timmy Documentation

Release 1.26.8

Mirantis

Apr 12, 2017

Contents

Contents:

CHAPTER 1

Specification

OpenStack Ansible-like tool for parallel node operations: two-way data transfer, log collection, remote command execution

- The tool is based on https://etherpad.openstack.org/p/openstack-diagnostics
- Should work fine in environments deployed by Fuel versions: 4.x, 5.x, 6.x, 7.0, 8.0, 9.0, 9.1, 9.2
- Operates non-destructively.
- Can be launched on any host within admin network, provided the fuel node IP is specified and access to Fuel and other nodes is possible via ssh from the local system.
- Parallel launch - only on the nodes that are 'online'. Some filters for nodes are also available.
- Commands (from ./cmds directory) are separated according to roles (detected automatically) by the symlinks. Thus, the command list may depend on release, roles and OS. In addition, there can be some commands that run everywhere. There are also commands that are executed only on one node according to its role, using the first node of this type they encounter.
- Modular: possible to create a special package that contains only certain required commands.
- Collects log files from the nodes using filename and timestamp filters
- Packs collected data
- Checks are implemented to prevent filesystem overfilling due to log collection, appropriate error shown.
- Can be imported into other python scripts (ex. https://github.com/f3flight/timmy-customtest) and used as a transport and structure to access node parameters known to Fuel, run commands on nodes, collect outputs, etc. with ease.

CHAPTER 2

General configuration

All default configuration values are defined in `timmy/conf.py`. Timmy works with these values if no configuration file is provided. If a configuration file is provided via `-c | --config` option, it overlays the default configuration. An example of a configuration file is `timmy_data/rq/config/example.yaml`.

Some of the parameters available in configuration file:

- **ssh_opts** - parameters to send to ssh command directly (recommended to leave at default), such as connection timeout, etc. See `timmy/conf.py` to review defaults.
- **env_vars** - environment variables to pass to the commands and scripts - you can use these to expand variables in commands or scripts
- **fuel_ip** - the IP address of the master node in the environment
- **fuel_user** - username to use for accessing Nailgun API
- **fuel_pass** - password to access Nailgun API
- **fuel_tenant** - Fuel Keystone tenant to use when accessing Nailgun API
- **fuel_port** - port to use when connecting to Fuel Nailgun API
- **fuel_keystone_port** - port to use when getting a Keystone token to access Nailgun API
- **fuelclient** - True/False - whether to use fuelclient library to access Nailgun API
- **fuel_skip_proxy** - True/False - ignore `http(s)_proxy` environment variables when connecting to Nailgun API
- **rqdir** - the path to the directory containing rqfiles, scripts to execute, and filelists to pass to rsync
- **rqfile - list of dicts:**
  - **file** - path to an rqfile containing actions and/or other configuration parameters
  - **default** - True/False - this option is used to make **logs_no_default** work (see below). Optional.
- **logs_no_default** - True/False - do not collect logs defined in any rqfile for which "default" is True
- **logs_days** - how many past days of logs to collect. This option will set **start** parameter for each **logs** action if not defined in it.

- **logs_speed_limit** - True/False - enable speed limiting of log transfers (total transfer speed limit, not per-node)
- **logs_speed_default** - Mbit/s - used when autodetect fails
- **logs_speed** - Mbit/s - manually specify max bandwidth
- **logs_size_coefficient** - a float value used to check local free space; 'logs size * coefficient' must be > free space; values lower than 0.3 are not recommended and will likely cause local disk fillup during log collection
- **do_print_results** - print outputs of commands and scripts to stdout
- **clean** - True/False - erase previous results in outdir and archive_dir dir, if any
- **outdir** - directory to store output data. **WARNING: this directory is WIPED by default at the beginning of data collection. Be careful with what you define here.**
- **archive_dir** - directory to put resulting archives into
- **timeout** - timeout for SSH commands and scripts in seconds

CHAPTER 3

Configuring actions

Actions can be configured in a separate yaml file (by default `timmy_data/rq/default.yaml` is used) and / or defined in the main config file or passed via command line options `-P`, `-C`, `-S`, `-G`.

