os-faults Documentation

Release 0.1.12.dev6

OpenStack Foundation

Apr 13, 2017

Contents

OpenStack fault-injection library

The library does destructive actions inside an OpenStack cloud. It provides an abstraction layer over different types of cloud deployments. The actions are implemented as drivers (e.g. DevStack driver, Fuel driver, Libvirt driver, IPMI driver).

CHAPTER 1

Contents

Quickstart

This section describes how to start using os-faults.

Installation

At the command line:

```
$ pip install os-faults
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv os-faults
$ pip install os-faults
```

The library contains optional libvirt driver, if you plan to use it, please use the following command to install os-faults with extra dependencies:

```
pip install os-faults[libvirt]
```

Basics

Configuration file

The cloud deployment configuration schema is an extension to the cloud config used by the os-client-config library:

```
cloud_management:
  driver: devstack
  args:
    address: 192.168.1.240
```

```
    username: ubuntu
    iface: enp0s3

power_managements:
- driver: libvirt
  args:
    connection_uri: qemu+ssh://ubuntu@10.0.1.50/system
```

By default, the library reads configuration from a file and the file can be in the following three formats: `os-faults.{json,yaml,yml}`. The configuration file will be searched in the default locations:

- current directory
- ~/.config/os-faults
- /etc/openstack

Also, the configuration file can be specified in the `OS_FAULTS_CONFIG` environment variable:

```
$ export OS_FAULTS_CONFIG=/home/alex/my-os-faults-config.yaml
```

Running

Establish a connection to the cloud and verify it:

```
import os_faults
destructor = os_faults.connect(config_filename='os-faults.yaml')
destructor.verify()
```

or via CLI:

```
$ os-faults -c os-faults.yaml verify
```

Make some destructive actions:

```
destructor.get_service(name='keystone').restart()
```

or via CLI:

```
$ os-inject-fault -c os-faults.yaml restart keystone service
```

API

The library operates with 2 types of objects:

- *service* - is a software that runs in the cloud, e.g. *nova-api*
- *nodes* - nodes that host the cloud, e.g. a hardware server with a hostname

Simplified API

Simplified API is used to inject faults in a human-friendly form.

```
import os_faults
destructor = os_faults.connect(config_filename='os-faults.yaml')
os_faults.human_api(destructor, 'restart keystone service')
```

Service-oriented command performs specified *action* against *service* on all, on one random node or on the node specified by FQDN:

```
<action> <service> service [on (random|one|single|<fqdn> node[s])]
```

Examples:

- *Restart Keystone service* - restarts Keystone service on all nodes.
- *kill nova-api service on one node* - restarts Nova API on one randomly-picked node.

Node-oriented command performs specified *action* on node specified by FQDN or set of service's nodes:

```
<action> [random|one|single|<fqdn>] node[s] [with <service> service]
```

Examples:

- *Reboot one node with mysql* - reboots one random node with MySQL.
- *Reset node-2.domain.tld node* - reset node *node-2.domain.tld*.

Network-oriented command is a subset of node-oriented and performs network management operation on selected nodes:

```
<action> <network> network on [random|one|single|<fqdn>] node[s]
    [with <service> service]
```

Examples:

- *Disconnect management network on nodes with rabbitmq service* - shuts down management network interface on all nodes where rabbitmq runs.
- *Connect storage network on node-1.domain.tld node* - enables storage network interface on node-1.domain.tld.

Extended API

1. Service actions

Get a service and restart it:

```
destructor = os_faults.connect(cloud_config)
service = destructor.get_service(name='glance-api')
service.restart()
```

Available actions:

- *start* - start Service
- *terminate* - terminate Service gracefully
- *restart* - restart Service
- *kill* - terminate Service abruptly
- *unplug* - unplug Service out of network
- *plug* - plug Service into network

2. Node actions

Get all nodes in the cloud and reboot them:

```
nodes = destructor.get_nodes()
nodes.reboot()
```

Available actions:

- *reboot* - reboot all nodes gracefully
- *poweroff* - power off all nodes abruptly
- *reset* - reset (cold restart) all nodes
- *oom* - fill all node's RAM
- *disconnect* - disable network with the specified name on all nodes
- *connect* - enable network with the specified name on all nodes

3. Operate with nodes

Get all nodes where a service runs, pick one of them and reset:

```
nodes = service.get_nodes()
one = nodes.pick()
one.reset()
```

Get nodes where l3-agent runs and disable the management network on them:

```
fqdns = neutron.l3_agent_list_hosting_router(router_id)
nodes = destructor.get_nodes(fqdns=fqdns)
nodes.disconnect(network_name='management')
```

4. Operate with services

Restart a service on a single node:

```
service = destructor.get_service(name='keystone')
nodes = service.get_nodes().pick()
service.restart(nodes)
```

Configuration specification

Configuration file contains the following parameters:

- cloud_management
- power_managements
- node_discover
- services

Each parameter specifies a driver or a list of drivers.

