Murano Administrator's Guide

Murano Administrator's Guide

v0.3

Table of Contents

List of Tables

Chapter 1. General Deployment Steps

Prepare A Lab For Murano

This section provides basic information about lab's system requirements. It also contains a description of a test which you may use to check if your hardware fits the requirements. To do this, run the test and compare the results with baseline data provided.

Lab Requirements

Table 1.1. Hardware requirements

Criteria	Minimal	Recommended
CPU	4 core @ 2.4 GHz	24 core @ 2.67 GHz
RAM	8 GB	24 GB or more
HDD	2 x 500 GB (7200 rpm)	4 x 500 GB (7200 rpm)
RAID	Software RAID-1 (use mdadm as it will improve read performance almost two times)	Hardware RAID-10

There are a few possible storage configurations except the shown above. All of them were tested and were working well.

- 1x SSD 500+ GB
- 1x HDD (7200 rpm) 500+ GB and 1x SSD 250+ GB (install the system onto the HDD and mount the SSD drive to folder where VM images are)
- 1x HDD (15000 rpm) 500+ GB

The list of OSes which can be used for a lab shown below:

Table 1.2. OS Requirements

List
Ubuntu Server 12.04 LTS
Centos 6.4

Test Your Lab Host Performance

We have measured time required to boot 1 to 5 instances of Windows system simultaneously. You can use this data as the baseline to check if your system is fast enough.

You should use sysprepped images for this test, to simulate VM first boot.

Steps to reproduce test:

1. Prepare Windows 2012 Standard (with GUI) image in QCOW2 format. Let's assume that its name is ws-2012-std.qcow2
2. Ensure that there is NO KVM PROCESSES on the host. To do this, run command:

```
># ps aux | grep kvm
```

3. Make 5 copies of Windows image file:

```
># for i in $(seq 5); do \
cp ws-2012-std.qcow2 ws-2012-std-$i.qcow2; done
```

4. Create script start-vm.sh in the folder with .qcow2 files:

```
#!/bin/bash
[ -z $1 ] || echo "VM count not provided!"; exit 1
for i in $(seq $1); do
echo "Starting VM $i ..."
kvm \
  -m 1024 \
  -drive file=ws-2012-std-$i.qcow2,if=virtio \
  -net user -net nic,model=virtio \
  -nographic \
  -usbdevice tablet \
  -vnc :$i &
done
```

5. Start ONE instance with command below (as root) and measure time between VM's launch and the moment when Server Manager window appears. To view VM's desktop, connect with VNC viewer to your host to VNC screen :1 (port 5901):

```
># ./start-vm.sh 1
```

6. Turn VM off. You may simply kill all KVM processes by

```
># killall kvm
```

7. Start FIVE instances with command below (as root) and measure time interval between ALL VM's launch and the moment when LAST Server Manager window appears. To view VM's desktops, connect with VNC viewer to your host to VNC screens :1 thru :5 (ports 5901-5905):

```
># ./start-vm.sh 5
```

8. Turn VMs off. You may simply kill all KVM processes by

```
># killall kvm
```

Baseline Data

The table below provides baseline data which we've got in our environment.

Avg. Time refers to the lab with recommended hardware configuration, while **Max. Time** refers to minimal hardware configuration.

	Boot ONE instance	**Boot FIVE instances**
Avg. Time	3m:40s	8m
Max. Time	5m	20m

Host Optimizations

Default KVM installation could be improved to provide better performance.

The following optimizations may improve host performance up to 30%:

- change default scheduler from **CFQ** to **Deadline**
- use **ksm**
- use **vhost-net**

Install OpenStack

Warning

Please keep in mind that in this guide we assume that Murano is installed in a separate devbox environment.

Murano devbox requires an OpenStack installed somewhere in your lab. This OpenStack installation must be reachable from the Murano box. Murano works great with Openstack packages installation as well as devstack installation. For now we support Openstack Grizzly and working on Havana integration.

Package-based Installation

To install Openstack Grizzly please consult the following sources:

- OpenStack Grizzly Install Guide [https://github.com/mseknibilel/OpenStack-Grizzly-Install-Guide]
- OpenStack Grizzly-g3 for Ubuntu 12.04 all-in-one installation [http://longgeek.com/2013/03/18/openstack-grizzly-g3-for-ubuntu-12-04-all-in-one-installation-2/?lang=en]

In addition to that Heat [https://wiki.openstack.org/wiki/Heat] should be installed. Follow the link to setup Heat on Ubuntu [http://openstack.redhat.com/Deploy_Heat_and_launch_your_first_Application] and/or on CentOS [http://docs.openstack.org/developer/heat/getting_started/on_ubuntu.html].

Devstack-based Installation

For a Devstack installation please vizit Devstack Home Page [http://devstack.org/] and clone devstack repository to your host.

For further installation process we recommend to use Devstack's Single VM Installation Guide [http://devstack.org/guides/single-vm.html].

The example of localrc configuration file is provided below:

localrc example.

```
HOST_IP=
```

```
FLAT_INTERFACE=
FLOATING_RANGE=

ADMIN_PASSWORD=swordfish
MYSQL_PASSWORD=swordfish
RABBIT_PASSWORD=swordfish
SERVICE_PASSWORD=swordfish
SERVICE_TOKEN=tokentoken

ENABLED_SERVICES+=,heat,h-api,h-api-cfn,h-api-cw,h-eng

# Image's cache is in $TOP_DIR/files
IMAGE_URLS+=",http://fedorapeople.org/groups/heat/prebuilt-jeos-images/"
IMAGE_URLS+="F17-x86_64-cfntools.qcow2"

# /etc/nova/nova.conf
EXTRA_OPTS=(force_config_drive=true libvirt_images_type=qcow2 force_raw_images=false)

# Logging
SCREEN_LOGDIR=/opt/stack/log/
LOGFILE=$SCREEN_LOGDIR/stack.sh.log
```

RabbitMQ additional instance

RabbitMQ is used for services interconnection in the OpenStack. Murano also uses RabbitMQ as "message queue" service but the separate instance. In the OpenStack normal installation "message queue" service resides in the management network segment and should not be reachable from any tenant networks to prevent security breach. Murano uses its own agent service running on deploying instance directly. Agent should have the ability to communicate with "message queue" service. Create one more "message queue" service instance in the external network, reachable from tenant networks through the OpenStack network router service (Quantum/Neutron).

Configuration steps

- Create file `/etc/default/rabbitmq-murano` with options listed below

```
#!/bin/sh
#
#
export RABBITMQ_NODENAME=murano@$(hostname)
export RABBITMQ_CONFIG_FILE=/etc/rabbitmq/rabbitmq-murano
export RABBITMQ_ENABLED_PLUGINS_FILE=/etc/rabbitmq/enabled_p
CONTROL="${CONTROL} -n ${RABBITMQ_NODENAME}"
PID_FILE=/var/run/rabbitmq/murano.pid
```

- Make copy of the original rabbitmq-server init script:

```
cd /etc/init.d
cp rabbitmq-server rabbitmq-server-murano
```

- Make changes inside new file `rabbitmq-server-murano`, after test calls:

```
...
test -x $DAEMON || exit 0
test -x $CONTROL || exit 0
. /etc/default/rabbitmq-murano
RETVAL=0
...
```

Fill in configuration files for new RabbitMQ instance.