The following actions are available for definition:

- **put** - a list of tuples / 2-element lists: [source, destination]. Passed to `scp` like so `scp source <node-ip>:destination`. Wildcards supported for source.
- **cmds** - a list of dicts: {'command-name':'command-string'}. Example: {'command-1': 'uptime'}. Command string is a bash string. Commands are executed in alphabetical order of their names.
- **scripts - a list of elements, each of which can be a string or a dict:**
  - string - represents a script filename located on a local system. If filename does not contain a path separator, the script is expected to be located inside `rqdir/scripts`. Otherwise the provided path is used to access the script. Example: `'./my-test-script.sh'`
  - dict - use this option if you need to pass variables to your script. Script parameters are not supported, but you can use env variables instead. A dict should only contain one key which is the script filename (read above), and the value is a Bash space-separated variable assignment string. Example: `'./my-test-script.sh': 'var1=123 var2="HELLO WORLD"'`
  - **LIMITATION**: if you use a script with the same name more than once for a given node, the collected output will only contain the result of the last execution.
  - **INFO**: Scripts are not copied to the destination system - script code is passed as stdin to *bash -s* executed via ssh or locally. Therefore passing parameters to scripts is not supported (unlike cmds where you can write any Bash string). You can use variables in your scripts instead. Scripts are executed in the following order: all scripts without variables, sorted by their full filename, then all scripts with variables, also sorted by full filename. Therefore if the order matters, it's better to put all scripts into the same folder and name them according to the order in which you want them executed on the same node. Mind that scripts with variables are executed after all scripts without variables. If you need to mix scripts with variables and without and maintain order, just use dict structure for all scripts, and set *null* as the value for those which do not need variables.
- **files** - a list of filenames to collect. passed to `scp`. Supports wildcards.

- **filelists** - a list of filelist filenames located on a local system. Filelist is a text file containing files and directories to collect, passed to rsync. Does not support wildcards. If the filename does not contain path separator, the filelist is expected to be located inside `rqdir/filelists`. Otherwise the provided path is used to read the filelist.
- **logs**
  - **path** - base path to scan for logs
  - **include** - list of regexp strings to match log files against for inclusion (if not set = include all). Optional.
  - **exclude** - list of regexp strings to match log files against. Excludes matched files from collection. Optional.
  - **start** - date or datetime string to collect only files modified on or after the specified time. Format - `YYYY-MM-DD` or `YYYY-MM-DD HH:MM:SS` or `N` where N = integer number of days (meaning last N days). Optional.

CHAPTER 4

Filtering nodes

- **soft_filter** - use to skip any operations on non-matching nodes
- **hard_filter** - same as above but also removes non-matching nodes from NodeManager.nodes dict - useful when using timmy as a module

Nodes can be filtered by the following parameters defined inside soft_filter and/or hard_filter:

- **roles** - the list of roles, ex. **['controller','compute']**
- **online** - enabled by default to skip non-accessible nodes
- **status** - the list of statuses. Default: **['ready', 'discover']**
- **ids** - the list of ids, ex. **[0,5,6]**
- any other attribute of Node object which is a simple type (int, float, str, etc.) or a list containing simple types

Lists match **any**, meaning that if any element of the filter list matches node value (if value is a list - any element in it), the node passes.

Negative filters are possible by prefacing filter parameter with **no_**, example: **no_id = [0]** will filter out Fuel.

Negative lists also match **any** - if any match / collision found, the node is skipped.

You can combine any number of positive and negative filters as long as their names differ (since this is a dict).

You can use both positive and negative parameters to match the same node parameter (though it does not make much sense): **roles = ['controller', 'compute'] no_roles = ['compute']** This will skip computes and run only on controllers. As already said, does not make much sense :)

CHAPTER 5

Parameter-based configuration

It is possible to define special **by_<parameter-name>** dicts in config to (re)define node parameters based on other parameters. For example:

```
by_roles:
  controller:
    cmds: {'check-uptime': 'uptime'}
```

In this example for any controller node, cmds setting will be reset to the value above. For nodes without controller role, default (none) values will be used.

Negative matches are possible via **no_** prefix:

```
by_roles:
  no_fuel:
    cmds: {'check-uptime': 'uptime'}
```

In this example **uptime** command will be executed on all nodes except Fuel server.

It is also possible to define a special **once_by_<parameter-name>** which works similarly, but will only result in attributes being assigned to a single (first in the list) matching node. Example:

```
once_by_roles:
  controller:
    cmds: {'check-uptime': 'uptime'}
```

Such configuration will result in *uptime* being executed on only one node with controller role, not on every controller.

