oaktree Documentation

Release

OpenStack Foundation

Mar 20, 2017

Contents

Contents:

CHAPTER 1

oaktree

Make your cloud throw some shade

oaktree is a gRPC interface for interacting with OpenStack clouds that is inherently interoperable and multi-cloud aware. It is based on the python shade library, which grew all of the logic needed to interact with OpenStack clouds and to work around differences in vendor deployment choices. Rather than keep all of that love in Python Library form, oaktree allows other languages to reap the benefits as well.

oaktree is not a replacement for all of the individual project REST APIs. Those are all essential for cross-project communication and are well suited for operators who can be expected to know things about how they have deployed their clouds - and who in fact WANT to be able to make changes in the cloud knowing deployment specifics. oaktree will never be for them.

oaktree is for end-users who do not and should not know what hypervisor, what storage driver or what network stack the deployer has chosen. The two sets of people are different audiences, so oaktree is a project to support the end user.

Using

Install oaktreemodel by hand. Then:

In one window:

```
python oaktree/server.py
```

oaktree/server.py assumes you have a clouds.yaml accessible.

In another window:

```
python -i devstack/test.py
```

You'll have an images and a flavors object you can poke at.

If you want to operate against a different cloud than *devstack*, you can pass it to devstack/test.py as the first command line argument.

Shape of the Project

oaktree should be super simple to deploy, and completely safe for deployers to upgrade from master constantly. Once it's released as a 1.0, it should NEVER EVER EVER EVER EVER EVER EVER have a backwards incompatible change. There is no reason, no justification, no obsession important enough to inflict such pain on the user.

The shade library will grow the ability to detect if a cloud has an oaktree api available, and if it does, it will use it. Hopefully we'll quickly reach a point where all deployers are deploying oaktree.

- Documentation: http://docs.openstack.org/developer/oaktree
- Source: http://git.openstack.org/cgit/openstack/oaktree
- Bugs: http://bugs.launchpad.net/oaktree

CHAPTER 2

Installation

At the command line:

```
$ pip install oaktree
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv oaktree
$ pip install oaktree
```

CHAPTER 3

Oaktree Design

Once 1.0.0 is released, oaktree pledges to never break backwards compatability.

Oaktree is intended to be safe for deployers to run CD from master. In fact, a deployer running a kilo OpenStack should be able to install tip of master of oaktree and have everything be perfectly fine.

Oaktree must be simple to install and operate. A single node install with no shared caching or locking is likely fine for most smaller clouds. For larger clouds, shared caching and locking are essential for scale out. Both must be supported, and simple.

Oaktree is not pluggable.

Oaktree does not allow selectively enabling or disabling features or part of its API.

Oaktree should be runnable by an end user pointed at a local clouds.yaml file.

Oaktree should be able to talk to other oaktrees.

Oaktree users should never need to know any information about the cloud other than the address of the oaktree endpoint. Cloud-specific information the user needs to know must be exposed via a capabilities API. For instance, in order for a user to upload an image to a cloud, the user must know what format the cloud requires the image to be in. The user must be able to ask oaktree what image format(s) the cloud accepts.

Data returned from oaktree should be normalized such that it is consistent no matter what drivers the cloud in question has chosen. This work is done in shade, but shapes the design of the protobuf messages.

All objects in oaktree should have a Location. A Location defines the cloud, the region, the zone and the project that contains the object. For objects that exist at a region and not a zone level, like flavors and images, zone will be null. For objects that exist at a cloud level, region will be null.

CHAPTER 4

Frequently Asked Questions

Why gRPC and not REST?

There are three main reasons.

We already have REST APIs. oaktree is not intended to replace them, but to supplement them to grease the 80% case that can be inter-operable.

gRPC comes out of the gate with direct support for a pile of languages, so supporting our non-Python friends is direct and straightforward.

A TON of time is spent in shade polling OpenStack for results. That may not sound like a problem - but when you spin up thousands of VMs a day like Infra does, the polling becomes a major engineering challenge. gRPC operates over http/2 and has support for bi-directional channels - which means you can just have a function notify you when something is done. That's a win for everyone

Why write it in Python rather than XXX?

The hard part of this isn't the gRPC api - it's the business logic that's in the shade library. If we wrote oaktree from scratch in C++ (because hello super-high-performance gRPC backend!) - we'd be faced with the task of re-implementing all of the shade business logic in C++. If you haven't looked, there is a LOT.

shade is what infra uses for nodepool. It has copious features in it already to deal with extremely high scale - including configurable caching, batched list update operations to prevent thundering herds and well exercised multi-threaded support.

The interesting part also isn't the server (it's a simple proxy layer) - it's the clients. THOSE definitely want much love in the different languages. The infrastructure is in place for Python, C++ and Go. Ruby, javascript and C# should follow asap.

Can I add support for my project?

Yes. It has to be added to shade first, which accepts patches from anything that can be tested consistently in a devstack job. We require all new features in shade to come with functional tests. Once it's in shade, it can be added as an API to oaktree.

However ... oaktree and shade both promise 100% backwards compatibility at all times. If your project is still young, be aware that once an API is added to shade or oaktree it will need to be supported until the end of time.

CHAPTER 5

Work Needed

Design the auth story

The native/default auth for gRPC is oauth. It has the ability for pluggable auth, but that would raise the barrier for new languages. I'd love it if we can come up with a story that involves making API users in keystone and authorizing them to use oaktree via an oauth transaction. The keystone auth backends currently are all about integrating with other auth management systems, which is great for environments where you have a web browser, but not so much for ones where you need to put your auth credentials into a file so that your scripts can work. I'm waving my hands wildly here - because all I really have are problems to solve and none of the solutions I have are great.

Design Glance Image / Swift Object Uploads and Downloads

Having those two data operations go through an API proxy seems inefficient. However, having them not in the API seems like a bad user experience. Perhaps if we take advantage of the gRPC streaming protocol support doing a direct streaming passthrough actually wouldn't be awful. Or maybe the better approach would be for the gRPC call to return a URL and token for a user to POST/PUT to directly. Literally no clue.

Design and implement Capabilities API

shade and the current oaktree codebase rely on os-client-config and clouds.yaml for information about the cloud and what it can do. As a service, some of the pieces of information in os-client-config need to be queriable by the user.

Implement API surfaces

In general, all of the API operations shade can perform should be exposed in oaktree. In order to shape that work, we should tackle them in the following order:

1. API surface needed for nodepool

2. API surface needed for existing Ansible modules
3. Everything else

The API surface needed for nodepool is:

list_flavors

create_image delete_image get_image list_images

create_keypair delete_keypair list_keypairs

create_server delete_server get_server list_servers

// These two require the most thought wait_for_server delete_unattached_floating_ips

Implement oaktree backend in shade

It's turtles all the way down. If shade sees that a cloud has an oaktree service, shade should talk to it over gRPC instead of talking to the REST APIs directly.

CHAPTER 6

Contributing

If you would like to contribute to the development of OpenStack, you must follow the steps in this page:

> http://docs.openstack.org/infra/manual/developers.html

If you already have a good understanding of how the system works and your OpenStack accounts are set up, you can skip to the development workflow section of this documentation to learn how changes to OpenStack should be submitted for review via the Gerrit tool:

> http://docs.openstack.org/infra/manual/developers.html#development-workflow

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Storyboard, not GitHub:

> https://storyboard.openstack.org/#!/project/855

CHAPTER 7

Indices and tables

- genindex
- modindex
- search