- Modify `/etc/rabbitmq/enabled_plugins.murano`

```
[rabbitmq_management]
```

- Modify `/etc/rabbitmq/rabbitmq-murano.config`

```
[
{rabbit, [
{tcp_listeners, [5674]},
{log_levels,[
{connection, error}
]}
]},
{rabbitmq_management, [
{listener, [{port, 15673}]}
]},
{rabbitmq_mochiweb, [
{listeners, [{mgmt, [{port, 55673}]}]}
]}
].
```

- Check that service works fine:

```
service rabbitmq-server-murano start
service rabbitmq-server-murano status
service rabbitmq-server-murano stop
```

- Enable service start at OS boot time:

```
update-rc.d rabbitmq-server-murano defaults
```

Warning

Don't forget about firewall rules for new RabbitMQ service!

Specify SecurityGroups quotas

Default quotas driver used by Neutron is `neutron.quota.ConfDriver`, all limits set in `/etc/neutron/ neutron.conf` - is not flexible. To extend functionality and flexibility, default quota driver should be changed to - `neutron.db.quota_db.DbQuotaDrive`.

- Change `/etc/neutron/neutron.conf` with the next values:

```
[QUOTAS]
...
#quota_driver = neutron.quota.ConfDriver
quota_driver = neutron.db.quota_db.DbQuotaDriver
...
```

- Restart all neutron services:

```
cd /etc/init.d/
for q in quantum-*; do restart $q; done
```

- Update required quota via Neutron CLI:

```
neutron quota-update --security_group 100 --tenant-id <tenant
+---------------------+-------+
| Field               | Value |
+---------------------+-------+
| floatingip          |    50 |
| network             |    10 |
| port                |    50 |
| router              |    10 |
| security_group      |   100 |
| security_group_rule |   100 |
| subnet              |    10 |
+---------------------+-------+
```

Reconfigure rate-limits for Nova

Please reconfigure rate-limits to at least 500-1000 hits per minute.

API calls rate limits could be configured using this manual [http://docs.openstack.org/grizzly/openstack-compute/admin/content/configuring-compute-API.html] or by disabling ratelimits in the `/etc/nova/api-paste.ini` file.

```
...
[composite:openstack_compute_api_v2]
use = call:nova.api.auth:pipeline_factory
#noauth = faultwrap sizelimit noauth ratelimit osapi_compute_app_v
#keystone = faultwrap sizelimit authtoken keystonecontext ratelimi
```

```
noauth = faultwrap sizelimit noauth osapi_compute_app_v2
keystone = faultwrap sizelimit authtoken keystonecontext osapi_com
keystone_nolimit = faultwrap sizelimit authtoken keystonecontext o
...
```

Configuring Neutron

Allow subnet ip-range overlapping

During environment deployment, Murano will create dedicated network for each of them, and every such network will have a subnet created. All these subnets will have identical ip-ranges. Theoretically this is perfectly fine, as these subnets belong to different isolated Networks (L2 segments) and are connected to different routers.

However, by default Neutron does not allow overlapping IPs for different subnets - even in different Networks. To override this restriction, change `/etc/neutron/neutron.conf`: uncomment `allow_overlapping_ips` parameter and change its value to `True`:

```
[DEFAULT]
...
# Enable or disable overlapping IPs for subnets
# Attention: the following parameter MUST be set to False if Neutr
# being used in conjunction with nova security groups
allow_overlapping_ips = True
```

Then, restart all neutron services:

```
cd /etc/init.d/
for q in quantum-*; do restart $q; done
```

Chapter 2. Install Murano Components

This chapter describes how to install Murano components on a separate devbox. We strongly recommend to use a separate host (virtual machine or real host) for Murano devbox as it prevents you from various dependency conflicts. You can refer this link [https://wiki.openstack.org/wiki/Murano/Requirements] to check supported version of all Murano dependencies.

Automatic Installation

There is a script to automate Murano installation onto devbox.

- Create a folder to hold cloned repositories

```
>#  mkdir -p /opt/git
```

- Clone murano-deployment repository

```
>#  cd /opt/git
>#  git clone git://github.com/stackforge/murano-deployment.git
```

- Set configuration and install prerequisites

```
>#  cd /opt/git/murano-deployment/devbox-scripts
>#  ./murano-git-install.sh prerequisites
```

Press Enter to edit `/etc/murano-deployment/lab-binding.rc`, then 'i' to enter INSERT mode. After editing press ESC and type :wq to write and exit from VI.

```
LAB_HOST=''

ADMIN_USER=''
ADMIN_PASSWORD=''

RABBITMQ_LOGIN=''
RABBITMQ_PASSWORD=''
RABBITMQ_VHOST=''
RABBITMQ_PORT=''
#RABBITMQ_HOST=''
#RABBITMQ_HOST_ALT=''

#FILE_SHARE_HOST=''

BRANCH_NAME='master'

# Only 'true' or 'false' values are allowed!
SSL_ENABLED='false'
```

```
SSL_CA_FILE=''
SSL_CERT_FILE=''
SSL_KEY_FILE=''

#BRANCH_MURANO_API=''
#BRANCH_MURANO_DASHBOARD=''
#BRANCH_MURANO_CLIENT=''
#BRANCH_MURANO_CONDUCTOR=''
#BRANCH_MURANO_REPOSITORY=''
```

- **LAB_HOST** - IP or hostname of the lab. This address/hostname should point to the host where Keystone is installed.
- **ADMIN_USER** - OpenStack admin user.
- **ADMIN_PASSWORD** - A password for OpenStack admin user.
- **RABBITMQ_USER** - User to connect to RabbitMQ host.
- **RABBITMQ_PASSWORD** - Password for that user.
- **RABBITMQ_VHOST** - vHost which will be used by Murano components. Provides additional layer of isolation from other devboxes.
- **RABBITMQ_PORT** - Port number for Murano components interconnection. It should be different from general RabbitMQ port to provide SSL opportunity.
- **RABBITMQ_HOST** - (optional) IP address or hostname of the host where RabbitMQ is installed IF it is not the same host as LAB_HOST points to. This parameter can be skipped and in this case LAB_HOST will be used.
- **RABBITMQ_HOST_ALT** - (optional) IP address or hostname of the RabbitMQ host to connect from inside the instance. In some cases the addresses like LAB_HOST or RABBITMQ_HOST are inaccessible from instances, and they must use different address.
- **FILE_SHARE_HOST** - (optional) IP address or hostname of the host where file share with prerequisites is located if it is not the same host as LAB_HOST points to.
- **BRANCH_MURANO_*** - (optional) code from specified branch name for a selected Murano component will be installed. By default code from 'master' branch will be fetched.
- **SSL_ENABLED** - Set '**true**' if OpenStack is configured with SSL support and '**false**' otherwise.
- **SSL_CA_FILE** - Path to CA certificate for certificate validation on client side. Leave it empty when used self-signed certificates.
- **SSL_CERT_FILE** - Path to the valid SSL certificate.
- **SSL_KEY_FILE** - Path to the valid key file.

