pylxd Documentation

Release

Canonical Ltd

Jul 28, 2017

Contents

1 Installation **3**

2 Getting started **5**
2.1 Client . . . 5

3 Client Authentication **7**
3.1 Generate a certificate . . . 7
3.2 Authenticate a new keypair . . . 7

4 Events **9**

5 Certificates **11**
5.1 Manager methods . . . 11
5.2 Certificate attributes . . . 11

6 Containers **13**
6.1 Manager methods . . . 13
6.2 Container attributes . . . 13
6.3 Container methods . . . 14
6.4 Examples . . . 14
6.5 Container Snapshots . . . 15
6.6 Container files . . . 15

7 Images **17**
7.1 Manager methods . . . 17
7.2 Image attributes . . . 17
7.3 Image methods . . . 18
7.4 Examples . . . 18

8 Networks **19**
8.1 Manager methods . . . 19
8.2 Network attributes . . . 19

9 Profiles **21**
9.1 Manager methods . . . 21
9.2 Profile attributes . . . 21
9.3 Profile methods . . . 21
9.4 Examples . . . 21

10 Contributing **23**
10.1 Code standards . . . 23
10.2 Testing . . . 23

11 API documentation **25**
11.1 Client . . . 25
11.2 Certificate . . . 26
11.3 Container . . . 26
11.4 Image . . . 26
11.5 Network . . . 26
11.6 Operation . . . 26
11.7 Profile . . . 26

12 Indices and tables **27**

Contents:

CHAPTER 1

Installation

If you're running on Ubuntu Xenial or greater:

```
sudo apt-get install python-pylxd lxd
```

Otherwise you can track LXD development on other Ubuntu releases:

```
sudo add-apt-repository ppa:ubuntu-lxc/lxd-git-master && sudo apt-get update
sudo apt-get install lxd
```

Or install pylxd using pip:

```
pip install pylxd
```

CHAPTER 2

Getting started

Client

Once you have *installed*, you're ready to instantiate an API client to start interacting with the LXD daemon on localhost:

```
>>> from pylxd import Client
>>> client = Client()
```

If your LXD instance is listening on HTTPS, you can pass a two part tuple of (cert, key) as the *cert* argument.

```
>>> from pylxd import Client
>>> client = Client(
...     endpoint='http://10.0.0.1:8443',
...     cert=('/path/to/client.crt', '/path/to/client.key'))
```

Note: in the case where the certificate is self signed (LXD default), you may need to pass *verify=False*.

Querying LXD

LXD exposes a number of objects via its REST API that are used to orchestrate containers. Those objects are all accessed via manager attributes on the client itself. This includes *certificates*, *containers*, *images*, *networks*, *operations*, and *profiles*. Each manager has methods for querying the LXD instance. For example, to get all containers in a LXD instance

For specific manager methods, please see the documentation for each object.

pylxd Objects

Each LXD object has an analagous pylxd object. Returning to the previous *client.containers.all* example, a *Container* object is manipulated as such:

Each pylxd object has a lifecycle which includes support for transactional changes. This lifecycle includes the following methods and attributes:

- *sync()* - Synchronize the object with the server. This method is called implicitly when accessing attributes that have not yet been populated, but may also be called explicitly. Why would attributes not yet be populated? When retrieving objects via *all*, LXD's API does not return a full representation.
- *dirty* - After setting attributes on the object, the object is considered "dirty".
- *rollback()* - Discard all local changes to the object, opting for a representation taken from the server.
- *save()* - Save the object, writing changes to the server.

Returning again to the *Container* example

A note about asynchronous operations

Some changes to LXD will return immediately, but actually occur in the background after the http response returns. All operations that happen this way will also take an optional *wait* parameter that, when *True*, will not return until the operation is completed.

CHAPTER 3

Client Authentication

When using LXD over https, LXD uses an asymmetric keypair for authentication. The keypairs are added to the authentication database after entering the LXD instance's "trust password".

