Table of Contents

ArangoDB v3.3.19 HTTP API Documentation

Welcome to the ArangoDB HTTP API documentation! This documentation is for API developers. As a user or administrator of ArangoDB you should not need the information provided herein.

In general, as a user of ArangoDB you will use one of the language drivers.

General HTTP Request Handling in ArangoDB

Protocol

ArangoDB exposes its API via HTTP, making the server accessible easily with a variety of clients and tools (e.g. browsers, curl, telnet). The communication can optionally be SSL-encrypted.

ArangoDB uses the standard HTTP methods (e.g. *GET*, *POST*, *PUT*, *DELETE*) plus the *PATCH* method described in RFC 5789.

Most server APIs expect clients to send any payload data in JSON format. Details on the expected format and JSON attributes can be found in the documentation of the individual server methods.

Clients sending requests to ArangoDB must use either HTTP 1.0 or HTTP 1.1. Other HTTP versions are not supported by ArangoDB and any attempt to send a different HTTP version signature will result in the server responding with an HTTP 505 (HTTP version not supported) error.

ArangoDB will always respond to client requests with HTTP 1.1. Clients should therefore support HTTP version 1.1.

Clients are required to include the *Content-Length* HTTP header with the correct content length in every request that can have a body (e.g. *POST*, *PUT* or *PATCH*) request. ArangoDB will not process requests without a *Content-Length* header - thus chunked transfer encoding for POST-documents is not supported.

HTTP Keep-Alive

ArangoDB supports HTTP keep-alive. If the client does not send a *Connection* header in its request, and the client uses HTTP version 1.1, ArangoDB will assume the client wants to keep alive the connection. If clients do not wish to use the keep-alive feature, they should explicitly indicate that by sending a *Connection: Close* HTTP header in the request.

ArangoDB will close connections automatically for clients that send requests using HTTP 1.0, except if they send an *Connection: Keep-Alive* header.

The default Keep-Alive timeout can be specified at server start using the *--http.keep-alive-timeout* parameter.

Establishing TCP connections is expensive, since it takes several ping pongs between the communication parties. Therefore you can use connection keepalive to send several HTTP request over one TCP-connection; Each request is treated independently by definition. You can use this feature to build up a so called *connection pool* with several established connections in your client application, and dynamically re-use one of those then idle connections for subsequent requests.

Blocking vs. Non-blocking HTTP Requests

ArangoDB supports both blocking and non-blocking HTTP requests.

ArangoDB is a multi-threaded server, allowing the processing of multiple client requests at the same time. Request/response handling and the actual work are performed on the server in parallel by multiple worker threads.

Still, clients need to wait for their requests to be processed by the server, and thus keep one connection of a pool occupied. By default, the server will fully process an incoming request and then return the result to the client when the operation is finished. The client must wait for the server's HTTP response before it can send additional requests over the same connection. For clients that are single-threaded and/or are blocking on I/O themselves, waiting idle for the server response may be non-optimal.

To reduce blocking on the client side, ArangoDB offers a generic mechanism for non-blocking, asynchronous execution: clients can add the HTTP header *x-arango-async: true* to any of their requests, marking them as to be executed asynchronously on the server. ArangoDB will put such requests into an in-memory task queue and return an *HTTP 202* (accepted) response to the client instantly and thus finish this HTTP-request. The server will execute the tasks from the queue asynchronously as fast as possible, while clients can continue to do other work. If the server queue is full (i.e. contains as many tasks as specified by the option "--scheduler.maximal-queue-size"), then the request will be rejected instantly with an *HTTP 500* (internal server error) response.

Asynchronous execution decouples the request/response handling from the actual work to be performed, allowing fast server responses and greatly reducing wait time for clients. Overall this allows for much higher throughput than if clients would always wait for the server's response.

Keep in mind that the asynchronous execution is just "fire and forget". Clients will get any of their asynchronous requests answered with a generic HTTP 202 response. At the time the server sends this response, it does not know whether the requested operation can be carried out successfully (the actual operation execution will happen at some later point). Clients therefore cannot make a decision based on the server response and must rely on their requests being valid and processable by the server.

Additionally, the server's asynchronous task queue is an in-memory data structure, meaning not-yet processed tasks from the queue might be lost in case of a crash. Clients should therefore not use the asynchronous feature when they have strict durability requirements or if they rely on the immediate result of the request they send.

For details on the subsequent processing read on under Async Result handling.

Authentication

Client authentication can be achieved by using the *Authorization* HTTP header in client requests. ArangoDB supports authentication via HTTP Basic or JWT.

Authentication is turned on by default for all internal database APIs but turned off for custom Foxx apps. To toggle authentication for incoming requests to the internal database APIs, use the option --server.authentication. This option is turned on by default so authentication is required for the database APIs.

Please note that requests using the HTTP OPTIONS method will be answered by ArangoDB in any case, even if no authentication data is sent by the client or if the authentication data is wrong. This is required for handling CORS preflight requests (see Cross Origin Resource Sharing requests). The response to an HTTP OPTIONS request will be generic and not expose any private data.

There is an additional option to control authentication for custom Foxx apps. The option --server.authentication-system-only controls whether authentication is required only for requests to the internal database APIs and the admin interface. It is turned on by default, meaning that other APIs (this includes custom Foxx apps) do not require authentication.

The default values allow exposing a public custom Foxx API built with ArangoDB to the outside world without the need for HTTP authentication, but still protecting the usage of the internal database APIs (i.e. */_api/*, */_admin/*) with HTTP authentication.

If the server is started with the *--server.authentication-system-only* option set to *false*, all incoming requests will need HTTP authentication if the server is configured to require HTTP authentication (i.e. *--server.authentication true*). Setting the option to *true* will make the server require authentication only for requests to the internal database APIs and will allow unauthenticated requests to all other URLs.

Here's a short summary:

- `--server.authentication true --server.authentication-system-only true` : this will require authentication for all requests to the internal database APIs but not custom Foxx apps. This is the default setting.
- `--server.authentication true --server.authentication-system-only false` : this will require authentication for all requests (including custom Foxx apps).
- `--server.authentication false` : authentication disabled for all requests

Whenever authentication is required and the client has not yet authenticated, ArangoDB will return *HTTP 401* (Unauthorized). It will also send the *WWW-Authenticate* response header, indicating that the client should prompt the user for username and password if supported. If the client is a browser, then sending back this header will normally trigger the display of the browser-side HTTP authentication dialog. As showing the browser HTTP authentication dialog is undesired in AJAX requests, ArangoDB can be told to not send the *WWW-Authenticate* header back to the client. Whenever a client sends the *X-Omit-WWW-Authenticate* HTTP header (with an arbitrary value) to ArangoDB, ArangoDB will only send status code 401, but no *WWW-Authenticate* header. This allows clients to implement credentials handling and bypassing the browser's built-in dialog.

Authentication via JWT

ArangoDB uses a standard JWT based authentication method. To authenticate via JWT you must first obtain a JWT token with a signature generated via HMAC with SHA-256. The secret may either be set using `--server.jwt-secret` or will be randomly generated upon server startup.

For more information on JWT please consult RFC7519 and https://jwt.io

User JWT-Token

To authenticate with a specific user you need to supply a JWT token containing the *preferred_username* field with the username. You can either let ArangoDB generate this token for you via an API call or you can generate it yourself (only if you know the JWT secret).

ArangoDB offers a REST API to generate user tokens for you if you know the username and password. To do so send a POST request to

/_open/auth

containing *username* and *password* JSON-encoded like so:

{"username":"root","password":"rootPassword"}

Upon success the endpoint will return a 200 OK and an answer containing the JWT in a JSON- encoded object like so:

```
{"jwt":"eyJhbGciOiJIUzI1NiI..x6EfI"}
```

This JWT should then be used within the Authorization HTTP header in subsequent requests:

```
Authorization: bearer eyJhbGciOiJIUzI1NiI..x6EfI
```

Please note that the JWT will expire after 1 month and needs to be updated. We encode the expiration date of the JWT token in the *exp* field in unix time. Please note that all JWT tokens must contain the *iss* field with string value `arangodb` . As an example the decoded JWT body would look like this:

```
{
"exp": 1540381557,
"iat": 1537789.55727901,
"iss": "arangodb",
"preferred_username": "root"
}
```

Superuser JWT-Token

To access specific internal APIs as well as Agency and DBServer instances a token generated via `/open/auth` is not good enough. For these special APIs you will need to generate a special JWT token which grants superuser access. Note that using superuser access for normal database operations is **NOT advised**.

Note: It is only possible to generate this JWT token with the knowledge of the JWT secret.

Should you whish to generate the JWT token yourself with a tool of your choice, you need to include the correct body. The body must contain the *iss* field with string value `arangodb` and the `server_id` field with an arbirtrary string identifier:

```
{
"exp": 1537900279,
"iat": 1537800279,
"iss": "arangodb",
"server_id": "myclient"
}
```

For example to generate a token via the jwtgen tool (note the lifetime of one hour):

```
jwtgen -s <my-secret> -e 3600 -v -a "HS256" -c 'iss=arangodb' -c 'server_id=myclient'
curl -v -H "Authorization: bearer $(jwtgen -s <my-secret> -e 3600 -a "HS256" -c 'iss=arangodb' -c 'server_id=myclient')" http:/
/<database-ip>:8529/_api/version
```

Error Handling

The following should be noted about how ArangoDB handles client errors in its HTTP layer:

- client requests using an HTTP version signature different than *HTTP/1.0* or *HTTP/1.1* will get an *HTTP 505* (HTTP version not supported) error in return.
- ArangoDB will reject client requests with a negative value in the *Content-Length* request header with *HTTP 411* (Length Required).
- Arangodb doesn't support POST with *transfer-encoding: chunked* which forbids the *Content-Length* header above.
- the maximum URL length accepted by ArangoDB is 16K. Incoming requests with longer URLs will be rejected with an *HTTP 414* (Request-URI too long) error.
- if the client sends a *Content-Length* header with a value bigger than 0 for an HTTP GET, HEAD, or DELETE request, ArangoDB will process the request, but will write a warning to its log file.
- when the client sends a *Content-Length* header that has a value that is lower than the actual size of the body sent, ArangoDB will respond with *HTTP 400* (Bad Request).
- if clients send a *Content-Length* value bigger than the actual size of the body of the request, ArangoDB will wait for about 90 seconds for the client to complete its request. If the client does not send the remaining body data within this time, ArangoDB will close the connection. Clients should avoid sending such malformed requests as this will block one tcp connection, and may lead to a temporary filedescriptor leak.
- when clients send a body or a *Content-Length* value bigger than the maximum allowed value (512 MB), ArangoDB will respond with *HTTP 413* (Request Entity Too Large).
- if the overall length of the HTTP headers a client sends for one request exceeds the maximum allowed size (1 MB), the server will fail with *HTTP 431* (Request Header Fields Too Large).
- if clients request an HTTP method that is not supported by the server, ArangoDB will return with *HTTP 405* (Method Not Allowed). ArangoDB offers general support for the following HTTP methods:
  - GET
  - POST
  - PUT
  - DELETE
  - HEAD
  - PATCH
  - OPTIONS

  Please note that not all server actions allow using all of these HTTP methods. You should look up up the supported methods for each method you intend to use in the manual.

  Requests using any other HTTP method (such as for example CONNECT, TRACE etc.) will be rejected by ArangoDB as mentioned before.

Cross-Origin Resource Sharing (CORS) requests

ArangoDB will automatically handle CORS requests as follows:

Preflight

When a browser is told to make a cross-origin request that includes explicit headers, credentials or uses HTTP methods other than `GET` or `POST`, it will first perform a so-called preflight request using the `OPTIONS` method.

ArangoDB will respond to `OPTIONS` requests with an HTTP 200 status response with an empty body. Since preflight requests are not expected to include or even indicate the presence of authentication credentials even when they will be present in the actual request, ArangoDB does not enforce authentication for `OPTIONS` requests even when authentication is enabled.

ArangoDB will set the following headers in the response:

- `access-control-allow-credentials`: will be set to `false` by default. For details on when it will be set to `true` see the next section on cookies.
- `access-control-allow-headers`: will be set to the exect value of the request's `access-control-request-headers` header or omitted if no such header was sent in the request.
- `access-control-allow-methods`: will be set to a list of all supported HTTP headers regardless of the target endpoint. In other words that a method is listed in this header does not guarantee that it will be supported by the endpoint in the actual request.

- `access-control-allow-origin` : will be set to the exact value of the request's `origin` header.
- `access-control-expose-headers` : will be set to a list of response headers used by the ArangoDB HTTP API.
- `access-control-max-age` : will be set to an implementation-specifc value.

Actual request

If a request using any other HTTP method than `OPTIONS` includes an `origin` header, ArangoDB will add the following headers to the response:

- `access-control-allow-credentials` : will be set to `false` by default. For details on when it will be set to `true` see the next section on cookies.
- `access-control-allow-origin` : will be set to the exact value of the request's `origin` header.
- `access-control-expose-headers` : will be set to a list of response headers used by the ArangoDB HTTP API.

When making CORS requests to endpoints of Foxx services, the value of the `access-control-expose-headers` header will instead be set to a list of response headers used in the response itself (but not including the `access-control-` headers). Note that Foxx services may override this behaviour.

Cookies and authentication

In order for the client to be allowed to correctly provide authentication credentials or handle cookies, ArangoDB needs to set the `access-control-allow-credentials` response header to `true` instead of `false` .

ArangoDB will automatically set this header to `true` if the value of the request's `origin` header matches a trusted origin in the `http.trusted-origin` configuration option. To make ArangoDB trust a certain origin, you can provide a startup option when running `arangod` like this:

```
--http.trusted-origin "http://localhost:8529"
```

To specify multiple trusted origins, the option can be specified multiple times. Alternatively you can use the special value `"*"` to trust any origin:

```
--http.trusted-origin "*"
```

Note that browsers will not actually include credentials or cookies in cross-origin requests unless explicitly told to do so:

- When using the Fetch API you need to set the `credentials` option to `include` .

```
fetch("./", { credentials:"include" }).then(/* … */)
```

- When using `XMLHttpRequest` you need to set the `withCredentials` option to `true` .

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://example.com/', true);
xhr.withCredentials = true;
xhr.send(null);
```

- When using jQuery you need to set the `xhrFields` option:

```
$.ajax({
   url: 'https://example.com',
   xhrFields: {
      withCredentials: true
   }
});
```

HTTP method overriding

ArangoDB provides a startup option *--http.allow-method-override*. This option can be set to allow overriding the HTTP request method (e.g. GET, POST, PUT, DELETE, PATCH) of a request using one of the following custom HTTP headers:

- *x-http-method-override*
- *x-http-method*
- *x-method-override*

This allows using HTTP clients that do not support all "common" HTTP methods such as PUT, PATCH and DELETE. It also allows bypassing proxies and tools that would otherwise just let certain types of requests (e.g. GET and POST) pass through.

Enabling this option may impose a security risk, so it should only be used in very controlled environments. Thus the default value for this option is *false* (no method overriding allowed). You need to enable it explicitly if you want to use this feature.

Load-balancer support

When running in cluster mode, ArangoDB exposes some APIs which store request state data on specific coordinator nodes, and thus subsequent requests which require access to this state must be served by the coordinator node which owns this state data. In order to support function behind a load-balancer, ArangoDB can transparently forward requests within the cluster to the correct node. If a request is forwarded, the response will contain the following custom HTTP header whose value will be the ID of the node which actually answered the request:

- *x-arango-request-served-by*

The following APIs may use request forwarding:

- `/_api/cursor`

Note: since forwarding such requests require an additional cluster-internal HTTP request, they should be avoided when possible for best performance. Typically this is accomplished either by directing the requests to the correct coordinator at a client-level or by enabling request "stickiness" on a load balancer. Since these approaches are not always possible in a given environment, we support the request forwarding as a fall-back solution.

HTTP Interface

Following you have ArangoDB's HTTP Interface for Documents, Databases, Edges and more.

There are also some examples provided for every API action.

You may also use the interactive Swagger documentation in the ArangoDB webinterface to explore the API calls below.

HTTP Interface for Databases

Address of a Database

Any operation triggered via ArangoDB's HTTP REST API is executed in the context of exactly one database. To explicitly specify the database in a request, the request URI must contain the database name in front of the actual path:

```
http://localhost:8529/_db/mydb/...
```

where ... is the actual path to the accessed resource. In the example, the resource will be accessed in the context of the database *mydb*. Actual URLs in the context of *mydb* could look like this:

```
http://localhost:8529/_db/mydb/_api/version
http://localhost:8529/_db/mydb/_api/document/test/12345
http://localhost:8529/_db/mydb/myapp/get
```

Database-to-Endpoint Mapping

If a database name is present in the URI as above, ArangoDB will consult the database-to-endpoint mapping for the current endpoint, and validate if access to the database is allowed on the endpoint. If the endpoint is not restricted to an array of databases, ArangoDB will continue with the regular authentication procedure. If the endpoint is restricted to an array of specified databases, ArangoDB will check if the requested database is in the array. If not, the request will be turned down instantly. If yes, then ArangoDB will continue with the regular authentication procedure.

If the request URI was *http:// localhost:8529/_db/mydb/...*, then the request to *mydb* will be allowed (or disallowed) in the following situations:

```
Endpoint-to-database mapping          Access to *mydb* allowed?
----------------------------          -------------------------
[ ]                                   yes
[ "_system" ]                         no
[ "_system", "mydb" ]                 yes
[ "mydb" ]                            yes
[ "mydb", "_system" ]                 yes
[ "test1", "test2" ]                  no
```

In case no database name is specified in the request URI, ArangoDB will derive the database name from the endpoint-to-database mapping of the endpoint the connection was coming in on.

If the endpoint is not restricted to an array of databases, ArangoDB will assume the *_system* database. If the endpoint is restricted to one or multiple databases, ArangoDB will assume the first name from the array.

Following is an overview of which database name will be assumed for different endpoint-to-database mappings in case no database name is specified in the URI:

```
Endpoint-to-database mapping          Database
----------------------------          --------
[ ]                                   _system
[ "_system" ]                         _system
[ "_system", "mydb" ]                 _system
[ "mydb" ]                            mydb
[ "mydb", "_system" ]                 mydb
```

Database Management

This is an introduction to ArangoDB's HTTP interface for managing databases.

The HTTP interface for databases provides operations to create and drop individual databases. These are mapped to the standard HTTP methods *POST* and *DELETE*. There is also the *GET* method to retrieve an array of existing databases.

Please note that all database management operations can only be accessed via the default database (*_system*) and none of the other databases.

Managing Databases using HTTP

Information of the database

retrieves information about the current database

`GET /_api/database/current`

Retrieves information about the current database

The response is a JSON object with the following attributes:

- *name*: the name of the current database
- *id*: the id of the current database
- *path*: the filesystem path of the current database
- *isSystem*: whether or not the current database is the *_system* database

Return Codes

- *200*: is returned if the information was retrieved successfully.
- *400*: is returned if the request is invalid.
- *404*: is returned if the database could not be found.

Examples

```
shell> curl --dump - http://localhost:8529/_api/database/current

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

List of accessible databases

retrieves a list of all databases the current user can access

`GET /_api/database/user`

Retrieves the list of all databases the current user can access without specifying a different username or password.

Return Codes

- *200*: is returned if the list of database was compiled successfully.
- *400*: is returned if the request is invalid.

Examples

```
shell> curl --dump - http://localhost:8529/_api/database/user

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

List of databases

retrieves a list of all existing databases

`GET /_api/database`

Retrieves the list of all existing databases

Note: retrieving the list of databases is only possible from within the *_system* database.

Note: You should use the *GET user API* to fetch the list of the available databases now.

Return Codes

- *200*: is returned if the list of database was compiled successfully.
- *400*: is returned if the request is invalid.
- *403*: is returned if the request was not executed in the *_system* database.

Examples

```
shell> curl --dump - http://localhost:8529/_api/database

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Create database

creates a new database

`POST /_api/database`

A JSON object with these properties is required:

- **name**: Has to contain a valid database name.
- **users**: Has to be an array of user objects to initially create for the new database. User information will not be changed for users that already exist. If *users* is not specified or does not contain any users, a default user *root* will be created with an empty string password. This ensures that the new database will be accessible after it is created. Each user object can contain the following attributes:
  - **username**: Loginname of the user to be created
  - **passwd**: The user password as a string. If not specified, it will default to an empty string.
  - **active**: A flag indicating whether the user account should be activated or not. The default value is *true*. If set to *false*, the user won't be able to log into the database.
  - **extra**: A JSON object with extra user information. The data contained in *extra* will be stored for the user but not be interpreted further by ArangoDB.

Creates a new database

The response is a JSON object with the attribute *result* set to *true*.

Note: creating a new database is only possible from within the *_system* database.

Return Codes

- *201*: is returned if the database was created successfully.
- *400*: is returned if the request parameters are invalid or if a database with the specified name already exists.
- *403*: is returned if the request was not executed in the *_system* database.
- *409*: is returned if a database with the specified name already exists.

Examples

Creating a database named *example*.

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/database <<EOF
{
  "name" : "example"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Creating a database named *mydb* with two users.

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/database <<EOF
{
  "name" : "mydb",
  "users" : [
    {
      "username" : "admin",
      "passwd" : "secret",
      "active" : true
    },
    {
      "username" : "tester",
      "passwd" : "test001",
      "active" : false
    }
  ]
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Drop database

drop an existing database

`DELETE /_api/database/{database-name}`

Path Parameters

- *database-name* (required): The name of the database

Drops the database along with all data stored in it.

Note: dropping a database is only possible from within the *_system* database. The *_system* database itself cannot be dropped.

Return Codes

- *200*: is returned if the database was dropped successfully.
- *400*: is returned if the request is malformed.
- *403*: is returned if the request was not executed in the *_system* database.
- *404*: is returned if the database could not be found.

Examples

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/database/example

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Notes on Databases

Please keep in mind that each database contains its own system collections, which need to set up when a database is created. This will make the creation of a database take a while. Replication is configured on a per-database level, meaning that any replication logging or applying for the a new database must be configured explicitly after a new database has been created. Foxx applications are also available only in the context of the database they have been installed in. A new database will only provide access to the system applications shipped with ArangoDB (that is the web interface at the moment) and no other Foxx applications until they are explicitly installed for the particular database.

Database

ArangoDB can handle multiple databases in the same server instance. Databases can be used to logically group and separate data. An ArangoDB database consists of collections and dedicated database-specific worker processes. A database contains its own collections (which cannot be accessed from other databases), Foxx applications and replication loggers and appliers. Each ArangoDB database contains its own system collections (e.g. _users, _graphs, ...).

There will always be at least one database in ArangoDB. This is the default database named _system. This database cannot be dropped and provides special operations for creating, dropping and enumerating databases. Users can create additional databases and give them unique names to access them later. Database management operations cannot be initiated from out of user-defined databases.

When ArangoDB is accessed via its HTTP REST API, the database name is read from the first part of the request URI path (e.g. /_db/_system/...). If the request URI does not contain a database name, the database name is automatically determined by the algorithm described in Database-to-Endpoint Mapping .

Database Name

A single ArangoDB instance can handle multiple databases in parallel. When multiple databases are used, each database must be given an unique name. This name is used to uniquely identify a database. The default database in ArangoDB is named *system*. *The database name is a string consisting of only letters, digits and the* (underscore) and - (dash) characters. User-defined database names must always start with a letter. Database names are case-sensitive.

Database Organization

A single ArangoDB instance can handle multiple databases in parallel. By default, there will be at least one database which is named _system. Databases are physically stored in separate sub-directories underneath the database directory, which itself resides in the instance's data directory.

Each database has its own sub-directory, named database-<id>. The database directory contains sub-directories for the collections of the database, and a file named parameter.json. This file contains the database id and name.

In an example ArangoDB instance which has two databases, the filesystem layout could look like this:

```
data/                    # the instance's data directory
  databases/             # sub-directory containing all databases' data
    database-<id>/       # sub-directory for a single database
      parameter.json     # file containing database id and name
      collection-<id>/   # directory containing data about a collection
    database-<id>/       # sub-directory for another database
      parameter.json     # file containing database id and name
      collection-<id>/   # directory containing data about a collection
      collection-<id>/   # directory containing data about a collection
```

Foxx applications are also organized in database-specific directories inside the application path. The filesystem layout could look like this:

```
apps/                  # the instance's application directory
  system/              # system applications (can be ignored)
  databases/           # sub-directory containing database-specific applications
    <database-name>/   # sub-directory for a single database
      <app-name>       # sub-directory for a single application
```

```
        <app-name>        # sub-directory for a single application
      <database-name>/    # sub-directory for another database
        <app-name>        # sub-directory for a single application
`
```

HTTP Interface for Collections

Collections

This is an introduction to ArangoDB's HTTP interface for collections.

Collection

A collection consists of documents. It is uniquely identified by its collection identifier. It also has a unique name that clients should use to identify and access it. Collections can be renamed. This will change the collection name, but not the collection identifier. Collections have a type that is specified by the user when the collection is created. There are currently two types: document and edge. The default type is document.

Collection Identifier

A collection identifier lets you refer to a collection in a database. It is a string value and is unique within the database. Up to including ArangoDB 1.1, the collection identifier has been a client's primary means to access collections. Starting with ArangoDB 1.2, clients should instead use a collection's unique name to access a collection instead of its identifier. ArangoDB currently uses 64bit unsigned integer values to maintain collection ids internally. When returning collection ids to clients, ArangoDB will put them into a string to ensure the collection id is not clipped by clients that do not support big integers. Clients should treat the collection ids returned by ArangoDB as opaque strings when they store or use it locally.

Note: collection ids have been returned as integers up to including ArangoDB 1.1

Collection Name

A collection name identifies a collection in a database. It is a string and is unique within the database. Unlike the collection identifier it is supplied by the creator of the collection. The collection name must consist of letters, digits, and the _ (underscore) and - (dash) characters only. Please refer to Naming Conventions in ArangoDB for more information on valid collection names.

Key Generator

ArangoDB allows using key generators for each collection. Key generators have the purpose of auto-generating values for the _key attribute of a document if none was specified by the user. By default, ArangoDB will use the traditional key generator. The traditional key generator will auto-generate key values that are strings with ever-increasing numbers. The increment values it uses are non-deterministic.

Contrary, the auto increment key generator will auto-generate deterministic key values. Both the start value and the increment value can be defined when the collection is created. The default start value is 0 and the default increment is 1, meaning the key values it will create by default are:

1, 2, 3, 4, 5, ...

When creating a collection with the auto increment key generator and an increment of 5, the generated keys would be:

1, 6, 11, 16, 21, ...

The auto-increment values are increased and handed out on each document insert attempt. Even if an insert fails, the auto-increment value is never rolled back. That means there may exist gaps in the sequence of assigned auto-increment values if inserts fails.

The basic operations (create, read, update, delete) for documents are mapped to the standard HTTP methods (*POST*, *GET*, *PUT*, *DELETE*).

Address of a Collection

All collections in ArangoDB have an unique identifier and a unique name. ArangoDB internally uses the collection's unique identifier to look up collections. This identifier however is managed by ArangoDB and the user has no control over it. In order to allow users use their own names, each collection also has a unique name, which is specified by the user. To access a collection from the user perspective, the

collection name should be used, i.e.:

```
http://server:port/_api/collection/collection-name
```

For example: Assume that the collection identifier is *7254820* and the collection name is *demo*, then the URL of that collection is:

```
http://localhost:8529/_api/collection/demo
```

Creating and Deleting Collections

Create collection

creates a collection

`POST /_api/collection`

A JSON object with these properties is required:

- **journalSize**: The maximal size of a journal or datafile in bytes. The value must be at least `1048576` (1 MiB). (The default is a configuration parameter) This option is meaningful for the MMFiles storage engine only.
- **replicationFactor**: (The default is *1*): in a cluster, this attribute determines how many copies of each shard are kept on different DBServers. The value 1 means that only one copy (no synchronous replication) is kept. A value of k means that k-1 replicas are kept. Any two copies reside on different DBServers. Replication between them is synchronous, that is, every write operation to the "leader" copy will be replicated to all "follower" replicas, before the write operation is reported successful. If a server fails, this is detected automatically and one of the servers holding copies take over, usually without an error being reported.
- **keyOptions**:
  - **allowUserKeys**: if set to *true*, then it is allowed to supply own key values in the *_key* attribute of a document. If set to *false*, then the key generator will solely be responsible for generating keys and supplying own key values in the *_key* attribute of documents is considered an error.
  - **type**: specifies the type of the key generator. The currently available generators are *traditional* and *autoincrement*.
  - **increment**: increment value for *autoincrement* key generator. Not used for other key generator types.
  - **offset**: Initial offset value for *autoincrement* key generator. Not used for other key generator types.
- **name**: The name of the collection.
- **waitForSync**: If *true* then the data is synchronized to disk before returning from a document create, update, replace or removal operation. (default: false)
- **doCompact**: whether or not the collection will be compacted (default is *true*) This option is meaningful for the MMFiles storage engine only.
- **isVolatile**: If *true* then the collection data is kept in-memory only and not made persistent. Unloading the collection will cause the collection data to be discarded. Stopping or re-starting the server will also cause full loss of data in the collection. Setting this option will make the resulting collection be slightly faster than regular collections because ArangoDB does not enforce any synchronization to disk and does not calculate any CRC checksums for datafiles (as there are no datafiles). This option should therefore be used for cache-type collections only, and not for data that cannot be re-created otherwise. (The default is *false*) This option is meaningful for the MMFiles storage engine only.
- **shardKeys**: (The default is *["_key"]*): in a cluster, this attribute determines which document attributes are used to determine the target shard for documents. Documents are sent to shards based on the values of their shard key attributes. The values of all shard key attributes in a document are hashed, and the hash value is used to determine the target shard. **Note**: Values of shard key attributes cannot be changed once set. This option is meaningless in a single server setup.
- **numberOfShards**: (The default is *1*): in a cluster, this value determines the number of shards to create for the collection. In a single server setup, this option is meaningless.
- **isSystem**: If *true*, create a system collection. In this case *collection-name* should start with an underscore. End users should normally create non-system collections only. API implementors may be required to create system collections in very special occasions, but normally a regular collection will do. (The default is *false*)
- **type**: (The default is *2*): the type of the collection to create. The following values for *type* are valid:
  - *2*: document collection
  - *3*: edges collection
- **indexBuckets**: The number of buckets into which indexes using a hash table are split. The default is 16 and this number has to be a power of 2 and less than or equal to 1024. For very large collections one should increase this to avoid long pauses when the hash table has to be initially built or resized, since buckets are resized individually and can be initially built in parallel. For example, 64 might be a sensible value for a collection with 100 000 000 documents. Currently, only the edge index respects this value, but other index types might follow in future ArangoDB versions. Changes (see below) are applied when the collection is loaded the next time. This option is meaningful for the MMFiles storage engine only.
- **distributeShardsLike**: (The default is *""*): in an enterprise cluster, this attribute binds the specifics of sharding for the newly created collection to follow that of a specified existing collection. **Note**: Using this parameter has consequences for the prototype collection. It can no longer be dropped, before sharding imitating collections are dropped. Equally, backups and restores of imitating

collections alone will generate warnings, which can be overridden, about missing sharding prototype.

Creates a new collection with a given name. The request must contain an object with the following attributes.

Query Parameters

- *waitForSyncReplication* (optional): Default is *1* which means the server will only report success back to the client if all replicas have created the collection. Set to *0* if you want faster server responses and don't care about full replication.
- *enforceReplicationFactor* (optional): Default is *1* which means the server will check if there are enough replicas available at creation time and bail out otherwise. Set to *0* to disable this extra check.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/collection <<EOF
{
  "name" : "testCollectionBasics"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/collection <<EOF
{
  "name" : "testCollectionUsers",
  "keyOptions" : {
    "type" : "autoincrement",
    "increment" : 5,
    "allowUserKeys" : true
  }
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Drops a collection

drops a collection

`DELETE /_api/collection/{collection-name}`

Path Parameters

- *collection-name* (required): The name of the collection to drop.

Query Parameters

- *isSystem* (optional): Whether or not the collection to drop is a system collection. This parameter must be set to *true* in order to drop a system collection.

Drops the collection identified by *collection-name*.

If the collection was successfully dropped, an object is returned with the following attributes:

- *error*: *false*
- *id*: The identifier of the dropped collection.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Using an identifier:

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/collection/9857

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Using a name:

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/collection/products1

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Dropping a system collection

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/collection/_example?
isSystem=true

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Truncate collection

truncates a collection

`PUT /_api/collection/{collection-name}/truncate`

Path Parameters

- *collection-name* (required): The name of the collection.

Removes all documents from the collection, but leaves the indexes intact.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

```
shell> curl -X PUT --dump - http://localhost:8529/_api/collection/products/truncate

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/truncate
```

show response body

Getting Information about a Collection

Return information about a collection

returns a collection

```
GET /_api/collection/{collection-name}
```

Path Parameters

- *collection-name* (required): The name of the collection.

The result is an object describing the collection with the following attributes:

- *id*: The identifier of the collection.
- *name*: The name of the collection.
- *status*: The status of the collection as number.
  - 1: new born collection
  - 2: unloaded
  - 3: loaded
  - 4: in the process of being unloaded
  - 5: deleted
  - 6: loading

Every other status indicates a corrupted collection.

- *type*: The type of the collection as number.
  - 2: document collection (normal case)
  - 3: edges collection
- *isSystem*: If *true* then the collection is a system collection.

Return Codes

- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Read properties of a collection

reads the properties of the specified collection

```
GET /_api/collection/{collection-name}/properties
```

Path Parameters

- *collection-name* (required): The name of the collection.

In addition to the above, the result will always contain the *waitForSync* attribute, and the *doCompact*, *journalSize*, and *isVolatile* attributes for the MMFiles storage engine. This is achieved by forcing a load of the underlying collection.

- *waitForSync*: If *true* then creating, changing or removing documents will wait until the data has been synchronized to disk.
- *doCompact*: Whether or not the collection will be compacted. This option is only present for the MMFiles storage engine.
- *journalSize*: The maximal size setting for journals / datafiles in bytes. This option is only present for the MMFiles storage engine.
- *keyOptions*: JSON object which contains key generation options:
  - *type*: specifies the type of the key generator. The currently available generators are *traditional* and *autoincrement*.
  - *allowUserKeys*: if set to *true*, then it is allowed to supply own key values in the *_key* attribute of a document. If set to *false*, then the key generator is solely responsible for generating keys and supplying own key values in the *_key* attribute of documents is considered an error.
- *isVolatile*: If *true* then the collection data will be kept in memory only and ArangoDB will not write or sync the data to disk. This option is only present for the MMFiles storage engine.

In a cluster setup, the result will also contain the following attributes:

- *numberOfShards*: the number of shards of the collection.
- *shardKeys*: contains the names of document attributes that are used to determine the target shard for documents.
- *replicationFactor*: contains how many copies of each shard are kept on different DBServers.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Using an identifier:

```
shell> curl --dump - http://localhost:8529/_api/collection/10216/properties

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/10216/properties
```

show response body
Using a name:

```
shell> curl --dump - http://localhost:8529/_api/collection/products/properties

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/properties
```

show response body

Return number of documents in a collection

Counts the documents in a collection

`GET /_api/collection/{collection-name}/count`

Path Parameters

- *collection-name* (required): The name of the collection.

In addition to the above, the result also contains the number of documents. **Note** that this will always load the collection into memory.

- *count*: The number of documents inside the collection.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Requesting the number of documents:

```
shell> curl --dump - http://localhost:8529/_api/collection/products/count
```

```
HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/count
```

show response body

Return statistics for a collection

Fetch the statistics of a collection

`GET /_api/collection/{collection-name}/figures`

Path Parameters

- *collection-name* (required): The name of the collection.

In addition to the above, the result also contains the number of documents and additional statistical information about the collection. **Note** : This will always load the collection into memory.

Note: collection data that are stored in the write-ahead log only are not reported in the results. When the write-ahead log is collected, documents might be added to journals and datafiles of the collection, which may modify the figures of the collection.

Additionally, the filesizes of collection and index parameter JSON files are not reported. These files should normally have a size of a few bytes each. Please also note that the *fileSize* values are reported in bytes and reflect the logical file sizes. Some filesystems may use optimisations (e.g. sparse files) so that the actual physical file size is somewhat different. Directories and sub-directories may also require space in the file system, but this space is not reported in the *fileSize* results.

That means that the figures reported do not reflect the actual disk usage of the collection with 100% accuracy. The actual disk usage of a collection is normally slightly higher than the sum of the reported *fileSize* values. Still the sum of the *fileSize* values can still be used as a lower bound approximation of the disk usage.

HTTP 200

A json document with these Properties is returned:

Returns information about the collection:

- **count**: The number of documents currently present in the collection.
- **journalSize**: The maximal size of a journal or datafile in bytes.
- **figures**:
  - **datafiles**:
    - **count**: The number of datafiles.
    - **fileSize**: The total filesize of datafiles (in bytes).
  - **uncollectedLogfileEntries**: The number of markers in the write-ahead log for this collection that have not been transferred to journals or datafiles.
  - **documentReferences**: The number of references to documents in datafiles that JavaScript code currently holds. This information can be used for debugging compaction and unload issues.
  - **compactionStatus**:
    - **message**: The action that was performed when the compaction was last run for the collection. This information can be used for debugging compaction issues.
    - **time**: The point in time the compaction for the collection was last executed. This information can be used for debugging compaction issues.
  - **compactors**:
    - **count**: The number of compactor files.
    - **fileSize**: The total filesize of all compactor files (in bytes).
  - **dead**:
    - **count**: The number of dead documents. This includes document versions that have been deleted or replaced by a newer version. Documents deleted or replaced that are contained the write-ahead log only are not reported in this figure.
    - **deletion**: The total number of deletion markers. Deletion markers only contained in the write-ahead log are not reporting

in this figure.
    - **size**: The total size in bytes used by all dead documents.
- **indexes**:
    - **count**: The total number of indexes defined for the collection, including the pre-defined indexes (e.g. primary index).
    - **size**: The total memory allocated for indexes in bytes.
- **readcache**:
    - **count**: The number of revisions of this collection stored in the document revisions cache.
    - **size**: The memory used for storing the revisions of this collection in the document revisions cache (in bytes). This figure does not include the document data but only mappings from document revision ids to cache entry locations.
- **waitingFor**: An optional string value that contains information about which object type is at the head of the collection's cleanup queue. This information can be used for debugging compaction and unload issues.
- **alive**:
    - **count**: The number of currently active documents in all datafiles and journals of the collection. Documents that are contained in the write-ahead log only are not reported in this figure.
    - **size**: The total size in bytes used by all active documents of the collection. Documents that are contained in the write-ahead log only are not reported in this figure.
- **lastTick**: The tick of the last marker that was stored in a journal of the collection. This might be 0 if the collection does not yet have a journal.
- **journals**:
    - **count**: The number of journal files.
    - **fileSize**: The total filesize of all journal files (in bytes).
- **revisions**:
    - **count**: The number of revisions of this collection managed by the storage engine.
    - **size**: The memory used for storing the revisions of this collection in the storage engine (in bytes). This figure does not include the document data but only mappings from document revision ids to storage engine datafile positions.

Return Codes

- *200*: Returns information about the collection:

Response Body

- **count**: The number of documents currently present in the collection.
- **journalSize**: The maximal size of a journal or datafile in bytes.
- **figures**:
    - **datafiles**:
        - **count**: The number of datafiles.
        - **fileSize**: The total filesize of datafiles (in bytes).
    - **uncollectedLogfileEntries**: The number of markers in the write-ahead log for this collection that have not been transferred to journals or datafiles.
    - **lastTick**: The tick of the last marker that was stored in a journal of the collection. This might be 0 if the collection does not yet have a journal.
    - **compactionStatus**:
        - **message**: The action that was performed when the compaction was last run for the collection. This information can be used for debugging compaction issues.
        - **time**: The point in time the compaction for the collection was last executed. This information can be used for debugging compaction issues.
    - **dead**:
        - **count**: The number of dead documents. This includes document versions that have been deleted or replaced by a newer version. Documents deleted or replaced that are contained the write-ahead log only are not reported in this figure.
        - **deletion**: The total number of deletion markers. Deletion markers only contained in the write-ahead log are not reporting in this figure.
        - **size**: The total size in bytes used by all dead documents.
    - **compactors**:
        - **count**: The number of compactor files.
        - **fileSize**: The total filesize of all compactor files (in bytes).
    - **readcache**:

        - **count**: The number of revisions of this collection stored in the document revisions cache.
        - **size**: The memory used for storing the revisions of this collection in the document revisions cache (in bytes). This figure does not include the document data but only mappings from document revision ids to cache entry locations.
    - **waitingFor**: An optional string value that contains information about which object type is at the head of the collection's cleanup queue. This information can be used for debugging compaction and unload issues.
    - **alive**:
        - **count**: The number of currently active documents in all datafiles and journals of the collection. Documents that are contained in the write-ahead log only are not reported in this figure.
        - **size**: The total size in bytes used by all active documents of the collection. Documents that are contained in the write-ahead log only are not reported in this figure.
    - **documentReferences**: The number of references to documents in datafiles that JavaScript code currently holds. This information can be used for debugging compaction and unload issues.
    - **indexes**:
        - **count**: The total number of indexes defined for the collection, including the pre-defined indexes (e.g. primary index).
        - **size**: The total memory allocated for indexes in bytes.
    - **journals**:
        - **count**: The number of journal files.
        - **fileSize**: The total filesize of all journal files (in bytes).
    - **revisions**:
        - **count**: The number of revisions of this collection managed by the storage engine.
        - **size**: The memory used for storing the revisions of this collection in the storage engine (in bytes). This figure does not include the document data but only mappings from document revision ids to storage engine datafile positions.
- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Using an identifier and requesting the figures of the collection:

```
shell> curl --dump - http://localhost:8529/_api/collection/products/figures

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/figures
```

show response body

Return collection revision id

Retrieve the collections revision id

`GET /_api/collection/{collection-name}/revision`

Path Parameters

- *collection-name* (required): The name of the collection.

In addition to the above, the result will also contain the collection's revision id. The revision id is a server-generated string that clients can use to check whether data in a collection has changed since the last revision check.

- *revision*: The collection revision id as a string.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Retrieving the revision of a collection

```
shell> curl --dump - http://localhost:8529/_api/collection/products/revision

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/revision
```

show response body

Return checksum for the collection

returns a checksum for the specified collection

`GET /_api/collection/{collection-name}/checksum`

Path Parameters

- *collection-name* (required): The name of the collection.

Query Parameters

- *withRevisions* (optional): Whether or not to include document revision ids in the checksum calculation.
- *withData* (optional): Whether or not to include document body data in the checksum calculation.

Will calculate a checksum of the meta-data (keys and optionally revision ids) and optionally the document data in the collection.

The checksum can be used to compare if two collections on different ArangoDB instances contain the same contents. The current revision of the collection is returned too so one can make sure the checksums are calculated for the same state of data.

By default, the checksum will only be calculated on the *_key* system attribute of the documents contained in the collection. For edge collections, the system attributes *_from* and *_to* will also be included in the calculation.

By setting the optional query parameter *withRevisions* to *true*, then revision ids (*_rev* system attributes) are included in the checksumming.

By providing the optional query parameter *withData* with a value of *true*, the user-defined document attributes will be included in the calculation too. **Note**: Including user-defined attributes will make the checksumming slower.

The response is a JSON object with the following attributes:

- *checksum*: The calculated checksum as a number.
- *revision*: The collection revision id as a string.

Note: this method is not available in a cluster.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Retrieving the checksum of a collection:

```
shell> curl --dump - http://localhost:8529/_api/collection/products/checksum

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/checksum
```

show response body

Retrieving the checksum of a collection including the collection data, but not the revisions:

```
shell> curl --dump - http://localhost:8529/_api/collection/products/checksum?
withRevisions=false&withData=true

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/checksum
```

show response body

reads all collections

returns all collections

`GET /_api/collection`

Query Parameters

- *excludeSystem* (optional): Whether or not system collections should be excluded from the result.

Returns an object with an attribute *collections* containing an array of all collection descriptions. The same information is also available in the *names* as an object with the collection names as keys.

By providing the optional query parameter *excludeSystem* with a value of *true*, all system collections will be excluded from the response.

