Chameleon Cloud Hammers Documentation

Release 0.11

Nick Timkovich

May 05, 2020

Contents:

Percussive maintenance

The "hammers" repo is an ad-hoc collection of tools that Nick used to automate fixes to the infrastructure, do some investigations for reported problems, and make common tasks quicker.

CHAPTER 1

Management Scripts ("Hammers")

These are the collection of scripts used to fix inconsistencies in the various OpenStack services. Not particularly bright tools, but good for a first-pass.

The tools are run in a Python virtualenv to avoid package clashes with the system Python. On the CHI@TACC and CHI@UC controllers, they live at `/root/scripts/hammers/venv`. The scripts can be called directly without any path shenanigans by providing the full path, e.g. `/root/scripts/hammers/venv/bin/conflict-macs info`, and that is how the cronjobs do it.

1.1 Setup

As mentioned in the intro, the scripts are run in a virtualenv. Here's how to set it up:

1. Get code

```
mkdir -p /root/scripts/hammers
cd /root/scripts/hammers
git clone https://github.com/ChameleonCloud/hammers.git
```

2. Create environment

```
virtualenv /root/scripts/hammers/venv
/root/scripts/hammers/venv/bin/pip install -r /root/scripts/hammers/hammers/
↪requirements.txt
/root/scripts/hammers/venv/bin/pip install -e /root/scripts/hammers/hammers
```

Note: Because the hammers repo was installed with `-e`, some updates in the future can be done by `cd`-ing into the directory and `git pull`-ing. Updates that change script entrypoints in `setup.py` will require a quick `pip install...`

3. Set up credentials for OpenStack and Slack

The *Puppet cronjobs* have a configuration variable that points to the OS shell var file, for instance `/root/adminrc`.

There is also a file for Slack vars, e.g. `/root/scripts/slack.json`. It is a JSON with a root key `"webhook"` that is a URL (keep secret!) to post to and another root key `"hostname_name"` that is a mapping of FQDNs to pretty names.

Example:

```
{
    "webhook": "https://hooks.slack.com/services/...super-seekrit...",
    "hostname_names": {
        "m01-07.chameleon.tacc.utexas.edu": "CHI@TACC",
        "m01-03.chameleon.tacc.utexas.edu": "KVM@TACC",
        "admin01.uc.chameleoncloud.org": "CHI@UC"
    }
}
```

1.2 Running

You can either `source venv/bin/activate` the virtualenv to put the scripts into the path, or directly execute them out of the directory, `venv/bin/neutron-reaper`

Common Options:

- `--slack <json-options>` - if provided, used to post notifications to Slack
- `--osrc <rc-file>` - alternate way to feed in the OS authentication vars

1.3 Script Descriptions

1.3.1 Neutron Resource "Reaper"

```
neutron-reaper {info, delete}                    {ip, port}                    <grace-
↪days>                    [ --dbversion ocata ]
```

Reclaims idle floating IPs and cleans up stale ports.

Required arguments, in order:

- `info` to just display what would be cleaned up, or actually clean it up with `delete`.
- Consider floating `ip`'s or `port`'s
- A project needs to be idle for `grace-days` days.

Optional arguments:

- `--dbversion ocata` needed for the Ocata release as the database schema changed slightly.

1.3.2 Conflicting Ironic/Neutron MACs

```
conflict-macs          {info, delete}          ( --ignore-from-ironic-config <path to␣
↪ironic.conf> |
      --ignore-subnet <subnet UUID> )
```

The Ironic subnet must be provided—directly via ID or determined from a config—otherwise the script would think that they are in conflict.

1.3.3 Undead Instances

Sometimes Nova doesn't seem to tell Ironic the instance went away on a node, then the next time it deploys to the same node, Ironic fails.

```
undead-instances {info, delete}
```

Running with `info` displays what it thinks is wrong, and with `delete` will clear the offending state from the nodes.

