Clusto API Server Documentation

Release 0.5.2

Jorge Gallegos

August 21, 2015

Contents

CHAPTER 1

About

The Clusto API server intends to replace the direct database approach used by most clusto commands and scripts right now. The goal is to make this API robust enough to permit all operations usually handled via clusto commands or the clusto shell but using a REST/HTTP interface.

CHAPTER 2

Version 0.5.2 (2015-08-21)

New feature/bugfix release

2.1 Features

- Adding developer documentation (Jorge Gallegos)

2.2 Bugfixes

- Fixed the OPTIONS dummy routers (Jorge Gallegos)

CHAPTER 3

Changelog

3.1 Version 0.2.0 (2014-09-25)

First "real" release, including additional tests and finally some PR contributions :) (Thanks James!)

3.1.1 Features

- Split install requirements into 3 files, which can be read either by `pip` or by the `setup.py` script.
  - `requirements.txt` is the usual file you install from
  - `test-requirements.txt` is a file intended for use when you are testing this repo.
  - `dev-requirements.txt` is the file you should install from if you are hacking in this repo.
- Added a custom `Clusto-Mode` header to change how the JSON data is presented back:
  - `compact` mode: is the default for API calls that return a list of items back to the requestor.
  - `expanded` mode: is the default for API calls that return a single item back to the requestor
- Added a `by-names` endpoint, it receives a list of parameters and will return back a list of items found

3.2 Version 0.2.1 (2014-09-26)

Small bugfix release

3.2.1 Bugfixes

- Adding `requirements.txt`, `dev-requirements.txt` and `test-requirements.txt` to the distribution tarball for source downloads

3.3 Version 0.3.0 (2015-02-13)

Feature release

3.3.1 Features

- The `/by-attr` endpoint, which translates to the main clusto query method `get_by_attr()`
- Pagination implementation for results. By default it is turned off, basically only useful for those cases where your data is still too much. One important caveat here is that this only affects the **presentation** of the data back to the client, not the **querying** of the data from the database. Pagination is not native to clusto, yet
- Minify implementation for results. All results by default are *pretty printed* but if you pass a request header they will be printed as a single string

3.4 Version 0.3.1 (2015-02-13)

New release only to deal with dumb pypi upload error hurr.

3.5 Version 0.3.2 (2015-02-17)

New bugfix release

3.5.1 Bugfixes

- Fixed slicing in pagination logic

3.6 Version 0.4.0 (2015-05-27)

New feature release with some API changes (read at the bottom) which will hopefully stop being a thing once we hit 1.0

3.6.1 Features

- Adding a main /driverlist endpoint to match clusto.driverlist
- Added support for extra server adapter options in config
- Better test cases for resource manager core application
- Support for pool removal with caveats, read below for changes

3.6.2 API Changes

- Pool insertion changed from a PUT to a generic POST action, briefly: PUT for `.insert()` didn't make much sense without support for `.remove()` but `.remove()` isn't properly a DELETE action. Ergo we changed it to be a generic POST to the endpoint with an `action=[insert,remove]` field

3.7 Version 0.4.1 (2015-06-04)

New feature release

3.7.1 Features

- Add get_ips list to the show method (James Cunningham)
- Add ability to post json bodies in attribute app (James Cunningham)
- Added expansion and paging to the entity app (James Cunningham)

3.8 Version 0.4.2 (2015-06-18)

New bugfix release

3.8.1 Bug fixes

- lists and dicts attribute values were returned as a unicode string representation which was unparseable as a regular json object.

3.9 Version 0.5.0 (2015-07-24)

New feature release

3.9.1 Features

- add and set attr marriage, bring a present (James Cunningham)
- add typecast util (James Cunningham)
- add ability to set global response headers in clusto.conf (James Cunningham)

3.10 Version 0.5.1 (2015-08-20)

New feature release

3.10.1 Features

- Add a generic OPTIONS router for (crude) CORS support

3.11 Version 0.5.2 (2015-08-21)

New feature/bugfix release

3.11.1 Features

- Adding developer documentation (Jorge Gallegos)

3.11.2 Bugfixes

- Fixed the OPTIONS dummy routers (Jorge Gallegos)

CHAPTER 4

User documentation

4.1 Main server components

4.1.1 *clustoapi.server*: Server main module

Overview

The Clusto API Server will work as an alternative to the direct database access traditional clusto commands and libraries use.

Sometimes your database has restricted access or you want to expose your clusto information across facilities but don't want to risk opening up your database port to the outside world or don't have the required know-how to do it in a secure manner.

Exposing an HTTP(S) endpoint is a more common problem and as such there are several well-understood solutions. Scalability and security are also points you have to consider.

The Clusto API Server should thus have the following required features:

- Complete object manipulation: create, delete, insert, and remove objects, besides displaying them.
- Complete object attribute manipulation: add, delete, update, query, and displaying attributes
- Resource manipulation: allocate and deallocate objects from resources
- Querying

Custom Headers

The Clusto API Server comes with the ability to pass certain headers to multiple operations.

Clusto-Mode Determines if an object is `compact` or `expanded`. Compaction of objects helps speed up response time for multiple object lookups. `expanded` is the default mode if the function returns only one object, and is `compact` by default for all listing functions.

Clusto-Per-Page Number of entities to return when pagination is requested. Defaults to `50`.

Clusto-Page Requests the current page in a list of entities, delimited by `Clusto-Per-Page`.

Clusto-Pages Response only. This header returns the total number of pages to the requester.

Clusto-Minify If set to `True` (not case sensitive), clusto will not give a response that has been pretty-printed.

Configurable Response Headers

The Clusto API Server comes with the ability to define and configure static response headers within the `clusto.conf`.

