Tempest Testing Project

Release 42.0.1.dev61

OpenStack Foundation

Feb 21, 2025

CONTENTS

CHAPTER

ONE

OVERVIEW

1.1 Tempest - The OpenStack Integration Test Suite

The documentation for Tempest is officially hosted at: https://docs.openstack.org/tempest/latest/

This is a set of integration tests to be run against a live OpenStack cluster. Tempest has batteries of tests for OpenStack API validation, scenarios, and other specific tests useful in validating an OpenStack deployment.

1.1.1 Team and repository tags

1.1.2 Design Principles

Tempest Design Principles that we strive to live by.

- Tempest should be able to run against any OpenStack cloud, be it a one node DevStack install, a 20 node LXC cloud, or a 1000 node KVM cloud.
- Tempest should be explicit in testing features. It is easy to auto discover features of a cloud incorrectly, and give people an incorrect assessment of their cloud. Explicit is always better.
- Tempest uses OpenStack public interfaces. Tests in Tempest should only touch public OpenStack APIs.
- Tempest should not touch private or implementation specific interfaces. This means not directly going to the database, not directly hitting the hypervisors, not testing extensions not included in the OpenStack base. If there are some features of OpenStack that are not verifiable through standard interfaces, this should be considered a possible enhancement.
- Tempest strives for complete coverage of the OpenStack API and common scenarios that demonstrate a working cloud.
- Tempest drives load in an OpenStack cloud. By including a broad array of API and scenario tests Tempest can be reused in whole or in parts as load generation for an OpenStack cloud.
- Tempest should attempt to clean up after itself, whenever possible we should tear down resources when done.
- Tempest should be self-testing.

1.1.3 Quickstart

To run Tempest, you first need to create a configuration file that will tell Tempest where to find the various OpenStack services and other testing behavior switches. Where the configuration file lives and how you interact with it depends on how youll be running Tempest. There are 2 methods of using Tempest. The first, which is a newer and recommended workflow treats Tempest as a system installed program. The second older method is to run Tempest assuming your working dir is the actually Tempest source repo, and there are a number of assumptions related to that. For this section well only cover the newer method as it is simpler, and quicker to work with.

1. You first need to install Tempest. This is done with pip after you check out the Tempest repo:

   ```
   $ git clone https://opendev.org/openstack/tempest
   $ pip install tempest/
   ```

   This can be done within a venv, but the assumption for this guide is that the Tempest CLI entry point will be in your shells PATH.

2. Installing Tempest may create a `/etc/tempest dir`, however if one isnt created you can create one or use `~/.tempest/etc` or `~/.config/tempest` in place of `/etc/tempest`. If none of these dirs are created Tempest will create `~/.tempest/etc` when its needed. The contents of this dir will always automatically be copied to all `etc/` dirs in local workspaces as an initial setup step. So if there is any common configuration youd like to be shared between local Tempest workspaces its recommended that you pre-populate it before running `tempest init`.

3. Setup a local Tempest workspace. This is done by using the tempest init command:

   ```
   $ tempest init cloud-01
   ```

   which also works the same as:

   ```
   $ mkdir cloud-01 && cd cloud-01 && tempest init
   ```

   This will create a new directory for running a single Tempest configuration. If youd like to run Tempest against multiple OpenStack deployments the idea is that youll create a new working directory for each to maintain separate configuration files and local artifact storage for each.

4. Then `cd` into the newly created working dir and also modify the local config files located in the `etc/` subdir created by the `tempest init` command. Tempest is expecting a `tempest.conf` file in etc/ so if only a sample exists you must rename or copy it to tempest.conf before making any changes to it otherwise Tempest will not know how to load it. For details on configuring Tempest refer to the Tempest Configuration

5. Once the configuration is done youre now ready to run Tempest. This can be done using the Tempest Run command. This can be done by either running:

   ```
   $ tempest run
   ```

   from the Tempest workspace directory. Or you can use the `--workspace` argument to run in the workspace you created regardless of your current working directory. For example:

   ```
   $ tempest run --workspace cloud-01
   ```

   There is also the option to use stestr directly. For example, from the workspace dir run:

```
$ stestr run --exclude-regex '\[.*\bslow\b.*\]' '^tempest\.(api|scenario)'
```

will run the same set of tests as the default gate jobs. Or you can use unittest compatible test runners such as stestr, pytest etc.

Tox also contains several existing job configurations. For example:

```
$ tox -e full
```

which will run the same set of tests as the OpenStack gate. (its exactly how the gate invokes Tempest) Or:

```
$ tox -e smoke
```

to run the tests tagged as smoke.

1.1.4 Library

Tempest exposes a library interface. This interface is a stable interface and should be backwards compatible (including backwards compatibility with the old tempest-lib package, with the exception of the import). If you plan to directly consume Tempest in your project you should only import code from the Tempest library interface, other pieces of Tempest do not have the same stable interface and there are no guarantees on the Python API unless otherwise stated.

For more details refer to the library documentation

1.1.5 Release Versioning

Tempest Release Notes shows what changes have been released on each version.

Tempests released versions are broken into 2 sets of information. Depending on how you intend to consume Tempest you might need

The version is a set of 3 numbers:

X.Y.Z

While this is almost semver like, the way versioning is handled is slightly different:

X is used to represent the supported OpenStack releases for Tempest tests in-tree, and to signify major feature changes to Tempest. Its a monotonically increasing integer where each version either indicates a new supported OpenStack release, the drop of support for an OpenStack release (which will coincide with the upstream stable branch going EOL), or a major feature lands (or is removed) from Tempest.

Y.Z is used to represent library interface changes. This is treated the same way as minor and patch versions from semver but only for the library interface. When Y is incremented weve added functionality to the library interface and when Z is incremented its a bug fix release for the library. Also note that both Y and Z are reset to 0 at each increment of X.

1.1.6 Configuration

Detailed configuration of Tempest is beyond the scope of this document, see Tempest Configuration Documentation for more details on configuring Tempest. The `etc/tempest.conf.sample` attempts to be a self-documenting version of the configuration.

You can generate a new sample tempest.conf file, run the following command from the top level of the Tempest directory:

```
$ tox -e genconfig
```

The most important pieces that are needed are the user ids, OpenStack endpoints, and basic flavors and images needed to run tests.

1.1.7 Unit Tests

Tempest also has a set of unit tests which test the Tempest code itself. These tests can be run by specifying the test discovery path:

```
$ stestr --test-path ./tempest/tests run
```

By setting `--test-path` option to ./tempest/tests it specifies that test discover should only be run on the unit test directory. The default value of `test_path` is `test_path=./tempest/test_discover` which will only run test discover on the Tempest suite.

Alternatively, there is the py39 tox job which will run the unit tests with the corresponding version of python.

One common activity is to just run a single test, you can do this with tox simply by specifying to just run py39 tests against a single test:

```
$ tox -e py39 -- -n tempest.tests.test_microversions.
↪TestMicroversionsTestsClass.test_config_version_none_23
```

Or all tests in the test_microversions.py file:

```
$ tox -e py39 -- -n tempest.tests.test_microversions
```

You may also use regular expressions to run any matching tests:

```
$ tox -e py39 -- test_microversions
```

Additionally, when running a single test, or test-file, the `-n/--no-discover` argument is no longer required, however it may perform faster if included.

For more information on these options and details about stestr, please see the stestr documentation.

1.1.8 Python 3.x

Starting during the Pike cycle Tempest has a gating CI job that runs Tempest with Python 3. Any Tempest release after 15.0.0 should fully support running under Python 3 as well as Python 2.7.

1.1.9 Legacy run method

The legacy method of running Tempest is to just treat the Tempest source code as a python unittest repository and run directly from the source repo. When running in this way you still start with a Tempest config file and the steps are basically the same except that it expects you know where the Tempest code lives on your system and requires a bit more manual interaction to get Tempest running. For example, when running Tempest this way things like a lock file directory do not get generated automatically and the burden is on the user to create and configure that.

To start you need to create a configuration file. The easiest way to create a configuration file is to generate a sample in the `etc/` directory

```
$ cd $TEMPEST_ROOT_DIR
$ oslo-config-generator --config-file \
    tempest/cmd/config-generator.tempest.conf \
    --output-file etc/tempest.conf
```

After that, open up the `etc/tempest.conf` file and edit the configuration variables to match valid data in your environment. This includes your Keystone endpoint, a valid user and credentials, and reference data to be used in testing.

Note

If you have a running DevStack environment, Tempest will be automatically configured and placed in `/opt/stack/tempest`. It will have a configuration file already set up to work with your DevStack installation.

Tempest is not tied to any single test runner, but stestr is the most commonly used tool. Also, the nosetests test runner is **not** recommended to run Tempest.

After setting up your configuration file, you can execute the set of Tempest tests by using `stestr`. By default, `stestr` runs tests in parallel

```
$ stestr run
```

To run one single test serially

```
$ stestr run --serial tempest.api.compute.servers.test_servers_negative.
↪ServersNegativeTestJSON.test_reboot_non_existent_server
```

CHAPTER

TWO

FIELD GUIDES

Tempest contains tests of many different types, the field guides attempt to explain these in a way that makes it easy to understand where your test contributions should go.

2.1 Tempest Field Guide Overview

Tempest is designed to be useful for a large number of different environments. This includes being useful for gating commits to OpenStack core projects, being used to validate OpenStack cloud implementations for both correctness, as well as a burn in tool for OpenStack clouds.

As such Tempest tests come in many flavors, each with its own rules and guidelines. Below is the overview of the Tempest repository structure to make this clear.

```
tempest/
   api/ - API tests
   scenario/ - complex scenario tests
   serial_tests/ - tests that run always in the serial mode
   tests/ - unit tests for Tempest internals
```

Each of these directories contains different types of tests. What belongs in each directory, the rules and examples for good tests, are documented in a README.rst file in the directory.

2.1.1 Tempest Field Guide to API tests

API tests are validation tests for the OpenStack API. They should not use the existing Python clients for OpenStack, but should instead use the Tempest implementations of clients. Having raw clients let us pass invalid JSON to the APIs and see the results, something we could not get with the native clients.

When it makes sense, API testing should be moved closer to the projects themselves, possibly as functional tests in their unit test frameworks.

2.1.2 Tempest Field Guide to Scenario tests

Scenario tests are complex through path tests for OpenStack functionality. They are typically a series of steps where a complicated state requiring multiple services is set up exercised, and torn down.

Scenario tests should not use the existing Python clients for OpenStack, but should instead use the Tempest implementations of clients.

2.1.3 Tempest Field Guide to Serial tests

Tests within this category will always be executed serially from the rest of the test cases.

2.1.4 Tempest Field Guide to Unit tests

Unit tests are the self checks for Tempest. 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 Tempest are working as expected.

2.2 Tempest Field Guide to API tests

2.2.1 What are these tests?

One of Tempests prime functions is to ensure that your OpenStack cloud works with the OpenStack API as documented. The current largest portion of Tempest code is devoted to test cases that do exactly this.

Its also important to test not only the expected positive path on APIs, but also to provide them with invalid data to ensure they fail in expected and documented ways. The latter type of tests is called `negative tests` in Tempest source code. Throughout the OpenStack project, Tempest has discovered many fundamental bugs by doing just this.

In order for some APIs to return meaningful results, there must be enough data in the system. This means these tests might start by spinning up a server, image, etc., and then operating on it.

2.2.2 Why are these tests in Tempest?

This is one of the core missions for the Tempest project, and where it started. Many people use this bit of function in Tempest to ensure their clouds havent broken the OpenStack API.

It could be argued that some of the negative testing could be done back in the projects themselves, and we might evolve there over time, but currently, in the OpenStack gate, this is a fundamentally important place to keep things.

2.2.3 Scope of these tests

API tests should always use the Tempest implementation of the OpenStack API, as we want to ensure that bugs arent hidden by the official clients.

They should test specific API calls and can build up complex states if its 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.

2.3 Tempest Field Guide to Scenario tests

2.3.1 What are these tests?

Scenario tests are through path tests of OpenStack function. Complicated setups where one part might depend on the completion of a previous part. They ideally involve the integration between multiple OpenStack services to exercise the touch points between them.

Any scenario test should have a real-life use case. An example would be:

- As an operator, I want to start with a blank environment:

1. upload a glance image
2. deploy a vm from it
3. ssh to the guest
4. create a snapshot of the vm

2.3.2 Why are these tests in Tempest?

This is one of Tempests core purposes, testing the integration between projects.

2.3.3 Scope of these tests

Scenario tests should always use the Tempest implementation of the OpenStack API, as we want to ensure that bugs arent 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.

2.3.4 Example of a good test

While we are looking for interaction of 2 or more services, be specific in your interactions. A giant this is my data center smoke test is hard to debug when it goes wrong.

A flow of interactions between Glance and Nova, like 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.

2.4 Tempest Field Guide to Unit tests

2.4.1 What are these tests?

Unit tests are the self checks for Tempest. 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 Tempest are working as expected. They should not require an external service to be running and should be able to run solely from the Tempest tree.

2.4.2 Why are these tests in Tempest?

These tests exist to make sure that the mechanisms that we use inside of Tempest are valid and remain functional. They are only here for self validation of Tempest.

2.4.3 Scope of these tests

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

THREE

USERS GUIDE

3.1 Tempest Configuration Guide

3.1.1 Tempest Configuration Guide

This guide is a starting point for configuring Tempest. It aims to elaborate on and explain some of the mandatory and common configuration settings and how they are used in conjunction. The source of truth on each option is the sample config file which explains the purpose of each individual option. You can see the sample config file here: *Sample Configuration File*

Test Credentials

Tempest allows for configuring a set of admin credentials in the `auth` section, via the following parameters:

1. `admin_username`
2. `admin_password`
3. `admin_project_name`
4. `admin_domain_name`

Admin credentials are not mandatory to run Tempest, but when provided they can be used to:

- Run tests for admin APIs
- Generate test credentials on the fly (see *Dynamic Credentials*)

When Keystone uses a policy that requires domain scoped tokens for admin actions, the flag `admin_domain_scope` must be set to `True`. The admin user configured, if any, must have a role assigned to the domain to be usable.

Tempest allows for configuring pre-provisioned test credentials as well. This can be done using the accounts.yaml file (see *Pre-Provisioned Credentials*). This file is used to specify an arbitrary number of users available to run tests with. You can specify the location of the file in the `auth` section in the tempest.conf file. To see the specific format used in the file please refer to the `accounts.yaml.sample` file included in Tempest.

Keystone Connection Info

In order for Tempest to be able to talk to your OpenStack deployment you need to provide it with information about how it communicates with keystone. This involves configuring the following options in the `identity` section:

- `auth_version`

- uri
- uri_v3

The auth_version option is used to tell Tempest whether it should be using Keystones v2 or v3 api for communicating with Keystone. The two uri options are used to tell Tempest the url of the keystone endpoint. The uri option is used for Keystone v2 request and uri_v3 is used for Keystone v3. You want to ensure that which ever version you set for auth_version has its uri option defined.

Credential Provider Mechanisms

Tempest currently has two different internal methods for providing authentication to tests: dynamic credentials and pre-provisioned credentials. Depending on which one is in use the configuration of Tempest is slightly different.

Dynamic Credentials

Dynamic Credentials (formerly known as Tenant isolation) was originally created to enable running Tempest in parallel. For each test class it creates a unique set of user credentials to use for the tests in the class. It can create up to three sets of username, password, and project names for a primary user, an admin user, and an alternate user. To enable and use dynamic credentials you only need to configure two things:

1. A set of admin credentials with permissions to create users and projects. This is specified in the auth section with the admin_username, admin_project_name, admin_domain_name and admin_password options
2. To enable dynamic credentials in the auth section with the use_dynamic_credentials option.

This is also currently the default credential provider enabled by Tempest, due to its common use and ease of configuration.

It is worth pointing out that depending on your cloud configuration you might need to assign a role to each of the users created by Tempests dynamic credentials. This can be set using the tempest_roles option. It takes in a list of role names each of which will be assigned to each of the users created by dynamic credentials. This option will not have any effect when Tempest is not configured to use dynamic credentials.

When the admin_domain_scope option is set to True, provisioned admin accounts will be assigned a role on domain configured in default_credentials_domain_name. This will make the accounts provisioned usable in a cloud where domain scoped tokens are required by Keystone for admin operations. Note that the initial pre-provision admin accounts, configured in tempest.conf, must have a role on the same domain as well, for Dynamic Credentials to work.

Pre-Provisioned Credentials

For a long time using dynamic credentials was the only method available if you wanted to enable parallel execution of Tempest tests. However, this was insufficient for certain use cases because of the admin credentials requirement to create the credential sets on demand. To get around that the accounts.yaml file was introduced and with that a new internal credential provider to enable using the list of credentials instead of creating them on demand. With pre-provisioned credentials (also known as locking test accounts) each test class will reserve a set of credentials from the accounts.yaml before executing any of its tests so that each class is isolated like with dynamic credentials.

To enable and use pre-provisioned credentials you need do a few things:

1. Create an accounts.yaml file which contains the set of pre-existing credentials to use for testing. To make sure you dont have a credentials starvation issue when running in parallel make sure you have at least two times the number of worker processes you are using to execute Tempest available in the file. (If running serially the worker count is 1.)

   You can check the accounts.yaml.sample file packaged in Tempest for the yaml format.

2. Provide Tempest with the location of your accounts.yaml file with the `test_accounts_file` option in the `auth` section

   NOTE: Be sure to use a full path for the file; otherwise Tempest will likely not find it.

3. Set `use_dynamic_credentials = False` in the `auth` group

It is worth pointing out that each set of credentials in the accounts.yaml should have a unique project. This is required to provide proper isolation to the tests using the credentials, and failure to do this will likely cause unexpected failures in some tests. Also, ensure that these projects and users used do not have any pre-existing resources created. Tempest assumes all tenants its using are empty and may sporadically fail if there are unexpected resources present.

When the Keystone in the target cloud requires domain scoped tokens to perform admin actions, all pre-provisioned admin users must have a role assigned on the domain where test accounts a provisioned. The option `admin_domain_scope` is used to tell Tempest that domain scoped tokens shall be used. `default_credentials_domain_name` is the domain where test accounts are expected to be provisioned if no domain is specified.

Note that if credentials are pre-provisioned via `tempest account-generator` the role on the domain will be assigned automatically for you, as long as `admin_domain_scope` as `default_credentials_domain_name` are configured properly in tempest.conf.

Pre-Provisioned Credentials are also known as accounts.yaml or accounts file.

Keystone Scopes & Roles Support in Tempest

For details on scope and roles support in Tempest, please refer to *this document*

Compute

Flavors

For Tempest to be able to create servers you need to specify flavors that it can use to boot the servers with. There are two options in the Tempest config for doing this:

1. `flavor_ref`
2. `flavor_ref_alt`

Both of these options are in the `compute` section of the config file and take in the flavor id (not the name) from Nova. The `flavor_ref` option is what will be used for booting almost all of the guests; `flavor_ref_alt` is only used in tests where two different-sized servers are required (for example, a resize test).

Using a smaller flavor is generally recommended. When larger flavors are used, the extra time required to bring up servers will likely affect the total run time and probably require tweaking timeout values to ensure tests have ample time to finish.

Images

Just like with flavors, Tempest needs to know which images to use for booting servers. There are two options in the compute section just like with flavors:

1. `image_ref`
2. `image_ref_alt`

Both options are expecting an image id (not name) from Nova. The `image_ref` option is what will be used for booting the majority of servers in Tempest. `image_ref_alt` is used for tests that require two images such as rebuild. If two images are not available you can set both options to the same image id and those tests will be skipped.

There are also options in the `scenario` section for images:

1. `img_file`
2. `img_container_format`
3. `img_disk_format`

However, unlike the other image options, these are used for a very small subset of scenario tests which are uploading an image. These options are used to tell Tempest where an image file is located and describe its metadata for when it is uploaded.

You first need to specify full path of the image using `img_file` option. If it is found then the `img_container_format` and `img_disk_format` options are used to upload that image to glance. If its not found, the tests requiring an image to upload will fail.

It is worth pointing out that using cirros is a very good choice for running Tempest. Its what is used for upstream testing, they boot quickly and have a small footprint.

Networking

OpenStack has a myriad of different networking configurations possible and depending on which of the two network backends, nova-network or Neutron, you are using things can vary drastically. Due to this complexity Tempest has to provide a certain level of flexibility in its configuration to ensure it will work against any cloud. This ends up causing a large number of permutations in Tempests config around network configuration.

Enabling Remote Access to Created Servers

Network Creation/Usage for Servers

When Tempest creates servers for testing, some tests require being able to connect those servers. Depending on the configuration of the cloud, the methods for doing this can be different. In certain configurations, it is required to specify a single network with server create calls. Accordingly, Tempest provides a few different methods for providing this information in configuration to try and ensure that regardless of the clouds configuration itll still be able to run. This section covers the different methods of configuring Tempest to provide a network when creating servers.

Fixed Network Name

This is the simplest method of specifying how networks should be used. You can just specify a single network name/label to use for all server creations. The limitation with this is that all projects and users must be able to see that network name/label if they are to perform a network list and be able to use it.

If no network name is assigned in the config file and none of the below alternatives are used, then Tempest will not specify a network on server creations, which depending on the cloud configuration might prevent them from booting.

To set a fixed network name simply:

1. Set the `fixed_network_name` option in the `compute` group

In the case that the configured fixed network name can not be found by a user network list call, it will be treated like one was not provided except that a warning will be logged stating that it couldnt be found.

Accounts File

If you are using an accounts file to provide credentials for running Tempest then you can leverage it to also specify which network should be used with server creations on a per project and user pair basis. This provides the necessary flexibility to work with more intricate networking configurations by enabling the user to specify exactly which network to use for which projects. You can refer to the accounts.yaml.sample file included in the Tempest repo for the syntax around specifying networks in the file.

However, specifying a network is not required when using an accounts file. If one is not specified you can use a fixed network name to specify the network to use when creating servers just as without an accounts file. However, any network specified in the accounts file will take precedence over the fixed network name provided. If no network is provided in the accounts file and a fixed network name is not set then no network will be included in create server requests.

If a fixed network is provided and the accounts.yaml file also contains networks this has the benefit of enabling a couple more tests which require a static network to perform operations like server lists with a network filter. If a fixed network name is not provided these tests are skipped. Additionally, if a fixed network name is provided it will serve as a fallback in case of a misconfiguration or a missing network in the accounts file.

With Dynamic Credentials

With dynamic credentials enabled and using nova-network, your only option for configuration is to either set a fixed network name or not. However, in most cases, it shouldnt matter because nova-network should have no problem booting a server with multiple networks. If this is not the case for your cloud then using an accounts file is recommended because it provides the necessary flexibility to describe your configuration. Dynamic credentials are not able to dynamically allocate things as necessary if Neutron is not enabled.

With Neutron and dynamic credentials enabled there should not be any additional configuration necessary to enable Tempest to create servers with working networking, assuming you have properly configured the `network` section to work for your cloud. Tempest will dynamically create the Neutron resources necessary to enable using servers with that network. Also, just as with the accounts file, if you specify a fixed network name while using Neutron and dynamic credentials it will enable running tests which require a static network and it will additionally be used as a fallback for server creation. However, unlike accounts.yaml this should never be triggered.

However, there is an option `create_isolated_networks` to disable dynamic credentialss automatic provisioning of network resources. If this option is set to `False` you will have to either rely on there only

being a single/default network available for the server creation, or use `fixed_network_name` to inform Tempest which network to use.

SSH Connection Configuration

There are also several different ways to actually establish a connection and authenticate/login on the server. After a server is booted with a provided network there are still details needed to know how to actually connect to the server. The `validation` group gathers all the options regarding connecting to and remotely accessing the created servers.

To enable remote access to servers, there are 3 options at a minimum that are used:

1. `run_validation`
2. `connect_method`
3. `auth_method`

The `run_validation` is used to enable or disable ssh connectivity for all tests (with the exception of scenario tests which do not have a flag for enabling or disabling ssh) To enable ssh connectivity this needs be set to `True`.

The `connect_method` option is used to tell Tempest what kind of IP to use for establishing a connection to the server. Two methods are available: `fixed` and `floating`, the later being set by default. If this is set to floating Tempest will create a floating ip for the server before attempted to connect to it. The IP for the floating ip is what is used for the connection.

For the `auth_method` option there is currently, only one valid option, `keypair`. With this set to `keypair` Tempest will create an ssh keypair and use that for authenticating against the created server.

Configuring Available Services

OpenStack is really a constellation of several different projects which are running together to create a cloud. However which projects youre running is not set in stone, and which services are running is up to the deployer. Tempest, however, needs to know which services are available so it can figure out which tests it is able to run and certain setup steps which differ based on the available services.

The `service_available` section of the config file is used to set which services are available. It contains a boolean option for each service (except for Keystone which is a hard requirement) set it to `True` if the service is available or `False` if it is not.

Service Catalog

Each project which has its own REST API contains an entry in the service catalog. Like most things in OpenStack this is also completely configurable. However, for Tempest to be able to figure out which endpoints should get REST API calls for each service, it needs to know how that project is defined in the service catalog. There are three options for each service section to accomplish this:

1. `catalog_type`
2. `endpoint_type`
3. `region`

Setting `catalog_type` and `endpoint_type` should normally give Tempest enough information to determine which endpoint it should pull from the service catalog to use for talking to that particular service. However, if your cloud has multiple regions available and you need to specify a particular one to use a service you can set the `region` option in that services section.

It should also be noted that the default values for these options are set to what DevStack uses (which is a de facto standard for service catalog entries). So often nothing actually needs to be set on these options to enable communication to a particular service. It is only if you are either not using the same `catalog_type` as DevStack or you want Tempest to talk to a different endpoint type instead of `publicURL` for a service that these need to be changed.

Note

Tempest does not serve all kinds of fancy URLs in the service catalog. The service catalog should be in a standard format (which is going to be standardized at the Keystone level). Tempest expects URLs in the Service catalog in the following format:

- `http://example.com:1234/<version-info>`

Examples:

- Good - `http://example.com:1234/v2.0`
- Wouldnt work - `http://example.com:1234/xyz/v2.0/` (adding prefix/suffix around version etc)

Service Feature Configuration

OpenStack provides its deployers a myriad of different configuration options to enable anyone deploying it to create a cloud tailor-made for any individual use case. It provides options for several different backend types, databases, message queues, etc. However, the downside to this configurability is that certain operations and features arent supported depending on the configuration. These features may or may not be discoverable from the API so the burden is often on the user to figure out what is supported by the cloud theyre talking to. Besides the obvious interoperability issues with this, it also leaves Tempest in an interesting situation trying to figure out which tests are expected to work. However, Tempest tests do not rely on dynamic API discovery for a feature (assuming one exists). Instead, Tempest has to be explicitly configured as to which optional features are enabled. This is in order to prevent bugs in the discovery mechanisms from masking failures.

The service `feature-enabled` config sections are how Tempest addresses the optional feature question. Each service that has tests for optional features contains one of these sections. The only options in it are boolean options with the name of a feature which is used. If it is set to false any test which depends on that functionality will be skipped. For a complete list of all these options refer to the sample config file.

API Extensions

The service feature-enabled sections often contain an `api-extensions` option (or in the case of Swift a `discoverable_apis` option). This is used to tell Tempest which API extensions (or configurable middleware) is used in your deployment. It has two valid config states: either it contains a single value `all` (which is the default) which means that every API extension is assumed to be enabled, or it is set to a list of each individual extension that is enabled for that service.

3.1.2 Sample Configuration File

The following is a sample Tempest configuration for adaptation and use. It is auto-generated from Tempest when this documentation is built, so if you are having issues with an option, please compare your version of Tempest with the version of this documentation.

The sample configuration can also be viewed in file form.

3.2 Command Documentation

3.2.1 Tempest Test-Account Generator Utility

Utility for creating `accounts.yaml` file for concurrent test runs. Creates one primary user, one alt user, one swift admin, one stack owner and one admin (optionally) for each concurrent thread. The utility creates user for each tenant. The `accounts.yaml` file will be valid and contain credentials for created users, so each user will be in separate tenant and have the username, tenant_name, password and roles.

Usage: `tempest account-generator [-h] [OPTIONS] accounts_file.yaml`

Positional Arguments

`accounts_file.yaml` (Required) Provide an output accounts yaml file. Utility creates a .yaml file in the directory where the command is ran. The appropriate name for the file is *accounts.yaml* and it should be placed in *tempest/etc* directory.

Authentication

Account generator creates users and tenants so it needs the admin credentials of your cloud to operate properly. The corresponding info can be given either through CLI options or environment variables.

Youre probably familiar with these, but just to remind:

Param	CLI	Environment Variable
Username	`--os-username`	OS_USERNAME
Password	`--os-password`	OS_PASSWORD
Project	`--os-project-name`	OS_PROJECT_NAME
Domain	`--os-domain-name`	OS_DOMAIN_NAME

Optional Arguments

- `-h, --help` (Optional) Shows help message with the description of utility and its arguments, and exits.
- `-c, --config-file /etc/tempest.conf` (Optional) Path to tempest config file. If not specified, it searches for tempest.conf in these locations:
  - ./etc/
  - /etc/tempest
  - ~/.tempest/
  - ~/
  - /etc/
- `--os-username <auth-user-name>` (Optional) Name used for authentication with the OpenStack Identity service. Defaults to env[OS_USERNAME]. Note: User should have permissions to create new user accounts and tenants.
- `--os-password <auth-password>` (Optional) Password used for authentication with the OpenStack Identity service. Defaults to env[OS_PASSWORD].
- `--os-project-name <auth-project-name>` (Optional) Project to request authorization on. Defaults to env[OS_PROJECT_NAME].

- `--os-domain-name <auth-domain-name>` (Optional) Domain the user and project belong to. Defaults to env[OS_DOMAIN_NAME].
- `--tag TAG` (Optional) Resources tag. Each created resource (user, project) will have the prefix with the given TAG in its name. Using tag is recommended for the further using, cleaning resources.
- `-r, --concurrency CONCURRENCY` (Optional) Concurrency count (default: 2). The number of accounts generated will be same as CONCURRENCY. The higher the number, the more tests will run in parallel. If you want to run tests sequentially then use 1 as value for concurrency (beware that tests that need more credentials will fail).
- `--with-admin` (Optional) Creates admin for each concurrent group (default: False).
- `-i, --identity-version VERSION` (Optional) Provisions accounts using the specified version of the identity API. (default: 3).

To see help on specific argument, please do: `tempest account-generator [OPTIONS] <accounts_file.yaml> -h`.

3.2.2 Post Tempest Run Cleanup Utility

Utility for cleaning up environment after Tempest test run

Usage: `tempest cleanup [--help] [OPTIONS]`

If run with no arguments, `tempest cleanup` will query your OpenStack deployment and build a list of resources to delete and destroy them. This list will exclude the resources from `saved_state.json` and will include the configured admin account if the `--delete-tempest-conf-objects` flag is specified. By default the admin project is not deleted and the admin user specified in `tempest.conf` is never deleted.

Example Run

Warning

We advice not to run tempest cleanup on production environments.

Warning

If step 1 is skipped in the example below, the cleanup procedure may delete resources that existed in the cloud before the test run. This may cause an unwanted destruction of cloud resources, so use caution with this command.

Examples:

```
$ tempest cleanup --init-saved-state
$ # Actual running of Tempest tests
$ tempest cleanup
```

Runtime Arguments

- `--init-saved-state`: Initializes the saved state of the OpenStack deployment and will output a `saved_state.json` file containing resources from your deployment that will be preserved from the cleanup command. This should be done prior to running Tempest tests. Note, that if other users of your cloud could have created resources after running `--init-saved-state`, it would not protect those resources as they wouldnt be present in the saved_state.json file.
- `--delete-tempest-conf-objects`: If option is present, then the command will delete the admin project in addition to the resources associated with them on clean up. If option is not present, the command will delete the resources associated with the Tempest and alternate Tempest users and projects but will not delete the projects themselves.
- `--dry-run`: Creates a report (`./dry_run.json`) of the projects that will be cleaned up (in the `_projects_to_clean` dictionary[1]) and the global objects that will be removed (domains, flavors, images, roles, projects, and users). Once the cleanup command is executed (e.g. run without parameters), running it again with `--dry-run` should yield an empty report. We STRONGLY ENCOURAGE to run `tempest cleanup` with `--dry-run` first and then verify that the resources listed in the `dry_run.json` file are meant to be deleted.
- `--prefix`: Only resources that match the prefix will be deleted. When this option is used, `saved_state.json` file is not needed (no need to run with `--init-saved-state` first).

  All tempest resources are created with the prefix value from the config option `resource_name_prefix` in tempest.conf. To cleanup only the resources created by tempest, you should use the prefix set in your tempest.conf (the default value of `resource_name_prefix` is `tempest`.

  Note, that some resources are not named thus they will not be deleted when filtering based on the prefix. This option will be ignored when `--init-saved-state` is used so that it can capture the true init state - all resources present at that moment. If there is any `saved_state.json` file present (e.g. if you ran the tempest cleanup with `--init-saved-state` before) and you run the tempest cleanup with `--prefix`, the `saved_state.json` file will be ignored and cleanup will be done based on the passed prefix only.
- `--resource-list`: Allows the use of file `./resource_list.json`, which contains all resources created by Tempest during all Tempest runs, to create another method for removing only resources created by Tempest. List of these resources is created when config option `record_resources` in default section is set to true. After using this option for cleanup, the existing `./resource_list.json` is cleared from deleted resources.

  When this option is used, `saved_state.json` file is not needed (no need to run with `--init-saved-state` first). If there is any `saved_state.json` file present and you run the tempest cleanup with `--resource-list`, the `saved_state.json` file will be ignored and cleanup will be done based on the `resource_list.json` only.

  If you run tempest cleanup with both `--prefix` and `--resource-list`, the `--resource-list` option will be ignored and cleanup will be done based on the `--prefix` option only.
- `--help`: Print the help text for the command and parameters.

[1] The `_projects_to_clean` dictionary in `dry_run.json` lists the projects that `tempest cleanup` will loop through to delete child objects, but the command will, by default, not delete the projects themselves. This may differ from the `projects` list as you can clean the Tempest and alternate Tempest users and projects but they will not be deleted unless the `--delete-tempest-conf-objects` flag is used to force their deletion.

Note

If during execution of `tempest cleanup` NotImplemented exception occurres, `tempest cleanup` wont fail on that, it will be logged only. NotImplemented errors are ignored because they are an outcome of some extensions being disabled and `tempest cleanup` is not checking their availability as it tries to clean up as much as possible without any complicated logic.

3.2.3 Subunit Describe Calls Utility

subunit-describe-calls is a parser for subunit streams to determine what REST API calls are made inside of a test and in what order they are called.

Runtime Arguments

- `--subunit, -s`: (Optional) The path to the subunit file being parsed, defaults to stdin
- `--non-subunit-name, -n`: (Optional) The file_name that the logs are being stored in
- `--output-file, -o`: (Optional) The path where the JSON output will be written to. This contains more information than is present in stdout.
- `--ports, -p`: (Optional) The path to a JSON file describing the ports being used by different services
- `--verbose, -v`: (Optional) Print Request and Response Headers and Body data to stdout in the non cliff deprecated CLI
- `--all-stdout, -a`: (Optional) Print Request and Response Headers and Body data to stdout

Usage

subunit-describe-calls will take in either stdin subunit v1 or v2 stream or a file path which contains either a subunit v1 or v2 stream passed via the `--subunit` parameter. This is then parsed checking for details contained in the file_bytes of the `--non-subunit-name` parameter (the default is pythonlogging which is what Tempest uses to store logs). By default the OpenStack default ports are used unless a file is provided via the `--ports` option. The resulting output is dumped in JSON output to the path provided in the `--output-file` option.

Ports file JSON structure

```
{
    "<port number>": "<name of service>",
    ...
}
```

Output file JSON structure

```
{
    "full_test_name[with_id_and_tags]": [
        {
            "name": "The ClassName.MethodName that made the call",
            "verb": "HTTP Verb",
```

(continues on next page)

(continued from previous page)

```
            "service": "Name of the service",
            "url": "A shortened version of the URL called",
            "status_code": "The status code of the response",
            "request_headers": "The headers of the request",
            "request_body": "The body of the request",
            "response_headers": "The headers of the response",
            "response_body": "The body of the response"
        }
    ]
}
```

3.2.4 Tempest Workspace

Manages Tempest workspaces

This command is used for managing tempest workspaces

Commands

list

Outputs the name and path of all known tempest workspaces

register

Registers a new tempest workspace via a given `--name` and `--path`

rename

Renames a tempest workspace from `--old-name` to `--new-name`

move

Changes the path of a given tempest workspace `--name` to `--path`

remove

Deletes the entry for a given tempest workspace `--name`

`--rmdir` Deletes the given tempest workspace directory

General Options

- `--workspace_path`: Allows the user to specify a different location for the workspace.yaml file containing the workspace definitions instead of `~/.tempest/workspace.yaml`

3.2.5 Tempest Run

Runs tempest tests

This command is used for running the tempest tests

Test Selection

Tempest run has several options:

- `--regex/-r`: This is a selection regex like what stestr uses. It will run any tests that match on re.match() with the regex
- `--smoke/-s`: Run all the tests tagged as smoke
- `--exclude-regex`: It allows to do simple test exclusion via passing a rejection/exclude regexp

There are also the `--exclude-list` and `--include-list` options that let you pass a filepath to tempest run with the file format being a line separated regex, with # used to signify the start of a comment on a line. For example:

```
# Regex file
^regex1 # Match these tests
.*regex2 # Match those tests
```

These arguments are just passed into stestr, you can refer to the stestr selection docs for more details on how these operate: http://stestr.readthedocs.io/en/latest/MANUAL.html#test-selection

You can also use the `--list-tests` option in conjunction with selection arguments to list which tests will be run.

You can also use the `--load-list` option that lets you pass a filepath to tempest run with the file format being in a non-regex format, similar to the tests generated by the `--list-tests` option. You can specify target tests by removing unnecessary tests from a list file which is generated from `--list-tests` option.

You can also use `--worker-file` option that let you pass a filepath to a worker yaml file, allowing you to manually schedule the tests run. For example, you can setup a tempest run with different concurrences to be used with different regexps. An example of worker file is showed below:

```
# YAML Worker file
- worker:
  # you can have more than one regex per worker
  - tempest.api.*
  - neutron_tempest_tests
- worker:
  - tempest.scenario.*
```

This will run test matching with tempest.api.* and neutron_tempest_tests against worker 1. Run tests matching with tempest.scenario.* under worker 2.

You can mix manual scheduling with the standard scheduling mechanisms by concurrency field on a worker. For example:

```
# YAML Worker file
- worker:
  # you can have more than one regex per worker
  - tempest.api.*
  - neutron_tempest_tests
  concurrency: 3
- worker:
  - tempest.scenario.*
  concurrency: 2
```

This will run tests matching with tempest.scenario.* against 2 workers.

This worker file is passed into stestr. For some more details on how it operates please refer to the stestr scheduling docs: https://stestr.readthedocs.io/en/stable/MANUAL.html#test-scheduling

Test Execution

There are several options to control how the tests are executed. By default tempest will run in parallel with a worker for each CPU present on the machine. If you want to adjust the number of workers use the `--concurrency` option and if you want to run tests serially use `--serial/-t`

Running with Workspaces

Tempest run enables you to run your tempest tests from any setup tempest workspace it relies on you having setup a tempest workspace with either the `tempest init` or `tempest workspace` commands. Then using the `--workspace` CLI option you can specify which one of your workspaces you want to run tempest from. Using this option you dont have to run Tempest directly with you current working directory being the workspace, Tempest will take care of managing everything to be executed from there.

Running from Anywhere

Tempest run provides you with an option to execute tempest from anywhere on your system. You are required to provide a config file in this case with the `--config-file` option. When run tempest will create a .stestr directory and a .stestr.conf file in your current working directory. This way you can use stestr commands directly to inspect the state of the previous run.

Test Output

By default tempest runs output to STDOUT will be generated using the subunit-trace output filter. But, if you would prefer a subunit v2 stream be output to STDOUT use the `--subunit` flag

Combining Runs

There are certain situations in which you want to split a single run of tempest across 2 executions of tempest run. (for example to run part of the tests serially and others in parallel) To accomplish this but still treat the results as a single run you can leverage the `--combine` option which will append the current runs results with the previous runs.

3.3 Supported OpenStack Releases and Python Versions

3.3.1 Supported OpenStack Releases and Python Versions

This document lists the officially supported OpenStack releases and python versions by Tempest.

Compatible OpenStack Releases

Tempest master supports the below OpenStack Releases:

- 2024.2
- 2024.1
- 2023.2

For older OpenStack Release:

For any older OpenStack Release than the listed above, Tempest master might work. But if Tempest master starts failing then, you can use the respective Tempest tag listed in OpenStack release page.

For example: OpenStack Stein: Tempest 20.0.0

- https://releases.openstack.org/stein/index.html#stein-tempest

How to use Tempest tag on Extended Maintenance stable branch:

- https://review.opendev.org/#/c/705098/

Supported Python Versions

Tempest master supports the below python versions:

- Python 3.9
- Python 3.10
- Python 3.11
- Python 3.12

3.4 Description of Tests

3.4.1 Description of Tests

OpenStack Services Integration Tests

scenario

scenario package

Submodules

scenario.manager module

class EncryptionScenarioTest(**args*, ***kwargs*)

Bases: *ScenarioTest*

Base class for encryption scenario tests

class NetworkScenarioTest(**args*, ***kwargs*)

Bases: *ScenarioTest*

Base class for network scenario tests.

This class provide helpers for network scenario tests, using the neutron API. Helpers from ancestor which use the nova network API are overridden with the neutron API.

This Class also enforces using Neutron instead of novanetwork. Subclassed tests will be skipped if Neutron is not enabled

class ObjectStorageScenarioTest(**args*, ***kwargs*)

Bases: *ScenarioTest*

Provide harness to do Object Storage scenario tests.

Subclasses implement the tests that use the methods provided by this class.

class ScenarioTest(*args, **kwargs)

Bases: BaseTestCase

Base class for scenario tests. Uses tempest own clients.

scenario.test_compute_unified_limits module

class ComputeProjectQuotaTest(*args, **kwargs)

Bases: ScenarioTest

The test base class for compute unified limits tests.

Dynamic credentials (unique tenants) are created on a per-class basis, so we test different quota limits in separate test classes to prevent a quota limit update in one test class from affecting a test running in another test class in parallel.

https://docs.openstack.org/tempest/latest/configuration.html#dynamic-credentials

class ServersQuotaTest(*args, **kwargs)

Bases: *ComputeProjectQuotaTest*

test_server_count_vcpu_memory_disk_quota()

Test idempotent id: 555d8bbf-d2ed-4e39-858c-4235899402d9

scenario.test_dashboard_basic_ops module

class HorizonHTMLParser(*, *convert_charrefs=True*)

Bases: HTMLParser

class TestDashboardBasicOps(*args, **kwargs)

Bases: BaseTestCase

The test suite for dashboard basic operations

This is a basic scenario test: * checks that the login page is available * logs in as a regular user * checks that the user home page loads without error

test_basic_scenario()

Test idempotent id: 4f8851b1-0e69-482b-b63b-84c6e76f6c80

scenario.test_encrypted_cinder_volumes module

class TestEncryptedCinderVolumes(*args, **kwargs)

Bases: EncryptionScenarioTest

The test suite for encrypted cinder volumes

This test is for verifying the functionality of encrypted cinder volumes.

For both LUKS (v1 & v2) and cryptsetup encryption types, this test performs the following:

- Boots an instance from an image (CONF.compute.image_ref)
- Creates an encryption type (as admin)
- Creates a volume of that encryption type (as a regular user)
- Attaches and detaches the encrypted volume to the instance

test_encrypted_cinder_volumes_cryptsetup()

Test idempotent id: cbc752ed-b716-4717-910f-956cce965722

test_encrypted_cinder_volumes_luks()

Test idempotent id: 79165fb4-5534-4b9d-8429-97ccffb8f86e

LUKs v1 decrypts volume through libvirt.

test_encrypted_cinder_volumes_luksv2()

Test idempotent id: 7abec0a3-61a0-42a5-9e36-ad3138fb38b4

LUKs v2 decrypts volume through os-brick.

scenario.test_instances_with_cinder_volumes module

class TestInstancesWithCinderVolumes(**args*, ***kwargs*)

Bases: ScenarioTest

This is cinder volumes test.

Tests are below: * test_instances_with_cinder_volumes_on_all_compute_nodes

test_instances_with_cinder_volumes_on_all_compute_nodes()

Test idempotent id: d0e3c1a3-4b0a-4b0e-8b0a-4b0e8b0a4b0e

Test instances with cinder volumes launches on all compute nodes

Steps:

1. Create an image
2. Create a keypair
3. Create a bootable volume from the image and of the given volume type
4. Boot an instance from the bootable volume on each available compute node, up to CONF.compute.min_compute_nodes
5. Create a volume using each volume_types_for_data_volume on all available compute nodes, up to CONF.compute.min_compute_nodes. Total number of volumes is equal to compute nodes * len(volume_types_for_data_volume)
6. Assign floating IP to all instances
7. Configure security group for ssh access to all instances
8. Confirm ssh access to all instances
9. Attach volumes to the instances; fixup device mapping if required
10. Run write test to all volumes through ssh connection per instance
11. Clean up the sources, an instance, volumes, keypair and image

scenario.test_minimum_basic module

class TestMinimumBasicScenario(**args*, ***kwargs*)

Bases: ScenarioTest

This is a basic minimum scenario test.

These tests below: * across the multiple components * as a regular user * with and without optional parameters * check command outputs

test_minimum_basic_instance_hard_reboot_after_vol_snap_deletion()

Test idempotent id: a8fd48ec-1d01-4895-b932-02321661ec1e

Test compute hard reboot after volume snapshot deleted

Steps: 1. Create image 2. Create keypair 3. Boot instance with keypair and get list of instances 4. Create volume and show list of volumes 5. Attach volume to instance and getlist of volumes 6. Create a snapshot from volume 7. Add IP to instance 8. Create and add security group to instance 9. Check SSH connection to instance 10. Write data timestamp to the attached volume 11. Delete volume snapshot before reboot instance 12. Reboot instance (HARD) 13. Check SSH connection to instance after reboot 14. Verify attached disk data timestamp post instance reboot

test_minimum_basic_scenario()

Test idempotent id: bdbb5441-9204-419d-a225-b4fdbfb1a1a8

This is a basic minimum scenario with multiple components

Steps: 1. Create image 2. Create keypair 3. Boot instance with keypair and get list of instances 4. Create volume and show list of volumes 5. Attach volume to instance and getlist of volumes 6. Add IP to instance 7. Create and add security group to instance 8. Check SSH connection to instance 9. Reboot instance 10. Check SSH connection to instance after reboot

scenario.test_network_advanced_server_ops module

class BaseTestNetworkAdvancedServerOps(**args*, ***kwargs*)

Bases: NetworkScenarioTest

Base class for defining methods used in tests.

class TestNetworkAdvancedServerMigrationWithHost(**args*, ***kwargs*)

Bases: *BaseTestNetworkAdvancedServerOps*

Check VM connectivity with specifying source and destination hosts:

- Resize an instance by creating server on configured source host
- **Migrate server by creating it on configured source host and migrate it**
  - Cold Migration
  - Cold Migration with revert
  - Live Migration

test_server_connectivity_cold_migration()

Test idempotent id: 14f0c9e6-79ae-11ee-b962-0242ac120002

test_server_connectivity_cold_migration_revert()

Test idempotent id: 2627789a-79ae-11ee-b962-0242ac120002

test_server_connectivity_live_migration()

Test idempotent id: 1c13933e-79ae-11ee-b962-0242ac120002

test_server_connectivity_resize()

Test idempotent id: 06e23934-79ae-11ee-b962-0242ac120002

class TestNetworkAdvancedServerOps(**args*, ***kwargs*)

Bases: *BaseTestNetworkAdvancedServerOps*

Check VM connectivity after some advanced instance operations executed:

- Stop/Start an instance
- Reboot an instance
- Rebuild an instance
- Pause/Unpause an instance
- Suspend/Resume an instance

test_server_connectivity_cold_migration()

Test idempotent id: a4858f6c-401e-4155-9a49-d5cd053d1a2f

test_server_connectivity_cold_migration_revert()

Test idempotent id: 25b188d7-0183-4b1e-a11d-15840c8e2fd6

test_server_connectivity_live_migration()

Test idempotent id: 03fd1562-faad-11e7-9ea0-fa163e65f5ce

test_server_connectivity_pause_unpause()

Test idempotent id: 2b2642db-6568-4b35-b812-eceed3fa20ce

test_server_connectivity_reboot()

Test idempotent id: 7b6860c2-afa3-4846-9522-adeb38dfbe08

test_server_connectivity_rebuild()

Test idempotent id: 88a529c2-1daa-4c85-9aec-d541ba3eb699

test_server_connectivity_resize()

Test idempotent id: 719eb59d-2f42-4b66-b8b1-bb1254473967

test_server_connectivity_stop_start()

Test idempotent id: 61f1aa9a-1573-410e-9054-afa557cab021

test_server_connectivity_suspend_resume()

Test idempotent id: 5cdf9499-541d-4923-804e-b9a60620a7f0

scenario.test_network_basic_ops module

class Floating_IP_tuple(*floating_ip*, *server*)

Bases: `tuple`

class TestNetworkBasicOps(**args*, ***kwargs*)

Bases: `NetworkScenarioTest`

The test suite of network basic operations

This smoke test suite assumes that Nova has been configured to boot VMs with Neutron-managed networking, and attempts to verify network connectivity as follows:

There are presumed to be two types of networks: tenant and public. A tenant network may or may not be reachable from the Tempest host. A public network is assumed to be reachable from the Tempest host, and it should be possible to associate a public (floating) IP address with a tenant (fixed) IP address to facilitate external connectivity to a potentially unroutable tenant IP address.

This test suite can be configured to test network connectivity to a VM via a tenant network, a public network, or both. If both networking types are to be evaluated, tests that need to be executed remotely on the VM (via ssh) will only be run against one of the networks (to minimize test execution time).

Determine which types of networks to test as follows:

- Configure tenant network checks (via the project_networks_reachable key) if the Tempest host should have direct connectivity to tenant networks. This is likely to be the case if Tempest is running on the same host as a single-node devstack installation with IP namespaces disabled.
- Configure checks for a public network if a public network has been configured prior to the test suite being run and if the Tempest host should have connectivity to that public network. Checking connectivity for a public network requires that a value be provided for public_network_id. A value can optionally be provided for public_router_id if tenants will use a shared router to access a public network (as is likely to be the case when IP namespaces are not enabled). If a value is not provided for public_router_id, a router will be created for each tenant and use the network identified by public_network_id as its gateway.

`test_connectivity_between_vms_on_different_networks()`

Test idempotent id: 1546850e-fbaa-42f5-8b5f-03d8a6a95f15

Test connectivity between VMs on different networks

For a freshly-booted VM with an IP address (port) on a given network:

- the Tempest host can ping the IP address.
- **the Tempest host can ssh into the VM via the IP address and** successfully execute the following:
  - ping an external IP address, implying external connectivity.
  - **ping an external hostname, implying that dns is correctly** configured.
  - **ping an internal IP address, implying connectivity to another** VM on the same network.
- **Create another network on the same tenant with subnet, create** an VM on the new network.
  - **Ping the new VM from previous VM failed since the new network** was not attached to router yet.
  - **Attach the new network to the router, Ping the new VM from** previous VM succeed.

`test_hotplug_nic()`

Test idempotent id: c5adff73-e961-41f1-b4a9-343614f18cfa

Test hotplug network interface

1. Create a network and a VM.
2. Check connectivity to the VM via a public network.
3. Create a new network, with no gateway.
4. Bring up a new interface
5. check the VM reach the new network

`test_mtu_sized_frames()`

Test idempotent id: b158ea55-472e-4086-8fa9-c64ac0c6c1d0

Validate that network MTU sized frames fit through.

`test_network_basic_ops()`

Test idempotent id: f323b3ba-82f8-4db7-8ea6-6a895869ec49

Basic network operation test

For a freshly-booted VM with an IP address (port) on a given network:

- **the Tempest host can ping the IP address. This implies, but**
  does not guarantee (see the ssh check that follows), that the VM has been assigned the correct IP address and has connectivity to the Tempest host.
- **the Tempest host can perform key-based authentication to an**
  ssh server hosted at the IP address. This check guarantees that the IP address is associated with the target VM.
- **the Tempest host can ssh into the VM via the IP address and**
  successfully execute the following:
  - ping an external IP address, implying external connectivity.
  - **ping an external hostname, implying that dns is correctly**
    configured.
  - **ping an internal IP address, implying connectivity to another**
    VM on the same network.
- **detach the floating-ip from the VM and verify that it becomes**
  unreachable
- **associate detached floating ip to a new VM and verify connectivity.**
  VMs are created with unique keypair so connectivity also asserts that floating IP is associated with the new VM instead of the old one

Verifies that floating IP status is updated correctly after each change

`test_port_security_macspoofing_port()`

Test idempotent id: 7c0bb1a2-d053-49a4-98f9-ca1a1d849f63

Tests port_security extension enforces mac spoofing

Neutron security groups always apply anti-spoof rules on the VMs. This allows traffic to originate and terminate at the VM as expected, but prevents traffic to pass through the VM. Anti-spoof rules are not required in cases where the VM routes traffic through it.

The test steps are:

1. Create a new network.
2. Connect (hotplug) the VM to a new network.
3. Check the VM can ping a server on the new network (peer)
4. Spoof the mac address of the new VM interface.
5. Check the Security Group enforces mac spoofing and blocks pings via spoofed interface (VM cannot ping the peer).
6. Disable port-security of the spoofed port- set the flag to false.
7. Retest 3rd step and check that the Security Group allows pings via the spoofed interface.

`test_preserve_preexisting_port()`

Test idempotent id: 759462e1-8535-46b0-ab3a-33aa45c55aaa

Test preserve pre-existing port

Tests that a pre-existing port provided on server boot is not deleted if the server is deleted.

Nova should unbind the port from the instance on delete if the port was not created by Nova as part of the boot request.

We should also be able to boot another server with the same port.

`test_router_rescheduling()`

Test idempotent id: 2e788c46-fb3f-4ac9-8f82-0561555bea73

Tests that router can be removed from agent and add to a new agent.

1. Verify connectivity
2. Remove router from all l3-agents
3. Verify connectivity is down
4. Assign router to new l3-agent (or old one if no new agent is available)
5. Verify connectivity

`test_subnet_details()`

Test idempotent id: d8bb918e-e2df-48b2-97cd-b73c95450980

Tests that subnets extra configuration details are affecting VMs.

This test relies on non-shared, isolated tenant networks.

NOTE: Neutron subnets push data to servers via dhcp-agent, so any update in subnet requires server to actively renew its DHCP lease.

1. Configure subnet with dns nameserver
2. retrieve the VMs configured dns and verify it matches the one configured for the subnet.
3. update subnets dns

4. retrieve the VMs configured dns and verify it matches the new one configured for the subnet.

TODO(yfried): add host_routes

any resolution check would be testing either:

- l3 forwarding (tested in test_network_basic_ops)
- Name resolution of an external DNS nameserver - out of scope for Tempest

test_update_instance_port_admin_state()

Test idempotent id: f5dfcc22-45fd-409f-954c-5bd500d7890b

Test to update admin_state_up attribute of instance port

1. **Check public and project connectivity before updating** admin_state_up attribute of instance port to False
2. **Check public and project connectivity after updating** admin_state_up attribute of instance port to False
3. **Check public and project connectivity after updating** admin_state_up attribute of instance port to True

test_update_router_admin_state()

Test idempotent id: 04b9fe4e-85e8-4aea-b937-ea93885ac59f

Test to update admin state up of router

1. **Check public connectivity before updating** admin_state_up attribute of router to False
2. **Check public connectivity after updating** admin_state_up attribute of router to False
3. **Check public connectivity after updating** admin_state_up attribute of router to True

scenario.test_network_qos_placement module

class MinBwAllocationPlacementTest(**args*, ***kwargs*)

Bases: *NetworkQoSPlacementTestBase*

test_empty_update()

Test idempotent id: 0805779e-e03c-44fb-900f-ce97a790653b

test_migrate_with_qos_min_bw_allocation()

Test idempotent id: 8a98150c-a506-49a5-96c6-73a5e7b04ada

Scenario to migrate VM with QoS min bw allocation in placement

Boot a VM like in test_qos_min_bw_allocation_basic, do the same checks, and * migrate the server * confirm the resize, if the VM state is VERIFY_RESIZE * If the VM goes to ACTIVE state check that allocations are as expected.

test_qos_min_bw_allocation_basic()

Test idempotent id: 78625d92-212c-400e-8695-dd51706858b8

Basic scenario with QoS min bw allocation in placement.

Steps: * Create prerequisites: ** VLAN type provider network with subnet. ** valid QoS policy with minimum bandwidth rule with min_kbps=1 (This is a simplification to skip the checks in placement for detecting the resource provider tree and inventories, as if bandwidth resource is available 1 kbs will be available). ** invalid QoS policy with minimum bandwidth rule with min_kbs=max integer from placement (this is a simplification again to avoid detection of RP tress and inventories, as placement will reject such big allocation). * Create port with valid QoS policy, and boot VM with that, it should pass. * Create port with invalid QoS policy, and try to boot VM with that, it should fail.

test_qos_min_bw_allocation_update_policy()

Test idempotent id: 79fdaa1c-df62-4738-a0f0-1cff9dc415f6

Test the update of QoS policy on bound port

Related RFE in neutron: #1882804 The scenario is the following: * Have a port with QoS policy and minimum bandwidth rule. * Boot a VM with the port. * Update the port with a new policy with different minimum bandwidth values. * The allocation on placement side should be according to the new rules.

test_qos_min_bw_allocation_update_policy_direction_change()

Test idempotent id: 372b2728-cfed-469a-b5f6-b75779e1ccbe

Test QoS min bw direction change on a bound port

Related RFE in neutron: #1882804 The scenario is the following: * Have a port with QoS policy and minimum bandwidth rule with ingress direction * Boot a VM with the port. * Update the port with a new policy to egress direction in minimum bandwidth rule. * The allocation on placement side should be according to the new rules.

test_qos_min_bw_allocation_update_policy_from_zero()

Test idempotent id: 9cfc3bb8-f433-4c91-87b6-747cadc8958a

Test port without QoS policy to have QoS policy

This scenario checks if updating a port without QoS policy to have QoS policy with minimum_bandwidth rule succeeds only on controlplane, but placement allocation remains 0.

test_qos_min_bw_allocation_update_policy_to_zero()

Test idempotent id: a9725a70-1d28-4e3b-ae0e-450abc235962

Test port with QoS policy to remove QoS policy

In this scenario port with QoS minimum_bandwidth rule update to remove QoS policy results in 0 placement allocation.

test_qos_min_bw_allocation_update_with_multiple_ports()

Test idempotent id: 756ced7f-6f1a-43e7-a851-2fcfc16f3dd7

test_resize_with_qos_min_bw_allocation()

Test idempotent id: c29e7fd3-035d-4993-880f-70819847683f

Scenario to resize VM with QoS min bw allocation in placement.

