

# Tempest Stable Interfaces for OpenStack Integration Testing

Andrea Frittoli  
andrea.frittoli@hpe.com  
andreaaf on Freenode

Apr 25, 2016

[https://github.com/andreafrittoli/tempest\\_stable\\_interfaces](https://github.com/andreafrittoli/tempest_stable_interfaces)

Neutron or Newton

/ˈnjuːt(r)ən/

# What is OpenStack QA?

- ▶ Official Mission Statement:  
*Develop, maintain, and initiate tools and plans to ensure the upstream stability and quality of OpenStack, and its release readiness at any point during the release cycle.*

## Current QA Projects

- ▶ `eslint-config-openstack`
- ▶ `bashate`
- ▶ `hacking`
- ▶ `tempest`
- ▶ `tempest-lib` (deprecated)
- ▶ `grenade`
- ▶ `devstack`
- ▶ `devstack-plugin-ceph`
- ▶ `devstack-vagrant`
- ▶ `stackviz`
- ▶ `openstack-health` dashboard
- ▶ `os-testr`
- ▶ `devstack-plugin-cookiecutter`
- ▶ `tempest-plugin-cookiecutter`
- ▶ `os-performance-tools`

# What is Tempest



# Tempest in the Big Tent

- ▶ `tempest_lib` or `tempest.lib`
- ▶ `tempest` plugins

## Current Projects QA directly supports in-tree

- ▶ Keystone
- ▶ Nova
- ▶ Glance
- ▶ Cinder
- ▶ Neutron
- ▶ Swift
  
- ▶ Tests executed in integrated-gate jobs

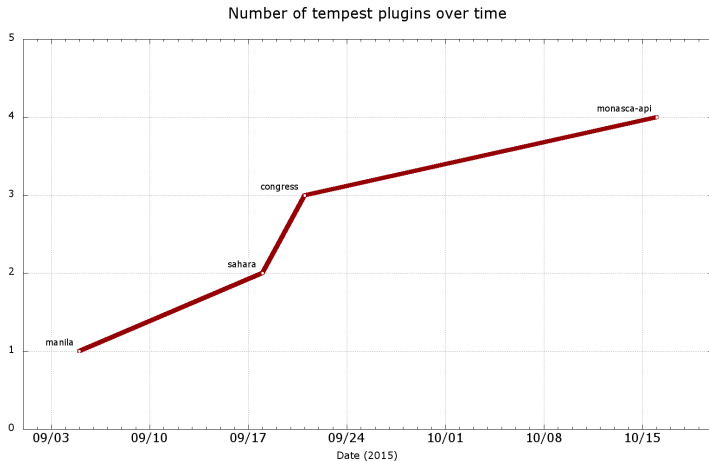
Where do other Tempest based tests live in OpenStack?



## Integration Tests outside of Tempest tree

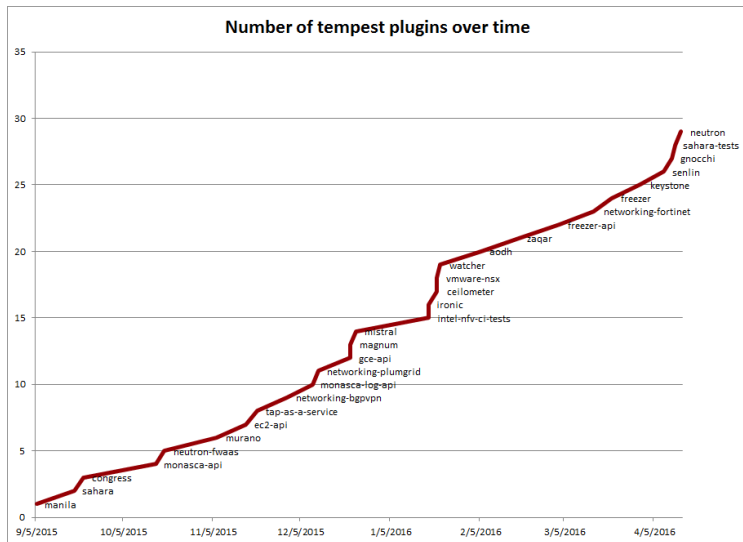
- ▶ Tempest Plugins (29)
  - ▶ CLI tests for clients (16)
  - ▶ Other functional/integration tests
- 
- ▶ Not executed against tempest
  - ▶ Should only use tempest stable interfaces

