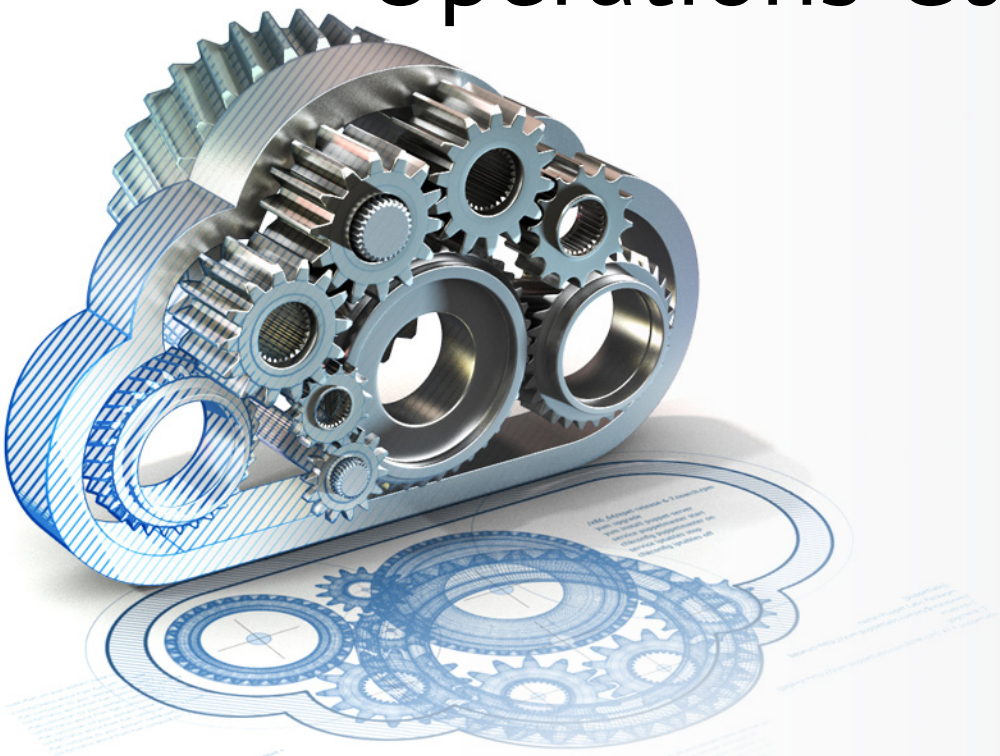
MIRANTIS

BUILT FOR openstack®

Mirantis OpenStack®

version 7.0

Operations Guide



Contents

Preface

This documentation provides information on how to use Fuel to deploy OpenStack environments. The information is for reference purposes and is subject to change.

Intended Audience

This documentation is intended for OpenStack administrators and developers; it assumes that you have experience with network and cloud concepts.

Documentation History

The following table lists the released revisions of this documentation:

Revision Date	Description
May, 2015	6.1 GA

Introduction

This is a collection of useful procedures for using and managing your Mirantis OpenStack environment. The information given here supplements the information in:

- OpenStack Admin User Guide
- OpenStack Cloud Administrator Guide

Accessing the shell on the nodes

Many maintenance and advanced configuration tasks require that you access the Fuel Master and target nodes at the shell level. All these systems run the **bash** shell and support standard Linux system commands. Each system has its own console that you can use directly but the standard practice is to use SSH to access the consoles of the different nodes.

Uploading Public Keys

You access the shell on the Fuel Master node and the target nodes using SSH, and you must define a Public Key to use SSH.

1. Generate a Public Key with the following command on your client:

   ```
   ssh-keygen -t rsa
   ```

2. Populate the *Public Key* field in the Fuel UI with this key. Fuel will upload this Public Key to each target node it deploys, but it does not upload it to nodes that are already deployed.

3. To upload the SSH key to the Fuel Master node or any deployed node, use the following command sequence:

   ```
   ssh-agent
   ssh-copy-id -i .ssh/id_rsa.pub root@<ip-addr>
   ```

   <*ip-addr*> is the IP address for the Fuel Master node, which is the same IP address you use to access the Fuel console.

   You can use this same command to add a public key to a deployed target node. See *SSH to target nodes* for information about getting the <*ip-addr*> values for the target nodes.

   You can instead add the content of your key (stored in the `.ssh/id_rsa.pub` file) to the node's `/root/.ssh/authorized_keys` file.

SSH to the Fuel Master Node

You can now use **ssh** to access the console of the Fuel Master Node or **scp** to securely copy a file to the Fuel Master node.

SSH to target nodes

You can SSH to any of the target nodes in your environment from the Fuel Master node.

Use the **fuel node list** command to get a list like the following:

```
id | status | name             | cluster | ip         | mac               | roles      | pe
---|--------|------------------|---------|------------|-------------------|------------|---
5  | ready  | Untitled (4d:4d) | 2       | 10.110.0.3 | b2:8b:55:17:ae:40 | controller |
8  | ready  | Untitled (3a:7f) | 2       | 10.110.0.6 | 92:93:99:70:14:4c | compute    |
```

```
6  | ready  | Untitled (34:84) | 2       | 10.110.0.4 | f2:b3:1a:74:da:41 | cinder     |
7  | ready  | Untitled (f0:9b) | 2       | 10.110.0.5 | 56:09:fe:c6:06:40 | compute    |
```

You can ssh to any of the nodes using the IP address. For example, to ssh to the Cinder node:

```
ssh 10.110.0.4
```

You can also use the "id" shown in the first column, for example:

```
ssh node-6
```

How To: Exclude some drives from RAID-1 array

RAID-1 spans all configured disks on a node, putting a boot partition on each disk because OpenStack does not have access to the BIOS. It is not currently possible to exclude some drives from the Fuel configuration on the Fuel UI. This means that one cannot, for example, configure some drives to be used for backup and recover or as b-cache.

You can work around this issue as follows. This example is for a system that has three disks: sda, sdb, and sdc. Fuel will provision sda and sdb as RAID-1 for OpenStack but sdc will not be used as part of the RAID-1 array:

1. Use the Fuel CLI to obtain provisioning data:

```
fuel provisioning --env-id 1 -d
```

2. Remove the drive which you do not want to be part of RAID:

```
- size: 300
  type: boot
- file_system: ext2
  mount: /boot
  name: Boot
  size: 200
  type: raid
```

3. Run deployment

```
fuel provisioning --env-id 1 -u
```

4. Confirm that your partition is not included in the RAID array:

```
[root@node-2 ~]# cat /proc/mdstat
Personalities : [raid1]
md0 : active raid1 sda3[0] sdb3[1] 204736 blocks
      super 1.0 [2/2] [UU]
```

How To: Modify Kernel Parameters

Kernel parameters are options that are passed to the kernel when a Linux system is booted. Kernel parameters can be set in any of the following ways:

- Use the Fuel Welcome screen to define kernel parameters that will be set for the Fuel Master node when it is installed.
- Use the *Initial parameters* field on the Settings tab to define kernel parameters that Fuel will set on the target nodes. This only affects target nodes that will be deployed in the future, not those that have already been deployed.
- Use the Cobbler web UI (see *Using the Cobbler web UI to set kernel parameters*) to change kernel parameters for all nodes using or for specific nodes. The nodes appear in Cobbler only after they are deployed; to change parameters before deployment, stop the deployment, change the parameters, then proceed.
- Issue the **dockerctl** command on each node where you want to set kernel parameters; see *Using the dockerctl command to set kernel parameters*.

Any kernel parameter supported by Ubuntu can be set for the target nodes and the Fuel Master node. Some parameters that are frequently set for Fuel and OpenStack are:

ttys0=<speed>
: serial console for videoless servers

console=ttyS0,9600
: enable serial console

nofb
: disable Linux framebuffer

nomodeset
: disable kernel handling of the video card; required for some older integrated server video chips

intel_iommu and amd_iommu
: enable/disable physical-to-virtual address translation for peripheral devices. Some devices (such as Mellanox cards) require this to be on; other peripheral devices may be incompatible with device virtual address space and may work only with real address space. If you are unable to boot a node or the node has a kernel panic soon after being booted, setting this parameter to "off" may fix the problem.

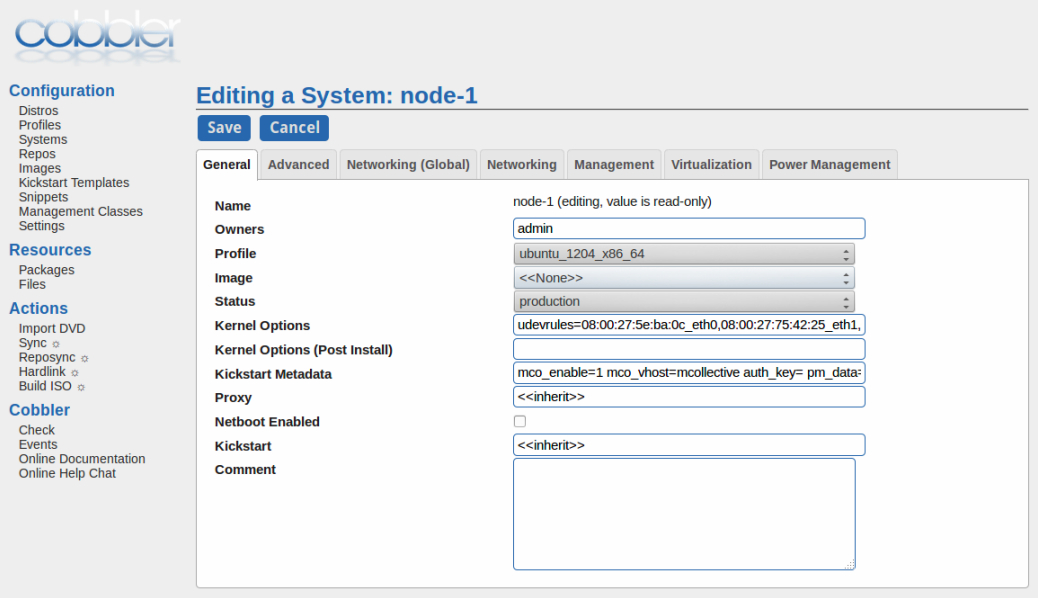
unsupported_hardware
: instructs the operating system to boot even if it does not recognize some configured hardware. If this parameter is not set, Linux does not boot when it detects critical hardware required at boot time (usually a new CPU model). Because most hardware provides backward compatibility with older versions, setting this kernel parameter may enable the system to boot. Note that, if no backward compatibility is provided, the system may panic or fail in other ways even with this parameter set.

Using the Cobbler web UI to set kernel parameters

You can use the Cobbler web UI to set kernel parameters:

- Use the https://<ip-addr>/cobbler_web URL to access the Cobbler web UI; replace <ip-addr> with the IP address for your Fuel Master Node.

- Log in, using the user name and password defined in the *cobbler* section of the */etc/fuel/astute.yaml* file.
- Select *Systems* from the menu on the right. This lists the nodes that are deployed in your environment. Select the node(s) for which you want to set new parameters and click "Edit". The following screen is displayed:



- Add the kernel parameters and values to the *Kernel Options (Post-install) field then click the `Save* button.

Using the dockerctl command to set kernel parameters

Use the **dockerctl** console command on the Fuel Master node to add a kernel parameter definition. For example, the following command sets the **intel_iommu=off** parameter:

```
`dockerctl shell cobbler cobbler profile edit --name bootstrap --kopts="intel_iommu=off" --
```

HowTo: Create an XFS disk partition

In most cases, Fuel creates the XFS partition for you. If for some reason you need to create it yourself, use this procedure:

> ***Note***
>
> Replace `/dev/sdb` with the appropriate block device you wish to configure.

1. Create the partition itself

```
fdisk /dev/sdb
  n(for new)
  p(for partition)
  <enter> (to accept the defaults)
  <enter> (to accept the defaults)
  w(to save changes)
```

2. Initialize the XFS partition

```
mkfs.xfs -i size=1024 -f /dev/sdb1
```

3. For a standard swift install, all data drives are mounted directly under /srv/node, so first create the mount point

```
mkdir -p /srv/node/sdb1
```

4. Finally, add the new partition to fstab so it mounts automatically, then mount all current partitions

```
echo "/dev/sdb1 /srv/node/sdb1 xfs \
noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab
mount -a
```

HowTo: Enable/Disable Galera Cluster Autorebuild Mechanism

By default, the autorebuild mechanism is enabled, so Fuel reassembles a Galera cluster automatically without any user interaction.

- The OCF Galera script checks every node in the Galera Cluster for the SEQNO position. This allows it to find the node with the most recent data.
- The script checks for the status of the current node, if it is synchronized with the quorum, the procedure stops; otherwise, SEQNO is obtained and stored in the Corosync CIB as a variable.
- The script sleeps for 300 seconds, allowing other nodes to join the Corosync quorum and push their UUIDs and SEQNOs, too.
- For every node in the quorum, the script compares the UUID and SEQNO. If at least one node has a higher SEQNO, it bootstraps the node as the Primary Component, allowing other nodes to join the newly formed cluster later;
- The Primary Component node is started with the *--wsrep-new-cluster* option, forming a new quorum.

To prevent the *autorebuild feature* you should do:

```
crm_resource unmanage clone_p_mysql
```

To re-enable *autorebuild feature* you should do:

```
crm_resource manage clone_p_mysql
```

To check GTID and SEQNO across all nodes saved in Corosync CIB you should do:

```
cibadmin --query --xpath "//nodes/*/*/nvpair[@name=\"gtid\"]"
```

To try an automated reassemble without reboot if cluster is broken just issue:

```
crm resource restart clone_p_mysql
```

To remove all GTIDs and SEQNOs from Corosync CIB and allow the OCF script to reread the data from the *grastate.dat* file, you should do:

```
cibadmin --delete-all --query --xpath "//nodes/*/*/nvpair[@name=\"gtid\"]" --force
```

HowTo: Backport Galera Pacemaker OCF script

Starting from Fuel 5.1, OCF script was completely redesigned which makes the *Galera cluster* cluster more reliable and predictable. The script can be backported to the Fuel pre-5.1 releases following the instructions below; similar steps could be used to backport to the older versions by adjusting the MySQL commands to match those used by the specific version of MySQL.

> ***Warning***
>
> Before performing any operations with Galera, you should schedule the maintenance window, perform backups of all databases, and stop all MySQL related services.

1. Set the p_mysql primitive in maintenance mode:

```
crm configure edit p_mysql \
meta is-managed=false
```

2. Check the status. It should show clone_p_mysql primitives as Unmanaged:

```
crm_mon -1
```

3. Download the latest OCF script from the fuel-library repository to the Fuel Master node:

```
wget --no-check-certificate -O \
/etc/puppet/modules/galera/files/ocf/mysql-wss \
https://raw.githubusercontent.com/stackforge/fuel-library/master/deployment/puppet/ \
galera/files/ocf/mysql-wss
```

4. The OCF script requires some modification as it was originally designed for MySQL 5.6:

```
perl -pi -e 's/--wsrep-new-cluster/--wsrep-cluster-address=gcomm:\/\//g' \
/etc/puppet/modules/galera/files/ocf/mysql-wss
```

5. Copy the script to all controllers:

```
for i in $(fuel nodes | awk '/ready.*controller.*True/{print $1}'); \
do scp /etc/puppet/modules/galera/files/ocf/mysql-wss \
node-$i:/etc/puppet/modules/galera/files/ocf/mysql-wss; done

for i in $(fuel nodes | awk '/ready.*controller.*True/{print $1}'); \
do scp /etc/puppet/modules/galera/files/ocf/mysql-wss \
node-$i:/usr/lib/ocf/resource.d/mirantis/mysql-wss; done
```

6. Configure the p_mysql resource for the new Galera OCF script:

```
crm configure edit p_mysql
```

Example of primitive for Ubuntu:

```
crm configure primitive p_mysql ocf:mirantis:mysql-wss \
       params socket="/var/run/mysqld/mysqld.sock" \
       pid="/var/run/mysqld/mysqld.pid" \
       test_passwd="password" test_user="wsrep_sst" \
       op monitor timeout="55" interval="60" enabled=true \
       op start timeout="475" interval="0" \
       op stop timeout="175" interval="0" \
       meta is-managed=true
```

Note

During this operation, the MySQL/Galera cluster will be restarted. This may take up to 5 minutes.

7. Check whether Galera Cluster is synced and functioning:

```
mysql -e "show global status like 'wsrep_cluster_status'"
```

8. IPs of all nodes should be present in Galera Cluster. This guarantees that all nodes participate in Cluster operations and acting properly.

```
mysql -e "show global status like 'wsrep_incoming_addresses'"
```

9. Restart MySQL related services:

   - Restart neutron on every Controller (if installed).
   - Restart the remaining OpenStack services on each Controller and Storage node.
   - Restart the OpenStack services on the Compute nodes.

HowTo: Backport Memcached backend fixes

Fuel 6.1 contains several High Availability (HA) fixes to the configuration of memcache_pool (see *Memcached*) backend for Keystone which makes response time of OpenStack services shorter if a memcached node fails. There is also a security fix to the Keystone configuration which makes the revocation of tokens to be kept in the MySQL backend instead of memcached. These patches can be backported to the Fuel 6.0 release following the instructions below. Note, Fuel 5.0 or older does not support memcache_pool backend and the fixes are not applicable for them.

> ***Warning***
>
> Before performing any operations with Keystone, you should schedule a maintenance window, perform backups of Keystone configuration files, and stop all Keystone related services.

1. Download the related fixes for puppet modules from the fuel-library repository to the Fuel Master node:

```
wget --no-check-certificate -O /etc/puppet/modules/keystone/manifests/init.pp \
   https://raw.githubusercontent.com/stack \
   forge/fuel-library/stable/6.0/deployment/puppet/keystone/manifests/init.pp
wget --no-check-certificate -O /etc/puppet/modules/keystone/spec/classes \
/keystone_spec.rb \
   https://raw.githubusercontent.com/stack \
   forge/fuel-library/stable/6.0/deployment/puppet/keystone/spec/classes \
/keystone_spec.rb
wget --no-check-certificate -O /etc/puppet/modules/openstack/manifests/keystone.pp \
   https://raw.githubusercontent.com/stack \
   forge/fuel-library/stable/6.0/deployment/puppet/deployment/puppet/openstack \
/manifests/keystone.pp
```

2. Copy the fixed files to all Controllers:

```
for i in $(fuel nodes --env <env_ID> | awk '/ready.*controller.*True/{print $1}'); do
  scp /etc/puppet/modules/keystone/manifests/init.pp \
  node-$i:/etc/puppet/modules/keystone/manifests/init.pp;\
  scp /etc/puppet/modules/keystone/spec/classes/keystone_spec.rb \
  node-$i:/etc/puppet/modules/keystone/spec/classes/keystone_spec.rb; \
  scp /etc/puppet/modules/openstack/manifests/keystone.pp \
  node-$i:/etc/puppet/modules/openstack/manifests/keystone.pp;\
done
```

> ***Note***
>
> This step assumes the environment id is a "1" and the controller nodes names have a standard Fuel notation, like "node-1", "node-42" and so on.

3. Update the */etc/keystone/keystone.conf* configuration file on all of the controller nodes as the following:

```
[revoke]
driver = keystone.contrib.revoke.backends.sql.Revoke

[cache]
memcache_dead_retry = 30
memcache_socket_timeout = 1
memcache_pool_maxsize = 1000

[token]
driver = keystone.token.persistence.backends.memcache_pool.Token
```

4. Restart all Keystone related services:

    - Restart Keystone on every Controller.
    - Restart Neutron on every Controller (if installed).
    - Restart the remaining OpenStack services on each Controller and Storage node.
    - Restart the OpenStack services on the Compute nodes.

HowTo: Backport RabbitMQ Pacemaker OCF script

Fuel 6.1 contains many fixes to the RabbitMQ OCF script which makes the *RabbitMQ* cluster more reliable and predictable. This can be backported to the Fuel 5.1 and 6.0 releases following the instructions below. The older Fuel versions do not use Pacemaker for RabbitMQ cluster management, hence the given changes to the OCF script are not applicable for them.

> ***Note***
>
> The OCF script in the Fuel 6.1 release also distributes and ensures the consistent Erlang cookie file among the all controller nodes. For backports to the older Fuel versions, this feature is disabled by default in the OCF script. If you think you want to enable it, please read carefully the details below.

1. Schedule maintenance window.

   > ***Warning***
   >
   > Before performing any operations with RabbitMQ, you should schedule maintenance window, perform backups of all RabbitMQ mnesia files and OCF scripts, and stop all OpenStack services an all environment nodes, see *HowTo: Manage OpenStack services* for details.

   Mnesia files are located at `/var/lib/rabbitmq/mnesia/` and OCF files can be found at `/usr/lib/ocf/resource.d/mirantis/`.

2. Inside the maintenance window, put the p_rabbitmq-server primitive in unmanaged state at one of the controller nodes:

```
pcs resource unmanage master_p_rabbitmq-server
```

or with the crm tool:

```
crm resource unmanage master_p_rabbitmq-server
```

> ***Note***
>
> Normally, the crm tool can be installed from the crmsh package, by commands:
>
> ```
> yum install crmsh || apt-get install crmsh
> ```

3. Check the status. It should show p_rabbitmq-server primitives as "Unmanaged":

```
pcs resource show
```

or with the crm tool:

```
crm_mon -1
```

Unmanaged p_rabbitmq-server resources should look like:

```
Master/Slave Set: master_p_rabbitmq-server [p_rabbitmq-server] (unmanaged) \
    p_rabbitmq-server  (ocf::fuel:rabbitmq-server):    Master node-1 (unmanaged) \
    p_rabbitmq-server  (ocf::fuel:rabbitmq-server):    Slave node-2 (unmanaged) \
    p_rabbitmq-server  (ocf::fuel:rabbitmq-server):    Slave node-3 (unmanaged) \
```

4. Download the latest OCF script from the fuel-library repository to the Fuel Master node:

```
wget --no-check-certificate -O /etc/puppet/modules/nova/files/ocf/rabbitmq \
https://raw.githubusercontent.com/stack\
forge/fuel-library/stable/6.0/deployment/puppet/nova/files/ocf/rabbitmq
chmod +x /etc/puppet/modules/nova/files/ocf/rabbitmq
```

> ***Note***
>
> For the Fuel 5.1 release update the link to use a "5.1" version in the download path.

5. Copy the script to all controllers:

```
for i in $(fuel nodes --env <env_ID> | awk '/ready.*controller.*True/{print $1}'); \
do scp /etc/puppet/modules/nova/files/ocf/rabbitmq \
node-$i:/etc/puppet/modules/nova/files/ocf/rabbitmq; done
for i in $(fuel nodes --env <env_ID> | awk '/ready.*controller.*True/{print $1}'); \
do scp /etc/puppet/modules/nova/files/ocf/rabbitmq \
node-$i:/usr/lib/ocf/resource.d/mirantis/rabbitmq-server; done
```

> ***Note***
>
> This step assumes the environment id is a "1" and the controller nodes names have a standard Fuel notation, like "node-1", "node-42", and so on.

6. Update the configuration of the p_rabbitmq-server resource for the new RabbitMQ OCF script at any controller node:

```
crm configure edit p_rabbitmq-server
```

An example primitive may look like:

```
primitive p_rabbitmq-server ocf:mirantis:rabbitmq-server \
        params node_port="5673" \
        meta failure-timeout="60s" migration-threshold="INFINITY" \
        op demote interval="0" timeout="60" \
        op notify interval="0" timeout="60" \
        op promote interval="0" timeout="120" \
        op start interval="0" timeout="120" \
        op monitor interval="30" timeout="60" \
        op stop interval="0" timeout="60" \
        op monitor interval="27" role="Master" timeout="60"
```

or in an XML notation:

```
xml <primitive class="ocf" id="p_rabbitmq-server" provider="mirantis" \
   type="rabbitmq-server">
```

```
  <operations>
    <op id="p_rabbitmq-server-monitor-30" interval="30" name="monitor" timeout="60"/> \
    <op id="p_rabbitmq-server-monitor-27" interval="27" name="monitor" \
        role="Master" timeout="60"/>
    <op id="p_rabbitmq-server-start-0" interval="0" \
        name="start" timeout="60"/>
    <op id="p_rabbitmq-server-stop-0" interval="0" \
        name="stop" timeout="60"/>
    <op id="p_rabbitmq-server-promote-0" interval="0" \
        name="promote" timeout="120"/>
    <op id="p_rabbitmq-server-demote-0" interval="0" \
        name="demote" timeout="60"/>
    <op id="p_rabbitmq-server-notify-0" interval="0" \
        name="notify" timeout="60"/>
  </operations> \
  <instance_attributes id="p_rabbitmq-server-instance_attributes"> \
    <nvpair id="p_rabbitmq-server-instance_attributes-node_port" \
     name="node_port" value="5673"/>
  </instance_attributes> \
  <meta_attributes id="p_rabbitmq-server-meta_attributes"> \
    <nvpair id="p_rabbitmq-server-meta_attributes-migration-threshold" \
     name="migration-threshold" value="INFINITY"/>
    <nvpair id="p_rabbitmq-server-meta_attributes-failure-timeout" \
     name="failure-timeout" value="60s"/> \
  </meta_attributes> \
</primitive>
#vim:set syntax=pcmk
```

Make sure the following changes are applied:

- To the *params* stanza:
  - Add the parameter `command_timeout` with the value `--signal=KILL`

    > ***Note***
    >
    > The `command_timeout` parameter value is given for Ubuntu OS.

    Use `some_param="some_value"` notation, or for the XML case:

    ```
    <nvpair id="p_rabbitmq-server-instance_attributes-some_param" \
     name="some_param" value="some_value"/>
    ```

  - Add the `erlang_cookie` parameter with the value `false`

> ***Note***
>
> If you want to allow the OCF script to manage the Erlang cookie files, provide the existing Erlang cookie from `/var/lib/rabbitmq/.erlang.cookie` as an `erlang_cookie` parameter, otherwise set this parameter to a `false`. Note, that a different Erlang cookie would require to erase mnesia files for all controller nodes as well.

> ***Warning***
>
> Erasing the mnesia files will also erase all custom users, vhosts, queues, and other RabbitMQ entities, if any.

- To the *meta* stanza:
  - Set the `failure-timeout` to a `"360s"`
- To the *op* stanzas:
  - Set the `notify interval` to a `"0"` and the `timeout` to a `"180"`
  - Set the `start interval` to a `"0"` and the `timeout` to a `"360"`

Or the same with the pcs tool:

```
pcs resource meta p_rabbitmq-server failure-timeout=360s
pcs resource op remove p_rabbitmq-server notify interval=0 timeout=60
pcs resource op add p_rabbitmq-server notify interval=0 timeout=180
pcs resource op remove p_rabbitmq-server start interval=0 timeout=60
pcs resource op add p_rabbitmq-server start interval=0 timeout=360
```

> ***Note***
>
> Ignore messages like "Error: Unable to find operation matching:"

> ***Note***
>
> You cannot add resource attributes with pcs tool, you should install crmsh package and use crm tool in order to update `command_timeout` and `erlang_cookie` parameters, see details above.

As a result, the given example resource should look like:

```
# pcs resource show p_rabbitmq-server

 Resource: p_rabbitmq-server (class=ocf provider=fuel type=rabbitmq-server)
  Attributes: node_port=5673 debug=false command_timeout=--signal=KILL erlang_cookie=EC
  Meta Attrs: migration-threshold=INFINITY failure-timeout=360s
  Operations: monitor interval=30 timeout=60 (p_rabbitmq-server-monitor-30)
              monitor interval=27 role=Master timeout=60 (p_rabbitmq-server-monitor-27)
              monitor interval=103 role=Slave timeout=60 OCF_CHECK_LEVEL=30 (p_rabbitmq
              start interval=0 timeout=360 (p_rabbitmq-server-start-0)
              stop interval=0 timeout=120 (p_rabbitmq-server-stop-0)
              promote interval=0 timeout=120 (p_rabbitmq-server-promote-0)
              demote interval=0 timeout=120 (p_rabbitmq-server-demote-0)
              notify interval=0 timeout=180 (p_rabbitmq-server-notify-0)
```

or with the crm tool:

```
# crm configure show p_rabbitmq-server
primitive p_rabbitmq-server ocf:fuel:rabbitmq-server \
 op monitor interval=30 timeout=60 \
 op monitor interval=27 role=Master timeout=60 \
 op monitor interval=103 role=Slave timeout=60 OCF_CHECK_LEVEL=30 \
 op start interval=0 timeout=360 \
 op stop interval=0 timeout=120 \
 op promote interval=0 timeout=120 \
 op demote interval=0 timeout=120 \
 op notify interval=0 timeout=180 \
 params node_port=5673 debug=false command_timeout="--signal=KILL" erlang_cookie=EOKOWX
 meta migration-threshold=INFINITY failure-timeout=360s
```

The output also may have an XML notation and may look like:

```
 xml <primitive class="ocf" id="p_rabbitmq-server" provider="mirantis" \
     type="rabbitmq-server"> \
  <operations> \
    <op id="p_rabbitmq-server-monitor-30" interval="30" name="monitor" \
     timeout="60"/>
    <op id="p_rabbitmq-server-monitor-27" interval="27" name="monitor" \
      role="Master" timeout="60"/>
```

```
    <op id="p_rabbitmq-server-start-0" interval="0" name="start" \
     timeout="360"/>
    <op id="p_rabbitmq-server-stop-0" interval="0" name="stop" \
     timeout="60"/>
    <op id="p_rabbitmq-server-promote-0" interval="0" name="promote" \
     timeout="120"/>
    <op id="p_rabbitmq-server-demote-0" interval="0" name="demote" \
     timeout="60"/>
    <op id="p_rabbitmq-server-notify-0" interval="0" name="notify" \
     timeout="180"/>
  </operations> \
  <instance_attributes id="p_rabbitmq-server-instance_attributes"> \
    <nvpair id="p_rabbitmq-server-instance_attributes-node_port" \
     name="node_port" value="5673"/>
    <nvpair id="p_rabbitmq-server-instance_attributes-command_timeout" \
     name="command_timeout" value="--signal=KILL"/>
    <nvpair id="p_rabbitmq-server-instance_attributes-erlang_cookie" \
     name="erlang_cookie" value="EOKOWXQREETZSHFNTPEY"/> \
  </instance_attributes> \
  <meta_attributes id="p_rabbitmq-server-meta_attributes"> \
    <nvpair id="p_rabbitmq-server-meta_attributes-migration-threshold" \
     name="migration-threshold" value="INFINITY"/>
    <nvpair id="p_rabbitmq-server-meta_attributes-failure-timeout" \
     name="failure-timeout" value="360s"/> \
  </meta_attributes> \
</primitive>
```

7. Put the p_rabbitmq-server to management state and restart it:

```
pcs resource manage master_p_rabbitmq-server
pcs resource disable master_p_rabbitmq-server
pcs resource enable master_p_rabbitmq-server
pcs resource cleanup master_p_rabbitmq-server
```

   or with the crm tool:

```
crm resource manage master_p_rabbitmq-server
crm resource restart master_p_rabbitmq-server
crm resource cleanup master_p_rabbitmq-server
```

> ***Note***
>
> During this operation, the RabbitMQ cluster will be restarted. This may take from a 1 up to 20 minutes. If there are any issues, see *crm - Cluster Resource Manager*.

8. Check whether the RabbitMQ cluster is functioning on each controller node:

```
rabbitmqctl cluster_status
rabbitmqctl list_users
```

9. Restart RabbitMQ related services:

   See *HowTo: Manage OpenStack services* for details.

HowTo: Manage OpenStack services

1. Stop or start OpenStack services

   In order to get a full list of OpenStack services from the corresponding list of OpenStack projects and their statuses in SysV or Upstart use the following commands:

   On Ubuntu:

```
services=$(curl http://git.openstack.org/cgit/open\
stack/governance/plain/reference/projects.yaml | \
  egrep -v 'Security|Documentation|Infrastructure' | \
  perl -n -e'/^(\w+):$/ && print "openstack-",lc $1,".*\$|",lc $1,".*\$|"')

initctl list | grep -oE $services | grep start
```

   Now you can start, stop, or restart the OpenStack services, see details about recommended order below.

> ***Warning***
>
> Fuel configures some services, like Neutron agents, Heat engine, Ceilometer agents, to be managed by Pacemaker instead of generic init scripts. These services should be managed only with the pcs or crm tools!

   In order to figure out the list of services managed by Pacemaker, you should first find disabled or not running services with the command:

   On Ubuntu:

```
initctl list | grep -oE $services | grep stop
```

   Next, you should inspect the output of command `pcs resource` (or `crm resource list`) and find the corresponding services listed, if any.