- Install Murano components

```
>#  ./murano-git-install.sh install
```

- Login to the Dashboard using URL **http://your_VM_IP/horizon** on Ubuntu or **http://your_VM_IP/dashboard** on CentOS.

Manual Installation

This chapter describes manual installation and configuration of Murano services.

Note that all Murano modules can be downloaded from our page [https://launchpad.net/murano/] on launchpad.

Automatic installation

Murano can be installed in automatic way. Script will install all necessary packages to your system. Find out more about this in Getting Started Guide [http://murano-docs.github.io/0.4/getting-started/content/ch04s02.html]

Pre-Requisites

Murano supports the following operating systems:

1. Ubuntu 12.04
2. RHEL/CentOS 6.4

These system packages are required for Murano:

Ubuntu

1. gcc
2. python-pip
3. python-dev
4. libxml2-dev
5. libxslt-dev
6. libffi-dev

CentOS

1. gcc
2. python-pip
3. python-devel
4. libxml2-devel
5. libxslt-devel
6. libffi-devel

All these packages will be installed in murano-installation scripts. In addition to these packages some repositories are required. Please follow the instructions in the appendix to prepare your environment for murano installation.

Murano API Service

Murano API provides access to the Murano orchestration engine via API.

This chapter describes the procedure of installation and configuration of Murano API.

Install

- Superuser privileges is required to install and configure system packages. Let's switch to root account:

```
sudo su -
```

- Make sure that additional linux repositories are installed. See the appendix for information about preparing a virtual machine for murano installation.
- Clone Murano API git repository:

```
git clone https://github.com/stackforge/murano-api
```

Stable version one of our releases can be checked by the tag:

```
cd murano-api && git checkout 0.4
```

- And perform installation:

Ubuntu

```
sh setup.sh install
```

CentOS

```
sh setup-centos.sh install
```

- Successful installation ends with message like this:

```
Successfully installed muranoapi
Cleaning up...
LOG:> Making sample configuration files at "/etc/murano"
LOG:> Reloading initctl
```

```
LOG:> Please, make proper configuration,located at "/etc/murano", before starting
```

Configure

- Copy and edit configuration files:

```
cd /etc/murano
cp murano-api.conf.sample murano-api.conf
cp murano-api-paste.ini.sample murano-api-paste.ini
```

- Configure `murano-api.conf` according to your environment:
  - *[DEFAULT]* section sets up logging.
  - *[database]*sets database connection.
  - *[reports]* section defines names of RabbitMQ queues. This naming should correspond to a similar parameter in conductor config file.
  - In *[rabbitmq]* section you can set up host configuration where rabbitMQ with just created user and vhost is running. If you consider to use Murano in production it's better to use separate vhosts in RabbitMQ. To add new vhost and user with administrator rights perform:

```
rabbitmqctl add_user muranouser murano
rabbitmqctl set_user_tags muranouser administrator
rabbitmqctl add_vhost muranovhost
rabbitmqctl set_permissions -p muranovhost muranouser ".*" ".*" ".*"
```

  - *[ssl]* sets up SSL parameters - paths to required files in case of ssl connection. For more information how to configure SSL take a look at SSL configuration chapter
  - In *[keystone_authtoken]* configure parameters of Openstack Keystone service. For more information see Auth-Token Middleware with Username and Password [http://docs.openstack.org/developer/keystone/configuringservices.html]
  - Another murano-api configuration file located at `/etc/murano/murano-api-paste.ini` and does not require any changes.
-

  Register murano-api service in Openstack. To do that perform the following commands:

  ### Note

  You need to be authorized in Openstack to run this commands

```
$ keystone service-create --name muranoapi --type murano --description "Murano-Ap

$ keystone endpoint-create
 --region RegionOne
 --service-id The ID field returned by the keystone service-create
 --publicurl http://x.x.x.x:8082 (where x.x.x.x - host ip where murano-api instal
 --internalurl the same as publicurl
 --adminurl the same as publicurl
```

Run

- Run Murano API service:

  Ubuntu

  ```
  service murano-api start
  ```

  CentOS

  ```
  initctl start murano-api
  ```

Conductor Service

Conductor is a Murano orchestration engine that transforms object model sent by REST API service into a series of Heat and Murano-Agent commands.

This chapter describes Conductor for contributors of the project.

Install

- Murano Conductor uses OpenStack Heat for new virtual machines creation, therefore Heat should been installed and configured. Some services require the Internet access for virtual machines to successful deployment.

  The detailed information about Heat configuration is described here. [http://docs.openstack.org/developer/heat/getting_started/index.html]
- OpenStack Heat requires Key Pair for Load Balancer instances. Murano Conductor uses LoadBalancer for IIS Farms and ASP.NET Farms.
- Superuser privileges is required to install and configure system packages. Let's switch to root account:

  ```
  sudo su -
  ```

- Make sure that additional repositories are installed. See the appendix for information about preparing a virtual machine for murano installation.
- Clone Murano Conductor repository from the github.

```
git clone https://github.com/stackforge/murano-conductor
```

Stable version one of our releases can be checked out by tag:

```
cd murano-conductor && git checkout 0.4
```

- And then perform installation

Ubuntu

```
sh setup.sh install
```

CentOS

```
sh setup-centos.sh install
```

Configure

- Copy example of the configuration file:

```
cd /etc/murano
cp conductor.conf.sample conductor.conf
```