> **Example** To enable CORS, a line in the `clusto.conf` would read: `response_headers = Access-Control-Allow-Origin:*`

API Docs

`clustoapi.server.`**`build_docs`**(*module='clustoapi.server'*)

This will build documentation for the given module and all its methods. If python-rest is available, it will attempt to parse it as a restructured text document. You can get to the docs by going to the __doc__ endpoint on each mounted application, the main __doc__ endpoint, or on the main endpoint:

```
$ ${get} ${server_url}/__doc__
<?xml version="1.0" encoding="utf-8" ?>
...
HTTP: 200
Content-type: text/html; charset=UTF-8
```

If you pass the `Accept` headers and specify `text/plain`, you should get the plain text version back

```
$ ${get} -H 'Accept: text/plain' ${server_url}/__doc__
...
HTTP: 200
Content-type: text/plain

$ diff -q <( curl -s -H 'Accept: text/plain' ${server_url}/__doc__ ) <( curl -s -H 'Accept: text
equal
```

In the test config, CORS is configured to be returned with every response header.

Or:

If you try to get a non-configured header, it shouldn't be in the output

`clustoapi.server.`**`favicon`**()

Send an HTTP code to clients so they stop asking for favicon. Example:

```
$ ${get} -o /dev/null ${server_url}/favicon.ico
HTTP: 410
Content-type: text/html; charset=UTF-8
```

`clustoapi.server.`**`get_by_attr`**()

One of the main `clusto` operations. Parameters:

- Required: the `key` parameter
- Optional: the `subkey` parameter
- Optional: the `value` parameter

Examples:

```
$ ${get} ${server_url}/by-attr
"Provide a key to use get_by_attr"
HTTP: 412
Content-type: application/json

$ ${get} -d 'key=nonkey' ${server_url}/by-attr
```

```
    []
    HTTP: 200
    Content-type: application/json

    $ ${get} -d 'key=key1' -d 'subkey=subkey1' -d 'value=value1' ${server_url}/by-attr
    [
        "/basicserver/testserver1"
    ]
    HTTP: 200
    Content-type: application/json

    $ ${get} -H 'Clusto-Mode: expanded' -d 'key=key1' ${server_url}/by-attr
    [
        {
            "attrs": [
                {
                    "datatype": "string",
                    "key": "key1",
                    "number": null,
                    "subkey": "subkey1",
                    "value": "value1"
                }
            ],
            "contents": [],
            "driver": "basicserver",
            "ips": [],
            "name": "testserver1",
            "parents": [
                "/pool/singlepool",
                "/pool/multipool"
            ]
        },
        {
            "attrs": [
                {
                    "datatype": "string",
                    "key": "key1",
                    "number": null,
                    "subkey": "subkey2",
                    "value": "value2"
                }
            ],
            "contents": [],
            "driver": "basicserver",
            "ips": [],
            "name": "testserver2",
            "parents": [
                "/pool/multipool"
            ]
        }
    ]
    HTTP: 200
    Content-type: application/json
```

clustoapi.server.**get_by_name**(*name*)

One of the main `clusto` operations. Parameters:

- Required path parameter: `name` - The name you're looking for

- Optional: `driver` - If provided, a driver check will be added to ensure the resulting object is the type you're expecting

---

Note: This function returns expanded objects by default in order to reduce the amount of required custom headers. Therefore, the header is not required to receive expanded objects.

Examples:

```
$ ${get} ${server_url}/by-name/nonserver
"Object "nonserver" not found (nonserver does not exist.)"
HTTP: 404
Content-type: application/json

$ ${get} -H 'Clusto-Mode: compact' ${server_url}/by-name/testserver1
"/basicserver/testserver1"
HTTP: 200
Content-type: application/json

$ ${get} ${server_url}/by-name/testserver1
{
    "attrs": [
        {
            "datatype": "string",
            "key": "key1",
            "number": null,
            "subkey": "subkey1",
            "value": "value1"
        }
    ],
    "contents": [],
    "driver": "basicserver",
    "ips": [],
    "name": "testserver1",
    "parents": [
        "/pool/singlepool",
        "/pool/multipool"
    ]
}
HTTP: 200
Content-type: application/json

$ ${get} -d 'driver=pool' ${server_url}/by-name/testserver1
"The driver for object "testserver1" is not "pool""
HTTP: 409
Content-type: application/json

$ ${get} -d 'driver=nondriver' ${server_url}/by-name/testserver1
"The driver "nondriver" is not a valid driver"
HTTP: 412
Content-type: application/json
```

`clustoapi.server.`**`get_by_names`**`()`

One of the main `clusto` operations. Parameters:

- Required parameter: At least one `name` parameter

Returns `HTTP: 404` when all entites requested do not exist and `HTTP: 206` when a percent of entities requested do not exist.

Examples:

```
$ ${get} ${server_url}/by-names
"Provide at least one name to get data from"
HTTP: 412
Content-type: application/json

$ ${get} -d 'name=nonserver' ${server_url}/by-names
[
    null
]
HTTP: 404
Content-type: application/json

$ ${get} -d 'name=testserver1' -d 'name=nonserver' ${server_url}/by-names
[
    "/basicserver/testserver1",
    null
]
HTTP: 206
Content-type: application/json

$ ${get} -H 'Clusto-Mode: expanded' -d 'name=testserver1' -d 'name=testserver2' ${server_url}/by
[
    {
        "attrs": [
            {
                "datatype": "string",
                "key": "key1",
                "number": null,
                "subkey": "subkey1",
                "value": "value1"
            }
        ],
        "contents": [],
        "driver": "basicserver",
        "ips": [],
        "name": "testserver1",
        "parents": [
            "/pool/singlepool",
            "/pool/multipool"
        ]
    },
    {
        "attrs": [
            {
                "datatype": "string",
                "key": "key1",
                "number": null,
                "subkey": "subkey2",
                "value": "value2"
            }
        ],
        "contents": [],
        "driver": "basicserver",
        "ips": [],
        "name": "testserver2",
        "parents": [
        "/pool/multipool"
```

```
            ]
        }
    ]
    HTTP: 200
    Content-type: application/json

    $ ${get} -d 'name=nonserver1' -d 'name=nonserver2' ${server_url}/by-names
    [
        null,
        null
    ]
    HTTP: 404
    Content-type: application/json
```

clustoapi.server.**get_driverlist**()

Returns clusto.driverlist, a list of drivers and their qualified class paths.