CHAPTER 6

rqfile format

`rqfile` format is a bit different from config. The basic difference:

config:

```
scripts: [a ,b, c]
by_roles:
  compute:
    scripts: [d, e, f]
```

rqfile:

```
scripts:
  __default: [a, b, c]
  by_roles:
    compute: [d, e, f]
```

The **config** and **rqfile** definitions presented above are equivalent. It is possible to define actions in a config file using the **config** format, or in an **rqfile** using **rqfile** format, linking to the **rqfile** in config with `rqfile` setting. It is also possible to define part here and part there. Mixing identical parameters in both places is not recommended - the results may be unpredictable (such a scenario has not been thoroughly tested). In general, **rqfile** is the preferred place to define actions.

CHAPTER 7

Configuration application order

Configuration is assembled and applied in a specific order:

1. default configuration is initialized. See `timmy/conf.py` for details.
2. command line parameters, if defined, are used to modify the configuration.
3. **rqfile**, if defined (default - `rq.yaml`), is converted and injected into the configuration. At this stage the configuration is in its final form.
4. **for every node, configuration is applied, except `once_by_` directives:**
    (a) first the top-level attributes are set
    (b) then `by_<attribute-name>` parameters are iterated to override settings and append(accumulate) actions
5. finally `once_by_`<attribute-name>` parameters are applied - only for one matching node for any set of matching values. This is useful, for example, if you want a specific file or command from only a single node matching a specific role, like running `nova list` only on one controller.

Once you are done with the configuration, you might want to familiarize yourself with *Usage*.

CHAPTER 8

Usage

NOTICE: Even though Timmy uses nice and ionice to limit impact on the cloud, you should still expect 1 core utilization both locally (where Timmy is launched) and on each node where commands are executed or logs collected. Additionally, if logs are collected, local disk (log destination directory) may get utilized significantly.

WARNING If modifying the `outdir` config parameter, please first read the related warning on *configuration </configuration>* page.

The easiest way to launch Timmy would be running the `timmy.py` script / `timmy` command: * Timmy will perform all actions defined in the `default.yaml` rq-file. The file is located in `timmy_data/rq` folder in Python installation directory. Specifically:

- run diagnostic scripts on all nodes, including Fuel server
- collect configuration files for all nodes
- Timmy will **NOT** collect log files when executed this way.

Basically, `timmy.py` is a simple wrapper that launches `cli.py`. * Current page does not reference all available CLI options. Full *reference* for command line interface. * You may also want to create a custom *configuration* for Timmy, depending on your use case.

Basic parameters:

- `--only-logs` only collect logs (skip files, filelists, commands and scripts)
- `-l`, `--logs` also collect logs (logs are not collected by default due to their size)
- `-e`, `--env` filter by environment ID
- `-R`, `--role` filter by role
- `-c`, `--config` use custom configuration file to overwrite defaults. See `timmy_data/config/example.yaml` as an example
- `-j`, `--nodes-json` use json file instead of polling Fuel (to generate json file use `fuel node --json`) - speeds up initialization
- `-o`, `--dest-file` the name/path for output archive, default is `general.tar.gz` and put into `/tmp/timmy/archives`. A folder will be created if it does not exist. It's not recommended to use `/var/log` as

destination because subsequent runs with log collection may cause Timmy to collect it's own previously created files or even update them while reading from them. The general idea is that a destination directory should contain enough space to hold all collected data and should not be in collection paths.

- `-v`, `--verbose` verbose(INFO) logging. Use `-vv` to enable DEBUG logging.

CHAPTER 9

Shell Mode

Shell Mode is activated whenever any of the following parameters are used via CLI: `-C`, `-S`, `-P`, `-G`.

A mode of execution which makes the following changes:

- rqfile (`timmy_data/rq/default.yaml` by default) is skipped
- Fuel node is skipped. If for some reason you need to run specific scripts/actions via Timmy on Fuel and on other nodes at the same time, create an rqfile instead (see *configuration* for details, see `timmy_data/rq/neutron.yaml` as an example), coupled with `--rqfile` option or a custom config file to override default rqfile.
- outputs of commands (specified with `-C` options) and scripts (specified with `-S`) are printed on screen
- any actions (cmds, scripts, files, filelists, put, **except** logs) and Parameter Based configuration defined in config are ignored.