Return Codes

- *200*: The list of collections

Examples

Return information about all collections:

```
shell> curl --dump - http://localhost:8529/_api/collection

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Modifying a Collection

Load collection

loads a collection

`PUT /_api/collection/{collection-name}/load`

Path Parameters

- *collection-name* (required): The name of the collection.

Loads a collection into memory. Returns the collection on success.

The request body object might optionally contain the following attribute:

- *count*: If set, this controls whether the return value should include the number of documents in the collection. Setting *count* to *false* may speed up loading a collection. The default value for *count* is *true*.

On success an object with the following attributes is returned:

- *id*: The identifier of the collection.
- *name*: The name of the collection.
- *count*: The number of documents inside the collection. This is only returned if the *count* input parameters is set to *true* or has not been specified.
- *status*: The status of the collection as number.
- *type*: The collection type. Valid types are:
  - 2: document collection
  - 3: edges collection
- *isSystem*: If *true* then the collection is a system collection.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

```
shell> curl -X PUT --dump - http://localhost:8529/_api/collection/products/load

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/load
```

show response body

Unload collection

unloads a collection

`PUT /_api/collection/{collection-name}/unload`

Path Parameters

- *collection-name* (required):

Removes a collection from memory. This call does not delete any documents. You can use the collection afterwards; in which case it will be loaded into memory, again. On success an object with the following attributes is returned:

- *id*: The identifier of the collection.
- *name*: The name of the collection.
- *status*: The status of the collection as number.
- *type*: The collection type. Valid types are:
  - 2: document collection
  - 3: edges collection
- *isSystem*: If *true* then the collection is a system collection.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

```
shell> curl -X PUT --dump - http://localhost:8529/_api/collection/products/unload

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/unload
```

show response body

Load Indexes into Memory

Load Indexes into Memory

`PUT /_api/collection/{collection-name}/loadIndexesIntoMemory`

Path Parameters

- *collection-name* (required):

This route tries to cache all index entries of this collection into the main memory. Therefore it iterates over all indexes of the collection and stores the indexed values, not the entire document data, in memory. All lookups that could be found in the cache are much faster than lookups not stored in the cache so you get a nice performance boost. It is also guaranteed that the cache is consistent with the stored data.

For the time being this function is only useful on RocksDB storage engine, as in MMFiles engine all indexes are in memory anyways.

On RocksDB this function honors all memory limits, if the indexes you want to load are smaller than your memory limit this function guarantees that most index values are cached. If the index is larger than your memory limit this function will fill up values up to this limit and for the time being there is no way to control which indexes of the collection should have priority over others.

On sucess this function returns an object with attribute `result` set to `true`

Return Codes

- *200*: If the indexes have all been loaded
- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

```
shell> curl -X PUT --dump -
http://localhost:8529/_api/collection/products/loadIndexesIntoMemory

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/loadIndexesIntoMemory
```

show response body

Change properties of a collection

changes a collection

`PUT /_api/collection/{collection-name}/properties`

Path Parameters

- *collection-name* (required): The name of the collection.

Changes the properties of a collection. Expects an object with the attribute(s)

- *waitForSync*: If *true* then creating or changing a document will wait until the data has been synchronized to disk.
- *journalSize*: The maximal size of a journal or datafile in bytes. The value must be at least `1048576` (1 MB). Note that when changing the journalSize value, it will only have an effect for additional journals or datafiles that are created. Already existing journals or datafiles will not be affected.

On success an object with the following attributes is returned:

- *id*: The identifier of the collection.
- *name*: The name of the collection.
- *waitForSync*: The new value.
- *journalSize*: The new value.
- *status*: The status of the collection as number.
- *type*: The collection type. Valid types are:
  - 2: document collection
  - 3: edges collection
- *isSystem*: If *true* then the collection is a system collection.
- *isVolatile*: If *true* then the collection data will be kept in memory only and ArangoDB will not write or sync the data to disk.
- *doCompact*: Whether or not the collection will be compacted.
- *keyOptions*: JSON object which contains key generation options:
  - *type*: specifies the type of the key generator. The currently available generators are *traditional* and *autoincrement*.
  - *allowUserKeys*: if set to *true*, then it is allowed to supply own key values in the *_key* attribute of a document. If set to *false*, then the key generator is solely responsible for generating keys and supplying own key values in the *_key* attribute of documents is considered an error.

Note: except for *waitForSync*, *journalSize* and *name*, collection properties **cannot be changed** once a collection is created. To rename a collection, the rename endpoint must be used.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/collection/products/properties <<EOF
{
  "waitForSync" : true
}
EOF

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/properties
```

show response body

Rename collection

renames a collection

`PUT /_api/collection/{collection-name}/rename`

Path Parameters

- *collection-name* (required): The name of the collection to rename.

Renames a collection. Expects an object with the attribute(s)

- *name*: The new name.

It returns an object with the attributes

- *id*: The identifier of the collection.
- *name*: The new name of the collection.
- *status*: The status of the collection as number.
- *type*: The collection type. Valid types are:
  - 2: document collection
  - 3: edges collection
- *isSystem*: If *true* then the collection is a system collection.

If renaming the collection succeeds, then the collection is also renamed in all graph definitions inside the `_graphs` collection in the current database.

Note: this method is not available in a cluster.

Return Codes

- *400*: If the *collection-name* is missing, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/collection/products1/rename <<EOF
{
  "name" : "newname"
}
EOF
```

```
HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products1/rename
```

show response body

Rotate journal of a collection

rotates the journal of a collection

`PUT /_api/collection/{collection-name}/rotate`

Path Parameters

- *collection-name* (required): The name of the collection.

Rotates the journal of a collection. The current journal of the collection will be closed and made a read-only datafile. The purpose of the rotate method is to make the data in the file available for compaction (compaction is only performed for read-only datafiles, and not for journals).

Saving new data in the collection subsequently will create a new journal file automatically if there is no current journal.

It returns an object with the attributes

- *result*: will be *true* if rotation succeeded

Note: this method is specific for the MMFiles storage engine, and there it is not available in a cluster.

Return Codes

- *400*: If the collection currently has no journal, *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Rotating the journal:

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/collection/products/rotate <<EOF
{
}
EOF

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
location: /_api/collection/products/rotate
```

show response body
Rotating if no journal exists:

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/collection/products/rotate <<EOF
{
}
EOF
```

```
HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

```

show response body

HTTP Interface for Documents

In this chapter we describe the REST API of ArangoDB for documents.

Basics and Terminology

Documents, Keys, Handles and Revisions

Documents in ArangoDB are JSON objects. These objects can be nested (to any depth) and may contain lists. Each document has a unique primary key which identifies it within its collection. Furthermore, each document is uniquely identified by its document handle across all collections in the same database. Different revisions of the same document (identified by its handle) can be distinguished by their document revision. Any transaction only ever sees a single revision of a document.

Here is an example document:

```
{
  "_id" : "myusers/3456789",
  "_key" : "3456789",
  "_rev" : "14253647",
  "firstName" : "John",
  "lastName" : "Doe",
  "address" : {
    "street" : "Road To Nowhere 1",
    "city" : "Gotham"
  },
  "hobbies" : [
    {name: "swimming", howFavorite: 10},
    {name: "biking", howFavorite: 6},
    {name: "programming", howFavorite: 4}
  ]
}
```

All documents contain special attributes: the document handle is stored as a string in `_id`, the document's primary key in `_key` and the document revision in `_rev`. The value of the `_key` attribute can be specified by the user when creating a document. `_id` and `_key` values are immutable once the document has been created. The `_rev` value is maintained by ArangoDB automatically.

Document Handle

A document handle uniquely identifies a document in the database. It is a string and consists of the collection's name and the document key (`_key` attribute) separated by `/`.

Document Key

A document key uniquely identifies a document in the collection it is stored in. It can and should be used by clients when specific documents are queried. The document key is stored in the `_key` attribute of each document. The key values are automatically indexed by ArangoDB in a collection's primary index. Thus looking up a document by its key is a fast operation. The _key value of a document is immutable once the document has been created. By default, ArangoDB will auto-generate a document key if no _key attribute is specified, and use the user-specified _key otherwise.

This behavior can be changed on a per-collection level by creating collections with the `keyOptions` attribute.

Using `keyOptions` it is possible to disallow user-specified keys completely, or to force a specific regime for auto-generating the `_key` values.

Document Revision

As ArangoDB supports MVCC (Multiple Version Concurrency Control), documents can exist in more than one revision. The document revision is the MVCC token used to specify a particular revision of a document (identified by its `_id`). It is a string value currently containing an integer number and is unique within the list of document revisions for a single document. Document revisions can be used to conditionally query, update, replace or delete documents in the database. In order to find a particular revision of a document, you need the document handle or key, and the document revision.

ArangoDB uses 64bit unsigned integer values to maintain document revisions internally. When returning document revisions to clients, ArangoDB will put them into a string to ensure the revision is not clipped by clients that do not support big integers. Clients should treat the revision returned by ArangoDB as an opaque string when they store or use it locally. This will allow ArangoDB to change the format of revisions later if this should be required. Clients can use revisions to perform simple equality/non-equality comparisons (e.g. to check whether a document has changed or not), but they should not use revision ids to perform greater/less than comparisons with them to check if a document revision is older than one another, even if this might work for some cases.

Document Etag

ArangoDB tries to adhere to the existing HTTP standard as far as possible. To this end, results of single document queries have the HTTP header `Etag` set to the document revision enclosed in double quotes.

The basic operations (create, read, exists, replace, update, delete) for documents are mapped to the standard HTTP methods (*POST*, *GET*, *HEAD*, *PUT*, *PATCH* and *DELETE*).

If you modify a document, you can use the *If-Match* field to detect conflicts. The revision of a document can be checking using the HTTP method *HEAD*.

Multiple Documents in a single Request

Beginning with ArangoDB 3.0 the basic document API has been extended to handle not only single documents but multiple documents in a single request. This is crucial for performance, in particular in the cluster situation, in which a single request can involve multiple network hops within the cluster. Another advantage is that it reduces the overhead of the HTTP protocol and individual network round trips between the client and the server. The general idea to perform multiple document operations in a single request is to use a JSON array of objects in the place of a single document. As a consequence, document keys, handles and revisions for preconditions have to be supplied embedded in the individual documents given. Multiple document operations are restricted to a single document or edge collections. See the API descriptions for details.

Note that the *GET*, *HEAD* and *DELETE* HTTP operations generally do not allow to pass a message body. Thus, they cannot be used to perform multiple document operations in one request. However, there are other endpoints to request and delete multiple documents in one request. FIXME: ADD SENSIBLE LINKS HERE.

URI of a Document

Any document can be retrieved using its unique URI:

```
http://server:port/_api/document/<document-handle>
```

For example, assuming that the document handle is `demo/362549736`, then the URL of that document is:

```
http://localhost:8529/_api/document/demo/362549736
```

The above URL schema does not specify a database name explicitly, so the default database `_system` will be used. To explicitly specify the database context, use the following URL schema:

```
http://server:port/_db/<database-name>/_api/document/<document-handle>
```

Example:

```
http://localhost:8529/_db/mydb/_api/document/demo/362549736
```

Note: The following examples use the short URL format for brevity.

The document revision is returned in the "Etag" HTTP header when requesting a document.

If you obtain a document using *GET* and you want to check whether a newer revision is available, then you can use the *If-None-Match* header. If the document is unchanged, a *HTTP 412* (precondition failed) error is returned.

If you want to query, replace, update or delete a document, then you can use the *If-Match* header. If the document has changed, then the operation is aborted and an *HTTP 412* error is returned.

Working with Documents using REST

Read document

reads a single document

`GET /_api/document/{document-handle}`

Path Parameters

- *document-handle* (required): The handle of the document.

Header Parameters

- *If-None-Match* (optional): If the "If-None-Match" header is given, then it must contain exactly one Etag. The document is returned, if it has a different revision than the given Etag. Otherwise an *HTTP 304* is returned.
- *If-Match* (optional): If the "If-Match" header is given, then it must contain exactly one Etag. The document is returned, if it has the same revision as the given Etag. Otherwise a *HTTP 412* is returned.

Returns the document identified by *document-handle*. The returned document contains three special attributes: *_id* containing the document handle, *_key* containing key which uniquely identifies a document in a given collection and *_rev* containing the revision.

Return Codes

- *200*: is returned if the document was found
- *304*: is returned if the "If-None-Match" header is given and the document has the same version
- *404*: is returned if the document or collection was not found
- *412*: is returned if an "If-Match" header is given and the found document has a different version. The response will also contain the found document's current revision in the *_rev* attribute. Additionally, the attributes *_id* and *_key* will be returned.

Examples

Use a document handle:

```
shell> curl --dump - http://localhost:8529/_api/document/products/10676

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: "_XnCMnMy--_"
```

show response body
Use a document handle and an Etag:

```
shell> curl --header 'If-None-Match: "_XnCMnRG--_"' --dump -
http://localhost:8529/_api/document/products/10724
```

Unknown document handle:

```
shell> curl --dump - http://localhost:8529/_api/document/products/unknownhandle

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Changes in 3.0 from 2.8:

The *rev* query parameter has been withdrawn. The same effect can be achieved with the *If-Match* HTTP header.

Read document header

reads a single document head

```
HEAD /_api/document/{document-handle}
```

Path Parameters

- *document-handle* (required): The handle of the document.

Header Parameters

- *If-None-Match* (optional): If the "If-None-Match" header is given, then it must contain exactly one Etag. If the current document revision is not equal to the specified Etag, an *HTTP 200* response is returned. If the current document revision is identical to the specified Etag, then an *HTTP 304* is returned.
- *If-Match* (optional): If the "If-Match" header is given, then it must contain exactly one Etag. The document is returned, if it has the same revision as the given Etag. Otherwise a *HTTP 412* is returned.

Like *GET*, but only returns the header fields and not the body. You can use this call to get the current revision of a document or check if the document was deleted.

Return Codes

- *200*: is returned if the document was found
- *304*: is returned if the "If-None-Match" header is given and the document has the same version
- *404*: is returned if the document or collection was not found
- *412*: is returned if an "If-Match" header is given and the found document has a different version. The response will also contain the found document's current revision in the *Etag* header.

Examples

```
shell> curl -X HEAD --dump - http://localhost:8529/_api/document/products/10715
```

Changes in 3.0 from 2.8:

The *rev* query parameter has been withdrawn. The same effect can be achieved with the *If-Match* HTTP header.

Read all documents

reads all documents from collection

```
PUT /_api/simple/all-keys
```

Query Parameters

- *collection* (optional): The name of the collection. **This parameter is only for an easier migration path from old versions.** In ArangoDB versions < 3.0, the URL path was */_api/document* and this was passed in via the query parameter "collection". This combination was removed. The collection name can be passed to */_api/simple/all-keys* as body parameter (preferred) or as query parameter.

A JSON object with these properties is required:

- **type**: The type of the result. The following values are allowed:

  - *id*: returns an array of document ids (*_id* attributes)
  - *key*: returns an array of document keys (*_key* attributes)
  - *path*: returns an array of document URI paths. This is the default.
- **collection**: The collection that should be queried

Returns an array of all keys, ids, or URI paths for all documents in the collection identified by *collection*. The type of the result array is determined by the *type* attribute.

Note that the results have no defined order and thus the order should not be relied on.

Return Codes

- *201*: All went well.
- *404*: The collection does not exist.

Examples

Return all document paths

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/all-keys
<<EOF
{
  "collection" : "products"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Return all document keys

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/all-keys
<<EOF
{
  "collection" : "products",
  "type" : "id"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Collection does not exist

```
shell> curl --dump - http://localhost:8529/_api/document/doesnotexist

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Changes in 3.0 from 2.8:

The collection name should now be specified in the URL path. The old way with the URL path */_api/document* and the required query parameter *collection* still works.

Create document

creates documents

```
POST /_api/document/{collection}
```

Path Parameters

- *collection* (required): The *collection* in which the collection is to be created.

Request Body (required)

A JSON representation of a single document or of an array of documents.

Query Parameters

- *collection* (optional): The name of the collection. This is only for backward compatibility. In ArangoDB versions < 3.0, the URL path was */_api/document* and this query parameter was required. This combination still works, but the recommended way is to specify the collection in the URL path.
- *waitForSync* (optional): Wait until document has been synced to disk.
- *returnNew* (optional): Additionally return the complete new document under the attribute *new* in the result.
- *silent* (optional): If set to *true*, an empty object will be returned as response. No meta-data will be returned for the created document. This option can be used to save some network traffic.

Creates a new document from the document given in the body, unless there is already a document with the *_key* given. If no *_key* is given, a new unique *_key* is generated automatically.

The body can be an array of documents, in which case all documents in the array are inserted with the same semantics as for a single document. The result body will contain a JSON array of the same length as the input array, and each entry contains the result of the operation for the corresponding input. In case of an error the entry is a document with attributes *error* set to *true* and errorCode set to the error code that has happened.

Possibly given *_id* and *_rev* attributes in the body are always ignored, the URL part or the query parameter collection respectively counts.

If the document was created successfully, then the *Location* header contains the path to the newly created document. The *Etag* header field contains the revision of the document. Both are only set in the single document case.

If *silent* is not set to *true*, the body of the response contains a JSON object (single document case) with the following attributes:

- *_id* contains the document handle of the newly created document
- *_key* contains the document key
- *_rev* contains the document revision

In the multi case the body is an array of such objects.

If the collection parameter *waitForSync* is *false*, then the call returns as soon as the document has been accepted. It will not wait until the documents have been synced to disk.

Optionally, the query parameter *waitForSync* can be used to force synchronization of the document creation operation to disk even in case that the *waitForSync* flag had been disabled for the entire collection. Thus, the *waitForSync* query parameter can be used to force synchronization of just this specific operations. To use this, set the *waitForSync* parameter to *true*. If the *waitForSync* parameter is not specified or set to *false*, then the collection's default *waitForSync* behavior is applied. The *waitForSync* query parameter cannot be used to disable synchronization for collections that have a default *waitForSync* value of *true*.

If the query parameter *returnNew* is *true*, then, for each generated document, the complete new document is returned under the *new* attribute in the result.

Return Codes

- *201*: is returned if the documents were created successfully and *waitForSync* was *true*.
- *202*: is returned if the documents were created successfully and *waitForSync* was *false*.
- *400*: is returned if the body does not contain a valid JSON representation of one document or an array of documents. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.
- *409*: is returned in the single document case if a document with the same qualifiers in an indexed attribute conflicts with an already existing document and thus violates that unique constraint. The response body contains an error document in this case. In the array case only 201 or 202 is returned, but if an error occurred, the additional HTTP header *X-Arango-Error-Codes* is set, which contains a map of the error codes that occurred together with their multiplicities, as in: *1205:10,1210:17* which means that in 10 cases the error 1205 "illegal document handle" and in 17 cases the error 1210 "unique constraint violated" has happened.

Examples

Create a document in a collection named *products*. Note that the revision identifier might or might not by equal to the auto-generated key.

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/document/products
<<EOF
{ "Hello": "World" }
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
etag: "_XnCMnHm--_"
location: /_db/_system/_api/document/products/10638
```

show response body

Create a document in a collection named *products* with a collection-level *waitForSync* value of *false*.

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/document/products
<<EOF
{ "Hello": "World" }
EOF

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
etag: "_XnCMnFa--_"
location: /_db/_system/_api/document/products/10626
```

show response body

Create a document in a collection with a collection-level *waitForSync* value of *false*, but using the *waitForSync* query parameter.

```
shell> curl -X POST --data-binary @- --dump -
http://localhost:8529/_api/document/products?waitForSync=true <<EOF
{ "Hello": "World" }
EOF
```

```
HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
etag: "_XnCMnLK--_"
location: /_db/_system/_api/document/products/10668
```

show response body

Unknown collection name

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/document/products
<<EOF
{ "Hello": "World" }
EOF

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Illegal document

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/document/products
<<EOF
{ 1: "World" }
EOF

HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Insert multiple documents:

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/document/products
<<EOF
[{"Hello":"Earth"}, {"Hello":"Venus"}, {"Hello":"Mars"}]
EOF

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

[
  {
    "_id" : "products/10646",
    "_key" : "10646",
    "_rev" : "_XnCMnJG--_"
  },
  {
    "_id" : "products/10650",
    "_key" : "10650",
    "_rev" : "_XnCMnJG--B"
```

```
  },
  {
    "_id" : "products/10652",
    "_key" : "10652",
    "_rev" : "_XnCMnJK--_"
  }
]
```

Use of returnNew:

```
shell> curl -X POST --data-binary @- --dump -
http://localhost:8529/_api/document/products?returnNew=true <<EOF
{"Hello":"World"}
EOF

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
etag: "_XnCMnKG--_"
location: /_db/_system/_api/document/products/10660
```

show response body

Changes in 3.0 from 2.8:

The collection name should now be specified in the URL path. The old way with the URL path */_api/document* and the required query parameter *collection* still works. The possibility to insert multiple documents with one operation is new and the query parameter *returnNew* has been added.

Replace document

replaces a document

`PUT /_api/document/{document-handle}`

Request Body (required)

A JSON representation of a single document.

Path Parameters

- *document-handle* (required): This URL parameter must be a document handle.

Query Parameters

- *waitForSync* (optional): Wait until document has been synced to disk.
- *ignoreRevs* (optional): By default, or if this is set to *true*, the *_rev* attributes in the given document is ignored. If this is set to *false*, then the *_rev* attribute given in the body document is taken as a precondition. The document is only replaced if the current revision is the one specified.
- *returnOld* (optional): Return additionally the complete previous revision of the changed document under the attribute *old* in the result.
- *returnNew* (optional): Return additionally the complete new document under the attribute *new* in the result.
- *silent* (optional): If set to *true*, an empty object will be returned as response. No meta-data will be returned for the replaced document. This option can be used to save some network traffic.

Header Parameters

- *If-Match* (optional): You can conditionally replace a document based on a target revision id by using the *if-match* HTTP header.

Replaces the document with handle with the one in the body, provided there is such a document and no precondition is violated.

If the *If-Match* header is specified and the revision of the document in the database is unequal to the given revision, the precondition is violated.

If *If-Match* is not given and *ignoreRevs* is *false* and there is a *_rev* attribute in the body and its value does not match the revision of the document in the database, the precondition is violated.

If a precondition is violated, an *HTTP 412* is returned.

If the document exists and can be updated, then an *HTTP 201* or an *HTTP 202* is returned (depending on *waitForSync*, see below), the *Etag* header field contains the new revision of the document and the *Location* header contains a complete URL under which the document can be queried.

Optionally, the query parameter *waitForSync* can be used to force synchronization of the document replacement operation to disk even in case that the *waitForSync* flag had been disabled for the entire collection. Thus, the *waitForSync* query parameter can be used to force synchronization of just specific operations. To use this, set the *waitForSync* parameter to *true*. If the *waitForSync* parameter is not specified or set to *false*, then the collection's default *waitForSync* behavior is applied. The *waitForSync* query parameter cannot be used to disable synchronization for collections that have a default *waitForSync* value of *true*.

If *silent* is not set to *true*, the body of the response contains a JSON object with the information about the handle and the revision. The attribute *_id* contains the known *document-handle* of the updated document, *_key* contains the key which uniquely identifies a document in a given collection, and the attribute *_rev* contains the new document revision.

If the query parameter *returnOld* is *true*, then the complete previous revision of the document is returned under the *old* attribute in the result.

If the query parameter *returnNew* is *true*, then the complete new document is returned under the *new* attribute in the result.

If the document does not exist, then a *HTTP 404* is returned and the body of the response contains an error document.

Return Codes

- *201*: is returned if the document was replaced successfully and *waitForSync* was *true*.
- *202*: is returned if the document was replaced successfully and *waitForSync* was *false*.
- *400*: is returned if the body does not contain a valid JSON representation of a document. The response body contains an error document in this case.
- *404*: is returned if the collection or the document was not found.
- *412*: is returned if the precondition was violated. The response will also contain the found documents' current revisions in the *_rev* attributes. Additionally, the attributes *_id* and *_key* will be returned.

Examples

Using a document handle

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/document/products/10733 <<EOF
{"Hello": "you"}
EOF

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
etag: "_XnCMnXm--B"
location: /_db/_system/_api/document/products/10733
```

show response body
Unknown document handle

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/document/products/10755 <<EOF
{}
EOF

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Produce a revision conflict

```
shell> curl -X PUT --header 'If-Match: "_XnCMnZS--B"' --data-binary @- --dump -
http://localhost:8529/_api/document/products/10743 <<EOF
{"other":"content"}
EOF

HTTP/1.1 412 Precondition Failed
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: "_XnCMnZS--_"
```

show response body

Changes in 3.0 from 2.8:

There are quite some changes in this in comparison to Version 2.8, but few break existing usage:

- the *rev* query parameter is gone (was duplication of If-Match)
- the *policy* query parameter is gone (was non-sensical)
- the *ignoreRevs* query parameter is new, the default *true* gives the traditional behavior as in 2.8
- the *returnNew* and *returnOld* query parameters are new

There should be very few changes to behavior happening in real-world situations or drivers. Essentially, one has to replace usage of the *rev* query parameter by usage of the *If-Match* header. The non-sensical combination of *If-Match* given and *policy=last* no longer works, but can easily be achieved by leaving out the *If-Match* header.

The collection name should now be specified in the URL path. The old way with the URL path */_api/document* and the required query parameter *collection* still works.

Replace documents

replaces multiple documents

`PUT /_api/document/{collection}`

Request Body (required)

A JSON representation of an array of documents.

Path Parameters

- *collection* (required): This URL parameter is the name of the collection in which the documents are replaced.

Query Parameters

- *waitForSync* (optional): Wait until the new documents have been synced to disk.

- *ignoreRevs* (optional): By default, or if this is set to *true*, the *_rev* attributes in the given documents are ignored. If this is set to *false*, then any *_rev* attribute given in a body document is taken as a precondition. The document is only replaced if the current revision is the one specified.
- *returnOld* (optional): Return additionally the complete previous revision of the changed documents under the attribute *old* in the result.
- *returnNew* (optional): Return additionally the complete new documents under the attribute *new* in the result.

Replaces multiple documents in the specified collection with the ones in the body, the replaced documents are specified by the *_key* attributes in the body documents.

If *ignoreRevs* is *false* and there is a *_rev* attribute in a document in the body and its value does not match the revision of the corresponding document in the database, the precondition is violated.

If the document exists and can be updated, then an *HTTP 201* or an *HTTP 202* is returned (depending on *waitForSync*, see below).

Optionally, the query parameter *waitForSync* can be used to force synchronization of the document replacement operation to disk even in case that the *waitForSync* flag had been disabled for the entire collection. Thus, the *waitForSync* query parameter can be used to force synchronization of just specific operations. To use this, set the *waitForSync* parameter to *true*. If the *waitForSync* parameter is not specified or set to *false*, then the collection's default *waitForSync* behavior is applied. The *waitForSync* query parameter cannot be used to disable synchronization for collections that have a default *waitForSync* value of *true*.

The body of the response contains a JSON array of the same length as the input array with the information about the handle and the revision of the replaced documents. In each entry, the attribute *_id* contains the known *document-handle* of each updated document, *_key* contains the key which uniquely identifies a document in a given collection, and the attribute *_rev* contains the new document revision. In case of an error or violated precondition, an error object with the attribute *error* set to *true* and the attribute *errorCode* set to the error code is built.

If the query parameter *returnOld* is *true*, then, for each generated document, the complete previous revision of the document is returned under the *old* attribute in the result.

If the query parameter *returnNew* is *true*, then, for each generated document, the complete new document is returned under the *new* attribute in the result.

Note that if any precondition is violated or an error occurred with some of the documents, the return code is still 201 or 202, but the additional HTTP header *X-Arango-Error-Codes* is set, which contains a map of the error codes that occurred together with their multiplicities, as in: *1200:17,1205:10* which means that in 17 cases the error 1200 "revision conflict" and in 10 cases the error 1205 "illegal document handle" has happened.

Return Codes

- *201*: is returned if the documents were replaced successfully and *waitForSync* was *true*.
- *202*: is returned if the documents were replaced successfully and *waitForSync* was *false*.
- *400*: is returned if the body does not contain a valid JSON representation of an array of documents. The response body contains an error document in this case.
- *404*: is returned if the collection was not found.

Changes in 3.0 from 2.8:

The multi document version is new in 3.0.

Update document

updates a document

```
PATCH /_api/document/{document-handle}
```

Request Body (required)

A JSON representation of a document update as an object.

Path Parameters

- *document-handle* (required): This URL parameter must be a document handle.

Query Parameters

- *keepNull* (optional): If the intention is to delete existing attributes with the patch command, the URL query parameter *keepNull* can be used with a value of *false*. This will modify the behavior of the patch command to remove any attributes from the existing document that are contained in the patch document with an attribute value of *null*.
- *mergeObjects* (optional): Controls whether objects (not arrays) will be merged if present in both the existing and the patch document. If set to *false*, the value in the patch document will overwrite the existing document's value. If set to *true*, objects will be merged. The default is *true*.
- *waitForSync* (optional): Wait until document has been synced to disk.
- *ignoreRevs* (optional): By default, or if this is set to *true*, the *_rev* attributes in the given document is ignored. If this is set to *false*, then the *_rev* attribute given in the body document is taken as a precondition. The document is only updated if the current revision is the one specified.
- *returnOld* (optional): Return additionally the complete previous revision of the changed document under the attribute *old* in the result.
- *returnNew* (optional): Return additionally the complete new document under the attribute *new* in the result.
- *silent* (optional): If set to *true*, an empty object will be returned as response. No meta-data will be returned for the updated document. This option can be used to save some network traffic.

Header Parameters

- *If-Match* (optional): You can conditionally update a document based on a target revision id by using the *if-match* HTTP header.

Partially updates the document identified by *document-handle*. The body of the request must contain a JSON document with the attributes to patch (the patch document). All attributes from the patch document will be added to the existing document if they do not yet exist, and overwritten in the existing document if they do exist there.

Setting an attribute value to *null* in the patch document will cause a value of *null* to be saved for the attribute by default.

If the *If-Match* header is specified and the revision of the document in the database is unequal to the given revision, the precondition is violated.

If *If-Match* is not given and *ignoreRevs* is *false* and there is a *_rev* attribute in the body and its value does not match the revision of the document in the database, the precondition is violated.

If a precondition is violated, an *HTTP 412* is returned.

If the document exists and can be updated, then an *HTTP 201* or an *HTTP 202* is returned (depending on *waitForSync*, see below), the *Etag* header field contains the new revision of the document (in double quotes) and the *Location* header contains a complete URL under which the document can be queried.

Optionally, the query parameter *waitForSync* can be used to force synchronization of the updated document operation to disk even in case that the *waitForSync* flag had been disabled for the entire collection. Thus, the *waitForSync* query parameter can be used to force synchronization of just specific operations. To use this, set the *waitForSync* parameter to *true*. If the *waitForSync* parameter is not specified or set to *false*, then the collection's default *waitForSync* behavior is applied. The *waitForSync* query parameter cannot be used to disable synchronization for collections that have a default *waitForSync* value of *true*.

If *silent* is not set to *true*, the body of the response contains a JSON object with the information about the handle and the revision. The attribute *_id* contains the known *document-handle* of the updated document, *_key* contains the key which uniquely identifies a document in a given collection, and the attribute *_rev* contains the new document revision.

If the query parameter *returnOld* is *true*, then the complete previous revision of the document is returned under the *old* attribute in the result.

If the query parameter *returnNew* is *true*, then the complete new document is returned under the *new* attribute in the result.

If the document does not exist, then a *HTTP 404* is returned and the body of the response contains an error document.

Return Codes

- *201*: is returned if the document was updated successfully and *waitForSync* was *true*.
- *202*: is returned if the document was updated successfully and *waitForSync* was *false*.
- *400*: is returned if the body does not contain a valid JSON representation of a document. The response body contains an error document in this case.
- *404*: is returned if the collection or the document was not found.
- *412*: is returned if the precondition was violated. The response will also contain the found documents' current revisions in the *_rev* attributes. Additionally, the attributes *_id* and *_key* will be returned.

Examples

Patches an existing document with new content.

```
shell> curl -X PATCH --data-binary @- --dump -
http://localhost:8529/_api/document/products/10595 <<EOF
{
  "hello" : "world"
}
EOF

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
etag: "_XnCMnDa--B"
location: /_db/_system/_api/document/products/10595
```

show response body

Merging attributes of an object using `mergeObjects`:

```
shell> curl --dump - http://localhost:8529/_api/document/products/10611

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: "_XnCMnEa--_"
```

show response body

Changes in 3.0 from 2.8:

There are quite some changes in this in comparison to Version 2.8, but few break existing usage:

- the *rev* query parameter is gone (was duplication of If-Match)
- the *policy* query parameter is gone (was non-sensical)
- the *ignoreRevs* query parameter is new, the default *true* gives the traditional behavior as in 2.8
- the *returnNew* and *returnOld* query parameters are new

There should be very few changes to behavior happening in real-world situations or drivers. Essentially, one has to replace usage of the *rev* query parameter by usage of the *If-Match* header. The non-sensical combination of *If-Match* given and *policy=last* no longer works, but can easily be achieved by leaving out the *If-Match* header.

The collection name should now be specified in the URL path. The old way with the URL path */_api/document* and the required query parameter *collection* still works.

Update documents

updates multiple documents

```
PATCH /_api/document/{collection}
```

Request Body (required)

A JSON representation of an array of document updates as objects.

Path Parameters

- *collection* (required): This URL parameter is the name of the collection in which the documents are updated.

Query Parameters

- *keepNull* (optional): If the intention is to delete existing attributes with the patch command, the URL query parameter *keepNull* can be used with a value of *false*. This will modify the behavior of the patch command to remove any attributes from the existing document that are contained in the patch document with an attribute value of *null*.
- *mergeObjects* (optional): Controls whether objects (not arrays) will be merged if present in both the existing and the patch document. If set to *false*, the value in the patch document will overwrite the existing document's value. If set to *true*, objects will be merged. The default is *true*.
- *waitForSync* (optional): Wait until the new documents have been synced to disk.
- *ignoreRevs* (optional): By default, or if this is set to *true*, the *_rev* attributes in the given documents are ignored. If this is set to *false*, then any *_rev* attribute given in a body document is taken as a precondition. The document is only updated if the current revision is the one specified.
- *returnOld* (optional): Return additionally the complete previous revision of the changed documents under the attribute *old* in the result.
- *returnNew* (optional): Return additionally the complete new documents under the attribute *new* in the result.

Partially updates documents, the documents to update are specified by the *_key* attributes in the body objects. The body of the request must contain a JSON array of document updates with the attributes to patch (the patch documents). All attributes from the patch documents will be added to the existing documents if they do not yet exist, and overwritten in the existing documents if they do exist there.

Setting an attribute value to *null* in the patch documents will cause a value of *null* to be saved for the attribute by default.

If *ignoreRevs* is *false* and there is a *_rev* attribute in a document in the body and its value does not match the revision of the corresponding document in the database, the precondition is violated.

If the document exists and can be updated, then an *HTTP 201* or an *HTTP 202* is returned (depending on *waitForSync*, see below).

Optionally, the query parameter *waitForSync* can be used to force synchronization of the document replacement operation to disk even in case that the *waitForSync* flag had been disabled for the entire collection. Thus, the *waitForSync* query parameter can be used to force synchronization of just specific operations. To use this, set the *waitForSync* parameter to *true*. If the *waitForSync* parameter is not specified or set to *false*, then the collection's default *waitForSync* behavior is applied. The *waitForSync* query parameter cannot be used to disable synchronization for collections that have a default *waitForSync* value of *true*.

The body of the response contains a JSON array of the same length as the input array with the information about the handle and the revision of the updated documents. In each entry, the attribute *_id* contains the known *document-handle* of each updated document, *_key* contains the key which uniquely identifies a document in a given collection, and the attribute *_rev* contains the new document revision. In case of an error or violated precondition, an error object with the attribute *error* set to *true* and the attribute *errorCode* set to the error code is built.

If the query parameter *returnOld* is *true*, then, for each generated document, the complete previous revision of the document is returned under the *old* attribute in the result.

If the query parameter *returnNew* is *true*, then, for each generated document, the complete new document is returned under the *new* attribute in the result.

Note that if any precondition is violated or an error occurred with some of the documents, the return code is still 201 or 202, but the additional HTTP header *X-Arango-Error-Codes* is set, which contains a map of the error codes that occurred together with their multiplicities, as in: *1200:17,1205:10* which means that in 17 cases the error 1200 "revision conflict" and in 10 cases the error 1205 "illegal document handle" has happened.

Return Codes

- *201*: is returned if the documents were updated successfully and *waitForSync* was *true*.
- *202*: is returned if the documents were updated successfully and *waitForSync* was *false*.
- *400*: is returned if the body does not contain a valid JSON representation of an array of documents. The response body contains an error document in this case.
- *404*: is returned if the collection was not found.

Changes in 3.0 from 2.8:

The multi document version is new in 3.0.

Removes a document

removes a document

`DELETE /_api/document/{document-handle}`

Path Parameters

- *document-handle* (required): Removes the document identified by *document-handle*.

Query Parameters

- *waitForSync* (optional): Wait until deletion operation has been synced to disk.
- *returnOld* (optional): Return additionally the complete previous revision of the changed document under the attribute *old* in the result.
- *silent* (optional): If set to *true*, an empty object will be returned as response. No meta-data will be returned for the removed document. This option can be used to save some network traffic.

Header Parameters

- *If-Match* (optional): You can conditionally remove a document based on a target revision id by using the *if-match* HTTP header.

If *silent* is not set to *true*, the body of the response contains a JSON object with the information about the handle and the revision. The attribute *_id* contains the known *document-handle* of the removed document, *_key* contains the key which uniquely identifies a document in a given collection, and the attribute *_rev* contains the document revision.

If the *waitForSync* parameter is not specified or set to *false*, then the collection's default *waitForSync* behavior is applied. The *waitForSync* query parameter cannot be used to disable synchronization for collections that have a default *waitForSync* value of *true*.

If the query parameter *returnOld* is *true*, then the complete previous revision of the document is returned under the *old* attribute in the result.

Return Codes

- *200*: is returned if the document was removed successfully and *waitForSync* was *true*.
- *202*: is returned if the document was removed successfully and *waitForSync* was *false*.
- *404*: is returned if the collection or the document was not found. The response body contains an error document in this case.
- *412*: is returned if a "If-Match" header or *rev* is given and the found document has a different version. The response will also contain the found document's current revision in the *_rev* attribute. Additionally, the attributes *_id* and *_key* will be returned.

Examples

Using document handle:

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/document/products/10529

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
```

```
x-content-type-options: nosniff
etag: "_XnCMm7a--_"
location: /_db/_system/_api/document/products/10529
```

show response body
Unknown document handle:

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/document/products/10573

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Revision conflict:

```
shell> curl -X DELETE --header 'If-Match: "_XnCMm9K--B"' --dump -
http://localhost:8529/_api/document/products/10539

HTTP/1.1 412 Precondition Failed
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: "_XnCMm9K--_"
```

show response body

Changes in 3.0 from 2.8:

There are only very few changes in this in comparison to Version 2.8:

- the *rev* query parameter is gone (was duplication of If-Match)
- the *policy* query parameter is gone (was non-sensical)
- the *returnOld* query parameter is new

There should be very few changes to behavior happening in real-world situations or drivers. Essentially, one has to replace usage of the *rev* query parameter by usage of the *If-Match* header. The non-sensical combination of *If-Match* given and *policy=last* no longer works, but can easily be achieved by leaving out the *If-Match* header.

Removes multiple documents

removes multiple document

`DELETE /_api/document/{collection}`

Request Body (required)

A JSON array of strings or documents.

Path Parameters

- *collection* (required): Collection from which documents are removed.

Query Parameters

- *waitForSync* (optional): Wait until deletion operation has been synced to disk.
- *returnOld* (optional): Return additionally the complete previous revision of the changed document under the attribute *old* in the result.

- *ignoreRevs* (optional): If set to *true*, ignore any *_rev* attribute in the selectors. No revision check is performed.

The body of the request is an array consisting of selectors for documents. A selector can either be a string with a key or a string with a document handle or an object with a *_key* attribute. This API call removes all specified documents from *collection*. If the selector is an object and has a *_rev* attribute, it is a precondition that the actual revision of the removed document in the collection is the specified one.

The body of the response is an array of the same length as the input array. For each input selector, the output contains a JSON object with the information about the outcome of the operation. If no error occurred, an object is built in which the attribute *_id* contains the known *document-handle* of the removed document, *_key* contains the key which uniquely identifies a document in a given collection, and the attribute *_rev* contains the document revision. In case of an error, an object with the attribute *error* set to *true* and *errorCode* set to the error code is built.

If the *waitForSync* parameter is not specified or set to *false*, then the collection's default *waitForSync* behavior is applied. The *waitForSync* query parameter cannot be used to disable synchronization for collections that have a default *waitForSync* value of *true*.

If the query parameter *returnOld* is *true*, then the complete previous revision of the document is returned under the *old* attribute in the result.

Note that if any precondition is violated or an error occurred with some of the documents, the return code is still 200 or 202, but the additional HTTP header *X-Arango-Error-Codes* is set, which contains a map of the error codes that occurred together with their multiplicities, as in: *1200:17,1205:10* which means that in 17 cases the error 1200 "revision conflict" and in 10 cases the error 1205 "illegal document handle" has happened.

Return Codes

- *200*: is returned if *waitForSync* was *true*.
- *202*: is returned if *waitForSync* was *false*.
- *404*: is returned if the collection was not found. The response body contains an error document in this case.

Examples

Using document handle:

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/document/products/10563

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
etag: "_XnCMn_S--_"
location: /_db/_system/_api/document/products/10563
```

show response body
Unknown document handle:

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/document/products/10584

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Revision conflict:

```
shell> curl -X DELETE --header 'If-Match: "_XnCMn-O--B"' --dump -
http://localhost:8529/_api/document/products/10551

HTTP/1.1 412 Precondition Failed
```

```
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: "_XnCMn-O--_"
```

show response body

Changes in 3.0 from 2.8:

This variant is new in 3.0. Note that it requires a body in the DELETE request.

HTTP Interface for Edges

This is an introduction to ArangoDB's REST interface for edges.

ArangoDB offers graph functionality; Edges are one part of that.

Address and Etag of an Edge

All documents in ArangoDB have a document handle. This handle uniquely identifies a document. Any document can be retrieved using its unique URI:

```
http://server:port/_api/document/<document-handle>
```

Edges are a special variation of documents. To access an edge use the same URL format as for a document:

```
http://server:port/_api/document/<document-handle>
```

For example, assumed that the document handle, which is stored in the *_id* attribute of the edge, is *demo/362549736*, then the URL of that edge is:

```
http://localhost:8529/_api/document/demo/362549736
```

The above URL scheme does not specify a database name explicitly, so the default database will be used. To explicitly specify the database context, use the following URL schema:

```
http://server:port/_db/<database-name>/_api/document/<document-handle>
```

Example:

```
http://localhost:8529/_db/mydb/_api/document/demo/362549736
```

Note: that the following examples use the short URL format for brevity.

Working with Edges using REST

This is documentation to ArangoDB's REST interface for edges.

Edges are documents with two additional attributes: *_from* and *_to*. These attributes are mandatory and must contain the document-handle of the from and to vertices of an edge.

Use the general document REST api for create/read/update/delete.

Read in- or outbound edges

get edges

`GET /_api/edges/{collection-id}`

Path Parameters

- *collection-id* (required): The id of the collection.

Query Parameters

- *vertex* (required): The id of the start vertex.
- *direction* (optional): Selects *in* or *out* direction for edges. If not set, any edges are returned.

Returns an array of edges starting or ending in the vertex identified by *vertex-handle*.

Return Codes

- *200*: is returned if the edge collection was found and edges were retrieved.
- *400*: is returned if the request contains invalid parameters.
- *404*: is returned if the edge collection was not found.

Examples

Any direction

```
shell> curl --dump - http://localhost:8529/_api/edges/edges?vertex=vertices/1

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
In edges

```
shell> curl --dump - http://localhost:8529/_api/edges/edges?vertex=vertices/1&direction=in

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Out edges

```
shell> curl --dump - http://localhost:8529/_api/edges/edges?
vertex=vertices/1&direction=out
```

```
HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

General Graphs

This chapter describes the REST interface for the multi-collection graph module. It allows you to define a graph that is spread across several edge and document collections. There is no need to include the referenced collections within the query, this module will handle it for you.

Manage your graphs

The graph module provides functions dealing with graph structures. Examples will explain the REST API on the social graph:

List all graphs

Lists all graphs known to the graph module.

`GET /_api/gharial`

Lists all graph names stored in this database.

Return Codes

- *200*: Is returned if the module is available and the graphs could be listed.