Examples:

```
    $ ${get} ${server_url}/driverlist
    {
        "basicappliance": "clusto.drivers.devices.appliance.basicappliance.BasicAppliance",
        "basiccage": "clusto.drivers.locations.datacenters.basiccage.BasicCage",
        "basicconsoleserver": "clusto.drivers.devices.consoleservers.basicconsoleserver.BasicConsole
        "basicdatacenter": "clusto.drivers.locations.datacenters.basicdatacenter.BasicDatacenter",
        "basicnetworkswitch": "clusto.drivers.devices.networkswitches.basicnetworkswitch.BasicNetwor
        "basicpowerstrip": "clusto.drivers.devices.powerstrips.basicpowerstrip.BasicPowerStrip",
        "basicrack": "clusto.drivers.locations.racks.basicrack.BasicRack",
        "basicserver": "clusto.drivers.devices.servers.basicserver.BasicServer",
        "basicvirtualserver": "clusto.drivers.devices.servers.basicserver.BasicVirtualServer",
        "basiczone": "clusto.drivers.locations.zones.basiczone.BasicZone",
        "clustometa": "clusto.drivers.base.clustometa.ClustoMeta",
        "device": "clusto.drivers.base.device.Device",
        "entity": "clusto.drivers.base.driver.Driver",
        "exclusive_pool": "clusto.drivers.categories.pool.ExclusivePool",
        "ipmanager": "clusto.drivers.resourcemanagers.ipmanager.IPManager",
        "location": "clusto.drivers.base.location.Location",
        "pool": "clusto.drivers.categories.pool.Pool",
        "resourcemanager": "clusto.drivers.base.resourcemanager.ResourceManager",
        "simpleentitynamemanager": "clusto.drivers.resourcemanagers.simplenamemanager.SimpleEntityNa
        "simplenamemanager": "clusto.drivers.resourcemanagers.simplenamemanager.SimpleNameManager",
        "unique_pool": "clusto.drivers.categories.pool.UniquePool"
    }
    HTTP: 200
    Content-type: application/json
```

clustoapi.server.**get_from_pools**()

One of the main `clusto` operations. Parameters:

- Required: at least one `pool` parameter
- Optional: one or more `driver` parameter to filter out results
- Optional: one or more `type` parameter to filter out results
- Optional: a boolean `children` parameter to search for children recursively (True by default)

Examples:

```
    $ ${get} ${server_url}/from-pools
    "Provide at least one pool to get data from"
```

```
    HTTP: 412
    Content-type: application/json

    $ ${get} -H 'Clusto-Page: notanint' -d 'pool=emptypool' ${server_url}/from-pools
    "invalid literal for int() with base 10: 'notanint'"
    HTTP: 400
    Content-type: application/json

    $ ${get} -d 'pool=emptypool' ${server_url}/from-pools
    []
    HTTP: 200
    Content-type: application/json

    $ ${get} -d 'pool=singlepool' -d 'pool=multipool' ${server_url}/from-pools
    [
        "/basicserver/testserver1"
    ]
    HTTP: 200
    Content-type: application/json

    $ ${get} -H 'Clusto-Mode: expanded' -d 'pool=multipool' ${server_url}/from-pools
    [
        {
            "attrs": [
                {
                    "datatype": "string",
                    "key": "key1",
                    "number": null,
                    "subkey": "subkey1",
                    "value": "value1"
                }
            ],
            "contents": [],
            "driver": "basicserver",
            "ips": [],
            "name": "testserver1",
            "parents": [
                "/pool/singlepool",
                "/pool/multipool"
            ]
        },
        {
            "attrs": [
                {
                    "datatype": "string",
                    "key": "key1",
                    "number": null,
                    "subkey": "subkey2",
                    "value": "value2"
                }
            ],
            "contents": [],
            "driver": "basicserver",
            "ips": [],
            "name": "testserver2",
            "parents": [
                "/pool/multipool"
            ]
```

```
        }
    ]
    HTTP: 200
    Content-type: application/json

    $ ${get} -H 'Clusto-Page: 1'  -H 'Clusto-Per-Page: 1' -d 'pool=multipool' ${server_url}/from-poo
    [
        "/basicserver/testserver1"
    ]
    HTTP: 200
    Content-type: application/json

    $ ${get} -H 'Clusto-Page: 1'  -H 'Clusto-Per-Page: 100' -d 'pool=multipool' ${server_url}/from-p
    [
        "/basicserver/testserver1",
        "/basicserver/testserver2"
    ]
    HTTP: 200
    Content-type: application/json

    $ ${get} -H 'Clusto-Page: 100'  -H 'Clusto-Per-Page: 100' -d 'pool=multipool' ${server_url}/from
    []
    HTTP: 200
    Content-type: application/json

    $ ${get} -H 'Clusto-Minify: True' -d 'pool=multipool' ${server_url}/from-pools
    ["/basicserver/testserver1", "/basicserver/testserver2"]
    HTTP: 200
    Content-type: application/json
```

clustoapi.server.**main**()
Main entry point for the clusto-apiserver console program

clustoapi.server.**meta**()
This call just returns a mapping of all currently installed applications.

```
    $ ${get} ${server_url}/__meta__
    ...
    HTTP: 200
    Content-type: application/json
```

clustoapi.server.**options**(***kwargs*)
Defined from w3.org:

> "The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval."

