Red Hat OpenStack Platform 13

OpenStack Integration Test Suite Guide

Introduction to the OpenStack Integration Test Suite

Last Updated: 2022-10-13

Red Hat OpenStack Platform 13 OpenStack Integration Test Suite Guide

Introduction to the OpenStack Integration Test Suite

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This guide contains information about installing, configuring, and managing the OpenStack Integration Test Suite in a Red Hat OpenStack Platform environment.

Table of Contents

PREFACE

This guide contains information about installing, configuring, and managing 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 the interoperability of the projects within your OpenStack cluster. The OpenStack Integration Test Suite (tempest) automates the integration testing of your Red Hat OpenStack Platform deployment. Running tests ensures that 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. The Integration Test Suite performs black box testing using the OpenStack public APIs, with tempest 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 contain information about implementing each of these tests.

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 configuration, tests the integration between services, and is then removed automatically. Tag the tests with the services that 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. You can use both valid and invalid JSON to ensure that error responses are valid. You can run tests independently and you do not have to rely on the previous test state.

CHAPTER 3. INSTALLING THE OPENSTACK INTEGRATION TEST SUITE

This section contains information about installing 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 home directory of the **stack** user. Ensure that the **enable_tempest** parameter is set to **true**:

```
enable_tempest = true
```

If your undercloud is already installed, you can edit the **undercloud.conf** file and then run the **openstack undercloud install** command to include the extra configuration in the undercloud:

```
$ openstack undercloud upgrade
```

You are now ready to install the **tempest** packages and plugins, 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 must first install the necessary packages and create a configuration file that informs the Integration Test Suite where to find the various OpenStack services and other testing behaviour switches.

To install the OpenStack Integration Test Suite, the following networks must be available 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. Specify the following options according to your network deployment:

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

Create the public network. Specify the following options according to your network deployment:

```
$ 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:

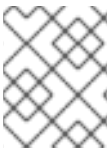
   ```
   $ sudo yum -y install openstack-tempest
   ```

   This command does not install any tempest plugins. You must install the plugins manually, depending on your OpenStack installation.

2. Install the appropriate tempest plugin for each component in your environment. For example, run the following command to install the keystone, horizon, neutron, cinder, and telemetry plugins:

   ```
   $ sudo yum install python-keystone-tests-tempest python-horizon-tests-tempest python-neutron-tests-tempest python-cinder-tests-tempest python-telemetry-tests-tempest
   ```

   See Section 3.3.1, "List of Tempest Plug-in Packages" for a list of the tempest plugins for each OpenStack component.



NOTE

You can also install the **openstack-tempest-all** package. This package contains all of the tempest plugins.

3.3.1. List of Tempest Plug-in Packages

Run the following command to retrieve a list of tempest test packages:

```
$ sudo yum search $(openstack service list -c Name -f value) 2>/dev/null | grep test | awk '{print $1}'
```

Component	Package Name
barbican	python-barbican-tests-tempest
cinder	python-cinder-tests-tempest
designate	python-designate-tests-tempest
ec2-api	python-ec2api-tests-tempest
heat	python-heat-tests-tempest
horizon	python-horizon-tests-tempest
ironic	python-ironic-tests-tempest

Component	Package Name
keystone	python-keystone-tests-tempest
kuryr	python-kuryr-tests-tempest
manila	python-manila-tests-tempest
mistral	python-mistral-tests-tempest
networking-bgvpn	python-networking-bgpvpn-tests-tempest
networking-l2gw	python-networking-l2gw-tests-tempest
neutron	python-neutron-tests-tempest
nova-join	python-novajoin-tests-tempest
octavia	python-octavia-tests-tempest
patrole	python-patrole-tests-tempest
sahara	python-sahara-tests-tempest
telemetry	python-telemetry-tests-tempest
tripleo-common	python-tripleo-common-tests-tempest
zaqar	python-zaqar-tests-tempest

Tempest test packages are specific to a Python version. For example, if your system uses Python 2, you must replace **python-** with **python2-** when installing the Tempest test packages.

NOTE

The **python-telemetry-tests-tempest** package contains plugins for aodh, panko, gnocchi, and ceilometer tests. The **python-ironic-tests-tempest** package contains plugins for ironic and ironic-inspector.

CHAPTER 4. CONFIGURING THE OPENSTACK INTEGRATION TEST SUITE

4.1. CREATING A WORKSPACE

1. Source the credentials for the target deployment:
   - If the target is in the undercloud, source the credentials for the undercloud:

     ```
     # source stackrc
     ```

   - If the target is in the overcloud, source the credentials for the overcloud:

     ```
     # source overcloudrc
     ```

2. Initialize **tempest**:

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

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

   Run the following command to 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 --network-id <UUID>
   ```

