Poppy Documentation

Release 1.0.10.dev10

OpenStack Foundation

April 28, 2016

Contents

Poppy is an OpenStack-related project to provide a generic and modular vendor-neutral API which wraps provisioning instructions for CDN vendors that support it.

Poppy, as with all OpenStack projects, is written with the following design guidelines in mind:

- **Component-based architecture**: Quickly add new behaviors
- **Highly available**: Scale to very serious workloads
- **Fault tolerant**: Isolated processes avoid cascading failures
- **Recoverable**: Failures should be easy to diagnose, debug, and rectify
- **Open standards**: Be a reference implementation for a community-driven api

This documentation is generated by the Sphinx toolkit and lives in the source tree. Additional draft and project documentation on Poppy and other components of OpenStack can be found on the OpenStack wiki. Cloud administrators, refer to docs.openstack.org.

CHAPTER 1

Concepts

1.1 Glossary

1.2 Concepts

Poppy is an OpenStack-related project to provide a generic and modular vendor-neutral API that wraps provisioning instructions for CDN vendors that support it.

Caching Rule A caching rule provides the user with fine-grained control over the time-to-live (TTL) of an object. When the TTL expires for an object, the edge node pulls the object from the origin again.

Domain A domain represents the domain name through which visitors retrieve content. The underlying site may be served through a CDN. A service can have multiple domains. A user typically uses CNAME for this domain to their CDN provider.

Driver Poppy has a modular API where many components are interchangeable. These components are known as drivers (see Stevedore Framework). It is possible to use different transport drivers, manager drivers, storage drivers, and provider drivers.

Edge Node CDN providers have many points-of-presence (POP) servers around the world. These servers are known as edge nodes. These edge nodes cache the content and serve it directly to customers, thus reducing transit time to a customers location.

Flavor A flavor allows the user to decide what CDN providers they would like their service to use. Operators can define the flavors offered, and assign a CDN provider belonging to that flavor. Use flavors to abstract away the underlying provider used.

Manager Driver A manager driver contains the business logic within the application. This driver is responsible for delegating tasks to Storage and Provider Drivers.

Origin An origin is an address (ip or domain) from which the CDN provider pulls content. A service can have multiple origins.

Provider There are many established CDN vendors in the market. A provider is one of these vendors, who has decided to participate in the Poppy project. These participating providers will have a provider driver that can communicate with their API.

Provider Driver A provider driver is responsible for communicating with the third party providers who are participating in the Poppy project.

Purge Purging removes content from the edge servers, so it can be refreshed from your origin servers.

Restriction A restriction enables the user to define rules about who can or cannot access content from the cache. Examples of a restriction are allowing requests only from certain domains, geographies, or IP addresses.

Service A service represents a customers' application that has its content cached to the edge nodes.

Status The time it takes for a service configuration to be distributed amongst a CDN provider cache can vary. The status indicates the current state of the service.

Storage Driver A storage driver is responsible for communicating with the chosen data store to store service configurations.

Transport Driver A transport driver handles the incoming requests to the API. The recommended transport driver for Poppy is the Pecan Driver based on WSGI.

1.3 Provider Details

Header	Poppy	Akamai	CloudFront	Fastly	MaxCDN
Name of CDN site	Service	Policy	Distribution	Service	Pull Zone
Name of Origin	Origin	Origin	Origin	Backend	Origin
Access URL	Custom URL	Vanity URL	CloudFront URL	Domain URL	Custom Domain URL
Interface Format	JSON	JSON	XML	JSON	JSON
Python client	None	None	boto	fastly-py, fastly-python	python-maxcdn
Logs	•		S3	S3/syslog/FTP/Storm	S3/ReportsAPI/RawLogs
Subaccount support	N/A	Yes	Yes (AWS IAM)	Yes	Yes
Propagation Time					
Create	N/A	15 mins	15 mins	100 ms	100 ms
Update	N/A	15 mins	15 mins	100 ms	100 ms
Purge	N/A	?	?	300 ms	100 ms
Data transfer rate			1,000 Mbps		
Requests per second			1000		
CDNs/account			Default: 200	Default: 20?	
RTMPs/account			100		
CNAMEs/service			100		
Origins/service			25		
Cache behaviors/service			25		
Whitelisted headers/cache			10		
Whitelisted cookies/cache			10		
Analytics			Hourly(Web)	Realtime	Realtime
Failover			AWS Route53	Yes	?
Loadbalancing			AWS ELB	Yes	Yes
Billing			Pay as you go	Pay as you go	Various plans available
HTTP Accelerators			No Information	Varnish	Varnish
Web TTL					

Continued on next page

Table 1.1 – continued from previous page

Header	Poppy	Akamai	CloudFront	Fastly	MaxCDN
Minimum			0 Seconds	0 Seconds	0 Seconds
Maximum			Year: 2038	> 30 days	> 30 days
Tick			Seconds	Seconds	Seconds (Using headers)
Default			24 Hours	1 Hour	24 Hours
Media TTL					
Minimum			1 Hour		
Maximum			Year: 2038	> 30 days	> 30 days
Tick			Seconds	Seconds	Seconds (Using headers)
Default			24 Hours	1 Hour	24 Hours

CHAPTER 2

Getting Started

2.1 Installing Poppy

Note: These instructions are for running a local instance of Poppy. We suggest you run this inside a virtualenv.