The clusto-apiserver team plans to roll this out to individual resources once it has been used a proper amount, but for now we will return OPTIONS with the minimum amount of required headers (and an empty content) no matter what resource is requested.

```
    $ ${head} -X OPTIONS ${server_url}/
    HTTP/1.0 204 No Content
    ...
    Content-Length: 0

    $ ${head} -X OPTIONS ${server_url}/return/headers/no/matter/where
```

```
    HTTP/1.0 204 No Content
    ...
    Content-Length: 0
```

The same applies to any mounted application:

```
    $ ${head} -X OPTIONS ${server_url}/entity/
    HTTP/1.0 204 No Content
    ...
    Content-Length: 0

    $ ${head} -X OPTIONS ${server_url}/attribute/who/knows/where
    HTTP/1.0 204 No Content
    ...
    Content-Length: 0
```

clustoapi.server.**version**()
This shows the current version running, example

```
    $ ${get} ${server_url}/__version__
    "${server_version}"
    HTTP: 200
    Content-type: application/json
```

If you make a HEAD request to the / endpoint, the response is also the version string, as that's less heavy to build than the regular / page:

```
    $ ${head} ${server_url}/
    HTTP/1.0 200 OK
    ...
```

4.1.2 *clustoapi.util*: Miscellaneous util module

clustoapi.util.**dumps**(*obj*, *code=200*, *headers={}*)
Dumps a given object as a JSON string in an HTTP Response object. Will circumvent pretty-printing if Clusto-Minify header is True.

clustoapi.util.**get**(*name*, *driver=None*)
Tries to fetch a clusto object from a given name, optionally validating the driver given. Returns:

- HTTP Error 404 if the object could not be found
- HTTP Error 409 if the object does not match the expected driver
- Clusto object otherwise

clustoapi.util.**page**(*ents*, *current=1*, *per=50*)
Takes a list of entities and drops all from a list but the current page. Returns a tuple that has the entities and also a page total so it may be returned to the client.

clustoapi.util.**show**(*obj*, *mode=''*)
Will return the expanded or compact representation of a given object

clustoapi.util.**typecast**(*value*, *datatype*, *mask='%Y-%m-%dT%H:%M:%S.%f'*)
Takes a string and a valid clusto datatype and attempts to cast the value to the specified datatype. Will error out if a ValueError is incurred or a relation does not exist. Will aslo take a strptime format as `mask` because typcasting datetimes is hard.

clustoapi.util.**unclusto**(*obj*)
Convert an object to a representation that can be safely serialized into JSON.

4.2 Core applications

4.2.1 *clustoapi.apps.attribute*: Attribute Application

The `attribute` application handles all attribute specific operations like querying, adding, deleting and updating attributes.

`clustoapi.apps.attribute.`**`add_attr`**(*name*, ***kwargs*)

Add an attribute to this object.

- Requires parameters `name`, `key`, and `value`
- Optional parameters are `subkey`, `number`, and `datatype`
- Additionally, `mask` can be provided for a datetime attribute.
- These parameters can be either be passed with a querystring
- or a json body. If json is supplied, multiple attributes may be
- added at the same time.

Example:

```
$ ${post} -d 'name=addattrserver' ${server_url}/entity/basicserver
[
    "/basicserver/addattrserver"
]
HTTP: 201
Content-type: application/json
```

```
$ ${post} -d 'key=group' -d 'value=web' ${server_url}/attribute/addattrserver
[
    {
        "datatype": "string",
        "key": "group",
        "number": null,
        "subkey": null,
        "value": "web"
    }
]
HTTP: 201
Content-type: application/json
```

Will:

1. Create an entity called `addattrserver`
2. Add the attribute with `key=group` and `value=web` to it

Example:

```
$ ${post} -d 'key=group' -d 'subkey=owner' -d 'value=web' ${server_url}/attribute/addattrserver
[
    {
        "datatype": "string",
        "key": "group",
        "number": null,
        "subkey": null,
        "value": "web"
    },
```

```
        {
            "datatype": "string",
            "key": "group",
            "number": null,
            "subkey": "owner",
            "value": "web"
        }
    ]
    HTTP: 201
    Content-type: application/json
```

Will add the attribute with key `group` *and* subkey `owner` *and* value `joe` to the previously created entity `addattrserver`

```
    $ ${post} -d 'key=group' -d 'subkey=id' -d 'value=6' -d 'datatype=int' ${server_url}/attribute/a
    [
    ...
        {
            "datatype": "int",
            "key": "group",
            "number": null,
            "subkey": "id",
            "value": 6
        }
    ]
    HTTP: 201
    Content-type: application/json
```

Will add the attribute with key `group` *and* subkey `id` *and* value `1` with the correct datatype to the previously created entity `addattrserver`

```
    $ ${post} -d 'key=inception' -d 'value=/basicserver/addattrserver' -d 'datatype=relation' ${serv
    [
    ...
        {
            "datatype": "relation",
            "key": "inception",
            "number": null,
            "subkey": null,
            "value": "/basicserver/addattrserver"
        }
    ]
    HTTP: 201
    Content-type: application/json
```

Will add the attribute with key `inception` *and* itself as a relation using the `relation` datatype.

```
    $ ${post} -d 'key=birthday' -d 'subkey=jtcunning' -d 'value=1991-07-09T14:46:51.321435' -d 'data
    [
        {
            "datatype": "datetime",
            "key": "birthday",
            "number": null,
            "subkey": "jtcunning",
            "value": "1991-07-09T14:46:51.321435"
        },
    ...
    ]
    HTTP: 201
```

```
    Content-type: application/json
```

Will add the attribute with key `birthday` *and* subkey `jtcunning` with the `datetime` python object as the datatype.

```
    $ ${post} -H 'Content-Type: application/json' -d '${sample_json_attrs}' ${server_url}/attribute/
    [
        ...
        {
            "datatype": "string",
            "key": "group",
            "number": null,
            "subkey": "admin",
            "value": "apache"
        },
        {
            "datatype": "string",
            "key": "group",
            "number": null,
            "subkey": "member",
            "value": "webapp"
        },
        ...
    ]
    HTTP: 201
    Content-type: application/json
```

Will add two attributes in bulk by stating that the content type is `application/json`.

clustoapi.apps.attribute.**attrs**(*name*, *key=None*, *subkey=None*, *number=None*)

Query attributes from this object.