   The Pacemaker resources list is:

```
root@node-1:~# crm status

Stack: corosync
Current DC: node-1.domain.tld (1) - partition with quorum
Version: 1.1.12-561c4cf
3 Nodes configured
43 Resources configured

Online: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]

 Clone Set: clone_p_vrouter [p_vrouter]
```

```
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
vip__management (ocf::fuel:ns_IPaddr2): Started node-1.domain.tld
vip__public_vrouter (ocf::fuel:ns_IPaddr2): Started node-1.domain.tld
vip__management_vrouter (ocf::fuel:ns_IPaddr2): Started node-1.domain.tld
vip__public (ocf::fuel:ns_IPaddr2): Started node-2.domain.tld
Master/Slave Set: master_p_conntrackd [p_conntrackd]
    Masters: [ node-1.domain.tld ]
    Slaves: [ node-2.domain.tld node-5.domain.tld ]
Clone Set: clone_p_haproxy [p_haproxy]
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
Clone Set: clone_p_dns [p_dns]
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
Clone Set: clone_p_mysql [p_mysql]
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
Master/Slave Set: master_p_rabbitmq-server [p_rabbitmq-server]
    Masters: [ node-1.domain.tld ]
    Slaves: [ node-2.domain.tld node-5.domain.tld ]
Clone Set: clone_p_heat-engine [p_heat-engine]
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
Clone Set: clone_p_neutron-plugin-openvswitch-agent [p_neutron-plugin-openvswitch-agen
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
Clone Set: clone_p_neutron-dhcp-agent [p_neutron-dhcp-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
Clone Set: clone_p_neutron-metadata-agent [p_neutron-metadata-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
Clone Set: clone_p_neutron-l3-agent [p_neutron-l3-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
Clone Set: clone_p_ntp [p_ntp]
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
Clone Set: clone_ping_vip__public [ping_vip__public]
    Started: [ node-1.domain.tld node-2.domain.tld node-5.domain.tld ]
```

You may notice, that there is only a heat-engine service is managed by Pacemaker and disabled in OS. At any controller node, use the following command to start or stop cluster-wide:

```
pcs resource enable clone_p_openstack-heat-engine
pcs resource disable clone_p_openstack-heat-engine
```

or with crm tool:

```
crm resource start clone_p_openstack-heat-engine
crm resource stop clone_p_openstack-heat-engine
```

2. Start, stop, restart order for OpenStack services.

    - Start/stop/restart keystone on every Controller.

- Start/stop/restart neutron-server and agents on every Controller (if installed).

  > ***Note***
  >
  > Use pcs or crm tools for corresponding services, when managed by Pacemaker

- Start/stop/restart the remaining OpenStack services on each Controller and Storage node, in any order.

  > ***Note***
  >
  > Use pcs or crm tools for corresponding services, when managed by Pacemaker

- Start/stop/restart the OpenStack services on the Compute nodes, in any order.

3. Unmanage, manage services controlled by Pacemaker.

   In order to put a resource in uncontrolled state, use the following commands:

   ```
   pcs resource unmanage <some_resource_name>
   ```

   or with crm tool

   ```
   crm resource unmanage <some_resource_name>
   ```

   This will not stop the running resources.

   And to bring the resource back to be managed by Pacemaker:

   ```
   pcs resource manage <some_resource_name>
   ```

   or with crm tool

   ```
   crm resource manage <some_resource_name>
   ```

Adding, Redeploying, and Replacing Nodes

This section discusses how to add, redeploy, and replace *nodes* in your OpenStack environment. Compute and Storage nodes have always been expandable; Controller nodes could not be added or replaced before Mirantis OpenStack 5.1.

Redeploy a Non-Controller Node

Redeploying a *node* refers to the process of changing the roles that are assigned to it. For example, you may have several nodes that run both Compute and Storage roles and you want to redeploy some of those nodes to be only Storage nodes while others become only Compute nodes. Or you might want to redeploy some Compute and Storage nodes to be MongoDB nodes.

To redeploy a non-controller node, follow these steps:

1. Use live migration to move instances from the Compute nodes you are going to redeploy.
2. If appropriate, back up or copy information from the Operating System nodes being redeployed.
3. Use the procedure described in *Assign a role or roles to each node server* to remove the node from your environment. Select the node(s) to be deleted then click on the "Delete Nodes" button.
4. Click on the "Deploy Changes" button on that screen.
5. Wait for the node to become available as an unallocated node.
6. Use the same Fuel screen to assign an appropriate role to each node being redeployed.
7. Click on the "Deploy Changes" button.
8. Wait for the environment to be deployed.

After redeploying an Operating System node, you will have to manually apply any configuration changes you made and reinstall the software that was running on the node or restore the system from the backup you made before redeploying the node.

Add a Non-Controller Node

Non-controller nodes can be added to your OpenStack environment.

To add a non-controller node to your environment, follow these steps:

1. Physically configure the node into your hardware environment.
2. Wait for the new node to show up as an "Unallocated Node" on your Fuel dashboard.
3. Click the "Add Node" button; on the screen described in *Assign a role or roles to each node server*; the unallocated node will be displayed. Assign the role or roles to the node that you want.
4. Click the "Deploy Changes" button and wait for the node to be deployed.

   The cluster must be redeployed to update the configuration files. Most of the services that are running are not affected but the redeployment process restarts HAProxy and a few other services.

Add a MongoDB node

Additional *MongoDB* roles can be added to an existing deployment by using shell commands. Any number of MongoDB roles (or standalone nodes) can be deployed into an OpenStack environment using the Fuel Web UI during the initial deployment but you cannot use the Fuel Web UI to add MongoDB nodes to an existing environment.

Fuel installs MongoDB as a backend for *Ceilometer (OpenStack Telemetry)*. Ideally, you should configure one MongoDB node for each Controller node in the environment so, if you add Controller nodes, you should also add MongoDB nodes.

To add one or more MongoDB nodes to the environment:

1. Add an entry for each new MongoDB node to the *connection* parameter in the *ceilometer.conf* file on each Controller node. This entry needs to specify the new node's IP address for the Management *logical network*.
2. Open the *astute.yaml* file on any deployed MongoDB node and determine which node has the *primary-mongo* role; see *MongoDB nodes configuration*. Write down the value of the **fqdn** parameter; you will use this to **ssh** to this node.
3. Retrieve the *db_password* value from the *Ceilometer configuration* section of same file. You will use this password to access the primary MongoDB node.
4. Connect to the MongoDB node that has the *primary-mongo* role and log into Mongo:

```
ssh ... <fqdn-of-primary-mongo-node>
mongo -u admin -p <db_password> admin
```

5. Configure each MongoDB node to be added to the environment:

```
ceilometer:PRIMARY> rs.add ("<management-ip-address-of-node>")
```

6. Restart the ceilometer services.

Add a Controller node

Mirantis OpenStack 5.1 and later allows you to add Controller nodes to your environment without redeploying the entire environment.

You may want to add a Controller node to your environment for any of the following reasons:

- You deployed a single-Controller, HA-ready environment; it is not actually highly-available until at least three Controller nodes are configured but it is run using *Pacemaker*, *Corosync*, and the other utilities used to manage an HA environment. To make it highly available, add two or more Controller nodes.
- The resources of your existing Controller nodes are being exhausted and you want to supplement them.
- You are replacing a Controller node that failed; you will first need to remove the failed Controller as discussed in *Remove a Controller node*.

Each Controller cluster should include an odd number of nodes -- 1 node, 3 nodes, 5 nodes, et cetera.

To add Controllers to your environment:

1. Physically configure the servers in your hardware environment and wait for them to be discovered and reflected in the "Unallocated Node" count on your Fuel dashboard.

2. Use the *Assign a role or roles to each node server* screen to assign the Controller role to each node.
3. Check the connectivity between the nodes by running *Verify Networks*.
4. Click "Deploy changes" and wait for Fuel to redeploy the environment.
5. Run the *post-deployment checks* `.

Remove a Controller node

You may need to remove a Controller node from your environment, usually because you want to replace it with a different server. This may be because of a catastrophic hardware failure on the node's server or because you want to replace the server with a more powerful system.

To remove a Controller node:

1. Remove Controller(s) from environment by going to the Nodes tab in the Fuel desktop, selecting the node(s) to be deleted, and click on the "Delete Nodes" button.

   Puppet removes the controller(s) from the configuration files and retriggers services.
2. Physically remove the controller from the configuration.

Configuring an Operating System node

Fuel provisions the *Operating System Role* with the supported operating system that was selected for the environment but *Puppet* does not deploy other packages on this node. You can find out more about what Fuel installs on such nodes in *How the Operating System Role is provisioned*.

You can access an Operating System node using **ssh**, just as you would access any other node; see *Accessing the shell on the nodes*. Some general administrative tasks you may need to perform are:

- Create file systems on partitions you created and populate the *fstab* file so they will mount automatically.
- Configure additional *logical networks* you need; Fuel only configures the Admin/PXE network.
- Set up any monitoring facilites you want to use such as **monit** and **atop**; configure **syslog** to send error messages to a centralized syslog server.
- Tune kernel resources to optimize performance for the particular applications you plan to run here.

You are pretty much free to install and configure this node any way you like. By default, all the repositories from the Fuel Master node are configured so you can install packages from these repositories by running the **apt-get install <package-name>** command. You can also use **scp** to copy other software packages to this node and then install them using **apt-get** or **yum**.

How To: Safely remove a Ceph OSD node

Before a Ceph OSD node can be deleted from an environment all data must be moved to other OSDs. Fuel prevents the deletion of OSDs that still have data on them. The following process will move any data present on the soon-to-be-deleted node to other OSDs in the cluster.

1. Determine which OSD processes are running on the target node (node-35 in this case):

```
# ceph osd tree
   # id    weight  type name       up/down reweight
   -1      0.4499  root default
   -2      0.09998         host node-35
   0       0.04999                 osd.0   up      1
   1       0.04999                 osd.1   up      1
```

   From this output we can see that OSDs 0 and 1 are running on this node.

2. Remove the OSDs from the Ceph cluster:

```
# ceph osd out 0
# ceph osd out 1
```

   This will trigger a rebalance. Placement groups will be moved to other OSDs. Once that has completed we can finish removing the OSD. This process can take minutes or hours depending on the amount of data to be rebalanced. While the rebalance is in progress the cluster state will look something like:

```
# ceph -s
cluster 7fb97281-5014-4a39-91a5-918d525f25a9
 health HEALTH_WARN recovery 2/20 objects degraded (10.000%)
 monmap e1: 1 mons at {node-33=10.108.2.4:6789/0}, election epoch 1, quorum 0 node-33
 osdmap e172: 7 osds: 7 up, 5 in
  pgmap v803: 960 pgs, 6 pools, 4012 MB data, 10 objects
        10679 MB used, 236 GB / 247 GB avail
        2/20 objects degraded (10.000%)
               1 active
             959 active+clean
```

   Once the rebalance is complete it will look like:

```
# ceph -s

cluster 7fb97281-5014-4a39-91a5-918d525f25a9
 health HEALTH_OK
 monmap e1: 1 mons at {node-33=10.108.2.4:6789/0}, election epoch 1, quorum 0 node-33
 osdmap e172: 7 osds: 7 up, 5 in
  pgmap v804: 960 pgs, 6 pools, 4012 MB data, 10 objects
```

```
        10679 MB used, 236 GB / 247 GB avail
             960 active+clean
```

When the cluster is in state HEALTH_OK the OSD(s) can be removed from the CRUSH map:

```
On Ubuntu hosts:
  # stop ceph-osd id=0

# ceph osd crush remove osd.0
# ceph auth del osd.0
# ceph osd rm osd.0
```

After all OSDs have been deleted the host can be removed from the CRUSH map:

```
# ceph osd crush remove node-35
```

3. This node can now be deleted from the environment with Fuel.

Caveats:

- Ensure that at least replica count hosts remain in the cluster. If the replica count is 3 then there should always be at least 3 hosts in the cluster.
- Do not let the cluster reach its full ratio. By default when the cluster reaches 95% utilization Ceph will prevent clients from writing data to it. See Ceph documentation for additional details.

How To: Adjust Placement Groups when adding additional Ceph OSD node(s)

When adding additional Ceph OSD nodes to an environment, it may be necessary to addjust the placement groups (pg_num) and the placement groups for placement (pgp_num) for the pools. The following process can be used to update these two values for the Ceph cluster. These adjustments may not be necessary unless you add a significant number of OSDs to an existing cluster. Fuel attempts to calculate appropriate values for the initial deployment but does not adjust them when adding additional OSDs.

1. Determine the current values for pg_num and pgp_num for each pool that will be touched. The pools that may need to be adjusted are the 'backups', 'images', 'volumes' and 'compute' pools.

   First get the list of pools that are currently configured.

   ```
   # ceph osd lspools
   0 data,1 metadata,2 rbd,3 images,4 volumes,5 compute,
   ```

   The pools that may need to be adjusted are the 'backups', 'images', 'volumes', and 'compute' pools. You can query each pool to see what the current value of pg_num and pgp_num is for a pool individually.

   ```
   # ceph osd pools get {pool-name} pg_num
   # ceph osd pools get {pool-name} pgp_num
   ```

2. Calculate the correct value for pg_num and pgp_num that should be used based on the number of OSDs the cluster has. See the Ceph Placement Groups documentation for additional details on how to properly calculate these values. Ceph.com has a Ceph PGs Per Pool Calculator that can be helpful in this calculation.

3. Adjust the pg_num and pgp_num for each of the pools as necessary.

   ```
   # ceph osd pool set {pool-name} pg_num {pg_num}
   # ceph osd pool set {pool-name} pgp_num {pgp_num}
   ```

   It should be noted that this will cause a cluster rebalance to occur which may have performance impacts on services consuming Ceph.

Caveats:

- It is also not advisable to set pg_num and pgp_num to large values unless necessary as it does have an impact on the amount of resources required. See the Choosing the Number of Placement Groups documentation for additional details.

HowTo: Shut down the whole cluster

To shut down the whole cluster, follow these steps:

1. Stop all virtual machines.
2. Either power off all the nodes (with clicking *poweroff* button or running *poweroff* command) at once or shut them down in the following order:
    - Computes
    - Controllers (one by one or all in one)
    - Cinder/Ceph/Swift
    - Other/Neutron (if separate Neutron node exists)

Starting up cluster

To start up cluster, power on all the nodes.

If you have a cluster with Neutron, take the instructions below into consideration.

Cinder or Ceph require Neutron, Neutron node requires database and Controllers, so the following sequence is possible:

1. Power on Ceph/Cinder/Swift/MongoDB/Zabbix nodes.
2. Start all controllers and Neutron (if separate Neutron node exists) and wait until RabbitMQ, Neutron agents and Galera get synced by Pacemaker. It should take no more than 5 minutes.
3. Ensure that HAProxy is OK. Note, that HAProxy is under Pacemaker so you should use *pcs resource <command> clone_p_haproxy* to manage it.
4. Ensure that RabbitMQ cluster is OK and fix it if there are failed nodes.
5. Ensure that Galera cluster is OK and fix it if necessary. Note, that Galera is under Pacemaker, so you should use *pcs resource* to manage it.
6. Run *pcs resource restart clone_p_neutron-metadata-agent* command.
7. Run *pcs resource restart clone_p_neutron-plugin-openvswitch-agent*
8. Wait several minutes and check the cluster state with *crm status* command. All Neutron agents including L3 and DHCP should be started.
9. Restart all Openstack services.
10. Power on compute nodes.

Creating and Configuring ML2 Drivers for Neutron

Fuel 5.1 and later supports *ML2* mechanism drivers for Neutron. Some network configurations, such as advanced features provided by *Mellanox ConnectX-3 network adapters*, require ML2 mechanism driver configuration.

You can add ML2 configuration data to the quantum_settings section of the *node.yaml* file (see *Using YAML configuration files*); this updates the *astute.yaml* file:

```
quantum_settings:
  server:
    service_plugins:
        'neutron.services.l3_router.l3_router_plugin.L3RouterPlugin,
        neutron.services.firewall.fwaas_plugin.FirewallPlugin,
        neutron.services.metering.metering_plugin.MeteringPlugin'
  L2:
    provider:'ml2'
    mechanism_drivers: 'openvswitch'
    type_drivers: "local,flat,l2[:segmentation_type]"
    tenant_network_types: "local,flat,l2[:segmentation_type]"
    flat_networks: '*'
    segmentation-type:'vlan'
    tunnel_types: l2[:segmentation_type]
    tunnel_id_ranges: l2[:tunnel_id_ranges]
    vxlan_group: 'None'
    vni_ranges: l2[:tunnel_id_ranges]
```

Note the following:

- The following values should be set only if L2[enable_tunneling] is true: tunnel_types, tunnel_id_ranges, vxlan_group, vni_ranges.
- The l2[:item] settings refer to values that are already in the quantum_settings.
- This only shows new items that are related to ml2 configuration. The values shown are the defaults that are used if no other value is set.
- All Neutron component packages are available to download.

Configuring Multiple Cluster Networks

You can configure multiple network domains per single OpenStack environment using *Fuel CLI*. You should run the commands described here after Fuel is installed but before booting the target nodes. For background information about how this feature is implemented, see *Implementing Multiple Cluster Networks*.

Requirements for an environment that runs multiple cluster networks are:

- Must use an encapsulation protocol such as *Neutron GRE*.
- A gateway must be defined for each *logical network* when the cluster has multiple *Node Groups*.
- All controllers must be members of the same Node Group; if they are not, the *HAProxy* VIP does not work.

> ***Note***
>
> It is possible to use ECMP to work around this restriction; documenting how to do this is beyond the scope of this document.

To configure an environment that uses multiple cluster networks, do the following.

1. Add each DHCP network into the *dnsmasq.template* file.
2. Start a **dhcp-helper** process:

   ```
   dhcp-helper -s <IP-of-fuel-master> -i <IP-of-DHCP-NIC>
   ```

   -s specifies the system to which packets are relayed, so is set to the IP address of the Fuel Master node. **-i** specifies the local interface where DHCP packets arrive, so is set to the IP address of the NIC that is connected to your DHCP network.
3. Create each new Node Group with a command like this:

   ```
   fuel --env 1 nodegroup --create --name "alpha"
   ```

   Assign a meaningful name to each *group* you create.
4. Configure the new Network Group(s) by uploading a new *network-1.yaml* file and modifying it as discussed in the reference page, then upload the modified file.
5. Boot the nodes for all environments.
6. When you assign roles to your target nodes, Fuel tries to automatically determine the node's group based on the DHCP address. If that fails, the group can be manually assigned with the following command:

   ```
   fuel nodegroup --env 1 --assign --node <list-of-node-ids> --group <nodegroup-id>
   ```

   See *Node group* for complete information about using the Fuel CLI to manage node groups.

Using YAML configuration files

Fuel uses YAML files to pass configuration attributes to *puppet*

Passing custom attributes are useful when you have some Puppet manifests that should be run but are not supported by Fuel itself.

> ***Warning***
>
> Be very careful when modifying the configuration files. A simple typo when editing these files may severely damage your environment. When you modify the YAML files, you will receive a warning that some attributes were modified from the outside. Some features may become inaccessible from the UI after you do this. Please, read *Using Fuel CLI* section carefully.

To do this:

1. Create an environment following the instructions in *Create a new OpenStack environment*.
2. Assign roles to nodes following the instructions in *Assign a role or roles to each node server* but do not start deployment yet.
3. Log into the Fuel Master node and dump provisioning information using this *fuel CLI* command:

   ```
   fuel --env 1 provisioning default
   ```

   where `--env 1` that points to the specific environment (id=1 in this example).

   To dump deployment information, the command is:

   ```
   fuel --env 1 deployment default
   ```

   For a full list of configuration types that can be dumped, see the *fuel CLI* section.

   These commands create a directories called *provisioning_1* or *deployment_1*, which include a number of YAML files that correspond to roles that are currently assigned to nodes. Each file includes parameters for current role, so you can freely modify and save them.

Customizing Passwords

A few service passwords are not managed by the *Keystone* based *access control* facility that was introduced in Fuel 5.1. You may want to set your own passwords in some cases.

You can edit files and modify password values. For example, you can set the MySQL root password in this block:

```
"mysql": {
    "root_password": "mynewpassword"
},
```

Adding new modules

You can extend Fuel functionality by adding new Puppet modules. You can do this either by adding them to the */etc/puppet/modules* file on the Fuel Master node, or you can edit existing modules to change the deployment behavior in some way.

As an example, let's add a new module called *'packages'* that installs some useful packages from the repository that is located on the Master node.

The module should have the following structure:

```
packages/
packages/manifests
packages/manifests/init.pp
```

init.pp should have this content:

```
class profile {
        $tools = $::fuel_settings['tools']
        package { $tools :
        ensure => installed,
        }
}
```

To implement this module:

1. Copy this module to the */etc/puppet/modules* directory on the Master node.
2. Add 'include profile' to the end of the */etc/puppet/manifests/site.pp* file to enable this module. Placing new *include* statements in the middle of the file may break the deployment process and/or its dependencies.
3. As you can see, there is list of packages to install that should be passed through the Fuel parameters system.

   Let's add this attribute to the downloaded file's top level hash:

   ```
   “tools”: [
           “htop”,
           “tmux”,
   ]
   ```

   Provisioned nodes will have this addition in their parameters and our 'profile' module will be able to access their values and install the given list of packages during node deployment.
4. Upload the modified configuration:

```
fuel --env 1 deployment upload
```

You can also use the `--dir` option to set a directory from which to load the parameters.

5. Start the deployment process as usual.

This operation has following effects:

- Parameters that are about to be sent to the orchestrator are replaced completely with the ones you specified.
- The cluster sets the *is_customized* flag, which is checked by the UI so you will get a message about attributes customization.

Docker Containers and Dockerctl

Docker provides user-friendly commands that can be used to deploy *LXC (Linux containers)* containers.

- Docker brings LXC to the foreground by wrapping it with user-friendly commands.
- Dockerctl is a simple wrapper for Docker that improves Docker's offline image handling and container versioning. It adds additional management tools that are useful when managing your Fuel deployment.

See *Docker containers and dockerctl* for more background information.

Container types

Application Container

An application container is the most common type of container. It usually runs a single process in the foreground and writes logs to stdout/stderr. Docker traps these logs and saves them automatically in its database.

Storage Container

A storage container is a minimalistic container that runs Busybox and acts as a sharer of one or more directories. It needs to run only one time and then spends the majority of its existence in Exited state.

Command reference

Below is a list of commands that are useful when managing LXC containers on the Fuel Master.

Basic usage

Get a list of available commands:

```
docker help
```

Get a list of all running containers:

```
docker ps
```

Get a list of all containers available:

```
docker ps -a
```

> ***Note***
>
> the storage containers used for sharing files among application containers are usually in Exited state. Exited state means that the container exists, but no processes inside are running.

Start a new Docker container with the specified commands:

```
docker run [options] imagename [command]
```

Example: The command below creates a temporary postgres container that is ephemeral and not tied to any other containers. This is useful for testing without impacting production containers.

```
docker run --rm -i -t fuel/postgres /bin/bash
```

Import a Docker image:

```
docker load -i (archivefile)
```

Loads in a Docker image in the following formats: .tar, .tar.gz, .tar.xz. lrz is not supported.

Save a Docker image to a file:

```
docker save image > image.tar
```

Dockerctl

Build and run storage containers, then run application containers:

```
dockerctl build all
```

> ***Note***
>
> This can take a few minutes, depending on your hardware.

Launch a container from its image with the necessary options. If the container already exists, will ensure that this container is in a running state:

```
dockerctl start <appname> [--attach]
```

Optionally, *--attach* option can be used to monitor the process and view its stdout and stderr.

Display the entire container log for /app/. Useful for troubleshooting:

```
dockerctl logs <appname>
```

Stop or restart a container:

```
dockerctl stop|restart <appname>
```

Create a shell or run a command:

```
dockerctl shell <appname> [command]
```

> ***Note***
>
> The container must be running first in order to use this feature. Additionally, quotes must be escaped if your command requires them.

Stop and destroy a container:

```
dockerctl destroy <appname>
```

> ***Note***
>
> This is not reversible, so use with caution.

Find containets names by running:

```
dockerctl list
```

System changes for Docker affecting Fuel 5.0 and later

The Fuel Master base system is modified in 5.0. These changes were made mostly to enable directory sharing between containers to operate smoothly:

- /etc/astute.yaml moved to /etc/fuel/astute.yaml
- /etc/nailgun/version.yaml moved to /etc/fuel/version.yaml
- Base OS puppet is now run from /etc/puppet/modules/nailgun/examples/host-only.pp
- Postgres DB is now inside a container. You can access it if you run "dockerctl shell postgres" or connect to localhost from base host.
- DNS resolution is now performed inside the cobbler container. Additional custom entries should be added inside /etc/dnsmasq.d/ inside the cobbler container or via Cobbler itself.

- Starting with Fuel 6.1, Docker containers with host networking are used. This means that dhcrelay is not used anymore because cobbler/dnsmasq are bound to the host system.
- Application logs are inside /var/log/docker-logs, including astute, nailgun, cobbler, and others.
- Supervisord configuration is located inside /etc/supervisord.d/(CurrentRelease)/
- Containers are automatically restarted by supervisord. If you need to stop a container for any reason, first run supervisorctl stop /app/, and then dockerctl stop /app/

Fuel Master architecture changes for Docker

Starting with Fuel 5.1, in order to enable containerization of the Fuel Master's services, several pieces of the Fuel Master node design were changed. Most of that change came from Puppet, but below is a list of modifications to Fuel to enable Docker:

- DNS lookups come from Cobbler container
- App containers launch in order, but not in a synchronous manner. Retries were added to several sections of deployment in case a dependent service is not yet ready.
- The version.yaml file is extended to include production key with values docker and docker-build.
- Extended Docker's default iptables rules to ensure that traffic visibility is appropriate for each service.

Enable Experimental Features

Beginning with Fuel 5.1 and in later release versions, *experimental features* are disabled by default. Experimental features provide useful functionality but may not be mature enough to be appropriate for environments that require high levels of stability.

To enable experimental features, execute the following on a running Fuel Master node. You should do this before *creating* and *configuring* your environment if you want to have access to Experimental features during those phases, or after you have deployed your cloud if you want to access the experimental features post-deployment. You can also enable Experimental Features after you *upgrade* your Fuel Master to 5.1.

- Log into your Fuel Master console as *root*.
- Manually modify the */etc/fuel/version.yaml* file to add "experimental" to the "feature_groups" list in the "VERSION" section. For example:

```
VERSION:
  ...
  feature_groups:
    - mirantis
    - experimental
```

- Restart the *Nailgun* container with dependencies by running:

```
dockerctl restart nailgun
dockerctl restart nginx
dockerctl shell cobbler
cobbler sync
exit
```

For more details about configuring nailgun settings see Extending OpenStack Settings.

Alternatively, you can build a custom ISO with the experimental features enabled:

```
make FEATURE_GROUPS=experimental iso
```

Fuel Access Control

Access to the Fuel Dashboard is controlled in Mirantis OpenStack 5.1 and later. Authentication is under control of *Keystone*.

The default username/password can be changed:

- During Fuel installation; see *Fuel login*.
- From the main Fuel UI screen; see *Launch Wizard to Create New Environment*.
- Using the Fuel CLI; see *Change and Set Fuel password*

If the password for the Fuel Dashboard is changed using the Fuel UI or the Fuel CLI, the new password is stored only in Keystone; it is not written to any file. If you forget the password, you can change it by using the **keystone** command on the Fuel Master Node:

```
keystone --os-endpoint=http://<your_master_ip>:35357/v2.0 --os-token=<admin_token> password
```

The default value of <your_master_ip> is 10.20.0.2. The port number of 35357 never changes.

The <admin_token> is stored in the */etc/fuel/astute.yaml* file on the Fuel Master node.

To run or disable authentication, modify */etc/nailgun/settings.yaml* (AUTHENTICATION_METHOD) in the Nailgun container.

All endpoints except the agent updates and version endpoint are protected by an authentication token, obtained from Keystone by logging into Fuel as the *admin* user with the appropriate password.

Services such as *Astute*, *Cobbler*, Postgres, MCollective, and *Keystone*), which used to be protected with the default password, are now each protected by a user/password pair that is unique for each Fuel installation.

Beginning with release 6.0, the *Nailgun* and *OSTF* services endpoints are added to Keystone and now it is possible to use the Keystone service catalog to obtain URLs of those services instead of hardcoding them.

Fuel Authentication is implemented by a dedicated Keystone instance that Fuel installs in a new *docker* container on the Fuel Master.

- Fuel Menu generates passwords for fresh installations; the upgrade script generates passwords when upgrading. The password is stored in the Keystone database.
- The *nailgun* and *ostf* users are created in the *services* project with admin roles. They are used to authenticate requests in middleware, rather than requiring that each request by middleware be validated using the Keystone admin token as was done in Release 5.1.
- Some Nailgun URLs are not protected; they are defined in *nailgun/middleware/keystone.py* in the public_url section.
- The authentication token does not expire for 24 hours so it is not necessary to store the username and password in the browser cache.
- A cron script runs daily in the Keystone container to delete outdated tokens using the **keystone-manage token_flush** command. It can be seen using the **crontab -l** command in the Keystone container.
- Support for storing authentication token in cookies is added in releases 5.1.1 and later; this allows the API to be tested from the browser.

- The **keystonemiddleware** python package replaces the deprecated **keystoneclinet.middleware** package; this is an internal change that makes the implementation more stable. All recent fixes and changes are made to **keystonemiddleware**; which was extracted from **keystoneclinet.middleware** in earlier releases.

Beginning with releases 5.1.1 and later, the user must supply a password when upgrading Fuel from an earlier release. This password can be supplied on the command line when running the installation script or in response to the prompt (this is the same password that is used to access Fuel UI).

Managing your Ceph cluster

Accessing the Puppet manifest for Ceph

The **node** parameter defines the names of the Ceph pools to pre-create. By default, *volumes* and *images* are required to set up the OpenStack hooks.

```
node 'default' {
  ...
}
```

The **class** section configures components for all Ceph-OSD nodes in the environment:

```
class { 'ceph::deploy':
    auth_supported   => 'cephx',
    osd_journal_size => '2048',
    osd_mkfs_type    => 'xfs',
}
```

You can modify the authentication type, Journal size (specified in KB), and the filesystem architecture to use.

Verify the deployment

Use the **ceph -s** or **ceph health** command on one of the Controller or Ceph-OSD nodes to check the current status of the Ceph cluster.

The output of this command looks like:

```
root@fuel-ceph-02:~# ceph -s
   health HEALTH_OK
   monmap e1: 2 mons at {fuel-ceph-01=10.0.0.253:6789/0,fuel-ceph-02=10.0.0.252:6789/0}, el
   osdmap e23: 4 osds: 4 up, 4 in
   pgmap v275: 448 pgs: 448 active+clean; 9518 bytes data, 141 MB used, 28486 MB / 28627 MB
   mdsmap e4: 1/1/1 up {0=fuel-ceph-02.local.try=up:active}
```

Look for the following two lines:

- monmap: should display the correct number of Ceph-MON processes.
- osdmap: should display the correct number of Ceph-OSD instances (one per node per volume)

ceph -s may return an error similar to the following:

```
root@fuel-ceph-01:~# ceph -s
health HEALTH_WARN 63 pgs peering; 54 pgs stuck inactive; 208 pgs stuck unclean; recovery 2
...
```

ceph commands return "missing keyring" error, such as:

```
2013-08-22 00:06:19.513437 7f79eedea760 -1 monclient(hunting): ERROR: missing keyring, cann
2013-08-22 00:06:19.513466 7f79eedea760 -1 ceph_tool_common_init failed.
```

To analyze the problem:

- Check the links in */root/ceph*.keyring*. There should be one for each admin, osd, and mon role that is configured. If any are missing, it may be the cause of the error. To correct use soft-links:

  ```
  ceph-deploy gatherkeys {monitor host}
  ```

  Place the downloaded keys to /etc/ceph/. Remove the original files from /root and create symlinks with the same names in /root, pointing to the actual files in /etc/ceph/.

- Try to run the following command:

  ```
   ceph-deploy gatherkeys {mon-server-name}

  If this does not work,
  an error may have occurred when initializing the cluster.
  ```

- Run the following command to find running ceph processes:

  ```
  ps axu | grep ceph
  ```

  If this lists a python process running for *ceph-create-keys*, it usually indicates that the Ceph-MON processes are unable to communicate with each other.

  - Check the network and firewall for each Ceph-MON. Ceph-MON defaults to port 6789.
  - If *public_network* is defined in the *ceph.conf* file, *mon_host* and DNS names must be inside the *public_network* or **ceph-deploy** does not create the Ceph-MON processes.

