Tempest Scenario Tests
An Introduction for Testing your OpenStack cloud

OpenStack Summit May 2014 Atlanta
Monday, May-12 3:40pm-4:20pm

Masayuki Igawa
NEC Solution Innovators, Ltd.
Agenda

- Who am I?
- Overview of OpenStack
  - Operator’s wish
- What is Tempest?
  - Purpose and use cases of Tempest
  - Key points of Tempest
- What is ‘Scenario test’?
  - Overview of the current Scenario tests
- How to use Scenario Tests
- Demonstration
- Issue/Future of Scenario tests
- Summary
- Q&A
Who am I?

- OpenStack Active Technical Contributor
  - Since the Grizzly release

- Tempest core
  - Proposed **Scenario Tests** at Havana Summit

- Software Design & Production Specialist & Project Manager
  - At NEC Solution Innovators, Ltd.
Overview of OpenStack
Overview of OpenStack

- Most popular OSS Cloud infrastructure Software in the world!
- Consists of several loosely-coupled components
- Many features are being evolved with six month release cycle

Verifying multiple components is one of the greatest concerns from developer’s and operator’s viewpoint.
Operator’s wish

They want to verify their cloud works well **easily in a short time**

- **Why?**
  - To avoid any regression
  - To compare the stability of different software version

- **When?**
  - During/after setting up
  - After adding compute/controller nodes
  - Minor software update
  - Minor bug fix
  - Periodically

Do you want to verify?
... How?

How do you ensure the stability?
We need tests for it!
Do you know "Tempest"?

http://en.wikipedia.org/wiki/The_Tempest
What is Tempest?
What is Tempest? (1/2)

Tempest is one of the projects of OpenStack Programs

OpenStack Programs
- Compute (Nova)
- Object Storage (Swift)
- Image Service (Glance)
- Identity (Keystone)
- Dashboard (Horizon)
- Networking (Neutron)
- Block Storage (Cinder)
- Telemetry (Ceilometer)
- Orchestration (Heat)
- Database Service (Trove)
- Bare metal (Ironic)
- Queue service (Marconi)
- Data processing (Sahara)
- Key management (Barbican)
- Common Libraries (Oslo)
- Infrastructure
- Documentation

- **Quality Assurance (QA)**
  - Deployment (TripleO)
  - Devstack (DevStack)
  - Release cycle management

Projects
- **Tempest**
  - A set of integration tests to be run against a live OpenStack cluster.
- Grenade
  - Grenade is a test harness to exercise the OpenStack upgrade process between releases.

https://wiki.openstack.org/wiki/Programs
Tempest runs in the Zuul at every patch codes. We need to have Tempest code for integrated projects to ensure their validity.
Purpose and use cases of Tempest (1/2)

For Developers

1. New code can be tested to check the expected behavior
2. To verify whether new code introduce any regression.
3. On each new patch, Tempest runs in Zuul to make sure it qualify the expected quality/stability.

https://wiki.openstack.org/wiki/Gerrit_Workflow#Gerrit_Workflow_Quick_Reference
Purpose and use cases of Tempest (2/2)

For Operators
1. Checking their cloud works correctly → test suite for production environments
2. To check/avoid regression while software upgrade etc.
3. Can compare the stability of different software versions

RefStack/DefCore
- RefStack uses Tempest as a verifying tool set.

What is Refstack?
- An existence proof of the certified openstack APIs.
- A reference OpenStack environment for tools developers.
- A certification process for OpenStack service and product vendors.

http://refstack.org/
Implementing of Tempest tests is one of the minimal graduation requirements.

http://git.openstack.org/cgit/openstack/governance/tree/reference/incubation-integration-requirements

```
= Graduation to integrated =

* QA
** Project must have a devstack-gate job running. This gate job should install the project using devstack and then run tempest tests. This job should run and vote in the check and gate pipelines for the project. It is *not* required that this job is running for the projects it depends on. This demonstrates that it would be easy to add the project to the integrated gate after graduation.
```
There are 12 services in Tempest now.