Boot a VM like in test_qos_min_bw_allocation_basic, do the same checks, and * resize the server with new flavor * confirm the resize, if the VM state is VERIFY_RESIZE * If the VM goes to ACTIVE state check that allocations are as expected.

class NetworkQoSPlacementTestBase(**args*, ***kwargs*)

Bases: `NetworkScenarioTest`

Base class for Network QoS testing

Base class for testing Network QoS scenarios involving placement resource allocations.

class QoSBandwidthAndPacketRateTests(**args*, ***kwargs*)

Bases: *NetworkQoSPlacementTestBase*

test_interface_attach_detach()

Test idempotent id: 0393d038-03ad-4844-a0e4-83010f69dabb

test_qos_policy_update_on_bound_port()

Test idempotent id: fdb260e3-caa5-482d-ac7c-8c22adf3d750

test_qos_policy_update_on_bound_port_additional_rule()

Test idempotent id: f5864761-966c-4e49-b430-ac0044b7d658

test_qos_policy_update_on_bound_port_from_null_policy()

Test idempotent id: e6a20125-a02e-49f5-bcf6-894305ee3715

test_qos_policy_update_on_bound_port_to_null_policy()

Test idempotent id: fbbb9c81-ed21-48c3-bdba-ce2361e93aad

test_server_create_delete()

Test idempotent id: 93d1a88d-235e-4b7b-b44d-2a17dcf4e213

test_server_create_no_valid_host_due_to_bandwidth()

Test idempotent id: 915dd2ce-4890-40c8-9db6-f3e04080c6c1

test_server_create_no_valid_host_due_to_packet_rate()

Test idempotent id: 2d4a755e-10b9-4ac0-bef2-3f89de1f150b

test_server_live_migrate()

Test idempotent id: 36ffdb85-6cc2-4cc9-a426-cad5bac8626b

test_server_migrate()

Test idempotent id: bdd0b31c-c8b0-4b7b-b80a-545a46b32abe

test_server_resize()

Test idempotent id: 69d93e4f-0dfc-4d17-8d84-cc5c3c842cd5

test_server_resize_revert()

Test idempotent id: d01d4aee-ca06-4e4e-add7-8a47fe0daf96

scenario.test_network_v6 module

class TestGettingAddress(**args*, ***kwargs*)

Bases: `NetworkScenarioTest`

Test Summary:

1. **Create network with subnets:**
   1.1. one IPv4 and 1.2. one or more IPv6 in a given address mode
2. Boot 2 VMs on this network
3. Allocate and assign 2 FIP4
4. Check that vNICs of all VMs gets all addresses actually assigned
5. Each VM will ping the others v4 private address
6. If ping6 available in VM, each VM will ping all of the others v6 addresses as well as the routers

test_dhcp6_stateless_from_os()

Test idempotent id: d7e1f858-187c-45a6-89c9-bdafde619a9f

test_dualnet_dhcp6_stateless_from_os()

Test idempotent id: 76f26acd-9688-42b4-bc3e-cd134c4cb09e

test_dualnet_multi_prefix_dhcpv6_stateless()

Test idempotent id: cf1c4425-766b-45b8-be35-e2959728eb00

test_dualnet_multi_prefix_slaac()

Test idempotent id: 9178ad42-10e4-47e9-8987-e02b170cc5cd

test_dualnet_slaac_from_os()

Test idempotent id: b6399d76-4438-4658-bcf5-0d6c8584fde2

test_multi_prefix_dhcpv6_stateless()

Test idempotent id: 7ab23f41-833b-4a16-a7c9-5b42fe6d4123

test_multi_prefix_slaac()

Test idempotent id: dec222b1-180c-4098-b8c5-cc1b8342d611

test_slaac_from_os()

Test idempotent id: 2c92df61-29f0-4eaa-bee3-7c65bef62a43

scenario.test_object_storage_basic_ops module

class TestObjectStorageBasicOps(**args*, ***kwargs*)

Bases: ObjectStorageScenarioTest

test_swift_acl_anonymous_download()

Test idempotent id: 916c7111-cb1f-44b2-816d-8f760e4ea910

This test will cover below steps:

1. Create container
2. Upload object to the new container
3. Change the ACL of the container
4. Check if the object can be download by anonymous user
5. Delete the object and container

`test_swift_basic_ops()`

Test idempotent id: b920faf1-7b8a-4657-b9fe-9c4512bfb381

Test swift basic ops.

- get swift stat.
- create container.
- upload a file to the created container.
- list containers objects and assure that the uploaded file is present.
- download the object and check the content
- delete object from container.
- list containers objects and assure that the deleted file is gone.
- delete a container.

scenario.test_security_groups_basic_ops module

`class TestSecurityGroupsBasicOps(*args, **kwargs)`

Bases: `NetworkScenarioTest`

The test suite for security groups

This test suite assumes that Nova has been configured to boot VMs with Neutron-managed networking, and attempts to verify cross tenant connectivity as follows

ssh:
in order to overcome ip namespace, each tenant has an access point VM with floating-ip open to incoming ssh connection allowing network commands (ping/ssh) to be executed from within the tenant-network-namespace Tempest host performs key-based authentication to the ssh server via floating IP address

connectivity test is done by pinging destination server via source server ssh connection. success - ping returns failure - ping_timeout reached

multi-node:
Multi-Node mode is enabled when CONF.compute.min_compute_nodes > 1. Tests connectivity between servers on different compute nodes. When enabled, test will boot each new server to different compute nodes.

setup:

for primary tenant:

1. create a network&subnet
2. create a router (if public router isnt configured)
3. connect tenant network to public network via router
4. **create an access point:**
   a. a security group open to incoming ssh connection
   b. a VM with a floating ip

5. create a general empty security group (same as default, but without rules allowing in-tenant traffic)

tests:

1. _verify_network_details
2. _verify_mac_addr: for each access point verify that (subnet, fix_ip, mac address) are as defined in the port list
3. _test_in_tenant_block: test that in-tenant traffic is disabled without rules allowing it
4. _test_in_tenant_allow: test that in-tenant traffic is enabled once an appropriate rule has been created
5. _test_cross_tenant_block: test that cross-tenant traffic is disabled without a rule allowing it on destination tenant
6. **_test_cross_tenant_allow:**
   - test that cross-tenant traffic is enabled once an appropriate rule has been created on destination tenant.
   - test that reverse traffic is still blocked
   - test than reverse traffic is enabled once an appropriate rule has been created on source tenant
7. **_test_port_update_new_security_group:**
   - test that traffic is blocked with default security group
   - test that traffic is enabled after updating port with new security group having appropriate rule
8. _test_multiple_security_groups: test multiple security groups can be associated with the vm

assumptions:

1. alt_tenant/user existed and is different from primary_tenant/user
2. Public network is defined and reachable from the Tempest host
3. **Public router can either be:**
   - defined, in which case all tenants networks can connect directly to it, and cross tenant check will be done on the private IP of the destination tenant

   or

   - not defined (empty string), in which case each tenant will have its own router connected to the public network

`test_boot_into_disabled_port_security_network_without_secgroup()`

Test idempotent id: 13ccf253-e5ad-424b-9c4a-97b88a026699

`test_cross_tenant_traffic()`

Test idempotent id: e79f879e-debb-440c-a7e4-efeda05b6848

`test_in_tenant_traffic()`

Test idempotent id: 63163892-bbf6-4249-aa12-d5ea1f8f421b

test_multiple_security_groups()

Test idempotent id: d2f77418-fcc4-439d-b935-72eca704e293

Verify multiple security groups and checks that rules

provided in the both the groups is applied onto VM

test_port_security_disable_security_group()

Test idempotent id: 7c811dcc-263b-49a3-92d2-1b4d8405f50c

Verify the default security group rules is disabled.

test_port_update_new_security_group()

Test idempotent id: f4d556d7-1526-42ad-bafb-6bebf48568f6

Verifies the traffic after updating the vm port

With new security group having appropriate rule.

scenario.test_server_advanced_ops module

class TestServerAdvancedOps(**args*, ***kwargs*)

Bases: ScenarioTest

The test suite for server advanced operations

This test case stresses some advanced server instance operations:

- Sequence suspend resume

test_server_sequence_suspend_resume()

Test idempotent id: 949da7d5-72c8-4808-8802-e3d70df98e2c

scenario.test_server_basic_ops module

class TestServerBasicOps(**args*, ***kwargs*)

Bases: ScenarioTest

The test suite for server basic operations

This smoke test case follows this basic set of operations:

- Create a keypair for use in launching an instance
- Create a security group to control network access in instance
- Add simple permissive rules to the security group
- Launch an instance
- Perform ssh to instance
- Verify metadata service
- Verify metadata on config_drive
- Terminate the instance

test_server_basic_ops()

Test idempotent id: 7fff3fb3-91d8-4fd0-bd7d-0204f1f180ba

scenario.test_server_multinode module

class TestServerMultinode(**args*, ***kwargs*)

Bases: ScenarioTest

This is a set of tests specific to multinode testing.

test_schedule_to_all_nodes()

Test idempotent id: 9cecbe35-b9d4-48da-a37e-7ce70aa43d30

scenario.test_server_volume_attachment module

class BaseAttachmentTest(**args*, ***kwargs*)

Bases: ScenarioTest

class TestServerVolumeAttachScenarioOldVersion(**args*, ***kwargs*)

Bases: *BaseAttachmentTest*

test_old_versions_reject()

Test idempotent id: 6f4d2144-99f4-495c-8b0b-c6a537971418

class TestServerVolumeAttachmentScenario(**args*, ***kwargs*)

Bases: *BaseAttachmentTest*

Test server attachment behaviors

This tests that volume attachments to servers may not be removed directly and are only allowed through the compute service (bug #2004555).

test_server_detach_rules()

Test idempotent id: be615530-f105-437a-8afe-ce998c9535d9

Test that various methods of detaching a volume honors the rules

scenario.test_shelve_instance module

class TestShelveInstance(**args*, ***kwargs*)

Bases: ScenarioTest

This test shelves then unshelves a Nova instance

The following is the scenario outline:

- boot an instance and create a timestamp file in it
- shelve the instance
- unshelve the instance
- check the existence of the timestamp file in the unshelved instance
- check the existence of the timestamp file in the unshelved instance, after a cold migrate

test_cold_migrate_unshelved_instance()

Test idempotent id: 1295fd9e-193a-4cf8-b211-55358e021bae

test_shelve_instance()

Test idempotent id: 1164e700-0af0-4a4c-8792-35909a88743c

test_shelve_volume_backed_instance()

Test idempotent id: c1b6318c-b9da-490b-9c67-9339b627271f

scenario.test_snapshot_pattern module

class TestSnapshotPattern(**args*, ***kwargs*)

Bases: ScenarioTest

This test is for snapshotting an instance and booting with it.

The following is the scenario outline:

- boot an instance and create a timestamp file in it
- snapshot the instance
- add version metadata to the snapshot image
- boot a second instance from the snapshot
- check the existence of the timestamp file in the second instance
- snapshot the instance again

test_snapshot_pattern()

Test idempotent id: 608e604b-1d63-4a82-8e3e-91bc665c90b4

scenario.test_stamp_pattern module

class TestStampPattern(**args*, ***kwargs*)

Bases: ScenarioTest

The test suite for both snapshoting and attaching of volume

This test is for snapshotting an instance/volume and attaching the volume created from snapshot to the instance booted from snapshot. The following is the scenario outline: 1. Boot an instance instance1 2. Create a volume volume1 3. Attach volume1 to instance1 4. Create a filesystem on volume1 5. Mount volume1 6. Create a file which timestamp is written in volume1 7. Unmount volume1 8. Detach volume1 from instance1 9. Get a snapshot snapshot_from_volume of volume1 10. Get a snapshot snapshot_from_instance of instance1 11. Boot an instance instance2 from snapshot_from_instance 12. Create a volume volume2 from snapshot_from_volume 13. Attach volume2 to instance2 14. Check the existence of a file which created at 6. in volume2

test_stamp_pattern()

Test idempotent id: 10fd234a-515c-41e5-b092-8323060598c5

scenario.test_unified_limits module

class ImageQuotaTest(**args*, ***kwargs*)

Bases: ScenarioTest

test_image_count_quota()

Test idempotent id: 9b74fe24-183b-41e6-bf42-84c2958a7be8

test_image_count_uploading_quota()

Test idempotent id: b103788b-5329-4aa9-8b0d-97f8733460db

test_image_size_quota()

Test idempotent id: 05e8d064-c39a-4801-8c6a-465df375ec5b

test_image_stage_quota()

Test idempotent id: fc76b8d9-aae5-46fb-9285-099e37f311f7

scenario.test_volume_backup_restore module

class TestVolumeBackupRestore(*args, **kwargs)

Bases: ScenarioTest

Test cinder backup and restore

This testcase verifies content preservation after backup and restore operations by booting a server from a restored backup and check the connectivity to it.

The following is the scenario outline: 1. Create volume from image. 2. Create a backup for the volume. 3. Restore the backup. 4. Boot a server from the restored backup. 5. Create a floating ip. 6. Check server connectivity.

test_volume_backup_restore()

Test idempotent id: 2ce5e55c-4085-43c1-98c6-582525334ad7

scenario.test_volume_boot_pattern module

class TestVolumeBootPattern(*args, **kwargs)

Bases: EncryptionScenarioTest

test_boot_server_from_encrypted_volume_luks()

Test idempotent id: cb78919a-e553-4bab-b73b-10cf4d2eb125

LUKs v1 decrypts volume through libvirt.

test_boot_server_from_encrypted_volume_luksv2()

Test idempotent id: 5ab6100f-1b31-4dd0-a774-68cfd837ef77

LUKs v2 decrypts volume through os-brick.

test_bootable_volume_snapshot_stop_start_instance()

Test idempotent id: e3f4f2fc-5c6a-4be6-9c54-aedfc0954da7

test_create_server_from_volume_snapshot()

Test idempotent id: 05795fb2-b2a7-4c9f-8fac-ff25aedb1489

test_image_defined_boot_from_volume()

Test idempotent id: 36c34c67-7b54-4b59-b188-02a2f458a63b

test_volume_boot_pattern()

Test idempotent id: 557cd2c2-4eb8-4dce-98be-f86765ff311b

This test case attempts to reproduce the following steps:

- Create in Cinder some bootable volume importing a Glance image
- Boot an instance from the bootable volume
- Write content to the volume

- Delete an instance and Boot a new instance from the volume
- Check written content in the instance
- Create a volume snapshot while the instance is running
- Boot an additional instance from the new snapshot based volume
- Check written content in the instance booted from snapshot

scenario.test_volume_migrate_attached module

class TestVolumeMigrateRetypeAttached(**args*, ***kwargs*)

Bases: ScenarioTest

This test case attempts to reproduce the following steps:

- Create 2 volume types representing 2 different backends
- Create in Cinder some bootable volume importing a Glance image using
- volume_type_1
- Boot an instance from the bootable volume
- Write to the volume
- Perform a cinder retype on-demand of the volume to type of backend #2
- Check written content of migrated volume
- Check the type of the volume has been updated.
- Check the volume is still in-use and the migration was successful.
- Check that the same volume is attached to the instance.

test_volume_migrate_attached()

Test idempotent id: fe47b1ed-640e-4e3b-a090-200e25607362

test_volume_migrate_attached_data_volume()

Test idempotent id: 1b8661cb-db93-4110-860b-201295027b78

test_volume_retype_attached()

Test idempotent id: deadd2c2-beef-4dce-98be-f86765ff311b

test_volume_retype_attached_data_volume()

Test idempotent id: 122e070c-a5b2-470c-af2b-81e9dbefb9e8

Module contents

OpenStack Services API Tests

compute

compute package

Subpackages

compute.admin package

Submodules

compute.admin.test_aggregates_negative module

class AggregatesAdminNegativeTestJSON(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Tests Aggregates API that require admin privileges

test_aggregate_add_existent_host()

Test idempotent id: 19dd44e1-c435-4ee1-a402-88c4f90b5950

Adding already existing host to aggregate should fail

test_aggregate_add_host_as_user()

Test idempotent id: 7324c334-bd13-4c93-8521-5877322c3d51

Regular user is not allowed to add a host to an aggregate

test_aggregate_add_non_exist_host()

Test idempotent id: 0ef07828-12b4-45ba-87cc-41425faf5711

Adding a non-exist host to an aggregate should fail

test_aggregate_create_aggregate_name_length_exceeds_255()

Test idempotent id: 4c194563-543b-4e70-a719-557bbe947fac

The length of aggregate name should <=255

test_aggregate_create_aggregate_name_length_less_than_1()

Test idempotent id: 3b8a1929-3793-4e92-bcb4-dfa572ee6c1d

The length of aggregate name should >=1

test_aggregate_create_as_user()

Test idempotent id: 86a1cb14-da37-4a70-b056-903fd56dfe29

Regular user is not allowed to create an aggregate

test_aggregate_create_with_existent_aggregate_name()

Test idempotent id: 9c23a291-b0b1-487b-b464-132e061151b3

Creating an aggregate with existent aggregate name is forbidden

test_aggregate_delete_as_user()

Test idempotent id: cd6de795-c15d-45f1-8d9e-813c6bb72a3d

Regular user is not allowed to delete an aggregate

test_aggregate_delete_with_invalid_id()

Test idempotent id: c74f4bf1-4708-4ff2-95a0-f49eaca951bd

Delete an aggregate with invalid id should raise exceptions

test_aggregate_get_details_as_user()

Test idempotent id: 557cad12-34c9-4ff4-95f0-22f0dfbaf7dc

Regular user is not allowed to get aggregate details

test_aggregate_get_details_with_invalid_id()

Test idempotent id: 3c916244-2c46-49a4-9b55-b20bb0ae512c

Get aggregate details with invalid id should raise exceptions

test_aggregate_list_as_user()

Test idempotent id: b7d475a6-5dcd-4ff4-b70a-cd9de66a6672

Regular user is not allowed to list aggregates

test_aggregate_remove_host_as_user()

Test idempotent id: 7a53af20-137a-4e44-a4ae-e19260e626d9

Regular user is not allowed to remove a host from an aggregate

test_aggregate_remove_nonexistent_host()

Test idempotent id: 95d6a6fa-8da9-4426-84d0-eec0329f2e4d

Removing not existing host from aggregate should fail

compute.admin.test_assisted_volume_snapshots module

class VolumesAssistedSnapshotsTest(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test volume assisted snapshots

test_volume_assisted_snapshot_create_delete()

Test idempotent id: 8aee84a3-1b1f-42e4-9b00-613931ccac24

Test create/delete volume assisted snapshot

compute.admin.test_auto_allocate_network module

class AutoAllocateNetworkTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Tests auto-allocating networks with the v2.37 microversion.

These tests rely on Neutron being enabled. Also, the tenant must not have any network resources available to it so we can make sure that Nova calls to Neutron to automatically allocate the network topology.

test_server_create_no_allocate()

Test idempotent id: 5eb7b8fa-9c23-47a2-9d7d-02ed5809dd34

Tests that no networking is allocated for the server.

test_server_multi_create_auto_allocate()

Test idempotent id: 2e6cf129-9e28-4e8a-aaaa-045ea826b2a6

Tests that networking is auto-allocated for multiple servers.

compute.admin.test_availability_zone module

class AZAdminV2TestJSON(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Tests Availability Zone API List

test_get_availability_zone_list()

Test idempotent id: d3431479-8a09-4f76-aa2d-26dc580cb27c

Test listing availability zones

test_get_availability_zone_list_detail()

Test idempotent id: ef726c58-530f-44c2-968c-c7bed22d5b8c

Test listing availability zones with detail

compute.admin.test_availability_zone_negative module

class AZAdminNegativeTestJSON(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Negative Tests of Availability Zone API List

test_get_availability_zone_list_detail_with_non_admin_user()

Test idempotent id: bf34dca2-fdc3-4073-9c02-7648d9eae0d7

Test listing availability zone with detail by non-admin user

List of availability zones and available services with non-administrator user is not allowed.

compute.admin.test_create_server module

class ServersWithSpecificFlavorTestJSON(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Test creating servers with specific flavor

test_verify_created_server_ephemeral_disk()

Test idempotent id: b3c7bcfc-bb5b-4e22-b517-c7f686b802ca

Verify that the ephemeral disk is created when creating server

compute.admin.test_delete_server module

class DeleteServersAdminTestJSON(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Test deletion of servers

test_admin_delete_servers_of_others()

Test idempotent id: 73177903-6737-4f27-a60c-379e8ae8cf48

Administrator can delete servers of others

test_delete_server_while_in_error_state()

Test idempotent id: 99774678-e072-49d1-9d2a-49a59bc56063

Delete a server while its VM state is error

compute.admin.test_flavors module

class FlavorsAdminTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Tests Flavors API Create and Delete that require admin privileges

test_create_flavor_using_string_ram()

Test idempotent id: 3b541a2e-2ac2-4b42-8b8d-ba6e22fcd4da

Test creating flavor with ram of type string

test_create_flavor_verify_entry_in_list_details()

Test idempotent id: 8261d7b0-be58-43ec-a2e5-300573c3f6c5

Create a flavor and ensure its details are listed

This operation requires the user to have admin role

test_create_flavor_with_int_id()

Test idempotent id: 8b4330e1-12c4-4554-9390-e6639971f086

Test creating flavor with id of type integer

test_create_flavor_with_none_id()

Test idempotent id: f83fe669-6758-448a-a85e-32d351f36fe0

Test creating flavor without id specified

If nova receives a request with None as flavor_id, nova generates flavor_id of uuid.

test_create_flavor_with_uuid_id()

Test idempotent id: 94c9bb4e-2c2a-4f3c-bb1f-5f0daf918e6d

Test creating flavor with id of type uuid

test_create_list_flavor_without_extra_data()

Test idempotent id: 63dc64e6-2e79-4fdf-868f-85500d308d66

Create a flavor and ensure it is listed

This operation requires the user to have admin role

test_create_server_with_non_public_flavor()

Test idempotent id: bcc418ef-799b-47cc-baa1-ce01368b8987

Create a flavor with os-flavor-access:is_public false

test_is_public_string_variations()

Test idempotent id: fb9cbde6-3a0e-41f2-a983-bdb0a823c44e

Test creating public and non public flavors

test_list_non_public_flavor()

Test idempotent id: be6cc18c-7c5d-48c0-ac16-17eaf03c54eb

Create a flavor with os-flavor-access:is_public false.

The flavor should not be present in list_details as the tenant is not automatically added access list. This operation requires the user to have admin role

test_list_public_flavor_with_other_user()

Test idempotent id: b345b196-bfbd-4231-8ac1-6d7fe15ff3a3

Create a Flavor with public access.

Try to List/Get flavor with another user

compute.admin.test_flavors_access module

class FlavorsAccessTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Tests Flavor Access API extension.

Add and remove Flavor Access require admin privileges.

test_flavor_access_add_remove()

Test idempotent id: 59e622f6-bdf6-45e3-8ba8-fedad905a6b4

Test add/remove flavor access to a given project

test_flavor_access_list_with_private_flavor()

Test idempotent id: ea2c2211-29fa-4db9-97c3-906d36fad3e0

Test listing flavor access for a private flavor

Listing flavor access on a newly created private flavor will return an empty access list.

compute.admin.test_flavors_access_negative module

class FlavorsAccessNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Tests Flavor Access API extension.

Add and remove Flavor Access require admin privileges.

test_add_flavor_access_duplicate()

Test idempotent id: f3592cc0-0306-483c-b210-9a7b5346eddc

Test adding duplicate flavor access to same flavor should fail

test_flavor_access_list_with_public_flavor()

Test idempotent id: 0621c53e-d45d-40e7-951d-43e5e257b272

Test listing flavor access of a public flavor should fail

test_flavor_non_admin_add()

Test idempotent id: 41eaaade-6d37-4f28-9c74-f21b46ca67bd

Test adding flavor access by a non-admin user is forbidden

test_flavor_non_admin_remove()

Test idempotent id: 073e79a6-c311-4525-82dc-6083d919cb3a

Test removing flavor access by a non-admin user should fail

test_remove_flavor_access_not_found()

Test idempotent id: 1f710927-3bc7-4381-9f82-0ca6e42644b7

Test removing non existent flavor access should fail

compute.admin.test_flavors_extra_specs module

class FlavorMetadataValidation(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

test_flavor_update_with_custom_namespace()

Test idempotent id: d3114f03-b0f2-4dc7-be11-70c0abc178b3

Test flavor creation with a custom namespace, key and value

class FlavorsExtraSpecsTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Tests Flavor Extra Spec API extension.

SET, UNSET, UPDATE Flavor Extra specs require admin privileges. GET Flavor Extra specs can be performed even by without admin privileges.

test_flavor_non_admin_get_all_keys()

Test idempotent id: a99dad88-ae1c-4fba-aeb4-32f898218bd0

Test non admin user getting all flavor extra spec keys

test_flavor_non_admin_get_specific_key()

Test idempotent id: 12805a7f-39a3-4042-b989-701d5cad9c90

Test non admin user getting specific flavor extra spec key

test_flavor_set_get_update_show_unset_keys()

Test idempotent id: 0b2f9d4b-1ca2-4b99-bb40-165d4bb94208

Test flavor extra spec operations by admin user

Test to SET, GET, UPDATE, SHOW, UNSET flavor extra spec as a user with admin privileges.

compute.admin.test_flavors_extra_specs_negative module

class FlavorsExtraSpecsNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Negative Tests Flavor Extra Spec API extension.

SET, UNSET, UPDATE Flavor Extra specs require admin privileges.

test_flavor_get_nonexistent_key()

Test idempotent id: 329a7be3-54b2-48be-8052-bf2ce4afd898

Getting non existence flavor extra spec key should fail

test_flavor_non_admin_set_keys()

Test idempotent id: a00a3b81-5641-45a8-ab2b-4a8ec41e1d7d

Test to SET flavor extra spec as a user without admin privileges

test_flavor_non_admin_unset_keys()

Test idempotent id: 28f12249-27c7-44c1-8810-1f382f316b11

non admin user is not allowed to unset flavor extra spec

test_flavor_non_admin_update_specific_key()

Test idempotent id: 1ebf4ef8-759e-48fe-a801-d451d80476fb

non admin user is not allowed to update flavor extra spec

test_flavor_unset_nonexistent_key()

Test idempotent id: 440b9f3f-3c7f-4293-a106-0ceda350f8de

Unsetting non existence flavor extra spec key should fail

test_flavor_update_mismatch_key()

Test idempotent id: 25b822b8-9f49-44f6-80de-d99f0482e5cb

Updating unmatched flavor extra spec key should fail

The key to be updated should match the key in the body

test_flavor_update_more_key()

Test idempotent id: f5889590-bf66-41cc-b4b1-6e6370cfd93f

Updating multiple flavor spec keys should fail

There should be just one item in the request body

compute.admin.test_flavors_microversions module

class FlavorsV255TestJSON(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Test flavors API with compute microversion greater than 2.54

test_crud_flavor()

Test idempotent id: 61976b25-488d-41dc-9dcb-cb9693a7b075

Test create/show/update/list flavor

Check the response schema of flavors API with microversion greater than 2.54.

class FlavorsV261TestJSON(*args, **kwargs)

Bases: *FlavorsV255TestJSON*

Test flavors API with compute microversion greater than 2.60

compute.admin.test_hosts module

class HostsAdminTestJSON(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Tests nova hosts API using admin privileges.

test_list_hosts()

Test idempotent id: 9bfaf98d-e2cb-44b0-a07e-2558b2821e4f

Listing nova hosts

test_list_hosts_with_a_blank_zone()

Test idempotent id: 9af3c171-fbf4-4150-a624-22109733c2a6

Listing nova hosts with blank availability zone

If send the request with a blank zone, the request will be successful and it will return all the hosts list

test_list_hosts_with_nonexistent_zone()

Test idempotent id: c6ddbadb-c94e-4500-b12f-8ffc43843ff8

Listing nova hosts with not existing availability zone.

If send the request with a nonexistent zone, the request will be successful and no hosts will be returned

test_list_hosts_with_zone()

Test idempotent id: 5dc06f5b-d887-47a2-bb2a-67762ef3c6de

Listing nova hosts with specified availability zone

test_show_host_detail()

Test idempotent id: 38adbb12-aee2-4498-8aec-329c72423aa4

Showing nova host details

compute.admin.test_hosts_negative module

class HostsAdminNegativeTestJSON(**args*, ***kwargs*)

Bases: `BaseV2ComputeAdminTest`

Tests hosts API using admin privileges.

test_list_hosts_with_non_admin_user()

Test idempotent id: dd032027-0210-4d9c-860e-69b1b8deed5f

Non admin user is not allowed to list hosts

test_reboot_host_with_non_admin_user()

Test idempotent id: 02d79bb9-eb57-4612-abf6-2cb38897d2f8

Non admin user is not allowed to reboot host

test_reboot_nonexistent_host()

Test idempotent id: f86bfd7b-0b13-4849-ae29-0322e83ee58b

Rebooting not existing host should fail

test_show_host_detail_with_non_admin_user()

Test idempotent id: 19ebe09c-bfd4-4b7c-81a2-e2e0710f59cc

Non admin user is not allowed to show host details

test_show_host_detail_with_nonexistent_hostname()

Test idempotent id: e75b0a1a-041f-47a1-8b4a-b72a6ff36d3f

Showing host detail with not existing hostname should fail

test_shutdown_host_with_non_admin_user()

Test idempotent id: a803529c-7e3f-4d3c-a7d6-8e1c203d27f6

Non admin user is not allowed to shutdown host

test_shutdown_nonexistent_host()

Test idempotent id: 9e637444-29cf-4244-88c8-831ae82c31b6

Shutting down not existing host should fail

test_startup_host_with_non_admin_user()

Test idempotent id: 9f4ebb7e-b2ae-4e5b-a38f-0fd1bb0ddfca

Non admin user is not allowed to startup host

test_startup_nonexistent_host()

Test idempotent id: 0d981ac3-4320-4898-b674-82b61fbb60e4

Starting up not existing host should fail

test_update_host_with_invalid_maintenance_mode()

Test idempotent id: ab1e230e-5e22-41a9-8699-82b9947915d4

Updating host to invalid maintenance mode should fail

maintenance_mode can only be enable or disable.

test_update_host_with_invalid_status()

Test idempotent id: fbe2bf3e-3246-4a95-a59f-94e4e298ec77

Updating host to invalid status should fail

status can only be enable or disable.

test_update_host_with_non_admin_user()

Test idempotent id: e40c72b1-0239-4ed6-ba21-81a184df1f7c

Non admin user is not allowed to update host

test_update_host_without_param()

Test idempotent id: 0cd85f75-6992-4a4a-b1bd-d11e37fd0eee

Updating host without param should fail

status or maintenance_mode is needed for host update

test_update_nonexistent_host()

Test idempotent id: 23c92146-2100-4d68-b2d6-c7ade970c9c1

Updating not existing host should fail

compute.admin.test_hypervisor module

class HypervisorAdminTestBase(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Tests Hypervisors API that require admin privileges

class HypervisorAdminTestJSON(**args*, ***kwargs*)

Bases: *HypervisorAdminTestBase*

Tests Hypervisors API that require admin privileges

test_get_hypervisor_list()

Test idempotent id: 7f0ceacd-c64d-4e96-b8ee-d02943142cc5

List of hypervisor and available hypervisors hostname

test_get_hypervisor_list_details()

Test idempotent id: 1e7fdac2-b672-4ad1-97a4-bad0e3030118

Display the details of the all hypervisor

test_get_hypervisor_show_details()

Test idempotent id: 94ff9eae-a183-428e-9cdb-79fde71211cc

Display the details of the specified hypervisor

test_get_hypervisor_stats()

Test idempotent id: 797e4f28-b6e0-454d-a548-80cc77c00816

Verify the stats of the all hypervisor

test_get_hypervisor_uptime()

Test idempotent id: 91a50d7d-1c2b-4f24-b55a-a1fe20efca70

Verify that GET shows the specified hypervisor uptime

class HypervisorAdminUnderV252Test(**args*, ***kwargs*)

Bases: *HypervisorAdminTestBase*

Tests Hypervisors API below 2.53 that require admin privileges

test_get_hypervisor_show_servers()

Test idempotent id: e81bba3f-6215-4e39-a286-d52d2f906862

Test showing instances about the specific hypervisors

test_search_hypervisor()

Test idempotent id: d7e1805b-3b14-4a3b-b6fd-50ec6d9f361f

Test searching for hypervisors by its name

class HypervisorAdminV228Test(**args*, ***kwargs*)

Bases: *HypervisorAdminTestBase*

Tests Hypervisors API higher than 2.27 that require admin privileges

test_get_list_hypervisor_details()

Test idempotent id: d46bab64-0fbe-4eb8-9133-e6ee56188cc5

Test listing and showing hypervisor details

class HypervisorAdminV253TestBase(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Tests Hypervisors API above 2.53 that require admin privileges

test_list_show_detail_hypervisors()

Test idempotent id: 4ab54a14-77a2-4e39-b9d2-1306d157c705

Verify the list, list details, and show hypevisors

This verify the Hypervisor API response schema with v2.53 microversion

compute.admin.test_hypervisor_negative module

class HypervisorAdminNegativeTestBase(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Tests Hypervisors API that require admin privileges

class HypervisorAdminNegativeTestJSON(*args, **kwargs)

Bases: *HypervisorAdminNegativeTestBase*

Tests Hypervisors API that require admin privileges

test_get_hypervisor_list_details_with_non_admin_user()

Test idempotent id: dc02db05-e801-4c5f-bc8e-d915290ab345

Test listing hypervisor details by non admin user should fail

test_get_hypervisor_list_with_non_admin_user()

Test idempotent id: 51b3d536-9b14-409c-9bce-c6f7c794994e

Test listing hypervisors by non admin user should fail

test_get_hypervisor_stats_with_non_admin_user()

Test idempotent id: e2b061bb-13f9-40d8-9d6e-d5bf17595849

Test getting hypervisor stats by non admin user should fail

test_get_hypervisor_uptime_with_non_admin_user()

Test idempotent id: 6c3461f9-c04c-4e2a-bebb-71dc9cb47df2

Test showing uptime of hypervisor by non admin user should fail

test_get_nonexistent_hypervisor_uptime()

Test idempotent id: f60aa680-9a3a-4c7d-90e1-fae3a4891303

Test showing uptime of non existent hypervisor should fail

test_show_hypervisor_with_non_admin_user()

Test idempotent id: 51e663d0-6b89-4817-a465-20aca0667d03

Test showing hypervisor by non admin user should fail

test_show_nonexistent_hypervisor()

Test idempotent id: c136086a-0f67-4b2b-bc61-8482bd68989f

Test showing non existent hypervisor should fail

class HypervisorAdminNegativeUnderV252Test(*args, **kwargs)

Bases: *HypervisorAdminNegativeTestBase*

Tests Hypervisors API below ver 2.53 that require admin privileges

test_search_hypervisor_with_non_admin_user()

Test idempotent id: 5b6a6c79-5dc1-4fa5-9c58-9c8085948e74

Test searching hypervisor by non admin user should fail

test_search_nonexistent_hypervisor()

Test idempotent id: 19a45cc1-1000-4055-b6d2-28e8b2ec4faa

Test searching non existent hypervisor should fail

test_show_servers_with_non_admin_user()

Test idempotent id: 2a0a3938-832e-4859-95bf-1c57c236b924

Test showing hypervisor servers by non admin user should fail

test_show_servers_with_nonexistent_hypervisor()

Test idempotent id: 02463d69-0ace-4d33-a4a8-93d7883a2bba

Test showing servers on non existent hypervisor should fail

compute.admin.test_instance_usage_audit_log module

class InstanceUsageAuditLogTestJSON(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Test instance usage audit logs API

test_get_instance_usage_audit_log()

Test idempotent id: 6e40459d-7c5f-400b-9e83-449fbc8e7feb

Test getting instance usage audit log before specified time

test_list_instance_usage_audit_logs()

Test idempotent id: 25319919-33d9-424f-9f99-2c203ee48b9d

Test listing instance usage audit logs

compute.admin.test_instance_usage_audit_log_negative module

class InstanceUsageAuditLogNegativeTestJSON(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Negative tests of instance usage audit logs

test_get_instance_usage_audit_logs_with_invalid_time()

Test idempotent id: 9b952047-3641-41c7-ba91-a809fc5974c8

Test showing instance usage audit logs with invalid time

Showing instance usage audit logs with invalid time should fail.

test_instance_usage_audit_logs_with_nonadmin_user()

Test idempotent id: a9d33178-d2c9-4131-ad3b-f4ca8d0308a2

Test list/show instance usage audit logs by non-admin should fail

The instance_usage_audit_logs API just can be accessed by admin user.

compute.admin.test_keypairs_v210 module

class KeyPairsV210TestJSON(**args*, ***kwargs*)

Bases: BaseKeypairTest

Tests KeyPairs API with microversion higher than 2.9

test_admin_manage_keypairs_for_other_users()

Test idempotent id: 3c8484af-cfb3-48f6-b8ba-d5d58bbf3eac

Test admin managing keypairs for other users

First admin creates a keypair for an other user, then admin lists keypairs filtered by that user, and keypairs created for that user should appear in the result and keypairs not created for that user should not appear in the result.

compute.admin.test_live_migration module

class LiveAutoBlockMigrationV225Test(**args*, ***kwargs*)

Bases: *LiveMigrationTest*

class LiveMigrationRemoteConsolesV26Test(**args*, ***kwargs*)

Bases: *LiveMigrationTestBase*

test_live_migration_serial_console()

Test idempotent id: 6190af80-513e-4f0f-90f2-9714e84955d7

Test the live-migration of an instance which has a serial console

The serial console feature of an instance uses ports on the host. These ports need to be updated when they are already in use by another instance on the target host. This test checks if this update behavior is correctly done, by connecting to the serial consoles of the instances before and after the live migration.

class LiveMigrationTest(**args*, ***kwargs*)

Bases: *LiveMigrationTestBase*

test_live_block_migration()

Test idempotent id: 1dce86b8-eb04-4c03-a9d8-9c1dc3ee0c7b

Test live migrating an active server

test_live_block_migration_paused()

Test idempotent id: 1e107f21-61b2-4988-8f22-b196e938ab88

Test live migrating a paused server

test_live_block_migration_with_attached_volume()

Test idempotent id: e19c0cc6-6720-4ed8-be83-b6603ed5c812

Test the live-migration of an instance with an attached volume.

This tests the live-migration of an instance with both a local disk and attach volume. This differs from test_volume_backed_live_migration above that tests live-migration with only an attached volume.

test_live_migration_with_trunk()

Test idempotent id: 0022c12e-a482-42b0-be2d-396b5f0cffe3

Test live migration with trunk and subport

test_volume_backed_live_migration()

Test idempotent id: 5071cf17-3004-4257-ae61-73a84e28badd

Test live migrating an active server booted from volume

class LiveMigrationTestBase(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test live migration operations supported by admin user

compute.admin.test_live_migration_negative module

class LiveMigrationNegativeTest(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Negative tests of live migration

test_invalid_host_for_migration()

Test idempotent id: 7fb7856e-ae92-44c9-861a-af62d7830bcb

Test migrating to an invalid host should not change the status

test_live_block_migration_suspended()

Test idempotent id: 6e2f94f5-2ee8-4830-bef5-5bc95bb0795b

Test migrating a suspended server should not change the status

compute.admin.test_migrations module

class MigrationsAdminTest(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test migration operations supported by admin user

test_cold_migration()

Test idempotent id: 4bf0be52-3b6f-4746-9a27-3143636fe30d

Test cold migrating server and then confirm the migration

test_list_migrations()

Test idempotent id: 75c0b83d-72a0-4cf8-a153-631e83e7d53f

Test admin user can get the migrations list

test_list_migrations_in_flavor_resize_situation()

Test idempotent id: 1b512062-8093-438e-b47a-37d2f597cd64

Admin can get the migrations list containing the resized server

test_resize_server_revert_deleted_flavor()

Test idempotent id: 33f1fec3-ba18-4470-8e4e-1d888e7c3593

Test reverting resized server with original flavor deleted

Tests that we can revert the resize on an instance whose original flavor has been deleted.

test_revert_cold_migration()

Test idempotent id: caa1aa8b-f4ef-4374-be0d-95f001c2ac2d

Test cold migrating server and then revert the migration

compute.admin.test_networks module

class NetworksTest(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Tests Nova Networks API that usually requires admin privileges.

API docs: https://docs.openstack.org/api-ref/compute/#networks-os-networks-deprecated

test_get_network()

Test idempotent id: d206d211-8912-486f-86e2-a9d090d1f416

Test getting network from nova side

test_list_all_networks()

Test idempotent id: df3d1046-6fa5-4b2c-ad0c-cfa46a351cb9

Test getting all networks from nova side

compute.admin.test_quotas module

class QuotaClassesAdmin257Test(**args*, ***kwargs*)

Bases: *QuotaClassesAdminTestJSON*

Test compute quotas with microversion greater than 2.56

NOTE(gmann): This test tests the Quota class APIs response schema # for 2.57 microversion. No specific assert or behaviour verification # is needed.

class QuotaClassesAdminTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Tests the os-quota-class-sets API to update default quotas.

test_update_default_quotas()

Test idempotent id: 7932ab0f-5136-4075-b201-c0e2338df51a

Test updating default compute quota class set

class QuotasAdminTestBase(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

class QuotasAdminTestJSON(**args*, ***kwargs*)

Bases: *QuotasAdminTestBase*

Test compute quotas by admin user

test_delete_quota()

Test idempotent id: 389d04f0-3a41-405f-9317-e5f86e3c44f0

Test admin can delete the compute quota set for a project

test_get_default_quotas()

Test idempotent id: 3b0a7c8f-cf58-46b8-a60c-715a32a8ba7d

Test admin can get the default compute quota set for a project

test_get_updated_quotas()

Test idempotent id: ce9e0815-8091-4abd-8345-7fe5b85faa1d

Test that GET shows the updated quota set of project

test_update_all_quota_resources_for_tenant()

Test idempotent id: 55fbe2bf-21a9-435b-bbd2-4162b0ed799a

Test admin can update all the compute quota limits for a project

class QuotasAdminTestV236(**args*, ***kwargs*)

Bases: *QuotasAdminTestBase*

Test compute quotas with microversion greater than 2.35

NOTE(gmann): This test tests the Quota APIs response schema # for 2.36 microversion. No specific assert or behaviour verification # is needed.

test_get_updated_quotas()

Test idempotent id: 4268b5c9-92e5-4adc-acf1-3a2798f3d803

Test compute quotas API with microversion greater than 2.35

Checking compute quota update, get, get details APIs response schema.

class QuotasAdminTestV257(**args*, ***kwargs*)

Bases: *QuotasAdminTestBase*

Test compute quotas with microversion greater than 2.56

NOTE(gmann): This test tests the Quota APIs response schema # for 2.57 microversion. No specific assert or behaviour verification # is needed.

test_get_updated_quotas()

Test idempotent id: e641e6c6-e86c-41a4-9e5c-9493c0ae47ad

Test compute quotas API with microversion greater than 2.56

Checking compute quota update, get, get details APIs response schema.

compute.admin.test_quotas_negative module

class QuotasAdminNegativeTest(**args*, ***kwargs*)

Bases: *QuotasAdminNegativeTestBase*

Negative tests of nova quotas

test_create_server_when_cpu_quota_is_full()

Test idempotent id: 91058876-9947-4807-9f22-f6eb17140d9b

Disallow server creation when tenants vcpu quota is full

test_create_server_when_instances_quota_is_full()

Test idempotent id: 7c6be468-0274-449a-81c3-ac1c32ee0161

Once instances quota limit is reached, disallow server creation

test_create_server_when_memory_quota_is_full()

Test idempotent id: 6fdd7012-584d-4327-a61c-49122e0d5864

Disallow server creation when tenants memory quota is full

test_update_quota_normal_user()

Test idempotent id: 733abfe8-166e-47bb-8363-23dbd7ff3476

Test updating nova quota by normal user should fail

class QuotasAdminNegativeTestBase(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

class QuotasSecurityGroupAdminNegativeTest(**args*, ***kwargs*)

Bases: *QuotasAdminNegativeTestBase*

Negative tests of nova security group quota

test_security_groups_exceed_limit()

Test idempotent id: 7c6c8f3b-2bf6-4918-b240-57b136a66aa0

Negative test: Creation Security Groups over limit should FAIL

test_security_groups_rules_exceed_limit()

Test idempotent id: 6e9f436d-f1ed-4f8e-a493-7275dfaa4b4d

Negative test: Creation of Security Group Rules should FAIL

compute.admin.test_security_groups module

class SecurityGroupsTestAdminJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test security groups API that requires admin privilege

Test security groups API that requires admin privilege with compute microversion less than 2.36

test_list_security_groups_list_all_tenants_filter()

Test idempotent id: 49667619-5af9-4c63-ab5d-2cfdd1c8f7f1

Test listing security groups with all_tenants filter

1. Create two security groups for non-admin user
2. Create two security groups for admin user
3. Fetch all security groups based on all_tenants search filter by admin, check that all four created security groups are present in fetched list
4. Fetch all security groups based on all_tenants search filter by non-admin, check only two security groups created by the provided non-admin user are present in fetched list

compute.admin.test_server_diagnostics module

class ServerDiagnosticsTest(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test server diagnostics with compute microversion less than 2.48

test_get_server_diagnostics()

Test idempotent id: 31ff3486-b8a0-4f56-a6c0-aab460531db3

Test getting server diagnostics

class ServerDiagnosticsV248Test(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test server diagnostics with compute microversion greater than 2.47

test_get_server_diagnostics()

Test idempotent id: 64d0d48c-dff1-11e6-bf01-fe55135034f3

Test getting server diagnostics

compute.admin.test_server_diagnostics_negative module

class ServerDiagnosticsNegativeTest(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Negative tests of server diagnostics

test_get_server_diagnostics_by_non_admin()

Test idempotent id: e84e2234-60d2-42fa-8b30-e2d3049724ac

Test getting server diagnostics by non-admin user is forbidden

Non-admin user cannot view server diagnostics according to policy.

compute.admin.test_server_external_events module

class ServerExternalEventsTest(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test server external events test

test_create_server_external_events()

Test idempotent id: 6bbf4723-61d2-4372-af55-7ba27f1c9ba6

Test create a server and add some external events

compute.admin.test_servers module

class ServersAdmin275Test(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test compute server with microversion greater than 2.75

NOTE(gmann): This test tests the Server APIs response schema # for 2.75 microversion. No specific assert or behaviour verification # is needed.

test_rebuild_update_server_275()

Test idempotent id: bf2b4a00-73a3-4d53-81fa-acbcd97d6339

class ServersAdminTestJSON(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Tests Servers API using admin privileges

test_create_server_with_scheduling_hint()

Test idempotent id: fdcd9b33-0903-4e00-a1f7-b5f6543068d6

Test creating server with scheduling hint

test_inject_network_info()

Test idempotent id: 7a1323b4-a6a2-497a-96cb-76c07b945c71

Test injecting network info of a server

test_list_servers_by_admin()

Test idempotent id: 51717b38-bdc1-458b-b636-1cf82d99f62f

Test listing servers by admin without other projects

Listing servers by admin user returns a list which doesnt contain the other projects server by default.

test_list_servers_by_admin_with_all_tenants()

Test idempotent id: 9f5579ae-19b4-4985-a091-2a5d56106580

Test listing servers by admin with all tenants

Listing servers by admin user with all tenants parameter, all servers should be listed.

test_list_servers_by_admin_with_specified_tenant()

Test idempotent id: 7e5d6b8f-454a-4ba1-8ae2-da857af8338b

Test listing servers by admin with specified project

In nova v2, tenant_id is ignored unless all_tenants is specified.

test_list_servers_detailed_filter_by_invalid_status()

Test idempotent id: d56e9540-73ed-45e0-9b88-98fc419087eb

Test filtering the list of servers by invalid server status

test_list_servers_filter_by_error_status()

Test idempotent id: 06f960bb-15bb-48dc-873d-f96e89be7870

Test filtering the list of servers by server error status

test_list_servers_filter_by_exist_host()

Test idempotent id: 86c7a8f7-50cf-43a9-9bac-5b985317134f

Test filtering the list of servers by existent host

test_rebuild_server_in_error_state()

Test idempotent id: 682cb127-e5bb-4f53-87ce-cb9003604442

Test rebuilding server in error state

The server in error state should be rebuilt using the provided image and changed to ACTIVE state.

test_reset_state_server()

Test idempotent id: ee8ae470-db70-474d-b752-690b7892cab1

Test resetting server state to error/active

compute.admin.test_servers_negative module

class ServersAdminNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Negative Tests of Servers API using admin privileges

test_migrate_non_existent_server()

Test idempotent id: 46a4e1ca-87ae-4d28-987a-1b6b136a0221

Test migrating a non existent server should fail

test_migrate_server_invalid_state()

Test idempotent id: b0b17f83-d14e-4fc4-8f31-bcc9f3cfa629

Test migrating a server with invalid state should fail

test_reset_state_server_invalid_state()

Test idempotent id: b0b4d8af-1256-41ef-9ee7-25f1c19dde80

Test resetting server state to invalid state value should fail

test_reset_state_server_invalid_type()

Test idempotent id: 4cdcc984-fab0-4577-9a9d-6d558527ee9d

Test resetting server state to invalid state type should fail

test_reset_state_server_nonexistent_server()

Test idempotent id: e741298b-8df2-46f0-81cb-8f814ff2504c

Test resetting a non existent servers state should fail

test_resize_server_using_overlimit_ram()

Test idempotent id: 28dcec23-f807-49da-822c-56a92ea3c687

Test resizing server using over limit ram should fail

test_resize_server_using_overlimit_vcpus()

Test idempotent id: 7368a427-2f26-4ad9-9ba9-911a0ec2b0db

Test resizing server using over limit vcpus should fail

test_restore_server_invalid_state()

Test idempotent id: 7fcadfab-bd6a-4753-8db7-4a51e51aade9

Restore-deleting a server not in soft-delete state should fail

We can restore a soft deleted server, but cant restore a server that is not in soft-delete state.

compute.admin.test_servers_on_multinodes module

class ServersOnMultiNodesTest(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Test creating servers on multiple nodes with scheduler_hints.

test_create_server_with_scheduler_hint_group_affinity()

Test idempotent id: 9d2e924a-baf4-11e7-b856-fa163e65f5ce

Tests the ServerGroupAffinityFilter

Creates two servers in an affinity server group and asserts the servers are in the group and on same host.

Uses the /servers multi-create API.

test_create_server_with_scheduler_hint_group_anti_affinity()

Test idempotent id: f8bd0867-e459-45f5-ba53-59134552fe04

Tests the ServerGroupAntiAffinityFilter

Creates two servers in an anti-affinity server group and asserts the servers are in the group and on different hosts.

Uses the /servers multi-create API.

test_create_servers_on_different_hosts()

Test idempotent id: cc7ca884-6e3e-42a3-a92f-c522fcf25e8e

Test creating servers with hints of single different_host

test_create_servers_on_different_hosts_with_list_of_servers()

Test idempotent id: 7869cc84-d661-4e14-9f00-c18cdc89cf57

Test creating servers with hints of a list of different_host

test_create_servers_on_same_host()

Test idempotent id: 26a9d5df-6890-45f2-abc4-a659290cb130

Test creating servers with hints same_host

class UnshelveToHostMultiNodesTest(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Test to unshelve server in between hosts.

test_unshelve_to_specific_host()

Test idempotent id: b5cc0889-50c2-46a0-b8ff-b5fb4c3a6e20

Test unshelve to a specific host, new behavior introduced in microversion 2.91. 1. Shelve offload server. 2. Request unshelve to original host and verify server land on it. 3. Shelve offload server again. 4. Request unshelve to the other host and verify server land on it.

compute.admin.test_services module

class ServicesAdminTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Tests Nova Services API.

