os-faults Documentation

Release 0.2.6.dev2

OpenStack Foundation

Mar 09, 2020

Contents

OpenStack fault-injection library

OS-Faults library provides an unified abstract API for performing destructive actions against OpenStack cloud. The library is extendable using drivers. Basic drivers for DevStack, Linux services and power management are already included.

CHAPTER 1

Contents

1.1 Quickstart

This section describes how to start using os-faults.

1.1.1 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]
```

The library relies on Ansible which needs to be installed separately. Please refer to [https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html] for installation instructions.

1.1.2 Basics

Configuration file

The cloud deployment configuration schema has simple YAML/JSON format:

```
cloud_management:
  driver: devstack
  args:
    address: 192.168.1.240
    auth:
      username: stack
      private_key_file: cloud_key
    iface: enp0s8

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

By default, the library reads configuration from a file with one of the following names: `os-faults.{json,yaml,yml}`. The configuration file is searched in one of default locations:

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

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

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

Execution

Establish a connection to the cloud and verify it:

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

or via CLI:

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

Make some destructive action:

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

or via CLI:

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

1.1.3 API

The library operates with different types of objects:

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

Human API

Human API is used to specify faults as normal English sentences.

```
import os_faults
cloud_management = os_faults.connect(config_filename='os-faults.yaml')
os_faults.human_api(cloud_management, '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:

```
cloud_management = os_faults.connect(cloud_config)
service = cloud_management.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. Container actions

Get a container and restart it:

```
cloud_management = os_faults.connect(cloud_config)
container = cloud_management.get_container(name='neutron_api')
container.restart()
```

Available actions:

- *start* - start Container
- *terminate* - terminate Container gracefully
- *restart* - restart Container

3. Node actions

Get all nodes in the cloud and reboot them:

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

Available actions:

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

4. 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 = cloud_management.get_nodes(fqdns=fqdns)
nodes.disconnect(network_name='management')
```

5. Operate with services

Restart a service on a single node:

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

1.1.4 Configuration specification

Configuration file contains the following parameters:

- cloud_management
- power_managements
- node_discover
- services
- containers

Each parameter specifies a driver or a list of drivers.

Example configuration:

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

(continues on next page)

(continued from previous page)

```
      grep: glance-api
      window_name: g-api
    hosts:
    - 192.168.1.240

containers:
  neutron_api:
    driver: docker_container
    args:
      container_name: neutron_api
```

cloud_management

This parameter specifies cloud management driver and its arguments. `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
    auth:
      username: ubuntu
      iface: enp0s3
```

Drivers can support discovering of cloud nodes. For example, `saltcloud` 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
    hosts:                # list of hosts 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 `hosts` parameter which controls discovering of hosts where the service is running. If `hosts` specified, then service discovering is disabled for this service and hosts specified in `hosts` will be used, otherwise, service will be searched across all nodes.

List of supported drivers for services: *Service drivers*

containers

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

```
containers:
  neutron_api:                    # name of the container
    driver: docker_container      # name of the container driver
    args:                         # arguments for the driver
      container_name: neutron_api
    hosts:                        # list of hosts where this container running
    - 192.168.1.240
```

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

List of supported drivers for containers: *Container drivers*

1.1.5 OS-Faults + Rally

Combination of OS-Faults and Rally gives a powerful tool to test OpenStack high availability and fail-over under the load.

Fault injection is implemented with help of Rally Fault Injection Hook. Following is an example of Rally scenario performing Keystone authentication with restart of one of Memcached services:

```
---
  Authenticate.keystone:
    -
      runner:
        type: "constant_for_duration"
        duration: 30
        concurrency: 5
      context:
        users:
          tenants: 1
          users_per_tenant: 1
      hooks:
        -
          name: fault_injection
          args:
            action: restart memcached service on one node
          trigger:
            name: event
            args:
              unit: iteration
              at: [100]
```

The moment of fault injection can be specified as iteration number or in time relative to the beginning of the test:

```
trigger:
  name: event
  args:
    unit: time
    at: [10]
```

Parameter *action* contains fault specification in human-friendly format, see *Human API* for details.