Examples

```
shell> curl --dump - http://localhost:8529/_api/gharial

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Create a graph

Create a new graph in the graph module.

`POST /_api/gharial`

The creation of a graph requires the name of the graph and a definition of its edges. See also edge definitions.

A JSON object with these properties is required:

- **orphanCollections**: An array of additional vertex collections.
- **edgeDefinitions**: An array of definitions for the edge
- **name**: Name of the graph.
- **isSmart**: Define if the created graph should be smart. This only has effect in Enterprise version.
- **options**:
  - **smartGraphAttribute**: The attribute name that is used to smartly shard the vertices of a graph. Every vertex in this Graph has to have this attribute. Cannot be modified later.
  - **numberOfShards**: The number of shards that is used for every collection within this graph. Cannot be modified later.

Return Codes

- *201*: Is returned if the graph could be created and waitForSync is enabled for the `_graphs` collection. The response body contains the graph configuration that has been stored.
- *202*: Is returned if the graph could be created and waitForSync is disabled for the `_graphs` collection. The response body contains the graph configuration that has been stored.
- *409*: Returned if there is a conflict storing the graph. This can occur either if a graph with this name is already stored, or if there is one edge definition with a the same edge collection but a different signature used in any other graph.

Examples

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/gharial <<EOF
```

```
{
  "name" : "myGraph",
  "edgeDefinitions" : [
    {
      "collection" : "edges",
      "from" : [
        "startVertices"
      ],
      "to" : [
        "endVertices"
      ]
    }
  ]
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMhNa--_
```

show response body

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/gharial <<EOF
{
  "name" : "myGraph",
  "edgeDefinitions" : [
    {
      "collection" : "edges",
      "from" : [
        "startVertices"
      ],
      "to" : [
        "endVertices"
      ]
    }
  ],
  "isSmart" : true,
  "options" : {
    "numberOfShards" : 9,
    "smartGraphAttribute" : "region"
  }
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMhQi--_
```

show response body

Get a graph

Get a graph from the graph module.

`GET /_api/gharial/{graph-name}`

Gets a graph from the collection *_graphs*. Returns the definition content of this graph.

Path Parameters

- *graph-name* (required): The name of the graph.

Return Codes

- *200*: Returned if the graph could be found.
- *404*: Returned if no graph with this name could be found.

Examples

```
shell> curl --dump - http://localhost:8529/_api/gharial/myGraph

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMhkS--_
```

show response body

Drop a graph

delete an existing graph

`DELETE /_api/gharial/{graph-name}`

Removes a graph from the collection *_graphs*.

Path Parameters

- *graph-name* (required): The name of the graph.

Query Parameters

- *dropCollections* (optional): Drop collections of this graph as well. Collections will only be dropped if they are not used in other graphs.

Return Codes

- *201*: Is returned if the graph could be dropped and waitForSync is enabled for the `_graphs` collection.
- *202*: Returned if the graph could be dropped and waitForSync is disabled for the `_graphs` collection.
- *404*: Returned if no graph with this name could be found.

Examples

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/gharial/social?
dropCollections=true

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

List vertex collections

Lists all vertex collections used in this graph.

`GET /_api/gharial/{graph-name}/vertex`

Lists all vertex collections within this graph.

Path Parameters

- *graph-name* (required): The name of the graph.

Return Codes

- *200*: Is returned if the collections could be listed.
- *404*: Returned if no graph with this name could be found.

Examples

```
shell> curl --dump - http://localhost:8529/_api/gharial/social/vertex

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Add vertex collection

Add an additional vertex collection to the graph.

`POST /_api/gharial/{graph-name}/vertex`

Adds a vertex collection to the set of collections of the graph. If the collection does not exist, it will be created.

Path Parameters

- *graph-name* (required): The name of the graph.

Return Codes

- *201*: Returned if the edge collection could be added successfully and waitForSync is true.
- *202*: Returned if the edge collection could be added successfully and waitForSync is false.
- *404*: Returned if no graph with this name could be found.

Examples

```
shell> curl -X POST --data-binary @- --dump -
http://localhost:8529/_api/gharial/social/vertex <<EOF
{
  "collection" : "otherVertices"
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMhI6--_
```

show response body

Remove vertex collection

Remove a vertex collection form the graph.

`DELETE /_api/gharial/{graph-name}/vertex/{collection-name}`

Removes a vertex collection from the graph and optionally deletes the collection, if it is not used in any other graph.

Path Parameters

- *graph-name* (required): The name of the graph.
- *collection-name* (required): The name of the vertex collection.

Query Parameters

- *dropCollection* (optional): Drop the collection as well. Collection will only be dropped if it is not used in other graphs.

Return Codes

- *201*: Returned if the vertex collection was removed from the graph successfully and waitForSync is true.
- *202*: Returned if the request was successful but waitForSync is false.
- *400*: Returned if the vertex collection is still used in an edge definition. In this case it cannot be removed from the graph yet, it has to be removed from the edge definition first.
- *404*: Returned if no graph with this name could be found.

Examples

You can remove vertex collections that are not used in any edge collection:

```
shell> curl -X DELETE --dump -
http://localhost:8529/_api/gharial/social/vertex/otherVertices

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMiHC--_
```

show response body

You cannot remove vertex collections that are used in edge collections:

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/gharial/social/vertex/male

HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

List edge definitions

Lists all edge definitions

`GET /_api/gharial/{graph-name}/edge`

Lists all edge collections within this graph.

Path Parameters

- *graph-name* (required): The name of the graph.

Return Codes

- *200*: Is returned if the edge definitions could be listed.
- *404*: Returned if no graph with this name could be found.

Examples

```
shell> curl --dump - http://localhost:8529/_api/gharial/social/edge

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Add edge definition

Add a new edge definition to the graph

`POST /_api/gharial/{graph-name}/edge`

Adds an additional edge definition to the graph.

This edge definition has to contain a *collection* and an array of each *from* and *to* vertex collections. An edge definition can only be added if this definition is either not used in any other graph, or it is used with exactly the same definition. It is not possible to store a definition "e" from "v1" to "v2" in the one graph, and "e" from "v2" to "v1" in the other graph.

A JSON object with these properties is required:

- **to** (string): One or many vertex collections that can contain target vertices.
- **from** (string): One or many vertex collections that can contain source vertices.
- **collection**: The name of the edge collection to be used.

Path Parameters

- *graph-name* (required): The name of the graph.

Return Codes

- *201*: Returned if the definition could be added successfully and waitForSync is enabled for the `_graphs` collection.
- *202*: Returned if the definition could be added successfully and waitForSync is disabled for the `_graphs` collection.
- *400*: Returned if the defininition could not be added, the edge collection is used in an other graph with a different signature.
- *404*: Returned if no graph with this name could be found.

Examples

```
shell> curl -X POST --data-binary @- --dump -
http://localhost:8529/_api/gharial/social/edge <<EOF
{
  "collection" : "works_in",
  "from" : [
    "female",
    "male"
  ],
  "to" : [
    "city"
  ]
}
EOF
```

```
HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMhAK--_
```

show response body

Replace an edge definition

Replace an existing edge definition

`PUT /_api/gharial/{graph-name}/edge/{definition-name}`

Change one specific edge definition. This will modify all occurrences of this definition in all graphs known to your database.

A JSON object with these properties is required:

- **to** (string): One or many vertex collections that can contain target vertices.
- **from** (string): One or many vertex collections that can contain source vertices.
- **collection**: The name of the edge collection to be used.

Path Parameters

- *graph-name* (required): The name of the graph.
- *definition-name* (required): The name of the edge collection used in the definition.

Return Codes

- *201*: Returned if the request was successful and waitForSync is true.
- *202*: Returned if the request was successful but waitForSync is false.
- *400*: Returned if no edge definition with this name is found in the graph.
- *404*: Returned if no graph with this name could be found.

Examples

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/gharial/social/edge/relation <<EOF
{
  "collection" : "relation",
  "from" : [
    "female",
    "male",
    "animal"
  ],
  "to" : [
    "female",
    "male",
    "animal"
  ]
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMiQW--_
```

show response body

Remove an edge definition from the graph

Remove an edge definition form the graph

`DELETE /_api/gharial/{graph-name}/edge/{definition-name}`

Remove one edge definition from the graph. This will only remove the edge collection, the vertex collections remain untouched and can still be used in your queries.

Path Parameters

- *graph-name* (required): The name of the graph.
- *definition-name* (required): The name of the edge collection used in the definition.

Query Parameters

- *dropCollection* (optional): Drop the collection as well. Collection will only be dropped if it is not used in other graphs.

Return Codes

- *201*: Returned if the edge definition could be removed from the graph and waitForSync is true.
- *202*: Returned if the edge definition could be removed from the graph and waitForSync is false.
- *400*: Returned if no edge definition with this name is found in the graph.
- *404*: Returned if no graph with this name could be found.

Examples

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/gharial/social/edge/relation

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMhd2--_
```

show response body

Handling Vertices

Examples will explain the REST API to the graph module on the social graph:

Create a vertex

create a new vertex

`POST /_api/gharial/{graph-name}/vertex/{collection-name}`

Adds a vertex to the given collection.

Path Parameters

- *graph-name* (required): The name of the graph.
- *collection-name* (required): The name of the vertex collection the vertex belongs to.

Query Parameters

- *waitForSync* (optional): Define if the request should wait until synced to disk.

Request Body (required)

The body has to be the JSON object to be stored.

Return Codes

- *201*: Returned if the vertex could be added and waitForSync is true.
- *202*: Returned if the request was successful but waitForSync is false.
- *404*: Returned if no graph or no vertex collection with this name could be found.

Examples

```
shell> curl -X POST --data-binary @- --dump -
http://localhost:8529/_api/gharial/social/vertex/male <<EOF
{
  "name" : "Francis"
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMhFu--_
```

show response body

Get a vertex

fetches an existing vertex

`GET /_api/gharial/{graph-name}/vertex/{collection-name}/{vertex-key}`

Gets a vertex from the given collection.

Path Parameters

- *graph-name* (required): The name of the graph.

- *collection-name* (required): The name of the vertex collection the vertex belongs to.
- *vertex-key* (required): The *_key* attribute of the vertex.

Header Parameters

- *if-match* (optional): If the "If-Match" header is given, then it must contain exactly one Etag. The document is returned, if it has the same revision as the given Etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the Etag in an attribute rev in the URL.

Return Codes

- *200*: Returned if the vertex could be found.
- *404*: Returned if no graph with this name, no vertex collection or no vertex with this id could be found.
- *412*: Returned if if-match header is given, but the documents revision is different.

Examples

```
shell> curl --dump - http://localhost:8529/_api/gharial/social/vertex/female/alice

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMhnm--_
```

show response body

Modify a vertex

replace an existing vertex

`PATCH /_api/gharial/{graph-name}/vertex/{collection-name}/{vertex-key}`

Updates the data of the specific vertex in the collection.

Path Parameters

- *graph-name* (required): The name of the graph.
- *collection-name* (required): The name of the vertex collection the vertex belongs to.
- *vertex-key* (required): The *_key* attribute of the vertex.

Query Parameters

- *waitForSync* (optional): Define if the request should wait until synced to disk.
- *keepNull* (optional): Define if values set to null should be stored. By default the key is not removed from the document.

Header Parameters

- *if-match* (optional): If the "If-Match" header is given, then it must contain exactly one Etag. The document is updated, if it has the same revision as the given Etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the Etag in an attribute rev in the URL.

Request Body (required)

The body has to contain a JSON object containing exactly the attributes that should be replaced.

Return Codes

- *200*: Returned if the vertex could be updated.
- *202*: Returned if the request was successful but waitForSync is false.

- *404*: Returned if no graph with this name, no vertex collection or no vertex with this id could be found.
- *412*: Returned if if-match header is given, but the documents revision is different.

Examples

```
shell> curl -X PATCH --data-binary @- --dump -
http://localhost:8529/_api/gharial/social/vertex/female/alice <<EOF
{
  "age" : 26
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMh8q--_
```

show response body

Replace a vertex

replaces an existing vertex

`PUT /_api/gharial/{graph-name}/vertex/{collection-name}/{vertex-key}`

Replaces the data of a vertex in the collection.

Path Parameters

- *graph-name* (required): The name of the graph.
- *collection-name* (required): The name of the vertex collection the vertex belongs to.
- *vertex-key* (required): The *_key* attribute of the vertex.

Query Parameters

- *waitForSync* (optional): Define if the request should wait until synced to disk.

Header Parameters

- *if-match* (optional): If the "If-Match" header is given, then it must contain exactly one Etag. The document is updated, if it has the same revision as the given Etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the Etag in an attribute rev in the URL.

Request Body (required)

The body has to be the JSON object to be stored.

Return Codes

- *200*: Returned if the vertex could be replaced.
- *202*: Returned if the request was successful but waitForSync is false.
- *404*: Returned if no graph with this name, no vertex collection or no vertex with this id could be found.
- *412*: Returned if if-match header is given, but the documents revision is different.

Examples

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/gharial/social/vertex/female/alice <<EOF
{
```

```
  "name" : "Alice Cooper",
  "age" : 26
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMiXu--_
```

show response body

Remove a vertex

removes a vertex from a graph

`DELETE /_api/gharial/{graph-name}/vertex/{collection-name}/{vertex-key}`

Removes a vertex from the collection.

Path Parameters

- *graph-name* (required): The name of the graph.
- *collection-name* (required): The name of the vertex collection the vertex belongs to.
- *vertex-key* (required): The *_key* attribute of the vertex.

Query Parameters

- *waitForSync* (optional): Define if the request should wait until synced to disk.

Header Parameters

- *if-match* (optional): If the "If-Match" header is given, then it must contain exactly one Etag. The document is updated, if it has the same revision as the given Etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the Etag in an attribute rev in the URL.

Return Codes

- *200*: Returned if the vertex could be removed.
- *202*: Returned if the request was successful but waitForSync is false.
- *404*: Returned if no graph with this name, no vertex collection or no vertex with this id could be found.
- *412*: Returned if if-match header is given, but the documents revision is different.

Examples

```
shell> curl -X DELETE --dump -
http://localhost:8529/_api/gharial/social/vertex/female/alice

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Handling Edges

Examples will explain the REST API for manipulating edges of the graph module on the knows graph:

Create an edge

Creates an edge in an existing graph

`POST /_api/gharial/{graph-name}/edge/{collection-name}`

Creates a new edge in the collection. Within the body the has to contain a *_from* and *_to* value referencing to valid vertices in the graph. Furthermore the edge has to be valid in the definition of this edge collection.

Path Parameters

- *graph-name* (required): The name of the graph.
- *collection-name* (required): The name of the edge collection the edge belongs to.

Query Parameters

- *waitForSync* (optional): Define if the request should wait until synced to disk.
- *_from* (required):
- *_to* (required):

Request Body (required)

The body has to be the JSON object to be stored.

Return Codes

- *201*: Returned if the edge could be created.
- *202*: Returned if the request was successful but waitForSync is false.
- *404*: Returned if no graph with this name, no edge collection or no edge with this id could be found.

Examples

```
shell> curl -X POST --data-binary @- --dump -
http://localhost:8529/_api/gharial/social/edge/relation <<EOF
{
  "type" : "friend",
  "_from" : "female/alice",
  "_to" : "female/diana"
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMg8y--_
```

show response body

Get an edge

fetch an edge

`GET /_api/gharial/{graph-name}/edge/{collection-name}/{edge-key}`

Gets an edge from the given collection.

Path Parameters

- *graph-name* (required): The name of the graph.
- *collection-name* (required): The name of the edge collection the edge belongs to.
- *edge-key* (required): The *_key* attribute of the vertex.

Header Parameters

- *if-match* (optional): If the "If-Match" header is given, then it must contain exactly one Etag. The document is returned, if it has the same revision as the given Etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the Etag in an attribute rev in the URL.

Return Codes

- *200*: Returned if the edge could be found.
- *404*: Returned if no graph with this name, no edge collection or no edge with this id could be found.
- *412*: Returned if if-match header is given, but the documents revision is different.

Examples

```
shell> curl --dump - http://localhost:8529/_api/gharial/social/edge/relation/8617

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMhhK--L
```

show response body

Examples will explain the API on the social graph:

Modify an edge

modify an existing edge

`PATCH /_api/gharial/{graph-name}/edge/{collection-name}/{edge-key}`

Updates the data of the specific edge in the collection.

Path Parameters

- *graph-name* (required): The name of the graph.
- *collection-name* (required): The name of the edge collection the edge belongs to.
- *edge-key* (required): The *_key* attribute of the vertex.

Query Parameters

- *waitForSync* (optional): Define if the request should wait until synced to disk.
- *keepNull* (optional): Define if values set to null should be stored. By default the key is not removed from the document.

Request Body (required)

The body has to be a JSON object containing the attributes to be updated.

Return Codes

- *200*: Returned if the edge could be updated.
- *202*: Returned if the request was successful but waitForSync is false.
- *404*: Returned if no graph with this name, no edge collection or no edge with this id could be found.

Examples

```
shell> curl -X PATCH --data-binary @- --dump -
http://localhost:8529/_api/gharial/social/edge/relation/9173 <<EOF
{
  "since" : "01.01.2001"
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMiAG--_
```

show response body

Replace an edge

replace the content of an existing edge

`PUT /_api/gharial/{graph-name}/edge/{collection-name}/{edge-key}`

Replaces the data of an edge in the collection.

Path Parameters

- *graph-name* (required): The name of the graph.
- *collection-name* (required): The name of the edge collection the edge belongs to.
- *edge-key* (required): The *_key* attribute of the vertex.

Query Parameters

- *waitForSync* (optional): Define if the request should wait until synced to disk.

Header Parameters

- *if-match* (optional): If the "If-Match" header is given, then it must contain exactly one Etag. The document is updated, if it has the same revision as the given Etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the Etag in an attribute rev in the URL.

Request Body (required)

The body has to be the JSON object to be stored.

Return Codes

- *201*: Returned if the request was successful but waitForSync is true.
- *202*: Returned if the request was successful but waitForSync is false.
- *404*: Returned if no graph with this name, no edge collection or no edge with this id could be found.
- *412*: Returned if if-match header is given, but the documents revision is different.

Examples

```
shell> curl -X PUT --data-binary @- --dump -
```

```
http://localhost:8529/_api/gharial/social/edge/relation/9247 <<EOF
{
  "type" : "divorced",
  "_from" : "female/alice",
  "_to" : "male/bob"
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
content-type: application/json; charset=utf-8
etag: _XnCMiDe--_
```

show response body

Remove an edge

removes an edge from graph

`DELETE /_api/gharial/{graph-name}/edge/{collection-name}/{edge-key}`

Removes an edge from the collection.

Path Parameters

- *graph-name* (required): The name of the graph.
- *collection-name* (required): The name of the edge collection the edge belongs to.
- *edge-key* (required): The *_key* attribute of the vertex.

Query Parameters

- *waitForSync* (optional): Define if the request should wait until synced to disk.

Header Parameters

- *if-match* (optional): If the "If-Match" header is given, then it must contain exactly one Etag. The document is updated, if it has the same revision as the given Etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the Etag in an attribute rev in the URL.

Return Codes

- *200*: Returned if the edge could be removed.
- *202*: Returned if the request was successful but waitForSync is false.
- *404*: Returned if no graph with this name, no edge collection or no edge with this id could be found.
- *412*: Returned if if-match header is given, but the documents revision is different.

Examples

```
shell> curl -X DELETE --dump -
http://localhost:8529/_api/gharial/social/edge/relation/8337

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

HTTP Interface for Traversals

Traversals

ArangoDB's graph traversals are executed on the server. Traversals can be initiated by clients by sending the traversal description for execution to the server.

Traversals in ArangoDB are used to walk over a graph stored in one edge collection. It can easily be described which edges of the graph should be followed and which actions should be performed on each visited vertex. Furthermore the ordering of visiting the nodes can be specified, for instance depth-first or breadth-first search are offered.

Executing Traversals via HTTP

executes a traversal

execute a server-side traversal

`POST /_api/traversal`

Starts a traversal starting from a given vertex and following. edges contained in a given edgeCollection. The request must contain the following attributes.

A JSON object with these properties is required:

- **sort**: body (JavaScript) code of a custom comparison function for the edges. The signature of this function is *(l, r) -> integer* (where l and r are edges) and must return -1 if l is smaller than, +1 if l is greater than, and 0 if l and r are equal. The reason for this is the following: The order of edges returned for a certain vertex is undefined. This is because there is no natural order of edges for a vertex with multiple connected edges. To explicitly define the order in which edges on the vertex are followed, you can specify an edge comparator function with this attribute. Note that the value here has to be a string to conform to the JSON standard, which in turn is parsed as function body on the server side. Furthermore note that this attribute is only used for the standard expanders. If you use your custom expander you have to do the sorting yourself within the expander code.
- **direction**: direction for traversal
  - *if set*, must be either *"outbound"*, *"inbound"*, or *"any"*
  - *if not set*, the *expander* attribute must be specified
- **minDepth**: ANDed with any existing filters): visits only nodes in at least the given depth
- **startVertex**: id of the startVertex, e.g. *"users/foo"*.
- **visitor**: body (JavaScript) code of custom visitor function function signature: *(config, result, vertex, path, connected) -> void* The visitor function can do anything, but its return value is ignored. To populate a result, use the *result* variable by reference. Note that the *connected* argument is only populated when the *order* attribute is set to *"preorder-expander"*.
- **itemOrder**: item iteration order can be *"forward"* or *"backward"*
- **strategy**: traversal strategy can be *"depthfirst"* or *"breadthfirst"*
- **filter**: default is to include all nodes: body (JavaScript code) of custom filter function function signature: *(config, vertex, path) -> mixed* can return four different string values:
  - *"exclude"* -> this vertex will not be visited.
  - *"prune"* -> the edges of this vertex will not be followed.
  - *""* or *undefined* -> visit the vertex and follow its edges.
  - *Array* -> containing any combination of the above. If there is at least one *"exclude"* or *"prune"* respectivly is contained, it's effect will occur.
- **init**: body (JavaScript) code of custom result initialization function function signature: *(config, result) -> void* initialize any values in result with what is required
- **maxIterations**: Maximum number of iterations in each traversal. This number can be set to prevent endless loops in traversal of cyclic graphs. When a traversal performs as many iterations as the *maxIterations* value, the traversal will abort with an error. If *maxIterations* is not set, a server-defined value may be used.
- **maxDepth**: ANDed with any existing filters visits only nodes in at most the given depth
- **uniqueness**: specifies uniqueness for vertices and edges visited. If set, must be an object like this: `"uniqueness": {"vertices": "none"|"global"|"path", "edges": "none"|"global"|"path"}`

- **order**: traversal order can be *"preorder"*, *"postorder"* or *"preorder-expander"*
- **graphName**: name of the graph that contains the edges. Either *edgeCollection* or *graphName* has to be given. In case both values are set the *graphName* is prefered.
- **expander**: body (JavaScript) code of custom expander function *must* be set if *direction* attribute is **not** set function signature: *(config, vertex, path) -> array* expander must return an array of the connections for *vertex* each connection is an object with the attributes *edge* and *vertex*
- **edgeCollection**: name of the collection that contains the edges.

If the Traversal is successfully executed *HTTP 200* will be returned. Additionally the *result* object will be returned by the traversal.

For successful traversals, the returned JSON object has the following properties:

- *error*: boolean flag to indicate if an error occurred (*false* in this case)
- *code*: the HTTP status code
- *result*: the return value of the traversal

If the traversal specification is either missing or malformed, the server will respond with *HTTP 400*.

The body of the response will then contain a JSON object with additional error details. The object has the following attributes:

- *error*: boolean flag to indicate that an error occurred (*true* in this case)
- *code*: the HTTP status code
- *errorNum*: the server error number
- *errorMessage*: a descriptive error message

Return Codes

- *200*: If the traversal is fully executed *HTTP 200* will be returned.
- *400*: If the traversal specification is either missing or malformed, the server will respond with *HTTP 400*.
- *404*: The server will responded with *HTTP 404* if the specified edge collection does not exist, or the specified start vertex cannot be found.
- *500*: The server will responded with *HTTP 500* when an error occurs inside the traversal or if a traversal performs more than *maxIterations* iterations.

Examples

In the following examples the underlying graph will contain five persons *Alice*, *Bob*, *Charlie*, *Dave* and *Eve*. We will have the following directed relations:

- *Alice* knows *Bob*
- *Bob* knows *Charlie*
- *Bob* knows *Dave*
- *Eve* knows *Alice*
- *Eve* knows *Bob*

The starting vertex will always be Alice.

Follow only outbound edges

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "outbound"
}
EOF

HTTP/1.1 200 OK
```

```
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body

Follow only inbound edges

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "inbound"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body

Follow any direction of edges

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "any",
  "uniqueness" : {
    "vertices" : "none",
    "edges" : "global"
  }
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body

Excluding *Charlie* and *Bob*

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "outbound",
  "filter" : "if (vertex.name === \"Bob\" ||     vertex.name === \"Charlie\") {  return
\"exclude\";}return;"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
```

```
x-content-type-options: nosniff
```

show response body

Do not follow edges from *Bob*

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "outbound",
  "filter" : "if (vertex.name === \"Bob\") {return \"prune\";}return;"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Visit only nodes in a depth of at least 2

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "outbound",
  "minDepth" : 2
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Visit only nodes in a depth of at most 1

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "outbound",
  "maxDepth" : 1
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Using a visitor function to return vertex ids only

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "outbound",
  "visitor" : "result.visited.vertices.push(vertex._id);"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Count all visited nodes and return a list of nodes only

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "outbound",
  "init" : "result.visited = 0; result.myVertices = [ ];",
  "visitor" : "result.visited++; result.myVertices.push(vertex);"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Expand only inbound edges of *Alice* and outbound edges of *Eve*

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "expander" : "var connections = [ ];if (vertex.name === \"Alice\")
{config.datasource.getInEdges(vertex).forEach(function (e) {connections.push({ vertex:
require(\"internal\").db._document(e._from), edge: e});});}if (vertex.name === \"Eve\")
{config.datasource.getOutEdges(vertex).forEach(function (e) {connections.push({vertex:
require(\"internal\").db._document(e._to), edge: e});});}return connections;"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Follow the *depthfirst* strategy

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "any",
  "strategy" : "depthfirst"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Using *postorder* ordering

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "any",
  "order" : "postorder"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Using *backward* item-ordering:

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "any",
  "itemOrder" : "backward"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Edges should only be included once globally, but nodes are included every time they are visited

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
```

```
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "any",
  "uniqueness" : {
    "vertices" : "none",
    "edges" : "global"
  }
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

If the underlying graph is cyclic, *maxIterations* should be set

The underlying graph has two vertices *Alice* and *Bob*. With the directed edges:

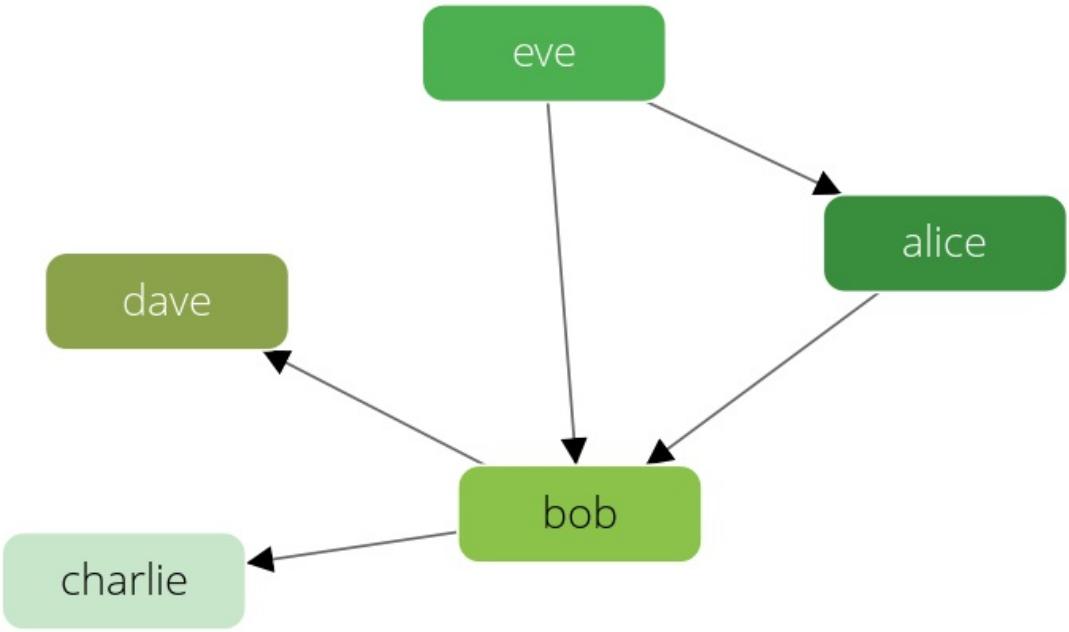
- *Alice* knows *Bob*
- *Bob* knows *Alice*

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/traversal <<EOF
{
  "startVertex" : "persons/alice",
  "graphName" : "knows_graph",
  "direction" : "any",
  "uniqueness" : {
    "vertices" : "none",
    "edges" : "none"
  },
  "maxIterations" : 5
}
EOF

HTTP/1.1 500 Internal Server Error
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

All examples were using this graph:


eve
alice
dave
bob
charlie

HTTP Interface for AQL Query Cursors

Database Cursors

This is an introduction to ArangoDB's HTTP Interface for Queries. Results of AQL and simple queries are returned as cursors in order to batch the communication between server and client. Each call returns a number of documents in a batch and an indication if the current batch has been the final batch. Depending on the query, the total number of documents in the result set might or might not be known in advance. In order to free server resources the client should delete the cursor as soon as it is no longer needed.

To execute a query, the query details need to be shipped from the client to the server via an HTTP POST request.

Retrieving query results

Select queries are executed on-the-fly on the server and the result set will be returned back to the client.

There are two ways the client can get the result set from the server:

- In a single roundtrip
- Using a cursor

Single roundtrip

The server will only transfer a certain number of result documents back to the client in one roundtrip. This number is controllable by the client by setting the *batchSize* attribute when issuing the query.

If the complete result can be transferred to the client in one go, the client does not need to issue any further request. The client can check whether it has retrieved the complete result set by checking the *hasMore* attribute of the result set. If it is set to *false*, then the client has fetched the complete result set from the server. In this case no server side cursor will be created.

```
> curl --data @- -X POST --dump - http://localhost:8529/_api/cursor
{ "query" : "FOR u IN users LIMIT 2 RETURN u", "count" : true, "batchSize" : 2 }

HTTP/1.1 201 Created
Content-type: application/json

{
  "hasMore" : false,
  "error" : false,
  "result" : [
    {
      "name" : "user1",
      "_rev" : "210304551",
      "_key" : "210304551",
      "_id" : "users/210304551"
    },
    {
      "name" : "user2",
      "_rev" : "210304552",
      "_key" : "210304552",
      "_id" : "users/210304552"
    }
  ],
  "code" : 201,
  "count" : 2
}
```

Using a cursor

If the result set contains more documents than should be transferred in a single roundtrip (i.e. as set via the *batchSize* attribute), the server will return the first few documents and create a temporary cursor. The cursor identifier will also be returned to the client. The server will put the cursor identifier in the *id* attribute of the response object. Furthermore, the *hasMore* attribute of the response object will be set to *true*. This is an indication for the client that there are additional results to fetch from the server.

Examples:

Create and extract first batch:

```
> curl --data @- -X POST --dump - http://localhost:8529/_api/cursor
{ "query" : "FOR u IN users LIMIT 5 RETURN u", "count" : true, "batchSize" : 2 }

HTTP/1.1 201 Created
Content-type: application/json

{
  "hasMore" : true,
  "error" : false,
  "id" : "26011191",
```

```
  "result" : [
    {
      "name" : "user1",
      "_rev" : "258801191",
      "_key" : "258801191",
      "_id" : "users/258801191"
    },
    {
      "name" : "user2",
      "_rev" : "258801192",
      "_key" : "258801192",
      "_id" : "users/258801192"
    }
  ],
  "code" : 201,
  "count" : 5
}
```

Extract next batch, still have more:

```
> curl -X PUT --dump - http://localhost:8529/_api/cursor/26011191

HTTP/1.1 200 OK
Content-type: application/json

{
  "hasMore" : true,
  "error" : false,
  "id" : "26011191",
  "result": [
    {
      "name" : "user3",
      "_rev" : "258801193",
      "_key" : "258801193",
      "_id" : "users/258801193"
    },
    {
      "name" : "user4",
      "_rev" : "258801194",
      "_key" : "258801194",
      "_id" : "users/258801194"
    }
  ],
  "code" : 200,
  "count" : 5
}
```

Extract next batch, done:

```
> curl -X PUT --dump - http://localhost:8529/_api/cursor/26011191

HTTP/1.1 200 OK
Content-type: application/json

{
  "hasMore" : false,
  "error" : false,
  "result" : [
    {
      "name" : "user5",
      "_rev" : "258801195",
      "_key" : "258801195",
      "_id" : "users/258801195"
    }
  ],
  "code" : 200,
  "count" : 5
}
```

Do not do this because *hasMore* now has a value of false:

```
> curl -X PUT --dump - http://localhost:8529/_api/cursor/26011191
```

```
HTTP/1.1 404 Not Found
Content-type: application/json

{
  "errorNum": 1600,
  "errorMessage": "cursor not found: disposed or unknown cursor",
  "error": true,
  "code": 404
}
```

Modifying documents

The `_api/cursor` endpoint can also be used to execute modifying queries.

The following example appends a value into the array `arrayValue` of the document with key `test` in the collection `documents`. Normal update behavior is to replace the attribute completely, and using an update AQL query with the `PUSH()` function allows to append to the array.

```
curl --data @- -X POST --dump http://127.0.0.1:8529/_api/cursor
{ "query": "FOR doc IN documents FILTER doc._key == @myKey UPDATE doc._key WITH { arrayValue: PUSH(doc.arrayValue, @value) } IN
 documents","bindVars": { "myKey": "test", "value": 42 } }

HTTP/1.1 201 Created
Content-type: application/json; charset=utf-8

{
  "result" : [],
  "hasMore" : false,
  "extra" : {
    "stats" : {
      "writesExecuted" : 1,
      "writesIgnored" : 0,
      "scannedFull" : 0,
      "scannedIndex" : 1,
      "filtered" : 0
    },
    "warnings" : []
  },
  "error" : false,
  "code" : 201
}
```

Setting a memory limit

To set a memory limit for the query, the *memoryLimit* option can be passed to the server. The memory limit specifies the maximum number of bytes that the query is allowed to use. When a single AQL query reaches the specified limit value, the query will be aborted with a *resource limit exceeded* exception. In a cluster, the memory accounting is done per shard, so the limit value is effectively a memory limit per query per shard.

```
> curl --data @- -X POST --dump - http://localhost:8529/_api/cursor
{ "query" : "FOR i IN 1..100000 SORT i RETURN i", "memoryLimit" : 100000 }

HTTP/1.1 500 Internal Server Error
Server: ArangoDB
Connection: Keep-Alive
Content-Type: application/json; charset=utf-8
Content-Length: 115

{"error":true,"errorMessage":"query would use more memory than allowed (while executing)","code":500,"errorNum":32}
```

If no memory limit is specified, then the server default value (controlled by startup option *--query.memory-limit* will be used for restricting the maximum amount of memory the query can use. A memory limit value of *0* means that the maximum amount of memory for the query is not restricted.

Accessing Cursors via HTTP

Create cursor

create a cursor and return the first results

`POST /_api/cursor`

A JSON object describing the query and query parameters.

A JSON object with these properties is required:

- **count**: indicates whether the number of documents in the result set should be returned in the "count" attribute of the result. Calculating the "count" attribute might have a performance impact for some queries in the future so this option is turned off by default, and "count" is only returned when requested.
- **batchSize**: maximum number of result documents to be transferred from the server to the client in one roundtrip. If this attribute is not set, a server-controlled default value will be used. A *batchSize* value of *0* is disallowed.
- **cache**: flag to determine whether the AQL query cache shall be used. If set to *false*, then any query cache lookup will be skipped for the query. If set to *true*, it will lead to the query cache being checked for the query if the query cache mode is either *on* or *demand*.
- **memoryLimit**: the maximum number of memory (measured in bytes) that the query is allowed to use. If set, then the query will fail with error "resource limit exceeded" in case it allocates too much memory. A value of *0* indicates that there is no memory limit.
- **ttl**: The time-to-live for the cursor (in seconds). The cursor will be removed on the server automatically after the specified amount of time. This is useful to ensure garbage collection of cursors that are not fully fetched by clients. If not set, a server-defined value will be used.
- **query**: contains the query string to be executed
- **bindVars** (object): key/value pairs representing the bind parameters.
- **options**:
  - **failOnWarning**: When set to *true*, the query will throw an exception and abort instead of producing a warning. This option should be used during development to catch potential issues early. When the attribute is set to *false*, warnings will not be propagated to exceptions and will be returned with the query result. There is also a server configuration option `--query.fail-on-warning` for setting the default value for *failOnWarning* so it does not need to be set on a per-query level.
  - **profile**: If set to *true*, then the additional query profiling information will be returned in the sub-attribute *profile* of the *extra* return attribute if the query result is not served from the query cache.
  - **maxTransactionSize**: Transaction size limit in bytes. Honored by the RocksDB storage engine only.
  - **skipInaccessibleCollections**: AQL queries (especially graph traversals) will treat collection to which a user has no access rights as if these collections were empty. Instead of returning a forbidden access error, your queries will execute normally. This is intended to help with certain use-cases: A graph contains several collections and different users execute AQL queries on that graph. You can now naturally limit the accessible results by changing the access rights of users on collections. This feature is only available in the Enterprise Edition.
  - **maxWarningCount**: Limits the maximum number of warnings a query will return. The number of warnings a query will return is limited to 10 by default, but that number can be increased or decreased by setting this attribute.
  - **intermediateCommitCount**: Maximum number of operations after which an intermediate commit is performed automatically. Honored by the RocksDB storage engine only.
  - **satelliteSyncWait**: This *enterprise* parameter allows to configure how long a DBServer will have time to bring the satellite collections involved in the query into sync. The default value is *60.0* (seconds). When the max time has been reached the query will be stopped.
  - **fullCount**: if set to *true* and the query contains a *LIMIT* clause, then the result will have an *extra* attribute with the sub-attributes *stats* and *fullCount*, `{ ... , "extra": { "stats": { "fullCount": 123 } } }`. The *fullCount* attribute will contain the number of documents in the result before the last LIMIT in the query was applied. It can be used to count the number of documents that match certain filter criteria, but only return a subset of them, in one go. It is thus similar to MySQL's *SQL_CALC_FOUND_ROWS* hint. Note that setting the option will disable a few LIMIT optimizations and may lead to more documents being processed, and thus make queries run longer. Note that the *fullCount* attribute will only be present in the result if the query has a LIMIT clause and the LIMIT clause is actually used in the query.
  - **intermediateCommitSize**: Maximum total size of operations after which an intermediate commit is performed automatically. Honored by the RocksDB storage engine only.
  - **optimizer.rules** (string): A list of to-be-included or to-be-excluded optimizer rules can be put into this attribute, telling the

optimizer to include or exclude specific rules. To disable a rule, prefix its name with a `-`, to enable a rule, prefix it with a `+`. There is also a pseudo-rule `all`, which will match all optimizer rules.

- **maxPlans**: Limits the maximum number of plans that are created by the AQL query optimizer.

The query details include the query string plus optional query options and bind parameters. These values need to be passed in a JSON representation in the body of the POST request.

HTTP 201

A json document with these Properties is returned:

is returned if the result set can be created by the server.

- **count**: the total number of result documents available (only available if the query was executed with the *count* attribute set)
- **code**: the HTTP status code
- **extra**: an optional JSON object with extra information about the query result contained in its *stats* sub-attribute. For data-modification queries, the *extra.stats* sub-attribute will contain the number of modified documents and the number of documents that could not be modified due to an error (if *ignoreErrors* query option is specified)
- **cached**: a boolean flag indicating whether the query result was served from the query cache or not. If the query result is served from the query cache, the *extra* return attribute will not contain any *stats* sub-attribute and no *profile* sub-attribute.
- **hasMore**: A boolean indicator whether there are more results available for the cursor on the server
- **result** (anonymous json object): an array of result documents (might be empty if query has no results)
- **error**: A flag to indicate that an error occurred (*false* in this case)
- **id**: id of temporary cursor created on the server (optional, see above)

HTTP 400

A json document with these Properties is returned:

is returned if the JSON representation is malformed or the query specification is missing from the request. If the JSON representation is malformed or the query specification is missing from the request, the server will respond with *HTTP 400*. The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- **errorMessage**: a descriptive error message If the query specification is complete, the server will process the query. If an error occurs during query processing, the server will respond with *HTTP 400*. Again, the body of the response will contain details about the error. A list of query errors can be found here.
- **errorNum**: the server error number
- **code**: the HTTP status code
- **error**: boolean flag to indicate that an error occurred (*true* in this case)

Return Codes

- *201*: is returned if the result set can be created by the server.

Response Body

- **count**: the total number of result documents available (only available if the query was executed with the *count* attribute set)
- **code**: the HTTP status code
- **extra**: an optional JSON object with extra information about the query result contained in its *stats* sub-attribute. For data-modification queries, the *extra.stats* sub-attribute will contain the number of modified documents and the number of documents that could not be modified due to an error (if *ignoreErrors* query option is specified)
- **cached**: a boolean flag indicating whether the query result was served from the query cache or not. If the query result is served from the query cache, the *extra* return attribute will not contain any *stats* sub-attribute and no *profile* sub-attribute.
- **hasMore**: A boolean indicator whether there are more results available for the cursor on the server
- **result** (anonymous json object): an array of result documents (might be empty if query has no results)
- **error**: A flag to indicate that an error occurred (*false* in this case)
- **id**: id of temporary cursor created on the server (optional, see above)

- *400*: is returned if the JSON representation is malformed or the query specification is missing from the request.

If the JSON representation is malformed or the query specification is missing from the request, the server will respond with *HTTP 400*.

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

Response Body

- **errorMessage**: a descriptive error message If the query specification is complete, the server will process the query. If an error occurs during query processing, the server will respond with *HTTP 400*. Again, the body of the response will contain details about the error. A list of query errors can be found here.
- **code**: the HTTP status code
- **errorNum**: the server error number
- **error**: boolean flag to indicate that an error occurred (*true* in this case)

If the query specification is complete, the server will process the query. If an error occurs during query processing, the server will respond with *HTTP 400*. Again, the body of the response will contain details about the error.

A list of query errors can be found here.

- *404*: The server will respond with *HTTP 404* in case a non-existing collection is accessed in the query.
- *405*: The server will respond with *HTTP 405* if an unsupported HTTP method is used.

Examples

Execute a query and extract the result in a single go

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/cursor <<EOF
{
  "query" : "FOR p IN products LIMIT 2 RETURN p",
  "count" : true,
  "batchSize" : 2
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Execute a query and extract a part of the result

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/cursor <<EOF
{
  "query" : "FOR p IN products LIMIT 5 RETURN p",
  "count" : true,
  "batchSize" : 2
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Using the query option "fullCount"

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/cursor <<EOF
{
  "query" : "FOR i IN 1..1000 FILTER i > 500 LIMIT 10 RETURN i",
```

```
  "count" : true,
  "options" : {
    "fullCount" : true
  }
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Enabling and disabling optimizer rules

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/cursor <<EOF
{
  "query" : "FOR i IN 1..10 LET a = 1 LET b = 2 FILTER a + b == 3 RETURN i",
  "count" : true,
  "options" : {
    "maxPlans" : 1,
    "optimizer" : {
      "rules" : [
        "-all",
        "+remove-unnecessary-filters"
      ]
    }
  }
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Execute a data-modification query and retrieve the number of modified documents

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/cursor <<EOF
{
  "query" : "FOR p IN products REMOVE p IN products"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Execute a data-modification query with option *ignoreErrors*

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/cursor <<EOF
{
```

```
  "query" : "REMOVE 'bar' IN products OPTIONS { ignoreErrors: true }"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Bad query - Missing body

```
shell> curl -X POST --dump - http://localhost:8529/_api/cursor

HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Bad query - Unknown collection

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/cursor <<EOF
{
  "query" : "FOR u IN unknowncoll LIMIT 2 RETURN u",
  "count" : true,
  "batchSize" : 2
}
EOF

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Bad query - Execute a data-modification query that attempts to remove a non-existing document

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/cursor <<EOF
{
  "query" : "REMOVE 'foo' IN products"
}
EOF

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Read next batch from cursor

return the next results from an existing cursor

`PUT /_api/cursor/{cursor-identifier}`

Path Parameters

- *cursor-identifier* (required): The name of the cursor

If the cursor is still alive, returns an object with the following attributes:

- *id*: the *cursor-identifier*
- *result*: a list of documents for the current batch
- *hasMore*: *false* if this was the last batch
- *count*: if present the total number of elements

Note that even if *hasMore* returns *true*, the next call might still return no documents. If, however, *hasMore* is *false*, then the cursor is exhausted. Once the *hasMore* attribute has a value of *false*, the client can stop.

Return Codes

- *200*: The server will respond with *HTTP 200* in case of success.
- *400*: If the cursor identifier is omitted, the server will respond with *HTTP 404*.
- *404*: If no cursor with the specified identifier can be found, the server will respond with *HTTP 404*.

Examples

Valid request for next batch

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/cursor <<EOF
{
  "query" : "FOR p IN products LIMIT 5 RETURN p",
  "count" : true,
  "batchSize" : 2
}
EOF

shell> curl -X PUT --dump - http://localhost:8529/_api/cursor/10399

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Missing identifier

```
shell> curl -X PUT --dump - http://localhost:8529/_api/cursor

HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Unknown identifier

```
shell> curl -X PUT --dump - http://localhost:8529/_api/cursor/123123

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Delete cursor

dispose an existing cursor

`DELETE /_api/cursor/{cursor-identifier}`

Path Parameters

- *cursor-identifier* (required): The id of the cursor

Deletes the cursor and frees the resources associated with it.

The cursor will automatically be destroyed on the server when the client has retrieved all documents from it. The client can also explicitly destroy the cursor at any earlier time using an HTTP DELETE request. The cursor id must be included as part of the URL.

Note: the server will also destroy abandoned cursors automatically after a certain server-controlled timeout to avoid resource leakage.

Return Codes

- *202*: is returned if the server is aware of the cursor.
- *404*: is returned if the server is not aware of the cursor. It is also returned if a cursor is used after it has been destroyed.