The following parameters ("actions") are available via CLI:

- `-C <command>` - Bash command (string) to execute on nodes. Using multiple `-C` statements will produce the same result as using one with several commands separated by `;` (traditional Shell syntax), but for each `-C` statement a new SSH connection is established.
- `-S <script>` - name of the Bash script file to execute on nodes (if you do not have a path separator in the filename, you need to put the file into `scripts` folder inside a path specified by `rqdir` config parameter, defaults to `rq`. If a path separator is present, the given filename will be used directly as provided)
- `-P <file/path> <dest>` - upload local data to nodes (wildcards supported). You must specify 2 values for each `-P` switch.
- `-G <file/path>` - download (collect) data from nodes

CHAPTER 10

Logs

It's possible to specify custom log collection when using CLI: * `-L <base-path> <include-regex> <exclude-regex>`, `--get-logs` - specify a base path, include regex and exclude regex to collect logs. This option can be specified more than once, in this case log lists will be united. This option **does not** disable default log collection defined in `timmy_data/rq/default.yaml`. * `--logs-no-default` - use this option of you **only** need logs specified via `-L`.

CHAPTER 11

Execution order

Specified actions are executed for all applicable nodes, always in the following order: 1. put 2. commands 3. scripts 4. get, filelists 5. logs

CHAPTER 12

Examples

- `timmy` - run according to the default configuration and default actions. Default actions are defined in `timmy_data/rq/default.yaml`. Logs are not collected.
- `timmy -l` - run default actions and also collect logs. Such execution is similar to Fuel's "diagnostic snapshot" action, but will finish faster and collect less logs. There is a default log collection period based on file modification time, only files modified within the last 30 days are collected.
- `timmy -l --days 3` - same as above but only collect log files updated within the last 3 days.
- `timmy --only-logs` - only collect logs, no actions (files, filelists, commands, scripts, put, get) performed.
- `timmy -C 'uptime; free -m'` - check uptime and memory on all nodes
- `timmy -G /etc/nova/nova.conf` - get `nova.conf` from all nodes
- `timmy -R controller -P package.deb '' -C 'dpkg -i package.deb' -C 'rm package.deb' -C 'dpkg -l | grep [p]ackage'` - push a package to all nodes, install it, remove the file and check that it is installed. Commands are executed in the order in which they are provided.
- `timmy - myconf.yaml` - use a custom config file and run the program according to it. Custom config can specify any actions, log setup, and other settings. See configuration doc for more details.

CHAPTER 13

Using custom configuration file

If you want to perform a set of actions on the nodes without writing a long command line (or if you want to use the options only available in config), you may want to set up config file instead. An example config structure would be:

```
rqdir: './pacemaker-debug' # a folder which should contain any filelists and/or␣
↪scripts if they are defined later, should contain folders 'filelists' and/or
↪'scripts'
rqfile: null # explicitly undefine rqfile to skip default filelists and scripts
hard_filter:
  roles: # only execute on Fuel and controllers
    - fuel
    - controller
cmds: # some commands to run on all nodes (after filtering). cmds syntax is {name:␣
↪value, ...}. cmds are executed in alphabetical order.
  01-my-first-command: 'uptime'
  02-disk-check: 'df -h'
  and-also-ram: 'free -m'
logs:
  path: '/var/log' # base path to search for logs
  exclude: # a list of exclude regexes
    - '.*' # exclude all logs by default - does not make much sense - just an example.
↪ If the intention is to not collect all logs then this 'logs' section can be␣
↪removed altogether, just ensure that either rqfile is custom or 'null', or '--logs-
↪no-default' is set via CLI / 'logs_no_default: True' set in config.
logs_days: 5 # collect only log files updated within the last 5 days
# an example of parameter-based configuration is below:
by_roles:
  controller:
    scripts: # I use script here to not overwrite the cmds we have already defined␣
↪for all nodes
      - pacemaker-debug.sh # the name of the file inside 'scripts' folder inside
↪'rqdir' path, which will be executed (by default) on all nodes
    files:
      - '/etc/coros*' # get all files from /etc/coros* wildcard path
  fuel:
    logs:
```

```
      path: '/var/log/remote'
      include: # include regexp - non-matching log files will be excluded.
        - 'crmd|lrmd|corosync|pacemaker'
```

Then you would run `timmy -l -c my-config.yaml` to execute Timmy with such config.

