lxdapi Documentation

Release 0.0.0

James Pic

Sep 01, 2017

Contents

This library includes basic helpers encapsulating boring business logic code for interacting with the lxd daemon. It is an experimental alternative to the pylxd approach. The purpose of this library is **not** to be a full blown framework for interacting with lxd completely abstracting HTTP transactions in an hermetic way, pylxd already does a good job at that.

Contents:

CHAPTER 1

HTTP API transactions

API

HTTP API classes for interfacing with the LXD API.

This module leverages the composition pattern, providing 3 main classes:

- *API*: the entry point for dealing with the HTTP API,
- *APIResult*: returned by requests made with *API*, it represents an HTTP transaction.
- *APIException*: raised when the server responded with HTTP 400.

The *API* object wraps around requests, note that its constructor takes a debug keyword argument to enable printouts of HTTP transactions, that can also be enabled with the `DEBUG` environment variable.

classmethod `API.`**`factory`**(*endpoint=None*, *default_version=None*, ***kwargs*)

Instanciate an *API* with the right endpoint and session.

Example:

```
# Connect to a local socket
api = lxd.API.factory()

# Or, connect to a remote server (untested so far)
api = lxd.API.factory(base_url='http://example.com:12345')
```

class `lxdapi.api.`**`API`**(*session*, *endpoint*, *default_version=None*, *debug=False*)

Main entry point to interact with the HTTP API.

Once you have an instance of *API*, which is easier to make with *factory()* than with the constructor, use the *get()*, *post()*, *delete()*, *put()* or *request()* directly. Since *request()* is used by the other methods, refer to to *request()* for details.

Example:

```
api = lxd.API.factory()
api.post('images', json=data_dict).wait()
```

API.**request**(*method*, *url*, **args*, ***kwargs*)
Execute an HTTP request, return an *APIResult*.

Note that it calls *APIResult.validate()*, which may raise *APIException* or *APINotFoundException*.

If debug is True, then this will dump HTTP request and response data.

Extra args and kwargs are passed to requests.Session.request().

If the server responds with HTTP/400 then an APIException will be raised. It has the APIResult as result attribute and has the error message from the server as error. It looks like this:

```
APIException: GET http+unix://%2Fvar%2Flib%2Flxd%2Funix.socket/1.0/operations/
↪1c34b923-57c8-4fce-b349-e4a1c61b8803/wait?timeout=30 Error calling 'lxd forkstart␣
↪pylxd-test-container /var/lib/lxd/containers /var/log/lxd/pylxd-test-container/lxc.
↪conf': err='exit status 1'
```

API.**delete**(*url*, **args*, ***kwargs*)
Calls *request()* with method=DELETE.

API.**get**(*url*, **args*, ***kwargs*)
Calls *request()* with method=GET.

API.**post**(*url*, **args*, ***kwargs*)
Calls *request()* with method=POST.

API.**put**(*url*, **args*, ***kwargs*)
Calls *request()* with method=PUT.

Debugging

The purpose of this lib is to help the user to have feedback from HTTP transactions. If anything fails, it should help a lot to re-run the program with the DEBUG env var, and such output will be printed out for every transaction:

```
PUT http+unix://%2Fvar%2Flib%2Flxd%2Funix.socket/1.0/containers/pylxd-test-container/
↪state
{
        "action": "stop",
        "timeout": 30
} HTTP/202
{
        "status": "Operation created",
        "status_code": 100,
        "operation": "/1.0/operations/70eb42bf-5e79-42de-8230-2162e9e8b612",
        "type": "async",
        "metadata": {
                "status": "Running",
                "err": "",
                "status_code": 103,
                "created_at": "2016-06-23T15:05:45.435413676+02:00",
                "updated_at": "2016-06-23T15:05:45.435413676+02:00",
                "class": "task",
                "may_cancel": false,
                "id": "70eb42bf-5e79-42de-8230-2162e9e8b612",
```

```
                "resources": {
                        "containers": [
                                "/1.0/containers/pylxd-test-container"
                        ]
                },
                "metadata": null
        }
}
```

APIResult

class lxdapi.api.**APIResult**(*api*, *response*)

Represent an HTTP transaction, return by API calls using *API*.

api

API object which returned this result.

data

JSON data from the response.

request

Request object from the requests library.

response

Response object from the requests library.

request_summary()

Return a string with the request method, url, and data.

response_summary()

Return a string with the response status code and data.

validate()

Recursive status code checker for this result's response.

If the response's status code is 404 then raise *APINotFoundException*.

If the response's status code is anything superior or equal to 400 then raise *APIException*

It'll use *validate_metadata()* to check metadata.

validate_metadata(*data*)

Recursive function to check status code for a metadata dict.

Each metadata may contain more metadata. Each metadata may have a status_code, if it's superior or equal to 400 then an *APIException* is raised.

This is used by *validate()* which should be used in general instead of this method.

wait(*timeout=None*)

Execute the wait API call for the operation in this result.

Exceptions

APIException

class lxdapi.api.**APIException**(*result*)
Raised by *API* on HTTP/400 responses.

It will try to find the error message in the HTTP response and use it if it find it, otherwise will use the response data as message.

result
The *APIResult* this was raised for.

APINotFoundException

class lxdapi.api.**APINotFoundException**(*result*)
Child of APIException for 404.

CHAPTER 2

Shortcut functions

Idempotent functions and shortcuts.

Functions here have the following similarities:

- take a *API* as first argument,
- return True if something has changed, False otherwise,
- except `_get()` functions such as `container_get()` which return *APIResult* for an *lxdapi.api.API.get()* or False.

`lxdapi.shortcuts.`**`container_absent`**(*api*, *container*)

Ensure a container is absent.

Container is an *APIResult* for the container, to be able to compare the configuration with.

It is expected that the user manages the HTTP transactions, here's an example usage:

```
container_absent(api, container_get('yourcontainer'))
```

`lxdapi.shortcuts.`**`container_apply_config`**(*api*, *container*, *config*)

Apply a configuration on a container.

Container is an:class:'lxdapi.api.APIResult'for the container, to be able to compare the configuration with.

Config is the dict to pass as JSON to the HTTP API.

Example usage:

```
container_apply_config(api, container_get('yourcontainer'))
```

`lxdapi.shortcuts.`**`container_apply_status`**(*api*, *container*, *status*)

Apply an LXD status to a container.

Container is an:class:'lxdapi.api.APIResult'for the container, to be able to compare the status with.

Status is a string, choices are: Running, Stopped, Frozen.

Example usage:

```
container_apply_status(api, container_get('yourcontainer'), 'Running')
```

lxdapi.shortcuts.**container_get**(*api*, *name*)
Return the:class:‘lxdapi.api.APIResult‘for a container or False.

lxdapi.shortcuts.**image_absent**(*api*, *fingerprint*)
Return False if the image is absent, otherwise delete it and return True.

lxdapi.shortcuts.**image_alias_present**(*api*, *name*, *target*, *description=None*)
Ensure an image has an alias.

lxdapi.shortcuts.**image_get**(*api*, *fingerprint*)
Return the APIResult for a fingerprint or False.

lxdapi.shortcuts.**image_get_fingerprint**(*path*)
Return the fingerprint for an image.

lxdapi.shortcuts.**image_present**(*api*, *path*, *fingerprint=None*)
Ensure an image is present.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index