Missing OSD instances

For the default configuration, you should have one Ceph-OSD instance per volume for each Ceph-OSD node listed in the configuration. If one or more of these is missing, it may indicate a problem initializing and mounting the disks. Common causes:

- The disk or volume is in use.
- The disk partition did not refresh in the kernel.
- Customized Ceph.conf may contain errors on mount parameters string. Ceph-deploy fails to mount such partitions.

Check the osd tree:

```
#ceph osd tree
```

```
 # id    weight  type name       up/down reweight
-1      6       root default
-2      2               host controller-1
0       1                       osd.0   up      1
3       1                       osd.3   up      1
-3      2               host controller-2
1       1                       osd.1   up      1
4       1                       osd.4   up      1
-4      2               host controller-3
2       1                       osd.2   up      1
5       1                       osd.5   up      1
```

Ceph pools

To see which Ceph pools have been created, use the **ceph osd lspools** command:

```
# ceph osd lspools
0 data,1 metadata,2 rbd,3 images,4 volumes,
```

By default, two pools -- *image* and *volumes* are created. In this case, we also have *data*, *metadata*, and *rbd* pools.

Test that Ceph works with OpenStack components

Glance

Upload an image to Glance to see if it is saved in Ceph:

```
source ~/openrc
glance image-create --name cirros --container-format bare \
--disk-format qcow2 --is-public yes --location \
https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-disk.img
```

This should return something similar to the following:

```
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | None                                 |
| container_format | bare                                 |
| created_at       | 2013-08-22T19:54:28                  |
| deleted          | False                                |
| deleted_at       | None                                 |
| disk_format      | qcow2                                |
| id               | f52fb13e-29cf-4a2f-8ccf-a170954907b8 |
| is_public        | True                                 |
| min_disk         | 0                                    |
```

```
| min_ram          | 0                                    |
| name             | cirros                               |
| owner            | baa3187b7df94d9ea5a8a14008fa62f5     |
| protected        | False                                |
| size             | 0                                    |
| status           | active                               |
| updated_at       | 2013-08-22T19:54:30                  |
+------------------+--------------------------------------+
```

Then check rbd:

```
rbd ls images
rados -p images df
```

Cinder

Create a small volume and see if it is saved in Cinder:

```
source openrc
cinder create 1
```

This will instruct Cinder to create a 1 GB volume. This should return something similar to the following:

```
+=====================+======================================+
|       Property      |                Value                 |
+=====================+======================================+
|     attachments     |                  []                  |
+---------------------+--------------------------------------+
| availability_zone   |                 nova                 |
+---------------------+--------------------------------------+
|       bootable      |                 false                |
+---------------------+--------------------------------------+
|      created_at     |      2013-08-30T00:01:39.011655      |
+---------------------+--------------------------------------+
| display_description |                 None                 |
+---------------------+--------------------------------------+
|     display_name    |                 None                 |
+---------------------+--------------------------------------+
|          id         | 78bf2750-e99c-4c52-b5ca-09764af367b5 |
+---------------------+--------------------------------------+
|        metadata     |                  {}                  |
+---------------------+--------------------------------------+
|         size        |                  1                   |
+---------------------+--------------------------------------+
|     snapshot_id     |                 None                 |
+---------------------+--------------------------------------+
|     source_volid    |                 None                 |
```

```
+---------------------+--------------------------------------+
|        status       |               creating               |
+---------------------+--------------------------------------+
|     volume_type     |                 None                 |
+---------------------+--------------------------------------+
```

Check the status of the image using its *id* with the *cinder show* <*id*> command:

```
cinder show 78bf2750-e99c-4c52-b5ca-09764af367b5

+==============================+======================================+
|           Property           |                Value                 |
+==============================+======================================+
|         attachments          |                  []                  |
+------------------------------+--------------------------------------+
|      availability_zone       |                 nova                 |
+------------------------------+--------------------------------------+
|           bootable           |                false                 |
+------------------------------+--------------------------------------+
|          created_at          |      2013-08-30T00:01:39.000000      |
+------------------------------+--------------------------------------+
|     display_description      |                 None                 |
+------------------------------+--------------------------------------+
|         display_name         |                 None                 |
+------------------------------+--------------------------------------+
|              id              | 78bf2750-e99c-4c52-b5ca-09764af367b5 |
+------------------------------+--------------------------------------+
|           metadata           |                  {}                  |
+------------------------------+--------------------------------------+
|    os-vol-host-attr:host     |       controller-19.domain.tld       |
+------------------------------+--------------------------------------+
| os-vol-tenant-attr:tenant_id |   b11a96140e8e4522b81b0b58db6874b0   |
+------------------------------+--------------------------------------+
|             size             |                  1                   |
+------------------------------+--------------------------------------+
|         snapshot_id          |                 None                 |
+------------------------------+--------------------------------------+
|         source_volid         |                 None                 |
+------------------------------+--------------------------------------+
|            status            |              available               |
+------------------------------+--------------------------------------+
|          volume_type         |                 None                 |
+------------------------------+--------------------------------------+
```

If the image shows *status available*, it was successfully created in Ceph. You can check this with the *rbd ls volumes command*.

```
rbd ls volumes
volume-78bf2750-e99c-4c52-b5ca-09764af367b5
```

Rados GW

First confirm that the cluster is *HEALTH_OK* using *ceph -s* or *ceph health detail*. If the cluster is not healthy most of these tests will not function.

> ***Note***
>
> RedHat distros: mod_fastcgi's /etc/httpd/conf.d/fastcgi.conf must have FastCgiWrapper Off or rados calls will return 500 errors.

Rados relies on the service *radosgw* (Debian) *ceph-radosgw* (RHEL) to run and create a socket for the webserver's script service to talk to. If the radosgw service is not running, or not staying running then you need to inspect it closer.

The service script for radosgw might *exit 0* and not start the service. An easy way to test this is to simply *service ceph-radosgw restart* if the service script can not stop the service, it was not running in the first place.

You can also check to see if the radosgw service is be running with the *ps axu | grep radosgw* command, but this might also show the webserver script server processes as well.

Most commands from *radosgw-admin* will work regardless of whether the radosgw service is running or not.

Swift

Create a new user:

```
 radosgw-admin user create --uid=test --display-name="username" --email="username@domai
{ "user_id": "test",
  "display_name": "username",
  "email": "username@domain.com",
  "suspended": 0,
  "max_buckets": 1000,
  "auid": 0,
  "subusers": [],
  "keys": [
        { "user": "test",
          "access_key": "CVMC8OX9EMBRE2F5GA8C",
          "secret_key": "P3H4Ilv8Lhx0srz8ALO\/7udwkJd6raIz11s71FIV"}],
  "swift_keys": [],
  "caps": []}
```

Swift authentication works with subusers. In OpenStack this will be *tenant:user*, so you need to mimic it:

```
radosgw-admin subuser create --uid=test --subuser=test:swift --access=full

{ "user_id": "test",
  "display_name": "username",
  "email": "username@domain.com",
  "suspended": 0,
  "max_buckets": 1000,
  "auid": 0,
  "subusers": [
        { "id": "test:swift",
          "permissions": "full-control"}],
  "keys": [
        { "user": "test",
          "access_key": "CVMC8OX9EMBRE2F5GA8C",
          "secret_key": "P3H4Ilv8Lhx0srz8ALO\/7udwkJd6raIz11s71FIV"}],
  "swift_keys": [],
  "caps": []}
```

Generate a secret key.

> ***Note***
>
> *--gen-secret* is required in Cuttlefish and newer.

```
radosgw-admin key create --subuser=test:swift --key-type=swift --gen-secret
{ "user_id": "test",
  "display_name": "username",
  "email": "username@domain.com",
  "suspended": 0,
  "max_buckets": 1000,
  "auid": 0,
  "subusers": [
        { "id": "test:swift",
          "permissions": "full-control"}],
  "keys": [
        { "user": "test",
          "access_key": "CVMC8OX9EMBRE2F5GA8C",
          "secret_key": "P3H4Ilv8Lhx0srz8ALO\/7udwkJd6raIz11s71FIV"}],
  "swift_keys": [
        { "user": "test:swift",
          "secret_key": "hLyMvpVNPez7lBqFlLjcefsZnU0qlCezyE2IDRsp"}],
  "caps": []}
```

Sample test commands should look as follows:

```
swift -A http://localhost:6780/auth/1.0 -U test:swift -K "eRYvzUr6vubg93dMRMk60RWYiGdJG
swift -A http://localhost:6780/auth/1.0 -U test:swift -K "eRYvzUr6vubg93dMRMk60RWYiGdJG
swift -A http://localhost:6780/auth/1.0 -U test:swift -K "eRYvzUr6vubg93dMRMk60RWYiGdJG
```

Reset the Ceph cluster

You can reset the Ceph cluster if necessary after correcting configuration errors. This is often easier than having to re-deploy the entire OpenStack environment.

To do this, create a simple shell script with the following contents (below), then edit the string *export all="compute-4 controller-1 controller-2 controller-3"* and define the variable *$all* to contain all nodes that contain Ceph-MON and Ceph-OSD services that you want to re-initialize as well as all Compute nodes.

You can get the node names with the *fuel node list* command.

```
export all="compute-4 controller-1 controller-2 controller-3"
for node in $all
do
   ssh $node 'service ceph -a stop ;
   umount /var/lib/ceph/osd/ceph* ;
done;
ceph-deploy purgedata $all;
ceph-deploy purge $all;
yum install -y ceph-deploy;
rm ~/ceph* ;
ceph-deploy install $all
```

S3 API in Ceph RADOS Gateway

Introduction

Ceph RADOS Gateway offers accessing the same objects and containers using many different APIs. The two most important are: OpenStack Object Storage API v1 (aka Swift API) and Amazon S3 (Simple Storage Service). Beside these, radosgw supports several internal interfaces dedicated for logging, replication, and administration. Covering them is not the purpose of this document.

Getting started

Assumption has been made that you posses a working Ceph cluster and the radosgw is able to access the cluster. In case of using cephx security system, which is the default scenario, both radosgw and cluster must authenticate to each other. Please note this is not related to any user-layer authentication mechanism used in radosgw like Keystone, TempURL, or TempAuth. If radosgw is deployed with Fuel, cephx should work out of the box. In case of manual deployment, official documentation will be helpful.

To enable or just verify whether S3 has been properly configured, the configuration file used by radosgw (usually */etc/ceph/ceph.conf*) should be inspected. Please take a look at radosgw's section (usually *client.radosgw.gateway*) and consider the following options:

- *rgw_enable_apis* - if present, it must contain at least `s3`

User authentication

The component providing S3 API implementation inside radosgw actually supports two methods of user authentication: Keystone-based and RADOS-based (internal). Each of them may be separately enabled or disabled with an appropriate configuration option. The first one takes precedence over the second. That is, if both methods are enabled and Keystone authentication fails for any reason (wrong credentials, connectivity problems etc.), the RADOS-based will be treated as fallback.

Keystone-based

Configuration

Keystone authentication for S3 is not enabled by default, even if using Fuel. Please take a look at appropriate section (usually *client.radosgw.gateway*) in radosgw's configuration file and consider following options:

Option name	Default value in rgw	Comment	Present in Fuel-deployed configuration
rgw_s3_auth_use_keystone	false	must be present and set to true	no
rgw_keystone_url	empty	must be present and set to point admin interface of Keystone (usually port 35357, *some versions of Fuel wrongly set the port to 5000*).	yes
rgw_keystone_admin_token	empty	must be present and match the admin token set in Keystone configuration	yes
rgw_keystone_accepted_roles	Member, admin	should correspond the Keystone schema. Fuel setting seems to be OK	yes

In case of using Keystone-based authentication, user management is fully delegated to Keystone. You may use **keystone** CLI command to do that. Please be aware that the EC2/S3 *<AccessKeyId>:<secret>* credentials pair do not map well into Keystone (there is no a direct way to specify a tenant), so special compatibility layer has been introduced. Practically, it means you need to tell Keystone about the mapping parameters manually. For example:

```
keystone ec2-credentials-create --tenant-id=68a23e70b5854263ab64f2ddc16c2a38 --user-id=2ccd

WARNING: Bypassing authentication using a token & endpoint (authentication credentials are

+-----------+----------------------------------+
|  Property |              Value               |
+-----------+----------------------------------+
|   access  | 3862b51ecc6a43a78ffca23a05e7c0ad |
|   secret  | a0b4cb375d5a409893b05e36812fb811 |
| tenant_id | 68a23e70b5854263ab64f2ddc16c2a38 |
|  trust_id |                                  |
|  user_id  | 2ccdd07ae153484296d308eab10c85dd |
+-----------+----------------------------------+
```

`access` and `secret` are the parameters needed to authenticate a client to S3 API.

Performance Impact

Please be aware that Keystone's PKI tokens are not available together with S3 API. Moreover, radosgw doesn't cache Keystone responses while using S3 API. This could lead to authorization service overload.

RADOS-based (internal)

Configuration

The RADOS-based authentication mechanism should work out of the box. It is enabled by default in radosgw and Fuel does not change this setting. However, in case of necessity to disable it, the *rgw_s3_auth_use_rados* may be set to *false*.

User management could be performed with command line utility **radosgw-admin** provided with Ceph. For example, to create a new user the following command should be executed:

```
radosgw-admin user create --uid=ant --display-name="aterekhin"
{ "user_id": "ant",
 "display_name": "aterekhin",
 "email": "",
 "suspended": 0,
 "max_buckets": 1000,
 "auid": 0,
 "subusers": [],
 "keys": [
       { "user": "ant",
         "access_key": "9TEP7FTSYTZF2HZD284A",
         "secret_key": "8uNAjUZ+u0CcpbJsQBgpoVgHkm+PU8e3cXvyMclY"}],
 "swift_keys": [],
 "caps": [],
 "op_mask": "read, write, delete",
 "default_placement": "",
 "placement_tags": [],
 "bucket_quota": { "enabled": false,
     "max_size_kb": -1,
     "max_objects": -1},
 "user_quota": { "enabled": false,
     "max_size_kb": -1,
     "max_objects": -1},
 "temp_url_keys": []}
```

`access_key` and `secret_key` are the parameters needed to authenticate a client to S3 API.

Verification

To check whether everything works fine a low-level S3 API client might be very useful, especially if it can provide assistance in the matter of authentication signature generation. S3 authentication model requires that the client provides a key identifier (*AccessKeyId*) and HMAC-based authentication signature, which is calculated against a user key (*secret*) and some HTTP headers present in the request. The well-known solution is s3curl application. However, unpatched versions contain severe bugs (see LP1446704). We fixed them already and sent a pull request to its author. However, until it is not merged, we may recommend trying this version of s3curl.

Step-by-step instruction

1. Install the *libdigest-hmac-perl* package
2. Download the S3 API client using the following link:

```
git clone https://github.com/rzarzynski/s3curl
```

3. Set permisions for s3curl.pl:

```
chmod u+x s3curl.pl
```

4. Create *.s3curl* file in your home directory. This file should contain your *AccessKeyId* and *SecretAccessKey* pairs.

```
%awsSecretAccessKeys = (
    # your account
    ant => {
        id => '9TEP7FTSYTZF2HZD284A',
        key => '8uNAjUZ+u0CcpbJsQBgpoVgHkm+PU8e3cXvyMclY',
    },
);
```

5. Set the S3 endpoint in *s3curl.pl* file, for example:

```
my @endpoints = ('172.16.0.2');
```

or use *s3curl.pl* script to add it:

```
./s3curl.pl --id ant --endpoint <s3-endpoint-host>
```

Example:

```
./s3curl.pl --id ant --endpoint 172.16.0.2
```

Note

You can get your S3 endpoint using the keystone CLI command as follows:

```
keystone endpoint-get --service 's3'
+--------------+------------------------+
|   Property   |          Value         |
+--------------+------------------------+
```

```
| s3.publicURL | http://172.16.0.2:8080 |
+--------------+------------------------+
```

6. Try to run the s3curl command to test S3 API, for example:

    - To get an object

      ```
      ./s3curl.pl --id <frienfdly name> -- <endpoint>/<bucket name>/<key name>
      ```

      Example:

      ```
      ./s3curl.pl --id ant -- http://172.16.0.2:8080/bucket/key
      ```

    - Upload a file

      ```
      ./s3curl.pl --id <frienfdly name> --put <path to file> -- <endpoint>/<bucket name>/
      ```

      Example:

      ```
      ./s3curl.pl --id ant --put file -- http://172.16.0.2:8080/bucket/key
      ```

Note

Known issues: LP1477457 in Fuel 6.1.

Sahara Deployment

Sahara is a service for launching Hadoop clusters on OpenStack. It is vendor-agnostic and currently supports the following distributions:

- Vanilla Apache Hadoop
- Hortonworks Data Platform (HDP)
- Cloudera Hadoop Distribution (CDH)
- Apache Spark
- MapR

Sahara can install Hadoop clusters on demand. The user must populate several parameters such as the Hadoop version and cluster topology and Sahara will deploy this cluster in a few minutes. It can also scale the cluster by adding or removing nodes as needed.

- For Sahara usage guidelines, read the User Guide section of the Sahara documentation.
- The list of prebuilt images is available here: *Sahara Images*. The images are usually provided in a QCOW2 format which works with the default OpenStack hypervisors QEMU/KVM. If the cloud is installed with a different hypervisor the image should be converted to an appropriate format before being uploaded to Glance. The instructions for the Vmware Nova backend are here *Image Requirements*.
- Planning considerations for running Sahara are in *Planning a Sahara Deployment*.
- Installation instructions for Sahara are in *Installing Sahara*

This section provides additional information about running Sahara.

Sahara Security Groups

Sahara uses auto-security groups for opening required ports for each plugin.

To learn about the open ports for each plugin, please refer to the official documentation:

- Apache Hadoop
- Hortonworks Hadoop Version 2.2 and Version 2.3
- Apach Spark
- Cloudera
- MapR

Preparing Sahara for Testing

You can run Platform Tests to verify whether Sahara is functioning correctly. To run the tests, Sahara must be deployed and configured.

You run the tests in the tenant you specified in the *OpenStack Settings* tab during the OpenStack installation. By default, the *admin* tenant is used for the tests.

You must have at least 4096 MB of available RAM in the tenant you are using for Sahara Platform Tests otherwise some tests may fail.

To prepare Sahara for testing, you should get an image and register it with Sahara image registry:

1. Download the image with Hadoop for Sahara
2. Register it with Sahara:
    a. Upload the image into Glance into the *admin* tenant. Name it *sahara*.
    b. In Horizon, access the *Data Processing* panel under the *Project* menu.
    c. Switch to the *admin* tenant.
    d. Go to the *Image Registry* menu.
    e. Click the *Register Image* button. The *Image registration* dialog appears.
    f. Select the image you have just uploaded.
    g. Set username to `ubuntu`.
    h. Select the tags: `plugin=vanilla` and `version=2.4.1`.
    i. Click `Add plugin tags`.
    j. Click `Done`.

Now you can test Sahara.

Image Requirements

Vmware Nova backend requires VMDK image format. You may use qemu-img utility to convert a QCOW2 image to VMDK.

```
qemu-img convert -O vmdk <original_image>.qcow2 <converted_image>.vmdk
```

Sahara Test Details

Hadoop cluster operations test

checks whether Sahara can launch a Hadoop cluster using the Vanilla plugin

Target component

Sahara

Scenario

1. Login to the OpenStack dashboard.

2. Register an image:
    - Navigate to *Project* > *Data Processing* > *Image Registry*.
    - Click on *Register Image*.
    - Select an image for registering from the drop-down list in the *Image* field.
    - Specify the `User Name` value for this OS.
    - Set the following values: `Plugin=vanilla`, `Version=2.6.0`.
    - Click *Add plugin tags* and *Done*.
3. Create a master node group template:
    - Navigate to *Project* > *Data Processing* > *Node Group Templates*.
    - Click *Create Template*.
    - In the *Create Node Group template* dialog, set the following values: `Plugin name=Vanilla Apache Hadoop`, `Hadoop version=2.6.0`. Click *Create* to proceed.
    - In the second *Create Node Group template* dialog, set the following values: `Template Name=vanilla2-master`, `OpenStack Flavor=m1.small`, `Floating IP pool=(external network)`, check `namenode`, `secondarynamenode`, `resourcemanager`, `historyserver`, and `oozie` in the *Process* section.
    - Click *Create*.
4. Create a worker node group template:
    - Navigate to *Project* > *Data Processing* > *Node Group Templates*.
    - Click *Create Template*.
    - In the *Create Node Group template* dialog, set the following values: `Plugin name=Vanilla Apache Hadoop` and `Hadoop version=2.6.0`. Click *Create* to proceed.
    - In the second *Create Node Group template* dialog, set the following values: `Template Name=vanilla2-worker`, `OpenStack Flavor=m1.small`, `Floating IP pool=(external network)`; check `datanode` and `nodemanager` in the *Process* section.
    - Click *Create*.

5. Create a cluster template:
    - Navigate to *Project* > *Data Processing* > *Cluster Templates*.
    - Click *Create Template*.
    - In the *Create Cluster Template* dialog, set the following values: `Plugin name=Vanilla Apache Hadoop`, `Hadoop version=2.6.0`. Click *Create* to proceed.
    - In the second *Create Cluster Template* dialog, set the following values:
        - the *Details* tab: specify `Template Name=vanilla2-template`;
        - the *Node Groups* tab: add `vanilla2-master` and `vanilla2-worker`;
    - Click *Create*.
6. Launch the cluster:
    - Navigate to *Project* > *Data Processing* > *Clusters*.
    - Click *Launch Cluster*.
    - In the *Launch Cluster* dialog, set the following values: `Plugin name=Vanilla Apache Hadoop`, `Hadoop version=2.6.0`. Click *Create* to proceed.
    - In the second *Launch Cluster* dialog, set `Cluster Name=vanilla2-cluster`.
    - Click *Create* and wait until the cluster gets the `Active` status.
7. Delete the cluster:
    - On the *Clusters* page, select the `vanilla2-cluster` cluster from the table and click *Delete Cluster*.
8. Delete the templates:
    - Navigate to *Project* > *Data Processing* > *Cluster Templates*.
    - Select the *vanilla2-template* templates and click *Delete Templates*.
    - Navigate to *Project* > *Data Processing* > *Node Group Templates*.
    - Select *vanilla2-master* and *vanilla2-worker* templates, and click *Delete Templates*.

For more information, see Sahara documentation.

Sahara Images

Prepared images can be downloaded from the following locations:

- Ubuntu 12.04 for CDH 5.4.0
- Ubuntu 14.04 for Vanilla Hadoop 2.6.0
- Ubuntu 14.04 for Spark 1.3.1
- Ubuntu 14.04 for MapR 4.0.2
- CentOS 6.6 for Vanilla Hadoop 2.6.0
- CentOS 6.6 for HDP 2.2
- CentOS 6.6 for HDP 2.3

- CentOS 6.6 for CDH 5.4.0
- CentOS 6.6 for MapR 4.0.2

> ***Note***
>
> For the HDP 2.2 and the HDP 2.3 installations, you use the same image.

The default username for these images depends on the distribution:

Operating System	Username
Ubuntu 12.04	ubuntu
Ubuntu 14.04	ubuntu
CentOS 6.6	cloud-user

You can find MD5 checksum of an image by adding the `.md5` suffix to the image URL.

For example:

http://sahara-files.mirantis.com/mos70/sahara-kilo-vanilla-2.6.0-ubuntu-14.04.qcow2.md5.

To check an `.iso` file with an MD5 hash, run:

```
$ md5sum -c sahara-kilo-vanilla-2.6.0-ubuntu-14.04.qcow2.md5
..
```

Murano Deployment Notes

Murano provides an application catalog that application developers and administrators can use to publish various cloud-ready applications in a browsable, categorized catalog. Users can select applications from this catalog and deploy them easily.

Some highlights of Murano features include:

- Native to OpenStack
- Application catalog; see https://wiki.openstack.org/wiki/Murano
- Introduction of abstraction level
- Support for Availability Zones and Disaster Recovery scenarios
- Use of native Windows features to provide HA solutions

Note that, because Microsoft Windows and other necessary components can only be obtained directly from Microsoft, Murano is still to some degree a do-it-yourself project. Fuel is able to configure Murano's dashboard, API, and engine services, but you will need to read documentation on the steps to set up a Murano base image; see Creating a Murano Image. Images can be uploaded via Glance.

Murano Components

Dashboard

The Murano Dashboard is available after Murano is installed. Use the same credentials to log into Murano as you use for Horizon (via Keystone). From the Murano Dashboard, you can manage the application catalog.

Murano API

The Murano API provides the ability to manage applications in the OpenStack clouds. For further reading, refer to Murano API Specification

Engine

The Murano orchestration engine transforms objects into a series of Heat and Murano API commands. The Murano dashboard sends requests to the Murano API, which sends these requests to the engine.

Preparing Murano for Testing

The platform tests are run in the tenant you've specified in the 'OpenStack Settings' tab during OpenStack installation. The 'admin' tenant is selected by default.

To prepare Murano for linux-based services deployment testing, add a Linux based image to Murano:

1. Download the following image:

   http://murano-files.mirantis.com/ubuntu_14_04-murano-agent_stable_juno.qcow2

> ***Note***
>
> The base operating system of the Murano image is not related to the base operating system of the OpenStack environment.

If you want to build your own Linux image, you can use the Murano agent. For instructions, see the Murano documentation (Linux Image Builder).

> ***Note***
>
> The instructions in the official Murano documentation may be outdated; following these instructions cannot guarantee success with image creation.

2. Upload the image to the *admin* tenant in the OpenStack Image Service (Glance).
3. Open the 'Murano' tab.
4. Navigate to the 'Manage' submenu
5. Click the 'Images' menu.
6. Click 'Mark Image'. The Image registration window displays.
7. Select the Linux image with the Murano Agent.
8. In the 'Title' field, set the title for this image.
9. Select the 'Generic Linux' type.
10. Click 'Mark'.

Murano is ready for testing.

Murano platform test details

The Platform Tests run as part of the Fuel Health Test suite and test Murano functionality when Murano is installed in the OpenStack environment. This document describes the actual tests that are run.

Murano environment with WordPress application deployment

The test verifies that the user can deploy the WordPress application in the Murano environment.

Target component: Murano

Scenario:

1. Send request to create environment.
2. Send request to create session for environment.

3. Send request to create MySQL.
4. Send request to create Linux-based service Apache.
5. Send request to create WordPress.
6. Request to deploy session.
7. Checking environment status.
8. Checking deployments status.
9. Checking ports availability.
10. Checking WordPress path.
11. Send request to delete environment.

For more information, see: Murano documentation

Murano environment with Linux Apache service deployment

The test verifies that the Murano service can create and deploy the Linux Apache service.

Target component: Murano

Scenario:

1. Check linux image with Murano agent installed in Glance.
2. Send request to create environment.
3. Send request to create session for environment.
4. Send request to create service Linux Apache.
5. Request to deploy session.
6. Checking environment status.
7. Checking deployments status.
8. Send request to delete environment.

For more information, see: Murano documentation

Troubleshooting Murano

There are no known issues with deploying Murano in Mirantis OpenStack at the time of this writing.

Maintenance Mode

Overview

Maintenance mode (MM) is the state of operating system when it has only critical set of working services which are needed for basic network and disk operations. Also node in MM is reachable with ssh from network.

You can put your system in *Maintenance Mode* for system repair or other service operations.

The maintenance mode is enforced by using the `umm` parameter in one of the following ways:

- by selecting the respective option in the boot menu;
- by forcing the reboot into maintenance mode from shell with the **umm on** command;
- automatically, by reaching an number of unclean-reboots specified in `REBOOT_COUNT` parameters.

Unclean reboot means that system reboots unexpectedly without a direct call from the user.

You can also disable the maintenance mode functionality if you do not need it (for example, you do not want to be automatically booted into it every time).

You can operate in maintenance mode through ssh or tty2.

A return back into normal mode is issued with the **umm off** command.

> ***Note***
>
> If you manually start a service in the maintenance mode, it will not be automatically restarted when you put the system back in the normal mode with the **umm off** command.

Using the umm command

There are several parameters to use with the **umm** command:

- `umm on [cmd]` - enter the maintenance mode, and execute cmd when MM is reached;
- `umm status` - check the mode status. There are three statuses:
  - `runlevel` - the system is in the normal mode.
  - `reboot` - the system is starting to enter the maintenance mode.
  - `umm` - the system is in the maintenance mode.
  - `disabled` - the maintenance mode functionality is disabled.
- `umm off [reboot]` - resume boot or reboot into normal mode.
- `umm enable` - enable the maintenance mode functionality.
- `umm disable` - disable the maintenance mode functionality.

Configuring the *UMM.conf* file

You can automate the maintenance mode start by editing the */etc/umm.conf* file.

The configuration options are:

- UMM=yes
- REBOOT_COUNT=2
- COUNTER_RESET_TIME=10

where:

UMM

tells the system to go into the maintenance mode based on the `REBOOT_COUNT` and `COUNTER_RESET_TIME` values. If the value is anything other than `yes` (or if the *UMM.conf* file is missing), the system will go into the native Ubuntu recovery mode.

REBOOT_COUNT

determines the number of unclean reboots that trigger the system to go into the maintenance mode.

COUNTER_RESET_TIME

determines the period of time (in minutes) before the *Unclean reboot* counter reset.

Example of using MM on one node

- Switching node into MM:

```
 1 root@node-1:~#umm on
 2 umm-gr start/running, process 6657
 3
 4 Broadcast message from root@node-1
 5 (/dev/pts/0) at 14:29 ...
 6
 7 The system is going down for reboot NOW!
 8 root@node-1:~# umm status
 9 rebooting
10 root@node-1:~# Connection to node-1 closed by remote host.
11 Connection node-1:~# closed.
12 root@fuel:~#:~$
13
14 root@node-1:~#ssh
15
16 root@node-1:~# umm status
17 umm
18 root@node-1:~#ps -Af
```

We can see only small set of working processes.

- Start the service:

```
1 root@node-1:~# /etc/init.d/apache2 start
2 root@node-1:~# /etc/init.d/apache2 status
3 Apache2 is running (pid 1907).
```

- Switch back to the working mode:

```
1 root@node-1:~#umm off
```

- Continue booting into working mode:

```
1 root@node-1:~#umm status
2 runlevel N 2
3 root@node-1:~#/etc/init.d/apache2 status
4 Apache2 is running (pid 1907).
```

We can see that service was not restarted during switching from MM to working mode.

- Check the state of the OpenStack services:

```
1 root@node-1:~#crm status
```

- If you want to reach working mode by reboot, you should use the following command:

```
1 root@node-1:~# umm off reboot umm-gr start/running, process 2825
2
3 Broadcast message from root@node-1
4 (/dev/pts/0) at 11:23 ...
5
6 The system is going down for reboot NOW!
7 root@node-1:~# Connection to node-1 closed by remote host.
8 Connection to node-1 closed.
9 [root@fuel ~]#
```

Example of putting all nodes into the maintenance mode at the same time

The following maintenance mode sequence is called *Last input First out*. This guarantees that there is going to be the most recent data on the Cloud Infrastructure Controller (CIC) that comes back first.

