Test Plan for SwiftStack Fuel Plugin

Release 0.3-0.3.0-1

SwiftStack Inc.

Jun 17, 2016

Contents

1 Revision history

Version	Revision Date	Editor	Comment
1.0	27.08.2015	Charles Hsu(chsu@swiftstack.com)	Created
2.0	06.07.2016	Charles Hsu(chsu@swiftstack.com)	Revised for Fuel 8.0
2.1	06.15.2016	Charles Hsu(chsu@swiftstack.com)	Update contents and rewrite in RST

2 SwiftStack Fuel Plugin

Allow Mirantis OpenStack environment able to use an existing Swift cluster managed by SwiftStack Controller. SwiftStack plugin will disable the swift cluster deployed in the nodes are role **Controller** or **Primary-controller**. And then reconfigures API endpoints, keystone and glance settings to point an existing SwiftStack Swift cluster.

Here are two basic Fuel OpenStack environments as follows:

2.1 Use SwiftStack On-Premises Controller

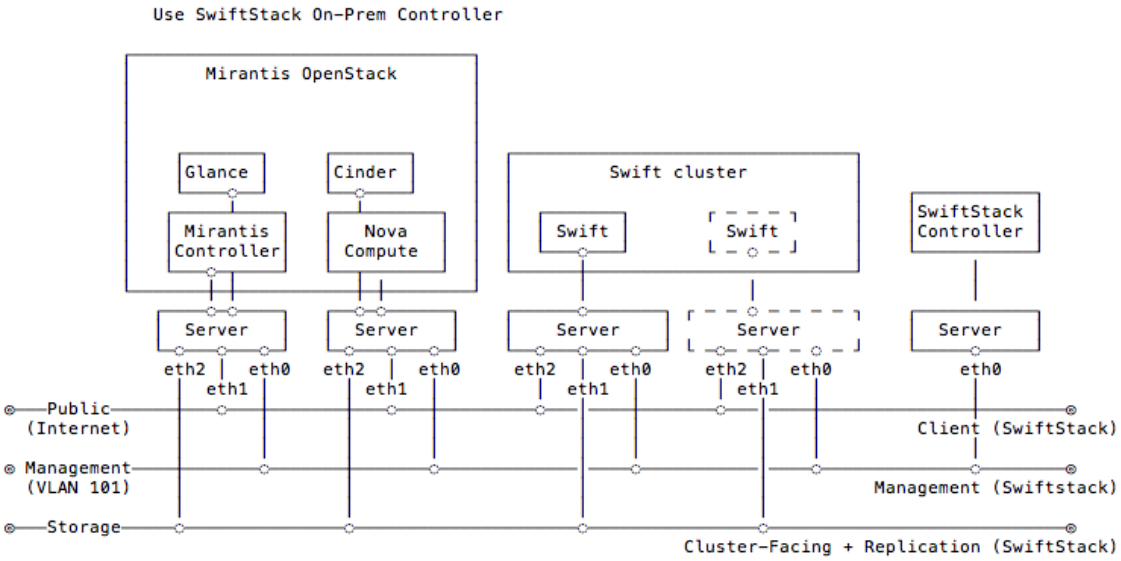
Please setup an On-Premises SwiftStack controller first, and then setup a single node Swift cluster with SwiftStack controller, here is our quick start guide.

- 1 SwiftStack On-Premises controller
- 1 Swift cluster (single node)

Also prepare a Fuel environment using Slave nodes according to the Fuel Install Guide.

Note: In this diagram, the Swift cluster is also connected to Fuel Storage network for SwiftStack cluster-facing and data replication network, if you have performance concern, please consider to separate Swift cluster-facing and data replication network out of Fuel networks. That prevents network starvation on Fuel Storage network when Swift service daemons are moving data or clients upload large data into the Swift cluster.

Also, SwiftStack Nodes need to communicate with the On-Premises controller over Fuel Management network, so please make sure the On-Premises controller also connected to Fuel Management network. You can run a CLI command `ssdiag` on SwiftStack nodes to check the connectivity between SwiftStack Nodes and Controller.