# Tempest Plugins at the end of Liberty



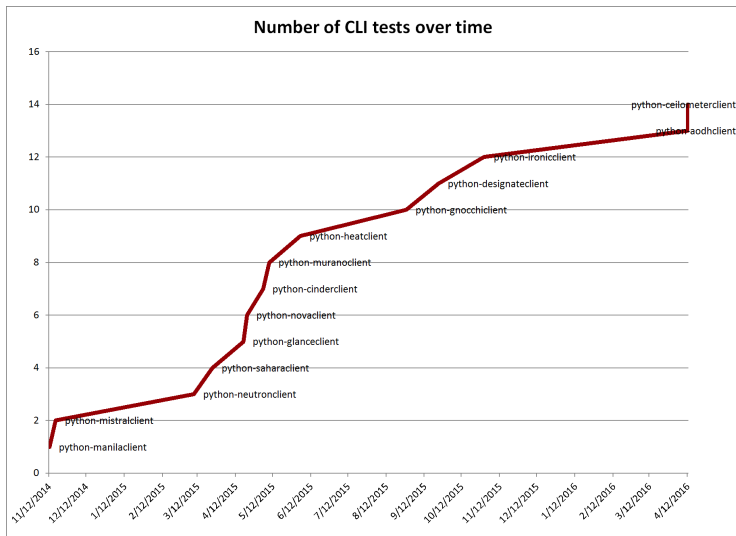
Source: `git blame -L '/^tempest/,+1' setup.cfg | awk '{print $1}' | xargs git log -1 --format=%cd --date=short`

# Tempest Plugins over time



Source: `git blame -L '/^tempest/,+1' setup.cfg | awk '{print $1}' | xargs git log -1 --format=%cd --date=short`

# CLI Tests over time



Source: `git blame -L '/^(from|import) tempest[_.]+lib.cli/,+1' *.py`

## Other Tempest Tests

- ▶ WIP Tempest Plugins
  - ▶ kingbird
  - ▶ vitrage
- ▶ Tempest Tests (potential plugins)
  - ▶ blazar
  - ▶ designate
  - ▶ networking-l2gw
  - ▶ networking-vsphere
  - ▶ neutron-lbaas
  - ▶ neutron-vpnaas

## Functional/Integration Tests

- ▶ cerberus (rest\_client, auth, clients)
- ▶ cue (rest\_client, test base class)
- ▶ solum (rest\_client, auth)
  
- ▶ astara (utils)
- ▶ barbican (utils)
- ▶ python-keystoneclient (test base class)
- ▶ python-openstackclient (utils)
- ▶ tacker (test base class)

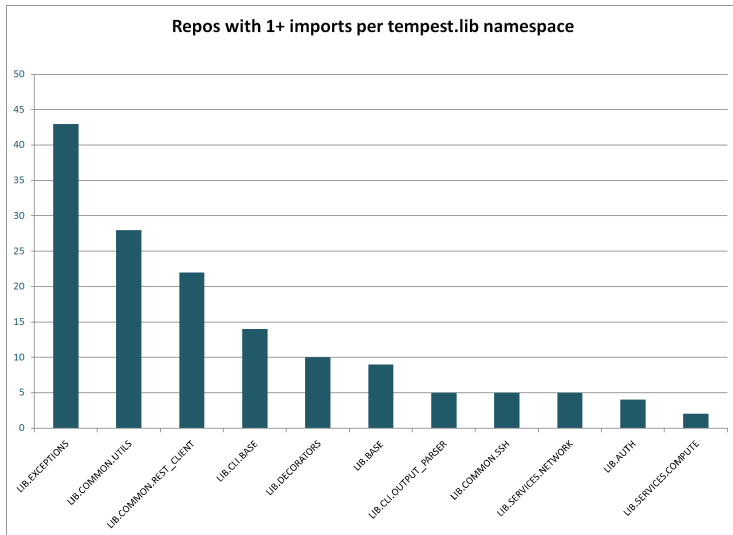
What are Tempest (Stable) Interfaces, how are they used?

# Tempest Stable APIs

- ▶ Common: rest client, microversion, ssh client, utils
- ▶ Service clients: identity, network, compute
- ▶ Authentication providers
- ▶ CLI test framework
- ▶ Decorators, exceptions
- ▶ Base test class
- ▶ Commands: check\_uuid, skip tracker