1.3.4 Ironic Node Error Resetter

Basic Usage:

```
ironic-error-resetter {info, reset}
```

Resets Ironic nodes in error state with a known, common error. Started out looking for IPMI-related errors, but isn't intrinsically specific to them over any other error that shows up on the nodes. Records the resets it performs on the node metadata (`extra` field) and refuses after some number (currently 3) of accumulated resets.

Currently watches out for:

```
^Failed to tear down\. Error: Failed to set node power state to power off\.
^Failed to tear down\. Error: IPMI call failed: power status\.
(?s)^Failed to tear down. Error: Unable to clear binding profile for neutron port [0-
↪9a-f-]+\. Error:.+?502 Proxy Error
^During sync_power_state, max retries exceeded for node [0-9a-f-]+, node state␣
↪(None|power (on|off)) does not match expected state 'power (on|off)'\.
```

1.3.5 Dirty Ports

Basic Usage:

```
dirty-ports {info, clean}
```

There was/is an issue where a non-empty value in an Ironic node's port's `internal_info` field would cause a new instance to fail deployment on the node. This notifies (`info`) or `clean`s up if there is info on said ports on nodes that are in the "available" state.

1.3.6 Orphan Resource Providers

```
orphan-resource-providers {info, update}
```

Occasionally, compute nodes are recreated in the Nova database with new UUIDs, but resource providers in the Placement API database are not updated and still refer to the old UUIDs. This causes failures to post allocations and results in errors when launching instances. This detects the issue (`info`) and fixes it (`update`) by updating the `uuid` field of resource providers.

1.3.7 Curiouser

Note: Not well-tested, may be slightly buggy with Chameleon phase 2 updates.

```
curiouser
```

Displays Ironic nodes that are in an error state, but not in maintenance. The Ironic Error Resetter can fix some error states automatically.

1.3.8 Metadata Sync

Synchronizes the metadata contained in the G5K API to Blazar's "extra capabilities". Keys not in Blazar are created, those not in G5K are deleted.

If using the soft removal option, you could follow up with a manual query to clean up the empty strings:

```
DELETE FROM blazar.computehost_extra_capabilities WHERE capability_value='';
```

1.3.9 GPU Resource "Lease Stacking"

1.4 Puppet Directives

Add cronjob(s) to Puppet. These expect that the above *setup* is already done.

```
$slack_json_loc = '/root/scripts/slack.json'
$osrc_loc = '/root/adminrc'
$venv_bin = '/root/scripts/hammers/venv/bin'

cron { 'hammers-neutronreaper-ip':
    command => "$venv_bin/neutron-reaper delete ip 14 --dbversion ocata --slack
↪$slack_json_loc --osrc $osrc_loc 2>&1 | /usr/bin/logger -t hammers-neutronreaper-ip
↪",
    user => 'root',
    hour => 5,
    minute => 20,
}
cron { 'hammers-ironicerrorresetter':
    command => "$venv_bin/ironic-error-resetter info --slack $slack_json_loc --osrc
↪$osrc_loc 2>&1 | /usr/bin/logger -t hammers-ironicerrorresetter",
    user => 'root',
    hour => 5,
    minute => 25,
}
cron { 'hammers-conflictmacs':
    command => "$venv_bin/conflict-macs info --slack $slack_json_loc --osrc $osrc_loc␣
↪--ignore-from-ironic-conf /etc/ironic/ironic.conf 2>&1 | /usr/bin/logger -t hammers-
↪conflictmacs",
    user => 'root',
    hour => 5,
    minute => 30,
}
```

(continues on next page)

(continued from previous page)

```
cron { 'hammers-undeadinstances':
    command => "$venv_bin/undead-instances info --slack $slack_json_loc --osrc $osrc_
↪loc 2>&1 | /usr/bin/logger -t hammers-undeadinstances",
    user => 'root',
    hour => 5,
    minute => 35,
}
cron { 'hammers-orphanresourceproviders':
  command => "$venv_bin/orphan-resource-providers info --slack $slack_json_loc 2>&1 |␣
↪/usr/bin/logger -t hammers-orphanresourceproviders",
  user => 'root',
  hour => 5,
  minute => 40,
}
cron { 'hammers-gpuleasestacking':
  command => "$venv_bin/lease-stack-reaper delete --slack $slack_json_loc 2>&1 | /usr/
↪bin/logger -t hammers-leasestacking",
  user => 'root',
  hour => 5,
  minute => 40,
}
```

CHAPTER 2

Client Tools ("ccmanage")

These are the collection of modules and scripts used to perform common tasks like launching a single node. They generally use the Python APIs, so are kept separate in the `ccmanage` package.