- Determine which nodes have Controller (CIC) role:

```
1 [root@fuel ~]# fuel nodes
2 id | status | name              | cluster| ip        | mac               | roles
3 ---|--------|-------------------|--------|-----------|-------------------|-----------
4 2  | ready  | Untitled (c0:02) | 1      | 10.20.0.4 | e6:6a:42:96:a4:45 | controller
5 4  | ready  | Untitled (c0:04) | 1      | 10.20.0.6 | 66:10:2e:0c:12:4a | compute
```

```
6 1  | ready  | Untitled (c0:01) | 1      | 10.20.0.3 | fa:a1:39:94:7f:4c | controller
7 3  | ready  | Untitled (c0:03) | 1      | 10.20.0.5 | 82:cb:bb:50:40:47 | controller
```

- Copy `id_rsa` to the CICs for passwordless ssh authentification:

```
[root@fuel ~]# scp .ssh/id_rsa node-1:.ssh/id_rsa
Warning: Permanently added 'node-1' (RSA) to the list of known hosts.
id_rsa                                        100% 1675   1.6KB/s   00:00
[root@fuel ~]# scp .ssh/id_rsa node-2:.ssh/id_rsa
Warning: Permanently added 'node-2' (RSA) to the list of known hosts.
id_rsa                                        100% 1675   1.6KB/s   00:00
[root@fuel ~]# scp .ssh/id_rsa node-3:.ssh/id_rsa
Warning: Permanently added 'node-3' (RSA) to the list of known hosts.
id_rsa                                        100% 1675   1.6KB/s   00:00
```

- Enforce switching into MM mode on all nodes:

```
[root@fuel ~]# ssh node-1 umm on ssh node-2 umm on ssh node-3 umm on
Warning: Permanently added 'node-1' (RSA) to the list of known hosts.
umm-gr start/running, process 24318
Connection to node-1 closed by remote host.
Connection to node-1 closed.
[root@fuel ~]#
[root@fuel ~]# ssh -tt node-1 ssh -tt node-2 ssh -tt node-3 sleep 1
Warning: Permanently added 'node-1' (RSA) to the list of known hosts.
ECDSA key fingerprint is 84:17:0d:ea:27:1f:4e:08:f7:54:b2:8c:fe:8a:13:1a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'node-2,10.20.0.4' (ECDSA)
to the list of known hosts. established.
ECDSA key fingerprint is
c3:c6:ca:7d:11:d3:53:01:15:64:20:f7:c7:44:fb:d1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'node-3,192.168.0.6' (ECDSA)
to the list of known hosts.
Connection to node-3 closed.
Connection to node-2 closed.
Connection to node-1 closed. [root@fuel ~]#
```

- Wait until the last node reboots:

```
[root@fuel ~]# ssh node-3
Warning: Permanently added 'node-3' (RSA) to the list of known hosts.
Welcome to Ubuntu 12.04.4 LTS (GNU/Linux 3.13.0-32-generic x86_64)
 * Documentation:  https://help.ubuntu.com/
Last login: Tue Dec 23 05:55:47 2014 from 10.20.0.2
root@node-3:~#
Broadcast message from root@node-3
```

```
 8 (unknown) at 6:00 ...
 9 The system is going down for reboot NOW!
10 Connection to node-3 closed by remote host.
11 Connection to node-3 closed.
12 [root@fuel ~]#
```

- Perform all the steps planned for MM.
- Enforce a return back into normal mode in reverse state:

```
1 [root@fuel ~]# ssh node-3 umm off
2 Warning: Permanently added 'node-3' (RSA) to the list of known hosts.
3 [root@fuel ~]# ssh node-2 umm off
4 Warning: Permanently added 'node-2' (RSA) to the list of known hosts.
5 [root@fuel ~]# ssh node-1 umm off
6 Warning: Permanently added 'node-1' (RSA) to the list of known hosts.
```

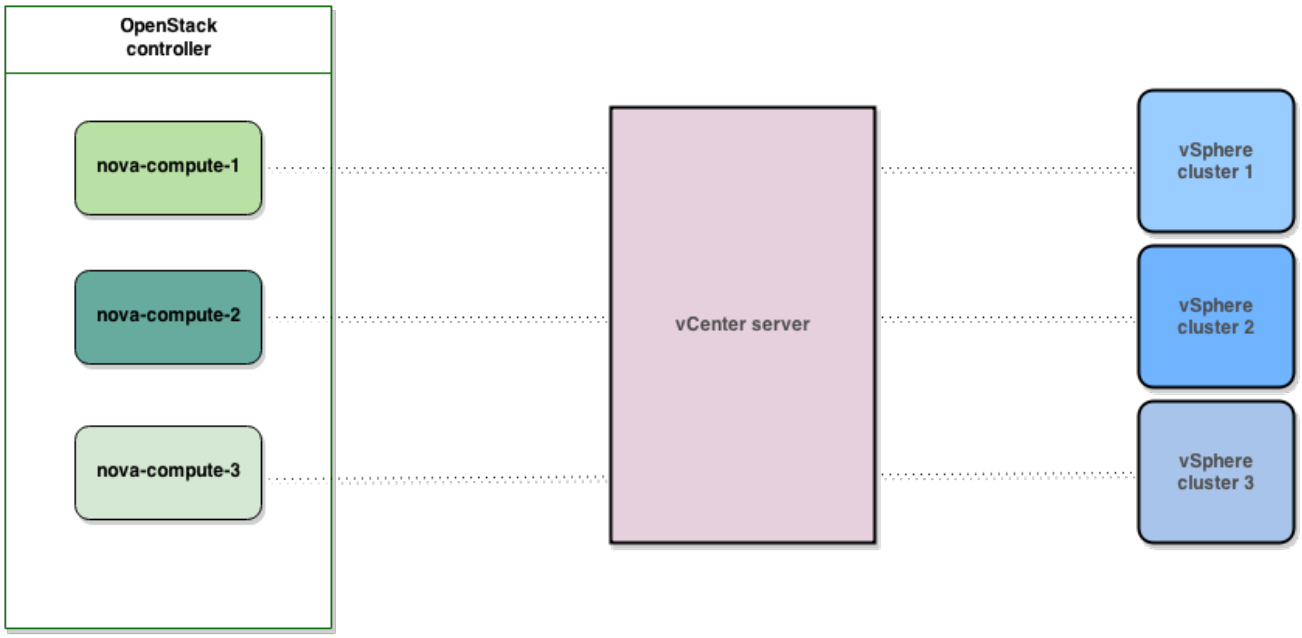
Running vCenter

After the OpenStack environment that is integrated with the vCenter server is deployed, you can manage the VMware cluster using the Horizon dashboard and/or the Nova CLI:

- Log into the Horizon dashboard.
- Open the Hypervisor tab and select the VMware vCenter Server.
- To boot a new VM in vCenter:
    - Open the "Manage Compute" tab and go to the Instances page.
    - Press the "Launch instance" button and specify the VM parameters in the pop-up window. This launches a new VM which can be seen in a vSphere web client.
- Use the Horizon UI or the Nova CLI to stop or delete booted VMs in the vCenter.
- Use the Nova CLI to see the VMware cluster resources or to boot a new VM in vCenter.

Nova-compute and vSphere clusters mapping

In earlier Fuel releases, 1-N mapping between nova-compute service and vSphere cluster (cluster that is formed from ESXi hosts by vCenter server) was used. In most cases, a single nova-compute service instance uses many vSphere clusters, managed by a single vCenter. Beginning with 6.1 Fuel release, this behaviour was changed to 1-1 mapping, so that a single nova-compute service instance now interacts with a single vSphere cluster.



Performance Notes

Keystone Token Cleanup

The *Keystone* service creates new tokens in the Keystone database each time an external query is run against OpenStack but it does not automatically clean up expired tokens because they may be required for forensics work such as that required after a security breach. However, the accumulation of the expired tokens in the database can seriously degrade the performance of the entire OpenStack.

Beginning with version 5.0, Mirantis OpenStack includes the pt-archiver command from the Percona Toolkit. We recommend using pt-archiver to set up a cleanup job that runs periodically; the cleanup-keystone-tokens.sh script from TripleO is a good example:

```
pt-archiver --source h=$DB_HOST,u=$DB_USER,p=$DB_PASS,D=$DB_NAME,t=token \
    --charset utf8 \
    --where "expires < UTC_TIMESTAMP()" \
    --purge \
    --txn-size 500 \
    --run-time 59m \
    --statistics \
    --primary-key-only
```

It is better to use pt-archiver instead of deleting the expired tokens using standard database manipulation commands because it prevents the Keystone database from being blocked for significant time periods while the rows with expired tokens are deleted.

HowTo: Backup and restore Fuel Master

Because the Fuel Master itself is not available with high availability, it is strongly recommended to create a backup after each deployment. This allows for recovery in the event of data corruption or hardware loss. The backup process requires the operator to place the backup on a remote location so that it survives a complete system failure. Note that this backup is only for the Fuel Master itself and not for OpenStack deployments. In order to be able to manage existing OpenStack environments deployed by Fuel, it is necessary to restore a backed up Fuel Master if a reinstall was performed.

You can back up your Fuel Master without downtime. What this means is Fuel API, Fuel UI, DHCP, DNS, and NTP services continue to operate during the backup process, causing no impact for any deployed or bootstrapped nodes.

In order to back up the Fuel Master, you need to meet these requirements:

- No deployment tasks are currently running
- You have at least 11GB free disk space

The backup contains the following items:

- All docker containers (including Fuel DB)
- PXE deployment configuration
- All OpenStack environment configurations
- Package repositories
- Deployment SSH keys
- Puppet manifests

Items not backed up include logs and host network configuration. If preserving log data is important, back up the */var/log* directory separately. This could be done by using *scp* to transfer */var/log* to another host. Network configuration needs to be done manually via Fuel Setup if you reinstall your Fuel Master before restoring it.

Running the backup

To start a backup, run **dockerctl backup**. Optionally, you can specify a path for backup. The default path is */var/backup/fuel*. This process takes approximately 30 minutes and is dependent on the performance of your hardware. After the backup is done, you may want to copy the backup to a separate storage medium.

> ***Note***
>
> If you make further changes to your environment after a backup, you should make a new backup.

Restoring Fuel Master

The restore is quite similar to the backup process. This process can be run any time after installing a Fuel Master node. Before starting a restore operation, ensure the following:

- The Fuel version is the same release as the backup
- There are no deployments running
- At least 11GB free space in */var*

If you reinstall your Fuel Master host, you need to configure your network settings via Fuel Setup the same way it was configured originally. It is particularly important that the configuration for the Admin (PXE) network is the same as before.

To run the restore, simply run **dockerctl restore /path/to/backup**.

How slave nodes choose the interface to use for PXE booting

Fuel configures the NIC name/order based on the data seen by the Nailgun agent (*/opt/nailgun/bin/agent*) on the discovered nodes. This is in turn the result of how the NICs are named/ordered by the bootstrap node.

The device used by the Admin (PXE) network will be the interface that is directly attached to this network. If one is not available, it will fall back to the interface with the default gateway.

For example:

Physical device	Interface	MAC
0	eth0	:FE:A0
1	eth1	:BC:6D
2	eth2	:E1:B2

If physical device 0 is connected to the Admin (PXE) network, then eth0 will be the admin interface in Fuel.

If instead physical device 1 is connected to the Admin (PXE) network, then eth1 will be the admin interface in Fuel.

A common issue here is that physical device 0, may not always be assigned device eth0. You may see:

Physical device	Interface	MAC
0	eth2	:FE:A0
1	eth0	:BC:6D
2	eth1	:E1:B2

In this case, having physical device 0 connected to the Admin (PXE) network will result in the eth2 interface being used as the admin interface in Fuel.

You can confirm that the right interface is in use because the MAC address did not change even though the device name did.

Running Ceilometer

Fuel can deploy the OpenStack Telemetry component *Ceilometer* to run in your OpenStack environment. When enabled, Ceilometer collects and shares measurement data gathered from all OpenStack components. This data can be used for monitoring and capacity planning purposes as well as for an alarming service. Ceilometer's REST API can also provide data to external monitoring software for a customer's billing system. See *Ceilometer and MongoDB* for information about the resources required to run Ceilometer.

For complete information about configuring and running Ceilometer, see Ceilometer Developer Documentation.

Configuring Ceilometer

Three types of measurement are defined:

- Cumulative -- increasing over time (instance hours)
- Gauge -- Discrete items (floating IPs, image uploads) and fluctuating values (disk I/O)
- Delta -- Changing over time (bandwidth)

For a complete list of meter types by component that are currently implemented, see http://docs.openstack.org/developer/ceilometer/measurements.html

For a complete list of Configuration Options, see http://docs.openstack.org/developer/ceilometer/configuration.html

Configuring Ceilometer for vCenter

The default Fuel deployment does not configure Ceilometer to collect metering information from vCenter. This also means that, if the vCenter installation is used with Heat, autoscaling does not work correctly because the alarms sent to Heat are implemented using meters.

To configure Ceilometer to collect metering information for vCenter:

Install **ceilometer-compute-agent** on the Controller node using the appropriate command:

```
apt-get install ceilometer-agent-compute (Debian)
```

Configure Ceilometer by adding the following lines to the *ceilometer.conf* file:

```
[DEFAULT]
default_log_levels=amqp=WARN,amqplib=WARN,boto=WARN,qpid=WARN, \
   sqlalchemy=WARN,suds=INFO,iso8601=WARN, \
   requests.packages.urllib3.connectionpool=WARN,oslo.vmware=WARN
hypervisor_inspector=vsphere

[vmware]
# See Note [1] below.

[alarm]
evaluation_interval=<Period of evaluation cycle> # See Note [2] below.
```

Note [1]: Configure according to the VMware documentation.

Note [2]: This value of <Period of evaluation cycle> should be >= than the configured pipeline interval for the collection of the underlying metrics.

You must also patch a known Icehouse bug with the VMware inspector. Use this fix.

When you have completed these steps, restart **ceilometer-agent-compute**, **ceilometer-alarm-evaluator** and **ceilometer-alarm-notifier**.

Performance and database backend

Ceilometer can be configured to collect a large amount of metering data and thus perform a high volume of database writes. For example, with 100 resources and default configs Ceilometer collects around 16k samples per hour.

Starting with version 5.0, Mirantis OpenStack defaults to installing *MongoDB* as the recommended back-end database for Ceilometer. The Fuel Master Node enables you to choose the installation of MongoDB as a role onto a node; see *Assign a role or roles to each node server* for instructions. This resolves the Ceilometer performance issues caused by the volume of concurrent read/write operations.

Preparing Ceilometer for Testing

The Platform Tests that are run as a part of the Health Tests test Ceilometer if the Environment is configured accordingly. Before running the test, please read the requirements below.

Note that the tests are run in the tenant that you specify in the *OpenStack Settings* tab during the OpenStack installation. By default, the *admin* tenant is used for the tests.

To run Ceilometer Platform Tests, you must have at least 6 GB of RAM on each controller node.

To run the "Ceilometer test to check notifications from Sahara" test, you must have at least 4096 MB of available RAM in the tenant you are using for the Ceilometer Platform Tests, otherwise the test may fail. Also you must get an image with Hadoop for Sahara and register it with Sahara. For instructions, see the Sahara deployment (Preparing Sahara for Testing).

To run the "Ceilometer test to check notifications from Nova" test, you must have at least 100 MB of available RAM in the tenant you are using for the Ceilometer Platform Tests, otherwise the test may fail.

If the requirements above are satisfied, Ceilometer is ready to be tested.

Heat Deployment Notes

Overview

Heat is the OpenStack Orchestration service. Its main goal is to implement a framework for managing the entire lifecycle of your infrastructure inside the OpenStack cloud. Heat uses human-readable templates to describe the deployment process of instances, but also supports AWS CloudFormation template format.

Heat Components

heat-api

This component provides a native REST API and processes API requests.

heat-api-cfn

This component is similar to *heat-api*, but provides AWS CloudFormation compatible API.

heat-engine

This is the main component of the Heat framework. It does all the work of reading templates, launching instances and providing events to the API users.

Installation

When using Fuel 4.1 and later, Heat is installed by default when you deploy your environment.

For more information:

- The official documentation of the Heat project
- Development documentation
- Mirantis blog record about Heat.

Details of Heat platform tests

Typical stack actions: create, update, delete, show details, etc

The test verifies that the Heat service can create, update and delete a stack and shows details of the stack and its resources, events and template.

Target component: Heat

Scenario:

1. Create a stack.
2. Wait for the stack status to change to 'CREATE_COMPLETE'.
3. Get the details of the created stack by its name.
4. Get the resources list of the created stack.

5. Get the details of the stack resource.
6. Get the events list of the created stack.
7. Get the details of the stack event.
8. Update the stack.
9. Wait for the stack to update.
10. Get the stack template details.
11. Get the resources list of the updated stack.
12. Delete the stack.
13. Wait for the stack to be deleted.

Check stack autoscaling

The test verifies that the Heat service can scale the stack capacity up and down automatically according to the current conditions.

Target component: Heat

Scenario:

1. Image with cfntools package should be imported.
2. Create a flavor.
3. Create a keypair.
4. Save the generated private key to a file on Controller node.
5. Create a security group.
6. Create a stack.
7. Wait for the stack status to change to 'CREATE_COMPLETE'.
8. Create a floating IP.
9. Assign the floating IP to the instance of the stack.
10. Wait for the cloud_init procedure to be completed on the instance.
11. Load the instance CPU to initiate the stack scaling up.
12. Wait for the second instance to be launched.
13. Release the instance CPU to initiate the stack scaling down.
14. Wait for the second instance to be terminated.
15. Delete the file with the private key.
16. Delete the stack.
17. Wait for the stack to be deleted.

Check stack rollback

The test verifies that the Heat service can rollback the stack if its creation failed.

Target component: Heat

Scenario:

1. Start stack creation with rollback enabled.
2. Verify the stack appears with status 'CREATE_IN_PROGRESS'.
3. Wait for the stack to be deleted as a result of the rollback after the expiration of the timeout defined in the WaitHandle resource of the stack.
4. Verify if the instance of the stack has been deleted.

Check advanced stack actions: suspend, resume

The test verifies that the Heat service can suspend and resume the stack.

Target component: Heat

Scenario:

1. Create a stack.
2. Wait until the stack status changes to 'CREATE_COMPLETE'.
3. Call stack suspend action.
4. Wait until the stack status changes to 'SUSPEND_COMPLETE'.
5. Call stack resume action.
6. Wail until the stack status changes to 'RESUME_COMPLETE'.
7. Call stack check action.
8. Wail until the stack status changes to 'CHECK_COMPLETE'.
9. Delete the stack and wait for the stack to be deleted.

Horizon Deployment Notes

Overview

Horizon is the OpenStack web dashboard which provides a web based user interface to OpenStack services including Nova, Swift, Keystone, etc.

Details of Health Checks

The Health Checks are run from the Fuel console. This section provides background details on the actual checks these tests perform.

Sanity tests description

Sanity checks work by sending a query to all OpenStack components to get a response back from them. Many of these tests are simple in that they ask each service for a list of their associated objects and then wait for a response. The response can be something, nothing, an error, or a timeout, so there are several ways to determine if a service is up. The following list includes the suite of sanity tests implemented:

- Instance list availability
- Images list availability
- Volume list availability
- Snapshots list availability
- Flavor list availability
- Limits list availability
- Services list availability
- User list availability
- Stack list availability
- Check if all the services execute normally
- Check Internet connectivity from a Compute node
- Check DNS resolution on a Compute node
- Murano environment and service creation, listing and deletion
- Networks availability
- Ceilometer meter, alarm and resource list availability

Functional tests description

Functional tests verify how your system handles basic OpenStack operations under normal circumstances. The Functional Test series gives you information about the speed of your environment and runs timeout tests.

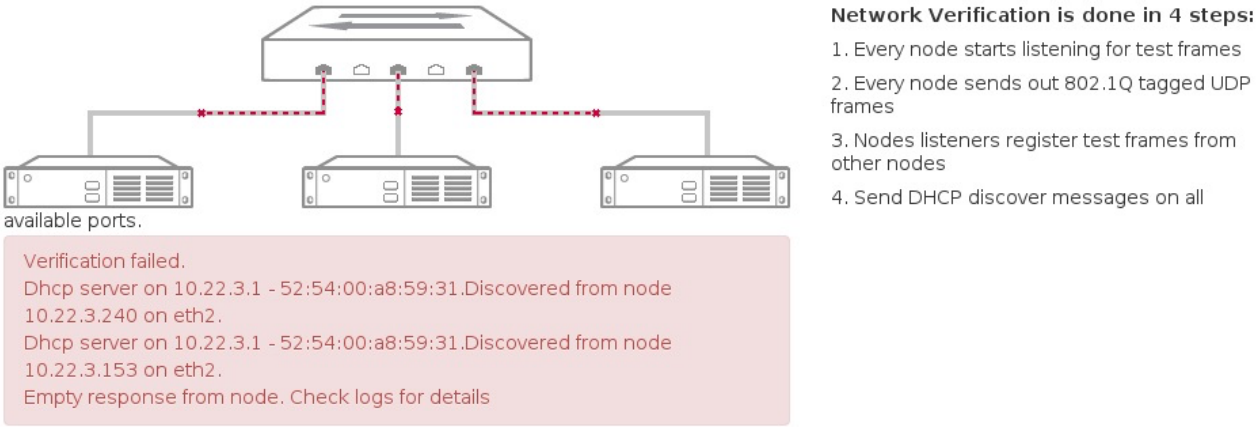
All tests use the basic OpenStack services (Nova, Glance, Keystone, Cinder, etc), so, if any of these are inactive, the test using it will fail. You should run all sanity checks before running the functional checks to verify that all services are alive. This helps ensure that you do not get false negatives. The following is a description of each sanity test available:

- Create instance flavor
- Create instance volume
- Launch instance, create snapshot, launch instance from snapshot

- Keypair creation
- Security group creation
- Check networks parameters
- Launch instance
- Assign floating IP
- Check that VM is accessible via floating IP address
- Check network connectivity from instance via floating IP
- Check network connectivity from instance without floating IP
- Launch instance with file injection
- Launch instance and perform live migration
- User creation and authentication in Horizon

Network Issues

Fuel has the built-in capability to run a network check before or after OpenStack deployment. The network check includes tests for connectivity between nodes via configured VLANs on configured host interfaces. Additionally, checks for an unexpected DHCP server are done to ensure outside DHCP servers will not interfere with deployment. The image below shows a sample result of the check. If there are errors, it is either in your configuration of interfaces or possibly the VLAN tagging feature is disabled on your switch port.



HA tests description

HA tests verify the High Availability (HA) architecture. The following is a description of each HA test available:

- Check data replication over MySQL
- Check if the amount of tables in OS databases is the same on each node
- Check Galera environment state
- RabbitMQ availability
- RabbitMQ replication
- Check Pacemaker status

Configuration tests description

Configuration tests verify if the default user data (e.g. username, password for OpenStack cluster) were changed. The following is a description of each test available:

- Check usage of the default credentials (password) for root user to SSH on the Fuel Master node. If the default password was not changed, the test will fail with a recommendation to change it.
- Check usage of the default credentials for OpenStack cluster. If the default values are used for the admin user, the test will fail with a recommendation to change the password/username for the OpenStack user with the admin role.

Cloud validation tests description

The following tests for cloud validation are implemented:

- Check disk space outage on the Controller and Compute nodes
- Check log rotation configuration on all nodes

Notes on Corosync and Pacemaker

Keep in mind the following on Corosync and Pacemaker:

- If you restart the Corosync service, you will need to restart the Pacemaker as well.
- Corosync 1.x cannot be upgraded to 2.x without full cluster downtime. All Pacemaker resources, such as Neutron agents, DB and AMQP clusters and others, will be hit by the downtime as well.
- All location constraints for cloned the Pacemaker resources must be removed as a part of the Corosync version upgrade procedure.

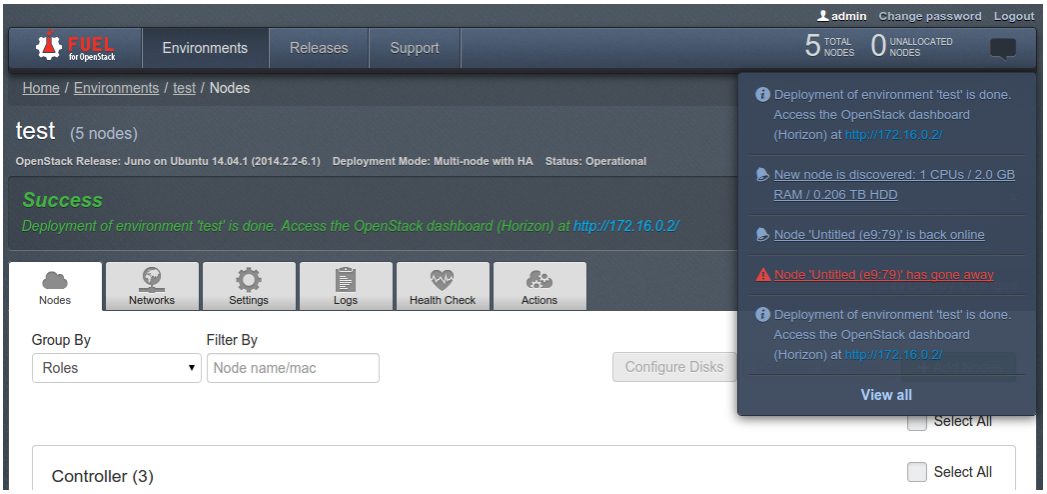
Troubleshooting

Logs and messages

A number of logs are available to help you understand and troubleshoot what is going on in your environment.

Screen messages

Clicking on the icon in the upper right corner of the Fuel UI displays a drop-down menu of notifications posted as the environment was deployed. If notifications have been posted that you have not yet viewed, this icon is an orange square with a number that indicates the number of unread notifications. If there are no unread notifications, the icon looks as it does here:

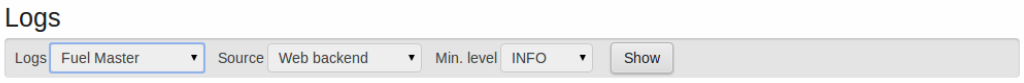


These notifications tell you about the nodes that were deployed and when nodes go on and offline. Click on any notification to get a summary configuration view of the node.

Viewing Logs through Fuel

The Logs tab on the Fuel UI allows you to view key logs for any node in the environment. Use the drop-down list on the left to select whether to view Fuel Master logs or the logs of other servers in the environment.

When you choose the "Fuel Master" logs, you get this display:



Select the log to view by setting the fields:

Logs: Choose between the Fuel Master or "Other servers". When you choose "Other servers", the display changes to provide a drop-down list of all nodes in the environment.

Source: Log you want to view. See the lists below.

Min Level: By default, this shows "INFO" messages only. If you are running your environment in debug mode, you can use this field to filter out some of the messages.

When you have set all the fields, click on "SHOW" at the right to display the requested log for the specified server.

Viewing Fuel Master logs

The following logs can be chosen from the "Source" list for the "Fuel Master" node:

puppet: Logs activity of the *Puppet* configuration management system.
anaconda: Logs activities of the Anaconda installation agent used for provisioning.
syslog: Shows the **syslog** entries that will be sent to the **rsyslog** server (Fuel Master by default).

Other install logs

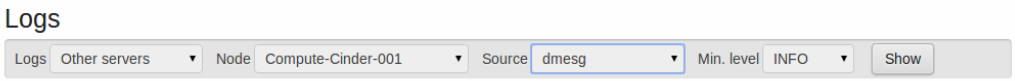
storage: Log entries for disk partitioning.
kickstart-pre: Shows activities before the *Cobbler* kickstart mechanism runs.
kickstart-post: Shows activities after the Cobbler kickstart mechanism runs.

Bootstrap logs

Web backend: Logs each connection from the Master Node to the Internet.
REST API: *Nailgun* API activities.
RPC consumer: Logs messaging between *Nailgun* and the *orchestration service*
Astute: Records the activity of the *Astute* agents that implements the *Nailgun* configuration tasks.
Health Check: Displays the results of the most recent run of the tests run from the "Health Check" tab.

Viewing logs for target nodes ("Other servers")

When you choose "Other Servers", the display includes an extra field that you use to choose the node whose logs you want to view:



Many of the same logs shown for the Fuel Master node are also shown for the target nodes. The difference is in the nodes given for "Bootstrap logs", plus the controller node includes a set of OpenStack logs that shows logs for services that run on the Controller node.

The "Bootstrap logs" for "Other servers" are:

Bootstrap logs

dmesg: Standard Linux dmesg log that displays log messages from the most recent system startup.
messages: Logs all kernel messages for the node.
mcollective: Logs activities of Mcollective
agent: Logs activities of the *Nailgun* agent.

syslog

OpenStack uses the standard Linux **syslog**/**rsyslog** facilities to manage logging in the environment. Fuel puts the appropriate templates into the */etc/rsyslog.d* directory on each target node.

By default, Fuel sets up the Fuel Master node to be the remote syslog server. See *Configuring syslog* for instructions about how to configure the environment to use a different server as the **rsyslog** server. Note that Fuel configures all the files required for **rsyslog** when you use the Fuel Master node as the remote server; if you specify another server, you must configure that server to handle messages from the OpenStack environment.

/var/logs

Logs for each node are written to the node's */var/logs* directory and can be viewed there. Under this directory, you will find subdirectories for the major services that run on that node such as nova, cinder, glance, and heat.

On the Fuel Master node, */var/log/remote* is a symbolic link to the */var/log/docker-logs/remote* directory.

Logging events from Fuel OCF agents are collected both locally in */var/log/daemon.log* and remotely. The naming convention for a remote log file is the following:

```
ocf-<AGENT-NAME>.log
```

For example:

```
ocf-foo-agent.log
```

> ***Note***
>
> RabbitMQ logs its OCF events to the *lrmd.log* file to keep it backwards compatible.

atop logs

Fuel installs and runs atop on all deployed nodes in the environment. The **atop** service uses screen to display detailed information about resource usage on each node. The data shows usage of the hardware resources that are most important from a performance standpoint: CPU, memory, disk, and network interfaces and can be used to troubleshoot performance and scalability issues.

The implementation is:

- By default, **atop** takes a snapshot of system status every 20 seconds and stores this information in binary form.
- The binary data is stored locally on each node in the */var/log/atop* directory.
- Data is kept for seven days; a logrotate job deletes logs older than seven days.
- Data is stored locally on each target node; it is not aggregated for the whole environment.
- The data consumes no more than 2GB of disk space on each node with the default configuration settings.

- The **atop** service can be disabled from the Linux shell. A Puppet run (either done manually or when patching OpenStack) re-enables **atop**.
- The Diagnostic Logs snapshot includes the *atop* data for the current day only.

To view the **atop** data, run the atop(1) command on the shell of the node you are analyzing:

```
atop -r /var/log/atop/<filename>
```

Use **t** and **T** to navigate through the state snapshots.

You can also search the **atop** data. For example, the following command reports on all **sh** processes in the **atop** binary file that ran between 10:00 and 12:00 (searching the *atop_current* binary file). For each process, the **PPRG** flag causes it to report when it started, when it ended, and the exit code provided:

```
atop -PPRG -r /var/log/atop/atop_current -b "10:00" -e "12:00" | grep 'sh'
```

See the atop(1) man page for a description of the **atop** options and commands.

Each target node has a configuration file that controls the behavior of **atop** on that node. On Ubuntu nodes, this is the */etc/default/atop* file. The contents of the file are:

```
INTERVAL=20
LOGPATH="/var/log/atop"
OUTFILE="$LOGPATH/daily.log"`
```

Modifying the value of the *INTERVAL* parameter or the **logrotate** settings affects the size of the logs maintained. For the most efficient log size, use a larger interval setting and a smaller rotate setting. To modify the rotate setting, edit the */etc/logrotate.d/atop* file and make both of the following modifications; the value of *X* should be the same in both cases:

- Modify the value of the **rotate X** setting.
- Modify the value of the **mtime +X** argument to the **lastaction** setting.

Fuel Master node log rotation

The Fuel Master node provides an ability to configure log rotation using the template file.

- By default, the logrotate script calls */etc/logrotate.d/fuel.nodaily* file every 15 minutes to verify whether one of the following conditions is met:
  - age - if the log has not been rotated in more than the specified period of time (weekly by default), and the file is larger than 10MB on the Fuel Master node or 5MB on slave nodes;
  - size - if a log file exceeds 100MB on the Fuel Master node or 20MB on slave nodes.

  > ***Warning***
  >
  > Fuel enforces the following global changes to the *logrotate* configuration: `delaycompress` and `copytruncate`.

- You can run a quick test to check if the logrotate script works.

  Find the biggest file in `/var/log/`, and check date time stamps in the first and the last line:

  ```
  biggest_file=$(find /var/log/ -type f | xargs du -h | sort -h | tail -n 1 | cut -f2);
  ls -lah $biggest_file;
  head -n1 $biggest_file | head -c35; echo "";
  tail -n1 $biggest_file | head -c35; echo "";
  ```

  If it is older than your rotation schedule and bigger than maxsize, then logrotate is not working correctly.

  To debug, type:

  ```
  logrotate -v -d /etc/logrotate.d/fuel.nodaily
  ```

  The output of this command is a list of files examined by *logrotate*, including whether they should be rotated or not.

- You can find an example of a writing rate evaluation for the `neutron-server` log file: LP1382515.
- When backporting the reworked logrotate configuration to the older Fuel releases, purge old template files:

  ```
  rm /etc/logrotate.d/{1,2}0-fuel*
  ```

  The */usr/bin/fuel-logrotate* script is needed as well as a new cron job to perform the rotation with it.

Enabling debug logging for OpenStack services

Most OpenStack services use the same configuration options to control log level and logging method. If you need to troubleshoot a specific service, locate its config file under */etc* (e.g. */etc/nova/nova.conf*) and revert the values of **debug** and **use_syslog** flags like this:

debug=True use_syslog=False

Disabling syslog will protect the Fuel master node from being overloaded from a flood of debug messages sent from that node to the rsyslog server. Do not forget to revert both flags back to original values when done with troubleshooting.