List and Enable/Disable require admin privileges.

test_get_service_by_host_name()

Test idempotent id: affb42d5-5b4b-43c8-8b0b-6dca054abcca

Listing nova services by host name

test_get_service_by_service_binary_name()

Test idempotent id: f345b1ec-bc6e-4c38-a527-3ca2bc00bef5

Listing nova services by binary name

test_list_services()

Test idempotent id: 5be41ef4-53d1-41cc-8839-5c2a48a1b283

Listing nova services

compute.admin.test_services_negative module

class ServicesAdminNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Tests Services API. List and Enable/Disable require admin privileges.

test_get_service_by_invalid_params()

Test idempotent id: d0884a69-f693-4e79-a9af-232d15643bf7

Test listing services by invalid filter should return all services

Expect all services to be returned when the request contains invalid parameters.

test_get_service_by_invalid_service_and_valid_host()

Test idempotent id: 1e966d4a-226e-47c7-b601-0b18a27add54

Test listing services by invalid service and valid host value

test_get_service_with_valid_service_and_invalid_host()

Test idempotent id: 64e7e7fb-69e8-4cb6-a71d-8d5eb0c98655

Test listing services by valid service and invalid host value

test_list_services_with_non_admin_user()

Test idempotent id: 1126d1f8-266e-485f-a687-adc547492646

Non admin user is not allowed to list nova services

class ServicesAdminNegativeV253TestJSON(**args*, ***kwargs*)

Bases: *ServicesAdminNegativeTestJSON*

test_disable_log_reason_with_invalid_service_id()

Test idempotent id: f46a9d91-1e85-4b96-8e7a-db7706fa2e9a

Test updating non existing service to disabled with reason

test_disable_service_with_invalid_service_id()

Test idempotent id: a9eeeade-42b3-419f-87aa-c9342aa068cf

Test updating non existing service to status disabled

test_enable_service_with_invalid_service_id()

Test idempotent id: 508671aa-c929-4479-bd10-8680d40dd0a6

Test updating non existing service to status enabled

compute.admin.test_simple_tenant_usage module

class TenantUsagesTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test tenant usages

test_get_usage_tenant()

Test idempotent id: 94135049-a4c5-4934-ad39-08fa7da4f22e

Test getting usage for a specific tenant

test_get_usage_tenant_with_non_admin_user()

Test idempotent id: 9d00a412-b40e-4fd9-8eba-97b496316116

Test getting usage for a specific tenant with non admin user

test_list_usage_all_tenants()

Test idempotent id: 062c8ae9-9912-4249-8b51-e38d664e926e

Test getting usage for all tenants

compute.admin.test_simple_tenant_usage_negative module

class TenantUsagesNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Negative tests of compute tenant usages API

test_get_usage_tenant_with_empty_tenant_id()

Test idempotent id: 8b21e135-d94b-4991-b6e9-87059609c8ed

Test getting tenant usage with empty tenant id should fail

test_get_usage_tenant_with_invalid_date()

Test idempotent id: 4079dd2a-9e8d-479f-869d-6fa985ce45b6

Test getting tenant usage with invalid time range should fail

test_list_usage_all_tenants_with_non_admin_user()

Test idempotent id: bbe6fe2c-15d8-404c-a0a2-44fad0ad5cc7

Test listing usage of all tenants by non-admin user is forbidden

compute.admin.test_spice module

class SpiceDirectConsoleTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test the spice-direct console

test_spice_direct()

Test idempotent id: 80f4460d-1a06-403c-9e93-cf434c70be05

Test accessing spice-direct console of server

compute.admin.test_volume module

class AttachSCSIVolumeTestJSON(**args*, ***kwargs*)

Bases: *BaseAttachSCSIVolumeTest*

Test attaching scsi volume to server

test_attach_scsi_disk_with_config_drive()

Test idempotent id: 777e468f-17ca-4da4-b93d-b7dbf56c0494

Test the attach/detach volume with config drive/scsi disk

Enable the config drive, followed by booting an instance from an image with meta properties hw_cdrom: scsi and use virtio-scsi mode with further asserting list volume attachments in instance after attach and detach of the volume.

class BaseAttachSCSIVolumeTest(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Base class for the admin volume tests in this module.

compute.admin.test_volume_swap module

class TestMultiAttachVolumeSwap(**args*, ***kwargs*)

Bases: *TestVolumeSwapBase*

Test swapping volume attached to multiple servers

Test swapping volume attached to multiple servers with microversion greater than 2.59

test_volume_swap_with_multiattach()

Test idempotent id: e8f8f9d1-d7b7-4cd2-8213-ab85ef697b6e

Test swapping volume attached to multiple servers

The following is the scenario outline:

1. Create a volume volume1 with non-admin.
2. Create a volume volume2 with non-admin.
3. Boot 2 instances server1 and server2 with non-admin.
4. Attach volume1 to server1 with non-admin.
5. Attach volume1 to server2 with non-admin.
6. Swap volume1 to volume2 on server1

7. Check volume1 is attached to server2 and not attached to server1
8. Check volume2 is attached to server1.

class TestVolumeSwap(**args*, ***kwargs*)

Bases: *TestVolumeSwapBase*

The test suite for swapping of volume with admin user

test_volume_swap()

Test idempotent id: 1769f00d-a693-4d67-a631-6a3496773813

Test swapping of volume attached to server with admin user

The following is the scenario outline:

1. Create a volume volume1 with non-admin.
2. Create a volume volume2 with non-admin.
3. Boot an instance instance1 with non-admin.
4. Attach volume1 to instance1 with non-admin.
5. Swap volume from volume1 to volume2 as admin.
6. Check the swap volume is successful and volume2 is attached to instance1 and volume1 is in available state.
7. Swap volume from volume2 to volume1 as admin.
8. Check the swap volume is successful and volume1 is attached to instance1 and volume2 is in available state.

class TestVolumeSwapBase(**args*, ***kwargs*)

Bases: `BaseV2ComputeAdminTest`

compute.admin.test_volumes_negative module

class UpdateMultiattachVolumeNegativeTest(**args*, ***kwargs*)

Bases: `BaseV2ComputeAdminTest`

Negative tests of swapping volume attached to multiple servers

Negative tests of swapping volume attached to multiple servers with compute microversion greater than 2.59 and volume microversion greater than 3.26

test_multiattach_rw_volume_update_failure()

Test idempotent id: 7576d497-b7c6-44bd-9cc5-c5b4e50fec71

Test swapping volume attached to multi-servers with read-write mode

1. Create two volumes vol1 and vol2
2. Create two instances server1 and server2
3. Attach vol1 to both of these instances
4. By default both of these attachments should have an attach_mode of read-write, so trying to swap vol1 to vol2 should fail
5. Check vol1 is still attached to both servers

6. Check vol2 is not attached to any server

class VolumesAdminNegativeTest(*args, **kwargs)

Bases: BaseV2ComputeAdminTest

Negative tests of volume swapping

test_update_attached_volume_with_nonexistent_volume_in_body()

Test idempotent id: 7dcac15a-b107-46d3-a5f6-cb863f4e454a

Test swapping volume to a non existence volume should fail

test_update_attached_volume_with_nonexistent_volume_in_uri()

Test idempotent id: 309b5ecd-0585-4a7e-a36f-d2b2bf55259d

Test swapping non existent volume should fail

Module contents

compute.certificates package

Module contents

compute.flavors package

Submodules

compute.flavors.test_flavors module

class FlavorsV2TestJSON(*args, **kwargs)

Bases: BaseV2ComputeTest

Tests Flavors

test_get_flavor()

Test idempotent id: 1f12046b-753d-40d2-abb6-d8eb8b30cb2f

The expected flavor details should be returned

test_list_flavors()

Test idempotent id: e36c0eaa-dff5-4082-ad1f-3f9a80aa3f59

List of all flavors should contain the expected flavor

test_list_flavors_detailed_filter_by_min_disk()

Test idempotent id: 3df2743e-3034-4e57-a4cb-b6527f6eac79

The detailed list of flavors should be filtered by disk space

test_list_flavors_detailed_filter_by_min_ram()

Test idempotent id: 09fe7509-b4ee-4b34-bf8b-39532dc47292

The detailed list of flavors should be filtered by RAM

test_list_flavors_detailed_limit_results()

Test idempotent id: b26f6327-2886-467a-82be-cef7a27709cb

Only the expected number of flavors(detailed) should be returned

test_list_flavors_detailed_using_marker()

Test idempotent id: 6db2f0c0-ddee-4162-9c84-0703d3dd1107

The list of flavors should start from the provided marker

test_list_flavors_filter_by_min_disk()

Test idempotent id: 10645a4d-96f5-443f-831b-730711e11dd4

The list of flavors should be filtered by disk space

test_list_flavors_filter_by_min_ram()

Test idempotent id: 935cf550-e7c8-4da6-8002-00f92d5edfaa

The list of flavors should be filtered by RAM

test_list_flavors_limit_results()

Test idempotent id: 8d7691b3-6ed4-411a-abc9-2839a765adab

Only the expected number of flavors should be returned

test_list_flavors_using_marker()

Test idempotent id: e800f879-9828-4bd0-8eae-4f17189951fb

The list of flavors should start from the provided marker

test_list_flavors_with_detail()

Test idempotent id: 6e85fde4-b3cd-4137-ab72-ed5f418e8c24

Detailed list of all flavors should contain the expected flavor

compute.flavors.test_flavors_negative module

class FlavorsV2NegativeTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

test_boot_with_low_ram()

Test idempotent id: 90f0d93a-91c1-450c-91e6-07d18172cefe

Try boot a vm with lower than min ram

Create an image with min_ram value Try to create server with flavor of insufficient ram size from that image

Module contents

compute.floating_ips package

Submodules

compute.floating_ips.base module

class BaseFloatingIPsTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

compute.floating_ips.test_floating_ips_actions module

class FloatingIPsAssociationTestJSON(**args*, ***kwargs*)

Bases: BaseFloatingIPsTest

Test floating ips association with microversion less than 2.44

test_associate_already_associated_floating_ip()

Test idempotent id: 6edef4b2-aaf1-4abc-bbe3-993e2561e0fe

Test associating an already associated floating ip

First associate a floating ip to server1, then associate the floating ip to server2, the floating ip will be associated to server2 and no longer associated to server1.

test_associate_disassociate_floating_ip()

Test idempotent id: 307efa27-dc6f-48a0-8cd2-162ce3ef0b52

Test associate/disassociate floating ip to a server

class FloatingIPsTestJSON(**args*, ***kwargs*)

Bases: BaseFloatingIPsTest

Test floating ips API with compute microversion less than 2.36

test_allocate_floating_ip()

Test idempotent id: f7bfb946-297e-41b8-9e8c-aba8e9bb5194

Test allocating a floating ip to a project

test_delete_floating_ip()

Test idempotent id: de45e989-b5ca-4a9b-916b-04a52e7bbb8b

Test deleting a valid floating ip from project

compute.floating_ips.test_floating_ips_actions_negative module

class FloatingIPsAssociationNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseFloatingIPsTest

Test floating ips API with compute microversion less than 2.44

test_associate_ip_to_server_with_floating_ip()

Test idempotent id: 58a80596-ffb2-11e6-9393-fa163e4fa634

Test associating floating ip to server already with floating ip

1. The VM have one port
2. Associate floating IP A to the VM
3. Associate floating IP B which is from same pool with floating IP A to the VM, should raise BadRequest exception

test_associate_ip_to_server_without_passing_floating_ip()

Test idempotent id: 804b4fcb-bbf5-412f-925d-896672b61eb3

Test associating empty floating ip to server should fail

test_associate_nonexistent_floating_ip()

Test idempotent id: 595fa616-1a71-4670-9614-46564ac49a4c

Test associating non existent floating ip to server should fail

test_dissociate_nonexistent_floating_ip()

Test idempotent id: 0a081a66-e568-4e6b-aa62-9587a876dca8

Test dissociating non existent floating ip should fail

class FloatingIPsNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseFloatingIPsTest

Test floating ips API with compute microversion less than 2.36

test_allocate_floating_ip_from_nonexistent_pool()

Test idempotent id: 6e0f059b-e4dd-48fb-8207-06e3bba5b074

Test allocating floating ip from non existent pool should fail

test_delete_nonexistent_floating_ip()

Test idempotent id: ae1c55a8-552b-44d4-bfb6-2a115a15d0ba

Test deleting non existent floating ip should fail

compute.floating_ips.test_list_floating_ips module

class FloatingIPDetailsTestJSON(**args*, ***kwargs*)

Bases: BaseFloatingIPsTest

Test floating ip details with compute microversion less than 2.36

test_get_floating_ip_details()

Test idempotent id: eef497e0-8ff7-43c8-85ef-558440574f84

Test getting floating ip details

test_list_floating_ips()

Test idempotent id: 16db31c3-fb85-40c9-bbe2-8cf7b67ff99f

Test listing floating ips

compute.floating_ips.test_list_floating_ips_negative module

class FloatingIPDetailsNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseFloatingIPsTest

Negative tests of floating ip detail

Negative tests of floating ip detail with compute microversion less than 2.36.

test_get_nonexistent_floating_ip_details()

Test idempotent id: 7ab18834-4a4b-4f28-a2c5-440579866695

Test getting non existent floating ip should fail

Module contents

compute.images package

Submodules

compute.images.test_image_metadata module

class ImagesMetadataTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test image metadata with compute microversion less than 2.39

test_delete_image_metadata_item()

Test idempotent id: a013796c-ba37-4bb5-8602-d944511def14

Test deleting image metadata item

The metadata value/key pair should be deleted from the image.

test_get_image_metadata_item()

Test idempotent id: 4f5db52f-6685-4c75-b848-f4bb363f9aa6

Test getting image metadata item

The value for a specific metadata key should be returned.

test_list_image_metadata()

Test idempotent id: 37ec6edd-cf30-4c53-bd45-ae74db6b0531

Test listing image metadata

All metadata key/value pairs for an image should be returned.

test_set_image_metadata()

Test idempotent id: ece7befc-d3ce-42a4-b4be-c3067a418c29

Test setting image metadata

The metadata for the image should match the new values.

test_set_image_metadata_item()

Test idempotent id: f2de776a-4778-4d90-a5da-aae63aee64ae

Test setting image metadata item

The value provided for the given meta item should be set for the image.

test_update_image_metadata()

Test idempotent id: 7b491c11-a9d5-40fe-a696-7f7e03d3fea2

Test updating image medata

The metadata for the image should match the updated values.

compute.images.test_image_metadata_negative module

class ImagesMetadataNegativeTestJSON(*args, **kwargs)

Bases: BaseV2ComputeTest

Negative tests of image metadata

Negative tests of image metadata with compute microversion less than 2.39.

test_delete_nonexistent_image_metadata_item()

Test idempotent id: 848e157f-6bcf-4b2e-a5dd-5124025a8518

Test deleting metadata item of a non existence image should fail

test_get_nonexistent_image_metadata_item()

Test idempotent id: 41ae052c-6ee6-405c-985e-5712393a620d

Test getting metadata of a non existence image should fail

test_list_nonexistent_image_metadata()

Test idempotent id: 94069db2-792f-4fa8-8bd3-2271a6e0c095

Test listing metadata of a non existence image should fail

test_set_nonexistent_image_metadata()

Test idempotent id: dc64f2ce-77e8-45b0-88c8-e15041d08eaf

Test setting metadata of a non existence image should fail

test_set_nonexistent_image_metadata_item()

Test idempotent id: 2154fd03-ab54-457c-8874-e6e3eb56e9cf

Test setting metadata item of a non existence image should fail

test_update_nonexistent_image_metadata()

Test idempotent id: a403ef9e-9f95-427c-b70a-3ce3388796f1

Test updating metadata of a non existence image should fail

compute.images.test_images module

class ImagesTestJSON(*args, **kwargs)

Bases: BaseV2ComputeTest

Test server images

test_create_image_from_paused_server()

Test idempotent id: 71bcb732-0261-11e7-9086-fa163e4fa634

Test creating server image from paused server

test_create_image_from_stopped_server()

Test idempotent id: aaacd1d0-55a2-4ce8-818a-b5439df8adc9

Test creating server image from stopped server

test_create_image_from_suspended_server()

Test idempotent id: 8ca07fec-0262-11e7-907e-fa163e4fa634

Test creating server image from suspended server

test_create_server_from_snapshot()

Test idempotent id: f3cac456-e3fe-4183-a7a7-a59f7f017088

test_delete_saving_image()

Test idempotent id: aa06b52b-2db5-4807-b218-9441f75d74e3

Test deleting server image while it is in SAVING state

compute.images.test_images_negative module

class ImagesDeleteNegativeTestJSON(**args*, ***kwargs*)

Bases: *ImagesNegativeTestBase*

Negative tests of server image

Negative tests of server image with compute microversion less than 2.36.

test_delete_image_blank_id()

Test idempotent id: e6e41425-af5c-4fe6-a4b5-7b7b963ffda5

Check trying to delete an image with blank id should fail

test_delete_image_negative_image_id()

Test idempotent id: 68e2c175-bd26-4407-ac0f-4ea9ce2139ea

Check trying to delete an image with negative id should fail

test_delete_image_non_hex_string_id()

Test idempotent id: 924540c3-f1f1-444c-8f58-718958b6724e

Check trying to delete an image with non hex id should fail

test_delete_image_with_id_over_character_limit()

Test idempotent id: b340030d-82cd-4066-a314-c72fb7c59277

Check trying to delete image with id over limit should fail

test_delete_image_with_invalid_image_id()

Test idempotent id: 381acb65-785a-4942-94ce-d8f8c84f1f0f

Check an image should not be deleted with invalid image id

test_delete_non_existent_image()

Test idempotent id: 137aef61-39f7-44a1-8ddf-0adf82511701

Check trying to delete a non-existent image should fail

class ImagesNegativeTestBase(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

class ImagesNegativeTestJSON(**args*, ***kwargs*)

Bases: *ImagesNegativeTestBase*

Negative tests of server image

test_create_image_from_deleted_server()

Test idempotent id: 6cd5a89d-5b47-46a7-93bc-3916f0d84973

Check server image should not be created if the server is removed

test_create_image_from_invalid_server()

Test idempotent id: 82c5b0c4-9dbd-463c-872b-20c4755aae7f

Check server image should not be created with invalid server id

test_create_image_specify_uuid_35_characters_or_less()

Test idempotent id: ec176029-73dc-4037-8d72-2e4ff60cf538

Check server image should not be created for invalid server id

Return an error if server id passed is 35 characters or less

test_create_image_specify_uuid_37_characters_or_more()

Test idempotent id: 36741560-510e-4cc2-8641-55fe4dfb2437

Check server image should not be created for invalid server id

Return an error if sever id passed is 37 characters or more

compute.images.test_images_oneserver module

class ImagesOneServerTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test server images API

test_create_delete_image()

Test idempotent id: 3731d080-d4c5-4872-b41a-64d0d0021314

Test create/delete server image

test_create_image_specify_multibyte_character_image_name()

Test idempotent id: 3b7c6fe4-dfe7-477c-9243-b06359db51e6

Test creating server image with multibyte character image name

compute.images.test_images_oneserver_negative module

class ImagesOneServerNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Negative tests of server images

test_create_image_specify_invalid_metadata()

Test idempotent id: 55d1d38c-dd66-4933-9c8e-7d92aeb60ddc

Test creating server image with invalid metadata should fail

test_create_image_specify_metadata_over_limits()

Test idempotent id: 3d24d11f-5366-4536-bd28-cff32b748eca

Test creating server image with metadata over 255 should fail

test_create_image_specify_name_over_character_limit()

Test idempotent id: 084f0cbc-500a-4963-8a4e-312905862581

Test creating server image with image name over 255 should fail

test_create_second_image_when_first_image_is_being_saved()

Test idempotent id: 0460efcf-ee88-4f94-acef-1bf658695456

Test creating another server image when first image is being saved

Creating another server image when first image is being saved is not allowed.

test_delete_image_that_is_not_yet_active()

Test idempotent id: 0894954d-2db2-4195-a45b-ffec0bc0187e

Test deleting a non-active server image should fail

compute.images.test_list_image_filters module

class ListImageFiltersTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test listing server images with compute microversion less than 2.36

test_list_images_filter_by_changes_since()

Test idempotent id: 18bac3ae-da27-436c-92a9-b22474d13aab

Test listing images filtered by changes-since

The list of images should contain only images updated since the provided changes-since value.

test_list_images_filter_by_name()

Test idempotent id: 33163b73-79f5-4d07-a7ea-9213bcc468ff

Test listing server images filtered by image name

The list of images should contain only images with the provided image name.

test_list_images_filter_by_server_id()

Test idempotent id: 9f238683-c763-45aa-b848-232ec3ce3105

Test listing images filtered by server id

The list of images should contain only images with the provided server id.

test_list_images_filter_by_server_ref()

Test idempotent id: 05a377b8-28cf-4734-a1e6-2ab5c38bf606

Test listing images filtered by server link href

The list of images should contain only images with the provided server link href.

test_list_images_filter_by_status()

Test idempotent id: a3f5b513-aeb3-42a9-b18e-f091ef73254d

Test listing server images filtered by image status

The list of images should contain only images with the provided image status.

test_list_images_filter_by_type()

Test idempotent id: e3356918-4d3e-4756-81d5-abc4524ba29f

Test listing images filtered by image type

The list of images should contain only images with the provided image type.

test_list_images_limit_results()

Test idempotent id: 3a484ca9-67ba-451e-b494-7fcf28d32d62

Test listing images with limited count

If we use limit=1 when listing images, then only 1 image should be returned.

test_list_images_with_detail_filter_by_changes_since()

Test idempotent id: 7d439e18-ac2e-4827-b049-7e18004712c4

Test listing images details filtered by changes-since

The list of images should contain only images updated since the provided changes-since value.

test_list_images_with_detail_filter_by_name()

Test idempotent id: 644ea267-9bd9-4f3b-af9f-dffa02396a17

Test listing server images details filtered by image name

The list of images should contain only images with the provided image name.

test_list_images_with_detail_filter_by_server_ref()

Test idempotent id: 8c78f822-203b-4bf6-8bba-56ebd551cf84

Test listing images details filtered by server link href

The list of images should contain only images with the provided server link href.

test_list_images_with_detail_filter_by_status()

Test idempotent id: 9b0ea018-6185-4f71-948a-a123a107988e

Test listing server images details filtered by image status

The list of images should contain only images with the provided image status.

test_list_images_with_detail_filter_by_type()

Test idempotent id: 888c0cc0-7223-43c5-9db0-b125fd0a393b

Test listing images details filtered by image type

The list of images should contain only images with the provided image type.

test_list_images_with_detail_limit_results()

Test idempotent id: ba2fa9a9-b672-47cc-b354-3b4c0600e2cb

Test listing images details with limited count

If we use limit=1 when listing images with full details, then only 1 image should be returned.

compute.images.test_list_image_filters_negative module

class ListImageFiltersNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Negative tests of listing images using compute images API

Negative tests of listing images using compute images API with microversion less than 2.36.

test_get_nonexistent_image()

Test idempotent id: 391b0440-432c-4d4b-b5da-c5096aa247eb

Test getting a non existent image should fail

compute.images.test_list_images module

class ListImagesTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test listing server images with compute microversion less than 2.36

test_get_image()

Test idempotent id: 490d0898-e12a-463f-aef0-c50156b9f789

Test getting the correct details for a single server image

test_list_images()

Test idempotent id: fd51b7f4-d4a3-4331-9885-866658112a6f

Test listing server images

The list of all images should contain the image

test_list_images_with_detail()

Test idempotent id: 9f94cb6b-7f10-48c5-b911-a0b84d7d4cd6

Test listing server images with detail

Detailed list of all images should contain the expected images

Module contents

compute.keypairs package

Submodules

compute.keypairs.base module

class BaseKeypairTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Base test case class for all keypair API tests.

compute.keypairs.test_keypairs module

class KeyPairsV2TestJSON(**args*, ***kwargs*)

Bases: BaseKeypairTest

Test keypairs API with compute microversion less than 2.2

test_get_keypair_detail()

Test idempotent id: a4233d5d-52d8-47cc-9a25-e1864527e3df

Test getting keypair detail by keypair name

test_keypair_create_delete()

Test idempotent id: 6c1d3123-4519-4742-9194-622cb1714b7d

Test create/delete keypair

test_keypair_create_with_pub_key()

Test idempotent id: 39c90c6a-304a-49dd-95ec-2366129def05

Test creating keypair with a given public key

test_keypairs_create_list_delete()

Test idempotent id: 1d1dbedb-d7a0-432a-9d09-83f543c3c19b

Test create/list/delete keypairs

Keypairs created should be available in the response list

compute.keypairs.test_keypairs_negative module

class KeyPairsNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseKeypairTest

Negative tests of keypairs API

test_create_keypair_invalid_name()

Test idempotent id: 45fbe5e0-acb5-49aa-837a-ff8d0719db91

Test keypairs with an invalid name should not be created

test_create_keypair_when_public_key_bits_exceeds_maximum()

Test idempotent id: fc100c19-2926-4b9c-8fdc-d0589ee2f9ff

Test keypair should not be created when public key are too long

test_create_keypair_with_duplicate_name()

Test idempotent id: 0359a7f1-f002-4682-8073-0c91e4011b7c

Test keypairs with duplicate names should not be created

test_create_keypair_with_empty_name_string()

Test idempotent id: 1398abe1-4a84-45fb-9294-89f514daff00

Test keypairs with empty name should not be created

test_create_keypair_with_empty_public_key()

Test idempotent id: dade320e-69ca-42a9-ba4a-345300f127e0

Test keypair should not be created with an empty public key

test_create_keypair_with_long_keynames()

Test idempotent id: 3faa916f-779f-4103-aca7-dc3538eee1b7

Test keypairs with name longer than 255 should not be created

test_keypair_create_with_invalid_pub_key()

Test idempotent id: 29cca892-46ae-4d48-bc32-8fe7e731eb81

Test keypair should not be created with a non RSA public key

test_keypair_delete_nonexistent_key()

Test idempotent id: 7cc32e47-4c42-489d-9623-c5e2cb5a2fa5

Test non-existent key deletion should throw a proper error

compute.keypairs.test_keypairs_v22 module

class KeyPairsV22TestJSON(**args*, ***kwargs*)

Bases: KeyPairsV2TestJSON

Test keypairs API with compute microversion greater than 2.1

test_keypairsv22_create_list_show()

Test idempotent id: 8726fa85-7f98-4b20-af9e-f710a4f3391c

Test create/list/show keypair

test_keypairsv22_create_list_show_with_type()

Test idempotent id: 89d59d43-f735-441a-abcf-0601727f47b6

Test create/list/show keypair with keypair type

Module contents

compute.limits package

Submodules

compute.limits.test_absolute_limits module

class AbsoluteLimitsTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test compute absolute limits

Test compute absolute limits with compute microversion less than 2.57

test_absLimits_get()

Test idempotent id: b54c66af-6ab6-4cf0-a9e5-a0cb58d75e0b

Test getting nova absolute limits

class AbsoluteLimitsV257TestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test compute absolute limits

Test compute absolute limits with compute microversion greater than 2.56

compute.limits.test_absolute_limits_negative module

class AbsoluteLimitsNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Negative tests of nova absolute limits

test_max_metadata_exceed_limit()

Test idempotent id: 215cd465-d8ae-49c9-bf33-9c911913a5c8

Test creating server with metadata over limit should fail

We should not create server with metadata over maxServerMeta limit

Module contents

compute.security_groups package

Submodules

compute.security_groups.base module

class BaseSecurityGroupsTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

compute.security_groups.test_security_group_rules module

class SecurityGroupRulesTestJSON(**args*, ***kwargs*)

Bases: BaseSecurityGroupsTest

Test security group rules API

Test security group rules API with compute microversion less than 2.36.

test_security_group_rules_create()

Test idempotent id: 850795d7-d4d3-4e55-b527-a774c0123d3a

Test creating security group rules

test_security_group_rules_create_with_optional_cidr()

Test idempotent id: 7a01873e-3c38-4f30-80be-31a043cfe2fd

Test creating security group rules with optional field cidr

test_security_group_rules_create_with_optional_group_id()

Test idempotent id: 7f5d2899-7705-4d4b-8458-4505188ffab6

Test creating security group rules with optional field group id

test_security_group_rules_delete_when_peer_group_deleted()

Test idempotent id: fc5c5acf-2091-43a6-a6ae-e42760e9ffaf

Test security group rule gets deleted when peer group is deleted

test_security_group_rules_list()

Test idempotent id: a6154130-5a55-4850-8be4-5e9e796dbf17

Test listing security group rules

compute.security_groups.test_security_group_rules_negative module

class SecurityGroupRulesNegativeTestJSON(*args, **kwargs)

Bases: BaseSecurityGroupsTest

Negative tests of security group rules API

Negative tests of security group rules API with compute microversion less than 2.36.

test_create_security_group_rule_duplicate()

Test idempotent id: 8bd56d02-3ffa-4d67-9933-b6b9a01d6089

Test creating duplicate security group rule should fail

test_create_security_group_rule_with_invalid_from_port()

Test idempotent id: 12bbc875-1045-4f7a-be46-751277baedb9

Test creating security group rule with invalid from_port

Negative test: Creation of security group rule should fail with invalid from_port.

test_create_security_group_rule_with_invalid_id()

Test idempotent id: 2244d7e4-adb7-4ecb-9930-2d77e123ce4f

Test creating security group rule with invalid parent group id

Negative test: Creation of security group rule should fail with parent group id which is not integer.

test_create_security_group_rule_with_invalid_ip_protocol()

Test idempotent id: 84c81249-9f6e-439c-9bbf-cbb0d2cddbdf

Test creating security group rule with invalid ip protocol

Negative test: Creation of security group rule should fail with invalid ip_protocol.

test_create_security_group_rule_with_invalid_port_range()

Test idempotent id: 00296fa9-0576-496a-ae15-fbab843189e0

Test creating security group rule with invalid port range

Negative test: Creation of security group rule should fail with invalid port range.

test_create_security_group_rule_with_invalid_to_port()

Test idempotent id: ff88804d-144f-45d1-bf59-dd155838a43a

Test creating security group rule with invalid to_port

Negative test: Creation of security group rule should fail with invalid to_port.

test_create_security_group_rule_with_non_existent_id()

Test idempotent id: 1d507e98-7951-469b-82c3-23f1e6b8c254

Test creating security group rule with non existent parent group

Negative test: Creation of security group rule should fail with non existent parent group id.

test_delete_security_group_rule_with_non_existent_id()

Test idempotent id: 56fddcca-dbb8-4494-a0db-96e9f869527c

Test deleting non existent security group rule should fail

compute.security_groups.test_security_groups module

class SecurityGroupsTestJSON(*args, **kwargs)

Bases: BaseSecurityGroupsTest

Test security groups API with compute microversion less than 2.36

test_list_security_groups_by_server()

Test idempotent id: 79517d60-535a-438f-af3d-e6feab1cbea7

Test listing security groups by server

Create security groups and add them to a server, then list security groups by server, the added security groups should be in the list.

test_security_group_create_get_delete()

Test idempotent id: ecc0da4a-2117-48af-91af-993cca39a615

Test create/get/delete security group

Security group should be created, fetched and deleted with char space between name along with leading and trailing spaces.

test_security_groups_create_list_delete()

Test idempotent id: eb2b087d-633d-4d0d-a7bd-9e6ba35b32de

Test create/list/delete security groups

Positive test: Should return the list of security groups.

test_server_security_groups()

Test idempotent id: fe4abc0d-83f5-4c50-ad11-57a1127297a2

Test adding security groups to a server

Checks that security groups may be added and linked to a server and not deleted if the server is active.

test_update_security_groups()

Test idempotent id: 7d4e1d3c-3209-4d6d-b020-986304ebad1f

Test updating security group name and description

compute.security_groups.test_security_groups_negative module

class SecurityGroupsNegativeTestJSON(*args, **kwargs)

Bases: BaseSecurityGroupsTest

Negative tests of security groups API

Negative tests of security groups API with compute microversion less than 2.36.

test_delete_nonexistent_security_group()

Test idempotent id: 6727c00b-214c-4f9e-9a52-017ac3e98411

Test deleting non existent security group should fail

test_delete_security_group_without_passing_id()

Test idempotent id: 1438f330-8fa4-4aeb-8a94-37c250106d7f

Test deleting security group passing empty group id should fail

test_delete_the_default_security_group()

Test idempotent id: 36a1629f-c6da-4a26-b8b8-55e7e5d5cd58

Test deleting default security group should fail

test_security_group_create_with_duplicate_name()

Test idempotent id: 9fdb4abc-6b66-4b27-b89c-eb215a956168

Test creating security group with duplicate name should fail

test_security_group_create_with_invalid_group_description()

Test idempotent id: 777b6f14-aca9-4758-9e84-38783cfa58bc

Test creating security group with invalid group description

Negative test: Security group should not be created with description longer than 255 chars. Empty description is allowed by the API reference, however.

test_security_group_create_with_invalid_group_name()

Test idempotent id: 1759c3cb-b0fc-44b7-86ce-c99236be911d

Test creating security group with invalid group name should fail

Negative test: Security group should not be created with group name as an empty string, or group name with white spaces, or group name with chars more than 255.

test_security_group_get_nonexistent_group()

Test idempotent id: 673eaec1-9b3e-48ed-bdf1-2786c1b9661c

Test getting non existent security group details should fail

test_update_non_existent_security_group()

Test idempotent id: 27edee9c-873d-4da6-a68a-3c256efebe8f

Test updating a non existent security group should fail

test_update_security_group_with_invalid_sg_des()

Test idempotent id: 97d12b1c-a610-4194-93f1-ba859e718b45

Test updating security group to invalid description should fail

test_update_security_group_with_invalid_sg_id()

Test idempotent id: 00579617-fe04-4e1c-9d08-ca7467d2e34b

Test updating security group with invalid group id should fail

test_update_security_group_with_invalid_sg_name()

Test idempotent id: cda8d8b4-59f8-4087-821d-20cf5a03b3b1

Test updating security group to invalid group name should fail

Module contents

compute.servers package

Submodules

compute.servers.test_attach_interfaces module

class AttachInterfacesTestBase(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

class AttachInterfacesTestJSON(**args*, ***kwargs*)

Bases: *AttachInterfacesTestBase*

Test attaching interfaces

test_create_list_show_delete_interfaces_by_fixed_ip()

Test idempotent id: d290c06c-f5b3-11e7-8ec8-002293781009

Test create/list/show/delete interfaces by fixed ip

test_create_list_show_delete_interfaces_by_network_port()

Test idempotent id: 73fe8f02-590d-4bf1-b184-e9ca81065051

Test create/list/show/delete interfaces by network port

test_reassign_port_between_servers()

Test idempotent id: 2f3a0127-95c7-4977-92d2-bc5aec602fb4

Tests reassigning port between servers

1. Create two servers in Nova.
2. Create a port in Neutron.
3. Attach the port to the first server.
4. Detach the port from the first server.
5. Attach the port to the second server.
6. Detach the port from the second server.

class AttachInterfacesUnderV243Test(**args*, ***kwargs*)

Bases: *AttachInterfacesTestBase*

Test attaching interfaces with compute microversion less than 2.44

test_add_remove_fixed_ip()

Test idempotent id: c7e0e60b-ee45-43d0-abeb-8596fd42a2f9

Test adding and removing fixed ip from server

class AttachInterfacesV270Test(**args*, ***kwargs*)

Bases: *AttachInterfacesTestBase*

Test interface API with microversion greater than 2.69

test_create_get_list_interfaces()

Test idempotent id: 2853f095-8277-4067-92bd-9f10bd4f8e0c

Test interface API with microversion greater than 2.69

Checking create, get, list interface APIs response schema.

compute.servers.test_availability_zone module

class AZV2TestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Tests Availability Zone API List

test_get_availability_zone_list_with_non_admin_user()

Test idempotent id: a8333aa2-205c-449f-a828-d38c2489bf25

List of availability zone with non-administrator user

compute.servers.test_create_server module

class ServersTestBootFromVolume(**args*, ***kwargs*)

Bases: *ServersTestJSON*

Test creating server and verifying the server attributes

This is to create server booted from volume and with disk_config AUTO

class ServersTestFqdnHostnames(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test creating server with FQDN hostname and verifying attributes

Starting Wallaby release, Nova sanitizes freeform characters in server hostname with dashes. This test verifies the same.

test_create_server_with_fqdn_name()

Test idempotent id: 622066d2-39fc-4c09-9eeb-35903c114a0a

Test to create an instance with FQDN type name scheme

class ServersTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test creating server and verifying the server attributes

This is to create server booted from image and with disk_config AUTO

test_host_name_is_same_as_server_name()

Test idempotent id: ac1ad47f-984b-4441-9274-c9079b7a0666

Verify the instance host name is the same as the server name

test_list_servers()

Test idempotent id: 9a438d88-10c6-4bcd-8b5b-5b6e25e1346f

The created server should be in the list of all servers

test_list_servers_with_detail()

Test idempotent id: 585e934c-448e-43c4-acbf-d06a9b899997

The created server should be in the detailed list of all servers

test_verify_created_server_vcpus()

Test idempotent id: cbc0f52f-05aa-492b-bdc1-84b575ca294b

The created server should have the same specification as the flavor

Verify that the number of vcpus reported by the instance matches the amount stated by the flavor

test_verify_server_details()

Test idempotent id: 5de47127-9977-400a-936f-abcfbec1218f

Verify the specified server attributes are set correctly

class ServersTestManualDisk(**args*, ***kwargs*)

Bases: *ServersTestJSON*

Test creating server and verifying the server attributes

This is to create server booted from image and with disk_config MANUAL

class ServersV294TestFqdnHostnames(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test creating server with FQDN hostname and verifying attributes

Starting Antelope release, Nova allows to set hostname as an FQDN type and allows free form characters in hostname using hostname parameter with API above 2.94 .

This is to create server with hostname having FQDN type value having more than 64 characters

test_verify_hostname_allows_fqdn()

Test idempotent id: e7b05488-f9d5-4fce-91b3-e82216c52017

Test to verify hostname allows FQDN type name scheme

Verify the hostname has FQDN value and Freeform characters in the hostname are allowed

compute.servers.test_create_server_multi_nic module

class ServersTestMultiNic(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test multiple networks in servers

test_verify_duplicate_network_nics()

Test idempotent id: 1678d144-ed74-43f8-8e57-ab10dbf9b3c2

Test multiple duplicate networks can be used to create server

Creating server with networks [net1, net2, net1], the server can be created successfully and all three networks are in the server addresses.

test_verify_multiple_nics_order()

Test idempotent id: 0578d144-ed74-43f8-8e57-ab10dbf9b3c2

Test verifying multiple networks order in server

The networks order given at the server creation is preserved within the server.

get_subnets(*count=2*)

Returns a list of requested subnets from project_network_cidr block.

Args:
count (int): Number of blocks required.

Returns:

CIDRs as a list of strings
e.g. [19.80.0.0/24, 19.86.0.0/24]

compute.servers.test_delete_server module

class DeleteServersTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test deleting servers in various states

test_delete_active_server()

Test idempotent id: 925fdfb4-5b13-47ea-ac8a-c36ae6fddb05

Test deleting a server while its VM state is Active

test_delete_server_while_in_attached_volume()

Test idempotent id: d0f3f0d6-d9b6-4a32-8da4-23015dcab23c

Test deleting a server while a volume is attached to it

test_delete_server_while_in_building_state()

Test idempotent id: 9e6e0c87-3352-42f7-9faf-5d6210dbd159

Test deleting a server while its VM state is Building

test_delete_server_while_in_pause_state()

Test idempotent id: 943bd6e8-4d7a-4904-be83-7a6cc2d4213b

Test deleting a server while its VM state is Pause

test_delete_server_while_in_shelved_state()

Test idempotent id: bb0cb402-09dd-4947-b6e5-5e7e1cfa61ad

Test deleting a server while its VM state is Shelved

test_delete_server_while_in_shutoff_state()

Test idempotent id: 546d368c-bb6c-4645-979a-83ed16f3a6be

Test deleting a server while its VM state is Shutoff

test_delete_server_while_in_suspended_state()

Test idempotent id: 1f82ebd3-8253-4f4e-b93f-de9b7df56d8b

Test deleting a server while its VM state is Suspended

test_delete_server_while_in_verify_resize_state()

Test idempotent id: ab0c38b4-cdd8-49d3-9b92-0cb898723c01

Test deleting a server while its VM state is VERIFY_RESIZE

compute.servers.test_device_tagging module

class DeviceTaggingBase(*args, **kwargs)

Bases: BaseV2ComputeTest

class TaggedAttachmentsTest(*args, **kwargs)

Bases: *DeviceTaggingBase*

Test tagged attachments with compute microversion greater than 2.48

test_tagged_attachment()

Test idempotent id: 3e41c782-2a89-4922-a9d2-9a188c4e7c7c

Test tagged attachment

1. Create network
2. Create subnet
3. Create volume
4. Create server
5. Attach tagged networks and volume
6. Verify tagged devices are in server via metadata service
7. Detach tagged networks and volume

class TaggedBootDevicesTest(*args, **kwargs)

Bases: *DeviceTaggingBase*

Test tagged boot device with compute microversion equals 2.32

test_tagged_boot_devices()

Test idempotent id: a2e65a6c-66f1-4442-aaa8-498c31778d96

Test tagged boot devices

1. Create volumes
2. Create networks
3. Create subnets
4. Create ports
5. Create server, specifying tags for items in networks and block_device_mapping_v2,
6. Verify tagged devices are in server via metadata service

class TaggedBootDevicesTest_v242(*args, **kwargs)

Bases: *TaggedBootDevicesTest*

Test tagged boot devices with compute microversion greater than 2.41

compute.servers.test_disk_config module

class ServerDiskConfigTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test disk config option of server

test_rebuild_server_with_auto_disk_config()

Test idempotent id: 9c9fae77-4feb-402f-8450-bf1c8b609713

A server should be rebuilt using the auto disk config option

test_rebuild_server_with_manual_disk_config()

Test idempotent id: bef56b09-2e8c-4883-a370-4950812f430e

A server should be rebuilt using the manual disk config option

test_resize_server_from_auto_to_manual()

Test idempotent id: 693d16f3-556c-489a-8bac-3d0ca2490bad

A server should be resized from auto to manual disk config

test_resize_server_from_manual_to_auto()

Test idempotent id: 414e7e93-45b5-44bc-8e03-55159c6bfc97

A server should be resized from manual to auto disk config

test_update_server_from_auto_to_manual()

Test idempotent id: 5ef18867-358d-4de9-b3c9-94d4ba35742f

A server should be updated from auto to manual disk config

compute.servers.test_instance_actions module

class InstanceActionsTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test instance actions API

test_get_instance_action()

Test idempotent id: aacc71ca-1d70-4aa5-bbf6-0ff71470e43c

Test getting the action details of the provided server

test_list_instance_actions()

Test idempotent id: 77ca5cc5-9990-45e0-ab98-1de8fead201a

Test listing actions of the provided server

class InstanceActionsV221TestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test instance actions with compute microversion greater than 2.20

test_get_list_deleted_instance_actions()

Test idempotent id: 0a0f85d4-10fa-41f6-bf80-a54fb4aa2ae1

Test listing actions for deleted instance

Listing actions for deleted instance should succeed and the returned actions should contain create and delete.

compute.servers.test_instance_actions_negative module

class InstanceActionsNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Negative tests of instance actions

test_get_instance_action_invalid_request()

Test idempotent id: 0269f40a-6f18-456c-b336-c03623c897f1

Test getting instance action with invalid request_id should fail

test_list_instance_actions_non_existent_server()

Test idempotent id: 67e1fce6-7ec2-45c6-92d4-0a8f1a632910

Test listing actions for non existent instance should fail

compute.servers.test_list_server_filters module

class ListServerFiltersTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test listing servers filtered by specified attribute

test_list_servers_detailed_filter_by_flavor()

Test idempotent id: 80c574cc-0925-44ba-8602-299028357dd9

Filter the detailed list of servers by flavor

test_list_servers_detailed_filter_by_image()

Test idempotent id: b3304c3b-97df-46d2-8cd3-e2b6659724e7

Filter the detailed list of servers by image

test_list_servers_detailed_filter_by_server_name()

Test idempotent id: f9eb2b70-735f-416c-b260-9914ac6181e4

Filter the detailed list of servers by server name

test_list_servers_detailed_filter_by_server_status()

Test idempotent id: de2612ab-b7dd-4044-b0b1-d2539601911f

Filter the detailed list of servers by server status

test_list_servers_detailed_limit_results()

Test idempotent id: 67aec2d0-35fe-4503-9f92-f13272b867ed

Filter the detailed list of servers by limit 1

Verify only the expected number of servers are returned (one server)

test_list_servers_filter_by_active_status()

Test idempotent id: ca78e20e-fddb-4ce6-b7f7-bcbf8605e66e

Filter the list of servers by server active status

test_list_servers_filter_by_exceed_limit()

Test idempotent id: 37791bbd-90c0-4de0-831e-5f38cba9c6b3

Filter the list of servers by exceeded limit

Verify only the expected number of servers are returned (all servers)

test_list_servers_filter_by_flavor()

Test idempotent id: 573637f5-7325-47bb-9144-3476d0416908

Filter the list of servers by flavor

test_list_servers_filter_by_image()

Test idempotent id: 05e8a8e7-9659-459a-989d-92c2f501f4ba

Filter the list of servers by image

test_list_servers_filter_by_limit()

Test idempotent id: 614cdfc1-d557-4bac-915b-3e67b48eee76

Filter the list of servers by limit 1

Verify only the expected number of servers are returned (one server)

test_list_servers_filter_by_server_name()

Test idempotent id: 9b067a7b-7fee-4f6a-b29c-be43fe18fc5a

Filter the list of servers by server name

test_list_servers_filter_by_shutoff_status()

Test idempotent id: 451dbbb2-f330-4a9f-b0e1-5f5d2cb0f34c

Filter the list of servers by server shutoff status

test_list_servers_filter_by_zero_limit()

Test idempotent id: b1495414-2d93-414c-8019-849afe8d319e

Filter the list of servers by limit 0

Verify only the expected number of servers are returned (no server)

test_list_servers_filtered_by_ip()

Test idempotent id: 43a1242e-7b31-48d1-88f2-3f72aa9f2077

Filter the list of servers by server ip address

test_list_servers_filtered_by_ip_regex()

Test idempotent id: a905e287-c35e-42f2-b132-d02b09f3654a

Filter the list of servers by part of server ip address

test_list_servers_filtered_by_name_regex()

Test idempotent id: 24a89b0c-0d55-4a28-847f-45075f19b27b

Filter the list of servers by server name regular expression

test_list_servers_filtered_by_name_wildcard()

Test idempotent id: e9f624ee-92af-4562-8bec-437945a18dcb

Filter the list of servers by part of server name

compute.servers.test_list_servers_negative module

class ListServersNegativeTestJSON(*args, **kwargs)

Bases: BaseV2ComputeTest

Negative tests of listing servers

test_list_servers_by_changes_since_future_date()

Test idempotent id: 74745ad8-b346-45b5-b9b8-509d7447fc1f

Test listing servers by a future changes-since date

Return an empty list when a date in the future is passed as changes-since value.

test_list_servers_by_changes_since_invalid_date()

Test idempotent id: 87d12517-e20a-4c9c-97b6-dd1628d6d6c9

Test listing servers by invalid changes-since format should fail

test_list_servers_by_limits_greater_than_actual_count()

Test idempotent id: d47c17fb-eebd-4287-8e95-f20a7e627b18

Test listing servers by limit greater than actual count

Listing servers by limit greater than actual count should return all servers.

test_list_servers_by_limits_pass_negative_value()

Test idempotent id: 62610dd9-4713-4ee0-8beb-fd2c1aa7f950

Test listing servers by negative limit should fail

test_list_servers_by_limits_pass_string()

Test idempotent id: 679bc053-5e70-4514-9800-3dfab1a380a6

Test listing servers by non-integer limit should fail

test_list_servers_by_non_existing_flavor()

Test idempotent id: 5913660b-223b-44d4-a651-a0fbfd44ca75

Test listing servers by non existing flavor returns empty list

test_list_servers_by_non_existing_image()

Test idempotent id: ff01387d-c7ad-47b4-ae9e-64fa214638fe

Test listing servers for a non existing image returns empty list

test_list_servers_by_non_existing_server_name()

Test idempotent id: e2c77c4a-000a-4af3-a0bd-629a328bde7c

Test listing servers for a non existent server name

Listing servers for a non existent server name should return empty list.

test_list_servers_detail_server_is_deleted()

Test idempotent id: 93055106-2d34-46fe-af68-d9ddbf7ee570

Test listing servers detail should not contain deleted server

test_list_servers_status_non_existing()

Test idempotent id: fcdf192d-0f74-4d89-911f-1ec002b822c4

Test listing servers with non existing status

When invalid status is specified, up to microversion 2.37, an empty list is returned, and starting from microversion 2.38, a 400 error is returned in that case.

test_list_servers_with_a_deleted_server()

Test idempotent id: 24a26f1a-1ddc-4eea-b0d7-a90cc874ad8f

Test that deleted servers do not show by default in list servers

compute.servers.test_multiple_create module

class MultipleCreateTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test creating multiple servers in one request

test_multiple_create()

Test idempotent id: 61e03386-89c3-449c-9bb1-a06f423fd9d1

Test creating multiple servers in one request

Creating server with min_count=2, 2 servers will be created.

test_multiple_create_with_reservation_return()

Test idempotent id: 864777fb-2f1e-44e3-b5b9-3eb6fa84f2f7

Test creating multiple servers with return_reservation_id=True

Creating multiple servers with return_reservation_id=True, reservation_id will be returned.

compute.servers.test_multiple_create_negative module

class MultipleCreateNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Negative tests of creating multiple servers in one request

test_max_count_less_than_min_count()

Test idempotent id: 476da616-f1ef-4271-a9b1-b9fc87727cdf

Test creating server with max_count < min_count should fail

test_max_count_less_than_one()

Test idempotent id: a6f9c2ab-e060-4b82-b23c-4532cb9390ff

Test creating server with max_count < 1 should fail

test_max_count_non_integer()

Test idempotent id: 9c5698d1-d7af-4c80-b971-9d403135eea2

Test creating server with non-integer max_count should fail

test_min_count_less_than_one()

Test idempotent id: daf29d8d-e928-4a01-9a8c-b129603f3fc0

Test creating server with min_count=0 should fail

test_min_count_non_integer()

Test idempotent id: 999aa722-d624-4423-b813-0d1ac9884d7a

Test creating server with non-integer min_count should fail

compute.servers.test_novnc module

class NoVNCConsoleTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test novnc console

test_novnc()

Test idempotent id: c640fdff-8ab4-45a4-a5d8-7e6146cbd0dc

Test accessing novnc console of server

test_novnc_bad_token()

Test idempotent id: f9c79937-addc-4aaa-9e0e-841eef02aeb7

Test accessing novnc console with bad token

Do the WebSockify HTTP Request to novnc proxy with a bad token, the novnc proxy should reject the connection and closed it.

compute.servers.test_server_actions module

class ServerActionsBase(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test server actions

class ServerActionsTestJSON(**args*, ***kwargs*)

Bases: *ServerActionsBase*

test_change_server_password()

Test idempotent id: 6158df09-4b82-4ab3-af6d-29cf36af858d

Test changing servers password

The servers password should be set to the provided password and the user can authenticate with the new password.

test_get_console_output()

Test idempotent id: 4b8867e6-fffa-4d54-b1d1-6fdda57be2f3

Test getting console output for a server

Should be able to GET the console output for a given server_id and number of lines.

test_lock_unlock_server()

Test idempotent id: 80a8094c-211e-440a-ab88-9e59d556c7ee

Test locking and unlocking server

Lock the server, and trying to stop it will fail because locked server is not allowed to be stopped by non-admin user. Then unlock the server, now the server can be stopped and started.

test_pause_unpause_server()

Test idempotent id: bd61a9fd-062f-4670-972b-2d6c3e3b9e73

Test pausing and unpausing server

test_reboot_server_hard()

Test idempotent id: 2cb1baf6-ac8d-4429-bf0d-ba8a0ba53e32

Test hard rebooting server

The server should be power cycled.

test_rebuild_server()

Test idempotent id: aaa6cdf3-55a7-461a-add9-1c8596b9a07c

Test rebuilding server

The server should be rebuilt using the provided image and data.

test_resize_server_confirm()

Test idempotent id: 1499262a-9328-4eda-9068-db1ac57498d2

Test resizing server and then confirming

test_resize_server_revert()

Test idempotent id: c03aab19-adb1-44f5-917d-c419577e9e68

Test resizing server and then reverting

The servers RAM and disk space should return to its original values after a resize is reverted.

test_stop_start_server()

Test idempotent id: af8eafd4-38a7-4a4b-bdbc-75145a580560

Test stopping and starting server

test_suspend_resume_server()

Test idempotent id: 0d8ee21e-b749-462d-83da-b85b41c86c7f

Test suspending and resuming server

class ServerActionsTestOtherA(**args*, ***kwargs*)

Bases: *ServerActionsBase*

test_rebuild_server_in_stop_state()

Test idempotent id: 30449a88-5aff-4f9b-9866-6ee9b17f906d

Test rebuilding server in stop state

The server in stop state should be rebuilt using the provided image and remain in SHUTOFF state.

test_rebuild_server_with_volume_attached()

Test idempotent id: b68bd8d6-855d-4212-b59b-2e704044dace

Test rebuilding server with volume attached

The volume should be attached to the instance after rebuild.

test_remove_server_all_security_groups()

Test idempotent id: 1d1c9104-1b0a-11e7-a3d4-fa163e65f5ce

Test removing all security groups from server

test_resize_volume_backed_server_confirm()

Test idempotent id: e6c28180-7454-4b59-b188-0257af08a63b

Test resizing a volume backed server and then confirming

class ServerActionsTestOtherB(**args*, ***kwargs*)

Bases: *ServerActionsBase*

test_create_backup()

Test idempotent id: b963d4f1-94b3-4c40-9e97-7b583f46e470

Test creating server backup

1. create server backup1 with rotation=2, there are 1 backup.
2. create server backup2 with rotation=2, there are 2 backups.
3. create server backup3, due to the rotation is 2, the first one (backup1) will be deleted, so now there are still 2 backups.

test_get_console_output_server_id_in_shutoff_status()

Test idempotent id: 5b65d4e7-4ecd-437c-83c0-d6b79d927568

Test getting console output for a server in SHUTOFF status

Should be able to GET the console output for a given server_id in SHUTOFF status.

test_get_console_output_with_unlimited_size()

Test idempotent id: 89104062-69d8-4b19-a71b-f47b7af093d7

Test getting servers console output with unlimited size

The console output lines length should be bigger than the one of test_get_console_output.

test_get_vnc_console()

Test idempotent id: c6bc11bf-592e-4015-9319-1c98dc64daf5

Test getting vnc console from a server

The returned vnc console url should be in valid format.

test_resize_server_confirm_from_stopped()

Test idempotent id: 138b131d-66df-48c9-a171-64f45eb92962

Test resizing a stopped server and then confirming

test_resize_server_revert_with_volume_attached()

Test idempotent id: fbbf075f-a812-4022-bc5c-ccb8047eef12

Test resizing a volume attached server and then reverting

Tests attaching a volume to a server instance and then resizing the instance. Once the instance is resized, revert the resize which should move the instance and volume attachment back to the original compute host.

test_shelve_paused_server()

Test idempotent id: 8cf9f450-a871-42cf-9bef-77eba189c0b0

Test shelving a paused server

test_shelve_unshelve_server()

Test idempotent id: 77eba8e0-036e-4635-944b-f7a8f3b78dc9

Test shelving and unshelving server

class ServerActionsV293TestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

test_rebuild_volume_backed_server()

Test idempotent id: 6652dab9-ea24-4c93-ab5a-93d79c3041cf

Test rebuilding a volume backed server

class ServersAaction247Test(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test compute server with microversion greater than 2.47

NOTE(gmann): This test tests the Server create backup APIs # response schema for 2.47 microversion. No specific assert # or behaviour verification is needed.

test_create_backup()

Test idempotent id: 252a4bdd-6366-4dae-9994-8c30aa660f23

compute.servers.test_server_addresses module

class ServerAddressesTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test server addresses

test_list_server_addresses()

Test idempotent id: 6eb718c0-02d9-4d5e-acd1-4e0c269cef39

Test listing server address

All public and private addresses for a server should be returned.

test_list_server_addresses_by_network()

Test idempotent id: 87bbc374-5538-4f64-b673-2b0e4443cc30

Test listing server addresses filtered by network addresses

Providing a network address should filter the addresses same with the specified one.

compute.servers.test_server_addresses_negative module

class ServerAddressesNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Negative tests of listing server addresses

test_list_server_addresses_by_network_neg()

Test idempotent id: a2ab5144-78c0-4942-a0ed-cc8edccfd9ba

List addresses by network should fail if network name not valid

test_list_server_addresses_invalid_server_id()

Test idempotent id: 02c3f645-2d2e-4417-8525-68c0407d001b

List addresses request should fail if server id not in system

compute.servers.test_server_group module

class ServerGroup264TestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

These tests check for the server-group APIs 2.64 microversion.

This tests is only to verify the POST, GET server-groups APIs response schema with 2.64 microversion

test_create_get_server_group()

Test idempotent id: b52f09dd-2133-4037-9a5d-bdb260096a88

class ServerGroupTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

These tests check for the server-group APIs.

They create/delete server-groups with different policies. policies = affinity/anti-affinity It also adds the tests for list and get details of server-groups

test_create_delete_multiple_server_groups_with_same_name_policy()

Test idempotent id: 154dc5a4-a2fe-44b5-b99e-f15806a4a113

Test Create/Delete the server-groups with same name and policy

test_create_delete_server_group_with_affinity_policy()

Test idempotent id: 5dc57eda-35b7-4af7-9e5f-3c2be3d2d68b

Test Create/Delete the server-group with affinity policy

test_create_delete_server_group_with_anti_affinity_policy()

Test idempotent id: 3645a102-372f-4140-afad-13698d850d23

Test Create/Delete the server-group with anti-affinity policy

test_create_server_with_scheduler_hint_group()

Test idempotent id: ed20d3fb-9d1f-4329-b160-543fbd5d9811

Test creating a server with the scheduler hint group

test_list_server_groups()

Test idempotent id: d4874179-27b4-4d7d-80e4-6c560cdfe321

Test listing the server-groups

test_show_server_group()

Test idempotent id: b3545034-dd78-48f0-bdc2-a4adfa6d0ead

Test getting the server-group detail

compute.servers.test_server_metadata module

class ServerMetadataTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test server metadata

test_delete_server_metadata_item()

Test idempotent id: 127642d6-4c7b-4486-b7cd-07265a378658

Test deleting server metadata item

The metadata value/key pair should be deleted from the server.

test_get_server_metadata_item()

Test idempotent id: 3043c57d-7e0e-49a6-9a96-ad569c265e6a

Test getting specific server metadata item

test_list_server_metadata()

Test idempotent id: 479da087-92b3-4dcf-aeb3-fd293b2d14ce

Test listing server metadata

All metadata key/value pairs for a server should be returned.

test_set_server_metadata()

Test idempotent id: 211021f6-21de-4657-a68f-908878cfe251

Test setting server metadata

The servers metadata should be replaced with the provided values

test_set_server_metadata_item()

Test idempotent id: 58c02d4f-5c67-40be-8744-d3fa5982eb1c

Test updating specific server metadata item

The metadata items value should be updated to the provided value.

test_update_metadata_empty_body()

Test idempotent id: 0f58d402-e34a-481d-8af8-b392b17426d9

Test updating server metadata to empty values

The original server metadata should not be lost if empty metadata body is passed.

test_update_server_metadata()

Test idempotent id: 344d981e-0c33-4997-8a5d-6c1d803e4134

Test updating server metadata

The servers metadata values should be updated to the provided values.

compute.servers.test_server_metadata_negative module

class ServerMetadataNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Negative tests of server metadata

test_create_server_metadata_blank_key()

Test idempotent id: 92431555-4d8b-467c-b95b-b17daa5e57ff

Test creating server with blank metadata key should fail

test_delete_metadata_non_existent_server()

Test idempotent id: 6bbd88e1-f8b3-424d-ba10-ae21c45ada8d

Test deleting metadata item from a non existent server

Should not be able to delete metadata item from a non-existent server.

test_list_server_metadata_non_existent_server()

Test idempotent id: f408e78e-3066-4097-9299-3b0182da812e

Test listing metadata for a non existent server should fail

test_metadata_items_limit()

Test idempotent id: d8c0a210-a5c3-4664-be04-69d96746b547

Test set/update server metadata over limit should fail

A 403 Forbidden or 413 Overlimit (old behaviour) exception will be raised while exceeding metadata items limit for project.

test_server_create_metadata_key_too_long()

Test idempotent id: fe114a8f-3a57-4eff-9ee2-4e14628df049

Test creating server with too long metadata key should fail

test_server_metadata_non_existent_server()

Test idempotent id: 4d9cd7a3-2010-4b41-b8fe-3bbf0b169466

Test getting metadata item for a non existent server should fail

test_set_metadata_invalid_key()

Test idempotent id: 0025fbd6-a4ba-4cde-b8c2-96805dcfdabc

Test setting server metadata item with wrong key in body

Raise BadRequest if key in uri does not match the key passed in body.

test_set_metadata_non_existent_server()

Test idempotent id: 0df38c2a-3d4e-4db5-98d8-d4d9fa843a12

Test setting metadata for a non existent server should fail

test_set_server_metadata_blank_key()

Test idempotent id: 96100343-7fa9-40d8-80fa-d29ef588ce1c

Test setting server metadata with blank key should fail

test_set_server_metadata_missing_metadata()

Test idempotent id: 64a91aee-9723-4863-be44-4c9d9f1e7d0e

Test setting server metadata without metadata field should fail

test_update_metadata_non_existent_server()

Test idempotent id: 904b13dc-0ef2-4e4c-91cd-3b4a0f2f49d8

Test updating metadata for a non existent server should fail

test_update_metadata_with_blank_key()

Test idempotent id: a452f38c-05c2-4b47-bd44-a4f0bf5a5e48

Test updating server metadata to blank key should fail

compute.servers.test_server_password module

class ServerPasswordTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test server password

test_delete_server_password()

Test idempotent id: f8229e8b-b625-4493-800a-bde86ac611ea

Test deleting password from a server

test_get_server_password()

Test idempotent id: f83b582f-62a8-4f22-85b0-0dee50ff783a

Test getting password of a server

compute.servers.test_server_personality module

class ServerPersonalityTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test servers with injected files

test_can_create_server_with_max_number_personality_files()

Test idempotent id: 52f12ee8-5180-40cc-b417-31572ea3d555

Test creating server with maximum allowed number of injected files

Server should be created successfully if maximum allowed number of files is injected into the server during creation.

test_create_server_with_personality()

Test idempotent id: 3cfe87fd-115b-4a02-b942-7dc36a337fdf

Test creating server with file injection

test_personality_files_exceed_limit()

Test idempotent id: 176cd8c9-b9e8-48ee-a480-180beab292bf

Test creating server with injected files over limitation

Server creation should fail if greater than the maximum allowed number of files are injected into the server.

test_rebuild_server_with_personality()

Test idempotent id: 128966d8-71fc-443c-8cab-08e24114ecc9

Test injecting file when rebuilding server

compute.servers.test_server_rescue module

class BaseServerStableDeviceRescueTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

class ServerBootFromVolumeStableRescueTest(**args*, ***kwargs*)

Bases: *BaseServerStableDeviceRescueTest*

Test rescuing server specifying type of device for the rescue disk

Test rescuing server specifying type of device for the rescue disk with compute microversion greater than 2.86.

test_stable_device_rescue_bfv_blank_volume()

Test idempotent id: 48f123cb-922a-4065-8db6-b9a9074a556b

Test rescuing server with blank volume as block_device_mapping_v2

Create a server with block_device_mapping_v2 with blank volume, then rescue the server with disk and virtio as the rescue disk.

test_stable_device_rescue_bfv_image_volume()

Test idempotent id: e4636333-c928-40fc-98b7-70a23eef4224

Test rescuing server with blank volume as block_device_mapping_v2

Create a server with block_device_mapping_v2 with image volume, then rescue the server with disk and virtio as the rescue disk.

class ServerRescueTestBase(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

class ServerRescueTestJSON(**args*, ***kwargs*)

Bases: *ServerRescueTestBase*

Test server rescue

test_rescue_unrescue_instance()

Test idempotent id: fd032140-714c-42e4-a8fd-adcd8df06be6

Test rescue/unrescue server

class ServerRescueTestJSONUnderV235(**args*, ***kwargs*)

Bases: *ServerRescueTestBase*

Test server rescue with compute microversion less than 2.36

test_rescued_vm_add_remove_security_group()

Test idempotent id: affca41f-7195-492d-8065-e09eee245404

Test add/remove security group to for rescued server

test_rescued_vm_associate_dissociate_floating_ip()

Test idempotent id: 4842e0cf-e87d-4d9d-b61f-f4791da3cacc

Test associate/dissociate floating ip for rescued server

class ServerStableDeviceRescueTest(**args*, ***kwargs*)

Bases: *BaseServerStableDeviceRescueTest*

Test rescuing server specifying type of device for the rescue disk

test_stable_device_rescue_disk_scsi()

Test idempotent id: 12340157-6306-4745-bdda-cfa019908b48

Test rescuing server with disk and scsi as the rescue disk

test_stable_device_rescue_disk_usb()

Test idempotent id: 647d04cf-ad35-4956-89ab-b05c5c16f30c

Test rescuing server with disk and usb as the rescue disk

test_stable_device_rescue_disk_virtio()

Test idempotent id: 16865750-1417-4854-bcf7-496e6753c01e

Test rescuing server with disk and virtio as the rescue disk

test_stable_device_rescue_disk_virtio_with_volume_attached()

Test idempotent id: a3772b42-00bf-4310-a90b-1cc6fd3e7eab

Test rescuing server with volume attached

Attach a volume to the server and then rescue the server with disk and virtio as the rescue disk.

class ServerStableDeviceRescueTestIDE(**args*, ***kwargs*)

Bases: *BaseServerStableDeviceRescueTest*

Test rescuing server using an IDE device for the rescue disk

test_stable_device_rescue_cdrom_ide()

Test idempotent id: 947004c3-e8ef-47d9-9f00-97b74f9eaf96

Test rescuing server with cdrom and ide as the rescue disk

compute.servers.test_server_rescue_negative module

class ServerRescueNegativeTestJSON(**args*, ***kwargs*)

Bases: `BaseV2ComputeTest`

Negative tests of server rescue

test_rescue_non_existent_server()

Test idempotent id: 6dfc0a55-3a77-4564-a144-1587b7971dde

Test rescuing a non-existing server should fail

test_rescue_paused_instance()

Test idempotent id: cc3a883f-43c0-4fb6-a9bb-5579d64984ed

Test rescuing a paused server should fail

test_rescued_vm_attach_volume()

Test idempotent id: d0ccac79-0091-4cf4-a1ce-26162d0cc55f

Test attaching volume to a rescued server should fail

test_rescued_vm_detach_volume()

Test idempotent id: f56e465b-fe10-48bf-b75d-646cda3a8bc9

Test detaching volume from a rescued server should fail

test_rescued_vm_reboot()