2.2 Use SwiftStack Public Controller (Platform)

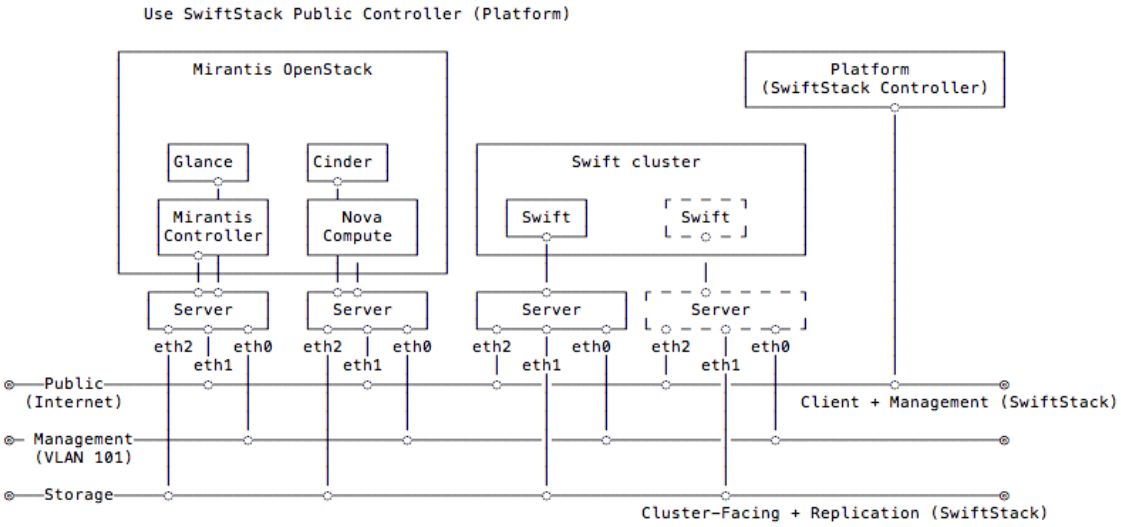
Please setup a single node Swift cluster with our public controller, here is our quick start guide.

- 1 Swift cluster (single node)

Also prepare a Fuel environment using Slave nodes according to the Fuel Install Guide.

Note: In this diagram, the Swift cluster is also connected to Fuel Storage network for SwiftStack cluster-facing and data replication network, if you have performance concern, please consider to separate Swift cluster-facing and data replication network out of Fuel networks. That prevents network starvation on Fuel Storage network when Swift service daemons are moving data or clients upload large data into the Swift cluster.

Also, SwiftStack Nodes need to communicate with SwiftStack Public controller over Fuel Public network, so please make sure SwiftStack Nodes able to reach Internet.



3 Developer's specification

See developers specification in this repository: https://github.com/openstack/fuel-plugin-swiftstack

4 Limitations

The plugin only supports a running SwiftStack Swift cluster and it able to reach from the OpenStack environment. Make sure you have the correct network configuration for the Swift cluster and Mirantis OpenStack environment before you enable this plugin.

5 Test strategy

5.1 Types of tests included

- Check Swift API endpoints
- Basic Swift APIs operations
- System Tests

5.2 Types of tests not included

- Swift cluster operations

5.3 Acceptance criteria

These tests should be pass:

- Check Swift API endpoints
- Basic Swift APIs operation

- System Tests

5.4 Test environment and infrastructure

Before install the plugin to your fuel environment, make sure you have a running SwiftStack Swift cluster managed by SwiftStack Platform controller or On-Premises controller. You'll need to configure few middlewares from the controller and push configuration to the SwiftStack cluster.