You must have CassandraDB installed and running. We recommend using Docker (see below)

1. From your home folder, create the `~/.poppy` folder and clone the repo:

```
$ cd
$ mkdir .poppy
$ git clone https://github.com/openstack/poppy.git
```

2. Copy the Poppy config files to the directory `~/.poppy`:

```
$ cp poppy/etc/poppy.conf ~/.poppy/poppy.conf
$ cp poppy/etc/logging.conf ~/.poppy/logging.conf
```

3. Find the `[drivers:storage:cassandradb]` section in `~/.poppy/poppy.conf` and modify the URI to point to your local casssandra cluster:

```
[drivers:storage:cassandra]
cluster = "localhost"
keyspace = poppy
```

4. You need to create the default keyspace "poppy" on your cassandra host/cluster. Log into cqlsh, do:

```
cqlsh> CREATE KEYSPACE poppy WITH REPLICATION = { 'class' : 'SimpleStrategy' , 'replication_fact
```

5. For logging, find the `[DEFAULT]` section in `~/.poppy/poppy.conf` and modify as desired:

```
log_file = server.log
```

6. Change directories back to your local copy of the repo:

```
$ cd poppy
```

7. Run the following command so you can see the results of any changes you make to the code without having to reinstall the package each time:

```
$ pip install -e .
```

8. Start the Poppy server:

```
$ poppy-server
```

9. Test that Poppy is working by requesting the home doc (with a sample project ID):

```
$ curl -i -X GET http://0.0.0.0:8888/v1.0/123
```

You should get an **HTTP 200** along with some headers that look similar to the following example:

```
HTTP/1.0 200 OK
Date: Thu, 13 Feb 2014 14:34:21 GMT
Server: WSGIServer/0.1 Python/2.7.3
Content-Length: 464
Content-Type: application/json-home
Cache-Control: max-age=86400
```

2.1.1 Installing CassandraDB (using Docker)

1. From the *docker/cassandra* folder:

```
$ docker build -t db .
```

2. Open the 9160 and 9042 ports for Cassandra. Name the container 'cassandra':

```
$ docker run -d -p 9160:9160 -p 9042:9042 --name cassandra db
```

3. Test the running cassandra instance (you may need to `pip install cqlsh`):

```
$ cqlsh <local ip> 9160
```

Where local ip is the ip address of your running docker container

4. Import the schema file from the poppy/storage/cassandra/schema.cql file.

2.2 Running tests

First, install the additional requirements using the following command:

```
$ pip install tox
```

Then run tests using the following command:

```
$ tox
```

Tox checks that Poppy works against the following environments:

```
python 2.6
python 2.7
python 3.3
pypy
```

Tox also performs the following coding enforcement checks:

```
pep8
code coverage (100% required)
```

2.3 Contributing to Poppy

2.3.1 First steps

Interested in contributing to Poppy? That's great to hear!

First of all, make sure to join our communication forums:

- Subscribe to our [[MailingLists|mailing lists]].
- Join us on IRC! You can chat with us directly in the ‘’‘#openstack-poppy’‘’ channel on ‘’‘irc.freenode.org’‘’. Don't know to use IRC? You can find some directions in [[UsingIRC]] wiki page.
- Answer and ask questions on [https://ask.openstack.org/ Ask OpenStack].

2.3.2 How can I contribute?

You can contribute to Poppy in many ways. Of course, coding is one, but you can also contribute as a tester, documenter, designer, or translator.

Coding

Bug fixing and triaging

The first area where you can help is bug fixing. ‘’Confirmed” bugs are usually your best choice. ‘’Triaged bugs” should even contain tips on how you can fix them.

Once you selected the bug you want to work on, go ahead and assign it to yourself, branch the code, implement the fix, and propose your change for merging into trunk!

Some easy-to-fix bugs may be marked with the ‘’‘low-hanging-fruit’‘’ tag. Those are good targets for a beginner.

Reported bugs need care: prioritizing them correctly, confirming them, making sure they don't go stale... All those tasks help immensely. If you want to start contributing in coding but you are not a hardcore developer, consider helping in this area!

Bugs can be marked with different tags according to their status, as follows: * ‘’New” bugs are those bugs that have been reported by a user but haven't been verified by the community yet. * ‘’Confirmed” bugs are those bugs that have been reproduced by someone else than the reporter. * ‘’Triaged” bugs are those bugs that have been reproduced by a core developer. * ‘’Incomplete” bugs are those bugs that don't have enough information to be reproduced. * ‘’In Progress” bugs are those bugs that are being fixed by a developer. * ‘’Invalid” bugs are those bugs that don't qualify as a bug. These usually stem from a support request or something unrelated to the project.

You can learn more, see Launchpad's [http://blog.launchpad.net/general/of-bugs-and-statuses Of Bugs and Statuses].