Test idempotent id: db22b618-f157-4566-a317-1b6d467a8094

Test rebooing a rescued server should fail

test_rescued_vm_rebuild()

Test idempotent id: 70cdb8a1-89f8-437d-9448-8844fd82bf46

Test rebuilding a rescued server should fail

compute.servers.test_server_tags module

class ServerTagsTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test server tags with compute microversion greater than 2.25

test_check_tag_existence()

Test idempotent id: 81279a66-61c3-4759-b830-a2dbe64cbe08

Test checking server tag existence

test_create_delete_tag()

Test idempotent id: 8d95abe2-c658-4c42-9a44-c0258500306b

Test creating and deleting server tag

test_delete_all_tags()

Test idempotent id: a63b2a74-e918-4b7c-bcab-10c855f3a57e

Test deleting all server tags

test_update_all_tags()

Test idempotent id: a2c1af8c-127d-417d-974b-8115f7e3d831

Test updating all server tags

compute.servers.test_servers module

class ServerShowV247Test(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test servers API with compute microversion greater than 2.46

test_show_server()

Test idempotent id: 88b0bdb2-494c-11e7-a919-92ebcb67fe33

Test getting server detail

test_update_rebuild_list_server()

Test idempotent id: 8de397c2-57d0-4b90-aa30-e5d668f21a8b

Test update/rebuild/list server

class ServerShowV263Test(*args, **kwargs)

Bases: BaseV2ComputeTest

Test servers API with compute microversion greater than 2.62

test_show_update_rebuild_list_server()

Test idempotent id: 71b8e3d5-11d2-494f-b917-b094a4afed3c

Test show/update/rebuild/list server

class ServersListShow296Test(*args, **kwargs)

Bases: BaseV2ComputeTest

Test compute server with microversion >= than 2.96

This test tests the Server APIs response schema for 2.96 microversion. No specific assert or behaviour verification is needed.

test_list_show_server_296()

Test idempotent id: 4eee1ffe-9e00-4c99-a431-0d3e0f323a8f

class ServersTestJSON(*args, **kwargs)

Bases: BaseV2ComputeTest

Test servers API

test_create_server_specify_multibyte_character_name()

Test idempotent id: defbaca5-d611-49f5-ae21-56ee25d2db49

Test creating server with multi character name

prefix character is: http://unicode.org/cldr/utility/character.jsp?a=20A1

We use a string with 3 byte utf-8 character due to nova will return 400(Bad Request) if we attempt to send a name which has 4 byte utf-8 character.

test_create_server_with_admin_password()

Test idempotent id: b92d5ec7-b1dd-44a2-87e4-45e888c46ef0

Test creating server with admin password

If an admin password is provided on server creation, the servers root password should be set to that password.

test_create_server_with_ipv6_addr_only()

Test idempotent id: 38fb1d02-c3c5-41de-91d3-9bc2025a75eb

Test creating server with ipv6 address only(no ipv4 address)

test_create_specify_keypair()

Test idempotent id: f9e15296-d7f9-4e62-b53f-a04e89160833

Test creating server with keypair

test_create_with_existing_server_name()

Test idempotent id: 8fea6be7-065e-47cf-89b8-496e6f96c699

Test creating a server with already existing name is allowed

test_update_access_server_address()

Test idempotent id: 89b90870-bc13-4b73-96af-f9d4f2b70077

Test updating servers access addresses to the provided value

test_update_server_name()

Test idempotent id: 5e6ccff8-349d-4852-a8b3-055df7988dd2

Test updating server name to the provided value

compute.servers.test_servers_microversions module

class ServerShowV254Test(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test servers API schema for compute microversion greater than 2.53

test_rebuild_server()

Test idempotent id: 09170a98-4940-4637-add7-1a35121f1a5a

Test rebuilding server with microversion greater than 2.53

class ServerShowV257Test(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test servers API schema for compute microversion greater than 2.56

test_rebuild_server()

Test idempotent id: 803df848-080a-4261-8f11-b020cd9b6f60

Test rebuilding server with microversion greater than 2.56

compute.servers.test_servers_negative module

class ServersNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Negative tests of servers

test_create_numeric_server_name()

Test idempotent id: fd57f159-68d6-4c2a-902b-03070828a87e

Creating a server with numeric server name should fail

test_create_server_from_non_bootable_volume()

Test idempotent id: 74085be3-a370-4ca2-bc51-2d0e10e0f573

Creating a server from a non bootable volume should fail

test_create_server_invalid_bdm_in_2nd_dict()

Test idempotent id: 12146ac1-d7df-4928-ad25-b1f99e5286cd

Creating a server with invalid block_device_mapping_v2 should fail

Create a server with invalid block_device_mapping_v2, an error is returned.

test_create_server_metadata_exceeds_length_limit()

Test idempotent id: 7fc74810-0bd2-4cd7-8244-4f33a9db865a

Creating a server with metadata longer than limit should fail

test_create_server_name_length_exceeds_256()

Test idempotent id: c3e0fb12-07fc-4d76-a22e-37409887afe8

Creating a server with name length exceeding limit should fail

Create a server with name length exceeding 255 characters, an error is returned.

test_create_with_invalid_flavor()

Test idempotent id: 18f5227f-d155-4429-807c-ccb103887537

Creating a server with an unknown flavor should fail

test_create_with_invalid_image()

Test idempotent id: fcba1052-0a50-4cf3-b1ac-fae241edf02f

Creating a server with an unknown image should fail

test_create_with_invalid_network_uuid()

Test idempotent id: 4e72dc2d-44c5-4336-9667-f7972e95c402

Creating a server with invalid network uuid should fail

test_create_with_non_existent_keypair()

Test idempotent id: 7a2efc39-530c-47de-b875-2dd01c8d39bd

Creating a server with non-existent keypair should fail

test_create_with_nonexistent_security_group()

Test idempotent id: c5fa6041-80cd-483b-aa6d-4e45f19d093c

Creating a server with a nonexistent security group should fail

test_delete_non_existent_server()

Test idempotent id: 1041b4e6-514b-4855-96a5-e974b60870a3

Deleting a non existent server should fail

test_delete_server_pass_id_exceeding_length_limit()

Test idempotent id: f4d7279b-5fd2-4bf2-9ba4-ae35df0d18c5

Deleting server with a server ID exceeding length limit should fail

Pass a server ID that exceeds length limit to delete server, an error is returned.

test_delete_server_pass_negative_id()

Test idempotent id: 75f79124-277c-45e6-a373-a1d6803f4cc4

Passing an invalid string parameter to delete server should fail

test_force_delete_nonexistent_server_id()

Test idempotent id: 6f47992b-5144-4250-9f8b-f00aa33950f3

Force-deleting a non existent server should fail

test_get_console_output_of_non_existent_server()

Test idempotent id: 7dd919e7-413f-4198-bebb-35e2a01b13e9

Getting the console output for a non existent server should fail

test_get_non_existent_server()

Test idempotent id: 3436b02f-1b1e-4f03-881e-c6a602327439

Getting a non existent server details should fail

test_invalid_access_ip_v4_address()

Test idempotent id: 7f70a4d1-608f-4794-9e56-cb182765972c

Creating a server with invalid ipv4 ip address should fail

An access IPv4 address must match a valid address pattern

test_invalid_ip_v6_address()

Test idempotent id: 5226dd80-1e9c-4d8a-b5f9-b26ca4763fd0

Creating a server with invalid ipv6 ip address should fail

An access IPv6 address must match a valid address pattern

test_pause_non_existent_server()

Test idempotent id: 6a8dc0c6-6cd4-4c0a-9f32-413881828091

Pausing a non existent server should fail

test_pause_paused_server()

Test idempotent id: d1417e7f-a509-41b5-a102-d5eed8613369

Pausing a paused server should fail

test_personality_file_contents_not_encoded()

Test idempotent id: b8a7235e-5246-4a8f-a08e-b34877c6586f

Using an unencoded injected file to create server should fail

test_reboot_deleted_server()

Test idempotent id: 581a397d-5eab-486f-9cf9-1014bbd4c984

Rebooting a deleted server should fail

test_reboot_non_existent_server()

Test idempotent id: d4c023a0-9c55-4747-9dd5-413b820143c7

Rebooting a non existent server should fail

test_rebuild_deleted_server()

Test idempotent id: 98fa0458-1485-440f-873b-fe7f0d714930

Rebuilding a deleted server should fail

test_rebuild_non_existent_server()

Test idempotent id: d86141a7-906e-4731-b187-d64a2ea61422

Rebuilding a non existent server should fail

test_resize_nonexistent_server()

Test idempotent id: 7ea45b3e-e770-46fa-bfcc-9daaf6d987c0

Resizing a non-existent server should fail

test_resize_server_with_non_existent_flavor()

Test idempotent id: ced1a1d7-2ab6-45c9-b90f-b27d87b30efd

Resizing a server with non existent flavor should fail

test_resize_server_with_null_flavor()

Test idempotent id: 45436a7d-a388-4a35-a9d8-3adc5d0d940b

Resizing a server with null flavor should fail

test_restore_nonexistent_server_id()

Test idempotent id: 9c6d38cc-fcfb-437a-85b9-7b788af8bf01

Restore-deleting a non existent server should fail

We can restore a soft deleted server, but cant restore a non existent server.

test_resume_non_existent_server()

Test idempotent id: 221cd282-bddb-4837-a683-89c2487389b6

Resuming a non existent server should fail

test_resume_server_invalid_state()

Test idempotent id: ccb6294d-c4c9-498f-8a43-554c098bfadb

Resuming an active server should fail

test_server_name_blank()

Test idempotent id: dbbfd247-c40c-449e-8f6c-d2aa7c7da7cf

Creating a server with name parameter empty should fail

test_shelve_non_existent_server()

Test idempotent id: abca56e2-a892-48ea-b5e5-e07e69774816

Shelving a non existent server should fail

test_shelve_shelved_server()

Test idempotent id: 443e4f9b-e6bf-4389-b601-3a710f15fddd

Shelving a shelved server should fail

test_stop_non_existent_server()

Test idempotent id: a31460a9-49e1-42aa-82ee-06e0bb7c2d03

Stopping a non existent server should fail

test_suspend_non_existent_server()

Test idempotent id: d1f032d5-7b6e-48aa-b252-d5f16dd994ca

Suspending a non existent server should fail

test_suspend_server_invalid_state()

Test idempotent id: 7f323206-05a9-4bf8-996b-dd5b2036501b

Suspending a suspended server should fail

test_unpause_non_existent_server()

Test idempotent id: 705b8e3a-e8a7-477c-a19b-6868fc24ac75

Unpausing a non existent server should fail

test_unpause_server_invalid_state()

Test idempotent id: c8e639a7-ece8-42dd-a2e0-49615917ba4f

Unpausing an active server should fail

test_unshelve_non_existent_server()

Test idempotent id: 23d23b37-afaf-40d7-aa5d-5726f82d8821

Unshelving a non existent server should fail

test_unshelve_server_invalid_state()

Test idempotent id: 8f198ded-1cca-4228-9e65-c6b449c54880

Unshelving an active server should fail

test_update_name_of_non_existent_server()

Test idempotent id: aa8eed43-e2cb-4ebf-930b-da14f6a21d81

Updating name of a non-existent server should fail

test_update_server_name_length_exceeds_256()

Test idempotent id: 5c8e244c-dada-4590-9944-749c455b431f

Updating name of server exceeding the name length limit should fail

Update name of server exceeding the name length limit, an error is returned.

test_update_server_set_empty_name()

Test idempotent id: 38204696-17c6-44da-9590-40f87fb5a899

Updating name of the server to an empty string should fail

class ServersNegativeTestMultiTenantJSON(*args, **kwargs)

Bases: BaseV2ComputeTest

Negative tests of servers for multiple projects

test_delete_a_server_of_another_tenant()

Test idempotent id: 5c75009d-3eea-423e-bea3-61b09fd25f9c

Deleting a server that belongs to another project should fail

test_update_server_of_another_tenant()

Test idempotent id: 543d84c1-dd2e-4c6d-8cb2-b9da0efaa384

Updating server that belongs to another project should fail

Update name of a server that belongs to another project, an error is returned.

Module contents

compute.volumes package

Submodules

compute.volumes.test_attach_volume module

class AttachVolumeMultiAttachTest(**args*, ***kwargs*)

Bases: *BaseAttachVolumeTest*

Test attaching one volume to multiple servers

Test attaching one volume to multiple servers with compute microversion greater than 2.59.

test_boot_from_multiattach_volume()

Test idempotent id: 65e33aa2-185b-44c8-b22e-e524973ed625

Simple test to boot an instance from a multiattach volume.

test_boot_from_multiattach_volume_direct_lun(*boot=False*)

Test idempotent id: bfe61d6e-767a-4f93-9de8-054355536475

test_boot_with_multiattach_volume_direct_lun(*boot=False*)

Test idempotent id: 07eb6686-571c-45f0-9d96-446b120f1121

test_list_get_volume_attachments_multiattach()

Test idempotent id: 8d5853f7-56e7-4988-9b0c-48cea3c7049a

Test listing and getting multiattached volume attachments

Attach a single volume to two servers, list attachments from the volume and make sure the server uuids are in the list, then detach the volume from servers one by one.

test_resize_server_with_multiattached_volume()

Test idempotent id: f01c7169-a124-4fc7-ae60-5e380e247c9c

Test resizing servers with multiattached volume

Attach a single volume to multiple servers, then resize the servers

test_snapshot_volume_backed_multiattach()

Test idempotent id: 885ac48a-2d7a-40c5-ae8b-1993882d724c

Boots a server from a multiattach volume and snapshots the server.

Creating the snapshot of the server will also create a snapshot of the volume.

class AttachVolumeShelveTestJSON(**args*, ***kwargs*)

Bases: *BaseAttachVolumeTest*

Testing volume with shelved instance.

This test checks the attaching and detaching volumes from a shelved or shelved offload instance.

Note that these are uncommon scenarios until blueprint detach-boot-volume is implemented in the compute service.

test_attach_volume_shelved_or_offload_server()

Test idempotent id: 13a940b6-3474-4c3c-b03f-29b89112bfee

Test attaching volume to shelved server

Create server, count number of volumes on it, shelve server and attach pre-created volume to shelved server, then unshelve the server and check that attached volume exists.

test_detach_volume_shelved_or_offload_server()

Test idempotent id: b54e86dd-a070-49c4-9c07-59ae6dae15aa

Test detaching volume from shelved server

Count number of volumes on server, shelve server and attach pre-created volume to shelved server, then detach the volume, unshelve the instance and check that we have the expected number of volume(s).

class AttachVolumeTestJSON(**args*, ***kwargs*)

Bases: *BaseAttachVolumeTest*

Test attaching volume to server

test_attach_detach_volume()

Test idempotent id: 52e9045a-e90d-4c0d-9087-79d657faffff

Test attaching and detaching volume from server

Stop and Start a server with an attached volume, ensuring that the volume remains attached.

test_list_get_volume_attachments()

Test idempotent id: 7fa563fe-f0f7-43eb-9e22-a1ece036b513

Test listing and getting volume attachments

First we attach one volume to the server, check listing and getting the volume attachment of the server. Then we attach another volume to the server, check listing and getting the volume attachments of the server. Finally we detach the volumes from the server one by one.

class BaseAttachVolumeTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Base class for the attach volume tests in this module.

compute.volumes.test_attach_volume_negative module

class AttachVolumeNegativeTest(**args*, ***kwargs*)

Bases: BaseAttachVolumeTest

Negative tests of volume attaching

test_attach_attached_volume_to_different_server()

Test idempotent id: ee37a796-2afb-11e7-bc0f-fa163e65f5ce

Test attaching attached volume to different server should fail

test_attach_attached_volume_to_same_server()

Test idempotent id: aab919e2-d992-4cbb-a4ed-745c2475398c

Test attaching attached volume to same server should fail

Test attaching the same volume to the same instance once its already attached. The nova/cinder validation for this differs depending on whether or not cinder v3.27 is being used to attach the volume to the instance.

test_delete_attached_volume()

Test idempotent id: a313b5cd-fbd0-49cc-94de-870e99f763c7

Test deleting attachemd volume should fail

compute.volumes.test_volume_snapshots module

class VolumesSnapshotsTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test volume snapshots with compute microversion less than 2.36

test_volume_snapshot_create_get_list_delete()

Test idempotent id: cd4ec87d-7825-450d-8040-6e2068f2da8f

Test create/get/list/delete volume snapshot

compute.volumes.test_volumes_get module

class VolumesGetTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test compute volumes API with microversion less than 2.36

test_volume_create_get_delete()

Test idempotent id: f10f25eb-9775-4d9d-9cbe-1cf54dae9d5f

Test create/get/delete volume

compute.volumes.test_volumes_list module

class VolumesTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test listing volumes with compute microversion less than 2.36

test_volume_list()

Test idempotent id: bc2dd1a0-15af-48e5-9990-f2e75a48325d

Test listing volumes should return all volumes

test_volume_list_param_limit()

Test idempotent id: 1048ed81-2baf-487a-b284-c0622b86e7b8

Test listing volumes based on limit set

If we list volumes with limit=2, then only 2 volumes should be returned.

test_volume_list_param_offset_and_limit()

Test idempotent id: 51c22651-a074-4ea7-af0b-094f9331303e

Test listing volumes based on offset and limit set

If we list volumes with offset=1 and limit=1, then 1 volume located in the position 1 in the all volumes list should be returned. (The items in the all volumes list start from position 0.)

test_volume_list_with_detail_param_limit()

Test idempotent id: 33985568-4965-49d5-9bcc-0aa007ca5b7a

Test listing volumes with detail based on limit set

If we list volumes with detail with limit=2, then only 2 volumes with detail should be returned.

test_volume_list_with_detail_param_offset_and_limit()

Test idempotent id: 06b6abc4-3f10-48e9-a7a1-3facc98f03e5

Test listing volumes with detail based on offset and limit set

If we list volumes with detail with offset=1 and limit=1, then 1 volume with detail located in the position 1 in the all volumes list should be returned. (The items in the all volumes list start from position 0.)

test_volume_list_with_details()

Test idempotent id: bad0567a-5a4f-420b-851e-780b55bb867c

Test listing volumes with detail should return all volumes

compute.volumes.test_volumes_negative module

class VolumesNegativeTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Negative tests of volumes with compute microversion less than 2.36

test_create_volume_with_invalid_size()

Test idempotent id: 5125ae14-152b-40a7-b3c5-eae15e9022ef

Test creating volume with invalid size should fail

test_create_volume_with_size_zero()

Test idempotent id: 8cce995e-0a83-479a-b94d-e1e40b8a09d1

Test creating volume with size=0 should fail

test_create_volume_without_passing_size()

Test idempotent id: 131cb3a1-75cc-4d40-b4c3-1317f64719b0

Test creating volume without specifying size should fail

test_delete_invalid_volume_id()

Test idempotent id: 62972737-124b-4513-b6cf-2f019f178494

Test deleting volume with an invalid volume id should fail

test_delete_volume_without_passing_volume_id()

Test idempotent id: 0d1417c5-4ae8-4c2c-adc5-5f0b864253e5

Test deleting volume without volume id should fail

test_get_volume_without_passing_volume_id()

Test idempotent id: 62bab09a-4c03-4617-8cca-8572bc94af9b

Test getting volume details without volume id should fail

test_volume_delete_nonexistent_volume_id()

Test idempotent id: 54a34226-d910-4b00-9ef8-8683e6c55846

Test deleting a nonexistent volume should fail

test_volume_get_nonexistent_volume_id()

Test idempotent id: c03ea686-905b-41a2-8748-9635154b7c57

Test getting details of a non existent volume should fail

Module contents

Submodules

compute.api_microversion_fixture module

class APIMicroversionFixture(*compute_microversion*)

Bases: Fixture

compute.base module

class BaseV2ComputeAdminTest(**args*, ***kwargs*)

Bases: *BaseV2ComputeTest*

Base test case class for Compute Admin API tests.

class BaseV2ComputeTest(**args*, ***kwargs*)

Bases: *BaseMicroversionTest*, BaseTestCase

Base test case class for all Compute API tests.

compute.test_extensions module

class ExtensionsTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Tests Compute Extensions API

test_get_extension()

Test idempotent id: 05762f39-bdfa-4cdb-9b46-b78f8e78e2fd

Test getting specified compute extension details

test_list_extensions()

Test idempotent id: 3bb27738-b759-4e0d-a5fa-37d7a6df07d1

Test listing compute extensions

compute.test_networks module

class ComputeNetworksTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test compute networks API with compute microversion less than 2.36

test_list_networks()

Test idempotent id: 3fe07175-312e-49a5-a623-5f52eeada4c2

Test listing networks using compute networks API

compute.test_quotas module

class QuotasTestJSON(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test compute quotas

test_compare_tenant_quotas_with_default_quotas()

Test idempotent id: cd65d997-f7e4-4966-a7e9-d5001b674fdc

Test tenants are created with the default compute quota values

test_get_default_quotas()

Test idempotent id: 9bfecac7-b966-4f47-913f-1a9e2c12134a

Test user can get the default compute quota set for its project

test_get_quotas()

Test idempotent id: f1ef0a97-dbbb-4cca-adc5-c9fbc4f76107

Test user can get the compute quota set for its project

compute.test_tenant_networks module

class ComputeTenantNetworksTest(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

Test compute tenant networks API with microversion less than 2.36

test_list_show_tenant_networks()

Test idempotent id: edfea98e-bbe3-4c7a-9739-87b986baff26

Test list/show tenant networks

Fetch all networks that are visible to the tenant: this may include shared and external networks.

compute.test_versions module

class TestVersions(**args*, ***kwargs*)

Bases: BaseV2ComputeTest

test_get_version_details()

Test idempotent id: b953a29e-929c-4a8e-81be-ec3a7e03cb76

Test individual version endpoints info works.

In addition to the GET / version request, there is also a version info document stored at the top of the versioned endpoints. This provides access to details about that endpoint, including min / max version if that implements microversions.

This test starts with the version list, iterates all the returned endpoints, and fetches them. This will also ensure that all the version links are followable constructs which

will help detect configuration issues when SSL termination isnt done completely for a site.

test_list_api_versions()

Test idempotent id: 6c0a0990-43b6-4529-9b61-5fd8daf7c55c

Test that a get of the unversioned url returns the choices doc.

A key feature in OpenStack services is the idea that you can GET / on the service and get a list of the versioned endpoints that you can access. This comes back as a status 300 request. Its important that this is available to API consumers to discover the API they can use.

Module contents

identity

identity package

Subpackages

identity.admin package

Subpackages

identity.admin.v3 package

Submodules

identity.admin.v3.test_application_credentials module

class ApplicationCredentialsV3AdminTest(**args*, ***kwargs*)

Bases: BaseApplicationCredentialsV3Test, BaseIdentityV3AdminTest

Test keystone application credentials

test_create_application_credential_with_roles()

Test idempotent id: 3b3dd48f-3388-406a-a9e6-4d078a552d0e

Test creating keystone application credential with roles

identity.admin.v3.test_credentials module

class CredentialsTestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test keystone credentials

test_credentials_create_get_update_delete()

Test idempotent id: 7cd59bf9-bda4-4c72-9467-d21cab278355

Test creating, getting, updating, deleting of credentials

test_credentials_list_delete()

Test idempotent id: 13202c00-0021-42a1-88d4-81b44d448aab

Test listing credentials

identity.admin.v3.test_default_project_id module

class TestDefaultProjectId(*args, **kwargs)

Bases: BaseIdentityV3AdminTest

Test creating a token without project will default to users project

test_default_project_id()

Test idempotent id: d6110661-6a71-49a7-a453-b5e26640ff6d

Creating a token without project will default to users project

identity.admin.v3.test_domain_configuration module

class DomainConfigurationTestJSON(*args, **kwargs)

Bases: BaseIdentityV3AdminTest

Test domain configuration

test_create_domain_config_and_show_config_groups_and_options()

Test idempotent id: 9e3ff13c-f597-4f01-9377-d6c06c2a1477

Test creating and showing keystone config groups and options

test_create_update_and_delete_domain_config()

Test idempotent id: 7161023e-5dd0-4612-9da0-1bac6ac30b63

Test creating, updating and deleting keystone domain config

test_create_update_and_delete_domain_config_groups_and_opts()

Test idempotent id: c7510fa2-6661-4170-9c6b-4783a80651e9

Test create/update/delete keystone domain config groups and opts

test_show_default_group_config_and_options()

Test idempotent id: 11a02bf0-6f94-4380-b3b0-c8dc18fc0d22

Test showing default keystone group config and options

The API supports only the identity and ldap groups. For the ldap group, a valid value is url or user_tree_dn. For the identity group, a valid value is driver.

identity.admin.v3.test_domains module

class DomainsTestJSON(*args, **kwargs)

Bases: BaseIdentityV3AdminTest

Test identity domains

test_create_domain_with_disabled_status()

Test idempotent id: 036df86e-bb5d-42c0-a7c2-66b9db3a6046

Test creating domain with disabled status

test_create_domain_without_description()

Test idempotent id: 2abf8764-309a-4fa9-bc58-201b799817ad

Test creating domain without description

test_create_update_delete_domain()

Test idempotent id: f2f5b44a-82e8-4dad-8084-0661ea3b18cf

Test creating, updating and deleting domain

test_domain_delete_cascades_content()

Test idempotent id: d8d318b7-d1b3-4c37-94c5-3c5ba0b121ea

Test deleting domain will delete its associated contents

test_list_domains()

Test idempotent id: 8cf516ef-2114-48f1-907b-d32726c734d4

Test listing domains

test_list_domains_filter_by_enabled()

Test idempotent id: 3fd19840-65c1-43f8-b48c-51bdd066dff9

Test listing domains filtering by enabled domains

test_list_domains_filter_by_name()

Test idempotent id: c6aee07b-4981-440c-bb0b-eb598f58ffe9

Test listing domains filtering by name

identity.admin.v3.test_domains_negative module

class DomainsNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Negative tests of identity domains

test_create_domain_with_empty_name()

Test idempotent id: 9018461d-7d24-408d-b3fe-ae37e8cd5c9e

Test creating domain with empty name should fail

Domain name should not be empty

test_create_domain_with_name_length_over_64()

Test idempotent id: 37b1bbf2-d664-4785-9a11-333438586eae

Test creating domain with name over length

Domain name length should not ne greater than 64 characters

test_delete_active_domain()

Test idempotent id: 1f3fbff5-4e44-400d-9ca1-d953f05f609b

Test deleting active domain should fail

test_delete_non_existent_domain()

Test idempotent id: 43781c07-764f-4cf2-a405-953c1916f605

Test attempting to delete a non existent domain should fail

test_domain_create_duplicate()

Test idempotent id: e6f9e4a2-4f36-4be8-bdbc-4e199ae29427

Test creating domain with duplicate name should fail

identity.admin.v3.test_endpoint_groups module

class EndPointGroupsTest(*args, **kwargs)

Bases: BaseIdentityV3AdminTest

Test endpoint groups

test_create_list_show_check_delete_endpoint_group()

Test idempotent id: 7c69e7a1-f865-402d-a2ea-44493017315a

Test create/list/show/check/delete of endpoint group

test_update_endpoint_group()

Test idempotent id: 51c8fc38-fa84-4e76-b5b6-6fc37770fb26

Test updating endpoint group

identity.admin.v3.test_endpoints module

class EndPointsTestJSON(*args, **kwargs)

Bases: BaseIdentityV3AdminTest

Test keystone endpoints

test_create_list_show_delete_endpoint()

Test idempotent id: 0e2446d2-c1fd-461b-a729-b9e73e3e3b37

Test creating, listing, showing and deleting keystone endpoint

test_list_endpoints()

Test idempotent id: c19ecf90-240e-4e23-9966-21cee3f6a618

Test listing keystone endpoints by filters

test_update_endpoint()

Test idempotent id: 37e8f15e-ee7c-4657-a1e7-f6b61e375eff

Test updating keystone endpoint

identity.admin.v3.test_endpoints_negative module

class EndpointsNegativeTestJSON(*args, **kwargs)

Bases: BaseIdentityV3AdminTest

Negative tests of endpoint

test_create_with_enabled_False()

Test idempotent id: ac6c137e-4d3d-448f-8c83-4f13d0942651

Test creating endpoint with invalid enabled value False

Enabled should be a boolean, not a string like False

test_create_with_enabled_True()

Test idempotent id: 9c43181e-0627-484a-8c79-923e8a59598b

Test creating endpoint with invalid enabled value True

Enabled should be a boolean, not a string like True

test_update_with_enabled_False()

Test idempotent id: 65e41f32-5eb7-498f-a92a-a6ccacf7439a

Test updating endpoint with invalid enabled value False

Enabled should be a boolean, not a string like False

test_update_with_enabled_True()

Test idempotent id: faba3587-f066-4757-a48e-b4a3f01803bb

Test updating endpoint with invalid enabled value True

Enabled should be a boolean, not a string like True

identity.admin.v3.test_groups module

class GroupsV3TestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test keystone groups

test_group_create_update_get()

Test idempotent id: 2e80343b-6c81-4ac3-88c7-452f3e9d5129

Test creating, updating and getting keystone group

test_group_users_add_list_delete()

Test idempotent id: 1598521a-2f36-4606-8df9-30772bd51339

Test adding/listing/deleting group users

test_list_groups()

Test idempotent id: cc9a57a5-a9ed-4f2d-a29f-4f979a06ec71

Test listing groups

test_list_user_groups()

Test idempotent id: 64573281-d26a-4a52-b899-503cb0f4e4ec

Test listing user groups when the user is in two groups

identity.admin.v3.test_inherits module

class InheritsV3TestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test keystone inherits

test_inherit_assign_check_revoke_roles_on_projects_group()

Test idempotent id: 26021436-d5a4-4256-943c-ded01e0d4b45

Test assign/list/check/revoke inherited role on project group

test_inherit_assign_check_revoke_roles_on_projects_user()

Test idempotent id: 18b70e45-7687-4b72-8277-b8f1a47d7591

Test assign/list/check/revoke inherited role on project user

test_inherit_assign_list_check_revoke_roles_on_domains_group()

Test idempotent id: c7a8dda2-be50-4fb4-9a9c-e830771078b1

Test assign/list/check/revoke inherited role on domain group

test_inherit_assign_list_check_revoke_roles_on_domains_user()

Test idempotent id: 4e6f0366-97c8-423c-b2be-41eae6ac91c8

Test assign/list/check/revoke inherited role on domain user

test_inherit_assign_list_revoke_user_roles_on_domain()

Test idempotent id: 3acf666e-5354-42ac-8e17-8b68893bcd36

Test assign/list/check/revoke inherited role on domain

test_inherit_assign_list_revoke_user_roles_on_project_tree()

Test idempotent id: 9f02ccd9-9b57-46b4-8f77-dd5a736f3a06

Test assign/list/check/revoke inherited role on project tree

identity.admin.v3.test_list_projects module

class BaseListProjectsTestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

class ListProjectsStaticTestJSON(**args*, ***kwargs*)

Bases: *BaseListProjectsTestJSON*

Test listing projects

These tests can be executed in clouds using the pre-provisioned users

test_list_projects()

Test idempotent id: 1d830662-22ad-427c-8c3e-4ec854b0af44

Test listing projects

test_list_projects_with_domains()

Test idempotent id: fab13f3c-f6a6-4b9f-829b-d32fd44fdf10

Test listing projects filtered by domain

test_list_projects_with_name()

Test idempotent id: fa178524-4e6d-4925-907c-7ab9f42c7e26

Test listing projects filtered by name

class ListProjectsTestJSON(**args*, ***kwargs*)

Bases: *BaseListProjectsTestJSON*

Test listing projects

test_list_projects_with_enabled()

Test idempotent id: 0fe7a334-675a-4509-b00e-1c4b95d5dae8

Test listing the projects with enabled

test_list_projects_with_parent()

Test idempotent id: 6edc66f5-2941-4a17-9526-4073311c1fac

Test listing projects with parent

identity.admin.v3.test_list_users module

class UsersV3TestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test listing keystone users

test_get_user()

Test idempotent id: b4baa3ae-ac00-4b4e-9e27-80deaad7771f

Get a user detail

test_list_user_domains()

Test idempotent id: 08f9aabb-dcfe-41d0-8172-82b5fa0bd73d

List users with domain

test_list_users()

Test idempotent id: b30d4651-a2ea-4666-8551-0c0e49692635

List users

test_list_users_with_name()

Test idempotent id: c285bb37-7325-4c02-bff3-3da5d946d683

List users with name

test_list_users_with_not_enabled()

Test idempotent id: bff8bf2f-9408-4ef5-b63a-753c8c2124eb

List the users with not enabled

identity.admin.v3.test_oauth_consumers module

class OAUTHConsumersV3Test(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

test_create_and_show_consumer()

Test idempotent id: c8307ea6-a86c-47fd-ae7b-5b3b2caca76d

Tests to make sure that a consumer with parameters is made

test_delete_consumer()

Test idempotent id: fdfa1b7f-2a31-4354-b2c7-f6ae20554f93

Tests the delete function.

test_list_consumers()

Test idempotent id: 09ca50de-78f2-4ffb-ac71-f2254036b2b8

Test for listing consumers

test_update_consumer()

Test idempotent id: 080a9b1a-c009-47c0-9979-5305bf72e3dc

Tests the update functionality

identity.admin.v3.test_policies module

class PoliciesTestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test keystone policies

test_create_update_delete_policy()

Test idempotent id: e544703a-2f03-4cf2-9b0f-350782fdb0d3

Test to update keystone policy

test_list_policies()

Test idempotent id: 1a0ad286-2d06-4123-ab0d-728893a76201

Test to list keystone policies

identity.admin.v3.test_project_tags module

class IdentityV3ProjectTagsTest(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test keystone project tags

test_list_update_delete_project_tags()

Test idempotent id: 7c123aac-999d-416a-a0fb-84b915ab10de

Test listing, updating and deleting of project tags

identity.admin.v3.test_projects module

class ProjectsTestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test identity projects

test_associate_user_to_project()

Test idempotent id: 59398d4a-5dc5-4f86-9a4c-c26cc804d6c6

Test associating a user to a project

test_create_is_domain_project()

Test idempotent id: a7eb9416-6f9b-4dbb-b71b-7f73aaef59d5

Test creating is_domain project

test_project_create_enabled()

Test idempotent id: 1f66dc76-50cc-4741-a200-af984509e480

Test creating a project that is enabled

test_project_create_not_enabled()

Test idempotent id: 78f96a9c-e0e0-4ee6-a3ba-fbf6dfd03207

Test creating a project that is not enabled

test_project_create_with_description()

Test idempotent id: 0ecf465c-0dc4-4532-ab53-91ffeb74d12d

Test creating project with a description

test_project_create_with_domain()

Test idempotent id: 5f50fe07-8166-430b-a882-3b2ee0abe26f

Test creating project with a domain

test_project_create_with_parent()

Test idempotent id: 1854f9c0-70bc-4d11-a08a-1c789d339e3d

Test creating root project without providing a parent_id

test_project_get_equals_list()

Test idempotent id: d1db68b6-aebe-4fa0-b79d-d724d2e21162

Test the result of getting project equals that of listing

test_project_update_desc()

Test idempotent id: f138b715-255e-4a7d-871d-351e1ef2e153

Test updating description attribute of a project

test_project_update_enable()

Test idempotent id: b6b25683-c97f-474d-a595-55d410b68100

Test updating the enabled attribute of a project

test_project_update_name()

Test idempotent id: f608f368-048c-496b-ad63-d286c26dab6b

Test updating name attribute of a project

identity.admin.v3.test_projects_negative module

class ProjectsNegativeStaticTestJSON(**args*, ***kwargs*)

Bases: `BaseIdentityV3AdminTest`

Negative tests of projects

These tests can be executed in clouds using the pre-provisioned users

test_create_project_by_unauthorized_user()

Test idempotent id: 8fba9de2-3e1f-4e77-812a-60cb68f8df13

Non-admin user should not be authorized to create a project

test_create_project_with_empty_name()

Test idempotent id: 7828db17-95e5-475b-9432-9a51b4aa79a9

Project name should not be empty

test_create_projects_name_length_over_64()

Test idempotent id: 502b6ceb-b0c8-4422-bf53-f08fdb21e2f0

Project name length should not be greater than 64 characters

test_delete_non_existent_project()

Test idempotent id: 7965b581-60c1-43b7-8169-95d4ab7fc6fb

Attempt to delete a non existent project should fail

test_list_projects_by_unauthorized_user()

Test idempotent id: 24c49279-45dd-4155-887a-cb738c2385aa

Non-admin user should not be able to list projects

test_project_create_duplicate()

Test idempotent id: 874c3e84-d174-4348-a16b-8c01f599561b

Project names should be unique

class ProjectsNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Negative tests of projects

test_project_delete_by_unauthorized_user()

Test idempotent id: 8d68c012-89e0-4394-8d6b-ccd7196def97

Non-admin user should not be able to delete a project

identity.admin.v3.test_regions module

class RegionsTestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test regions

test_create_region_with_specific_id()

Test idempotent id: 2c12c5b5-efcf-4aa5-90c5-bff1ab0cdbe2

Test creating region with specific id

test_create_update_get_delete_region()

Test idempotent id: 56186092-82e4-43f2-b954-91013218ba42

Test creating, updating, getting and updating region

test_list_regions()

Test idempotent id: d180bf99-544a-445c-ad0d-0c0d27663796

Test getting a list of regions

test_list_regions_filter_by_parent_region_id()

Test idempotent id: 2d1057cb-bbde-413a-acdf-e2d265284542

Test listing regions filtered by parent region id

identity.admin.v3.test_roles module

class RolesV3TestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test roles

test_assignments_for_domain_roles()

Test idempotent id: 3859df7e-5b78-4e4d-b10e-214c8953842a

Test assignments for domain roles

test_assignments_for_implied_roles_create_delete()

Test idempotent id: c8828027-df48-4021-95df-b65b92c7429e

Test assignments when implied roles are created and deleted

test_domain_roles_create_delete()

Test idempotent id: d92a41d2-5501-497a-84bb-6e294330e8f8

Test creating, listing and deleting domain roles

test_grant_list_revoke_role_to_group_on_domain()

Test idempotent id: 4bf8a70b-e785-413a-ad53-9f91ce02faa7

Test granting, listing, revoking role to group on domain

test_grant_list_revoke_role_to_group_on_project()

Test idempotent id: cbf11737-1904-4690-9613-97bcbb3df1c4

Test granting, listing, revoking role to group on project

test_grant_list_revoke_role_to_group_on_system()

Test idempotent id: c888fe4f-8018-48db-b959-542225c1b4b6

test_grant_list_revoke_role_to_user_on_domain()

Test idempotent id: 6c9a2940-3625-43a3-ac02-5dcec62ef3bd

Test granting, listing, revoking role to user on domain

test_grant_list_revoke_role_to_user_on_project()

Test idempotent id: c6b80012-fe4a-498b-9ce8-eb391c05169f

Test granting, listing, revoking role to user on project

test_grant_list_revoke_role_to_user_on_system()

Test idempotent id: e5a81737-d294-424d-8189-8664858aae4c

test_implied_domain_roles()

Test idempotent id: eb1e1c24-1bc4-4d47-9748-e127a1852c82

Test creating implied roles when roles are in domains

test_implied_roles_create_check_show_delete()

Test idempotent id: c90c316c-d706-4728-bcba-eb1912081b69

Test creating, checking, showing and deleting implied roles

test_list_all_implied_roles()

Test idempotent id: 3748c316-c18f-4b08-997b-c60567bc6235

Test listing all implied roles

test_list_roles()

Test idempotent id: f5654bcc-08c4-4f71-88fe-05d64e06de94

Test listing roles

test_role_create_update_show_list()

Test idempotent id: 18afc6c0-46cf-4911-824e-9989cc056c3a

Test creating, updating, showing and listing a role

test_roles_hierarchy()

Test idempotent id: dc6f5959-b74d-4e30-a9e5-a8255494ff00

Test creating implied role and listing role inferences rules

identity.admin.v3.test_services module

class ServicesTestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test keystone services

test_create_service_without_description()

Test idempotent id: d1dcb1a1-2b6b-4da8-bbb8-5532ef6e8269

Create a keystone service only with name and type

test_create_update_get_service()

Test idempotent id: 5193aad5-bcb7-411d-85b0-b3b61b96ef06

Test creating, updating and getting of keystone service

test_list_services()

Test idempotent id: e55908e8-360e-439e-8719-c3230a3e179e

Create, List, Verify and Delete Keystone Services

identity.admin.v3.test_tokens module

class TokensV3TestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test tokens

test_get_available_domain_scopes()

Test idempotent id: ec5ecb05-af64-4c04-ac86-4d9f6f12f185

Test getting available domain scopes

To verify that listing domain scopes for a user works if the user has a domain role or belongs to a group that has a domain role. For this test, admin client is used to add roles to alt user, which performs API calls, to avoid 401 Unauthorized errors.

test_get_available_project_scopes()

Test idempotent id: 08ed85ce-2ba8-4864-b442-bcc61f16ae89

Test getting available project scopes

test_rescope_token()

Test idempotent id: 565fa210-1da1-4563-999b-f7b5b67cf112

Rescope a token.

An unscoped token can be requested, that token can be used to request a scoped token. The scoped token can be revoked, and the original token used to get a token in a different project.

identity.admin.v3.test_trusts module

class TrustsV3TestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3AdminTest

Test keystone trusts

test_get_trusts_all()

Test idempotent id: 4773ebd5-ecbf-4255-b8d8-b63e6f72b65d

Test getting all keystone trusts

test_get_trusts_query()

Test idempotent id: 6268b345-87ca-47c0-9ce3-37792b43403a

Test getting keystone trusts

test_trust_expire()

Test idempotent id: 0ed14b66-cefd-4b5c-a964-65759453e292

Test expire attribute of keystone trust

To check we can create, get and delete a trust with an expiry specified

test_trust_expire_invalid()

Test idempotent id: 3e48f95d-e660-4fa9-85e0-5a3d85594384

Test invalid expire attribute of a keystone trust

To check an invalid expiry time is rejected with the correct error

test_trust_impersonate()

Test idempotent id: 5a0a91a4-baef-4a14-baba-59bf4d7fcace

Test keystone trust with impersonation enabled

To check we can create, get and delete a trust. Updates are not supported for trusts

test_trust_noimpersonate()

Test idempotent id: ed2a8779-a7ac-49dc-afd7-30f32f936ed2

Test keystone trust with impersonation disabled

To check we can create, get and delete a trust with impersonation=False

identity.admin.v3.test_users module

class UsersV3TestJSON(*args, **kwargs)

Bases: BaseIdentityV3AdminTest

Test keystone users

test_get_user()

Test idempotent id: c10dcd90-461d-4b16-8e23-4eb836c00644

Test getting a user detail

test_list_user_projects()

Test idempotent id: a831e70c-e35b-430b-92ed-81ebbc5437b8

Test listing the projects that a user has access upon

test_password_history_not_enforced_in_admin_reset()

Test idempotent id: 568cd46c-ee6c-4ab4-a33a-d3791931979e

Test setting same password when password history is not enforced

test_update_user_password()

Test idempotent id: 2d223a0e-e457-4a70-9fb1-febe027a0ff9

Test updating user password

test_user_update()

Test idempotent id: b537d090-afb9-4519-b95d-270b0708e87e

Test case to check if updating of user attributes is successful

identity.admin.v3.test_users_negative module

class UsersNegativeTest(*args, **kwargs)

Bases: BaseIdentityV3AdminTest

Negative tests of keystone users

test_authentication_for_disabled_user()

Test idempotent id: b3c9fccc-4134-46f5-b600-1da6fb0a3b1f

Attempt to authenticate for disabled user should fail

test_create_user_for_non_existent_domain()

Test idempotent id: e75f006c-89cc-477b-874d-588e4eab4b17

Attempt to create a user in a non-existent domain should fail

Module contents

Module contents

identity.v3 package

Submodules

identity.v3.test_access_rules module

class AccessRulesV3Test(*args, **kwargs)

Bases: BaseIdentityV3Test

test_delete_access_rule()

Test idempotent id: 278757e9-e193-4bf8-adf2-0b0a229a17d0

test_list_access_rules()

Test idempotent id: 2354c498-5119-4ba5-9f0d-44f16f78fb0e

test_show_access_rule()

Test idempotent id: 795dd507-ca1e-40e9-ba90-ff0a08689ba4

identity.v3.test_api_discovery module

class TestApiDiscovery(*args, **kwargs)

Bases: BaseIdentityV3Test

Tests for identity API discovery features.

test_api_media_types()

Test idempotent id: 657c1970-4722-4189-8831-7325f3bc4265

Test showing identity v3 api version media type

test_api_version_resources()

Test idempotent id: b9232f5e-d9e5-4d97-b96c-28d3db4de1bd

Test showing identity v3 api version resources

test_api_version_statuses()

Test idempotent id: 8879a470-abfb-47bb-bb8d-5a7fd279ad1e

Test showing identity v3 api version status

test_identity_v3_existence()

Test idempotent id: 79aec9ae-710f-4c54-a4fc-3aa25b4feac3

Test that identity v3 version should exist

test_list_api_versions()

Test idempotent id: 721f480f-35b6-46c7-846e-047e6acea0dc

Test listing identity api versions

NOTE: Actually this API doesnt depend on v3 API at all, because the API operation is GET / without v3s endpoint. The reason of this test path is just v3 API is CURRENT on Keystone side.

identity.v3.test_application_credentials module

class ApplicationCredentialsV3Test(*args, **kwargs)

Bases: BaseApplicationCredentialsV3Test

Test application credentials

test_create_application_credential()

Test idempotent id: 8080c75c-eddc-4786-941a-c2da7039ae61

Test creating application credential

test_create_application_credential_access_rules()

Test idempotent id: 529936eb-aa5d-463d-9f79-01c113d3b88f

test_create_application_credential_expires()

Test idempotent id: 852daf0c-42b5-4239-8466-d193d0543ed3

Test creating application credential with expire time

test_list_application_credentials()

Test idempotent id: ff0cd457-6224-46e7-b79e-0ada4964a8a6

Test listing application credentials

test_query_application_credentials()

Test idempotent id: 9bb5e5cc-5250-493a-8869-8b665f6aa5f6

Test listing application credentials filtered by name

identity.v3.test_catalog module

class IdentityCatalogTest(**args*, ***kwargs*)

Bases: BaseIdentityV3Test

Test services catalog type values

test_catalog_standardization()

Test idempotent id: 56b57ced-22b8-4127-9b8a-565dfb0207e2

Test that every service has a standard catalog type value

identity.v3.test_domains module

class DefaultDomainTestJSON(**args*, ***kwargs*)

Bases: BaseIdentityV3Test

Test identity default domains

test_default_domain_exists()

Test idempotent id: 17a5de24-e6a0-4e4a-a9ee-d85b6e5612b5

Test showing default domain

identity.v3.test_ec2_credentials module

class EC2CredentialsTest(**args*, ***kwargs*)

Bases: BaseIdentityV3Test

test_create_ec2_credential()

Test idempotent id: b0f55a29-54e5-4166-999d-712347e0c920

Create user ec2 credential.

test_delete_ec2_credential()

Test idempotent id: 9408d61b-8be0-4a8d-9b85-14f61edb456b

Delete user ec2 credential.

test_list_ec2_credentials()

Test idempotent id: 897813f0-160c-4fdc-aabc-24ee635ce4a9

Get the list of user ec2 credentials.

test_show_ec2_credential()

Test idempotent id: 8b8d1010-5958-48df-a6cd-5e3df72e6bcf

Get the definite user ec2 credential.

identity.v3.test_projects module

class IdentityV3ProjectsTest(**args*, ***kwargs*)

Bases: BaseIdentityV3Test

Test identity projects

test_list_projects_returns_only_authorized_projects()

Test idempotent id: 86128d46-e170-4644-866a-cc487f699e1d

Test listing projects only returns authorized projects

identity.v3.test_tokens module

class TokensV3Test(**args*, ***kwargs*)

Bases: BaseIdentityV3Test

Test identity tokens

test_create_token()

Test idempotent id: 6f8e4436-fc96-4282-8122-e41df57197a9

Test creating token for user

test_token_auth_creation_existence_deletion()

Test idempotent id: 0f9f5a5f-d5cd-4a86-8a5b-c5ded151f212

Test auth/check existence/delete token for user

Tests basic token auth functionality in a way that is compatible with pre-provisioned credentials. The default user is used for token authentication.

test_validate_token()

Test idempotent id: a9512ac3-3909-48a4-b395-11f438e16260

Test validating token for user

identity.v3.test_users module

class IdentityV3UsersTest(**args*, ***kwargs*)

Bases: BaseIdentityV3Test

Test identity user password

test_password_history_check_self_service_api()

Test idempotent id: 941784ee-5342-4571-959b-b80dd2cea516

Test checking password changing history

test_user_account_lockout()

Test idempotent id: a7ad8bbf-2cff-4520-8c1d-96332e151658

Test locking out user account after failure attempts

test_user_update_own_password()

Test idempotent id: ad71bd23-12ad-426b-bb8b-195d2b635f27

Test updating users own password

Module contents

Submodules

identity.base module

class BaseApplicationCredentialsV3Test(*args, **kwargs)

Bases: *BaseIdentityV3Test*

class BaseIdentityTest(*args, **kwargs)

Bases: BaseTestCase

class BaseIdentityV3AdminTest(*args, **kwargs)

Bases: *BaseIdentityV3Test*

class BaseIdentityV3Test(*args, **kwargs)

Bases: *BaseIdentityTest*

Module contents

setup_package()

image

image package

Subpackages

image.v2 package

Subpackages

image.v2.admin package

Submodules

image.v2.admin.test_image_caching module

class ImageCachingTest(*args, **kwargs)

Bases: BaseV2ImageTest

Here we test the caching operations for image

test_image_caching_cycle()

Test idempotent id: 4bf6adba-2f9f-47e9-a6d5-37f21ad4387c

Test image cache APIs

test_remote_and_self_cache()

Test idempotent id: 0a6b7e10-bc30-4a41-91ff-69fb4f5e65f2

Test image cache works with self and remote glance service

image.v2.admin.test_image_task module

class ImageTaskCreate(*args, **kwargs)

Bases: BaseV2ImageAdminTest

Test image task operations

test_image_tasks_create()

Test idempotent id: 669d5387-0340-4abf-b62d-7cc89f539c8c

Test task type import image

test_task_create_fake_image_location()

Test idempotent id: ad6450c6-7060-4ee7-a2d1-41c2604b446c

image.v2.admin.test_images module

class BasicOperationsImagesAdminTest(*args, **kwargs)

Bases: BaseV2ImageAdminTest

Test image operations about image owner

test_create_image_owner_param()

Test idempotent id: 646a6eaa-135f-4493-a0af-12583021224e

Test creating image with specified owner

test_list_public_image()

Test idempotent id: f6ab4aa0-035e-4664-9f2d-c57c6df50605

Test create image as admin and list public image as none admin

test_update_image_owner_param()

Test idempotent id: 525ba546-10ef-4aad-bba1-1858095ce553

Test updating image owner

class ImageLocationsAdminTest(*args, **kwargs)

Bases: BaseV2ImageAdminTest

test_delete_locations()

Test idempotent id: 8a648de4-b745-4c28-a7b5-20de1c3da4d2

class ImportCopyImagesTest(*args, **kwargs)

Bases: BaseV2ImageAdminTest

Test the import copy-image operations

test_image_copy_image_import()

Test idempotent id: 9b3b644e-03d1-11eb-a036-fa163e2eaf49

Test copy-image import functionalities

Create image, import image with copy-image method and verify that import succeeded.

class MultiStoresImagesTest(*args, **kwargs)

Bases: BaseV2ImageAdminTest, BaseV2ImageTest

Test importing and deleting image in multiple stores

test_delete_image_from_specific_store()

Test idempotent id: 1ecec683-41d4-4470-a0df-54969ec74514

Test delete image from specific store

image.v2.admin.test_images_metadefs_namespace_objects module

class MetadataNamespaceObjectsTest(*args, **kwargs)

Bases: BaseV2ImageAdminTest

Test the Metadata definition namespace objects basic functionality

test_create_update_delete_meta_namespace_objects()

Test idempotent id: b1a3775e-3b5c-4f6a-a3b4-1ba3574ae718

Test creating/updating/deleting image metadata namespace objects

test_list_meta_namespace_objects()

Test idempotent id: a2a3615e-3b5c-3f6a-a2b1-1ba3574ae738

Test listing image metadata namespace objects

test_show_meta_namespace_objects()

Test idempotent id: b1a3674e-3b4c-3f6a-a3b4-1ba3573ca768

Test showing image metadata namespace object

image.v2.admin.test_images_metadefs_namespace_properties module

class MetadataNamespacePropertiesTest(*args, **kwargs)

Bases: BaseV2ImageAdminTest

Test the Metadata definition namespace property basic functionality

test_basic_meta_def_namespace_property()

Test idempotent id: b1a3765e-3a5d-4f6d-a3a7-3ca3476ae768

Test operations of image metadata definition namespace property

image.v2.admin.test_images_metadefs_namespace_tags module

class MetadataNamespaceTagsTest(**args*, ***kwargs*)

Bases: BaseV2ImageAdminTest

Test the Metadata definition namespace tags basic functionality

test_create_list_delete_namespace_tags()

Test idempotent id: a2a3765e-3a6d-4f6d-a3a7-3cc3476aa876

Test creating/listing/deleting image metadata namespace tags

test_create_update_delete_tag()

Test idempotent id: a2a3765e-1a2c-3f6d-a3a7-3cc3466ab875

Test creating/updating/deleting image metadata namespace tag

image.v2.admin.test_images_metadefs_namespaces module

class MetadataNamespacesTest(**args*, ***kwargs*)

Bases: BaseV2ImageAdminTest

Test the Metadata definition Namespaces basic functionality

test_basic_metadata_definition_namespaces()

Test idempotent id: 319b765e-7f3d-4b3d-8b37-3ca3876ee768

Test operations of image metadata definition namespaces

image.v2.admin.test_images_metadefs_resource_types module

class MetadataResourceTypesTest(**args*, ***kwargs*)

Bases: BaseV2ImageAdminTest

Test the Metadata definition resource types basic functionality

test_basic_meta_def_resource_type_association()

Test idempotent id: 6f358a4e-5ef0-11e6-a795-080027d0d606

Test image resource type associations

Module contents

Submodules

image.v2.test_images module

class BasicOperationsImagesTest(**args*, ***kwargs*)

Bases: BaseV2ImageTest

Here we test the basic operations of images

test_deactivate_reactivate_image()

Test idempotent id: 951ebe01-969f-4ea9-9898-8a3f1f442ab0

Test deactivating and reactivating an image

test_delete_image()

Test idempotent id: f848bb94-1c6e-45a4-8726-39e3a5b23535

Test deleting an image by image_id

test_register_upload_get_image_file()

Test idempotent id: 139b765e-7f3d-4b3d-8b37-3ca3876ee318

Here we test these functionalities

Register image, upload the image file, get image and get image file apis

test_update_image()

Test idempotent id: f66891a7-a35c-41a8-b590-a065c2a1caa6

Test updating an image by image_id

class ImageLocationsTest(*args, **kwargs)

Bases: BaseV2ImageTest

test_location_after_upload()

Test idempotent id: 58b0fadc-219d-40e1-b159-1c902cec323a

test_replace_location()

Test idempotent id: bf6e0009-c039-4884-b498-db074caadb10

test_set_location()

Test idempotent id: 37599b8a-d5c0-4590-aee5-73878502be15

test_set_location_bad_scheme()

Test idempotent id: a9a20396-8399-4b36-909d-564949be098f

test_set_location_with_hash()

Test idempotent id: 42d6f7db-c9f5-4bae-9e15-a90262fe445a

test_set_location_with_hash_not_matching()

Test idempotent id: f3ce99c2-9ffb-4b9f-b2cb-876929382553

test_set_location_with_hash_second_matching()

Test idempotent id: 304c8a19-aa86-47dd-a022-ec4c7f433f1b

class ImportImagesTest(*args, **kwargs)

Bases: BaseV2ImageTest

Here we test the import operations for image

test_image_glance_direct_import()

Test idempotent id: 32ca0c20-e16f-44ac-8590-07869c9b4cc2

Test glance-direct import functionalities

Create image, stage image data, import image and verify that import succeeded.

test_image_glance_download_import_bad_endpoint()

Test idempotent id: 77644240-dbbe-4744-ae28-09b2ac12e218

test_image_glance_download_import_bad_missing_image()

Test idempotent id: c7edec8e-24b5-416a-9d42-b3e773bab62c

test_image_glance_download_import_bad_uuid()

Test idempotent id: 36d4b546-64a2-4bb9-bdd0-ba676aa48f2c

test_image_glance_download_import_success()

Test idempotent id: 8876c818-c40e-4b90-9742-31d231616305

test_image_web_download_import()

Test idempotent id: f6feb7a4-b04f-4706-a011-206129f83e62

Test web-download import functionalities

Create image, import image and verify that import succeeded.

test_remote_delete()

Test idempotent id: 44d60544-1524-42f7-8899-315301105dd8

Test image delete against a different worker than stage.

This creates and stages an image against the primary API worker, but then calls delete on a secondary worker (if available) to test that distributed image import works (i.e. proxies the delete request to the proper worker).

test_remote_import()

Test idempotent id: e04761a1-22af-42c2-b8bc-a34a3f12b585

Test image import against a different worker than stage.

This creates and stages an image against the primary API worker, but then calls import on a secondary worker (if available) to test that distributed image import works (i.e. proxies the import request to the proper worker).

class ListSharedImagesTest(**args*, ***kwargs*)

Bases: BaseV2ImageTest

Here we test the listing of a shared image information

test_list_images_param_member_status()

Test idempotent id: 3fa50be4-8e38-4c02-a8db-7811bb780122

Test listing images by member_status and visibility

class ListUserImagesTest(**args*, ***kwargs*)

Bases: BaseV2ImageTest

Here we test the listing of image information

test_get_image_schema()

Test idempotent id: 622b925c-479f-4736-860d-adeaf13bc371

Test to get image schema

test_get_images_schema()

Test idempotent id: 25c8d7b2-df21-460f-87ac-93130bcdc684

Test to get images schema

test_list_hidden_image()

Test idempotent id: d43f3efc-da4c-4af9-b636-868f0c6acedb

test_list_image_param_owner()

Test idempotent id: e9a44b91-31c8-4b40-a332-e0a39ffb4dbb

Test to get images by owner

test_list_images_param_container_format()

Test idempotent id: 9959ca1d-1aa7-4b7a-a1ea-0fff0499b37e

Test to get all images with a specific container_format

test_list_images_param_disk_format()

Test idempotent id: 4a4735a7-f22f-49b6-b0d9-66e1ef7453eb

Test to get all images with disk_format = raw

test_list_images_param_limit()

Test idempotent id: e914a891-3cc8-4b40-ad32-e0a39ffbddbb

Test to get images by limit

test_list_images_param_min_max_size()

Test idempotent id: 4ad8c157-971a-4ba8-aa84-ed61154b1e7f

Test to get all images with min size and max size

test_list_images_param_name()

Test idempotent id: 55c8f5f5-bfed-409d-a6d5-4caeda985d7b

Test to get images by name

test_list_images_param_size()

Test idempotent id: cf1b9a48-8340-480e-af7b-fe7e17690876

Test to get all images by size

test_list_images_param_sort()

Test idempotent id: eeadce49-04e0-43b7-aec7-52535d903e7a

Test listing images sorting in descending order

test_list_images_param_sort_key_dir()

Test idempotent id: 9faaa0c2-c3a5-43e1-8f61-61c54b409a49

Test listing images sorting by size in descending order

test_list_images_param_status()

Test idempotent id: 7fc9e369-0f58-4d05-9aa5-0969e2d59d15

Test to get all active images

test_list_images_param_tag()

Test idempotent id: aa8ac4df-cff9-418b-8d0f-dd9c67b072c9

Test to get images matching a tag

test_list_images_param_visibility()

Test idempotent id: 7a95bb92-d99e-4b12-9718-7bc6ab73e6d2

Test to get all images with visibility = private

test_list_no_params()

Test idempotent id: 1e341d7a-90a9-494c-b143-2cdf2aeb6aee

Simple test to see all fixture images returned

test_list_update_hidden_image()

Test idempotent id: fdb96b81-257b-42ac-978b-ddeefa3760e4

class MultiStoresImportImagesTest(**args*, ***kwargs*)

Bases: BaseV2ImageTest

Test importing image in multiple stores

test_glance_direct_import_image_to_all_stores()

Test idempotent id: bf04ff00-3182-47cb-833a-f1c6767b47fd

Test image is imported in all available stores

Create image, import image to all available stores using glance-direct import method and verify that import succeeded.

test_glance_direct_import_image_to_specific_stores()

Test idempotent id: 82fb131a-dd2b-11ea-aec7-340286b6c574

Test image is imported in all available stores

Create image, import image to specified store(s) using glance-direct import method and verify that import succeeded.

image.v2.test_images_dependency module

class ImageDependencyTests(**args*, ***kwargs*)

Bases: BaseV2ImageTest, BaseV2ComputeTest, ScenarioTest

Test image, instance, and snapshot dependency.