- Configure `conductor.conf` file according to your environment.
  - *[DEFAULT]* section is responsible for logging.
  - *[keystone]* defines where kestone is located.
  - *[heat]*points where heat is running.
  - *[neutron]* sets up parameters for interconnection with neutron.
  - *[rabbitmq]*section points where your rabbitMQ installed and configured.

```
[DEFAULT]

# Path where log will be written
log_file = /tmp/conductor.log

# Log verbosity
debug=True
verbose=True

# Provide directory with initialization scripts
init_scripts_dir = etc/init-scripts

# Provide directory with agent configs
agent_config_dir = etc/agent-config

# Directory for data cache, OS temp directory is used by default
#data_dir = /tmp/muranoconductor-cache

# Provide url to Murano Metadata repository
# Comment this line if you registered murano-metadata in keystone catalog
murano_metadata_url = http://localhost:8084/v1

# Maximum number of environments that can be processed simultaneously
max_environments = 20

# Maximum number of VMs per environment
max_hosts = 250

# Template IP address for generating environment subnet cidrs
env_ip_template = 10.0.0.0

# Enforces default network topology.
# Allowed values: nova, flat, routed
# default is routed
network_topology = routed

[keystone]
# URL of OpenStack KeyStone service REST API.
# Typically only hostname (or IP) needs to be changed
auth_url = http://localhost:5000/v2.0

# Keystone SSL parameters
# Optional CA cert file to use in SSL connections
#ca_file =
# Optional PEM-formatted certificate chain file
#cert_file =
# Optional PEM-formatted file that contains the private key
#key_file =
# If set then the server's certificate will not be verified
insecure = False

[heat]
```

```
# Heat SSL parameters
# Optional CA cert file to use in SSL connections
#ca_file =
# Optional PEM-formatted certificate chain file
#cert_file =
# Optional PEM-formatted file that contains the private key
#key_file =
# If set then the server's certificate will not be verified
insecure = False
# Valid endpoint types: publicURL (default), internalURL, adminURL
endpoint_type = publicURL

[neutron]
# Optional CA cert file to use in SSL connections
#ca_cert =
# Allow self signed server certificate
insecure = False
# Valid endpoint types: publicURL (default), internalURL, adminURL
endpoint_type = publicURL

[rabbitmq]
# Connection parameters to RabbitMQ service

# Hostname or IP address where RabbitMQ is located.
# !!! Change localhost to your real IP or hostname as this address must be reacha
host = localhost

# RabbitMQ port (5672 is a default)
port = 5672

# Use SSL for RabbitMQ connections (True or False)
ssl = False

# Path to SSL CA certificate or empty to allow self signed server certificate
#ca_certs =

# RabbitMQ credentials. Fresh RabbitMQ installation has "guest" account with "gue
# It is recommended to create dedicated user account for Murano using RabbitMQ we
login = guest
password = guest

# RabbitMQ virtual host (vhost). Fresh RabbitMQ installation has "/" vhost precon
# It is recommended to create dedicated vhost for Murano using RabbitMQ web conso
virtual_host = /
```

Run

- Run Murano Conductor service:

  Ubuntu

```
service murano-conductor start
```

CentOS

```
initctl start murano-conductor
```

Murano Repository Service

Murano Repository provides access to metadata for Murano Conductor and Murano Dashboard. It also allows to manage metadata objects via API. Editing service definitions (and other data stored in Murano Repository) is made separately for each tenant.

This chapter describes the procedure of installation and configuration of Murano Repository.

Install

- Superuser privileges is required to install and configure system packages. Let's switch to root account:

```
sudo su -
```

- Make sure that additional linux repositories are installed. See the appendix for information about preparing a virtual machine for murano installation.
- Clone Murano Repository from git:

```
git clone https://github.com/stackforge/murano-repository
```

Stable version one of our releases can be checked by the tag:

```
cd murano-repository && git checkout 0.4
```

- And perform installation:

Ubuntu

```
sh setup.sh install
```

CentOS

```
sh setup-centos.sh install
```

- Successful installation ends with message like this:

```
Successfully installed muranorepository
Cleaning up...
LOG:> Making sample configuration files at "/etc/murano"
LOG:> Reloading initctl
LOG:> Please, make proper configuration,located at "/etc/murano", before starting
```

Configure

- Copy and edit configuration files:

```
cd /etc/murano
cp murano-repository.conf.sample murano-repository.conf
```

- Configure `murano-repository.conf` according to your environment:
  - *[DEFAULT]* section sets up main server parameters: port and address. Folder to store cache and logging parameters also defines in this section.
    - *manifests* parameter points out to directory with metadata objects: manifests should be on the first level. All other objects are kept in a separates folders. This folders are configured in the parameters described below:
    - *ui* sets up directory name for keeping Murano dashboard ui definitions
    - *workflows* sets up directory name for keeping Murano Conductor workflows
    - *heat* sets up directory name for keeping Heat templates
    - *agent*sets up directory name for keeping Murano Agent templates
    - *scripts* sets up directory name for keeping Murano Execution Plans
  - *[output]* defines result archive structure. This parameters applies to archive structure that clients (Murano Conductor) request from Metadata Repository.
  - In *[keystone]* configure parameters of Openstack Keystone service. For more information see Auth-Token Middleware with Username and Password [http://docs.openstack.org/developer/keystone/configuringservices.html]
-

Register murano-repository service in Openstack. To do that perform the following commands:

Note

You need to be authorized in Openstack to run this commands

```
$ keystone service-create --name murano-metadata --type murano-metadata --descrip

$ keystone endpoint-create
 --region RegionOne
 --service-id The ID field returned by the keystone service-create
 --publicurl http://x.x.x.x:8084/v1 (where x.x.x.x - host ip where murano-api ins
          8084 - port number, also can be changed,
          v1 - metadata API version)
 --internalurl the same as publicurl
 --adminurl the same as publicurl
```

Run

- Run Murano Repository service:

  Ubuntu

  ```
  service murano-repository start
  ```

  CentOS

  ```
  initctl start murano-repository
  ```

Murano Dashboard

Murano Dashboard provides Web UI for Murano Project.

Warning

This installation is not capable with Horizon installed by devstack

Install

- Superuser privileges is required to install and configure system packages. Let's switch to root account:

```
sudo su -
```

- Make sure that additional repositories are installed and your system is updated and upgraded. Please check from with steps in the appendix.
- If there is no openstack dashboard package in your environment install it now with all dependencies. Deleting an Ubuntu theme is an optional step but recommended. In Centos open up the firewall ports for HTTP.

  ## Note

  Horizon installed by devstack is not capable for a murano installation.

  Ubuntu

  ```
  apt-get install memcached libapache2-mod-wsgi openstack-dashboard
  dpkg --purge openstack-dashboard-ubuntu-theme
  ```

  CentOS

  ```
  yum install make gcc memcached python-memcached \
  mod_wsgi openstack-dashboard python-netaddr.noarch
  #> lokkit -p http:tcp
  #> lokkit -p https:tcp
  ```

- Clone Murano Dashboard repository from the github:

  ```
  git clone https://github.com/stackforge/murano-dashboard
  ```

- Stable version one of our releases can be checked out by tag:

  ```
  cd murano-dashboard && git checkout 0.4
  ```

- Switch to just created directory and run installation script

  Ubuntu

  ```
  sh setup.sh install
  ```

  CentOS

```
sh setup-centos.sh install
```

Configure

- Murano installation script makes all needed changes in horizon (openstack dashboard) configs. All you have to do is to configure horizon in appropriate way. Set OPENSTACK_HOST in your horizon local settings which located in `/etc/openstack-dashboard/local_settings.py`.. For more information visit official horizon documentation. [http://docs.openstack.org/developer/horizon/]

Run

Since all required settings are made Apache service need to be restarted to apply all changes.