- A running SwiftStack Swift cluster

5.5 Product compatibility matrix

Issue	Version
Mirantis OpenStack	8.0

6 Check Swift API endpoints

- Swift API endpoint should be like http[s]://<SWIFT_API_IP_or_HOSTNAME>:[80|443]/v1/KEY_%(tenant_id)s
  - Disable TLS with Swift endpoint
    - http://<SWIFT_API_IP_or_HOSTNAME>:80/v1/KEY_%(tenant_id)s
  - Enable TLS with Swift endpoint
    - https://<SWIFT_API_IP_or_HOSTNAME>:443/v1/KEY_%(tenant_id)s

7 Basic Swift APIs operation

1. Verify Swift account with Keystone authentication.

```
# Login to one of nodes of Swift cluster.

# Test admin account
root@node-23:~$ cat rc.admin
export ST_AUTH=http://192.168.0.2:5000/v2.0
export ST_USER=admin:admin
export ST_KEY=admin
export ST_AUTH_VERSION=2

~$ source rc.admin

~$ swift stat
        Account: KEY_8408a5a799364d06b81542a8017e7975
     Containers: 0
        Objects: 0
          Bytes: 0
X-Put-Timestamp: 1465289155.00899
    X-Timestamp: 1465289155.00899
     X-Trans-Id: tx1d35d328e5294c88baf5e-00575689c2
   Content-Type: text/plain; charset=utf-8
```

2. Upload/download object and check md5sum

```
~$ swift upload test rc.admin
rc.admin

~$ swift stat test rc.admin
       Account: KEY_8408a5a799364d06b81542a8017e7975
     Container: test
        Object: rc.admin
  Content Type: application/octet-stream
Content Length: 117
 Last Modified: Tue, 07 Jun 2016 08:46:16 GMT
          ETag: 4a97d36410af1b380fe5b014a6cd8db5
    Meta Mtime: 1465288847.821181
 Accept-Ranges: bytes
   X-Timestamp: 1465289175.47789
    X-Trans-Id: tx13823ad38c084e529c20d-00575689de

~$ swift download test rc.admin -o rc.admin.download
rc.admin [auth 1.171s, headers 1.452s, total 1.460s, 0.000 MB/s]

~$ md5sum rc.admin*
4a97d36410af1b380fe5b014a6cd8db5  rc.admin
4a97d36410af1b380fe5b014a6cd8db5  rc.admin.download
```

8 System testing

8.1 Install plugin and deploy environment

Test Case ID	install_plugin_deploy_env
Steps	1. Copy the plugin to the Fuel Master node 2. Install the plugin. 3. Ensure that plugin is installed successfully with running `fuel plugins --list` command in the Fuel CLI. 4. Create an environment with enabled plugin in the Fuel Web UI. 5. Add 3 nodes with Controller role and 1 node with Compute and Cinder role. 6. Finalize environment configuration (e.g. networking, nodes interfaces). 7. Run network verification check. 8. Deploy the cluster. 9. Run OSTF.
Expected Result	1. Plugin is installed successfully at the Fuel Master node and the corresponding output appears in the CLI. 2. Cluster is created and network verification check is passed. 3. OSTF tests (Health Checks) are passed. 4. Environment is deployed successfully.

8.2 Install plugin and deploy environment with Swift API hostname

Test Case ID	install_plugin_deploy_env_with_swift_hostname
Steps	1. Copy the plugin to the Fuel Master node 2. Install the plugin. 3. Ensure that plugin is installed successfully with running `fuel plugins --list` command in the Fuel CLI. 4. Create an environment with enabled plugin in the Fuel Web UI. 5. Add 3 nodes with Controller role and 1 node with Compute and Cinder role. 6. Finalize environment configuration (e.g. networking, nodes interfaces). 7. Run network verification check. 8. Enabled Swift API hostname in the plugin section 9. Deploy the cluster. 10. Run OSTF.
Expected Result	1. Plugin is installed successfully at the Fuel Master node and the corresponding output appears in the CLI. 2. Cluster is created and network verification check is passed. 3. OSTF tests (Health Checks) are passed. 4. Environment is deployed successfully.