Examples

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/cursor <<EOF
{
  "query" : "FOR p IN products LIMIT 5 RETURN p",
  "count" : true,
  "batchSize" : 2
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

HTTP Interface for AQL Queries

Explaining and parsing queries

ArangoDB has an HTTP interface to syntactically validate AQL queries. Furthermore, it offers an HTTP interface to retrieve the execution plan for any valid AQL query.

Both functionalities do not actually execute the supplied AQL query, but only inspect it and return meta information about it.

Explain an AQL query

explain an AQL query and return information about it

`POST /_api/explain`

A JSON object describing the query and query parameters.

A JSON object with these properties is required:

- **query**: the query which you want explained; If the query references any bind variables, these must also be passed in the attribute *bindVars*. Additional options for the query can be passed in the *options* attribute.
- **options**:
  - **optimizer.rules** (string): an array of to-be-included or to-be-excluded optimizer rules can be put into this attribute, telling the optimizer to include or exclude specific rules. To disable a rule, prefix its name with a `-`, to enable a rule, prefix it with a `+`. There is also a pseudo-rule `all`, which will match all optimizer rules.
  - **maxNumberOfPlans**: an optional maximum number of plans that the optimizer is allowed to generate. Setting this attribute to a low value allows to put a cap on the amount of work the optimizer does.
  - **allPlans**: if set to *true*, all possible execution plans will be returned. The default is *false*, meaning only the optimal plan will be returned.
- **bindVars** (object): key/value pairs representing the bind parameters.

To explain how an AQL query would be executed on the server, the query string can be sent to the server via an HTTP POST request. The server will then validate the query and create an execution plan for it. The execution plan will be returned, but the query will not be executed.

The execution plan that is returned by the server can be used to estimate the probable performance of the query. Though the actual performance will depend on many different factors, the execution plan normally can provide some rough estimates on the amount of work the server needs to do in order to actually run the query.

By default, the explain operation will return the optimal plan as chosen by the query optimizer The optimal plan is the plan with the lowest total estimated cost. The plan will be returned in the attribute *plan* of the response object. If the option *allPlans* is specified in the request, the result will contain all plans created by the optimizer. The plans will then be returned in the attribute *plans*.

The result will also contain an attribute *warnings*, which is an array of warnings that occurred during optimization or execution plan creation. Additionally, a *stats* attribute is contained in the result with some optimizer statistics. If *allPlans* is set to *false*, the result will contain an attribute *cacheable* that states whether the query results can be cached on the server if the query result cache were used. The *cacheable* attribute is not present when *allPlans* is set to *true*.

Each plan in the result is a JSON object with the following attributes:

- *nodes*: the array of execution nodes of the plan. The array of available node types can be found here
- *estimatedCost*: the total estimated cost for the plan. If there are multiple plans, the optimizer will choose the plan with the lowest total cost.
- *collections*: an array of collections used in the query
- *rules*: an array of rules the optimizer applied. An overview of the available rules can be found here
- *variables*: array of variables used in the query (note: this may contain internal variables created by the optimizer)

Return Codes

- *200*: If the query is valid, the server will respond with *HTTP 200* and return the optimal execution plan in the *plan* attribute of the response. If option *allPlans* was set in the request, an array of plans will be returned in the *allPlans* attribute instead.
- *400*: The server will respond with *HTTP 400* in case of a malformed request, or if the query contains a parse error. The body of the response will contain the error details embedded in a JSON object. Omitting bind variables if the query references any will also result in an *HTTP 400* error.
- *404*: The server will respond with *HTTP 404* in case a non-existing collection is accessed in the query.

Examples

Valid query

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/explain <<EOF
{
  "query" : "FOR p IN products RETURN p"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

A plan with some optimizer rules applied

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/explain <<EOF
{
  "query" : "FOR p IN products LET a = p.id FILTER a == 4 LET name = p.name SORT p.id
LIMIT 1 RETURN name"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Using some options

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/explain <<EOF
{
  "query" : "FOR p IN products LET a = p.id FILTER a == 4 LET name = p.name SORT p.id
LIMIT 1 RETURN name",
  "options" : {
    "maxNumberOfPlans" : 2,
    "allPlans" : true,
    "optimizer" : {
      "rules" : [
        "-all",
        "+use-index-for-sort",
        "+use-index-range"
      ]
    }
  }
```

```
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Returning all plans

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/explain <<EOF
{
  "query" : "FOR p IN products FILTER p.id == 25 RETURN p",
  "options" : {
    "allPlans" : true
  }
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

A query that produces a warning

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/explain <<EOF
{
  "query" : "FOR i IN 1..10 RETURN 1 / 0"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Invalid query (missing bind parameter)

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/explain <<EOF
{
  "query" : "FOR p IN products FILTER p.id == @id LIMIT 2 RETURN p.n"
}
EOF

HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

The data returned in the **plan** attribute of the result contains one element per AQL top-level statement (i.e. `FOR`, `RETURN`, `FILTER` etc.). If the query optimizer removed some unnecessary statements, the result might also contain less elements than there were top-level statements in the AQL query.

The following example shows a query with a non-sensible filter condition that the optimizer has removed so that there are less top-level statements.

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/explain <<EOF
{ "query" : "FOR i IN [ 1, 2, 3 ] FILTER 1 == 2 RETURN i" }
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Parse an AQL query

parse an AQL query and return information about it

`POST /_api/query`

This endpoint is for query validation only. To actually query the database, see `/api/cursor`.

A JSON object with these properties is required:

- **query**: To validate a query string without executing it, the query string can be passed to the server via an HTTP POST request.

Return Codes

- *200*: If the query is valid, the server will respond with *HTTP 200* and return the names of the bind parameters it found in the query (if any) in the *bindVars* attribute of the response. It will also return an array of the collections used in the query in the *collections* attribute. If a query can be parsed successfully, the *ast* attribute of the returned JSON will contain the abstract syntax tree representation of the query. The format of the *ast* is subject to change in future versions of ArangoDB, but it can be used to inspect how ArangoDB interprets a given query. Note that the abstract syntax tree will be returned without any optimizations applied to it.
- *400*: The server will respond with *HTTP 400* in case of a malformed request, or if the query contains a parse error. The body of the response will contain the error details embedded in a JSON object.

Examples

a Valid query

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/query <<EOF
{ "query" : "FOR p IN products FILTER p.name == @name LIMIT 2 RETURN p.n" }
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

an Invalid query

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/query <<EOF
{ "query" : "FOR p IN products FILTER p.name = @name LIMIT 2 RETURN p.n" }
EOF
```

```
HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body

Query tracking

ArangoDB has an HTTP interface for retrieving the lists of currently executing AQL queries and the list of slow AQL queries. In order to make meaningful use of these APIs, query tracking needs to be enabled in the database the HTTP request is executed for.

Returns the properties for the AQL query tracking

returns the configuration for the AQL query tracking

`GET /_api/query/properties`

Returns the current query tracking configuration. The configuration is a JSON object with the following properties:

- *enabled*: if set to *true*, then queries will be tracked. If set to *false*, neither queries nor slow queries will be tracked.
- *trackSlowQueries*: if set to *true*, then slow queries will be tracked in the list of slow queries if their runtime exceeds the value set in *slowQueryThreshold*. In order for slow queries to be tracked, the *enabled* property must also be set to *true*.
- *trackBindVars*: if set to *true*, then bind variables used in queries will be tracked.
- *maxSlowQueries*: the maximum number of slow queries to keep in the list of slow queries. If the list of slow queries is full, the oldest entry in it will be discarded when additional slow queries occur.
- *slowQueryThreshold*: the threshold value for treating a query as slow. A query with a runtime greater or equal to this threshold value will be put into the list of slow queries when slow query tracking is enabled. The value for *slowQueryThreshold* is specified in seconds.
- *maxQueryStringLength*: the maximum query string length to keep in the list of queries. Query strings can have arbitrary lengths, and this property can be used to save memory in case very long query strings are used. The value is specified in bytes.

Return Codes

- *200*: Is returned if properties were retrieved successfully.
- *400*: The server will respond with *HTTP 400* in case of a malformed request,

Changes the properties for the AQL query tracking

changes the configuration for the AQL query tracking

`PUT /_api/query/properties`

A JSON object with these properties is required:

- **maxSlowQueries**: The maximum number of slow queries to keep in the list of slow queries. If the list of slow queries is full, the oldest entry in it will be discarded when additional slow queries occur.
- **slowQueryThreshold**: The threshold value for treating a query as slow. A query with a runtime greater or equal to this threshold value will be put into the list of slow queries when slow query tracking is enabled. The value for *slowQueryThreshold* is specified in seconds.
- **enabled**: If set to *true*, then queries will be tracked. If set to *false*, neither queries nor slow queries will be tracked.
- **maxQueryStringLength**: The maximum query string length to keep in the list of queries. Query strings can have arbitrary lengths, and this property can be used to save memory in case very long query strings are used. The value is specified in bytes.
- **trackSlowQueries**: If set to *true*, then slow queries will be tracked in the list of slow queries if their runtime exceeds the value set in *slowQueryThreshold*. In order for slow queries to be tracked, the *enabled* property must also be set to *true*.
- **trackBindVars**: If set to *true*, then the bind variables used in queries will be tracked along with queries.

The properties need to be passed in the attribute *properties* in the body of the HTTP request. *properties* needs to be a JSON object.

After the properties have been changed, the current set of properties will be returned in the HTTP response.

Return Codes

- *200*: Is returned if the properties were changed successfully.
- *400*: The server will respond with *HTTP 400* in case of a malformed request,

Returns the currently running AQL queries

returns a list of currently running AQL queries

```
GET /_api/query/current
```

Returns an array containing the AQL queries currently running in the selected database. Each query is a JSON object with the following attributes:

- *id*: the query's id
- *query*: the query string (potentially truncated)
- *bindVars*: the bind parameter values used by the query
- *started*: the date and time when the query was started
- *runTime*: the query's run time up to the point the list of queries was queried
- *state*: the query's current execution state (as a string)

Return Codes

- *200*: Is returned when the list of queries can be retrieved successfully.
- *400*: The server will respond with *HTTP 400* in case of a malformed request,

Returns the list of slow AQL queries

returns a list of slow running AQL queries

```
GET /_api/query/slow
```

Returns an array containing the last AQL queries that are finished and have exceeded the slow query threshold in the selected database. The maximum amount of queries in the list can be controlled by setting the query tracking property `maxSlowQueries`. The threshold for treating a query as *slow* can be adjusted by setting the query tracking property `slowQueryThreshold`.

Each query is a JSON object with the following attributes:

- *id*: the query's id
- *query*: the query string (potentially truncated)
- *bindVars*: the bind parameter values used by the query
- *started*: the date and time when the query was started
- *runTime*: the query's total run time
- *state*: the query's current execution state (will always be "finished" for the list of slow queries)

Return Codes

- *200*: Is returned when the list of queries can be retrieved successfully.
- *400*: The server will respond with *HTTP 400* in case of a malformed request,

Clears the list of slow AQL queries

clears the list of slow AQL queries

```
DELETE /_api/query/slow
```

Clears the list of slow AQL queries

Return Codes

- *200*: The server will respond with *HTTP 200* when the list of queries was cleared successfully.
- *400*: The server will respond with *HTTP 400* in case of a malformed request.

Killing queries

Running AQL queries can also be killed on the server. ArangoDB provides a kill facility via an HTTP interface. To kill a running query, its id (as returned for the query in the list of currently running queries) must be specified. The kill flag of the query will then be set, and the query will be aborted as soon as it reaches a cancelation point.

Kills a running AQL query

kills an AQL query

```
DELETE /_api/query/{query-id}
```

Path Parameters

- *query-id* (required): The id of the query.

Kills a running query. The query will be terminated at the next cancelation point.

Return Codes

- *200*: The server will respond with *HTTP 200* when the query was still running when the kill request was executed and the query's kill flag was set.
- *400*: The server will respond with *HTTP 400* in case of a malformed request.
- *404*: The server will respond with *HTTP 404* when no query with the specified id was found.

HTTP Interface for the AQL query cache

This section describes the API methods for controlling the AQL query cache.

Clears any results in the AQL query cache

clears the AQL query cache

```
DELETE /_api/query-cache
```

clears the query cache

Return Codes

- *200*: The server will respond with *HTTP 200* when the cache was cleared successfully.
- *400*: The server will respond with *HTTP 400* in case of a malformed request.

Returns the global properties for the AQL query cache

returns the global configuration for the AQL query cache

```
GET /_api/query-cache/properties
```

Returns the global AQL query cache configuration. The configuration is a JSON object with the following properties:

- *mode*: the mode the AQL query cache operates in. The mode is one of the following values: *off*, *on* or *demand*.
- *maxResults*: the maximum number of query results that will be stored per database-specific cache.

Return Codes

- *200*: Is returned if the properties can be retrieved successfully.
- *400*: The server will respond with *HTTP 400* in case of a malformed request,

Globally adjusts the AQL query result cache properties

changes the configuration for the AQL query cache

```
PUT /_api/query-cache/properties
```

After the properties have been changed, the current set of properties will be returned in the HTTP response.

Note: changing the properties may invalidate all results in the cache. The global properties for AQL query cache. The properties need to be passed in the attribute *properties* in the body of the HTTP request. *properties* needs to be a JSON object with the following properties:

A JSON object with these properties is required:

- **mode**: the mode the AQL query cache should operate in. Possible values are *off*, *on* or *demand*.
- **maxResults**: the maximum number of query results that will be stored per database-specific cache.

Return Codes

- *200*: Is returned if the properties were changed successfully.
- *400*: The server will respond with *HTTP 400* in case of a malformed request,

HTTP Interface for AQL User Functions Management

AQL User Functions Management

This is an introduction to ArangoDB's HTTP interface for managing AQL user functions. AQL user functions are a means to extend the functionality of ArangoDB's query language (AQL) with user-defined JavaScript code.

For an overview of how AQL user functions and their implications, please refer to the Extending AQL chapter.

The HTTP interface provides an API for adding, deleting, and listing previously registered AQL user functions.

All user functions managed through this interface will be stored in the system collection *_aqlfunctions*. Documents in this collection should not be accessed directly, but only via the dedicated interfaces.

Create AQL user function

create a new AQL user function

`POST /_api/aqlfunction`

A JSON object with these properties is required:

- **isDeterministic**: an optional boolean value to indicate that the function results are fully deterministic (function return value solely depends on the input value and return value is the same for repeated calls with same input). The *isDeterministic* attribute is currently not used but may be used later for optimisations.
- **code**: a string representation of the function body.
- **name**: the fully qualified name of the user functions.

In case of success, the returned JSON object has the following properties:

- *error*: boolean flag to indicate that an error occurred (*false* in this case)
- *code*: the HTTP status code

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- *error*: boolean flag to indicate that an error occurred (*true* in this case)
- *code*: the HTTP status code
- *errorNum*: the server error number
- *errorMessage*: a descriptive error message

Return Codes

- *200*: If the function already existed and was replaced by the call, the server will respond with *HTTP 200*.
- *201*: If the function can be registered by the server, the server will respond with *HTTP 201*.
- *400*: If the JSON representation is malformed or mandatory data is missing from the request, the server will respond with *HTTP 400*.

Examples

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/aqlfunction <<EOF
{
  "name" : "myfunctions::temperature::celsiustofahrenheit",
  "code" : "function (celsius) { return celsius * 1.8 + 32; }"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
```

```
x-content-type-options: nosniff

{
  "error" : false,
  "code" : 201
}
```

Remove existing AQL user function

remove an existing AQL user function

`DELETE /_api/aqlfunction/{name}`

Path Parameters

- *name* (required): the name of the AQL user function.

Query Parameters

- *group* (optional): If set to *true*, then the function name provided in *name* is treated as a namespace prefix, and all functions in the specified namespace will be deleted. If set to *false*, the function name provided in *name* must be fully qualified, including any namespaces.

Removes an existing AQL user function, identified by *name*.

In case of success, the returned JSON object has the following properties:

- *error*: boolean flag to indicate that an error occurred (*false* in this case)
- *code*: the HTTP status code

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- *error*: boolean flag to indicate that an error occurred (*true* in this case)
- *code*: the HTTP status code
- *errorNum*: the server error number
- *errorMessage*: a descriptive error message

Return Codes

- *200*: If the function can be removed by the server, the server will respond with *HTTP 200*.
- *400*: If the user function name is malformed, the server will respond with *HTTP 400*.
- *404*: If the specified user user function does not exist, the server will respond with *HTTP 404*.

Examples

deletes a function:

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/aqlfunction/square::x::y

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

{
  "error" : false,
  "code" : 200
}
```

function not found:

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/aqlfunction/myfunction::x::y

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Return registered AQL user functions

gets all reqistered AQL user functions

`GET /_api/aqlfunction`

Query Parameters

- *namespace* (optional): Returns all registered AQL user functions from namespace *namespace*.

Returns all registered AQL user functions.

The call will return a JSON array with all user functions found. Each user function will at least have the following attributes:

- *name*: The fully qualified name of the user function
- *code*: A string representation of the function body

Return Codes

- *200*: if success *HTTP 200* is returned.

Examples

```
shell> curl --dump - http://localhost:8529/_api/aqlfunction

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

[
  {
    "name" : "myfunctions::temperature::celsiustofahrenheit",
    "code" : "function (celsius) { return celsius * 1.8 + 32; }"
  }
]
```

HTTP Interface for Simple Queries

Simple Queries

This is an introduction to ArangoDB's HTTP interface for simple queries.

Simple queries can be used if the query condition is straight forward simple, i.e., a document reference, all documents, a query-by-example, or a simple geo query. In a simple query you can specify exactly one collection and one condition. The result can then be sorted and can be split into pages.

Working with Simples Queries using HTTP

To limit the amount of results to be transferred in one batch, simple queries support a *batchSize* parameter that can optionally be used to tell the server to limit the number of results to be transferred in one batch to a certain value. If the query has more results than were transferred in one go, more results are waiting on the server so they can be fetched subsequently. If no value for the *batchSize* parameter is specified, the server will use a reasonable default value.

If the server has more documents than should be returned in a single batch, the server will set the *hasMore* attribute in the result. It will also return the id of the server-side cursor in the *id* attribute in the result. This id can be used with the cursor API to fetch any outstanding results from the server and dispose the server-side cursor afterwards.

Return all documents

returns all documents of a collection

`PUT /_api/simple/all`

Request Body (required)

Contains the query.

Returns all documents of a collections. The call expects a JSON object as body with the following attributes:

- *collection*: The name of the collection to query.
- *skip*: The number of documents to skip in the query (optional).
- *limit*: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. (optional)

Returns a cursor containing the result, see Http Cursor for details.

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

Limit the amount of documents using *limit*

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/all <<EOF
{ "collection": "products", "skip": 2, "limit" : 2 }
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Using a *batchSize* value

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/all <<EOF
{ "collection": "products", "batchSize" : 3 }
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Simple query by-example

returns all documents of a collection matching a given example

`PUT /_api/simple/by-example`

A JSON object with these properties is required:

- **skip**: The number of documents to skip in the query (optional).
- **batchSize**: maximum number of result documents to be transferred from the server to the client in one roundtrip. If this attribute is not set, a server-controlled default value will be used. A *batchSize* value of *0* is disallowed.
- **limit**: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. (optional)
- **example**: The example document.
- **collection**: The name of the collection to query.

This will find all documents matching a given example.

Returns a cursor containing the result, see Http Cursor for details.

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

Matching an attribute

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/by-example
<<EOF
{
  "collection" : "products",
  "example" : {
    "i" : 1
  }
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Matching an attribute which is a sub-document

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/by-example
<<EOF
{
  "collection" : "products",
  "example" : {
    "a.j" : 1
  }
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Matching an attribute within a sub-document

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/by-example
<<EOF
{
  "collection" : "products",
  "example" : {
    "a" : {
      "j" : 1
    }
  }
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Find documents matching an example

returns one document of a collection matching a given example

`PUT /_api/simple/first-example`

A JSON object with these properties is required:

- **example**: The example document.
- **collection**: The name of the collection to query.

This will return the first document matching a given example.

Returns a result containing the document or *HTTP 404* if no document matched the example.

If more than one document in the collection matches the specified example, only one of these documents will be returned, and it is undefined which of the matching documents is returned.

Return Codes

- *200*: is returned when the query was successfully executed.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

If a matching document was found

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/first-
example <<EOF
{
  "collection" : "products",
  "example" : {
    "i" : 1
  }
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
If no document was found

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/first-
example <<EOF
{
  "collection" : "products",
  "example" : {
    "l" : 1
  }
}
EOF

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Find documents by their keys

fetches multiple documents by their keys

`PUT /_api/simple/lookup-by-keys`

A JSON object with these properties is required:

- **keys** (string): array with the _keys of documents to remove.
- **collection**: The name of the collection to look in for the documents

Looks up the documents in the specified collection using the array of keys provided. All documents for which a matching key was specified in the *keys* array and that exist in the collection will be returned. Keys for which no document can be found in the underlying collection are ignored, and no exception will be thrown for them.

The body of the response contains a JSON object with a *documents* attribute. The *documents* attribute is an array containing the matching documents. The order in which matching documents are present in the result array is unspecified.

Return Codes

- *200*: is returned if the operation was carried out successfully.
- *404*: is returned if the collection was not found. The response body contains an error document in this case.
- *405*: is returned if the operation was called with a different HTTP METHOD than PUT.

Examples

Looking up existing documents

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/lookup-by-
keys <<EOF
{
  "keys" : [
    "test0",
    "test1",
    "test2",
    "test3",
    "test4",
    "test5",
    "test6",
    "test7",
    "test8",
    "test9"
  ],
  "collection" : "test"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Looking up non-existing documents

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/lookup-by-
keys <<EOF
{
  "keys" : [
    "foo",
    "bar",
    "baz"
  ],
  "collection" : "test"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Return a random document

returns a random document from a collection

`PUT /_api/simple/any`

Returns a random document from a collection. The call expects a JSON object as body with the following attributes:

A JSON object with these properties is required:

- **collection**: The identifier or name of the collection to query. Returns a JSON object with the document stored in the attribute *document* if the collection contains at least one document. If the collection is empty, the *document* attrbute contains null.

Return Codes

- *200*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/any <<EOF
{
  "collection" : "products"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Remove documents by their keys

removes multiple documents by their keys

`PUT /_api/simple/remove-by-keys`

A JSON object with these properties is required:

- **keys** (string): array with the _keys of documents to remove.
- **options**:
  - **returnOld**: if set to *true* and *silent* above is *false*, then the above information about the removed documents contains the complete removed documents.
  - **silent**: if set to *false*, then the result will contain an additional attribute *old* which contains an array with one entry for each removed document. By default, these entries will have the *_id*, *_key* and *_rev* attributes.
  - **waitForSync**: if set to true, then all removal operations will instantly be synchronized to disk. If this is not specified, then the collection's default sync behavior will be applied.
- **collection**: The name of the collection to look in for the documents to remove

Looks up the documents in the specified collection using the array of keys provided, and removes all documents from the collection whose keys are contained in the *keys* array. Keys for which no document can be found in the underlying collection are ignored, and no exception will be thrown for them.

The body of the response contains a JSON object with information how many documents were removed (and how many were not). The *removed* attribute will contain the number of actually removed documents. The *ignored* attribute will contain the number of keys in the request for which no matching document could be found.

Return Codes

- *200*: is returned if the operation was carried out successfully. The number of removed documents may still be 0 in this case if none of the specified document keys were found in the collection.
- *404*: is returned if the collection was not found. The response body contains an error document in this case.
- *405*: is returned if the operation was called with a different HTTP METHOD than PUT.

Examples

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/remove-by-
keys <<EOF
{
  "keys" : [
    "test0",
    "test1",
    "test2",
    "test3",
    "test4",
    "test5",
    "test6",
    "test7",
    "test8",
    "test9"
  ],
  "collection" : "test"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/remove-by-
keys <<EOF
{
  "keys" : [
    "foo",
    "bar",
    "baz"
  ],
  "collection" : "test"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Remove documents by example

removes all documents of a collection that match an example

`PUT /_api/simple/remove-by-example`

A JSON object with these properties is required:

- **example**: An example document that all collection documents are compared against.
- **collection**: The name of the collection to remove from.
- **options**:
  - **limit**: an optional value that determines how many documents to delete at most. If *limit* is specified but is less than the number of documents in the collection, it is undefined which of the documents will be deleted.
  - **waitForSync**: if set to true, then all removal operations will instantly be synchronized to disk. If this is not specified, then the collection's default sync behavior will be applied.

This will find all documents in the collection that match the specified example object.

Note: the *limit* attribute is not supported on sharded collections. Using it will result in an error.

Returns the number of documents that were deleted.

Return Codes

- *200*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/remove-by-
example <<EOF
{
  "collection" : "products",
  "example" : {
    "a" : {
      "j" : 1
    }
  }
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Using Parameter: waitForSync and limit

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/remove-by-
example <<EOF
{
  "collection" : "products",
  "example" : {
    "a" : {
```

```
      "j" : 1
    }
  },
  "waitForSync" : true,
  "limit" : 2
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Using Parameter: waitForSync and limit with new signature

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/remove-by-example <<EOF
{
  "collection" : "products",
  "example" : {
    "a" : {
      "j" : 1
    }
  },
  "options" : {
    "waitForSync" : true,
    "limit" : 2
  }
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
replaces the body of all documents of a collection that match an example @RESTHEADER{PUT /_api/simple/replace-by-example, Replace documents by example}

A JSON object with these properties is required:

- **options**:
  - **limit**: an optional value that determines how many documents to replace at most. If *limit* is specified but is less than the number of documents in the collection, it is undefined which of the documents will be replaced.
  - **waitForSync**: if set to true, then all removal operations will instantly be synchronized to disk. If this is not specified, then the collection's default sync behavior will be applied.
- **example**: An example document that all collection documents are compared against.
- **collection**: The name of the collection to replace within.
- **newValue**: The replacement document that will get inserted in place of the "old" documents.

This will find all documents in the collection that match the specified example object, and replace the entire document body with the new value specified. Note that document meta-attributes such as *_id*, *_key*, *_from*, *_to* etc. cannot be replaced.

Note: the *limit* attribute is not supported on sharded collections. Using it will result in an error.

Returns the number of documents that were replaced.

Return Codes

- *200*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/replace-by-example <<EOF
{
  "collection" : "products",
  "example" : {
    "a" : {
      "j" : 1
    }
  },
  "newValue" : {
    "foo" : "bar"
  },
  "limit" : 3
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Using new Signature for attributes WaitForSync and limit

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/replace-by-example <<EOF
{
  "collection" : "products",
  "example" : {
    "a" : {
      "j" : 1
    }
  },
  "newValue" : {
    "foo" : "bar"
  },
  "options" : {
    "limit" : 3,
    "waitForSync" : true
  }
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
partially updates the body of all documents of a collection that match an example @RESTHEADER{PUT /_api/simple/update-by-example, Update documents by example}

A JSON object with these properties is required:

- **options**:
  - **keepNull**: This parameter can be used to modify the behavior when handling *null* values. Normally, *null* values are stored in the database. By setting the *keepNull* parameter to *false*, this behavior can be changed so that all attributes in *data* with *null* values will be removed from the updated document.
  - **mergeObjects**: Controls whether objects (not arrays) will be merged if present in both the existing and the patch document. If set to false, the value in the patch document will overwrite the existing document's value. If set to true, objects will be merged. The default is true.
  - **limit**: an optional value that determines how many documents to update at most. If *limit* is specified but is less than the number of documents in the collection, it is undefined which of the documents will be updated.
  - **waitForSync**: if set to true, then all removal operations will instantly be synchronized to disk. If this is not specified, then the collection's default sync behavior will be applied.
- **example**: An example document that all collection documents are compared against.
- **collection**: The name of the collection to update within.
- **newValue**: A document containing all the attributes to update in the found documents.

This will find all documents in the collection that match the specified example object, and partially update the document body with the new value specified. Note that document meta-attributes such as *_id*, *_key*, *_from*, *_to* etc. cannot be replaced.

Note: the *limit* attribute is not supported on sharded collections. Using it will result in an error.

Returns the number of documents that were updated.

Return Codes

- *200*: is returned if the collection was updated successfully and *waitForSync* was *true*.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

using old syntax for options

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/update-by-example <<EOF
{
  "collection" : "products",
  "example" : {
    "a" : {
      "j" : 1
    }
  },
  "newValue" : {
    "a" : {
      "j" : 22
    }
  },
  "limit" : 3
}
EOF

HTTP/1.1 200 OK
```

```
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body
using new signature for options

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/update-by-
example <<EOF
{
  "collection" : "products",
  "example" : {
    "a" : {
      "j" : 1
    }
  },
  "newValue" : {
    "a" : {
      "j" : 22
    }
  },
  "options" : {
    "limit" : 3,
    "waitForSync" : true
  }
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body

Simple range query

returns all documents of a collection within a range

`PUT /_api/simple/range`

A JSON object with these properties is required:

- **right**: The upper bound.
- **attribute**: The attribute path to check.
- **collection**: The name of the collection to query.
- **limit**: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. (optional)
- **closed**: If *true*, use interval including *left* and *right*, otherwise exclude *right*, but include *left*.
- **skip**: The number of documents to skip in the query (optional).
- **left**: The lower bound.

This will find all documents within a given range. In order to execute a range query, a skip-list index on the queried attribute must be present.

Returns a cursor containing the result, see Http Cursor for details.

Note: the *range* simple query is **deprecated** as of ArangoDB 2.6. The function may be removed in future versions of ArangoDB. The preferred way for retrieving documents from a collection within a specific range is to use an AQL query as follows:

```
FOR doc IN @@collection
  FILTER doc.value >= @left && doc.value < @right
  LIMIT @skip, @limit
  RETURN doc`
```

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown or no suitable index for the range query is present. The response body contains an error document in this case.

Examples

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/range <<EOF
{
  "collection" : "products",
  "attribute" : "i",
  "left" : 2,
  "right" : 4
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Returns documents near a coordinate

returns all documents of a collection near a given location

`PUT /_api/simple/near`

A JSON object with these properties is required:

- **distance**: If given, the attribute key used to return the distance to the given coordinate. (optional). If specified, distances are returned in meters.
- **skip**: The number of documents to skip in the query. (optional)
- **longitude**: The longitude of the coordinate.
- **limit**: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. The default is 100. (optional)
- **collection**: The name of the collection to query.
- **latitude**: The latitude of the coordinate.
- **geo**: If given, the identifier of the geo-index to use. (optional)

The default will find at most 100 documents near the given coordinate. The returned array is sorted according to the distance, with the nearest document being first in the return array. If there are near documents of equal distance, documents are chosen randomly from this set until the limit is reached.

In order to use the *near* operator, a geo index must be defined for the collection. This index also defines which attribute holds the coordinates for the document. If you have more than one geo-spatial index, you can use the *geo* field to select a particular index.

Returns a cursor containing the result, see Http Cursor for details.

Note: the *near* simple query is **deprecated** as of ArangoDB 2.6. This API may be removed in future versions of ArangoDB. The preferred way for retrieving documents from a collection using the near operator is to issue an AQL query using the *NEAR* function as follows:

```
FOR doc IN NEAR(@@collection, @latitude, @longitude, @limit)
  RETURN doc`
```

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

Without distance

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/near <<EOF
{
  "collection" : "products",
  "latitude" : 0,
  "longitude" : 0,
  "skip" : 1,
  "limit" : 2
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
With distance

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/near <<EOF
{
  "collection" : "products",
  "latitude" : 0,
  "longitude" : 0,
  "skip" : 1,
  "limit" : 3,
  "distance" : "distance"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Find documents within a radius around a coordinate

returns all documents of a collection within a given radius

`PUT /_api/simple/within`

A JSON object with these properties is required:

- **distance**: If given, the attribute key used to return the distance to the given coordinate. (optional). If specified, distances are returned in meters.
- **skip**: The number of documents to skip in the query. (optional)
- **longitude**: The longitude of the coordinate.
- **radius**: The maximal radius (in meters).
- **collection**: The name of the collection to query.
- **latitude**: The latitude of the coordinate.
- **limit**: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. The default is 100. (optional)
- **geo**: If given, the identifier of the geo-index to use. (optional)

This will find all documents within a given radius around the coordinate (*latitude*, *longitude*). The returned list is sorted by distance.

In order to use the *within* operator, a geo index must be defined for the collection. This index also defines which attribute holds the coordinates for the document. If you have more than one geo-spatial index, you can use the *geo* field to select a particular index.

Returns a cursor containing the result, see Http Cursor for details.

Note: the *within* simple query is **deprecated** as of ArangoDB 2.6. This API may be removed in future versions of ArangoDB. The preferred way for retrieving documents from a collection using the near operator is to issue an AQL query using the *WITHIN* function as follows:

```
FOR doc IN WITHIN(@@collection, @latitude, @longitude, @radius, @distanceAttributeName)
  RETURN doc
```

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

Without distance

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/near <<EOF
{
  "collection" : "products",
  "latitude" : 0,
  "longitude" : 0,
  "skip" : 1,
  "limit" : 2,
  "radius" : 500
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
With distance

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/near <<EOF
{
  "collection" : "products",
  "latitude" : 0,
```

```
  "longitude" : 0,
  "skip" : 1,
  "limit" : 3,
  "distance" : "distance",
  "radius" : 300
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Within rectangle query

returns all documents of a collection within a rectangle

`PUT /_api/simple/within-rectangle`

A JSON object with these properties is required:

- **latitude1**: The latitude of the first rectangle coordinate.
- **skip**: The number of documents to skip in the query. (optional)
- **latitude2**: The latitude of the second rectangle coordinate.
- **longitude2**: The longitude of the second rectangle coordinate.
- **longitude1**: The longitude of the first rectangle coordinate.
- **limit**: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. The default is 100. (optional)
- **collection**: The name of the collection to query.
- **geo**: If given, the identifier of the geo-index to use. (optional)

This will find all documents within the specified rectangle (determined by the given coordinates (*latitude1*, *longitude1*, *latitude2*, *longitude2*).

In order to use the *within-rectangle* query, a geo index must be defined for the collection. This index also defines which attribute holds the coordinates for the document. If you have more than one geo-spatial index, you can use the *geo* field to select a particular index.

Returns a cursor containing the result, see Http Cursor for details.

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/within-
rectangle <<EOF
{
  "collection" : "products",
  "latitude1" : 0,
  "longitude1" : 0,
  "latitude2" : 0.2,
  "longitude2" : 0.2,
  "skip" : 1,
  "limit" : 2
}
```

```
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Fulltext index query

returns documents of a collection as a result of a fulltext query

`PUT /_api/simple/fulltext`

A JSON object with these properties is required:

- **index**: The identifier of the fulltext-index to use.
- **attribute**: The attribute that contains the texts.
- **collection**: The name of the collection to query.
- **limit**: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. (optional)
- **skip**: The number of documents to skip in the query (optional).
- **query**: The fulltext query. Please refer to Fulltext queries for details.

This will find all documents from the collection that match the fulltext query specified in *query*.

In order to use the *fulltext* operator, a fulltext index must be defined for the collection and the specified attribute.

Returns a cursor containing the result, see Http Cursor for details.

Note: the *fulltext* simple query is **deprecated** as of ArangoDB 2.6. This API may be removed in future versions of ArangoDB. The preferred way for retrieving documents from a collection using the near operator is to issue an AQL query using the *FULLTEXT* AQL function as follows:

```
FOR doc IN FULLTEXT(@@collection, @attributeName, @queryString, @limit)
  RETURN doc
```

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/fulltext
<<EOF
{
  "collection" : "products",
  "attribute" : "text",
  "query" : "word"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

HTTP Interface for Async Results Management

Request Execution

ArangoDB provides various methods of executing client requests. Clients can choose the appropriate method on a per-request level based on their throughput, control flow, and durability requirements.

Blocking execution

ArangoDB is a multi-threaded server, allowing the processing of multiple client requests at the same time. Communication handling and the actual work can be performed by multiple worker threads in parallel.

Though multiple clients can connect and send their requests in parallel to ArangoDB, clients may need to wait for their requests to be processed.

By default, the server will fully process an incoming request and then return the result to the client. The client must wait for the server's response before it can send additional requests over the connection. For clients that are single-threaded or not event-driven, waiting for the full server response may be non-optimal.

Furthermore, please note that even if the client closes the HTTP connection, the request running on the server will still continue until it is complete and only then notice that the client no longer listens. Thus closing the connection does not help to abort a long running query! See below under Async Execution and later Result Retrieval and HttpJobPutCancel for details.

Fire and Forget

To mitigate client blocking issues, ArangoDB since version 1.4. offers a generic mechanism for non-blocking requests: if clients add the HTTP header *x-arango-async: true* to their requests, ArangoDB will put the request into an in-memory task queue and return an HTTP 202 (accepted) response to the client instantly. The server will execute the tasks from the queue asynchronously, decoupling the client requests and the actual work.

This allows for much higher throughput than if clients would wait for the server's response. The downside is that the response that is sent to the client is always the same (a generic HTTP 202) and clients cannot make a decision based on the actual operation's result at this point. In fact, the operation might have not even been executed at the time the generic response has reached the client. Clients can thus not rely on their requests having been processed successfully.

The asynchronous task queue on the server is not persisted, meaning not-yet processed tasks from the queue will be lost in case of a crash. However, the client will not know whether they were processed or not.

Clients should thus not send the extra header when they have strict durability requirements or if they rely on result of the sent operation for further actions.

The maximum number of queued tasks is determined by the startup option *-scheduler.maximal-queue-size*. If more than this number of tasks are already queued, the server will reject the request with an HTTP 500 error.

Finally, please note that it is not possible to cancel such a fire and forget job, since you won't get any handle to identify it later on. If you need to cancel requests, use Async Execution and later Result Retrieval and HttpJobPutCancel below.

Async Execution and later Result Retrieval

By adding the HTTP header *x-arango-async: store* to a request, clients can instruct the ArangoDB server to execute the operation asynchronously as above, but also store the operation result in memory for a later retrieval. The server will return a job id in the HTTP response header *x-arango-async-id*. The client can use this id in conjunction with the HTTP API at */_api/job*, which is described in detail in this manual.

Clients can ask the ArangoDB server via the async jobs API which results are ready for retrieval, and which are not. Clients can also use the async jobs API to retrieve the original results of an already executed async job by passing it the originally returned job id. The server will then return the job result as if the job was executed normally. Furthermore, clients can cancel running async jobs by their job id, see HttpJobPutCancel.

ArangoDB will keep all results of jobs initiated with the *x-arango-async: store* header. Results are removed from the server only if a client explicitly asks the server for a specific result.

The async jobs API also provides methods for garbage collection that clients can use to get rid of "old" not fetched results. Clients should call this method periodically because ArangoDB does not artificially limit the number of not-yet-fetched results.

It is thus a client responsibility to store only as many results as needed and to fetch available results as soon as possible, or at least to clean up not fetched results from time to time.

The job queue and the results are kept in memory only on the server, so they will be lost in case of a crash.

Canceling asynchronous jobs

As mentioned above it is possible to cancel an asynchronously running job using its job ID. This is done with a PUT request as described in HttpJobPutCancel.

However, a few words of explanation about what happens behind the scenes are in order. Firstly, a running async query can internally be executed by C++ code or by JavaScript code. For example CRUD operations are executed directly in C++, whereas AQL queries and transactions are executed by JavaScript code. The job cancelation only works for JavaScript code, since the mechanism used is simply to trigger an uncatchable exception in the JavaScript thread, which will be caught on the C++ level, which in turn leads to the cancelation of the job. No result can be retrieved later, since all data about the request is discarded.

If you cancel a job running on a coordinator of a cluster (Sharding), then only the code running on the coordinator is stopped, there may remain tasks within the cluster which have already been distributed to the DBservers and it is currently not possible to cancel them as well.

Async Execution and Authentication

If a request requires authentication, the authentication procedure is run before queueing. The request will only be queued if it valid credentials and the authentication succeeds. If the request does not contain valid credentials, it will not be queued but rejected instantly in the same way as a "regular", non-queued request.

Managing Async Results via HTTP

Return result of an async job

fetches a job result and removes it from the queue

`PUT /_api/job/{job-id}`

Path Parameters

- *job-id* (required): The async job id.

Returns the result of an async job identified by job-id. If the async job result is present on the server, the result will be removed from the list of result. That means this method can be called for each job-id once. The method will return the original job result's headers and body, plus the additional HTTP header x-arango-async-job-id. If this header is present, then the job was found and the response contains the original job's result. If the header is not present, the job was not found and the response contains status information from the job manager.

Return Codes

- *204*: is returned if the job requested via job-id is still in the queue of pending (or not yet finished) jobs. In this case, no x-arango-async-id HTTP header will be returned.
- *400*: is returned if no job-id was specified in the request. In this case, no x-arango-async-id HTTP header will be returned.
- *404*: is returned if the job was not found or already deleted or fetched from the job result list. In this case, no x-arango-async-id HTTP header will be returned.

Examples

Not providing a job-id:

```
shell> curl -X PUT --dump - http://localhost:8529/_api/job

HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Providing a job-id for a non-existing job:

```
shell> curl -X PUT --dump - http://localhost:8529/_api/job/notthere

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Fetching the result of an HTTP GET job:

```
shell> curl -X PUT --header 'x-arango-async: store' --dump -
http://localhost:8529/_api/version

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676912
content-type: text/plain; charset=utf-8

shell> curl -X PUT --dump - http://localhost:8529/_api/job/154003193676912

HTTP/1.1 200 OK
x-content-type-options: nosniff
x-arango-async-id: 154003193676912
content-type: application/json; charset=utf-8
```

show response body

Fetching the result of an HTTP POST job that failed:

```
shell> curl -X PUT --header 'x-arango-async: store' --data-binary @- --dump -
http://localhost:8529/_api/collection <<EOF
{
  "name" : " this name is invalid "
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676917
content-type: text/plain; charset=utf-8

shell> curl -X PUT --dump - http://localhost:8529/_api/job/154003193676917

HTTP/1.1 400 Bad Request
```

```
x-content-type-options: nosniff
x-arango-async-id: 154003193676917
content-type: application/json; charset=utf-8
```

show response body

Cancel async job

cancels an async job

`PUT /_api/job/{job-id}/cancel`

Path Parameters

- *job-id* (required): The async job id.

Cancels the currently running job identified by job-id. Note that it still might take some time to actually cancel the running async job.

Return Codes

- *200*: cancel has been initiated.
- *400*: is returned if no job-id was specified in the request. In this case, no x-arango-async-id HTTP header will be returned.
- *404*: is returned if the job was not found or already deleted or fetched from the job result list. In this case, no x-arango-async-id HTTP header will be returned.

Examples

```
shell> curl -X POST --header 'x-arango-async: store' --data-binary @- --dump -
http://localhost:8529/_api/cursor <<EOF
{
  "query" : "FOR i IN 1..10 FOR j IN 1..10 LET x = sleep(1.0) FILTER i == 5 && j == 5
RETURN 42"
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676875
content-type: text/plain; charset=utf-8

shell> curl --dump - http://localhost:8529/_api/job/pending

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

[
  "154003193676875"
]
shell> curl -X PUT --dump - http://localhost:8529/_api/job/154003193676875/cancel

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Deletes async job

deletes an async job result

`DELETE /_api/job/{type}`

Path Parameters

- *type* (required): The type of jobs to delete. type can be:
- *all*: Deletes all jobs results. Currently executing or queued async jobs will not be stopped by this call.
- *expired*: Deletes expired results. To determine the expiration status of a result, pass the stamp query parameter. stamp needs to be a UNIX timestamp, and all async job results created at a lower timestamp will be deleted.
- *an actual job-id*: In this case, the call will remove the result of the specified async job. If the job is currently executing or queued, it will not be aborted.

Query Parameters

- *stamp* (optional): A UNIX timestamp specifying the expiration threshold when type is expired.

Deletes either all job results, expired job results, or the result of a specific job. Clients can use this method to perform an eventual garbage collection of job results.

Return Codes

- *200*: is returned if the deletion operation was carried out successfully. This code will also be returned if no results were deleted.
- *400*: is returned if type is not specified or has an invalid value.
- *404*: is returned if type is a job-id but no async job with the specified id was found.

Examples

Deleting all jobs:

```
shell> curl -X PUT --header 'x-arango-async: store' --dump -
http://localhost:8529/_api/version

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676884
content-type: text/plain; charset=utf-8

shell> curl -X DELETE --dump - http://localhost:8529/_api/job/all

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

{
  "result" : true
}
```

Deleting expired jobs:

```
shell> curl -X PUT --header 'x-arango-async: store' --dump -
http://localhost:8529/_api/version

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676889
content-type: text/plain; charset=utf-8
```

```
shell> curl --dump - http://localhost:8529/_admin/time

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Deleting the result of a specific job:

```
shell> curl -X PUT --header 'x-arango-async: store' --dump -
http://localhost:8529/_api/version

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676898
content-type: text/plain; charset=utf-8

shell> curl -X DELETE --dump - http://localhost:8529/_api/job/154003193676898

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

{
  "result" : true
}
```

Deleting the result of a non-existing job:

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/job/AreYouThere

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Returns async job

Returns the status of a specific job

`GET /_api/job/{job-id}`

Path Parameters

- *job-id* (required): The async job id.