Example configuration:

```
cloud_management:
  driver: devstack
  args:
    address: 192.168.1.240
    username: ubuntu
    iface: enp0s3

power_managements:
- driver: libvirt
  args:
    connection_uri: qemu+ssh://ubuntu@10.0.1.50/system

- driver: ipmi
  args:
    fqdn_to_bmc:
      node-1.domain.tld:
        address: 120.10.30.65
        username: alex
        password: super-secret

node_discover:
  driver: node_list
  args:
  - fqdn: node-1.domain.tld
    ip: 192.168.1.240
    mac: 1e:24:c3:75:dd:2c

services:
  glance-api:
    driver: screen
    args:
      grep: glance-api
      window_name: g-api
    ips:
    - 192.168.1.240
```

cloud_management

This parameter specifies cloud managment driver and its argumanets. `cloud_management` is responsible for configuring connection to nodes and contains arguments such as SSH username/password/key/proxy.

```
cloud_management:
  driver: devstack  # name of the driver
  args:             # arguments for the driver
    address: 192.168.1.240
    username: ubuntu
    iface: enp0s3
```

Also, such drivers can support discovering of cloud nodes. For example, `fuel`, `tcpcloud` drives allow discovering information about nodes through master/config node of the cloud.

List of supported drivers for cloud_management: *Cloud management*

power_managements

This parameter specifies list of power management drivers. Such drivers allow controlling power state of cloud nodes.

```
power_managements:
- driver: libvirt    # name of the driver
  args:              # arguments for the driver
    connection_uri: qemu+ssh://ubuntu@10.0.1.50/system

- driver: ipmi       # name of the driver
  args:              # arguments for the driver
    fqdn_to_bmc:
      node-1.domain.tld:
        address: 120.10.30.65
        username: alex
        password: super-secret
```

List of supported drivers for power_managements: *Power management*

node_discover

This parameter specifies node discover driver. `node_discover` is responsible for fetching list of hosts for the cloud. If `node_discover` is specified in configuration then `cloud_management` will only control connection options to the nodes.

```
node_discover:
  driver: node_list
  args:
  - fqdn: node-1.domain.tld
    ip: 192.168.1.240
    mac: 1e:24:c3:75:dd:2c
```

List of supported drivers for node_discover: *Node discover*

services

This parameter specifies list of services and their types. This parameter allows updating/adding services which are embedded in `cloud_management` driver.

```
services:
  glance-api:            # name of the service
    driver: screen       # name of the service driver
    args:                # arguments for the driver
      grep: glance-api
      window_name: g-api
    ips:                 # list of ips where this service running
    - 192.168.1.240
  mysql:                 # name of the service
    driver: process      # name of the service driver
    args:                # arguments for the driver
      grep: mysqld
      port:
      - tcp
      - 3307
      restart_cmd: sudo service mysql restart
```

```
      start_cmd: sudo service mysql start
      terminate_cmd: sudo service mysql stop
```

Service driver contains optional `ips` parameter which controls discovering of hosts where the service is running. If `ips` specified, then service discovering is disabled for this service and hosts specified in `ips` will be used, otherwise, service will be searched across all nodes.