8.3 Modifying env with enabled plugin (removing/adding controller nodes)

Test Case ID	modify_env_with_plugin_remove_add_controller
Steps	1. Copy the plugin to the Fuel Master node 2. Install the plugin. 3. Ensure that plugin is installed successfully with running `fuel plugins --list` command in the Fuel CLI. 4. Create an environment with enabled plugin in the Fuel Web UI. 5. Add 3 nodes with Controller role and 1 node with Compute and Cinder role. 6. Finalize environment configuration (e.g. networking, nodes interfaces). 7. Enable the plugin and configure it following the instructions from the Plugin Guide. 8. Run network verification check. 9. Deploy the cluster. 10. Run OSTF. 11. Remove 1 node with Controller role 12. Re-deploy the cluster. 13. Run OSTF. 14. Add 1 new node with Controller role. 15. Re-deploy the cluster. 16. Run OSTF.
Expected Result	1. Plugin is installed successfully at the Fuel Master node and the corresponding output appears in the CLI. 2. Cluster is created and network verification check is passed. 3. Plugin is enabled and configured in the Fuel Web UI. 4. OSTF tests (Health Checks) are passed. 5. Environment is deployed successfully. 6. When adding/removing Controller node (where plugin-related services are run): (a) all plugins resources are migrated to another Controller node (b) the environment is redeployed successfully when adding/removing Controller node.

8.4 Modifying env with enabled plugin (removing/adding compute node)

Test Case ID	modify_env_with_plugin_remove_add_compute
Steps	1. Copy the plugin to the Fuel Master node 2. Install the plugin. 3. Ensure that plugin is installed successfully with running `fuel plugins --list` command in the Fuel CLI. 4. Create an environment with enabled plugin in the Fuel Web UI. 5. Add 3 nodes with Controller role and 1 node with Compute and Cinder role. 6. Finalize environment configuration 7. Enable the plugin and configure it following the instructions from the Plugin Guide. 8. Run network verification check. 9. Deploy the cluster. 10. Run OSTF. 11. Add 1 new node with Compute role. 12. Re-deploy the cluster. 13. Run OSTF. 14. Remove 1 node with Compute role 15. Re-deploy the cluster. 16. Run OSTF.
Expected Result	1. Plugin is installed successfully at the Fuel Master node and the corresponding output appears in the CLI. 2. Cluster is created and network verification check is passed. 3. Plugin is enabled and configured in the Fuel Web UI. 4. OSTF tests (Health Checks) are passed. 5. Environment is deployed successfully. 6. When adding/removing Compute node (where plugin-related services are run): (a) all plugins resources are migrated to another Compute node (b) the environment is re-deployed successfully when adding/removing Compute node

8.5 Fuel create mirror and update (setup) of core repos

Test Case ID	Fuel_create_mirror_update_core_repos
Steps	1. Copy the plugin to the Fuel Master node 2. Install the plugin. 3. Ensure that plugin is installed successfully with running `fuel plugins --list` command in the Fuel CLI. 4. Create an environment with enabled plugin in the Fuel Web UI. 5. Add 3 nodes with Controller role and 1 node with Compute and Cinder role. 6. Finalize environment configuration (e.g. networking, nodes interfaces). 7. Enable the plugin and configure it following the instructions from the Plugin Guide. 8. Run network verification check. 9. Deploy the cluster. 10. Run OSTF. 11. Go in cli through controller / compute / storage /etc nodes and get pid of services which were launched by plugin and store them. 12. Launch the following command on the Fuel Master node: fuel-createmirror -M 13. Launch the following command on the Fuel Master node: (a) For MOS < 8.0: fuel –env <ENV_ID> node –node-id <NODE_ID1> <NODE_ID2> <NODE_ID_N> –tasks upload_core_repos (b) For MOS 8.0: fuel –env <ENV_ID> node –node-id <NODE_ID1> <NODE_ID2> <NODE_ID_N> –tasks setup_repositories 14. Go to controller/plugin/storage node and check if plugin's services are alive and aren't changed their pid. 15. Check with fuel nodes command that all nodes are remain in ready status. 16. Rerun OSTF.
Expected Result	1. Plugin is installed successfully at the Fuel Master node and the corresponding output appears in the CLI. 2. Cluster is created and network verification check is passed. 3. Plugin is enabled and configured in the Fuel Web UI. 4. OSTF tests (Health Checks) are passed. 5. Environment is deployed successfully. 6. Plugin's services shouldn't be restarted after corresponding task was executed. If they are restarted as some exception, this information should be added to plugin's User Guide. 7. Cluster (nodes) should remain in ready state. 8. OSTF test should be passed on rerun.