Returns the processing status of the specified job. The processing status can be determined by peeking into the HTTP response code of the response.

Return Codes

- *200*: is returned if the job requested via job-id has been executed and its result is ready to fetch.
- *204*: is returned if the job requested via job-id is still in the queue of pending (or not yet finished) jobs.
- *404*: is returned if the job was not found or already deleted or fetched from the job result list.

Examples

Querying the status of a done job:

```
shell> curl -X PUT --header 'x-arango-async: store' --dump -
http://localhost:8529/_api/version

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676939
content-type: text/plain; charset=utf-8

shell> curl -X PUT --dump - http://localhost:8529/_api/job/154003193676939

HTTP/1.1 200 OK
x-content-type-options: nosniff
x-arango-async-id: 154003193676939
content-type: application/json; charset=utf-8
```

show response body
Querying the status of a pending job: (therefore we create a long runnging job...)

```
shell> curl -X POST --header 'x-arango-async: store' --data-binary @- --dump -
http://localhost:8529/_api/transaction <<EOF
{
  "collections" : {
    "read" : [
      "_frontend"
    ]
  },
  "action" : "function () {require('internal').sleep(15.0);}"
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676944
content-type: text/plain; charset=utf-8

shell> curl --dump - http://localhost:8529/_api/job/154003193676944

HTTP/1.1 204 No Content
content-type: text/plain; charset=utf-8
x-content-type-options: nosniff
```

Returns list of async jobs

Returns the ids of job results with a specific status

`GET /_api/job/{type}`

Path Parameters

- *type* (required): The type of jobs to return. The type can be either done or pending. Setting the type to done will make the method return the ids of already completed async jobs for which results can be fetched. Setting the type to pending will return the ids of not

yet finished async jobs.

Query Parameters

- *count* (optional):

The maximum number of ids to return per call. If not specified, a server-defined maximum value will be used.

Returns the list of ids of async jobs with a specific status (either done or pending). The list can be used by the client to get an overview of the job system status and to retrieve completed job results later.

Return Codes

- *200*: is returned if the list can be compiled successfully. Note: the list might be empty.
- *400*: is returned if type is not specified or has an invalid value.

Examples

Fetching the list of done jobs:

```
shell> curl -X PUT --header 'x-arango-async: store' --dump -
http://localhost:8529/_api/version

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676922
content-type: text/plain; charset=utf-8

shell> curl --dump - http://localhost:8529/_api/job/done

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

[
  "154003193676922"
]
```

Fetching the list of pending jobs:

```
shell> curl -X PUT --header 'x-arango-async: store' --dump -
http://localhost:8529/_api/version

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676927
content-type: text/plain; charset=utf-8

shell> curl --dump - http://localhost:8529/_api/job/pending

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

[ ]
```

Querying the status of a pending job: (we create a sleep job therefore...)

```
shell> curl -X POST --header 'x-arango-async: store' --data-binary @- --dump -
```

```
http://localhost:8529/_api/transaction <<EOF
{
  "collections" : {
    "read" : [
      "_frontend"
    ]
  },
  "action" : "function () {require('internal').sleep(15.0);}"
}
EOF

HTTP/1.1 202 Accepted
x-content-type-options: nosniff
x-arango-async-id: 154003193676932
content-type: text/plain; charset=utf-8

shell> curl --dump - http://localhost:8529/_api/job/pending

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

[
  "154003193676932"
]
shell> curl -X DELETE --dump - http://localhost:8529/_api/job/154003193676932

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

{
  "result" : true
}
```

HTTP Interface for Bulk Imports

ArangoDB provides an HTTP interface to import multiple documents at once into a collection. This is known as a bulk import.

The data uploaded must be provided in JSON format. There are two mechanisms to import the data:

- self-contained JSON documents: in this case, each document contains all attribute names and values. Attribute names may be completely different among the documents uploaded
- attribute names plus document data: in this case, the first array must contain the attribute names of the documents that follow. The following arrays containing only the attribute values. Attribute values will be mapped to the attribute names by positions.

The endpoint address is */_api/import* for both input mechanisms. Data must be sent to this URL using an HTTP POST request. The data to import must be contained in the body of the POST request.

The *collection* query parameter must be used to specify the target collection for the import. Importing data into a non-existing collection will produce an error.

The *waitForSync* query parameter can be set to *true* to make the import only return if all documents have been synced to disk.

The *complete* query parameter can be set to *true* to make the entire import fail if any of the uploaded documents is invalid and cannot be imported. In this case, no documents will be imported by the import run, even if a failure happens at the end of the import.

If *complete* has a value other than *true*, valid documents will be imported while invalid documents will be rejected, meaning only some of the uploaded documents might have been imported.

The *details* query parameter can be set to *true* to make the import API return details about documents that could not be imported. If *details* is *true*, then the result will also contain a *details* attribute which is an array of detailed error messages. If the *details* is set to *false* or omitted, no details will be returned.

imports document values

imports documents from JSON-encoded lists

`POST /_api/import#document`

Request Body (required)

The body must consist of JSON-encoded arrays of attribute values, with one line per document. The first row of the request must be a JSON-encoded array of attribute names. These attribute names are used for the data in the subsequent lines.

Query Parameters

- *collection* (required): The collection name.
- *fromPrefix* (optional): An optional prefix for the values in `_from` attributes. If specified, the value is automatically prepended to each `_from` input value. This allows specifying just the keys for `_from`.
- *toPrefix* (optional): An optional prefix for the values in `_to` attributes. If specified, the value is automatically prepended to each `_to` input value. This allows specifying just the keys for `_to`.
- *overwrite* (optional): If this parameter has a value of `true` or `yes`, then all data in the collection will be removed prior to the import. Note that any existing index definitions will be preseved.
- *waitForSync* (optional): Wait until documents have been synced to disk before returning.
- *onDuplicate* (optional): Controls what action is carried out in case of a unique key constraint violation. Possible values are:
- *error*: this will not import the current document because of the unique key constraint violation. This is the default setting.
- *update*: this will update an existing document in the database with the data specified in the request. Attributes of the existing document that are not present in the request will be preseved.
- *replace*: this will replace an existing document in the database with the data specified in the request.
- *ignore*: this will not update an existing document and simply ignore the error caused by the unique key constraint violation.

Note that *update*, *replace* and *ignore* will only work when the import document in the request contains the *_key* attribute. *update* and *replace* may also fail because of secondary unique key constraint violations.

- *complete* (optional): If set to `true` or `yes`, it will make the whole import fail if any error occurs. Otherwise the import will continue even if some documents cannot be imported.
- *details* (optional): If set to `true` or `yes`, the result will include an attribute `details` with details about documents that could not be imported.

NOTE Swagger examples won't work due to the anchor.

Creates documents in the collection identified by `collection-name`. The first line of the request body must contain a JSON-encoded array of attribute names. All following lines in the request body must contain JSON-encoded arrays of attribute values. Each line is interpreted as a separate document, and the values specified will be mapped to the array of attribute names specified in the first header line.

The response is a JSON object with the following attributes:

- `created`: number of documents imported.
- `errors`: number of documents that were not imported due to an error.
- `empty`: number of empty lines found in the input (will only contain a value greater zero for types `documents` or `auto`).
- `updated`: number of updated/replaced documents (in case `onDuplicate` was set to either `update` or `replace`).
- `ignored`: number of failed but ignored insert operations (in case `onDuplicate` was set to `ignore`).
- `details`: if query parameter `details` is set to true, the result will contain a `details` attribute which is an array with more detailed information about which documents could not be inserted.

Return Codes

- *201*: is returned if all documents could be imported successfully.
- *400*: is returned if `type` contains an invalid value, no `collection` is specified, the documents are incorrectly encoded, or the request is malformed.
- *404*: is returned if `collection` or the `_from` or `_to` attributes of an imported edge refer to an unknown collection.
- *409*: is returned if the import would trigger a unique key violation and `complete` is set to `true`.
- *500*: is returned if the server cannot auto-generate a document key (out of keys error) for a document with no user-defined key.

Examples

Importing two documents, with attributes `_key`, `value1` and `value2` each. One line in the import data is empty

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products <<EOF
[ "_key", "value1", "value2" ]
[ "abc", 25, "test" ]

[ "foo", "bar", "baz" ]
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Importing into an edge collection, with attributes `_from`, `_to` and `name`

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
```

```
collection=links <<EOF
[ "_from", "_to", "name" ]
[ "products/123","products/234", "some name" ]
[ "products/332", "products/abc", "other name" ]
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Importing into an edge collection, omitting `_from` or `_to`

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=links&details=true <<EOF
[ "name" ]
[ "some name" ]
[ "other name" ]
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Violating a unique constraint, but allow partial imports

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products&details=true <<EOF
[ "_key", "value1", "value2" ]
[ "abc", 25, "test" ]
["abc", "bar", "baz" ]
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Violating a unique constraint, not allowing partial imports

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products&complete=true <<EOF
[ "_key", "value1", "value2" ]
[ "abc", 25, "test" ]
["abc", "bar", "baz" ]
EOF

HTTP/1.1 409 Conflict
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Using a non-existing collection

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products <<EOF
[ "_key", "value1", "value2" ]
[ "abc", 25, "test" ]
["foo", "bar", "baz" ]
EOF

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Using a malformed body

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products <<EOF
{ "_key": "foo", "value1": "bar" }
EOF

HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

imports documents from JSON

imports documents from JSON

`POST /_api/import#json`

Request Body (required)

The body must either be a JSON-encoded array of objects or a string with multiple JSON objects separated by newlines.

Query Parameters

- *type* (required): Determines how the body of the request will be interpreted. `type` can have the following values:
- `documents`: when this type is used, each line in the request body is expected to be an individual JSON-encoded document. Multiple JSON objects in the request body need to be separated by newlines.
- `list`: when this type is used, the request body must contain a single JSON-encoded array of individual objects to import.
- `auto`: if set, this will automatically determine the body type (either `documents` or `list`).
- *collection* (required): The collection name.
- *fromPrefix* (optional): An optional prefix for the values in `_from` attributes. If specified, the value is automatically prepended to each `_from` input value. This allows specifying just the keys for `_from`.
- *toPrefix* (optional): An optional prefix for the values in `_to` attributes. If specified, the value is automatically prepended to each `_to` input value. This allows specifying just the keys for `_to`.
- *overwrite* (optional): If this parameter has a value of `true` or `yes`, then all data in the collection will be removed prior to the import. Note that any existing index definitions will be preseved.
- *waitForSync* (optional): Wait until documents have been synced to disk before returning.

- *onDuplicate* (optional): Controls what action is carried out in case of a unique key constraint violation. Possible values are:

- *error*: this will not import the current document because of the unique key constraint violation. This is the default setting.
- *update*: this will update an existing document in the database with the data specified in the request. Attributes of the existing document that are not present in the request will be preseved.
- *replace*: this will replace an existing document in the database with the data specified in the request.
- *ignore*: this will not update an existing document and simply ignore the error caused by a unique key constraint violation.

Note that that *update*, *replace* and *ignore* will only work when the import document in the request contains the *_key* attribute. *update* and *replace* may also fail because of secondary unique key constraint violations.

- *complete* (optional): If set to `true` or `yes`, it will make the whole import fail if any error occurs. Otherwise the import will continue even if some documents cannot be imported.

- *details* (optional): If set to `true` or `yes`, the result will include an attribute `details` with details about documents that could not be imported.

NOTE Swagger examples won't work due to the anchor.

Creates documents in the collection identified by `collection-name`. The JSON representations of the documents must be passed as the body of the POST request. The request body can either consist of multiple lines, with each line being a single stand-alone JSON object, or a singe JSON array with sub-objects.

The response is a JSON object with the following attributes:

- `created`: number of documents imported.

- `errors`: number of documents that were not imported due to an error.

- `empty`: number of empty lines found in the input (will only contain a value greater zero for types `documents` or `auto`).

- `updated`: number of updated/replaced documents (in case `onDuplicate` was set to either `update` or `replace`).

- `ignored`: number of failed but ignored insert operations (in case `onDuplicate` was set to `ignore`).

- `details`: if query parameter `details` is set to true, the result will contain a `details` attribute which is an array with more detailed information about which documents could not be inserted.

Return Codes

- *201*: is returned if all documents could be imported successfully.

- *400*: is returned if `type` contains an invalid value, no `collection` is specified, the documents are incorrectly encoded, or the request is malformed.

- *404*: is returned if `collection` or the `_from` or `_to` attributes of an imported edge refer to an unknown collection.

- *409*: is returned if the import would trigger a unique key violation and `complete` is set to `true`.

- *500*: is returned if the server cannot auto-generate a document key (out of keys error) for a document with no user-defined key.

Examples

Importing documents with heterogenous attributes from a JSON array

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products&type=list <<EOF
[
  {
    "_key" : "abc",
    "value1" : 25,
    "value2" : "test",
    "allowed" : true
  },
  {
    "_key" : "foo",
```

```
    "name" : "baz"
  },
  {
    "name" : {
      "detailed" : "detailed name",
      "short" : "short name"
    }
  }
]
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Importing documents from individual JSON lines

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products&type=documents <<EOF
{ "_key": "abc", "value1": 25, "value2": "test","allowed": true }
{ "_key": "foo", "name": "baz" }

{ "name": { "detailed": "detailed name", "short": "short name" } }

EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Using the auto type detection

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products&type=auto <<EOF
[
  {
    "_key" : "abc",
    "value1" : 25,
    "value2" : "test",
    "allowed" : true
  },
  {
    "_key" : "foo",
    "name" : "baz"
  },
  {
    "name" : {
      "detailed" : "detailed name",
      "short" : "short name"
    }
  }
```

```
]
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Importing into an edge collection, with attributes `_from`, `_to` and `name`

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=links&type=documents <<EOF
{ "_from": "products/123", "_to": "products/234" }
{"_from": "products/332", "_to": "products/abc",   "name": "other name" }
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Importing into an edge collection, omitting `_from` or `_to`

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=links&type=list&details=true <<EOF
[
  {
    "name" : "some name"
  }
]
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Violating a unique constraint, but allow partial imports

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products&type=documents&details=true <<EOF
{ "_key": "abc", "value1": 25, "value2": "test" }
{ "_key": "abc", "value1": "bar", "value2": "baz" }
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Violating a unique constraint, not allowing partial imports

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products&type=documents&complete=true <<EOF
{ "_key": "abc", "value1": 25, "value2": "test" }
{ "_key": "abc", "value1": "bar", "value2": "baz" }
EOF

HTTP/1.1 409 Conflict
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Using a non-existing collection

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products&type=documents <<EOF
{ "name": "test" }
EOF

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Using a malformed body

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/import?
collection=products&type=list <<EOF
{ }
EOF

HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Importing Self-Contained JSON Documents

This import method allows uploading self-contained JSON documents. The documents must be uploaded in the body of the HTTP POST request. Each line of the body will be interpreted as one stand-alone document. Empty lines in the body are allowed but will be skipped. Using this format, the documents are imported line-wise.

Example input data: { "_key": "key1", ... } { "_key": "key2", ... } ...

To use this method, the *type* query parameter should be set to *documents*.

It is also possible to upload self-contained JSON documents that are embedded into a JSON array. Each element from the array will be treated as a document and be imported.

Example input data for this case:

```
[
  { "_key": "key1", ... },
  { "_key": "key2", ... },
  ...
]
```

This format does not require each document to be on a separate line, and any whitespace in the JSON data is allowed. It can be used to import a JSON-formatted result array (e.g. from arangosh) back into ArangoDB. Using this format requires ArangoDB to parse the complete array and keep it in memory for the duration of the import. This might be more resource-intensive than the line-wise processing.

To use this method, the *type* query parameter should be set to *array*.

Setting the *type* query parameter to *auto* will make the server auto-detect whether the data are line-wise JSON documents (type = documents) or a JSON array (type = array).

Examples

```
curl --data-binary @- -X POST --dump - "http://localhost:8529/_api/import?type=documents&collection=test"
{ "name" : "test", "gender" : "male", "age" : 39 }
{ "type" : "bird", "name" : "robin" }

HTTP/1.1 201 Created
Server: ArangoDB
Connection: Keep-Alive
Content-type: application/json; charset=utf-8

{"error":false,"created":2,"empty":0,"errors":0}
```

The server will respond with an HTTP 201 if everything went well. The number of documents imported will be returned in the *created* attribute of the response. If any documents were skipped or incorrectly formatted, this will be returned in the *errors* attribute. There will also be an attribute *empty* in the response, which will contain a value of *0*.

If the *details* parameter was set to *true* in the request, the response will also contain an attribute *details* which is an array of details about errors that occurred on the server side during the import. This array might be empty if no errors occurred.

Importing Headers and Values

When using this type of import, the attribute names of the documents to be imported are specified separate from the actual document value data. The first line of the HTTP POST request body must be a JSON array containing the attribute names for the documents that follow. The following lines are interpreted as the document data. Each document must be a JSON array of values. No attribute names are needed or allowed in this data section.

Examples

```
curl --data-binary @- -X POST --dump - "http://localhost:8529/_api/import?collection=test"
[ "firstName", "lastName", "age", "gender" ]
[ "Joe", "Public", 42, "male" ]
[ "Jane", "Doe", 31, "female" ]

HTTP/1.1 201 Created
Server: ArangoDB
Connection: Keep-Alive
Content-type: application/json; charset=utf-8

{"error":false,"created":2,"empty":0,"errors":0}
```

The server will again respond with an HTTP 201 if everything went well. The number of documents imported will be returned in the *created* attribute of the response. If any documents were skipped or incorrectly formatted, this will be returned in the *errors* attribute. The number of empty lines in the input file will be returned in the *empty* attribute.

If the *details* parameter was set to *true* in the request, the response will also contain an attribute *details* which is an array of details about errors that occurred on the server side during the import. This array might be empty if no errors occurred.

Importing into Edge Collections

Please note that when importing documents into an edge collection, it is mandatory that all imported documents contain the *_from* and *_to* attributes, and that these contain references to existing collections.

HTTP Interface for Batch Requests

Clients normally send individual operations to ArangoDB in individual HTTP requests. This is straightforward and simple, but has the disadvantage that the network overhead can be significant if many small requests are issued in a row.

To mitigate this problem, ArangoDB offers a batch request API that clients can use to send multiple operations in one batch to ArangoDB. This method is especially useful when the client has to send many HTTP requests with a small body/payload and the individual request results do not depend on each other.

Clients can use ArangoDB's batch API by issuing a multipart HTTP POST request to the URL */_api/batch* handler. The handler will accept the request if the Content-type is *multipart/form-data* and a boundary string is specified. ArangoDB will then decompose the batch request into its individual parts using this boundary. This also means that the boundary string itself must not be contained in any of the parts. When ArangoDB has split the multipart request into its individual parts, it will process all parts sequentially as if it were a standalone request. When all parts are processed, ArangoDB will generate a multipart HTTP response that contains one part for each part operation result. For example, if you send a multipart request with 5 parts, ArangoDB will send back a multipart response with 5 parts as well.

The server expects each part message to start with exactly the following "header":

```
Content-type: application/x-arango-batchpart
```

You can optionally specify a *Content-Id* "header" to uniquely identify each part message. The server will return the *Content-Id* in its response if it is specified. Otherwise, the server will not send a Content-Id "header" back. The server will not validate the uniqueness of the Content-Id. After the mandatory *Content-type* and the optional *Content-Id* header, two Windows line breaks (i.e. *\r\n\r\n*) must follow. Any deviation of this structure might lead to the part being rejected or incorrectly interpreted. The part request payload, formatted as a regular HTTP request, must follow the two Windows line breaks literal directly.

Note that the literal *Content-type: application/x-arango-batchpart* technically is the header of the MIME part, and the HTTP request (including its headers) is the body part of the MIME part.

An actual part request should start with the HTTP method, the called URL, and the HTTP protocol version as usual, followed by arbitrary HTTP headers. Its body should follow after the usual *\r\n\r\n* literal. Part requests are therefore regular HTTP requests, only embedded inside a multipart message.

The following example will send a batch with 3 individual document creation operations. The boundary used in this example is *XXXsubpartXXX*.

Examples

```
> curl -X POST --data-binary @- --header "Content-type: multipart/form-data; boundary=XXXsubpartXXX" http://localhost:8529/_api
/batch
--XXXsubpartXXX
Content-type: application/x-arango-batchpart
Content-Id: 1

POST /_api/document?collection=xyz HTTP/1.1

{"a":1,"b":2,"c":3}
--XXXsubpartXXX
Content-type: application/x-arango-batchpart
Content-Id: 2

POST /_api/document?collection=xyz HTTP/1.1

{"a":1,"b":2,"c":3,"d":4}
--XXXsubpartXXX
Content-type: application/x-arango-batchpart
Content-Id: 3

POST /_api/document?collection=xyz HTTP/1.1

{"a":1,"b":2,"c":3,"d":4,"e":5}
--XXXsubpartXXX--
```

The server will then respond with one multipart message, containing the overall status and the individual results for the part operations. The overall status should be 200 except there was an error while inspecting and processing the multipart message. The overall status therefore does not indicate the success of each part operation, but only indicates whether the multipart message could be handled successfully.

Each part operation will return its own status value. As the part operation results are regular HTTP responses (just included in one multipart response), the part operation status is returned as a HTTP status code. The status codes of the part operations are exactly the same as if you called the individual operations standalone. Each part operation might also return arbitrary HTTP headers and a body/payload:

Examples

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-type: multipart/form-data; boundary=XXXsubpartXXX
Content-length: 1055

--XXXsubpartXXX
Content-type: application/x-arango-batchpart
Content-Id: 1

HTTP/1.1 202 Accepted
Content-type: application/json; charset=utf-8
Etag: "9514299"
Content-length: 53

{"error":false,"_id":"xyz/9514299","_key":"9514299","_rev":"9514299"}
--XXXsubpartXXX
Content-type: application/x-arango-batchpart
Content-Id: 2

HTTP/1.1 202 Accepted
Content-type: application/json; charset=utf-8
Etag: "9579835"
Content-length: 53

{"error":false,"_id":"xyz/9579835","_key":"9579835","_rev":"9579835"}
--XXXsubpartXXX
Content-type: application/x-arango-batchpart
Content-Id: 3

HTTP/1.1 202 Accepted
Content-type: application/json; charset=utf-8
Etag: "9645371"
Content-length: 53

{"error":false,"_id":"xyz/9645371","_key":"9645371","_rev":"9645371"}
--XXXsubpartXXX--
```

In the above example, the server returned an overall status code of 200, and each part response contains its own status value (202 in the example):

When constructing the multipart HTTP response, the server will use the same boundary that the client supplied. If any of the part responses has a status code of 400 or greater, the server will also return an HTTP header *x-arango-errors* containing the overall number of part requests that produced errors:

Examples

```
> curl -X POST --data-binary @- --header "Content-type: multipart/form-data; boundary=XXXsubpartXXX" http://localhost:8529/_api
/batch
--XXXsubpartXXX
Content-type: application/x-arango-batchpart

POST /_api/document?collection=nonexisting

{"a":1,"b":2,"c":3}
--XXXsubpartXXX
Content-type: application/x-arango-batchpart

POST /_api/document?collection=xyz
```

```
{"a":1,"b":2,"c":3,"d":4}
--XXXsubpartXXX--
```

In this example, the overall response code is 200, but as some of the part request failed (with status code 404), the *x-arango-errors* header of the overall response is *1*:

Examples

```
HTTP/1.1 200 OK
x-arango-errors: 1
Content-type: multipart/form-data; boundary=XXXsubpartXXX
Content-length: 711

--XXXsubpartXXX
Content-type: application/x-arango-batchpart

HTTP/1.1 404 Not Found
Content-type: application/json; charset=utf-8
Content-length: 111

{"error":true,"code":404,"errorNum":1203,"errorMessage":"collection \/_api\/collection\/nonexisting not found"}
--XXXsubpartXXX
Content-type: application/x-arango-batchpart

HTTP/1.1 202 Accepted
Content-type: application/json; charset=utf-8
Etag: "9841979"
Content-length: 53

{"error":false,"_id":"xyz/9841979","_key":"9841979","_rev":"9841979"}
--XXXsubpartXXX--
```

Please note that the database used for all part operations of a batch request is determined by scanning the original URL (the URL that contains */_api/batch*). It is not possible to override the database name in part operations of a batch. When doing so, any other database name used in a batch part will be ignored.

executes a batch request

executes a batch request

`POST /_api/batch`

Request Body (required)

The multipart batch request, consisting of the envelope and the individual batch parts.

Executes a batch request. A batch request can contain any number of other requests that can be sent to ArangoDB in isolation. The benefit of using batch requests is that batching requests requires less client/server roundtrips than when sending isolated requests.

All parts of a batch request are executed serially on the server. The server will return the results of all parts in a single response when all parts are finished.

Technically, a batch request is a multipart HTTP request, with content-type `multipart/form-data`. A batch request consists of an envelope and the individual batch part actions. Batch part actions are "regular" HTTP requests, including full header and an optional body. Multiple batch parts are separated by a boundary identifier. The boundary identifier is declared in the batch envelope. The MIME content-type for each individual batch part must be `application/x-arango-batchpart`.

Please note that when constructing the individual batch parts, you must use CRLF (`\ \`) as the line terminator as in regular HTTP messages.

The response sent by the server will be an `HTTP 200` response, with an optional error summary header `x-arango-errors`. This header contains the number of batch part operations that failed with an HTTP error code of at least 400. This header is only present in the response if the number of errors is greater than zero.

The response sent by the server is a multipart response, too. It contains the individual HTTP responses for all batch parts, including the full HTTP result header (with status code and other potential headers) and an optional result body. The individual batch parts in the result are seperated using the same boundary value as specified in the request.

The order of batch parts in the response will be the same as in the original client request. Client can additionally use the `Content-Id` MIME header in a batch part to define an individual id for each batch part. The server will return this id is the batch part responses, too.

Return Codes

- *200*: is returned if the batch was received successfully. HTTP 200 is returned even if one or multiple batch part actions failed.
- *400*: is returned if the batch envelope is malformed or incorrectly formatted. This code will also be returned if the content-type of the overall batch request or the individual MIME parts is not as expected.
- *405*: is returned when an invalid HTTP method is used.

Examples

Sending a batch request with five batch parts:

- GET /_api/version
- DELETE /_api/collection/products
- POST /_api/collection/products
- GET /_api/collection/products/figures
- DELETE /_api/collection/products

The boundary (`SomeBoundaryValue`) is passed to the server in the HTTP `Content-Type` HTTP header. *Please note the reply is not displayed all accurate.*

```
shell> curl -X POST --header 'Content-Type: multipart/form-data;
boundary=SomeBoundaryValue' --data-binary @- --dump - http://localhost:8529/_api/batch
<<EOF
--SomeBoundaryValue
Content-Type: application/x-arango-batchpart
Content-Id: myId1

GET /_api/version HTTP/1.1
--SomeBoundaryValue
Content-Type: application/x-arango-batchpart
Content-Id: myId2

DELETE /_api/collection/products HTTP/1.1
--SomeBoundaryValue
Content-Type: application/x-arango-batchpart
Content-Id: someId

POST /_api/collection/products HTTP/1.1

{"name": "products" }
--SomeBoundaryValue
Content-Type: application/x-arango-batchpart
Content-Id: nextId

GET /_api/collection/products/figures HTTP/1.1
--SomeBoundaryValue
Content-Type: application/x-arango-batchpart
Content-Id: otherId

DELETE /_api/collection/products HTTP/1.1
--SomeBoundaryValue--

EOF

HTTP/1.1 200 OK
```

```
x-content-type-options: nosniff
content-type: multipart/form-data; boundary=SomeBoundaryValue
x-arango-errors: 1

"--SomeBoundaryValue\r\nContent-Type: application/x-arango-batchpart\r\nContent-Id:
myId1\r\n\r\nHTTP/1.1 200 OK\r\nServer: \r\nConnection: \r\nContent-Type:
application/json; charset=utf-8\r\nContent-Length:
60\r\n\r\n{\"server\":\"arango\",\"version\":\"3.3.19\",\"license\":\"community\"}\r\n--
SomeBoundaryValue\r\nContent-Type: application/x-arango-batchpart\r\nContent-Id:
myId2\r\n\r\nHTTP/1.1 404 Not Found\r\nServer: \r\nConnection: \r\nContent-Type:
application/json; charset=utf-8\r\nContent-Length:
79\r\n\r\n{\"error\":true,\"errorMessage\":\"collection not
found\",\"code\":404,\"errorNum\":1203}\r\n--SomeBoundaryValue\r\nContent-Type:
application/x-arango-batchpart\r\nContent-Id: someId\r\n\r\nHTTP/1.1 200 OK\r\nServer:
\r\nConnection: \r\nContent-Type: application/json; charset=utf-8\r\nContent-Length:
324\r\n\r\n{\"code\":200,\"error\":false,\"status\":3,\"statusString\":\"loaded\",\"name\"
:\"products\",\"keyOptions\":
{\"type\":\"traditional\",\"allowUserKeys\":true,\"lastValue\":0},\"type\":2,\"indexBucket
s\":8,\"globallyUniqueId\":\"h5486C74C94E1/9839\",\"doCompact\":true,\"waitForSync\":false
,\"id\":\"9839\",\"isSystem\":false,\"journalSize\":33554432,\"isVolatile\":false}\r\n--
SomeBoundaryValue\r\nContent-Type: application/x-arango-batchpart\r\nContent-Id:
nextId\r\n\r\nHTTP/1.1 200 OK\r\nServer: \r\nLocation:
/_api/collection/products/figures\r\nConnection: \r\nContent-Type: application/json;
charset=utf-8\r\nContent-Length:
831\r\n\r\n{\"code\":200,\"error\":false,\"statusString\":\"loaded\",\"name\":\"products\"
,\"keyOptions\":
{\"type\":\"traditional\",\"allowUserKeys\":true,\"lastValue\":0},\"journalSize\":33554432
,\"isVolatile\":false,\"isSystem\":false,\"status\":3,\"count\":0,\"figures\":
{\"indexes\":{\"count\":1,\"size\":32128},\"documentReferences\":0,\"waitingFor\":\"-
\",\"alive\":{\"count\":0,\"size\":0},\"dead\":
{\"count\":0,\"size\":0,\"deletion\":0},\"compactionStatus\":{\"message\":\"compaction not
yet started\",\"time\":\"2018-10-
20T10:39:04Z\",\"count\":0,\"filesCombined\":0,\"bytesRead\":0,\"bytesWritten\":0},\"dataf
iles\":{\"count\":0,\"fileSize\":0},\"journals\":
{\"count\":0,\"fileSize\":0},\"compactors\":{\"count\":0,\"fileSize\":0},\"revisions\":
{\"count\":0,\"size\":48192},\"lastTick\":0,\"uncollectedLogfileEntries\":0},\"doCompact\"
:true,\"globallyUniqueId\":\"h5486C74C94E1/9839\",\"type\":2,\"indexBuckets\":8,\"waitForS
ync\":false,\"id\":\"9839\"}\r\n--SomeBoundaryValue\r\nContent-Type: application/x-arango-
batchpart\r\nContent-Id: otherId\r\n\r\nHTTP/1.1 200 OK\r\nServer: \r\nConnection:
\r\nContent-Type: application/json; charset=utf-8\r\nContent-Length:
38\r\n\r\n{\"code\":200,\"error\":false,\"id\":\"9839\"}\r\n--SomeBoundaryValue--"
```

Sending a batch request, setting the boundary implicitly (the server will in this case try to find the boundary at the beginning of the request body).

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/batch <<EOF
--SomeBoundaryValue
Content-Type: application/x-arango-batchpart

DELETE /_api/collection/notexisting1 HTTP/1.1
--SomeBoundaryValue
Content-Type: application/x-arango-batchpart

DELETE _api/collection/notexisting2 HTTP/1.1
--SomeBoundaryValue--
```

```
EOF

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type:
x-arango-errors: 2

"--SomeBoundaryValue\r\nContent-Type: application/x-arango-batchpart\r\n\r\nHTTP/1.1 404
Not Found\r\nServer: \r\nConnection: \r\nContent-Type: application/json; charset=utf-
8\r\nContent-Length: 79\r\n\r\n{\"error\":true,\"errorMessage\":\"collection not
found\",\"code\":404,\"errorNum\":1203}\r\n--SomeBoundaryValue\r\nContent-Type:
application/x-arango-batchpart\r\n\r\nHTTP/1.1 404 Not Found\r\nServer: \r\nConnection:
\r\nContent-Type: application/json; charset=utf-8\r\nContent-Length:
101\r\n\r\n{\"error\":true,\"code\":404,\"errorNum\":404,\"errorMessage\":\"unknown path
'_api/collection/notexisting2'\"}\r\n--SomeBoundaryValue--"
```

HTTP Interface for Exporting Documents

Create export cursor

export all documents from a collection, using a cursor

`POST /_api/export`

A JSON object with these properties is required:

- **count**: boolean flag that indicates whether the number of documents in the result set should be returned in the "count" attribute of the result (optional). Calculating the "count" attribute might in the future have a performance impact so this option is turned off by default, and "count" is only returned when requested.
- **restrict**:
  - **fields** (string): Contains an array of attribute names to *include* or *exclude*. Matching of attribute names for *inclusion* or *exclusion* will be done on the top level only. Specifying names of nested attributes is not supported at the moment.
  - **type**: has to be be set to either *include* or *exclude* depending on which you want to use
- **batchSize**: maximum number of result documents to be transferred from the server to the client in one roundtrip (optional). If this attribute is not set, a server-controlled default value will be used.
- **flush**: if set to *true*, a WAL flush operation will be executed prior to the export. The flush operation will start copying documents from the WAL to the collection's datafiles. There will be an additional wait time of up to *flushWait* seconds after the flush to allow the WAL collector to change the adjusted document meta-data to point into the datafiles, too. The default value is *false* (i.e. no flush) so most recently inserted or updated documents from the collection might be missing in the export.
- **flushWait**: maximum wait time in seconds after a flush operation. The default value is 10. This option only has an effect when *flush* is set to *true*.
- **limit**: an optional limit value, determining the maximum number of documents to be included in the cursor. Omitting the *limit* attribute or setting it to 0 will lead to no limit being used. If a limit is used, it is undefined which documents from the collection will be included in the export and which will be excluded. This is because there is no natural order of documents in a collection.
- **ttl**: an optional time-to-live for the cursor (in seconds). The cursor will be removed on the server automatically after the specified amount of time. This is useful to ensure garbage collection of cursors that are not fully fetched by clients. If not set, a server-defined value will be used.

A call to this method creates a cursor containing all documents in the specified collection. In contrast to other data-producing APIs, the internal data structures produced by the export API are more lightweight, so it is the preferred way to retrieve all documents from a collection.

Documents are returned in a similar manner as in the `/_api/cursor` REST API. If all documents of the collection fit into the first batch, then no cursor will be created, and the result object's *hasMore* attribute will be set to *false*. If not all documents fit into the first batch, then the result object's *hasMore* attribute will be set to *true*, and the *id* attribute of the result will contain a cursor id.

The order in which the documents are returned is not specified.

By default, only those documents from the collection will be returned that are stored in the collection's datafiles. Documents that are present in the write-ahead log (WAL) at the time the export is run will not be exported.

To export these documents as well, the caller can issue a WAL flush request before calling the export API or set the *flush* attribute. Setting the *flush* option will trigger a WAL flush before the export so documents get copied from the WAL to the collection datafiles.

If the result set can be created by the server, the server will respond with *HTTP 201*. The body of the response will contain a JSON object with the result set.

The returned JSON object has the following properties:

- *error*: boolean flag to indicate that an error occurred (*false* in this case)
- *code*: the HTTP status code
- *result*: an array of result documents (might be empty if the collection was empty)
- *hasMore*: a boolean indicator whether there are more results available for the cursor on the server
- *count*: the total number of result documents available (only available if the query was executed with the *count* attribute set)

- *id*: id of temporary cursor created on the server (optional, see above)

If the JSON representation is malformed or the query specification is missing from the request, the server will respond with *HTTP 400*.

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- *error*: boolean flag to indicate that an error occurred (*true* in this case)
- *code*: the HTTP status code
- *errorNum*: the server error number
- *errorMessage*: a descriptive error message

Clients should always delete an export cursor result as early as possible because a lingering export cursor will prevent the underlying collection from being compacted or unloaded. By default, unused cursors will be deleted automatically after a server-defined idle time, and clients can adjust this idle time by setting the *ttl* value.

Note: this API is currently not supported on cluster coordinators.

Query Parameters

- *collection* (required): The name of the collection to export.

Return Codes

- *201*: is returned if the result set can be created by the server.
- *400*: is returned if the JSON representation is malformed or the query specification is missing from the request.
- *404*: The server will respond with *HTTP 404* in case a non-existing collection is accessed in the query.
- *405*: The server will respond with *HTTP 405* if an unsupported HTTP method is used.
- *501*: The server will respond with *HTTP 501* if this API is called on a cluster coordinator.

HTTP Interface for Indexes

Indexes

This is an introduction to ArangoDB's HTTP interface for indexes in general. There are special sections for various index types.

Index

Indexes are used to allow fast access to documents. For each collection there is always the primary index which is a hash index for the document key (_key attribute). This index cannot be dropped or changed. edge collections will also have an automatically created edges index, which cannot be modified. This index provides quick access to documents via the `_from` and `_to` attributes.

Most user-land indexes can be created by defining the names of the attributes which should be indexed. Some index types allow indexing just one attribute (e.g. fulltext index) whereas other index types allow indexing multiple attributes.

Using the system attribute `_id` in user-defined indexes is not supported by any index type.

Index Handle

An index handle uniquely identifies an index in the database. It is a string and consists of a collection name and an index identifier separated by /. Geo Index: A geo index is used to find places on the surface of the earth fast. Hash Index: A hash index is used to find documents based on examples. A hash index can be created for one or multiple document attributes. A hash index will only be used by queries if all indexed attributes are present in the example or search query, and if all attributes are compared using the equality (== operator). That means the hash index does not support range queries.

If the index is declared unique, then access to the indexed attributes should be fast. The performance degrades if the indexed attribute(s) contain(s) only very few distinct values.

Edges Index

An edges index is automatically created for edge collections. It contains connections between vertex documents and is invoked when the connecting edges of a vertex are queried. There is no way to explicitly create or delete edge indexes.

Skiplist Index

A skiplist is a sorted index that can be used to find individual documents or ranges of documents.

Persistent Index

A persistent index is a sorted index that can be used for finding individual documents or ranges of documents. In constrast to the other indexes, the contents of a persistent index are stored on disk and thus do not need to be rebuilt in memory from the documents when the collection is loaded.

Fulltext Index:

A fulltext index can be used to find words, or prefixes of words inside documents. A fulltext index can be set on one attribute only, and will index all words contained in documents that have a textual value in this attribute. Only words with a (specifiable) minimum length are indexed. Word tokenization is done using the word boundary analysis provided by libicu, which is taking into account the selected language provided at server start. Words are indexed in their lower-cased form. The index supports complete match queries (full words) and prefix queries.

The basic operations (create, read, update, delete) for documents are mapped to the standard HTTP methods (*POST*, *GET*, *PUT*, *DELETE*).

Address of an Index

All indexes in ArangoDB have an unique handle. This index handle identifies an index and is managed by ArangoDB. All indexes are found under the URI

```
http://server:port/_api/index/index-handle
```

For example: Assume that the index handle is *demo/63563528* then the URL of that index is:

```
http://localhost:8529/_api/index/demo/63563528
```

Working with Indexes using HTTP

Read index

returns an index

`GET /_api/index/{index-handle}`

Path Parameters

- *index-handle* (required): The index-handle.

The result is an object describing the index. It has at least the following attributes:

- *id*: the identifier of the index
- *type*: the index type

All other attributes are type-dependent. For example, some indexes provide *unique* or *sparse* flags, whereas others don't. Some indexes also provide a selectivity estimate in the *selectivityEstimate* attribute of the result.

Return Codes

- *200*: If the index exists, then a *HTTP 200* is returned.
- *404*: If the index does not exist, then a *HTTP 404* is returned.

Examples

```
shell> curl --dump - http://localhost:8529/_api/index/products/0

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Create index

creates an index

`POST /_api/index#general`

Query Parameters

- *collection* (required): The collection name.

Request Body (required)

NOTE Swagger examples won't work due to the anchor.

Creates a new index in the collection *collection*. Expects an object containing the index details.

The type of the index to be created must specified in the *type* attribute of the index details. Depending on the index type, additional other attributes may need to specified in the request in order to create the index.

Indexes require the to be indexed attribute(s) in the *fields* attribute of the index details. Depending on the index type, a single attribute or multiple attributes can be indexed. In the latter case, an array of strings is expected.

Indexing the system attribute *_id* is not supported for user-defined indexes. Manually creating an index using *_id* as an index attribute will fail with an error.

Some indexes can be created as unique or non-unique variants. Uniqueness can be controlled for most indexes by specifying the *unique* flag in the index details. Setting it to *true* will create a unique index. Setting it to *false* or omitting the *unique* attribute will create a non-unique index.

Note: The following index types do not support uniqueness, and using the *unique* attribute with these types may lead to an error:

- geo indexes
- fulltext indexes

Note: Unique indexes on non-shard keys are not supported in a cluster.

Hash, skiplist and persistent indexes can optionally be created in a sparse variant. A sparse index will be created if the *sparse* attribute in the index details is set to *true*. Sparse indexes do not index documents for which any of the index attributes is either not set or is *null*.

The optional attribute **deduplicate** is supported by array indexes of type *hash* or *skiplist*. It controls whether inserting duplicate index values from the same document into a unique array index will lead to a unique constraint error or not. The default value is *true*, so only a single instance of each non-unique index value will be inserted into the index per document. Trying to insert a value into the index that already exists in the index will always fail, regardless of the value of this attribute.

Return Codes

- *200*: If the index already exists, then an *HTTP 200* is returned.
- *201*: If the index does not already exist and could be created, then an *HTTP 201* is returned.
- *400*: If an invalid index description is posted or attributes are used that the target index will not support, then an *HTTP 400* is returned.
- *404*: If *collection* is unknown, then an *HTTP 404* is returned.

Delete index

deletes an index

`DELETE /_api/index/{index-handle}`

Path Parameters

- *index-handle* (required): The index handle.

Deletes an index with *index-handle*.

Return Codes

- *200*: If the index could be deleted, then an *HTTP 200* is returned.
- *404*: If the *index-handle* is unknown, then an *HTTP 404* is returned.

Examples

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/index/products/11236

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Read all indexes of a collection

returns all indexes of a collection

`GET /_api/index`

Query Parameters

- *collection* (required): The collection name.

Returns an object with an attribute *indexes* containing an array of all index descriptions for the given collection. The same information is also available in the *identifiers* as an object with the index handles as keys.

Return Codes

- *200*: returns a JSON object containing a list of indexes on that collection.

Examples

Return information about all indexes

```
shell> curl --dump - http://localhost:8529/_api/index?collection=products

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Working with Hash Indexes

If a suitable hash index exists, then */_api/simple/by-example* will use this index to execute a query-by-example.

Create hash index

creates a hash index

`POST /_api/index#hash`

Query Parameters

- *collection-name* (required): The collection name.

A JSON object with these properties is required:

- **fields** (string): an array of attribute paths.
- **unique**: if *true*, then create a unique index.
- **type**: must be equal to *"hash"*.
- **sparse**: if *true*, then create a sparse index.
- **deduplicate**: if *false*, the deduplication of array values is turned off.

NOTE Swagger examples won't work due to the anchor.

Creates a hash index for the collection *collection-name* if it does not already exist. The call expects an object containing the index details.

In a sparse index all documents will be excluded from the index that do not contain at least one of the specified index attributes (i.e. *fields*) or that have a value of *null* in any of the specified index attributes. Such documents will not be indexed, and not be taken into account for uniqueness checks if the *unique* flag is set.

In a non-sparse index, these documents will be indexed (for non-present indexed attributes, a value of *null* will be used) and will be taken into account for uniqueness checks if the *unique* flag is set.

Note: unique indexes on non-shard keys are not supported in a cluster.

Return Codes

- *200*: If the index already exists, then a *HTTP 200* is returned.
- *201*: If the index does not already exist and could be created, then a *HTTP 201* is returned.
- *400*: If the collection already contains documents and you try to create a unique hash index in such a way that there are documents violating the uniqueness, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Creating an unique constraint

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/index?
collection=products <<EOF
{
  "type" : "hash",
  "unique" : true,
  "fields" : [
    "a",
    "b"
  ]
}
EOF

HTTP/1.1 201 Created
```

```
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body

Creating a non-unique hash index

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/index?
collection=products <<EOF
{
  "type" : "hash",
  "unique" : false,
  "fields" : [
    "a",
    "b"
  ]
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body

Creating a sparse index

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/index?
collection=products <<EOF
{
  "type" : "hash",
  "unique" : false,
  "sparse" : true,
  "fields" : [
    "a"
  ]
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body

Simple query by-example

returns all documents of a collection matching a given example

`PUT /_api/simple/by-example`

A JSON object with these properties is required:

- **skip**: The number of documents to skip in the query (optional).
- **batchSize**: maximum number of result documents to be transferred from the server to the client in one roundtrip. If this attribute is not set, a server-controlled default value will be used. A *batchSize* value of *0* is disallowed.