Code names:
- Ironic (EC2 compatible)
- Nova
- Sahara
- Trove
- Keystone
- Glance
- Neutron
- Swift
- Heat
- Marconi
- Ceilometer
- Cinder

http://git.openstack.org/cgit/openstack/tempest/tree/tempest/services
Key points of Tempest (3/5)

Growth of the Tempest code

Line of Tempest code

LoC (Python only)

<table>
<thead>
<tr>
<th>Line</th>
<th>Loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diablo</td>
<td>6745</td>
</tr>
<tr>
<td>Essex</td>
<td></td>
</tr>
<tr>
<td>Folsom</td>
<td></td>
</tr>
<tr>
<td>Grizzly</td>
<td></td>
</tr>
<tr>
<td>Havana</td>
<td></td>
</tr>
<tr>
<td>Icehouse</td>
<td>77602</td>
</tr>
</tbody>
</table>

10 times more than Diablo!
### Tempest directories

<table>
<thead>
<tr>
<th>Category</th>
<th>Subdirectories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>api</td>
</tr>
<tr>
<td></td>
<td>stress</td>
</tr>
<tr>
<td></td>
<td>cli</td>
</tr>
<tr>
<td></td>
<td>thirdparty</td>
</tr>
<tr>
<td></td>
<td>scenario</td>
</tr>
<tr>
<td></td>
<td>tests</td>
</tr>
</tbody>
</table>

Tempest source code directories are separated by its category.
### Characteristics of Tempest tests

<table>
<thead>
<tr>
<th>Category (type)</th>
<th>Summary</th>
<th>Scope</th>
<th>Access client</th>
</tr>
</thead>
<tbody>
<tr>
<td>api</td>
<td>Functional tests for OpenStack APIs</td>
<td>single component</td>
<td>Tempest original</td>
</tr>
<tr>
<td>cli</td>
<td>Tests for OpenStack Official command line interface tools</td>
<td>single component</td>
<td>Official clients</td>
</tr>
<tr>
<td>stress</td>
<td>Stress tests</td>
<td>single component</td>
<td>Tempest original</td>
</tr>
<tr>
<td>thirdparty</td>
<td>Tests for non native OpenStack APIs such as EC2 API of Nova</td>
<td>single component</td>
<td>Tempest original</td>
</tr>
<tr>
<td>tests</td>
<td>Unit tests for Tempest</td>
<td>single component</td>
<td>---</td>
</tr>
<tr>
<td><strong>scenario</strong></td>
<td><strong>Through path Tests for between multiple OpenStack services</strong></td>
<td><strong>multiple components</strong></td>
<td><strong>Official clients</strong></td>
</tr>
</tbody>
</table>

Scenario tests are “system tests” for verifying between multiple OpenStack components.
What is ‘Scenario test’?
What is ‘Scenario test’? (1/3)

Key points of Scenario tests

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

Typical Scenario

1. create a flavor
2. create a image
3. create a network
4. create & configure a project, a quota, a role, a user
5. create a keypair
6. boot a instance
7. list & show the instance
8. create a volume
9. list & show the volume
10. attach the volume

Across the multiple components & sequential testing
What is ‘Scenario test’? (2/3)

Key points of Scenario tests

- Scenario tests should use **the official python client** libraries
- Tests should be **tagged with which services they exercise**, as determined by which client libraries are used directly by the test.

```python
@test.services('compute', 'volume', 'image', 'network')
def test_minimum_basics_scenario(self):
    self.glance_image_create()
    self.nova_keypair_add()
    self.nova_boot()
    self.nova_list()
    self.nova_show()
    self.cinder_create()
    self.cinder_list()
    self.cinder_show()
    self.nova_volume_attach()
```
The goal of the scenario tests

- For operators
  - Operators can use scenario tests for validating their cloud with simple process such as one command or one click.

- For developers
  - They can check whether new code will cause any regression on the other components easily.
    - Because scenario tests are comprehensive tests.
    - Example: Grenade[1] is using them now. As a result, we have been able to verify the OpenStack upgrade process in a short time.