- *Ubuntu*

```
# service apache2 restart
```

- *CentOS*

```
# service httpd restart
```

- Check that "Environments" panel appears at the horizon "Project" tab. To see how to operate with Murano dashboard plugin check out Murano User Guide. [http://murano-docs.github.io/latest/user-guide/content/ch01.html]

SSL configuration

Murano components are able to work with SSL. This chapter will help your to make proper settings with SSL configuration.

HTTPS for Murano API

SSL for Murano API service can be configured in *ssl* section in `/etc/murano/murano-api.conf`. Just point to a valid SSL certificate. See the example below:

```
[ssl]
cert_file = PATH
key_file = PATH
ca_file = PATH
```

- *cert_file=PATH:* Path to the certificate file the server should use when binding to an SSL-wrapped socket.

- *key_file=PATH:* Path to the private key file the server should use when binding to an SSL-wrapped socket.
- *ca_file=PATH:* Path to the CA certificate file the server should use to validate client certificates provided during an SSL handshake. This is ignored if cert_file and "key_file" are not set.

The use of SSL is automatically started after point to HTTPS protocol instead of HTTP during registration Murano API service in endpoints (Change publicurl argument to start with https://). See here how to register Murano API in Openstack Keystone.

SSL for Murano API is implemented like in any other Openstack component. This realization is based on ssl python module so more information about it can be found here. [http://docs.python.org/2/library/ssl.html]

SSL for RabbitMQ

All Murano components communicate with each other by RabbitMQ. This interaction can be encrypted with SSL. By default all messages in Rabbit MQ are not encrypted. Each RabbitMQ Exchange should be configured separately.

Murano API -> Rabbit MQ exchange

Edit *rabbitmq* section in `/etc/murano/murano-api.conf` and set ssl option to True to enable SSL. Specify the path to the SSL CA certificate in regular format: /path/to/file without quotes or leave it empty to allow self-signed certificates.

```
 [rabbitmq]

# Use SSL for RabbitMQ connections (True or False)
ssl = True

# Path to SSL CA certificate or empty to allow self signed server certificate
ca_certs =
```

Rabbit MQ -> Murano Conductor exchange

Open */etc/murano/conductor.conf* and configure *rabbitmq* section in the same way: enable ssl option to True and set CA certificate path or leave it empty to allow self-signed certificates.

```
 [rabbitmq]

# Use SSL for RabbitMQ connections (True or False)
ssl = True

# Path to SSL CA certificate or empty to allow self signed server certificate
ca_certs = /home/user/certificates/example.crt
```

Murano Agent -> Rabbit MQ exchange

By default all Murano Conductor configuration settings apply to Murano Agent. If you want to configure Murano Agent in a different way change the default template. It can be found here:*/etc/murano/data/templates/agent-config/Default.template*. Take a look at appSettings section:

```
<appSettings>
 <add key="rabbitmq.host" value="%RABBITMQ_HOST%"/>
 <add key="rabbitmq.port" value="%RABBITMQ_PORT%"/>
 <add key="rabbitmq.user" value="%RABBITMQ_USER%"/>
 <add key="rabbitmq.password"
   value="%RABBITMQ_PASSWORD%"/>
 <add key="rabbitmq.vhost" value="%RABBITMQ_VHOST%"/>
 <add key="rabbitmq.inputQueue"
   value="%RABBITMQ_INPUT_QUEUE%"/>
 <add key="rabbitmq.resultExchange" value=""/>
 <add key="rabbitmq.resultRoutingKey"
   value="%RESULT_QUEUE%"/>
 <add key="rabbitmq.durableMessages" value="true"/>

 <add key="rabbitmq.ssl" value="%RABBITMQ_SSL%"/>
 <add key="rabbitmq.allowInvalidCA" value="true"/>
 <add key="rabbitmq.sslServerName" value=""/>
</appSettings>
```

Desired parameter should be set directly to the value of the key that you want to change. Quotes are need to be kept. Thus you can change "rabbitmq.ssl" and "rabbitmq.port" values to make Rabbit MQ work with this exchange in a different from Murano-Conductor way.

SSL for Murano Dashboard

If you are going not to use self-signed certificates additional configuration do not need to be done. Just point https in the URL. Otherwise, set *MURANO_API_INSECURE = True* on horizon config. You can find it in `/etc/openstack-dashboard/local_settings.py`..

Chapter 3. Building Windows Image

Murano requires a Windows Image in QCOW2 format to be built and uploaded into Glance.

The easiest way to build Windows image for Murano is to build it on the host where your OpenStack is installed.

Install Required Packages

Note

Please check that hardware virtualization supported and enabled in BIOS.

The following packages should be installed on any host which will be used to build Windows Image:

- ipxe-qemu
- kvm-ipxe
- qemu-kvm
- python-libvirt
- libvirt-bin
- libvirt0
- virt-goodies
- virt-manager
- virt-top
- virt-what
- virtinst
- python

On Ubuntu you could install them using the command below:

```
>#  apt-get install ipxe-qemu kvm-ipxe qemu-kvm virt-goodies \
  virtinst virt-manager libvirt0 libvirt-bin \
  python python-libvirt \
  python-libxml2 python-minimal python-pycurl \
  python-pyorbit python-requests python-six \
  samba samba-common openssh-server virt-top virt-what
```

Configure Shared Resource

Configure samba based share.

```
>#  mkdir -p /opt/samba/share
>#  chown -R nobody:nogroup /opt/samba/share
```

Configure samba server (/etc/samba/smb.conf).

```
...
[global]
   ...
   security = user
...
[share]
   comment = Deployment Share
   path = /opt/samba/share
   browsable = yes
   read only = no
   create mask = 0755
   guest ok = yes
   guest account = nobody
...
```

Restart services.

```
>#  service smbd restart
>#  service nmbd restart
```

Prerequisites

Download the files below and copy them into their places in your **${SHARE_PATH}** folder (we usually use **/opt/samba/share** as **${SHARE_PATH}**):

- Windows 2012 Server ISO evaluation version
  - ${SHARE_PATH}/libvirt/images/ws-2012-eval.iso
  - http://technet.microsoft.com/en-us/evalcenter/hh670538.aspx [http://technet.microsoft.com/en-us/evalcenter/hh670538.aspx]
- VirtIO drivers for Windows
  - ${SHARE_PATH}/libvirt/images/virtio-win-0.1-52.iso
  - http://alt.fedoraproject.org/pub/alt/virtio-win/stable/virtio-win-0.1-52.iso [http://alt.fedoraproject.org/pub/alt/virtio-win/stable/virtio-win-0.1-52.iso]
- CloudBase-Init for Windows
  - ${SHARE_PATH}/share/files/CloudbaseInitSetup_Beta.msi
  - http://www.cloudbase.it/downloads/CloudbaseInitSetup_Beta.msi [http://www.cloudbase.it/downloads/CloudbaseInitSetup_Beta.msi]
- Far Manager
  - ${SHARE_PATH}/share/files/Far30b3367.x64.20130426.msi
  - http://www.farmanager.com/files/Far30b3525.x64.20130717.msi [http://www.farmanager.com/files/Far30b3525.x64.20130717.msi]

