OS Brick Documentation

Release 5.2.5.dev1

Cinder Contributors

Oct 03, 2023

CONTENTS

os-brick is a Python package containing classes that help with volume discovery and removal from a host.

CHAPTER

ONE

INSTALLATION GUIDE

1.1 Installation

At the command line:

```
$ pip install os-brick
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv os-brick
$ pip install os-brick
```

Or, from source:

```
$ git clone https://opendev.org/openstack/os-brick
$ cd os-brick
$ python setup.py install
```

CHAPTER

TWO

USAGE GUIDE

2.1 Tutorial

This tutorial is intended as an introduction to working with **os-brick**.

2.1.1 Prerequisites

Before we start, make sure that you have the **os-brick** distribution *installed*. In the Python shell, the following should run without raising an exception:

```
>>> import os_brick
```

2.1.2 Fetch all of the initiator information from the host

An example of how to collect the initiator information that is needed to export a volume to this host.

```
from os_brick.initiator import connector

# what helper do you want to use to get root access?
root_helper = "sudo"
# The ip address of the host you are running on
my_ip = "192.168.1.1"
# Do you want to support multipath connections?
multipath = True
# Do you want to enforce that multipath daemon is running?
enforce_multipath = False
initiator = connector.get_connector_properties(root_helper, my_ip,
                                               multipath,
                                               enforce_multipath)
```

CHAPTER

THREE

REFERENCE

3.1 API Documentation

The **os-brick** package provides the ability to collect host initiator information as well as discovery volumes and removal of volumes from a host.

3.1.1 os_brick OpenStack Brick library

Sub-modules:

initiator Initiator

Bricks Initiator module.

The initator module contains the capabilities for discovering the initiator information as well as discovering and removing volumes from a host.

Sub-modules:

connector Connector

Brick Connector objects for each supported transport protocol.

The connectors here are responsible for discovering and removing volumes for each of the supported transport protocols.

class os_brick.initiator.connector.**InitiatorConnector**

static factory(*protocol*, *root_helper*, *driver=None*, *use_multipath=False*, *device_scan_attempts=3*, *arch=None*, **args*, ***kwargs*)

Build a Connector object based upon protocol and architecture.

class os_brick.initiator.connector.**ISCSIConnector**(*root_helper: str*, *driver=None*, *execute=None*, *use_multipath: bool = False*, *device_scan_attempts: int = 3*, *transport='default'*, **args*, ***kwargs*)

Connector class to attach/detach iSCSI volumes.

connect_volume(*connection_properties: dict*)

Attach the volume to instance_name.

Parameters **connection_properties** (*dict*) The valid dictionary that describes all of the target volume attributes.

Returns dict

connection_properties for iSCSI must include: target_portal(s) - ip and optional port target_iqn(s) - iSCSI Qualified Name target_lun(s) - LUN id of the volume Note that plural keys may be used when use_multipath=True

disconnect_volume(*connection_properties*, *device_info*, *force=False*, *ignore_errors=False*)
Detach the volume from instance_name.

Parameters

- **connection_properties** (*dict that must include: target_portal(s) - IP and optional port target_iqn(s) - iSCSI Qualified Name target_lun(s) - LUN id of the volume*) The dictionary that describes all of the target volume attributes.
- **device_info** (*dict*) historical difference, but same as connection_props
- **force** (*bool*) Whether to forcefully disconnect even if flush fails.
- **ignore_errors** (*bool*) When force is True, this will decide whether to ignore errors or raise an exception once finished the operation. Default is False.

class os_brick.initiator.connector.**FibreChannelConnector**(*root_helper*, *driver=None*, *execute=None*, *use_multipath=False*, *device_scan_attempts=3*, **args*, ***kwargs*)
Connector class to attach/detach Fibre Channel volumes.

connect_volume(*connection_properties*)
Attach the volume to instance_name.

Parameters **connection_properties** (*dict*) The dictionary that describes all of the target volume attributes.

Returns dict

connection_properties for Fibre Channel must include: target_wwn - World Wide Name target_lun - LUN id of the volume

disconnect_volume(*connection_properties*, *device_info*, *force=False*, *ignore_errors=False*)
Detach the volume from instance_name.

Parameters

- **connection_properties** (*dict*) The dictionary that describes all of the target volume attributes.
- **device_info** (*dict*) historical difference, but same as connection_props

connection_properties for Fibre Channel must include: target_wwn - World Wide Name target_lun - LUN id of the volume

class os_brick.initiator.connector.**LocalConnector**(*root_helper*, *driver=None*, **args*, ***kwargs*)
Connector class to attach/detach File System backed volumes.

connect_volume(*connection_properties*)
Connect to a volume.

Parameters **connection_properties** (*dict*) The dictionary that describes all of the target volume attributes. connection_properties must include:

- device_path - path to the volume to be connected

Returns dict

disconnect_volume(*connection_properties*, *device_info*, *force=False*, *ignore_errors=False*)
Disconnect a volume from the local host.

Parameters

- **connection_properties** (*dict*) The dictionary that describes all of the target volume attributes.
- **device_info** (*dict*) historical difference, but same as connection_props

class os_brick.initiator.connector.**HuaweiStorHyperConnector**(*root_helper*, *driver=None*, **args*, ***kwargs*)
Connector class to attach/detach SDSHypervisor volumes.

connect_volume(*connection_properties*)
Connect to a volume.

Parameters **connection_properties** (*dict*) The dictionary that describes all of the target volume attributes.

Returns dict

disconnect_volume(*connection_properties*, *device_info*, *force=False*, *ignore_errors=False*)
Disconnect a volume from the local host.

Parameters

- **connection_properties** (*dict*) The dictionary that describes all of the target volume attributes.
- **device_info** (*dict*) historical difference, but same as connection_props

exception Exceptions

Exceptions for the Brick library.

class os_brick.exception.**BrickException**(*message=None*, ***kwargs*)
Base Brick Exception

To correctly use this class, inherit from it and define a message property. That message will get printfd with the keyword arguments provided to the constructor.

class os_brick.exception.**NotFound**(*message=None*, ***kwargs*)

class os_brick.exception.**Invalid**(*message=None*, ***kwargs*)

class os_brick.exception.**InvalidParameterValue**(*message=None*, ***kwargs*)

class os_brick.exception.**NoFibreChannelHostsFound**(*message=None*, ***kwargs*)

class os_brick.exception.**NoFibreChannelVolumeDeviceFound**(*message=None*, ***kwargs*)

class os_brick.exception.**VolumeDeviceNotFound**(*message=None*, ***kwargs*)

class os_brick.exception.**ProtocolNotSupported**(*message=None*, ***kwargs*)

CHAPTER

FOUR

CONTRIBUTING

4.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

The os-brick library is maintained by the OpenStack Cinder project. To understand our development process and how you can contribute to it, please look at the Cinder projects general contributors page: http://docs.openstack.org/cinder/latest/contributor/contributing.html