Alternative Cinder Scheduler Classes Documentation

Release 0.1.1

Gorka Eguileor

Jul 13, 2017

Contents

Contents:

CHAPTER 1

Alternative Cinder Scheduler Classes

Alternative Classes such as filters, host managers, etc. for Cinder, the OpenStack Block Storage service.

The main purpose of this library is to illustrate the broad range of possibilities of the Cinder services provided by their flexible mechanisms.

Originally it was only meant to contain Scheduler related code, hence the name, but now it's also including API code.

Currently there's 2 interesting features for the Scheduler, which are the possibility of changing the default provisioning type on volume creation for volumes that don't specify the type using the *provisioning:type* extra spec and an alternative calculation of the free space consumption.

Scheduler's original approach to space consumption by new volumes is conservative to prevent backends from filling up due to a sudden burst of volume creations.

The alternative approach is more aggressive and is adequate for deployments where the workload is well know and a lot of thin volumes could be requested at the same time.

It's important to notice that even though the Schedulers will be able to understand *provisioining:type* extra spec it will depend on the backend if this parameter is actually used or not.

Another interesting possibility, added this time at the API level, is having default volume types for specific projects and/or users.

The precedence would be:

1- Source volume type: volume, snapshot, image metadata 2- User's default volume type 3- Project's default volume type 4- Cinder configured *default_volume_type*

User and project default volume types must be defined in Keystone's DB in the *extra* field of *user* and *project* tables. It is not necessary to define it for all projects and users, if they none is defined Cinder's default will be used.

The *extra* field is a JSON string and may already contain extra information like an email address.

The key used to define the default volume types is *default_vol_type* and its value can be an UUID or a name.

Since there is no CRUD REST API available in Keystone for custom values in the *extra* field we'll need to change this manually in the DB.

- Free software: Apache Software License 2.0

- Documentation: https://alt-cinder-sch.readthedocs.io.

Features

- Can default capacity calculations to thin or thick.
- Less conservative approach to free space consumption calculations.
- Per project and/or user default volume types.

Usage

First we'll need to have the package installed:

```
# pip install alt_cinder-sch
```

For Cinder's scheduler features we'll have to change the configuration to use the package:

```
scheduler_host_manager = alt_cinder_sch.host_managers.HostManagerThin
scheduler_default_filters = AvailabilityZoneFilter,AltCapacityFilter,
↪CapabilitiesFilter
scheduler_driver = alt_cinder_sch.scheduler_drivers.FilterScheduler
```

And finally restart scheduler services.

For Cinder's API feature the Cinder configuration is just:

```
volume_api_class = alt_cinder_sch.api.DefaultVolumeTypeAPI
```

As well as changing Keystone's *user* and *project* tables.

CHAPTER 2

Installation

Stable release

To install Alternative Cinder Scheduler Classes, run this command in your terminal:

```
$ pip install alt_cinder_sch
```

This is the preferred method to install Alternative Cinder Scheduler Classes, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The easiest way of installing the package from source is using pip:

```
$ pip install git+https://github.com/Akrog/alt_cinder_sch.git
```

Alternative the sources for Alternative Cinder Scheduler Classes can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/akrog/alt_cinder_sch
```

Or download the tarball:

```
$ curl  -OL https://github.com/akrog/alt_cinder_sch/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

Proportional Free Space Calculation

To use Alternative Cinder Scheduler Classes in a Cinder deployment the package will need to be first installed in all scheduler nodes as instructed in the *installation guide*.

Then configuration files will need to be updated to use the classes:

```
scheduler_host_manager = alt_cinder_sch.host_managers.HostManagerThin
scheduler_default_filters = AvailabilityZoneFilter,AltCapacityFilter,
↪CapabilitiesFilter
scheduler_driver = alt_cinder_sch.scheduler_drivers.FilterScheduler
```

Scheduler's default filters could vary depending on your configuration, but the only filter provided by this package at the moment is the AltCapacityFilter.

In above example we were defaulting to thin provisioning calculations for any backend that supported thin provisioning, but we can also default to thick provisioning is we use *HostManagerThick* instead as the *scheduler_default_filters*.

Default Volume Types

To support Default Volume Types based on users or projects the package needs to be installed in all API nodes as instructed in the *installation guide*.

Then configuration files on the API nodes will need to be updated to use our custome API class:

```
volume_api_class = alt_cinder_sch.api.DefaultVolumeTypeAPI
```

Since we are only changing the configuration for the API service only these will need to be restarted, leaving Scheduler, Backup, and Volume services as they were.

And the default volume types will need to be added to the users and/or projects in Keystone directly in the DB (there's no REST API).

Data must be added to extra DB field as JSON with key *default_vol_type*.

If there is no data in the user's extra field we can run:

```
UPDATE user
SET extra='{"default_vol_type": "iscsi"}'
WHERE id=$USER_UUID;
```

If the project's extra field already had info, like an email, we could do:

```
UPDATE project
SET extra=CONCAT(SUBSTRING(extra, 1, LENGTH(extra) - 1),
                 ', "default_vol_type": 'iscsi"}')
WHERE name='admin';
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/akrog/alt_cinder_sch/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

Write Documentation

Alternative Cinder Scheduler Classes could always use more documentation, whether as part of the official Alternative Cinder Scheduler Classes docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/akrog/alt_cinder_sch/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *alt_cinder_sch* for local development.

1. Fork the *alt_cinder_sch* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/alt_cinder_sch.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv alt_cinder_sch
   $ cd alt_cinder_sch/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 alt_cinder_sch tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/akrog/alt_cinder_sch/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_alt_cinder_sch
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search