- Git client
  - ${SHARE_PATH}/share/files/Git-1.8.1.2-preview20130201.exe
  - https://msysgit.googlecode.com/files/Git-1.8.3-preview20130601.exe [https://msysgit.googlecode.com/files/Git-1.8.3-preview20130601.exe]
- Sysinternals Suite
  - ${SHARE_PATH}/share/files/SysinternalsSuite.zip
  - http://download.sysinternals.com/files/SysinternalsSuite.zip [http://download.sysinternals.com/files/SysinternalsSuite.zip]
- unzip.exe tool
  - ${SHARE_PATH}/share/files/unzip.exe
  - https://www.dropbox.com/sh/zthldcxnp6r4flm/-k1Om_V6XR [https://www.dropbox.com/sh/zthldcxnp6r4flm/-k1Om_V6XR]
- PowerShell v3
  - ${SHARE_PATH}/share/files/Windows6.1-KB2506143-x64.msu
  - http://www.microsoft.com/en-us/download/details.aspx?id=34595 [http://www.microsoft.com/en-us/download/details.aspx?id=34595]
- .NET 4.0
  - ${SHARE_PATH}/share/files/dotNetFx40_Full_x86_x64.exe
  - http://www.microsoft.com/en-us/download/details.aspx?id=17718 [http://www.microsoft.com/en-us/download/details.aspx?id=17718]
- .NET 4.5
  - ${SHARE_PATH}/share/files/dotNetFx45_Full_setup.exe
  - http://www.microsoft.com/en-us/download/details.aspx?id=30653 [http://www.microsoft.com/en-us/download/details.aspx?id=30653]
- Murano Agent
  - ${SHARE_PATH}/share/files/MuranoAgent.zip
  - https://www.dropbox.com/sh/zthldcxnp6r4flm/-k1Om_V6XR [https://www.dropbox.com/sh/zthldcxnp6r4flm/-k1Om_V6XR]

Additional Software

This section describes additional software which is required to build a Windows Image.

Windows ADK

Windows Assessment and Deployment Kit (ADK) for Windows® 8 is required to build your own answer files for auto unattended Windows installation.

You can dowload it from http://www.microsoft.com/en-us/download/details.aspx?id=30652 [http://www.microsoft.com/en-us/download/details.aspx?id=30652].

PuTTY

PuTTY is a useful tool to manage your Linux boxes via SSH.

You can download it from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html [http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html].

Windows Server 2012 ISO image

We use the following Windows installation images:

- Windows Server 2008 R2
  - Image Name: 7601.17514.101119-1850_x64fre_server_eval_en-us-GRMSXEVAL_EN_DVD.iso
  - URL: http://www.microsoft.com/en-us/download/details.aspx?id=11093 [http://www.microsoft.com/en-us/download/details.aspx?id=11093]
- Windows Server 2012
  - Image Name: 9200.16384.WIN8_RTM.120725-1247_X64FRE_SERVER_EVAL_EN-US-HRM_SSS_X64FREE_EN-US_DV5.iso
  - URL: http://technet.microsoft.com/en-US/evalcenter/hh670538.aspx?ocid=&wt.mc_id=TEC_108_1_33 [http://technet.microsoft.com/en-US/evalcenter/hh670538.aspx?ocid=&wt.mc_id=TEC_108_1_33]

VirtIO Red Hat drivers ISO image

Warning

Please, choose stable version instead of latest, We've got errors with unstable drivers during guest unattended install.

Download drivers from http://alt.fedoraproject.org/pub/alt/virtio-win/stable/ [http://alt.fedoraproject.org/pub/alt/virtio-win/stable/]

Floppy Image With Unattended File

Run following commands as root:

1. Create emtpy floppy image in your home folder

```
>#  dd bs=512 count=2880 \
  if=/dev/zero of=~/floppy.img \
  mkfs.msdos ~/floppy.img
```

2. Mount the image to **/media/floppy**

```
>#  mkdir /media/floppy mount -o loop \
  ~/floppy.img /media/floppy
```

3. Download **autounattend.xml** file from https://raw.github.com/stackforge/murano-deployment/master/image-builder/share/files/ws-2012-std/autounattend.xml [https://raw.github.com/stackforge/murano-deployment/master/image-builder/share/files/ws-2012-std/autounattend.xml]

```
>#  cd ~
>#  wget https://raw.github.com/stackforge/murano-deployment\
  /master/image-builder/share/files/ws-2012-std/autounattend.xml
```

4. Copy our **autounattend.xml** to **/media/floppy**

```
>#  cp ~/autounattend.xml /media/floppy
```

5. Unmount the image

```
>#  umount /media/floppy
```

Build Windows Image (Automatic Way)

1. Clone **murano-deployment** repository

```
>#  git clone git://github.com/stackforge/murano-deployment.git
```

2. Change directory to **murano-deployment/image-builder** folder.

3. Create folder structure for image builder

```
>#  make build-root
```

4. Create shared resource

   Add to /etc/samba/smb.conf.

```
[image-builder-share]
   comment = Image Builder Share
   browsable = yes
   path = /opt/image-builder/share
   guest ok = yes
   guest user = nobody
   read only = no
   create mask = 0755
```

   Restart samba services.

```
>#  restart smbd && restart nmbd
```

5. Test that all required files are in place

```
>#  make test-build-files
```

6. Get list of available images

```
># make
```

7. Run image build process

```
># make ws-2012-std
```

8. Wait until process finishes

9. The image file **ws-2012-std.qcow2** should be stored under **/opt/image-builder/share/images** folder.

Build Windows Image (Manual Way)

Warning

Please note that the preferred way to build images is to use **Automated Build** described in the previous chapter.

Get Post-Install Scripts

There are a few scripts which perform all the required post-installation tasks.

Package installation tasks are performed by script named **wpi.ps1**.

Download it from https://raw.github.com/stackforge/murano-deployment/master/image-builder/share/scripts/ws-2012-std/wpi.ps1 [https://raw.github.com/stackforge/murano-deployment/master/image-builder/share/scripts/ws-2012-std/wpi.ps1]

Note

There are a few scripts named **wpi.ps1**, each supports only one version of Windows image. The script above is intended to be used to create Windows Server 2012 Standard. To build other version of Windows please use appropriate script from **scripts** folder.

Clean-up actions to finish image preparation are performed by **Start-Sysprep.ps1** script.

Download it from https://raw.github.com/stackforge/murano-deployment/master/image-builder/share/scripts/ws-2012-std/Start-Sysprep.ps1 [https://raw.github.com/stackforge/murano-deployment/master/image-builder/share/scripts/ws-2012-std/Start-Sysprep.ps1]

These scripts should be copied to the shared resource folder, subfolder **Scripts**.

