Concurrent Aquari Storage Test Plan

Document History

Version	Revision Date	Description
0.1	DD-MM-YYYY	Initial Version

1 Test strategy

We have qualified Aquari Storage ,as a Mirantis PopenStack (MOS) backend storage, using Fuel Ceph plugin. Functional, performance and resiliency testing was performed to ensure Aquari Storage is functioning properly.

Functional tests include verification of storage for Glance, Cinder and Nova. Performance testing benchmarked Aquari Storage throughput.

Resiliency testing evaluates the reliability and fault tolerance of the Aquari Storage,.

2 Functional Testing

2.1 Health Check

Mirantis OpenStack (MOS) has built-in system called Health Check for quick, high-level verification that a cloud is working properly. Health Check tests the basic operations of the OpenStack API, validates HA functionality (checks MySQL and RabbitMQ cluster status) and performs other critical functions. A list of the available tests can be found in Appendix 5.1 and on github.

2.1.1 How to use Health Checks

There are two ways to launch the Health Checks:

- From the Fuel Master Web UI (Health Check tab).
- From the Fuel Master node CLI by running the <*fuel --env end_id health* --check smoke,ha,etc> command (the OSTF service runs inside the Docker container).

Please note that the logs are stored in the /var/log/ostf/ folder of the Fuel Master node.

2.2 Tempest

The Openstack community has developed two instruments for functional tests – Tempest and Rally. Tempest is an original test framework aimed to cover all possible API functionality available in Openstack components or added by third-party modules/drivers/plugins. A list of Tempest tests is in Appendix 5.2. Mirantis recommends using MOS-Tempest-runner to easily launch this tool. All Tempest tests will be run except for a small number of tests known as 'should-fail' for a particular MOS version. Tempest also covers negative scenarios such as authorization violation or parameter mismatches.

2.2.1 Tips on Using Tempest

- WARNING: It is potentially dangerous to use the mos-tempest-runner script to run Tempest against PRODUCTION OpenStack clouds. Doing so may break the cluster under test in ways

that are difficult to diagnose and repair. Please note that mos-tempest-runner was initially designed to run Tempest on CI and test OpenStack environments, only.

- Mos-tempest-runner must be run on the Fuel master node only. It is best to run it from a virtualenv.
- The latest oslo.log version supported by Tempest may not be the same as the version installed on the cluster under test. Verify it with <pip search oslo.log> and install the appropriate version of oslo.log on the cluster if required.
- Tempest should be checked out to the appropriate branch.
- To authenticate with the cluster, please copy the openrc file from the controllers to the folder on the Fuel Master node, containing the script.
- The Fuel master node must have an Internet connection to run Tempest.
- The script tempest_config_creator.sh should be created and should receive 755 permissions.
- Edit tempest.conf according to your cloud settings. Example of such tempest config file could be found in Appendix 5.7
- <run_tests --smoke> will launch only the smoke test group. It's also possible to run single test as described in the Readme file.
- Tempest tests can be launched from the Rally framework, using the <rally verify> command. This may be the most convenient way of doing so.
- WARNING: After testing, use the cleanup_cloud.sh script to clean up the OpenStack cloud (the script removes user "demo," tenant "demo," roles, flavors, uploaded CirrOS image and reverts Keystone endpoints), reverting the cloud to its initial state. Note that this script does not clean up resources created by Tempest, but only by mos-tempest-runner.

2.3 Rally

Rally is used to test API functionality and performance. Rally is a benchmarking tool widely used across the Openstack ecosystem. A list of available Rally tests is in Appendix 5.3

2.3.1 Tips on Using Rally

There are many ways to install and use Rally. Here, we provide instructions on how to install Rally on the compute node, using Docker:

- Extract the Rally archive on the fuel-master node

- Edit rally_conf/config.json for your environment settings. An example of such a tempest config file may be found in Appendix 5.8
- Run ./rally_setup.sh compute_node_address
- ssh to the selected compute node
- Change user to rally <su rally>
- Use rally_docker alias to start the container with rally
- Once inside container, run conf/create_deployment.sh
- Run Rally benchmarks using <rally task start scenarios/nova.yaml> (or copy any other task from the official repository)

2.4 Test cases

2.4.1 MOS Integration

Purpose: validate Aquari integration with MOS
Tasks:

- Install Fuel Ceph Plugin
- Provision 3 pools for Glance, Cinder and Nova in Aquari UI
- Configure Fuel Ceph Plugin to use newly provisioned Aquari Storage pools

2.4.2 Validate storage for Glance

Purpose: validate storage for OS images
Tasks:

- Upload various OS images to Glance
- Verify that images can be restored

2.4.3 Validate storage for Nova

Purpose: validate storage for instances
Tasks:

- Provision multiple instances
- Verify that instances work (star, run, suspend) properly

2.4.4 Validate Cinder storage

Purpose: validate Cinder storage backend
Tasks:

- Provision multiple volumes
- Attach volumes to instances
- Verify that instances can access volumes

2.4.5 Teptest testing

Purpose: validate genral MOS functionality
Tasks: execute selected temptest scripts.

3 Non-Functional Testing

3.1 Aquari Storage performance testing

Purpose: benchmark Aquari Storage performance for Sequential Read, Random Read and Sequential Write workloads on S3 pools using large block size IO

4 Resiliency testing

Aquari Storage is based on Ceph, highly resilient software defined storage. The purpose of this testing is to ensure MOS cloud availability in the case of disk's or entire node's failure. Aquari Storage supports Replicated and Erasure Coded (EC) pools.