The tests create image and remove the base image that other snapshots were depend on.In OpenStack, images and snapshots should be separate, but in some configurations like Glance with Ceph storage, there were cases where images couldnt be removed. This was fixed in glance store for RBD backend.

- **Dependency scenarios:**
  - image > instance -> snapshot dependency

NOTE: volume -> image dependencies tests are in cinder-tempest-plugin

test_image_volume_server_snapshot_dependency()

Test idempotent id: f0c8a35d-8f8f-443c-8bcb-85a9c0f87d19

Test with image > volume > instance > snapshot dependency.

We are going to perform the following steps in the test: * Create image * Create a bootable volume from Image * Launch an instance from the bootable volume * Take snapshot of the instance which creates the volume snapshot * Delete the image.

This will test the dependency chain of image -> volume -> snapshot.

test_nova_image_snapshot_dependency()

Test idempotent id: d19b0731-e98e-4103-8b0e-02f651b8f586

Test with image > instance > snapshot dependency.

Create instance snapshot and check if we able to delete base image

image.v2.test_images_formats module

class ImagesFormatTest(**args*, ***kwargs*)

Bases: BaseV2ImageTest, BaseV2ComputeTest

test_accept_reject_formats_import()

Test idempotent id: 7c7c2f16-2e97-4dce-8cb4-bc10be031c85

Make sure glance rejects invalid images during conversion.

test_accept_usable_formats()

Test idempotent id: a245fcbe-63ce-4dc1-a1d0-c16d76d9e6df

test_compute_rejects_format_mismatch()

Test idempotent id: ffe21610-e801-4992-9b81-e2d646e2e2e9

Make sure compute rejects any image with a format mismatch.

test_compute_rejects_invalid()

Test idempotent id: f77394bc-81f4-4d54-9f5b-e48f3d6b5376

Make sure compute rejects invalid/insecure images.

load_tests(*loader*, *suite*, *pattern*)

Generate scenarios from the image manifest.

image.v2.test_images_member module

class ImagesMemberTest(**args*, ***kwargs*)

Bases: BaseV2MemberImageTest

Test image members

test_get_image_member()

Test idempotent id: a6ee18b9-4378-465e-9ad9-9a6de58a3287

Test getting image members after the image is accepted

test_get_image_member_schema()

Test idempotent id: 634dcc3f-f6e2-4409-b8fd-354a0bb25d83

Test getting image member schema

test_get_image_members_schema()

Test idempotent id: 6ae916ef-1052-4e11-8d36-b3ae14853cbb

Test getting image members schema

test_image_share_accept()

Test idempotent id: 5934c6ea-27dc-4d6e-9421-eeb5e045494a

Test sharing and accepting an image

test_image_share_reject()

Test idempotent id: d9e83e5f-3524-4b38-a900-22abcb26e90e

Test sharing and rejecting an image

test_remove_image_member()

Test idempotent id: 72989bc7-2268-48ed-af22-8821e835c914

Test removing image members after the image is accepted

image.v2.test_images_member_negative module

class ImagesMemberNegativeTest(**args*, ***kwargs*)

Bases: BaseV2MemberImageTest

Negative tests of image members

test_image_share_invalid_status()

Test idempotent id: b79efb37-820d-4cf0-b54c-308b00cf842c

Test updating image member status to invalid status should fail

test_image_share_owner_cannot_accept()

Test idempotent id: 27002f74-109e-4a37-acd0-f91cd4597967

Test that image owner cant accept image shared to other member

image.v2.test_images_metadefs_schema module

class MetadataSchemaTest(**args*, ***kwargs*)

Bases: BaseV2ImageTest

Test to get image metadata schema

test_get_metadata_namespace_schema()

Test idempotent id: e9e44891-3cb8-3b40-a532-e0a39fea3dab

Test to get image namespace schema

test_get_metadata_namespaces_schema()

Test idempotent id: ffe44891-678b-3ba0-a3e2-e0a3967b3aeb

Test to get image namespaces schema

test_get_metadata_object_schema()

Test idempotent id: dff4a891-b38b-3bf0-a3b2-e03ee67b3a3b

Test to get image object schema

test_get_metadata_objects_schema()

Test idempotent id: dee4a891-b38b-3bf0-a3b2-e03ee67b3a3c

Test to get image objects schema

test_get_metadata_properties_schema()

Test idempotent id: dce4a891-b38b-3bf0-a3b2-e03ee67b3a3e

Test to get image properties schema

test_get_metadata_property_schema()

Test idempotent id: dae4a891-b38b-3bf0-a3b2-e03ee67b3a3d

Test to get image property schema

test_get_metadata_resource_type_schema()

Test idempotent id: fde34891-678b-3b40-ae32-e0a3e67b6beb

Test to get image resource_type schema

test_get_metadata_resources_types_schema()

Test idempotent id: dfe4a891-b38b-3bf0-a3b2-e03ee67b3a3a

Test to get image resource_types schema

test_get_metadata_tag_schema()

Test idempotent id: dde4a891-b38b-3bf0-a3b2-e03ee67b3a3e

Test to get image tag schema

test_get_metadata_tags_schema()

Test idempotent id: cde4a891-b38b-3bf0-a3b2-e03ee67b3a3a

Test to get image tags schema

image.v2.test_images_negative module

class ImagesNegativeTest(*args, **kwargs)

Bases: BaseV2ImageTest

here we have -ve tests for show_image and delete_image api

Tests

** get non-existent image ** get image with image_id=NULL ** get the deleted image ** delete non-existent image ** delete image with image_id=NULL ** delete the deleted image

test_create_image_reserved_property()

Test idempotent id: e3fb7df8-2742-4143-8976-f1b26870f0a0

Attempt to create an image with a reserved property.

Glance bans some internal-use-only properties such that they will cause an image create to fail. Since os_glance_* is banned, we can use a key in that namespace here.

test_delete_image_null_id()

Test idempotent id: 32248db1-ab88-4821-9604-c7c369f1f88c

Delete image with image_id=NULL

test_delete_non_existing_image()

Test idempotent id: 6fe40f1c-57bd-4918-89cc-8500f850f3de

Delete non-existent image

test_delete_protected_image()

Test idempotent id: ab980a34-8410-40eb-872b-f264752f46e5

Create a protected image

test_get_delete_deleted_image()

Test idempotent id: e57fc127-7ba0-4693-92d7-1d8a05ebcba9

Get and delete the deleted image

test_get_image_null_id()

Test idempotent id: ef45000d-0a72-4781-866d-4cb7bf2562ad

Get image with image_id = NULL

test_get_non_existent_image()

Test idempotent id: 668743d5-08ad-4480-b2b8-15da34f81d9f

Get the non-existent image

test_register_with_invalid_container_format()

Test idempotent id: 292bd310-369b-41c7-a7a3-10276ef76753

Create image with invalid container format

Negative tests for invalid data supplied to POST /images

test_register_with_invalid_disk_format()

Test idempotent id: 70c6040c-5a97-4111-9e13-e73665264ce1

Create image with invalid disk format

test_update_image_reserved_property()

Test idempotent id: a0ae75d4-ce9c-4576-b823-aba04c8acabd

Attempt to add a reserved property to an image.

Glance bans some internal-use-only properties such that they will cause a PATCH to fail. Since os_glance_* is banned, we can use a key in that namespace here.

class ImportImagesNegativeTest(**args*, ***kwargs*)

Bases: BaseV2ImageTest

Here we test the import operations for image

test_image_web_download_import_with_bad_url()

Test idempotent id: c52f6a77-f522-4411-8dbe-9d14f2b1ba6b

Test web-download import functionalities

Make sure that web-download with invalid URL fails properly.

image.v2.test_images_tags module

class ImagesTagsTest(**args*, ***kwargs*)

Bases: BaseV2ImageTest

Test image tags

test_update_delete_tags_for_image()

Test idempotent id: 10407036-6059-4f95-a2cd-cbbbee7ed329

Test adding and deleting image tags

image.v2.test_images_tags_negative module

class ImagesTagsNegativeTest(**args*, ***kwargs*)

Bases: BaseV2ImageTest

Negative tests of image tags

test_delete_non_existing_tag()

Test idempotent id: 39c023a2-325a-433a-9eea-649bf1414b19

Delete non existing image tag

test_update_tags_for_non_existing_image()

Test idempotent id: 8cd30f82-6f9a-4c6e-8034-c1b51fba43d9

Update image tag with non existing image

image.v2.test_versions module

class VersionsTest(**args*, ***kwargs*)

Bases: BaseV2ImageTest

Test image versions

test_list_versions()

Test idempotent id: 659ea30a-a17c-4317-832c-0f68ed23c31d

Test listing image versions

Module contents

Submodules

image.base module

class BaseImageTest(**args*, ***kwargs*)

Bases: BaseTestCase

Base test class for Image API tests.

class BaseV2ImageAdminTest(**args*, ***kwargs*)

Bases: *BaseV2ImageTest*

class BaseV2ImageTest(**args*, ***kwargs*)

Bases: *BaseImageTest*

class BaseV2MemberImageTest(**args*, ***kwargs*)

Bases: *BaseV2ImageTest*

Module contents

network

network package

Subpackages

network.admin package

Submodules

network.admin.test_dhcp_agent_scheduler module

class DHCPAgentSchedulersTestJSON(**args*, ***kwargs*)

Bases: BaseAdminNetworkTest

Test network DHCP agent scheduler extension

test_add_remove_network_from_dhcp_agent()

Test idempotent id: a0856713-6549-470c-a656-e97c8df9a14d

Test adding and removing network from a DHCP agent

test_dhcp_port_status_active()

Test idempotent id: f164801e-1dd8-4b8b-b5d3-cc3ac77cfaa5

test_list_dhcp_agent_hosting_network()

Test idempotent id: 5032b1fe-eb42-4a64-8f3b-6e189d8b5c7d

Test Listing DHCP agents hosting a network

test_list_networks_hosted_by_one_dhcp()

Test idempotent id: 30c48f98-e45d-4ffb-841c-b8aad57c7587

Test Listing networks hosted by a DHCP agent

network.admin.test_external_network_extension module

class ExternalNetworksTestJSON(**args*, ***kwargs*)

Bases: BaseAdminNetworkTest

Test external networks

test_create_external_network()

Test idempotent id: 462be770-b310-4df9-9c42-773217e4c8b1

Test creating external network

Create a network as an admin user specifying the external network extension attribute

test_delete_external_networks_with_floating_ip()

Test idempotent id: 82068503-2cf2-4ed4-b3be-ecb89432e4bb

Test deleting external network with unassociated floating ips

Verifies external network can be deleted while still holding (unassociated) floating IPs

test_list_external_networks()

Test idempotent id: 39be4c9b-a57e-4ff9-b7c7-b218e209dfcc

Test listing external networks

test_show_external_networks_attribute()

Test idempotent id: 2ac50ab2-7ebd-4e27-b3ce-a9e399faaea2

Test showing external network attribute

test_update_external_network()

Test idempotent id: 4db5417a-e11c-474d-a361-af00ebef57c5

Test updating external network

Update a network as an admin user specifying the external network extension attribute

network.admin.test_external_networks_negative module

class ExternalNetworksAdminNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseAdminNetworkTest

Negative tests of external network

test_create_port_with_precreated_floatingip_as_fixed_ip()

Test idempotent id: d402ae6c-0be0-4d8e-833b-a738895d98d0

Test creating port with precreated floating ip as fixed ip

NOTE: External networks can be used to create both floating-ip as well as instance-ip. So, creating an instance-ip with a value of a pre-created floating-ip should be denied.

network.admin.test_floating_ips_admin_actions module

class FloatingIPAdminTestJSON(**args*, ***kwargs*)

Bases: BaseAdminNetworkTest

Test floating ips

test_create_list_show_floating_ip_with_tenant_id_by_admin()

Test idempotent id: 32727cc3-abe2-4485-a16e-48f2d54c14f2

Verify if admin can create/list/show floating ip with tenant id

test_list_floating_ips_from_admin_and_nonadmin()

Test idempotent id: 64f2100b-5471-4ded-b46c-ddeeeb4f231b

Test listing floating ips from admin and non admin users

This test performs below operations: 1. Create couple floating ips for admin and non-admin users. 2. Verify if admin can access all floating ips including other user and non-admin user can only access its own floating ips.

network.admin.test_metering_extensions module

class MeteringIpV6TestJSON(**args*, ***kwargs*)

Bases: *MeteringTestJSON*

class MeteringTestJSON(**args*, ***kwargs*)

Bases: BaseAdminNetworkTest

Tests the following operations in the Neutron API:

List, Show, Create, Delete Metering labels List, Show, Create, Delete Metering labels rules

test_create_delete_metering_label_rule_with_filters()

Test idempotent id: f4d547cd-3aee-408f-bf36-454f8825e045

Verifies creating and deleting metering label rule with filters

test_create_delete_metering_label_with_filters()

Test idempotent id: ec8e15ff-95d0-433b-b8a6-b466bddb1e50

Verifies creating and deleting metering label with filters

test_list_metering_label_rules()

Test idempotent id: cc832399-6681-493b-9d79-0202831a1281

Verifies listing metering label rules

test_list_metering_labels()

Test idempotent id: e2fb2f8c-45bf-429a-9f17-171c70444612

Verify listing metering labels

test_show_metering_label()

Test idempotent id: 30abb445-0eea-472e-bd02-8649f54a5968

Verifies the details of a metering label

test_show_metering_label_rule()

Test idempotent id: b7354489-96ea-41f3-9452-bace120fb4a7

Verifies the metering details of a rule

network.admin.test_ports module

class PortsAdminExtendedAttrsIpV6TestJSON(**args*, ***kwargs*)

Bases: *PortsAdminExtendedAttrsTestJSON*

class PortsAdminExtendedAttrsTestJSON(**args*, ***kwargs*)

Bases: BaseAdminNetworkTest

Test extended attributes of ports

test_create_port_binding_ext_attr()

Test idempotent id: 8e8569c1-9ac7-44db-8bc1-f5fb2814f29b

Test creating port with extended attribute

test_list_ports_binding_ext_attr()

Test idempotent id: 1c82a44a-6c6e-48ff-89e1-abe7eaf8f9f8

Test updating and listing ports extended attribute

test_show_port_binding_ext_attr()

Test idempotent id: b54ac0ff-35fc-4c79-9ca3-c7dbd4ea4f13

Test showing ports extended attribute

test_update_port_binding_ext_attr()

Test idempotent id: 6f6c412c-711f-444d-8502-0ac30fbf5dd5

Test updating ports extended attribute

network.admin.test_routers module

class RoutersAdminTest(**args*, ***kwargs*)

Bases: BaseAdminNetworkTest

Test routers operation supported by admin

test_create_router_set_gateway_with_fixed_ip()

Test idempotent id: cbe42f84-04c2-11e7-8adb-fa163e4fa634

Test creating router setting gateway with fixed ip

test_create_router_setting_project_id()

Test idempotent id: e54dd3a3-4352-4921-b09d-44369ae17397

test_create_router_with_default_snat_value()

Test idempotent id: 847257cc-6afd-4154-b8fb-af49f5670ce8

Create a router with default snat rule

test_create_router_with_snat_explicit()

Test idempotent id: ea74068d-09e9-4fd7-8995-9b6a1ace920f

Test creating router with specified enable_snat value

test_update_router_reset_gateway_without_snat()

Test idempotent id: f2faf994-97f4-410b-a831-9bc977b64374

Test updating routers gateway to be with snat not enabled

test_update_router_set_gateway()

Test idempotent id: 6cc285d8-46bf-4f36-9b1a-783e3008ba79

Test updating routers gateway info

test_update_router_set_gateway_with_snat_explicit()

Test idempotent id: b386c111-3b21-466d-880c-5e72b01e1a33

Test setting routers gateway with snat enabled

test_update_router_set_gateway_without_snat()

Test idempotent id: 96536bc7-8262-4fb2-9967-5c46940fa279

Test setting routers gateway with snat not enabled

test_update_router_unset_gateway()

Test idempotent id: ad81b7ee-4f81-407b-a19c-17e623f763e8

Test unsetting routers gateway

class RoutersIpV6AdminTest(**args*, ***kwargs*)

Bases: *RoutersAdminTest*

network.admin.test_routers_dvr module

class RoutersTestDVR(**args*, ***kwargs*)

Bases: BaseAdminNetworkTest

test_centralized_router_creation()

Test idempotent id: 8a0a72b4-7290-4677-afeb-b4ffe37bc352

Test centralized router creation

Test uses administrative credentials to creates a CVR (Centralized Virtual Routing) router using the distributed=False.

Acceptance The router is created and the distributed attribute is set to False, thus making it a Centralized Virtual Router as opposed to a Distributed Virtual Router

test_centralized_router_update_to_dvr()

Test idempotent id: acd43596-c1fb-439d-ada8-31ad48ae3c2e

Test centralized router update

Test uses administrative credentials to creates a CVR (Centralized Virtual Routing) router using the distributed=False. Then it will update the router distributed attribute to True

Acceptance The router is created and the distributed attribute is set to False. Once the router is updated, the distributed attribute will be set to True

test_distributed_router_creation()

Test idempotent id: 08a2a0a8-f1e4-4b34-8e30-e522e836c44e

Test distributed router creation

Test uses administrative credentials to creates a DVR (Distributed Virtual Routing) router using the distributed=True.

Acceptance The router is created and the distributed attribute is set to True

network.admin.test_routers_negative module

class RoutersAdminNegativeIpV6Test(**args*, ***kwargs*)

Bases: *RoutersAdminNegativeTest*

class RoutersAdminNegativeTest(**args*, ***kwargs*)

Bases: BaseAdminNetworkTest

Admin negative tests of routers

test_router_set_gateway_used_ip_returns_409()

Test idempotent id: 7101cc02-058a-11e7-93e1-fa163e4fa634

Test creating router with gateway set to used ip should fail

Module contents

Submodules

network.base module

class BaseAdminNetworkTest(**args*, ***kwargs*)

Bases: *BaseNetworkTest*

class BaseNetworkTest(**args*, ***kwargs*)

Bases: BaseTestCase

Base class for the Neutron tests.

Per the Neutron API Guide, API v1.x was removed from the source code tree (docs.openstack.org/api/openstack-network/2.0/content/Overview-d1e71.html) Therefore, v2.x of the Neutron API is assumed. It is also assumed that the following options are defined in the [network] section of etc/tempest.conf:

project_network_cidr with a block of cidrs from which smaller blocks can be allocated for project networks

project_network_mask_bits with the mask bits to be used to partition the block defined by project-network_cidr

Finally, it is assumed that the following option is defined in the [service_available] section of etc/tempest.conf

neutron as True

network.base_security_groups module

class BaseSecGroupTest(**args*, ***kwargs*)

Bases: BaseNetworkTest

network.test_agent_management_negative module

class AgentManagementNegativeTest(**args*, ***kwargs*)

Bases: BaseNetworkTest

test_list_agents_non_admin()

Test idempotent id: e335be47-b9a1-46fd-be30-0874c0b751e6

Validate that non-admin user cannot list agents.

network.test_allowed_address_pair module

class AllowedAddressPairIpV6TestJSON(**args*, ***kwargs*)

Bases: *AllowedAddressPairTestJSON*

class AllowedAddressPairTestJSON(*args, **kwargs)

Bases: BaseNetworkTest

Tests the Neutron Allowed Address Pair API extension

The following API operations are tested with this extension:

create port list ports update port show port

v2.0 of the Neutron API is assumed. It is also assumed that the following options are defined in the [network-feature-enabled] section of etc/tempest.conf

api_extensions

test_create_list_port_with_address_pair()

Test idempotent id: 86c3529b-1231-40de-803c-00e40882f043

Create and list port with allowed address pair attribute

test_update_port_with_address_pair()

Test idempotent id: 9599b337-272c-47fd-b3cf-509414414ac4

Update port with allowed address pair

test_update_port_with_cidr_address_pair()

Test idempotent id: 4d6d178f-34f6-4bff-a01c-0a2f8fe909e4

Update allowed address pair with cidr

test_update_port_with_multiple_ip_mac_address_pair()

Test idempotent id: b3f20091-6cd5-472b-8487-3516137df933

Update allowed address pair port with multiple ip and mac

network.test_dhcp_ipv6 module

class NetworksTestDHCPv6(*args, **kwargs)

Bases: BaseNetworkTest

test_dhcp_stateful()

Test idempotent id: 4ab211a0-276f-4552-9070-51e27f58fecf

Test creating port when setting stateful for subnets

NOTE: With all options below, DHCPv6 shall allocate address from subnet pool to port.

test_dhcp_stateful_fixedips()

Test idempotent id: 51a5e97f-f02e-4e4e-9a17-a69811d300e3

Test creating port with fixed ip when setting stateful for subnets

NOTE: With all options below, port shall be able to get requested IP from fixed IP range not depending on DHCP stateful (not SLAAC!) settings configured.

test_dhcp_stateful_fixedips_duplicate()

Test idempotent id: 57b8302b-cba9-4fbb-8835-9168df029051

Test creating port with duplicate fixed ip

NOTE: When port gets IP address from fixed IP range it shall be checked if its not duplicate.

test_dhcp_stateful_fixedips_outrange()

Test idempotent id: 98244d88-d990-4570-91d4-6b25d70d08af

Test creating port with fixed ip that is not in the range

NOTE: When port gets IP address from fixed IP range it shall be checked if its from subnets range.

test_dhcp_stateful_router()

Test idempotent id: e98f65db-68f4-4330-9fea-abd8c5192d4d

Test creating router with dhcp stateful

NOTE: With all options below the router interface shall receive DHCPv6 IP address from allocation pool.

test_dhcpv6_64_subnets()

Test idempotent id: 4256c61d-c538-41ea-9147-3c450c36669e

Test eui64 ip when setting slaac and stateless for subnets

NOTE: When one IPv6 subnet configured with dnsmasq SLAAC or DHCP stateless and other IPv4 is with DHCP of IPv4, port shall receive EUI-64 IP addresses from first subnet and IPv4 DHCP address from second one. Order of subnet creating should be unimportant.

test_dhcpv6_invalid_options()

Test idempotent id: 81f18ef6-95b5-4584-9966-10d480b7496a

Different configurations for radvd and dnsmasq are not allowed

test_dhcpv6_stateless_eui64()

Test idempotent id: e5517e62-6f16-430d-a672-f80875493d4c

Test eui64 ip when setting slaac and statelss for subnet

NOTE: When subnets configured with RAs SLAAC (AOM=100) and DHCP stateless (AOM=110) both for radvd and dnsmasq, port shall receive IP address calculated from its MAC.

test_dhcpv6_stateless_no_ra()

Test idempotent id: ae2f4a5d-03ff-4c42-a3b0-ce2fcb7ea832

Test eui64 ip when setting stateless and no radvd for subnets

NOTE: When subnets configured with dnsmasq SLAAC and DHCP stateless and there is no radvd, port shall receive IP address calculated from its MAC and mask of subnet.

test_dhcpv6_stateless_no_ra_no_dhcp()

Test idempotent id: 21635b6f-165a-4d42-bf49-7d195e47342f

Test eui64 ip when setting no radvd and no dnsmasq for subnets

NOTE: If no radvd option and no dnsmasq option is configured port shall receive IP from fixed IPs list of subnet.

test_dhcpv6_two_subnets()

Test idempotent id: 4544adf7-bb5f-4bdc-b769-b3e77026cef2

Test eui64 ip when creating port under network with two subnets

NOTE: When one IPv6 subnet configured with dnsmasq SLAAC or DHCP stateless and other IPv6 is with DHCP stateful, port shall receive EUI-64 IP addresses from first subnet and DHCP address from second one. Order of subnet creating should be unimportant.

network.test_extensions module

class ExtensionsTestJSON(**args*, ***kwargs*)

Bases: BaseNetworkTest

Tests the following operations in the Neutron API:

List all available extensions

v2.0 of the Neutron API is assumed. It is also assumed that api-extensions option is defined in the [network-feature-enabled] section of etc/tempest.conf.

test_list_show_extensions()

Test idempotent id: ef28c7e6-e646-4979-9d67-deb207bc5564

List available extensions and show the detail of each extension

network.test_extra_dhcp_options module

class ExtraDHCPOptionsIpV6TestJSON(**args*, ***kwargs*)

Bases: *ExtraDHCPOptionsTestJSON*

class ExtraDHCPOptionsTestJSON(**args*, ***kwargs*)

Bases: BaseNetworkTest

Tests the following operations with the Extra DHCP Options:

port create port list port show port update

v2.0 of the Neutron API is assumed. It is also assumed that the Extra DHCP Options extension is enabled in the [network-feature-enabled] section of etc/tempest.conf

test_create_list_port_with_extra_dhcp_options()

Test idempotent id: d2c17063-3767-4a24-be4f-a23dbfa133c9

Test creating a port with Extra DHCP Options and list those

test_update_show_port_with_extra_dhcp_options()

Test idempotent id: 9a6aebf4-86ee-4f47-b07a-7f7232c55607

Test updating port with extra DHCP options and show that port

network.test_floating_ips module

class FloatingIPTestJSON(**args*, ***kwargs*)

Bases: BaseNetworkTest

Tests the following operations in the Neutron API:

Create a Floating IP Update a Floating IP Delete a Floating IP List all Floating IPs Show Floating IP details Associate a Floating IP with a port and then delete that port Associate a Floating IP with a port and then with a port on another router

v2.0 of the Neutron API is assumed. It is also assumed that the following options are defined in the [network] section of etc/tempest.conf:

public_network_id which is the id for the external network present

test_create_floating_ip_specifying_a_fixed_ip_address()

Test idempotent id: 36de4bd0-f09c-43e3-a8e1-1decc1ffd3a5

Test creating floating ip with specified fixed ip

test_create_list_show_update_delete_floating_ip()

Test idempotent id: 62595970-ab1c-4b7f-8fcc-fddfe55e8718

Test create/list/show/update/delete floating ip

test_create_update_floatingip_with_port_multiple_ip_address()

Test idempotent id: 45c4c683-ea97-41ef-9c51-5e9802f2f3d7

Test updating floating ips fixed_ips to another ip of same port

First we create a port with 2 fixed ips, then we create a floating ip with one of the fixed ips, and then we update the floating ip to another fixed ip of that port.

test_floating_ip_delete_port()

Test idempotent id: e1f6bffd-442f-4668-b30e-df13f2705e77

Test deleting floating ips port

1. Create a floating ip
2. Create a port
3. Update the floating ips port_id to the created port
4. Delete the port
5. Verify that the port details are cleared from the floating ip

test_floating_ip_update_different_router()

Test idempotent id: 1bb2f731-fe5a-4b8c-8409-799ade1bed4d

Test associating a floating ip to a port on different router

network.test_floating_ips_negative module

class FloatingIPNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseNetworkTest

Test the following negative operations for floating ips:

Create floatingip with a port that is unreachable to external network Create floatingip in private network Associate floatingip with port that is unreachable to external network

test_associate_floatingip_port_ext_net_unreachable()

Test idempotent id: 6b3b8797-6d43-4191-985c-c48b773eb429

Associate floating ip to port with unreachable external network

test_create_floatingip_in_private_network()

Test idempotent id: 50b9aeb4-9f0b-48ee-aa31-fa955a48ff54

Test creating floating in private network

test_create_floatingip_with_port_ext_net_unreachable()

Test idempotent id: 22996ea8-4a81-4b27-b6e1-fa5df92fa5e8

Creating floating ip when ports external network is unreachable

network.test_networks module

class BaseNetworkTestResources(**args*, ***kwargs*)

Bases: BaseNetworkTest

Test networks

class BulkNetworkOpsIpV6Test(**args*, ***kwargs*)

Bases: *BulkNetworkOpsTest*

class BulkNetworkOpsTest(**args*, ***kwargs*)

Bases: BaseNetworkTest

Tests the following operations in the Neutron API:

bulk network creation bulk subnet creation bulk port creation list projects networks

v2.0 of the Neutron API is assumed. It is also assumed that the following options are defined in the [network] section of etc/tempest.conf:

project_network_cidr with a block of cidrs from which smaller blocks can be allocated for project networks

project_network_mask_bits with the mask bits to be used to partition the block defined by project-network_cidr

test_bulk_create_delete_network()

Test idempotent id: d4f9024d-1e28-4fc1-a6b1-25dbc6fa11e2

Verify creating and deleting multiple networks in one request

test_bulk_create_delete_port()

Test idempotent id: 48037ff2-e889-4c3b-b86a-8e3f34d2d060

Verify creating and deleting multiple ports in one request

test_bulk_create_delete_subnet()

Test idempotent id: 8936533b-c0aa-4f29-8e53-6cc873aec489

Verify creating and deleting multiple subnets in one request

class NetworksIpV6Test(**args*, ***kwargs*)

Bases: *NetworksTest*

test_create_delete_subnet_with_default_gw()

Test idempotent id: ebb4fd95-524f-46af-83c1-0305b239338f

Verify creating and deleting subnet without specified gateway

test_create_delete_subnet_with_gw()

Test idempotent id: e41a4888-65a6-418c-a095-f7c2ef4ad59a

Verify creating and deleting subnet with gateway

test_create_list_subnet_with_no_gw64_one_network()

Test idempotent id: a9653883-b2a4-469b-8c3c-4518430a7e55

Verify subnets with and without gateway are in one network

First we create a network, then we create one ipv6 subnet with gateway and one ipv4 subnet without gateway, the two subnets should be in the same network

class NetworksIpV6TestAttrs(**args*, ***kwargs*)

Bases: *BaseNetworkTestResources*

test_create_delete_slaac_subnet_with_ports()

Test idempotent id: 88554555-ebf8-41ef-9300-4926d45e06e9

Test deleting subnet with SLAAC ports

Create subnet with SLAAC, create ports in network and then you shall be able to delete subnet without port deletion. But you still can not delete the network.

test_create_delete_stateless_subnet_with_ports()

Test idempotent id: 2de6ab5a-fcf0-4144-9813-f91a940291f1

Test deleting subnet with DHCPv6 stateless ports

Create subnet with DHCPv6 stateless, create ports in network and then you shall be able to delete subnet without port deletion. But you still can not delete the network.

test_create_delete_subnet_with_v6_attributes_slaac()

Test idempotent id: 176b030f-a923-4040-a755-9dc94329e60c

Test create/delete subnet with ipv6 attributes slaac

test_create_delete_subnet_with_v6_attributes_stateful()

Test idempotent id: da40cd1b-a833-4354-9a85-cd9b8a3b74ca

Test create/delete subnet with ipv6 attributes stateful

test_create_delete_subnet_with_v6_attributes_stateless()

Test idempotent id: 7d410310-8c86-4902-adf9-865d08e31adb

Test create/delete subnet with ipv6 attributes stateless

class NetworksTest(**args*, ***kwargs*)

Bases: *BaseNetworkTestResources*

Tests the following operations in the Neutron API:

create a network for a project list projects networks show a project network details create a subnet for a project list projects subnets show a project subnet details network update subnet update delete a network also deletes its subnets list external networks

All subnet tests are run once with ipv4 and once with ipv6.

v2.0 of the Neutron API is assumed. It is also assumed that the following options are defined in the [network] section of etc/tempest.conf:

project_network_cidr with a block of cidrs from which smaller blocks can be allocated for project ipv4 subnets

project_network_v6_cidr is the equivalent for ipv6 subnets

project_network_mask_bits with the mask bits to be used to partition the block defined by project_network_cidr

project_network_v6_mask_bits is the equivalent for ipv6 subnets

test_create_delete_subnet_all_attributes()

Test idempotent id: a4d9ec4c-0306-4111-a75c-db01a709030b

Verify create/delete subnets all attributes

test_create_delete_subnet_with_allocation_pools()

Test idempotent id: bec949c4-3147-4ba6-af5f-cd2306118404

Verify creating and deleting subnet with allocation pools

test_create_delete_subnet_with_dhcp_enabled()

Test idempotent id: 94ce038d-ff0a-4a4c-a56b-09da3ca0b55d

Verify create/delete subnet with dhcp enabled

test_create_delete_subnet_with_gw()

Test idempotent id: 9393b468-186d-496d-aa36-732348cd76e7

Verify creating and deleting subnet with gateway

test_create_delete_subnet_with_gw_and_allocation_pools()

Test idempotent id: 8217a149-0c6c-4cfb-93db-0486f707d13f

Verify create/delete subnet with gateway and allocation pools

test_create_delete_subnet_with_host_routes_and_dns_nameservers()

Test idempotent id: d830de0a-be47-468f-8f02-1fd996118289

Verify create/delete subnet with host routes and name servers

test_create_delete_subnet_without_gateway()

Test idempotent id: d2d596e2-8e76-47a9-ac51-d4648009f4d3

Verify creating and deleting subnet without gateway

test_create_update_delete_network_subnet()

Test idempotent id: 0e269138-0da6-4efc-a46d-578161e7b221

Verify creating, updating and deleting network subnet

test_create_update_network_description()

Test idempotent id: c72c1c0c-2193-4aca-ccc4-b1442640bbbb

Verify creating and updating networks description

test_delete_network_with_subnet()

Test idempotent id: f04f61a9-b7f3-4194-90b2-9bcf660d1bfe

Verify deleting network with subnet

test_external_network_visibility()

Test idempotent id: af774677-42a9-4e4b-bb58-16fe6a5bc1ec

Verify external networks visibility

test_list_networks()

Test idempotent id: f7ffdeda-e200-4a7a-bcbe-05716e86bf43

Verify the network exists in the list of all networks

test_list_networks_fields()

Test idempotent id: 6ae6d24f-9194-4869-9c85-c313cb20e080

Verify specific fields of the networks

test_list_subnets()

Test idempotent id: db68ba48-f4ea-49e9-81d1-e367f6d0b20a

Verify the subnet exists in the list of all subnets

test_list_subnets_fields()

Test idempotent id: 842589e3-9663-46b0-85e4-7f01273b0412

Verify specific fields of subnets

test_show_network()

Test idempotent id: 2bf13842-c93f-4a69-83ed-717d2ec3b44e

Verify the details of a network

test_show_network_fields()

Test idempotent id: 867819bb-c4b6-45f7-acf9-90edcf70aa5e

Verify specific fields of a network

test_show_subnet()

Test idempotent id: bd635d81-6030-4dd1-b3b9-31ba0cfdf6cc

Verify the details of a subnet

test_show_subnet_fields()

Test idempotent id: 270fff0b-8bfc-411f-a184-1e8fd35286f0

Verify specific fields of a subnet

test_update_subnet_gw_dns_host_routes_dhcp()

Test idempotent id: 3d3852eb-3009-49ec-97ac-5ce83b73010a

Verify updating subnets gateway/nameserver/routes/dhcp

network.test_networks_negative module

class NetworksNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseNetworkTest

Negative tests of network

test_create_port_on_non_existent_network()

Test idempotent id: 13d3b106-47e6-4b9b-8d53-dae947f092fe

Test creating port on non existent network

test_delete_non_existent_network()

Test idempotent id: 03795047-4a94-4120-a0a1-bd376e36fd4e

Test deleting non existent network

test_delete_non_existent_port()

Test idempotent id: 49ec2bbd-ac2e-46fd-8054-798e679ff894

Test deleting non existent port

test_delete_non_existent_subnet()

Test idempotent id: a176c859-99fb-42ec-a208-8a85b552a239

Test deleting non existent subnet

test_show_non_existent_network()

Test idempotent id: 9293e937-824d-42d2-8d5b-e985ea67002a

Test showing non existent network

test_show_non_existent_port()

Test idempotent id: a954861d-cbfd-44e8-b0a9-7fab111f235d

Test showing non existent port

test_show_non_existent_subnet()

Test idempotent id: d746b40c-5e09-4043-99f7-cba1be8b70df

Test showing non existent subnet

test_update_non_existent_network()

Test idempotent id: 98bfe4e3-574e-4012-8b17-b2647063de87

Test updating non existent network

test_update_non_existent_port()

Test idempotent id: cf8eef21-4351-4f53-adcd-cc5cb1e76b92

Test updating non existent port

test_update_non_existent_subnet()

Test idempotent id: 1cc47884-ac52-4415-a31c-e7ce5474a868

Test updating non existent subnet

network.test_ports module

class PortsIpV6TestJSON(**args*, ***kwargs*)

Bases: *PortsTestJSON*

class PortsTestJSON(**args*, ***kwargs*)

Bases: BaseSecGroupTest

Test the following operations for ports:

port create port delete port list port show port update

test_create_bulk_port()

Test idempotent id: 67f1b811-f8db-43e2-86bd-72c074d4a42c

Test creating multiple ports in a single request

test_create_port_in_allowed_allocation_pools()

Test idempotent id: 0435f278-40ae-48cb-a404-b8a087bc09b1

Test creating port in allowed allocation pools

test_create_port_with_no_securitygroups()

Test idempotent id: 4179dcb9-1382-4ced-84fe-1b91c54f5735

Test creating port without security groups

test_create_show_delete_port_user_defined_mac()

Test idempotent id: 13e95171-6cbd-489c-9d7c-3f9c58215c18

Test creating port with user defined mac address

test_create_update_delete_port()

Test idempotent id: c72c1c0c-2193-4aca-aaa4-b1442640f51c

Test creating, updating and deleting port

test_create_update_port_with_second_ip()

Test idempotent id: 63aeadd4-3b49-427f-a3b1-19ca81f06270

Test updating port from 2 fixed ips to 1 fixed ip and vice versa

test_list_ports()

Test idempotent id: cf95b358-3e92-4a29-a148-52445e1ac50e

Verify the port exists in the list of all ports

test_list_ports_fields()

Test idempotent id: ff7f117f-f034-4e0e-abff-ccef05c454b4

Verify specific fields of ports

test_port_list_filter_by_ip()

Test idempotent id: e7fe260b-1e79-4dd3-86d9-bec6a7959fc5

Test listing ports filtered by ip

test_port_list_filter_by_ip_substr()

Test idempotent id: 79895408-85d5-460d-94e7-9531c5fd9123

Test listing ports filtered by part of ip address string

test_port_list_filter_by_router_id()

Test idempotent id: 5ad01ed0-0e6e-4c5d-8194-232801b15c72

Test listing ports filtered by router id

test_show_port()

Test idempotent id: c9a685bd-e83f-499c-939f-9f7863ca259f

Verify the details of port

test_show_port_fields()

Test idempotent id: 45fcdaf2-dab0-4c13-ac6c-fcddfb579dbd

Verify specific fields of a port

test_update_port_with_security_group_and_extra_attributes()

Test idempotent id: 58091b66-4ff4-4cc1-a549-05d60c7acd1a

Test updating ports security_group along with extra attributes

First we create a port with one security group, and then we update the ports security_group, in the same update request we also change the ports fixed ips.

test_update_port_with_two_security_groups_and_extra_attributes()

Test idempotent id: edf6766d-3d40-4621-bc6e-2521a44c257d

Test updating port with two security_groups and extra attributes

First we create a port with one security group, and then we update the port to two security_groups, in the same update request we also change the ports fixed ips.

network.test_routers module

class RoutersIpV6Test(**args*, ***kwargs*)

Bases: *RoutersTest*

class RoutersTest(**args*, ***kwargs*)

Bases: BaseNetworkTest

Test routers

test_add_multiple_router_interfaces()

Test idempotent id: 802c73c9-c937-4cef-824b-2191e24a6aab

Test adding multiple router interfaces

test_add_remove_router_interface_with_port_id()

Test idempotent id: 2b7d2f37-6748-4d78-92e5-1d590234f0d5

Test adding and removing router interface with port id

test_add_remove_router_interface_with_subnet_id()

Test idempotent id: b42e6e39-2e37-49cc-a6f4-8467e940900a

Test adding and removing router interface with subnet id

test_create_show_list_update_delete_router()

Test idempotent id: f64403e2-8483-4b34-8ccd-b09a87bcc68c

Test create/show/list/update/delete of a router

test_router_interface_port_update_with_fixed_ip()

Test idempotent id: 96522edf-b4b5-45d9-8443-fa11c26e6eff

Test updating router interface ports fixed ip

test_update_delete_extra_route()

Test idempotent id: c86ac3a8-50bd-4b00-a6b8-62af84a0765c

Test updating and deleting router with extra route

test_update_router_admin_state()

Test idempotent id: a8902683-c788-4246-95c7-ad9c6d63a4d9

Test updating routers admin state

network.test_routers_negative module

class DvrRoutersNegativeTest(**args*, ***kwargs*)

Bases: BaseNetworkTest

Negative tests of DVR router

test_router_create_tenant_distributed_returns_forbidden()

Test idempotent id: 4990b055-8fc7-48ab-bba7-aa28beaad0b9

Non admin user is not allowed to create distributed router

class RoutersNegativeIpV6Test(**args*, ***kwargs*)

Bases: *RoutersNegativeTest*

class RoutersNegativeTest(**args*, ***kwargs*)

Bases: BaseNetworkTest

Negative tests of routers

test_add_router_interfaces_on_overlapping_subnets_returns_400()

Test idempotent id: 957751a3-3c68-4fa2-93b6-eb52ea10db6e

Test adding router interface which is on overlapping subnets

test_delete_non_existent_router_returns_404()

Test idempotent id: c7edc5ad-d09d-41e6-a344-5c0c31e2e3e4

Test deleting non existent router

test_router_add_gateway_invalid_network_returns_404()

Test idempotent id: 37a94fc0-a834-45b9-bd23-9a81d2fd1e22

Test adding gateway with invalid network for router

test_router_add_gateway_net_not_external_returns_400()

Test idempotent id: 11836a18-0b15-4327-a50b-f0d9dc66bddd

Test adding gateway with not external network for router

test_router_remove_interface_in_use_returns_409()

Test idempotent id: 04df80f9-224d-47f5-837a-bf23e33d1c20

Test removing in-use interface from router

test_show_non_existent_router_returns_404()

Test idempotent id: c2a70d72-8826-43a7-8208-0209e6360c47

Test showing non existent router

test_update_non_existent_router_returns_404()

Test idempotent id: b23d1569-8b0c-4169-8d4b-6abd34fad5c7

Test updating non existent router

network.test_security_groups module

class SecGroupIPv6Test(**args*, ***kwargs*)

Bases: *SecGroupTest*

class SecGroupTest(**args*, ***kwargs*)

Bases: BaseSecGroupTest

Test security groups

test_create_list_update_show_delete_security_group()

Test idempotent id: bfd128e5-3c92-44b6-9d66-7fe29d22c802

Verify create/list/update/show/delete of security group

test_create_security_group_rule_with_additional_args()

Test idempotent id: 87dfbcf9-1849-43ea-b1e4-efa3eeae9f71

Verify security group rule with additional arguments works.

direction:ingress, ethertype:[IPv4/IPv6], protocol:tcp, port_range_min:77, port_range_max:77

test_create_security_group_rule_with_icmp_type_code()

Test idempotent id: c9463db8-b44d-4f52-b6c0-8dbda99f26ce

Verify security group rule for icmp protocol works.

Specify icmp type (port_range_min) and icmp code (port_range_max) with different values. A separate testcase is added for icmp protocol as icmp validation would be different from tcp/udp.

test_create_security_group_rule_with_protocol_integer_value()

Test idempotent id: 0a307599-6655-4220-bebc-fd70c64f2290

Verify creating security group rule with the integer protocol value

arguments : protocol: 17

test_create_security_group_rule_with_remote_group_id()

Test idempotent id: c2ed2deb-7a0c-44d8-8b4c-a5825b5c310b

Verify creating security group rule with remote_group_id works

test_create_security_group_rule_with_remote_ip_prefix()

Test idempotent id: 16459776-5da2-4634-bce4-4b55ee3ec188

Verify creating security group rule with remote_ip_prefix works

test_create_show_delete_security_group_rule()

Test idempotent id: cfb99e0e-7410-4a3d-8a0c-959a63ee77e9

Test create/show/delete of security group rule

test_delete_security_group_clear_associated_rules()

Test idempotent id: fd1ea1c5-eedc-403f-898d-2b562e853f2e

Verify delete security group.

its associated security group rules are also deleted

test_list_security_groups()

Test idempotent id: e30abd17-fef9-4739-8617-dc26da88e686

Verify that default security group exist

network.test_security_groups_negative module

class NegativeSecGroupIPv6Test(**args*, ***kwargs*)

Bases: *NegativeSecGroupTest*

test_create_security_group_rule_wrong_ip_prefix_version()

Test idempotent id: 7607439c-af73-499e-bf64-f687fd12a842

Test creating security group rule with wrong ip prefix version

class NegativeSecGroupTest(**args*, ***kwargs*)

Bases: BaseSecGroupTest

Negative tests of security groups

test_create_additional_default_security_group_fails()

Test idempotent id: 2323061e-9fbf-4eb0-b547-7e8fafc90849

Test creating additional default security group

Create security group named default, it should be failed.

test_create_duplicate_security_group_rule_fails()

Test idempotent id: 8fde898f-ce88-493b-adc9-4e4692879fc5

Test creating duplicate security group rule

Create duplicate security group rule, it should fail.

test_create_security_group_rule_with_bad_ethertype()

Test idempotent id: 5666968c-fff3-40d6-9efc-df1c8bd01abb

Test creating security group rule with bad bad ethertype

test_create_security_group_rule_with_bad_protocol()

Test idempotent id: 981bdc22-ce48-41ed-900a-73148b583958

Test creating security group rule with bad protocol

test_create_security_group_rule_with_bad_remote_ip_prefix()

Test idempotent id: 5f8daf69-3c5f-4aaa-88c9-db1d66f68679

Test creating security group rule with bad remote ip prefix

test_create_security_group_rule_with_invalid_ports()

Test idempotent id: 0d9c7791-f2ad-4e2f-ac73-abf2373b0d2d

Test creating security group rule with invalid ports

test_create_security_group_rule_with_non_existent_remote_groupid()

Test idempotent id: 4bf786fd-2f02-443c-9716-5b98e159a49a

Creating security group rule with non existent remote group id

test_create_security_group_rule_with_non_existent_security_group()

Test idempotent id: be308db6-a7cf-4d5c-9baf-71bafd73f35e

Creating security group rules with not existing security group

test_create_security_group_rule_with_remote_ip_and_group()

Test idempotent id: b5c4b247-6b02-435b-b088-d10d45650881

Test creating security group rule with remote ip and group

test_create_security_group_update_name_default()

Test idempotent id: 966e2b96-023a-11e7-a9e4-fa163e4fa634

Test updating security groups name to default

Update security group name to default, it should be failed.

test_delete_non_existent_security_group()

Test idempotent id: 1f1bb89d-5664-4956-9fcd-83ee0fa603df

Test deleting non existent security group

test_show_non_existent_security_group()

Test idempotent id: 424fd5c3-9ddc-486a-b45f-39bf0c820fc6

Test showing non existent security group

test_show_non_existent_security_group_rule()

Test idempotent id: 4c094c09-000b-4e41-8100-9617600c02a6

Test showing non existent security group rule

network.test_service_providers module

class ServiceProvidersTest(**args*, ***kwargs*)

Bases: BaseNetworkTest

Test network service providers

test_service_providers_list()

Test idempotent id: 2cbbeea9-f010-40f6-8df5-4eaa0c918ea6

Test listing network service providers

network.test_subnetpools_extensions module

class SubnetPoolsTestJSON(**args*, ***kwargs*)

Bases: BaseNetworkTest

Tests the following operations in the subnetpools API:

Create a subnet pool. Update a subnet pool. Delete a subnet pool. Lists subnet pool. Show subnet pool details.

v2.0 of the Neutron API is assumed. It is assumed that subnet_allocation options mentioned in the [network-feature-enabled] section and default_network option mentioned in the [network] section of etc/tempest.conf:

test_create_list_show_update_delete_subnetpools()

Test idempotent id: 62595970-ab1c-4b7f-8fcc-fddfe55e9811

Test create/list/show/update/delete of subnet pools

network.test_tags module

class TagsExtTest(**args*, ***kwargs*)

Bases: BaseNetworkTest

Tests the following operations in the tags API:

Update all tags. Delete all tags. Check tag existence. Create a tag. List tags. Remove a tag.

v2.0 of the Neutron API is assumed. The tag-ext or standard-attr-tag extension allows users to set tags on the following resources: subnets, ports, routers and subnetpools. from stein release the tag-ext has been renamed to standard-attr-tag

test_create_check_list_and_delete_tags()

Test idempotent id: c6231efa-9a89-4adf-b050-2a3156b8a1d9

Test tag operations on subnets/ports/routers/subnetpools

test_update_and_delete_all_tags()

Test idempotent id: 663a90f5-f334-4b44-afe0-c5fc1d408791

Test update/delete all tags on subnets/ports/routers/subnetpools

class TagsTest(**args*, ***kwargs*)

Bases: BaseNetworkTest

Tests the following operations in the tags API:

Update all tags. Delete all tags. Check tag existence. Create a tag. List tags. Remove a tag.

v2.0 of the Neutron API is assumed. The tag extension allows users to set tags on their networks. The extension supports networks only.

test_create_list_show_update_delete_tags()

Test idempotent id: ee76bfaf-ac94-4d74-9ecc-4bbd4c583cb1

Validate that creating a tag on a network resource works

network.test_versions module

class NetworksApiDiscovery(**args*, ***kwargs*)

Bases: BaseNetworkTest

test_api_version_resources()

Test idempotent id: cac8a836-c2e0-4304-b556-cd299c7281d1

Test that GET / returns expected resources.

The versions document returned by Neutron returns a few other resources other than just available API versions: it also states the status of each API version and provides links to schema.

test_show_api_v2_details()

Test idempotent id: e64b7216-3178-4263-967c-d389290988bf

Test that GET /v2.0/ returns expected resources.

Module contents

object_storage

object_storage package

Submodules

object_storage.base module

class BaseObjectTest(**args*, ***kwargs*)

Bases: BaseTestCase

object_storage.test_account_bulk module

class BulkTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test bulk operation of archived file

test_bulk_delete()

Test idempotent id: c075e682-0d2a-43b2-808d-4116200d736d

Test bulk operation of deleting multiple files

test_bulk_delete_by_POST()

Test idempotent id: dbea2bcb-efbb-4674-ac8a-a5a0e33d1d79

Test bulk operation of deleting multiple files by HTTP POST

test_extract_archive()

Test idempotent id: a407de51-1983-47cc-9f14-47c2b059413c

Test bulk operation of file upload with an archived file

object_storage.test_account_quotas module

class AccountQuotasTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test account quotas

test_admin_modify_quota()

Test idempotent id: 63f51f9f-5f1d-4fc6-b5be-d454d70949d6

Test ResellerAdmin can modify/remove the quota on a users account

Using the account client, the test modifies the quota successively to:

- 25: a random value different from the initial quota value.
- : an empty value, equivalent to the removal of the quota.