8.6 Uninstall of plugin in the deployed environment

Test Case ID	uninstall_plugin_with_deployed_env
Steps	1. Copy the plugin to the Fuel Master node 2. Install the plugin. 3. Ensure that plugin is installed successfully with running `fuel plugins --list` command in the Fuel CLI. 4. Add 3 nodes with Controller role and 1 node with Compute and Cinder role. 5. Finalize environment configuration (e.g. networking, nodes interfaces). 6. Enable the plugin and configure it following the instructions from the Plugin Guide. 7. Run network verification check. 8. Deploy the cluster. 9. Run OSTF. 10. Uninstall the plugin with running `fuel plugins --remove <plugin-name>==<plugin_version> (e.g. 1.0.1)'` Ensure that the following output appears in CLI: "400 Client Error: Bad Request (Can't delete plugin which is enabled for some environment.)"
Expected Result	1. Plugin is installed successfully at the Fuel Master node and the corresponding output appears in the CLI. 2. Cluster is created and network verification check is passed. 3. Plugin is enabled and configured in the Fuel Web UI. 4. OSTF tests (Health Checks) are passed. 5. Environment is deployed successfully. 6. Alert is displayed when trying the uninstall the plugin.

8.7 Uninstall of plugin in the non-deployed environment

Test Case ID	uninstall_plugin
Steps	1. Copy the plugin to the Fuel Master node 2. Install the plugin. 3. Ensure that plugin is installed successfully with running `fuel plugins --list` command in the Fuel CLI. 4. Add 3 nodes with Controller role and 1 node with Compute and Cinder role. 5. Finalize environment configuration (e.g. networking, nodes interfaces). 6. Enable the plugin and configure it following the instructions from the Plugin Guide. 7. Run network verification check. 8. Delete listed environment 9. Uninstall the plugin with running `fuel plugins --remove <plugin-name>==<plugin_version>` (e.g. `1.0.1`) for more details. 10. Install the plugin.
Expected Result	1. Plugin is installed successfully at the Fuel Master node and the corresponding output appears in the CLI. 2. Cluster is created and network verification check is passed. 3. Plugin is enabled and configured in the Fuel Web UI. 4. When uninstalling the plugin, no plugin-related elements are left in the environment (e.g. UI elements disappear, Nailgun database is restored to the default state, no output for command `fuel plugins --list`).

8.8 Apply maintenance updates to deployed environment

Mirantis OpenStack features the ability to receive patches via the common flow called Maintenance Updates. Please reach out to Partner Enablement team about the latest Maintenance Updates portion to test against with you plugin.

Test Case ID	apply_mu
Steps	1. Copy the plugin to the Fuel Master node 2. Install the plugin. 3. Ensure that plugin is installed successfully with running `fuel plugins --list` command in the Fuel CLI. 4. Add 3 nodes with Controller role and 1 node with Compute and cinder role. 5. Finalize environment configuration (e.g. networking, nodes interfaces). 6. Enable the plugin and configure it following the instructions from the Plugin Guide. 7. Run network verification check. 8. Deploy the cluster. 9. Run OSTF. 10. Once environment is deployed, apply maintenance updates following the instructions. 11. Check is plugin services continue running. 12. Make sure all nodes are in ready state and no regression is observed. 13. Run OSTF checks.
Expected Result	1. Plugin is installed successfully at the Fuel Master node and the corresponding output appears in the CLI. 2. Cluster is created and network verification check is passed. 3. Plugin is enabled and configured in the Fuel Web UI. 4. OSTF tests (Health Checks) are passed. 5. Environment is deployed successfully. 6. Maintenance Updates do not affect running services related to the plugin (e.g. the services aren't restarted). 7. Cluster remains in the fully operational state after applying Maintenance Updates.

9 Appendix

#	Resource title