Create a VM

This section describes steps required to build an image of Windows Virtual Machine which could be used with Murano. There are two possible ways to create it - from CLI or using GUI tools. We describe both in this section.

Note

Run all commands as root.

Way 1: Using CLI Tools

This section describes the required step to launch a VM using CLI tools only.

1. Preallocate disk image

```
>#  qemu-img create -f raw /var/lib/libvirt/images/ws-2012.img 40G
```

2. Start the VM

```
>#  virt-install --connect qemu:///system --hvm --name WinServ \
    --ram 2048 --vcpus 2 --cdrom /opt/samba/share/9200.16384.WIN8_RTM\
.120725-1247_X64FRE_SERVER_EVAL_EN-US-HRM_SSS_X64FREE_EN-US_DV5.ISO \
    --disk path=/opt/samba/share/virtio-win-0.1-52.iso,device=cdrom \
    --disk path=/opt/samba/share/floppy.img,device=floppy \
    --disk path=/var/lib/libvirt/images/ws-2012.qcow2\
,format=qcow2,bus=virtio,cache=none \
    --network network=default,model=virtio \
    --memballoon model=virtio --vnc --os-type=windows \
    --os-variant=win2k8 --noautoconsole \
    --accelerate --noapic --keymap=en-us --video=cirrus --force
```

Way 2: Using virt-manager UI

A VM also could be lauched via GUI tools like virt-manager.

1. Launch *virt-manager* from shell as root
2. Set a name for VM and select Local install media
3. Add one cdrom and attach Windows Server ISO image to it
4. Select OS type **Windows** and it's version **Windows Server 2008**
5. Set CPU and RAM amount
6. Deselect option **Enable storage for this virtual machine**
7. Select option **Customize configuration before install**
8. Add second cdrom for ISO image with virtio drivers
9. Add a floppy drive and attach our floppy image to it
10. Add (or create new) HDD image with Disk bus **VirtIO** and storage format **RAW**
11. Set network device model **VirtIO**
12. Start installation process and open guest vm screen through **Console** button

Convert the image from RAW to QCOW2 format. The image must be converted from RAW format to QCOW2 before being imorted into Glance.

```
>#  qemu-img convert -O qcow2 /var/lib/libvirt/images/ws-2012.raw \
```

```
/var/lib/libvirt/images/ws-2012-ref.qcow2
```

Upload Image Into Glance

Services deployed by Murano require specially prepared images, that can be created manually or via automated scripts. Please refer to corresponding chapters of this book to create image. After images are created they should be registered in Openstack Glance - image operation service.

1. Use the glance image-create command to import your disk image to Glance:

   ```
   >$ glance image-create --name <NAME> \
     --is-public true --disk-format qcow2 \
     --container-format bare \
     --file <IMAGE_FILE> \
     --property <IMAGE_METADATA>
   ```

   Replace the command line arguments to *glance image-create* with the appropriate values for your environment and disk image:

   - Replace <**NAME**> with the name that users will refer to the disk image by. E.g. '**ws-2012-std**'
   - Replace <**IMAGE_FILE**> with the local path to the image file to upload. E.g. '**ws-2012-std.qcow2**'.
   - Replace <**IMAGE_METADATA**> with the following property string (It **MUST** be one-line string!)

     ```
     murano_image_info='{"title": "Windows 2012 Standart Edition",
      "type": "windows.2012"}'
     ```

     where

     - **title** - user-friendly description of the image
     - **type** - an image type, for example 'windows.2012'

2. To update metadata of the existing image run the command:

   ```
   >$ glance image-update <IMAGE-ID> --property <IMAGE_MATADATA>
   ```

   - Replace <**IMAGE-ID**> with image id from the previous command output.
   - Replace <**IMAGE_METADATA**> with murano_image_info property, e.g.

     ```
     murano_image_info='{"title": "Windows 2012 Std", "type": "windows.2012"}'
     ```

   ### Warning

   The value of the **--property** argument named **murano_image_info** is a JSON string. Only double quotes are valid in JSON, so please type the string exactly as in the example above.

After these steps desired image can be chosen in Murano dashboard and used for services platform.

Chapter 4. Building Linux Image

Install Required Packages

Note

Please check that hardware virtualization is supported and enabled in BIOS.

The following packages should be installed on any host which will be used to build Linux Image:

- ipxe-qemu
- kvm-ipxe
- qemu-kvm
- munin-libvirt-plugins
- python-libvirt
- libvirt-bin
- libvirt0
- munin-libvirt-plugins
- python-libvirt
- virt-goodies
- virt-manager
- virt-top
- virt-what
- virtinst
- python

On Ubuntu you could install them using the command below:

```
># apt-get install ipxe-qemu kvm-ipxe qemu-kvm virt-goodies \
    virtinst virt-manager libvirt0 libvirt-bin \
    munin-libvirt-plugins python python-libvirt \
    python-libxml2 python-minimal python-pycurl \
    python-pyorbit python-requests python-six \
    samba samba-common openssh-server virt-top virt-what
```

Build Linux Image

Create a VM

This section describes steps required to build an image of Linux Virtual Machine which could be used with Murano. There are two possible ways to create it - from CLI or using GUI tools. We describe both in this section.

Note

Run all commands as root.

Way 1: Using CLI Tools

This section describes the required step to launch a VM using CLI tools only.

1. Preallocate disk image

```
>#  qemu-img create -f qcow2 /var/lib/libvirt/images/cloud-linux.img 10G
```

2. Start the VM

```
>#  virt-install --connect qemu:///system --hvm --name cloud-linux \
  --ram 2048 --vcpus 2 --cdrom /PATH_TO_YOUR_LINUX.ISO \
  --disk path=/var/lib/libvirt/images/cloud-linux.img, \
  format=qcow2,bus=virtio,cache=none \
  --network network=default,model=virtio \
  --memballoon model=virtio --vnc --os-type=linux \
  --accelerate --noapic --keymap=en-us --video=cirrus --force
```

Way 2: Using virt-manager UI

A VM also could be lauched via GUI tools like virt-manager.