List of supported drivers for services: *Service drivers*

CLI reference

os-inject-fault

```
usage: os-inject-fault [-h] [-c CONFIG] [-d] [-v] [command]

positional arguments:
  command               fault injection command, e.g. "restart keystone
                        service"

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        path to os-faults cloud connection config
  -d, --debug
  -v, --verify          verify connection to the cloud

Built-in drivers:
  salt_service - Service in salt
  tcpcloud - TCPCloud management driver
  linux_service - Service in init.d
  process - Service as process
  ipmi - IPMI power management driver
  screen - Service in screen
  pcs_or_linux_service - Service in pacemaker or init.d
  node_list - Reads hosts from configuration file
  libvirt - Libvirt power management driver
  pcs_service - Service in pacemaker
  fuel - Fuel 9.x cloud management driver
  devstack - DevStack management driver

*Service-oriented* commands perform specified action against service on
all, on one random node or on the node specified by FQDN:

  <action> <service> service [on (random|one|single|<fqdn> node[s])]

  where:
    action is one of:
      freeze - Pause service execution
      kill - Terminate Service abruptly on all nodes or on particular subset
      plug - Plug Service into network on all nodes or on particular subset
      restart - Restart Service on all nodes or on particular subset
      start - Start Service on all nodes or on particular subset
      terminate - Terminate Service gracefully on all nodes or on particular subset
      unfreeze - Resume service execution
      unplug - Unplug Service out of network on all nodes or on particular subset
```

```
    service is one of supported by driver:
      tcpcloud: cinder-api, cinder-backup, cinder-scheduler, cinder-volume,
                elasticsearch, glance-api, glance-registry,
                grafana-server, heat-api, heat-engine, horizon,
                influxdb, keystone, kibana, memcached, mysql, nagios3,
                neutron-dhcp-agent, neutron-l3-agent,
                neutron-metadata-agent, neutron-openvswitch-agent,
                neutron-server, nova-api, nova-cert, nova-compute,
                nova-conductor, nova-consoleauth, nova-novncproxy,
                nova-scheduler, rabbitmq
      fuel: cinder-api, cinder-backup, cinder-scheduler, cinder-volume,
            glance-api, glance-glare, glance-registry, heat-api,
            heat-engine, horizon, ironic-api, ironic-conductor,
            keystone, memcached, mysql, neutron-dhcp-agent,
            neutron-l3-agent, neutron-metadata-agent,
            neutron-openvswitch-agent, neutron-server, nova-api,
            nova-cert, nova-compute, nova-conductor, nova-consoleauth,
            nova-novncproxy, nova-scheduler, rabbitmq, swift-account,
            swift-account-auditor, swift-account-reaper,
            swift-account-replicator, swift-container,
            swift-container-auditor, swift-container-replicator,
            swift-container-sync, swift-container-updater,
            swift-object, swift-object-auditor,
            swift-object-replicator, swift-object-updater, swift-proxy
      devstack: glance-api, ironic-api, ironic-conductor, keystone, mysql, nova-api,
                nova-compute, nova-scheduler, rabbitmq

Examples:
    * "Restart Keystone service" - restarts Keystone service on all nodes.
    * "kill nova-api service on one node" - restarts Nova API on one
      randomly-picked node.

*Node-oriented* commands perform specified action on node specified by FQDN
or set of service's nodes:

  <action> [random|one|single|<fqdn>] node[s] [with <service> service]

  where:
    action is one of:
      connect - Connect nodes to <network_name> network
      disconnect - Disconnect nodes from <network_name> network
      oom - Fill all node's RAM
      poweroff - Power off all nodes abruptly
      poweron - Power on all nodes abruptly
      reboot - Reboot all nodes gracefully
      reset - Reset (cold restart) all nodes
      shutdown - Shutdown all nodes gracefully
    service is one of supported by driver:
      tcpcloud: cinder-api, cinder-backup, cinder-scheduler, cinder-volume,
                elasticsearch, glance-api, glance-registry,
                grafana-server, heat-api, heat-engine, horizon,
                influxdb, keystone, kibana, memcached, mysql, nagios3,
                neutron-dhcp-agent, neutron-l3-agent,
                neutron-metadata-agent, neutron-openvswitch-agent,
                neutron-server, nova-api, nova-cert, nova-compute,
                nova-conductor, nova-consoleauth, nova-novncproxy,
                nova-scheduler, rabbitmq
      fuel: cinder-api, cinder-backup, cinder-scheduler, cinder-volume,
```

```
            glance-api, glance-glare, glance-registry, heat-api,
            heat-engine, horizon, ironic-api, ironic-conductor,
            keystone, memcached, mysql, neutron-dhcp-agent,
            neutron-l3-agent, neutron-metadata-agent,
            neutron-openvswitch-agent, neutron-server, nova-api,
            nova-cert, nova-compute, nova-conductor, nova-consoleauth,
            nova-novncproxy, nova-scheduler, rabbitmq, swift-account,
            swift-account-auditor, swift-account-reaper,
            swift-account-replicator, swift-container,
            swift-container-auditor, swift-container-replicator,
            swift-container-sync, swift-container-updater,
            swift-object, swift-object-auditor,
            swift-object-replicator, swift-object-updater, swift-proxy
      devstack: glance-api, ironic-api, ironic-conductor, keystone, mysql, nova-api,
                nova-compute, nova-scheduler, rabbitmq

Examples:
    * "Reboot one node with mysql" - reboots one random node with MySQL.
    * "Reset node-2.domain.tld node" - reset node node-2.domain.tld.

*Network-oriented* commands are subset of node-oriented and perform network
management operation on selected nodes:

  [connect|disconnect] <network> network on [random|one|single|<fqdn>] node[s]
    [with <service> service]

  where:
    network is one of supported by driver:
      tcpcloud:
      fuel: management, private, public, storage
      devstack: all-in-one
    service is one of supported by driver:
      tcpcloud: cinder-api, cinder-backup, cinder-scheduler, cinder-volume,
                elasticsearch, glance-api, glance-registry,
                grafana-server, heat-api, heat-engine, horizon,
                influxdb, keystone, kibana, memcached, mysql, nagios3,
                neutron-dhcp-agent, neutron-l3-agent,
                neutron-metadata-agent, neutron-openvswitch-agent,
                neutron-server, nova-api, nova-cert, nova-compute,
                nova-conductor, nova-consoleauth, nova-novncproxy,
                nova-scheduler, rabbitmq
      fuel: cinder-api, cinder-backup, cinder-scheduler, cinder-volume,
            glance-api, glance-glare, glance-registry, heat-api,
            heat-engine, horizon, ironic-api, ironic-conductor,
            keystone, memcached, mysql, neutron-dhcp-agent,
            neutron-l3-agent, neutron-metadata-agent,
            neutron-openvswitch-agent, neutron-server, nova-api,
            nova-cert, nova-compute, nova-conductor, nova-consoleauth,
            nova-novncproxy, nova-scheduler, rabbitmq, swift-account,
            swift-account-auditor, swift-account-reaper,
            swift-account-replicator, swift-container,
            swift-container-auditor, swift-container-replicator,
            swift-container-sync, swift-container-updater,
            swift-object, swift-object-auditor,
            swift-object-replicator, swift-object-updater, swift-proxy
      devstack: glance-api, ironic-api, ironic-conductor, keystone, mysql, nova-api,
                nova-compute, nova-scheduler, rabbitmq
```

```
Examples:
    * "Disconnect management network on nodes with rabbitmq service" - shuts
      down management network interface on all nodes where rabbitmq runs.
    * "Connect storage network on node-1.domain.tld node" - enables storage
      network interface on node-1.domain.tld.

For more details please refer to docs: http://os-faults.readthedocs.io/
```