Generate a certificate

To generate a keypair, you should use the *openssl* command. As an example:

```
openssl req -newkey rsa:2048 -nodes -keyout lxd.key -out lxd.csr
openssl x509 -signkey lxd.key -in lxd.csr -req -days 365 -out lxd.crt
```

For more detail on the commands, or to customize the keys, please see the documentation for the *openssl* command.

Authenticate a new keypair

If a client is created using this keypair, it would originally be "untrusted", essentially meaning that the authentication has not yet occurred.

```
>>> from pylxd import Client
>>> client = Client(
...     endpoint='http://10.0.0.1:8443',
...     cert=('lxd.crt', 'lxd.key'))
>>> client.trusted
False
```

In order to authenticate the client, pass the lxd instance's trust password to *Client.authenticate*

```
>>> client.authenticate('a-secret-trust-password')
>>> client.trusted
>>> True
```

CHAPTER 4

Events

LXD provides an */events* endpoint that is upgraded to a streaming websocket for getting LXD events in real-time. The `Client`'s *events* method will return a websocket client that can interact with the web socket messages.

```
>>> ws_client = client.events()
>>> ws_client.connect()
>>> ws_client.run()
```

A default client class is provided, which will block indefinitely, and collect all json messages in a *messages* attribute. An optional *websocket_client* parameter can be provided when more functionality is needed. The *ws4py* library is used to establish the connection; please see the *ws4py* documentation for more information.

CHAPTER 5

Certificates

Certificates are used to manage authentications in LXD. Certificates are not editable. They may only be created or deleted. None of the certificate operations in LXD are asynchronous.

Manager methods

Certificates can be queried through the following client manager methods:

- *all()* - Retrieve all certificates.
- *get()* - Get a specifit certificate, by its fingerprint.
- *create()* - Create a new certificate. This method requires a first argument that is the LXD trust password, and the cert data, in binary format.

Certificate attributes

Certificates have the following attributes:

- *fingerprint* - The fingerprint of the certificate. Certificates are keyed off this attribute.
- *certificate* - The certificate itself, in PEM format.
- *type* - The certificate type (currently only "client")

CHAPTER 6

Containers

Container objects are the core of LXD. Containers can be created, updated, and deleted. Most of the methods for operating on the container itself are asynchronous, but many of the methods for getting information about the container are synchronous.

Manager methods

Containers can be queried through the following client manager methods:

- *all()* - Retrieve all containers.
- *get()* - Get a specific container, by its name.
- *create(config, wait=False)* - Create a new container. This method requires the container config as the first parameter. The config itself is beyond the scope of this documentation. Please refer to the LXD documentation for more information. This method will also return immediately, unless *wait* is *True*.

Container attributes

For more information about the specifics of these attributes, please see the LXD documentation.

- *architecture* - The container architecture.
- *config* - The container config
- *created_at* - The time the container was created
- *devices* - The devices for the container
- *ephemeral* - Whether the container is ephemeral
- *expanded_config* - An expanded version of the config
- *expanded_devices* - An expanded version of devices

- *name* - The name of the container. This attribute serves as the primary identifier of a container.
- *profiles* - A list of profiles applied to the container
- *status* - A string representing the status of the container
- *status_code* - A LXD status code of the container
- *stateful* - Whether the container is stateful

Container methods

- *rename* - Rename a container. Because *name* is the key, it cannot be renamed by simply changing the name of the container as an attribute and calling *save*. The new name is the first argument and, as the method is asynchronous, you may pass *wait=True* as well.
- *save* - Update container's configuration
- *state* - Get the expanded state of the container.
- *start* - Start the container
- *stop* - Stop the container
- *restart* - Restart the container
- *freeze* - Suspend the container
- *unfreeze* - Resume the container
- *execute* - Execute a command on the container. The first argument is a list, in the form of *subprocess.Popen* with each item of the command as a separate item in the list. Returns a two part tuple of *(stdout, stderr)*. This method will block while the command is executed.
- *migrate* - Migrate the container. The first argument is a client connection to the destination server. This call is asynchronous, so *wait=True* is optional. The container on the new client is returned.

