Red Hat OpenStack Platform 11

OpenStack Integration Test Suite Guide

Introduction to the OpenStack Integration Test Suite

Last Updated: 2017-09-04

Red Hat OpenStack Platform 11 OpenStack Integration Test Suite Guide

Introduction to the OpenStack Integration Test Suite

OpenStack Team
rhos-docs@redhat.com

Abstract

This guide provides instructions to install, configure and manage the OpenStack Integration Test Suite in a Red Hat OpenStack Platform environment.

Table of Contents

PREFACE

This guide provides instructions to install, configure and manage the OpenStack Integration Test Suite in a Red Hat OpenStack Platform environment.

CHAPTER 1. INTRODUCTION

As OpenStack consists of many different projects, it is important to test their interoperability within your OpenStack cluster. The OpenStack Integration Test Suite (tempest) automates the integration testing of your Red Hat OpenStack Platform deployment. Running tests ensures your cluster is working as expected, and can also provide early warning of potential problems, especially after an upgrade.

The Integration Test Suite contains tests for OpenStack API validation and scenario testing, as well as unit testing for self-validation. It performs black box testing using the OpenStack public APIs, with ostestr as the test runner.

CHAPTER 2. OPENSTACK INTEGRATION TEST SUITE TESTS

The OpenStack Integration Test Suite has many applications. It acts as a gate for commits to the OpenStack core projects, it can stress test to generate load on a cloud deployment, and it can perform CLI tests to check the response formatting of the command line. However, the functionality that we are concerned with are the `scenario tests` and `API tests`. These tests are run against your OpenStack cloud deployment. The following sections briefly describe each of these tests and how you can implement them.

2.1. SCENARIO TESTS

Scenario tests simulate a typical end user action workflow to test the integration points between services. The testing framework conducts the set up, tests the integration between services, and then it is torn down. You should tag the tests with which services they relate to, to make it clear which client libraries the test uses.

A scenario is based on a use case, for example:

- Upload an image to the Image Service
- Deploy an instance from the image
- Attach a volume to the instance
- Create a snapshot of the instance
- Detach the volume from the instance

2.2. API TESTS

API tests validate the OpenStack API. Tests use the OpenStack Integration Test Suite implementation of the Openstack API. Both valid and invalid JSON can be used to make sure error responses are valid. Tests can be run independently, and do not rely on the state left by the previous test.

CHAPTER 3. INSTALLING THE OPENSTACK INTEGRATION TEST SUITE

This section describes how to install the OpenStack Integration Test Suite either with the director or with a manual installation.

3.1. USING THE DIRECTOR

Edit the **`undercloud.conf`** file, located in the **`stack`** user's home directory. By default, **`enable_tempest`** is set to **`false`**. Change this to **`true`**.

```
enable_tempest = true
```

You are now ready to install the **`tempest`** packages and plug-ins, described in Section 3.3, "Installing the OpenStack Integration Test Suite Packages".

3.2. PREPARING A MANUAL INSTALLATION

To run the OpenStack Integration Test Suite, you need to first install the necessary packages and create a configuration file that will tell the Integration Test Suite where to find the various OpenStack services and other testing behaviour switches.

On the controller node, as a **`root`** user, create a virtual machine called **`tempest`**. This machine must run Red Hat Enterprise Linux 7.3 or greater. It also needs to be able to reach the cloud, but it does not have to be part of the cloud. For more information, see Creating Guests with Virt-Manager.

Also, before installing the OpenStack Integration Test Suite, the following networks are required within your Red Hat OpenStack Platform environment:

- An external network which can provide floating IP
- A private network

These networks must be connected through a router.

Create the private network:

```
$ openstack network create <network_name> --share
$ openstack subnet create <subnet_name> --subnet-range <address/prefix>
$ openstack router create <router_name>
$ openstack router add subnet <router_name> <subnet_name>
```

Create the public network:

```
$ openstack network create <network_name> --external \
  --provider-network-type flat
$ openstack subnet create <subnet_name> --subnet-range <address/prefix> \
  --gateway <default_gateway> --no-dhcp --network <network_name>
$ openstack router set <router_name> --external_gateway
<public_network_name>
```

You are now ready to install and configure the OpenStack Integration Test Suite within the `tempest` virtual machine. For more information, see Section 3.3, “Installing the OpenStack Integration Test Suite Packages”.

3.3. INSTALLING THE OPENSTACK INTEGRATION TEST SUITE PACKAGES

1. Install the packages related to the OpenStack Integration Test Suite:

   ```
   # yum -y install openstack-tempest
   ```

   However, this command will not install any tempest plug-ins. These have to be installed manually, depending on your OpenStack installation.

2. View all the OpenStack components installed on your machine:

   ```
   # openstack-status
   ```

3. Install the appropriate tempest plug-in for each component you have, for example:

   ```
   # yum install python-glance-tests python-keystone-tests python-
   horizon-tests-tempest python-neutron-tests python-cinder-tests
   python-nova-tests python-swift-tests python-ceilometer-tests python-
   gnocchi-tests python-aodh-tests
   ```

   See Section 3.3.1, “List of Tempest Plug-in Packages” for a list of the tempest plug-ins for each OpenStack component.