Example:

```
    $ ${post} -d 'name=attrpool1' ${server_url}/entity/pool
    [
        "/pool/attrpool1"
    ]
    HTTP: 201
    Content-type: application/json
```

```
    $ ${get} ${server_url}/attribute/attrpool1
    []
    HTTP: 200
    Content-type: application/json
```

Will show all the attributes from the object `attrpool1`:

```
    $ ${get} -d 'driver=pool' ${server_url}/attribute/attrpool1
    []
    HTTP: 200
    Content-type: application/json
```

Will show all the attributes from the object `attrpool1` **if** the driver for `attrpool1` is `pool`. In the same vein this code:

```
    $ ${get} -d 'driver=basicserver' ${server_url}/attribute/attrpool1
    ...
    HTTP: 409
    ...
```

Should fail, because the `attrpool1` object is of type `pool`, **not** `basicserver`

Example:

```
$ ${get} ${server_url}/attribute/attrpool1/owner
[]
HTTP: 200
Content-type: application/json
```

Will show the attributes for `server1` if their key is `owner`.

clustoapi.apps.attribute.**del_attrs**(*name*, *key*, *subkey=None*, *number=None*)

Deletes an attribute from this object

- Requires HTTP path `key`
- Optional parameters are `subkey`, `value`, and `number`

Examples:

```
$ ${post} -d 'name=deleteserver1' ${server_url}/entity/basicserver
[
    "/basicserver/deleteserver1"
]
HTTP: 201
Content-type: application/json
```

```
$ ${put} -d 'value=joe' ${server_url}/attribute/deleteserver1/group/owner
[
    {
        "datatype": "string",
        "key": "group",
        "number": null,
        "subkey": "owner",
        "value": "joe"
    }
]
HTTP: 200
Content-type: application/json
```

```
$ ${delete} ${server_url}/attribute/deleteserver1/group/owner
[]
HTTP: 200
Content-type: application/json
```

Will create a `basicserver` object called `deleteserver1`, then it will add an attribute (the only attribute so far), then it will delete it.

```
$ ${post} -d 'name=deleteserver2' ${server_url}/entity/basicserver
[
    "/basicserver/deleteserver2"
]
HTTP: 201
Content-type: application/json
```

```
$ ${put} -d 'value=engineering' ${server_url}/attribute/deleteserver2/group
[
    {
        "datatype": "string",
        "key": "group",
        "number": null,
```

```
            "subkey": null,
            "value": "engineering"
        }
    ]
    HTTP: 200
    Content-type: application/json
```

```
    $ ${put} -d 'value=joe' ${server_url}/attribute/deleteserver2/group/owner
    [
        {
            "datatype": "string",
            "key": "group",
            "number": null,
            "subkey": null,
            "value": "engineering"
        },
        {
            "datatype": "string",
            "key": "group",
            "number": null,
            "subkey": "owner",
            "value": "joe"
        }
    ]
    HTTP: 200
    Content-type: application/json
```

```
    $ ${delete} ${server_url}/attribute/deleteserver2/group/owner
    [
        {
            "datatype": "string",
            "key": "group",
            "number": null,
            "subkey": null,
            "value": "engineering"
        }
    ]
    HTTP: 200
    Content-type: application/json
```

This example should add two attributes with the same key, but different subkey, then it will delete only the second value.

clustoapi.apps.attribute.**set_attr**(*name*, ***kwargs*)

Sets an attribute from this object. If the attribute doesn't exist it will be added, if the attribute already exists then it will be updated.

- Requires HTTP parameters `key` and `value`
- Optional parameters are `subkey`', ' ''`number`, and `datatype`
- Additionally, `mask` can be provided for a datetime attribute.

Example:

```
    $ ${post} -d 'name=setattrserver' ${server_url}/entity/basicserver
    [
        "/basicserver/setattrserver"
    ]
```

```
HTTP: 201
Content-type: application/json
```

```
$ ${post} -d 'key=group' -d 'value=web' ${server_url}/attribute/setattrserver
[
    {
        "datatype": "string",
        "key": "group",
        "number": null,
        "subkey": null,
        "value": "web"
    }
]
HTTP: 201
Content-type: application/json
```

```
$ ${put} -d 'value=db' ${server_url}/attribute/setattrserver/group
[
    {
        "datatype": "string",
        "key": "group",
        "number": null,
        "subkey": null,
        "value": "db"
    }
]
HTTP: 200
Content-type: application/json
```

Will:

1.Create the entity `setattrserver`

2.Add the attribute with `key=group` and `value=web`

3.Update the attribute to `value=db`

Example:

```
$ ${post} -d 'name=setattrserver2' ${server_url}/entity/basicserver
[
    "/basicserver/setattrserver2"
]
HTTP: 201
Content-type: application/json
```

```
$ ${put} -d 'value=joe' ${server_url}/attribute/setattrserver2/group/owner
[
    {
        "datatype": "string",
        "key": "group",
        "number": null,
        "subkey": "owner",
        "value": "joe"
    }
]
HTTP: 200
Content-type: application/json
```

```
$ ${put} -d 'value=bob' ${server_url}/attribute/setattrserver2/group/owner
[
    {
        "datatype": "string",
        "key": "group",
        "number": null,
        "subkey": "owner",
        "value": "bob"
    }
]
HTTP: 200
Content-type: application/json
```

Will:

1. Create a new object `setattrserver2` of type `basicserver`
2. Set the attribute with key `group` *and* subkey `owner` with value `joe` to the object `setattrserver1`. Since this is the only attribute so far, this operation works just like `add_attr()`
3. Update the attribute we set above, now the `value` will read `bob`

4.2.2 *clustoapi.apps.entity*: Entity Application

The `entity` application will hold all methods related to entity management in clusto. That is: creation, querying, modification and overall entity manipulation.

