

erocci

A scalable model-driven REST framework

Jean Parpaillon <jean.parpaillon@free.fr>



erocci REST API : a user point of view

Chapter 2. General API information – OpenStack Networking API v2.0 (neutron) Reference – API v2.0 (neutron) and extensions – Mozilla Firefox

docs.openstack.org/api/openstack-network/2.0/content/General_API_Information-d1e436.html

OpenStack

Chapter 2. General API information

CONTENTS SEARCH

OPENSTACK MANUALS > OPENSTACK NETWORKING API v2.0 (NEUTRON) REFERENCE > API v2.0 (NEUTRON) AND EXTENSIONS

GET /v2.0/networks.xml?limit=2&marker=71c1e68c-171a-4aa2-aca5-50ea153a3718 HTTP/1.1
Host: 127.0.0.1:9696
Content-Type: application/xml
Accept: application/xml

Example 2.8. Network collection, last page: XML response

```
1 <?xml version="1.0" ?>
2 <networks xmlns="http://openstack.org/neutron/api/v2.0" xmlns:atom="http://www.w3.org/2005/Atom" xmlns:provider="http://docs.openstack.org/ext/pi
3 <network>
4   <status>ACTIVE</status>
5   <subnets neutron:type="list" />
6   <name>net1</name>
7   <provider:physical_network>physnet1</provider:physical_network>
8   <admin_state_up neutron:type="bool">True</admin_state_up>
9   <tenant_id>c05140b3dc7c4555aff9fab6b58dc2c</tenant_id>
10  <provider:network_type>vlan</provider:network_type>
11  <router:external neutron:type="bool">False</router:external>
12  <shared neutron:type="bool">False</shared>
13  <id>b3680498-03da-4691-896f-ef9ee1d856a7</id>
14  <provider:segmentation_id neutron:type="long">1000</provider:segmentation_id>
15 </network>
16 <atom:link href="http://127.0.0.1:9696/v2.0/networks.xml?limit=2&marker=b3680498-03da-4691-896f-ef9ee1d856a7&page_reverse=True"
17 </networks>
```

Sorting

You can use the `sort_key` and `sort_dir` parameters to sort the results of list operations. Currently sorting does not work with extended attributes of resource. The `sort_key` and `sort_dir` can be repeated, and the number of `sort_key` and `sort_dir` provided must be same. The `sort_dir` parameter indicates in which direction to sort. Acceptable values are `asc` (ascending) and `desc` (descending).

Sorting is optional feature of OpenStack Networking API, and it might be disabled. If sorting is disabled, the sorting parameters are ignored.

If a particular plug-in does not support sorting operations and sorting is enabled, the Networking API v2.0 emulates the sorting behavior so that users can expect the same behavior regardless of the particular plug-in that runs in the background.

Unfortunately OpenStack Networking does not provide a mechanism to tell users if specific plug-ins support or have enabled sorting.

Extensions

The Networking API v2.0 is extensible.

The purpose of Networking API v2.0 extensions is to:

- Introduce new features in the API without requiring a version change.

- Data format
 - JSON format
 - XML format
 - Scalar data types (string, integer, float, etc)
- Data access protocol
 - Filtering
 - Sorting
 - Pagination
 - Authnz (tokens, etc), return codes, etc.
- Finally...
 - Application level : data types (structures, relations, etc)



■ Data format

- JSON format
- XML format
- Scalar data types (string, integer, float, etc)

Schémas ?

■ Data access protocol

- Filtering
- Sorting
- Pagination
- Authnz (tokens, etc), return codes, etc.

One to rule them all ?

■ Finally...

- Application level : data types (structures, relations, etc)

A single meta-model ?





Let's speak about a standard ?

■ A standard

- Is not a catalog of schemas
- Is not a constraint for developers
- Must be implemented (implementable)

■ A good standard

- Should allow to concentrate on the design, not the implementation details
- Allow to build an ecosystem
- Allow to accelerate the development, not the contrary



- OCCI is typed
 - A resource is an instance of a kind
 - A kind is a named list of typed attributes and actions
 - e.g. : compute : # cores (integer), RAM (float), etc
 - Attribute : name, type, default value, mutability, etc.
 - Action : an invocable operation on a resource
 - Kinds are inheritable



- OCCI is extensible
 - Resource can be associated with mixins
 - Mixin: a named set of additional attributes and actions
 - e.g. : IPNetworkInterface adds IP, netmask, etc to a network interface
 - User Mixin : aka « tags »
 - e.g. : http://example.com/occi/mixins#my_project1



