# Red Hat OpenStack Platform 9
# OpenStack Integration Test Suite Guide

---

Introduction to the OpenStack Integration Test Suite

OpenStack Team

# Red Hat OpenStack Platform 9 OpenStack Integration Test Suite Guide

## Introduction to the OpenStack Integration Test Suite

OpenStack Team
rhos-docs@redhat.com

## Legal Notice



## 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

OpenStack Integration Test Suite (tempest) is a set of integration tests that are to be run against a live OpenStack cluster. The Integration Test Suite has a set of tests for OpenStack API validation, scenarios and other specific tests useful in validating your Red Hat OpenStack Platform deployment.

The OpenStack Integration Test Suite provides a stable library interface that provides external tools or test suites an interface for reusing pieces of **`tempest`** code. Any public interface that lives in **`tempest/lib`** in the OpenStack Integration Test Suite repository is treated as a stable public interface and it should be safe to external consume that. Every effort goes into maintaining backwards compatibility with any change. The library is self contained and does not have any dependency on other OpenStack Integration Test Suite internals outside of library.

Red Hat OpenStack Platform components have implemented API microversions. The Integration Test Suite provides stable interfaces to support to test the API microversions. Based on the microversion range coming from the combination of configuration options and each test case, APIs request is made with the selected microversion.

# CHAPTER 2. OPENSTACK INTEGRATION TEST SUITE SCRIPTS

The OpenStack Integration Test Suite provides some scripts you can use to install, and configure it and the **`run_tests.sh`** script allows you to run tests on your OpenStack deployment. You can find the entire list of the tools available for the OpenStack Integration Test Suite service in the **`/tools`** directory. The following is a brief summary on each of these scripts. You can get more details by running them with the option **`-h`**:

- **`configure-tempest-directory`** - The Integration Test Suite expects the tests it discovers to be in the current working directory. This script will add symbollic links and other files to the current directory so that you can run **`tempest`** against the cloud. You can run this script in different directories to run **`tempest`** against different clouds from the same machine.
- **`config_tempest.py`** - The OpenStack Integration Test Suite can be difficult to configure. You need to set various options, and different users, tenants, flavors, images, need to be created. The **`config_tempest.py`** script makes the configuration easier. Using the **`--create`** option, you can create the necessary resources and **`admin`** credentials. Red Hat recommends that you do not include the **`admin`** credentials while running the script against an enterprise cloud.
- **`run-tests.sh`** - This script is a wrapper around the **`testr`** test runner that provides a file with a set of tests to skip as well as producing optional **`xunit`** output. All the long arguments to the **`testr`** can be used with this script. See the **`testr`** documentation for more information.

For more details on these scripts, see List and Description of the OpenStack Integration Test Suite Scripts

# CHAPTER 3. OPENSTACK INTEGRATION TEST SUITE TESTS

The OpenStack Integration Test Suite is designed to be useful for a large number of different environments, for example, gating commits to OpenStack core projects, to validate OpenStack cloud implementations for both correctness, as well as a burn in tool for OpenStack clouds. As such the OpenStack Integration Test Suite tests come in many flavors, each with their own rules and guidelines. The different types of tests are as follows:

- Scenario Tests
- API Tests
- Stress Tests

While these tests are run against your OpenStack cloud deployment, the OpenStack Integration Test Suite also provides **`unit tests`** that you can run to verify your **`tempest`** implementation. The following sections briefly describes each of the tests and how you can implement them.

## 3.1. SCENARIO TESTS

Scenario tests are *through path* tests of OpenStack function. They are typically a series of steps where complicated state requiring multiple services is set up exercised, and torn down. They ideally involve the integration between multiple OpenStack services to exercise the touch points between them.

**Note**

Scenario tests should not use the existing python clients for OpenStack, but should instead use the **`tempest`** implementations of clients.

Any scenario test should have a real-life use case. For example, as an operator I want to start with a blank environment and then perform the following functions:

- Upload an Image Service image
- Deploy an instance from it
- Log in to the guest using **`ssh`**
- Create a snapshot of the instance

**Scope** - Scenario tests used for testing the integration between OpenStack projects is the core purpose of the OpenStack Integration Test Suite. Scenario tests should always use the OpenStack Integration Test Suite implementation of the OpenStack API, to ensure that bugs are not hidden by the official clients. Tests should be tagged with which services they exercise, as determined by which client libraries are used directly by the test.

**Example of a good scenario test** - Be specific in the interactions, while testing for an interaction between two or more services. A giant “this is my data center” smoke test is hard to debug when it goes wrong. A flow of interactions between Image Service and Compute, as mentioned in the introduction, is a good example. Especially if it involves a repeated interaction when a resource is setup, modified, detached, and then reused later again.

## 3.2. API TESTS

API tests are validation tests for the OpenStack API.

**Note**

API tests should not use the existing python clients for OpenStack, but should instead use the **`tempest`** implementations of clients.

This allows you to test JSON. Having raw clients also lets you pass invalid JSON to the APIs and see the results, something you could not get with the native clients.