1. Launch *virt-manager* from shell as root
2. Set a name for VM and select Local install media
3. Add one cdrom and attach your linux ISO image to it
4. Select OS type **Linux** and it's version **choose yours**
5. Set CPU and RAM amount
6. Deselect option **Enable storage for this virtual machine**
7. Select option **Customize configuration before install**
8. Add (or create new) HDD image with Disk bus **VirtIO** and storage format **QCOW2**
9. Set network device model **VirtIO**
10. Start installation process and open guest vm screen through **Console** button

Guest VM Linux OS preparation

Ubuntu 12.04 LTS x86_64

```
>#  for action in update upgrade dist-upgrade;do apt-get -y $action;done
>#  apt-get install -y git unzip make cmake gcc python-dev python-pip openssh-serve
```

CentOS 6.4 x86_64

```
>#  rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.
>#  for action in update upgrade;do yum -y $action; done
>#  yum install -y git unzip make cmake gcc python-devel python-pip openssh-server
```

murano-agent installation steps

```
>#  mkdir -p /opt/git
>#  cd /opt/git
>#  git clone https://github.com/stackforge/murano-agent.git
>#  cd murano-agent/python-agent
>#  git checkout release-0.3
>#  chmod a+x setup*.sh

# To install Murano Agent on Ubuntu run:
>#  ./setup.sh install
or
>#  ./setup-centos.sh install
```

cloud-init installation steps

- **Ubuntu**

  ```
  >#  apt-get install -y cloud-init cloud-initramfs-growroot
  ```

- **CentOS**

  ```
  >#  yum install -y cloud-init
  ```

  ## Note

  Ubuntu only

  ```
  >#  dpkg-reconfigure cloud-init
  ```

  Mark **EC2** data source support, save and exit or add manualy **Ec2** to the datasource_list variable in the /etc/cloud/cloud.cfg.d/90_dfkg.cfg

- **Minimal cloud-init configuration options**

```
>#  vi /etc/cloud/cloud.cfg:
 user: ec2-user
 disable_root: 1
 preserve_hostname: False
```

Security setup

Create user and make it able to run commands through sudo without password prompt.

- **Ubuntu**

```
>#  useradd -m -G sudo -s /bin/bash ec2-user
>#  passwd ec2-user
```

- **CentOS**

```
>#  useradd -m -G wheel -s /bin/bash ec2-user
>#  passwd ec2-user
```

- **Sudo**

```
>#  echo "ec2-user   ALL=(ALL)  NOPASSWD: ALL" > /etc/sudoers.d/ec2-user
>#  chmod 440 /etc/sudoers.d/ec2-user
```

- **Disable SSH password-based logins in the /etc/ssh/sshd_config.**

```
...
GSSAPIAuthentication no
PasswordAuthentication no
PermitRootLogin no
...
```

Network handling

- **Ubuntu**

```
>#  rm -rf /etc/udev/rules.d/70-persistent-net.rules
```

- **CentOS** Remove or comment out HWADDR and UUID in /etc/sysconfig/network-scripts/ifcfg-eth*

```
>#  rm -rf /etc/udev/rules.d/70-persistent-net.rules
```

Shutdown VM

Convert the image from RAW to QCOW2 format if you made it as RAW. The image must be converted from RAW format to QCOW2 before being imorted into Glance.

```
>#  qemu-img convert -O qcow2 /var/lib/libvirt/images/cloud-linux.img \
/var/lib/libvirt/images/cloud-linux.img.qcow2
```

Upload Image Into Glance

Services deployed by Murano require specially prepared images. After images are created they should be registered in Openstack Glance - image operation service.

```
>#  glance image-create --disk-format=qcow2 --container-format=bare --is-public=tru
```

Note

Image should be marked with an appropriate type. That could be done through the Horizon UI. Navigate to Project -> Environments -> Marked Images -> Mark Image and fill up form:

- **Image** - cloud-linux
- **Title** - My Cloud-ready Linux
- **Type** - Generic Linux

After these steps desired image can be chosen in Murano dashboard and used for services platform.

Chapter 5. Troubleshooting

General Notes. The following debug sequence should be used when you have no idea about why the system isn't working. If you have one, you may skip unnecessary sections.

Set debug options to "True" in the following Murano configuration files:

- /etc/murano-api/murano-api.conf
- /etc/murano-conductor/conductor.conf

Stop both **murano-api** and **murano-conductor** services. We will start them one by one from the console.

murano-api. First, the murano-api must be started.

- Open new console
- Start **murano-api** service manually

```
>\# murano-api --config-dir /etc/murano-api 2>&1 >\
 /var/log/murano-api-live.log &
>\# tailf /var/log/murano-api-live.log
```

- Open dashboard, create and send to deploy some simple environment.
- Open RabbitMQ web console, open your vhost and ensure that queues were created and there is at least one message.
- Check log for errors - there shouldn't be any
- Keep **murano-api** service running

murano-conductor. Next to the **murano-api** the **murano-conductor** should be started

- Open new console
- Start conductor from console

```
>\# muranoconductor --config-dir /etc/murano-conductor >\
  /var/log/murano-conductor-live.log &
>\# tailf /var/log/murano-conductor-live.log
```

- Check that there is no python exceptions in the log. Some errors like 404 are ok, as conductor tries to delete environment that doesn't exist
- Check heat stack status. It should not be in FAILED state. If it is - check heat and nova error log to find the cause.
- Keep murano-conductor service running.

Log Files. There are various log files which will help you to debug the system.

Murano Log Files

- /var/log/murano-api.log

- /var/log/murano-conductor.log
- /var/log/apache2/errors.log
- /var/log/httpd/errors.log

Windows Log Files

- C:\Program Files (x86)\CloudBase Solutions\logs\log.txt
- C:\Murano\Agent\log.txt
- C:\Murano\PowerShell.log

Chapter 6. Appendix

Murano VM

Note

Your VM MUST be attached to the network with Internet access and openstack management network (lab network) access.

Ubuntu Server 12.04 LTS x86_64

Installation steps:

- Install minimal version of the system
- When prompted, mark OpenSSH Server to be installed
- Login as root
- Enable Cloud Archive repository

  Create and add the following lines to the `/etc/apt/sources.list.d/grizzly.list` file

```
deb http://ubuntu-cloud.archive.canonical.com/ubuntu \
                    precise-updates/grizzly main
deb http://archive.gplhost.com/debian grizzly main
deb http://archive.gplhost.com/debian grizzly-backports main
```

- Update installed OS and packages

```
>#  apt-get update
>#  apt-get install ubuntu-cloud-keyring
>#  apt-get install gplhost-archive-keyring
>#  apt-get install mc unzip git make gcc python-setuptools python-pip
>#  apt-get upgrade
```

CentOS 6.4 x86_64

Installation steps:

- Install minimal version of the system.
- When prompted, mark OpenSSH Server to be installed
- Login as root
- Enable RedHat Openstack and Epel repository
- Update system and add repositories and update OS

```
>#  yum install -y http://rdo.fedorapeople.org/openstack/openstack-grizzly/rdo-re
```

```
># yum install -y http://mirror.us.leaseweb.net/epel/6/x86_64/epel-release-6-8.no
># yum install -y mc unzip git make gcc python-setuptools python-pip upstart
># yum update
># yum upgrade
```

Most of dependent packages will be installed automatically with setup scripts.

Note

In order to have proper versions of python dependency packages installed, pip version MUST be 1.4 or higher!

View version of pip on your system:

```
># pip --version
```

Upgrade pip to latest version if it is older than 1.4:

```
># pip install --upgrade pip
```

Note

During upgrade the location of pip executable could be changed. If this is happened, you have to create a simlink pointing to the new location:

```
># rm /usr/bin/pip
># ln -s /usr/local/bin/pip /usr/bin/pip
```