- 20: set the quota to its initial value.

test_upload_valid_object()

Test idempotent id: a22ef352-a342-4587-8f47-3bbdb5b039c4

Test uploading valid object

object_storage.test_account_quotas_negative module

class AccountQuotasNegativeTest(**args*, ***kwargs*)

Bases: BaseObjectTest

test_user_modify_quota()

Test idempotent id: d1dc5076-555e-4e6d-9697-28f1fe976324

Test that a user cannot modify or remove a quota on its account.

object_storage.test_account_services module

class AccountTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test account metadata and containers

test_list_account_metadata()

Test idempotent id: 4894c312-6056-4587-8d6f-86ffbf861f80

Test listing account metadata

test_list_containers()

Test idempotent id: 3499406a-ae53-4f8c-b43a-133d4dc6fe3f

Test listing containers

test_list_containers_reverse_order()

Test idempotent id: b1811cff-d1ed-4c15-a52e-efd8de41cf34

Test listing containers in reverse order

test_list_containers_with_end_marker()

Test idempotent id: 5ca164e4-7bde-43fa-bafb-913b53b9e786

Test listing containers with end_marker parameter

First expect to get 0 container as we specified first container as end_marker, second expect to get the top half of the containers

test_list_containers_with_format_json()

Test idempotent id: 1c7efa35-e8a2-4b0b-b5ff-862c7fd83704

Test listing containers setting format parameter to json

test_list_containers_with_format_xml()

Test idempotent id: 4477b609-1ca6-4d4b-b25d-ad3f01086089

Test listing containers setting format parameter to xml

test_list_containers_with_limit()

Test idempotent id: 5cfa4ab2-4373-48dd-a41f-a532b12b08b2

Test listing containers with limit parameter

Listing containers limited to one of them, half of them, and then all of them.

test_list_containers_with_limit_and_end_marker()

Test idempotent id: 888a3f0e-7214-4806-8e50-5e0c9a69bb5e

Test listing containers combining end_marker and limit parameter

Result are always limited by the limit whatever the end_marker.

test_list_containers_with_limit_and_marker()

Test idempotent id: f7064ae8-dbcc-48da-b594-82feef6ea5af

Test listing containers combining marker and limit parameter

Result are always limited by the limit whatever the marker.

test_list_containers_with_limit_and_marker_and_end_marker()

Test idempotent id: 8cf98d9c-e3a0-4e44-971b-c87656fdddbd

Test listing containers combining marker and end_marker and limit

Result are always limited by the limit whatever the marker and the end_marker.

test_list_containers_with_marker()

Test idempotent id: 638f876d-6a43-482a-bbb3-0840bca101c6

Test listing containers with marker parameter

First expect to get 0 container as we specified the last container as marker, second expect to get the bottom half of the containers.

test_list_containers_with_marker_and_end_marker()

Test idempotent id: ac8502c2-d4e4-4f68-85a6-40befea2ef5e

Test listing containers with marker and end_marker parameter

If we use the first container as marker, and the last container as end_marker, then we should get all containers excluding the first one and the last one.

test_list_containers_with_prefix()

Test idempotent id: 365e6fc7-1cfe-463b-a37c-8bd08d47b6aa

Test listing containers that have a name starting with a prefix

test_list_extensions()

Test idempotent id: 6eb04a6a-4860-4e31-ba91-ea3347d76b58

Test listing capabilities

test_list_no_account_metadata()

Test idempotent id: b904c2e3-24c2-4dba-ad7d-04e90a761be5

Test listing account metadata for account without metadata

test_list_no_containers()

Test idempotent id: 884ec421-fbad-4fcc-916b-0580f2699565

Test listing containers for an account without container

test_update_account_metadata_with_create_and_delete_metadata()

Test idempotent id: 8e5fc073-59b9-42ee-984a-29ed11b2c749

Test adding and deleting metadata simultaneously

Send a request adding and deleting metadata requests simultaneously, both adding and deleting of metadata will succeed.

test_update_account_metadata_with_create_metadata()

Test idempotent id: e2a08b5f-3115-4768-a3ee-d4287acd6c08

Test adding metadata to account

test_update_account_metadata_with_create_metadata_key()

Test idempotent id: 64fd53f3-adbd-4639-af54-436e4982dbfb

Test adding metadata to account with empty value

Adding metadata with empty value to account, the metadata is not registered.

test_update_account_metadata_with_delete_metadata()

Test idempotent id: 9f60348d-c46f-4465-ae06-d51dbd470953

Test deleting metadata from account

test_update_account_metadata_with_delete_metadata_key()

Test idempotent id: d4d884d3-4696-4b85-bc98-4f57c4dd2bf1

Test deleting metadata from account with empty value

Although the value of metadata is not set, the feature of deleting metadata is valid, so the metadata is removed from account.

object_storage.test_account_services_negative module

class AccountNegativeTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Negative tests of account

test_list_containers_with_non_authorized_user()

Test idempotent id: 070e6aca-6152-4867-868d-1118d68fb38c

Test listing containers using non-authorized user

object_storage.test_container_acl module

class ObjectTestACLs(**args*, ***kwargs*)

Bases: BaseObjectTest

Test object ACLs

test_read_object_with_rights()

Test idempotent id: a3270f3f-7640-4944-8448-c7ea783ea5b6

Test reading object using authorized user

test_write_object_with_rights()

Test idempotent id: aa58bfa5-40d9-4bc3-82b4-d07f4a9e392a

Test writing object using authorized user

object_storage.test_container_acl_negative module

class ObjectACLsNegativeTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Negative tests of object ACLs

test_delete_object_with_non_authorized_user()

Test idempotent id: 7343ac3d-cfed-4198-9bb0-00149741a492

Test deleting object with non-authorized user

test_delete_object_without_using_creds()

Test idempotent id: af85af0b-a025-4e72-a90e-121babf55720

Test deleting object without using credentials

test_delete_object_without_write_rights()

Test idempotent id: b4e366f8-f185-47ab-b789-df4416f9ecdb

Test deleting object without write rights

test_read_object_with_non_authorized_user()

Test idempotent id: abf63359-be52-4feb-87dd-447689fc77fd

Test reading object with non-authorized user

test_read_object_without_rights()

Test idempotent id: 9ed01334-01e9-41ea-87ea-e6f465582823

Test reading object without rights

test_write_object_with_non_authorized_user()

Test idempotent id: 63d84e37-55a6-42e2-9e5f-276e60e26a00

Test writing object with non-authorized user

test_write_object_without_rights()

Test idempotent id: a3a585a7-d8cf-4b65-a1a0-edc2b1204f85

Test writing object without rights

test_write_object_without_using_creds()

Test idempotent id: af587587-0c24-4e15-9822-8352ce711013

Test writing object without using credentials

test_write_object_without_write_rights()

Test idempotent id: 8ba512ad-aa6e-444e-b882-2906a0ea2052

Test writing object without write rights

object_storage.test_container_quotas module

class ContainerQuotasTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Attempts to test the perfect behavior of quotas in a container.

test_upload_large_object()

Test idempotent id: 22eeeb2b-3668-4160-baef-44790f65a5a0

Attempts to upload an object larger than the bytes quota.

test_upload_too_many_objects()

Test idempotent id: 3a387039-697a-44fc-a9c0-935de31f426b

Attempts to upload many objects that exceeds the count limit.

test_upload_valid_object()

Test idempotent id: 9a0fb034-86af-4df0-86fa-f8bd7db21ae0

Attempts to uploads an object smaller than the bytes quota.

object_storage.test_container_services module

class ContainerTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test containers

test_create_container()

Test idempotent id: 92139d73-7819-4db1-85f8-3f2f22a8d91f

Test creating container

test_create_container_overwrite()

Test idempotent id: 49f866ed-d6af-4395-93e7-4187eb56d322

Test overwriting container with the same name

test_create_container_with_metadata_key()

Test idempotent id: c2ac4d59-d0f5-40d5-ba19-0635056d48cd

Test creating container with the blank value of metadata

test_create_container_with_metadata_value()

Test idempotent id: e1e8df32-7b22-44e1-aa08-ccfd8d446b58

Test creating container with metadata value

test_create_container_with_remove_metadata_key()

Test idempotent id: 24d16451-1c0c-4e4f-b59c-9840a3aba40e

Test creating container with the blank value of remove metadata

test_create_container_with_remove_metadata_value()

Test idempotent id: 8a21ebad-a5c7-4e29-b428-384edc8cd156

Test creating container with remove metadata

test_delete_container()

Test idempotent id: 95d3a249-b702-4082-a2c4-14bb860cf06a

Test deleting container

test_list_container_contents()

Test idempotent id: 312ff6bd-5290-497f-bda1-7c5fec6697ab

Test getting container contents list

test_list_container_contents_with_delimiter()

Test idempotent id: fe323a32-57b9-4704-a996-2e68f83b09bc

Test getting container contents list using delimiter param

test_list_container_contents_with_end_marker()

Test idempotent id: 55b4fa5c-e12e-4ca9-8fcf-a79afe118522

Test getting container contents list using end_marker param

test_list_container_contents_with_format_json()

Test idempotent id: 196f5034-6ab0-4032-9da9-a937bbb9fba9

Test getting container contents list using format_json param

test_list_container_contents_with_format_xml()

Test idempotent id: 655a53ca-4d15-408c-a377-f4c6dbd0a1fa

Test getting container contents list using format_xml param

test_list_container_contents_with_limit()

Test idempotent id: 297ec38b-2b61-4ff4-bcd1-7fa055e97b61

Test getting container contents list using limit param

test_list_container_contents_with_marker()

Test idempotent id: c31ddc63-2a58-4f6b-b25c-94d2937e6867

Test getting container contents list using marker param

test_list_container_contents_with_no_object()

Test idempotent id: 4646ac2d-9bfb-4c7d-a3c5-0f527402b3df

Test getting empty container contents list

test_list_container_contents_with_path()

Test idempotent id: 58ca6cc9-6af0-408d-aaec-2a6a7b2f0df9

Test getting container contents list using path param

test_list_container_contents_with_prefix()

Test idempotent id: 77e742c7-caf2-4ec9-8aa4-f7d509a3344c

Test getting container contents list using prefix param

test_list_container_metadata()

Test idempotent id: 96e68f0e-19ec-4aa2-86f3-adc6a45e14dd

Test listing container metadata

test_list_no_container_metadata()

Test idempotent id: a2faf936-6b13-4f8d-92a2-c2278355821e

Test listing container without metadata

test_update_container_metadata_with_create_and_delete_metadata()

Test idempotent id: cf19bc0b-7e16-4a5a-aaed-cb0c2fe8deef

Test updating container with adding and deleting metadata

Send one request of adding and deleting metadata.

test_update_container_metadata_with_create_metadata()

Test idempotent id: 2ae5f295-4bf1-4e04-bfad-21e54b62cec5

Test updating container metadata using add metadata

test_update_container_metadata_with_create_metadata_key()

Test idempotent id: 31f40a5f-6a52-4314-8794-cd89baed3040

Test updating container metadata with a blank value of metadata

test_update_container_metadata_with_delete_metadata()

Test idempotent id: 3a5ce7d4-6e4b-47d0-9d87-7cd42c325094

Test updating container metadata using delete metadata

test_update_container_metadata_with_delete_metadata_key()

Test idempotent id: a2e36378-6f1f-43f4-840a-ffd9cfd61914

Test updating container metadata with a blank value of metadata

object_storage.test_container_services_negative module

class ContainerNegativeTest(*args, **kwargs)

Bases: BaseObjectTest

Negative tests of containers

test_create_container_metadata_exceeds_overall_metadata_count()

Test idempotent id: ac666539-d566-4f02-8ceb-58e968dfb732

Test creating container with metadata exceeding default count

test_create_container_metadata_name_exceeds_max_length()

Test idempotent id: 41e645bf-2e68-4f84-bf7b-c71aa5cd76ce

Test creating container with metadata name longer than max

test_create_container_metadata_value_exceeds_max_length()

Test idempotent id: 81e36922-326b-4b7c-8155-3bbceecd7a82

Test creating container with metadata value longer than max

test_create_container_name_exceeds_max_length()

Test idempotent id: 30686921-4bed-4764-a038-40d741ed4e78

Test creating container with name longer than max

test_delete_non_empty_container()

Test idempotent id: 42da116e-1e8c-4c96-9e06-2f13884ed2b1

Test deleting a container with object in it

test_delete_with_nonexistent_container_name()

Test idempotent id: 65387dbf-a0e2-4aac-9ddc-16eb3f1f69ba

Test deleting metadata using a non existent container name

test_get_metadata_headers_with_invalid_container_name()

Test idempotent id: 1a95ab2e-b712-4a98-8a4d-8ce21b7557d6

Test getting metadata headers with invalid container name

test_list_all_container_objects_on_deleted_container()

Test idempotent id: 86b2ab08-92d5-493d-acd2-85f0c848819e

Test getting a list of all objects on a deleted container

test_list_all_container_objects_with_nonexistent_container()

Test idempotent id: 14331d21-1e81-420a-beea-19cb5e5207f5

Test getting a list of all objects on a non existent container

test_update_metadata_with_nonexistent_container_name()

Test idempotent id: 125a24fa-90a7-4cfc-b604-44e49d788390

Test updating metadata using a nonexistent container name

object_storage.test_container_staticweb module

class StaticWebTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test static web

test_web_error()

Test idempotent id: f18b4bef-212e-45e7-b3ca-59af3a465f82

Test web error

test_web_index()

Test idempotent id: c1f055ab-621d-4a6a-831f-846fcb578b8b

Test web index

test_web_listing()

Test idempotent id: 941814cf-db9e-4b21-8112-2b6d0af10ee5

Test web listing

test_web_listing_css()

Test idempotent id: bc37ec94-43c8-4990-842e-0e5e02fc8926

Test web listing css

object_storage.test_container_sync module

class ContainerSyncTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test container synchronization

test_container_synchronization()

Test idempotent id: be008325-1bba-4925-b7dd-93b58f22ce9b

Test container synchronization

object_storage.test_container_sync_middleware module

class ContainerSyncMiddlewareTest(**args*, ***kwargs*)

Bases: ContainerSyncTest

Test containers synchronization specifying realm and cluster

test_container_synchronization()

Test idempotent id: ea4645a1-d147-4976-82f7-e5a7a3065f80

Test container synchronization specifying realm and cluster

object_storage.test_crossdomain module

class CrossdomainTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test crossdomain policy

test_get_crossdomain_policy()

Test idempotent id: d1b8b031-b622-4010-82f9-ff78a9e915c7

Test getting crossdomain policy

object_storage.test_healthcheck module

class HealthcheckTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test healthcheck

test_get_healthcheck()

Test idempotent id: db5723b1-f25c-49a9-bfeb-7b5640caf337

Test getting healthcheck

object_storage.test_object_expiry module

class ObjectExpiryTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test object expiry

test_get_object_after_expiry_time()

Test idempotent id: fb024a42-37f3-4ba5-9684-4f40a7910b41

Test object is expired after x-delete-after time

test_get_object_at_expiry_time()

Test idempotent id: e592f18d-679c-48fe-9e36-4be5f47102c5

Test object is expired at x-delete-at time

object_storage.test_object_formpost module

class ObjectFormPostTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test object post with form

test_post_object_using_form()

Test idempotent id: 80fac02b-6e54-4f7b-be0d-a965b5cbef76

Test posting object using form

object_storage.test_object_formpost_negative module

class ObjectFormPostNegativeTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Negative tests of object post with form

test_post_object_using_form_expired()

Test idempotent id: d3fb3c4d-e627-48ce-9379-a1631f21336d

Test posting object using expired form

test_post_object_using_form_invalid_signature()

Test idempotent id: b277257f-113c-4499-b8d1-5fead79f7360

Test posting object using form with invalid signature

object_storage.test_object_services module

class ObjectTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test storage object

test_copy_object_2d_way()

Test idempotent id: 06f90388-2d0e-40aa-934c-e9a8833e958a

Test copying objects data to the new object using COPY

test_copy_object_across_containers()

Test idempotent id: aa467252-44f3-472a-b5ae-5b57c3c9c147

Test copying object to another container

test_copy_object_in_same_container()

Test idempotent id: 1a9ab572-1b66-4981-8c21-416e2a5e6011

Test copying object to another object in same container

test_copy_object_to_itself()

Test idempotent id: 2248abba-415d-410b-9c30-22dff9cd6e67

Test changing the content type of an existing object

test_copy_object_with_x_fresh_metadata()

Test idempotent id: 5a9e2cc6-85b6-46fc-916d-0cbb7a88e5fd

Test copying objectwith x_fresh_metadata

test_copy_object_with_x_object_meta()

Test idempotent id: edabedca-24c3-4322-9b70-d6d9f942a074

Test copying object with x_object_meta

test_copy_object_with_x_object_metakey()

Test idempotent id: a28a8b99-e701-4d7e-9d84-3b66f121460b

Test copying object with x_object_metakey

test_create_object()

Test idempotent id: 5b4ce26f-3545-46c9-a2ba-5754358a4c62

Test creating object and checking the objects uploaded content

test_create_object_with_content_disposition()

Test idempotent id: 5daebb1d-f0d5-4dc9-b541-69672eff00b0

Test creating object with content-disposition

test_create_object_with_content_encoding()

Test idempotent id: 605f8317-f945-4bee-ae91-013f1da8f0a0

Test creating object with content-encoding

test_create_object_with_etag()

Test idempotent id: 73820093-0503-40b1-a478-edf0e69c7d1f

Test creating object with Etag

test_create_object_with_expect_continue()

Test idempotent id: 84dafe57-9666-4f6d-84c8-0814d37923b8

Test creating object with expect_continue

test_create_object_with_transfer_encoding()

Test idempotent id: 4f84422a-e2f2-4403-b601-726a4220b54e

Test creating object with transfer_encoding

test_create_object_with_x_fresh_metadata()

Test idempotent id: 0f3d62a6-47e3-4554-b0e5-1a5dc372d501

Test creating object with x-fresh-metadata

The previous added metadata will be cleared.

test_create_object_with_x_object_meta()

Test idempotent id: 1c7ed3e4-2099-406b-b843-5301d4811baf

Test creating object with x-object-meta

test_create_object_with_x_object_metakey()

Test idempotent id: e4183917-33db-4153-85cc-4dacbb938865

Test creating object with the blank value of metadata

test_create_object_with_x_remove_object_meta()

Test idempotent id: ce798afc-b278-45de-a5ce-2ea124b98b99

Test creating object with x-remove-object-meta

The metadata will be removed from the object.

test_create_object_with_x_remove_object_metakey()

Test idempotent id: ad21e342-7916-4f9e-ab62-a1f885f2aaf9

Test creating object with the blank value of remove metadata

Creating object with blank metadata X-Remove-Object-Meta-test-meta, metadata x-object-meta-test-meta will be removed from the object.

test_delete_object()

Test idempotent id: 17738d45-03bd-4d45-9e0b-7b2f58f98687

Test deleting object

test_get_object()

Test idempotent id: 02610ba7-86b7-4272-9ed8-aa8d417cb3cd

Test retrieving objects data (in response body)

test_get_object_if_different()

Test idempotent id: 50d01f12-526f-4360-9ac2-75dd508d7b68

Test getting object content only when the local file is different

http://en.wikipedia.org/wiki/HTTP_ETag Make a conditional request for an object using the If-None-Match header, it should get downloaded only if the local file is different, otherwise the response code should be 304 Not Modified

test_get_object_with_if_match()

Test idempotent id: c05b4013-e4de-47af-be84-e598062b16fc

Test getting object with if_match

test_get_object_with_if_modified_since()

Test idempotent id: be133639-e5d2-4313-9b1f-2d59fc054a16

Test getting object with if_modified_since

test_get_object_with_if_none_match()

Test idempotent id: 641500d5-1612-4042-a04d-01fc4528bc30

Test getting object with if_none_match

test_get_object_with_if_unmodified_since()

Test idempotent id: 0aa1201c-10aa-467a-bee7-63cbdd463152

Test getting object with if_unmodified_since

test_get_object_with_metadata()

Test idempotent id: 005f9bf6-e06d-41ec-968e-96c78e0b1d82

Test getting object with metadata

test_get_object_with_range()

Test idempotent id: 05a1890e-7db9-4a6c-90a8-ce998a2bddfa

Test getting object with range

test_get_object_with_x_newest()

Test idempotent id: 94587078-475f-48f9-a40f-389c246e31cd

Test getting object with x_newest

test_get_object_with_x_object_manifest()

Test idempotent id: 11b4515b-7ba7-4ca8-8838-357ded86fc10

Test getting object with x_object_manifest

test_list_no_object_metadata()

Test idempotent id: 170fb90e-f5c3-4b1f-ae1b-a18810821172

Test listing object metadata for object without metadata

test_list_object_metadata()

Test idempotent id: 9a447cf6-de06-48de-8226-a8c6ed31caf2

Test listing object metadata

test_list_object_metadata_with_x_object_manifest()

Test idempotent id: 23a3674c-d6de-46c3-86af-ff92bfc8a3da

Test getting object metadata with x_object_manifest

test_object_upload_in_segments()

Test idempotent id: e3e6a64a-9f50-4955-b987-6ce6767c97fb

Test uploading object in segments

test_update_object_metadata()

Test idempotent id: 7a94c25d-66e6-434c-9c38-97d4e2c29945

Test updating object metadata

test_update_object_metadata_with_create_and_remove_metadata()

Test idempotent id: f726174b-2ded-4708-bff7-729d12ce1f84

Test updating object with creation and deletion of metadata

Update object with creation and deletion of metadata with one request, both operations will succeed.

test_update_object_metadata_with_remove_metadata()

Test idempotent id: 48650ed0-c189-4e1e-ad6b-1d4770c6e134

Test updating object metadata with remove metadata

test_update_object_metadata_with_x_object_manifest()

Test idempotent id: 08854588-6449-4bb7-8cca-f2e1040f5e6f

Test updating object metadata with x_object_manifest

test_update_object_metadata_with_x_object_metakey()

Test idempotent id: 0dbbe89c-6811-4d84-a2df-eca2bdd40c0e

Test updating object metadata with a blank value of metadata

test_update_object_metadata_with_x_remove_object_metakey()

Test idempotent id: 9a88dca4-b684-425b-806f-306cd0e57e42

Test updating object metadata with blank remove metadata value

class PublicObjectTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test public storage object

test_access_public_container_object_without_using_creds()

Test idempotent id: 07c9cf95-c0d4-4b49-b9c8-0ef2c9b27193

Test accessing public container object without using credentials

Make container public-readable and access an object in it object anonymously, without using credentials.

test_access_public_object_with_another_user_creds()

Test idempotent id: 54e2a2fe-42dc-491b-8270-8e4217dd4cdc

Test accessing public object with another users credentials

Make container public-readable and access an object in it using another users credentials.

object_storage.test_object_slo module

class ObjectSloTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test static large object

test_delete_large_object()

Test idempotent id: 87b6dfa1-abe9-404d-8bf0-6c3751e6aa77

Test deleting static large object using multipart manifest

test_list_large_object_metadata()

Test idempotent id: e69ad766-e1aa-44a2-bdd2-bf62c09c1456

Test listing static large object metadata

List static large object metadata using multipart manifest

test_retrieve_large_object()

Test idempotent id: 49bc49bc-dd1b-4c0f-904e-d9f10b830ee8

Test listing static large object using multipart manifest

test_upload_manifest()

Test idempotent id: 2c3f24a6-36e8-4711-9aa2-800ee1fc7b5b

Test creating static large object from multipart manifest

object_storage.test_object_temp_url module

class ObjectTempUrlTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test object temp url

test_get_object_using_temp_url()

Test idempotent id: f91c96d4-1230-4bba-8eb9-84476d18d991

Test getting object using temp url

test_get_object_using_temp_url_key_2()

Test idempotent id: 671f9583-86bd-4128-a034-be282a68c5d8

Test getting object using metadata Temp-URL-Key-2

test_get_object_using_temp_url_with_inline_query_parameter()

Test idempotent id: 9d9cfd90-708b-465d-802c-e4a8090b823d

Test getting object using temp url with inline query parameter

test_head_object_using_temp_url()

Test idempotent id: 249a0111-5ad3-4534-86a7-1993d55f9185

Test HEAD operation of object using temp url

test_put_object_using_temp_url()

Test idempotent id: 9b08dade-3571-4152-8a4f-a4f2a873a735

Test putting object using temp url

object_storage.test_object_temp_url_negative module

class ObjectTempUrlNegativeTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Negative tests of object temp url

test_get_object_after_expiration_time()

Test idempotent id: 5a583aca-c804-41ba-9d9a-e7be132bdf0b

Test getting object after expiration time

object_storage.test_object_version module

class ContainerTest(**args*, ***kwargs*)

Bases: BaseObjectTest

Test versioned container

test_versioned_container()

Test idempotent id: a151e158-dcbf-4a1f-a1e7-46cd65895a6f

Test versioned container

1. create container1
2. create container2, with container1 as X-versions-Location header
3. create object1 in container1
4. create 2nd version of object1
5. delete object version 2
6. delete object version 1

Module contents

volume

volume package

Subpackages

volume.admin package

Submodules

volume.admin.test_backends_capabilities module

class BackendsCapabilitiesAdminTestsJSON(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test backends capabilities

test_compare_volume_stats_values()

Test idempotent id: a9035743-d46a-47c5-9cb7-3c80ea16dea0

Test comparing volume stats values

Compare volume stats between show_backend_capabilities and show_pools.

test_get_capabilities_backend()

Test idempotent id: 3750af44-5ea2-4cd4-bc3e-56e7e6caf854

Test getting backend capabilities

volume.admin.test_encrypted_volumes_extend module

class EncryptedVolumesExtendAttachedTest(*args, **kwargs)

Bases: BaseVolumesExtendAttachedTest, BaseVolumeAdminTest

Tests extending the size of an attached encrypted volume.

test_extend_attached_encrypted_volume_luksv1()

Test idempotent id: e93243ec-7c37-4b5b-a099-ebf052c13216

LUKs v1 decrypts and extends through libvirt.

test_extend_attached_encrypted_volume_luksv2()

Test idempotent id: 381a2a3a-b2f4-4631-a910-720881f2cc2f

LUKs v2 decrypts and extends through os-brick.

volume.admin.test_group_snapshots module

class BaseGroupSnapshotsTest(*args, **kwargs)

Bases: BaseVolumeAdminTest

class GroupSnapshotsTest(*args, **kwargs)

Bases: *BaseGroupSnapshotsTest*

Test group snapshot

test_create_group_from_group_snapshot()

Test idempotent id: eff52c70-efc7-45ed-b47a-4ad675d09b81

Test creating group from group snapshot

1. Create volume type volume_type1
2. Create group type group_type1
3. Create group group1 with group_type1 and volume_type1
4. Create volume volume1 with volume_type1 and group1
5. Create group snapshot group_snapshot1 with group1
6. Check snapshot created from volume1 reaches available status
7. Create group group2 from group_snapshot1
8. Check the volumes belonging to group2 reach available status
9. Check group2 reaches available status

test_delete_group_snapshots_following_updated_volumes()

Test idempotent id: 7d7fc000-0b4c-4376-a372-544116d2e127

Test deleting group snapshot following updated volumes

1. Create volume type volume_type1
2. Create group type group_type1
3. Create group group1 with group_type1 and volume_type1
4. Create 2 volumes volume1 and volume2 with volume_type1 and group1

5. For each created volume, removing and then adding back to group1
6. Create group snapshot group_snapshot1 with group1
7. Check snapshots created from volume1 and volume2 reach available status
8. Delete group_snapshot1
9. Check snapshots created from volume1 and volume2 are deleted

test_group_snapshot_create_show_list_delete()

Test idempotent id: 1298e537-f1f0-47a3-a1dd-8adec8168897

Test create/show/list/delete group snapshot

1. Create volume type volume_type1
2. Create group type group_type1
3. Create group group1 with group_type1 and volume_type1
4. Create volume volume1 with volume_type1 and group1
5. Create group snapshot group_snapshot1 with group1
6. Check snapshot created from volume1 reaches available status
7. Check the created group snapshot group_snapshot1 is in the list of all group snapshots
8. Delete group snapshot group_snapshot1

class GroupSnapshotsV319Test(**args*, ***kwargs*)

Bases: *BaseGroupSnapshotsTest*

Test group snapshot with volume microversion greater than 3.18

test_reset_group_snapshot_status()

Test idempotent id: 3b42c9b9-c984-4444-816e-ca2e1ed30b40

Test resetting group snapshot status to creating/available/error

volume.admin.test_group_type_specs module

class GroupTypeSpecsTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test group type specs

test_group_type_specs_create_show_update_list_delete()

Test idempotent id: bb4e30d0-de6e-4f4d-866c-dcc48d023b4e

Test create/show/update/list/delete group type specs

volume.admin.test_group_types module

class GroupTypesTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test group types

test_group_type_create_list_update_show_delete()

Test idempotent id: dd71e5f9-393e-4d4f-90e9-fa1b8d278864

Test create/list/update/show/delete group type

test_group_type_list_by_optional_params()

Test idempotent id: 3d5e5cec-72b4-4511-b135-7cc2b7a053ae

Test list group type sort/public

volume.admin.test_groups module

class GroupsTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Tests of volume groups with microversion greater than 3.12

test_group_create_show_list_delete()

Test idempotent id: 4b111d28-b73d-4908-9bd2-03dc2992e4d4

Test creating, showing, listing and deleting of volume group

test_group_update()

Test idempotent id: 4a8a6fd2-8b3b-4641-8f54-6a6f99320006

Test updating volume group

class GroupsV314Test(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Tests of volume groups with microversion greater than 3.13

test_create_group_from_group()

Test idempotent id: 2424af8c-7851-4888-986a-794b10c3210e

Test creating volume group from volume group

class GroupsV320Test(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Tests of volume groups with microversion greater than 3.19

test_reset_group_status()

Test idempotent id: b20c696b-0cbc-49a5-8b3a-b1fb9338f45c

Test resetting volume group status to creating/available/error

volume.admin.test_multi_backend module

class VolumeMultiBackendTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test volume multi backends

test_backend_name_distinction()

Test idempotent id: 46435ab1-a0af-4401-8373-f14e66b0dd58

Test volume backend distinction when type is without prefix

1. For each backend, create volume type with volume_backend_name as extra spec key
2. Create volumes using the created volume types
3. Check os-vol-host-attr:host of each created volume is different.

test_backend_name_distinction_with_prefix()

Test idempotent id: 4236305b-b65a-4bfc-a9d2-69cb5b2bf2ed

Test volume backend distinction when type is with prefix

1. For each backend, create volume type with capabilities:volume_backend_name as extra spec key
2. Create volumes using the created volume types
3. Check os-vol-host-attr:host of each created volume is different.

test_backend_name_reporting()

Test idempotent id: c1a41f3f-9dad-493e-9f09-3ff197d477cc

Test backend name reporting for volume when type is without prefix

1. Create volume type, with volume_backend_name as extra spec key
2. Create volume using the created volume type
3. Check os-vol-host-attr:host of the volume info, the value should contain @ character, like cinder@CloveStorage#tecs_backend

test_backend_name_reporting_with_prefix()

Test idempotent id: f38e647f-ab42-4a31-a2e7-ca86a6485215

Test backend name reporting for volume when type is with prefix

1. Create volume type, with capabilities:volume_backend_name as extra spec key
2. Create volume using the created volume type
3. Check os-vol-host-attr:host of the volume info, the value should contain @ character, like cinder@CloveStorage#tecs_backend

volume.admin.test_qos module

class QosSpecsTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test the Cinder QoS-specs.

Tests for create, list, delete, show, associate, disassociate, set/unset key APIs.

test_associate_disassociate_qos()

Test idempotent id: 1dd93c76-6420-485d-a771-874044c416ac

Test the following operations :

1. associate_qos
2. get_association_qos
3. disassociate_qos
4. disassociate_all_qos

test_create_delete_qos_with_back_end_consumer()

Test idempotent id: b115cded-8f58-4ee4-aab5-9192cfada08f

Tests the creation and deletion of QoS specs

With consumer as back-end

test_create_delete_qos_with_both_consumer()

Test idempotent id: f88d65eb-ea0d-487d-af8d-71f4011575a4

Tests the creation and deletion of QoS specs

With consumer as both front end and back end

test_create_delete_qos_with_front_end_consumer()

Test idempotent id: 7e15f883-4bef-49a9-95eb-f94209a1ced1

Tests the creation and deletion of QoS specs

With consumer as front end

test_get_qos()

Test idempotent id: 7aa214cc-ac1a-4397-931f-3bb2e83bb0fd

Tests the detail of a given qos-specs

test_list_qos()

Test idempotent id: 75e04226-bcf7-4595-a34b-fdf0736f38fc

Tests the list of all qos-specs

test_set_unset_qos_key()

Test idempotent id: ed00fd85-4494-45f2-8ceb-9e2048919aed

Test the addition of a specs key to qos-specs

volume.admin.test_snapshot_manage module

class SnapshotManageAdminTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Unmanage & manage snapshots

This feature provides the ability to import/export volume snapshot from one Cinder to another and to import snapshots that have not been managed by Cinder from a storage back end to Cinder

test_snapshot_manage_with_attached_volume()

Test idempotent id: 7c735385-e953-4198-8534-68137f72dbdc

Test manage a snapshot with an attached volume.

The case validates manage snapshot operation while the parent volume is attached to an instance.

test_unmanage_manage_snapshot()

Test idempotent id: 0132f42d-0147-4b45-8501-cc504bbf7810

volume.admin.test_snapshots_actions module

class SnapshotsActionsTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test volume snapshot actions

test_reset_snapshot_status()

Test idempotent id: 3e13ca2f-48ea-49f3-ae1a-488e9180d535

Test resetting snapshot status to creating

test_snapshot_force_delete_when_snapshot_is_creating()

Test idempotent id: 05f711b6-e629-4895-8103-7ca069f2073a

Test force delete when status of snapshot is creating

test_snapshot_force_delete_when_snapshot_is_deleting()

Test idempotent id: 92ce8597-b992-43a1-8868-6316b22a969e

Test force delete when status of snapshot is deleting

test_snapshot_force_delete_when_snapshot_is_error()

Test idempotent id: 645a4a67-a1eb-4e8e-a547-600abac1525d

Test force delete when status of snapshot is error

test_snapshot_force_delete_when_snapshot_is_error_deleting()

Test idempotent id: bf89080f-8129-465e-9327-b2f922666ba5

Test force delete when status of snapshot is error_deleting

test_update_snapshot_status()

Test idempotent id: 41288afd-d463-485e-8f6e-4eea159413eb

Test updating snapshot

Update snapshot status to error and progress to 80%.

volume.admin.test_user_messages module

class UserMessagesTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test volume messages with microversion greater than 3.2

test_delete_message()

Test idempotent id: c6eb6901-cdcc-490f-b735-4fe251842aed

Test deleting volume messages

test_list_show_messages()

Test idempotent id: 50f29e6e-f363-42e1-8ad1-f67ae7fd4d5a

Test listing and showing volume messages

volume.admin.test_volume_hosts module

class VolumeHostsAdminTestsJSON(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test fetching volume hosts info by admin users

test_list_hosts()

Test idempotent id: d5f3efa2-6684-4190-9ced-1c2f526352ad

Test listing volume hosts

test_show_host()

Test idempotent id: 21168d57-b373-4b71-a3ac-f2c88f0c5d31

Test getting volume host details

volume.admin.test_volume_manage module

class VolumeManageAdminTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test volume manage by admin users

test_unmanage_manage_volume()

Test idempotent id: 70076c71-0ce1-4208-a8ff-36a66e65cc1e

Test unmanaging and managing volume

volume.admin.test_volume_pools module

class VolumePoolsAdminTestsJSON(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test getting volume pools by admin users

test_get_pools_with_details()

Test idempotent id: d4bb61f7-762d-4437-b8a4-5785759a0ced

Test getting volume pools with detail

test_get_pools_without_details()

Test idempotent id: 0248a46c-e226-4933-be10-ad6fca8227e7

Test getting volume pools without detail

volume.admin.test_volume_quota_classes module

class VolumeQuotaClassesTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test volume quota classes

test_show_default_quota()

Test idempotent id: abb9198e-67d0-4b09-859f-4f4a1418f176

Test showing default volume quota class set

test_update_default_quota()

Test idempotent id: a7644c63-2669-467a-b00e-452dd5c5397b

Test updating default volume quota class set

Check current project and new projects default quota are updated to the provided one.

volume.admin.test_volume_quotas module

class VolumeQuotasAdminTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test volume quotas with admin privilege

test_delete_quota()

Test idempotent id: 874b35a9-51f1-4258-bec5-cd561b6690d3

Test admin can delete the volume quota set for a project

test_list_default_quotas()

Test idempotent id: 2be020a2-5fdd-423d-8d35-a7ffbc36e9f7

Test showing volume default quota set

test_list_quotas()

Test idempotent id: 59eada70-403c-4cef-a2a3-a8ce2f1b07a0

Test showing volume quota set

test_quota_usage()

Test idempotent id: ae8b6091-48ad-4bfa-a188-bbf5cc02115f

Test volume quota usage is updated after creating volume

test_quota_usage_after_volume_transfer()

Test idempotent id: 8911036f-9d54-4720-80cc-a1c9796a8805

Test volume quota usage is updated after transferring volume

test_show_quota_usage()

Test idempotent id: 18c51ae9-cb03-48fc-b234-14a19374dbed

Test showing volume quota usage

test_update_all_quota_resources_for_tenant()

Test idempotent id: 3d45c99e-cc42-4424-a56e-5cbd212b63a6

Test admin can update all the volume quota limits for a project

volume.admin.test_volume_quotas_negative module

class VolumeQuotasNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Negative tests of volume quotas

test_quota_volume_gigabytes()

Test idempotent id: 2dc27eee-8659-4298-b900-169d71a91374

Creating volume with size larger than allowed quota will fail

test_quota_volumes()

Test idempotent id: bf544854-d62a-47f2-a681-90f7a47d86b6

Creating more volume than allowed quota will fail

test_volume_extend_gigabytes_quota_deviation()

Test idempotent id: d321dc21-d8c6-401f-95fe-49f4845f1a6d

Extending volume with size larger than allowed quota will fail

volume.admin.test_volume_retype module

class VolumeRetypeTest(*args, **kwargs)

Bases: BaseVolumeAdminTest

class VolumeRetypeWithMigrationTest(*args, **kwargs)

Bases: *VolumeRetypeTest*

Test volume retype with migration

test_available_volume_retype_with_migration()

Test idempotent id: a1a41f3f-9dad-493e-9f09-3ff197d477cd

Test volume retype with migration

1. Create volume1 with volume_type1
2. Retype volume1 to volume_type2 with migration_policy=on-demand
3. Check volume1s volume_type is changed to volume_type2, and os-vol-host-attr:host in the volume info is changed.

test_volume_from_snapshot_retype_with_migration()

Test idempotent id: d0d9554f-e7a5-4104-8973-f35b27ccb60d

Test volume created from snapshot retype with migration

1. Create volume1 from snapshot with volume_type1
2. Retype volume1 to volume_type2 with migration_policy=on-demand
3. Check volume1s volume_type is changed to volume_type2, and os-vol-host-attr:host in the volume info is changed.

class VolumeRetypeWithoutMigrationTest(*args, **kwargs)

Bases: *VolumeRetypeTest*

Test volume retype without migration

test_available_volume_retype()

Test idempotent id: b90412ee-465d-46e9-b249-ec84a47d5f25

Test volume retype without migration

1. Create volume1 with volume_type1

2. Retype volume1 to volume_type2 with migration_policy=never
3. Check volume1s volume_type is changed to volume_type2, and os-vol-host-attr:host in the volume info is not changed.

volume.admin.test_volume_services module

class VolumesServicesTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Tests Volume Services API.

volume service list requires admin privileges.

test_get_service_by_host_name()

Test idempotent id: 178710e4-7596-4e08-9333-745cb8bc4f8d

Test getting volume service by service host name

test_get_service_by_service_and_host_name()

Test idempotent id: ffa6167c-4497-4944-a464-226bbdb53908

Test getting volume service by binary name and host name

test_get_service_by_service_binary_name()

Test idempotent id: 63a3e1ca-37ee-4983-826d-83276a370d25

Test getting volume service by binary name

test_get_service_by_volume_host_name()

Test idempotent id: 67ec6902-f91d-4dec-91fa-338523208bbc

Test getting volume service by volume host name

test_list_services()

Test idempotent id: e0218299-0a59-4f43-8b2b-f1c035b3d26d

Test listing volume services

volume.admin.test_volume_services_negative module

class VolumeServicesNegativeTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Negative tests of volume services

test_disable_log_reason_with_no_reason()

Test idempotent id: 77767b36-5e8f-4c68-a0b5-2308cc21ec64

Test disabling volume service with none reason should fail

test_disable_service_with_invalid_binary()

Test idempotent id: c571f179-c6e6-4c50-a0ab-368b628a8ac1

Test disabling volume service with invalid binary should fail

test_enable_service_with_invalid_host()

Test idempotent id: 3246ce65-ba70-4159-aa3b-082c28e4b484

Test enabling volume service with invalid host should fail

test_freeze_host_with_invalid_host()

Test idempotent id: 712bfab8-1f44-4eb5-a632-fa70bf78f05e

Test freezing volume service with invalid host should fail

test_thaw_host_with_invalid_host()

Test idempotent id: 7c6287c9-d655-47e1-9a11-76f6657a6dce

Test thawing volume service with invalid host should fail

volume.admin.test_volume_snapshot_quotas_negative module

class VolumeSnapshotQuotasNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Negative tests of volume snapshot quotas

test_quota_volume_gigabytes_snapshots()

Test idempotent id: c99a1ca9-6cdf-498d-9fdf-25832babef27

Test creating snapshot exceeding gigabytes quota should fail

test_quota_volume_snapshots()

Test idempotent id: 02bbf63f-6c05-4357-9d98-2926a94064ff

Test creating snapshot exceeding snapshots quota should fail

volume.admin.test_volume_type_access module

class VolumeTypesAccessTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test volume type access

test_volume_type_access_add()

Test idempotent id: d4dd0027-835f-4554-a6e5-50903fb79184

Test adding volume type access for non-admin project

test_volume_type_access_list()

Test idempotent id: 5220eb28-a435-43ce-baaf-ed46f0e95159

Test listing volume type access

volume.admin.test_volume_types module

class VolumeTypesTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test volume types

test_volume_crud_with_volume_type_and_extra_specs()

Test idempotent id: c03cc62c-f4e9-4623-91ec-64ce2f9c1260

Test create/update/get/delete volume with volume_type

test_volume_type_create_get_delete()

Test idempotent id: 4e955c3b-49db-4515-9590-0c99f8e471ad

Test create/get/delete volume type

test_volume_type_encryption_create_get_update_delete()

Test idempotent id: 7830abd0-ff99-4793-a265-405684a54d46

Test create/get/update/delete volume encryption type

test_volume_type_list()

Test idempotent id: 9d9b28e3-1b2e-4483-a2cc-24aa0ea1de54

Test listing volume types

test_volume_type_update()

Test idempotent id: cf9f07c6-db9e-4462-a243-5933ad65e9c8

Test updating volume type details

volume.admin.test_volume_types_extra_specs module

class VolumeTypesExtraSpecsTest(**args*, ***kwargs*)

Bases: `BaseVolumeAdminTest`

Test volume type extra specs

test_volume_type_extra_spec_create_get_delete()

Test idempotent id: d4772798-601f-408a-b2a5-29e8a59d1220

Test Create/Get/Delete volume type extra specs

test_volume_type_extra_specs_list()

Test idempotent id: b42923e9-0452-4945-be5b-d362ae533e60

Test listing volume type extra specs

test_volume_type_extra_specs_update()

Test idempotent id: 0806db36-b4a0-47a1-b6f3-c2e7f194d017

Test updating volume type extra specs

volume.admin.test_volume_types_extra_specs_negative module

class ExtraSpecsNegativeTest(**args*, ***kwargs*)

Bases: `BaseVolumeAdminTest`

Negative tests of volume type extra specs

test_create_invalid_body()

Test idempotent id: bc772c71-1ed4-4716-b945-8b5ed0f15e87

Test creating volume type extra spec with invalid POST body

Creating volume type extra spec with invalid POST body should fail.

test_create_none_body()

Test idempotent id: c821bdc8-43a4-4bf4-86c8-82f3858d5f7d

Test creating volume type extra spec with none POST body

Creating volume type extra spec with none POST body should fail.

test_create_nonexistent_type_id()

Test idempotent id: 49d5472c-a53d-4eab-a4d3-450c4db1c545

Test creating volume type extra specs for non existent volume type

Creating volume type extra specs for non existent volume type should fail.

test_delete_nonexistent_volume_type_id()

Test idempotent id: 031cda8b-7d23-4246-8bf6-bbe73fd67074

Test deleting volume type extra spec for non existent volume type

Deleting volume type extra spec for non existent volume type should fail.

test_get_nonexistent_extra_spec_name()

Test idempotent id: c881797d-12ff-4f1a-b09d-9f6212159753

Test getting volume type extra spec for non existent spec name

Getting volume type extra spec for non existent extra spec name should fail.

test_get_nonexistent_volume_type_id()

Test idempotent id: 9f402cbd-1838-4eb4-9554-126a6b1908c9

Test getting volume type extra spec for non existent volume type

Getting volume type extra spec for non existent volume type should fail.

test_list_nonexistent_volume_type_id()

Test idempotent id: dee5cf0c-cdd6-4353-b70c-e847050d71fb

Test listing volume type extra spec for non existent volume type

Listing volume type extra spec for non existent volume type should fail.

test_update_multiple_extra_spec()

Test idempotent id: a77dfda2-9100-448e-9076-ed1711f4bdfc

Test updating multiple volume type extra specs should fail

test_update_no_body()

Test idempotent id: 08961d20-5cbb-4910-ac0f-89ad6dbb2da1

Test updating volume type extra specs with no body should fail

test_update_none_extra_spec_id()

Test idempotent id: 9bf7a657-b011-4aec-866d-81c496fbe5c8

Test updating volume type extra specs without name

Updating volume type extra specs without extra spec name should fail.

test_update_nonexistent_extra_spec_id()

Test idempotent id: 25e5a0ee-89b3-4c53-8310-236f76c75365

Test updating volume type extra specs with non existent name

Updating volume type extra specs with non existent extra spec name should fail.

test_update_nonexistent_type_id()

Test idempotent id: 474090d2-0824-eb3b-9335-f506b4aa49d8

Test update volume type extra specs for non existent volume type

Update volume type extra specs for non existent volume type should fail.

volume.admin.test_volume_types_negative module

class VolumeTypesNegativeTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Negative tests of volume type

test_create_volume_type_encryption_nonexistent_type_id()

Test idempotent id: a5924b5f-b6c1-49ba-994c-b4af55d26e52

Test create encryption with nonexistent type id will fail

test_create_volume_with_private_volume_type()

Test idempotent id: 8c09f849-f225-4d78-ba87-bffd9a5e0c6f

Test creating volume with private volume type will fail

test_create_with_empty_name()

Test idempotent id: 878b4e57-faa2-4659-b0d1-ce740a06ae81

Test creating volume type with an empty name will fail

test_create_with_repeated_name()

Test idempotent id: 969b10c7-3d77-4e1b-a4f2-2d265980f7e5

Test creating volume type with a repeated name will fail

test_delete_nonexistent_type_id()

Test idempotent id: 6b3926d2-7d73-4896-bc3d-e42dfd11a9f6

Test deleting volume type with nonexistent type id will fail

test_get_nonexistent_type_id()

Test idempotent id: 994610d6-0476-4018-a644-a2602ef5d4aa

Test getting volume type with nonexistent type id will fail

volume.admin.test_volumes_actions module

class VolumesActionsTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test volume actions

test_force_detach_volume()

Test idempotent id: d38285d9-929d-478f-96a5-00e66a115b81

Test force detaching volume when its status is error

test_volume_force_delete_when_volume_is_attaching()

Test idempotent id: db8d607a-aa2e-4beb-b51d-d4005c232011

Test force deleting volume when its status is attaching

test_volume_force_delete_when_volume_is_creating()

Test idempotent id: 21737d5a-92f2-46d7-b009-a0cc0ee7a570

Test force deleting volume when its status is creating

test_volume_force_delete_when_volume_is_error()

Test idempotent id: 3e33a8a8-afd4-4d64-a86b-c27a185c5a4a

Test force deleting volume when its status is error

test_volume_force_delete_when_volume_is_maintenance()

Test idempotent id: b957cabd-1486-4e21-90cf-a9ed3c39dfb2

Test force deleting volume when its status is maintenance

test_volume_reset_status()

Test idempotent id: d063f96e-a2e0-4f34-8b8a-395c42de1845

Test resetting volume status

Reset volume status to available->error->available->maintenance

volume.admin.test_volumes_backup module

class VolumesBackupsAdminTest(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test volume backups

test_volume_backup_export_import()

Test idempotent id: a99c54a1-dd80-4724-8a13-13bf58d4068d

Test backup export import functionality.

Cinder allows exporting DB backup information through its API so it can be imported back in case of a DB loss.

test_volume_backup_reset_status()

Test idempotent id: 47a35425-a891-4e13-961c-c45deea21e94

Test resetting volume backup status to error

volume.admin.test_volumes_list module

class VolumesListAdminTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeAdminTest

Test listing volumes with admin privilege

test_volume_list_param_tenant()

Test idempotent id: 5866286f-3290-4cfd-a414-088aa6cdc469

Test admin can list volumes belonging to specified project

Module contents

Submodules

volume.api_microversion_fixture module

class APIMicroversionFixture(*volume_microversion*)

Bases: Fixture

volume.base module

class BaseVolumeAdminTest(**args*, ***kwargs*)

Bases: *BaseVolumeTest*

Base test case class for all Volume Admin API tests.

class BaseVolumeTest(**args*, ***kwargs*)

Bases: *BaseMicroversionTest*, BaseTestCase

Base test case class for all Cinder API tests.

volume.test_availability_zone module

class AvailabilityZoneTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeTest

Tests Availability Zone API List

test_get_availability_zone_list()

Test idempotent id: 01f1ae88-eba9-4c6b-a011-6f7ace06b725

Test listing volume available zones

volume.test_extensions module

class ExtensionsTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeTest

Test volume extensions

test_list_extensions()

Test idempotent id: 94607eb0-43a5-47ca-82aa-736b41bd2e2c

Test listing volume extensions

volume.test_image_metadata module

class VolumesImageMetadata(**args*, ***kwargs*)

Bases: BaseVolumeTest

Test volume image metadata

test_update_show_delete_image_metadata()

Test idempotent id: 03efff0b-5c75-4822-8f10-8789ac15b13e

Test update/show/delete volumes image metadata

volume.test_snapshot_metadata module

class SnapshotMetadataTestJSON(*args, **kwargs)

Bases: BaseVolumeTest

Test snapshot metadata

test_crud_snapshot_metadata()

Test idempotent id: a2f20f99-e363-4584-be97-bc33afb1a56c

Test create/get/update/delete snapshot metadata

test_update_show_snapshot_metadata_item()

Test idempotent id: e8ff85c5-8f97-477f-806a-3ac364a949ed

Test update/show snapshot metadata item

volume.test_versions module

class VersionsTest(*args, **kwargs)

Bases: BaseVolumeTest

Test volume versions

test_list_versions()

Test idempotent id: 77838fc4-b49b-4c64-9533-166762517369

Test listing volume versions

test_show_version()

Test idempotent id: 7f755ae2-caa9-4049-988c-331d8f7a579f

Test getting volume version details

volume.test_volume_absolute_limits module

class AbsoluteLimitsTests(*args, **kwargs)

Bases: BaseVolumeAdminTest

Test volume absolute limits

test_get_volume_absolute_limits()

Test idempotent id: 8e943f53-e9d6-4272-b2e9-adcf2f7c29ad

Test getting volume absolute limits

volume.test_volume_delete_cascade module

class VolumesDeleteCascade(*args, **kwargs)

Bases: BaseVolumeTest

Delete a volume with associated snapshots.

Cinder provides the ability to delete a volume with its associated snapshots. It is allow a volume and its snapshots to be removed in one operation both for usability and performance reasons.

test_volume_delete_cascade()

Test idempotent id: 994e2d40-de37-46e8-b328-a58fba7e4a95

Test deleting a volume with associated snapshots

The case validates the ability to delete a volume with associated snapshots.

test_volume_from_snapshot_cascade_delete()

Test idempotent id: 59a77ede-609b-4ee8-9f68-fc3c6ffe97b5

Test deleting a volume with associated volume-associated snapshot

The case validates the ability to delete a volume with associated snapshot while there is another volume created from that snapshot.

volume.test_volume_metadata module

class VolumesMetadataTest(*args, **kwargs)

Bases: BaseVolumeTest

Test volume metadata

test_crud_volume_metadata()

Test idempotent id: 6f5b125b-f664-44bf-910f-751591fe5769

Test creating, getting, updating and deleting of volume metadata

test_update_show_volume_metadata_item()

Test idempotent id: 862261c5-8df4-475a-8c21-946e50e36a20

Test updating and getting single volume metadata item

volume.test_volume_transfers module

class VolumesTransfersTest(*args, **kwargs)

Bases: BaseVolumeTest

Test volume transfer

test_create_get_list_accept_volume_transfer()

Test idempotent id: 4d75b645-a478-48b1-97c8-503f64242f1a

Test creating, getting, listing and accepting of volume transfer

test_create_list_delete_volume_transfer()

Test idempotent id: ab526943-b725-4c07-b875-8e8ef87a2c30

Test creating, listing and deleting volume transfer

class VolumesTransfersV355Test(*args, **kwargs)

Bases: VolumesTransfersTest

Test volume transfer for the new Transfers API mv 3.55

test_create_get_list_accept_volume_transfer()

Test idempotent id: 9f36bb2b-619f-4507-b246-76aeb9a28851

Test create, get, list, accept with volume-transfers API mv 3.55

test_create_list_delete_volume_transfer()

Test idempotent id: af4a5b97-0859-4f31-aa3c-85b05bb63322

Test create, list, delete with volume-transfers API mv 3.55

class VolumesTransfersV357Test(*args, **kwargs)

Bases: VolumesTransfersV355Test

Test volume transfer for the new Transfers API mv 3.57

test_create_get_list_accept_volume_transfer()

Test idempotent id: d746bd69-bb30-4414-9a1c-577959fac6a1

Test create, get, list, accept with volume-transfers API mv 3.57

test_create_list_delete_volume_transfer()

Test idempotent id: d4b20ec2-e1bb-4068-adcf-6c20020a8e05

Test create, list, delete with volume-transfers API mv 3.57

volume.test_volumes_actions module

class VolumesActionsTest(*args, **kwargs)

Bases: BaseVolumeTest

Test volume actions

test_attach_detach_volume_to_instance()

Test idempotent id: fff42874-7db5-4487-a8e1-ddda5fb5288d

Test attaching and detaching volume to instance

test_get_volume_attachment()

Test idempotent id: 9516a2c8-9135-488c-8dd6-5677a7e5f371

Test getting volume attachments

Attach a volume to a server, and then retrieve volumes attachments info.

test_reserve_unreserve_volume()

Test idempotent id: 92c4ef64-51b2-40c0-9f7e-4749fbaaba33

Test reserving and unreserving volume

test_volume_bootable()

Test idempotent id: 63e21b4c-0a0c-41f6-bfc3-7c2816815599

Test setting and retrieving bootable flag of a volume

test_volume_readonly_update()

Test idempotent id: fff74e1e-5bd3-4b33-9ea9-24c103bc3f59

Test updating and retrieve volumes readonly flag

test_volume_upload()

Test idempotent id: d8f1ca95-3d5b-44a3-b8ca-909691c9532d

Test uploading volume to create an image

volume.test_volumes_backup module

class VolumesBackupsTest(**args*, ***kwargs*)

Bases: BaseVolumeTest

Test volumes backup

test_backup_create_attached_volume()

Test idempotent id: 07af8f6d-80af-44c9-a5dc-c8427b1b62e6

Test backup create using force flag.

Cinder allows to create a volume backup, whether the volume status is available or in-use.

test_bootable_volume_backup_and_restore()

Test idempotent id: 2a8ba340-dff2-4511-9db7-646f07156b15

Test backuping and restoring a bootable volume

1. Create volume1 from image
2. Create backup1 from volume1
3. Restore backup1
4. Verify the restored backup volume is bootable

test_volume_backup_create_get_detailed_list_restore_delete()

Test idempotent id: a66eb488-8ee1-47d4-8e9f-575a095728c6

Test create/get/list/restore/delete volume backup

1. Create volume1 with metadata
2. Create backup1 from volume1
3. Show backup1
4. List backups with detail
5. Restore backup1
6. Verify backup1 has been restored successfully with the metadata of volume1

test_volume_backup_incremental()

Test idempotent id: f86eff09-2a6d-43c1-905e-8079e5754f1e

Test create a backup when latest incremental backup is deleted

class VolumesBackupsV39Test(*args, **kwargs)

Bases: BaseVolumeTest

Test volumes backup with volume microversion greater than 3.8

test_update_backup()

Test idempotent id: 9b374cbc-be5f-4d37-8848-7efb8a873dcc

Test updating backups name and description

volume.test_volumes_clone module

class VolumesCloneTest(*args, **kwargs)

Bases: BaseVolumeTest

Test volume clone

test_create_from_bootable_volume()

Test idempotent id: cbbcd7c6-5a6c-481a-97ac-ca55ab715d16

Test cloning a bootable volume

test_create_from_volume()

Test idempotent id: 9adae371-a257-43a5-9555-dc7c88e66e0e

Test cloning a volume with increasing size

volume.test_volumes_clone_negative module

class VolumesCloneNegativeTest(*args, **kwargs)

Bases: BaseVolumeTest

Negative tests of volume clone

test_create_from_volume_decreasing_size()

Test idempotent id: 9adae371-a257-43a5-459a-dc7c88e66e0e

Test cloning a volume with decreasing size will fail

volume.test_volumes_extend module

class BaseVolumesExtendAttachedTest(*args, **kwargs)

Bases: BaseVolumeTest

Tests extending the size of an attached volume.

class VolumesExtendAttachedTest(*args, **kwargs)

Bases: *BaseVolumesExtendAttachedTest*

test_extend_attached_volume()

Test idempotent id: 301f5a30-1c6f-4ea0-be1a-91fd28d44354

class VolumesExtendTest(*args, **kwargs)

Bases: BaseVolumeTest

Test volume extend

test_volume_extend()

Test idempotent id: 9a36df71-a257-43a5-9555-dc7c88e66e0e

Test extend a volume

test_volume_extend_when_volume_has_snapshot()

Test idempotent id: 86be1cba-2640-11e5-9c82-635fb964c912

Test extending a volume which has a snapshot

volume.test_volumes_get module

class VolumesGetTest(**args*, ***kwargs*)

Bases: BaseVolumeTest

Test getting volume info

test_volume_create_get_update_delete()

Test idempotent id: 27fb0e9f-fb64-41dd-8bdb-1ffa762f0d51

Test Create/Get/Update/Delete of a blank volume

test_volume_create_get_update_delete_as_clone()

Test idempotent id: 3f591b4a-7dc6-444c-bd51-77469506b3a1

Test Create/Get/Update/Delete of a cloned volume

test_volume_create_get_update_delete_from_image()

Test idempotent id: 54a01030-c7fc-447c-86ee-c1182beae638

Test Create/Get/Update/Delete of a volume created from image

class VolumesSummaryTest(**args*, ***kwargs*)

Bases: BaseVolumeTest

Test volume summary

test_show_volume_summary()

Test idempotent id: c4f2431e-4920-4736-9e00-4040386b6feb

Test showing volume summary

volume.test_volumes_list module

class VolumesListTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeTest

Test listing volumes

NOTE: This test creates a number of 1G volumes. To run it successfully, ensure that the backing file for the volume group that Cinder uses has space for at least 3 1G volumes! If you are running a Devstack environment, ensure that the VOLUME_BACKING_FILE_SIZE is at least 4G in your localrc

test_volume_list()

Test idempotent id: 0b6ddd39-b948-471f-8038-4787978747c4

Test getting a list of volumes

test_volume_list_by_name()

Test idempotent id: a28e8da4-0b56-472f-87a8-0f4d3f819c02

Test getting a list of volumes filtered by volume name

test_volume_list_details_by_name()

Test idempotent id: 2de3a6d4-12aa-403b-a8f2-fdeb42a89623

Test getting a list of detailed volumes filtered by volume name

test_volume_list_details_pagination()

Test idempotent id: e9138a2c-f67b-4796-8efa-635c196d01de

Test listing volumes with details by pagination

All volumes will be returned by multiple requests, and the number of limit volumes will be returned at a time.

test_volume_list_details_with_multiple_params()

Test idempotent id: 2a7064eb-b9c3-429b-b888-33928fc5edd3

Test listing volumes detail using combined filtering condition

test_volume_list_pagination()

Test idempotent id: af55e775-8e4b-4feb-8719-215c43b0238c

Test listing volumes by pagination

All volumes will be returned by multiple requests, and the number of limit volumes will be returned at a time.

test_volume_list_param_display_name_and_status()

Test idempotent id: 777c87c1-2fc4-4883-8b8e-5c0b951d1ec8

Test listing volume when display name and status param is given

test_volume_list_with_detail_param_display_name_and_status()

Test idempotent id: 856ab8ca-6009-4c37-b691-be1065528ad4

Test listing volume when name and status param is given

test_volume_list_with_detail_param_marker()

Test idempotent id: 46eff077-100b-427f-914e-3db2abcdb7e2

Test listing volumes with details from the specified marker

Choose a volume id from all volumes as a marker, list volumes with that marker, only volumes with id greater than marker will be returned.

test_volume_list_with_detail_param_metadata()

Test idempotent id: 1ca92d3c-4a8e-4b43-93f5-e4c7fb3b291d

Test listing volumes details when metadata param is given

test_volume_list_with_details()

Test idempotent id: adcbb5a7-5ad8-4b61-bd10-5380e111a877

Test getting a list of detailed volumes

test_volume_list_with_param_metadata()

Test idempotent id: b5ebea1b-0603-40a0-bb41-15fcd0a53214

Test listing volumes when metadata param is given

test_volumes_list_by_availability_zone()

Test idempotent id: c0cfa863-3020-40d7-b587-e35f597d5d87

Test getting a list of volumes filtered by availability zone

test_volumes_list_by_bootable()

Test idempotent id: 2016a942-3020-40d7-95ce-7613bf8407ce

Check out volumes.