You only have to worry about ‘’New” bugs. If you can reproduce them, you can mark them as ‘’Confirmed’‘. If you cannot reproduce them, you can ask the reporter to provide more information and mark them as ‘’Incomplete’‘. If you think that they aren't bugs, mark them as “Invalid”. (Be careful! Asking someone else in Poppy is always a good idea.)

Also, you can contribute instructions about how to fix a given bug.

Check out the [[BugTriage|Bug Triage]] wiki for more information.

Reviewing

Every patch submitted to OpenStack gets reviewed before it can be approved and merged. We get a lot of contributions and everyone can - and is encouraged! [https://review.openstack.org/#/q/status:open+project:openstack/poppy,n,z Review Poppy's existing patches]. Pick an open review and go through it. Test it if possible, and leave a comment with a +1 or -1 vote describing what you discovered. If you're planning to submit patches of your own, this is a great way to learn about what the community cares about and to learn about the code base.

Feature development

Once you get familiar with the code, you can start to contribute new features. New features get implemented every 6 months in a [[ReleaseCycle|development cycle]]. We use Launchpad [[Blueprints]] to track the design and implementation of significant features, and we use Design Summits every 6 months to discuss them in public. Code should be proposed for inclusion before we reach the final feature milestone of the development cycle.

Testing

Testing efforts are highly related to coding. If you find that there are test cases missing or that some tests could be improved, you are encouraged to report it as a bug, and then provide your fix. Learn more about this in Write The Tests!

Documenting

You can contribute to Poppy's Users Guide and Poppy's Wiki. See Documentation/HowTo for details, as well as Documentation/HowTo/FirstTimers, which has some other info that may be useful.

To fix a documentation bug, check the bugs marked with the 'doc' tag in Poppy's [https://bugs.launchpad.net/poppy/+bugs?field.tag=doc bug list]. In case that you want to report a documentation bug, then don't forget to add the 'doc' tag to it :)

You can also start by reading the developer documentation which is created using Sphinx as part of the code in the /doc/source/ directory and published to [https://poppy.readthedocs.org Read The Docs].

Also, monitor [http://ask.openstack.org Ask OpenStack] to curate the best answers that can be folded into the documentation.

2.4 Using Gerrit

2.4.1 Before you begin

To familiarize yourself with Poppy, try it out using the information in our [https://github.com/openstack/poppy repo]. When you are ready to start contributing, you will need to execute an [http://docs.openstack.org/infra/manual/developers.html#account-setup OpenStack CLA]. This is required before you can submit reviews to our [https://git.openstack.org/cgit/openstack/poppy Poppy StackForge Repo]. For information about how prepare for contribution, please consult the [http://docs.openstack.org/infra/manual/developers.html developer guide]].

Learn about Gerrit

Be sure to read the [[Gerrit_Workflow|Gerrit Workflow]] wiki page for information about how to submit your commit for review so it can be merged into the Poppy code base.

2.4.2 Setting up your git review settings

git config –global user.name "Firstname Lastname" git config –global user.email "your_email@youremail.com" git config –global gitreview.username "your_launchpad_username"

To check your git configuration:

git config –list

2.4.3 Installing git-review

On Ubuntu, MacOSX, or most other Unix-like systems, use the following command: pip install git-review

There are other installation options detailed in the [[Gerrit_Workflow#Git_Review_Installation|Installation Instructions]]. You can now check out the Poppy code and begin working on it.

2.5 Your first commit

2.5.1 Set up your local branch

Use the following commands to set up your local branch:

git clone git://git.openstack.org/openstack/poppy cd poppy git checkout -b [branch name] git review -s

Create a topic branch to hold your work and switch to it. If you are working on a blueprint, name your topic branch bp/BLUEPRINT where BLUEPRINT is the name of a blueprint in launchpad (for example, "bp/authentication"). The general convention when working on bugs is to name the branch bug/BUG-NUMBER (for example, "bug/1234567"). Otherwise, give it a meaningful name because it will show up as the topic for your change in Gerrit.

2.5.2 Write some awesome code

At this point can write your code and push it to Gerrit for review by using the following commands:

git add <list of files you added/changed> git commit -a git review -v –draft

Once you are happy with your code and want it to be reviewed, you want to convert it from a Draft. "Sign In" at https://review.openstack.org/ and after verifying the review yourself, hit the "Publish" button on the page.

If you know you are ready for others to review your code, you can skip the draft step and use: git review -v

If you want to revise your patchset in the review system in response to feedback, make your changes, then use: git commit -a –amend git review -v

Upon approval of the review, your code is automatically merged.

Reviews

The OpenStack CI system uses the concept of core reviewers. These are individuals who have consistently reviewed code for the project, and helped over a considerable period of time to improve the quality and consistency of what we merge into the code base. Project contributors feel that this reviewer is a positive influence on the team and that they maintain the values and traditions of the OpenStack development community.

Policies

Existing core reviewers may nominate new ones in an ML thread. Consent among the current reviewers shall result in the declaration of the new core reviewer by the PTL. Lack of unanimous consent shall be carefully considered, and a final decision informed by input from from active team members shall be made by the PTL. Core reviewers who are judged by their peers in the core review group to fall short of the expectations for contribution of a core reviewer may be nominated for return to regular reviewer status.

