C12G LABS

# Private Cloud Computing with OpenNebula 1.4

**C12G Labs S.L.**

Rev20100611

Copyright

©2010 C12G Labs

Although the information in this document has been carefully reviewed, C12G Labs does not warrant it to be free of errors or omissions. C12G Labs reserves the right to make corrections, updates, revisions, or changes to the information in this document.

UNLESS OTHERWISE EXPRESSLY STATED BY C12G LABS THE SOFTWARE DESCRIBED IN THIS DOCUMENT IS PROVIDED ON "AS IS" BASIS, WITHOUT ANY WARRANTIES OF ANY KIND, INCLUDING, BUT NOT LIMITED TO, WARRANTIES CONCERNING THE INSTALLATION, USE OR PERFORMANCE OF PRODUCT. C12G AND ITS SUPPLIERS DISCLAIM ANY AND ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY FITNESS FOR A PARTICULAR PURPOSE AND/OR NON-INFRINGEMENT. C12G AND ITS SUPPLIERS DO NOT WARRANT THAT PRODUCT WILL MEET USER'S REQUIREMENTS OR THAT THE OPERATION THEREOF WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT ERRORS WILL BE CORRECTED. YOU ARE SOLELY RESPONSIBLE FOR DETERMINING THE APPROPRIATENESS OF USING THE WORK AND ASSUME ANY RISKS ASSOCIATED WITH YOUR EXERCISE OF PERMISSIONS UNDER THIS LICENSE.

Document redistribution and translation

This document is protected by copyright and you may not redistribute it or translate it into another language, in part or in whole.

Trademarks

C12G is a pending trademark in the European Union and in the United States. All other tradermakrs are property of their respective owners. Other product or company names mentioned may be trademarks or trade names of their respective companies.

# Contents

# Chapter 1

# Getting Started

## 1.1 Building a Private Cloud

### 1.1.1 What is a Private Cloud?

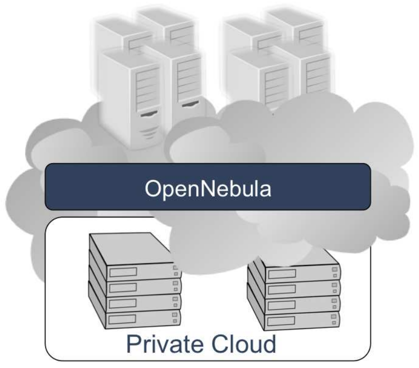


Figure 1.1:

The aim of a Private Cloud is not to expose to the world a cloud interface to sell capacity over the Internet, but to **provide local users with a flexible and agile private infrastructure to run virtualized service workloads within the administrative domain**. OpenNebula virtual infrastructure interfaces expose **user and administrator functionality for virtualization, networking, image and physical resource configuration, management, monitoring and accounting**.

### 1.1.2 The User View

An OpenNebula Private Cloud provides infrastructure users with an **elastic platform for fast delivery and scalability of services to meet dynamic demands of service end-users**. Services are hosted in VMs, and then submitted, monitored and controlled in the Cloud by using the virtual infrastructure interfaces:

- Command line interface
- XML[1]-RPC[2] API[3]
- Libvirt virtualization API[3] or any of its management tools

[1]Extensible Markup Language
[2]Remote Procedure Call
[3]Application Programming Interface

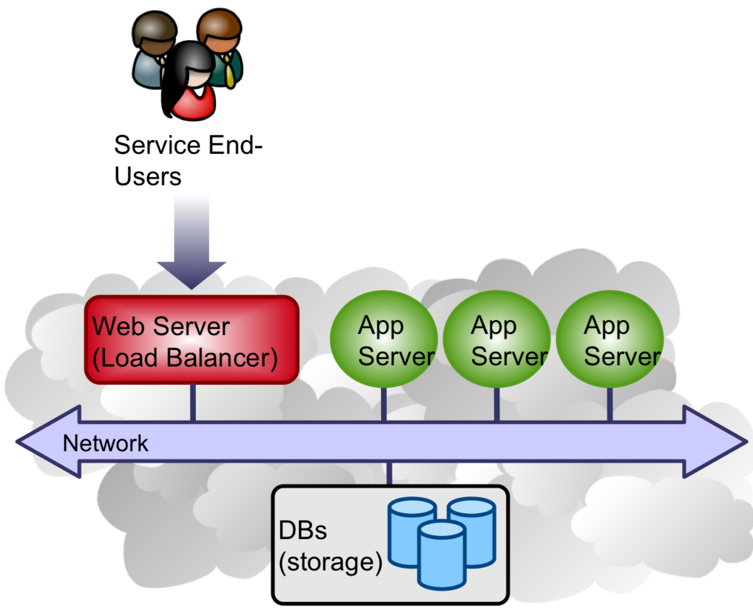


Figure 1.2:

Lets do a **sample session to illustrate the functionality provided by the OpenNebula CLI for Private Cloud Computing**. First thing to do, **check the hosts in the physical cluster**:

We can then **submit a VM** to OpenNebula, by using `onevm`. We are going to build a VM template to submit the image we had previously placed in the `/opt/nebula/images` directory.

```
CPU    = 0.5
MEMORY = 128
OS     = [
  kernel   = "/boot/vmlinuz-2.6.18-4-xen-amd64",
  initrd   = "/boot/initrd.img-2.6.18-4-xen-amd64",
  root     = "sda1" ]
DISK   = [
  source   = "/opt/nebula/images/disk.img",
  target   = "sda1",
  readonly = "no" ]
DISK   = [
  type     = "swap",
  size     = 1024,
  target   = "sdb"]
NIC    = [ NETWORK = "Public VLAN" ]
```

Once we have tailored the requirements to our needs (specially, CPU and MEMORY fields), ensuring that the VM *fits* into at least one of both hosts, let's submit the VM (assuming you are currently in your home folder):

This should come back with an ID, that we can use to identify the VM for **monitoring and controlling**, again through the use of the `onevm` command:

The **STAT** field tells the state of the virtual machine. If there is an **runn** state, the virtual machine is up and running. Depending on how we set up the image, we may be aware of it's IP address. If that is the case we can try now and log into the VM.

To **perform a migration**, we use yet again the `onevm` command. Let's move the VM (with VID=0) to *host02* (HID=1):

This will move the VM from *host01* to *host02*. The `onevm list` shows something like the following:

### 1.1.3 How the System Operates

**OpenNebula does the following**:

- **Manages Virtual Networks**. Virtual networks interconnect VMs. Each Virtual Networks includes a description.
- **Creates VMs**. The VM description is added to the database.
- **Deploys VMs**. According to the allocation policy, the scheduler decides where to execute the VMs.
- **Manages VM Images**. Before execution, VM images are transferred to the host and swap disk images are created. After execution, VM images may be copied back to the repository.
- **Manages Running VMs**. VM are started, periodically polled to get their consumption and state, and can be shutdown, suspended, stopped or migrated.