More on reliability testing of OpenStack:

- Reliability Test Plan in OpenStack performance documentation
- Keystone authentication with restart memcached report collected in OpenStack deployed by Fuel
- Introduction into reliability metrics video cast

1.2 CLI reference

1.2.1 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:
  devstack - DevStack driver
  docker_container - Docker container
  ipmi - IPMI power management driver
  libvirt - Libvirt power management driver
  node_list - Reads hosts from configuration file
  process - Service as process
  salt_service - Service in salt
  saltcloud - SaltCloud management driver
  system_service - System Service (systemd, upstart, SysV, etc.)
  universal - Universal cloud 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:
      devstack: cinder-api, cinder-scheduler, cinder-volume, etcd, glance-api,
                heat-api, heat-engine, keystone, memcached, mysql,
                neutron-dhcp-agent, neutron-l3-agent,
                neutron-meta-agent, neutron-openvswitch-agent,
                neutron-server, nova-api, nova-compute,
                nova-scheduler, placement-api, rabbitmq
      saltcloud: cinder-api, cinder-backup, cinder-scheduler, cinder-volume,
                 elasticsearch, glance-api, glance-glare,
                 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
```

(continues on next page)

(continued from previous page)

```
      universal: /no built-in support/

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
      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
      stress - Stress node OS and hardware
    service is one of supported by driver:
      devstack: cinder-api, cinder-scheduler, cinder-volume, etcd, glance-api,
                heat-api, heat-engine, keystone, memcached, mysql,
                neutron-dhcp-agent, neutron-l3-agent,
                neutron-meta-agent, neutron-openvswitch-agent,
                neutron-server, nova-api, nova-compute,
                nova-scheduler, placement-api, rabbitmq
      saltcloud: cinder-api, cinder-backup, cinder-scheduler, cinder-volume,
                 elasticsearch, glance-api, glance-glare,
                 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
      universal: /no built-in support/

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:
      devstack: all-in-one
      saltcloud: /no built-in support/
      universal: /no built-in support/
    service is one of supported by driver:
```

(continues on next page)

(continued from previous page)

```
      devstack: cinder-api, cinder-scheduler, cinder-volume, etcd, glance-api,
                heat-api, heat-engine, keystone, memcached, mysql,
                neutron-dhcp-agent, neutron-l3-agent,
                neutron-meta-agent, neutron-openvswitch-agent,
                neutron-server, nova-api, nova-compute,
                nova-scheduler, placement-api, rabbitmq
      saltcloud: cinder-api, cinder-backup, cinder-scheduler, cinder-volume,
                 elasticsearch, glance-api, glance-glare,
                 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
      universal: /no built-in support/

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/
```

1.2.2 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 output config file
  drivers   List os-faults drivers
  nodes     List cloud nodes
  verify    Verify connection to the cloud
```

1.2.3 os-faults verify

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

  Verify connection to the cloud

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

1.2.4 os-faults discover

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

  Discover services/nodes and save them to output config file

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

1.2.5 os-faults nodes

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

  List cloud nodes

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

1.2.6 os-faults drivers

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

  List os-faults drivers

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

1.3 Drivers

1.3.1 Cloud management

universal [CloudManagement]

Universal cloud management driver

This driver is suitable for the most abstract (and thus universal) case. The driver does not have any built-in services nor node discovery capabilities. All services need to be listed explicitly in a config file. Node list is specified using *node_list* node discovery driver.

Example of multi-node configuration:

Note that in this configuration a node discovery driver is required.

```
cloud_management:
  driver: universal

node_discover:
  driver: node_list
  args:
```

(continues on next page)

(continued from previous page)

```
    - ip: 192.168.5.149
      auth:
        username: developer
        private_key_file: cloud_key
        become_password: my_secret_password
    - ip: 192.168.5.150
      auth:
        username: developer
        private_key_file: cloud_key
        become_password: my_secret_password
```

devstack [CloudManagement, NodeDiscover]

Driver for DevStack.

This driver requires DevStack installed with Systemd (USE_SCREEN=False). Supports discovering of node MAC addresses.