`clustoapi.apps.entity.`**`action`**(*driver*, *name*)

Inserts/removes the given device from the request parameters into/from the object

Example:

```
$ ${post} -d 'name=pool1' ${server_url}/entity/pool
[
    "/pool/pool1"
]
HTTP: 201
Content-type: application/json
```

```
$ ${post} -d 'name=server1' ${server_url}/entity/basicserver
[
    "/basicserver/server1"
]
HTTP: 201
Content-type: application/json
```

```
$ ${post} -d 'device=server1' -d 'action=insert' ${server_url}/entity/pool/pool1
{
    "attrs": [],
    "contents": [
        "/basicserver/server1"
    ],
    "driver": "pool",
    "name": "pool1",
    "parents": []
}
HTTP: 200
Content-type: application/json
```

```
$ ${post} -d 'device=server1' -d 'action=remove' ${server_url}/entity/pool/pool1
{
    "attrs": [],
    "contents": [],
    "driver": "pool",
    "name": "pool1",
    "parents": []
}
HTTP: 200
Content-type: application/json
```

Will:

1. Create a pool entity called `pool1`
2. Create a basicserver entity called `server1`
3. Insert the entity `server1` into the entity `pool1`
4. Remove the entity `server1` from the entity `pool1`

Examples:

```
$ ${post} -d 'name=pool2' ${server_url}/entity/pool
[
    "/pool/pool2"
]
HTTP: 201
Content-type: application/json
```

```
$ ${post} -d 'name=server2' -d 'name=server3' ${server_url}/entity/basicserver
[
    "/basicserver/server2",
    "/basicserver/server3"
]
HTTP: 201
Content-type: application/json
```

```
$ ${post} -d 'device=server2' -d 'device=server3' -d 'action=insert' ${server_url}/entity/pool/p
{
    "attrs": [],
    "contents": [
        "/basicserver/server2",
        "/basicserver/server3"
    ],
    "driver": "pool",
    "name": "pool2",
    "parents": []
}
HTTP: 200
Content-type: application/json
```

```
$ ${post} -d 'device=server2' -d 'device=server3' -d 'action=remove' ${server_url}/entity/pool/p
{
    "attrs": [],
    "contents": [],
    "driver": "pool",
    "name": "pool2",
    "parents": []
}
```

```
    HTTP: 200
    Content-type: application/json
```

The above will:

1.Create a pool entity called `pool2`

2.Create two basicserver entities called `server2` and `server3`

3.Insert both basicserver entities into the pool entity

4.Remove both basicserver entities from the pool entity

`clustoapi.apps.entity.`**`create`**(*driver*)

Creates a new object of the given driver.

•Requires HTTP parameters `name`

Example:

```
    $ ${post} -d 'name=createpool1' ${server_url}/entity/pool
    [
        "/pool/createpool1"
    ]
    HTTP: 201
    Content-type: application/json
```

Will create a new `pool1` object with a `pool` driver. If the `pool1` object already exists, the status code returned will be 202, and you will see whatever warnings in the `Warnings` header:

```
    $ ${post_i} -d 'name=createpool1' ${server_url}/entity/pool
    HTTP/1.0 202 Accepted
    ...
    Warnings: Entity(s) /pool/createpool1 already exist(s)...
    [
        "/pool/createpool1"
    ]
```

If you try to create a server of an unknown driver, you should receive a 412 status code back:

```
    $ ${post} -d 'name=createobject' ${server_url}/entity/nondriver
    "Requested driver "nondriver" does not exist"
    HTTP: 412
    Content-type: application/json
```

The following example:

```
    $ ${post_i} -d 'name=createpool1' -d 'name=createpool2' ${server_url}/entity/pool
    HTTP/1.0 202 Accepted
    ...
    Warnings: Entity(s) /pool/createpool1 already exist(s)...
    [
        "/pool/createpool1",
        "/pool/createpool2"
    ]
```

Will attempt to create new objects `createpool1` and `createpool2` with a `pool` driver. As all objects are validated prior to creation, if any of them already exists the return code will be 202 (Accepted) and you will get an extra header `Warnings` with the message.

`clustoapi.apps.entity.`**`delete`**(*driver*, *name*)

Deletes an object if it matches the given driver

•Requires HTTP parameters `name`

Examples:

```
$ ${post} -d 'name=servercreated' ${server_url}/entity/basicserver
[
    "/basicserver/servercreated"
]
HTTP: 201
Content-type: application/json
```

```
$ ${delete} ${server_url}/entity/nondriver/servercreated
"Requested driver "nondriver" does not exist"
HTTP: 412
Content-type: application/json
```

```
$ ${delete} ${server_url}/entity/basicserver/servercreated
HTTP: 204
Content-type:
```

```
$ ${delete} ${server_url}/entity/basicserver/servercreated
HTTP: 404
Content-type: None
```

Will create a new `servercreated` object with a `basicserver` driver. Then it will proceed to delete it. If the operation succeeded, it will return a 200, if the object doesn't exist, it will return a 404.