This test function is aim at check out whether all of the volumes in volume_list are not a bootable volume.

test_volumes_list_by_status()

Test idempotent id: 39654e13-734c-4dab-95ce-7613bf8407ce

Test getting a list of volumes filtered by volume status

test_volumes_list_details_by_availability_zone()

Test idempotent id: e1b80d13-94f0-4ba2-a40e-386af29f8db1

Test getting a list of detailed volumes by availability zone

test_volumes_list_details_by_bootable()

Test idempotent id: 2016a939-72ec-482a-bf49-d5ca06216b9f

Test getting a list of detailed volumes filtered by bootable

test_volumes_list_details_by_status()

Test idempotent id: 2943f712-71ec-482a-bf49-d5ca06216b9f

Test getting a list of detailed volumes filtered by status

volume.test_volumes_negative module

class VolumesNegativeTest(**args*, ***kwargs*)

Bases: BaseVolumeTest

Negative tests of volumes

test_attach_volumes_with_nonexistent_volume_id()

Test idempotent id: f5e56b0a-5d02-43c1-a2a7-c9b792c2e3f6

Test attaching non existent volume to server should fail

test_create_volume_from_deactivated_image()

Test idempotent id: d15e7f35-2cfc-48c8-9418-c8223a89bcbb

Test creating volume from deactivated image should fail

test_create_volume_from_image_with_decreasing_size()

Test idempotent id: 5b810c91-0ad1-47ce-aee8-615f789be78f

Test creating volume from image with decreasing size should fail

test_create_volume_with_invalid_size()

Test idempotent id: 1ed83a8a-682d-4dfb-a30e-ee63ffd6c049

Test creating volume with invalid size should fail

test_create_volume_with_nonexistent_snapshot_id()

Test idempotent id: 0c36f6ae-4604-4017-b0a9-34fdc63096f9

Test creating volume with non existent snapshot should fail

test_create_volume_with_nonexistent_source_volid()

Test idempotent id: 47c73e08-4be8-45bb-bfdf-0c4e79b88344

Test creating volume with non existent source volume should fail

test_create_volume_with_nonexistent_volume_type()

Test idempotent id: 10254ed8-3849-454e-862e-3ab8e6aa01d2

Test creating volume with non existent volume type should fail

test_create_volume_with_size_negative()

Test idempotent id: 8b472729-9eba-446e-a83b-916bdb34bef7

Test creating volume with negative size should fail

test_create_volume_with_size_zero()

Test idempotent id: 41331caa-eaf4-4001-869d-bc18c1869360

Test creating volume with zero size should fail

test_create_volume_without_passing_size()

Test idempotent id: 9387686f-334f-4d31-a439-33494b9e2683

Test creating volume with empty size should fail

test_delete_invalid_volume_id()

Test idempotent id: 1f035827-7c32-4019-9240-b4ec2dbd9dfd

Test deleting volume with invalid volume id should fail

test_delete_volume_without_passing_volume_id()

Test idempotent id: 441a1550-5d44-4b30-af0f-a6d402f52026

Test deleting volume with empty volume id should fail

test_detach_volumes_with_invalid_volume_id()

Test idempotent id: 9f9c24e4-011d-46b5-b992-952140ce237a

Test detaching volume with invalid volume id should fail

test_get_invalid_volume_id()

Test idempotent id: 30799cfd-7ee4-446c-b66c-45b383ed211b

Test getting volume with invalid volume id should fail

test_get_volume_without_passing_volume_id()

Test idempotent id: c6c3db06-29ad-4e91-beb0-2ab195fe49e3

Test getting volume with empty volume id should fail

test_list_volumes_detail_with_invalid_status()

Test idempotent id: ba94b27b-be3f-496c-a00e-0283b373fa75

Test listing volume details with invalid status

Listing volume details with invalid status should get nothing

test_list_volumes_detail_with_nonexistent_name()

Test idempotent id: 9ca17820-a0e7-4cbd-a7fa-f4468735e359

Test listing volume details with non existent name

Listing volume details with non existent name should get nothing.

test_list_volumes_with_invalid_status()

Test idempotent id: 143b279b-7522-466b-81be-34a87d564a7c

Test listing volumes with invalid status should get nothing

test_list_volumes_with_nonexistent_name()

Test idempotent id: 0f4aa809-8c7b-418f-8fb3-84c7a5dfc52f

Test listing volumes with non existent name should get nothing

test_reserve_volume_with_negative_volume_status()

Test idempotent id: 449c4ed2-ecdd-47bb-98dc-072aeccf158c

Test reserving already reserved volume should fail

test_reserve_volume_with_nonexistent_volume_id()

Test idempotent id: ac6084c0-0546-45f9-b284-38a367e0e0e2

Test reserving non existent volume should fail

test_unreserve_volume_with_nonexistent_volume_id()

Test idempotent id: eb467654-3dc1-4a72-9b46-47c29d22654c

Test unreserving non existent volume should fail

test_update_volume_with_empty_volume_id()

Test idempotent id: 72aeca85-57a5-4c1f-9057-f320f9ea575b

Test updating volume with empty volume id should fail

test_update_volume_with_invalid_volume_id()

Test idempotent id: e66e40d6-65e6-4e75-bdc7-636792fa152d

Test updating volume with invalid volume id should fail

test_update_volume_with_nonexistent_volume_id()

Test idempotent id: 0186422c-999a-480e-a026-6a665744c30c

Test updating non existent volume should fail

test_volume_delete_nonexistent_volume_id()

Test idempotent id: 555efa6e-efcd-44ef-8a3b-4a7ca4837a29

Test deleting non existent volume should fail

test_volume_extend_with_None_size()

Test idempotent id: 355218f1-8991-400a-a6bb-971239287d92

Test extending volume with none size should fail

test_volume_extend_with_non_number_size()

Test idempotent id: 5d0b480d-e833-439f-8a5a-96ad2ed6f22f

Test extending volume with non-integer size should fail

test_volume_extend_with_nonexistent_volume_id()

Test idempotent id: 8f05a943-013c-4063-ac71-7baf561e82eb

Test extending non existent volume should fail

test_volume_extend_with_size_smaller_than_original_size()

Test idempotent id: e0c75c74-ee34-41a9-9288-2a2051452854

Test extending volume with decreasing size should fail

test_volume_extend_without_passing_volume_id()

Test idempotent id: aff8ba64-6d6f-4f2e-bc33-41a08ee9f115

Test extending volume without passing volume id should fail

test_volume_get_nonexistent_volume_id()

Test idempotent id: f131c586-9448-44a4-a8b0-54ca838aa43e

Test getting non existent volume should fail

volume.test_volumes_snapshots module

class VolumesSnapshotTestJSON(**args*, ***kwargs*)

Bases: `BaseVolumeTest`

Test volume snapshots

test_snapshot_backup()

Test idempotent id: bbcfa285-af7f-479e-8c1a-8c34fc16543c

Test creating backup from snapshot and volume

1. Create snapshot1 from volume1
2. Create backup from volume1 and snapshot1
3. Check the created backups volume is volume1 and snapshot is snapshot1

test_snapshot_create_delete_with_volume_in_use()

Test idempotent id: 8567b54c-4455-446d-a1cf-651ddeaa3ff2

Test create/delete snapshot from volume attached to server

test_snapshot_create_get_list_update_delete()

Test idempotent id: 2a8abbe4-d871-46db-b049-c41f5af8216e

Test create/get/list/update/delete snapshot

test_snapshot_create_offline_delete_online()

Test idempotent id: 5210a1de-85a0-11e6-bb21-641c676a5d61

Test creating snapshots when volume is detached and attached

1. Create snapshot1 from volume1(not attached to any server)
2. Attach volume1 to server1
3. Create snapshot2 and snapshot3 from volume1
4. Delete snapshot3, snapshot1, snapshot2

test_volume_from_snapshot()

Test idempotent id: 677863d1-3142-456d-b6ac-9924f667a7f4

Test creating volume from snapshot with extending size

test_volume_from_snapshot_no_size()

Test idempotent id: 053d8870-8282-4fff-9dbb-99cb58bb5e0a

Test creating volume from snapshot with original size

volume.test_volumes_snapshots_list module

class VolumesSnapshotListTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeTest

Test listing volume snapshots

test_snapshot_list_param_limit()

Test idempotent id: db4d8e0a-7a2e-41cc-a712-961f6844e896

Test listing snapshot with limit returns the limited elements

If listing snapshots with limit=1, then 1 snapshot is returned.

test_snapshot_list_param_limit_equals_infinite()

Test idempotent id: a1427f61-420e-48a5-b6e3-0b394fa95400

Test listing snapshot with infinite limit

If listing snapshots with limit greater than the count of all snapshots, then all snapshots are returned.

test_snapshot_list_param_limit_equals_zero()

Test idempotent id: e3b44b7f-ae87-45b5-8a8c-66110eb24d0a

Test listing snapshot with zero limit should return empty list

test_snapshot_list_param_marker()

Test idempotent id: 05489dde-44bc-4961-a1f5-3ce7ee7824f7

Test listing snapshots with marker

The list of snapshots should end before the provided marker

test_snapshot_list_param_offset()

Test idempotent id: ca96d551-17c6-4e11-b0e8-52d3bb8a63c7

Test listing snapshots with offset and limit

If listing snapshots with offset=2 and limit=3, then at most 3(limit) snapshots located in the position 2(offset) in the all snapshots list should be returned. (The items in the all snapshots list start from position 0.)

test_snapshot_list_param_sort_created_at_asc()

Test idempotent id: 4052c3a0-2415-440a-a8cc-305a875331b0

Test listing snapshots sort by created_at ascendingly

test_snapshot_list_param_sort_created_at_desc()

Test idempotent id: dcbbe24a-f3c0-4ec8-9274-55d48db8d1cf

Test listing snapshots sort by created_at descendingly

test_snapshot_list_param_sort_id_asc()

Test idempotent id: c5513ada-64c1-4d28-83b9-af3307ec1388

Test listing snapshots sort by id ascendingly

test_snapshot_list_param_sort_id_desc()

Test idempotent id: 8a7fe058-0b41-402a-8afd-2dbc5a4a718b

Test listing snapshots sort by id descendingly

test_snapshot_list_param_sort_name_asc()

Test idempotent id: d58b5fed-0c37-42d3-8c5d-39014ac13c00

Test listing snapshots sort by display_name ascendingly

test_snapshot_list_param_sort_name_desc()

Test idempotent id: 96ba6f4d-1f18-47e1-b4bc-76edc6c21250

Test listing snapshots sort by display_name descendingly

test_snapshots_list_details_with_params()

Test idempotent id: 220a1022-1fcd-4a74-a7bd-6b859156cda2

Test listing snapshot details with params

test_snapshots_list_with_params()

Test idempotent id: 59f41f43-aebf-48a9-ab5d-d76340fab32b

Test listing snapshots with params

volume.test_volumes_snapshots_negative module

class VolumesSnapshotNegativeTestJSON(**args*, ***kwargs*)

Bases: BaseVolumeTest

Negative tests of volume snapshot

test_create_snapshot_with_nonexistent_volume_id()

Test idempotent id: e3e466af-70ab-4f4b-a967-ab04e3532ea7

Test creating snapshot from non existent volume should fail

test_create_snapshot_without_passing_volume_id()

Test idempotent id: bb9da53e-d335-4309-9c15-7e76fd5e4d6d

Test creating snapshot without passing volume_id should fail

test_list_snapshot_invalid_param_limit()

Test idempotent id: 8fd92339-e22f-4591-86b4-1e2215372a40

Test listing snapshots with invalid limit param should fail

test_list_snapshots_invalid_param_marker()

Test idempotent id: b68deeda-ca79-4a32-81af-5c51179e553a

Test listing snapshots with invalid marker should fail

test_list_snapshots_invalid_param_sort()

Test idempotent id: 27b5f37f-bf69-4e8c-986e-c44f3d6819b8

Test listing snapshots with invalid sort key should fail

test_volume_from_snapshot_decreasing_size()

Test idempotent id: 677863d1-34f9-456d-b6ac-9924f667a7f4

Test creating volume from snapshot with decreasing size

creating volume from snapshot with decreasing size should fail.

Module contents

Serial Tests

serial_tests

serial_tests package

Subpackages

serial_tests.api package

Subpackages

serial_tests.api.compute package

Subpackages

serial_tests.api.compute.admin package

Submodules

serial_tests.api.compute.admin.test_aggregates module

class AggregatesAdminTestBase(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Tests Aggregates API that require admin privileges

class AggregatesAdminTestJSON(**args*, ***kwargs*)

Bases: *AggregatesAdminTestBase*

Tests Aggregates API that require admin privileges

test_aggregate_add_host_create_server_with_az()

Test idempotent id: 96be03c7-570d-409c-90f8-e4db3c646996

Test adding a host to the given aggregate and creating a server

test_aggregate_add_host_get_details()

Test idempotent id: eeef473c-7c52-494d-9f09-2ed7fc8fc036

Test showing aggregate contains the host added to the aggregate

Add a host to the given aggregate and get details.

test_aggregate_add_host_list()

Test idempotent id: 7f6a1cc5-2446-4cdb-9baa-b6ae0a919b72

Test listing aggregate contains the host added to the aggregate

Add a host to the given aggregate and list.

test_aggregate_add_remove_host()

Test idempotent id: c8e85064-e79b-4906-9931-c11c24294d02

Test adding host to and removing host from aggregate

test_aggregate_create_delete()

Test idempotent id: 0d148aa3-d54c-4317-aa8d-42040a475e20

Test create/delete aggregate

test_aggregate_create_delete_with_az()

Test idempotent id: 5873a6f8-671a-43ff-8838-7ce430bb6d0b

Test create/delete aggregate with availability_zone

test_aggregate_create_update_metadata_get_details()

Test idempotent id: 36ec92ca-7a73-43bc-b920-7531809e8540

Test set/get aggregate metadata

test_aggregate_create_update_with_az()

Test idempotent id: 4d2b2004-40fa-40a1-aab2-66f4dab81beb

Test create/update aggregate with availability_zone

test_aggregate_create_verify_entry_in_list()

Test idempotent id: 68089c38-04b1-4758-bdf0-cf0daec4defd

Test listing aggregate should contain the created aggregate

class AggregatesAdminTestV241(**args*, ***kwargs*)

Bases: *AggregatesAdminTestBase*

Tests Aggregates API that require admin privileges

Tests Aggregates API that require admin privileges with compute microversion greater than 2.40.

test_create_update_show_aggregate_add_remove_host()

Test idempotent id: fdf24d9e-8afa-4700-b6aa-9c498351504f

Test response schema of aggregates API

Test response schema of aggregates API(create/update/show/add host/ remove host) with compute microversion greater than 2.40.

serial_tests.api.compute.admin.test_server_affinity module

class ServersAffinityTest(**args*, ***kwargs*)

Bases: BaseV2ComputeAdminTest

Test creating servers without multi-create with scheduler_hints.

The server affinity tests in ServersOnMultiNodesTest use the /servers multi-create API and therefore do not test affinity behavior when server group members already exist on hosts.

These tests must be run in serial because they will be disabling compute hosts in order to verify affinity behavior.

test_create_server_with_affinity()

Test idempotent id: 28ef4c29-09db-40a8-aacd-dc5fa321f35e

test_create_server_with_affinity_negative()

Test idempotent id: 3ac1ff4e-0fa2-4069-ae59-695cf829275b

test_create_server_with_anti_affinity()

Test idempotent id: 475e9db0-5512-41cb-a6b2-4bd6fb3c7603

test_create_server_with_anti_affinity_max_server_per_host()

Test idempotent id: 88a8c3d4-c0e8-4873-ba6f-006004779f29

test_create_server_with_anti_affinity_negative()

Test idempotent id: c5e43585-0fdd-42a9-a525-2b99465c28df

test_create_server_with_soft_affinity()

Test idempotent id: 99cf4819-479c-4176-a9a6-ad501f6fc4b7

test_create_server_with_soft_anti_affinity()

Test idempotent id: ef2bc189-5ecc-4a23-8c1b-0d70a9138a77

Module contents

Module contents

Module contents

serial_tests.scenario package

Submodules

serial_tests.scenario.test_aggregates_basic_ops module

class TestAggregatesBasicOps(**args*, ***kwargs*)

Bases: ScenarioTest

Creates an aggregate within an availability zone

Adds a host to the aggregate Checks aggregate details Updates aggregates name Removes host from aggregate Deletes aggregate

test_aggregate_basic_ops()

Test idempotent id: cb2b4c4f-0c7c-4164-bdde-6285b302a081

Module contents

Module contents

CHAPTER

FOUR

FOR CONTRIBUTORS

- If you are a new contributor to Tempest please refer: *So You Want to Contribute*

4.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with Tempest.

4.1.1 Communication

- IRC channel `#openstack-qa` at OFTC
- Mailing list (prefix subjects with `[qa]` for faster responses)

4.1.2 Contacting the Core Team

Please refer to the Tempest Core Team contacts.

4.1.3 New Feature Planning

If you want to propose a new feature please read Feature Proposal Process Tempest features are tracked on Launchpad BP. It also helps to bring the feature up during the next PTG and contact the current PTL of QA project. Information about PTG is always posted on the Mailing list.

4.1.4 Task Tracking

We track our tasks in Launchpad.

If youre looking for some smaller, easier work items to pick up and get started on, search for the low-hanging-fruit tag.

4.1.5 Reporting a Bug

Have you found an issue and want to make sure we are aware of it? You can do so on Launchpad. More info about Launchpad usage can be found on OpenStack docs page

4.1.6 Getting Your Patch Merged

All changes proposed to the Tempest require a single `Code-Review +2` vote from a Tempest core followed by a `Workflow +1` vote. More detailed guidelines for reviewers are available at *Reviewing Tempest Code*.

4.1.7 Project Team Lead Duties

All common PTL duties are enumerated in the PTL guide.

The Release Process for QA is documented in QA Release Process.

CHAPTER

FIVE

DEVELOPERS GUIDE

5.1 Development

5.1.1 Tempest Coding Guide

- Step 1: Read the OpenStack Style Commandments https://docs.openstack.org/hacking/latest/
- Step 2: Read on

Tempest Specific Commandments

- [T102] Cannot import OpenStack python clients in tempest/api & tempest/scenario tests
- [T104] Scenario tests require a services decorator
- [T105] Tests cannot use setUpClass/tearDownClass
- [T107] Check that a service tag isnt in the module path
- [T108] Check no hyphen at the end of rand_name() argument
- [T109] Cannot use testtools.skip decorator; instead use decorators.skip_because from tempest.lib
- [T110] Check that service client names of GET should be consistent
- [T111] Check that service client names of DELETE should be consistent
- [T112] Check that tempest.lib should not import local tempest code
- [T113] Check that tests use data_utils.rand_uuid() instead of uuid.uuid4()
- [T114] Check that tempest.lib does not use tempest config
- [T115] Check that admin tests should exist under admin path
- [N322] Methods default argument shouldnt be mutable
- [T116] Unsupported message Exception attribute in PY3
- [T117] Check negative tests have `@decorators.attr(type=['negative'])` applied.
- [T118] LOG.warn is deprecated. Enforce use of LOG.warning.

It is recommended to use `tox -eautopep8` before submitting a patch.

Test Data/Configuration

- Assume nothing about existing test data
- Tests should be self contained (provide their own data)
- Clean up test data at the completion of each test
- Use configuration files for values that will vary by environment

Supported OpenStack Components

Tempests *Tempest Library Documentation* and *plugin interface* can be leveraged to support integration testing for virtually any OpenStack component.

However, Tempest only offers **in-tree** integration testing coverage for the following components:

- Cinder
- Glance
- Keystone
- Neutron
- Nova
- Swift

Historically, Tempest offered in-tree testing for other components as well, but since the introduction of the External Plugin Interface, Tempests in-tree testing scope has been limited to the projects above. Integration tests for projects not included above should go into one of the relevant plugin projects.

Exception Handling

According to the `The Zen of Python` the `Errors should never pass silently.` Tempest usually runs in special environment (jenkins gate jobs), in every error or failure situation we should provide as much error related information as possible, because we usually do not have the chance to investigate the situation after the issue happened.

In every test case the abnormal situations must be very verbosely explained, by the exception and the log.

In most cases the very first issue is the most important information.

Try to avoid using `try` blocks in the test cases, as both the `except` and `finally` blocks could replace the original exception, when the additional operations leads to another exception.

Just letting an exception to propagate, is not a bad idea in a test case, at all.

Try to avoid using any exception handling construct which can hide the errors origin.

If you really need to use a `try` block, please ensure the original exception at least logged. When the exception is logged you usually need to `raise` the same or a different exception anyway.

Use of `self.addCleanup` is often a good way to avoid having to catch exceptions and still ensure resources are correctly cleaned up if the test fails part way through.

Use the `self.assert*` methods provided by the unit test framework. This signals the failures early on.

Avoid using the `self.fail` alone, its stack trace will signal the `self.fail` line as the origin of the error.

Avoid constructing complex boolean expressions for assertion. The `self.assertTrue` or `self.assertFalse` without a `msg` argument, will just tell you the single boolean value, and you will not know anything about the values used in the formula, the `msg` argument might be good enough for providing more information.

Most other assert method can include more information by default. For example `self.assertIn` can include the whole set.

It is recommended to use testtools matcher for the more tricky assertions. You can implement your own specific matcher as well.

If the test case fails you can see the related logs and the information carried by the exception (exception class, backtrack and exception info). This and the service logs are your only guide to finding the root cause of flaky issues.

Test cases are independent

Every `test_method` must be callable individually and MUST NOT depends on, any other `test_method` or `test_method` ordering.

Test cases MAY depend on commonly initialized resources/facilities, like credentials management, testresources and so on. These facilities, MUST be able to work even if just one `test_method` is selected for execution.

Service Tagging

Service tagging is used to specify which services are exercised by a particular test method. You specify the services with the `tempest.common.utils.services` decorator. For example:

```
@utils.services('compute', 'image')
```

Valid service tag names are the same as the list of directories in tempest.api that have tests.

For scenario tests having a service tag is required. For the API tests service tags are only needed if the test method makes an API call (either directly or indirectly through another service) that differs from the parent directory name. For example, any test that make an API call to a service other than Nova in `tempest.api.compute` would require a service tag for those services, however they do not need to be tagged as `compute`.

Test Attributes

Tempest leverages test attributes which are a simple but effective way of distinguishing between different types of API tests. A test can be tagged with such attributes using the `decorators.attr` decorator, for example:

```
@decorators.attr(type=['negative'])
def test_aggregate_create_aggregate_name_length_less_than_1(self):
    [...]
```

These test attributes can be used for test selection via regular expressions. For example, `(?!.*\[.*\bslow\b.*\])(^tempest\.scenario)` runs all the tests in the `scenario` test module, *except* for those tagged with the `slow` attribute (via a negative lookahead in the regular expression). These attributes are used in Tempests `tox.ini` as well as Tempests Zuul job definitions for specifying particular batches of Tempest test suites to run.

Negative Attribute

The `type='negative'` attribute is used to signify that a test is a negative test, which is a test that handles invalid input gracefully. This attribute should be applied to all negative test scenarios.

This attribute must be applied to each test that belongs to a negative test class, i.e. a test class name ending with Negative.* substring.

Slow Attribute

The `type='slow'` attribute is used to signify that a test takes a long time to run, relatively speaking. This attribute is usually applied to *scenario tests*, which involve a complicated series of API operations, the total runtime of which can be relatively long. This long runtime has performance implications on Zuul jobs, which is why the `slow` attribute is leveraged to run slow tests on a selective basis, to keep total Zuul job runtime down to a reasonable time frame.

Smoke Attribute

The `type='smoke'` attribute is used to signify that a test is a so-called smoke test, which is a type of test that tests the most vital OpenStack functionality, like listing servers or flavors or creating volumes. The attribute should be sparingly applied to only the tests that sanity-check the most essential functionality of an OpenStack cloud.

Multinode Attribute

The `type='multinode'` attribute is used to signify that a test is desired to be executed in a multinode environment. By marking the tests with this attribute we can avoid running tests which arent that beneficial for the multinode setup and thus reduce the consumption of resources.

Test fixtures and resources

Test level resources should be cleaned-up after the test execution. Clean-up is best scheduled using `addCleanup` which ensures that the resource cleanup code is always invoked, and in reverse order with respect to the creation order.

Test class level resources should be defined in the `resource_setup` method of the test class, except for any credential obtained from the credentials provider, which should be set-up in the `setup_credentials` method. Cleanup is best scheduled using `addClassResourceCleanup` which ensures that the cleanup code is always invoked, and in reverse order with respect to the creation order.

In both cases - test level and class level cleanups - a wait loop should be scheduled before the actual delete of resources with an asynchronous delete.

The test base class `BaseTestCase` defines Tempest framework for class level fixtures. `setUpClass` and `tearDownClass` are defined here and cannot be overwritten by subclasses (enforced via hacking rule T105).

Set-up is split in a series of steps (setup stages), which can be overwritten by test classes. Set-up stages are:

- `skip_checks`
- `setup_credentials`
- `setup_clients`
- `resource_setup`

Tear-down is also split in a series of steps (teardown stages), which are stacked for execution only if the corresponding setup stage had been reached during the setup phase. Tear-down stages are:

- `clear_credentials` (defined in the base test class)
- `resource_cleanup`

Skipping Tests

Skipping tests should be based on configuration only. If that is not possible, it is likely that either a configuration flag is missing, or the test should fail rather than be skipped. Using discovery for skipping tests is generally discouraged.

When running a test that requires a certain feature in the target cloud, if that feature is missing we should fail, because either the test configuration is invalid, or the cloud is broken and the expected feature is not there even if the cloud was configured with it.

Negative Tests

Error handling is an important aspect of API design and usage. Negative tests are a way to ensure that an application can gracefully handle invalid or unexpected input. However, as a black box integration test suite, Tempest is not suitable for handling all negative test cases, as the wide variety and complexity of negative tests can lead to long test runs and knowledge of internal implementation details. The bulk of negative testing should be handled with project function tests. All negative tests should be based on API-WG guideline . Such negative tests can block any changes from accurate failure code to invalid one.

If facing some gray area which is not clarified on the above guideline, propose a new guideline to the API-WG. With a proposal to the API-WG we will be able to build a consensus across all OpenStack projects and improve the quality and consistency of all the APIs.

In addition, we have some guidelines for additional negative tests.

- About BadRequest(HTTP400) case: We can add a single negative tests of BadRequest for each resource and method(POST, PUT). Please dont implement more negative tests on the same combination of resource and method even if API request parameters are different from the existing test.
- About NotFound(HTTP404) case: We can add a single negative tests of NotFound for each resource and method(GET, PUT, DELETE, HEAD). Please dont implement more negative tests on the same combination of resource and method.

The above guidelines dont cover all cases and we will grow these guidelines organically over time. Patches outside of the above guidelines are left up to the reviewers discretion and if we face some conflicts between reviewers, we will expand the guideline based on our discussion and experience.

Test skips because of Known Bugs

If a test is broken because of a bug it is appropriate to skip the test until bug has been fixed. You should use the `skip_because` decorator so that Tempests skip tracking tool can watch the bug status.

Example:

```
@skip_because(bug="980688")
def test_this_and_that(self):
    ...
```

Guidelines

- Do not submit changesets with only testcases which are skipped as they will not be merged.
- Consistently check the status code of responses in testcases. The earlier a problem is detected the easier it is to debug, especially where there is complicated setup required.

Parallel Test Execution

Tempest by default runs its tests in parallel this creates the possibility for interesting interactions between tests which can cause unexpected failures. Dynamic credentials provides protection from most of the potential race conditions between tests outside the same class. But there are still a few of things to watch out for to try to avoid issues when running your tests in parallel.

- Resources outside of a project scope still have the potential to conflict. This is a larger concern for the admin tests since most resources and actions that require admin privileges are outside of projects.
- Races between methods in the same class are not a problem because parallelization in Tempest is at the test class level, but if there is a json and xml version of the same test class there could still be a race between methods.
- The rand_name() function from tempest.lib.common.utils.data_utils should be used anywhere a resource is created with a name. Static naming should be avoided to prevent resource conflicts.
- If the execution of a set of tests is required to be serialized then locking can be used to perform this. See usage of `LockFixture` for examples of using locking. However, LockFixture only helps if you want to separate the execution of two small sets of test cases. On the other hand, if you need to run a set of tests separately from potentially all other tests then `LockFixture` does not scale as you would need to take the lock in all the other tests too. In this case, you can use the `@serial` decorator on top of the test class holding the tests that need to run separately from the potentially parallel test set. See more in *Tempest Test Writing Guide*.

Sample Configuration File

The sample config file is autogenerated using a script. If any changes are made to the config variables in tempest/config.py then the sample config file must be regenerated. This can be done running:

```
tox -e genconfig
```

Unit Tests

Unit tests are a separate class of tests in Tempest. They verify Tempest itself, and thus have a different set of guidelines around them:

1. They can not require anything running externally. All you should need to run the unit tests is the git tree, python and the dependencies installed. This includes running services, a config file, etc.
2. The unit tests cannot use setUpClass, instead fixtures and testresources should be used for shared state between tests.

Test Documentation

For tests being added we need to require inline documentation in the form of docstrings to explain what is being tested. In API tests for a new API a class level docstring should be added to an API reference doc. If one doesnt exist a TODO comment should be put indicating that the reference needs to be added.

For individual API test cases a method level docstring should be used to explain the functionality being tested if the test name isnt descriptive enough. For example:

```
def test_get_role_by_id(self):
    """Get a role by its id."""
```

the docstring there is superfluous and shouldnt be added. but for a method like:

```
def test_volume_backup_create_get_detailed_list_restore_delete(self):
    pass
```

a docstring would be useful because while the test title is fairly descriptive the operations being performed are complex enough that a bit more explanation will help people figure out the intent of the test.

For scenario tests a class level docstring describing the steps in the scenario is required. If there is more than one test case in the class individual docstrings for the workflow in each test methods can be used instead. A good example of this would be:

```
class TestServerBasicOps(manager.ScenarioTest):

    """The test suite for server basic operations

    This smoke test case follows this basic set of operations:
     * Create a keypair for use in launching an instance
     * Create a security group to control network access in instance
     * Add simple permissive rules to the security group
     * Launch an instance
     * Perform ssh to instance
     * Verify metadata service
     * Verify metadata on config_drive
     * Terminate the instance
    """
```

Test Identification with Idempotent ID

Every function that provides a test must have an `idempotent_id` decorator that is a unique `uuid-4` instance. This ID is used to complement the fully qualified test name and track test functionality through refactoring. The format of the metadata looks like:

```
@decorators.idempotent_id('585e934c-448e-43c4-acbf-d06a9b899997')
def test_list_servers_with_detail(self):
    # The created server should be in the detailed list of all servers
    ...
```

Tempest.lib includes a `check-uuid` tool that will test for the existence and uniqueness of idempotent_id metadata for every test. If you have Tempest installed you run the tool against Tempest by calling from the Tempest repo:

```
check-uuid
```

It can be invoked against any test suite by passing a package name:

```
check-uuid --package <package_name>
```

Tests without an `idempotent_id` can be automatically fixed by running the command with the `--fix` flag, which will modify the source package by inserting randomly generated uuids for every test that does not have one:

```
check-uuid --fix
```

The `check-uuid` tool is used as part of the Tempest gate job to ensure that all tests have an `idempotent_id` decorator.

Branchless Tempest Considerations

Starting with the OpenStack Icehouse release Tempest no longer has any stable branches. This is to better ensure API consistency between releases because the API behavior should not change between releases. This means that the stable branches are also gated by the Tempest master branch, which also means that proposed commits to Tempest must work against both the master and all the currently supported stable branches of the projects. As such there are a few special considerations that have to be accounted for when pushing new changes to Tempest.

1. New Tests for new features

When adding tests for new features that were not in previous releases of the projects the new test has to be properly skipped with a feature flag. This can be just as simple as using the `@utils.requires_ext()` or `testtools.skipUnless` decorators to check if the required extension (or discoverable optional API) or feature is enabled or can be as difficult as adding a new config option to the appropriate section. If there isnt a method of selecting the new **feature** from the config file then there wont be a mechanism to disable the test with older stable releases and the new test wont be able to merge.

Introduction of a new feature flag requires specifying a default value for the corresponding config option that is appropriate in the latest OpenStack release. Because Tempest is branchless, the feature flags default value will need to be overridden to a value that is appropriate in earlier releases in which the feature isnt available. In DevStack, this can be accomplished by modifying Tempests lib installation script for previous branches (because DevStack is branched).

2. Bug fix on core project needing Tempest changes

When trying to land a bug fix which changes a tested API youll have to use the following procedure:

```
1. Propose change to the project, get a +2 on the change even with failing
2. Propose skip on Tempest which will only be approved after the
  corresponding change in the project has a +2 on change
3. Land project change in master and all open stable branches (if required)
4. Land changed test in Tempest
```

Otherwise the bug fix wont be able to land in the project.

Handily, Zuuls cross-repository dependencies. can be leveraged to do without step 2 and to have steps 3 and 4 happen atomically. To do that, make the patch written in step 1 to depend (refer to Zuuls documentation above) on the patch written in step 4. The commit message for the Tempest change should have a link to the Gerrit review that justifies that change.

3. New Tests for existing features

If a test is being added for a feature that exists in all the current releases of the projects then the only concern is that the API behavior is the same across all the versions of the project being tested. If the behavior is not consistent the test will not be able to merge.

API Stability

For new tests being added to Tempest the assumption is that the API being tested is considered stable and adheres to the OpenStack API stability guidelines. If an API is still considered experimental or in development then it should not be tested by Tempest until it is considered stable.

5.1.2 Tempest Field Guide to Serial tests

What are these tests?

Tempest can run tests serially as well as in parallel, depending on the configuration that is fully up to the user. However, sometimes you need to make sure that tests are not interfering with each other via OpenStack resources with the other tests running in parallel. Tempest creates separate projects for each test class to separate project based resources between test cases.

If your tests use resources outside of projects, e.g. host aggregates then you might need to explicitly separate interfering test cases. If you only need to separate a small set of test cases from each other then you can use the `LockFixture`.

However, in some cases, a small set of tests needs to be run serially. For example, some of the host aggregate and availability zone testing needs compute nodes without any running nova server to be able to move compute hosts between availability zones. But many tempest tests start one or more nova servers.

Why are these tests in Tempest?

This is one of Tempests core purposes, testing the integration between projects.

Scope of these tests

The tests should always use the Tempest implementation of the OpenStack API, as we want to ensure that bugs arent 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 test

While we are looking for interaction of 2 or more services, be specific in your interactions. A giant this is my data center smoke test is hard to debug when it goes wrong.

The tests that need to be run serially need to be marked with the `@serial` class decorator. This will make sure that even if tempest is configured to run the tests in parallel, these tests will always be executed separately from the rest of the test cases.

Please note that due to test ordering optimization reasons test cases marked for `@serial` execution need to be put under `tempest/serial_tests` directory. This will ensure that the serial tests will block the parallel tests in the least amount of time.

5.1.3 Reviewing Tempest Code

To start read the OpenStack Common Review Checklist

Ensuring code is executed

For any new or change to a test it has to be verified in the gate. This means that the first thing to check with any change is that a gate job actually runs it. Tests which arent executed either because of configuration or skips should not be accepted.

If a new test is added that depends on a new config option (like a feature flag), the commit message must reference a change in DevStack or DevStack-Gate that enables the execution of this newly introduced test. This reference could either be a Cross-Repository Dependency or a simple link to a Gerrit review.

Execution time

While checking in the job logs that a new test is actually executed, also pay attention to the execution time of that test. Keep in mind that each test is going to be executed hundreds of time each day, because Tempest tests run in many OpenStack projects. Its worth considering how important/critical the feature under test is with how costly the new test is.

Unit Tests

For any change that adds new functionality to either common functionality or an out-of-band tool unit tests are required. This is to ensure we dont introduce future regressions and to test conditions which we may not hit in the gate runs. API and scenario tests arent required to have unit tests since they should be self-verifying by running them in the gate. All service clients, on the other hand, must have unit tests, as they belong to `tempest/lib`.

API Stability

Tests should only be added for published stable APIs. If a patch contains tests for an API which hasnt been marked as stable or for an API which doesnt conform to the API stability guidelines then it should not be approved.

Reject Copy and Paste Test Code

When creating new tests that are similar to existing tests it is tempting to simply copy the code and make a few modifications. This increases code size and the maintenance burden. Such changes should not be approved if it is easy to abstract the duplicated code into a function or method.

Tests overlap

When a new test is being proposed, question whether this feature is not already tested with Tempest. Tempest has more than 1200 tests, spread amongst many directories, so its easy to introduce test duplication. For example, testing volume attachment to a server could be a compute test or a volume test, depending on how you see it. So one must look carefully in the entire code base for possible overlap. As a rule of thumb, the older a feature is, the more likely its already tested.

Being explicit

When tests are being added that depend on a configurable feature or extension, polling the API to discover that it is enabled should not be done. This will just result in bugs being masked because the test can be skipped automatically. Instead the config file should be used to determine whether a test should be skipped or not. Do not approve changes that depend on an API call to determine whether to skip or not.

Configuration Options

With the introduction of the Tempest external test plugin interface we needed to provide a stable contract for Tempests configuration options. This means we can no longer simply remove a configuration option when its no longer used. Patches proposed that remove options without a deprecation cycle should not be approved. Similarly when changing default values with configuration we need to similarly be careful that we dont break existing functionality. Also, when adding options, just as before, we need to weigh the benefit of adding an additional option against the complexity and maintenance overhead having it costs.

Test Documentation

When a new test is being added refer to the *Test Documentation* section in hacking to see if the requirements are being met. With the exception of a class level docstring linking to the API ref doc in the API tests and a docstring for scenario tests this is up to the reviewers discretion whether a docstring is required or not.

Test Removal and Refactoring

Make sure that any test that is renamed, relocated (e.g. moved to another class), or removed does not belong to the interop testing suite which includes a select suite of Tempest tests for the purposes of validating that OpenStack vendor clouds are interoperable or a projects whitelist or blacklist files.

It is of critical importance that no interop, whitelist or blacklist test reference be broken by a patch set introduced to Tempest that renames, relocates or removes a referenced test.

Please check the existence of code which references Tempest tests with: http://codesearch.openstack.org/

Interop

Make sure that modifications to an interop test are backwards-compatible. This means that code modifications to tests should not undermine the quality of the validation currently performed by the test or significantly alter the behavior of the test.

Removal

Reference the *Tempest Test Removal Procedure* guidelines for understanding best practices associated with test removal.

Release Notes

Release notes are how we indicate to users and other consumers of Tempest what has changed in a given release. Since Tempest 10.0.0 weve been using reno to manage and build the release notes. There are certain types of changes that require release notes and we should not approve them without including a release note. These include but arent limited to, any addition, deprecation or removal from the lib interface, any change to configuration options (including deprecation), CLI additions or deprecations, major feature additions, and anything backwards incompatible or would require a user to take note or do something extra.

Deprecated Code

Sometimes we have some bugs in deprecated code. Basically, we leave it. Because we dont need to maintain it. However, if the bug is critical, we might need to fix it. When it will happen, we will deal with it on a case-by-case basis.

When to approve

- Its OK to hold off on an approval until a subject matter expert reviews it.
- Every patch needs at least single +2s before being approved. A single Tempest core reviewer can approve patches but can always wait for another +2 in any case. Following cases where single +2 can be used without any issue:
  - If any trivial patch set fixes one of the items below:
    * Documentation or code comment typo
    * Documentation ref link
    * Example: example

    Note

    Any other small documentation, CI job, or code change does not fall under this category.

  - If the patch **unblocks** a failing project gate, provided that:
    * the projects PTL +1s the change
    * the patch does not affect any other projects testing gates
    * the patch does not cause any negative side effects
  - If fixing and removing the faulty plugin (which leads to fail voting `tempest-tox-plugin-sanity-check` job) and unblock the tempest gate

5.1.4 Microversion Testing With Tempest

Many OpenStack Services provide their APIs with microversion support and want to test them in Tempest.

This document covers how to test microversions for each project and whether tests should live in Tempest or on project side.

Tempest Scope For Microversion Testing

APIs microversions for any OpenStack service grow rapidly and testing each and every microversion in Tempest is not feasible and efficient way. Also not every API microversion changes the complete system behavior and many of them only change the API or DB layer to accept and return more data on API.

Tempest is an integration test suite, but not all API microversion testing fall under this category. As a result, Tempest mainly covers integration test cases for microversions, Other testing coverage for microversion should be hosted on project side as functional tests or via Tempest plugin as per project guidelines.

Note

Integration tests are those tests which involve more than one service to verify the expected behavior by single or combination of API requests. If a test is just to verify the API behavior as success and failure cases or verify its expected response object, then it does not fall under integration tests.

Tempest will cover only integration testing of applicable microversions with below exceptions:

1. Test covers a feature which is important for interoperability. This covers tests requirement from Defcore.

2. Test needed to fill Schema gaps. Tempest validates API responses with defined JSON schema. API responses can be different on each microversion and the JSON schemas need to be defined separately for the microversion. While implementing new integration tests for a specific microversion, there may be a gap in the JSON schemas (caused by previous microversions) implemented in Tempest. Filling that gap while implementing the new integration test cases is not efficient due to many reasons:

   - Hard to review
   - Sync between multiple integration tests patches which try to fill the same schema gap at same time
   - Might delay the microversion change on project side where project team wants Tempest tests to verify the results.

   Tempest will allow to fill the schema gaps at the end of each cycle, or more often if required. Schema gap can be filled with testing those with a minimal set of tests. Those tests might not be integration tests and might be already covered on project side also. This exception is needed because:

   - Allow to create microversion response schema in Tempest at the same time that projects are implementing their API microversions. This will make implementation easier for adding required tests before a new microversion change can be merged in the corresponding project and hence accelerate the development of microversions.
   - New schema must be verified by at least one test case which exercises such schema.

   For example:
   If any projects implemented 4 API microversion say- v2.3, v2.4, v2.5, v2.6 Assume microversion v2.3, v2.4, v2.6 change the API Response which means Tempest needs to add JSON schema for v2.3, v2.4, v2.6. In that case if only 1 or 2 tests can verify all new schemas then we do not need separate tests for each new schemas. In worst case, we have to add 3 separate tests.

3. Test covers service behavior at large scale with involvement of more deep layer like hypervisor etc not just API/DB layer. This type of tests will be added case by case basis and with project team consultation about why it cannot be covered on project side and worth to test in Tempest.

Project Scope For Microversion Testing

All microversions testing which are not covered under Tempest as per above section, should be tested on project side as functional tests or as Tempest plugin as per project decision.

Configuration options for Microversion

- Add configuration options for specifying test target Microversions. We need to specify test target Microversions because the supported Microversions may be different between OpenStack clouds. For operating multiple Microversion tests in a single Tempest operation, configuration options should represent the range of test target Microversions. New configuration options are:
  - min_microversion
  - max_microversion

  Those should be defined under respective section of each service. For example:

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

How To Implement Microversion Tests

Tempest provides stable interfaces to test API Microversion. For Details, see: API Microversion testing Framework This document explains how to implement Microversion tests using those interfaces.

Step1: Add skip logic based on configured Microversion range

Add logic to skip the tests based on Tests class and configured Microversion range. api_version_utils.check_skip_with_microversion function can be used to automatically skip the tests which do not fall under configured Microversion range. For example:

```
class BaseTestCase1(api_version_utils.BaseMicroversionTest):

    [..]
    @classmethod
    def skip_checks(cls):
        super(BaseTestCase1, cls).skip_checks()
        api_version_utils.check_skip_with_microversion(cls.min_microversion,
                                                       cls.max_microversion,
                                                       CONF.compute.min_
↪microversion,
                                                       CONF.compute.max_
↪microversion)
```

Skip logic can be added in tests base class or any specific test class depends on tests class structure.

Step2: Selected API request microversion

Select appropriate Microversion which needs to be used to send with API request. api_version_utils.select_request_microversion function can be used to select the appropriate Microversion which will be used for API request. For example:

```
@classmethod
def resource_setup(cls):
    super(BaseTestCase1, cls).resource_setup()
    cls.request_microversion = (
        api_version_utils.select_request_microversion(
            cls.min_microversion,
            CONF.compute.min_microversion))
```

Step3: Set Microversion on Service Clients

Microversion selected by Test Class in previous step needs to be set on service clients so that APIs can be requested with selected Microversion.

Microversion can be defined as global variable on service clients which can be set using fixture. Also Microversion header name needs to be defined on service clients which should be constant because it is not supposed to be changed by project as per API contract. For example:

```
COMPUTE_MICROVERSION = None

class BaseClient1(rest_client.RestClient):
    api_microversion_header_name = 'X-OpenStack-Nova-API-Version'
```

Now test class can set the selected Microversion on required service clients using fixture which can take care of resetting the same once tests is completed. For example:

```
def setUp(self):
    super(BaseTestCase1, self).setUp()
    self.useFixture(api_microversion_fixture.APIMicroversionFixture(
        self.request_microversion))
```

Service clients needs to add set Microversion in API request header which can be done by overriding the get_headers() method of rest_client. For example:

```
COMPUTE_MICROVERSION = None

class BaseClient1(rest_client.RestClient):
    api_microversion_header_name = 'X-OpenStack-Nova-API-Version'

    def get_headers(self):
        headers = super(BaseClient1, self).get_headers()
        if COMPUTE_MICROVERSION:
            headers[self.api_microversion_header_name] = COMPUTE_MICROVERSION
        return headers
```

Step4: Separate Test classes for each Microversion

This is last step to implement Microversion test class.

For any Microversion tests, basically we need to implement a separate test class. In addition, each test class defines its Microversion range with class variable like min_microversion and max_microversion. Tests will be valid for that defined range. If that range is out of configured Microversion range then, test will be skipped.

Note

Microversion testing is supported at test class level not at individual test case level.

For example:

Below test is applicable for Microversion from 2.2 till 2.9:

```
class BaseTestCase1(api_version_utils.BaseMicroversionTest,
                    tempest.test.BaseTestCase):

    [..]


class Test1(BaseTestCase1):
```

(continues on next page)

(continued from previous page)

```
    min_microversion = '2.2'
    max_microversion = '2.9'

    [..]
```

Below test is applicable for Microversion from 2.10 till latest:

```
class Test2(BaseTestCase1):
    min_microversion = '2.10'
    max_microversion = 'latest'

    [..]
```

Notes about Compute Microversion Tests

Some of the compute Microversion tests have been already implemented with the Microversion testing framework. So for further tests only step 4 is needed.

Along with that JSON response schema might need versioning if needed.

Compute service clients strictly validate the response against defined JSON schema and does not allow additional elements in response. So if that Microversion changed the API response then schema needs to be versioned. New JSON schema file needs to be defined with new response attributes and service client methods will select the schema based on requested microversion.

If Microversion tests are implemented randomly meaning not in sequence order(v2.20 tests added and previous Microversion tests are not yet added) then, still schema might need to be version for older Microversion if they changed the response. This is because Nova Microversion includes all the previous Microversions behavior.

For Example:

Implementing the v2.20 Microversion tests before v2.9 and 2.19- v2.20 API request will respond as latest behavior of Nova till v2.20, and in v2.9 and 2.19, server response has been changed so response schema needs to be versioned accordingly.

That can be done by using the get_schema method in below module:

The base_compute_client module

class BaseComputeClient(*auth_provider*, *service*, *region*, *endpoint_type='publicURL'*, *build_interval=1*, *build_timeout=60*, *disable_ssl_certificate_validation=False*, *ca_certs=None*, *trace_requests=''*, *name=None*, *http_timeout=None*, *proxy_url=None*, *follow_redirects=True*)

Base compute service clients class to support microversion.

This class adds microversion to API request header if that is set and provides interface to select appropriate JSON schema file for response validation.

Parameters

- **auth_provider** An auth provider object used to wrap requests in auth
- **service** (*str*) The service name to use for the catalog lookup

- **region** (*str*) The region to use for the catalog lookup
- **kwargs** kwargs required by rest_client.RestClient

Microversion tests implemented in Tempest

- Compute
  - 2.1
  - 2.2
  - 2.3
  - 2.6
  - 2.8
  - 2.9
  - 2.10
  - 2.19
  - 2.20
  - 2.21
  - 2.25
  - 2.26
  - 2.28
  - 2.32
  - 2.33
  - 2.36
  - 2.37
  - 2.39
  - 2.41
  - 2.42
  - 2.45
  - 2.47
  - 2.48
  - 2.49
  - 2.50
  - 2.53
  - 2.54
  - 2.55
  - 2.57
  - 2.59

  - 2.60
  - 2.61
  - 2.63
  - 2.64
  - 2.70
  - 2.71
  - 2.73
  - 2.75
  - 2.79
  - 2.86
  - 2.96
- Volume
  - 3.3
  - 3.9
  - 3.11
  - 3.12
  - 3.13
  - 3.14
  - 3.19
  - 3.20
  - 3.55

5.1.5 Tempest Test Removal Procedure

Historically, Tempest was the only way of doing functional testing and integration testing in OpenStack. This was mostly only an artifact of Tempest being the only proven pattern for doing this, not an artifact of a design decision. However, moving forward, as functional testing is being spun up in each individual project, we really only want Tempest to be the integration test suite it was intended to be: testing the high-level interactions between projects through REST API requests. In this model, there are probably existing tests that arent the best fit living in Tempest. However, since Tempest is largely still the only gating test suite in this space we cant carelessly rip out everything from the tree. This document outlines the procedure which was developed to ensure we minimize the risk for removing something of value from the Tempest tree.

This procedure might seem overly conservative and slow-paced, but this is by design to try to ensure we dont remove something that is actually providing value. Having potential duplication between testing is not a big deal especially compared to the alternative of removing something which is actually providing value and is actively catching bugs, or blocking incorrect patches from landing.

Proposing a test removal

3 prong rule for removal

In the proposal etherpad well be looking for answers to 3 questions:

1. The tests proposed for removal must have equiv. coverage in a different projects test suite (whether this is another gating test project, or an in tree functional test suite). For API tests preferably the other project will have a similar source of friction in place to prevent breaking API changes so that we dont regress and let breaking API changes slip through the gate.
2. The test proposed for removal has a failure rate < 0.50% in the gate over the past release (the value and interval will likely be adjusted in the future)
3. There must not be an external user/consumer of Tempest that depends on the test proposed for removal

The answers to 1 and 2 are easy to verify. For 1 just provide a link to the new test location. If you are linking to the Tempest removal patch please also put a Depends-On in the commit message for the commit which moved the test into another repo.

For prong 2 you can use subunit2sql:

Using subunit2sql directly

`SELECT * from tests where test_id like "%test_id%";` (where `$test_id` is the full test_id, but truncated to the class because of `setUpClass` or `tearDownClass` failures)

You can access the infra mysql subunit2sql db w/ read-only permissions with:

- hostname: logstash.openstack.org
- username: query
- password: query
- db_name: subunit2sql

For example if you were trying to remove the test with the id: `tempest.api.compute.admin.test_flavors_negative.FlavorsAdminNegativeTestJSON.test_get_flavor_details_for_deleted_flavor` you would run the following:

1. run the command: `mysql -u query -p -h logstash.openstack.org subunit2sql` to connect to the subunit2sql db
2. run the query:

   ```
   MySQL [subunit2sql]> select * from tests where test_id like \
   "tempest.api.compute.admin.test_flavors_negative.
   ↪FlavorsAdminNegativeTestJSON%";
   ```

   which will return a table of all the tests in the class (but it will also catch failures in `setUpClass` and `tearDownClass`)
3. paste the output table with numbers and the mysql command you ran to generate it into the etherpad.

Eventually, a CLI interface will be created to make that a bit more friendly. Also a dashboard is in the works so we dont need to manually run the command.

The intent of the 2nd prong is to verify that moving the test into a project specific testing is preventing bugs (assuming the Tempest tests were catching issues) from bubbling up a layer into Tempest jobs. If were seeing failure rates above a certain threshold in the gate checks that means the functional testing isnt really being effective in catching that bug (and therefore blocking it from landing) and having the testing run in Tempest still has value.

However, for the 3rd prong verification is a bit more subjective. The original intent of this prong was mostly for interop/refstack and also for things that running on the stable branches. We dont want to remove any tests if that would break our API consistency checking between releases, or something that interop/refstack is depending on being in Tempest. Its worth pointing out that if a test is used in interop_wg as part of interop testing then it will probably have continuing value being in Tempest as part of the integration/integrated tests in general. This is one area where some overlap is expected between testing in projects and Tempest, which is not a bad thing.

Discussing the 3rd prong

There are 2 approaches to addressing the 3rd prong. Either it can be raised during a QA meeting during the Tempest discussion. Please put it on the agenda well ahead of the scheduled meeting. Since the meeting time will be well known ahead of time anyone who depends on the tests will have ample time beforehand to outline any concerns on the before the meeting. To give ample time for people to respond to removal proposals please add things to the agenda by the Monday before the meeting.

The other option is to raise the removal on the openstack-discuss mailing list. (for example see: http://lists.openstack.org/pipermail/openstack-dev/2016-February/086218.html or http://lists.openstack.org/pipermail/openstack-discuss/2019-March/003574.html) This will raise the issue to the wider community and attract at least the same (most likely more) attention than discussing it during the irc meeting. The only downside is that it might take more time to get a response, given the nature of ML.

Exceptions to this procedure

For the most part, all Tempest test removals have to go through this procedure there are a couple of exceptions though:

1. The class of testing has been decided to be outside the scope of Tempest.
2. A revert for a patch which added a broken test, or testing which didnt actually run in the gate (basically any revert for something which shouldnt have been added)
3. Tests that would become out of scope as a consequence of an API change, as described in *API Compatibility*. Such tests cannot live in Tempest because of the branchless nature of Tempest. Such tests must still honor *prong #3*.

For the first exception type, the only types of testing in the tree which have been declared out of scope at this point are:

- The CLI tests (which should be completely removed at this point)
- Neutron Adv. Services testing (which should be completely removed at this point)
- XML API Tests (which should be completely removed at this point)
- EC2 API/boto tests (which should be completely removed at this point)

For tests that fit into this category, the only criteria for removal is that there is equivalent testing elsewhere.

Tempest Scope

Starting in the liberty cycle Tempest, has defined a set of projects which are defined as in scope for direct testing in Tempest. As of today that list is:

- Keystone
- Nova
- Glance
- Cinder
- Neutron
- Swift

Anything that lives in Tempest which doesnt test one of these projects can be removed assuming there is equivalent testing elsewhere. Preferably using the tempest plugin mechanism to maintain continuity after migrating the tests out of Tempest.

API Compatibility

If an API introduces a non-discoverable, backward-incompatible change, and such a change is not backported to all versions supported by Tempest, tests for that API cannot live in Tempest anymore. This is because tests would not be able to know or control which API response to expect, and thus would not be able to enforce a specific behavior.

If a test exists in Tempest that would meet these criteria as a consequence of a change, the test must be removed according to the procedure discussed in this document. The API change should not be merged until all conditions required for test removal can be met.

5.1.6 Tempest Test Writing Guide

This guide serves as a starting point for developers working on writing new Tempest tests. At a high level, tests in Tempest are just tests that conform to the standard python unit test framework. But there are several aspects of that are unique to Tempest and its role as an integration test suite running against a real cloud.

Note

This guide is for writing tests in the Tempest repository. While many parts of this guide are also applicable to Tempest plugins, not all the APIs mentioned are considered stable or recommended for use in plugins. Please refer to *Tempest Test Plugin Interface* for details about writing plugins

Adding a New TestCase

The base unit of testing in Tempest is the TestCase (also called the test class). Each TestCase contains test methods which are the individual tests that will be executed by the test runner. But, the TestCase is the smallest self contained unit for tests from the Tempest perspective. Its also the level at which Tempest is parallel safe. In other words, multiple TestCases can be executed in parallel, but individual test methods in the same TestCase can not. Also, all test methods within a TestCase are assumed to be executed serially. As such you can use the test case to store variables that are shared between methods.

In standard unittest the lifecycle of a TestCase can be described in the following phases:

1. setUpClass
2. setUp
3. Test Execution
4. tearDown
5. doCleanups
6. tearDownClass

setUpClass

The setUpClass phase is the first phase executed by the test runner and is used to perform any setup required for all the test methods to be executed. In Tempest this is a very important step and will automatically do the necessary setup for interacting with the configured cloud.

To accomplish this you do **not** define a setUpClass function, instead there are a number of predefined phases to setUpClass that are used. The phases are:

- skip_checks
- setup_credentials
- setup_clients
- resource_setup

which is executed in that order. Cleanup of resources provisioned during the resource_setup must be scheduled right after provisioning using the addClassResourceCleanup helper. The resource cleanups stacked this way are executed in reverse order during tearDownClass, before the cleanup of test credentials takes place. An example of a TestCase which defines all of these would be:

```
from tempest.common import waiters
from tempest import config
from tempest.lib.common.utils import test_utils
from tempest import test

CONF = config.CONF


class TestExampleCase(test.BaseTestCase):

    @classmethod
    def skip_checks(cls):
        """This section is used to evaluate config early and skip all test
           methods based on these checks
        """
        super(TestExampleCase, cls).skip_checks()
        if not CONF.section.foo
            cls.skip('A helpful message')

    @classmethod
    def setup_credentials(cls):
        """This section is used to do any manual credential allocation and␣
```

(continues on next page)

(continued from previous page)

```
↪also
           in the case of dynamic credentials to override the default network
           resource creation/auto allocation
        """
        # This call is used to tell the credential allocator to not create any
        # network resources for this test case. It also enables selective
        # creation of other neutron resources. NOTE: it must go before the
        # super call
        cls.set_network_resources()
        super(TestExampleCase, cls).setup_credentials()

    @classmethod
    def setup_clients(cls):
        """This section is used to setup client aliases from the manager␣
↪object
           or to initialize any additional clients. Except in a few very
           specific situations you should not need to use this.
        """
        super(TestExampleCase, cls).setup_clients()
        cls.servers_client = cls.os_primary.servers_client

    @classmethod
    def resource_setup(cls):
        """This section is used to create any resources or objects which are
           going to be used and shared by **all** test methods in the
           TestCase. Note then anything created in this section must also be
           destroyed in the corresponding resource_cleanup() method (which␣
↪will
           be run during tearDownClass())
        """
        super(TestExampleCase, cls).resource_setup()
        cls.shared_server = cls.servers_client.create_server(...)
        cls.addClassResourceCleanup(waiters.wait_for_server_termination,
                                    cls.servers_client,
                                    cls.shared_server['id'])
        cls.addClassResourceCleanup(
            test_utils.call_and_ignore_notfound_exc(
                cls.servers_client.delete_server,
                cls.shared_server['id']))
```

Allocating Credentials

Since Tempest tests are all about testing a running cloud, every test will need credentials to be able to make API requests against the cloud. Since this is critical to operation and, when running in parallel, easy to make a mistake, the base TestCase class will automatically allocate a regular user for each TestCase during the setup_credentials() phase. During this process it will also initialize a client manager object using those credentials, which will be your entry point into interacting with the cloud. For more details on how credentials are allocated the *Test Credentials* section of the Tempest Configuration Guide provides more details on the operation of this.

There are some cases when you need more than a single set of credentials, or credentials with a more specialized set of roles. To accomplish this you have to set a class variable `credentials` on the TestCase directly. For example:

```
from tempest import test

class TestExampleAdmin(test.BaseTestCase):

    credentials = ['primary', 'admin']

    @classmethod
    def skip_checks(cls):
        ...
```

In this example the `TestExampleAdmin` TestCase will allocate 2 sets of credentials, one regular user and one admin user. The corresponding manager objects will be set as class variables `cls.os_primary` and `cls.os_admin` respectively. You can also allocate a second user by putting **alt** in the list too. A set of alt credentials are the same as primary but can be used for tests cases that need a second user/project.

You can also specify credentials with specific roles assigned. This is useful for cases where there are specific RBAC requirements hard coded into an API. The canonical example of this are swift tests which often want to test swifts concepts of operator and reseller_admin. An actual example from Tempest on how to do this is:

```
class PublicObjectTest(base.BaseObjectTest):

    credentials = [['operator', CONF.object_storage.operator_role],
                   ['operator_alt', CONF.object_storage.operator_role]]

    @classmethod
    def setup_credentials(cls):
        super(PublicObjectTest, cls).setup_credentials()
        ...
```

In this case the manager objects will be set to `cls.os_roles_operator` and `cls.os_roles_operator_alt` respectively.

There is no limit to how many credentials you can allocate in this manner, however in almost every case you should **not** need more than 3 sets of credentials per test case.

To figure out the mapping of manager objects set on the TestCase and the requested credentials you can reference:

Credentials Entry	Manager Variable
primary	cls.os_primary
admin	cls.os_admin
alt	cls.os_alt
[$label, $role]	cls.os_roles_$label

By default cls.os_primary is available since it is allocated in the base Tempest test class (located in tempest/test.py). If your TestCase inherits from a different direct parent class (itll still inherit from the BaseTestCase, just not directly) be sure to check if that class overrides allocated credentials.

Dealing with Network Allocation

When Neutron is enabled and a testing requires networking this isnt normally automatically setup when a tenant is created. Since Tempest needs isolated tenants to function properly it also needs to handle network allocation. By default the base test class will allocate a network, subnet, and router automatically (this depends on the configured credential provider, for more details see: *Network Creation/Usage for Servers*). However, there are situations where you do no need all of these resources allocated (or your TestCase inherits from a class that overrides the default in tempest/test.py). There is a class level mechanism to override this allocation and specify which resources you need. To do this you need to call *cls.set_network_resources()* in the *setup_credentials()* method before the *super()*. For example:

```
from tempest import test


class TestExampleCase(test.BaseTestCase):

    @classmethod
    def setup_credentials(cls):
        cls.set_network_resources(network=True, subnet=True, router=False)
        super(TestExampleCase, cls).setup_credentials()
```

There are 2 quirks with the usage here. First for the set_network_resources function to work properly it **must be called before super()**. This is so that children classes settings are always used instead of a parent classes. The other quirk here is that if you do not want to allocate any network resources for your test class simply call *set_network_resources()* without any arguments. For example:

```
from tempest import test


class TestExampleCase(test.BaseTestCase):

    @classmethod
    def setup_credentials(cls):
        cls.set_network_resources()
        super(TestExampleCase, cls).setup_credentials()
```

This will not allocate any networking resources. This is because by default all the arguments default to False.

Its also worth pointing out that it is common for base test classes for different services (and scenario tests) to override this setting. When inheriting from classes other than the base TestCase in tempest/test.py it is worth checking the immediate parent for what is set to determine if your class needs to override that setting.

Running some tests in serial

Tempest potentially runs test cases in parallel, depending on the configuration. However, sometimes you need to make sure that tests are not interfering with each other via OpenStack resources. Tempest creates separate projects for each test class to separate project based resources between test cases.

If your tests use resources outside of projects, e.g. host aggregates then you might need to explicitly separate interfering test cases. If you only need to separate a small set of testcases from each other then you can use the `LockFixture`.

However, in some cases a small set of tests needs to be run independently from the rest of the test cases. For example, some of the host aggregate and availability zone testing needs compute nodes without any running nova server to be able to move compute hosts between availability zones. But many tempest tests start one or more nova servers. In this scenario you can mark the small set of tests that needs to be independent from the rest with the `@serial` class decorator. This will make sure that even if tempest is configured to run the tests in parallel the tests in the marked test class will always be executed separately from the rest of the test cases.

Please note that due to test ordering optimization reasons test cases marked for `@serial` execution need to be put under `tempest/serial_tests` directory. This will ensure that the serial tests will block the parallel tests in the least amount of time.