Example configuration:

```
cloud_management:
  driver: devstack
  args:
    address: 192.168.1.10
    auth:
      username: ubuntu
      password: ubuntu_pass
      private_key_file: ~/.ssh/id_rsa_devstack
    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)
- **iface** - network interface name to retrieve mac address (optional)

Default services:

- cinder-api
- cinder-scheduler
- cinder-volume
- etcd
- glance-api
- heat-api
- heat-engine
- keystone
- memcached

- mysql
- neutron-dhcp-agent
- neutron-l3-agent
- neutron-meta-agent
- neutron-openvswitch-agent
- neutron-server
- nova-api
- nova-compute
- nova-scheduler
- placement-api
- rabbitmq

saltcloud [CloudManagement, NodeDiscover]

Driver for OpenStack cloud managed by Salt.

Supports discovering of slave nodes.

Example configuration:

```
cloud_management:
  driver: saltcloud
  args:
    address: 192.168.1.10
    auth:
      username: root
      password: root_pass
      private_key_file: ~/.ssh/id_rsa_tcpcloud
    slave_auth:
      username: ubuntu
      password: ubuntu_pass
      become_username: root
    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 sudo 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)
- **serial** - how many hosts Ansible should manage at a single time. (optional) default: 10

Default services:

- cinder-api
- cinder-backup
- cinder-scheduler
- cinder-volume
- elasticsearch
- glance-api
- glance-glare
- 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

1.3.2 Power management

libvirt [PowerDriver]

Libvirt driver.

Example configuration:

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

parameters:

- **connection_uri** - libvirt uri

Note that Libvirt domain name should be specified as node attribute. Refer to node discover (node_list driver) for details.

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

1.3.3 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
    libvirt_name: node1
  - ip: 192.168.1.50
    mac: aa:bb:cc:dd:ee:02
    fqdn: node2.local
    auth:
      username: user1
      password: secret1
      jump:
        host: 10.0.0.52
        username: ubuntu
        private_key_file: /path/to/file
  - ip: 10.0.0.53
    mac: aa:bb:cc:dd:ee:03
    fqdn: node3.local
    become_password: my_secret_password
```

node parameters:

- **ip** - ip/host of the node
- **mac** - MAC address of the node (optional). MAC address is used for libvirt driver.
- **fqdn** - FQDN of the node (optional). FQDN is used for filtering only.
- **libvirt_name** - Libvirt domain name (optional).
- **auth - SSH related parameters (optional):**
  - **username** - SSH username (optional)
  - **password** - SSH password (optional)
  - **private_key_file** - SSH key file (optional)
  - **become_password** - privilege escalation password (optional)
  - **jump - SSH proxy parameters (optional):**
    - **host** - SSH proxy host
    - **username** - SSH proxy user
    - **private_key_file** - SSH proxy key file (optional)

1.3.4 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, 'ingress']
```

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 or three values - protocol, port number, direction (optional)

Note that network operations are based on iptables. They are applied to the whole host and not restricted to a single process.

system_service [ServiceAsProcess]

System service

This is universal driver for any system services supported by Ansible (e.g. systemd, upstart). Please refer to Ansible documentation http://docs.ansible.com/ansible/latest/service_module.html for the whole list.

Example configuration:

```
services:
  app:
    driver: system_service
    args:
      service_name: app
      grep: my_app
      port: ['tcp', 4242, 'ingress']
```

parameters:

- **service_name** - name of a service
- **grep** - regexp for grep to find process PID
- **port** - tuple with two or three values - protocol, port number, direction (optional)

salt_service [ServiceAsProcess]

Salt service

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

Example configuration:

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

parameters:

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

1.3.5 Container drivers

docker_container [Container]

Docker container

This is docker container driver for any docker containers supported by Ansible. Please refer to Ansible documentation https://docs.ansible.com/ansible/latest/modules/docker_container_module.html for the whole list.