os-faults

```
Usage: os-faults [OPTIONS] COMMAND [ARGS]...

Options:
  -d, --debug  Enable debug logs
  --version    Show version and exit.
  --help       Show this message and exit.

Commands:
  discover  Discover services/nodes and save them to...
  drivers   List os-faults drivers
  nodes     List cloud nodes
  verify    Verify connection to the cloud
```

os-faults verify

```
Usage: os-faults verify [OPTIONS]

  Verify connection to the cloud

Options:
  -c, --config PATH  path to os-faults cloud connection config
  --help             Show this message and exit.
```

os-faults discover

```
Usage: os-faults discover [OPTIONS]

  Discover services/nodes and save them to config file

Options:
  --dst-path PATH    Alternate location for discovered config
  -c, --config PATH  path to os-faults cloud connection config
  --help             Show this message and exit.
```

os-faults nodes

```
Usage: os-faults nodes [OPTIONS]

  List cloud nodes
```

```
Options:
  -c, --config PATH  path to os-faults cloud connection config
  --help             Show this message and exit.
```

os-faults drivers

```
Usage: os-faults drivers [OPTIONS]

  List os-faults drivers

Options:
  --help  Show this message and exit.
```

Drivers

Cloud management

devstack [CloudManagement, NodeDiscover]

Devstack driver.

This driver requires devstack installed in screen mode (USE_SCREEN=True). Supports discovering of node MAC addreses.

Example configuration:

```
cloud_management:
  driver: devstack
  args:
    address: 192.168.1.10
    username: ubuntu
    password: ubuntu_pass
    private_key_file: ~/.ssh/id_rsa_devstack
    slaves:
    - 192.168.1.11
    - 192.168.1.12
    iface: eth1
```

parameters:

- **address** - ip address of any devstack node
- **username** - username for all nodes
- **password** - password for all nodes (optional)
- **private_key_file** - path to key file (optional)
- **slaves** - list of ips for additional nodes (optional)
- **iface** - network interface name to retrive mac address (optional)

Default services:

- glance-api

- ironic-api
- ironic-conductor
- keystone
- mysql
- nova-api
- nova-compute
- nova-scheduler
- rabbitmq

fuel [CloudManagement, NodeDiscover]

Fuel driver.