Because they use the Python APIs, they may be sensitive to the versions installed; consult the `requirements.txt` file for what's known to work. Notably, the Blazar client comes from a repo rather than PyPI, so may be a bit volatile.

These APIs are also used by Abracadabra, the automated Chameleon appliance builder thing.

2.1 Quicknode

The quicknode script creates a 24-hour lease, launches an instance on it, then binds a floating IP to it. One command, a 10 to 15 minute wait, and you can SSH in to your very own bare metal node.

`ccmanage.quicknode.`**`main`**(*argv=None*)

This script must be run as a module using `python -m ccmanage.quicknode`. In the future it could be configured as an entry point in `setup.py` and installed like the hammers scripts.

```
$ python -m ccmanage.quicknode --help
usage: quicknode.py [-h] [--osrc OSRC] [--node-type NODE_TYPE]
                    [--key-name KEY_NAME] [--image IMAGE] [--no-clean]
                    [--net-name NET_NAME] [--no-floatingip]

Fire up a single node on Chameleon to do something with.

optional arguments:
-h, --help            show this help message and exit
--osrc OSRC           OpenStack parameters file that overrides envvars.
                        (default: None)
--node-type NODE_TYPE
                        Node type to launch. May be custom or likely one of:
                        'compute_skylake', 'gpu_p100',
                        'gpu_p100_nvlink', 'gpu_k80', 'gpu_m40',
```

(continues on next page)

(continued from previous page)

```
                        'compute_haswell_ib', 'storage', 'atom',
                        'compute_haswell', 'storage_hierarchy', 'arm64',
                        'fpga', 'lowpower_xeon' (default: compute_haswell)
--key-name KEY_NAME   SSH keypair name on OS used to create an instance.
                        Must exist in Nova (default: default)
--image IMAGE         Name or ID of image to launch. (default: CC-CentOS7)
--no-clean            Do not clean up on failure. (default: False)
--net-name NET_NAME   Name of network to connect to. (default: sharednet1)
--no-floatingip       Skip assigning a floating IP. (default: False)
```

It can either read the environment variables (i.e. you did a `source osvars.rc`) or be given a file with them—including the password—in it (`--osrc`). There must be a key pair loaded into Nova that matches the option for `--key-name`. A basic run:

```
$ python -m ccmanage.quicknode --image CC-CentOS7
Lease: creating...started <Lease 'lease-JTCMZOKMHE' on chi.uc.chameleoncloud.org␣
↪(ad67ccb1-edeb-462b-a9b3-83727578b937)>
Server: creating...building...started <Server 'instance-KYJCM5N55A' on chi.uc.
↪chameleoncloud.org (36d52a0d-428d-45a8-88fb-232898aff0cb)>...bound ip 192.5.87.37␣
↪to server.

'ssh cc@192.5.87.37' available.
Press enter to terminate lease and server.
```

It attempts to remove the lease after hitting enter, the instance is deleted along with it.

The main function is also a handy reference for how the other objects in `ccmanage` work, specifically *ccmanage.lease.Lease* and *ccmanage.server.Server* (created in a factory method of *Lease*)

2.2 Authentication

Generate "real" Keystone auth objects versus the DIY methods like in *hammers.osapi*

`ccmanage.auth.`**`add_arguments`**(*parser*)
: Inject our args into the user's `ArgumentParser` *parser*. The resulting argument namespace can be inspected by *session_from_args()*.

`ccmanage.auth.`**`auth_from_rc`**(*rc*)
: Generates a Keystone Auth object from an OS parameter dictionary. Dict key format is the same as environment variables (`OS_AUTH_URL`, et al.)