- **limit**: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. (optional)
- **example**: The example document.
- **collection**: The name of the collection to query.

This will find all documents matching a given example.

Returns a cursor containing the result, see Http Cursor for details.

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

Matching an attribute

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/by-example
<<EOF
{
  "collection" : "products",
  "example" : {
    "i" : 1
  }
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Matching an attribute which is a sub-document

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/by-example
<<EOF
{
  "collection" : "products",
  "example" : {
    "a.j" : 1
  }
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Matching an attribute within a sub-document

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/by-example
<<EOF
```

```
{
  "collection" : "products",
  "example" : {
    "a" : {
      "j" : 1
    }
  }
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Find documents matching an example

returns one document of a collection matching a given example

`PUT /_api/simple/first-example`

A JSON object with these properties is required:

- **example**: The example document.
- **collection**: The name of the collection to query.

This will return the first document matching a given example.

Returns a result containing the document or *HTTP 404* if no document matched the example.

If more than one document in the collection matches the specified example, only one of these documents will be returned, and it is undefined which of the matching documents is returned.

Return Codes

- *200*: is returned when the query was successfully executed.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

If a matching document was found

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/first-
example <<EOF
{
  "collection" : "products",
  "example" : {
    "i" : 1
  }
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

If no document was found

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/first-
example <<EOF
{
  "collection" : "products",
  "example" : {
    "l" : 1
  }
}
EOF

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Working with Skiplist Indexes

If a suitable skip-list index exists, then /_api/simple/range and other operations will use this index to execute queries.

Create skip list

creates a skip-list

`POST /_api/index#skiplist`

Query Parameters

- *collection-name* (required): The collection name.

A JSON object with these properties is required:

- **fields** (string): an array of attribute paths.
- **unique**: if *true*, then create a unique index.
- **type**: must be equal to *"skiplist"*.
- **sparse**: if *true*, then create a sparse index.
- **deduplicate**: if *false*, the deduplication of array values is turned off.

Creates a skip-list index for the collection *collection-name*, if it does not already exist. The call expects an object containing the index details.

In a sparse index all documents will be excluded from the index that do not contain at least one of the specified index attributes (i.e. *fields*) or that have a value of *null* in any of the specified index attributes. Such documents will not be indexed, and not be taken into account for uniqueness checks if the *unique* flag is set.

In a non-sparse index, these documents will be indexed (for non-present indexed attributes, a value of *null* will be used) and will be taken into account for uniqueness checks if the *unique* flag is set.

Note: unique indexes on non-shard keys are not supported in a cluster.

Return Codes

- *200*: If the index already exists, then a *HTTP 200* is returned.
- *201*: If the index does not already exist and could be created, then a *HTTP 201* is returned.
- *400*: If the collection already contains documents and you try to create a unique skip-list index in such a way that there are documents violating the uniqueness, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Creating a skiplist index

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/index?
collection=products <<EOF
{
  "type" : "skiplist",
  "unique" : false,
  "fields" : [
    "a",
    "b"
  ]
}
EOF

HTTP/1.1 201 Created
```

```
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body

Creating a sparse skiplist index

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/index?
collection=products <<EOF
{
  "type" : "skiplist",
  "unique" : false,
  "sparse" : true,
  "fields" : [
    "a"
  ]
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body

Working with Persistent Indexes

If a suitable persistent index exists, then /_api/simple/range and other operations will use this index to execute queries.

Create a persistent index

creates a persistent index

`POST /_api/index#persistent`

Query Parameters

- *collection-name* (required): The collection name.

A JSON object with these properties is required:

- **fields** (string): an array of attribute paths.
- **unique**: if *true*, then create a unique index.
- **type**: must be equal to *"persistent"*.
- **sparse**: if *true*, then create a sparse index.

Creates a persistent index for the collection *collection-name*, if it does not already exist. The call expects an object containing the index details.

In a sparse index all documents will be excluded from the index that do not contain at least one of the specified index attributes (i.e. *fields*) or that have a value of *null* in any of the specified index attributes. Such documents will not be indexed, and not be taken into account for uniqueness checks if the *unique* flag is set.

In a non-sparse index, these documents will be indexed (for non-present indexed attributes, a value of *null* will be used) and will be taken into account for uniqueness checks if the *unique* flag is set.

Note: unique indexes on non-shard keys are not supported in a cluster.

Return Codes

- *200*: If the index already exists, then a *HTTP 200* is returned.
- *201*: If the index does not already exist and could be created, then a *HTTP 201* is returned.
- *400*: If the collection already contains documents and you try to create a unique persistent index in such a way that there are documents violating the uniqueness, then a *HTTP 400* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Creating a persistent index

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/index?
collection=products <<EOF
{
  "type" : "persistent",
  "unique" : false,
  "fields" : [
    "a",
    "b"
  ]
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
```

```
x-content-type-options: nosniff
```

show response body

Creating a sparse persistent index

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/index?
collection=products <<EOF
{
  "type" : "persistent",
  "unique" : false,
  "sparse" : true,
  "fields" : [
    "a"
  ]
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Working with Geo Indexes

Create geo-spatial index

creates a geo index

`POST /_api/index#geo`

Query Parameters

- *collection* (required): The collection name.

A JSON object with these properties is required:

- **fields** (string): An array with one or two attribute paths. If it is an array with one attribute path *location*, then a geo-spatial index on all documents is created using *location* as path to the coordinates. The value of the attribute must be an array with at least two double values. The array must contain the latitude (first value) and the longitude (second value). All documents, which do not have the attribute path or with value that are not suitable, are ignored. If it is an array with two attribute paths *latitude* and *longitude*, then a geo-spatial index on all documents is created using *latitude* and *longitude* as paths the latitude and the longitude. The value of the attribute *latitude* and of the attribute *longitude* must a double. All documents, which do not have the attribute paths or which values are not suitable, are ignored.
- **type**: must be equal to *"geo"*.
- **geoJson**: If a geo-spatial index on a *location* is constructed and *geoJson* is *true*, then the order within the array is longitude followed by latitude. This corresponds to the format described in http://geojson.org/geojson-spec.html#positions

NOTE Swagger examples won't work due to the anchor.

Creates a geo-spatial index in the collection *collection-name*, if it does not already exist. Expects an object containing the index details.

Geo indexes are always sparse, meaning that documents that do not contain the index attributes or have non-numeric values in the index attributes will not be indexed.

Return Codes

- *200*: If the index already exists, then a *HTTP 200* is returned.
- *201*: If the index does not already exist and could be created, then a *HTTP 201* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Creating a geo index with a location attribute

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/index?
collection=products <<EOF
{
  "type" : "geo",
  "fields" : [
    "b"
  ]
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Creating a geo index with latitude and longitude attributes

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/index?
collection=products <<EOF
{
  "type" : "geo",
  "fields" : [
    "e",
    "f"
  ]
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Returns documents near a coordinate

returns all documents of a collection near a given location

`PUT /_api/simple/near`

A JSON object with these properties is required:

- **distance**: If given, the attribute key used to return the distance to the given coordinate. (optional). If specified, distances are returned in meters.
- **skip**: The number of documents to skip in the query. (optional)
- **longitude**: The longitude of the coordinate.
- **limit**: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. The default is 100. (optional)
- **collection**: The name of the collection to query.
- **latitude**: The latitude of the coordinate.
- **geo**: If given, the identifier of the geo-index to use. (optional)

The default will find at most 100 documents near the given coordinate. The returned array is sorted according to the distance, with the nearest document being first in the return array. If there are near documents of equal distance, documents are chosen randomly from this set until the limit is reached.

In order to use the *near* operator, a geo index must be defined for the collection. This index also defines which attribute holds the coordinates for the document. If you have more than one geo-spatial index, you can use the *geo* field to select a particular index.

Returns a cursor containing the result, see Http Cursor for details.

Note: the *near* simple query is **deprecated** as of ArangoDB 2.6. This API may be removed in future versions of ArangoDB. The preferred way for retrieving documents from a collection using the near operator is to issue an AQL query using the *NEAR* function as follows:

```
FOR doc IN NEAR(@@collection, @latitude, @longitude, @limit)
  RETURN doc`
```

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

Without distance

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/near <<EOF
{
  "collection" : "products",
  "latitude" : 0,
  "longitude" : 0,
  "skip" : 1,
  "limit" : 2
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

With distance

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/near <<EOF
{
  "collection" : "products",
  "latitude" : 0,
  "longitude" : 0,
  "skip" : 1,
  "limit" : 3,
  "distance" : "distance"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Find documents within a radius around a coordinate

returns all documents of a collection within a given radius

`PUT /_api/simple/within`

A JSON object with these properties is required:

- **distance**: If given, the attribute key used to return the distance to the given coordinate. (optional). If specified, distances are returned in meters.
- **skip**: The number of documents to skip in the query. (optional)
- **longitude**: The longitude of the coordinate.
- **radius**: The maximal radius (in meters).
- **collection**: The name of the collection to query.
- **latitude**: The latitude of the coordinate.
- **limit**: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. The default is 100. (optional)
- **geo**: If given, the identifier of the geo-index to use. (optional)

This will find all documents within a given radius around the coordinate (*latitude*, *longitude*). The returned list is sorted by distance.

In order to use the *within* operator, a geo index must be defined for the collection. This index also defines which attribute holds the coordinates for the document. If you have more than one geo-spatial index, you can use the *geo* field to select a particular index.

Returns a cursor containing the result, see Http Cursor for details.

Note: the *within* simple query is **deprecated** as of ArangoDB 2.6. This API may be removed in future versions of ArangoDB. The preferred way for retrieving documents from a collection using the near operator is to issue an AQL query using the *WITHIN* function as follows:

```
FOR doc IN WITHIN(@@collection, @latitude, @longitude, @radius, @distanceAttributeName)
  RETURN doc
```

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

Without distance

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/near <<EOF
{
  "collection" : "products",
  "latitude" : 0,
  "longitude" : 0,
  "skip" : 1,
  "limit" : 2,
  "radius" : 500
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
With distance

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/near <<EOF
{
  "collection" : "products",
  "latitude" : 0,
  "longitude" : 0,
  "skip" : 1,
  "limit" : 3,
  "distance" : "distance",
  "radius" : 300
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Fulltext

If a fulltext index exists, then /_api/simple/fulltext will use this index to execute the specified fulltext query.

Create fulltext index

creates a fulltext index

`POST /_api/index#fulltext`

Query Parameters

- *collection-name* (required): The collection name.

A JSON object with these properties is required:

- **fields** (string): an array of attribute names. Currently, the array is limited to exactly one attribute.
- **type**: must be equal to *"fulltext"*.
- **minLength**: Minimum character length of words to index. Will default to a server-defined value if unspecified. It is thus recommended to set this value explicitly when creating the index.

NOTE Swagger examples won't work due to the anchor.

Creates a fulltext index for the collection *collection-name*, if it does not already exist. The call expects an object containing the index details.

Return Codes

- *200*: If the index already exists, then a *HTTP 200* is returned.
- *201*: If the index does not already exist and could be created, then a *HTTP 201* is returned.
- *404*: If the *collection-name* is unknown, then a *HTTP 404* is returned.

Examples

Creating a fulltext index

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/index?
collection=products <<EOF
{
  "type" : "fulltext",
  "fields" : [
    "text"
  ]
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Fulltext index query

returns documents of a collection as a result of a fulltext query

`PUT /_api/simple/fulltext`

A JSON object with these properties is required:

- **index**: The identifier of the fulltext-index to use.
- **attribute**: The attribute that contains the texts.
- **collection**: The name of the collection to query.
- **limit**: The maximal amount of documents to return. The *skip* is applied before the *limit* restriction. (optional)
- **skip**: The number of documents to skip in the query (optional).
- **query**: The fulltext query. Please refer to Fulltext queries for details.

This will find all documents from the collection that match the fulltext query specified in *query*.

In order to use the *fulltext* operator, a fulltext index must be defined for the collection and the specified attribute.

Returns a cursor containing the result, see Http Cursor for details.

Note: the *fulltext* simple query is **deprecated** as of ArangoDB 2.6. This API may be removed in future versions of ArangoDB. The preferred way for retrieving documents from a collection using the near operator is to issue an AQL query using the *FULLTEXT* AQL function as follows:

```
FOR doc IN FULLTEXT(@@collection, @attributeName, @queryString, @limit)
  RETURN doc
```

Return Codes

- *201*: is returned if the query was executed successfully.
- *400*: is returned if the body does not contain a valid JSON representation of a query. The response body contains an error document in this case.
- *404*: is returned if the collection specified by *collection* is unknown. The response body contains an error document in this case.

Examples

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/simple/fulltext
<<EOF
{
  "collection" : "products",
  "attribute" : "text",
  "query" : "word"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

HTTP Interface for Transactions

Transactions

ArangoDB's transactions are executed on the server. Transactions can be initiated by clients by sending the transaction description for execution to the server.

Transactions in ArangoDB do not offer separate *BEGIN*, *COMMIT* and *ROLLBACK* operations as they are available in many other database products. Instead, ArangoDB transactions are described by a JavaScript function, and the code inside the JavaScript function will then be executed transactionally. At the end of the function, the transaction is automatically committed, and all changes done by the transaction will be persisted. If an exception is thrown during transaction execution, all operations performed in the transaction are rolled back.

For a more detailed description of how transactions work in ArangoDB please refer to Transactions.

Execute transaction

execute a server-side transaction

```
POST /_api/transaction
```

A JSON object with these properties is required:

- **maxTransactionSize**: Transaction size limit in bytes. Honored by the RocksDB storage engine only.
- **lockTimeout**: an optional numeric value that can be used to set a timeout for waiting on collection locks. If not specified, a default value will be used. Setting *lockTimeout* to *0* will make ArangoDB not time out waiting for a lock.
- **waitForSync**: an optional boolean flag that, if set, will force the transaction to write all data to disk before returning.
- **params**: optional arguments passed to *action*.
- **action**: the actual transaction operations to be executed, in the form of stringified JavaScript code. The code will be executed on server side, with late binding. It is thus critical that the code specified in *action* properly sets up all the variables it needs. If the code specified in *action* ends with a return statement, the value returned will also be returned by the REST API in the *result* attribute if the transaction committed successfully.
- **collections**: *collections* must be a JSON object that can have either or both sub-attributes *read* and *write*, each being an array of collection names or a single collection name as string. Collections that will be written to in the transaction must be declared with the *write* attribute or it will fail, whereas non-declared collections from which is solely read will be added lazily. The optional sub-attribute *allowImplicit* can be set to *false* to let transactions fail in case of undeclared collections for reading. Collections for reading should be fully declared if possible, to avoid deadlocks.

The transaction description must be passed in the body of the POST request.

If the transaction is fully executed and committed on the server, *HTTP 200* will be returned. Additionally, the return value of the code defined in *action* will be returned in the *result* attribute.

For successfully committed transactions, the returned JSON object has the following properties:

- *error*: boolean flag to indicate if an error occurred (*false* in this case)
- *code*: the HTTP status code
- *result*: the return value of the transaction

If the transaction specification is either missing or malformed, the server will respond with *HTTP 400*.

The body of the response will then contain a JSON object with additional error details. The object has the following attributes:

- *error*: boolean flag to indicate that an error occurred (*true* in this case)
- *code*: the HTTP status code
- *errorNum*: the server error number
- *errorMessage*: a descriptive error message

If a transaction fails to commit, either by an exception thrown in the *action* code, or by an internal error, the server will respond with an error. Any other errors will be returned with any of the return codes *HTTP 400*, *HTTP 409*, or *HTTP 500*.

Return Codes

- *200*: If the transaction is fully executed and committed on the server, *HTTP 200* will be returned.
- *400*: If the transaction specification is either missing or malformed, the server will respond with *HTTP 400*.
- *404*: If the transaction specification contains an unknown collection, the server will respond with *HTTP 404*.
- *500*: Exceptions thrown by users will make the server respond with a return code of *HTTP 500*

Examples

Executing a transaction on a single collection

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/transaction <<EOF
{
  "collections" : {
    "write" : "products"
  },
  "action" : "function () { var db = require('@arangodb').db; db.products.save({});
return db.products.count(); }"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Executing a transaction using multiple collections

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/transaction <<EOF
{
  "collections" : {
    "write" : [
      "products",
      "materials"
    ]
  },
  "action" : "function () {var db =
require('@arangodb').db;db.products.save({});db.materials.save({});return 'worked!';}"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Aborting a transaction due to an internal error

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/transaction <<EOF
{
  "collections" : {
```

```
    "write" : "products"
  },
  "action" : "function () {var db = require('@arangodb').db;db.products.save({ _key:
'abc'});db.products.save({ _key: 'abc'});}"
}
EOF

HTTP/1.1 409 Conflict
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Aborting a transaction by explicitly throwing an exception

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/transaction <<EOF
{
  "collections" : {
    "read" : "products"
  },
  "action" : "function () { throw 'doh!'; }"
}
EOF

HTTP/1.1 500 Internal Server Error
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Referring to a non-existing collection

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/transaction <<EOF
{
  "collections" : {
    "read" : "products"
  },
  "action" : "function () { return true; }"
}
EOF

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

HTTP Interface for Replication

Replication

This is an introduction to ArangoDB's HTTP replication interface. The replication architecture and components are described in more details in Replication.

The HTTP replication interface serves four main purposes:

- fetch initial data from a server (e.g. for a backup, or for the initial synchronization of data before starting the continuous replication applier)
- querying the state of a master
- fetch continuous changes from a master (used for incremental synchronization of changes)
- administer the replication applier (starting, stopping, configuring, querying state) on a slave

Please note that all replication operations work on a per-database level. If an ArangoDB server contains more than one database, the replication system must be configured individually per database, and replicating the data of multiple databases will require multiple operations.

Replication Dump Commands

The *inventory* method can be used to query an ArangoDB database's current set of collections plus their indexes. Clients can use this method to get an overview of which collections are present in the database. They can use this information to either start a full or a partial synchronization of data, e.g. to initiate a backup or the incremental data synchronization.

Return inventory of collections and indexes

Returns an overview of collections and their indexes

```
GET /_api/replication/inventory
```

Query Parameters

- *includeSystem* (optional): Include system collections in the result. The default value is *true*.

Returns the array of collections and indexes available on the server. This array can be used by replication clients to initiate an initial sync with the server.

The response will contain a JSON object with the *collection* and *state* and *tick* attributes.

collections is an array of collections with the following sub-attributes:

- *parameters*: the collection properties
- *indexes*: an array of the indexes of a the collection. Primary indexes and edge indexes are not included in this array.

The *state* attribute contains the current state of the replication logger. It contains the following sub-attributes:

- *running*: whether or not the replication logger is currently active. Note: since ArangoDB 2.2, the value will always be *true*
- *lastLogTick*: the value of the last tick the replication logger has written
- *time*: the current time on the server

Replication clients should note the *lastLogTick* value returned. They can then fetch collections' data using the dump method up to the value of lastLogTick, and query the continuous replication log for log events after this tick value.

To create a full copy of the collections on the server, a replication client can execute these steps:

- call the */inventory* API method. This returns the *lastLogTick* value and the array of collections and indexes from the server.
- for each collection returned by */inventory*, create the collection locally and call */dump* to stream the collection data to the client, up to the value of *lastLogTick*. After that, the client can create the indexes on the collections as they were reported by */inventory*.

If the clients wants to continuously stream replication log events from the logger server, the following additional steps need to be carried out:

- the client should call */logger-follow* initially to fetch the first batch of replication events that were logged after the client's call to */inventory*.

  The call to */logger-follow* should use a *from* parameter with the value of the *lastLogTick* as reported by */inventory*. The call to */logger-follow* will return the *x-arango-replication-lastincluded* which will contain the last tick value included in the response.

- the client can then continuously call */logger-follow* to incrementally fetch new replication events that occurred after the last transfer.

  Calls should use a *from* parameter with the value of the *x-arango-replication-lastincluded* header of the previous response. If there are no more replication events, the response will be empty and clients can go to sleep for a while and try again later.

Note: on a coordinator, this request must have the query parameter *DBserver* which must be an ID of a DBserver. The very same request is forwarded synchronously to that DBserver. It is an error if this attribute is not bound in the coordinator case.

Return Codes

- *200*: is returned if the request was executed successfully.
- *405*: is returned when an invalid HTTP method is used.

- *500*: is returned if an error occurred while assembling the response.

Examples

```
shell> curl --dump - http://localhost:8529/_api/replication/inventory

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body
With some additional indexes:

```
shell> curl --dump - http://localhost:8529/_api/replication/inventory

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff


```

show response body
The *batch* method will create a snapshot of the current state that then can be dumped. A batchId is required when using the dump api with rocksdb.

Create new dump batch

handle a dump batch command

`POST /_api/replication/batch`

Note: These calls are uninteresting to users.

A JSON object with these properties is required:

- **ttl**: the time-to-live for the new batch (in seconds) A JSON object with the batch configuration.

Creates a new dump batch and returns the batch's id.

The response is a JSON object with the following attributes:

- *id*: the id of the batch

Note: on a coordinator, this request must have the query parameter *DBserver* which must be an ID of a DBserver. The very same request is forwarded synchronously to that DBserver. It is an error if this attribute is not bound in the coordinator case.

Return Codes

- *200*: is returned if the batch was created successfully.
- *400*: is returned if the ttl value is invalid or if *DBserver* attribute is not specified or illegal on a coordinator.
- *405*: is returned when an invalid HTTP method is used.

Deletes an existing dump batch

handle a dump batch command

`DELETE /_api/replication/batch/{id}`

Note: These calls are uninteresting to users.

Path Parameters

- *id* (required): The id of the batch.

Deletes the existing dump batch, allowing compaction and cleanup to resume.

Note: on a coordinator, this request must have the query parameter *DBserver* which must be an ID of a DBserver. The very same request is forwarded synchronously to that DBserver. It is an error if this attribute is not bound in the coordinator case.

Return Codes

- *204*: is returned if the batch was deleted successfully.
- *400*: is returned if the batch was not found.
- *405*: is returned when an invalid HTTP method is used.

Prolong existing dump batch

handle a dump batch command

```
PUT /_api/replication/batch/{id}
```

Note: These calls are uninteresting to users.

A JSON object with these properties is required:

- **ttl**: the time-to-live for the new batch (in seconds)

Extends the ttl of an existing dump batch, using the batch's id and the provided ttl value.

If the batch's ttl can be extended successfully, the response is empty.

Note: on a coordinator, this request must have the query parameter *DBserver* which must be an ID of a DBserver. The very same request is forwarded synchronously to that DBserver. It is an error if this attribute is not bound in the coordinator case.

Path Parameters

- *id* (required): The id of the batch.

Return Codes

- *204*: is returned if the batch's ttl was extended successfully.
- *400*: is returned if the ttl value is invalid or the batch was not found.
- *405*: is returned when an invalid HTTP method is used. The *dump* method can be used to fetch data from a specific collection. As the results of the dump command can be huge, *dump* may not return all data from a collection at once. Instead, the dump command may be called repeatedly by replication clients until there is no more data to fetch. The dump command will not only return the current documents in the collection, but also document updates and deletions.

Please note that the *dump* method will only return documents, updates and deletions from a collection's journals and datafiles. Operations that are stored in the write-ahead log only will not be returned. In order to ensure that these operations are included in a dump, the write-ahead log must be flushed first.

To get to an identical state of data, replication clients should apply the individual parts of the dump results in the same order as they are provided.

Return data of a collection

returns the whole content of one collection

```
GET /_api/replication/dump
```

Query Parameters

- *collection* (required): The name or id of the collection to dump.
- *chunkSize* (optional): Approximate maximum size of the returned result.
- *batchId* (required): rocksdb only - The id of the snapshot to use
- *from* (optional): mmfiles only - Lower bound tick value for results.

- *to* (optional): mmfiles only - Upper bound tick value for results.
- *includeSystem* (optional): mmfiles only - Include system collections in the result. The default value is *true*.
- *ticks* (optional): mmfiles only - Whether or not to include tick values in the dump. The default value is *true*.
- *flush* (optional): mmfiles only - Whether or not to flush the WAL before dumping. The default value is *true*.

Returns the data from the collection for the requested range.

When the *from* query parameter is not used, collection events are returned from the beginning. When the *from* parameter is used, the result will only contain collection entries which have higher tick values than the specified *from* value (note: the log entry with a tick value equal to *from* will be excluded).

The *to* query parameter can be used to optionally restrict the upper bound of the result to a certain tick value. If used, the result will only contain collection entries with tick values up to (including) *to*.

The *chunkSize* query parameter can be used to control the size of the result. It must be specified in bytes. The *chunkSize* value will only be honored approximately. Otherwise a too low *chunkSize* value could cause the server to not be able to put just one entry into the result and return it. Therefore, the *chunkSize* value will only be consulted after an entry has been written into the result. If the result size is then bigger than *chunkSize*, the server will respond with as many entries as there are in the response already. If the result size is still smaller than *chunkSize*, the server will try to return more data if there's more data left to return.

If *chunkSize* is not specified, some server-side default value will be used.

The *Content-Type* of the result is *application/x-arango-dump*. This is an easy-to-process format, with all entries going onto separate lines in the response body.

Each line itself is a JSON object, with at least the following attributes:

- *tick*: the operation's tick attribute
- *key*: the key of the document/edge or the key used in the deletion operation
- *rev*: the revision id of the document/edge or the deletion operation
- *data*: the actual document/edge data for types 2300 and 2301. The full document/edge data will be returned even for updates.
- *type*: the type of entry. Possible values for *type* are:
  - 2300: document insertion/update
  - 2301: edge insertion/update
  - 2302: document/edge deletion

Note: there will be no distinction between inserts and updates when calling this method.

Return Codes

- *200*: is returned if the request was executed successfully and data was returned. The header `x-arango-replication-lastincluded` is set to the tick of the last document returned.
- *204*: is returned if the request was executed successfully, but there was no content available. The header `x-arango-replication-lastincluded` is `0` in this case.
- *400*: is returned if either the *from* or *to* values are invalid.
- *404*: is returned when the collection could not be found.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred while assembling the response.

Examples

Empty collection:

```
shell> curl --dump - http://localhost:8529/_api/replication/dump?collection=testCollection
```

```
HTTP/1.1 204 No Content
x-content-type-options: nosniff
content-type: application/x-arango-dump; charset=utf-8
x-arango-replication-checkmore: false
x-arango-replication-lastincluded: 0
```

Non-empty collection:

```
shell> curl --dump - http://localhost:8529/_api/replication/dump?collection=testCollection

HTTP/1.1 200 OK
x-content-type-options: nosniff
content-type: application/x-arango-dump; charset=utf-8
x-arango-replication-checkmore: false
x-arango-replication-lastincluded: 11274

"{\"tick\":\"11268\",\"type\":2300,\"data\":
{\"_id\":\"testCollection/123456\",\"_key\":\"123456\",\"_rev\":\"_XnCMqDy--
D\",\"b\":1,\"c\":false,\"d\":\"additional
value\"}}\n{\"tick\":\"11272\",\"type\":2302,\"data\":
{\"_key\":\"foobar\",\"_rev\":\"_XnCMqD2--
B\"}}\n{\"tick\":\"11274\",\"type\":2302,\"data\":
{\"_key\":\"abcdef\",\"_rev\":\"_XnCMqD2--D\"}}\n"
```

Synchronize data from a remote endpoint

start a replication

`PUT /_api/replication/sync`

A JSON object with these properties is required:

- **username**: an optional ArangoDB username to use when connecting to the endpoint.
- **includeSystem**: whether or not system collection operations will be applied
- **endpoint**: the master endpoint to connect to (e.g. "tcp://192.168.173.13:8529").
- **initialSyncMaxWaitTime**: the maximum wait time (in seconds) that the initial synchronization will wait for a response from the master when fetching initial collection data. This wait time can be used to control after what time the initial synchronization will give up waiting for a response and fail. This value will be ignored if set to *0*.
- **database**: the database name on the master (if not specified, defaults to the name of the local current database).
- **restrictType**: an optional string value for collection filtering. When specified, the allowed values are *include* or *exclude*.
- **incremental**: if set to *true*, then an incremental synchronization method will be used for synchronizing data in collections. This method is useful when collections already exist locally, and only the remaining differences need to be transferred from the remote endpoint. In this case, the incremental synchronization can be faster than a full synchronization. The default value is *false*, meaning that the complete data from the remote collection will be transferred.
- **restrictCollections** (string): an optional array of collections for use with *restrictType*. If *restrictType* is *include*, only the specified collections will be sychronised. If *restrictType* is *exclude*, all but the specified collections will be synchronized.
- **password**: the password to use when connecting to the endpoint.

Starts a full data synchronization from a remote endpoint into the local ArangoDB database.

The *sync* method can be used by replication clients to connect an ArangoDB database to a remote endpoint, fetch the remote list of collections and indexes, and collection data. It will thus create a local backup of the state of data at the remote ArangoDB database. *sync* works on a per-database level.

sync will first fetch the list of collections and indexes from the remote endpoint. It does so by calling the *inventory* API of the remote database. It will then purge data in the local ArangoDB database, and after start will transfer collection data from the remote database to the local ArangoDB database. It will extract data from the remote database by calling the remote database's *dump* API until all data are

fetched.

In case of success, the body of the response is a JSON object with the following attributes:

- *collections*: an array of collections that were transferred from the endpoint
- *lastLogTick*: the last log tick on the endpoint at the time the transfer was started. Use this value as the *from* value when starting the continuous synchronization later.

WARNING: calling this method will sychronize data from the collections found on the remote endpoint to the local ArangoDB database. All data in the local collections will be purged and replaced with data from the endpoint.

Use with caution!

Note: this method is not supported on a coordinator in a cluster.

Return Codes

- *200*: is returned if the request was executed successfully.
- *400*: is returned if the configuration is incomplete or malformed.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred during sychronization.
- *501*: is returned when this operation is called on a coordinator in a cluster.

Return cluster inventory of collections and indexes

returs an overview of collections and indexes in a cluster

```
GET /_api/replication/clusterInventory
```

Query Parameters

- *includeSystem* (optional): Include system collections in the result. The default value is *true*.

Returns the array of collections and indexes available on the cluster.

The response will be an array of JSON objects, one for each collection. Each collection containscontains exactly two keys "parameters" and "indexes". This information comes from Plan/Collections/{DB-Name}/ *in the agency, just that the* indexes* attribute there is relocated to adjust it to the data format of arangodump.

Return Codes

- *200*: is returned if the request was executed successfully.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred while assembling the response.

Replication Logger Commands

Previous versions of ArangoDB allowed starting, stopping and configuring the replication logger. These commands are superfluous in ArangoDB 2.2 as all data-modification operations are written to the server's write-ahead log and are not handled by a separate logger anymore.

The only useful operations remaining since ArangoDB 2.2 are to query the current state of the logger and to fetch the latest changes written by the logger. The operations will return the state and data from the write-ahead log.

Return replication logger state

returns the state of the replication logger

`GET /_api/replication/logger-state`

Returns the current state of the server's replication logger. The state will include information about whether the logger is running and about the last logged tick value. This tick value is important for incremental fetching of data.

The body of the response contains a JSON object with the following attributes:

- *state*: the current logger state as a JSON object with the following sub-attributes:
  - *running*: whether or not the logger is running
  - *lastLogTick*: the tick value of the latest tick the logger has logged. This value can be used for incremental fetching of log data.
  - *totalEvents*: total number of events logged since the server was started. The value is not reset between multiple stops and re-starts of the logger.
  - *time*: the current date and time on the logger server
- *server*: a JSON object with the following sub-attributes:
  - *version*: the logger server's version
  - *serverId*: the logger server's id
- *clients*: returns the last fetch status by replication clients connected to the logger. Each client is returned as a JSON object with the following attributes:
  - *serverId*: server id of client
  - *lastServedTick*: last tick value served to this client via the *logger-follow* API
  - *time*: date and time when this client last called the *logger-follow* API
  - *expires*: date and time when this client would expire without an established connection

Return Codes

- *200*: is returned if the logger state could be determined successfully.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if the logger state could not be determined.

Examples

Returns the state of the replication logger.

```
shell> curl --dump - http://localhost:8529/_api/replication/logger-state

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
To query the latest changes logged by the replication logger, the HTTP interface also provides the `logger-follow` method.

This method should be used by replication clients to incrementally fetch updates from an ArangoDB database.

Returns log entries

Fetch log lines from the server

`GET /_api/replication/logger-follow`

Query Parameters

- *from* (optional): Lower bound tick value for results.
- *to* (optional): Upper bound tick value for results.
- *chunkSize* (optional): Approximate maximum size of the returned result.
- *includeSystem* (optional): Include system collections in the result. The default value is *true*.

Returns data from the server's replication log. This method can be called by replication clients after an initial synchronization of data. The method will return all "recent" log entries from the logger server, and the clients can replay and apply these entries locally so they get to the same data state as the logger server.

Clients can call this method repeatedly to incrementally fetch all changes from the logger server. In this case, they should provide the *from* value so they will only get returned the log events since their last fetch.

When the *from* query parameter is not used, the logger server will return log entries starting at the beginning of its replication log. When the *from* parameter is used, the logger server will only return log entries which have higher tick values than the specified *from* value (note: the log entry with a tick value equal to *from* will be excluded). Use the *from* value when incrementally fetching log data.

The *to* query parameter can be used to optionally restrict the upper bound of the result to a certain tick value. If used, the result will contain only log events with tick values up to (including) *to*. In incremental fetching, there is no need to use the *to* parameter. It only makes sense in special situations, when only parts of the change log are required.

The *chunkSize* query parameter can be used to control the size of the result. It must be specified in bytes. The *chunkSize* value will only be honored approximately. Otherwise a too low *chunkSize* value could cause the server to not be able to put just one log entry into the result and return it. Therefore, the *chunkSize* value will only be consulted after a log entry has been written into the result. If the result size is then bigger than *chunkSize*, the server will respond with as many log entries as there are in the response already. If the result size is still smaller than *chunkSize*, the server will try to return more data if there's more data left to return.

If *chunkSize* is not specified, some server-side default value will be used.

The *Content-Type* of the result is *application/x-arango-dump*. This is an easy-to-process format, with all log events going onto separate lines in the response body. Each log event itself is a JSON object, with at least the following attributes:

- *tick*: the log event tick value
- *type*: the log event type

Individual log events will also have additional attributes, depending on the event type. A few common attributes which are used for multiple events types are:

- *cid*: id of the collection the event was for
- *tid*: id of the transaction the event was contained in
- *key*: document key
- *rev*: document revision id
- *data*: the original document data

A more detailed description of the individual replication event types and their data structures can be found in the manual.

The response will also contain the following HTTP headers:

- *x-arango-replication-active*: whether or not the logger is active. Clients can use this flag as an indication for their polling frequency. If the logger is not active and there are no more replication events available, it might be sensible for a client to abort, or to go to sleep for a long time and try again later to check whether the logger has been activated.
- *x-arango-replication-lastincluded*: the tick value of the last included value in the result. In incremental log fetching, this value can be used as the *from* value for the following request. **Note** that if the result is empty, the value will be *0*. This value should not be used as *from* value by clients in the next request (otherwise the server would return the log events from the start of the log again).
- *x-arango-replication-lasttick*: the last tick value the logger server has logged (not necessarily included in the result). By comparing the the last tick and last included tick values, clients have an approximate indication of how many events there are still left to fetch.
- *x-arango-replication-checkmore*: whether or not there already exists more log data which the client could fetch immediately. If there is more log data available, the client could call *logger-follow* again with an adjusted *from* value to fetch remaining log entries until there are no more.

  If there isn't any more log data to fetch, the client might decide to go to sleep for a while before calling the logger again.

Note: this method is not supported on a coordinator in a cluster.

Return Codes

- *200*: is returned if the request was executed successfully, and there are log events available for the requested range. The response body will not be empty in this case.
- *204*: is returned if the request was executed successfully, but there are no log events available for the requested range. The response body will be empty in this case.
- *400*: is returned if either the *from* or *to* values are invalid.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred while assembling the response.
- *501*: is returned when this operation is called on a coordinator in a cluster.

Examples

No log events available

```
shell> curl --dump - http://localhost:8529/_api/replication/logger-follow?from=11321

HTTP/1.1 204 No Content
x-arango-replication-frompresent: true
x-arango-replication-lastscanned: 11321
x-content-type-options: nosniff
x-arango-replication-lastincluded: 0
content-type: application/x-arango-dump; charset=utf-8
x-arango-replication-checkmore: false
x-arango-replication-lasttick: 11321
x-arango-replication-active: true
```

A few log events

```
shell> curl --dump - http://localhost:8529/_api/replication/logger-follow?from=11321

HTTP/1.1 200 OK
x-arango-replication-frompresent: true
x-arango-replication-lastscanned: 11334
x-content-type-options: nosniff
x-arango-replication-lastincluded: 11334
content-type: application/x-arango-dump; charset=utf-8
x-arango-replication-checkmore: false
```

```
x-arango-replication-lasttick: 11334
x-arango-replication-active: true

"
{\"tick\":\"11323\",\"type\":2000,\"database\":\"1\",\"cid\":\"11322\",\"cname\":\"product
s\",\"data\":
{\"allowUserKeys\":true,\"cid\":\"11322\",\"count\":0,\"deleted\":false,\"doCompact\":true
,\"globallyUniqueId\":\"h5486C74C94E1/11322\",\"id\":\"11322\",\"indexBuckets\":8,\"indexe
s\":[{\"fields\":
[\"_key\"],\"id\":\"0\",\"selectivityEstimate\":1,\"sparse\":false,\"type\":\"primary\",\"
unique\":true}],\"isSmart\":false,\"isSystem\":false,\"isVolatile\":false,\"journalSize\":
33554432,\"keyOptions\":
{\"allowUserKeys\":true,\"lastValue\":0,\"type\":\"traditional\"},\"name\":\"products\",\"
numberOfShards\":1,\"planId\":\"11322\",\"replicationFactor\":1,\"shardKeys\":
[\"_key\"],\"shards\":
{},\"status\":3,\"type\":2,\"version\":6,\"waitForSync\":false}}\n{\"tick\":\"11327\",\"ty
pe\":2300,\"tid\":\"0\",\"database\":\"1\",\"cid\":\"11322\",\"cname\":\"products\",\"data
\":{\"_id\":\"_unknown/p1\",\"_key\":\"p1\",\"_rev\":\"_XnCMspW--_\",\"name\":\"flux
compensator\"}}\n{\"tick\":\"11329\",\"type\":2300,\"tid\":\"0\",\"database\":\"1\",\"cid\
":\"11322\",\"cname\":\"products\",\"data\":
{\"_id\":\"_unknown/p2\",\"_key\":\"p2\",\"_rev\":\"_XnCMspW--
B\",\"hp\":5100,\"name\":\"hybrid
hovercraft\"}}\n{\"tick\":\"11331\",\"type\":2302,\"tid\":\"0\",\"database\":\"1\",\"cid\"
:\"11322\",\"cname\":\"products\",\"data\":{\"_key\":\"p1\",\"_rev\":\"_XnCMspW--
D\"}}\n{\"tick\":\"11333\",\"type\":2300,\"tid\":\"0\",\"database\":\"1\",\"cid\":\"11322\
",\"cname\":\"products\",\"data\":
{\"_id\":\"_unknown/p2\",\"_key\":\"p2\",\"_rev\":\"_XnCMspa--
_\",\"hp\":5100,\"name\":\"broken
hovercraft\"}}\n{\"tick\":\"11334\",\"type\":2001,\"database\":\"1\",\"cid\":\"11322\",\"c
name\":\"products\",\"data\":
{\"cuid\":\"h5486C74C94E1/11322\",\"id\":\"11322\",\"name\":\"products\"}}\n"
```

More events than would fit into the response

```
shell> curl --dump - http://localhost:8529/_api/replication/logger-follow?
from=11308&chunkSize=400

HTTP/1.1 200 OK
x-arango-replication-frompresent: true
x-arango-replication-lastscanned: 11310
x-content-type-options: nosniff
x-arango-replication-lastincluded: 11310
content-type: application/x-arango-dump; charset=utf-8
x-arango-replication-checkmore: true
x-arango-replication-lasttick: 11321
x-arango-replication-active: true

"
{\"tick\":\"11310\",\"type\":2000,\"database\":\"1\",\"cid\":\"11309\",\"cname\":\"product
s\",\"data\":
{\"allowUserKeys\":true,\"cid\":\"11309\",\"count\":0,\"deleted\":false,\"doCompact\":true
,\"globallyUniqueId\":\"h5486C74C94E1/11309\",\"id\":\"11309\",\"indexBuckets\":8,\"indexe
s\":[{\"fields\":
[\"_key\"],\"id\":\"0\",\"selectivityEstimate\":1,\"sparse\":false,\"type\":\"primary\",\"
unique\":true}],\"isSmart\":false,\"isSystem\":false,\"isVolatile\":false,\"journalSize\":
33554432,\"keyOptions\":
```

```
{\"allowUserKeys\":true,\"lastValue\":0,\"type\":\"traditional\"},\"name\":\"products\",\"
numberOfShards\":1,\"planId\":\"11309\",\"replicationFactor\":1,\"shardKeys\":
[\"_key\"],\"shards\":{},\"status\":3,\"type\":2,\"version\":6,\"waitForSync\":false}}\n"
```

To check what range of changes is available (identified by tick values), the HTTP interface provides the methods `logger-first-tick` and `logger-tick-ranges`. Replication clients can use the methods to determine if certain data (identified by a tick *date*) is still available on the master.

Returns the first available tick value

Return the first available tick value from the server

`GET /_api/replication/logger-first-tick`

Returns the first available tick value that can be served from the server's replication log. This method can be called by replication clients after to determine if certain data (identified by a tick value) is still available for replication.

The result is a JSON object containing the attribute *firstTick*. This attribute contains the minimum tick value available in the server's replication log.

Note: this method is not supported on a coordinator in a cluster.

Return Codes

- *200*: is returned if the request was executed successfully.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred while assembling the response.
- *501*: is returned when this operation is called on a coordinator in a cluster.

Examples

Returning the first available tick

```
shell> curl --dump - http://localhost:8529/_api/replication/logger-first-tick

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

"{\"firstTick\":\"5\"}"
```

Return the tick ranges available in the WAL logfiles

returns the tick value ranges available in the logfiles

`GET /_api/replication/logger-tick-ranges`

Returns the currently available ranges of tick values for all currently available WAL logfiles. The tick values can be used to determine if certain data (identified by tick value) are still available for replication.

The body of the response contains a JSON array. Each array member is an object that describes a single logfile. Each object has the following attributes:

- *datafile*: name of the logfile
- *status*: status of the datafile, in textual form (e.g. "sealed", "open")
- *tickMin*: minimum tick value contained in logfile
- *tickMax*: maximum tick value contained in logfile

Return Codes

- *200*: is returned if the tick ranges could be determined successfully.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if the logger state could not be determined.
- *501*: is returned when this operation is called on a coordinator in a cluster.

Examples

Returns the available tick ranges.