Fuel Master and Docker disk space troubleshooting

Overview

One major consideration in maintaining a Fuel Master node is managing disk space. While there is currently no monitoring for Fuel Master, it is important to budget enough disk space. Failure to do so may lead to logs overwhelming the /var partition. For example, enabling Ceilometer and debug logging will quickly fill up disk space. The following sections describe failures that may occur if the disk fills up and gives solutions for resolving them.

If the solution to your issue requires rebuilding Docker containers, take note that data recovery is not necessary for the following containers: mcollective, nginx, ostf, nailgun, rsyslog, keystone, rabbitmq. This is because the data on these containers is stateless. For astute, cobbler, and postgres containers it is necessary to recover stateful data from the affected container. Instructions below will guide you through the processes.

PostgreSQL database inconsistency

Diagnosis

The following symptoms will be present:

- Fuel Web UI fails to work
- The `dockerctl list -l` output reports that the nailgun, ostf, and/or keystone container is down
- The output of the `fuel task` command reports an error similar to the following:

```
HTTP Error 400: Bad Request (This Session's transaction has been rolled back
due to a previous exception during flush. To begin a new transaction with
this Session, first issue Session.rollback(). Original exception was:
(InternalError) index "notifications_pkey" contains unexpected zero page at
block 26
HINT: Please REINDEX it.
```

Solution

The postgres container should still be running, so you simply need to run an SQL command to correct the fault. Before attempting to fix the database, make a quick backup of it:

```
date=$(date --rfc-3339=date)
dockerctl shell postgres su - postgres -c 'pg_dumpall --clean' \
   > /root/postgres_backup_$(date +"%F-%T").sql
```

Now try to reindex nailgun database:

```
dockerctl shell postgres su - postgres -c \
"psql nailgun -S -c \"select pg_terminate_backend(pid) from pg_stat_activity \
where datname='nailgun';\""

dockerctl shell postgres su - postgres -c "psql nailgun -c 'reindex database nailgun;'"
```

Lastly, check proper function:

```
dockerctl check all
```

> ***Note***
>
> You may need to restart the postgres, keystone, nailgun, nginx or ostf Docker container using the **dockerctl restart CONTAINERNAME** command.

Docker metadata corruption loses containers

Diagnosis

The following symptoms will be present:

- Deployment fails for some reason
- One or more Docker containers is missing from docker ps -a
- /var/log/docker contains the following message:

```
Cannot start container fuel-core-6.1-postgres: Error getting container
273c9b19ea61414d8838772aa3aeb0f6f1b982a74555fb6631adb6232459fe80 from driver
devicemapper: Error writing metadata to
/var/lib/docker/devicemapper/devicemapper/.json325916422: write
/var/lib/docker/devicemapper/devicemapper/.json325916422: no space left on device
```

Solution

This solution requires data recovery, described in the Summary above. It is necessary to recover data manually using the **dmsetup** and **mount** commands.

First, you need the full UID of the docker container that was lost. In the log message above, we can see the ID is 273c9b19ea61414d8838772aa3aeb0f6f1b982a74555fb6631adb6232459fe80. If you are missing such a message, it can be found this way:

```
container=postgres
container_id=$(sqlite3 /var/lib/docker/linkgraph.db \
"select entity_id from edge where name like '%$container%'")
```

```
echo $container_id
#should look like:
273c9b19ea61414d8838772aa3aeb0f6f1b982a74555fb6631adb6232459fe80
```

Once you have the container ID, you need to get the devicemapper block device ID for the container:

```
device_id=$(python -c 'import sys; import json; input = json.load(sys.stdin);\
[sys.stdout.write(str(v["device_id"])) for k, v in input["Devices"].items() if \
k == sys.argv[1]]' "$container_id" < /var/lib/docker/devicemapper/devicemapper/json)
echo $device_id
```

Now activate the volume and mount it:

```
# Verify the your device_id and container variables are defined
echo $device_id
echo $container
pool=$(echo /dev/mapper/docker*pool)
dmsetup create "${container}_recovery" --table "0 20971520 thin $pool $device_id"
mkdir -p "/mnt/${container}_recovery"
mkdir -p "/root/${container}_recovery"
mount -t ext4 -o rw,relatime,barrier=1,stripe=16,data=ordered,discard \
   "/dev/mapper/${container}_recovery" "/mnt/${container}_recovery"
```

Verify that data is present in the mounted directory:

- for PostgreSQL:

  ```
  ls -la /mnt/${container}_recovery/rootfs/var/lib/pgsql/9.3/data/
  ```

- for Astute:

  ```
  ls -la /mnt/${container}_recovery/rootfs/var/lib/astute
  ```

- for Cobbler:

  ```
  ls -la /mnt/${container}_recovery/rootfs/var/lib/cobbler
  ```

Next, it is necessary to purge the container record from the Docker sqlite database. You may see an issue when running **dockerctl start CONTAINER**:

```
Abort due to constraint violation: constraint failed
```

Run this command before trying to restore the container data or if you are simply destroying and recreating it:

```
#Make a backup dump of docker sqlite DB
cp /var/lib/docker/linkgraph.db /root/linkgraph_$(date +"%F-%T").db
```

```
container_id=$(sqlite3 /var/lib/docker/linkgraph.db \
"select entity_id from edge where name like '%$container%'")
echo "Deleting container ID ${container_id}..."
sqlite3 /var/lib/docker/linkgraph.db "delete from entity where\
   id='${container_id}';delete from edge where entity_id='${container_id}';"
```

Now perform the following recovery actions, which vary depending on whether you need to recover data from Cobbler, Astute, or PostgreSQL:

For Cobbler:

```
cp -Rp /mnt/cobbler_recovery/rootfs/var/lib/cobbler /root/cobbler_recovery
dockerctl destroy cobbler
dockerctl start cobbler
dockerctl copy "/root/cobbler_recovery/*" cobbler:/var/lib/cobbler/
dockerctl restart cobbler
```

For PostgreSQL:

```
cp -Rp /mnt/postgres_recovery/rootfs/var/lib/pgsql /root/postgres_recovery
dockerctl destroy postgres
dockerctl start postgres
dockerctl shell postgres mv /var/lib/pgsql /root/pgsql_old
dockerctl copy /root/postgres_recovery/pgsql postgres:/var/lib/
dockerctl shell postgres chown -R postgres:postgres /var/lib/pgsql
dockerctl restart postgres keystone nailgun nginx ostf
```

You may want to make a PostgreSQL backup at this point:

```
dockerctl shell postgres su - postgres -c 'pg_dumpall --clean' \
      > /root/postgres_backup_$(date +"%F-%T").sql
```

To recover a corrupted PostgreSQL database, you can import the dump to another PostgreSQL installation with the same version, as on fuel master(in 6.0 it is 9.3.5) There you can get a clean dump that you then import to your PostgreSQL container:

```
yum install postgresql-server
cp -rf data/ /var/lib/pgsql/
service postgresql start
su - postgres -c 'pg_dumpall --clean' > dump.sql
service postgresql stop
```

Now import the *dump.sql* file to the postgres container's database:

```
dockerctl shell postgres su - postgres -c "psql nailgun" < dump.sql
```

For Astute:

```
cp -Rp /mnt/astute_recovery/var/lib/astute /root/astute_recovery
dockerctl destroy astute
dockerctl start astute
dockerctl copy "/var/lib/astute/*" astute:/var/lib/astute/
dockerctl restart astute
```

Finally, clean up the recovery mount point:

```
umount "/mnt/${container}_recovery"
dmsetup clear $device_id
```

Read-only containers

Symptoms

- Fuel Web UI does not work
- Fuel CLI fails to report any commands
- Some containers may be failing and stopped
- Trying to run **dockerctl shell CONTAINER touch /root/test** results in "Read-only filesystem" error

Solution

Because of bugs in docker-io 0.10, the only way to correct this issue is to restart the Fuel Master node. If it still fails with the same issue, you may have a corrupt filesystem. See the next section for more details.

Corrupt ext4 filesystem on Docker container

Symptoms

Error:

```
Cannot start container fuel-core-6.1-rsync: Error getting container
df5f1adfe6858a13b0a9fe81217bf7db33d41a3d4ab8088d12d4301023d4cca3 from driver
devicemapper: Error mounting
'/dev/mapper/docker-253:2-341202-df5f1adfe6858a...d41a3d4ab8088d12d4301023d4cca3'
on
'/var/lib/docker/devicemapper/mnt/df5f1adfe6858a...d41a3d4ab8088d12d4301023d4cca3':
invalid argument
```

Solution

If the container affected is stateful, it is necessary to recover the data. Otherwise, you can simply destroy and recreate stateless containers.

For stateless containers:

```
container="rsync" # Change container name
dockerctl destroy $container
dockerctl start $container
```

For stateful containers:

```
container_id=$(sqlite3 /var/lib/docker/linkgraph.db \
"select entity_id from edge where name like '%$container%'")
echo $container_id
umount -l /dev/mapper/docker-*$container_id
fsck -y /dev/mapper/docker-*$container_id
dockerctl start $container
```

How To Troubleshoot Corosync/Pacemaker

Pacemaker and Corosync come with several CLI utilities that can help you troubleshoot and understand what is going on.

> ***Note***
>
> In Mirantis OpenStack 6.0 and later, multiple *L3 agents* are configured as clones, one on each Controller. When troubleshooting Corosync and Pacemaker, the **clone_p_neutron-l3-agent** resource (new in 6.0) is used to act on all L3 agent clones in the environment. The **p_neutron-l3-agent** resource is still provided, to act on a specific resource on a specific Controller node;

crm - Cluster Resource Manager

The **crm** utility shows you the state of the *Pacemaker* cluster and can be used for maintenance and to analyze whether the cluster is consistent. This section discusses some frequently-used commands.

crm status

This command shows you the main information about the Pacemaker cluster and the state of the resources being managed.

On the conrtoller node run:

```
root@node-1:~# crm
```

Then run the **status** command:

```
crm(live)#
crm(live)# status
```

Example of the output for the above command:

```
============
Last updated: Tue Jun 23 08:47:23 2015
Last change: Mon Jun 22 17:24:32 2015
Stack: corosync
Current DC: node-1.domain.tld (1) - partition with quorum
Version: 1.1.12-561c4cf
3 Nodes configured
43 Resources configured
============

Online: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]

Clone Set: clone_p_vrouter [p_vrouter]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
vip__management (ocf::fuel:ns_IPaddr2): Started node-1.domain.tld
vip__public_vrouter (ocf::fuel:ns_IPaddr2): Started node-1.domain.tld
vip__management_vrouter (ocf::fuel:ns_IPaddr2): Started node-1.domain.tld
vip__public (ocf::fuel:ns_IPaddr2): Started node-2.domain.tld
Master/Slave Set: master_p_conntrackd [p_conntrackd]
    Masters: [ node-1.domain.tld ]
    Slaves: [ node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_haproxy [p_haproxy]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_dns [p_dns]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_mysql [p_mysql]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Master/Slave Set: master_p_rabbitmq-server [p_rabbitmq-server]
    Masters: [ node-1.domain.tld ]
    Slaves: [ node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_heat-engine [p_heat-engine]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-plugin-openvswitch-agent [p_neutron-plugin-openvswitch-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-dhcp-agent [p_neutron-dhcp-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-metadata-agent [p_neutron-metadata-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-l3-agent [p_neutron-l3-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_ntp [p_ntp]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_ping_vip__public [ping_vip__public]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
```

Or run the **crm status** command on the controller node:

```
root@node-1:~# crm status
```

Example of the output for the above command:

```
============
Last updated: Tue Jun 23 08:47:52 2015
Last change: Mon Jun 22 17:24:32 2015
Stack: corosync
Current DC: node-1.domain.tld (1) - partition with quorum
Version: 1.1.12-561c4cf
3 Nodes configured
43 Resources configured
============

Online: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]

Clone Set: clone_p_vrouter [p_vrouter]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
vip__management (ocf::fuel:ns_IPaddr2): Started node-1.domain.tld
vip__public_vrouter (ocf::fuel:ns_IPaddr2): Started node-1.domain.tld
vip__management_vrouter (ocf::fuel:ns_IPaddr2): Started node-1.domain.tld
vip__public (ocf::fuel:ns_IPaddr2): Started node-2.domain.tld
Master/Slave Set: master_p_conntrackd [p_conntrackd]
    Masters: [ node-1.domain.tld ]
    Slaves: [ node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_haproxy [p_haproxy]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_dns [p_dns]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_mysql [p_mysql]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Master/Slave Set: master_p_rabbitmq-server [p_rabbitmq-server]
    Masters: [ node-1.domain.tld ]
    Slaves: [ node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_heat-engine [p_heat-engine]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-plugin-openvswitch-agent [p_neutron-plugin-openvswitch-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-dhcp-agent [p_neutron-dhcp-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-metadata-agent [p_neutron-metadata-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-l3-agent [p_neutron-l3-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_ntp [p_ntp]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
```

```
Clone Set: clone_ping_vip__public [ping_vip__public]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
```

crm(live)# resource

Here you can enter resource-specific commands:

```
root@node-1:~# crm
crm(live)# resource
crm(live)resource# status
```

Example of the output for the above command:

```
Clone Set: clone_p_vrouter [p_vrouter]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
vip__management        (ocf::fuel:ns_IPaddr2): Started
vip__public_vrouter    (ocf::fuel:ns_IPaddr2): Started
vip__management_vrouter        (ocf::fuel:ns_IPaddr2): Started
vip__public    (ocf::fuel:ns_IPaddr2): Started
Master/Slave Set: master_p_conntrackd [p_conntrackd]
    Masters: [ node-1.domain.tld ]
    Slaves: [ node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_haproxy [p_haproxy]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_dns [p_dns]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_mysql [p_mysql]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Master/Slave Set: master_p_rabbitmq-server [p_rabbitmq-server]
    Masters: [ node-1.domain.tld ]
    Slaves: [ node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_heat-engine [p_heat-engine]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-plugin-openvswitch-agent [p_neutron-plugin-openvswitch-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-dhcp-agent [p_neutron-dhcp-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-metadata-agent [p_neutron-metadata-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_neutron-l3-agent [p_neutron-l3-agent]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_p_ntp [p_ntp]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
Clone Set: clone_ping_vip__public [ping_vip__public]
    Started: [ node-1.domain.tld node-2.domain.tld node-3.domain.tld ]
```

crm(live)resource# start|restart|stop|cleanup <resource_name>

These commands, in order, allow you to start, stop, and restart resources.

crm(live)resource#cleanup

The cleanup command resets a resource's state on the node if it is currently in a failed state because of some unexpected operation, such as some side effects of a System V **init** operation on the resource. If this happens, Pacemaker can do the clean-up, deciding which node will run the resource.

Example:

```
3 Nodes configured, 3 expected votes
3 Resources configured.
============

3 Nodes configured, 3 expected votes
16 Resources configured.


Online: [ controller-01 controller-02 controller-03 ]

 vip__management      (ocf::heartbeat:IPaddr2):         Started controller-01
 vip__public          (ocf::heartbeat:IPaddr2):         Started controller-02
 Clone Set: clone_p_haproxy [p_haproxy]
  Started: [ controller-01 controller-02 controller-03 ]
 Clone Set: clone_p_mysql [p_mysql]
  Started: [ controller-01 controller-02 controller-03 ]
 Clone Set: clone_p_neutron-openvswitch-agent [p_neutron-openvswitch-agent]
  Started: [ controller-01 controller-02 controller-03 ]
 Clone Set: clone_p_neutron-metadata-agent [p_neutron-metadata-agent]
  Started: [ controller-01 controller-02 controller-03 ]
 Clone Set: clone_p_neutron-metadata-agent [p_neutron-dhcp-agent]
  Started: [ controller-01 controller-02 controller-03 ]
 p_neutron-dhcp-agent   (ocf::mirantis:neutron-agent-dhcp): Started controller-01
 Clone Set: clone_p_neutron-l3-agent [p_neutron-l3-agent]
  Started: [ controller-01 controller-02 controller-03 ]
```

In this case, **crm** found residual OpenStack agent processes that had been started by Pacemaker because of network failure and cluster partitioning. After the restoration of connectivity, Pacemaker saw these duplicate resources running on different nodes. You can let it clean up this situation automatically or, if you do not want to wait, cleanup them manually.

For more information, see crm interactive help and documentation.

Sometimes a cluster gets split into several parts. In this case, `crm status` shows something like this:

```
On ctrl1
============
….
Online: [ ctrl1 ]
```

```
On ctrl2
============
….
Online: [ ctrl2 ]

On ctrl3
============
….
Online: [ ctrl3 ]
```

You can troubleshoot this by checking connectivity between nodes. Look for the following:

1. By default Fuel configures corosync over UDP. Security Appliances shouldn't block UDP traffic for 5404, 5405 ports. Deep traffic inspection should be turned off for these ports. These ports should be accepted on the management network between all controllers.
2. Corosync should start after the network interfaces are activated.
3. *bindnetaddr* should be located in the management network or at least in the same reachable segment.

corosync-cfgtool -s

This command displays the cluster connectivity status.:

```
Printing ring status.
Local node ID 50490378
RING ID 0
      id      = 10.107.0.8
      status  = ring 0 active with no faults
```

FAULTY status indicates connectivity problems.

corosync-objctl

This command can get/set runtime Corosync configuration values including the status of Corosync redundant ring members:

```
runtime.totem.pg.mrp.srp.members.134245130.ip=r(0) ip(10.107.0.8)
runtime.totem.pg.mrp.srp.members.134245130.join_count=1
...
runtime.totem.pg.mrp.srp.members.201353994.ip=r(0) ip(10.107.0.12)
runtime.totem.pg.mrp.srp.members.201353994.join_count=1
runtime.totem.pg.mrp.srp.members.201353994.status=joined
```

If the IP of the node is 127.0.0.1, it means that Corosync started when only the loopback interface was available and bound to it.

If the members list contains only one IP address or is incomplete, it indicates that there is a Corosync connectivity issue because this node does not see the other ones.

As **no-quorum-policy** is set to **stop** on fully functioning cluster, Pacemaker will stop all resources on quorumless partition. If quorum is present, the cluster will function normally, allowing to drop minor set of controllers. This eliminates split-brain scenarios where nodes doesn't have quorum or can't see each other.

In some scenarios, such as manual cluster recovery, **no-quorum-policy** can be set to **ignore**. This setting allows operator to start operations on single controller rather than waiting for for quorum.

```
pcs property set no-quorum-policy=ignore
```

Once quorum or cluster is restored, **no-quorum-policy** should be set back to its previous value.

Also, Fuel temporarily sets **no-quorum-policy** to **ignore** when Cloud Operator adds/removes a controller node to the cluster. This is required for scenarios when Cloud Operator adds more controller nodes than the cluster currently consist of. Once addition/removal of new controller node is done, Fuel sets **no-quorum-policy** to **stop** value.

It's also recommended to configure fencing (STONITH) for Pacemaker cluster. That could be done manually or with help of Fencing plugin[1]_ for Fuel. When STONITH enabled, **no-quorum-policy** could be set to **suicide** as well. When set to **suicide**, the node will shoot itself and any other nodes in the partition without quorum - but it won't try to shoot the nodes it can't see. When set to **ignore** (or when it has quorum), it will shoot anyone it can't see. For any other value, it won't shoot anyone when it doesn't have quorum.

Furthermore, Corosync will always try to automatically restore the cluster back into single partition and start all of the resources, if any were stopped, unless some controller nodes are damaged (cannot run the Corosync service for example). Such nodes cannot join back the cluster and must be fenced by the STONITH daemon. That is why production cluster should always have a fencing enabled.

How to verify that Neutron HA is working

To verify that Neutron HA is working, simply shut down the node hosting the Neutron agents (either gracefully or with a hard shutdown). You should see agents start on the other node:

and see corresponding Neutron interfaces on the new Neutron node:

```
# ip link show

11: tap7b4ded0e-cb: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
12: qr-829736b7-34: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
13: qg-814b8c84-8f: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
```

You can also check `ovs-vsctl show output` to see that all corresponding tunnels/bridges/interfaces are created and connected properly:

```
ce754a73-a1c4-4099-b51b-8b839f10291c
  Bridge br-mgmt
      Port br-mgmt
          Interface br-mgmt
              type: internal
      Port "eth1"
          Interface "eth1"
```

```
  Bridge br-ex
      Port br-ex
          Interface br-ex
              type: internal
      Port "eth0"
          Interface "eth0"
      Port "qg-814b8c84-8f"
          Interface "qg-814b8c84-8f"
              type: internal
  Bridge br-int
      Port patch-tun
          Interface patch-tun
              type: patch
              options: {peer=patch-int}
      Port br-int
          Interface br-int
              type: internal
      Port "tap7b4ded0e-cb"
          tag: 1
          Interface "tap7b4ded0e-cb"
              type: internal
      Port "qr-829736b7-34"
          tag: 1
          Interface "qr-829736b7-34"
              type: internal
  Bridge br-tun
      Port "gre-1"
          Interface "gre-1"
              type: gre
              options: {in_key=flow, out_key=flow, remote_ip="10.107.0.8"}
      Port "gre-2"
          Interface "gre-2"
              type: gre
              options: {in_key=flow, out_key=flow, remote_ip="10.107.0.5"}
      Port patch-int
          Interface patch-int
              type: patch
              options: {peer=patch-tun}
      Port "gre-3"
          Interface "gre-3"
              type: gre
              options: {in_key=flow, out_key=flow, remote_ip="10.107.0.6"}
      Port "gre-4"
          Interface "gre-4"
              type: gre
              options: {in_key=flow, out_key=flow, remote_ip="10.107.0.7"}
      Port br-tun
          Interface br-tun
```

```
            type: internal
ovs_version: "1.4.0+build0"
```

Corosync crashes without network connectivity

Depending on a wide range of systems and configurations in the network, it is possible for Corosync's networking protocol, Totem, to time out. If this happens for an extended period of time, Corosync may crash. In addition, MySQL may have stopped. This guide illustrates the process of working through issues with Corosync and MySQL.

Workaround:

1. Verify that Corosync is really broken:

   ```
   service corosync status
   ```

   - You should see this error:

     ```
     corosync dead but pid file exists
     ```

2. Start Corosync manually:

   ```
   service corosync start
   ```

3. Run the following command:

   ```
   ps -ef | grep mysql
   ```

   and kill ALL(!) **mysqld** and **mysqld_safe** processes.

4. Wait for Pacemaker to completely start MySQL processes.

   - Check it with the following command:

     ```
     ps -ef | grep mysql
     ```

   - If it doesn't start, run:

     ```
     crm resource p_mysql
     ```

1 Fencing plugin

5. To verify that this host is a member of the cluster and that **p_mysql** does not contain any "Failed actions", run the following command:

```
crm status
```

How To Troubleshoot AMQP issues

AMQP is the heart of OpenStack. If something gets wrong with the messaging layer, normally an OpenStack application can tolerate this by issuing reconnects, reporting some requests as failed or retrying them. While this depends on the particular application and the underlying Oslo messaging library design, there are also some generic health checks and troubleshooting steps for operators to know and to do.

Check if there is a problem in the Corosync/Pacemaker layer

Normally, failures of the RabbitMQ multistate resource are automatically fixed by Corosync and Pacemaker. But if a failure cannot be automatically fixed for some reason, the master_p_rabbitmq-server resource status will be considered "bad". The status is considered "good" when there exists a single master for the master_p_rabbitmq-server and the rest of the resource instances are reported as slaves. Note, that the command `pcs status` issued on all controllers should be enough to gather all the required information, see *crm - Cluster Resource Manager*. You can also try issuing an extended output with the crm tool alternative:

```
crm_mon -fotAW -1
```

If there are some RabbitMQ resource failures, they will be shown in the command output with the time stamps, so you can search for the events in the logs around that moment.

> ***Note***
>
> If there are split clusters of Corosync running, you should first fix your Corosync cluster, because you cannot resolve issues with the Pacemaker resources, including RabbitMQ cluster, when there is a split brain in the Corosync cluster.

How to recover

It is recommended to clean up and restart the master_p_rabbitmq-server Pacemaker resource, see *crm - Cluster Resource Manager*.

> ***Note***
>
> Restarting the RabbitMQ Pacemaker resource will introduce a full downtime for the AMQP cluster and OpenStack applications. The downtime may take from a few to up to 20 minutes.

Check if there is a problem in the RabbitMQ layer

Normally, failures of the RabbitMQ cluster are automatically healed by OCF resource agents in Pacemaker. But if this fails for some reason, there may be unsynchronized queues, wrong cluster membership reported, or

partitions detected by the rabbitmqctl tool, or even some list channels/queues requests may hang. Note, that the command `rabbitmqctl report` issued on all controllers should be enough to gather all the required information, but there is also a special group of Fuel OSTF HA health checks available in the Fuel UI and CLI. See also *RabbitMQ OSTF replication tests <https://blueprints.launchpad.net/fuel/+spec/ostf-rabbit-replication-tests>*.

How to recover

It is recommended to clean up and restart the master_p_rabbitmq-server Pacemaker resource, see *crm - Cluster Resource Manager*.

Check if there is a problem in the Oslo messaging layer

Note, that normally an OpenStack application should be able to reconnect the AMQP host and restore its operations, eventually. But if it cannot for some reason, there may be "down" status reports or failures of CLI commands and messaging related or publish/consume message related records in the log files of the OpenStack services relying on the Oslo messaging library. For example, there may be records in the log files similar to the following ones:

```
Timed out waiting for a reply to message...
```

How to recover

It is recommended to restart the affected OpenStack service or services, see *HowTo: Manage OpenStack services*.

> ***Note***
>
> Restarting the OpenStack service will introduce a short (near to zero) downtime for the related OpenStack application.

Check if there are AMQP problems with any of the OpenStack components

Note, that normally an OpenStack application should be able to reconnect the AMQP host and restore its operations, eventually. But if it cannot for some reason, there may be "down" status reports or failures of CLI commands and AMQP/messaging related records in the log files of the services belonging to the affected OpenStack component under verification. For Nova, for example, there may be records in the log files in the /var/log/nova/ directory similar to the following ones:

```
AMQP server on ... is unreachable: [Errno 113] EHOSTUNREACH...
```

How to recover

It is recommended to restart all instances of the OpenStack services related to the affected OpenStack component, see *HowTo: Manage OpenStack services*. For example, for Nova Compute component, you may want to restart all instances of the Nova services on the Controllers and Compute nodes affected by the AMQP issue.

How to make RabbitMQ OCF script tolerate rabbitmqctl timeouts

If on a node where RabbitMQ is deployed other processes consume a significant part of CPU, RabbitMQ starts responding slow to queries from `rabbitmqctl` utility. The utility is used by the OCF script to monitor the state of the RabbitMQ. When the utility fails to complete in a pre-defined period of time, the OCF script considers the RabbitMQ being down and restarts it, which might lead to a several minutes cloud downtime. Such restarts are undesirable, as they cause a downtime without any benefit. To mitigate the issue, you can configure the OCF script to tolerate certain amount of `rabbitmqctl` timeouts in a row using the following command:

```
crm_resource --resource p_rabbitmq-server --set-parameter \
    max_rabbitmqctl_timeouts --parameter-value N
```

Replace N with the number of timeouts. For instance, if it is set to *3*, the OCF script will tolerate two rabbitmqctl timeouts in a row, but fail if the third one occurs.

By default, the parameter is set to *1*, which means `rabbitmqctl` timeout is not tolerated at all. The downside of increasing the parameter is a delay in restarting RabbitMQ when it is down. For example, if a real issue occurs and causes a `rabbitmqctl` timeout, the OCF script will detect that only after N monitor runs and then restart RabbitMQ, which might fix the issue.

To understand that RabbitMQ's restart was caused by a `rabbitmqctl` timeout, you should examine *lrmd.log* of the corresponding Controller on the Fuel Master node in */var/log/docker-logs/remote/ directory* for the presence of the following lines:

```
"the invoked command exited 137: /usr/sbin/rabbitmqctl list_channels ..."
```

This indicates a rabbitmqctl timeout. The next line will explain if it caused restart or not. For example:

```
"rabbitmqctl timed out 2 of max. 3 time(s) in a row. Doing nothing for now."
```

Timeout In Connection to OpenStack API From Client Applications

If you use Java, Python or any other code to work with OpenStack API, all connections should be done over OpenStack Public network. To explain why we can not use Fuel network, let's try to run nova client with debug option enabled:

```
[root@controller-6 ~]# nova --debug list

REQ: curl -i http://192.168.0.2:5000/v2.0/tokens -X POST -H "Content-Type: appli
cation/json" -H "Accept: application/json" -H "User-Agent: python-novaclient" -d
'{"auth": {"tenantName": "admin", "passwordCredentials": {"username": "admin",
"password": "admin"}}}'

INFO (connectionpool:191) Starting new HTTP connection (1): 192.168.0.2
DEBUG (connectionpool:283) "POST /v2.0/tokens HTTP/1.1" 200 2702
RESP: [200] {'date': 'Tue, 06 Aug 2013 13:01:05 GMT', 'content-type': 'applicati
```

```
on/json', 'content-length': '2702', 'vary': 'X-Auth-Token'}
RESP BODY: {"access": {"token": {"issued_at": "2013-08-06T13:01:05.616481", "exp
ires": "2013-08-07T13:01:05Z", "id": "c321cd823c8a4852aea4b870a03c8f72", "tenant
": {"description": "admin tenant", "enabled": true, "id": "8eee400f7a8a4f35b7a92
bc6cb54de42", "name": "admin"}}, "serviceCatalog": [{"endpoints": [{"adminURL":
"http://192.168.0.2:8774/v2/8eee400f7a8a4f35b7a92bc6cb54de42", "region": "Region
One", "internalURL": "http://192.168.0.2:8774/v2/8eee400f7a8a4f35b7a92bc6cb54de4
2", "id": "6b9563c1e37542519e4fc601b994f980", "publicURL": "http://172.16.1.2:87
74/v2/8eee400f7a8a4f35b7a92bc6cb54de42"}], "endpoints_links": [], "type": "compu
te", "name": "nova"}, {"endpoints": [{"adminURL": "http://192.168.0.2:8080", "re
gion": "RegionOne", "internalURL": "http://192.168.0.2:8080", "id": "4db0e11de35
74c889179f499f1e53c7e", "publicURL": "http://172.16.1.2:8080"}], "endpoints_link
s": [], "type": "s3", "name": "swift_s3"}, {"endpoints": [{"adminURL": "http://1
92.168.0.2:9292", "region": "RegionOne", "internalURL": "http://192.168.0.2:9292
", "id": "960a3ad83e4043bbbc708733571d433b", "publicURL": "http://172.16.1.2:929
2"}], "endpoints_links": [], "type": "image", "name": "glance"}, {"endpoints": [
{"adminURL": "http://192.168.0.2:8776/v1/8eee400f7a8a4f35b7a92bc6cb54de42", "reg
ion": "RegionOne", "internalURL": "http://192.168.0.2:8776/v1/8eee400f7a8a4f35b7
a92bc6cb54de42", "id": "055edb2aface49c28576347a8c2a5e35", "publicURL": "http://
172.16.1.2:8776/v1/8eee400f7a8a4f35b7a92bc6cb54de42"}], "endpoints_links": [], "
type": "volume", "name": "cinder"}, {"endpoints": [{"adminURL": "http://192.168.
0.2:8773/services/Admin", "region": "RegionOne", "internalURL": "http://192.168.
0.2:8773/services/Cloud", "id": "1e5e51a640f94e60aed0a5296eebdb51", "publicURL":
"http://172.16.1.2:8773/services/Cloud"}], "endpoints_links": [], "type": "ec2"
, "name": "nova_ec2"}, {"endpoints": [{"adminURL": "http://192.168.0.2:8080/",
"region": "RegionOne", "internalURL": "http://192.168.0.2:8080/v1/AUTH_8eee400f
7a8a4f35b7a92bc6cb54de42", "id": "081a50a3c9fa49719673a52420a87557", "publicURL
": "http://172.16.1.2:8080/v1/AUTH_8eee400f7a8a4f35b7a92bc6cb54de42"}], "endpoi
nts_links": [], "type": "object-store", "name": "swift"}, {"endpoints": [{"admi
nURL": "http://192.168.0.2:35357/v2.0", "region": "RegionOne", "internalURL": "
http://192.168.0.2:5000/v2.0", "id": "057a7f8e9a9f4defb1966825de957f5b", "publi
cURL": "http://172.16.1.2:5000/v2.0"}], "endpoints_links": [], "type": "identit
y", "name": "keystone"}], "user": {"username": "admin", "roles_links": [], "id"
: "717701504566411794a9cfcea1a85c1f", "roles": [{"name": "admin"}], "name": "ad
min"}, "metadata": {"is_admin": 0, "roles": ["90a1f4f29aef48d7bce3ada631a54261"
]}}}

REQ: curl -i http://172.16.1.2:8774/v2/8eee400f7a8a4f35b7a92bc6cb54de42/servers/
detail -X GET -H "X-Auth-Project-Id: admin" -H "User-Agent: python-novaclient" -
H "Accept: application/json" -H "X-Auth-Token: c321cd823c8a4852aea4b870a03c8f72"

INFO (connectionpool:191) Starting new HTTP connection (1): 172.16.1.2
```

Even though initial connection was in 192.168.0.2, the client tries to access the Public network for Nova API. The reason is because Keystone returns the list of OpenStack services URLs, and for production-grade deployments it is required to access services over public network.

