Murano Getting Started Guide

Murano Getting Started Guide

v0.2

Publication date 2013-09-09

Abstract

This document is intended for individuals who wish to use our product or intend to contribute.

Table of Contents

1. About Murano 1
    Document change history 1
2. Before You Begin 2
    System Requirements 2
        OpenStack Lab 2
        Devbox 3
3. Setup Lab Environment 4
    Install OpenStack 4
    Prepare Shared Prerequisites 5
    Build Windows Image 6
4. Setup Devbox 9
    Vagrant Way 9
    Automated Way 10
    Murano Services Configuration 11
        RabbitMQ Configuration Notes 11
        Keystone Configuration Notes 11
        Murano-Api Configuration 12
        Murano-Conductor 13
        Murano-Dashboard 13
5. Troubleshooting 15
    Log files 16

Chapter 1. About Murano

Murano is a new service which allows a non-experienced user to deploy reliable Windows based environments in a “push-the-button” manner. The key goal is to provide a UI and API enabling the deployment and operation of Windows Environments at the Windows Services abstraction level. The service is able to orchestrate complex circular dependent cases in order to set up a complex Windows Environment with multiple dependant services.

The service addresses following use cases:

- Self-provisioning of predefined Windows services with their dependencies
- Automation of administrative tasks during data center roll-out
- Custom windows application as a windows service

The solution provides higher level of abstraction for manipulation Windows Environments. Key concepts are:

- Windows Service - a service such as Active Directory, MSSQL, or IIS, which usually consists of multiple virtual machines and has multiple dependencies.
- Windows Environment - a logical unit for all Services and represents a classical Windows Datacenter.
- Windows VM instance - a VM which hosts a Windows Service. A Windows Service might be deployed over several Windows VM instances.

The Key Features of the Service are the following:

1. Native to OpenStack
2. Introduces abstraction level for Windows Environments
3. Supports Availability Zones and Disaster Recovery scenarios
4. Uses native Windows features for HA solutions

Document change history

The following table describes the most recent changes:

Revision Date	Summary of Changes
September. 4, 2013	• Initial document creation.

Chapter 2. Before You Begin

Naming Conventions

To clearly separate commands and parts of configuration files we use boxes, like shown below:

```
...
# Part of config file here
```

All commands start either with '>$' mark or with '>#' mark. The difference is that first should be executed as regular user, while second - as superuser ('root').

```
>$ echo 'Execute this command as regular user'
>#  echo 'Execute this command as root'
```

Use Appropriate Branch

There are a few branches that can be found in all Murano repositories: master, release-0.1, release-0.2 and so on. Branch master is the most recent between the releases, but not as stable as release-x.x branches.

Use Separate vHost in RabbitMQ

In general it is OK to configure Murano services to use the root ('/') vHost in RabbitMQ and use the same user credentials as OpenStack services use. However, we recommend to create a separate vHost with separate user for each Murano devbox. There are a few reasons for that:

- this prevents queue name collisions
- this prevents message stealing from queues
- this simplify debugging

If you are planning to use only one devbox then you may stay with '/' vHost. Steps required to configure your own vHost are described in RabbitMQ Configuration Notes.

System Requirements

- *OpenStack Lab* - a single node Openstack environment.
- *Devbox* - a machine where all murano components and services are running. This machine should have access to the Openstack Lab.

OpenStack Lab

Supported OS

- Ubuntu Server 12.04 x64

Hardware Requirements

- CPU: 8+ cores

- RAM: 12+ GB
- HDD (select any option)
- NIC: 1x Ethernet 1+ Gbps

Software Requirements

- Heat installed
- RabbitMQ configured (+ additional vhosts and user accounts)
- Windows Server 2012 Standard image imported into Glance (described later)
- Samba share with prerequisites (described later)
- OpenStack metadata service.

Devbox

We suggest to use virtual machine for Murano, as it allows you to backup your VM easily. Any type of hypervisor software which supports linux as guest OS could be used. KVM, VMWare and VirtualBox were tested with success.

Supported OS

- Ubuntu Server 12.04 LTS x86_64
- CentOS 6.4 x86_64

Hardware Requirements

- RAM: 512 MB
- CPU: 1 core
- HDD: 20 GB
- 1 NIC

Software Requirements

- Packages:
- X Server is NOT required and system runlevel 3 is preferred.

Chapter 3. Setup Lab Environment

Install OpenStack

To install Murano you need a working copy of OpenStack. If you already have a Openstack installation make sure it meets the requirements described in this chapter. To install Openstack that will be ready to use with Murano follow the instructions below.