  We do some dumb gymnastics because everything expects the parameters in their own cap/delim format:

  - envvar name: `OS_AUTH_URL`
  - loader option name: `auth-url`
  - loader argument name: `auth_url`

`ccmanage.auth.`**`session_from_vars`**(*os_vars*)
: Generates a `keystoneauth1.session.Session` object from an OS parameter dictionary akin to *auth_from_rc()*. This one is generally more useful as the session object can be used directly with most clients:

```
>>> from novaclient.client import Client as NovaClient
>>> from ccmanage.auth import session_from_vars
```

(continues on next page)

(continued from previous page)

```
>>> session = session_from_vars({'OS_AUTH_URL': ...})
>>> nova = NovaClient('2', session=session)
```

ccmanage.auth.**session_from_args**(*args=None*, *rc=False*)

Combine the `osrc` attribute in the namespace *args* (if provided) with the environment vars and produce a Keystone session for use by clients.

Optionally return the RC dictionary with the OS vars used to construct the session as the second value in a 2-tuple if *rc* is true.

2.3 Leases

Lease management

class ccmanage.lease.**Lease**(*keystone_session*, *, *sequester=False*, *_no_clean=False*, ***lease_kwargs*)

Creates and manages a lease, optionally with a context manager (`with`).

```
with Lease(session, node_type='compute_haswell') as lease:
    instance = lease.create_server()
    ...
```

When using the context manager, on entering it will wait for the lease to launch, then on exiting it will delete the lease, which in-turn also deletes the instances launched with it.

Parameters

- **keystone_session** – session object
- **sequester** (*bool*) – If the context manager catches that an instance failed to start, it will not delete the lease, but rather extend it and rename it with the ID of the instance that failed.
- **_no_clean** (*bool*) – Don't delete the lease at the end of a context manager
- **lease_kwargs** – Parameters passed through to *lease_create_nodetype()* and in turn *lease_create_args()*

create_server(**server_args*, ***server_kwargs*)

Generates instances using the resource of the lease. Arguments are passed to *ccmanage.server.Server* and returns same object.

delete()

Deletes the lease

classmethod from_existing(*keystone_session*, *id*)

Attach to an existing lease by ID. When using in conjunction with the context manager, it will *not* delete the lease at the end.

ready

Returns True if the lease has started.

refresh()

Updates the lease data

status

Refreshes and returns the status of the lease.

wait()

Blocks for up to 150 seconds, waiting for the lease to be ready. Raises a RuntimeError if it times out.

ccmanage.lease.**lease_create_nodetype**(**args*, ***, *node_type*, ***kwargs*)

Wrapper for *lease_create_args()* that adds the `resource_properties` payload to specify node type.

Parameters **node_type** (*str*) – Node type to filter by, `compute_haswell`, et al.

Raises **ValueError** – if there is no *node_type* named argument.

ccmanage.lease.**lease_create_args**(*name=None*, *start='now'*, *length=None*, *end=None*, *nodes=1*, *resource_properties=''*)

Generates the nested object that needs to be sent to the Blazar client to create the lease. Provides useful defaults for Chameleon.

Parameters

- **name** (*str*) – name of lease. If `None`, generates a random name.
- **start** (*str/datetime*) – when to start lease as a `datetime.datetime` object, or if the string `'now'`, starts in about a minute.
- **length** – length of time as a `datetime.timedelta` object or number of seconds as a number. Defaults to 1 day.
- **end** (*datetime.datetime*) – when to end the lease. Provide only this or *length*, not both.
- **nodes** (*int*) – number of nodes to reserve.
- **resource_properties** – object that is JSON-encoded and sent as the `resource_properties` value to Blazar. Commonly used to specify node types.

2.4 Servers

class ccmanage.server.**Server**(*session*, *id=None*, *lease=None*, *key='default'*, *image='CC-CentOS7'*, *net_ids=None*, *net_name=None*, ***extra*)

Launches an instance on a lease.