The **main functional components of an OpenNebula Private Cloud** are the following:

- **Hypervisor**: Virtualization manager installed in the resources of the cluster that OpenNebula leverages for the management of the VMs within each host.
- **Virtual Infrastructure Manager**: Centralized manager of VMs and resources, providing virtual network management, VM life-cycle management, VM image management and fault tolerance.
- **Scheduler**: VM placement policies for balance of workload, server consolidation, placement constraints, affinity, advance reservation of capacity and SLA commitment.

## 1.2 Planning the Installation

### 1.2.1 Overview

OpenNebula assumes that your physical infrastructure adopts a classical cluster-like architecture with a front-end, and a set of cluster nodes where Virtual Machines will be executed. There is at least one physical network joining all the cluster nodes with the front-end.

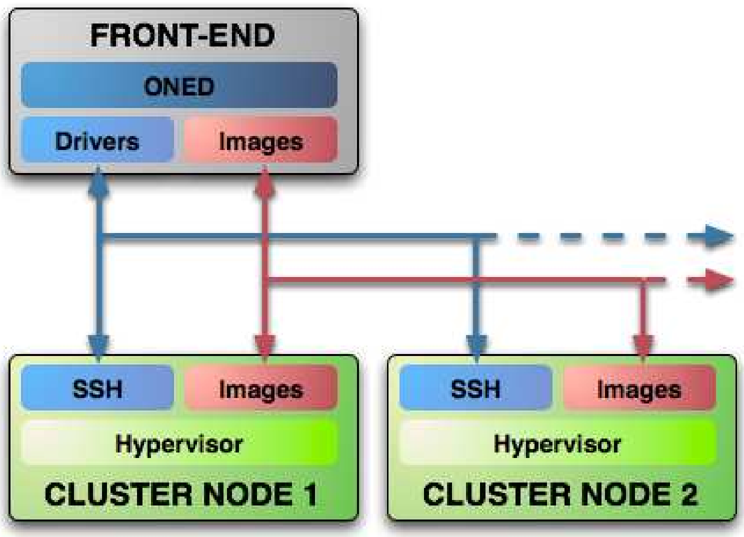


Figure 1.3: high level architecture of cluster, its components and relationship

The basic components of an OpenNebula system are:

- **Front-end**, executes the OpenNebula and cluster services.
- **Nodes**, hypervisor-enabled hosts that provides the resources needed by the Virtual Machines.
- **Image repository**, any storage medium that holds the base images of the VMs.
- **OpenNebula daemon**, is the core service of the system. It manages the life-cycle of the VMs and orchestrates the cluster subsystems (network, storage and hypervisors)
- **Drivers**, programs used by the core to interface with an specific cluster subsystem, e.g. a given hypervisor or storage file system.
- **oneadmin**, is the administrator of the private cloud that performs any operation on the VMs, virtual networks, nodes or users.
- **Users**, use the OpenNebula facilities to create and manage their own virtual machines and virtual networks.

### 1.2.2 System Requirements

#### Cluster Front-End

This section details the software that you need to install in the front-end to run and build OpenNebula. The front-end will access the image repository that should be big enough to store the VM images for your private cloud. Usually, these are master images that are cloned (copied) when you start the VM. So you need to plan your storage requirements depending on the number of VMs you'll run in your virtual infrastructure (see the  section below for more information). The base installation of OpenNebula only takes 10MB.

**Installation Mode** OpenNebula can be installed in two modes:

- **system-wide**: binaries, log files and configuration files will be placed in standard UNIX location under the root file-system. You will need root access to perform this installation.
- **self-contained**: the OpenNebula distribution is placed in a self-contained location. This is the **preferred** installation mode.

In either case, you **do not need** the root account to run the OpenNebula services.

**Software Packages** This machine will act as the OpenNebula server and therefore needs to have installed the following software:

- **ruby** >= 1.8.6 and < 1.9.0
- **sqlite3** >= 3.5.2
- **xmlrpc-c** >= 1.06
- **openssl** >= 0.9
- **ssh**

Additionally, to build OpenNebula from source you neeed:

- Development versions of the **sqlite3**, **xmlrpc-c** and **openssl** packages, if your distribution does not install them with the libraries.
- **scons** >= 0.97
- **g++** >= 4
- **flex** >= 2.5 (optional, only needed to rebuild the parsers)
- **bison** >= 2.3 (optional, only needed to rebuild the parsers)

**Optional Packages** These packages are not needed to run or build OpenNebula. They improve the performance of the user-land libraries and tools of OpenNebula, nor the core system. You will probably experiment a more responsive CLI

First install rubygems and ruby development libraries:

- ruby-dev
- rubygems
- rake
- make

Then install the following packages:

- **ruby xmlparser**, some distributions include a binary package for this (libxml-parser-ruby1.8). If it is not avilable in your distribution install `expat` libraries with its development files and install xmlparser using gem:

Note the extra parameters to gem install. Some versions of xmlparser have problems building the documentation and we can use it without documentation installed.

Bear in mind that in some Linux flavors you will need to install the ”expat-devel” package to be able to install the xmlparser.

- **ruby nokogiri**, to install this gem you will need libxml2 and libxslt libraries and their development versions. The we can install `nokogiri` library:

#### Cluster Node

The nodes will run the VMs and does not need any additional storage requirement, see the storage section for more details.

#### Software Packages

These are the requirements in the cluster nodes that will run the VM's:

- ssh server running
- hypervisor working properly configured
- **ruby** >= 1.8.5

### 1.2.3 Preparing the Cluster

This section guides you through steps needed to prepare your cluster to run a private cloud.

#### Storage

In this guide we assume that the image repository and the OpenNebula `var` directory can be access in the cluster nodes by a Shared FS of any kind. In this way you can take full advantage of the hypervisor capabilities (i.e. live migration) and OpenNebula Storage module (i.e. no need to always clone images).

OpenNebula **can work without a Shared FS**, this will make you to always clone the images and you will only be able to do *cold* migrations. However this non-shared configuration does not impose any significant storage requirements. If you want to use this configuration check the Storage Customization guide and skip this section.

The cluster front-end will export the image repository and the OpenNebula installation directory to the cluster nodes. The size of the image repository depends on the number of images (and size) you want to store. Also when you start a VM you will be usually cloning (copying) it, so you must be sure that there is enough space to store all the running VM images.