The current Gerrit policy is:

> label-Code-Review = -2..+2 group poppy-core label-Approved = +0..+1 group poppy-core

Patches require a core reviewer to mark a review as "Approved" before they are merged.

Review Guidelines

2.5.3 Code Approval for Merge

- For Approval, two core reviewers shall supply a <code>+2</code>.

2.5.4 Continuing Someone Else's Contribution

- If a patch submitted by one contributor is picked up and completed by another contributor, [http://www.mail-archive.com/openstack-dev@lists.openstack.org/msg05998.html special handling] of the resolution should be used.

2.5.5 Advice for Reviewers

- A <code>-1</code> vote is an opportunity to make our code better before it is merged. Please do your best to make helpful, actionable -1 votes.
- Avoid the temptation to blindly <code>+1</code> code without reviewing it in sufficient detail to form an opinion.
- When voting <code>-1</code> on a patch, it means that you want the submitter to make a revision in accordance with your feedback before core reviewers should consider this code for merge.
- If you ask a question, you should vote <code>0</code> unless you anticipate that the answer to that question is likely to cause you to vote against the patch without further revisions.
- If you use a <code>-1</code> vote for a question, and the contributor answers the question, please respond acknowledging the question. Either change your vote or follow up with additional rationale for why this should remain a <code>-1</code> comment.
- A <code>-2</code> vote is a veto by a single core reviewer. It is sticky. That means that even if you revise your patch, that vote will persist. To allow your patch to merge, that same reviewer must clear the <code>-2</code> vote first. This vote is used when you have contributed something that is not in alignment with the current project vision, or is implemented in a way that can not be accepted. For example, security concerns that a core reviewer wants to individually re-evaluate before allowing the contribution to continue. It can also be used as a way to halt further gate testing of a patch, if something is included that may break the gate. It works even after a <code>2*+2,+A</code> approval for merge, but before the patch reaches MERGED state.
- To avoid a <code>-2</code> vote, discuss your plans with the development team prior to writing code, and post a WIP (workflow-1) patch while you are working on it, and ask for input before you submit it for merge review.

2.6 Testing

See our [[Poppy/Testing]] wiki.

CHAPTER 3

Operating Poppy

3.1 Operating Poppy

CHAPTER 4

Developer Docs

4.1 poppy package

class poppy.bootstrap.**Bootstrap**(*conf*)

Bases: object

Defines the CDN bootstrapper.

The bootstrap loads up drivers per a given configuration, and manages their lifetimes.

distributed_task

distributed task driver.

:returns distributed task driver

dns

DNS.

manager

manager.

:returns mgr driver

metrics

metrics driver.

:returns metrics driver

notification

notification.

:returns mgr

provider

provider.

:returns mgr

run()

storage

storage.

:returns mgr driver

transport

transport.

:returns mgr driver

Sub-modules:

4.1.1 poppy.model module

class poppy.model.service.**Service**(*service_id*, *name*, *domains*, *origins*, *flavor_id*, *caching=[]*, *restrictions=[]*, *log_delivery=None*, *operator_status='enabled'*, *project_id=''*)

Bases: poppy.model.common.DictSerializableModel

Service Class.

caching
Get or set caching.

domains
Get or set domains.

flavor_id
Get or set flavor ref.

classmethod init_from_dict(*project_id*, *input_dict*)
Construct a model instance from a dictionary.

This is only meant to be used for converting a response model into a model. When converting a model into a request model, use to_dict.

log_delivery
Get log_delivery.

name
Get or set name.

operator_status
Get operator status.

origins
Get or set origins.

project_id
Get project id.

provider_details
Get or set provider details.

restrictions
Get or set restrictions.

service_id
Get service id.

status
Get or set status.

:returns boolean

to_dict()
Construct a model instance from a dictionary.

This is only meant to be used for converting a response model into a model. When converting a model into a request model, use to_dict.

class `poppy.model.helpers.domain.`**`Domain`**(*domain*, *protocol='http'*, *certificate=None*)

Bases: `poppy.model.common.DictSerializableModel`

`certificate`

certificate option.

:returns certificate

`domain`

domain.

:returns domain

classmethod **`init_from_dict`**(*dict_obj*)

Construct a model instance from a dictionary.

This serves as a 2nd constructor

Parameters **`dict_obj`** – dictionary object

:returns o

`protocol`

`to_dict`()

class `poppy.model.helpers.origin.`**`Origin`**(*origin*, *hostheadertype='domain'*, *hostheadervalue='-'*, *port=80*, *ssl=False*, *rules=[]*)

Bases: `poppy.model.common.DictSerializableModel`

Origin.

`hostheadertype`

hostheadertype.

`hostheadervalue`

hostheadervalue.

classmethod **`init_from_dict`**(*dict_obj*)

Construct a model instance from a dictionary.

This serves as a 2nd constructor

Parameters **`dict_obj`** – dictionary object

:returns o

`origin`

origin.

`port`

port.

:returns port

`rules`

rules.

:returns rules

`ssl`

self.