CHAPTER 3

HTTP Helpers

These are DIY shims that access OpenStack services' APIs without requiring much more than Python requests. The objects they return are not intelligent, but are simple lists and dictionaries.

3.1 Auth Management

Tools to convert credentials into authentication and authorization in raw Python, as opposed to the Python APIs provided by `keystoneauth1` or the like. They were largely created out of frustration with the apparent moving target and inconsistencies of the OS client APIs, which was also exacerbated by the Blazar client being fairly nacent.

Compare and contrast

ccmanage.auth which *does* use the Python APIs.

`hammers.osapi.`**`add_arguments`**(*parser*)

Inject an arg into the user's parser. Intented to pair with *Auth.from_env_or_args()*: after argparse parses the args, feed that the args namespace.

`hammers.osapi.`**`load_osrc`**(*fn*, *get_pass=False*)

Parse a Bash RC file dumped out by the dashboard to a dict. Used to load the file specified by *add_arguments()*.

`class` `hammers.osapi.`**`Auth`**(*rc*)

The Auth object consumes credentials and provides tokens and endpoints. Create either directly by providing a mapping with the keys in `required_os_vars` or via the *Auth.from_env_or_args()* method.

`classmethod from_env_or_args`(*, *args=None*, *env=True*)

Loads the RC values from the file in the provided *args* namespace, falling back to the environment vars if *env* is true. *add_arguments()* is a helper function that will add the "osrc" argument to an argparse parser.

Returns an Auth object that's ready for use.

authenticate()
Authenticate with Keystone to get a token and endpoint listing

endpoint(*type*)
Find the endpoint for a given service *type*. Examples include `compute` for Nova, `reservation` for Blazar, or `image` for Glance.

token
Read-only property that returns an active token, reauthenticating if it has expired. Most services accept this in the HTTP request header under the key `X-Auth-Token`.

3.2 Service API Wrappers

For the below, the *auth* argument is an instance of *hammers.osapi.Auth*.

3.2.1 Blazar (Reservations)

hammers.osrest.blazar.**host**(*auth*, *host_id*)
Retrieves host by ID.

hammers.osrest.blazar.**hosts**(*auth*)
Retrieves all hosts, returning a dictionary keyed by ID.

hammers.osrest.blazar.**host_update**(*auth*, *host_id*, *values_payload*)
Updates host data

hammers.osrest.blazar.**lease**(*auth*, *lease_id*)
Retrieves a lease by ID

hammers.osrest.blazar.**leases**(*auth*)
Retrieves all leases, returning a dictionary keyed by ID

hammers.osrest.blazar.**lease_delete**(*auth*, *lease_id*)
Deletes a lease by ID

3.2.2 Glance (Image Store)

Glance API shims. See Glance HTTP API

hammers.osrest.glance.**image_properties**(*auth*, *image_id*, ***, *add=None*, *remove=None*, *replace=None*)
Add/remove/replace properties on the image. Some standard properties can be modified (name, visibility), some can't (id, checksum), but custom fields can be whatever.

Parameters

- **add** (*mapping*) – properties to add
- **remove** (*iterable*) – properties to delete
- **replace** (*mapping*) – properties to replace by key

hammers.osrest.glance.**image_create**(*auth*, *name*, ***, *disk_format='qcow2'*, *container_format='bare'*, *visibility='private'*, *extra=None*)
Creates empty image entry, ready to be filled by an upload command.

If provided, *extra* is a mapping to set custom properties.

hammers.osrest.glance.**image**(*auth*, *id=None*, *name=None*)
Looks up image by *id* or *name*. If *name* is not unique given the scope of authentication (e.g. private image owned by someone else may be hidden), an error is raised.

hammers.osrest.glance.**images**(*auth*, *query=None*)
Retrieves all images, filtered by *query*, if provided.

Doesn't support pagination. Don't request too many.