Instead of putting all structure in a config file you can move actions (cmds, files, filelists, scripts, logs) to an rqfile, and specify `rqfile` path in config (although in this example the config-way is more compact). `rqfile` structure is a bit different:

```
cmds: # top-level elements are node parameters, __default will be assigned to all␣
↪nodes
  __default:
    - 01-my-first-command: 'uptime'
    - 02-disk-check: 'df -h'
    - and-also-ram: 'free -m'
scripts:
  by_roles: # all non "__default" keys should match, "by_<parameter>"
    controller:
      - pacemaker-debug.sh
files:
  by_roles:
    controller:
      - '/etc/coros*'
logs:
  by_roles:
    fuel:
      path: '/var/log/remote'
      include:
        - 'crmd|lrmd|corosync|pacemaker'
  __default: # again, this default section is useless, just serving as an example␣
↪here.
    path: '/var/log'
    exclude:
      - '.*'
```

Then the config should look like this:

```
rqdir: './pacemaker-debug'
rqfile:
  - file: './pacemaker-rq.yaml'
hard_filter:
  roles:
    - fuel
    - controller
```

And you run `timmy -l -c my-config.yaml`.

Back to *Index*.

CHAPTER 14

CLI

Parallel remote command execution and file manipulation tool

```
usage: timmy [-V] [-c CONFIG] [-o DEST_FILE] [--log-file LOG_FILE] [-e ENV]
             [-r ROLE] [-i ID] [-d NUMBER] [-G PATH] [-C COMMAND] [-S SCRIPT]
             [--one-way] [--max-pairs NUMBER] [-P SOURCE DESTINATION]
             [-L PATH INCLUDE EXCLUDE] [--rqfile PATH] [-l]
             [--logs-no-default] [--logs-speed MBIT/S] [--logs-speed-auto]
             [--logs-coeff RATIO] [--only-logs] [--fake] [--fake-logs]
             [--no-archive] [--no-clean] [-q] [--maxthreads NUMBER]
             [--logs-maxthreads NUMBER] [-t] [-T] [-m INVENTORY MODULE] [-a]
             [--offline] [-v]
```

Named Arguments

-V, --version	Print Timmy version and exit. Default: False
-c, --config	Path to a YAML configuration file.
-o, --dest-file	Output filename for the archive in tar.gz format for command outputs and collected files. Overrides "**archive_**" config options. If logs are collected they will be placed in the same folder (but separate archives).
--log-file	Redirect Timmy log to a file.
-e, --env	Env ID. Run only on specific environment.
-r, --role	Can be specified multiple times. Run only on the specified role.
-i, --id	Can be specified multiple times. Run only on the node(s) with given IDs.
-d, --days	Define log collection period in days. Timmy will collect only logs updated on or more recently then today minus the given number of days. Default - 30.

-G, --get Enables shell mode. Can be specified multiple times. Filemask to collect via "scp -r". Result is placed into a folder specified by "outdir" config option. For help on shell mode, read timmy/conf.py.

-C, --command Enables shell mode. Can be specified multiple times. Shell command to execute. For help on shell mode, read timmy/conf.py.

-S, --script Enables shell mode. Can be specified multiple times. Bash script name to execute. Script must be placed in "scripts" folder inside a path specified by "rqdir" configuration parameter. For help on shell mode, read timmy/conf.py.

--one-way When executing scripts_all_pairs (if defined), for each pair of nodes [A, B] run client script only on A (A->B connection). Default is to run both A->B and B->A.

Default: False

--max-pairs When executing scripts_all_pairs (if defined), limit the amount of pairs processed simultaneously. Default is to run max number of pairs possible, which is num. nodes / 2.

-P, --put Enables shell mode. Can be specified multiple times. Upload filemask via"scp -r" to node(s). Each argument must contain two strings - source file/path/mask and dest. file/path. For help on shell mode, read timmy/conf.py.

-L, --get-logs Define specific logs to collect. Implies "-l". Each -L option requires 3 values in the following order: path, include, exclude. See configuration doc for details on each of these parameters. Values except path can be skipped by passing empty strings. Example: -L "/var/mylogs/" "" "exclude-string"

--rqfile Can be specified multiple times. Path to rqfile(s) in yaml format, overrides default.

-l, --logs Collect logs from nodes. Logs are not collected by default due to their size.