:returns ssl

`to_dict`()

class `poppy.model.helpers.cachingrule.`**`CachingRule`**(*name*, *ttl*, *rules=[]*)

Bases: `poppy.model.common.DictSerializableModel`

Parameters **`DictSerializableModel`** –

classmethod **`init_from_dict`**(*dict_obj*)

Construct a model instance from a dictionary.

This serves as a 2nd constructor

Parameters **`dict_obj`** – dictionary object

:returns o

`name`

name.

:returns name

`rules`

rules.

:returns rules

`to_dict`()

`ttl`

ttl.

:returns ttl

class `poppy.model.helpers.restriction.`**`Restriction`**(*name*, *access='whitelist'*, *rules=[]*)

Bases: `poppy.model.common.DictSerializableModel`

Restriction.

`access`

name.

:returns name

classmethod **`init_from_dict`**(*dict_obj*)

Construct a model instance from a dictionary.

This serves as a 2nd constructor

Parameters **`dict_obj`** – dictionary object

:returns o

`name`

name.

:returns name

`rules`

rules.

:returns rules

`to_dict`()

class `poppy.model.helpers.rule.`**`Rule`**(*name=None*, *referrer=None*, *http_host=None*, *client_ip=None*, *geography=None*, *http_method=None*, *request_url='/*'*)

Bases: `poppy.model.common.DictSerializableModel`

Rule.

client_ip

geography
http_host.

http_host
http_host.

http_method

name
name.

referrer

request_url

4.1.2 poppy.transport drivers

CDN Transport Drivers

class poppy.transport.base.**TransportDriverBase**(*conf*, *manager*)
Bases: object

Base class for Transport Drivers to document the expected interface.

Parameters **conf** (*oslo_config.cfg.CONF*) – configuration instance

app
Get app.

:returns app

conf
Get conf.

:returns conf

listen()
Start listening for client requests (self-hosting mode).

:raises NotImplementedError

manager
Get manager

:returns manager

poppy.transport.pecan driver

class poppy.transport.pecan.controllers.base.**Controller**(*driver*)
Bases: pecan.rest.RestController

add_controller(*path*, *controller*)

driver

class poppy.transport.pecan.controllers.root.**RootController**(*driver*)
Bases: *poppy.transport.pecan.controllers.base.Controller*

add_controller(*path*, *controller*)

class poppy.transport.pecan.driver.**PecanTransportDriver**(*conf*, *manager*)

Bases: *poppy.transport.base.TransportDriverBase*

listen()

Pecan v1.0 Controllers

4.1.3 poppy.manager drivers

poppy.manager.base base classes

class poppy.manager.base.driver.**ManagerDriverBase**(*conf*, *storage*, *providers*, *dns*, *distributed_task*, *notification*, *metrics*)

Bases: object

Base class for driver manager.

analytics_controller

Returns the driver's analytics controller

:raises NotImplementedError

conf

:returns conf

distributed_task

dns

flavors_controller

Returns the driver's flavors controller

:raises NotImplementedError

health_controller

Returns the driver's health controller

:raises NotImplementedError

metrics

notification

providers

:returns providers

services_controller

Returns the driver's services controller

:raises NotImplementedError

storage

:returns storage

class poppy.manager.base.controller.**ManagerControllerBase**(*driver*)

Bases: object

Top-level class for controllers.

Parameters **driver** – Instance of the driver instantiating this controller.

driver

class poppy.manager.base.services.**ServicesControllerBase**(*manager*)

Bases: *poppy.manager.base.controller.ManagerControllerBase*

Services controller base class.

create(*project_id*, *auth_token*, *service_obj*)

:param project_id :param service_obj :raises: NotImplementedError

delete(*project_id*, *service_id*)

DELETE

:param project_id :param service_id :raises: NotImplementedError

get(*project_id*, *service_id*)

GET

:param project_id :param service_id :raises: NotImplementedError

list(*project_id*, *marker=None*, *limit=None*)

:param project_id :param marker :limit :raises: NotImplementedError

purge(*project_id*, *service_id*, *hard=False*, *purge_url=None*)

If purge_url is none, all content of this service will be purge.

services_action(*project_id*, *action*, *domain=None*)

:param project_id :param action :param domain :raises ValueError

update(*project_id*, *service_id*, *service_obj*)

POST

:param project_id :param service_id :param service_obj :raises: NotImplementedError

class poppy.manager.base.providers.**ProviderWrapper**

Bases: object

"ProviderWrapper class.

create(*ext*, *service_obj*)

Create a provider

:param ext :param service_obj :returns: ext.obj.service_controller.create(service_obj)

create_certificate(*ext*, *cert_obj*, *enqueue*)

Create a provider

:param ext :param service_obj :returns: ext.obj.service_controller.create(service_obj)

delete(*ext*, *provider_details*, *project_id*)

purge(*ext*, *service_obj*, *provider_details*, *hard=False*, *purge_url=None*)

update(*ext*, *provider_details*, *service_obj*)

Update a provider

:param ext :param provider_details :param service_old :param service_updates :param service_obj

poppy.manager.default driver

Default manager driver implementation.

class poppy.manager.default.driver.**DefaultManagerDriver**(*conf*, *storage*, *providers*, *dns*, *distributed_task*, *notification*, *metrics*)

Bases: *poppy.manager.base.driver.ManagerDriverBase*

Default Manager Driver.

analytics_controller

background_job_controller

flavors_controller

health_controller

home_controller

services_controller

ssl_certificate_controller

class poppy.manager.default.services.**DefaultServicesController**(*manager*)

Bases: *poppy.manager.base.services.ServicesControllerBase*

Default Services Controller.

create(*project_id*, *auth_token*, *service_json*)

create.