   Replace **UUID** with the UUID of the external network. For more information about the options that you can include in the **discover-tempest-config** command, run **discover-tempest-config --help**. For example, use the **--image** option to define the image that you want to use. If you expllicitly assign an image with **--image** option, you may need to create a flavor which is suitable for the assigned image. The default flavors created by **discover-tempest-config**, m1.nano and m1.micro, may be too small for the image. After you create a flavor, you need to add the flavor id into **flavor_ref** and **flavor_ref_alt** in **tempest.conf**.

   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**.

NOTE

To generate the **etc/tempest.conf** file for the undercloud, ensure that the region name in the **tempest-deployer-input.conf** file is the same as the name in the undercloud deployment. If these names do not match, update the region name in the **tempest-deployer-input.conf** file to match the region name of your undercloud.

To inspect the region name of your undercloud, run the following commands:

```
$ source stackrc
$ openstack region list
```

To inspect the region name of your overcloud, run the following commands:

```
$ source overcloudrc
$ openstack region list
```

4.2. CONFIGURING TEMPEST MANUALLY

The **discover-tempest-config** command generates the **tempest.conf** file automatically. However, you must ensure that the **tempest.conf** file corresponds to the configuration of your environment.

4.2.1. Configuring Tempest Extension Lists Manually

The default **tempest.conf** file contains lists of extensions for each component. Inspect the **api_extensions** attribute for each component in the **tempest.conf** file and verify that the lists of extensions correspond to your deployment.

If the extensions that are available in your deployment do not correspond to the list of extensions in the **api_extensions** attribute of the **tempest.conf** file, the component fails tempest tests. To prevent this failure, you must identify the extensions that are available in your deployment and include them in the **api_extensions** parameter. To get a list of Network, Compute, Volume, or Identity extensions in your deployment, run the following command:

```
$ openstack extension list [--network] [--compute] [--volume] [--identity]
```

4.2.2. Configuring heat_plugin Manually

Configure **heat_plugin** plugin manually according to your deployment configuration. The following example contains the minimum **tempest.conf** configuration for **heat_plugin**:

```
[service_available]
heat_plugin = True

[heat_plugin]
username = demo
password = ***
project_name = demo
admin_username = admin
admin_password = ****
admin_project_name = admin
auth_url = http://10.0.0.110:5000//v3
auth_version = 3
user_domain_id = default
project_domain_id = default
user_domain_name = Default
project_domain_name = Default
region = regionOne
instance_type = m1.nano
minimal_instance_type = m1.micro
image_ref = 7faed41e-a56c-4971-bf48-24e4e23e69a5
minimal_image_ref = 7faed41e-a56c-4971-bf48-24e4e23e69a5
```

NOTE

You must set **heat_plugin** to **True** in the **[service_available]** section of the **tempest.conf** file, and the user in the **username** attribute of the **[heat_plugin]** section must have the role **_member_**. For example, run the following command to add the **_member_** role to the **demo** user:

```
$ openstack role add --user demo --project demo '_member_'
```

4.3. VERIFYING YOUR TEMPEST CONFIGURATION

Verify your current tempest configuration:

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

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

4.4. 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 you set the **log_config_append** attribute, Tempest ignores all other logging configuration in **tempest.conf**, including the **log_dir** attribute.

4.5. CONFIGURING MICROVERSION TESTS

The OpenStack Integration Test Suite provides stable interfaces to test the API microversions. This section contains information about implementing microversion tests using these interfaces.

First, you must configure options in the **tempest.conf** configuration file to specify the target microversions. Configure these options to ensure that the supported microversions correspond to the microversions used in the OpenStack cloud. You can specify a range of target microversions to run multiple microversion tests in a single Integration Test Suite operation.