Using Devstack

Currently the most simple way to build a lab is to use devstack. The steps are quite simple:

- Install and configure OS on your hardware. The recommended OS is Ubuntu Server 12.04 x64. Minimal configuration would be enough.
- Install all the latest updates.

```
>apt-get update
>apt-get -y upgrade
```

- Create a user *stack*

```
>adduser stack
```

- Add user *stack* to sudoers

```
>echo ‘stack ALL=(ALL) NOPASSWD:ALL' > /etc/sudoers.d/stack
>chmod 0440 /etc/sudoers.d/stack
```

- Create a folder for OpenStack installation files

```
>mkdir /opt/stack
>chown stack:stack /opt/stack
```

- Clone the *devstack* repo

```
>su stack
>cd
>git clone https://github.com/openstack-dev/devstack.git
```

- Checkout the stable/grizzly branch

```
>cd devstack
>git checkout stable/grizzly
```

- Get `localrc` and `local.sh` files

```
>$ wget https://raw.github.com/stackforge/murano-deployment/release-0.2/
                getting-started/localrc
>$ wget https://raw.github.com/stackforge/murano-deployment/release-0.2/
                getting-started/local.sh -O local.sh-O local.sh
>$ chmod +x local.sh
```

- Start *devstack*

```
>$ ./stack.sh
```

When stack.sh finishes execution your OpenStack installation is ready.

Prepare Shared Prerequisites

Configure Samba Share

- Install SAMBA

```
>$ apt-get update
>$ apt-get install samba
```

- Create shared folder

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

- Edit /etc/samba/smb.conf

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

- Restart services

```
>#  restart smbd
>#  restart nmdb
```

Copy Prerequisites Into The Share

- Create folder structure

```
>#  mkdir -p “/opt/samba/share/Prerequisites/IIS”
>#  mkdir -p “/opt/samba/share/Prerequisites/SQL Server/2012”
>#  mkdir -p “/opt/samba/share/Prerequisites/SQL Server/Tools”
```

- Download and put the following files into the specified paths under `/opt/samba/share` folder:

  1. ASP .NET MVC 4 setup package [http://www.asp.net/mvc/mvc4]: `Prerequisites/AspNetMVC4Setup.exe`
  2. Additional files to build ASP application [https://www.dropbox.com/sh/zthldcxnp6r4flm/Y8gxbIZGF_/WebApplications.exe]: should be placed to `Prerequisites/WebApplications.exe`
  3. MS SQL Server 2012 [http://www.microsoft.com/en-us/download/details.aspx?id=29066]: Extract the content of the ISO/EXE file to the `Prerequisites/SQL Server/2012/`. After that `setup.exe` should be located inside this directory.
  4. Microsoft® Windows PowerShell Extensions for Microsoft® SQL Server®2012 [http://go.microsoft.com/fwlink/?LinkID=239656&clcid=0x409]: `Prerequisites/SQL Server/Tools/PowerShellTools.msi`
  5. Microsoft® SQL Server® 2012 Shared Management Objects [http://go.microsoft.com/fwlink/?LinkID=239659&clcid=0x409]: `Prerequisites/SQL Server/Tools/SharedManagementObjects.msi`
  6. Microsoft® System CLR Types for Microsoft® SQL Server® 2012 [http://www.asp.net/mvc/mvc4]: `Prerequisites/SQL Server/Tools/SQLSysClrTypes.msi`

Build Windows Image

A pre-built Windows Image is required to create environments in Murano. Because of its size it’s better to build the image on the same host where OpenStack is installed. This section describe steps required to build such an image.

Prepare Build Environment

- Samba should be already installed (See Configure samba share)
- Clone murano-deployment repository

```
>#  cd /opt/git
```

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

- Change directory to `murano-deployment/image-builder` folder

```
>#  cd /opt/git/murano-deployment/image-builder
```

- Create folder structure for image builder

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

- Download and put the following files into the specified paths under `/opt/image-builder` folder:

  1. Windows 2012 Server ISO evaluation version [http://technet.microsoft.com/en-us/evalcenter/hh670538.aspx]: libvirt/images/ws2012eval.iso
  2. VirtIO drivers [http://alt.fedoraproject.org/pub/alt/virtio-win/stable/virtio-win-0.1-52.iso] should be placed to libvirt/images/virtio-win-0.1-52.iso
  3. CloudBase-Init for Windows: [http://www.cloudbase.it/downloads/CloudbaseInitSetup_Beta.msi] share/files/CloudbaseInitSetup_Beta.msi
  4. Far Manager: [http://www.farmanager.com/files/Far30b3525.x64.20130717.msi] share/files/Far30b3367.x64.20130426.msi
  5. Git client [https://msysgit.googlecode.com/files/Git-1.8.3-preview20130601.exe] share/files/Git-1.8.1.2-preview20130201.exe
  6. Sysinternals [http://download.sysinternals.com/files/SysinternalsSuite.zip] Suiteshare/files/SysinternalsSuite.zip
  7. unzip.exe tool [https://www.dropbox.com/sh/zthldcxnp6r4flm/-k1Om_V6XR] share/files/unzip.exe
  8. PowerShell v3 [http://www.microsoft.com/en-us/download/details.aspx?id=34595] share/files/Windows6.1-KB2506143-x64.msu
  9. .NET 4.0: [http://www.microsoft.com/en-us/download/details.aspx?id=17718] share/files/dotNetFx40_Full_x86_x64.exe
  10. .NET 4.5: [http://www.microsoft.com/en-us/download/details.aspx?id=30653] share/files/dotNetFx45_Full_setup.exe
  11. Murano Agent [https://www.dropbox.com/sh/zthldcxnp6r4flm/-k1Om_V6XR] share/files/MuranoAgent.zip

- Test that all required files are in place

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

Build The Image

- Get list of supported images

```
># make
```

- Run image build process

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

- Wait until process finishes
- The image file ws-2012-std.qcow2 will be stored in `/opt/image-builder/share/images` folder.

Import Windows Image Into Glance

Now when you've built a Windows Image it must be imported into Glance.

- Import openrc file which contains environment variables definitions required by OpenStack components

```
>$ source openrc
```

- Import the Windows Server 2012 image into Glance

```
>$ cd /opt/image-builder/share/images
>$ glance image-create --name ws-2012-std --disk-format qcow2           \
    --container-format bare --file ws-2012-std.qcow2 --is-public true     \
    --property murano_image_info='{"type":"ws-2012-std", "title":"Windows Server 2012 Stan
```

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.

Chapter 4. Setup Devbox

There are a few ways to install Murano devbox

- using Vagrant automation tool
- using a script to install all components automatically
- install everything manually

Note

The preferred way is to use script for automated installation. It is described in Automated Way section below.

Vagrant Way

The simplest way to get working Murano devbox is Vagrant tool. This tool is available for Ubuntu only.

Prepare Environment (Ubuntu)

- Install VirtualBox:

```
>$ apt-get install virtualbox
```

- Install VirtualBox Extension Pack.
- Install Vagrant:

```
>$ apt-get install vagrant --no-install-recommends
```

- Upgrade the Vagrant:

```
>$ wget http://files.vagrantup.com/
            packages/7ec0ee1d00a916f80b109a298bab08e391945243/
            vagrant_1.2.7_x86_64.deb


>$ dpkg --install vagrant_1.2.7_x86_64.deb
```

Launch The Box

- Clone murano-vagrant repository

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

- Change directory to cloned repository folder

```
>$ cd murano-deployment/getting-started
>$ git checkout -b release-0.2 origin/release-0.2
```

- IMPORTANT STEP: RabbitMQ credentials are specified to default in lab-binding.rc so don't forget to change them if necessary. For example: - replace all the markers '***' to your settings:

```
# Lab Settings
#-------------
# Address of the host which provides Keystone service.
#
# LAB_HOST='192.168.1.2'
LAB_HOST='***.***.***.***'
...
```

- Launch the box:

```
>$ ./launch-the-box.sh
```

- The script will do the following:
- Vagrant will do the rest:
- When everything is done open the http://127.0.0.1:8080/horizon link.

Automated Way

- Create a folder to hold git repositories

```
>$ mkdir -p /opt/git cd /opt/git && cd /opt/git
```

- Clone murano-deployment repository

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

- Change directory to murano-deployment and switch to release-0.2 branch

```
>$ cd /opt/git/murano-deployment
>$ git checkout -b release-0.2 origin/release-0.2
```

- Install prerequisites. On this step some additional system packages will be installed

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

- Configure lab binding configuration file `/etc/murano-deployment/lab-binding.rc`

```
LAB_HOST='***.***.***.***'

AUTH_URL="http://$LAB_HOST:5000/v2.0"

ADMIN_USER='admin'
ADMIN_PASSWORD='***'

RABBITMQ_LOGIN='muranouser'
RABBITMQ_PASSWORD='murano'
RABBITMQ_VHOST='muranovhost'
BRANCH_NAME='release-0.2'
```

It's recommended to use separate vHost in RabbitMQ

- Install Murano components

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

- Login to the Dashboard using URL http://uth_port = 35357;your VM IP>/dashboard

Murano Services Configuration

This chapter descrides how to configure Murano services and services communicating with Murano.