Default: False

--logs-no-default Do not use default log collection parameters, only use what has been provided either via -L or in rqfile(s). Implies "-l".

Default: False

--logs-speed Limit log collection bandwidth to 90% of the specified speed in Mbit/s.

--logs-speed-auto Limit log collection bandwidth to 90% of local admin interface speed. If speed detection fails, a default value will be used. See "logs_speed_default" in conf.py.

Default: False

--logs-coeff Estimated logs compression ratio - this value is used during free space check. Set to a lower value (default - 1.05) to collect logs of a total size larger than locally available. Values lower than 0.3 are not recommended and may result in filling up local disk.

--only-logs Only collect logs, do not run commands or collect files.

Default: False

--fake Do not run commands and scripts

Default: False

--fake-logs Do not collect logs, only calculate size.

Default: False

--no-archive Do not create results archive. By default, an archive with all outputs and files is created every time you run Timmy.

Default: False

--no-clean Do not clean previous results. Allows accumulating results across runs.

Default: False

-q, --quiet Print only command execution results and log messages. Good for quick runs / "watch" wrap. This option disables any -v parameters.

Default: False

--maxthreads Maximum simultaneous nodes for commandexecution.

--logs-maxthreads Maximum simultaneous nodes for log collection.

-t, --outputs-timestamp Add timestamp to outputs - allows accumulating outputs of identical commands/scripts across runs. Only makes sense with –no-clean for subsequent runs.

Default: False

-T, --dir-timestamp Add timestamp to output folders (defined by "outdir" and "archive_dir" config options). Makes each run store results in new folders. This way Timmy will always preserve previous results. Do not forget to clean up the results manually when using this option.

Default: False

-m, --module Use module to get node data

Default: "fuel"

-a, --analyze Analyze collected outputs to determine node orservice health and print results

Default: False

--offline Mark all nodes as offline, do not perform anyoperations on the nodes

Default: False

-v, --verbose This works for -vvvv, -vvv, -vv, -v, -v -v,etc, If no -v then logging.WARNING is selected if more -v are provided it will step to INFO and DEBUG unless the option -q(–quiet) is specified

Default: 0

Fuel module parameters

```
usage: timmy [-h] [--fuel-ip FUEL_IP] [--fuel-user FUEL_USER]
             [--fuel-pass FUEL_PASS] [--fuel-token FUEL_TOKEN]
             [--fuel-logs-no-remote] [--fuel-proxy] [-j NODES_JSON]
```

Named Arguments

--fuel-ip fuel ip address

--fuel-user fuel username

--fuel-pass fuel password

--fuel-token fuel auth token

--fuel-logs-no-remote Do not collect remote logs from Fuel.

Default: False

--fuel-proxy use os system proxy variables for fuelclient

Default: False

-j, --nodes-json Path to a json file retrieved via "fuel node –json". Useful to speed up initialization, skips "fuel node" call.

Local module parameters

```
usage: timmy [-h] -j NODES_JSON
```

Named Arguments

-j, --nodes-json Path to a json file containing host info: ip, roles, etc.

CHAPTER 15

Tools module

tools module

timmy.tools.**load_json_file**(*filename*)
Loads json data from file

timmy.tools.**load_yaml_file**(*filename*)
Loads yaml data from file

timmy.tools.**mdir**(*directory*)
Creates a directory if it doesn't exist

CHAPTER 16

Exit Codes

- *2* - SIGINT (Keyboard Interrupt) caught.
- *100* - not enough free space for logs. Decrease logs coefficient via CLI or config or free up space.
- *101* - **rqdir** configuration parameter points to a non-existing directory.
- *102* - could not load YAML file - I/O Error.
- *103* - could not load YAML file - Value Error, see log for details.
- *104* - could not load YAML file - Parser Error - incorrectly formatted YAML.
- *105* - could not retrieve information about nodes by any available means.
- *106* - **fuel_ip** configuration parameter not defined.
- *107* - could not load JSON file - I/O Error.
- *108* - could not load JSON file - Value Error, see log for details.
- *109* - subprocess (one of the node execution processes) exited with a Python exception.
- *110* - unable to create a directory.
- *111* - ip address must be defined for Node instance.
- *112* - one of the two parameters **fuel_user** or **fuel_pass** specified without the other.
- *113* - unhandled Python exception occured in main process.

CHAPTER 17

Indices and tables

Python Module Index

t

Index