Create the following hierarchy in the front-end root file system:

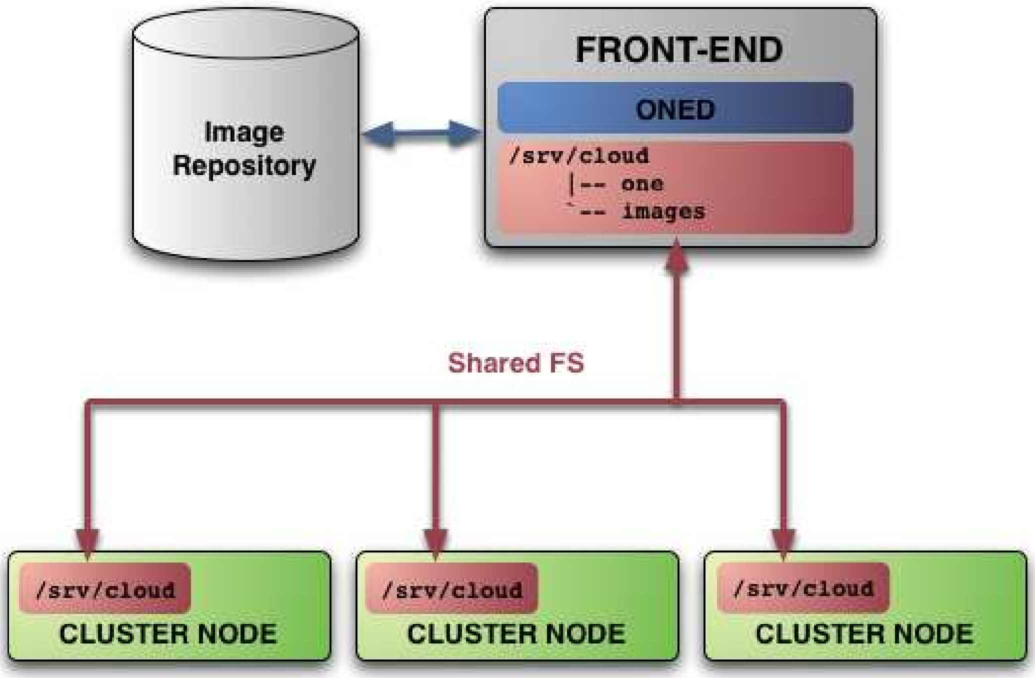


Figure 1.4: Storage Model : NFS

- `/srv/cloud/one`, will hold the OpenNebula installation and the clones for the running VMs
- `/srv/cloud/images`, will hold the master images and the repository

**Example**: A 64 core cluster will typically run around 80VMs, each VM will require an average of 10GB of disk space. So you will need ˜800GB for `/srv/cloud/one`, you will also want to store 10-15 master images so ˜200GB for `/srv/cloud/images`. A 1TB `/srv/cloud` will be enough for this example setup.

Export `/srv/cloud` to all the cluster nodes. For example, if you have all your physical nodes in a local network with address 192.168.0.0/24 you will need to add to your a line like this:

In each cluster node create `/srv/cloud` and mount this directory from the front-end.

### User Account

The Virtual Infrastructure is **administered** by the `oneadmin` account, this account will be used to run the OpenNebula services and to do regular administration and maintenance tasks.

❗ OpenNebula supports multiple users to create and manage virtual machines and networks. You will create these users later when configuring OpenNebula.

Follow these steps:

- Create the `cloud` group where OpenNebula administrator user will be:
- Create the OpenNebula administrative account (`oneadmin`), we will use OpenNebula directory as the home directory for this user:
- Get the user and group id of the OpenNebula administrative account. This id will be used later to create users in the cluster nodes with the same id: In this case the user id will be 1001 and group also 1001.
- Create the group account also on every node that run VMs. Make sure that its id is the same as in the frontend. In this example 1001:

You can use any other method to make a common `cloud` group and `oneadmin` account in the nodes, for example NIS[4].

### Network

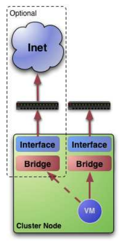


Figure 1.5:

There are no special requirements for networking, apart from those derived from the previous configuration steps. However to make an effective use of your VM deployments you'll probably need to make one or more physical networks accessible to them.

This is achieved with ethernet bridging, there are several guides that explain how to deal with this configuration, check for example the networking howto in the XEN documentation.

For example, a typical cluster node with two physical networks one for public IP addresses (attached to `eth0` NIC) and the other for private virtual LANs (NIC `eth1`) should have two bridges:

Please for **more details on using virtual networks** check the Virtual Network Usage Guide and the Networking Customization Guide

### Secure Shell Access

You need to create **`ssh` keys for `oneadmin` user** and configure machines so it can connect to them using `ssh` without need for a password.

- Generate `oneadmin` `ssh` keys: When prompted for password press enter so the private key is not encripted.

- Copy public key to `~/.ssh/authorized_keys` to let `oneadmin` user log without the need to type password. Do that also for the frontend:

- Tell ssh client to not ask before adding hosts to `known_hosts` file. This goes into `~/.ssh/config`:

Check that the `sshd` daemon is running in the cluster nodes. `oneadmin` must able to log in the cluster nodes without being promt for a password. Also remove any `Banner` option from the `sshd_config` file in the cluster nodes.

### Hypervisor

The virtualization technology installed in your cluster nodes, have to be configured so the `oneadmin` user can start, control and monitor VMs. This usually means the execution of commands with root privileges or making `oneadmin` part of a given group. Please take a look to the virtualization guide that fits your site:

- Xen Configuration Guide

- KVM Configuration Guide

- VMware Configuration Guide

[4]Network Information Service

### 1.2.4 Planning Checklist

If you have followed the previous steps your cluster should be ready to install and configure OpenNebula. You may want to print the following checklist to check your plan and proceed with the installation and configuration steps.

| **Software Requirements** | |
|---|---|
| **ACTION** | **DONE/COMMENTS** |
| Installation type: self-contained, system-wide | self-contained |
| Installation directory | `/srv/cloud/one` |
| OpenNebula software downloaded to `/srv/cloud/one/SRC` | |
| sqlite, g++, scons, ruby and software requirements installed | |
| **User Accounts** | |
| **ACTION** | **DONE/COMMENTS** |
| `oneadmin` account and `cloud` group ready in the nodes and front-end | |
| **Storage Checklist** | |
| **ACTION** | **DONE/COMMENTS** |
| `/srv/cloud` structure created in the front-end | |
| `/srv/cloud` exported and accessible from the cluster nodes | |
| mount point of `/srv/cloud` in the nodes if different | VMDIR=<mount_point>/var/ |
| **Cluster nodes Checklist** | |
| **ACTION** | **DONE/COMMENTS** |
| hostnames of cluster nodes | |
| ruby, sshd installed in the nodes | |
| `oneadmin` can ssh the nodes paswordless | |

## 1.3 Platform Notes

### 1.3.1 Ubuntu/Kubuntu

Ubuntu comes with OpenNebula packages, for easy installation:

- **libopennebula-dev** - OpenNebula client library - Development
- **libopennebula1** - OpenNebula client library - Runtime
- **opennebula** - OpenNebula controller
- **opennebula-common** - OpenNebula common files
- **opennebula-node** - OpenNebula node

If you want to build OpenNebula install the following packages:

- **libsqlite3-dev**
- **libxmlrpc-c3-dev**
- **scons**
- **g++**
- **ruby**
- **libssl-dev**

Optional software:

- **ruby-dev**
- **make**
- **rake**
- **rubygems**
- **libxml-parser-ruby1.8**
- **libxslt1-dev**
- **libxml2-dev**
- **nokogiri** (ruby gem)

### 1.3.2 Debian Lenny

To build OpenNebula install the following packages:

- **gcc c++ compiler**: g++
- **ruby**: ruby
- **sqlite3**: libsqlite3-0, sqlite3
- **sqlite3-dev** : libsqlite3-dev
- **sqlite3-ruby**: libsqlite3-ruby
- **libxmlrpc-c**: libxmlrpc-c3-dev, libxmlrpc-c3
- **libssl**: libssl-dev
- **scons**: scons

The xen packages on Lenny seems to be broken, and they don't work with the tap:aio interface. Sander Klous in the mailing proposes the following workaround:

# ln -s /usr/lib/xen-3.2-1/bin/tapdisk /usr/sbin # echo xenblktap >> /etc/modules # reboot

### 1.3.3 Fedora 8

Not known issues.