Interacting with Credentials and Clients

Once you have your basic TestCase setup youll want to start writing tests. To do that you need to interact with an OpenStack deployment. This section will cover how credentials and clients are used inside of Tempest tests.

Manager Objects

The primary interface with which you interact with both credentials and API clients is the client manager object. These objects are created automatically by the base test class as part of credential setup (for more details see the previous *Allocating Credentials* section). Each manager object is initialized with a set of credentials and has each client object already setup to use that set of credentials for making all the API requests. Each client is accessible as a top level attribute on the manager object. So to start making API requests you just access the clients method for making that call and the credentials are already setup for you. For example if you wanted to make an API call to create a server in Nova:

```
from tempest import test


class TestExampleCase(test.BaseTestCase):
    def test_example_create_server(self):
        self.os_primary.servers_client.create_server(...)
```

is all you need to do. As described previously, in the above example the `self.os_primary` is created automatically because the base test class sets the `credentials` attribute to allocate a primary credential set and initializes the client manager as `self.os_primary`. This same access pattern can be used for all of the clients in Tempest.

Credentials Objects

In certain cases you need direct access to the credentials (the most common use case would be an API request that takes a user or project id in the request body). If youre in a situation where you need to access this youll need to access the `credentials` object which is allocated from the configured credential provider in the base test class. This is accessible from the manager object via the managers `credentials` attribute. For example:

```
from tempest import test


class TestExampleCase(test.BaseTestCase):
```

(continues on next page)

(continued from previous page)

```
    def test_example_create_server(self):
        credentials = self.os_primary.credentials
```

The credentials object provides access to all of the credential information you would need to make API requests. For example, building off the previous example:

```
from tempest import test


class TestExampleCase(test.BaseTestCase):
    def test_example_create_server(self):
        credentials = self.os_primary.credentials
        username = credentials.username
        user_id = credentials.user_id
        password = credentials.password
        tenant_id = credentials.tenant_id
```

5.1.7 Requirements Upper Constraint for Tempest

Tempest is branchless and supported stable branches use Tempest master and all EM stable branches use old compatible Tempest version for their testing. This means the OpenStack installed upper-constraints might not be compatible with Tempest used for stable branch testing. For example, if Tempest master is used for testing the stable/stein then stable/stein constraint might not be compatible with Tempest master so we need to use master upper-constraints there. That is why we use virtual env for Tempest installation and running tests so that we can control Tempest required constraint from system wide installed constraints.

Devstack takes care of using the master upper-constraints when Tempest master is used. But when old Tempest is used then devstack alone cannot handle the compatible constraints because Tempest in-tree tox.ini also set the upper-constraints which are master constraints so if devstack set the different constraints than what we have in tox.ini we end up re-creation of venv which flush all previously installed tempest plugins in that venv. More details are on this ML thread

To solve that problem we have two ways:

1. Set UPPER_CONSTRAINTS_FILE to compatible constraint path This option is not easy as it requires to set this env var everywhere Tempest tox env is used like in devstack, grenade, projects side, zuulv3 roles etc.
2. Pin upper-constraints in tox.ini If we can pin the upper-constraints in tox.ini on every release with the branch constraint at the time of release then we can solve it in an easy way because tox can use the compatible constraint at the time of venv creation itself. But this can again mismatch with the devstack set constraint so we need to follow the below process to make it work.

How to pin upper-constraints in tox.ini

This has to be done exactly before we cut the Tempest new major version bump release for the cycle.

Step1: Add the pin constraint proposal in QA office hour.
Pin constraint proposal includes:

- pin constraint patch. Example patch 720578
- revert of pin constraint patch. Example patch 721724

Step2: Approve pin constraint and its revert patch together.
During office hour we need to check that there are no open patches for Tempest release and accordingly we fast approve the pin constraint and its revert patch during office hour itself. Remember pin constraint patch has to be the last commit to include in Tempest release.

Step3: Use pin constraint patch hash for the Tempest new release.
By using the pin constraint patch hash we make sure tox.ini in Tempest released tag has the compatible stable constraint not the master one. For Example Tempest 24.0.0

5.2 Plugins

5.2.1 Tempest Plugins Guide

Tempest Test Plugin Interface

Tempest has an external test plugin interface which enables anyone to integrate an external test suite as part of a Tempest run. This will let any project leverage being run with the rest of the Tempest suite while not requiring the tests live in the Tempest tree.

Creating a plugin

Creating a plugin is fairly straightforward and doesnt require much additional effort on top of creating a test suite using tempest.lib. One thing to note with doing this is that the interfaces exposed by Tempest are not considered stable (with the exception of configuration variables whichever effort goes into ensuring backward compatibility). You should not need to import anything from Tempest itself except where explicitly noted.

Stable Tempest APIs plugins may use

As noted above, several Tempest APIs are acceptable to use from plugins, while others are not. A list of stable APIs available to plugins is provided below:

- tempest.lib.*
- tempest.config
- tempest.test_discover.plugins
- tempest.common.credentials_factory
- tempest.clients
- tempest.test
- tempest.scenario.manager

If there is an interface from Tempest that you need to rely on in your plugin which is not listed above, it likely needs to be migrated to tempest.lib. In that situation, file a bug, push a migration patch, etc. to expedite providing the interface in a reliable manner.

Plugin Cookiecutter

In order to create the basic structure with base classes and test directories you can use the tempest-plugin-cookiecutter project:

```
> pip install -U cookiecutter && cookiecutter https://opendev.org/openstack/
↪tempest-plugin-cookiecutter.git

Cloning into 'tempest-plugin-cookiecutter'...
remote: Counting objects: 17, done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 17 (delta 1), reused 14 (delta 1)
Unpacking objects: 100% (17/17), done.
Checking connectivity... done.
project (default is "sample")? foo
testclass (default is "SampleTempestPlugin")? FooTempestPlugin
```

This would create a folder called `foo_tempest_plugin/` with all necessary basic classes. You only need to move/create your test in `foo_tempest_plugin/tests`.

Entry Point

Once youve created your plugin class you need to add an entry point to your project to enable Tempest to find the plugin. The entry point must be added to the tempest.test_plugins namespace.

If you are using pbr this is fairly straightforward, in the setup.cfg just add something like the following:

```
[entry_points]
tempest.test_plugins =
    plugin_name = module.path:PluginClass
```

Standalone Plugin vs In-repo Plugin

Since all thats required for a plugin to be detected by Tempest is a valid setuptools entry point in the proper namespace there is no difference from the Tempest perspective on either creating a separate Python package to house the plugin or adding the code to an existing Python project. However, there are tradeoffs to consider when deciding which approach to take when creating a new plugin.

If you create a separate Python project for your plugin this makes a lot of things much easier. Firstly it makes packaging and versioning much simpler, you can easily decouple the requirements for the plugin from the requirements for the other project. It lets you version the plugin independently and maintain a single version of the test code across project release boundaries (see the Branchless Tempest Spec for more details on this). It also greatly simplifies the install time story for external users. Instead of having to install the right version of a project in the same Python namespace as Tempest they simply need to pip install the plugin in that namespace. It also means that users dont have to worry about inadvertently installing a Tempest plugin when they install another package.

The sole advantage of integrating a plugin into an existing Python project is that it enables you to land code changes at the same time you land test changes in the plugin. This reduces some of the burden on contributors by not having to land 2 changes to add a new API feature and then test it, and do it as a single combined commit instead.

Plugin Class

To provide Tempest with all the required information it needs to be able to run your plugin you need to create a plugin class which Tempest will load and call to get information when it needs. To simplify

creating this Tempest provides an abstract class that should be used as the parent for your plugin. To use this you would do something like the following:

```
from tempest.test_discover import plugins

class MyPlugin(plugins.TempestPlugin):
```

Then you need to ensure you locally define all of the mandatory methods in the abstract class, you can refer to the API doc below for a reference of what that entails.

Abstract Plugin Class

class TempestPlugin

Provide basic hooks for an external plugin

To provide tempest the necessary information to run the plugin.

Plugin Structure

While there are no hard and fast rules for the structure of a plugin, there are basically no constraints on what the plugin looks like as long as the 2 steps above are done. However, there are some recommended patterns to follow to make it easy for people to contribute and work with your plugin. For example, if you create a directory structure with something like:

```
plugin_dir/
  config.py
  plugin.py
  tests/
    api/
    scenario/
  services/
    client.py
```

That will mirror what people expect from Tempest. The file

- **config.py**: contains any plugin specific configuration variables
- **plugin.py**: contains the plugin class used for the entry point
- **tests: the directory where test discovery will be run, all tests should** be under this dir
- **services**: where the plugin specific service clients are

Additionally, when youre creating the plugin you likely want to follow all of the Tempest developer and reviewer documentation to ensure that the tests being added in the plugin act and behave like the rest of Tempest.

Dealing with configuration options

Historically, Tempest didnt provide external guarantees on its configuration options. However, with the introduction of the plugin interface, this is no longer the case. An external plugin can rely on using any configuration option coming from Tempest, there will be at least a full deprecation cycle for any option before its removed. However, just the options provided by Tempest may not be sufficient for the plugin. If you need to add any plugin specific configuration options you should use the `register_opts` and

`get_opt_lists` methods to pass them to Tempest when the plugin is loaded. When adding configuration options the `register_opts` method gets passed the CONF object from Tempest. This enables the plugin to add options to both existing sections and also create new configuration sections for new options.

Service Clients

If a plugin defines a service client, it is beneficial for it to implement the `get_service_clients` method in the plugin class. All service clients which are exposed via this interface will be automatically configured and be available in any instance of the service clients class, defined in `tempest.lib.services.clients.ServiceClients`. In case multiple plugins are installed, all service clients from all plugins will be registered, making it easy to write tests which rely on multiple APIs whose service clients are in different plugins.

Example implementation of `get_service_clients`:

```
def get_service_clients(self):
    # Example implementation with two service clients
    my_service1_config = config.service_client_config('my_service')
    params_my_service1 = {
        'name': 'my_service_v1',
        'service_version': 'my_service.v1',
        'module_path': 'plugin_tempest_tests.services.my_service.v1',
        'client_names': ['API1Client', 'API2Client'],
    }
    params_my_service1.update(my_service_config)
    my_service2_config = config.service_client_config('my_service')
    params_my_service2 = {
        'name': 'my_service_v2',
        'service_version': 'my_service.v2',
        'module_path': 'plugin_tempest_tests.services.my_service.v2',
        'client_names': ['API1Client', 'API2Client'],
    }
    params_my_service2.update(my_service2_config)
    return [params_my_service1, params_my_service2]
```

Parameters:

- **name**: Name of the attribute used to access the `ClientsFactory` from the `ServiceClients` instance. See the example below.
- **service_version**: Tempest enforces a single implementation for each service client. Available service clients are held in a `ClientsRegistry` singleton, and registered with `service_version`, which means that `service_version` must be unique and it should represent the service API and version implemented by the service client.
- **module_path**: Relative to the service client module, from the root of the plugin.
- **client_names**: Name of the classes that implement service clients in the service clients module.

Example usage of the service clients in tests:

```
# my_creds is an instance of tempest.lib.auth.Credentials
# identity_uri is v2 or v3 depending on the configuration
from tempest.lib.services import clients
```

(continues on next page)

(continued from previous page)

```
my_clients = clients.ServiceClients(my_creds, identity_uri)
my_service1_api1_client = my_clients.my_service_v1.API1Client()
my_service2_api1_client = my_clients.my_service_v2.API1Client(my_args='any')
```

Automatic configuration and registration of service clients imposes some extra constraints on the structure of the configuration options exposed by the plugin.

Firstly, `service_version` should be in the format *service_config[.version]*. The *.version* part is optional, and should only be used if there are multiple versions of the same API available. The *service_config* must match the name of a configuration options group defined by the plugin. Different versions of one API must share the same configuration group.

Secondly, the configuration options group *service_config* must contain the following options:

- *catalog_type*: corresponds to *service* in the catalog
- *endpoint_type*

The following options will be honoured if defined, but they are not mandatory, as they do not necessarily apply to all service clients.

- *region*: default to identity.region
- *build_timeout*: default to compute.build_timeout
- *build_interval*: default to compute.build_interval

Thirdly, the service client classes should inherit from `RestClient`, should accept generic keyword arguments, and should pass those arguments to the `__init__` method of `RestClient`. Extra arguments can be added. For instance:

```
class MyAPIClient(rest_client.RestClient):

    def __init__(self, auth_provider, service, region,
                 my_arg, my_arg2=True, **kwargs):
        super(MyAPIClient, self).__init__(
            auth_provider, service, region, **kwargs)
        self.my_arg = my_arg
        self.my_args2 = my_arg
```

Finally, the service client should be structured in a Python module, so that all service client classes are importable from it. Each major API version should have its own module.

The following folder and module structure is recommended for a single major API version:

```
plugin_dir/
  services/
    __init__.py
    client_api_1.py
    client_api_2.py
```

The content of __init__.py module should be:

```
from client_api_1.py import API1Client
from client_api_2.py import API2Client

__all__ = ['API1Client', 'API2Client']
```

The following folder and module structure is recommended for multiple major API versions:

```
plugin_dir/
  services/
    v1/
       __init__.py
       client_api_1.py
       client_api_2.py
    v2/
       __init__.py
       client_api_1.py
       client_api_2.py
```

The content each of __init__.py module under vN should be:

```
from client_api_1.py import API1Client
from client_api_2.py import API2Client

__all__ = ['API1Client', 'API2Client']
```

Using Plugins

Tempest will automatically discover any installed plugins when it is run. So by just installing the Python packages, which contain your plugin, youll be using them with Tempest, nothing else is really required.

However, you should take care when installing plugins. By their very nature, there are no guarantees when running Tempest with plugins enabled about the quality of the plugin. Additionally, while there is no limitation on running with multiple plugins, its worth noting that poorly written plugins might not properly isolate their tests which could cause unexpected cross-interactions between plugins.

Notes for using plugins with virtualenvs

When using a Tempest inside a virtualenv (like when running under tox) you have to ensure that the package that contains your plugin is either installed in the venv too or that you have system site-packages enabled. The virtualenv will isolate the Tempest install from the rest of your system so just installing the plugin package on your system and then running Tempest inside a venv will not work.

For example, you can use tox to install and run tests from a tempest plugin like this:

```
[~/tempest] $ tox -e venv-tempest -- pip install (path to the plugin␣
↪directory)
[~/tempest] $ tox -e all
```

Stable Branch Support Policy

Stable Branch Support Policy

Since the Extended Maintenance policy for stable branches was adopted OpenStack projects will keep stable branches around after a stable or maintained period for a phase of indeterminate length called Extended Maintenance. Prior to this resolution Tempest supported all stable branches which were supported upstream. This policy does not scale under the new model as Tempest would be responsible for gating proposed changes against an ever increasing number of branches. Therefore due to resource constraints, Tempest will only provide support for branches in the Maintained phase from the documented Support Phases. When a branch moves from the *Maintained* to the *Extended Maintenance* phase, Tempest will tag the removal of support for that branch as it has in the past when a branch goes end of life.

The expectation for *Extended Maintenance* phase branches is that they will continue running Tempest during that phase of support. Since the REST APIs are stable interfaces across release boundaries, branches in these phases should run Tempest from master as long as possible. But, because we wont be actively testing branches in these phases, its possible that well introduce changes to Tempest on master which will break support on *Extended Maintenance* phase branches. When this happens the expectation for those branches is to either switch to running Tempest from a tag with support for the branch, or exclude a newly introduced test (if that is the cause of the issue). Tempest will not be creating stable branches to support *Extended Maintenance* phase branches, as the burden is on the *Extended Maintenance* phase branch maintainers, not the Tempest project, to support that branch.

Stable Branch Testing Policy

Stable Branch Testing Policy

Tempest and its plugins need to support the stable branches as per *Stable Branch Support Policy*.

Because of branchless model of Tempest and plugins, all the supported stable branches use the Tempest and plugins master version for their testing. That is done in devstack by using the master branch for the Tempest installation. To make sure the master version of Tempest or plugins (for any changes or adding new tests) is compatible for all the supported stable branches testing, Tempest and its plugins need to add the stable branches job on the master gate. That way can test the stable branches against master code and can avoid breaking supported branches accidentally.

Example:

- Stable jobs on Tempest master.
- Stable job on neutron tempest plugins

Once any stable branch is moved to the Extended Maintenance Phases and devstack start using the Tempest older version for that stable branch testing then we can remove that stable branch job from master gate.

Example: https://review.opendev.org/#/c/722183/

Tempest & Plugins Compatible Version Policy

Tempest and Plugins compatible version policy

Tempest and its plugins are responsible for the integrated testing of OpenStack. These tools have two use cases:

1. Testing upstream code at gate
2. Testing Production Cloud

Upstream code is tested by the master version of branchless Tempest & plugins for all supported stable branches in Maintained phase.

Production Cloud can be tested by using the compatible version or using master version. It depends on the testing strategy of cloud. To provide the compatible version of Tempest and its Plugins per OpenStack release, we started the coordinated release of all plugins and Tempest per OpenStack release. These versions are the first set of versions from Tempest and its Plugins to officially start the support of a particular OpenStack release. For example: OpenStack Train release first compatible versions Tempest plugins version.

Because of branchless nature of Tempest and its plugins, first version released during OpenStack release is not the last version to support that OpenStack release. This means the next (or master) versions can also be used for upstream testing as well as in production testing.

Since the Extended Maintenance policy for stable branch, Tempest started releasing the `end of support` version once stable release is moved to EM state, which used to happen on EOL of stable release. This is the last compatible version of Tempest for the OpenStack release moved to EM.

Because of branchless nature as explained above, we have a range of versions which can be considered a compatible version for particular OpenStack release. How we should release those versions is mentioned in the below table.

First compatible version ->	OpenStack XYZ	<- Last compatible version
This is the latest version released when OpenStack XYZ is released. Example: Tempest plugins version		This is the version released when OpenStack XYZ is moved to EM state. Hash used for this should be the hash from master at the time of branch is EM not the one used for First compatible version

Tempest & the Plugins should follow the above mentioned policy for the `First compatible version` and the `Last compatible version.` so that we provide the right set of compatible versions to Upstream as well as to Production Cloud testing.

Plugins Registry

Tempest Plugin Registry

Since weve created the external plugin mechanism, its gotten used by a lot of projects. The following is a list of plugins that currently exist.

Detected Plugins

The following are plugins that a script has found in the openstack/ namespace, which includes but is not limited to official OpenStack projects.

SR	Plugin Name	URL
1	airship/tempest-plugin	https://opendev.org/airship/tempest-plugin
2	openstack/barbican-tempest-plugin	https://opendev.org/openstack/barbican-tempest-plugin
3	openstack/blazar-tempest-plugin	https://opendev.org/openstack/blazar-tempest-plugin

continues on next page

Table 1 – continued from previous page

SR	Plugin Name	URL
4	openstack/cinder-tempest-plugin	https://opendev.org/openstack/cinder-tempest-plugin
5	openstack/cloudkitty-tempest-plugin	https://opendev.org/openstack/cloudkitty-tempest-plugin
6	openstack/cyborg-tempest-plugin	https://opendev.org/openstack/cyborg-tempest-plugin
7	openstack/designate-tempest-plugin	https://opendev.org/openstack/designate-tempest-plugin
8	openstack/freezer-tempest-plugin	https://opendev.org/openstack/freezer-tempest-plugin
9	openstack/glance-tempest-plugin	https://opendev.org/openstack/glance-tempest-plugin
10	openstack/heat-tempest-plugin	https://opendev.org/openstack/heat-tempest-plugin
11	openstack/ironic-tempest-plugin	https://opendev.org/openstack/ironic-tempest-plugin
12	openstack/keystone-tempest-plugin	https://opendev.org/openstack/keystone-tempest-plugin
13	openstack/magnum-tempest-plugin	https://opendev.org/openstack/magnum-tempest-plugin
14	openstack/manila-tempest-plugin	https://opendev.org/openstack/manila-tempest-plugin
15	openstack/mistral-tempest-plugin	https://opendev.org/openstack/mistral-tempest-plugin
16	openstack/monasca-tempest-plugin	https://opendev.org/openstack/monasca-tempest-plugin
17	openstack/networking-generic-switch	https://opendev.org/openstack/networking-generic-switch
18	openstack/neutron-tempest-plugin	https://opendev.org/openstack/neutron-tempest-plugin
19	openstack/octavia-tempest-plugin	https://opendev.org/openstack/octavia-tempest-plugin
20	openstack/telemetry-tempest-plugin	https://opendev.org/openstack/telemetry-tempest-plugin
21	openstack/trove-tempest-plugin	https://opendev.org/openstack/trove-tempest-plugin
22	openstack/venus-tempest-plugin	https://opendev.org/openstack/venus-tempest-plugin
23	openstack/vitrage-tempest-plugin	https://opendev.org/openstack/vitrage-tempest-plugin
24	openstack/watcher-tempest-plugin	https://opendev.org/openstack/watcher-tempest-plugin
25	openstack/whitebox-tempest-plugin	https://opendev.org/openstack/whitebox-tempest-plugin
26	openstack/zaqar-tempest-plugin	https://opendev.org/openstack/zaqar-tempest-plugin
27	openstack/zun-tempest-plugin	https://opendev.org/openstack/zun-tempest-plugin
28	x/almanach	https://opendev.org/x/almanach
29	x/gabbi-tempest	https://opendev.org/x/gabbi-tempest
30	x/gce-api	https://opendev.org/x/gce-api
31	x/glare	https://opendev.org/x/glare
32	x/group-based-policy	https://opendev.org/x/group-based-policy
33	x/intel-nfv-ci-tests	https://opendev.org/x/intel-nfv-ci-tests
34	x/kingbird	https://opendev.org/x/kingbird
35	x/mogan	https://opendev.org/x/mogan
36	x/networking-cisco	https://opendev.org/x/networking-cisco
37	x/networking-fortinet	https://opendev.org/x/networking-fortinet
38	x/networking-l2gw-tempest-plugin	https://opendev.org/x/networking-l2gw-tempest-plugin
39	x/networking-plumgrid	https://opendev.org/x/networking-plumgrid
40	x/networking-spp	https://opendev.org/x/networking-spp
41	x/novajoin-tempest-plugin	https://opendev.org/x/novajoin-tempest-plugin
42	x/ranger-tempest-plugin	https://opendev.org/x/ranger-tempest-plugin
43	x/trio2o	https://opendev.org/x/trio2o
44	x/valet	https://opendev.org/x/valet
45	x/vmware-nsx-tempest-plugin	https://opendev.org/x/vmware-nsx-tempest-plugin
46	x/whitebox-neutron-tempest-plugin	https://opendev.org/x/whitebox-neutron-tempest-plugin

Non Active Plugins

List of Tempest plugin projects that are stale or unmaintained for a long time (6 months or more). They can be moved out of nonactivelist state once one of the relevant patches gets merged: https://review.opendev.org/#/q/topic:tempest-sanity-gate+%28status:open%29

SR	Plugin Name	URL
1	x/gce-api	https://opendev.org/x/gce-api
2	x/glare	https://opendev.org/x/glare
3	x/group-based-policy	https://opendev.org/x/group-based-policy
4	x/intel-nfv-ci-tests	https://opendev.org/x/intel-nfv-ci-tests
5	openstack/networking-generic-switch	https://opendev.org/openstack/networking-generic-switch
6	x/networking-plumgrid	https://opendev.org/x/networking-plumgrid
7	x/networking-spp	https://opendev.org/x/networking-spp
8	openstack/neutron-dynamic-routing	https://opendev.org/openstack/neutron-dynamic-routing
9	openstack/neutron-vpnaas	https://opendev.org/openstack/neutron-vpnaas
10	x/valet	https://opendev.org/x/valet
11	x/kingbird	https://opendev.org/x/kingbird
12	x/mogan	https://opendev.org/x/mogan
13	x/vmware-nsx-tempest-pluginx/networking-l2gw-tempest-pluginx/novajoin-tempest-pluginx/ranger-tempest-pluginx/trio2ox/networking-fortinet	https://opendev.org/x/vmware-nsx-tempest-pluginx/networking-l2gw-tempest-pluginx/novajoin-tempest-pluginx/ranger-tempest-pluginx/trio2ox/networking-fortinet

5.3 Tempest & Plugins Compatible Version Policy

5.4 Keystone Scopes & Roles Support in Tempest

5.4.1 Keystone Scopes & Roles Support in Tempest

OpenStack Keystone supports different scopes in token, refer to the Keystone doc. Along with the scopes, keystone supports default roles, one of which is a reader role, for details refer to this keystone document.

Tempest supports those scopes and roles credentials that can be used to test APIs under different scope and roles.

Dynamic Credentials

Dynamic credential supports all the below set of personas and allows you to generate credentials tailored to a specific persona that you can use in your test.

Domain scoped personas:

1. Domain Admin: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['domain_admin']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_d_admin_client = (
            cls.os_domain_admin.availability_zone_client)
```

2. Domain Member: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['domain_member']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_d_member_client = (
            cls.os_domain_member.availability_zone_client)
```

3. Domain Reader: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['domain_reader']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_d_reader_client = (
            cls.os_domain_reader.availability_zone_client)
```

4. Domain other roles: This is supported and can be requested and used from the test as below:

   You need to use the `domain` as the prefix in credentials type, and based on that, Tempest will create test users under domain scope.

```
class TestDummy(base.DummyBaseTest):

    credentials = [['domain_my_role1', 'my_own_role1', 'admin']
                   ['domain_my_role2', 'my_own_role2']]

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_d_role1_client = (
            cls.os_domain_my_role1.availability_zone_client)
        cls.az_d_role2_client = (
            cls.os_domain_my_role2.availability_zone_client)
```

System scoped personas:

1. System Admin: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['system_admin']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_s_admin_client = (
            cls.os_system_admin.availability_zone_client)
```

2. System Member: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['system_member']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_s_member_client = (
            cls.os_system_member.availability_zone_client)
```

3. System Reader: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['system_reader']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_s_reader_client = (
            cls.os_system_reader.availability_zone_client)
```

4. System other roles: This is supported and can be requested and used from the test as below:

   You need to use the `system` as the prefix in credentials type, and based on that, Tempest will create test users under project scope.

```
class TestDummy(base.DummyBaseTest):

    credentials = [['system_my_role1', 'my_own_role1', 'admin']
                   ['system_my_role2', 'my_own_role2']]

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_s_role1_client = (
```

(continues on next page)

(continued from previous page)

```
            cls.os_system_my_role1.availability_zone_client)
        cls.az_s_role2_client = (
            cls.os_system_my_role2.availability_zone_client)
```

Project scoped personas:

1. Project Admin: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['project_admin']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_p_admin_client = (
            cls.os_project_admin.availability_zone_client)
```

2. Project Member: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['project_member']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_p_member_client = (
            cls.os_project_member.availability_zone_client)
```

3. Project Reader: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['project_reader']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_p_reader_client = (
            cls.os_project_reader.availability_zone_client)
```

> **Note**
>
> primary, project_admin, project_member, and project_reader credentials will be created under same project.

4. Project alternate Admin: This is supported and can be requested and used from the test

as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['project_alt_admin']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_p_alt_admin_client = (
            cls.os_project_alt_admin.availability_zone_client)
```

5. Project alternate Member: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['project_alt_member']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_p_alt_member_client = (
            cls.os_project_alt_member.availability_zone_client)
```

6. Project alternate Reader: This is supported and can be requested and used from the test as below:

```
class TestDummy(base.DummyBaseTest):

    credentials = ['project_alt_reader']

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_p_alt_reader_client = (
            cls.os_project_alt_reader.availability_zone_client)
```

> **Note**
>
> alt, project_alt_admin, project_alt_member, and project_alt_reader credentials will be created under same project.

7. Project other roles: This is supported and can be requested and used from the test as below:

   You need to use the `project` as the prefix in credentials type, and based on that, Tempest will create test users under project scope.

```
class TestDummy(base.DummyBaseTest):
```

(continues on next page)

(continued from previous page)

```
    credentials = [['project_my_role1', 'my_own_role1', 'admin']
                   ['project_my_role2', 'my_own_role2']]

    @classmethod
    def setup_clients(cls):
        super(TestDummy, cls).setup_clients()
        cls.az_role1_client = (
            cls.os_project_my_role1.availability_zone_client)
        cls.az_role2_client = (
            cls.os_project_my_role2.availability_zone_client)
```

Note

admin credenatials is considered and kept as legacy admin and will be created under new project. If any test want to test with admin role in projectA and non-admin/admin in projectB then test can request projectA admin using admin or project_alt_admin and non-admin in projectB using primary, project_member, or project_reader/admin in projectB using project_admin. Many existing tests using the admin with new project to assert on the resource list so we are keeping admin a kind of legacy admin.

Pre-Provisioned Credentials

Pre-Provisioned credentials support the below set of personas and can be used in the test as shown above in the `Dynamic Credentials` Section.

- Domain Admin
- Domain Member
- Domain Reader
- System Admin
- System Member
- System Reader
- Project Admin
- Project Member
- Project Reader

5.5 Stable Branch Support Policy

5.6 Stable Branch Testing Policy

5.7 Library

5.7.1 Tempest Library Documentation

Tempest 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 Tempest repo 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 doesnt have any dependency on other Tempest internals outside of lib (including no usage of Tempest configuration).

Stability

Any code that lives in tempest/lib will be treated as a stable interface. This means that any public interface under the tempest/lib directory is expected to be a stable interface suitable for public consumption. However, for any interfaces outside of tempest/lib in the tempest tree (unless otherwise noted) or any private interfaces the same stability guarantees dont apply.

Adding Interfaces

When adding an interface to tempest/lib we have to make sure there are no dependencies on any pieces of tempest outside of tempest/lib. This means if for example there is a dependency on the configuration file we need remove that. The other aspect when adding an interface is to make sure its really an interface ready for external consumption and something we want to commit to supporting.

Making changes

When making changes to tempest/lib you have to be conscious of the effect of any changes on external consumers. If your proposed change set will change the default behaviour of any interface, or make something which previously worked not after your change, then it is not acceptable. Every effort needs to go into preserving backwards compatibility in changes.

Reviewing

When reviewing a proposed change to tempest/lib code we need to be careful to ensure that we dont break backward compatibility. For patches that change existing interfaces, we have to be careful to make sure we dont break any external consumers. Some common red flags are:

- a change to an existing API requires a change outside the library directory where the interface is being consumed
- a unit test has to be significantly changed to make the proposed change pass

Testing

When adding a new interface to the library we need to at a minimum have unit test coverage. A proposed change to add an interface to tempest/lib that doesnt have unit tests shouldnt be accepted. Ideally, these unit tests will provide sufficient coverage to ensure a stable interface moving forward.

Current Library APIs

CLI Testing Framework Usage

The cli.base module

class CLIClient(*username=''*, *password=''*, *tenant_name=''*, *uri=''*, *cli_dir=''*, *insecure=False*, *prefix=''*, *user_domain_name=None*, *user_domain_id=None*, *project_domain_name=None*, *project_domain_id=None*, *identity_api_version=None*, **args*, ***kwargs*)

Class to use OpenStack official python client CLIs with auth

Parameters

- **username** (*string*) The username to authenticate with
- **password** (*string*) The password to authenticate with
- **tenant_name** (*string*) The name of the tenant to use with the client calls
- **uri** (*string*) The auth uri for the OpenStack Deployment
- **cli_dir** (*string*) The path where the python client binaries are installed. defaults to /usr/bin
- **insecure** (*boolean*) if True, insecure is passed to python client binaries.
- **prefix** (*string*) prefix to insert before commands
- **user_domain_name** (*string*) Users domain name
- **user_domain_id** (*string*) Users domain ID
- **project_domain_name** (*string*) Projects domain name
- **project_domain_id** (*string*) Projects domain ID
- **identity_api_version** (*string*) Version of the Identity API

class ClientTestBase(**args*, ***kwargs*)

Base test class for testing the OpenStack client CLI interfaces.

execute(*cmd*, *action*, *flags=''*, *params=''*, *fail_ok=False*, *merge_stderr=False*, *cli_dir='/usr/bin'*, *prefix=''*)

Executes specified command for the given action.

Parameters

- **cmd** (*string*) command to be executed
- **action** (*string*) string of the cli command to run
- **flags** (*string*) any optional cli flags to use
- **params** (*string*) string of any optional positional args to use
- **fail_ok** (*boolean*) boolean if True an exception is not raised when the cli return code is non-zero
- **merge_stderr** (*boolean*) boolean if True the stderr buffer is merged into stdout
- **cli_dir** (*string*) The path where the cmd can be executed
- **prefix** (*string*) prefix to insert before command

The cli.output_parser module

Collection of utilities for parsing CLI clients output.

details(*output_lines*, *with_label=False*)

Return dict with details of first item (table) found in output.

details_multiple(*output_lines*, *with_label=False*)

Return list of dicts with item details from cli output tables.

If with_label is True, key __label is added to each items dict. For more about label see OutputParser.tables().

listing(*output_lines*)

Return list of dicts with basic item info parsed from cli output.

table(*output_lines*)

Parse single table from cli output.

Return dict with list of column names in headers key and rows in values key.

tables(*output_lines*)

Find all ascii-tables in output and parse them.

Return list of tables parsed from cli output as dicts. (see OutputParser.table())

And, if found, label key (separated line preceding the table) is added to each tables dict.

Decorators Usage Guide

The decorators module

attr(***kwargs*)

A decorator which applies the testtools attr decorator

This decorator applies the testtools.testcase.attr if it is in the list of attributes to testtools we want to apply.

Parameters

condition Optional condition which if true will apply the attr. If a condition is specified which is false the attr will not be applied to the test function. If not specified, the attr is always applied.

class cleanup_order(*func*)

Descriptor for base create function to cleanup based on caller.

There are functions created as classmethod and the cleanup was managed by the class with addClassResourceCleanup, In case the function called from a class level (resource_setup) its ok But when it is called from testcase level there is no reason to delete the resource when class tears down.

The testcase results will not reflect the resources cleanup because test may pass but the class cleanup fails. if the resources were created by testcase its better to let the testcase delete them and report failure part of the testcase

idempotent_id(*id*)

Stub for metadata decorator

related_bug(*bug*, *status_code=None*, *bug_type='launchpad'*)

A decorator useful to know solutions from launchpad/storyboard reports

Parameters

- **bug** The launchpad/storyboard bug number causing the test bug
- **bug_type** launchpad or storyboard, default launchpad

- **status_code** The status code related to the bug report

serial(*cls*)

A decorator to mark a test class for serial execution

skip_because(**args*, ***kwargs*)

A decorator useful to skip tests hitting known bugs

bug must be a number and `condition` must be true for the test to skip.

Parameters

- **bug** bug number causing the test to skip (launchpad or storyboard)
- **bug_type** launchpad or storyboard, default launchpad
- **condition** optional condition to be True for the skip to have place

Raises

testtools.TestCase.skipException if `condition` is True and bug is included

unstable_test(**args*, ***kwargs*)

A decorator useful to run tests hitting known bugs and skip it if fails

This decorator can be used in cases like:

- We have skipped tests with some bug and now bug is claimed to be fixed. Now we want to check the test stability so we use this decorator. The number of skipped cases with that bug can be counted to mark test stable again.
- There is test which is failing often, but not always. If there is known bug related to it, and someone is working on fix, this decorator can be used instead of skip_because. That will ensure that test is still run so new debug data can be collected from jobs logs but it will not make life of other developers harder by forcing them to recheck jobs more often.

bug must be a number for the test to skip.

Parameters

- **bug** bug number causing the test to skip (launchpad or storyboard)
- **bug_type** launchpad or storyboard, default launchpad

Raises

testtools.TestCase.skipException if test actually fails, and bug is included

Rest Client Usage

The rest_client module

class ResponseBody(*response*, *body=None*)

Class that wraps an http response and dict body into a single value.

Callers that receive this object will normally use it as a dict but can extract the response if needed.

class ResponseBodyData(*response*, *data*)

Class that wraps an http response and string data into a single value.

class ResponseBodyList(*response*, *body=None*)

Class that wraps an http response and list body into a single value.

Callers that receive this object will normally use it as a list but can extract the response if needed.

class RestClient(*auth_provider*, *service*, *region*, *endpoint_type='publicURL'*, *build_interval=1*, *build_timeout=60*, *disable_ssl_certificate_validation=False*, *ca_certs=None*, *trace_requests=''*, *name=None*, *http_timeout=None*, *proxy_url=None*, *follow_redirects=True*)

Unified OpenStack RestClient class

This class is used for building openstack api clients on top of. It is intended to provide a base layer for wrapping outgoing http requests in keystone auth as well as providing response code checking and error handling.

Parameters

- **auth_provider** an auth provider object used to wrap requests in auth
- **service** (*str*) The service name to use for the catalog lookup
- **region** (*str*) The region to use for the catalog lookup
- **name** (*str*) The endpoint name to use for the catalog lookup; this returns only if the service exists
- **endpoint_type** (*str*) The endpoint type to use for the catalog lookup
- **build_interval** (*int*) Time in seconds between to status checks in wait loops
- **build_timeout** (*int*) Timeout in seconds to wait for a wait operation.
- **disable_ssl_certificate_validation** (*bool*) Set to true to disable ssl certificate validation
- **ca_certs** (*str*) File containing the CA Bundle to use in verifying a TLS server cert
- **trace_requests** (*str*) Regex to use for specifying logging the entirety of the request and response payload
- **http_timeout** (*str*) Timeout in seconds to wait for the http request to return
- **proxy_url** (*str*) http proxy url to use.
- **follow_redirects** (*bool*) Set to false to stop following redirects.

Utils Usage

The misc module

singleton(*cls*)

Simple wrapper for classes that should only have a single instance.

API Microversion Testing Support in Tempest

Framework to support API Microversion testing

Many of the OpenStack components have implemented API microversions. It is important to test those microversions in Tempest or external plugins. Tempest now provides stable interfaces to support testing the API microversions. Based on the microversion range coming from the combination of both configuration and each test case, APIs requests will be made with the selected microversion.

This document explains the interfaces needed for microversion testing.

The api_version_request module

class APIVersionRequest(*version_string=None*)

This class represents an API Version Request.

This class provides convenience methods for manipulation and comparison of version numbers that we need to do to implement microversions.

Parameters

version_string String representation of APIVersionRequest. Correct format is X.Y, where X and Y are int values. None value should be used to create Null APIVersionRequest, which is equal to 0.0

The api_version_utils module

class BaseMicroversionTest

Mixin class for API microversion test class.

assert_version_header_matches_request(*api_microversion_header_name*, *api_microversion*, *response_header*)

Checks API microversion in response header

Verify whether microversion is present in response header and with specified api_microversion value.

Parameters

- **api_microversion_header_name** Microversion header name Example- X-OpenStack-Nova-API-Version
- **api_microversion** Microversion number like 2.10, type str.
- **response_header** Response header where microversion is expected to be present.

check_skip_with_microversion(*test_min_version*, *test_max_version*, *cfg_min_version*, *cfg_max_version*)

Checks API microversions range and returns whether test needs to be skip

Compare the test and configured microversion range and returns whether test microversion range is out of configured one. This method can be used to skip the test based on configured and test microversion range.

Parameters

- **test_min_version** Test Minimum Microversion

- **test_max_version** Test Maximum Microversion
- **cfg_min_version** Configured Minimum Microversion
- **cfg_max_version** Configured Maximum Microversion

Returns
boolean

compare_version_header_to_response(*api_microversion_header_name*, *api_microversion*, *response_header*, *operation='eq'*)

Compares API microversion in response header to `api_microversion`.

Compare the `api_microversion` value in response header if microversion header is present in response, otherwise return false.

To make this function work for APIs which do not return microversion header in response (example compute v2.0), this function does *not* raise InvalidHTTPResponseHeader.

Parameters

- **api_microversion_header_name** Microversion header name. Example: Openstack-Api-Version.
- **api_microversion** Microversion number. Example:
  - 2.10 for the old-style header name, X-OpenStack-Nova-API-Version
  - Compute 2.10 for the new-style header name, Openstack-Api-Version
- **response_header** Response header where microversion is expected to be present.
- **operation** The boolean operation to use to compare the `api_microversion` to the microversion in `response_header`. Can be lt, eq, gt, le, ne, ge. Default is eq. The operation type should be based on the order of the arguments: `api_microversion` <operation> `response_header` microversion.

Returns
True if the comparison is logically true, else False if the comparison is logically false or if `api_microversion_header_name` is missing in the `response_header`.

Raises
InvalidParam If the operation is not lt, eq, gt, le, ne or ge.

select_request_microversion(*test_min_version*, *cfg_min_version*)

Select microversion from test and configuration min version.

Compare requested microversion and return the maximum microversion out of those.

Parameters

- **test_min_version** Test Minimum Microversion
- **cfg_min_version** Configured Minimum Microversion

Returns
Selected microversion string

Authentication Framework Usage

The auth module

class AuthProvider(*credentials*, *scope='project'*)

Provide authentication

class Credentials(***kwargs*)

Set of credentials for accessing OpenStack services

ATTRIBUTES: list of valid class attributes representing credentials.

class KeystoneAuthProvider(*credentials*, *auth_url*, *disable_ssl_certificate_validation=None*, *ca_certs=None*, *trace_requests=None*, *scope='project'*, *http_timeout=None*, *proxy_url=None*)

class KeystoneV2AuthProvider(*credentials*, *auth_url*, *disable_ssl_certificate_validation=None*, *ca_certs=None*, *trace_requests=None*, *scope='project'*, *http_timeout=None*, *proxy_url=None*)

Provides authentication based on the Identity V2 API

The Keystone Identity V2 API defines both unscoped and project scoped tokens. This auth provider only implements project.

class KeystoneV2Credentials(***kwargs*)

class KeystoneV3AuthProvider(*credentials*, *auth_url*, *disable_ssl_certificate_validation=None*, *ca_certs=None*, *trace_requests=None*, *scope='project'*, *http_timeout=None*, *proxy_url=None*)

Provides authentication based on the Identity V3 API

class KeystoneV3Credentials(***kwargs*)

Credentials suitable for the Keystone Identity V3 API

get_credentials(*auth_url*, *fill_in=True*, *identity_version='v2'*, *disable_ssl_certificate_validation=None*, *ca_certs=None*, *trace_requests=None*, *http_timeout=None*, *proxy_url=None*, ***kwargs*)

Builds a credentials object based on the configured auth_version

Parameters

- **(string)** (*identity_version*) Full URI of the OpenStack Identity API(Keystone) which is used to fetch the token from Identity service.
- **(boolean)** (*fill_in*) obtain a token and fill in all credential details provided by the identity service. When fill_in is not specified, credentials are not validated. Validation can be invoked by invoking `is_valid()`
- **(string)** identity API version is used to select the matching auth provider and credentials class
- **disable_ssl_certificate_validation** whether to enforce SSL certificate validation in SSL API requests to the auth system
- **ca_certs** CA certificate bundle for validation of certificates in SSL API requests to the auth system
- **trace_requests** trace in log API requests to the auth system

- **http_timeout** timeout in seconds to wait for the http request to return
- **proxy_url** URL of HTTP(s) proxy used when fill_in is True
- **(dict)** (*kwargs*) Dict of credential key/value pairs

Examples:

Returns credentials from the provided parameters: »> get_credentials(username=foo, password=bar)

Returns credentials including IDs: »> get_credentials(username=foo, password=bar, fill_in=True)

Service Clients Usage

Tests make requests against APIs using service clients. Service clients are specializations of the `RestClient` class. The service clients that cover the APIs exposed by a service should be grouped in a service clients module. A service clients module is a Python module where all service clients are defined. If major API versions are available, submodules should be defined, one for each version.

The `ClientsFactory` class helps to initialize all clients of a specific service client module from a set of shared parameters.

The `ServiceClients` class provides a convenient way to get access to all available service clients initialized with a provided set of credentials.

The clients management module

class ClientsFactory(*module_path*, *client_names*, *auth_provider*, ***kwargs*)

Builds service clients for a service client module

This class implements the logic of feeding service client parameters to service clients from a specific module. It allows setting the parameters once and obtaining new instances of the clients without the need of passing any parameter.

ClientsFactory can be used directly, or consumed via the *ServiceClients* class, which manages the authorization part.

class ServiceClients(*credentials*, *identity_uri*, *region=None*, *scope=None*, *disable_ssl_certificate_validation=True*, *ca_certs=None*, *trace_requests=''*, *client_parameters=None*, *proxy_url=None*)

Service client provider class

The ServiceClients object provides a useful means for tests to access service clients configured for a specified set of credentials. It hides some of the complexity from the authorization and configuration layers.

Examples:

```
# johndoe is a tempest.lib.auth.Credentials type instance
johndoe_clients = clients.ServiceClients(johndoe, identity_uri)

# List servers in default region
johndoe_servers_client = johndoe_clients.compute.ServersClient()
johndoe_servers = johndoe_servers_client.list_servers()
```

(continues on next page)

(continued from previous page)

```
# List servers in Region B
johndoe_servers_client_B = johndoe_clients.compute.ServersClient(
    region='B')
johndoe_servers = johndoe_servers_client_B.list_servers()
```

available_modules()

Set of service client modules available in Tempest and plugins

Set of stable service clients from Tempest and service clients exposed by plugins. This set of available modules can be used for automatic configuration.

Raises

PluginRegistrationException if a plugin exposes a service_version already defined by Tempest or another plugin.

Examples:

```
from tempest import config
params = {}
for service_version in available_modules():
    service = service_version.split('.')[0]
    params[service] = config.service_client_config(service)
service_clients = ServiceClients(creds, identity_uri,
                                 client_parameters=params)
```

tempest_modules()

Dict of service client modules available in Tempest.

Provides a dict of stable service modules available in Tempest, with `service_version` as key, and the module object as value.

Compute service client modules

Compute Client Usage

class ServersClient(*auth_provider*, *service*, *region*, *enable_instance_password=True*, ***kwargs*)

Service client for the resource /servers

Credential Providers

These library interfaces are used to deal with allocating credentials on demand either dynamically by calling Keystone to allocate new credentials, or from a list of preprovisioned credentials. These 2 modules are implementations of the same abstract credential providers class and can be used interchangeably. However, each implementation has some additional parameters that are used to influence the behavior of the modules. The API reference at the bottom of this doc shows the interface definitions for both modules, however, that may be a bit opaque. You can see some examples of how to leverage this interface below.

Initialization Example

This example is from Tempest itself (from tempest/common/credentials_factory.py just modified slightly) and is how it initializes the credential provider based on config:

```
from tempest import config
from tempest.lib.common import dynamic_creds
from tempest.lib.common import preprov_creds

CONF = config.CONF

def get_credentials_provider(name, network_resources=None,
                             force_tenant_isolation=False,
                             identity_version=None):
    # If a test requires a new account to work, it can have it via forcing
    # dynamic credentials. A new account will be produced only for that test.
    # In case admin credentials are not available for the account creation,
    # the test should be skipped else it will fail.
    identity_version = identity_version or CONF.identity.auth_version
    if CONF.auth.use_dynamic_credentials or force_tenant_isolation:
        admin_creds = get_configured_admin_credentials(
            fill_in=True, identity_version=identity_version)
        return dynamic_creds.DynamicCredentialProvider(
            name=name,
            network_resources=network_resources,
            identity_version=identity_version,
            admin_creds=admin_creds,
            identity_admin_domain_scope=CONF.identity.admin_domain_scope,
            identity_admin_role=CONF.identity.admin_role,
            extra_roles=CONF.auth.tempest_roles,
            neutron_available=CONF.service_available.neutron,
            project_network_cidr=CONF.network.project_network_cidr,
            project_network_mask_bits=CONF.network.project_network_mask_bits,
            public_network_id=CONF.network.public_network_id,
            create_networks=(CONF.auth.create_isolated_networks and not
                             CONF.network.shared_physical_network),
            resource_prefix='tempest',
            credentials_domain=CONF.auth.default_credentials_domain_name,
            admin_role=CONF.identity.admin_role,
            identity_uri=CONF.identity.uri_v3,
            identity_admin_endpoint_type=CONF.identity.v3_endpoint_type)
    else:
        if CONF.auth.test_accounts_file:
            # Most params are not relevant for pre-created accounts
            return preprov_creds.PreProvisionedCredentialProvider(
                name=name, identity_version=identity_version,
                accounts_lock_dir=lockutils.get_lock_path(CONF),
                test_accounts_file=CONF.auth.test_accounts_file,
                object_storage_operator_role=CONF.object_storage.operator_
↪role,
                object_storage_reseller_admin_role=reseller_admin_role,
```

(continues on next page)

(continued from previous page)

```
                credentials_domain=CONF.auth.default_credentials_domain_name,
                admin_role=CONF.identity.admin_role,
                identity_uri=CONF.identity.uri_v3,
                identity_admin_endpoint_type=CONF.identity.v3_endpoint_type)
        else:
            raise exceptions.InvalidConfiguration(
                'A valid credential provider is needed')
```

This function just returns an initialized credential provider class based on the config file. The consumer of this function treats the output as the same regardless of whether its a dynamic or preprovisioned provider object.

Dealing with Credentials

Once you have a credential provider object created the access patterns for allocating and removing credentials are the same across both the dynamic and preprovisioned credentials. These are defined in the abstract CredentialProvider class. At a high level, the credentials provider enables you to get 3 basic types of credentials at once (per object): primary, alt, and admin. Youre also able to allocate a credential by role. These credentials are tracked by the provider object and delete must be called manually, otherwise, the created resources will not be deleted (or returned to the pool in the case of preprovisioned creds).

Examples

Continuing from the example above, to allocate credentials by the 3 basic types you can do the following:

```
provider = get_credentials_provider('my_tests')
primary_creds = provider.get_primary_creds()
alt_creds = provider.get_alt_creds()
admin_creds = provider.get_admin_creds()
# Make sure to delete the credentials when you're finished
provider.clear_creds()
```

To create and interact with credentials by role you can do the following:

```
provider = get_credentials_provider('my_tests')
my_role_creds = provider.get_creds_by_role({'roles': ['my_role']})
# provider.clear_creds() will clear all creds including those allocated by
# role
provider.clear_creds()
```

When multiple roles are specified a set of creds with all the roles assigned will be allocated:

```
provider = get_credentials_provider('my_tests')
my_role_creds = provider.get_creds_by_role({'roles': ['my_role',
                                                      'my_other_role']})
# provider.clear_creds() will clear all creds including those allocated by
# role
provider.clear_creds()
```

If you need multiple sets of credentials with the same roles you can also do this by leveraging the `force_new` kwarg:

```
provider = get_credentials_provider('my_tests')
my_role_creds = provider.get_creds_by_role({'roles': ['my_role']})
my_role_other_creds = provider.get_creds_by_role({'roles': ['my_role']},
                                                 force_new=True)
# provider.clear_creds() will clear all creds including those allocated by
# role
provider.clear_creds()
```

API Reference

The dynamic credentials module

class DynamicCredentialProvider(*identity_version*, *name=None*, *network_resources=None*, *credentials_domain=None*, *admin_role=None*, *admin_creds=None*, *identity_admin_domain_scope=False*, *identity_admin_role='admin'*, *extra_roles=None*, *neutron_available=False*, *create_networks=True*, *project_network_cidr=None*, *project_network_mask_bits=None*, *public_network_id=None*, *resource_prefix=None*, *identity_admin_endpoint_type='public'*, *identity_uri=None*)

Creates credentials dynamically for tests

A credential provider that, based on an initial set of admin credentials, creates new credentials on the fly for tests to use and then discard.

Parameters

- **identity_version** (*str*) identity API version to use *v2* or *v3*
- **admin_role** (*str*) name of the admin role added to admin users
- **name** (*str*) names of dynamic resources include this parameter when specified
- **credentials_domain** (*str*) name of the domain where the users are created. If not defined, the project domain from admin_credentials is used
- **network_resources** (*dict*) network resources to be created for the created credentials
- **admin_creds** (Credentials) initial admin credentials
- **identity_admin_domain_scope** (*bool*) Set to true if admin should be scoped to the domain. By default this is False and the admin role is scoped to the project.
- **identity_admin_role** (*str*) The role name to use for admin
- **extra_roles** (*list*) A list of strings for extra roles that should be assigned to all created users
- **neutron_available** (*bool*) Whether we are running in an environment with neutron
- **create_networks** (*bool*) Whether dynamic project networks should be created or not

- **project_network_cidr** The CIDR to use for created project networks
- **project_network_mask_bits** The network mask bits to use for created project networks
- **public_network_id** The id for the public network to use
- **identity_admin_endpoint_type** The endpoint type for identity admin clients. Defaults to public.
- **identity_uri** Identity URI of the target cloud

The pre-provisioned credentials module

class PreProvisionedCredentialProvider(*identity_version*, *test_accounts_file*, *accounts_lock_dir*, *name=None*, *credentials_domain=None*, *admin_role=None*, *object_storage_operator_role=None*, *object_storage_reseller_admin_role=None*, *identity_uri=None*)

Credentials provider using pre-provisioned accounts

This credentials provider loads the details of pre-provisioned accounts from a YAML file, in the format specified by `etc/accounts.yaml.sample`. It locks accounts while in use, using the external locking mechanism, allowing for multiple python processes to share a single account file, and thus running tests in parallel.

The accounts_lock_dir must be generated using *lockutils.get_lock_path* from the oslo.concurrency library. For instance:

```
accounts_lock_dir = os.path.join(lockutils.get_lock_path(CONF),
                                 'test_accounts')
```

Role names for object storage are optional as long as the *operator* and *reseller_admin* credential types are not used in the accounts file.

Parameters

- **identity_version** identity version of the credentials
- **admin_role** name of the admin role
- **test_accounts_file** path to the accounts YAML file
- **accounts_lock_dir** the directory for external locking
- **name** name of the hash file (optional)
- **credentials_domain** name of the domain credentials belong to (if no domain is configured)
- **object_storage_operator_role** name of the role
- **object_storage_reseller_admin_role** name of the role
- **identity_uri** Identity URI of the target cloud

Validation Resources

The validation_resources module

class ValidationResourcesFixture(*clients*, *keypair=False*, *floating_ip=False*, *security_group=False*, *security_group_rules=False*, *ethertype='IPv4'*, *use_neutron=True*, *floating_network_id=None*, *floating_network_name=None*)

Fixture to provision and cleanup validation resources

clear_validation_resources(*clients*, *keypair=None*, *floating_ip=None*, *security_group=None*, *use_neutron=True*)

Cleanup resources for VM ping/ssh testing

Cleanup a set of resources provisioned via *create_validation_resources*. In case of errors during cleanup, the exception is logged and the cleanup process is continued. The first exception that was raised is re-raised after the cleanup is complete.

Parameters

- **clients** Instance of *tempest.lib.services.clients.ServiceClients* or of a subclass of it. Resources are provisioned using clients from *clients*.
- **keypair** A dictionary with the keypair to be deleted. Defaults to None.
- **floating_ip** A dictionary with the floating_ip to be deleted. Defaults to None.
- **security_group** A dictionary with the security_group to be deleted. Defaults to None.
- **use_neutron** When True resources are provisioned via neutron, when False resources are provisioned via nova.

Examples:

```
from tempest.common import validation_resources as vr
from tempest.lib import auth
from tempest.lib.services import clients

creds = auth.get_credentials('http://mycloud/identity/v3',
                             username='me', project_name='me',
                             password='secret', domain_name='Default')
osclients = clients.ServiceClients(creds, 'http://mycloud/identity/v3')
# Request keypair and floating IP
resources = dict(keypair=True, security_group=False,
                 security_group_rules=False, floating_ip=True)
resources = vr.create_validation_resources(
    osclients, validation_resources=resources, use_neutron=True,
    floating_network_id='4240E68E-23DA-4C82-AC34-9FEFAA24521C')

# Now cleanup the resources
try:
    vr.clear_validation_resources(osclients, use_neutron=True,
                                  **resources)
```

(continues on next page)

(continued from previous page)

```
except Exception as e:
    LOG.exception('Something went wrong during cleanup, ignoring')
```

create_ssh_security_group(*clients*, *add_rule=False*, *ethertype='IPv4'*, *use_neutron=True*)

Create a security group for ping/ssh testing

Create a security group to be attached to a VM using the nova or neutron clients. If rules are added, the group can be attached to a VM to enable connectivity validation over ICMP and further testing over SSH.

Parameters

- **clients** Instance of *tempest.lib.services.clients.ServiceClients* or of a subclass of it. Resources are provisioned using clients from *clients*.
- **add_rule** Whether security group rules are provisioned or not. Defaults to *False*.
- **ethertype** IPv4 or IPv6. Honoured only in case neutron is used.
- **use_neutron** When True resources are provisioned via neutron, when False resources are provisioned via nova.

Returns

A dictionary with the security group as returned by the API.

Examples:

```
from tempest.common import validation_resources as vr
from tempest.lib import auth
from tempest.lib.services import clients

creds = auth.get_credentials('http://mycloud/identity/v3',
                             username='me', project_name='me',
                             password='secret', domain_name='Default')
osclients = clients.ServiceClients(creds, 'http://mycloud/identity/v3')
# Security group for IPv4 tests
sg4 = vr.create_ssh_security_group(osclients, add_rule=True)
# Security group for IPv6 tests
sg6 = vr.create_ssh_security_group(osclients, ethertype='IPv6',
                                   add_rule=True)
```

create_validation_resources(*clients*, *keypair=False*, *floating_ip=False*, *security_group=False*, *security_group_rules=False*, *ethertype='IPv4'*, *use_neutron=True*, *floating_network_id=None*, *floating_network_name=None*)

Provision resources for VM ping/ssh testing

Create resources required to be able to ping / ssh a virtual machine: keypair, security group, security group rules and a floating IP. Which of those resources are required may depend on the cloud setup and on the specific test and it can be controlled via the corresponding arguments.

Provisioned resources are returned in a dictionary.

Parameters

- **clients** Instance of *tempest.lib.services.clients.ServiceClients* or of a subclass of it. Resources are provisioned using clients from *clients*.
- **keypair** Whether to provision a keypair. Defaults to False.
- **floating_ip** Whether to provision a floating IP. Defaults to False.
- **security_group** Whether to provision a security group. Defaults to False.
- **security_group_rules** Whether to provision security group rules. Defaults to False.
- **ethertype** IPv4 or IPv6. Honoured only in case neutron is used.
- **use_neutron** When True resources are provisioned via neutron, when False resources are provisioned via nova.
- **floating_network_id** The id of the network used to provision a floating IP. Only used if a floating IP is requested and with neutron.
- **floating_network_name** The name of the floating IP pool used to provision the floating IP. Only used if a floating IP is requested and with nova-net.

Returns

A dictionary with the resources in the format they are returned by the API. Valid keys are keypair, floating_ip and security_group.

Examples:

```
from tempest.common import validation_resources as vr
from tempest.lib import auth
from tempest.lib.services import clients

creds = auth.get_credentials('http://mycloud/identity/v3',
                             username='me', project_name='me',
                             password='secret', domain_name='Default')
osclients = clients.ServiceClients(creds, 'http://mycloud/identity/v3')
# Request keypair and floating IP
resources = dict(keypair=True, security_group=False,
                 security_group_rules=False, floating_ip=True)
resources = vr.create_validation_resources(
    osclients, use_neutron=True,
    floating_network_id='4240E68E-23DA-4C82-AC34-9FEFAA24521C',
    **resources)

# The floating IP to be attached to the VM
floating_ip = resources['floating_ip']['ip']
```

CHAPTER

SIX

SEARCH

- OpenStack wide search: Search the wider set of OpenStack documentation, including forums.

PYTHON MODULE INDEX

C

i

l

n

o

s

V

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V