**Scope** - API tests should always use the OpenStack Integration Test Suite implementation of the OpenStack API, to ensure that bugs are not hidden by the official clients. They should test specific API calls, and can build up complex state if it is needed for the API call to be meaningful. They should send not only good data, but bad data at the API and look for error codes. They should all be able to be run on their own, not depending on the state created by a previous test.

## 3.3. STRESS TESTS

Stress tests are designed to stress an OpenStack environment by running a high workload against it and seeing what breaks. The stress test framework runs several test jobs in parallel and can run any existing test in the OpenStack Integration Test Suite as a stress job.

**Environment** - This particular framework assumes your working Compute cluster understands **nova** API 2.0. The stress tests can read the logs from the cluster. To enable this, you have to provide the hostname to call **`nova-manage`** and the private key and user name for **`ssh`** to the cluster in the **`[stress]`** section of **`tempest.conf`**. You also need to provide the location of the log files:

```
target_logfiles = “regexp to all log files to be checked for errors”
target_private_key_path
= “private ssh key for controller and log file nodes” target_ssh_user =
“username for
controller and log file nodes” target_controller = “hostname or ip of
controller node (for
nova-manage) log_check_interval = “time between checking logs for
errors (default 60s)”
```

To activate logging on your console, make sure that you activate **`use_stderr`** in **`tempest.conf`** or use the default **`logging.conf.sample`** file.

**Running default stress test set** - The stress test framework can automatically discover test inside the OpenStack Integration Test Suite. All test flags with the **`@stresstest`** decorator will be executed. In order to use this discovery, you have to install the OpenStack Integration Test Suite CLI, navigate to the **`tempest root`** directory and execute the following:

```
# tempest run-stress -a -d 30
```

Run the sample test to test installation:

```
# tempest run-stress -t tempest/stress/etc/server-create-destroy-
test.json -d 30
```

This sample test tries to create a few virtual machines and kill a few virtual machines.

Sometimes the tests don't finish, or there are failures. In these cases, you may want to clean out the **nova** cluster. There are some scripts provided in the tools subdirectory. You can use the following script to destroy any keypairs, floating IP addresses, and servers:

```
# tempest/stress/tools/cleanup.py
```

## 3.4. UNIT TESTS

Unit tests are the self checks for the OpenStack Integration Test Suite. They provide functional verification and regression checking for the internal components of **`tempest`**. They should be used to just verify that the individual pieces of the OpenStack Integration Test Suite are working as expected. These tests can be run by specifying the test discovery path as follows:

```
# OS_TEST_PATH=./tempest/tests testr run --parallel
```

By setting the **`OS_TEST_PATH`** to **`./tempest/tests`**, you are specifying that the test discover should only be run on the unit test directory. The default value of **`OS_TEST_PATH`** is **`OS_TEST_PATH=./tempest/test_discover`** which will only run test discover on the OpenStack Integration Test Suite.

Alternatively, you can use the **`run_tests.sh`** script which will create a **`venv`** and run the unit tests. There are also the **`py27`** and **`py34`** tox jobs which will run the unit tests with the corresponding version of Python.

**Scope** - The unit tests exist to make sure that the mechanisms that used in the OpenStack Integration Test Suite to are valid and remain functional. They are only for self validation of the OpenStack Integration Test Suite. Unit tests should not require an external service to be running or any extra configuration to run. Any state that is required for a test should either be mocked out or created in a temporary test directory. (see test_wrappers.py for an example of using a temporary test directory)

# CHAPTER 4. INSTALL OPENSTACK INTEGRATION TEST SUITE

You can install the OpenStack Integration Test Suite using the director by enabling the **`enable_tempest`** in the Red Hat OpenStack Platform director undercloud. The **`enable_tempest`** parameter defines whether to install the validation tools. By default, this value is set to **`false`**.

For more information on the procedure to install the OpenStack Integration Test Suite manually, see Install OpenStack Integration Test Suite.

# CHAPTER 5. CONFIGURE THE OPENSTACK INTEGRATION TEST SUITE

The OpenStack Integration Test Suite provides the **`config_tempest.py`** script to configure the various options, for example, the users, tenants, flavors, images that need to be created. For more information, see **`/usr/share/openstack-tempest-10.0.0/tools/config_tempest.py`**.

For more information to configure the OpenStack Integration Test Suite manually to work with your cloud deployment, see Configure the OpenStack Integration Test Suite.

# APPENDIX A. LIST AND DESCRIPTION OF THE OPENSTACK INTEGRATION TEST SUITE SCRIPTS

The following sections list the **`tempest`** scripts available in **`/usr/share/openstack-tempest-10.0.0/tools/`** on the controller:

- **`configure-tempest-directory`**
- **`config_tempest.py`**
- **`run_tempest.sh`**
- **`run_tests.sh`**

# APPENDIX B. CONFIGURE AND IMPLEMENT 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.

Before you begin, you need to configure options in your **`tempest.conf`** configuration file to specify test target microversions to make sure that the supported mircoversions are the same as used in the OpenStack clouds. You can operate multiple microversion tests in a single Integration Test Suite operation by specifying a range of test target microversions.

In the **`[compute]`** section of your configuration file, add the values for the **`min_microversion`** and **`max_microversion`** parameters. For example,

```
[compute]
min_microversion = None
max_microversion = latest
```