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 TEMPEST

Run the commands in this section to perform various test operations. You can also combine multiple options in a single **tempest run** command.

5.1. LISTING AVAILABLE TESTS

Run the **tempest-run** command with either the **--list-tests** or **-l** options to get a list of available tempest tests:

```
# tempest run -l
```

5.2. 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.

```
# tempest run --smoke
```

5.3. PASSING TESTS USING WHITELIST FILES

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

Run the **tempest run** command with either the **--whitelist-file** or **-w** options to use a whitelist file:

```
# tempest run -w <whitelist_file>
```

5.4. SKIPPING TESTS USING BLACKLIST FILES

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

Run the **tempest run** command with either the **--blacklist-file** or **-b** options to use a blacklist file:

```
# tempest run -b <blacklist_file>
```

5.5. RUNNING TESTS IN PARALLEL CONCURRENTLY, OR SERIALLY

Run the tests serially:

```
# tempest run --serial
```

Run the tests in parallel. Parallel testing is the default:

```
# tempest run --parallel
```

Use the **--concurrency** or **-c** option to specify the number of workers to use when running tests in parallel:

```
# tempest run --concurrency <workers>
```

By default, the Integration Test Suite uses one worker for each CPU available.

5.6. RUNNING SPECIFIC TESTS

Run specific tests with the **--regex** regular expression option. The regular expression must be Python regular expression:

```
# tempest run --regex <regex>
```

For example, use the following example command to run all tests that have names beginning with **tempest.scenario**:

```
# tempest run --regex ^tempest.scenario
```

CHAPTER 6. RUNNING CONTAINERIZED TEMPEST

This section contains information about running tempest from a container on the undercloud. You can run tempest against the overcloud or the undercloud. Containerized tempest requires the same resources as non-containerized tempest.

This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

6.1. PREPARING THE TEMPEST CONTAINER

Complete the following steps to download and configure your tempest container:

1. Change to the **/home/stack** directory:

```
$ cd /home/stack
```

2. Download the tempest container:

```
$ docker pull registry.redhat.io/rhosp13/openstack-tempest
```

   This container includes all tempest plugins. Running tempest tests globally with this container includes tests for plugins. For example, if you run the **tempest run --regex '(*.)'** command, tempest runs all plugin tests. These tempest tests fail if your deployment does not contain configuration for all plugins. Run the **tempest list-plugins** command to view all installed plugins. To exclude tests, you must include the tests that you want to exclude in a blacklist file. For more information, see Chapter 5, *Using Tempest*.

3. Create directories to use for exchanging data between the host machine and the container:

```
$ mkdir container_tempest tempest_workspace
```

4. Copy the necessary files to the **container_tempest** directory. This directory is the file source for the container:

```
$ cp stackrc overcloudrc tempest-deployer-input.conf container_tempest
```

5. List the available docker images:

```
$ docker images
REPOSITORY                                                TAG       IMAGE ID            CREATED
SIZE
registry.redhat.io/rhosp13-beta/openstack-tempest   latest    881f7ac24d8f        10 days ago
641 MB
```

6. Create an alias to facilitate easier command entry. Ensure that you use absolute paths when mounting the directories:

```
$ alias docker-tempest="docker run -i \
    -v "$(pwd)"/container_tempest:/home/stack/container_tempest \
    -v "$(pwd)"/tempest_workspace:/home/stack/tempest_workspace \
```

```
    registry.redhat.io/rhosp13/openstack-tempest \
    /bin/bash"
```

7. To get a list of available tempest plugins in the container, run the following command:

```
$ docker-tempest -c "rpm -qa | grep tempest"
```

6.2. RUNNING CONTAINERIZED TEMPEST INSIDE THE CONTAINER

1. Create a tempest script that you can execute within the container to generate the **tempest.conf** file and run the tempest tests. The script performs the following actions:

   - Set the exit status for the command **set -e**.
   - Source the **overcloudrc** file if you want to run tempest against the overcloud. Source the **stackrc** file if you want to run tempest against the undercloud.
   - Run **tempest init** to create a tempest workspace. Use the shared directory so that the files are also accessible from the host.
   - Change directory to **tempest_workspace**
   - Export the TEMPESTCONF environment variable for ease of use at a later stage.
   - Execute **discover-tempest-config** to generate the **tempest.conf** file. For more information about the options that you can include in the **discover-tempest-config** command, run **discover-tempest-config --help**.
   - Set **--out** to **home/stack/tempest_workspace/tempest.conf** so that the **tempest.conf** file is accessible from the host machine.
   - Set **--deployer-input** to point to the **tempest-deployer-input.conf** file in the shared directory.
   - Run tempest tests. This example script runs the smoke test **tempest run --smoke**.

```
$ cat <<'EOF'>> /home/stack/container_tempest/tempest_script.sh
set -e
source /home/stack/container_tempest/overcloudrc
tempest init /home/stack/tempest_workspace
pushd /home/stack/tempest_workspace

export TEMPESTCONF="/usr/bin/discover-tempest-config"

$TEMPESTCONF \
  --out /home/stack/tempest_workspace/etc/tempest.conf \
  --deployer-input /home/stack/container_tempest/tempest-deployer-input.conf \
  --debug \
  --create \
  object-storage.reseller_admin ResellerAdmin

tempest run --smoke

EOF
```

If you already have a **tempest.conf** file and you want only to run the tempest tests, omit **TEMPESTCONF** from the script and replace it with a command to copy your **tempest.conf** file from the **container_tempest** directory to the **tempest_workspace/etc** directory:

```
$ cp /home/stack/container_tempest/tempest.conf
/home/stack/tempest_workspace/etc/tempest.conf
```

2. Set executable privileges on the **tempest_script.sh** script:

```
$ chmod +x container_tempest/tempest_script.sh
```

3. Run the tempest script from the container using the alias that you created in a previous step:

```
$ docker-tempest -c 'set -e; /home/stack/container_tempest/tempest_script.sh'
```

4. Inspect the **.stestr** directory for information about the test results.

5. If you want to rerun the tempest tests, you must first remove and recreate the tempest workspace:

```
$ sudo rm -rf /home/stack/tempest_workspace
$ mkdir /home/stack/tempest_workspace
```

6.3. RUNNING CONTAINERIZED TEMPEST OUTSIDE THE CONTAINER

The container generates or retrieves the **tempest.conf** file and runs tests. You can perform these operations from outside the container:

1. Source the **overcloudrc** file if you want to run tempest against the overcloud. Source the **stackrc** file if you want to run tempest against the undercloud:

```
# source /home/stack/container_tempest/overcloudrc
```

2. Run **tempest init** to create a tempest workspace. Use the shared directory so that the files are also accessible from the host:

```
# tempest init /home/stack/tempest_workspace
```

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

```
# discover-tempest-config \
--out /home/stack/tempest_workspace/tempest.conf \
--deployer-input /home/stack/container_tempest/tempest-deployer-input-conf \
--debug \
--create \
object-storage.reseller_admin ResellerAdmin
```

For more information about the options that you can include in the **discover-tempest-config** command, run **discover-tempest-config --help**.

4. Execute tempest tests. For example, run the following command to execute the tempest smoke test using the alias you created in a previous step:

```
# docker-tempest -c "tempest run --smoke"
```

5. Inspect the **.stestr** directory for information about the test results.

6. If you want to rerun the tempest tests, you must first remove and recreate the tempest workspace:

```
$ sudo rm -rf /home/stack/tempest_workspace
$ mkdir /home/stack/tempest_workspace
```

CHAPTER 7. CLEANING TEMPEST RESOURCES

After running **tempest**, there are files, users and projects created in the testing process that must be deleted. Being able to self-clean is one of the design principles of **tempest**.

7.1. 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 that you want cleanup to delete.

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

Run the cleanup:

```
# tempest cleanup
```

The **tempest cleanup** command deletes tempest resources but does not delete projects or the tempest administrator account.

7.2. PERFORMING A DRY RUN

It is recommended to perform a dry run before executing the real cleanup. 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. Check the **dry_run.json** file to ensure that the cleanup does not delete any files that you require for your environment.

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

7.3. DELETING TEMPEST OBJECTS

Run the following command to delete all tempest resources, including projects, but not the tempest administrator account:

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