For querying, accepts a dictionary. If the value is a non-string iterable, the key is repeated in the query with each element in the iterable.

hammers.osrest.glance.**image_delete**(*auth*, *id*)
Deletes image by ID

hammers.osrest.glance.**image_upload_curl**(*auth*, *id*, *filepath*)
Generates an cURL command to upload an image file at *filepath* to be associated with the Glance image. Includes authentication header, so is stateless and can be run most anywhere.

hammers.osrest.glance.**image_download_curl**(*auth*, *id*, *filepath=None*)
Generates a cURL command to download an image file to *filepath*. If *filepath* is not provided, dumps to `~/<image name>.img`. The request is authenticated, so can be run most anywhere.

3.2.3 Ironic (Bare Metal)

Shims for Ironic. See Ironic HTTP API Docs.

hammers.osrest.ironic.**node_update**(*auth*, *node*, * *add=None*, *remove=None*, *replace=None*)
Add/remove/replace properties on the node.

Parameters

- **add** (*mapping*) – properties to add
- **remove** (*iterable*) – properties to delete
- **replace** (*mapping*) – properties to replace by key

hammers.osrest.ironic.**node**(*auth*, *node*)
Get node by ID or the `uuid` key out of a dictionary.

hammers.osrest.ironic.**nodes**(*auth*, *details=False*)
Retrieves all nodes, with more info if *details* is true.

hammers.osrest.ironic.**node_set_state**(*auth*, *node*, *state*)
Set provision state of *node* to *state*.

See also:

Ironic's State Machine

hammers.osrest.ironic.**ports**(*auth*)
Retrieves all Ironic ports, returns a dictionary keyed by the port ID

3.2.4 Keystone (Authentication)

Keystone API shims. Requires v3 API. See Keystone HTTP API

hammers.osrest.keystone.**project**(*auth*, *id*)
Retrieve project by ID

hammers.osrest.keystone.**projects**(*auth*, ***params*)
Retrieve multiple projects, optionally filtered by *params*. Keyed by ID.

Example params: `name`, `enabled`, or stuff from https://developer.openstack.org/api-ref/identity/v3/?expanded=list-projects-detail#list-projects

hammers.osrest.keystone.**project_lookup**(*auth*, *name_or_id*)
Tries to find a single project by name or ID. Raises an error if none or multiple projects found.

hammers.osrest.keystone.**user**(*auth*, *id*)
Retrieves information about a user by ID

hammers.osrest.keystone.**users**(*auth*, *enabled=None*, *name=None*)
Retrieves multiple users, optionally filtered.

hammers.osrest.keystone.**user_lookup**(*auth*, *name_or_id*)
Tries to find a single user by name or ID. Raises an error if none or multiple users are found.

3.2.5 Neutron (Networking)

hammers.osrest.neutron.**floatingips**(*auth*)
Get all floating IPs, returns a dictionary keyed by ID.

hammers.osrest.neutron.**floatingip_delete**(*auth*, *floatingip*)
Frees a floating IP by ID

hammers.osrest.neutron.**network**(*auth*, *net*)
Gets a network by ID, or mapping containing an `'id'` key.

hammers.osrest.neutron.**networks**(*auth*)
Get all networks. Returns dictionary keyed by ID.

hammers.osrest.neutron.**ports**(*auth*)
Get all ports. Returns a dictionary keyed by port ID.

hammers.osrest.neutron.**port_delete**(*auth*, *port*)
Deletes a port by ID, or mapping containing an `'id'` key.

hammers.osrest.neutron.**subnet**(*auth*, *subnet*)
Get subnet info. Accepts ID or mapping containing an `'id'` key.

hammers.osrest.neutron.**subnets**(*auth*)
Get all subnets.

3.2.6 Nova (Compute)

CHAPTER 4

Direct Database Access

Some methods, properties are not fully exposed via the APIs, or are extremely slow or difficult to retrieve. Direct database access can be used with an abundance of caution and the caveat that it's not guaranteed to work in future releases without modification.

4.1 Credentials and Connecting

4.1.1 Credential Configuration

class hammers.mycnf.**MyCnf**(*paths=None*)

MySQL configuration fetcher. Attempts to emulate the behavior of the MySQL client to the best of its ability, falling through multiple locations where configuration could exist to determine its value.