Upgrade Environment (EXPERIMENTAL)

Overview

Upgrading Mirantis OpenStack from version 6.1 to version 7.0 involves building new environment, installing an HA Controller node in that environment, alongside with the old Controller nodes, switching all compute nodes to the new controller nodes, and then upgrading the Compute nodes.

You do not need any new hardware for the upgrade.

> ***Warning***
>
> During the upgrade, virtual machines and other resources might experience temporary network disconnects. Schedule the upgrade during a maintenance window.

Upgrade Scenario

The proposed solution includes the following general steps described below in more details:

- OpenStack environment of version 7.0 is created using Fuel API with the settings and network configurations matching the configuration of the original 6.1 environment.
- One of the *Cloud Infrastructure Controllers* (CICs) in the original 6.1 environment is deleted and moved to the 6.1 environment. It is then installed in that environment as a primary controller side-by-side with the original 6.1 environment.
- All OpenStack platform services are put into *Maintenance Mode* for the whole duration of the upgrade procedure to prevent user data loss and/or corruption.
- Copy the state databases of all the upgradeable OpenStack components to the new Controller and upgrade these with the standard 'database migration' feature of OpenStack.
- Reconfigure the Ceph cluster in such a way that the Monitors on the new 7.0 CICs replace the Monitors of the 6.1 environment, retaining the original IP addresses and configuration parameters.
- The 7.0 CIC is connected to the Management and External networks, while the original 6.1 ones are disconnected. The 7.0 CIC takes over Virtual IPs in the Management and Public *networks*.
- Control plane services on the Compute nodes in the 6.1 environment are upgraded to 7.0 without affecting the virtual server instances and workloads. After the upgrade, the Compute service reconnects to 7.0 CICs.
- Compute nodes from the 6.1 environment work with the CICs from the 7.0 environment, forming a temporary hybrid OpenStack environment that is only used to upgrade the Compute nodes one by one by re-assigning them to the 7.0 environment and re-installing with the new version.
- Ceph OSD nodes from the 6.1 environment transparently switch to the new Monitors without the actual data moving in the Ceph cluster.

- User data stored on OSD nodes must be preserved through re-installation of the nodes into a new release of the operating system and OpenStack services, and OSD nodes must connect to the Monitors without changing their original IDs and data set.

Every step requires certain actions. All of these actions are scripted as subcommands to the upgrade script called 'octane'.

Prerequisites and dependencies

The procedure of upgrading Mirantis OpenStack from 6.1 to 7.0 version has certain prerequisites and dependencies. You need to verify if your installation of Mirantis OpenStack meets these requirements.

Fuel installer

Mirantis OpenStack 6.1 environment must be deployed and managed by Fuel installer to be upgradeable. If you installed your environment without leveraging Fuel, or removed the *Fuel Master node* from the installation after a successful deployment, you will not be able to upgrade your environments using these instructions.

The upgrade scenario deviates from the standard sequence used in Fuel installer to deploy Mirantis OpenStack environment. These modifications to the behavior of the installer are implemented as modifications to the *deployment tasks* and extensions to certain components of Fuel. Patches are applied to the Fuel Master node as part of 'preparation' phase of the Upgrade scenario. See the sections below for the detailed description of which components are modified and why.

Architecture constraints

Make sure that your Mirantis OpenStack 6.1 environment meets the following architecture constraints. Otherwise, these instructions will not work for you:

Constraint	Check if comply
High Availability architecture	
Ubuntu 14.04 as an operating system	
Neutron networking manager with OVS+VLAN plugin	
Cinder virtual block storage volumes	
Ceph shared storage for volumes and ephemeral data	
Ceph shared storage for images and object store	

Fuel upgrade to 7.0

In this guide we assume that the user upgrades Fuel installer from version 6.1 to 7.0. The upgrade of Fuel installer is a standard feature of the system. Upgraded Fuel retains the ability to manage 6.1 environments, which is leveraged by the environment upgrade solution.

Additional hardware

The upgrade strategy requires installing 7.0 environment that will result in an OpenStack cluster along with the original environment. One of the Controller nodes from the original 6.1 environment will be deleted, added to the new 7.0 environment, and reinstalled. This allows performing an upgrade with no additional hardware.

> ***Note***
>
> The trade-off for using one of the existing controllers as a primary upgraded controller is that the 7.0 environment will not be highly available for some time during the maintenance window dedicated to the upgrade. Once the remaining controllers are moved from the 6.1 environment and reinstalled into the 7.0 environment, its High Availability is restored.

Upgrade Environment Quick Start

> ***Caution!***
>
> Only use this section as a reference. Read the detailed sections below to understand the upgrade scenario before running any of the listed commands.

This section lists commands required to upgrade a 6.1 environment to version 7.0 with brief comments. For the detailed description of what each command does and why, see the following sections:

- *Detailed upgrade procedure*
- *Detailed description of commands*

> ***Caution!***
>
> Do not run the following commands unless you understand exactly what you are doing. It can completely destroy your OpenStack environment. Please, read the detailed sections below carefully before you proceed with these commands.

1. Install the Upgrade Script

Run the following command on the Fuel Master node to download and install the Upgrade Script in the system:

```
yum install -y fuel-octane
```

Alternatively, you can install the script from Git repository.

1. Install dependency packages:

```
yum install -y git patch python-pip python-paramiko
```

2. Clone the `stable/7.0` branch of the `fuel-octane` repository by typing:

```
git clone https://git.openstack.org/openstack/fuel-octane -b stable/7.0
```

3. Install the script as a Python package by typing:

```
cd fuel-octane && pip install -e ./
```

2. Pick an OpenStack environment to upgrade

Run the following command and pick an environment to upgrade from the list:

```
fuel2 env list
```

Note the ID of the environment and store it in a variable:

```
export ORIG_ID=<ID>
```

3. Create an Upgrade Seed environment

Run the following command to create a new environment of version 7.0 and store its ID to a variable:

```
SEED_ID=$(octane upgrade-env $ORIG_ID)
```

4. Upgrade the first Controller

Pick one of the Controllers in your environment by ID and remember that ID:

```
fuel node list --env ${ORIG_ID} | awk -F\| '$7~/controller/{print($0)}'
```

> ***Note***
>
> Select a controller node with the lowest ID to ensures that you upgrade the *primary* controller first.

Use the ID of the Controller to upgrade it with the following command:

```
octane upgrade-node --isolated $SEED_ID <ID>
```

5. Upgrade State Database

After the first Controller in the 7.0 environment is deployed and ready, run the following command to upgrade the state databases the of OpenStack services:

```
octane upgrade-db $ORIG_ID $SEED_ID
```

6. Upgrade Ceph cluster

Run the following command to upgrade the Monitor node on the new Controller with the state and configuration of the original Ceph cluster:

```
octane upgrade-ceph $ORIG_ID $SEED_ID
```

7. Switch control plane to 7.0

Run the following command to switch the OpenStack environment to the 7.0 control plane:

```
octane upgrade-control $ORIG_ID $SEED_ID
```

8. Upgrade all Controllers

Identify the remaining Controllers in the 6.1 environment by IDs (ID1, ID2, etc):

```
fuel node list --env ${ORIG_ID} | awk -F\| '$7~/controller/{print($0)}'
```

Run the following command to upgrade the remaining 6.1 Controllers to version 7.0:

```
octane upgrade-node $SEED_ID <ID1> <ID2>
```

9. Upgrade Compute and Ceph OSD nodes

Repeat the following command for every node in the 6.1 environment identified by ID:

```
octane upgrade-node $SEED_ID <ID>
```

10. Delete the original 6.1 environment

After verification of the upgraded 7.0 environment, delete the original 6.1 environment with the following command:

```
fuel env --env $ORIG_ID --delete
```

Solution Overview

This section describes a solution that implements the upgrade strategy outlined in the previous section. It gives a step by step script of the procedure and explains every step of it. In the chapters that follow we will give detailed scripts with exact commands to upgrade your cluster.

Hardware considerations

For Mirantis OpenStack with *High Availability Reference Architecture* using at least 3 CICs is recommended. When you install a 7.0 Seed environment in HA mode, you can start with only 1 Controller. This procedure assumes that the node for this Controller is borrowed from the set of the Controllers in the original 6.1 environment.

Preparations and prerequisites

Before starting the upgrade itself, make sure that your system complies to the *architecture constraints* listed above. You will also need to make some preparations to provide prerequisites for the upgrade procedure. These preparations are automated using the Upgrade Script named `octane`.

Install the upgrade script on the Fuel Master node using `yum`:

```
yum install fuel-octane
```

Create 7.0 Seed environment

Our concept of the upgrade involves installing a CIC of version 7.0 side-by-side with the cloud being upgraded. We leverage Fuel installer for the task of deployment of nodes with 7.0 versions.

The way we create the upgraded environment is different from the way the ordinary OpenStack cluster is installed by Fuel 7.0. This section explains the specifics of the deployment of such a 'shadow' environment, also referred to as Upgrade Seed environment in this section.

Clone configuration of 6.1 environment

The first step in the upgrade procedure is to install a new 7.0 Upgrade Seed environment. The settings and attributes of the Seed environment must match the settings of the original environment, with the changes corresponding to the changes in the data model of Nailgun between releases.

As part of the upgrade procedure automation, an extension to Nailgun API was developed. This extension implements a resource `/api/v1/cluster/<:id>/upgrade/clone` which creates an Upgrade Seed environment based on the settings of the original one (identified by the `id` parameter).

A `POST` request to that resource must be sent to initiate the upgrade procedure.

An extension to the Fuel client was developed that allows sending the request from the CLI of the Fuel Master node.

Fuel client extension is installed in the *preparation phase* of the upgrade procedure.

Install 7.0 CIC

When the Fuel Master node and Upgrade Seed environment are prepared, you can start upgrading your 6.1 environment. First you need to pick a Controller node in the original environment. Upgrade script will move that node into the Upgrade Seed environment and reinstall it as a primary controller in that environment.

The Fuel installer deploys the Cloud Controller in a Seed environment with the following changes:

- 7.0 Controller will be isolated from the Management and Public networks on the data link layer (L2) to avoid conflicts with the Virtual IPs in the original 6.1 Controllers.

The nature of network isolation defines many aspects of the deployment process. To understand how it could be implemented, we need to analyze the configuration of the internal networking of Cloud Infrastructure Controller.

Fuel creates virtual bridges that connect the host to networks with different roles. Physical interfaces (e.g. `eth1`) are added to those bridges creating L2 connections to the corresponding networks.

On the other hand, L3 IP address is assigned to virtual bridge to the network of a given type. A virtual bridge to connect to Management network is called `br-mgmt`, to Public network - `br-ex`, and so on.

To install 7.0 Controller in isolation from these networks, the script configures the deployment information for the node in a way that physical interfaces are not added to certain virtual bridges when OpenStack is being deployed.

This ensures that the Controller has no physical connection to Management and Public network.

Using Fuel for isolated deployment

Fuel is responsible for the assignment of IP addresses to logical interfaces in the Management, Public and other types of networks. The environment cloning function in Nailgun API does copy the IP ranges and Virtual IP addressses of the original cluster.

Fuel configures Linux bridges and interfaces during the deployment of an environment. This configuration is managed by Puppet and is defined in the deployment settings. You can modify these settings to disable adding of certain physical interface to the bridge.

For deployment to succeed with the described schema, you need to ensure that no network checks break the installation by disabling a check for connectivity to the default gateway. Fuel installer expects the gateway to be in the Public network, which is not directly accessible from our isolated Controller.

The exact commands to create an isolated Upgrade Seed environment are listed in the *Upgrade Script* chapter.

Initial state of Ceph cluster

According to the upgrade scenario, Ceph cluster must be installed in a way that allows for replacing the original Monitors of the 6.1 environment with the new Monitors when 7.0 CICs take over. There is a way to install a cluster without OSD nodes and thus rule out the rebalance and data movement once the original OSD nodes start joining the cluster. However, it requires that the upload of the test VM image by Fuel is disabled before the deployment. We achieve this by disabling the corresponding tasks in the deployment graph: `upload_cirros` and `check_ceph_ready`.

Maintenance Mode

During the installation of 7.0 Seed cloud the original 6.1 environment continues to operate normally. Seed CIC do not interfere with the original CICs and the latter could continue the operation through the initial stages of the upgrade.

However, when it comes to the upgrade of state databases of OpenStack services, you need to make sure that no changes are made to the state data. Maintenance mode must be started before you download data from the state database of 6.1 OpenStack environment. Maintenance mode should last at least until the database upgrade is finished and 7.0 CICs take over the environment.

Note that Maintenance mode implemented according to these instructions does not impact operations of existing virtual server instances and other resources. It only affects OpenStack API endpoints which are the only way for the end user to change the state data of the cluster.

High Availability architecture of Mirantis OpenStack provides access to all OpenStack APIs at a single VIP address via HAProxy load balancer. You need to configure HAProxy server to return code `HTTP 503` on all requests to

the services listening on the Public VIP in 6.1 environment. This will not allow users to change the state of the virtual resources in the original cloud which can be lost after the data is downloaded from DB.

On 7.0 CIC, you must disable all OpenStack component services to make sure that they do not write to the state database while it is being upgraded. Otherwise, this might lead to data corruption and loss.

All the detailed commands used to put environments into Maintenance mode are listed in the Upgrade Script chapter below.

Upgrade databases

Database upgrade is a standard procedure provided by OpenStack upstream as a main upgrade feature. It allows converting data from state databases of all OpenStack component services from a previous to a new release version schema. It is necessary to fully preserve the status of the virtual resources provided by the cloud through the upgrade procedure.

The data is dumped from MySQL database on one of the CIC nodes in 6.1 environment. A text dump of the database is compressed and sent over to CIC node in 7.0 environment.

After uploading the data to MySQL on 7.0 CIC, use standard OpenStack methods to upgrade the database schema to the new release. Specific commands that upgrade the schema for particular components of the platform are listed in the Upgrade Script chapter below.

Configure Ceph Monitors

Architecture constraints for the upgrade procedure define that in the upgradeable configuration Ceph is used for all types of storage in the OpenStack platform: ephemeral storage, permanent storage, object storage and Glance image store. Ceph Monitors are essential for the Ceph cluster and must be upgraded seamlessly and transparently.

By default, Fuel installer creates a new Ceph cluster in the 7.0 Seed environment. You need to copy the configuration of the cluster from 6.1 environment to override the default configuration. This will allow OSD nodes from 6.1 environment to switch to the new Monitors when 7.0 CICs take over the control plane of the upgraded environment.

The Upgrade Script synchronizes the configuration of Ceph Monitors in the 6.1 and 7.0 clusters during the upgrade procedure.

Upgrade Control Plane

This step is called 'Upgrade', as it concludes with a new CIC of version 7.0 listening on the same set of IP addresses as the original 6.1 CICs. However, from the technical standpoint it is more of a switch to a new version of control plane. 7.0 Controller takes over the Virtual IP addresses of 6.1 environment, while the original CICs are disconnected from all networks except Admin. The sections that follow explain what happens and why at every stage of the upgrade process.

Start OpenStack services on 7.0 Controller

As part of Maintenance mode, OpenStack component services were shut down on 7.0 CIC before upgrading the database. These services include Nova, Glance, Keystone, Neutron and Cinder. Now it is time to restore them with a new data set created by the database migration procedure. This operation basically reverts the shutdown operation described above. It is automated in the Upgrade Script.

Note that Neutron restart involves creation of tenant networking resources on CIC nodes where Neutron agents run. This process can take longer than starting all other services, so check it carefully before you proceed with the upgrade.

Delete ports on 6.1 Controllers

Before 7.0 CIC can take over the virtual network addresses in the upgraded environment, you need to disconnect 6.1 CICs to release those addresses. Based on the CICs networking schema described above, to do that you need to delete patch ports from certain OVS bridges.

This procedure is automated by the upgrade script and executed as part of the `upgrade-cics` subcommand.

Connect 7.0 Controller

After 6.1 CICs are disconnected from all networks in the environment, 7.0 CIC can take over their former VIP addresses. The take-over procedure adds physical ports to the appropriate bridges and brings the ports up.

Upgrade hypervisor host

Hypervisor hosts provide their physical resources to run virtual machines. Physical resources are managed by hypervisor software, usually 'libvirt' and 'qemu-kvm' packages. With KVM hypervisor, all virtualization tasks are handled by the Linux kernel. Open vSwitch provides L2 network connectivity to virtual machines. All together, kernel, hypervisor and OVS constitute a data plane of Compute service.

You can upgrade data-plane software on a hypervisor host (or Compute node) by re-installing the operating system to a new version with Fuel installer. However, the deployment process takes time and impacts the virtual machines. To minimize the impact, leverage live migration to move all virtual machines from the Compute node before you start upgrading it. You can do that since Compute node's control plane is upgraded to 7.0.

Nailgun API extension installed by the Upgrade Script allows to move a node to the Upgrade Seed environment in runtime. It preserves the ID of the node, its hostname and configurations of its disks and interfaces.

When a node is added to the upgraded environment, the script provisions the node. When the provisioning is finished, the script runs the deployment of the node. As a result of the deployment, the node will be added to the environment as a fully capable Mirantis OpenStack 7.0 Compute node.

Upgrade of a single Compute node must be repeated for all the nodes of 6.1 environment in a rolling fashion. VMs must be gradually moved from the remaining 6.1 Compute nodes to the 7.0 ones with live migration.

Upgrade Ceph OSD node

In a Ceph cluster all data is stored on OSD nodes. These nodes have one or more storage devices (or disk partitions) dedicated to Ceph data and run ceph-osd daemon that is responsible for I/O operations on Ceph data.

Upgrading OSD node via Fuel means that the node must be redeployed. Per the requirement to minimize end-user impact and the move of data across the OpenStack cluster being upgraded, we developed a procedure to redeploy Ceph OSD nodes with the original data set. Although Fuel by default erases all data from disks of the node it deploys, you can patch and configure the installer to keep Ceph data on the devices intact.

There are several stages of the deployment when the data is erased from all disks in the Ceph OSD node. First, when you delete a Ceph node, Nailgun agent on that node does the erasing on all non-removable disks by writing 0s to the first 10MB of every disk. Then, at the provisioning stage, Ubuntu installer creates partitions on disks and formats them according to the disks configuration provided by Fuel orchestration components.

As part of the upgrade procedure, we provide patches for the components involved in volumes management that allow to keep data on certain partitions or disks. These patches are applied automatically by the Upgrade Script.

Disable rebalance

By default, Ceph initiates rebalance of data when OSD node goes down. Rebalancing means that the data of replicas is moved between the remaining nodes, which takes significant time and impacts end user's virtual machines and workloads. We disable the rebalance and recalculation of CRUSH maps when OSD node goes down. When a node is reinstalled, OSD connects to Ceph cluster with the original data set.

Finalizing the upgrade

When all nodes are reassigned to the 7.0 environment and upgraded, it is time to finalize the upgrade procedure with a few steps that allow Fuel installer to manage with the upgraded environment just as with vanilla 7.0 environment, installed from scratch:

- revert all patches applied to Fuel components;
- delete the original environment.

Upgrade Script Explained

In this chapter we explain how exactly we implement the steps described in *Solution Overview*. Sections of this chapter contain appendices with listings of commands and scripts that implement each particular action.

> ***Important***
>
> - All commands in this section must be executed on the Fuel Master node, unless specified otherwise.
> - You must use Bourne shell (`bash`) to execute commands.

Upgrade Prerequisites

There are certain prerequisites for upgrading Mirantis OpenStack environment managed by Fuel installer. You need to make sure that all the requirements are met before proceeding with the procedure. This section describes prerequisites and gives a list of commands used to check if the requirements are met.

Versions of Fuel installer and environment

First, you need to check the versions of Fuel installer and the environment you've picked for the upgrade. Version of Fuel must be equal to 7.0. Version of environment must be equal to 6.1. You can check versions using the Fuel Web UI or CLI client to Fuel API.

Configuration of environment

The configuration of the environment picked for the upgrade must comply to the architecture constraints for the upgrade procedure. You can check the applicability of the procedure to your configuration via Fuel Web UI.

Check Upgrade Prerequisites

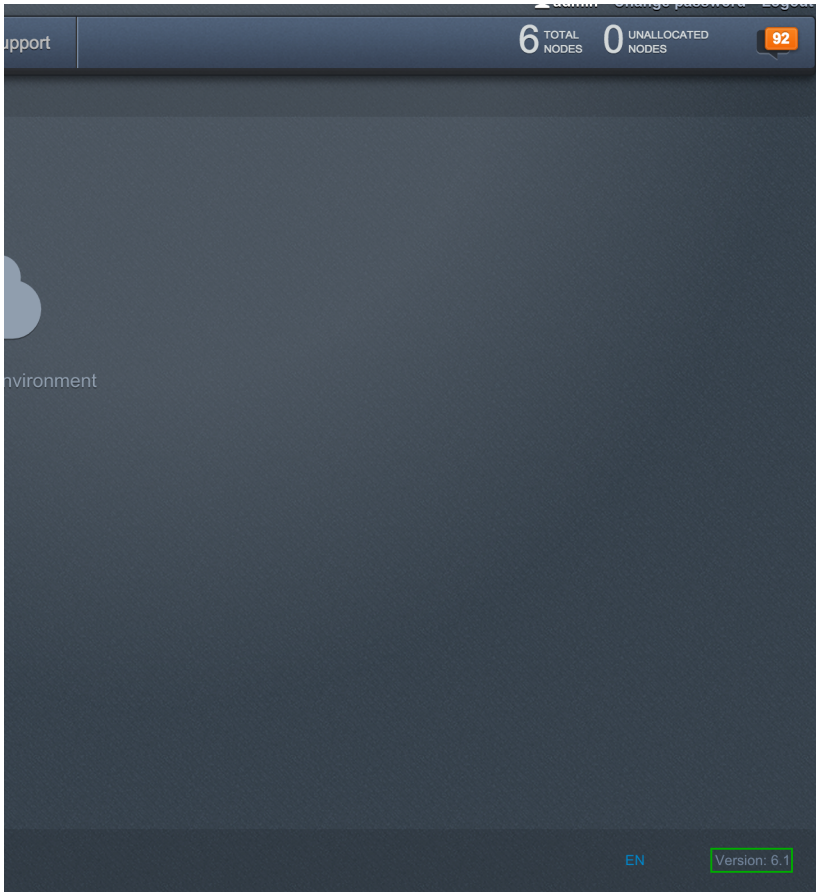
Pick environment to upgrade

Select an environment to upgrade from the list of environments in Fuel CLI and assign its ID to the `ORIG_ID` variable.

```
fuel env
<select from list of environments>
export ORIG_ID=<enter ID of environment here>
```

Check installer version in Fuel Web UI

Open the Fuel Web UI in your browser, log in and check the version number of Fuel installer in the lower right corner of the page:



Check installer version in Fuel CLI

Run the following command on your Fuel Master node to verify the version of the Fuel installer:

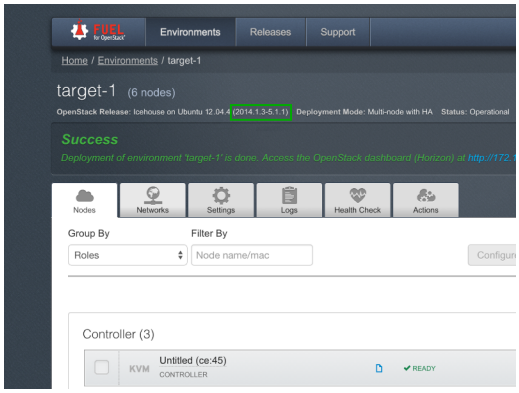
```
fuel release | grep available | grep -o 2015.1-7.0
```

You must see the following lines in the output:

```
2015.1-7.0
2015.1-7.0
```

Check environment version on the Fuel Web UI

Click on the environment you've picked for the upgrade. Check the environment version in the status line above the list of nodes:



Check environment version in Fuel CLI

Run the following command on your Fuel Master node to verify the version of the environment you pick for the upgrade:

```
fuel env | awk -F\| '$1~/'$ORIG_ID'/{print $5}' | tr -d ' ' \
| xargs -I@ bash -c "fuel release | awk -F\| '\$1~/@/{print \$5}'" | tr -d ' '
```

You must see the following line in the output:

```
2014.2.2-6.1
```

Check configuration of environment

You need to open the Fuel Web UI in your browser. Log in, click on the environment you'd like to upgrade and select the *Settings* tab. Check the following fields and verify they contain the values:

- Hypervisor type: **KVM**
- Ceph RBD for volumes: **Enabled**
- Ceph RBD for images: **Enabled**
- Ceph RBD for ephemeral volumes: **Enabled**
- Ceph RadosGW for objects: **Enabled**

Navigate to the *Networks* tab and check the second line after the tab title. It must state *Neutron with VLAN segmentation*.

Prepare Fuel Master

Before you start the upgrade procedure, you must prepare your Fuel installer. There are several changes to the default behavior and configuration of installer that the upgrade functions depend on. The configuration changes include installation of additional packages and setting environment variables.

In this section we briefly describe which components are affected by the change and why, and explain how to apply the patches correctly. Explanations of specific patches and their purpose will be given below in the sections dedicated to the steps that make use of those patches:

Install the Upgrade Script

The upgrade logic is automated in the script named `octane`. You need to download and install this script from RPM repository on your Fuel Master node.

Prepare environment variables

There are several variables that need to be set before you start the upgrade procedure. They will be used throughout the whole process. These variables include the ID and name of the environment picked for the upgrade.

Commands to Install the Upgrade Script

Install the `octane` application using `yum` package manager:

```
yum install -y fuel-octane
```

Alternatively, you can use the Git version control system to install the script from the source code that resides in upstream repository.

To install the script from the repository:

1. Install dependenecy packages using `yum install`:

```
yum install -y git patch python-pip python-paramiko
```

2. Clone the `fuel-octane` repository:

```
git clone https://git.openstack.org/openstack/fuel-octane -b stable/7.0
```

3. Change current directory to `fuel-octane`.
4. Install the upgrade script:

```
cd fuel-octane && pip install -e ./
```

Clone Environment settings

During the upgrade, the new Fuel environment of version 7.0 will be created with a copy of Network parameters and Settings from the upgrade target environment of version 6.1. The generated configurations, like credentials of the service users, are also copied from the original environment wherever possible.

Environment clone command

Run the following command to create Upgrade Seed environment:

```
octane upgrade-env ${ORIG_ID}
```

Store the ID of the Upgrade Seed environment displayed as a result of this command into variable:

```
export SEED_ID=<ID>
```

Install Controller

Installation of CIC node is performed by standard Fuel installer actions and is split into two distinct steps.

#. Prepare nodes settings and use the provisioning feature of Fuel to install an operating system and configure disks on nodes. #. Make modifications to the deployment information of the environment that affects all CIC nodes, and deploy OpenStack platform onto them.

Modified deployment settings

To deploy a 7.0 CIC node properly, we need to prepare deployment information to make Fuel configure nodes and OpenStack services with the following modifications:

1. Disable checking access to the default gateway in Public network
2. Skip adding physical interfaces to Linux bridges
3. Skip creation of the 'default' Fuel-defined networks in Neutron
4. Change default gateway addresses to the address of the Fuel Master node

Deployment settings can be downloaded from Fuel API as a set of files. The upgrade script updates the settings by changing those files and uploading the modified information back via Fuel API.

> ***Note***
>
> Always upgrade the controller with the lowest ID number first. The controller node with the lowest ID number is a so-called *primary* controller and you must upgrade before other controllers in the OpenStack environment.

The upgrade script keeps the deployment infromation for the environment in a cache directory (by default, `/tmp/octane/deployment`). The deployment settings for the nodes are stored to subdirectory `deployment_<cluster-id>`, where `<cluster-id>` is an ID of the environment.

The deployment tasks are kept in subdirectory `cluster_<cluster-id>` of the same cache directory. You can use them to check that the deployment is properly configured.

Install Controller commands

The upgrade automation script allows upgrading a controller that exists in the original 6.1 environment. Select the controller with minimal ID out of all controllers deployed in the environment and run the following command. Make sure to replace <NODE-ID> with the actual ID of 6.1 Controller you'd like to upgrade:

```
octane upgrade-node ${SEED_ID} isolated <NODE-ID>
```

Switch to 7.0 Control Plane

To switch the Controller services, the script transfers state data for those services from the original Controllers to 7.0 Seed Controllers and swaps the Controllers connections to the Management and External networks.

Maintenance mode

To prevent the loss of data in OpenStack state database, API interaction with the environment must be disabled. This mode of operations is also known as *Maintenance Mode*.

In maintenance mode, all services that write to DB are disabled. All communications to the control plane of the cluster are also disabled. VMs and other virtual resources must be able to continue to operate as usual.

> ### *Note*
>
> The Maintenance Mode is automatically set up by the Upgrade Script as soon as you start the upgrade of the state database. Make sure to carefully plan the maintenance window for that time and inform your users in advance.

Database migration

Before the databases could be upgraded, all OpenStack services on 7.0 Controller must be stopped to prevent corruption of the metadata. The upgrade script uses standard Ubuntu startup scripts from `/etc/init.d` on controllers to shut off the services.

Databases are dumped in text format from MySQL server on 6.1 CIC, copied to 7.0 CIC, and uploaded to the upgraded MySQL server. Standard OpenStack tools allow to upgrade the structure of databases saving the data itself via sqlalchemy-powered DB migrations.

Run the following command to set up Maintenance Mode and immediately start upgrading the state databases of OpenStack services:

```
octane upgrade-db ${ORIG_ID} ${SEED_ID}
```

Upgrade Ceph cluster

To replace Ceph Monitors on the same IP addresses, the upgrade script must preserve the cluster's identity and auth parameters. It copies the configuration files, keyrings and state dirs from 6.1 CICs to 7.0 CICs and uses Ceph management tools to restore cluster identity.

Run the following command to replicate the configuration of the Ceph cluster:

```
octane upgrade-ceph ${ORIG_ID} ${SEED_ID}
```

Upgrade CICs

The following section describes the procedure for replacing Controllers from 6.1 environment with the Controllers from 7.0 environment and then upgrading the 6.1 Controllers.

When the DB upgrade is finished, all OpenStack services on 7.0 CIC are started using Pacemaker and Upstart. Then the upgrade script disconnects the 6.1 CICs from the Management and Public networks by removing patch ports between the logical interfaces to the respective networks and physical interfaces connected to the network media.

On 7.0 CIC, the physical interface must be added to the Linux bridge corresponding to the Management network. This allows the Compute nodes to transparently switch from the old to the upgraded control plane without the need to reconfigure and renumber every service.

Commands to switch the Control Plane

Run the following command to switch from 6.1 to 7.0 Control Plane:

```
octane upgrade-control ${ORIG_ID} ${SEED_ID}
```

Upgrade Node

Node upgrade is essentially a reinstallation of an operating system and OpenStack platform services. We need to delete a node from the 6.1 environment and assign it to 7.0 Seed environment, then provision it and deploy OpenStack on it.

By default, Fuel installer erases all data from the disks on the node, creates new partitions structure, installs an operating system and deploys OpenStack.

Depending on the roles the node has in the 6.1 environment, we might need to make changes to the behavior. For example, to upgrade Ceph OSD node, we need to make Fuel keep data on Ceph partitions of that node. To upgrade Compute nodes, we need to use live migration to move users' VMs from it to other hypervisor hosts. There are also several steps in the upgrade procedure that are common to both supported roles ('compute' and 'ceph-osd'). This section describes these common steps to upgrade a node and how to use the Upgrade Script to upgrade a node with any of the supported roles.

Prepare Ceph OSD node to upgrade

Preparations for the upgrade of Ceph OSD node include steps to minimize data transfers inside the Ceph cluster during the upgrade. They also aim to ensure that the data on OSD devices is kept intact.

From the impact standpoint, the most optimal solution is to minimize data transfers over network during the upgrade of Ceph cluster. Ceph will normally rebalance its data when OSD node is down. However, since the described procedure preserves Ceph data on the node, rebalance must be turned off. We use the standard Ceph flag `noout` to disable the rebalance on a node outage.