Example configuration:

```
containers:
  app:
    driver: docker_container
    args:
      container_name: app
```

parameters:

- **container_name** - name of the container

1.4 API Reference

os_faults.**connect**(*cloud_config=None*, *config_filename=None*)

Connects 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**(*cloud_management*, *command*)
Executes a command written as English sentence

Parameters

- **cloud_management** – 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 **paths** – 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_container(*name*)
Get container with specified name

Parameters **name** – name of the container

Returns Container

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*, *hosts=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.container.**Container**(*container_name*, *config*, *node_cls*, *cloud_management*, *hosts=None*)

discover_nodes()
Discover nodes where this Container is running

Returns NodesCollection

get_nodes()
Get nodes where this Container is running

Returns NodesCollection

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

Parameters **nodes** – NodesCollection

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

Parameters **nodes** – NodesCollection

terminate(*nodes=None*)
Terminate Container gracefully 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

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

stress(*target*, *duration=None*)
Stress node OS and hardware

1.5 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

Note that the primary repo is https://opendev.org/performa/os-faults/ Repos located at GitHub are mirrors and may be out of sync.

Project bug tracker is Launchpad:

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

CHAPTER 2

Release Notes

2.1 CHANGES

- Add a job to mirror the code from OpenDev to Github
- Split requirements for py27 and >py36

2.1.1 0.2.5

- Update project links and package metadata
- Update jsonschema according to requirements

2.1.2 0.2.4

- Document how to use libvirt driver
- Fix reboot command
- Remove oom method since it has never been implemented
- Fix representation of signal.SIG* constants

2.1.3 0.2.3

- Re-implement network faults using standard iptables module
- Update constraints handling

2.1.4 0.2.2

- Fix service discovery
- Fix devstack and py27 jobs
- Update repo address in readme file
- OpenDev Migration Patch

2.1.5 0.2.1

- Declare support of python 3.7
- add python 3.7 unit test job
- Unpin pytest and its plugins
- Fix key management in devstack job
- Add DevStack plugin

2.1.6 0.2.0

- Use bindep to specify binary dependencies
- Update hacking version
- Add integration test for devstack cloud management driver
- Tests cleanup
- Rename ‘tcpcloud‘ driver into ‘saltcloud‘
- Unify auth parameters between drivers
- Specify auth parameters in devstack tests
- Remove deprecated methods
- Simplify universal driver
- Modify integration tests job to run on DevStack
- Change openstack-dev to openstack-discuss
- Remove old drivers
- Do not link with Ansible code
- Fix README markup
- Optimizing the safety of the http link site in HACKING.rst
- Move Zuul jobs from global project-config to the repo

2.1.7 0.1.18

- Specify webhook id for documentation build
- Update containers documentation
- Add docker containers support

- Typo fix in universal driver documantation
- fix tox python3 overrides
- Replaces yaml.load() with yaml.safe_load()
- Zuul: Remove project name
- Add section describing OS-Faults and Rally integration
- Update documentation and examples

2.1.8 0.1.17

- Add integration job for Zuul
- Fix custom module support in Ansible 2.4
- Add glance-glare service to the tcpdriver

2.1.9 0.1.16

- Fix for Ansible >= 2.4.0
- Add logging into human API call
- Universal cloud management driver
- Universal driver for any system services (systemd, upstart, etc.)

2.1.10 0.1.15

- Upper constraint ansible to fix library
- Move all service drivers under services/ package
- Refactor services module
- Make privilege escalation password configurable
- Group drivers modules by type
- Fix process name patterns in DevStack driver
- Implement stress injection

2.1.11 0.1.14

- Add Devstack Systemd driver
- Switch from oslosphinx to openstackdocstheme

2.1.12 0.1.13

- [trivial] Several typos fix in tcpcloud driver
- [trivial] Several typo fixes in devstack driver
- Skip checking of Devstack hosts

2.1.13 0.1.12

- Auth configuration for each node
- Allow adding libvirt name for node
- Add serial parameter to cloud drivers
- [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

2.1.14 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

2.1.15 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

2.1.16 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

2.1.17 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

2.1.18 0.1.7

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

2.1.19 0.1.6

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

2.1.20 0.1.5

- Revert “Change requirement link to sphinxcontrib-programoutput”

2.1.21 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

2.1.22 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

2.1.23 0.1.2

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

2.1.24 0.1.1

- Allow to run custom ansible modules on NodeCollection
- Fix docs build for readthedocs

CHAPTER 3

Indices and Tables

- genindex
- modindex
- search

Python Module Index

Index

S

T

U

V