### 1.3.4 Gentoo

When installing libxmlrpc you have to specify that it will be compiled with thread support:

```
# USE="threads" emerge xmlrpc-c
```

### 1.3.5 Arch

The xmlrpc-c package available in the extra repository is not compiled with support for the abyss server. Use the Arch Build System (ABS) to include support for the server. Just remove `disable-abyss` from the configure command in the `PKGBUILD` file, and install the package:

```
    cd $srcdir/$pkgname-$pkgver
    ./configure --prefix=/usr \
                --enable-libxml2-backend \
                --disable-cgi-server \
                --disable-libwww-client \
                --disable-wininet-client
```

### 1.3.6 CentOS 5 / RHEL 5

#### Ruby

We can install the standard packages directly with yum:

$ yum install ruby ruby-devel ruby-docs ruby-ri ruby-irb ruby-rdoc

To install rubygems we must activate the EPEL repository:

$ wget http://download.fedora.redhat.com/pub/epel/5/i386/epel-release-5-3.noarch.rpm $ yum localinstall epel-release-5-3.noarch.rpm $ yum install rubygems

Once rubygems is installed we can install the following gems:

gem install nokogiri rake xmlparser

#### scons

The version that comes with Centos is not compatible with our build scripts. To install a more recent version you can download the RPM at:

http://www.scons.org/download.php

$ wget http://prdownloads.sourceforge.net/scons/scons-1.2.0-1.noarch.rpm $ yum localinstall scons-1.2.0-1.noarch.rpm

#### xmlrpc-c

You can download the xmlrpc-c and xmlrpc-c packages from the rpm repository at http://centos.karan.org/.

$ wget http://centos.karan.org/el5/extras/testing/i386/RPMS/xmlrpc-c-1.06.18-1.el5.kb.i386.rpm $ wget http://centos.karan.org/el5/extras/testing/i386/RPMS/xmlrpc-c-devel-1.06.18-1.el5.kb.i386.rpm $ yum localinstall xmlrpc-c-1.06.18-1.el5.kb.i386.rpm xmlrpc-c-devel-1.06.18-1.el5.kb.i386.rpm

#### sqlite

This package should be installed from source, you can download the tar.gz from http://www.sqlite.org/download.html. It was tested with sqlite 3.5.9.

$ wget http://www.sqlite.org/sqlite-amalgamation-3.6.17.tar.gz $ tar xvzf sqlite-amalgamation-3.6.17.tar.gz $ cd sqlite-3.6.17/ $ ./configure $ make $ make install

If you do not install it to a system wide location (/usr or /usr/local) you need to add LD_LIBRARY_PATH and tell scons where to find the files:

$ scons sqlite=<path where you installed sqlite>

### 1.3.7 openSUSE 11

#### Building tools

By default openSUSE 11 does not include the standard building tools, so before any compilation is done you should install:

$ zypper install gcc gcc-c++ make patch

#### Required Libraries

Install these packages to satisfy all the dependencies of OpenNebula:

$ zypper install libopenssl-devel libcurl-devel scons pkg-config sqlite3-devel libxslt-devel libxmlrpc_server_abyss++3 libxmlrpc_client++3 libexpat-devel libxmlrpc_server++3

#### Ruby

We can install the standard packages directly with zypper:

$ zypper install ruby ruby-doc-ri ruby-doc-html ruby-devel rubygems

rubygems must be >=1.3.1, so to play it safe you can update it to the latest version:

$ wget http://rubyforge.org/frs/download.php/45905/rubygems-1.3.1.tgz $ tar zxvf rubygems-1.3.1.tgz $ cd rubygems-1.3.1 $ ruby setup.rb $ gem update –system

Once rubygems is installed we can install the following gems:

gem install nokogiri rake xmlparser

#### xmlrpc-c

xmlrpc-c must be built by downloading the latest svn release and compiling it. Read the README file included with the package for additional information.

svn co http://xmlrpc-c.svn.sourceforge.net/svnroot/xmlrpc-c/super_stable xmlrpc-c cd xmlrpc-c ./configure make make install

### 1.3.8 MAC OSX 10.4 10.5

OpenNebula frontend can be installed in Mac OS[5] X. Here are the instructions to build it in 10.5 (Leopard)

Requisites:

- **xcode** (you can install in from your Mac OS[5] X DVD)
- **macports** http://www.macports.org/

#### Getopt

This package is needed as `getopt` that comes with is BSD style.

$ sudo port install getopt

#### xmlrpc

$ sudo port install xmlrpc-c

#### scons

You can install scons using macports as this:

$ sudo port install scons

Unfortunately it will also compile python an lost of other packages. Another way of getting it is downloading the standalone package in http://www.scons.org/download.php. Look for scons-local Packages and download the Gzip tar file. In this example I am using version 1.2.0 of the package.

$ mkdir -p ~/tmp/scons $ cd ~/tmp/scons $ tar xvf ~/Downloads/scons-local-1.2.0.tar $ alias scons='python ~/tmp/scons/scons.py'

#### OpenNebula

$ scons -j2

## 1.4 Installing the Software

! Do not forget to check the Platform Notes for a list of specific software requirements to build OpenNebula.

Follow these simple steps to install the OpenNebula software:

- Download and untar the OpenNebula tarball.
- Change to the created folder and run scons to compile OpenNebula the argument expression [OPTIONAL] is used to set non-default paths for :

| OPTION | VALUE |
|---|---|
| sqlite | path-to-sqlite-install |
| xmlrpc | path-to-xmlrpc-install |
| parsers | **yes** if you want to rebuild flex/bison files |

[5]Operating System

- OpenNebula can be installed in two modes: `system-wide`, or in `self-contained` directory. In either case, you do not need to run OpenNebula as root. These options can be specified when running the install script: where <*install_options*> can be one or more of:

| **OPTION** | **VALUE** |
|---|---|
| **-u** | user that will run OpenNebula, defaults to user executing install.sh |
| **-g** | group of the user that will run OpenNebula, defaults to user executing install.sh |
| **-k** | keep current configuration files, useful when upgrading |
| **-d** | target installation directory. If defined, it will specified the path for the **self-contained** install. If not defined, the installation will be performed **system wide** |
| **-r** | remove Opennebula, only useful if -d was not specified, otherwise `rm -rf $ONE_LOCATION` would do the job |
| **-h** | prints installer help |

We will do a `self-contained` installation. As `oneadmin` user:
Now you need to configure OpenNebula to adjust it to your particular cluster.