:param project_id :param auth_token :param service_json :raises LookupError, ValueError

delete(*project_id*, *service_id*)

delete.

:param project_id :param service_id :raises LookupError

determine_sleep_times()

get(*project_id*, *service_id*)

get.

:param project_id :param service_id :return controller

get_certs_by_status(*status*)

get_domains_by_provider_url(*provider_url*)

get_service_by_domain_name(*domain_name*)

get_services_by_status(*status*)

get_services_limit(*project_id*)

list(*project_id*, *marker=None*, *limit=None*)

list.

:param project_id :param marker :param limit :return list

migrate_domain(*project_id*, *service_id*, *domain_name*, *new_cert*, *cert_status='deployed'*)

purge(*project_id*, *service_id*, *hard=False*, *purge_url=None*)

If purge_url is none, all content of this service will be purge.

services_action(*project_id*, *action*, *domain=None*)

perform action on services

:param project_id :param action :param domain

:raises ValueError

services_limit(*project_id*, *limit*)

set_service_provider_details(*project_id*, *service_id*, *auth_token*, *status*)

update (*project_id*, *service_id*, *auth_token*, *service_updates*, *force_update=False*)
update.

:param project_id :param service_id :param auth_token :param service_updates :param force_update :raises LookupError, ValueError

4.1.4 poppy.storage drivers

poppy.storage.base base classes

class poppy.storage.base.controller.**StorageControllerBase** (*driver*)
Bases: object

Top-level class for controllers.

Parameters **driver** – Instance of the driver instantiating this controller.

class poppy.storage.base.driver.**StorageDriverBase** (*conf*)
Bases: object

Interface definition for storage drivers.

Data plane storage drivers are responsible for implementing the core functionality of the system.

Connection information and driver-specific options are loaded from the config file.

Parameters **conf** (*oslo_config.ConfigOpts*) – Configuration containing options for this driver.

flavors_controller
Returns the driver's hostname controller.

:raise NotImplementedError

is_alive ()
Check whether the storage is ready.

:raise NotImplementedError

services_controller
Returns the driver's hostname controller.

:raise NotImplementedError

storage_name ()
For name.

:raise NotImplementedError

class poppy.storage.base.services.**ServicesControllerBase** (*driver*)
Bases: *poppy.storage.base.controller.StorageControllerBase*

Services Controller Base definition.

create (*project_id*, *service_id*, *service_json*)
:param project_id :param service_id :param service_json :raise NotImplementedError

create_cert (*project_id*, *cert_obj*)
:param project_id :param cert_obj :raise NotImplementedError

delete (*project_id*, *service_id*)
:param project_id :param service_id :raise NotImplementedError

static format_result (*result*)
:param result :raise NotImplementedError

get()
:raise NotImplementedError

get_provider_details(*project_id*, *service_id*)
:param project_id :param service_id :raise NotImplementedError

list(*project_id*, *marker=None*, *limit=None*)
:param project_id :param marker :param limit :raise NotImplementedError

update(*project_id*, *service_id*, *service_json*)
:param project_id :param service_id :param service_json

:returns service_obj :raise NotImplementedError

update_cert_info(*domain_name*, *cert_type*, *flavor_id*, *cert_details*)
update_cert_info.

:param domain_name :param cert_type :param flavor_id :param cert_info

update_provider_details(*provider_details*)
:param provider_details :raise NotImplementedError

update_state(*project_id*, *service_id*, *state*)
Update service state

:param project_id :param service_id :param state :raise NotImplementedError

poppy.storage.cassandra driver

Cassandra Storage Driver for CDN Exports Cassandra storage controllers.

Field Mappings: In order to reduce the disk / memory space used, fields name will be, most of the time, the first letter of their long name. Fields mapping will be updated and documented in each controller class.

Cassandra storage driver implementation.

class poppy.storage.cassandra.driver.**CassandraStorageDriver**(*conf*)
Bases: *poppy.storage.base.driver.StorageDriverBase*

Cassandra Storage Driver.

change_config_group(*options*, *group*)

change_namespace(*namespace*)
change_namespace.

:param namespace

close_connection()
close_connection.

connect()
connect.

:returns connection

connection
Cassandra connection instance.

database
database.

:returns session

delete_namespace(*namespace*)
delete_namespace.

:param namespace

flavors_controller
flavors_controller.

:returns flavor controller

is_alive()
Health check for Cassandra.

services_controller
services_controller.

:returns service controller

storage_name
storage name.