Prepare Compute node to upgrade

Compute node runs virtual machines in hypervisor processes. To satisfy the requirement to minimize the downtime of virtual resources, we must ensure that the VMs are moved from the node in preparation for the

reinstallation. The move must be done with the most seamless migration method available. This move is referred to as **evacuation**.

Using Ceph shared storage for the Nova instance's ephemeral disk allows using live migration of virtual instances. The sections below describe the steps required to live migrate all VMs from the hypervisor host picked for the upgrade.

To minimize the impact of the upgrade procedure on the end user workloads, we need to migrate all VMs off the hypervisor picked for the upgrade and make sure that no new VMs are provisioned to that host. Scheduling to the host can be ruled out by disabling Compute service on that host. This does not affect the ability to migrate VMs from that host.

Reassign node to Seed environment

There is an `/api/cluster/<:id>/upgrade` extension to the Fuel API that allows moving a node from an environment the its Upgrade Seed. So the reassignment is a matter of a single call to the Fuel API from the Upgrade Script.

Note that with this extension the reassigned node will not change its ID or host name.

Verify node upgrade

After the successful installation you need to make sure that the node is connected to CICs properly, according to its role. Ceph OSD node must be `up` in the OSD tree. The Compute node must be connected to the Nova controller services. Below in the list of commands you will find how to check these requirements.

Node Upgrade command

Run the following command to upgrade a node identified by the `NODE_ID` parameter:

```
octane upgrade-node ${SEED_ID} ${NODE_ID}
```

You can upgrade more than one node at a time. Run the following command to upgrade multiple nodes identified by the `NODE_ID1`, `NODE_ID2`, and `NODE_ID3` parameters:

```
octane upgrade-node ${SEED_ID} ${NODE_ID1} ${NODE_ID2} ${NODE_ID3}
```

> ***Note***
>
> Pay attention to the replication factor of your Ceph cluster and do not upgrade more Ceph OSD nodes than the value of the replication factor minus one.

Finalize Upgrade

This chapter contains actions required to finalize the upgrade procedure on an environment and on Fuel installer in general. Finalization involves the following steps:

- Restore the source code of the components of the installer by reverting the patches that implement modifications to the installer's behavior.
- Delete the original 6.1 environment and release the original controller nodes.

See the sections below for the detailed description of how to do that and the list of commands:

- *Uninstall upgrade-related components*
- *Decommission environment*

Uninstall the Upgrade Script

The final goal of the upgrade procedure is to get the upgraded environment as close as possible to the environment installed with the new release version and retain the ability to manage it in the new version of Fuel installer.

To return the Fuel Master node into original state, we just need to uninstall components that we installed for the upgrade purposes, specifically, extension to `fuelclient` CLI command that is used to call Upgrade extension in the Fuel API.

Commands to uninstall components

Run the following command to revert the changes made to the source code and configuration of components of the Fuel installer:

```
yum erase -y fuel-octane
```

Delete 6.1 environment

Delete the original 6.1 environment to release the Controller nodes and completely switch to use the 7.0 environment instead.

> **Note**
>
> The following operation may cause data loss if your upgrade operation was not completed successfully. Proceed with caution.

```
fuel env --env $ORIG_ID --delete
```

Enable Ubuntu bootstrap (EXPERIMENTAL)

By default, Fuel 7.0 uses CentOS 6.6 bootstrap operating system. Ubuntu 14.04 bootstrap is only available as an experimental feature. See *Release notes* for details about known issues with this feature.

To enable Ubuntu 14.04 bootstrap:

1. Enable *experimental features*.
2. Verify that you are logged as root into your Fuel Master node console and that your Master node has an access to the Internet.
3. Run the **fuel-bootstrap-image-set ubuntu** command.
4. Run the **ls -l /var/www/nailgun/bootstrap/ubuntu/root.squashfs** command to verify that the Ubuntu image is built successfully. The build log is available in */var/log/fuel-bootstrap-image-build.log*.
5. Reboot the discovered nodes.

HA testing scenarios

Currently, several testing scenarios are provided to check HA environment.

Regular testing scenarios

Nova-network

You can run the following tests on the supported operating system.

1. Deploy a cluster in HA mode with VLAN Manager. Steps to perform:
    - Create a cluster.
    - Add 3 nodes with controller role.
    - Add 2 nodes with compute role.
    - Set up a cluster to use Network VLAN manager with 8 networks.
    - Deploy the cluster.
    - Make sure that the cluster is configured correctly: there should be no dead services or no errors in the logs. Also, you should check that all nova services are running and they are in up state; TestVM must appear in Glance and only one nova-network should be present.
    - Run network verification test.
    - Run OSTF.
2. Deploy a cluster in HA mode with nova-network and Flat DHCP manager enabled. Steps to perform:
    - Create a cluster.
    - Add 3 nodes with controller role.
    - Add 2 nodes with compute role.
    - Deploy the cluster.
    - Make sure that the cluster is configured correctly: there should be no dead services or no errors in the logs. Also, you should check that all nova services are running and they are in up state; TestVM must appear in Glance and only one nova-network should be present.
    - Run network verification test.
    - Perform a security check: verify that it is impossible to access TCP or UDP unused ports.
    - Run OSTF.
3. Add a compute node to a cluster in HA mode with nova-network with Flat DHCP manager enabled. Steps to perform:
    - Create cluster
    - Add 3 nodes with controller role
    - Add 2 nodes with compute role.
    - Deploy the cluster.

- Make sure that the cluster is configured correctly: there should be no dead services or no errors in the logs. Also, you should check that all nova services are running and they are in up state; TestVM must appear in Glance and only one nova-network is present.
- Add one node with compute role.
- Re-deploy the cluster.
- Make sure that the cluster is configured correctly: there should be no dead services or no errors in the logs. Also, you should check that all nova services are running and they are in up state; TestVM must appear in Glance and only one nova-network should be present.
- Run network verification test.
- Run OSTF.

4. Deploy an HA cluster with Ceph and nova-network: Steps to perform:
    - Create a cluster: use Ceph for volumes and images.
    - Add 3 nodes with controller and Ceph OSD roles.
    - Add one node with Ceph OSD role.
    - Add 2 nodes with compute and Ceph OSD roles.
    - Start cluster deployment.
    - Check Ceph status with **ceph health** command. Command output should have **HEALTH_OK**.
    - Run OSTF.
5. Stop and reset nova-network cluster in HA mode. Steps to perform:
    - Create a cluster.
    - Add 3 nodes with controller role.
    - Start cluster deployment.
    - Stop deployment.
    - Reset settings.
    - Add 2 nodes with compute role.
    - Re-deploy the cluster.
    - Run OSTF.
    - Run network verification test.
6. Deploy nova-network cluster in HA mode with Ceilometer. Steps to perform:
    - Create a cluster. On **Settings** tab of the Fuel web UI, select **Install Ceilometer** option.
    - Add 3 nodes with controller role.
    - Add one node with compute role.
    - Add one node with MongoDB role.
    - Deploy the cluster.

- Check that partitions on MongoDB node are the same as those selected on the Fuel web UI.
- Make sure that Ceilometer API is running (it must be present in *ps ax* output).
- Run OSTF.

7. Check HA mode on scalability. Steps to perform:
    - Create a cluster.
    - Add 1 controller node.
    - Deploy the cluster.
    - Add 2 controller nodes.
    - Deploy the changes.
    - Check Pacemaker status: all nodes must be online after running **crm_mon -1** command.
    - Run network verification test.
    - Add 2 controller nodes.
    - Deploy the changes.
    - Check that public and management vIPs have started after running **crm_mon -1** command.
    - Run network verification test.
    - Run OSTF.
8. Backup/restore Fuel Master node with HA cluster. Steps to perform:
    - Create a cluster with 3 controllers and 2 compute nodes.
    - Backup Fuel Master node.
    - Check if the backup succeeded.
    - Run OSTF.
    - Add 1 node with compute role.
    - Restore Fuel Master node.
    - Check if restore procedure succeeded. Before backup, a file is created and its checksum is saved. After backuping and restoring the environment, you get the checksum of this file and verify that it is equal to the checksum that was saved before backup.
    - Run OSTF.

Neutron

You can run the following tests on the supported operating system.

1. Deploy a cluster in HA mode with Neutron GRE segmentation. Steps to perform:
    - Create a cluster.
    - Add 3 nodes with controller role.
    - Add 2 nodes with compute role.

- Deploy the cluster.
- Run network verification test.
- Run OSTF.

2. Deploy a cluster in HA mode with Neutron GRE segmentation and public network assigned to all nodes. Steps to perform:
    - Create a cluster.
    - Add 3 nodes with controller role.
    - Add 2 nodes with compute role.
    - On **Settings** tab of the Fuel web UI, select *Assign public networks to all nodes* option.
    - Deploy the cluster.
    - Check that public network is assigned to all nodes.
    - Run network verification test.
    - Perform a security check: verify that it is impossible to access TCP or UDP unused ports.
    - Run OSTF.
3. Deploy a cluster in HA mode with Neutron VLAN. Steps to perform:
    - Create a cluster.
    - Add 3 nodes with controller role.
    - Add 2 nodes with compute role.
    - Deploy the cluster.
    - Run network verification test.
    - Run OSTF.
4. Deploy cluster in HA mode with Neutron VLAN and public network assigned to all nodes. Steps to perform:
    - Create a cluster.
    - Add 3 nodes with controller role.
    - Add 2 nodes with compute role.
    - On **Settings** tab of the Fuel web UI, select *Assign public networks to all nodes* option.
    - Deploy the cluster.
    - Check that public network is assigned to all nodes.
    - Run network verification test.
    - Perform a security check: verify that it is impossible to access TCP or UDP unused ports.
    - Run OSTF.
5. Deploy a cluster in HA mode with Murano and Neutron GRE segmentation. Steps to perform:
    - Create a cluster. On **Settings** tab of the Fuel web UI, select *Install Murano* option.

- Add 3 nodes with controller role.
- Add 1 node with compute role.
- Deploy the cluster.
- Verify that Murano services are up and running (check that *murano-api* is present in 'ps ax' output on every controller).
- Run OSTF.
- Register Murano image.
- Run Murano platform OSTF tests.

6. Deploy Heat cluster in HA mode. Steps to perform:

    - Create a cluster.
    - Add 3 nodes with controller role.
    - Add one node with compute role.
    - Deploy the cluster.
    - Verify that Heat services are up and running (check that *heat-api* is present in 'ps ax' output on every controller).
    - Run OSTF.
    - Register Heat image.
    - Run OSTF platform tests.
7. Deploy a new Neutron GRE cluster in HA mode after Fuel Master is upgraded. Steps to perform:

    - Create a cluster with 1 controller with Ceph, 2 compute nodes with Ceph; Ceph for volumes and images should also be enabled.
    - Run upgrade on Fuel Master node.
    - Check that upgrade has succeeded.
    - Deploy a new upgraded cluster with HA Neutron VLAN manager, 3 controllers, 2 compute nodes and 1 Cinder.
    - Run OSTF.

Bonding

You can run the following tests on the supported operating system:

1. Deploy cluster in HA mode for Neutron VLAN with bonding. Steps to perform:

    - Create a cluster.
    - Add 3 nodes with controller role.
    - Add 2 nodes with compute role.
    - Set up bonding for all interfaces in **active-backup** mode.

- Deploy the cluster.
- Run network verification test.
- Run OSTF.

2. Deploy cluster in HA mode for Neutron GRE with bonding. Steps to perform:
    - Create a cluster.
    - Add 3 nodes with controller role.
    - Add 2 nodes with compute role.
    - Setup bonding for all interfaces in **balance-slb** mode.
    - Deploy the cluster.
    - Run network verification test.
    - Run OSTF.

Failover testing scenarios

> ***Warning***
>
> These scenarios are destructive and you should not try to reproduce them.

1. Neutron L3-agent rescheduling after L3-agent dies. Steps to perform:
    - Create a cluster (HA mode, Neutron with GRE segmentation).
    - Add 3 nodes with controller role.
    - Add 2 nodes with compute role.
    - Add one node with Cinder role.
    - Deploy the cluster.
    - Manually reschedule router from the primary controller to another one.
    - Stop L3-agent on a new node with **- pcs resource ban p_neutron-l3-agent NODE** command.
    - Check whether L3-agent has been rescheduled.
    - Check network connectivity from the instance with *dhcp namespace*.
    - Run OSTF.
2. Deploy nova-network environment with Ceph in HA mode. Steps to perform:
    - Create a cluster with Ceph for images and volumes.
    - Add 3 nodes with controller and Ceph OSD roles.

- Add 1 node with Ceph OSD role.
- Add 2 nodes with compute and Ceph OSD roles.
- Deploy the cluster.
- Check Ceph status with **ceph-health** command. Command output should have *HEALTH_OK*.
- Destroy a node with Ceph role and check Ceph status.
- Run OSTF and check Ceph status.
- Destroy the compute node with Ceph and check Ceph status.
- Run OSTF and check Ceph status.
- Restart 4 online nodes.
- Run OSTF and check Ceph status.

3. Monit on compute nodes for nova-network and Neutron. Steps to perform:

   - Deploy HA cluster with nova-network or Neutron 3 controllers and 2 compute nodes.
   - SSH to each compute node.
   - Kill nova-compute service.
   - Check that service has been restarted by Monit.

4. Pacemaker restarts heat-engine when AMQP connection is lost Steps to perform:

   - Deploy HA cluster with nova-network or Neutron, 3 controllers and 2 compute nodes.
   - SSH to any controller.
   - Check heat-engine status.
   - Block heat-engine AMQP connections.
   - Check that heat-engine has stopped on the current controller.
   - Unblock heat-engine AMQP connections.
   - Check that heat-engine process is running with new pid.
   - Check that AMQP connection has re-appeared for heat-engine.

The following testing scenarios (from 5 to 11) may be mixed with Nova or Neutron.

5. Shut down primary controller. Steps to perform:

- Deploy a cluster with 3 controllers and 2 compute nodes.
- Destroy the primary controller.
- Check Pacemaker status: all nodes must be online after running **crm_mon -1** command.
- Wait until MySQL Galera is up (command should return "On"):

```
SELECT VARIABLE_VALUE FROM information_schema.GLOBAL_STATUS WHERE VARIABLE_NAME \
  = 'wsrep_ready'
```

- Run OSTF.

6. Shut down non-primary controller. Steps to perform:

   - Deploy a cluster with 3 controllers and 2 compute nodes.
   - Destroy non-primary controller.
   - Check Pacemaker status: all nodes must be online after running **crm_mon -1** command.
   - Wait until MySQL Galera is up (it must return "On"):

```
"SELECT VARIABLE_VALUE FROM information_schema.GLOBAL_STATUS WHERE VARIABLE_NAME \
   = 'wsrep_ready'
```

   - Run OSTF.

7. Shut down management interface on the primary controller.

> ***Note***
>
> When you use **ifdown**, **ifup**, or commands that call them, it can cause the Corosync service to update the cluster state and in most cases leads to so-called split-brain: the test will fail. Instead, use **ip link set down <ethX>** or physically disconnect the interface.

   Steps to perform:

- Deploy a cluster with 3 controllers and 2 compute nodes.
- Disconnect eth2 of the first controller via iptables.
- Check Pacemaker status: all nodes must be online after running **crm_mon -1** command.
- Wait for vip__ resources to migrate to the working controllers.
- Run 'smoke' OSTF tests.
- Restore connectivity to the first controller.
- Wait until Pacemaker specifies the *lost* controller as *online*.
- Wait for Pacemaker resources to become operational on all controllers.
- Run "sanity" and "smoke" OSTF tests.
- Repeat steps described above (from disconnecting eth2) for another controller.
- Run OSTF.

8. Delete all management and public vIPs on all controller nodes: Steps to perform:
    - Delete all secondary vIPs.
    - Wait till it gets restored.
    - Ensure that vIp has restored.
    - Run OSTF.
9. Terminate HAProxy on all controllers one by one: Steps to perform:
    - Terminate HAProxy on every controller in cycle.
    - Wait till it gets restarted.
    - Go to another controller and repeat steps above.
    - Run OSTF.
10. Terminate MySQL on all controllers one by one Steps to perform:
    - Terminate MySQL on every controller in cycle.
    - Wait until it gets restarted.
    - Verify that MySQL has restarted.
    - Go to another controller.
    - Run OSTF.
11. Verify that resources are configured. Steps to perform:
    - SSH to controller node.
    - Verify that all resources are configured.
    - Go to another controller.

Rally

1. Run Rally for generating typical activity on a cluster (for example, create or delete instance and/or volumes). Shut down the primary controller and start Rally:
    - Ensure that vIP addresses have moved to another controller.
    - Ensure that VM is reachable from the outside world.
    - Check the state of Galera and RabbitMQ clusters.
2. HA load testing with Rally. Steps to perform:
    - Deploy HA cluster with Neutron GRE or VLAN, 3 MongoDB controllers and 4 Ceph compute nodes. You should also have Ceph volumes and images enabled for Storage.
    - Create an instance.
    - Wait until instance is created.
    - Delete the instance.
    - Run Rally for generating the same activity on the cluster. In average, 500-1000 VMs should be created using 50, 70 or 100 parallel requests.

OpenStack Database Backup and Restore with Percona XtraBackup

With the procedure described in this topic you will be able to back up and restore your OpenStack MySQL database.

You will need to put the OpenStack environment into maintenance mode.

In the maintenance mode the following services will be unavailable:

- MySQL and HAProxy on the selected controller node
- HAProxy on other controller nodes in the cluster for a short time

Backing up with Percona XtraBackup

1. Enable the HAProxy stats socket for every controller in a cluster:
    1. Open the */etc/haproxy/haproxy.cfg* file for editing.
    2. Find the *stats socket /var/lib/haproxy/stats* line in the global section and add *level admin* at the end of the line.
1. Restart HAProxy in one of the following ways:
    - Execute /usr/lib/ocf/resource.d/mirantis/ns_haproxy reload on every controller

    Or

    - Reload all HAProxy instances on all controllers in a cluster with a temporary services stop by running the *crm resource restart p_haproxy* command.
2. On the Fuel Master node, run the *fuel nodes | grep controller* command. If the node that you are going to back up is a host for a Neutron agent, you can move the agent to a different controller with the following command:

    ```
    ssh node-1
    pcs resource move agent_name node_name
    ```

    where "node-1" is the name of the node from which you would like to move.
3. For every controller in the cluster, put the MySQL service into maintenance mode by running the following command from the Fuel Master node:

    ```
    ssh -t node-1 'echo "disable server mysqld/node-1" | socat stdio /var/lib/haproxy/stats
    ```

4. Put the node into maintenance mode for Pacemaker:

    ```
    ssh node-1
    crm node maintenance
    ```

where "node-1" is the name of the node from which you would like to move.

5. Stop data replication on the selected MySQL instance:

```
mysql -e "SET GLOBAL wsrep_on=off;"
```

6. Run the backup:

```
xtrabackup --backup --stream=tar ./ | gzip - > backup.tar.gz
```

7. Make a streaming backup as described in Percona Guide.
8. Move the archive file to a safe place.
9. Re-enable the data replication:

```
mysql -e "SET GLOBAL wsrep_on=on;"
```

10. Take the MySQL service out of maintenance mode with the following command for every controller in the cluster:

```
ssh -t node-1 'echo "enable server mysqld/node-1" | socat stdio /var/lib/haproxy/stats'
```

11. Put the node into the ready mode:

```
ssh -t node-1 crm node ready
```

where "node-1" is the node that you have backed up.

Restoring with Percona XtraBackup

1. Remove grastate.dat (e.g. move to a different place) on all nodes:

```
ssh node-1 mv /var/lib/mysql/grastate.dat /var/lib/mysql/grastate.old
ssh node-2 mv /var/lib/mysql/grastate.dat /var/lib/mysql/grastate.old
ssh node-3 mv /var/lib/mysql/grastate.dat /var/lib/mysql/grastate.old
```

2. Extract the database backup file on the first controller:

```
ssh node-1 'cd /var/lib/mysql/ ;tar -xvzf clear-base.tgz'
```

where "node-1" is the node that you have backed up.

3. Change the owner:

```
chown -R mysql:mysql /var/lib/mysql
```

4. Export the variables for *mysql-wss* on all nodes:

```
export OCF_RESOURCE_INSTANCE=p_mysql
export OCF_ROOT=/usr/lib/ocf
export OCF_RESKEY_socket=/var/run/mysqld/mysqld.sock
```

5. Export the variable for *mysql-wss on* the first node:

```
export OCF_RESKEY_additional_parameters="--wsrep-new-cluster"
```

6. Start *mysqld* on the first controller:

```
/usr/lib/ocf/resource.d/fuel/mysql-wss start
```

7. Start *mysqld* on all other controllers:

```
/usr/lib/ocf/resource.d/fuel/mysql-wss start
```

8. Copy the extracted database backup.
9. Check the crm status for all nodes.

Writing a bootable Fuel ISO to a USB drive

Having downloaded a Fuel ISO, and having plugged in your USB drive, issue the following command:

```
# dd if=/way/to/your/ISO of=/way/to/your/USB/stick
```

where */way/to/your/ISO* is the path to your Fuel ISO, and */way/to/your/USB/stick* is the path to your USB drive.

For example, if your Fuel ISO is in the */home/user/fuel-isos/* folder and your USB drive is at */dev/sdc*, issue the following:

```
# dd if=/home/user/fuel-isos/fuel-7.0.iso of=/dev/sdc
```

Note

This operation will wipe all the data you have on on the USB drive and will place a bootable Fuel ISO on it. You also have to write the ISO to the USB drive itself, not to a partition on it.

Deploying an Empty Role through Fuel CLI

Make sure there are zero environments:

```
[root@nailgun tmp]# fuel env
id | status | name | mode | release_id | pending_release_id
---|--------|------|------|------------|-------------------
```

Check the operating systems:

```
[root@nailgun tmp]# fuel release
id | name                 | state       | operating_system | version
---|----------------------|-------------|------------------|-------------
2  | Kilo on Ubuntu 14.04 | available   | Ubuntu           | 2015.1.0-7.0
1  | Kilo on CentOS 6.5   | unavailable | CentOS           | 2015.1.0-7.0
```

Note down the numbers under the `id` column. You will need these later.

Check the existing nodes:

```
[root@nailgun tmp]# fuel node
id | status   | name              | cluster | ip          | mac               | roles |
---|----------|-------------------|---------|-------------|-------------------|-------|
10 | discover | Untitled (8a:15) | None    | 10.109.0.4  | 64:dd:40:75:8a:15 |       |
8  | discover | Untitled (96:c1) | None    | 10.109.0.5  | 64:2e:f0:06:96:c1 |       |
9  | discover | Untitled (8b:4a) | None    | 10.109.0.3  | 64:7b:44:59:8b:4a |       |
7  | discover | Untitled (d2:bf) | None    | 10.109.0.12 | 64:79:31:7a:d2:bf |       |

pending_roles | online | group_id
--------------|------------------
       True   | None
       True   | None
       True   | None
       False  | None
```

There are three nodes online: 8,9,10

Create a new environment:

```
[root@nailgun tmp]# fuel env create --name test --release 1
Environment 'test' with id=4, mode=ha_compact and network-mode=neutron was created!
```

Check if the environment has been created:

```
[root@nailgun tmp]# fuel env
id | status | name | mode       | release_id | pending_release_id
```

```
---|--------|------|------------|------------|-------------------
4  | new    | test | ha_compact | 1          | None
```

Note down the `id` of the environment. You will need this later.

Check the existing roles:

```
[root@nailgun tmp]# fuel role --release 1
name          | id
--------------|---
controller    | 1
compute       | 2
cinder        | 3
cinder-vmware | 4
ceph-osd      | 5
mongo         | 6
base-os       | 7
```

The role that you need is `base-os`.

Add the node whose `id` is 8 and the role is `base-os` to the environment whose `id` is 4:

```
[root@nailgun tmp]# fuel node set --env 4 --node 8 --role base-os
Nodes [8] with roles ['base-os'] were added to environment 4
```

Check the results:

```
[root@nailgun tmp]# fuel node
id | status   | name              | cluster | ip          | mac               | roles |
---|----------|-------------------|---------|-------------|-------------------|-------|
10 | discover | Untitled (8a:15) | None    | 10.109.0.4  | 64:dd:40:75:8a:15 |       |
8  | discover | Untitled (96:c1) | 4       | 10.109.0.5  | 64:2e:f0:06:96:c1 |       |
9  | discover | Untitled (8b:4a) | None    | 10.109.0.3  | 64:7b:44:59:8b:4a |       |
7  | discover | Untitled (d2:bf) | None    | 10.109.0.12 | 64:79:31:7a:d2:bf |       |

 pending_roles | online | group_id
---------------|--------|---------
               | True   | None
 base-os       | True   | 4
               | True   | None
               | False  | None
```

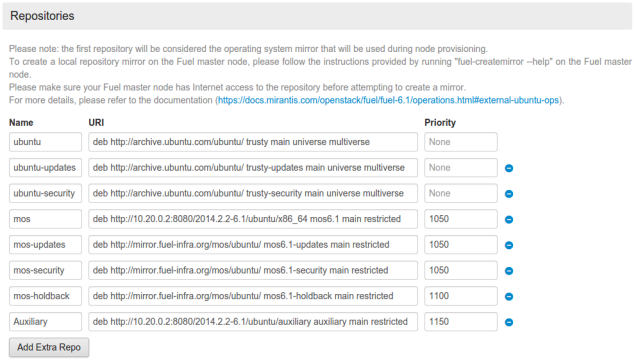
Your node with an empty role has been added to the cluster.

Configuring repositories

You may need to configure repositories to:

- *Download Ubuntu packages*
- *Apply patches*

By default, your environments will have the configuration of the repositories that point to the Mirantis update and security repository mirrors. There is also an 'Auxiliary' repository configured on the Fuel Master node which can be used to deliver packages to the nodes.



To change the list of repositories, you will need to amend the three fields which contain the required information for the repositories configuration depending on the distribution you install.

For Ubuntu

```
|repo-name|apt-sources-list-string|repo-priority|

my-repo deb http://my-domain.local/repo trusty main 1200
```

Repository priorities

The process of setting up repositories and repository priorities is the same one you normally do on your Linux distribution.

For more information, see the documentation to your Linux distribution.

Downloading Ubuntu system packages

In Fuel 6.0 and older there is an option to select an Ubuntu release in Fuel and deploy it, since all the Ubuntu packages are located on the Fuel Master node by default.

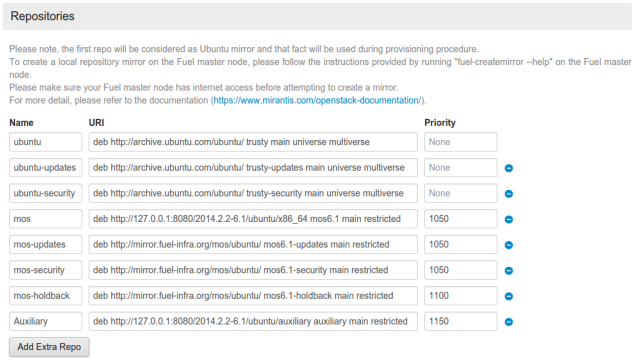
Now all Ubuntu packages are downloaded from the Ubuntu official mirrors by default, but you can specify another mirror in Fuel UI or by using Fuel CLI.

Updates to the Mirantis packages are fetched from the Mirantis mirrors by default.

> ***Note***
>
> To be able to download Ubuntu system packages from the official Ubuntu mirrors and Mirantis packages from the Mirantis mirrors you need to make sure your Fuel Master node and Slave nodes have Internet connectivity.

To change the Ubuntu system package repositories from the official ones to your company's local ones, do the following:

1. In Fuel web UI, navigate to the **Settings** tab and then scroll down to the **Repositories** section.
2. Change the path under **URI**:



> ***Note***
>
> You can also change the repositories after a node is deployed, but the new repository paths will only be used for the new nodes that you are going to add to a cluster.

See also *Configuring repositories*.

There is also a `fuel-createmirror` script on the Fuel Master node that you can use to synchronize Ubuntu packages to the Fuel Master node.

Setting up local mirrors

You can create and update local mirrors of Mirantis OpenStack and/or Ubuntu packages using the `fuel-createmirror` script.

> ***Note***
>
> The script supports only rsync mirrors. Please refer to the official upstream Ubuntu mirrors list.

The script uses a Docker container with Ubuntu to support dependencies resolution.

The script can be installed on any Red Hat based or Debian based system. On a Debian based system it requires only bash and rsync. On a Red Hat based system it also requires docker-io, dpkg, and dpkg-devel packages (from Fedora).

When run on the Fuel Master node, the script will attempt to set the created Mirantis OpenStack and/or Ubuntu local repositories as the default ones for new environments, and apply these repositories to all the existing environments in the "new" state. This behavior can be changed by using the command line options described below.

The script supports running behind an HTTP proxy as long as the proxy is configured to allow proxying to Port 873 (rsync). The following environment variables can be set either system-wide (via ~/.bashrc), or in the script configuration file (see below):

```
http_proxy=http://username:password@host:port/
RSYNC_PROXY=username:password@host:port
```

You may also want to configure Docker to use the proxy to download the Ubuntu image needed to resolve the packages dependencies. Add the above environment variables to the file */etc/sysconfig/docker*, and export them:

```
http_proxy=http://username:password@host:port/
RSYNC_PROXY=username:password@host:port
export http_proxy RYSNC_PROXY
```

Then, restart the docker daemon:

```
service docker restart
```

Or alternatively (recommended), reboot the Fuel Master node.

Issue the following command to get the *fuel-createmirror* help:

```
fuel-createmirror -h
```

OR

```
fuel-createmirror --help
```

To create or update a local Mirantis OpenStack mirror only, issue:

```
fuel-createmirror -M
```

OR

```
fuel-createmirror --mos
```

To create or update a local Ubuntu mirror only, issue:

```
fuel-createmirror -U
```

OR

```
fuel-createmirror --ubuntu
```

If no parameters are specified, the script will create/update both Mirantis OpenStack and Ubuntu mirrors.

Note

Options `-M/--mos` and `-U/--ubuntu` can't be used simultaneously.

To disable changing the default repositories for new environments, issue:

```
fuel-createmirror -d
```

OR

```
fuel-createmirror --no-default
```

To disable applying the created repositories to all environments, in the "new" state, issue:

```
fuel-createmirror -a
```

OR

```
fuel-createmirror --no-apply
```

> ***Note***
>
> If you change the default password (admin) in Fuel web UI, you will need to run the utility with the `--password` switch, or it will fail.

The following configuration file can be used to modify the script behavior:

```
/etc/fuel-createmirror/common.cfg
```

In this file you can redefine the upstream mirrors, set local paths for repositories, configure the upstream packages mirroring mode, set proxy settings, enable or disable using Docker, and set a path for logging. Please refer to the comments inside the file for more information.

The following configuration file contains the settings related to Fuel:

```
/etc/fuel-createmirror/fuel.cfg
```

If you run the script outside of Fuel node, you may need to redefine the FUEL_VERSION and the FUEL_SERVER parameters.

Installing on a Red Hat based server

1. Configure MOS RPM repository:

```
tee /etc/yum.repos.d/mos-rpm.repo <<EOF
[mos-rpm]
name=MOS RPM packages
baseurl=http://mirror.fuel-infra.org/fwm/6.1/centos/os/x86_64
gpgcheck=0
enabled=0
EOF
```

2. Install the package and its dependencies:

```
yum --enablerepo=mos-rpm install fuel-createmirror
```

3. Check and configure the settings in `/etc/fuel-createmirror/common.cfg`.
4. Make sure the Docker service is up and running.
5. Run `fuel-createmirror`

Debian-based server

1. Configure MOS DEB repository:

```
echo "deb http://mirror.fuel-infra.org/mos/ubuntu/ mos6.1 main restricted"\
| sudo tee /etc/apt/sources.list.d/mos-deb.list
```

2. Make `apt-get update`, then install the package `apt-get install fuel-createmirror`
3. Check and configure the settings in `/etc/fuel-createmirror/common.cfg`.
4. Run `fuel-createmirror`

Troubleshooting partial mirror

If there some packages required by your installation missing from from the partial mirror created by the script, add them to `/etc/fuel-createmirror/requirements-deb.txt`.

The package format to add to the `requirements-deb.txt` file is simple:

```
package1
package2
...
packageN
```

You can also look up the package names at the official Ubuntu website.

Having done that, restart the script. This will download all the missing packages and recreate a local partial mirror.

Applying patches

This section describes how to apply, rollback, and verify the patches applied to the Fuel Master node and the Fuel Slave nodes.