### 1.4.1 Verifying the Installation

Depending on the installation mode (defined by the presence or absence of the **-d** option in the installation script), there are two possibilities with respect to the layout of the OpenNebula files.

#### Self contained

Once the OpenNebula software is installed specifying a directory with the **-d** option, the next tree should be found under `$ONE_LOCATION`:

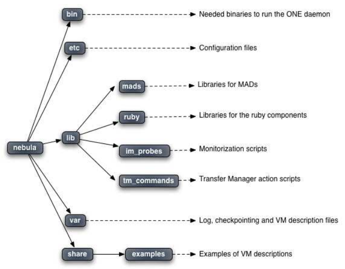

nebula
bin
Needed binaries to run the ONE daemon
etc
Configuration files
lib
mads
Libraries for MADs
ruby
Libraries for the ruby components
im_probes
Monitorization scripts
tm_commands
Transfer Manager action scripts
var
Log, checkpointing and VM description files
share
examples
Examples of VM descriptions

Figure 1.6:

#### System wide

Once the OpenNebula software is installed without specifying a directory with the **-d** option, the next tree reflects the files installed:

## 1.5 Configuring OpenNebula

### 1.5.1 OpenNebula Components

OpenNebula comprises the execution of three type of processes:

- The *OpenNebula daemon* (`oned`), to orchestrate the operation of all the modules and control the VM's life-cycle
- The *drivers* to access specific cluster systems (e.g. storage or hypervisors)
- The *scheduler* to take VM placement decisions

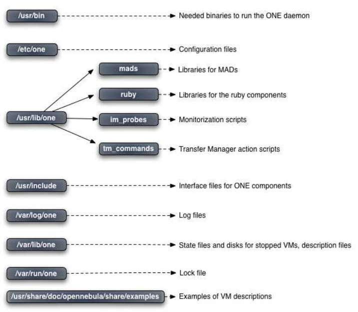

/usr/bin
Needed binaries to run the ONE daemon
/etc/one
Configuration files
mads
Libraries for MADs
ruby
Libraries for the ruby components
/usr/lib/one
im_probes
Monitorization scripts
tm_commands
Transfer Manager action scripts
/usr/include
Interface files for ONE components
/var/log/one
Log files
/var/lib/one
State files and disks for stopped VMs, description files
/var/run/one
Lock file
/usr/share/doc/opennebula/share/examples
Examples of VM descriptions

Figure 1.7:

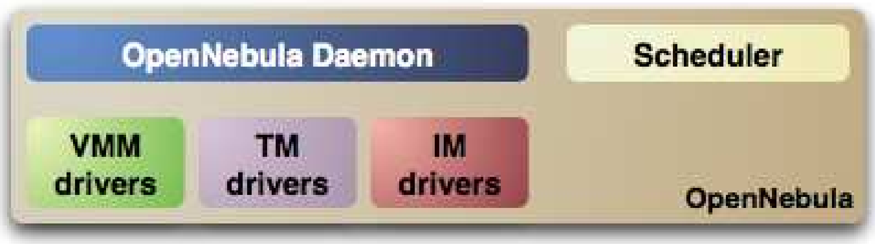


Figure 1.8: high level architecture of cluster, its components and relationship

In this section you'll learn how to configure and start these services.

**OpenNebula Daemon**

The configuration file for the daemon is called `oned.conf` and it is placed inside the `$ONE_LOCATION/etc` directory (or in `/etc/one` if OpenNebula was installed system wide).

❗ A detailed description of all the configuration options for the OpenNebula daemon can be found in the oned.conf reference document

The `oned.conf` file consists in the following sections:

- *General configuration attributes*, like the time between cluster nodes and VM monitorization actions or the MAC prefix to be used. See more details\ldots
- *Information Drivers*, the specific adaptors that will be used to monitor cluster nodes. See more details\ldots
- *Virtualization Drivers*, the adaptors that will be used to interface the hypervisors. See more details\ldots
- *Transfer Drivers*, that are used to interface with the storage system to clone, delete or move VM images. See more details\ldots
- *Hooks*, that are executed on specific events, e.g. VM creation. See more details\ldots

The following example will configure OpenNebula to work with KVM and a shared FS:

```
# Attributes
HOST_MONITORING_INTERVAL = 60
VM_POLLING_INTERVAL      = 60

VM_DIR = /srv/cloud/one/var #Path in the cluster nodes to store VM images

NETWORK_SIZE = 254     #default
MAC_PREFIX   = "00:03"
```

```
#Drivers
IM_MAD = [name="im_kvm", executable="one_im_ssh", arguments="im_kvm/im_kvm.conf"]
VM_MAD = [name="vmm_kvm",executable="one_vmm_kvm",
          default="vmm_kvm/vmm_kvm.conf", type= "kvm" ]
TM_MAD = [name="tm_nfs", executable="one_tm", arguments="tm_nfs/tm_nfs.conf" ]
```

Be sure that `VM_DIR` is set to the path where the front-end's $ONE_LOCATION/var directory is mounted in the cluster nodes.

**Scheduler**

The Scheduler module is in charge of the assignment between pending Virtual Machines and cluster nodes. OpenNebula's architecture defines this module as a separate process that can be started independently of `oned`. OpenNebula comes with a `match making` scheduler (*mm_sched*) that implements the *Rank Scheduling Policy*.

The goal of this policy is to prioritize those resources more suitable for the VM. You can configure several resource and load aware policies by simply specifying specific `RANK` expressions in the Virtual Machine definition files. Check the scheduling guide to learn how to configure these policies.

You can use OpenNebula without the scheduling process to operate it in a VM management mode. Start or migration of VMs in this case is explicitly performed using the `onevm` command.

The Haizea lease manager can also be used as a scheduling module in OpenNebula. Haizea allows OpenNebula to support advance reservation of resources and queuing of best effort requests.

**Drivers**

Drivers are separate processes that communicate with the OpenNebula core using an internal ASCII[6] protocol. Before loading a driver, two *run commands* (RC) files are sourced to optionally obtain environmental variables.

These two *RC* files are:

- `$ONE_LOCATION/etc/mad/defaultrc`. Global environment and tasks for all the drivers. Variables are defined **using sh** syntax, and upon read, exported to the driver's environment:

```
# Debug for MADs [0=ERROR, 1=DEBUG]
# If set, MADs will generate cores and logs in $ONE_LOCATION/var.
ONE_MAD_DEBUG=
# Nice Priority to run the drivers
PRIORITY=19
```

- Specific file for each driver, that may re-defined the `defaultrc` variables. Please see each driver's configuration guide for specific options:
  - Information Guide for the monitor drivers
  - Storage Guide for the transfer drivers
  - the Virtualization Adaptors of the Documentation for the different virtualization drivers

### 1.5.2 Start & Stop OpenNebula

When you execute OpenNebula for the first time it will create an administration account. Be sure to put the user and password in a single line as user:password in the `$ONE_AUTH` file.