Grenade: Ran 370 tests in **10mins**
Full test: Ran 2313 tests in **45mins**

## Overview of the current Scenario tests (1/2)

There are 24 scenarios but there are some missing services such as Telemetry, Database

<table>
<thead>
<tr>
<th>No.</th>
<th>file</th>
<th>test case</th>
<th>summary</th>
<th>services</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>orchestration/test_autoscaling.py</td>
<td>test_scale_up_then_down()</td>
<td>Scale up then down with heat</td>
<td>orchestration, compute</td>
</tr>
<tr>
<td>2</td>
<td>test_aggregates_basic_ops.py</td>
<td>test_aggregate_basic_ops()</td>
<td>Aggregates CRUD tests</td>
<td>compute</td>
</tr>
<tr>
<td>3</td>
<td>test_baremetal_basic_ops.py</td>
<td>test_baremetal_server_ops()</td>
<td>Baremetal basic ops</td>
<td>baremetal, compute, image, network</td>
</tr>
<tr>
<td>4</td>
<td>test_dashboard_basic_ops.py</td>
<td>test_basic_scenario()</td>
<td>Login to Horizon as a regular user and check the home page</td>
<td>dashboard</td>
</tr>
<tr>
<td>5</td>
<td>test_large_ops.py</td>
<td>test_large_ops_scenario()</td>
<td>Boot multiple instances in one nova call</td>
<td>compute, image</td>
</tr>
<tr>
<td>6</td>
<td>test_load_balancer_basic.py</td>
<td>test_load_balancer_basic()</td>
<td>Checking basic load balancing</td>
<td>compute, network</td>
</tr>
<tr>
<td>7</td>
<td>test_minimum_basic.py</td>
<td>test_minimum_basic_scenario()</td>
<td>Basic image, instance, volume, network CRUD test</td>
<td>compute, volume, image, network</td>
</tr>
<tr>
<td>8</td>
<td>test_network_advanced_server_ops.py</td>
<td>test_server_connectivity_stop_start()</td>
<td>This test case checks VM connectivity after some advanced instance operations executed * Stop/Start an instance</td>
<td>compute, network</td>
</tr>
<tr>
<td>9</td>
<td>test_network_advanced_server_ops.py</td>
<td>test_server_connectivity_reboot()</td>
<td>This test case checks VM connectivity after some advanced instance operations executed * Reboot an instance</td>
<td>compute, network</td>
</tr>
<tr>
<td>10</td>
<td>test_network_advanced_server_ops.py</td>
<td>test_server_connectivity_rebuild()</td>
<td>This test case checks VM connectivity after some advanced instance operations executed * Rebuild an instance</td>
<td>compute, network</td>
</tr>
<tr>
<td>11</td>
<td>test_network_advanced_server_ops.py</td>
<td>test_server_connectivity_pause_unpause()</td>
<td>This test case checks VM connectivity after some advanced instance operations executed * Pause/Unpause an instance</td>
<td>compute, network</td>
</tr>
<tr>
<td>12</td>
<td>test_network_advanced_server_ops.py</td>
<td>test_server_connectivity_suspend_resume()</td>
<td>This test case checks VM connectivity after some advanced instance operations executed * Suspend/Resume an instance</td>
<td>compute, network</td>
</tr>
<tr>
<td>13</td>
<td>test_network_advanced_server_ops.py</td>
<td>test_server_connectivity_resize()</td>
<td>This test case checks VM connectivity after some advanced instance operations executed * Resize an instance</td>
<td>compute, network</td>
</tr>
</tbody>
</table>
Overview of the current Scenario tests (2/2)

There are 24 scenarios but there are some missing services such as Telemetry, Database