:returns 'Cassandra'

class poppy.storage.cassandra.services.**ServicesController**(*driver*)
Bases: *poppy.storage.base.services.ServicesControllerBase*

Services Controller.

cert_already_exist(*domain_name*, *comparing_cert_type*, *comparing_flavor_id*, *comparing_project_id*)
Check if a cert with this domain name and type has already been created, or if the domain has been taken by other customers

:param domain_name :param comparing_cert_type :param comparing_flavor_id :param comparing_project_id

:returns Boolean if the cert with same type exists with another user.

create(*project_id*, *service_obj*)
create.

:param project_id :param service_obj

:raises ValueError

create_cert(*project_id*, *cert_obj*)

delete(*project_id*, *service_id*)
delete.

Archive local configuration storage

delete_cert(*project_id*, *domain_name*, *cert_type*)
Delete a certificate.

:param project_id :param domain_name :param cert_type

:raises ValueError

delete_provider_url(*provider_url*, *domain_name*)

delete_services_by_status(*project_id*, *service_id*, *status*)

domain_exists_elsewhere(*domain_name*, *service_id*)
Check if a service with this domain name has already been created.

:param domain_name :param service_id

:raises ValueError :returns Boolean if the service exists with another user.

static format_result (*result*)
format_result.

:param result :returns formatted result

get (*project_id*, *service_id*)
get.

:param project_id :param service_name

:returns result The requested service :raises ValueError

get_certs_by_domain (*domain_name*, *project_id=None*, *flavor_id=None*, *cert_type=None*)

get_certs_by_status (*status*)

get_domains_by_provider_url (*provider_url*)

get_provider_details (*project_id*, *service_id*)
get_provider_details.

:param project_id :param service_id :returns results Provider details

get_service_count (*project_id*)
Fetch Count of Services per project_id. :param project_id :returns count

get_service_details_by_domain_name (*domain_name*, *project_id=None*)
get_provider_details_by_domain_name.

:param domain_name :returns Provider details

get_service_limit (*project_id*)
Fetch Current limit on number of services per project_id.

:param project_id :raises ValueError :returns limit, if limit exists else default.

get_services_by_status (*status*)

list (*project_id*, *marker*, *limit*)
list.

:param project_id :param marker :param limit

:returns services

session
Get session.

:returns session

set_service_limit (*project_id*, *project_limit*)
Set Current limit on number of services per project_id.

:param project_id :param project_limit

set_service_provider_details (*project_id*, *service_id*, *status*)
Set current status on service_id under project_id.

:param project_id :param service_id

update (*project_id*, *service_id*, *service_obj*)
update.

:param project_id :param service_id :param service_obj

update_cert_info (*domain_name*, *cert_type*, *flavor_id*, *cert_details*)
update_cert_info.

:param domain_name :param cert_type :param flavor_id :param cert_info

update_provider_details (*project_id*, *service_id*, *provider_details*)
update_provider_details.

:param project_id :param service_id :param provider_details

update_state (*project_id*, *service_id*, *state*)
Update service state

:param project_id :param service_id :param state

:returns service_obj

poppy.storage.mongodb driver

poppy.storage.mockdb driver

Storage Driver for CDN Exports storage controllers.

Field Mappings: In order to reduce the disk / memory space used, fields name will be, most of the time, the first letter of their long name. Fields mapping will be updated and documented in each controller class.

Storage driver implementation.

class poppy.storage.mockdb.driver.**MockDBStorageDriver** (*conf*)
Bases: *poppy.storage.base.driver.StorageDriverBase*

close_connection ()

connect ()

connection
Connection instance.

database

flavors_controller

is_alive ()

services_controller

storage_name
For name.

class poppy.storage.mockdb.services.**ServicesController** (*driver*)
Bases: *poppy.storage.base.services.ServicesControllerBase*

create (*project_id*, *service_obj*)

create_cert (*project_id*, *cert_obj*)

delete (*project_id*, *service_id*)

delete_cert (*project_id*, *domain_name*, *cert_type*)

domain_exists_elsewhere (*domain_name*, *service_id*)

static format_result (*result*)

get (*project_id*, *service_id*)

get_certs_by_domain(*domain_name*, *project_id=None*, *flavor_id=None*, *cert_type=None*, *status=u'create_in_progress'*)

get_provider_details(*project_id*, *service_id*)

get_service_count(*project_id*)

get_service_details_by_domain_name(*domain_name*, *project_id=None*)

get_service_limit(*project_id*)

list(*project_id*, *marker=None*, *limit=None*)

session

set_service_limit(*project_id*, *project_limit*)

set_service_provider_details(*project_id*, *service_id*, *status*)

update(*project_id*, *service_id*, *service_json*)

update_cert_info(*domain_name*, *cert_type*, *flavor_id*, *cert_details*)

update_provider_details(*project_id*, *service_name*, *provider_details*)

update_state(*project_id*, *service_id*, *state*)

Update service state

:param project_id :param service_id :param state

:returns service_obj

4.1.5 poppy.provider extensions

poppy.provider.base base classes

class poppy.provider.base.driver.**ProviderDriverBase**(*conf*)

Bases: object

Interface definition for storage drivers.

Data plane storage drivers are responsible for implementing the core functionality of the system.

Connection information and driver-specific options are loaded from the config file.

Parameters **conf** (*oslo_config.ConfigOpts*) – Configuration containing options for this driver.

is_alive()

Check whether the storage is ready.