Cloud deployed by fuel. Supports discovering of slave nodes.

Example configuration:

```
cloud_management:
  driver: fuel
  args:
    address: 192.168.1.10
    username: root
    private_key_file: ~/.ssh/id_rsa_fuel
    slave_direct_ssh: True
```

parameters:

- **address** - ip address of fuel master node
- **username** - username for fuel master and slave nodes
- **private_key_file** - path to key file (optional)
- **slave_direct_ssh** - if *False* then fuel master is used as ssh proxy (optional)

Default services:

- cinder-api
- cinder-backup
- cinder-scheduler
- cinder-volume
- glance-api
- glance-glare
- glance-registry
- heat-api
- heat-engine
- horizon
- ironic-api
- ironic-conductor

- keystone
- memcached
- mysql
- neutron-dhcp-agent
- neutron-l3-agent
- neutron-metadata-agent
- neutron-openvswitch-agent
- neutron-server
- nova-api
- nova-cert
- nova-compute
- nova-conductor
- nova-consoleauth
- nova-novncproxy
- nova-scheduler
- rabbitmq
- swift-account
- swift-account-auditor
- swift-account-reaper
- swift-account-replicator
- swift-container
- swift-container-auditor
- swift-container-replicator
- swift-container-sync
- swift-container-updater
- swift-object
- swift-object-auditor
- swift-object-replicator
- swift-object-updater
- swift-proxy

tcpcloud [CloudManagement, NodeDiscover]

TCPCloud driver.

Supports discovering of slave nodes.

Example configuration:

```
cloud_management:
  driver: tcpcloud
  args:
    address: 192.168.1.10
    username: root
    password: root_pass
    private_key_file: ~/.ssh/id_rsa_tcpcloud
    slave_username: ubuntu
    slave_password: ubuntu_pass
    master_sudo: False
    slave_sudo: True
    slave_name_regexp: ^(?!cfg|mon)
    slave_direct_ssh: True
    get_ips_cmd: pillar.get _param:single_address
```

parameters:

- **address** - ip address of salt config node
- **username** - username for salt config node
- **password** - password for salt config node (optional)
- **private_key_file** - path to key file (optional)
- **slave_username** - username for salt minions (optional) *username* will be used if *slave_username* not specified
- **slave_password** - password for salt minions (optional) *password* will be used if *slave_password* not specified
- **master_sudo** - Use sudo on salt config node (optional)
- **slave_sudo** - Use sudi on salt minion nodes (optional)
- **slave_name_regexp** - regexp for minion FQDNs (optional)
- **slave_direct_ssh** - if *False* then salt master is used as ssh proxy (optional)
- **get_ips_cmd** - salt command to get IPs of minions (optional)

Default services:

- cinder-api
- cinder-backup
- cinder-scheduler
- cinder-volume
- elasticsearch
- glance-api
- glance-registry
- grafana-server
- heat-api
- heat-engine
- horizon
- influxdb
- keystone

- kibana
- memcached
- mysql
- nagios3
- neutron-dhcp-agent
- neutron-l3-agent
- neutron-metadata-agent
- neutron-openvswitch-agent
- neutron-server
- nova-api
- nova-cert
- nova-compute
- nova-conductor
- nova-consoleauth
- nova-novncproxy
- nova-scheduler
- rabbitmq

Power management

libvirt [PowerDriver]

Libvirt driver.

Example configuration:

```
power_managements:
- driver: libvirt
  args:
    connection_uri: qemu+unix:///system
```

parameters:

- **connection_uri** - libvirt uri

ipmi [PowerDriver]

IPMI driver.

Example configuration:

```
power_managements:
- driver: ipmi
  args:
    mac_to_bmc:
      aa:bb:cc:dd:ee:01:
        address: 170.0.10.50
```

```
        username: admin1
        password: Admin_123
      aa:bb:cc:dd:ee:02:
        address: 170.0.10.51
        username: admin2
        password: Admin_123
    fqdn_to_bmc:
      node3.local:
        address: 170.0.10.52
        username: admin1
        password: Admin_123
```

parameters:

- **mac_to_bmc** - list of dicts where keys are the node MACs and values are the corresponding BMC configurations with the folowing fields:
  - **address** - ip address of IPMI server
  - **username** - IPMI user
  - **password** - IPMI password

Node discover

node_list [NodeDiscover]

Node list.

Allows specifying list of nodes in configuration.