Examples

If you'd only like to fetch a single container by its name...

```
>>> client.containers.get('my-container')
<container.Container at 0x7f95d8af72b0>
```

If you're looking to operate on all containers of a LXD instance, you can get a list of all LXD containers with *all*.

```
>>> client.containers.all()
[<container.Container at 0x7f95d8af72b0>,]
```

In order to create a new `Container`, a container config dictionary is needed, containing a name and the source. A create operation is asynchronous, so the operation will take some time. If you'd like to wait for the container to be created before the command returns, you'll pass *wait=True* as well.

```
>>> config = {'name': 'my-container', 'source': {'type': 'none'}}
>>> container = client.containers.create(config, wait=False)
>>> container
<container.Container at 0x7f95d8af72b0>
```

If you were to use an actual image source, you would be able to operate on the container, starting, stopping, snapshotting, and deleting the container.

```
>>> config = {'name': 'my-container', 'source': {'type': 'image', 'alias': 'ubuntu/
↪trusty'}}
>>> container = client.containers.create(config, wait=True)
>>> container.start()
>>> container.freeze()
>>> container.delete()
```

To modify container's configuration method *save* should be called after `Container` attributes changes.

```
>>> container = client.containers.get('my-container')
>>> container.ephemeral = False
>>> container.devices = { 'root': { 'path': '/', 'type': 'disk', 'size': '7GB'} }
>>> container.save
```

Container Snapshots

Each container carries its own manager for managing `Snapshot` functionality. It has *get*, *all*, and *create* functionality.

Snapshots are keyed by their name (and only their name, in pylxd; LXD keys them by <container-name>/<snapshot-name>, but the manager allows us to use our own namespacing).

To create a new snapshot, use *create* with a *name* argument. If you want to capture the contents of RAM in the snapshot, you can use *stateful=True*. **Note: Your LXD requires a relatively recent version of CRIU for this.**

Container files

Containers also have a *files* manager for getting and putting files on the container.

CHAPTER 7

Images

Image objects are the base for which containers are built. Many of the methods of images are asynchronous, as they required reading and writing large files.

Manager methods

Images can be queried through the following client manager methods:

- *all()* - Retrieve all images.
- *get()* - Get a specific image, by its fingerprint.

And create through the following methods, theres also a copy method on an image:

- *create(data, public=False, wait=False)* - Create a new image. The first argument is the binary data of the image itself. If the image is public, set *public* to *True*.
- *create_from_simplestreams(server, alias, public=False, auto_update=False, wait=False)* - Create an image from simplestreams.
- *create_from_url(url, public=False, auto_update=False, wait=False)* - Create an image from a url.

Image attributes

For more information about the specifics of these attributes, please see the LXD documentation.

- *aliases* - A list of aliases for this image
- *auto_update* - Whether the image should auto-update
- *architecture* - The target architecture for the image
- *cached* - Whether the image is cached
- *created_at* - The date and time the image was created

- *expires_at* - The date and time the image expires
- *filename* - The name of the image file
- *fingerprint* - The image fingerprint, a sha2 hash of the image data itself. This unique key identifies the image.
- *last_used_at* - The last time the image was used
- *properties* - The configuration of image itself
- *public* - Whether the image is public or not
- *size* - The size of the image
- *uploaded_at* - The date and time the image was uploaded
- *update_source* - A dict of update informations

Image methods

- *export* - Export the image. Returns a file object with the contents of the image. *Note: Prior to pylxd 2.1.1, this method returned a bytestring with data; as it was not unbuffered, the API was severely limited.*
- *add_alias* - Add an alias to the image.
- *delete_alias* - Remove an alias.
- *copy* - Copy the image to another LXD client.

Examples

`Image` operations follow the same protocol from the client's *images* manager (i.e. *get*, *all*, and *create*). Images are keyed on a sha-1 fingerprint of the image itself. To get an image...

```
>>> image = client.images.get(
...     'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855')
>>> image
<image.Image at 0x7f95d8af72b0>
```

Once you have an image, you can operate on it as before:

```
>>> image.public
False
>>> image.public = True
>>> image.update()
```

To create a new Image, you'll open an image file, and pass that to *create*. If the image is to be public, *public=True*. As this is an asynchonous operation, you may also want to *wait=True*.

```
>>> image_data = open('an_image.tar.gz').read()
>>> image = client.images.create(image_data, public=True, wait=True)
>>> image.fingerprint
'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855'
```

CHAPTER 8

Networks

Network objects show the current networks available to lxd. They are read-only via the REST API.

