Tempest Stable Interfaces for OpenStack Integration Testing

Andrea Frittoli
andrea.frittoli@hpe.com
`andreaf on Freenode`

Apr 25, 2016

https://github.com/andreafrittoli/templest_stable_interfaces

Neutron or Newton

/ˈnjuːt(r)ɒn/

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

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

Current Projects QA directly supports in-tree

- Keystone
- Nova
- Glance
- Cinder
- Neutron
- Swift

Integration Tests in Tempest

- Six core services (and 1 horizon scenario)
  - tests executed in integrated-gate jobs
- Other services (to be moved out of tree)
  - data_processing, database, orchestration tests executed in layer4 job
  - telemetry tests executed in ceilometer-mysql-neutron job
  - ironic tests executed in ironic-agent_ssh job

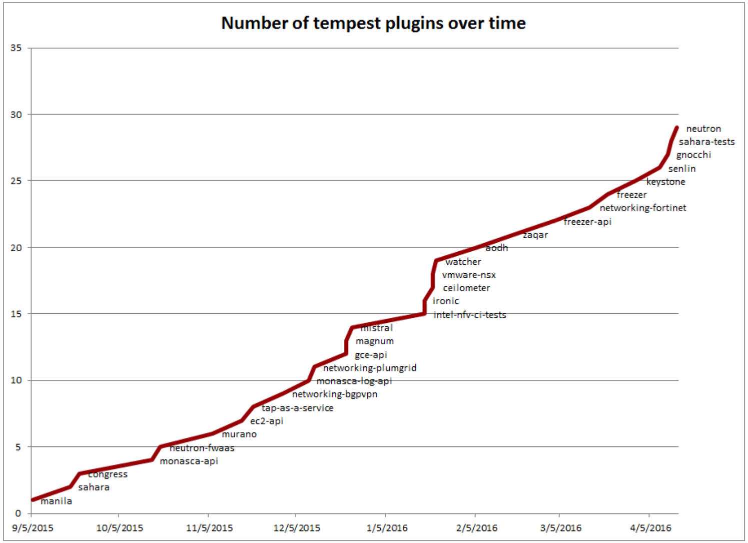
Who uses Tempest interfaces in OpenStack?

Integration Tests outside of tempest tree

- Tempest Plugins (29)
- CLI tests for clients (16)
- Other functional/integration tests

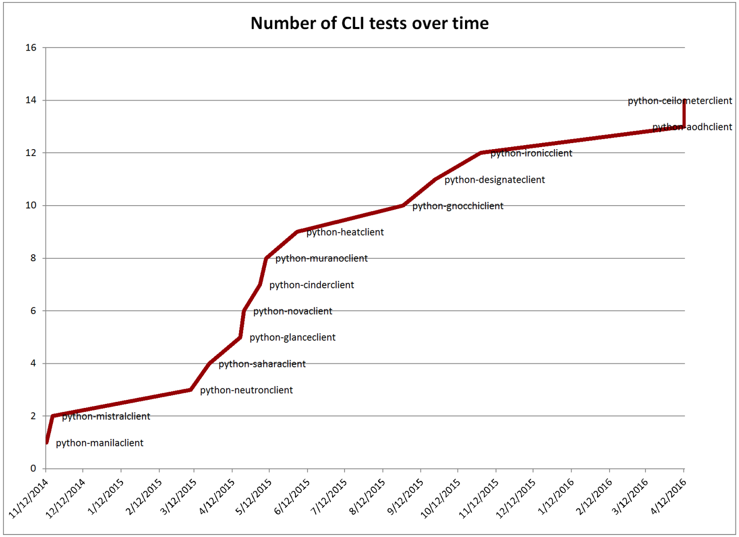
- Not executed against tempest
- Should only use tempest stable interfaces

Tempest plugins 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

- Tempest Plugins (WIP)
  - 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)

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 Internal APIs planned to become Stable

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

Which interfaces do you need to implement your 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

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/cgit/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 your CLI tests?

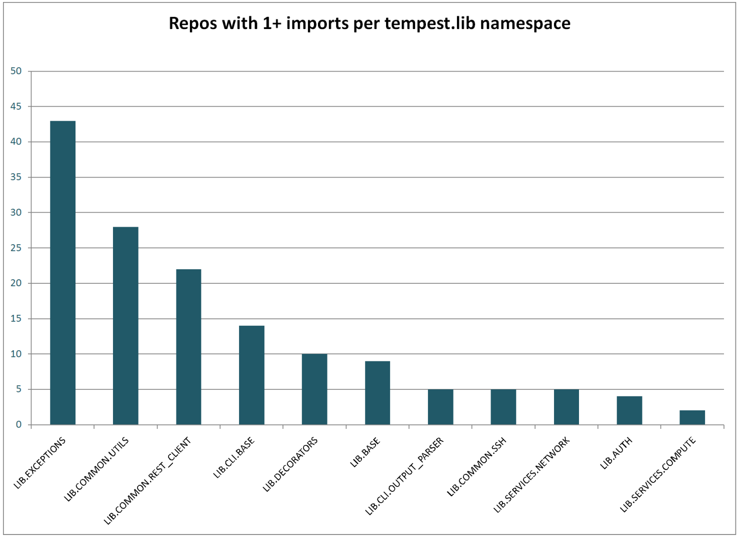
CLI Tests Base Class

- TBD

Output Parser

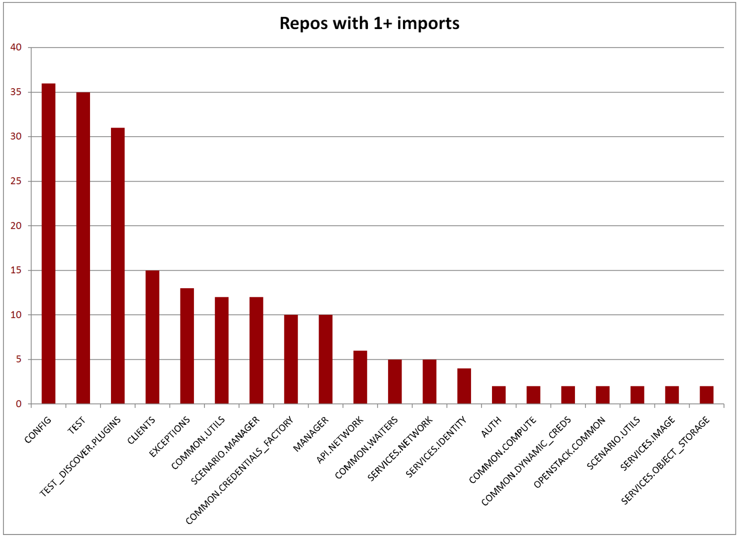
- TBD

Tempest Stable APIs



Source: codesearch.openstack.org

Tempest Internal APIs



Source: codesearch.openstack.org

Questions?