# Tempest Stable APIs

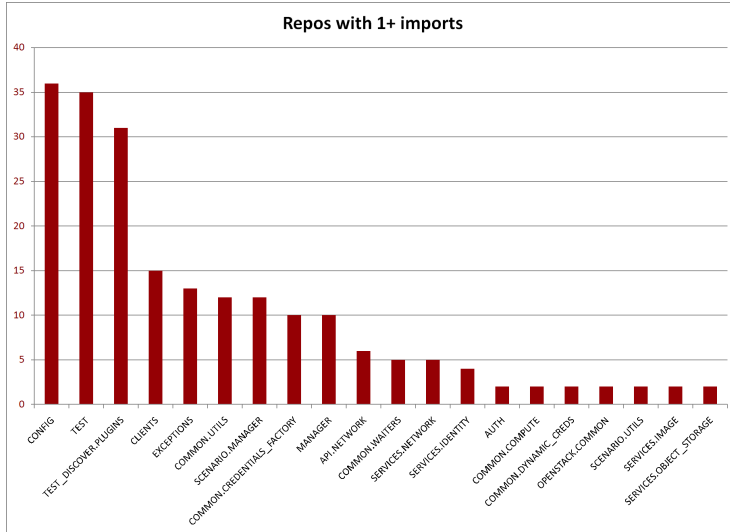


Source: [codesearch.openstack.org](https://codesearch.openstack.org)

## Tempest Internal APIs planned to become Stable

- ▶ Service clients: volume, image, object-storage
- ▶ Credential providers
- ▶ Client manager
- ▶ Plugin

# Tempest Internal APIs



Source: [codesearch.openstack.org](https://codesearch.openstack.org)

How to use these interfaces to write a Tempest plugin?

# Tempest Plugins Interface

- ▶ Integrates external tests into a tempest run
- ▶ Unifies configuration between plugin(s) and in-tree
- ▶ Integrates custom service clients (planned)
- ▶ Based on stevedore extension manager
- ▶ Automatically discovered when installed

# Tempest Plugins Interface

```
# Import config for 'register_opt_group'
from tempest import config
# Import the plugin base class
from tempest.test_discover import plugins

from manila_tempest_tests import config as config_share

class ManilaTempestPlugin(plugins.TempestPlugin):

    def register_opts(self, conf):
        config.register_opt_group(
            conf, config_share.service_available_group,
            config_share.ServiceAvailableGroup)
        config.register_opt_group(conf, config_share.share_group,
                                   config_share.ShareGroup)
```

► Full code at: [http://git.openstack.org/cgit/openstack/manila/tree/manila\\_tempest\\_tests/plugin.py](http://git.openstack.org/cgit/openstack/manila/tree/manila_tempest_tests/plugin.py)

## Rest Client and Service Clients

- ▶ ReST API Calls with different HTTP methods
- ▶ Decorate requests using supplied auth provider
- ▶ Validate HTTP return codes
- ▶ Handle HTTP non-2xx return codes as custom exceptions
  
- ▶ Methods for API calls
- ▶ Minimal response body parsing
- ▶ Pass any parameter to API calls

# Rest Client and Service Clients

```
from tempest.lib.common import rest_client

# Use tempest base client manager
from tempest import manager
from tempest.services.image.v1.json.images_client import ImagesClient

class TelemetryClient(rest_client.RestClient):

    def create_sample(self, meter_name, sample_list):
        uri = "%s/meters/%s" % (self.uri_prefix, meter_name)
        body = self.serialize(sample_list)
        resp, body = self.post(uri, body)
        self.expected_success(200, resp.status)
        body = self.deserialize(body)
        return rest_client.ResponseBody(resp, body)

class Manager(manager.Manager):

    def set_image_client(self):
        self.image_client = ImagesClient(self.auth_provider,
                                         **self.image_params)

    def set_telemetry_client(self):
        self.telemetry_client = TelemetryClient(self.auth_provider,
                                                **self.telemetry_params)
```

► Full code at: <http://git.openstack.org/cgit/openstack/ceilometer/tree/ceilometer/tests/tempest/service/client.py>



# Authentication Layer

- ▶ Credentials object
- ▶ Select endpoints from the catalogue
- ▶ Decorate requests with identity v2 and v3 auth data
- ▶ Inject alternate auth data

# Authentication Layer

```
from tempest.lib import auth

def get_auth_provider_class(credentials):
    if isinstance(credentials, auth.KeystoneV3Credentials):
        return auth.KeystoneV3AuthProvider, CONF.identity.uri_v3
    else:
        return auth.KeystoneV2AuthProvider, CONF.identity.uri

def get_auth_provider(credentials, pre_auth=False):
    default_params = {
        'disable_ssl_certificate_validation':
            CONF.identity.disable_ssl_certificate_validation,
        'ca_certs': CONF.identity.ca_certificates_file,
        'trace_requests': CONF.debug.trace_requests
    }
    auth_provider_class, auth_url = get_auth_provider_class(
        credentials)
    _auth_provider = auth_provider_class(credentials, auth_url,
                                         **default_params)
    if pre_auth:
        _auth_provider.set_auth()
    return _auth_provider
```