The OpenNebula daemon and the scheduler can be easily started with the `$ONE_LOCATION/bin/one` script. Just execute as the <oneadmin> user:

If you do not want to start the scheduler just use `oned`, check `oned -h` for options.

Now we should have running two process:

[6] American Standard Code for Information Interchange

* `oned` : Core process, attends the CLI requests, manages the pools and all the components
* `mm_sched` : Scheduler process, in charge of the VM to cluster node matching

If those process are running, you should see content in their log files (log files are placed in `/var/log/one/` if you installed OpenNebula system wide):

* `$ONE_LOCATION/var/oned.log`
* `$ONE_LOCATION/var/sched.log`

### 1.5.3 OpenNebula Users

There are two account types in the OpenNebula system:

* The **oneadmin** account is created **the first time** OpenNebula is started using the ONE_AUTH data, see below. `oneadmin` has enough privileges to perform any operation on any object (virtual machine, network, host or user)
* **Regular user** accounts must be created by <`oneadmin`> and they **can only manage their own objects** (virtual machines and networks)

Virtual Networks created by `oneadmin` are *public* and can be used by every other user.
OpenNebula users should have the following environment variables set:

| | |
|---|---|
| **ONE_AUTH** | Needs to point to **a file containing just a single line stating "username:password"**. If ONE_AUTH is not defined, $HOME/.one/one_auth will be used instead. If no auth file is present, OpenNebula cannot work properly, as this is needed by the core, the CLI, and the cloud components as well. |
| **ONE_LOCATION** | If OpenNebula was installed in **self-contained** mode, this variable must be set to <destination_folder>. Otherwise, in **system wide** mode, this variable must be unset. More info on installation modes can be found here |
| **ONE_XMLRPC** | http://localhost:2633/RPC2 |
| **PATH** | `$ONE_LOCATION/bin`:$PATH if **self-contained**. Otherwise this is not needed. |

#### Adding and Deleting Users

User accounts within the OpenNebula system are managed by <`oneadmin`> with the `oneuser` utility. Users can be easily added to the system like this:

In this case user `helen` should include the following content in the $ONE_AUTH file:

Users can be deleted by simply:

To list the users in the system just issue the command:

Detailed information of the `oneuser` utility can be found in the Command Line Reference

### 1.5.4 OpenNebula Hosts

Finally to set up the cluster, the nodes have to be added to the system as OpenNebula hosts. You need the following information:

* *Hostname* of the cluster node or IP
* *Information Driver* to be used to monitor the host, e.g. `im_kvm`.

- *Storage Driver* to clone, delete, move or copy images into the host, e.g. `tm_nfs`.
- *Virtualization Driver* to boot, stop, resume or migrate VMs in the host, e.g. `vmm_kvm`.

❗ Before adding a host check that you can ssh to it without being prompt for a password

#### Adding and Deleting Hosts

Hosts can be added to the system anytime with the `onehost` utility. You can add the cluster nodes to be used by OpenNebula, like this:

The status of the cluster can be check with the `list` command:

And specific information about a host with `show`:

If you want not to use a given host you can temporarily disable it:

A disable host should be listed with `STAT off` by `onehost list`. You can also remove a host permanently with:

❗ Detailed information of the `onehost` utility can be found in the Command Line Reference

### 1.5.5 Logging and Debugging

There are different log files corresponding to different OpenNebula components:

- **ONE Daemon**: The core component of OpenNebula dumps all its logging information onto `$ONE_LOCATION/var/oned.log`. Its verbosity is regulated by DEBUG_LEVEL in `$ONE_LOCATION/etc/oned.conf`.
- **Scheduler**: All the scheduler information is collected into the $ONE_LOCATION/var/sched.log file.
- **Virtual Machines**: All VMs controlled by OpenNebula have their folder, `$ONE_LOCATION/var/<VID>/` (or `/var/lib/one/<VID>` in a system wide installation). You can find the following information in it:
  - **Log file** : The information specific of the VM will be dumped in a file in this directory called **vm.log**. **Note**: These files are in `/var/log/one` if OpenNebula was installed system wide.
  - **Deployment description files** : Stored in `deployment.<EXECUTION>`, where `<EXECUTION>` is the sequence number in the execution history of the VM (deployment.0 for the first host, deployment.1 for the second and so on).
  - **Transfer description files** : Stored in `transfer.<EXECUTION>.<OPERATION>`, where `<EXECUTION>` is the sequence number in the execution history of the VM, `<OPERATION>` is the stage where the script was used, e.g. transfer.0.prolog, transfer.0.epilog, or transfer.1.cleanup.
  - **Save images**: Stored in `images/` sub-directory, images are in the form `disk.<id>`.
  - **Restore files** : check-pointing information is also stored in this directory to restore the VM in case of failure. The state information is stored in a file called `checkpoint`.
- **Drivers**: Each driver can have activated its **ONE_MAD_DEBUG** variable in their **RC** files (see the Drivers configuration section for more details). If so, error information will be dumped to `$ONE_LOCATION/var/name-of-the-driver-executable.log`; log information of the drivers is in `oned.log`.

# Chapter 2

# Configuring the Virtualization Adaptors

## 2.1 Xen Adaptor

The XEN hypervisor offers a powerful, efficient and secure feature set for virtualization of x86, IA64, PowerPC and other CPU architectures. It delivers both paravirtualization and full virtualization. This guide describes the use of Xen with OpenNebula, please refer to the Xen specific documentation for further information on the setup of the Xen hypervisor itself.

### 2.1.1 Xen Configuration

The cluster nodes must have a working installation of Xen that includes a Xen aware kernel running in Dom0 and the Xen utilities. In each cluster node you must perform the following steps to get the driver running:

- The remote hosts must have the xend daemon running (`/etc/init.d/xend`) and a `XEN` aware kernel running in Dom0
- The <`oneadmin`> user may need to execute Xen commands using root privileges. This can be done by adding this two lines to the `sudoers` file of the cluster nodes so <`oneadmin`> user can execute Xen commands as root (change paths to suit your installation):

```
%xen    ALL=(ALL) NOPASSWD: /usr/sbin/xm *
%xen    ALL=(ALL) NOPASSWD: /usr/sbin/xentop *
```

- You may also want to configure network for the virtual machines. OpenNebula assumes that the VMs have network access through standard bridging, please refer to the Xen documentation to configure the network for your site.
- Some distributions have **requiretty** option enabled in the `sudoers` file. It must be disabled to so ONE can execute commands using sudo. The line to comment out is this one:

```
Defaults requiretty
```

### 2.1.2 Driver Files

The driver consists of the following files:

- `$ONE_LOCATION/lib/mads/one_vmm_xen` : Shell script wrapper to the driver itself. Sets the environment and other bootstrap tasks.

- `$ONE_LOCATION/lib/mads/one_vmm_xen.rb` : The actual XEN driver.
- `$ONE_LOCATION/etc/vmm_xen/vmm_xenrc` : environment setup and bootstrap instructions
- `$ONE_LOCATION/etc/vmm_xen/vmm_xen.conf` : set here default values for XEN domain definitions.