<table>
<thead>
<tr>
<th>No.</th>
<th>file</th>
<th>test case</th>
<th>summary</th>
<th>services</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>test_network_basic_ops.py</td>
<td>test_network_basic_ops()</td>
<td>This smoke test suite assumes that Nova has been configured to boot VM's with Neutron-managed networking, and attempts to verify network connectivity</td>
<td>compute, network</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>test_hotplug_nic()</td>
<td>Checking hotplug a nic to a VM</td>
<td>compute, network</td>
</tr>
<tr>
<td>16</td>
<td>test_security_groups_basic_ops.py</td>
<td>test_cross_tenant_traffic()</td>
<td>This test suite assumes that Nova has been configured to boot VM's with Neutron-managed networking, and attempts to verify cross tenant connectivity</td>
<td>compute, network</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>test_in_tenant_traffic()</td>
<td>Test for in-tenant check</td>
<td>compute, network</td>
</tr>
<tr>
<td>18</td>
<td>test_server_advanced_ops.py</td>
<td>test_resize_server_confirm()</td>
<td>This test case stresses some advanced server instance operations * Resizing an instance</td>
<td>compute</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>test_server_sequence_suspend_resume()</td>
<td>This test case stresses some advanced server instance operations * Sequence suspend resume</td>
<td>compute</td>
</tr>
<tr>
<td>20</td>
<td>test_server_basic_ops.py</td>
<td>test_server_basicops()</td>
<td>This smoke test case follows this basic set of operations</td>
<td>compute, network</td>
</tr>
<tr>
<td>21</td>
<td>test_snapshot_pattern.py</td>
<td>test_snapshot_pattern()</td>
<td>This test is for snapshotting an instance and booting with it.</td>
<td>compute, network, image</td>
</tr>
<tr>
<td>22</td>
<td>test_stamp_pattern.py</td>
<td>test_stamp_pattern()</td>
<td>This test is for snapshotting an instance/volume and attaching the volume created from snapshot to the instance booted from snapshot.</td>
<td>compute, volume, image, network</td>
</tr>
<tr>
<td>23</td>
<td>test_swift_basic_ops.py</td>
<td>test_swift_basic_ops()</td>
<td>Swift basic operations test</td>
<td>object_storage</td>
</tr>
<tr>
<td>24</td>
<td>test_volume_boot_pattern.py</td>
<td>test_volume_boot_pattern()</td>
<td>This is a test for checking volume boot sequence.</td>
<td>compute, volume, image</td>
</tr>
</tbody>
</table>
How to use Scenario Tests
How to use Scenario tests (1/3)

1. clone the source code
   
   ```
   $ git clone git://git.openstack.org/openstack/tempest
   ```

2. copy & customize the configuration file (tempest.conf)
   
   ```
   $ vim tempest.conf
   ```

3. run Tempest
   
   ```
   $ testr run --parallel tempest.scenario
   ```
   or
   
   ```
   $ ./run_tempest.sh tempest.scenario
   ```

4. check the result

Requirements

- command line operation skill
- knowledge of tempest.conf
  - There are about 300 config options..

These execute all scenario tests. If you want to execute a specific test case, you can specify the test like this:

   ```
   ..... tempest.scenario.test_minimum_basic
   ```
How to use Scenario tests (2/3) – Check the result

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_server_sequence_suspendolume[compute]</td>
<td>OK</td>
<td>32.67</td>
</tr>
<tr>
<td>test_hotplug_nic[compute,gate,network,smoke]</td>
<td>FAIL</td>
<td></td>
</tr>
<tr>
<td>test_snapshot_pattern[compute,image,network]</td>
<td>OK</td>
<td>307.60</td>
</tr>
<tr>
<td>test_server_connectivity_pause_unpause[compute,network]</td>
<td>OK</td>
<td>187.68</td>
</tr>
<tr>
<td>test_minimum_basic[compute,image,network,volume]</td>
<td>OK</td>
<td>287.11</td>
</tr>
<tr>
<td>test_security_groups_basic_ops[compute,gate,network,smoke]</td>
<td>OK</td>
<td>314.48</td>
</tr>
<tr>
<td>test_cross_tenant_traffic[compute,gate,network,smoke]</td>
<td>OK</td>
<td>183.36</td>
</tr>
<tr>
<td>test_in_tenant_traffic[compute,gate,network,smoke]</td>
<td>OK</td>
<td>213.48</td>
</tr>
<tr>
<td>test_stomp_pattern[compute,image,network,volume]</td>
<td>SKIP</td>
<td>0.00</td>
</tr>
<tr>
<td>test_volume_boot_pattern[compute,image,network]</td>
<td>OK</td>
<td>232.72</td>
</tr>
<tr>
<td>test_volume_boot_patternV2[compute,image,network]</td>
<td>OK</td>
<td>389.60</td>
</tr>
<tr>
<td>tearDownClass (tempest.scenario.test_volume_boot_pattern)</td>
<td>FAIL</td>
<td></td>
</tr>
<tr>
<td>test_server_connectivity_rebuild[compute,network]</td>
<td>OK</td>
<td>258.70</td>
</tr>
<tr>
<td>test_server_connectivity_resize[compute,network]</td>
<td>OK</td>
<td>258.70</td>
</tr>
<tr>
<td>test_server_connectivity_stop_start[compute,network]</td>
<td>OK</td>
<td>91.37</td>
</tr>
<tr>
<td>test_server_connectivity_suspend_resume[compute,network]</td>
<td>OK</td>
<td>88.69</td>
</tr>
</tbody>
</table>