`clustoapi.apps.entity.`**`list`**(*driver=None*)

Returns all entities, or (optionally) all entities of the given driver

Example:

```
$ ${get} ${server_url}/entity/
[
    ...
]
HTTP: 200
Content-type: application/json
```

Will list all entities

Example:

```
$ ${get} ${server_url}/entity/clustometa
[
    "/clustometa/clustometa"
]
HTTP: 200
Content-type: application/json
```

Will list all entities that match the driver `clustometa`

The following example should fail because there is no driver `nondriver`:

```
$ ${get} ${server_url}/entity/nondriver
"The requested driver "nondriver" does not exist"
HTTP: 412
Content-type: application/json
```

`clustoapi.apps.entity.`**`show`**(*driver*, *name*)

Returns a json representation of the given object

Example:

```
$ ${post} -d 'name=showpool' ${server_url}/entity/pool
[
    "/pool/showpool"
]
HTTP: 201
Content-type: application/json
```

```
$ ${get} ${server_url}/entity/pool/showpool
{
    "attrs": [],
    "contents": [],
    "driver": "pool",
    "name": "showpool",
    "parents": []
}
HTTP: 200
Content-type: application/json
```

Will return a JSON representation of the previously created `showpool`.

```
$ ${get} ${server_url}/entity/basicserver/showpool
"The driver for object "showpool" is not "basicserver""
HTTP: 409
Content-type: application/json
```

Will yield a 409 (Conflict) because the object `showpool` is not a `basicserver` object.

4.2.3 *clustoapi.apps.resourcemanager*: Resource Manager Application

The `resourcemanager` application will hold all methods related to resource management in clusto. Pretty much allocating and deallocating resources.

`clustoapi.apps.resourcemanager.`**`allocate`**(*driver*, *manager*)

This allocates a new *resource* to a given *thing*. Said thing can be either a *driver* (and the result will be a newly created object subclasses from this driver) or an *object*, and the resource manager will allocate (bind) a resource to it.

Examples:

```
$ ${post} -d 'name=allocator' ${server_url}/resourcemanager/simpleentitynamemanager
{
    "attrs": [
        ...
    ],
    "contents": [],
    "count": 0,
    "driver": "simpleentitynamemanager",
    "name": "allocator",
    "parents": []
}
HTTP: 201
Content-type: application/json

$ ${post} -d 'driver=basicserver' ${server_url}/resourcemanager/simpleentitynamemanager/allocato
"/basicserver/01"
HTTP: 201
Content-type: application/json
```

Will request a new name from the object `allocator` (which is an object of `SimpleEntityManager` that we just created with default values) and then it will create a new `BasicServer` object.

```
$ ${post} -d 'driver=basicserver' -d 'resource=99' ${server_url}/resourcemanager/simpleentitynam
"/basicserver/99"
HTTP: 201
Content-type: application/json
```

Will create a new `BasicServer` object from the `testnames` resource manager with the specific name of `s99`.

clustoapi.apps.resourcemanager.**create**(*driver*)

This differs from the standard way of creating entities is that resource managers can have a number of extra parameters added to them that not necessarily match any of the other entities. These parameters are defined by each resource manager driver and are pretty much arbitrary. Seems like a good idea to separate these crucial differences.

Examples:

```
$ ${post} -d 'name=nameman1' ${server_url}/resourcemanager/simplenamemanager
{
    "attrs": [
        ...
    ],
    "contents": [],
    "count": 0,
    "driver": "simplenamemanager",
    "name": "nameman1",
    "parents": []
}
HTTP: 201
Content-type: application/json
```

Will create a `SimpleNameManager` resource manager named `namemgr1` with all default values set.

```
$ ${post} -d 'name=ipman1' -d 'gateway=192.168.1.1' -d 'netmask=255.255.255.0' -d 'baseip=192.16
{
    "attrs": [
        {
            "datatype": "string",
            "key": "baseip",
            "number": null,
            "subkey": "property",
            "value": "192.168.1.10"
        },
        {
            "datatype": "string",
            "key": "gateway",
            "number": null,
            "subkey": "property",
            "value": "192.168.1.1"
        },
        {
            "datatype": "string",
            "key": "netmask",
            "number": null,
            "subkey": "property",
            "value": "255.255.255.0"
```

```
            }
        ],
        "contents": [],
        "count": 0,
        "driver": "ipmanager",
        "name": "ipman1",
        "parents": []
    }
    HTTP: 201
    Content-type: application/json
```

Will create a `IPManager` resource manager named `ipman1` with some additional arguments such as `netmask`, `gateway` and `baseip`

clustoapi.apps.resourcemanager.**deallocate**(*driver*, *manager*)

Resource managers should allow you to deallocate *things* just the same as allocating *things*.

Examples:

```
    $ ${post} -d 'name=ipman2' -d 'gateway=192.168.1.1' -d 'netmask=255.255.255.0' -d 'baseip=192.16
    {
        "attrs": [
            {
                "datatype": "string",
                "key": "baseip",
                "number": null,
                "subkey": "property",
                "value": "192.168.1.10"
            },
            {
                "datatype": "string",
                "key": "gateway",
                "number": null,
                "subkey": "property",
                "value": "192.168.1.1"
            },
            {
                "datatype": "string",
                "key": "netmask",
                "number": null,
                "subkey": "property",
                "value": "255.255.255.0"
            }
        ],
        "contents": [],
        "count": 0,
        "driver": "ipmanager",
        "name": "ipman2",
        "parents": []
    }
    HTTP: 201
    Content-type: application/json

    $ ${post} -d 'name=names2' -d 'basename=a' ${server_url}/resourcemanager/simpleentitynamemanager
    {
        "attrs": [
            ...
        ],
        "contents": [],
```

```
        "count": 0,
        "driver": "simpleentitynamemanager",
        "name": "names2",
        "parents": []
    }
    HTTP: 201
    Content-type: application/json

    $ ${post} -d 'driver=basicserver' ${server_url}/resourcemanager/simpleentitynamemanager/names2
    "/basicserver/a01"
    HTTP: 201
    Content-type: application/json

    $ ${post} -d 'object=a01' ${server_url}/resourcemanager/ipmanager/ipman2
    {
        "datatype": "int",
        "key": "ip",
        "number": 0,
        "subkey": null,
        "value": 1084752130
    }
    HTTP: 201
    Content-type: application/json

    $ ${delete} -d 'object=a01' ${server_url}/resourcemanager/ipmanager/ipman2
    HTTP: 204
    Content-type:
```

clustoapi.apps.resourcemanager.**list**(*driver=None*)