**Note:** If OpenNebula was installed in **system wide** mode these directories become `/usr/lib/one/-mads` and `/etc/one/`, respectively. The rest of this guide refers to the $ONE_LOCATION paths (corresponding to **self contained** mode) and omits the equivalent **system wide** locations. More information on installation modes can be found here

### 2.1.3 Configuration

#### OpenNebula Configuration

OpenNebula needs to know if it is going to use the XEN Driver. To achieve this, two lines have to be placed within `$ONE_LOCATION/etc/oned.conf`, one for the VM driver and other for the IM driver:

```
IM_MAD = [
    name       = "im_xen",
    executable = "one_im_ssh",
    arguments  = "im_xen/im_xen.conf",
    default    = "im_xen/im_xen.conf" ]

VM_MAD = [
    name       = "vmm_xen",
    executable = "one_vmm_xen",
    default    = "vmm_xen/vmm_xen.conf",
    type       = "xen" ]
```

- The **name** of the driver needs to be provided at the time of adding a new cluster node to OpenNebula.
- **executable** points to the path of the driver executable file. It can be an absolute path or relative to `$ONE_LOCATION/lib/mads`.
- The **default** points to the configuration file for the driver (see below). It can be an absolute path or relative to `$ONE_LOCATION/etc`.
- **type** identifies this driver as a XEN driver.

#### Driver Configuration

The driver uses two configuration files, by default placed in `$ONE_LOCATION/etc/vmm_xen`:

- **Defaults** file, specified in the default attribute in the driver specification line in oned.conf, usually`$ONE_LOCATION/etc/vmm_xen/vmm_xen.conf`. This file is home for default values for domain definitions (in other words, OpenNebula templates). Let's go for a more concrete and VM related example. If the user wants to set a default value for CPU requirements for all of their XEN domain definitions, simply edit the `vmm_xenrc` file and set a

```
CPU=0.6
```

into it. Now, when defining a ONE template to be sent to a XEN resource, the user has the choice of "forgetting" to set the **CPU** requirement, in which case it will default to 0.6.

It is generally a good idea to place defaults for the XEN-specific attributes, that is, attributes mandatory in the XEN driver that are not mandatory for other hypervisors. Non mandatory attributes for XEN but specific to them are also recommended to have a default.

- **Run commands** file, the `$ONE_LOCATION/etc/vmm_xen/vmm_xenrc` file contains environment variables for the driver. You may need to tune the values for `XENTOP_PATH` and `XM_PATH`, if any of either `/usr/sbin/xentop` and `/usr/sbin/xm` do not live in their default locations in the remote hosts. This file can also hold instructions to be executed before the actual driver load to perform specific tasks or to pass environmental variables to the driver. The syntax used for the former is plain shell script that will be evaluated before the driver execution. For the latter, the syntax is the familiar:

```
ENVIRONMENT_VARIABLE=VALUE
```

### 2.1.4 Xen Specific Template Attributes

The following are template attributes specific to Xen, please refer to the OpenNebula user guide for a complete list of the attributes supported to define a VM.

#### Optional Attributes

- **CREDIT** : Xen comes with a credit scheduler. The credit scheduler is a proportional fair share CPU scheduler built from the ground up to be work conserving on SMP hosts. This attribute sets a 16 bit value that will represent the amount of sharing this VM will have respect to the others living in the same host. This value is set into the driver configuration file, is not intended to be defined per domain.

#### Additional Tunning

The **raw** attribute offers the end user the possibility of passing by attributes not known by OpenNebula to Xen. Basically, everything placed here will be written ad literally into the Xen deployment file.

```
RAW = [ type="xen", data="on_crash=destroy" ]
```

### 2.1.5 Testing

In order to test the Xen driver, the following template can be instantiated with appropriate values and sent to a Xen resource:

```
CPU      = 1
MEMORY   = 128
OS       = [kernel="/path-to-kernel",initrd= "/path-to-initrd",root="sda1" ]
DISK     = [source="/path-to-image-file",target="sda",readonly="no"]
NIC      = [mac="xx.xx.xx.xx.xx.xx", bridg="eth0"]
GRAPHICS = [type="vnc",listen="127.0.0.1",port="5900"]
```

## 2.2 KVM Adaptor

KVM (Kernel-based Virtual Machine) is a complete virtualization technique for Linux. It offers full virtualization, where each Virtual Machine interacts with its own virtualized hardware. This guide describes the use of the KVM virtualizer with OpenNebula, please refer to KVM specific documentation for further information on the setup of the KVM hypervisor itself.

### 2.2.1 KVM Configuration

The cluster nodes must have a working installation of KVM, that usually requires:

- CPU with VT extensions
- libvirt >= 0.4.0
- kvm kernel modules (kvm.ko, kvm-intel,amd.ko). Available from kernel 2.6.20 onwards.

- the qemu user-land tools

OpenNebula uses the libvirt interface to interact with KVM, so the following steps are required in the cluster nodes to get the KVM driver running:

- The remote hosts must have the libvirt daemon running.
- The user with access to these remotes hosts on behalf of OpenNebula (typically <`oneadmin`>) has to pertain to the <libvirtd> and <kvm> groups in order to use the deaemon and be able to launch VMs.

**Migration**

OpenNebula uses libvirt's migration capabilities. More precisely, it uses the TCP protocol offered by libvirt. In order to configure the physical nodes, the following files have to be modified:

- `/etc/libvirt/libvirtd.conf` : Uncomment "listen_tcp = 1". Security configuration is left to the admin's choice, file is full of useful comments to achieve a correct configuration.
- `/etc/default/libvirt-bin` : add **-l** option to `libvirtd_opts`

### 2.2.2 Driver Files

The driver consists of the following files:

- `$ONE_LOCATION/lib/mads/one_vmm_kvm` : Shell script wrapper to the driver itself. Sets the environment and other bootstrap tasks.
- `$ONE_LOCATION/lib/mads/one_vmm_kvm.rb` : The actual KVM driver.
- `$ONE_LOCATION/etc/vmm_kvm/vmm_kvmrc` : environment setup and bootstrap instructions
- `$ONE_LOCATION/etc/vmm_kvm/vmm_kvm.conf` : set here default values for KVM domain definitions.

**Note:** If OpenNebula was installed in **system wide** mode these directories become `/usr/lib/one/mads` and `/etc/one/`, respectively. The rest of this guide refers to the `$ONE_LOCATION` paths (corresponding to **self contained** mode) and omits the equivalent **system wide** locations. More information on installation modes can be found here.

### 2.2.3 Configuration

**OpenNebula Configuration**

OpenNebula needs to know if it is going to use the KVM Driver. To achieve this, two lines have to be placed within `$ONE_LOCATION/etc/oned.conf`, one for the VM driver and other for the information (IM) driver:

```
IM_MAD = [
    name       = "im_kvm",
    executable = "one_im_ssh",
    arguments  = "im_kvm/im_kvm.conf",
    default    = "im_kvm/im_kvm.conf" ]

VM_MAD = [
    name       = "vmm_kvm",
    executable = "one_vmm_kvm",
    default    = "vmm_kvm/vmm_kvm.conf",
    type       = "kvm" ]
```

- The **name** of the driver needs to be provided at the time of adding a new host to OpenNebula.