RabbitMQ Configuration Notes

OpenStack rabbitMQ credentials could be used for murano services, but prefered way is to make additional changes into rabbitMQ configuration, like own vhost, login and password. These steps require superuser rights and should be run on the OpenStack controller node where rabbitMQ service resides.

How to do this:

```
>#  rabbitmqctl add_user muranouser muranopassword
>#  rabbitmqctl set_user_tags muranouser administrator
>#  rabbitmqctl add_vhost muranovhost
>#  rabbitmqctl set_permissions -p muranovhost muranouser ".*" ".*" ".*"
```

Keystone Configuration Notes

Service entry for murano-api and endpoint associated with it could also be configured in the keystone. If there is no record about murano-api in the keystone murano-dashboard would try to reach murano-api service at localhost on the default murano-api port 8082.

How to add service record into the keystone using python keystone client:

```
>#  keystone service-create --name muranoapi --type murano \
                --description "Murano-Api Service"
```

This command returns UUID of the created service record, which should be used below.

```
>#  keystone endpoint-create \
              --service-id UUID_from_above \
              --publicurl http://murano_vm_address:8082 \
              --internalurl http://murano_vm_address:8082 \
              --adminurl http://murano_vm_address:8082 \
```

Murano-Api Configuration

Configuration files of the murano-api service reside at `/etc/murano-api` directory. Basic configuration parameters are listed below.

- murano-api.conf

```
[DEFAULT]
...
bind_host = 0.0.0.0
bind_port = 8082
log_file = /var/log/murano-api.log
...
…
[database]
...
#connection = mysql://mysqluser:mysqlpassword@mysqlhost:3306/murano

connection = sqlite:////etc/murano-api/murano.sqlite

…
...
[rabbitmq]
host = <rabbitmq ip>
port = 5672
login = <rabbitmq login>
password = <rabbitmq password>
virtual_host = <rabbitmq vhost>
...
```

- murano-api-paste.ini

```
...
[filter:authtoken]
auth_host = <keystone_ip>
```

```
auth_port = 35357
auth_protocol = http
admin_tenant_name = admin
admin_user = admin
admin_password = admin_password
...
```

Murano-Conductor

Configuration files of the murano-conductor service reside at `/etc/murano-conductor` directory. Basic configuration parameters are listed below:

- conductor.conf

```
[DEFAULT]
…
log_file = /var/log/murano-conductor.log

…
…
[keystone]
# URL of OpenStack KeyStone service REST
    API.

# Typically only hostname (or IP) needs to be
    changed

auth_url = http://keystone_ip:5000/v2.0
...
...
[rabbitmq]
host = <rabbitmq ip>
port = 5672
login = <rabbitmq login>
password = <rabbitmq password>
virtual_host = <rabbitmq vhost>
...
```

- conductor-paste.ini

It’s empty but must exist.

Murano-Dashboard

Murano-dashboard does not need to be configured, but we need to set up proper OPENSTACK_HOST variable in the OpenStack dashboard configuration file, which resides at `/etc/openstack-dashboard` directory. It should point to the OpenStack controller node.

```
…
# optional, but sometimes very useful to set DEBUG to ‘True’
```

```
DEBUG = True
...
OPENSTACK_HOST =
    “openstack_controller_address”
...
```

Chapter 5. Troubleshooting

Set debug options to "true" in all config files - dashboard, api, conductor.

Note

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.

First, stop both murano-api and murano-conductor services. We will start them one by one from the console.

Murano-Api

- Open new console
- Start api service manually

```
>#  murano-api --config-dir /etc/murano-api \
        > /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

- 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.

Now, the environment should be created, and instance(s) launched.

Next, check if instances were configured correctly by the cloudbase init tool.

Log in to any instance and open powershell log file at `C:\Murano\PowerShell.log`. There shouldn't be any exceptions logged. Other symptoms of successful configuration is that the instance was renamed and you have to press <Ctrl>+<Alt>+<Del> to log into. Unconfigured instance has autologon enabled for the first logon, so if console is open, the instance is not configured (yet).

Check that Murano Agent has correct config file. If there is a .bak file, then it was changed by the init script. Check the file, ensure that it has correct values.

Check Murano Agent log file. There should be logged all tasks received by the agent from the conductor.

Check PowerShell log. There should be messages about all functions, executed on the instance.

Log files

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:\Murano\PowerShell.log
- C:\Murano\Agent\log.txt