► Full code at: <http://git.openstack.org/cgi/openstack/tempest/tree/tempest/manager.py>

# Client Managers

- ▶ One object to access all service clients
- ▶ Bound to a set of credentials
- ▶ Hide the complexity of the auth layer
- ▶ Not yet in the lib namespace (WIP)
  
- ▶ Stable interface to register service clients from plugins
- ▶ Lazy loading of clients

# Client Managers

```
from tempest import manager

from neutron.tests.tempest.services.network.json.network_client import \
    NetworkClientJSON

class Manager(manager.Manager):

    def __init__(self, credentials=None, service=None):
        super(Manager, self).__init__(credentials=credentials)

        self.network_client = NetworkClientJSON(
            self.auth_provider,
            CONF.network.catalog_type,
            CONF.network.region or CONF.identity.region,
            endpoint_type=CONF.network.endpoint_type,
            build_interval=CONF.network.build_interval,
            build_timeout=CONF.network.build_timeout,
            **self.default_params)
```

- Full code at: <http://git.openstack.org/cgit/openstack/neutron/tree/neutron/tests/tempest/api/clients.py>

# Credential Providers

- ▶ Supply test cases with credentials
- ▶ Manage multiple test account for parallel test execution
- ▶ Manage account specific network resources
- ▶ Not yet in the lib namespace (WIP)
  
- ▶ Dynamic Credential Provider
- ▶ Preprovisioned Credential Provider

# Testing microversions

- ▶ Define acceptable microversion range for test class
- ▶ Match configured microversion range with tests
- ▶ Select microversion to be sent via API

# Testing microversions

```
from tempest.lib.common import api_version_utils
import tempest.test

class BaseV2ComputeTest(api_version_utils.BaseMicroversionTest,
                        tempest.test.BaseTestCase):

    @classmethod
    def skip_checks(cls):
        super(BaseV2ComputeTest, cls).skip_checks()
        if not CONF.service_available.nova:
            raise cls.skipException("Nova is not available")
        cfg_min_version = CONF.compute.min_microversion
        cfg_max_version = CONF.compute.max_microversion
        api_version_utils.check_skip_with_microversion(
            cls.min_microversion,
            cls.max_microversion,
            cfg_min_version,
            cfg_max_version)

    @classmethod
    def resource_setup(cls):
        super(BaseV2ComputeTest, cls).resource_setup()
        cls.request_microversion = (
            api_version_utils.select_request_microversion(
                cls.min_microversion,
                CONF.compute.min_microversion))
```

► Full code at: <http://git.openstack.org/cgit/openstack/tempest/tree/tempest/api/compute/base.py>

## Miscellaneous Utils

- ▶ Generate random test data
- ▶ SSH client
- ▶ Skip decorators
- ▶ Test Attributes (not yet stable)



Which interfaces do you need to implement CLI tests?

## CLI Tests Interfaces

- ▶ An *execute* command to drive clients via CLI
- ▶ A *CLIClient* class that wraps *execute* for clients
- ▶ An *output\_parser* module with helpers to parse clients output
- ▶ A *ClientTestBase* class with clients and output parsers

# CLI Tests Interfaces

```
from tempest_lib.cli import base

class MistralCLIAuth(base.ClientTestBase):

    def _get_clients(self):
        return base.CLIClient(
            username=creds['username'],
            password=creds['password'],
            tenant_name=creds['tenant_name'],
            uri=creds['auth_url'],
            cli_dir=CLI_DIR)

    def mistral(self, action, flags='', params='', fail_ok=False):
        """Executes Mistral command."""
        mistral_url_op = "--os-mistrmistralal-url %s" % self._mistral_url

        if 'WITHOUT_AUTH' in os.environ:
            return base.execute(
                'mistral %s' % mistral_url_op, action, flags, params,
                fail_ok, merge_stderr=False, cli_dir='')
        else:
            return self.clients.cmd_with_auth(
                'mistral %s' % mistral_url_op, action, flags, params,
                fail_ok)
```

► Full code at: <http://git.openstack.org/cgit/openstack/python-mistralclient/tree/mistralclient/tests/functional/cli/base.py>

How are Tempest interfaces used today in OpenStack?

## Where to get more information

- ▶ Tempest Stable Interfaces Docs:  
<http://docs.openstack.org/developer/tempest/library.html>
- ▶ Tempest Plugin Docs: <http://docs.openstack.org/developer/tempest/plugin.html>
- ▶ Tempest External Plugins Presentation:  
[https://github.com/mtreinish/external\\_plugins](https://github.com/mtreinish/external_plugins)
- ▶ #openstack-qa on Freenode

Questions?