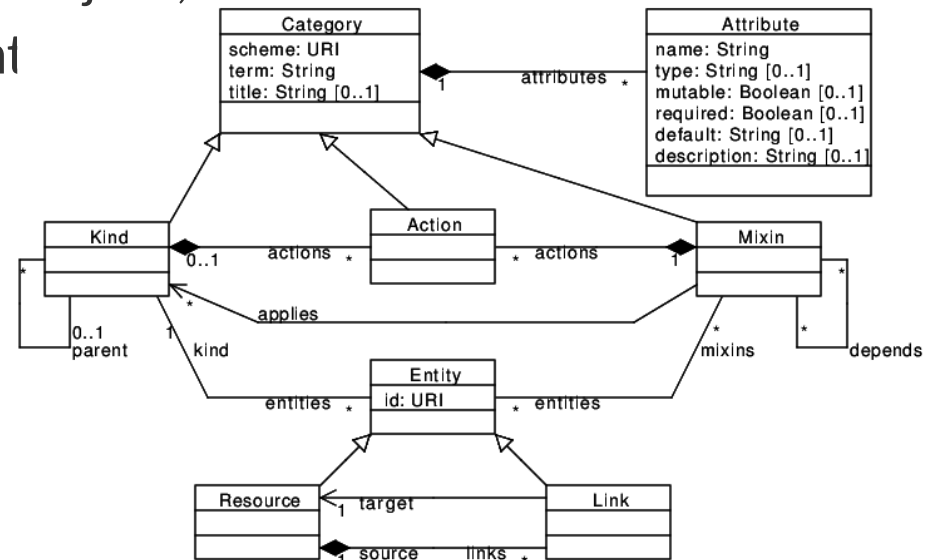
- OCCI is relational
 - Links are resources with additional attributes
 - `occi.core.source` : resource URI
 - `occi.core.target` : URI
 - A bounded collection is a list of resources of the same kind/mixin :
 - e.g. : `GET /collections/compute/`
 - An unbounded collection is a list of resources with same prefix :
 - e.g. : `GET /myresources/`



- OCCI is self-described
 - GET /-/ (capabilities)
 - List of supported kinds
 - List of supported mixins
 - Capabilities returns collection's URL
 - <http://schemas.org/occi/infrastructure#compute> →
/collections/compute/



- OCCI is meta-model based
 - Model consistency checking
 - Automatic model implementation
 - Rendering independant
 - text/plain, application/json, ...
 - Transport independant
 - HTTP, ...



■ Framework

- rOCCI (ruby)
- occi4java (not maintained)

■ Specific implementations

- CompatibleOne
- PyOCNI
- OpenStack, OpenNebula, etc

■ Tools

- DoYouSpeakOcci : tests
- Monitoring (Intel)

■ Limits

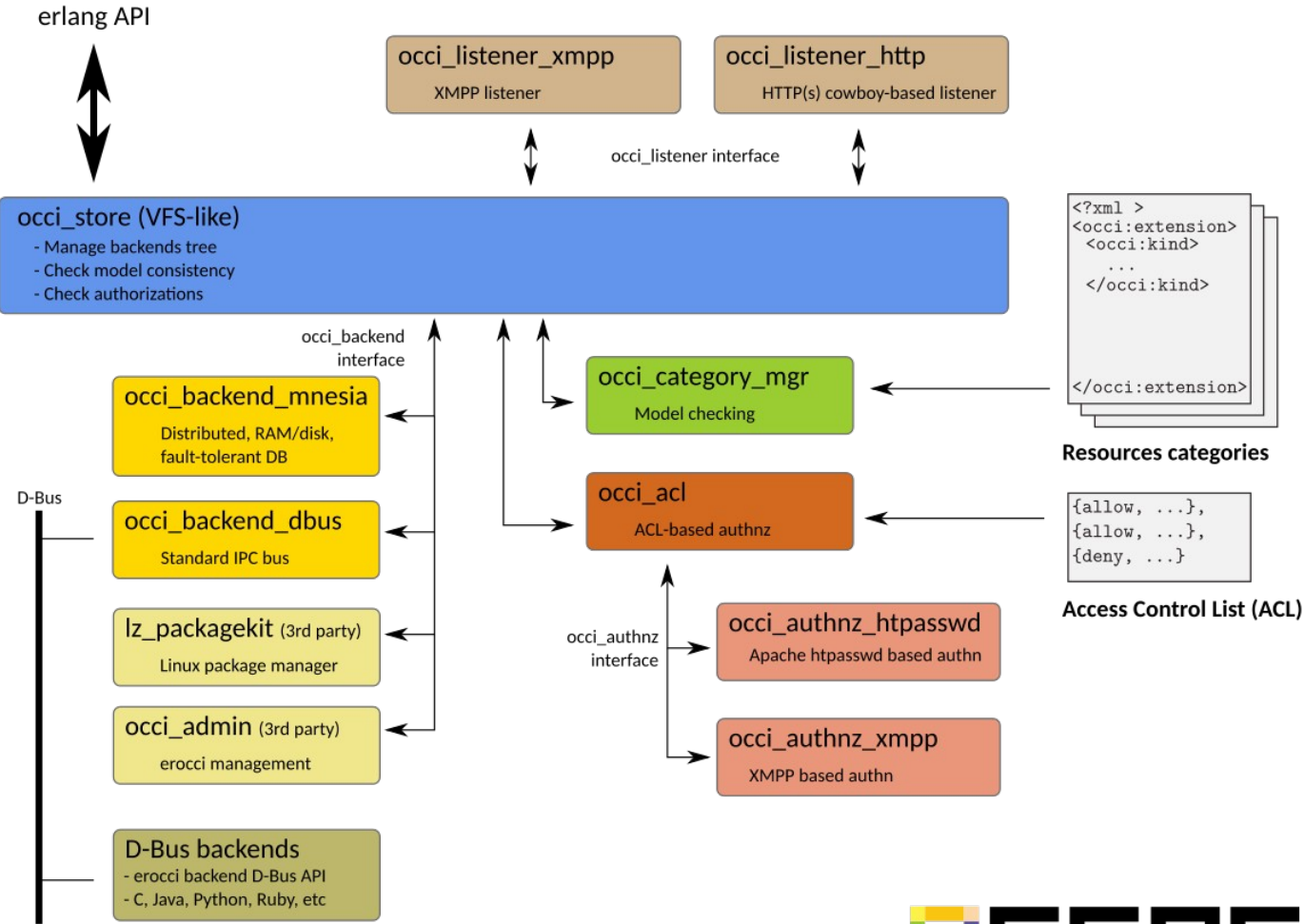
- Models are language/extensions/lib dependant
- See <http://occi-wg/community/implementations/>