Manager methods

Networks can be queried through the following client manager methods:

- *all()* - Retrieve all networks
- *get()* - Get a specific network, by its name.

Network attributes

- *name* - The name of the network
- *type* - The type of the network
- *used_by* - A list of containers using this network

CHAPTER 9

Profiles

Profile describe configuration options for containers in a re-usable way.

Manager methods

Profiles can be queried through the following client manager methods:

- *all()* - Retrieve all networks
- *get()* - Get a specific network, by its name.
- *create(name, config, devices)* - Create a new profile. The name of the profile is required. *config* and *devices* dictionaries are optional, and the scope of their contents is documented in the LXD documentation.

Profile attributes

- *name* - The name of the network
- *type* - The type of the network
- *used_by* - A list of containers using this network

Profile methods

- *rename* - Rename the profile.

Examples

`Profile` operations follow the same manager-style as Containers and Images. Profiles are keyed on a unique name.

```
>>> profile = client.profiles.get('my-profile')
>>> profile
<profile.Profile at 0x7f95d8af72b0>
```

The profile can then be modified and saved.

```
>>> profile.config = profile.config.update({'security.nesting': 'true'})
>>> profile.update()
```

To create a new profile, use *create* with a name, and optional *config* and *devices* config dictionaries.

```
>>> profile = client.profiles.create(
...     'an-profile', config={'security.nesting': 'true'},
...     devices={'root': {'path': '/', 'size': '10GB', 'type': 'disk'}})
```

CHAPTER 10

Contributing

pyLXD development is done on Github. Pull Requests and Issues should be filed there. We try and respond to PRs and Issues within a few days.

If you would like to contribute large features or have big ideas, it's best to post on to the lxc-users list to discuss your ideas before submitting PRs.

Code standards

pyLXD follows PEP 8 as closely as practical. To check your compliance, use the *pep8* tox target:

```
tox -epep8
```

Testing

pyLXD tries to follow best practices when it comes to testing. PRs are gated by Travis CI and CodeCov. It's best to submit tests with new changes, as your patch is unlikely to be accepted without them.

To run the tests, you can use nose:

```
nosetests pylxd
```

...or, alternatively, you can use *tox* (with the added bonus that it will test python 2.7, python 3, and pypy, as well as run pep8). This is the way that Travis will test, so it's recommended that you run this at least once before submitting a Pull Request.

CHAPTER 11

API documentation

Client

class pylxd.client.**Client**(*endpoint=None*, *version='1.0'*, *cert=None*, *verify=True*, *timeout=None*)

Client class for LXD REST API.

This client wraps all the functionality required to interact with LXD, and is meant to be the sole entry point.

containers

Instance of `Client.Containers`:

images

Instance of `Client.Images`.

operations

Instance of `Client.Operations`.

profiles

Instance of `Client.Profiles`.

api

This attribute provides tree traversal syntax to LXD's REST API for lower-level interaction.

Use the name of the url part as attribute or item of an api object to create another api object appended with the new url part name, ie:

```
>>> api = Client().api
# /
>>> response = api.get()
# Check status code and response
>>> print response.status_code, response.json()
# /containers/test/
>>> print api.containers['test'].get().json()
```

events(*websocket_client=None*)

Get a websocket client for getting events.

/events is a websocket url, and so must be handled differently than most other LXD API endpoints. This method returns a client that can be interacted with like any regular python socket.

An optional *websocket_client* parameter can be specified for implementation-specific handling of events as they occur.

Certificate

Container

Image

Network

Operation

Profile

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Index

A

api (Client attribute), 25

C

Client (class in pylxd.client), 25
containers (Client attribute), 25

E

events() (pylxd.client.Client method), 25

I

images (Client attribute), 25

O

operations (Client attribute), 25

P

profiles (Client attribute), 25