```
shell> curl --dump - http://localhost:8529/_api/replication/logger-tick-ranges

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

[
  {
    "datafile" : "/tmp/arangosh_Fqdn5s/tmp-8043-1455050571/data/journals/logfile-3.db",
    "status" : "collected",
    "tickMin" : "5",
    "tickMax" : "10200"
  },
  {
    "datafile" : "/tmp/arangosh_Fqdn5s/tmp-8043-1455050571/data/journals/logfile-38.db",
    "status" : "collected",
    "tickMin" : "10215",
    "tickMax" : "10271"
  },
  {
    "datafile" : "/tmp/arangosh_Fqdn5s/tmp-8043-1455050571/data/journals/logfile-61.db",
    "status" : "collected",
    "tickMin" : "10277",
    "tickMax" : "11274"
  },
  {
    "datafile" : "/tmp/arangosh_Fqdn5s/tmp-8043-1455050571/data/journals/logfile-
10203.db",
    "status" : "collected",
    "tickMin" : "11281",
    "tickMax" : "11283"
  },
  {
    "datafile" : "/tmp/arangosh_Fqdn5s/tmp-8043-1455050571/data/journals/logfile-
10274.db",
    "status" : "open",
    "tickMin" : "11288",
    "tickMax" : "11334"
  }
]
```

Replication Applier Commands

The applier commands allow to remotely start, stop, and query the state and configuration of an ArangoDB database's replication applier.

Return configuration of replication applier

fetch the current replication configuration

```
GET /_api/replication/applier-config
```

Returns the configuration of the replication applier.

The body of the response is a JSON object with the configuration. The following attributes may be present in the configuration:

- *endpoint*: the logger server to connect to (e.g. "tcp://192.168.173.13:8529").
- *database*: the name of the database to connect to (e.g. "_system").
- *username*: an optional ArangoDB username to use when connecting to the endpoint.
- *password*: the password to use when connecting to the endpoint.
- *maxConnectRetries*: the maximum number of connection attempts the applier will make in a row. If the applier cannot establish a connection to the endpoint in this number of attempts, it will stop itself.
- *connectTimeout*: the timeout (in seconds) when attempting to connect to the endpoint. This value is used for each connection attempt.
- *requestTimeout*: the timeout (in seconds) for individual requests to the endpoint.
- *chunkSize*: the requested maximum size for log transfer packets that is used when the endpoint is contacted.
- *autoStart*: whether or not to auto-start the replication applier on (next and following) server starts
- *adaptivePolling*: whether or not the replication applier will use adaptive polling.
- *includeSystem*: whether or not system collection operations will be applied
- *autoResync*: whether or not the slave should perform a full automatic resynchronization with the master in case the master cannot serve log data requested by the slave, or when the replication is started and no tick value can be found.
- *autoResyncRetries*: number of resynchronization retries that will be performed in a row when automatic resynchronization is enabled and kicks in. Setting this to *0* will effectively disable *autoResync*. Setting it to some other value will limit the number of retries that are performed. This helps preventing endless retries in case resynchronizations always fail.
- *initialSyncMaxWaitTime*: the maximum wait time (in seconds) that the initial synchronization will wait for a response from the master when fetching initial collection data. This wait time can be used to control after what time the initial synchronization will give up waiting for a response and fail. This value is relevant even for continuous replication when *autoResync* is set to *true* because this may re-start the initial synchronization when the master cannot provide log data the slave requires. This value will be ignored if set to *0*.
- *connectionRetryWaitTime*: the time (in seconds) that the applier will intentionally idle before it retries connecting to the master in case of connection problems. This value will be ignored if set to *0*.
- *idleMinWaitTime*: the minimum wait time (in seconds) that the applier will intentionally idle before fetching more log data from the master in case the master has already sent all its log data. This wait time can be used to control the frequency with which the replication applier sends HTTP log fetch requests to the master in case there is no write activity on the master. This value will be ignored if set to *0*.
- *idleMaxWaitTime*: the maximum wait time (in seconds) that the applier will intentionally idle before fetching more log data from the master in case the master has already sent all its log data and there have been previous log fetch attempts that resulted in no more log data. This wait time can be used to control the maximum frequency with which the replication applier sends HTTP log fetch

requests to the master in case there is no write activity on the master for longer periods. This configuration value will only be used if the option *adaptivePolling* is set to *true*. This value will be ignored if set to *0*.

- *requireFromPresent*: if set to *true*, then the replication applier will check at start whether the start tick from which it starts or resumes replication is still present on the master. If not, then there would be data loss. If *requireFromPresent* is *true*, the replication applier will abort with an appropriate error message. If set to *false*, then the replication applier will still start, and ignore the data loss.
- *verbose*: if set to *true*, then a log line will be emitted for all operations performed by the replication applier. This should be used for debugging replication problems only.
- *restrictType*: the configuration for *restrictCollections*
- *restrictCollections*: the optional array of collections to include or exclude, based on the setting of *restrictType*

Return Codes

- *200*: is returned if the request was executed successfully.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred while assembling the response.

Examples

```
shell> curl --dump - http://localhost:8529/_api/replication/applier-config

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Adjust configuration of replication applier

set configuration values of an applier

`PUT /_api/replication/applier-config`

A JSON object with these properties is required:

- **username**: an optional ArangoDB username to use when connecting to the endpoint.
- **includeSystem**: whether or not system collection operations will be applied
- **endpoint**: the logger server to connect to (e.g. "tcp://192.168.173.13:8529"). The endpoint must be specified.
- **verbose**: if set to *true*, then a log line will be emitted for all operations performed by the replication applier. This should be used for debugging replication problems only.
- **connectTimeout**: the timeout (in seconds) when attempting to connect to the endpoint. This value is used for each connection attempt.
- **autoResync**: whether or not the slave should perform a full automatic resynchronization with the master in case the master cannot serve log data requested by the slave, or when the replication is started and no tick value can be found.
- **database**: the name of the database on the endpoint. If not specified, defaults to the current local database name.
- **idleMinWaitTime**: the minimum wait time (in seconds) that the applier will intentionally idle before fetching more log data from the master in case the master has already sent all its log data. This wait time can be used to control the frequency with which the replication applier sends HTTP log fetch requests to the master in case there is no write activity on the master. This value will be ignored if set to *0*.
- **requestTimeout**: the timeout (in seconds) for individual requests to the endpoint.
- **requireFromPresent**: if set to *true*, then the replication applier will check at start whether the start tick from which it starts or resumes replication is still present on the master. If not, then there would be data loss. If *requireFromPresent* is *true*, the replication applier will abort with an appropriate error message. If set to *false*, then the replication applier will still start, and ignore the data loss.
- **idleMaxWaitTime**: the maximum wait time (in seconds) that the applier will intentionally idle before fetching more log data from

the master in case the master has already sent all its log data and there have been previous log fetch attempts that resulted in no more log data. This wait time can be used to control the maximum frequency with which the replication applier sends HTTP log fetch requests to the master in case there is no write activity on the master for longer periods. This configuration value will only be used if the option *adaptivePolling* is set to *true*. This value will be ignored if set to *0*.

- **restrictCollections** (string): the array of collections to include or exclude, based on the setting of *restrictType*
- **restrictType**: the configuration for *restrictCollections*; Has to be either *include* or *exclude*
- **initialSyncMaxWaitTime**: the maximum wait time (in seconds) that the initial synchronization will wait for a response from the master when fetching initial collection data. This wait time can be used to control after what time the initial synchronization will give up waiting for a response and fail. This value is relevant even for continuous replication when *autoResync* is set to *true* because this may re-start the initial synchronization when the master cannot provide log data the slave requires. This value will be ignored if set to *0*.
- **maxConnectRetries**: the maximum number of connection attempts the applier will make in a row. If the applier cannot establish a connection to the endpoint in this number of attempts, it will stop itself.
- **autoStart**: whether or not to auto-start the replication applier on (next and following) server starts
- **adaptivePolling**: if set to *true*, the replication applier will fall to sleep for an increasingly long period in case the logger server at the endpoint does not have any more replication events to apply. Using adaptive polling is thus useful to reduce the amount of work for both the applier and the logger server for cases when there are only infrequent changes. The downside is that when using adaptive polling, it might take longer for the replication applier to detect that there are new replication events on the logger server. Setting *adaptivePolling* to false will make the replication applier contact the logger server in a constant interval, regardless of whether the logger server provides updates frequently or seldom.
- **password**: the password to use when connecting to the endpoint.
- **connectionRetryWaitTime**: the time (in seconds) that the applier will intentionally idle before it retries connecting to the master in case of connection problems. This value will be ignored if set to *0*.
- **autoResyncRetries**: number of resynchronization retries that will be performed in a row when automatic resynchronization is enabled and kicks in. Setting this to *0* will effectively disable *autoResync*. Setting it to some other value will limit the number of retries that are performed. This helps preventing endless retries in case resynchronizations always fail.
- **chunkSize**: the requested maximum size for log transfer packets that is used when the endpoint is contacted.

Sets the configuration of the replication applier. The configuration can only be changed while the applier is not running. The updated configuration will be saved immediately but only become active with the next start of the applier.

In case of success, the body of the response is a JSON object with the updated configuration.

Return Codes

- *200*: is returned if the request was executed successfully.
- *400*: is returned if the configuration is incomplete or malformed, or if the replication applier is currently running.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred while assembling the response.

Examples

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/replication/applier-config <<EOF
{
  "endpoint" : "tcp://127.0.0.1:8529",
  "username" : "replicationApplier",
  "password" : "applier1234@foxx",
  "chunkSize" : 4194304,
  "autoStart" : false,
  "adaptivePolling" : true
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Start replication applier

start the replication applier

`PUT /_api/replication/applier-start`

Query Parameters

- *from* (optional): The remote *lastLogTick* value from which to start applying. If not specified, the last saved tick from the previous applier run is used. If there is no previous applier state saved, the applier will start at the beginning of the logger server's log.

Starts the replication applier. This will return immediately if the replication applier is already running.

If the replication applier is not already running, the applier configuration will be checked, and if it is complete, the applier will be started in a background thread. This means that even if the applier will encounter any errors while running, they will not be reported in the response to this method.

To detect replication applier errors after the applier was started, use the */_api/replication/applier-state* API instead.

Return Codes

- *200*: is returned if the request was executed successfully.
- *400*: is returned if the replication applier is not fully configured or the configuration is invalid.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred while assembling the response.

Examples

```
shell> curl -X PUT --dump - http://localhost:8529/_api/replication/applier-start

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Stop replication applier

stop the replication

`PUT /_api/replication/applier-stop`

Stops the replication applier. This will return immediately if the replication applier is not running.

Return Codes

- *200*: is returned if the request was executed successfully.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred while assembling the response.

Examples

```
shell> curl -X PUT --dump - http://localhost:8529/_api/replication/applier-stop

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
```

```
x-content-type-options: nosniff

```

show response body

State of the replication applier

output the current status of the replication

`GET /_api/replication/applier-state`

Returns the state of the replication applier, regardless of whether the applier is currently running or not.

The response is a JSON object with the following attributes:

- *state*: a JSON object with the following sub-attributes:
  - *running*: whether or not the applier is active and running
  - *lastAppliedContinuousTick*: the last tick value from the continuous replication log the applier has applied.
  - *lastProcessedContinuousTick*: the last tick value from the continuous replication log the applier has processed.

    Regularly, the last applied and last processed tick values should be identical. For transactional operations, the replication applier will first process incoming log events before applying them, so the processed tick value might be higher than the applied tick value. This will be the case until the applier encounters the *transaction commit* log event for the transaction.
  - *lastAvailableContinuousTick*: the last tick value the logger server can provide.
  - *time*: the time on the applier server.
  - *totalRequests*: the total number of requests the applier has made to the endpoint.
  - *totalFailedConnects*: the total number of failed connection attempts the applier has made.
  - *totalEvents*: the total number of log events the applier has processed.
  - *totalOperationsExcluded*: the total number of log events excluded because of *restrictCollections*.
  - *progress*: a JSON object with details about the replication applier progress. It contains the following sub-attributes if there is progress to report:
    - *message*: a textual description of the progress
    - *time*: the date and time the progress was logged
    - *failedConnects*: the current number of failed connection attempts
  - *lastError*: a JSON object with details about the last error that happened on the applier. It contains the following sub-attributes if there was an error:
    - *errorNum*: a numerical error code
    - *errorMessage*: a textual error description
    - *time*: the date and time the error occurred

    In case no error has occurred, *lastError* will be empty.
- *server*: a JSON object with the following sub-attributes:
  - *version*: the applier server's version
  - *serverId*: the applier server's id
- *endpoint*: the endpoint the applier is connected to (if applier is active) or will connect to (if applier is currently inactive)
- *database*: the name of the database the applier is connected to (if applier is active) or will connect to (if applier is currently inactive)

Return Codes

- *200*: is returned if the request was executed successfully.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred while assembling the response.

Examples

Fetching the state of an inactive applier:

```
shell> curl --dump - http://localhost:8529/_api/replication/applier-state

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Fetching the state of an active applier:

```
shell> curl --dump - http://localhost:8529/_api/replication/applier-state

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Turn the server into a slave of another

Changes role to slave

`PUT /_api/replication/make-slave`

A JSON object with these properties is required:

- **username**: an optional ArangoDB username to use when connecting to the master.
- **includeSystem**: whether or not system collection operations will be applied
- **endpoint**: the master endpoint to connect to (e.g. "tcp://192.168.173.13:8529").
- **verbose**: if set to *true*, then a log line will be emitted for all operations performed by the replication applier. This should be used for debugging replication problems only.
- **connectTimeout**: the timeout (in seconds) when attempting to connect to the endpoint. This value is used for each connection attempt.
- **autoResync**: whether or not the slave should perform an automatic resynchronization with the master in case the master cannot serve log data requested by the slave, or when the replication is started and no tick value can be found.
- **database**: the database name on the master (if not specified, defaults to the name of the local current database).
- **idleMinWaitTime**: the minimum wait time (in seconds) that the applier will intentionally idle before fetching more log data from the master in case the master has already sent all its log data. This wait time can be used to control the frequency with which the replication applier sends HTTP log fetch requests to the master in case there is no write activity on the master. This value will be ignored if set to *0*.
- **requestTimeout**: the timeout (in seconds) for individual requests to the endpoint.
- **restrictType**: an optional string value for collection filtering. When specified, the allowed values are *include* or *exclude*.
- **idleMaxWaitTime**: the maximum wait time (in seconds) that the applier will intentionally idle before fetching more log data from the master in case the master has already sent all its log data and there have been previous log fetch attempts that resulted in no more log data. This wait time can be used to control the maximum frequency with which the replication applier sends HTTP log fetch requests to the master in case there is no write activity on the master for longer periods. This configuration value will only be used if the option *adaptivePolling* is set to *true*. This value will be ignored if set to *0*.
- **initialSyncMaxWaitTime**: the maximum wait time (in seconds) that the initial synchronization will wait for a response from the

master when fetching initial collection data. This wait time can be used to control after what time the initial synchronization will give up waiting for a response and fail. This value is relevant even for continuous replication when *autoResync* is set to *true* because this may re-start the initial synchronization when the master cannot provide log data the slave requires. This value will be ignored if set to *0*.

- **restrictCollections** (string): an optional array of collections for use with *restrictType*. If *restrictType* is *include*, only the specified collections will be sychronised. If *restrictType* is *exclude*, all but the specified collections will be synchronized.
- **requireFromPresent**: if set to *true*, then the replication applier will check at start of its continuous replication if the start tick from the dump phase is still present on the master. If not, then there would be data loss. If *requireFromPresent* is *true*, the replication applier will abort with an appropriate error message. If set to *false*, then the replication applier will still start, and ignore the data loss.
- **adaptivePolling**: whether or not the replication applier will use adaptive polling.
- **maxConnectRetries**: the maximum number of connection attempts the applier will make in a row. If the applier cannot establish a connection to the endpoint in this number of attempts, it will stop itself.
- **password**: the password to use when connecting to the master.
- **connectionRetryWaitTime**: the time (in seconds) that the applier will intentionally idle before it retries connecting to the master in case of connection problems. This value will be ignored if set to *0*.
- **autoResyncRetries**: number of resynchronization retries that will be performed in a row when automatic resynchronization is enabled and kicks in. Setting this to *0* will effectively disable *autoResync*. Setting it to some other value will limit the number of retries that are performed. This helps preventing endless retries in case resynchronizations always fail.
- **chunkSize**: the requested maximum size for log transfer packets that is used when the endpoint is contacted.

Starts a full data synchronization from a remote endpoint into the local ArangoDB database and afterwards starts the continuous replication. The operation works on a per-database level.

All local database data will be removed prior to the synchronization.

In case of success, the body of the response is a JSON object with the following attributes:

- *state*: a JSON object with the following sub-attributes:
  - *running*: whether or not the applier is active and running
  - *lastAppliedContinuousTick*: the last tick value from the continuous replication log the applier has applied.
  - *lastProcessedContinuousTick*: the last tick value from the continuous replication log the applier has processed.

    Regularly, the last applied and last processed tick values should be identical. For transactional operations, the replication applier will first process incoming log events before applying them, so the processed tick value might be higher than the applied tick value. This will be the case until the applier encounters the *transaction commit* log event for the transaction.
  - *lastAvailableContinuousTick*: the last tick value the logger server can provide.
  - *time*: the time on the applier server.
  - *totalRequests*: the total number of requests the applier has made to the endpoint.
  - *totalFailedConnects*: the total number of failed connection attempts the applier has made.
  - *totalEvents*: the total number of log events the applier has processed.
  - *totalOperationsExcluded*: the total number of log events excluded because of *restrictCollections*.
  - *progress*: a JSON object with details about the replication applier progress. It contains the following sub-attributes if there is progress to report:
    - *message*: a textual description of the progress
    - *time*: the date and time the progress was logged
    - *failedConnects*: the current number of failed connection attempts
  - *lastError*: a JSON object with details about the last error that happened on the applier. It contains the following sub-attributes if there was an error:
    - *errorNum*: a numerical error code

    - *errorMessage*: a textual error description
    - *time*: the date and time the error occurred

  In case no error has occurred, *lastError* will be empty.

- *server*: a JSON object with the following sub-attributes:
  - *version*: the applier server's version
  - *serverId*: the applier server's id
- *endpoint*: the endpoint the applier is connected to (if applier is active) or will connect to (if applier is currently inactive)
- *database*: the name of the database the applier is connected to (if applier is active) or will connect to (if applier is currently inactive)

WARNING: calling this method will sychronize data from the collections found on the remote master to the local ArangoDB database. All data in the local collections will be purged and replaced with data from the master.

Use with caution!

Please also keep in mind that this command may take a long time to complete and return. This is because it will first do a full data synchronization with the master, which will take time roughly proportional to the amount of data.

Note: this method is not supported on a coordinator in a cluster.

Return Codes

- *200*: is returned if the request was executed successfully.
- *400*: is returned if the configuration is incomplete or malformed.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred during sychronization or when starting the continuous replication.
- *501*: is returned when this operation is called on a coordinator in a cluster.

Other Replication Commands

Return server id

fetch this server's unique identifier

`GET /_api/replication/server-id`

Returns the servers id. The id is also returned by other replication API methods, and this method is an easy means of determining a server's id.

The body of the response is a JSON object with the attribute *serverId*. The server id is returned as a string.

Return Codes

- *200*: is returned if the request was executed successfully.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if an error occurred while assembling the response.

Examples

```
shell> curl --dump - http://localhost:8529/_api/replication/server-id

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

{
  "serverId" : "92937846035681"
}
```

HTTP Interface for Sharding

Sharding only should be used by developers!

Execute cluster roundtrip

executes a cluster roundtrip for sharding

```
GET /_admin/cluster-test
```

Executes a cluster roundtrip from a coordinator to a DB server and back. This call only works in a coordinator node in a cluster. One can and should append an arbitrary path to the URL and the part after */_admin/cluster-test* is used as the path of the HTTP request which is sent from the coordinator to a DB node. Likewise, any form data appended to the URL is forwarded in the request to the DB node. This handler takes care of all request types (see below) and uses the same request type in its request to the DB node.

The following HTTP headers are interpreted in a special way:

- *X-Shard-ID*: This specifies the ID of the shard to which the cluster request is sent and thus tells the system to which DB server to send the cluster request. Note that the mapping from the shard ID to the responsible server has to be defined in the agency under *Current/ShardLocation/*. One has to give this header, otherwise the system does not know where to send the request.
- *X-Client-Transaction-ID*: the value of this header is taken as the client transaction ID for the request
- *X-Timeout*: specifies a timeout in seconds for the cluster operation. If the answer does not arrive within the specified timeout, an corresponding error is returned and any subsequent real answer is ignored. The default if not given is 24 hours.
- *X-Synchronous-Mode*: If set to *true* the test function uses synchronous mode, otherwise the default asynchronous operation mode is used. This is mainly for debugging purposes.
- *Host*: This header is ignored and not forwarded to the DB server.
- *User-Agent*: This header is ignored and not forwarded to the DB server.

All other HTTP headers and the body of the request (if present, see other HTTP methods below) are forwarded as given in the original request.

In asynchronous mode the DB server answers with an HTTP request of its own, in synchronous mode it sends a HTTP response. In both cases the headers and the body are used to produce the HTTP response of this API call.

Return Codes

The return code can be anything the cluster request returns, as well as:

- *200*: is returned when everything went well, or if a timeout occurred. In the latter case a body of type application/json indicating the timeout is returned.
- *403*: is returned if ArangoDB is not running in cluster mode.
- *404*: is returned if ArangoDB was not compiled for cluster operation.

Execute cluster roundtrip

executes a cluster roundtrip for sharding

```
POST /_admin/cluster-test
```

Request Body (required)

The body can be any type and is simply forwarded.

See GET method.

Return Codes

- *200*: is returned when everything went well.

Execute cluster roundtrip

executes a cluster roundtrip for sharding

```
PUT /_admin/cluster-test
```

Request Body (required)

See GET method. The body can be any type and is simply forwarded.

Return Codes

- *200*: is returned when everything went well.

Delete cluster roundtrip

executes a cluster roundtrip for sharding

```
DELETE /_admin/cluster-test
```

See GET method.

Return Codes

- *200*: is returned when everything went well.

Update cluster roundtrip

executes a cluster roundtrip for sharding

```
PATCH /_admin/cluster-test
```

Request Body (required)

See GET method. The body can be any type and is simply forwarded.

Return Codes

- *200*: is returned when everything went well.

Execute cluster roundtrip

executes a cluster roundtrip for sharding

```
HEAD /_admin/cluster-test
```

See GET method.

Return Codes

- *200*: is returned when everything went well.

Check port

allows to check whether a given port is usable

```
GET /_admin/clusterCheckPort
```

Query Parameters

- *port* (required):

Checks whether the requested port is usable.

Return Codes

- *200*: is returned when everything went well.
- *400*: the parameter port was not given or is no integer.

HTTP Interface for Administration and Monitoring

This is an introduction to ArangoDB's HTTP interface for administration and monitoring of the server.

Read global logs from the server

returns the server logs

```
GET /_admin/log
```

Query Parameters

- *upto* (optional): Returns all log entries up to log level *upto*. Note that *upto* must be:
- *fatal* or *0*
- *error* or *1*
- *warning* or *2*
- *info* or *3*
- *debug* or *4* The default value is *info*.
- *level* (optional): Returns all log entries of log level *level*. Note that the query parameters *upto* and *level* are mutually exclusive.
- *start* (optional): Returns all log entries such that their log entry identifier (*lid* value) is greater or equal to *start*.
- *size* (optional): Restricts the result to at most *size* log entries.
- *offset* (optional): Starts to return log entries skipping the first *offset* log entries. *offset* and *size* can be used for pagination.
- *search* (optional): Only return the log entries containing the text specified in *search*.
- *sort* (optional): Sort the log entries either ascending (if *sort* is *asc*) or descending (if *sort* is *desc*) according to their *lid* values. Note that the *lid* imposes a chronological order. The default value is *asc*.

Returns fatal, error, warning or info log messages from the server's global log. The result is a JSON object with the following attributes:

HTTP 200

A json document with these Properties is returned:

- **lid** (string): a list of log entry identifiers. Each log message is uniquely identified by its @LIT{lid} and the identifiers are in ascending order.
- **level**: A list of the loglevels for all log entries.
- **timestamp** (string): a list of the timestamps as seconds since 1970-01-01 for all log entries.
- **topic**: a list of the topics of all log entries
- **text**: a list of the texts of all log entries
- **totalAmount**: the total amount of log entries before pagination.

Return Codes

- *200*:

Response Body

- **lid** (string): a list of log entry identifiers. Each log message is uniquely identified by its @LIT{lid} and the identifiers are in ascending order.
- **level**: A list of the loglevels for all log entries.
- **text**: a list of the texts of all log entries
- **topic**: a list of the topics of all log entries
- **timestamp** (string): a list of the timestamps as seconds since 1970-01-01 for all log entries.
- **totalAmount**: the total amount of log entries before pagination.
- *400*: is returned if invalid values are specified for *upto* or *level*.
- *500*: is returned if the server cannot generate the result due to an out-of-memory error.

Return the current server loglevel

returns the current loglevel settings

```
GET /_admin/log/level
```

Returns the server's current loglevel settings. The result is a JSON object with the log topics being the object keys, and the log levels being the object values.

Return Codes

- *200*: is returned if the request is valid
- *500*: is returned if the server cannot generate the result due to an out-of-memory error.

Modify and return the current server loglevel

modifies the current loglevel settings

```
PUT /_admin/log/level
```

Modifies and returns the server's current loglevel settings. The request body must be a JSON object with the log topics being the object keys and the log levels being the object values.

The result is a JSON object with the adjusted log topics being the object keys, and the adjusted log levels being the object values.

It can set the loglevel of all facilities by only specifying the loglevel as string without json.

Possible loglevels are:

- FATAL - There will be no way out of this. ArangoDB will go down after this message.
- ERROR - This is an error. you should investigate and fix it. It may harm your production.
- WARNING - This may be serious application-wise, but we don't know.
- INFO - Something has happened, take notice, but no drama attached.
- DEBUG - output debug messages
- TRACE - trace - prepare your log to be flooded - don't use in production.

A JSON object with these properties is required:

- **audit-service**: One of the possible loglevels.
- **cache**: One of the possible loglevels.
- **syscall**: One of the possible loglevels.
- **communication**: One of the possible loglevels.
- **audit-authentication**: One of the possible loglevels.
- **agencycomm**: One of the possible loglevels.
- **startup**: One of the possible loglevels.
- **general**: One of the possible loglevels.
- **cluster**: One of the possible loglevels.
- **audit-view**: One of the possible loglevels.
- **collector**: One of the possible loglevels.
- **audit-documentation**: One of the possible loglevels.
- **engines**: One of the possible loglevels.
- **trx**: One of the possible loglevels.
- **mmap**: One of the possible loglevels.
- **agency**: One of the possible loglevels.
- **authentication**: One of the possible loglevels.
- **memory**: One of the possible loglevels.
- **performance**: One of the possible loglevels.
- **config**: One of the possible loglevels.
- **authorization**: One of the possible loglevels.
- **development**: One of the possible loglevels.
- **datafiles**: One of the possible loglevels.
- **views**: One of the possible loglevels.

- **ldap**: One of the possible loglevels.
- **replication**: One of the possible loglevels.
- **threads**: One of the possible loglevels.
- **audit-database**: One of the possible loglevels.
- **v8**: One of the possible loglevels.
- **ssl**: One of the possible loglevels.
- **pregel**: One of the possible loglevels.
- **audit-collection**: One of the possible loglevels.
- **rocksdb**: One of the possible loglevels.
- **supervision**: One of the possible loglevels.
- **graphs**: One of the possible loglevels.
- **compactor**: One of the possible loglevels.
- **queries**: One of the possible loglevels.
- **heartbeat**: One of the possible loglevels.
- **requests**: One of the possible loglevels.

Return Codes

- *200*: is returned if the request is valid
- *400*: is returned when the request body contains invalid JSON.
- *405*: is returned when an invalid HTTP method is used.
- *500*: is returned if the server cannot generate the result due to an out-of-memory error.

Reloads the routing information

Reload the routing table.

`POST /_admin/routing/reload`

Reloads the routing information from the collection *routing*.

Return Codes

- *200*: Routing information was reloaded successfully.

Read the statistics

return the statistics information

`GET /_admin/statistics`

Returns the statistics information. The returned object contains the statistics figures grouped together according to the description returned by *_admin/statistics-description*. For instance, to access a figure *userTime* from the group *system*, you first select the sub-object describing the group stored in *system* and in that sub-object the value for *userTime* is stored in the attribute of the same name.

In case of a distribution, the returned object contains the total count in *count* and the distribution list in *counts*. The sum (or total) of the individual values is returned in *sum*.

Return Codes

- *200*: Statistics were returned successfully.

Examples

```
shell> curl --dump - http://localhost:8529/_admin/statistics

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Statistics description

fetch descriptive info of statistics

`GET /_admin/statistics-description`

Returns a description of the statistics returned by */_admin/statistics*. The returned objects contains an array of statistics groups in the attribute *groups* and an array of statistics figures in the attribute *figures*.

A statistics group is described by

- *group*: The identifier of the group.
- *name*: The name of the group.
- *description*: A description of the group.

A statistics figure is described by

- *group*: The identifier of the group to which this figure belongs.
- *identifier*: The identifier of the figure. It is unique within the group.
- *name*: The name of the figure.
- *description*: A description of the figure.
- *type*: Either *current*, *accumulated*, or *distribution*.
- *cuts*: The distribution vector.
- *units*: Units in which the figure is measured.

Return Codes

- *200*: Description was returned successfully.

Examples

```
shell> curl --dump - http://localhost:8529/_admin/statistics-description

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Return role of a server in a cluster

Get to know whether this server is a Coordinator or DB-Server

`GET /_admin/server/role`

Returns the role of a server in a cluster. The role is returned in the *role* attribute of the result. Possible return values for *role* are:

- *SINGLE*: the server is a standalone server without clustering
- *COORDINATOR*: the server is a coordinator in a cluster
- *PRIMARY*: the server is a primary database server in a cluster
- *SECONDARY*: the server is a secondary database server in a cluster
- *AGENT*: the server is an agency node in a cluster
- *UNDEFINED*: in a cluster, *UNDEFINED* is returned if the server role cannot be determined.

Return Codes

- *200*: Is returned in all cases.

Return id of a server in a cluster

Get to know the internal id of the server

`GET /_admin/server/id`

Returns the id of a server in a cluster. The request will fail if the server is not running in cluster mode.

Return Codes

- *200*: Is returned when the server is running in cluster mode.
- *500*: Is returned when the server is not running in cluster mode.

Return whether or not a server is available

Return whether or not a server is available

`GET /_admin/server/availability`

Return availability information about a server.

This is a public API so it does *not* require authentication. It is meant to be used only in the context of server monitoring only.

Return Codes

- *200*: This API will return HTTP 200 in case the server is up and running and usable for arbitrary operations, is not set to read-only mode and is currently not a follower in case of an active failover setup.
- *503*: HTTP 503 will be returned in case the server is during startup or during shutdown, is set to read-only mode or is currently a follower in an active failover setup.

Cluster

Queries statistics of DBserver

allows to query the statistics of a DBserver in the cluster

`GET /_admin/clusterStatistics`

Query Parameters

- *DBserver* (required):

Queries the statistics of the given DBserver

Return Codes

- *200*: is returned when everything went well.
- *400*: the parameter DBserver was not given or is not the ID of a DBserver
- *403*: server is not a coordinator.

Queries the health of cluster for monitoring

Returns the health of the cluster as assessed by the supervision (agency)

`GET /_admin/cluster/health`

Queries the health of the cluster for monitoring purposes. The response is a JSON object, containing the standard `code`, `error`, `errorNum`, and `errorMessage` fields as appropriate. The endpoint-specific fields are as follows:

- `ClusterId`: A UUID string identifying the cluster
- `Health`: An object containing a descriptive sub-object for each node in the cluster. Each entry in `Health` will be keyed by the node ID and contain the the following attributes:
  - `Endpoint`: A string representing the network endpoint of the server.
  - `Role`: The role the server plays. Possible values are `"AGENT"`, `"COORDINATOR"`, and `"DBSERVER"`.
  - `CanBeDeleted`: Boolean representing whether the node can safely be removed from the cluster.

  Additionally, if the node is a Coordinator or DBServer, it will also have the following attributes:

- `Status` : A string indicating the health of the node as assessed by the supervision (agency). This should be considered primary source of truth for node health. If the node is responding normally to requests, it is `"GOOD"` . If it has missed one heartbeat, it is `"BAD"` . If it has been declared failed by the supervision, which occurs after missing heartbeats for about 15 seconds, it will be marked `"FAILED"` .
- `SyncStatus` : The last sync status reported by the node. This value is primarily used to determine the value of `Status` . Possible values include `"UNKNOWN"` , `"UNDEFINED"` , `"STARTUP"` , `"STOPPING"` , `"STOPPED"` , `"SERVING"` , `"SHUTDOWN"` .
- `ShortName` : A string representing the shortname of the server, e.g. `"DBServer1"` .
- `Timestamp` : ISO 8601 timestamp specifying the last heartbeat received.
- `Host` : An optional string, specifying the host machine if known.

Return Codes

- *200*: is returned when everything went well.

HTTP Interface for Endpoints

The API `/_api/endpoint` is *deprecated*. For cluster mode there is `/_api/cluster/endpoints` to find all current coordinator endpoints (see below).

The ArangoDB server can listen for incoming requests on multiple *endpoints*.

The endpoints are normally specified either in ArangoDB's configuration file or on the command-line, using the "--server.endpoint" option. The default endpoint for ArangoDB is *tcp://127.0.0.1:8529* or *tcp://localhost:8529*.

Please note that all endpoint management operations can only be accessed via the default database (*_system*) and none of the other databases.

Asking about Endpoints via HTTP

Get information about all coordinator endpoints

This API call returns information about all coordinator endpoints (cluster only).

`GET /_api/cluster/endpoints`

Returns an object with an attribute `endpoints`, which contains an array of objects, which each have the attribute `endpoint`, whose value is a string with the endpoint description. There is an entry for each coordinator in the cluster. This method only works on coordinators in cluster mode. In case of an error the `error` attribute is set to `true`.

Return Codes

- *200*: is returned when everything went well.
- *403*: server is not a coordinator or method was not GET.

Return list of all endpoints

This API call returns the list of all endpoints (single server).

`GET /_api/endpoint`

THIS API IS DEPRECATED

Returns an array of all configured endpoints the server is listening on.

The result is a JSON array of JSON objects, each with `"entrypoint"` as the only attribute, and with the value being a string describing the endpoint.

Note: retrieving the array of all endpoints is allowed in the system database only. Calling this action in any other database will make the server return an error.

Return Codes

- *200*: is returned when the array of endpoints can be determined successfully.
- *400*: is returned if the action is not carried out in the system database.
- *405*: The server will respond with *HTTP 405* if an unsupported HTTP method is used.

Examples

```
shell> curl --dump - http://localhost:8529/_api/endpoint

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

```
[
  {
    "endpoint" : "http://127.0.0.1:18328"
  }
]
```

Foxx HTTP API

These routes allow manipulating the Foxx services installed in a database.

For more information on Foxx and its JavaScript APIs see the Foxx chapter of the main documentation.

Foxx Service Management

This is an introduction to ArangoDB's HTTP interface for managing Foxx services.

List installed services

list installed services

`GET /_api/foxx`

Fetches a list of services installed in the current database.

Returns a list of objects with the following attributes:

- *mount*: the mount path of the service
- *development*: *true* if the service is running in development mode
- *legacy*: *true* if the service is running in 2.8 legacy compatibility mode
- *provides*: the service manifest's *provides* value or an empty object

Additionally the object may contain the following attributes if they have been set on the manifest:

- *name*: a string identifying the service type
- *version*: a semver-compatible version string

Query Parameters

- *excludeSystem* (optional): Whether or not system services should be excluded from the result.

Return Codes

- *200*: Returned if the request was successful.

Service description

service metadata

`GET /_api/foxx/service`

Fetches detailed information for the service at the given mount path.

Returns an object with the following attributes:

- *mount*: the mount path of the service
- *path*: the local file system path of the service
- *development*: *true* if the service is running in development mode
- *legacy*: *true* if the service is running in 2.8 legacy compatibility mode
- *manifest*: the normalized JSON manifest of the service

Additionally the object may contain the following attributes if they have been set on the manifest:

- *name*: a string identifying the service type
- *version*: a semver-compatible version string

Query Parameters

- *mount* (required): Mount path of the installed service.

Return Codes

- *200*: Returned if the request was successful.
- *400*: Returned if the mount path is unknown.

Install new service

install new service

`POST /_api/foxx`

Installs the given new service at the given mount path.

The request body can be any of the following formats:

- `application/zip` : a raw zip bundle containing a service
- `application/javascript` : a standalone JavaScript file
- `application/json` : a service definition as JSON
- `multipart/form-data` : a service definition as a multipart form

A service definition is an object or form with the following properties or fields:

- *configuration*: a JSON object describing configuration values
- *dependencies*: a JSON object describing dependency settings
- *source*: a fully qualified URL or an absolute path on the server's file system

When using multipart data, the *source* field can also alternatively be a file field containing either a zip bundle or a standalone JavaScript file.

When using a standalone JavaScript file the given file will be executed to define our service's HTTP endpoints. It is the same which would be defined in the field `main` of the service manifest.

If *source* is a URL, the URL must be reachable from the server. If *source* is a file system path, the path will be resolved on the server. In either case the path or URL is expected to resolve to a zip bundle, JavaScript file or (in case of a file system path) directory.

Note that when using file system paths in a cluster with multiple coordinators the file system path must resolve to equivalent files on every coordinator.

Query Parameters

- *mount* (required): Mount path the service should be installed at.
- *development* (optional): Set to `true` to enable development mode.
- *setup* (optional): Set to `false` to not run the service's setup script.
- *legacy* (optional): Set to `true` to install the service in 2.8 legacy compatibility mode.

Return Codes

- *201*: Returned if the request was successful.

Uninstall service

uninstall service

`DELETE /_api/foxx/service`

Removes the service at the given mount path from the database and file system.

Returns an empty response on success.

Query Parameters

- *mount* (required): Mount path of the installed service.
- *teardown* (optional): Set to `false` to not run the service's teardown script.

Return Codes

- *204*: Returned if the request was successful.

Replace service

replace a service

`PUT /_api/foxx/service`

Removes the service at the given mount path from the database and file system. Then installs the given new service at the same mount path.

This is a slightly safer equivalent to performing an uninstall of the old service followed by installing the new service. The new service's main and script files (if any) will be checked for basic syntax errors before the old service is removed.

The request body can be any of the following formats:

- `application/zip` : a raw zip bundle containing a service
- `application/javascript` : a standalone JavaScript file
- `application/json` : a service definition as JSON
- `multipart/form-data` : a service definition as a multipart form

A service definition is an object or form with the following properties or fields:

- *configuration*: a JSON object describing configuration values
- *dependencies*: a JSON object describing dependency settings
- *source*: a fully qualified URL or an absolute path on the server's file system

When using multipart data, the *source* field can also alternatively be a file field containing either a zip bundle or a standalone JavaScript file.

When using a standalone JavaScript file the given file will be executed to define our service's HTTP endpoints. It is the same which would be defined in the field `main` of the service manifest.

If *source* is a URL, the URL must be reachable from the server. If *source* is a file system path, the path will be resolved on the server. In either case the path or URL is expected to resolve to a zip bundle, JavaScript file or (in case of a file system path) directory.

Note that when using file system paths in a cluster with multiple coordinators the file system path must resolve to equivalent files on every coordinator.

Query Parameters

- *mount* (required): Mount path of the installed service.
- *teardown* (optional): Set to `false` to not run the old service's teardown script.
- *setup* (optional): Set to `false` to not run the new service's setup script.
- *legacy* (optional): Set to `true` to install the new service in 2.8 legacy compatibility mode.
- *force* (optional): Set to `true` to force service install even if no service is installed under given mount.

Return Codes

- *200*: Returned if the request was successful.

Upgrade service

upgrade a service

`PATCH /_api/foxx/service`

Installs the given new service on top of the service currently installed at the given mount path. This is only recommended for switching between different versions of the same service.

Unlike replacing a service, upgrading a service retains the old service's configuration and dependencies (if any) and should therefore only be used to migrate an existing service to a newer or equivalent service.

The request body can be any of the following formats:

- `application/zip` : a raw zip bundle containing a service
- `application/javascript` : a standalone JavaScript file
- `application/json` : a service definition as JSON
- `multipart/form-data` : a service definition as a multipart form

A service definition is an object or form with the following properties or fields:

- *configuration*: a JSON object describing configuration values
- *dependencies*: a JSON object describing dependency settings
- *source*: a fully qualified URL or an absolute path on the server's file system

When using multipart data, the *source* field can also alternatively be a file field containing either a zip bundle or a standalone JavaScript file.

When using a standalone JavaScript file the given file will be executed to define our service's HTTP endpoints. It is the same which would be defined in the field `main` of the service manifest.

If *source* is a URL, the URL must be reachable from the server. If *source* is a file system path, the path will be resolved on the server. In either case the path or URL is expected to resolve to a zip bundle, JavaScript file or (in case of a file system path) directory.

Note that when using file system paths in a cluster with multiple coordinators the file system path must resolve to equivalent files on every coordinator.

Query Parameters

- *mount* (required): Mount path of the installed service.
- *teardown* (optional): Set to `true` to run the old service's teardown script.
- *setup* (optional): Set to `false` to not run the new service's setup script.
- *legacy* (optional): Set to `true` to install the new service in 2.8 legacy compatibility mode.

Return Codes

- *200*: Returned if the request was successful.

Foxx Service configuration / dependencies

This is an introduction to ArangoDB's HTTP interface for managing Foxx services configuration and dependencies.

Get configuration options

get configuration options

`GET /_api/foxx/configuration`

Fetches the current configuration for the service at the given mount path.

Returns an object mapping the configuration option names to their definitions including a human-friendly *title* and the *current* value (if any).

Query Parameters

- *mount* (required): Mount path of the installed service.

Return Codes

- *200*: Returned if the request was sucessful.

Update configuration options

update configuration options

`PATCH /_api/foxx/configuration`

Replaces the given service's configuration.

Returns an object mapping all configuration option names to their new values.

Request Body (required)

A JSON object mapping configuration option names to their new values. Any omitted options will be ignored.

Query Parameters

- *mount* (required): Mount path of the installed service.
- *200*: Returned if the request was sucessful.

Replace configuration options

replace configuration options

`PUT /_api/foxx/configuration`

Replaces the given service's configuration completely.

Returns an object mapping all configuration option names to their new values.

Request Body (required)

A JSON object mapping configuration option names to their new values. Any omitted options will be reset to their default values or marked as unconfigured.

Query Parameters

- *mount* (required): Mount path of the installed service.
- *200*: Returned if the request was sucessful.

Get dependency options

get dependency options

`GET /_api/foxx/dependencies`

Fetches the current dependencies for service at the given mount path.

Returns an object mapping the dependency names to their definitions including a human-friendly *title* and the *current* mount path (if any).

Query Parameters

- *mount* (required): Mount path of the installed service.

Return Codes

- *200*: Returned if the request was sucessful.

Update dependencies options

update dependencies options

`PATCH /_api/foxx/dependencies`

Replaces the given service's dependencies.

Returns an object mapping all dependency names to their new mount paths.

Request Body (required)

A JSON object mapping dependency names to their new mount paths. Any omitted dependencies will be ignored.

Query Parameters

- *mount* (required): Mount path of the installed service.
- *200*: Returned if the request was sucessful.

Replace dependencies options

replace dependencies options

`PUT /_api/foxx/dependencies`

Replaces the given service's dependencies completely.

Returns an object mapping all dependency names to their new mount paths.

Request Body (required)

A JSON object mapping dependency names to their new mount paths. Any omitted dependencies will be disabled.

Query Parameters

- *mount* (required): Mount path of the installed service.
- *200*: Returned if the request was sucessful.

Foxx Service Miscellaneous

List service scripts

list service scripts

`GET /_api/foxx/scripts`

Fetches a list of the scripts defined by the service.

Returns an object mapping the raw script names to human-friendly names.

Query Parameters

- *mount* (required): Mount path of the installed service.

Return Codes

- *200*: Returned if the request was sucessful.

Run service script

run service script

`POST /_api/foxx/scripts/{name}`

Runs the given script for the service at the given mount path.

Returns the exports of the script, if any.

Request Body (optional)

An arbitrary JSON value that will be parsed and passed to the script as its first argument.

Query Parameters

- *name* (required): Name of the script to run.

Query Parameters

- *mount* (required): Mount path of the installed service.

Return Codes

- *200*: Returned if the request was sucessful.

Run service tests

run service tests

`POST /_api/foxx/tests`

Runs the tests for the service at the given mount path and returns the results.

Supported test reporters are:

- *default*: a simple list of test cases
- *suite*: an object of test cases nested in suites
- *stream*: a raw stream of test results
- *xunit*: an XUnit/JUnit compatible structure
- *tap*: a raw TAP compatible stream

The *Accept* request header can be used to further control the response format:

When using the *stream* reporter `application/x-ldjson` will result in the response body being formatted as a newline-delimited JSON stream.

When using the *tap* reporter `text/plain` or `text/*` will result in the response body being formatted as a plain text TAP report.

When using the *xunit* reporter `application/xml` or `text/xml` will result in the response body being formatted as XML instead of JSONML.

Otherwise the response body will be formatted as non-prettyprinted JSON.

Query Parameters

- *mount* (required): Mount path of the installed service.
- *reporter* (optional): Test reporter to use.
- *idiomatic* (optional): Use the matching format for the reporter, regardless of the *Accept* header.

Return Codes

- *200*: Returned if the request was sucessful.

Enable development mode

enable development mode

`POST /_api/foxx/development`

Puts the service into development mode.

While the service is running in development mode the service will be reloaded from the filesystem and its setup script (if any) will be re-executed every time the service handles a request.

When running ArangoDB in a cluster with multiple coordinators note that changes to the filesystem on one coordinator will not be reflected across the other coordinators. This means you should treat your coordinators as inconsistent as long as any service is running in development mode.

Query Parameters

- *mount* (required): Mount path of the installed service.

Return Codes

- *200*: Returned if the request was sucessful.

Disable development mode

disable development mode

`DELETE /_api/foxx/development`

Puts the service at the given mount path into production mode.

When running ArangoDB in a cluster with multiple coordinators this will replace the service on all other coordinators with the version on this coordinator.

Query Parameters

- *mount* (required): Mount path of the installed service.

Return Codes

- *200*: Returned if the request was sucessful.

Service README

service README

`GET /_api/foxx/readme`

Fetches the service's README or README.md file's contents if any.

Query Parameters

- *mount* (required): Mount path of the installed service.

Return Codes

- *200*: Returned if the request was sucessful.
- *204*: Returned if no README file was found.

Swagger description

swagger description

`GET /_api/foxx/swagger`

Fetches the Swagger API description for the service at the given mount path.

The response body will be an OpenAPI 2.0 compatible JSON description of the service API.

Query Parameters

- *mount* (required): Mount path of the installed service.

Return Codes

- *200*: Returned if the request was sucessful.

Download service bundle

download service bundle

`POST /_api/foxx/download`

Downloads a zip bundle of the service directory.

When development mode is enabled, this always creates a new bundle.

Otherwise the bundle will represent the version of a service that is installed on that ArangoDB instance.