Introduction

Patching in brief:

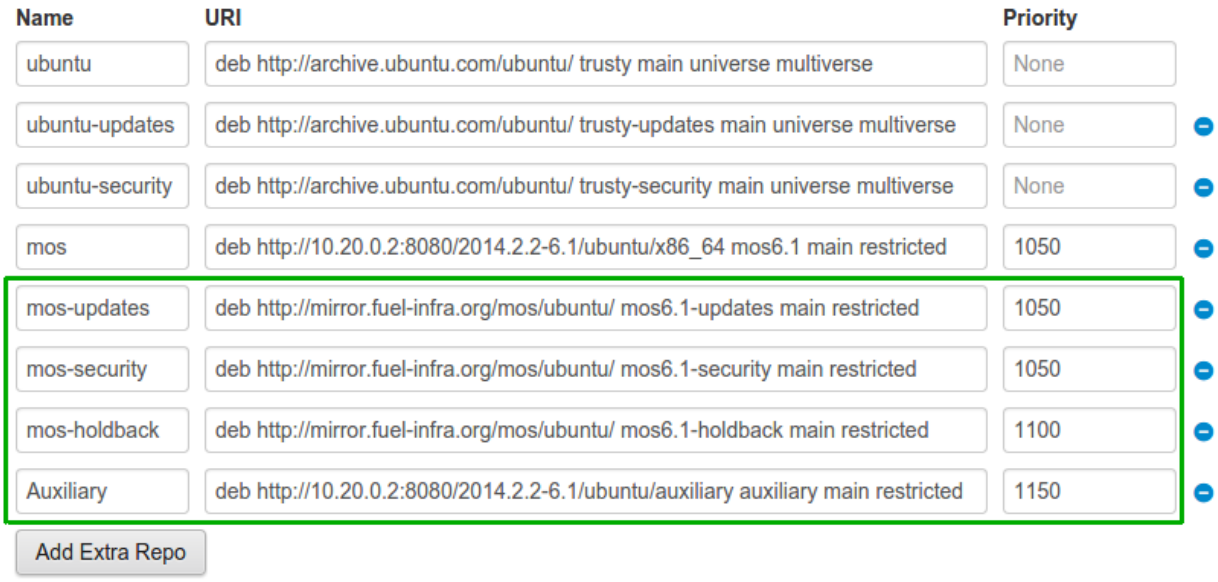
- The patching feature was introduced in Mirantis OpenStack 6.1 and will not work in older releases.
- There are two types of patches: bugfixes and security updates.
- Patches are downloaded from the Mirantis public repositories.
- You can always check what patches are available and get instructions on how to apply them at the Maintenance Update section of the Release Notes.
- The changes that the patches introduce will be applied to the new OpenStack environments.

Usage scenarios

Default scenario

In the default scenario you download patches from the default Mirantis mirrors.

- Make sure you are registered at at the official Mirantis website.
- Once you are registered, you will receive regular email notifications on the available patches with a link to the documentation on how to apply the patches. The documentation is always available in the Maintenance Update section of the Release Notes.
- Your repos by default are configured to download the patches from Mirantis. The patching repos are:



- Check each patching item and proceed with the instructions (plan accordingly: for example, schedule a maintenance slot to run the update).
    - Patching Fuel Master node:
        - Run the command specified in the documentation to install the patch.
        - After the patch is installed, restart the affected service as specified in the documentation.
    - Patching a slave node:
        - Run the command specified in the documentation to download the patch.
        - Run the command specified in the documentation to install the patch.

Custom scenario: deploying from local mirrors; patching from local mirrors

In this custom scenario you deploy from your local mirrors and download patches from your local mirrors.

For information on how to create and update local mirrors of Mirantis OpenStack see *Configuring repositories*.

- Make sure you are registered at at the official Mirantis website.
- Once you are registered, you will receive regular email notifications on the available patches with a link to the documentation on how to apply the patches. The documentation is always available in the Maintenance Update section of the Release Notes.
- Check each patching item and proceed with the instructions (plan accordingly).
    - Patching Fuel Master node:
        - Make sure your local mirror is up to date: run `fuel-createmirror -M`
        - Run the command specified in the documentation to download the patch.
        - Run the command specified in the documentation to install the patch.
        - After the patch is installed, restart the affected service as specified in the documentation.
    - Patching a slave node:
        - Run the command specified in the documentation to download the patch.
        - Run the command specified in the documentation to install the patch.

Custom scenario: deploying from Mirantis mirrors; patching from local mirrors

In this custom scenario you deploy from Mirantis mirrors and download patches from your local mirrors.

- Make sure you are registered at at the official Mirantis website.
- Configure your local mirrors to download patches from Mirantis mirrors as described in *Configuring repositories*.
- Once you are registered, you will receive regular email notifications on the available patches with a link to the documentation on how to apply the patches. The documentation is always available in the Maintenance Update section of the Release Notes.
- Check each patching item and proceed with the instructions (plan accordingly).

- Patching Fuel Master node:
  - Make sure your local mirror is up to date: run `fuel-createmirror -M`
  - Run the command specified in the documentation to download the patch.
  - Run the command specified in the documentation to install the patch.
  - After the patch is installed, restart the affected service as specified in the documentation.
- Patching a slave node:
  - Run the command specified in the documentation to download the patch.

Additional information

Rolling back patches

> ***Note***
>
> Use the instructions listed here only for Mirantis OpenStack 6.1 and 7.0.

> ***Note***
>
> The rollback instructions listed here are for advanced administrators. If you are not sure how to plan and execute the rollbacks, your best option is to contact Mirantis support.

Rolling back Fuel Master node

- Roll back the packages on the Fuel Master node. Refer to this article as an example.
- Roll back all the changes to the configuration you made when applying the patching instructions.
- Run `dockerctl destroy all`.
- Run `dockerctl start all`.
- Wait for bootstrap to complete.

Rolling back an Ubuntu slave node

- Evacuate all the running resources from the node.
- Make sure new workloads are not scheduled to the node: Put nova services in maintenance, turn on Pacemaker into maintenance mode etc.
- Look up the packages you want to roll back in `/var/log/apt/history.log` and `/var/log/dpkg.log`.

- Figure out where to get the old package version. Run `apt-cache policy`.
- Figure out if the old package version is available locally.
- If it is, install these versions using dpkg. Otherwise, check the snapshots of previous repositories on *http://mirror.fuel-infra.org/mos/snapshots* and pick the repository that contains the packages you need.
- Add this repository to the environment configuration.
- On the Fuel Master node run:

```
fuel node --node-id <comma_separated_list_of_nodes_you_want_to_update_repo> \
--tasks upload_core_repos
```

  This will propagate the new repos configuration.
- Install the packages with specific versions:

  ```
  apt-get install <pkg1>=<ver1> <pkg2>=<ver2>
  ```
- Roll back all the changes to the configuration you made when applying the patching instructions.
- Reboot the node.

Applying all accumulated changes in one go

> ***Note***
>
> This set of actions should be applied carefully and with consideration. It is strongly recommended that you do this on your test staging environment before applying the updates to production.

It is a good practice to apply the updates node by node so that you can stop the update procedure whenever an issue occurs. It is also strongly recommended to back up all sensitive data that can be altered continuously during the whole lifetime of your environment and the Fuel Master node.

These instructions assume that if you add any custom repositories to your environment configuration, these commands will update your environment taking packages from these repositories.

Patching Fuel Master node

- Back up your data with dockerctl backup. This will save the data to `/var/backup/fuel/`.
- Run `yum update`.
- Run `dockerctl destroy all`.
- Run `dockerctl start all`.
- Wait for the new containers deployment to finish.

Patching an Ubuntu slave node

- Run `apt-get update`.
- Run `apt-get upgrade`.
- Apply all the additional configuration options as described in the supporting documentation.
- Reboot the node.

Applying Puppet changes on a slave node

You may want to apply all changes on a slave node or run a single granular task so that Fuel Puppet changes take effect.

To run a complete Puppet cycle on a slave node, run:

- Update fuel-libraryX.X on Fuel Master `yum update`
- Run `fuel node --node NODE_ID --deploy`

If you want to just update Puppet manifests and apply a single task, then run:

- Update fuel-libraryX.X on Fuel Master `yum update`
- Run `fuel node --node node-XX --task rsync_core_puppet hiera globals TASK`

> ***Note***
>
> The tasks rsync_core_puppet, hiera, and globals are required for processing any Puppet changes.

Verifying the installed packages on the Fuel Master node

After you apply a patch to the Fuel Master node, you can verify that the Fuel Master node is using the latest packages.

To verify the packages on the Fuel Master node:

1. Log in to the Fuel Master node CLI.
2. Type:

```
yum clean expire-cache
yum -y update
```

Verifying the installed packages on the Fuel Slave nodes

When you apply a patch to the Fuel Slave nodes, ensure that the versions of packages on all Fuel Slave nodes are identical. Therefore, verify that the Fuel Slave nodes within one OpenStack environment have the same repository configuration, as well as the same versions of packages are installed on all nodes.

To verify the packages are up-to-date on the Fuel Slave nodes:

1. Log in to the Fuel Master node CLI.

2. Update the list of available packages:

```
apt-get update
```

3. Update all packages:

```
apt-get upgrade
```

4. Log in to the Fuel Master node GUI:
5. Click **Support**.
6. Generate and download a diagnostic snapshot by clicking **Generate Diagnostic Snapshot**.

   The Fuel Master node generates `ubuntu_installed_debs.txt`.
7. Analyze `ubuntu_installed_debs.txt` to verify the versions of the packages.

   Additionally, you can analyze the `ubuntu_repo_list.txt` file to verify the repositories.

Using the reduced footprint feature

With the reduced footprint feature you can spawn virtual machines on nodes.

This can be useful in the following scenarios (but not limited to them):

- Run a minimal two node cluster on a single physical machine.
- Put external services on the spawned virtual machines (e.g. a monitoring service).
- Run three controllers on virtual machines on three different physical machines.

Reduced footprint flow in brief

Minimal requirements:

- Two bare metal nodes. Alternatively, you can have one virtual machine (with Fuel installed on it) and one bare metal.
- Fuel 7.0 ISO.

Deployment flow:

1. Install Fuel on a bare metal or virtual machine.
2. Boot another bare metal machine via Fuel PXE.
3. Enable the **Advanced** feature group in Fuel.
4. Create a new environment in Fuel.
5. Optionally, modify the libvirt VM template on the Fuel Master node: `/etc/puppet/modules/osnailyfacter/templates/vm_libvirt.erb` The default template supports tunneling segmentation. If you use VLAN segmentation, change the bridge name 'br-mesh' to 'br-prv' and set type to 'openvswitch'. For example:

   ```
   <interface type='bridge'>
      <source bridge='br-prv'/>
      <virtualport type='openvswitch'/>
      <model type='virtio'/>
   </interface>
   ```

6. If you use tagged VLANs (VLAN segmentation or 'Use VLAN tagging' in the "Networks" tab), you should upload a network template. For details see *Using Networking Templates*. See also network template samples for reduced footprint:
   - VLAN segmentation
   - VLAN tagging
7. Assign the "virt" role to the discovered node.
8. Upload the virtual machine configuration to Fuel.
9. Provision the bare metal node with the "virt" role. This will also spawn the virtual machines.
10. Assign roles to the spawned and discovered virtual machines.

11. Deploy the environment.
12. Migrate the Fuel server as an additional virtual machine located on the physical server.

Reduced footprint flow detailed

1. Install Fuel on a bare metal or virtual machine. For details see *Download and Install Fuel*.
2. Boot another bare metal machine via Fuel PXE. For details see *Boot the node servers*.
3. Enable the **Advanced** feature group in Fuel. On the Fuel Master node edit the `/etc/fuel/version.yaml` file and add `advanced` under `feature groups` there. Here is a sample:

   ```
   VERSION:
     feature_groups:
       - mirantis
       - advanced
   ```

   Having added "advanced" to the yaml file, issue the following commands:

   ```
   dockerctl shell nailgun
   supervisorctl restart nailgun
   ```

4. Create a new environment in Fuel. For details see *Create a new OpenStack environment*.
5. Assign the "virt" role to the discovered node. On the Fuel Master node, issue the following command:

   ```
   fuel --env-id=<ENV_ID> node set --node-id=<NODE_ID> --role=virt
   ```

   where <NODE_ID> points to a specific node identified by its ID (a number) that you can get by issuing the `fuel nodes` command; <ENV_ID> points to the environment ID respectively; you can get the environment ID by issues the `fuel environment` command.

   For example:

   ```
   fuel --env-id=1 node set --node-id=1 --role=virt
   ```

6. Upload the virtual machine configuration to Fuel. On the Fuel Master node, issue the following command:

   ```
   fuel2 node create-vms-conf <NODE_ID> --conf '{"id":<VM_ID>, \
   "mem":<MEMORY_SIZE>,"cpu":<CPU_CORE_COUNT>}'
   ```

   For example:

   ```
   fuel2 node create-vms-conf 2 --conf '{"id":1,"mem":2,"cpu":4}'
   ```

   where <NODE_ID> is "virt" node ID, <VM_ID> is VM ID that should be unique on that "virt" node, <MEMORY_SIZE> is the memory amount in gigabytes, and <CPU_CORE_COUNT> is the number of CPUs.

7. Provision the bare metal node with the virtual role and spawn virtual machines. At this point you can go back to the Fuel UI. On the Dashboard there you will see the **Provision VMs** button that you need to click. Alternatively, you can do this through Fuel CLI on the Fuel Master node by issuing the following command:

   ```
   fuel2 env spawn-vms <CLUSTER_ID>
   ```

   For example:

   ```
   fuel2 env spawn-vms 1
   ```

8. Assign controller roles to the spawned virtual machines. For details see *Assign a role or roles to each node server*. Alternatively, you can do this through Fuel CLI by issuing the following command:

   ```
   fuel --env-id=<ENV_ID> node set --node-id=<NODE_ID> --role=controller
   ```

   You can specify several nodes with the `--node-id` parameter. For example:

   ```
   fuel --env-id=1 node set --node-id=2,3,4 --role=controller
   ```

9. Deploy the environment. For details see *Deploy Changes*. Alternatively, you can do this through Fuel CLI by issuing the following command:

   ```
   fuel --env <ENV_ID> node --deploy --node-id=<NODE_ID>
   ```

   You can specify several nodes with the `--node-id` parameter. For example:

   ```
   fuel --env 1 node --deploy --node-id=1,2,3,4
   ```

10. Use the fuel-migrate scrpt to migrate the Fuel Master node into into a virtual machine on a compute node. This allows for reduced resource use in small environments and lets the Fuel Master node run on physical or virtual machines by essentially making it host agnostic.

    To run the script issue the following command:

    ```
    fuel-migrate
    ```

    Note

    This will give you all the available parameters to properly do the migration with the `fuel-migrate` script.

Simple usage scenario:

1. Identify the node with the compute role by issuing the following command on the Fuel Master node (and checking its output):

   ```
   fuel node
   ```

2. Run the migration script:

   ```
   fuel-migrate <DESTINATION_COMPUTE>
   ```

   where <DESTINATION_COMPUTE> is the name or IP address of the destination compute node where the virtual machine will be created.

   For example:

   ```
   fuel-migrate node-1
   ```

   Or:

   ```
   fuel-migrate 192.168.116.1
   ```

   > ***Note***
   >
   > You can get the node name or the IP address by issuing the `fuel node` command.

Once you start the script, it will do the following:

1. Create a blank disk image on the destination node, define the virtual machine, start the virtual machine, and boot with Fuel PXE server.
2. Partition the disk on the destination node.
3. Reboot the Fuel Master node into maintenance mode and synchronize the data.
4. Swap the IP address on the source and destination Fuel Master nodes. It will then reboot the destination virtual machine.

An indication of that the script has run successfully will be the following message (with additional details on how to proceed) after you log in to the Fuel Master node via SSH:

```
Congratulation! You are on cloned Fuel now!
The migration tasks have completed. The clone should be up and
functioning correctly.
```

Additional notes:

- You can define the destination disk size in gigabytes with the `--fvm_disk_size` parameter.

For example:

```
fuel-migrate node-1 --fvm_disk_size=50g
```

- By default, the destination node will use the admin network interface. If you need to create additional interfaces, you can do so with the `--other_net_bridges` parameter.

  For example:

  ```
  fuel-migrate node-1 --other_net_bridges=eth1,,virbr13
  ```

  > ***Note***
  >
  > Pay attention that `--other_net_bridges` uses three parameters, and if you skip one of these as in this example, you still need to separate it with commas `,,`.

- By default, the migration log file is /var/log/fuel-migrate.log. If the migration fails, check the log for errors.

Custom usage example:

```
fuel-migrate 192.168.116.1 --admin_net_br=virbr12 --del_vm \
--other_net_bridges=eth1,,virbr13 --fvm_disk_size=100g \
--dkvm_folder=/var/lib/libvirt/images/
```

This example will do the following:

1. Set the destination compute node with the IP address 192.168.116.1
2. Use virbr12 on the host to connect to the admin interface (which is public network connected to the current Fuel Master node in this case).
3. Remove the destination virtual machine if it exists.
4. Use virbr13 for Ethernet 1.
5. Set the destination disk size to 100 GB.
6. Set the path to the folder on KVM hist where disk will be created to `/var/lib/libvirt/images/`

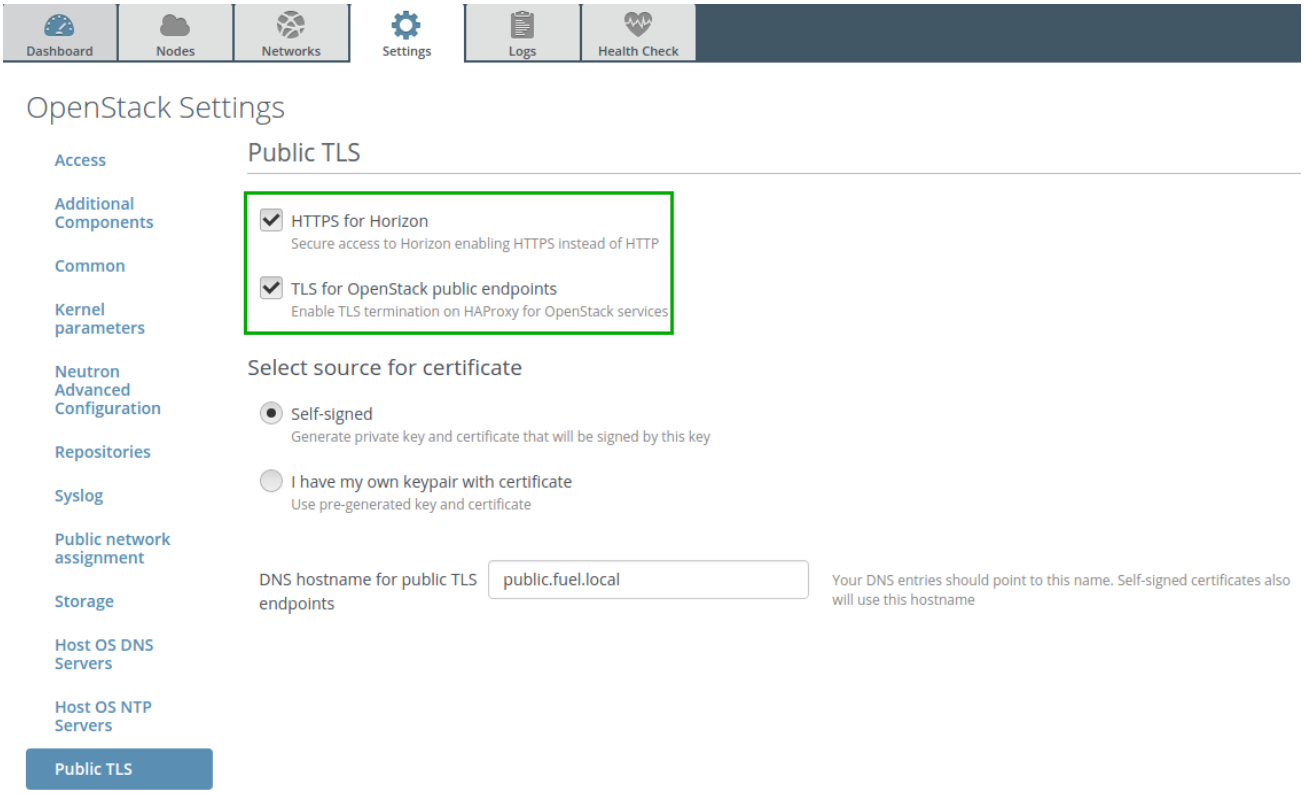
Switching on SSL and Secure Access

Starting with Fuel 7.0 there are two secure access options that you can enable on the *Settings tab* of Fuel Web UI:

- You can switch on SSL for the Horizon dashboard and the OpenStack publicURL endpoints.
- HTTPS access to the Fuel Master node.

Horizon dashboard and the OpenStack publicURL endpoints

Secure access is enabled by default:



The "HTTPS for Horizon" checkbox enables SSL access to the Horizon dashboard.

> ***Note***
>
> With the HTTPS enabled, you will not be able to access the Horizon dashboard through plain HTTP. You will automatically be redirected to HTTPS port 443.

The "TLS for OpenStack public endpoints" checkbox enables access to publicURL enpoints through HTTPS.

> ***Note***
>
> With the "TLS for OpenStack..." enabled, you will not be able to access the public endpoints through plain HTTP.

After enabling one or both of the secure access options, you will need to generate or upload a certificate and update your DNS entries:

1. Select the certificate source:
    - Self-signed -- The certificate will be generated before the environment deployment.
    - I have my own keypair with certificate -- You will need to upload a file with the certificate information and a private key that can be consumed by HAProxy. For detailed information read HOWTO SSL NATIVE IN HAPROXY.
2. Update your DNS entries -- Set the DNS hostname for public TLS endpoints. This hostname will be used in the two following cases:
    - When setting up DNS in the cluster.
    - As a name for OpenStack services when adding them to Identity. For example, you will see this name when you issue the `keystone --endpoint-list` command on one of the Controllers in a deployed cluster.

HTTPS access to the Fuel Master node

You can now access the Fuel Master node through HTTPS. To do this, you will need to use port 8443: https://10.20.0.2:8443

Additional information

- Changing keypairs for a cluster -- There is currently no automated way to do this. You can manually change the keypairs in `/var/lib/astute/haproxy/public_haproxy.pem` on Controller and Compute nodes. Make sure you restart the HAProxy service after you edit the file.
- Changing keypairs for the Fuel Master node -- You need to write the key to `/var/lib/fuel/keys/master/nginx/nginx.key` and the certificate to `/var/lib/fuel/keys/master/nginx/nginx.crt`. Make sure you restart nginx after that.
- Making access to the Fuel Master node HTTPS only -- Edit the `/etc/fuel/astute.yaml` file so that it contains the following:

```
SSL:
force_https: true
```

Note

Currently Fuel CLI does not support HTTPS. You will also need to update `fuel-nailgun-agent` on all nodes deployed with older than 7.0, otherwise they will be reported as inactive.

Using Networking Templates

Starting with Fuel 7.0 you can use networking templates. Templates allow for more flexible network configurations and provide you with the following abilities:

- Ability to create additional networks (e.g. an extra network for Swift) and delete networks.
- Have a specific set of network roles.
- Ability to create a network only if a relevant node role is present on the node.
- Ability to provide custom networking topologies (e.g. subinterface bonding).

Networking Templates Limitations

- Interdependencies between templates for different node roles cannot be set.
- Network roles to networks mapping and network topology cannot be set for nodes individually. They can only be set for node role or/and node group.
- There is no UI support for networking templates. You can only operate via CLI or API. The "Configure Interfaces" tab of Fuel Web UI will become inactive after you upload a networking template.

> ***Note***
>
> If you delete the template, Fuel's default network solution will automatically become live and all network related sections in Fuel Web UI will become available again.

Working with Networking Templates

A networking template is a YAML file in the following format:

```
network_template_<ENV_ID>.yaml
```

where <ENV_ID> is the ID of your OpenStack environment that you can get by issuing the `fuel environment` command.

For example, if the ID of your environment is `1`, the name of the template must be `network_template_1.yaml` to operate with the template via Fuel CLI.

Networking Templates Samples

You can download samples from the network_templates repository folder.

> ***Note***
>
> There is no default or generated template in your Fuel installation provided by default.

Networking Templates Structure

Each template consists of five major sections.

- `adv_net_template` -- This is the network configuration template for the environment. The template operates with *node groups*. Sample:

```
adv_net_template:
  default: # name of the node group
    nic_mapping:
      ...
    templates_for_node_role:
      ...
    network_assignments:
      ...
    network_scheme:
      ...
  group_11: # name of the node group
    nic_mapping:
    templates_for_node_role:
    network_assignments:
    network_scheme:
```

  The following four sections are defined for each node group in the environment. Definitions from the `default` node group will be used for the node groups not listed in the template.

- `nic_mapping` -- Aliases to NIC names mapping are set here.
- `templates_for_node_role` -- List of template names for every node role used in the environment.
- `network_assignments` -- Endpoints used in the template body. This is where the mapping is set between endpoints and network names to set the L3 configuration for the endpoints.
- `network_scheme` -- Template bodies for every template listed under `templates_for_node_role`

nic_mapping section detailed

Sample:

```
nic_mapping:
default:
      if1: eth0
      if2: eth1
```

```
      if3: eth2
      if4: eth3
node-33:
      if1: eth1
      if2: eth3
      if3: eth2
      if4: eth0
```

NIC aliases (e.g. "if1") are used in templates in the topology description in the `transformations` section. With `nic_mapping` you can set mapping of aliases to NIC names for different nodes.

The `default` mapping is set for all nodes that do not have name aliases. Custom mapping can be set for any particular node (by node name).

The number of NICs for any node may vary. It depends on the topologies defined for the nodes in templates in the `transformations` section.

Use of aliases in templates is optional. You can use NIC names if all nodes have the same set of NICs and they are connected in the same way.

templates_for_node_role section detailed

Sample:

```
templates_for_node_role:
controller:
      - public
      - private
      - storage
      - common
compute:
      - common
      - private
      - storage
ceph-osd:
      - common
      - storage
```

This is where you provide the list of template names for every node role used in the environment.

The order of templates matters. The description of the topology that is in the `transformations` section of the template is executed by Puppet in the order provided on its input. Also, the order of creating the networking objects cannot be arbitrary. For example, a bridge should be created first, and the subinterface that will carry its traffic should be created after that.

While templates can be reused for different node roles, each template is executed once for every node.

When several roles are mixed on one node, an alphabetical order of node roles is used to determine the final order of the templates.

network_assignments section detailed

Sample:

```
network_assignments:
storage:
      ep: br-storage
private:
      ep: br-prv
public:
      ep: br-ex
management:
      ep: br-mgmt
fuelweb_admin:
      ep: br-fw-admin
```

Endpoints are used in the template body. The mapping is set here between endpoints and network names to get the networks' L3 configuration to be set for endpoints.

The sample above shows the default mapping which is set without a template. The set of networks can be changed using API: networks can be created or deleted via API.

network_scheme section detailed

Sample:

```
network_scheme:
  storage: # template name
      transformations:
          ...
      endpoints:
          ...
      roles:
          ...
  private:
      transformations:
          ...
      endpoints:
          ...
      roles:
          ...
  ...
```

Each template has a name which is referenced in the sections above and consists of the three following sections:

- `transformations` -- A sequence of actions to build proper network topology is defined here. The "transformation" from physical interface to endpoint is described here. The transformations are applied by the Puppet l23network module and must be compatible with it.
- `endpoints` -- All endpoints introduced by the template.

- `roles` -- The mapping of network roles to endpoints. When several templates are used for one node there should be no contradictions in this mapping.

Operating with Networking Templates

> ***Note***
>
> The order in which you add or remove networks and load the the template does not matter. However, adding or removing networks will not make sense if a template is not uploaded for the environment at all, because the default network solution takes into account only the networks created by default.

To upload a networking template, on the Fuel Master node issue the following command:

```
fuel --env <ENV_ID> network-template --upload --dir <PATH>
```

where where <ENV_ID> is the ID of your OpenStack environment that you can get by issuing the `fuel environment` command; <PATH> is the path to where your template is.

For example:

```
fuel --env 1 network-template --upload --dir /home/stack/
```

To download a networking template to the current directory, on the Fuel Master node issue the following command:

```
fuel --env <ENV_ID> network-template --download
```

For example:

```
fuel --env 1 network-template --download
```

To delete an existing networking template, on the Fuel Master node issue the following command:

```
fuel --env <ENV_ID> network-template --delete
```

For example:

```
fuel --env 1 network-template --delete
```

To create a network group, issue the following command:

```
fuel network-group --create --node-group <GROUP_ID> --name \
"<GROUP_NAME>" --release <RELEASE_ID> --vlan <VLAN_ID> \
--cidr <NETWORK_CIDR>
```

where <GROUP_ID> is the ID of your *Node group* that you can get by issuing the `fuel nodegroup` command; <GROUP_NAME> is the name that you would like to assign to your group; <RELEASE_ID> is the ID of your release; <VLAN_ID> is the VLAN ID; <NETWORK_CIDR> is an IP address with an associated routing prefix.

For example:

```
fuel network-group --create --node-group 1 --name \
"new network" --release 2 --vlan 100 --cidr 10.0.0.0/24
```

To list all available network groups issue the following command:

```
fuel network-group list
```

To filter network groups by node group:

```
fuel network-group --node-group <GROUP_ID>
```

For example:

```
fuel network-group --node-group 1
```

To delete network groups:

```
fuel network-group --delete --network <GROUP_ID>
```

For example:

```
fuel network-group --delete --network 1
```

You can also specify multiple groups to delete:

```
fuel network-group --delete --network 2,3,4
```

Network Template Examples

You can use network templates to configure Fuel to use one or two networks for all OpenStack network traffic.

Configuring Two Networks

Fuel supports the two-network configuration where one network interface is dedicated for PXE traffic and another network interface, or bonding, for all other traffic.

To configure two networks:

1. Create a new network for all non-PXE traffic:

```
# fuel network-group --create --name everything --cidr <cidr>
--gateway <gateway> --nodegroup <nodegroup>
```

2. Set the `render_addr_mask` parameter to *internal* for this network by typing:

```
# fuel network-group --set --network 39 --meta '{"name":
"everything", "notation": "cidr", "render_type": null, "map_priority": 2,
"configurable": true, "use_gateway": true, "render_addr_mask":
"internal", "vlan_start": null, "cidr": "10.108.31.0/24"}'
```

   This parameter is required by the Fuel library. The Fuel library requires a value called `internal_address` for each node. This value is set to the node's IP address from a network group which has `render_addr_mask` set to *internal* in its metadata. Therefore, update `render_addr_mask` for this network.

3. Save network template for two networks as `network_template_<env id>.yaml`.

> ***Note***
>
> Verify that `nic_mapping` matches your configuration.

4. Upload the network template by typing:

```
# fuel network-template --upload --env <env id>
```

5. Deploy the environment.
6. After Fuel completes the deployment, verify that only one bridge is configured by typing:

```
# ip -4 a

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
```

```
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
8: br-fw-admin: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
inet 10.108.5.3/24 brd 10.108.5.255 scope global br-fw-admin
valid_lft forever preferred_lft forever
16: vr-host-base: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
inet 240.0.0.5/30 scope global vr-host-base
valid_lft forever preferred_lft forever
30: hapr-host: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP gro
inet 240.0.0.1/30 scope global hapr-host
valid_lft forever preferred_lft forever
```

Configuring a Single Network

Fuel supports a single network configuration where one network interface is responsible for all OpenStack traffic. This configuration is common in the proof of concept deployments where no additional network interfaces are available.

To configure a single network:

1. Modify the admin network through the database by typing:

```
# dockerctl shell postgres
# sudo -u postgres psql nailgun
nailgun=# UPDATE network_groups SET meta='{"unmovable": true, "use_gateway":
    true, "notation": "ip_ranges", "render_addr_mask": "internal",
    "render_type": null, "map_priority": 0, "configurable": false}'
    WHERE id=1;
```

> ***Note***
>
> You cannot modify the admin network using CLI.

2. Save network template for one network as `network_template_<env id>.yaml`.
3. Upload the network template by typing:

```
# fuel network-template --upload --env <env id>
```

4. Deploy the environment.
5. Proceed to *Configure Neutron*.

Configure Neutron

After you deploy your environment, allocate the correct floating IP pool to the network.

To allocate the correct floating IP pool:

1. Clear the gateway from *router04*.
2. Delete the *net04_ext__subnet* subnet.
3. Create a new subnet with the floating IP pool from the single network.
4. Set gateway on *router04*.

Index