:raises NotImplementedError

provider_name

provider name.

:raises NotImplementedError

service_controller

Returns the driver's hostname controller.

:raises NotImplementedError

class poppy.provider.base.controller.**ProviderControllerBase**(*driver*)

Bases: object

Top-level class for controllers.

Parameters **driver** – Instance of the driver instantiating this controller.

class poppy.provider.base.responder.**Responder**(*provider_type*)

Bases: object

Responder Class.

created(*provider_service_id*, *links*, ***extras*)

created.

:param provider_service_id :param links :param **extras :returns provider msg{id, links}

deleted(*provider_service_id*)

deleted.

:param provider_service_id :returns provider msg{provider service id}

failed(*msg*)

failed.

:param msg :returns provider msg{msg, error details}

get(*domain_list*, *origin_list*, *cache_list*)

get.

:param domain_list :param origin_list :param cache_list :returns provider msg{domain, origins, caching}

purged(*provider_service_id*, *purge_url*)

purged.

:param provider_service_id :param purge_url :param hard :returns provider msg{provider service id, purge urls}

ssl_certificate_provisioned(*cert_domain*, *extra_info=None*)

ssl_certificate_provisioned.

:param cert_domain :param extra_info :returns provider msg{cert_domain, extra_info}

updated(*provider_service_id*, *links*, ***extras*)

updated.

:param provider_service_id :param links :param **extras :returns provider msg{provider service id}

class poppy.provider.base.services.**ServicesControllerBase**(*driver*)

Bases: *poppy.provider.base.controller.ProviderControllerBase*

Services Controller Base.

create(*service_name*, *service_obj*)

create.

:param service_name :param service_obj :raises NotImplementedError

current_customer()

Return the current customer for a provider.

This will needed call each provider's customer API, useful for certain providers (e.g fastly) and manage master-sub account.

:param service_name :raises NotImplementedError

delete(*project_id*, *provider_service_id*)

delete.

:param project_id :param provider_service_id :raises NotImplementedError

get (*service_name*)
Get details of the service, as stored by the provider.

:param service_name :raises NotImplementedError

get_metrics_by_domain (*project_id*, *domain_name*, *region*, ***extras*)
get analytics metrics by domain from provider

:param project_id :param domain_name :param regions :raises NotImplementedError

get_provider_service_id (*service_obj*)
Get the provider side service id for the service object.

:param service_obj :raises NotImplementedError

purge (*provider_service_id*, *hard=True*, *purge_url='/*'*)
purge.

:param provider_service_id :param purge_url :raises NotImplementedError

update (*provider_service_id*, *service_obj*)
update.

:raises NotImplementedError

poppy.provider.fastly extension

Fastly CDN Extension for CDN Exports Fastly CDN controllers.

Field Mappings: In order to reduce the disk / memory space used, fields name will be, most of the time, the first letter of their long name. Fields mapping will be updated and documented in each controller class.

Fastly CDN Provider implementation.

class poppy.provider.fastly.driver.**CDNProvider** (*conf*)
Bases: *poppy.provider.base.driver.ProviderDriverBase*

Fastly CNDProvider.

client
client to this provider.

:return client

is_alive ()
is_alive.

:return boolean

provider_name
provider name.

:return 'Fastly'

service_controller
Hook for service controller.

:return service controller

class poppy.provider.fastly.services.**ServiceController** (*driver*)
Bases: *poppy.provider.base.services.ServicesControllerBase*

Fastly Service Controller Class.

client

create (*service_obj*)

current_customer

delete (*project_id*, *provider_service_id*)

get (*service_name*)

get_metrics_by_domain (*project_id*, *domain_name*, *regions*, ***extras*)
Use Fastly's API to get the metrics by domain.

get_provider_service_id (*service_obj*)

purge (*service_id*, *hard=True*, *purge_url='/*'*)

update (*provider_service_id*, *service_obj*)

poppy.provider.mock extension

CDN Extension for CDN Exports Sample CDN controllers.

Field Mappings: In order to reduce the disk / memory space used, fields name will be, most of the time, the first letter of their long name. Fields mapping will be updated and documented in each controller class.

CDN Provider implementation.

class poppy.provider.mock.driver.**CDNProvider** (*conf*)
Bases: *poppy.provider.base.driver.ProviderDriverBase*

Mock CDNProvider.

is_alive ()
is_alive.

:return True

provider_name
provider name.

:return 'Mock'

service_controller
Hook for service controller.

:return service controller

class poppy.provider.mock.services.**ServiceController** (*driver*)
Bases: *poppy.provider.base.services.ServicesControllerBase*

Mock Service Controller.

create (*service_obj*)

current_customer
return current_customer for Mock. We can return a None.

delete (*project_id*, *provider_service_id*)

get (*service_name*)

get_metrics_by_domain (*project_id*, *domain_name*, *regions*, ***extras*)

get_provider_service_id (*service_obj*)

purge (*provider_service_id*, *hard=True*, *purge_url='/*'*)

update (*service_name*, *service_obj*)

CHAPTER 5

Using Poppy's API

b

m

p

s

t

A

B

C

H

I

L

M

N

O

P

R

S

T

U