Slowest 10 tests took 2593.52 secs:
- tempest.scenario.test_minimum_basic[compute,network]
These are details of the failure. You may need to check these and also server's log.
Demonstration
Demonstration

- Demo environment
  - DevStack (May 6)
- Execute a scenario test case
  - No.7 test_minimum_basic.py:
    - Basic image, instance, volume, network CRUD test
- Check the result

Let’s see the demo.

A portion of the scenario test code

```python
@test.services('compute', 'volume', 'image', 'network')
def test_minimum_basic_scenario(self):
    self.glance_image_create()
    self.nova_keypair_add()
    self.nova_boot()
    self.nova_list()
    self.nova_show()
    self.cinder_create()
    self.cinder_list()
    self.cinder_show()
    self.nova_volume_attach()
    self.addCleanup(self.nova_volume_detach)
    self.cinder_show()
    self.nova_volume_detach()
    self.nova_keypair_delete()
    self.cinder_delete()
    self.nova_delete()
    self.check_partitions()
```

https://github.com/openstack/tempest/blob/master/tempest/scenario/test_minimum_basic.py
Issue/Future of Scenario tests
Issues and Future of scenario tests

Issues

- **Need more scenarios**
  - More services such as Telemetry(Ceilometer), Database(Trove)
  - More complicated but popular scenarios
- It’s **difficult to write scenarios** because it needs python development skills.
- It’s **difficult to prepare settings for existing clouds**.
  - ‘tempest.conf’ has about 300 configuration items.
- It’s **difficult to analyze** the cause of failures.
  - ‘Elastic Recheck’ is one of the solutions for developers but not for operators…

Future

- Easier
  - Operators want to verify their cloud without command line skills.
    - Tempest GUI!
- More useful
  - We can specified nodes and zones by settings.
    - Under reviewing the patch now: https://review.openstack.org/#/c/66882/
Summary
Summary

OpenStack has an official test suite: **Tempest**
This is **not for only developers but also for operators** of OpenStack cloud.
Especially, **scenario tests** are test cases across multiple components, it can be used for system testing for OpenStack cloud.
By using the scenario tests, it is possible to reduce the evaluation cost of your cloud environment in comparison with making a test suite from scratch.

Please join us to enhance scenario tests!
- OpenStack Development Mailing List (not for usage questions)
  - [openstack-dev@lists.openstack.org](mailto:openstack-dev@lists.openstack.org)
  - Please add ‘[qa]’ tag to the subject for QA things.
- IRC
  - #openstack-qa channel on Freenode
Q&A
Appendix
Links

- OpenStack Development Mailing List (not for usage questions)
  - openstack-dev@lists.openstack.org

- Tempest scenario test code

- Gerrit workflow
  - https://wiki.openstack.org/wiki/Gerrit_Workflow

- How To Contribute to OpenStack
  - https://wiki.openstack.org/wiki/How_To_Contribute
Overview of Tempest GUI (Now Proposing)

- Horizon
  - python-tempestclient

- Tempest Server
  - REST APIs for List, Show, Run... Tempest tests and results

- OpenStack Cloud
  - Execute Tempest tests

- Results Store
  - New modules: python-tempestclient
  - Tempest Server
  - Results Store
How to add more scenarios

Requirements
- python development skill
- OpenStack python library knowledge

Design a scenario
- What do you want to verify?

Write the code
- Before writing the code, I recommend you look at the code of existing scenario tests.
  - You will get the knowledge of how to implement scenarios.

Why Contribute the code to the OpenStack community?
- It increases OpenStack stability.