See also:

- MySQL 5.7 Reference Manual, 4.2.6 Using Option Files
- Configuring MariaDB with my.cnf

class hammers.mysqlargs.**MySqlArgs**(*defaults*, *mycnfpaths=None*)

Argument manager that combines command-line arguments with configuration files to determine MySQL connection info including the username, password, hostname, and port.

The *defaults* provided take the lowest priority. If any value is found among the configuration files with a higher priority, it overrides it. The key names used are `user`, `password`, `host`, and `port`.

connect()

Uses the prepared connection arguments and creates a *hammers.mysqlshim.MySqlShim* object that connects to the database.

extract(*args*)

Parses the arguments in the namespace returned by `argparse.ArgumentParser.parse_args()` to generate the final set of connection arguments.

inject(*parser*)
Adds arguments to a `argparse.ArgumentParser`.

- `-u/--db-user`
- `-p/--password`
- `-H/--host`
- `-P/--port`
- `--service-conf`: A configuration file like `/etc/ironic/ironic.conf` that contains a database connection string.

4.1.2 Connecting

class hammers.mysqlshim.**MySqlShim**(***connect_args*)
Connection manager and query executor. `connect_args` is passed directly to `MySQLdb.connect()`

This class provides some quality-of-life stuff like providing column names and emitting dictionaries for rows.

columns()
Returns column names from the most recent query.

query(**cargs*, ***ckwargs*)
Parameters not listed are passed into the `MySQLdb.cursors.Cursor`'s execute function. One that is common would be `args` for parameterized queries.

Parameters

- **no_rows** (*bool*) – Executes the query and returns the number of rows updated. Very likely what you want for anything that modifies the database or else you may not complete the transaction.
- **immediate** (*bool*) – If true, immediately runs the query and puts it into a list. Otherwise, an iterator is returned.

4.2 Queries

hammers.query.**query**(*q*)
Decorator to include all the queries into a dictionary

hammers.query.**project_col**(*version*)
The name of the column changed somewhere between L and O.

Should be pretty basic to avoid any SQL injection.

hammers.query.**idle_projects**(*db*)
Returns rows enumerating all projects that are currently idle (number of running instances = 0). Also provides since when the project has been idle (when the latest running instance was deleted)

There may be NULLs emitted for "latest_deletion" if a project hasn't ever had an instance (like an admin project...).

hammers.query.**latest_instance_interaction**(*db*, *kvm*, *nova_db_name='nova'*)
Get the latest interaction date with instances on the target database name. Combine as you so desire.

hammers.query.**owned_ips**(*db*, *project_ids*)
Return all IPs associated with *project_ids*

Maria 5.5 in production doesn't seem to like this, but works fine with a local MySQL 5.7. Is it Maria? 5.5? Too many? See owned_ip_single for one that works, but need to call multiple times.

hammers.query.**future_reservations**(*db*)
Get project IDs with lease end dates in the future that haven't been deleted. This will also grab *active* leases, but that's erring on the safe side.

hammers.query.**clear_ironic_port_internalinfo**(*db*, *port_id*)
Remove internal_info data from ports. When the data wasn't cleaned up, it appeared to block other instances from spawning on the node. Now it may not be required? More research needed.

hammers.query.**remove_extra_capability**(*db*, *host_id*, *capability_name*)
Remove an extra capability by name from *host_id*. (HTTP API doesn't support this as of Feb 2018)

hammers.query.**main**(*argv*)
Run queries!

CHAPTER 5

Miscellany

5.1 Slack

Slack integration that lets the scripts pipe up in #notifications is provided by `hammers.slack.Slackbot`

5.2 Util

Helper functions

`hammers.util.`**`drop_prefix`**`(`*s*, *start*`)`
Remove prefix *start* from sting *s*. Raises a `ValueError` if *s* didn't start with *start*.

`hammers.util.`**`nullcontext`**`(`**args*, ***kwargs*`)`
With `with`, wiff (do nothing). Added to stdlib in 3.7 as `contextlib.nullcontext()`

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

c

h

Index

L

M

N

O

P

Q

R

S

T

U

W