Query Parameters

- *mount* (required): Mount path of the installed service.

Return Codes

- *200*: Returned if the request was sucessful.
- *400*: Returned if the mount path is unknown.

Commit local service state

commit local service state

`POST /_api/foxx/commit`

Commits the local service state of the coordinator to the database.

This can be used to resolve service conflicts between coordinators that can not be fixed automatically due to missing data.

Query Parameters

- *replace* (optional): Overwrite existing service files in database even if they already exist.
- *204*: Returned if the request was sucessful.

HTTP Interface for User Management

This is an introduction to ArangoDB's HTTP interface for managing users.

The interface provides a simple means to add, update, and remove users. All users managed through this interface will be stored in the system collection *_users*. You should never manipulate the *_users* collection directly.

This specialized interface intentionally does not provide all functionality that is available in the regular document REST API.

Please note that user operations are not included in ArangoDB's replication.

Create User

Create a new user.

`POST /_api/user`

A JSON object with these properties is required:

- **passwd**: The user password as a string. If no password is specified, the empty string will be used. If you pass the special value *ARANGODB_DEFAULT_ROOT_PASSWORD*, then the password will be set the value stored in the environment variable `ARANGODB_DEFAULT_ROOT_PASSWORD`. This can be used to pass an instance variable into ArangoDB. For example, the instance identifier from Amazon.
- **active**: An optional flag that specifies whether the user is active. If not specified, this will default to true
- **user**: The name of the user as a string. This is mandatory.
- **extra**: An optional JSON object with arbitrary extra data about the user.

Create a new user. You need server access level *Administrate* in order to execute this REST call.

Return Codes

- *201*: Returned if the user can be added by the server
- *400*: If the JSON representation is malformed or mandatory data is missing from the request.
- *401*: Returned if you have *No access* database access level to the *_system* database.
- *403*: Returned if you have *No access* server access level.
- *409*: Returned if a user with the same name already exists.

Examples

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/user <<EOF
{
  "user" : "admin@example",
  "passwd" : "secure"
}
EOF

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Set the database access level

Set the database access level.

`PUT /_api/user/{user}/database/{dbname}`

A JSON object with these properties is required:

- **grant**: Use "rw" to set the database access level to *Administrate* . Use "ro" to set the database access level to *Access*. Use "none" to set the database access level to *No access*.

Sets the database access levels for the database *dbname* of user *user*. You need the *Administrate* server access level in order to execute this REST call.

Path Parameters

- *user* (required): The name of the user.
- *dbname* (required): The name of the database.

Return Codes

- *200*: Returned if the access level was changed successfully.
- *400*: If the JSON representation is malformed or mandatory data is missing from the request.
- *401*: Returned if you have *No access* database access level to the *_system* database.
- *403*: Returned if you have *No access* server access level.

Examples

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/user/admin@myapp/database/_system <<EOF
{
  "grant" : "rw"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Set the collection access level

Set the collection access level.

`PUT /_api/user/{user}/database/{dbname}/{collection}`

A JSON object with these properties is required:

- **grant**: Use "rw" to set the collection level access to *Read/Write*. Use "ro" to set the collection level access to *Read Only*. Use "none" to set the collection level access to *No access*.

Sets the collection access level for the *collection* in the database *dbname* for user *user*. You need the *Administrate* server access level in order to execute this REST call.

Path Parameters

- *user* (required): The name of the user.
- *dbname* (required): The name of the database.
- *collection* (required): The name of the collection.

Return Codes

- *200*: Returned if the access permissions were changed successfully.
- *400*: If the JSON representation is malformed or mandatory data is missing from the request.

- *401*: Returned if you have *No access* database access level to the *_system* database.
- *403*: Returned if you have *No access* server access level.

Examples

```
shell> curl -X PUT --data-binary @- --dump -
http://localhost:8529/_api/user/admin@myapp/database/_system/reports <<EOF
{
  "grant" : "rw"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Clear the database access level

Clear the database access level, revert back to the default access level

`DELETE /_api/user/{user}/database/{dbname}`

Path Parameters

- *user* (required): The name of the user.
- *dbname* (required): The name of the database.

Clears the database access level for the database *dbname* of user *user*. As consequence the default database access level is used. If there is no defined default database access level, it defaults to *No access*. You need permission to the *_system* database in order to execute this REST call.

Return Codes

- *202*: Returned if the access permissions were changed successfully.
- *400*: If the JSON representation is malformed or mandatory data is missing from the request.

Examples

```
shell> curl -X DELETE --dump -
http://localhost:8529/_api/user/admin@myapp/database/_system

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

"{\"error\":false,\"code\":202}"
```

Clear the collection access level

Clear the collection access level, revert back to the default access level

`DELETE /_api/user/{user}/database/{dbname}/{collection}`

Path Parameters

- *user* (required): The name of the user.

- *dbname* (required): The name of the database.
- *collection* (required): The name of the collection.

Clears the collection access level for the collection *collection* in the database *dbname* of user *user*. As consequence the default collection access level is used. If there is no defined default collection access level, it defaults to *No access*. You need permissions to the *_system* database in order to execute this REST call.

Return Codes

- *202*: Returned if the access permissions were changed successfully.
- *400*: If there was an error

Examples

```
shell> curl -X DELETE --dump -
http://localhost:8529/_api/user/admin@myapp/database/_system/reports

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

"{\"error\":false,\"code\":202}"
```

List the accessible databases for a user

List the accessible databases for a user

`GET /_api/user/{user}/database/`

Path Parameters

- *user* (required): The name of the user for which you want to query the databases.

Query Parameters

- *full* (optional): Return the full set of access levels for all databases and all collections.

Fetch the list of databases available to the specified *user*. You need *Administrate* for the server access level in order to execute this REST call.

The call will return a JSON object with the per-database access privileges for the specified user. The *result* object will contain the databases names as object keys, and the associated privileges for the database as values.

In case you specified *full*, the result will contain the permissions for the databases as well as the permissions for the collections.

Return Codes

- *200*: Returned if the list of available databases can be returned.
- *400*: If the access privileges are not right etc.
- *401*: Returned if you have *No access* database access level to the *_system* database.
- *403*: Returned if you have *No access* server access level.

Examples

```
shell> curl --dump - http://localhost:8529/_api/user/anotherAdmin@secapp/database/

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
With the full response format:

```
shell> curl --dump - http://localhost:8529/_api/user/anotherAdmin@secapp/database?
full=true

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Get the database access level

Get specific database access level

`GET /_api/user/{user}/database/{database}`

Path Parameters

- *user* (required): The name of the user for which you want to query the databases.
- *database* (required): The name of the database to query

Fetch the database access level for a specific database

Return Codes

- *200*: Returned if the acccess level can be returned
- *400*: If the access privileges are not right etc.
- *401*: Returned if you have *No access* database access level to the *_system* database.
- *403*: Returned if you have *No access* server access level.

Examples

```
shell> curl --dump - http://localhost:8529/_api/user/anotherAdmin@secapp/database/_system

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Get the specific collection access level

Get the collection access level

`GET /_api/user/{user}/database/{database}/{collection}`

Path Parameters

- *user* (required): The name of the user for which you want to query the databases.
- *database* (required): The name of the database to query
- *collection* (required): The name of the collection

Returns the collection access level for a specific collection

Return Codes

- *200*: Returned if the acccess level can be returned
- *400*: If the access privileges are not right etc.
- *401*: Returned if you have *No access* database access level to the *_system* database.
- *403*: Returned if you have *No access* server access level.

Examples

```
shell> curl --dump -
http://localhost:8529/_api/user/anotherAdmin@secapp/database/_system/_users

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Replace User

Replace an existing user.

`PUT /_api/user/{user}`

Path Parameters

- *user* (required): The name of the user

A JSON object with these properties is required:

- **passwd**: The user password as a string. Specifying a password is mandatory, but the empty string is allowed for passwords
- **active**: An optional flag that specifies whether the user is active. If not specified, this will default to true
- **extra**: An optional JSON object with arbitrary extra data about the user.

Replaces the data of an existing user. The name of an existing user must be specified in *user*. You need server access level *Administrate* in order to execute this REST call. Additionally, a user can change his/her own data.

Return Codes

- *200*: Is returned if the user data can be replaced by the server.
- *400*: The JSON representation is malformed or mandatory data is missing from the request
- *401*: Returned if you have *No access* database access level to the *_system* database.
- *403*: Returned if you have *No access* server access level.
- *404*: The specified user does not exist

Examples

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/user/admin@myapp
<<EOF
{
  "passwd" : "secure"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Modify User

Modify attributes of an existing user

`PATCH /_api/user/{user}`

Path Parameters

- *user* (required): The name of the user

A JSON object with these properties is required:

- **passwd**: The user password as a string. Specifying a password is mandatory, but the empty string is allowed for passwords
- **active**: An optional flag that specifies whether the user is active. If not specified, this will default to true
- **extra**: An optional JSON object with arbitrary extra data about the user.

Partially updates the data of an existing user. The name of an existing user must be specified in *user*. You need server access level *Administrate* in order to execute this REST call. Additionally, a user can change his/her own data.

Return Codes

- *200*: Is returned if the user data can be replaced by the server.
- *400*: The JSON representation is malformed or mandatory data is missing from the request.
- *401*: Returned if you have *No access* database access level to the *_system* database.
- *403*: Returned if you have *No access* server access level.
- *404*: The specified user does not exist

Examples

```
shell> curl -X PATCH --data-binary @- --dump - http://localhost:8529/_api/user/admin@myapp
<<EOF
{
  "passwd" : "secure"
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Remove User

delete a user permanently.

`DELETE /_api/user/{user}`

Path Parameters

- *user* (required): The name of the user

Removes an existing user, identified by *user*. You need *Administrate* for the server access level in order to execute this REST call.

Return Codes

- *202*: Is returned if the user was removed by the server
- *401*: Returned if you have *No access* database access level to the *_system* database.

- *403*: Returned if you have *No access* server access level.
- *404*: The specified user does not exist

Examples

```
shell> curl -X DELETE --data-binary @- --dump -
http://localhost:8529/_api/user/userToDelete@myapp <<EOF
{
}
EOF

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

{
  "error" : false,
  "code" : 202
}
```

Fetch User

fetch the properties of a user.

`GET /_api/user/{user}`

Path Parameters

- *user* (required): The name of the user

Fetches data about the specified user. You can fetch information about yourself or you need the *Administrate* server access level in order to execute this REST call.

Return Codes

- *200*: The user was found.
- *401*: Returned if you have *No access* database access level to the *_system* database.
- *403*: Returned if you have *No access* server access level.
- *404*: The user with the specified name does not exist.

Examples

```
shell> curl --dump - http://localhost:8529/_api/user/admin@myapp

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

List available Users

fetch the properties of a user.

`GET /_api/user/`

Fetches data about all users. You need the *Administrate* server access level in order to execute this REST call. Otherwise, you will only get information about yourself.

The call will return a JSON object with at least the following attributes on success:

- *user*: The name of the user as a string.
- *active*: An optional flag that specifies whether the user is active.
- *extra*: An optional JSON object with arbitrary extra data about the user.

Return Codes

- *200*: The users that were found.
- *401*: Returned if you have *No access* database access level to the *_system* database.
- *403*: Returned if you have *No access* server access level.

Examples

```
shell> curl --dump - http://localhost:8529/_api/user

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

HTTP tasks Interface

Following you have ArangoDB's HTTP Interface for Tasks.

There are also some examples provided for every API action.

Fetch all tasks or one task

Retrieves all currently active server tasks

`GET /_api/tasks/`

fetches all existing tasks on the server

- *200*: The list of tasks

Examples

Fetching all tasks

```
shell> curl --dump - http://localhost:8529/_api/tasks

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

[
  {
    "id" : "statistics-gc",
    "name" : "statistics-gc",
    "created" : 1540031948.5958247,
    "type" : "periodic",
    "period" : 450,
    "offset" : 224.786226,
    "command" : "(function (params) { require('@arangodb/statistics').garbageCollector();
} )(params);",
    "database" : "_system"
  },
  {
    "id" : "statistics-average-collector",
    "name" : "statistics-average-collector",
    "created" : 1540031948.5957427,
    "type" : "periodic",
    "period" : 900,
    "offset" : 20,
    "command" : "(function (params) { require('@arangodb/statistics').historianAverage();
} )(params);",
    "database" : "_system"
  },
  {
    "id" : "statistics-collector",
    "name" : "statistics-collector",
    "created" : 1540031948.5956516,
    "type" : "periodic",
    "period" : 10,
    "offset" : 1,
    "command" : "(function (params) { require('@arangodb/statistics').historian(); } )
```

```
(params);",
    "database" : "_system"
  },
  {
    "id" : "86",
    "name" : "user-defined task",
    "created" : 1540031938.657401,
    "type" : "periodic",
    "period" : 1,
    "offset" : 0.000001,
    "command" : "(function (params) { (function () {\n
require('@arangodb/foxx/queues/manager').manage();\n      })(params) } )(params);",
    "database" : "_system"
  }
]
```

Fetch one task with id

Retrieves one currently active server task

`GET /_api/tasks/{id}`

- *id* (required): The id of the task to fetch.

fetches one existing tasks on the server specified by *id*

- *200*: The requested task

Examples

Fetching a single task by its id

```
shell> curl --dump - http://localhost:8529/_api/tasks/statistics-average-collector

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
trying to fetch a non-existing task

```
shell> curl --dump - http://localhost:8529/_api/tasks/non-existing-task

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

creates a task

creates a new task

`POST /_api/tasks`

A JSON object with these properties is required:

- **params**: The parameters to be passed into command

- **offset**: Number of seconds initial delay
- **command**: The JavaScript code to be executed
- **name**: The name of the task
- **period**: number of seconds between the executions

creates a new task with a generated id

Return Codes

- *400*: If the post body is not accurate, a *HTTP 400* is returned.

Examples

```
shell> curl -X POST --data-binary @- --dump - http://localhost:8529/_api/tasks/ <<EOF
{
  "name" : "SampleTask",
  "command" : "(function(params) { require('@arangodb').print(params); })(params)",
  "params" : {
    "foo" : "bar",
    "bar" : "foo"
  },
  "period" : 2
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

creates a task with id

registers a new task with a pre-defined id

`PUT /_api/tasks/{id}`

- *id* (required): The id of the task to create

AJSON object with these properties is required:

- **params**: The parameters to be passed into command
- **offset**: Number of seconds initial delay
- **command**: The JavaScript code to be executed
- **name**: The name of the task
- **period**: number of seconds between the executions

registers a new task with the specified id

Return Codes

- *400*: If the task *id* already exists or the rest body is not accurate, *HTTP 400* is returned.

Examples

```
shell> curl -X PUT --data-binary @- --dump - http://localhost:8529/_api/tasks/sampleTask
<<EOF
{
  "id" : "SampleTask",
  "name" : "SampleTask",
  "command" : "(function(params) { require('@arangodb').print(params); })(params)",
```

```
  "params" : {
    "foo" : "bar",
    "bar" : "foo"
  },
  "period" : 2
}
EOF

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

deletes the task with id

deletes one currently active server task

`DELETE /_api/tasks/{id}`

- *id* (required): The id of the task to delete.

Deletes the task identified by *id* on the server.

Return Codes

- *404*: If the task *id* is unknown, then an *HTTP 404* is returned.

Examples

trying to delete non existing task

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/tasks/NoTaskWithThatName

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Remove existing Task

```
shell> curl -X DELETE --dump - http://localhost:8529/_api/tasks/SampleTask

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff

{
  "error" : false,
  "code" : 200
}
```

HTTP Interface for Agency feature

The Agency is the ArangoDB component which manages the entire ArangoDB cluster. ArangoDB itself mainly uses the Agency as a central place to store the configuration and the cluster nodes health management. It implements the Raft consensus protocol to act as the single-source of truth for the entire cluster. You may know other software providing similar functionality e.g. Apache Zookeeper, etcd or Consul.

To an end-user the Agency is essentially a fault-tolerant Key-Value Store with a simple REST-API. It is possible to use the Agency API for a variety of use-cases, for example:

Centralized configuration repository Service discovery registry Distributed synchronization service Distributed Lock-Manager

Note 1: To access the Agency API with authentication enabled, you need to include an authorization header with every request. The authorization header must contain a superuser JWT Token; For more information see the authentication section.

Note 2: The key-prefix /arango contains ArangoDBs internal configuration. You should never change any values below the arango key.

Key-Value store APIs

Generally, all document IO to and from the key-value store consists of JSON arrays. The outer Array is an envelope for multiple read or write transactions. The results are arrays are an envelope around the results corresponding to the order of the incoming transactions.

Consider the following write operation into a pristine agency:

```
curl -L http://$SERVER:$PORT/_api/agency/write -d '[[{"a":{"op":"set","new":{"b":{"c":[1,2,3]},"e":12}},"d":{"op":"set","new":f
alse}}]]'
```

```
[{results:[1]}]
```

And the subsequent read operation

```
curl -L http://$SERVER:$PORT/_api/agency/read -d '[["/"]]'
```

```
[
  {
    "a": {
      "b": {
        "c": [1,2,3]
      },
      "e": 12
    },
    "d": false
  }
]
```

In the first step we committed a single transaction that commits the JSON document inside the inner transaction array to the agency. The result is `[1]`, which is the replicated log index. Repeated invocation will yield growing log numbers 2, 3, 4, etc.

The read access is a complete access to the key-value store indicated by access to it's root element and returns the result as an array corresponding to the outermost array in the read transaction.

Let's dig in some deeper.

Read API

Let's start with the above initialized key-value store in the following. Let us visit the following read operations:

```
curl -L http://$SERVER:$PORT/_api/agency/read -d '[["/a/b"]]'
```

```
[
  {
    "a": {
      "b": {
        "c": [1,2,3]
      }
    }
  }
]
```

And

```
curl -L http://$SERVER:$PORT/_api/agency/read -d '[["/a/b/c"]]'
```

```
[
  {
    "a": {
      "b": {
        "c": [1,2,3]
      }
    }
  }
]
```

Note that the above results are identical, meaning that results obtained from the agency are always return with full path.

The second outer array brackets in read operations correspond to transactions, meaning that the result is guaranteed to have been acquired without a write transaction in between:

```
curl -L http://$SERVER:$PORT/_api/agency/read -d '[["/a/e"],["/d","/a/b"]]'
```

```
[
  {
    "a": {
      "e": 12
    }
  },
  {
    "a": {
      "b": {
        "c": [1,2,3
        ]
      }
    },
    "d": false
  }
]
```

While the first transaction consists of a single read access to the key-value-store thus stretching the meaning of the word transaction, the second bracket actually hold two disjunct read accesses, which have been joined within zero-time, i.e. without a write access in between. That is to say that `"/d"` cannot have changed before `"/a/b"` had been acquired.

Let's try to fetch a value from the key-value-store, which does not exist:

```
curl -L http://$SERVER:$PORT/_api/agency/read -d '[["/a/b/d"]]'
```

```
[
  {
    "a": {
      "b": {}
    }
  }
]
```

The result returns the cross section of the requested path and the key-value-store contents. `"/a/b"` exists, but there is no key `"/a/b/d"`. Thus the following transaction will yield:

```
curl -L http://$SERVER:$PORT/_api/agency/read -d '[["/a/b/d","/d"]]'
```

```
[
  {
    "a": {
      "b": {}
    },
    "d": false
  }
]
```

And this last read operation should return:

```
curl -L http://$SERVER:$PORT/_api/agency/read -d '[["/a/b/c"],["/a/b/d"],["/a/x/y"],["/y"],["/a/b","/a/x" ]]'
```

```
[
  {"a":{"b":{"c":[1,2,3]}}},
  {"a":{"b":{}}},
  {"a":{}},
  {},
  {"a":{"b":{"c":[1,2,3]}}}
]
```

Write API

The write API must obviously be more versatile and needs a more detailed appreciation. Write operations are arrays of transactions with preconditions, i.e. `[[U,P]]`, where the system tries to apply all updates in the outer array in turn, rejecting those whose precondition is not fulfilled by the current state. It is guaranteed that the transactions in the write request are sequenced adjacent to each other (with no intervention from other write requests). Only the ones with failed preconditions are left out.

For `P`, the value of a key is an object with attributes `"old"`, `"oldNot"`, `"oldEmpty"` or `"isArray"`. With `"old"` one can specify a JSON value that has to be present for the condition to be fulfilled. With `"oldNot"` one may check for a value to not be equal to the test. While with `"oldEmpty"`, which can take a boolean value, one can specify that the key value needs to be not set `true` or set to an arbitrary value `false`. With `"isArray"` one can specify that the value must be an array. As a shortcut, `"old"` values of scalar or array type may be stored directly in the attribute. Examples:

```
{ "/a/b/c": { "old": [1,2,3] }}
```

is a precondition specifying that the previous value of the key `"/a/b/c"` key must be `[1,2,3]`. If and only if the value of the precondition is not an object we provide a notation, where the keywork `old` may be omitted. Thus, the above check may be shortcut as

```
{ "/a/b/c": [1, 2, 3] }
```

Consider the agency in initialized as above let's review the responses from the agency as follows:

```
curl -L http://$SERVER:$PORT/_api/agency/write -d '[[{"/a/b/c":{"op":"set","new":[1,2,3,4]},"/a/b/pi":{"op":"set","new":"some t
ext"}},{"/a/b/c":{"old":[1,2,3]}}]]'
```

```
{
  "results": [19]
}
```

The condition is fulfilled in the first run and would be wrong in a second returning

```
{
  "results": [0]
```

```
}
```

`0` as a result means that the precondition failed and no "real" log number was returned.

```
{ "/a/e": { "oldEmpty": false } }
```

means that the value of the key `"a/e"` must be set (to something, which can be `null`!). The condition

```
{ "/a/e": { "oldEmpty": true } }
```

means that the value of the key `"a/e"` must be unset. The condition

```
{ "/a/b/c": { "isArray": true } }
```

means that the value of the key `"a/b/c"` must be an array.

The update value U is an object, the attribute names are again key strings and the values are objects with optional attributes `"new"`, `"op"` and `"ttl"`. They have the following meaning:

`"op"` determines the operation, possible values are `"set"` (the default, if left out), `"delete"`, `"increment"`, `"decrement"`, `"push"`, `"pop"`, `"shift"` or `"prepend"`

`"new"` is the new value, can be omitted for the `"delete"` operation and for `"increment"` and `"decrement"`, where `1` is implied

`"ttl"`, if present, the new value that is being set gets a time to live in seconds, given by a numeric value in this attribute. It is only guaranteed that the actual removal of the value is done according to the system clock, so up to clock skew between servers. The removal is done by an additional write transaction that is automatically generated between the regular writes.

Additional rule: If none of `"new"` and `"op"` is set or the value is not even an object, then this is to be interpreted as if it were

```
{ "op": "set", "new": <VALUE> }
```

which amounts to setting the value with no precondition.

Examples:

```
{ "/a": { "op": "set", "new": 12 } }
```

sets the value of the key `"/a"` to `12`. The same could have been achieved by

```
{ "/a": 12 }
```

or by

```
{ "/a": { "new": 12} }
```

The operation

```
{ "/a/b": { "new": { "c": [1,2,3,4] } } }
```

sets the key `"/a/b"` to `{"c": [1,2,3,4]}`. Note that in the above example this is the same as setting the value of `"/a/b/c"` to `[1,2,3,4]`. The difference is, that if `a/b` had other sub attributes, then this transaction would delete all these other attributes and make `"/a/b"` equal to `{"c": [1,2,3,4]}`, whereas setting `"/a/b/c"` to `[1,2,3,4]` would retain all attributes other than `"c"` in `"/a/b"`.

Here are some more examples for full transactions (update/precondition pairs). The transaction

```
[ { "/a/b": { "new": { "c": [1,2,3,4] } } },
  { "/a/b": { "old": { "c": [1,2,3] } } } ]
```

sets the key `"/a/b"` to `{"c":[1,2,3,4]}` if and only if it was `{"c":[1,2,3]}` before. Note that this fails if `"/a/b"` had other attributes than `"c"`. The transaction

```
[ { "/x": { "op": "delete" } },
  { "/x": { "old": false } } ]
```

clears the value of the key `"/x"` if this old value was false.

```
[ { "/y": { "new": 13 },
  { "/y": { "oldEmpty": true } } }
```

sets the value of `"/y"` to `13`, but only, if it was unset before.

```
[ { "/z": { "op": "push", "new": "Max" } } ]
```

appends the string `"Max"` to the end of the list stored in the `"z"` attribute, or creates an array `["Max"]` in `"z"` if it was unset or not an array.

```
[ { "/u": { "op": "pop" } } ]
```

removes the last entry of the array stored under `"u"`, if the value of `"u"` is not set or not an array.

HTTP-headers for write operations

`X-ArangoDB-Agency-Mode` with possible values `"waitForCommitted"`, `"waitForSequenced"` and `"noWait"`.

In the first case the write operation only returns when the commit to the replicated log has actually happened. In the second case the write operation returns when the write transactions that fulfilled their preconditions have been sequenced and thus it is known, which of the write transactions in the given array had fulfilled preconditions. In both cases the body is a JSON array containing the indexes of the transactions in the list that had fulfilled preconditions.

In the last case, `"noWait"`, the operation returns immediately, an empty body is returned. To get any information about the result of the operation one has to specify a tag (see below) and ask about the status later on.

`X-ArangoDB-Agency-Tag` with an arbitrary UTF-8 string value.

Observers

External services to the agency may announce themselves or others to be observers of arbitrary existing or future keys in the key-value-store. The agency must then inform the observing service of any changes to the subtree below the observed key. The notification is done by virtue of POST requests to a required valid URL.

In order to observe any future modification below say `"/a/b/c"`, a observer is announced through posting the below document to the agency's write REST handler:

```
[ { "/a/b/c":
    { "op":  "observe",
      "url": "http://<host>:<port>/<path>"
    }
  } ]
```

The observer is notified of any changes to that target until such time that it removes itself as an observer of that key through

```
[ { "/a/b/c":
    { "op":  "unobserve",
      "url": “http://<host>:<port>/<path>" } } ]
```

Note that the last document removes all observations from entities below `"/a/b/c"`. In particular, issuing

```
[ { "/": "unobserve", "url": "http://<host>:<port>/<path>"} ]
```

will result in the removal of all observations for URL `"http://<host>:<port>/<path>"` . The notifying POST requests are submitted immediately with any complete array of changes to the read db of the leader of create, modify and delete events accordingly; The body

```
{ "term": "5",
  "index": 167,
  "/": {
    "/a/b/c" : { "op": "modify", "old": 1, "new": 2 } },
    "/constants/euler" : {"op": "create", "new": 2.718281828459046 },
    "/constants/pi": { "op": "delete" } } }
```

Configuration

At all times, i.e. regardless of the state of the agents and the current health of the RAFT consensus, one can invoke the configuration API:

curl http://$SERVER:$PORT/_api/agency/config

Here, and in all subsequent calls, we assume that `$SERVER` is replaced by the server name and `$PORT` is replaced by the port number. We use `curl` throughout for the examples, but any client library performing HTTP requests should do. The output might look somewhat like this

```
{
  "term": 1,
  "leaderId": "f5d11cde-8468-4fd2-8747-b4ef5c7dfa98",
  "lastCommitted": 1,
  "lastAcked": {
    "ac129027-b440-4c4f-84e9-75c042942171": 0.21,
    "c54dbb8a-723d-4c82-98de-8c841a14a112": 0.21,
    "f5d11cde-8468-4fd2-8747-b4ef5c7dfa98": 0
  },
  "configuration": {
    "pool": {
      "ac129027-b440-4c4f-84e9-75c042942171": "tcp://localhost:8531",
      "c54dbb8a-723d-4c82-98de-8c841a14a112": "tcp://localhost:8530",
      "f5d11cde-8468-4fd2-8747-b4ef5c7dfa98": "tcp://localhost:8529"
    },
    "active": [
      "ac129027-b440-4c4f-84e9-75c042942171",
      "c54dbb8a-723d-4c82-98de-8c841a14a112",
      "f5d11cde-8468-4fd2-8747-b4ef5c7dfa98"
    ],
    "id": "f5d11cde-8468-4fd2-8747-b4ef5c7dfa98",
    "agency size": 3,
    "pool size": 3,
    "endpoint": "tcp://localhost:8529",
    "min ping": 0.5,
    "max ping": 2.5,
    "supervision": false,
    "supervision frequency": 5,
    "compaction step size": 1000,
    "supervision grace period": 120
  }
}
```

This is the actual output of a healthy agency. The configuration of the agency is found in the `configuration` section as you might have guessed. It is populated by static information on the startup parameters like `agency size` , the once generated `unique id` etc. It holds information on the invariants of the RAFT algorithm and data compaction.

The remaining data reflect the variant entities in RAFT, as `term` and `leaderId` , also some debug information on how long the last leadership vote was received from any particular agency member. Low term numbers on a healthy network are an indication of good operation environment, while often increasing term numbers indicate, that the network environment and stability suggest to raise the RAFT parameters `min ping` and 'max ping' accordingly.

HTTP Interface for Miscellaneous functions

This is an overview of ArangoDB's HTTP interface for miscellaneous functions.

Return server version

returns the server version number

`GET /_api/version`

Query Parameters

- *details* (optional): If set to *true*, the response will contain a *details* attribute with additional information about included components and their versions. The attribute names and internals of the *details* object may vary depending on platform and ArangoDB version.

Returns the server name and version number. The response is a JSON object with the following attributes:

HTTP 200

A json document with these Properties is returned:

is returned in all cases.

- **version**: the server version string. The string has the format "*major.minor.sub*". *major* and *minor* will be numeric, and *sub* may contain a number or a textual version.
- **details**: an optional JSON object with additional details. This is returned only if the *details* query parameter is set to *true* in the request.
- **server**: will always contain *arango*

Return Codes

- *200*: is returned in all cases.

Response Body

- **version**: the server version string. The string has the format "*major.minor.sub*". *major* and *minor* will be numeric, and *sub* may contain a number or a textual version.
- **details**: an optional JSON object with additional details. This is returned only if the *details* query parameter is set to *true* in the request.
- **server**: will always contain *arango*

Examples

Return the version information

```
shell> curl --dump - http://localhost:8529/_api/version

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body
Return the version information with details

```
shell> curl --dump - http://localhost:8529/_api/version?details=true

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

```
```

show response body

Return server database engine type

returns the engine the type the server is running with

`GET /_api/engine`

Returns the storage engine the server is configured to use. The response is a JSON object with the following attributes:

HTTP 200

A json document with these Properties is returned:

is returned in all cases.

- **name**: will be *mmfiles* or *rocksdb*

Return Codes

- *200*: is returned in all cases.

Response Body

- **name**: will be *mmfiles* or *rocksdb*

Examples

Return the active storage engine

```
shell> curl --dump - http://localhost:8529/_api/engine

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
x-content-type-options: nosniff
```

show response body

Flushes the write-ahead log

Sync the WAL to disk.

`PUT /_admin/wal/flush`

Query Parameters

- *waitForSync* (optional): Whether or not the operation should block until the not-yet synchronized data in the write-ahead log was synchronized to disk.
- *waitForCollector* (optional): Whether or not the operation should block until the data in the flushed log has been collected by the write-ahead log garbage collector. Note that setting this option to *true* might block for a long time if there are long-running transactions and the write-ahead log garbage collector cannot finish garbage collection.

Flushes the write-ahead log. By flushing the currently active write-ahead logfile, the data in it can be transferred to collection journals and datafiles. This is useful to ensure that all data for a collection is present in the collection journals and datafiles, for example, when dumping the data of a collection.

Return Codes

- *200*: Is returned if the operation succeeds.
- *405*: is returned when an invalid HTTP method is used.

Retrieves the configuration of the write-ahead log

fetch the current configuration.

`GET /_admin/wal/properties`

Retrieves the configuration of the write-ahead log. The result is a JSON object with the following attributes:

- *allowOversizeEntries*: whether or not operations that are bigger than a single logfile can be executed and stored
- *logfileSize*: the size of each write-ahead logfile
- *historicLogfiles*: the maximum number of historic logfiles to keep
- *reserveLogfiles*: the maximum number of reserve logfiles that ArangoDB allocates in the background
- *syncInterval*: the interval for automatic synchronization of not-yet synchronized write-ahead log data (in milliseconds)
- *throttleWait*: the maximum wait time that operations will wait before they get aborted if case of write-throttling (in milliseconds)
- *throttleWhenPending*: the number of unprocessed garbage-collection operations that, when reached, will activate write-throttling. A value of *0* means that write-throttling will not be triggered.

Return Codes

- *200*: Is returned if the operation succeeds.
- *405*: is returned when an invalid HTTP method is used.

Configures the write-ahead log

configure parameters of the wal

`PUT /_admin/wal/properties`

Configures the behavior of the write-ahead log. The body of the request must be a JSON object with the following attributes:

- *allowOversizeEntries*: whether or not operations that are bigger than a single logfile can be executed and stored
- *logfileSize*: the size of each write-ahead logfile
- *historicLogfiles*: the maximum number of historic logfiles to keep
- *reserveLogfiles*: the maximum number of reserve logfiles that ArangoDB allocates in the background
- *throttleWait*: the maximum wait time that operations will wait before they get aborted if case of write-throttling (in milliseconds)
- *throttleWhenPending*: the number of unprocessed garbage-collection operations that, when reached, will activate write-throttling. A value of *0* means that write-throttling will not be triggered.

Specifying any of the above attributes is optional. Not specified attributes will be ignored and the configuration for them will not be modified.

Return Codes

- *200*: Is returned if the operation succeeds.
- *405*: is returned when an invalid HTTP method is used.

Returns information about the currently running transactions

returns information about the currently running transactions

`GET /_admin/wal/transactions`

Returns information about the currently running transactions. The result is a JSON object with the following attributes:

- *runningTransactions*: number of currently running transactions
- *minLastCollected*: minimum id of the last collected logfile (at the start of each running transaction). This is *null* if no transaction is running.
- *minLastSealed*: minimum id of the last sealed logfile (at the start of each running transaction). This is *null* if no transaction is running.

Return Codes

- *200*: Is returned if the operation succeeds.
- *405*: is returned when an invalid HTTP method is used.

Return system time

Get the current time of the system

`GET /_admin/time`

The call returns an object with the attribute *time*. This contains the current system time as a Unix timestamp with microsecond precision.

Return Codes

- *200*: Time was returned successfully.

Return current request

Send back what was sent in, headers, post body etc.

`GET /_admin/echo`

The call returns an object with the following attributes:

- *headers*: object with HTTP headers received
- *requestType*: the HTTP request method (e.g. GET)
- *parameters*: object with query parameters received

Return Codes

- *200*: Echo was returned successfully.

Return the required version of the database

returns the version of the database.

`GET /_admin/database/target-version`

Returns the database-version that this server requires. The version is returned in the *version* attribute of the result.

Return Codes

- *200*: Is returned in all cases.

Initiate shutdown sequence

initiates the shutdown sequence

`DELETE /_admin/shutdown`

This call initiates a clean shutdown sequence. Requires administrive privileges

Return Codes

- *200*: is returned in all cases.

Runs tests on server

show the available unittests on the server.

`POST /_admin/test`

Request Body (required)

A JSON object containing an attribute *tests* which lists the files containing the test suites.

Executes the specified tests on the server and returns an object with the test results. The object has an attribute "error" which states whether any error occurred. The object also has an attribute "passed" which indicates which tests passed and which did not.

Return Codes

- *200*: is returned when everything went well.

Execute program

Execute a script on the server.

`POST /_admin/execute`

Request Body (required)

The body to be executed.

Executes the javascript code in the body on the server as the body of a function with no arguments. If you have a *return* statement then the return value you produce will be returned as content type *application/json*. If the parameter *returnAsJSON* is set to *true*, the result will be a JSON object describing the return value directly, otherwise a string produced by JSON.stringify will be returned.

Note that this API endpoint will only be present if the server was started with the option `--javascript.allow-admin-execute true`.

The default value of this option is `false`, which disables the execution of user-defined code and disables this API endpoint entirely. This is also the recommended setting for production.

Return Codes

- *200*: is returned when everything went well.

Return status information

returns status information of the server.

`GET /_admin/status`

Returns status information about the server.

This is intended for manual use by the support and should never be used for monitoring or automatic tests. The results are subject to change without notice.

The call returns an object with the following attributes:

- *server*: always *arango*.
- *license*: either *community* or *enterprise*.
- *version*: the server version as string.
- *mode* : either *server* or *console*.
- *host*: the hostname, see *ServerState*.
- *serverInfo.role*: either *SINGLE*, *COORDINATOR*, *PRIMARY*, *AGENT*.
- *serverInfo.writeOpsEnabled*: boolean, true if writes are enabled.
- *serverInfo.maintenance*: boolean, true if maintenace mode is enabled.
- *agency.endpoints*: a list of possible agency endpoints.

An agent, coordinator or primary will also have

- *serverInfo.persistedId*: the persisted ide, e. g. *"CRDN-e427b441-5087-4a9a-9983-2fb1682f3e2a"*.

A coordinator or primary will also have

- *serverInfo.state*: *SERVING*
- *serverInfo.address*: the address of the server, e. g. *tcp://[::1]:8530*.
- *serverInfo.serverId*: the server ide, e. g. *"CRDN-e427b441-5087-4a9a-9983-2fb1682f3e2a"*.

A coordintor will also have

- *coordinator.foxxmaster*: the server id of the foxx master.
- *coordinator.isFoxxmaster*: boolean, true if the server is the foxx master.

An agent will also have

- *agent.id*: server id of this agent.
- *agent.leaderId*: server id of the leader.
- *agent.leading*: boolean, true if leading.
- *agent.endpoint*: the endpoint of this agent.
- *agent.term*: current term number.

Return Codes

- *200*: Status information was returned successfully.

Repair Jobs

distributeShardsLike

Before versions 3.2.12 and 3.3.4 there was a bug in the collection creation which could lead to a violation of the property that its shards were distributed on the DBServers exactly as the prototype collection from the `distributeShardsLike` setting.

Please read everything carefully before using this API!

There is a job that can restore this property safely. However, while the job is running,

- the `replicationFactor` *must not be changed* for any affected collection or prototype collection (i.e. set in `distributeShardsLike`, including SmartGraphs),
- *neither should shards be moved* of one of those prototypes
- and shutdown of DBServers *should be avoided* during the repairs. Also only one repair job should run at any given time. Failure to meet those requirements will mostly cause the job to abort, but still allow to restart it safely. However, changing the `replicationFactor` during repairs may leave it in a state that is not repairable without manual intervention!

Shutting down the coordinator which executes the job will abort it, but it can safely be restarted on another coordinator. However, there may still be a shard move ongoing even after the job stopped. If the job is started again before the move is finished, repairing the affected collection will fail, but the repair can be restarted safely.

If there is any affected collection which `replicationFactor` is equal to the total number of DBServers, the repairs might abort. In this case, it is necessary to reduce the `replicationFactor` by one (or add a DBServer). The job will not do that automatically.

Generally, the job will abort if any of its assumptions fail, at the start or during the repairs. It can be started again and will resume from the current state.

Testing with `GET /_admin/repairs/distributeShardsLike`

Using `GET` will **not** trigger any repairs, but only calculate and return the operations necessary to repair the cluster. This way, you can also check if there is something to repair.

```
$ wget -qSO - http://localhost:8529/_admin/repair/distributeShardsLike | jq .
  HTTP/1.1 200 OK
  X-Content-Type-Options: nosniff
  Server: ArangoDB
  Connection: Keep-Alive
  Content-Type: application/json; charset=utf-8
  Content-Length: 53
{
  "error": false,
  "code": 200,
  "message": "Nothing to do."
}
```

In the example above, all collections with `distributeShardsLike` have their shards distributed correctly. The response if something is broken looks like this:

```
{
  "error": false,
  "code": 200,
  "collections": {
    "_system/someCollection": {
      "PlannedOperations": [
        {
          "BeginRepairsOperation": {
            "database": "_system",
            "collection": "someCollection",
            "distributeShardsLike": "aPrototypeCollection",
            "renameDistributeShardsLike": true,
            "replicationFactor": 4
          }
```

```
        },
        {
          "MoveShardOperation": {
            "database": "_system",
            "collection": "someCollection",
            "shard": "s2000109",
            "from": "PRMR-6b8c84be-1e80-4085-9065-177c6e31a702",
            "to": "PRMR-d3e62c96-c3f7-4766-bac6-f3bf8026f59a",
            "isLeader": false
          }
        },
        {
          "MoveShardOperation": {
            "database": "_system",
            "collection": "someCollection",
            "shard": "s2000109",
            "from": "PRMR-ee3d7af6-1fbf-4ab7-bfd1-56d0a1c1c9b9",
            "to": "PRMR-6b8c84be-1e80-4085-9065-177c6e31a702",
            "isLeader": true
          }
        },
        {
          "FixServerOrderOperation": {
            "database": "_system",
            "collection": "someCollection",
            "distributeShardsLike": "aPrototypeCollection",
            "shard": "s2000109",
            "distributeShardsLikeShard": "s2000092",
            "leader": "PRMR-6b8c84be-1e80-4085-9065-177c6e31a702",
            "followers": [
              "PRMR-99c2ac17-f417-4710-82aa-8350417dd089",
              "PRMR-3b0b85de-882b-4eb2-bbf2-ef1018bdc81e",
              "PRMR-d3e62c96-c3f7-4766-bac6-f3bf8026f59a"
            ],
            "distributeShardsLikeFollowers": [
              "PRMR-d3e62c96-c3f7-4766-bac6-f3bf8026f59a",
              "PRMR-99c2ac17-f417-4710-82aa-8350417dd089",
              "PRMR-3b0b85de-882b-4eb2-bbf2-ef1018bdc81e"
            ]
          }
        },
        {
          "FinishRepairsOperation": {
            "database": "_system",
            "collection": "someCollection",
            "distributeShardsLike": "aPrototypeCollection",
            "shards": [
              {
                "shard": "s2000109",
                "protoShard": "s2000092",
                "dbServers": [
                  "PRMR-6b8c84be-1e80-4085-9065-177c6e31a702",
                  "PRMR-d3e62c96-c3f7-4766-bac6-f3bf8026f59a",
                  "PRMR-99c2ac17-f417-4710-82aa-8350417dd089",
                  "PRMR-3b0b85de-882b-4eb2-bbf2-ef1018bdc81e"
                ]
              },
              {
                "shard": "s2000110",
                "protoShard": "s2000093",
                "dbServers": [
                  "PRMR-d3e62c96-c3f7-4766-bac6-f3bf8026f59a",
                  "PRMR-ee3d7af6-1fbf-4ab7-bfd1-56d0a1c1c9b9",
                  "PRMR-6b8c84be-1e80-4085-9065-177c6e31a702",
                  "PRMR-99c2ac17-f417-4710-82aa-8350417dd089"
                ]
              },
[...]
            ]
          }
        }
      ],
      "error": false
    }
  }
}
```

If something is to be repaired, the response will have the property `collections` with an entry `<db>/<collection>` for each collection which has to be repaired. Each collection also as a separate `error` property which will be `true` iff an error occured for this collection (and `false` otherwise). If `error` is `true`, the properties `errorNum` and `errorMessage` will also be set, and in some cases also `errorDetails` with additional information on how to handle a specific error.

Repairing with `POST /_admin/repairs/distributeShardsLike`

As this job possibly has to move a lot of data around, it can take a while depending on the size of the affected collections. So this should *not be called synchronously*, but only via Async Results: i.e., set the header `x-arango-async: store` to put the job into background and get its results later. Otherwise the request will most probably result in a timeout and the response will be lost! The job will still continue unless the coordinator is stopped, but there is no way to find out if it is still running, or get success or error information afterwards.

Starting the job in background can be done like so:

```
$ wget --method=POST --header='x-arango-async: store' -qSO - http://localhost:8529/_admin/repair/distributeShardsLike
  HTTP/1.1 202 Accepted
  X-Content-Type-Options: nosniff
  X-Arango-Async-Id: 152223973119118
  Server: ArangoDB
  Connection: Keep-Alive
  Content-Type: text/plain; charset=utf-8
  Content-Length: 0
```

This line is of notable importance:

```
  X-Arango-Async-Id: 152223973119118
```

as it contains the job id which can be used to fetch the state and results of the job later. `GET` ting `/_api/job/pending` and `/_api/job/done` will list job ids of jobs that are pending or done, respectively.

This can also be done with the `GET` method for testing.

The job api must be used to fetch the state and results. It will return a `204` while the job is running. The actual response will be returned only once, after that the job is deleted and the api will return a `404`. It is therefore recommended to write the response directly to a file for later inspection. Fetching the result is done by calling `/_api/job` via `PUT`:

```
$ wget --method=PUT -qSO - http://localhost:8529/_api/job/152223973119118 | jq .
  HTTP/1.1 200 OK
  X-Content-Type-Options: nosniff
  X-Arango-Async-Id: 152223973119118
  Server: ArangoDB
  Connection: Keep-Alive
  Content-Type: application/json; charset=utf-8
  Content-Length: 53
{
  "error": false,
  "code": 200,
  "message": "Nothing to do."
}
```

The final response will look like the response of the `GET` call. If an error occured the response should contain details on how to proceed. If in doubt, ask as on Slack: https://arangodb.com/community/