Example configuration:

```
node_discover:
  driver: node_list
  args:
  - ip: 10.0.0.51
    mac: aa:bb:cc:dd:ee:01
    fqdn: node1.local
  - ip: 10.0.0.52
    mac: aa:bb:cc:dd:ee:02
    fqdn: node2.local
  - ip: 10.0.0.53
    mac: aa:bb:cc:dd:ee:03
    fqdn: node3.local
```

Service drivers

process [Service]

Service as process

"process" is a basic service driver that uses *ps* and *kill* in actions like kill / freeze / unfreeze. Commands for start / restart / terminate should be specified in configuration, otherwise the commands will fail at runtime.

Example configuration:

```
services:
  app:
    driver: process
    args:
      grep: my_app
      restart_cmd: /bin/my_app --restart
      terminate_cmd: /bin/stop_my_app
      start_cmd: /bin/my_app
      port: ['tcp', 4242]
```

parameters:

- **grep** - regexp for grep to find process PID
- **restart_cmd** - command to restart service (optional)
- **terminate_cmd** - command to terminate service (optional)
- **start_cmd** - command to start service (optional)
- **port** - tuple with two values - potocol, port number (optional)

linux_service [ServiceAsProcess]

Linux service

Service that is defined in init.d and can be controled by *service* CLI tool.

Example configuration:

```
services:
  app:
    driver: linux_service
    args:
      linux_service: app
      grep: my_app
      port: ['tcp', 4242]
```

parameters:

- **linux_service** - name of a service
- **grep** - regexp for grep to find process PID
- **port** - tuple with two values - potocol, port number (optional)

screen [ServiceAsProcess]

Service in Screen

This driver controls service that is started in a window of *screen* tool.

Example configuration:

```
services:
  app:
    driver: screen
    args:
      window_name: app
```

```
      grep: my_app
      port: ['tcp', 4242]
```

parameters:

- **window_name** - name of a service
- **grep** - regexp for grep to find process PID
- **port** - tuple with two values - potocol, port number (optional)

salt_service [ServiceAsProcess]

Salt service

Service that can be controled by *salt service.** commands.

Example configuration:

```
services:
  app:
    driver: salt_service
    args:
      salt_service: app
      grep: my_app
      port: ['tcp', 4242]
```

parameters:

- **salt_service** - name of a service
- **grep** - regexp for grep to find process PID
- **port** - tuple with two values - potocol, port number (optional)

pcs_service [ServiceAsProcess]

Service as a resource in Pacemaker

Service that can be controled by *pcs resource* CLI tool.

Example configuration:

```
services:
  app:
    driver: pcs_service
    args:
      pcs_service: app
      grep: my_app
      port: ['tcp', 4242]
```

parameters:

- **pcs_service** - name of a service
- **grep** - regexp for grep to find process PID
- **port** - tuple with two values - potocol, port number (optional)

pcs_or_linux_service [ServiceAsProcess]

Service as a resource in Pacemaker or Linux service

Service that can be controled by *pcs resource* CLI tool or linux *service* tool. This is a hybrid driver that tries to find service in Pacemaker and uses linux *service* if it is not found there.

Example configuration:

```
services:
  app:
    driver: pcs_or_linux_service
    args:
      pcs_service: p_app
      linux_service: app
      grep: my_app
      port: ['tcp', 4242]
```

parameters:

- **pcs_service** - name of a service in Pacemaker
- **linux_service** - name of a service in init.d
- **grep** - regexp for grep to find process PID
- **port** - tuple with two values - potocol, port number (optional)

API Reference

os_faults.**connect**(*cloud_config=None*, *config_filename=None*)
Connect to the cloud

Parameters

- **cloud_config** – dict with cloud and power management params
- **config_filename** – name of the file where to read config from

Returns CloudManagement object

os_faults.**discover**(*cloud_config*)
Connect to the cloud and discover nodes and services

Parameters **cloud_config** – dict with cloud and power management params

Returns config dict with discovered nodes/services

os_faults.**human_api**(*distractor*, *command*)
Execute high-level text command with specified destructor

Parameters

- **destructor** – library instance as returned by :connect: function
- **command** – text command

os_faults.**register_ansible_modules**(*paths*)
Registers ansible modules by provided paths

Allows to use custom ansible modules in NodeCollection.run_task method

Parameters **path** – list of paths to folders with ansible modules

class os_faults.api.cloud_management.**CloudManagement**

execute_on_cloud(*hosts*, *task*, *raise_on_error=True*)
Execute task on specified hosts within the cloud.