3.3.1. List of Tempest Plug-in Packages

Component	Package Name
aodh	python-aodh-tests
ceilometer	python-ceilometer-tests
cinder	python-cinder-tests
designate	python-designate-tests-tempest
glance	python-glance-tests
gnocchi	python-gnocchi-tests
heat	python-heat-tests
horizon	python-horizon-tests-tempest
ironic	python-ironic-tests

Component	Package Name
ironic-inspector	python-ironic-inspector-tests
keystone	python-keystone-tests
magnum	python-magnum-tests
manila	python-manila-tests
mistral	python-mistral-tests
murano	python-murano-tests
neutron	python-neutron-tests
neutron-fwaas	python-neutron-fwaas-tests
neutron-lbaas	python-neutron-lbaas-tests
neutron-vpnaas	python-neutron-vpnaas-tests
nova	python-nova-tests
sahara	python-sahara-tests-tempest
swift	python-swift-tests
trove	python-trove-tests
watcher	python-watcher-tests-tempest
zaqar	python-zaqar-tests

CHAPTER 4. CONFIGURING THE OPENSTACK INTEGRATION TEST SUITE

4.1. CREATING A WORKSPACE

1. Source the admin credentials:
   In the undercloud:

   ```
   # source stackrc
   ```

   Or in the overcloud:

   ```
   # source overcloudrc
   ```

2. Initialize **tempest**:

   ```
   # tempest init mytempest
   # cd mytempest
   ```

   This creates a tempest workspace named **mytempest**.

   You can view a list of existing workspaces:

   ```
   # tempest workspace list
   ```

3. Generate the **etc/tempest.conf** file:

   ```
   # discover-tempest-config --deployer-input ~/tempest-deployer-
   input.conf \
   --debug --create identity.uri $OS_AUTH_URL identity.admin_password \
   $OS_PASSWORD --network-id <UUID>
   ```

   uuid is the UUID of the external network.

   discover-tempest-config was formerly called **config_tempest.py** and takes the same parameters. It is provided by **python-tempestconf** which is installed as a dependency of **openstack-tempest**.

4.2. VERIFYING YOUR TEMPEST CONFIGURATION

Verify your current tempest configuration:

```
# tempest verify-config -o <output>
```

output is the output file where your updated configuration is written. This is different from your original configuration file.

4.3. CHANGING THE LOGGING CONFIGURATION

The default location for log files is the **logs** directory within your tempest workspace.

To change this directory, in **`tempest.conf`**, under the **`[DEFAULT]`** section, set **`log_dir`** to the desired directory:

```
[DEFAULT]
log_dir = <directory>
```

If you have your own logging configuration file, in **`tempest.conf`**, under the **`[DEFAULT]`** section, set **`log_config_append`** to your file:

```
[DEFAULT]
log_config_append = <file>
```

If this is set, all other logging configuration in **`tempest.conf`** will be ignored, including **`log_dir`**.

4.4. CONFIGURING MICROVERSION TESTS

The OpenStack Integration Test Suite provides stable interfaces to test the API microversions. This section describes how to implement microversion tests using these interfaces.

You first need to configure options in the **`tempest.conf`** configuration file to specify the target microversions. This is to make sure that the supported microversions match the microversions used in the OpenStack cloud. You can run multiple microversion tests in a single Integration Test Suite operation by specifying a range of target microversions.

For example, to limit the range of microversions for the **`compute`** service, in the **`[compute]`** section of your configuration file, assign values to the **`min_microversion`** and **`max_microversion`** parameters:

```
[compute]
min_microversion = 2.14
max_microversion = latest
```

CHAPTER 5. USING OSTESTR TO RUN TEMPEST

Ostestr is an OpenStack wrapper for the `testr` test runner.

5.1. RUNNING SMOKE TESTS

Smoke testing is a type of preliminary testing which only covers the most important functionality. While they are not comprehensive, running smoke tests can save time if they do identify a problem.

To run the smoke tests:

```
# ostestr '.*smoke'
```

5.2. PASSING TESTS USING WHITELIST FILES

A whitelist file is a file which contains regular expressions to select tests to include. Regular expressions are separated by a newline.

To use a whitelist file:

```
# ostestr --whitelist-file <whitelist_file>
```

Alternatively:

```
# ostestr -w <whitelist_file>
```

5.3. SKIPPING TESTS USING BLACKLIST FILES

A blacklist file is a file which contains regular expressions to select tests to exclude. Regular expressions are separated by a newline.

To use a blacklist file:

```
# ostestr --blacklist-file <blacklist_file>
```

Alternatively:

```
# ostestr -b <blacklist_file>
```

5.4. RUNNING TESTS IN PARALLEL CONCURRENTLY, OR SERIALLY

Run the tests serially:

```
# ostestr --serial
```

Run the tests in parallel (this is the default):

```
# ostestr --parallel
```

Specify the number of workers to use when running tests in parallel:

```
# ostestr --concurrency <workers>
```

Alternatively:

```
# ostestr -c <workers>
```

By default, this is set to the number of CPUs.

CHAPTER 6. CLEANING TEMPEST RESOURCES

After running **`tempest`**, there will be files, users and tenants created in the testing process that need to be deleted. Being able to self clean is one of the design principles of **`tempest`**.

6.1. PERFORMING A CLEAN UP

First you must initialize the saved state. This creates the file **`saved_state.json`**, which prevents the cleanup from deleting objects that need to be kept. Typically you would run cleanup with **`--init-saved-state`** prior to a tempest run. If this is not the case, **`saved_state.json`** must be edited to remove objects you want cleanup to delete.

```
# tempest cleanup --init-saved-state
```

Run the cleanup:

```
# tempest cleanup
```

6.2. PERFORMING A DRY RUN

A dry run lists the files that would be deleted by a cleanup, but does not delete any files. The files are listed in the **`dry_run.json`** file.

```
# tempest cleanup --dry-run
```

6.3. DELETING TEMPEST OBJECTS

Delete users and tenants created by tempest:

```
# tempest cleanup --delete-tempest-conf-objects
```