Why erocci ?

- Generic : a framework to build OCCI applications
- Less code, more design
- Small, tested, reliable
- Easy to deploy
- Scalable
 - Runs embedded in connected objects (IoT)
 - Scales out to serve millions of users
- Extensible

NETWORK PROTOCOLS



APPLICATION MODEL

DATA SOURCES



- Only OCCI meta-model is hard-coded
- Everything is pluggable :
 - Authentication mechanism
 - Transport protocol (HTTP, XMPP, etc)
 - Data sources (backends)
- Allow composition of services by plugging data sources (backends) at runtime



- A schema for resources' categories (kinds, mixins)
- ACL-based authorizations



- Small
 - Around 12k lines
- Tested
 - Continuous integration
- Reliable
 - Erlang cowboy web server (LeoFS, MongooseIM, etc)
 - erlang/OTP platform provides fault tolerance
 - Process supervision

- All included erlang/OTP platform
- Self contained : no web frontend
- Just try it !



- Thanks to erlang/OTP
 - Runs on RaspberryPI (~ 100 req/sec)
 - Scales out to a full cluster



- Pluggable mechanisms
- For those who don't like erlang
 - D-Bus backend API (experimental)



(1) use case : a REST backend for blog

- No need for ORM, web server, database, templating, etc.
- Design your model
 - Blog entry kind
 - User kind
 - Contribution kind (link between a user and an entry)
- Choose a storage backend
- Choose a transport (HTTP?)
- Defines ACL
- And that's all



(2) use case : API adapter

- Backend API : write a new data source for your API
 - Erlang API
 - D-Bus API (any language)
 - In any case, implements CRUD operations
- Choose a transport
- Enjoy scalability, supervision, transports, ACL, etc.



(3) use case : a protocol adapter

- You have a resource oriented protocol and want to expose it through HTTP/OCCI
- Use `occi_store` for storing your data
- `erocci` deals with storage, rendering, scalability, ...



■ Authn/authz

- x509 based authn
- oauth

■ Testing

- Improve functional tests
- Improve unit testing

■ occi_store

- Optimizations
- Integrate authnz, pagination (wip)

■ XMPP

- Enable pubsub (notifications)

■ New storage backends

- Riak (wip)
- ODBC (SQL)
- ...

- (1) consortium
 - Research, industry, standardization
- (3) pillars
 - Theory : proven meta-model (OCCL)
 - Design : Eclipse-based tools
 - Runtime : erocci, ...
- (*) use cases
 - IaaS, PaaS, SaaS, IoT, ...
 - Every RESTful services



- Website (WIP)
 - <http://erocci.ow2.org/>
- Source code
 - <http://github.com/erocci/erocci>
- Continuous integration
 - <https://travis-ci.org/jeanparpaillon/erocci>
- Mailing lists
 - erocci-dev@ow2.org / erocci-info@ow2.org



Questions ?