Parameters

- **hosts** – List of host FQDNs
- **task** – Ansible task
- **raise_on_error** – throw exception in case of error

Returns Ansible execution result (list of records)

get_nodes(*fqdns=None*)
Get nodes in the cloud

This function returns NodesCollection representing all nodes in the cloud or only those that has specified FQDNs. :param fqdns list of FQDNs or None to retrieve all nodes :return: NodesCollection

get_service(*name*)
Get service with specified name

Parameters **name** – name of the service

Returns Service

classmethod list_supported_networks()
Lists all networks supported by nodes returned by this driver

Returns [String] list of network names

classmethod list_supported_services()
Lists all services supported by this driver

Returns [String] list of service names

verify()
Verify connection to the cloud.

class os_faults.api.service.**Service**(*service_name*, *config*, *node_cls*, *cloud_management*, *ips=None*)

discover_nodes()
Discover nodes where this Service is running

Returns NodesCollection

freeze(*nodes=None*, *sec=None*)
Pause service execution

Send SIGSTOP to Service into network on all nodes or on particular subset. If sec is defined - it mean Service will be stopped for a wile.

Parameters

- **nodes** – NodesCollection
- **sec** – int

get_nodes()
Get nodes where this Service is running

Returns NodesCollection

kill(*nodes=None*)
Terminate Service abruptly on all nodes or on particular subset

Parameters **nodes** – NodesCollection

plug(*nodes=None*)
Plug Service into network on all nodes or on particular subset

Parameters **nodes** – NodesCollection

restart(*nodes=None*)
Restart Service on all nodes or on particular subset

Parameters **nodes** – NodesCollection

start(*nodes=None*)
Start Service on all nodes or on particular subset

Parameters **nodes** – NodesCollection

terminate(*nodes=None*)
Terminate Service gracefully on all nodes or on particular subset

Parameters **nodes** – NodesCollection

unfreeze(*nodes=None*)
Resume service execution

Send SIGCONT to Service into network on all nodes or on particular subset.

Parameters **nodes** – NodesCollection

unplug(*nodes=None*)
Unplug Service out of network on all nodes or on particular subset

Parameters **nodes** – NodesCollection

class os_faults.api.node_collection.**NodeCollection**(*cloud_management=None*, *hosts=None*)

connect(*network_name*)
Connect nodes to <network_name> network

Parameters **network_name** – name of network

disconnect(*network_name*)
Disconnect nodes from <network_name> network

Parameters **network_name** – name of network

oom()
Fill all node's RAM

pick(*count=1*)
Pick one Node out of collection

Returns NodeCollection consisting just one node

poweroff()
Power off all nodes abruptly

poweron()
Power on all nodes abruptly

reboot()
Reboot all nodes gracefully

reset()
Reset (cold restart) all nodes

revert(*snapshot_name*, *resume=True*)
Revert snapshot for all nodes

run_task(*task*, *raise_on_error=True*)
Run ansible task on node colection

Parameters

- **task** – ansible task as dict
- **raise_on_error** – throw exception in case of error

Returns AnsibleExecutionRecord with results of task

shutdown()
Shutdown all nodes gracefully

snapshot(*snapshot_name*, *suspend=True*)
Create snapshot for all nodes

Contributing

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

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

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

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

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Launchpad, not GitHub:

https://bugs.launchpad.net/os-faults

CHAPTER 2

Release Notes

CHANGES

- [docs] Add documentation for config file
- [CLI] Add os-faults discover command
- [Core] Predefined ips for services in config
- [Core] Allow adding services in config
- [Core] Services as drivers
- Add ‘slave_direct_ssh’ parameter to fuel driver

0.1.11

- Bump ansible version
- Add ConnectTimeout=60 to default ssh args
- Allow binding BMC by node fqdn in ipmi driver
- Add shutdown method to all power drivers
- Update requirements
- Add monitoring services to tcpcloud driver
- Load modules before creating ansible playbook

0.1.10

- Remove brackets from Service.GREP variables
- Add password auth to devstack and tcpcloud

- Add docs for drivers
- [tcpcloud] Additional configuration parameters
- Allow usage of multiple power drivers at once
- Add node_list driver
- Use filter in get_nodes

0.1.9

- Add ironic services to devstack driver
- Support for multi-node devstack
- Add neutron services to tcpcloud driver
- Allow filtering of node collection
- Skip monitoring nodes in TcpCloud

0.1.8

- New configuration parameters for TCPCloud driver
- Add glance-registry and cinder-backup to TcpCloud driver
- Add missing Nova serices to TcpCloud driver
- Search all nodes except master on TcpCloud driver
- Add Swift services to Fuel driver
- Add Cinder and Glance services to Fuel driver
- Add Nova services to fuel driver
- Add destroy before revert when using libvirt driver