Lists all resource managers found in the clusto database. Optionally you can list all resource managers that match the given `driver`

Examples:

```
    $ ${post} -d 'name=zmanager' ${server_url}/resourcemanager/simplenamemanager
    {
        "attrs": [
            ...
        ],
        "contents": [],
        "count": 0,
        "driver": "simplenamemanager",
        "name": "zmanager",
        "parents": []
    }
    HTTP: 201
    Content-type: application/json
```

The above will create a simple name manager called "zmanager"

```
    $ ${get} ${server_url}/resourcemanager/
    [
        "/simpleentitynamemanager/testnames",
        "/simplenamemanager/zmanager"
    ]
    HTTP: 200
    Content-type: application/json
```

The above will list all resource managers in clusto, which should have "zmanager"

```
$ ${get} ${server_url}/resourcemanager/simpleentitynamemanager
[
    "/simpleentitynamemanager/testnames"
]
HTTP: 200
Content-type: application/json
```

Will list all resource managers of driver `SimpleNameManager`

```
$ ${get} ${server_url}/resourcemanager/notadriver
"Not a valid driver "notadriver" (driver name notadriver doesn't exist.)"
HTTP: 404
Content-type: application/json
```

Will return a `404` error because that resource manager driver doesn't exist

clustoapi.apps.resourcemanager.**show**(*driver*, *manager*)

Shows the details of the given resource manager, if it is a resource manager

Examples:

```
$ ${post} -d 'name=nameman2' ${server_url}/resourcemanager/simplenamemanager
{
    "attrs": [
        ...
    ],
    "contents": [],
    "count": 0,
    "driver": "simplenamemanager",
    "name": "nameman2",
    "parents": []
}
HTTP: 201
Content-type: application/json

$ ${get} ${server_url}/resourcemanager/simplenamemanager/nameman1
{
    "attrs": [
        ...
    ],
    "contents": [],
    "count": 0,
    "driver": "simplenamemanager",
    "name": "nameman1",
    "parents": []
}
HTTP: 200
Content-type: application/json
```

Will create the `nameman2` resource manager, then show its details. In this case both operations yield the same data.

```
$ ${get} ${server_url}/resourcemanager/simpleentitynamemanager/nonames
"Object "nonames" not found (nonames does not exist.)"
HTTP: 404
Content-type: application/json
```

Will return a `404` error since the resource manager wasn't found

```
$ ${get} ${server_url}/resourcemanager/nomanager/testnames
"The driver "nomanager" is not a valid driver"
HTTP: 412
Content-type: application/json
```

Will return a `412` because the driver `nomanager` doesn't exist

```
$ ${get} ${server_url}/resourcemanager/basicserver/testserver1
"The object "testserver1" is not a resource manager"
HTTP: 409
Content-type: application/json
```

Will return a `412` instead because even though the driver `basicserver` exists, it is not a resource manager driver

CHAPTER 5

Developer documentation

5.1 Releasing a new version

Once you are comfortable with the changes to be released, there is a few things you have to do to consider this software "released".

It is worth mentioning that I will very probably script all (or almost all) of this, once I have the time / decide how to do it.

5.1.1 Pre-requisites

- You need a github account with permissions for pushing new tags/releases
- You need to be added to the PyPi project so you can submit new releases
- Your `~/.pypirc` is configured with your PyPi username and password
- You need your GPG key signed by one of the other maintainers

5.1.2 Cutting the release

Usually involves a few steps (for simplicity, using version 0.0.1 to illustrate the examples)

1. *Updating the changelog*: You create a new page in `docs/changelog` named `docs/changelog/v0.0.1.rst`
2. *Re-linking current changelog to new changelog*: You re-point the symlink from `docs/changelog/current.rst` to `docs/changelog/v0.0.1.rst`
3. *Commit your changelog change*
4. *Update the version*: Need to bump the version in `clustoapi/__init__.py` to reflect the new 0.0.1 version
5. *Commit your version bump change*
6. *Tag the release*: You need to `git tag -s v0.0.1` at this point. Pay attention to the `-s` switch, this implies the tag will be signed with your GPG key (optionally, use `-u` to specify which key to use if you don't want to use the default)

5.1.3 Publishing the release

This entails publishing the release on github. Once you publish the release on github, a tweet will be posted shortly (< 1 hour after) to the @clustodotorg account.

1. *Push your changes up to the repo*: You need to both `git push` and `git push --tags` for this operation, your changes should be visible now, and the new tag should appear in https://github.com/clusto/clusto-apiserver/tags
2. *Edit the tag so it turns into a release in github*: Draft a new release https://github.com/clusto/clusto-apiserver/releases/new and pick the tag you recently created. The title of the release should be "Version 0.0.1" (to continue with the sample version) and the body should be the contents of the `docs/changelog/v0.0.1.rst` file. Keep in mind that the format is Markdown while the documentation is ReStructured Text, so adjust as necessary (mainly titles change i.e. replace ^ for −)

At this point the release should appear in the github releases page https://github.com/clusto/clusto-apiserver/releases/

5.1.4 Uploading to PyPi

Finally, publishing the module / source code to PyPi should be a single step:

- *Sign and upload the new version tarball*: Basically run (from the project's root dir):

```
python setup.py sdist upload --sign [--identity <GPG identity>]
```

What this does is:

1. Creates a tarball under the `dist/` directory (the `sdist` command)
2. Using the `--sign` flag signs the tarball using your GPG key (depending on your setup, you may get prompted for your GPG password). Optionally, you can use the `--identity` argument if you aren't using your default identity to sign
3. Uploads the recently created and signed tarball to PyPi (the `upload` command)

At this point, the new version should be visible in https://pypi.python.org/pypi/clusto-apiserver

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

C