0.1.7

- Add neutron agents to Fuel driver
- Add neutron-server
- Add snapshot and revert to libvirt driver

0.1.6

- Extend tcpcloud configuration
- Add start/terminate for all services
- Fix ssh to slave nodes
- Ironic service for Fuel driver

0.1.5

- Revert "Change requirement link to sphinxcontrib-programoutput"

0.1.4

- horizon and cinder-volume service for TCPCloud and Fuel
- Change requirement link to sphinxcontrib-programoutput
- Add Cinder services to TCPCloud and Fuel drivers

0.1.3

- Add Services to TCPCloud driver
- Add devstack services
- Support TCPCloud platform
- Add restart command for Fuel MySQL service
- Add support of standard operators to NodeCollection
- Add NodeCollection.get_fqdns method
- Fix RESTART_CMD for Fuel services
- Fix result logs in AnsibleRunner.execute
- Improve logging
- Fixed link to launchpad in docs
- Fix readthedocs build
- Move services to common module and reuse them in libvirt driver

0.1.2

- Fix AnsibleRunner.execute
- Return result from NodeCollection.run_task
- Fix release notes build for readthedocs

0.1.1

- Allow to run custom ansible modules on NodeCollection
- Fix docs build for readthedocs
- Raise ServiceError in case of unknown service requested
- Decorator for required variables
- Improvements for version
- Move plug/unplug bash command to ansible module
- Make libvirt-python an extra requirement

- Fuel services refactoring
- Fix doc for libvirt driver
- Rename private_key to private_key_file for devstack driver

0.1.0

- Configuration validation
- Update configuration
- Add usage and API reference into docs
- Improve help in command-line utility
- Load drivers dynamically
- Fix the issue with getting nodes by FQDNs
- Enable release notes translation
- Unit tests for os_faults.ansible.executor
- Rename and fix network-management commands in NodesCollection
- Add missing unit tests for libvirt and ipmi drivers
- Unit test for os_faults.cmd
- Small cleanup before the first release
- Move unit tests from os_faults/tests/ to os_faults/tests/unit/
- Improve logging for os_faults/drivers/fuel.py module
- Implement reboot action in Fuel driver
- Fix printing large output when getting hosts in debug mode
- Unit test for os_faults.utils
- Restart for nova-api, glance-api services
- Support nova compute/scheduller, neutron L2/L3 agent, heat api/engine
- Fix py35 tests
- Add human-friendly interface
- Unit tests for fuel network manegement ansible module
- Add memcached service operations
- Fix FuelService._run_task
- Fix RabbitMQ commands
- Allow to view logs from pytest
- Unit tests for devstack driver
- Unit tests for FuelService
- Tests for os_faults.connect
- Unit tests for FuelNodeCollection
- Use pytest to run test in tox

- Fix FuelManagement._retrieve_hosts_fqdn
- Unit tests for FuelManagement
- OS-Faults mechanism to close/open service ports on nodes
- Fix Fuel key distribution script
- Adding unit tests for IPMI driver
- Specify which key to use to access the cloud
- Adding unit tests for libvirt driver
- Docs cleanup
- SIGSTOP/SIGCONT signals for RabbitMQ/MySQL/Keystone-API/Nova-API services
- Read configuration from file or standard location
- Examples cleanup
- Fuel driver code improvements
- Rename os-failures into os-faults
- Update .gitreview post project rename
- Adding threads to IPMI driver
- Sigkill signal for RabbitMQ/MySQL/Keystone-API/Nova-API/Glance-API services
- Require Ansible 2 or higher
- Service discovering
- Adding threds to libvirt driver
- Add IPMI driver
- Adding a simple libvirt driver
- Fix project name in .gitreview
- Docs cleanup
- Run DevStack driver with sudo
- Add sample DevStack driver
- Fix formatting in readme
- Document use cases in readme file
- Add simple network management module
- Cleanup code and more logging
- Remove unnecessary utils module
- Enable logging
- Add network failures API
- Added get_nodes by fqdn
- Document API
- Added ky distribution script
- Fix PEP8 errors

- Cleanup Ansible executor
- Add connection verification
- Flatten client configuration
- Added power management
- Add node operations
- Service API prototype
- Added Ansible runner
- Tune tox.ini and update readme
- Initial Cookiecutter Commit

CHAPTER 3

Indices and Tables

- genindex
- modindex
- search

Python Module Index

o

Index

T

U

V