- **executable** points to the path of the driver executable file. It can be an absolute path or relative to `$ONE_LOCATION/lib/mads`.
- The **default** points to the configuration file for the driver (see below). It can be an absolute path or relative to `$ONE_LOCATION/etc`.
- **type** identifies this driver as a KVM driver.

#### Driver Configuration

The driver uses two configuration files, by default placed in `$ONE_LOCATION/etc/vmm_kvm`:

- **Defaults** file, specified in the **default** attribute in the driver specification line in `oned.conf`, usually `$ONE_LOCATION/etc/vmm_kvm/vmm_kvm.conf`. This file is home for default values for domain definitions (in other words, for OpenNebula templates). For example, if the user wants to set a default value for **CPU** requirements for all of their KVM domain definitions, simply edit the `$ONE_LOCATION/etc/vmm_kvm/vmm_kvm.conf` file and set a

```
CPU=0.6
```

into it. Now, when defining a VM to be sent to a KVM resource, the user has the choice of ”forgetting” to set the CPU requirement, in which case it will default to 0.6.

It is generally a good idea to place defaults for the KVM-specific attributes, that is, attributes mandatory in the KVM driver that are not mandatory for other hypervisors. Non mandatory attributes for KVM but specific to them are also recommended to have a default.

- **Run Commands** file, the `$ONE_LOCATION/etc/vmm_kvm/vmm_kvmrc` file holds instructions to be executed before the actual driver load to perform specific tasks or to pass environmental variables to the driver. The syntax used for the former is plain shell script that will be evaluated before the driver execution. For the latter, the syntax is the familiar:

```
ENVIRONMENT_VARIABLE=VALUE
```

### 2.2.4 KVM Specific Template Attributes

The following are template attributes specific to KVM, please refer to the OpenNebula user guide for a complete list of the attributes supported to define a VM.

#### Mandatory Attributes

Specify the boot device to consider in the **OS**[1] attribute, using **BOOT**. Valid values for **BOOT** are `fd`, `hd`, `cdrom` or `network`, that corresponds to the `boot [a|c|d|n]` option of the `kvm` command, respectively. For example:

```
  OS=[
    KERNEL = /vmlinuz,
    BOOT   = hd]
```

#### Optional Attributes

In general you will not need to set the following attributes for you VMs, they are provided to let you fine tune the VM deployment with KVM.

[1] Operating System

**DISK**

- **type**, This attribute defines the type of the media to be exposed to the VM, possible values are: `disk` (default), `cdrom` or `floppy`. This attribute corresponds to the `media` option of the `-driver` argument of the `kvm` command.
- **bus**, specifies the type of disk device to emulate; possible values are driver specific, with typical values being `ide`, `scsi` or `pflash`. This attribute corresponds to the `if` option of the `-driver` argument of the `kvm` command.

**NIC**

- **target**, name for the tun device created for the VM. It corresponds to the `ifname` option of the '-net' argument of the `kvm` command.
- **script**, name of a shell script to be executed after creating the tun device for the VM. It corresponds to the `script` option of the '-net' argument of the `kvm` command.
- **model**, ethernet hardware to emulate. You can get the list of available models with this command:

```
$ kvm -net nic,model=? -nographic /dev/null
```

**Virtio** Virtio is the framework for IO virtualization in KVM. You will need a linux kernel with the virtio drivers for the guest, check check the KVM documentation for more info.

If you want to use the virtio drivers add the following attributes to your devices:

- `DISK`, add the attribute `bus=virtio`
- `NIC`, add the attribute `model=virtio`

**FEATURES**

- **pae**: Physical address extension mode allows 32-bit guests to address more than 4 GB[2] of memory:
- **acpi**: useful for power management, for example, with KVM guests it is required for graceful shutdown to work.

Format and valid values:

```
FEATURES=[
    pae={yes|no},
    acpi={yes|no} ]
```

Default values for this features can be set in the driver configuration file so they don't need to be specified for every VM.

### Additional Tunning

The **raw** attribute offers the end user the possibility of passing by attributes not known by OpenNebula to KVM. Basically, everything placed here will be written literally into the KVM deployment file (**use libvirt xml format and semantics**).

```
  RAW = [ type = "kvm",
          data = "<devices><serial type=\"pty\"><source path=\"/dev/pts/5\"/><target
port=\"0\"/></serial><console type=\"pty\" tty=\"/dev/pts/5\"><source path=\"/dev/pts/5\"/><target
port=\"0\"/></console></devices>" ]
```

[2] Gigabyte

### 2.2.5 Testing

In order to test the KVM driver, the following template can be instantiated with appropriate values and sent to a KVM resource:

```
NAME     = KVM-TEST
CPU      = 1
MEMORY   = 128
OS       = [kernel="/path-to-kernel",initrd="/path-to-initrd",boot=hd ]
DISK     = [source="/path-to-image-file",clone=yes,target=hda,readonly=no]
NIC      = [mac="xx.xx.xx.xx.xx.xx"]
```

## 2.3 VMware Adaptor

The VMware Infrastructure API[3] (VI API[3]) provides a complete set of language-neutral interfaces to the VMware virtual infrastructure management framework. By targeting the VI API[3], the OpenNebula VMware drivers are able to manage various flavors of VMware hypervisors: ESXi (free), ESX and VMware Server.

### 2.3.1 VMware Configuration

#### Requirements

The **front-end** where OpenNebula is installed needs the following software:

- VMware VI SDK 2.5
- Apache Axis 1.4

Besides, a VMware hypervisor is needed in the **cluster nodes**. You can choose and install one of the following hypervisors:

- ESXi *(free)*
- ESX
- VMware Server

#### User Configuration

All the VMware hypervisors (that will be accessed by the same set of drivers) needs to share the same user with the same password. This user can be created through the VMware Infrastructure Client (VIC), in the `User & Groups`. This user needs to have the same UID as the oneadmin user has in the OpenNebula front-end, and this can be set through the VIC as well.

! The VIC that can be downloaded via https://$<$esx-hostname$>$:443/. Please be aware that you will need a Windows machine in order to run it.

#### Datastore Configuration

The default recommended configuration of the storage for VMware hypervisors is to use a shared filesystem between them, ideally `NFS`. This share has to be accessible to the OpenNebula front-end. Its name is identified by $DATASTORE (all the hypervisors have to mount the share with the same name), and its location in the OpenNebula front-end by the environmental variable $DATASTORE_PATH.

This configuration can be set through the VIC as well, `Configuration` tab, Storage link in the left panel, ”Add Storage” in the top right. You will need a `NFS` export with the ”no_root_squash” option enabled (the VIC needs root permission to create and manage VMs).

[3]Application Programming Interface

### 2.3.2 Front-end Configuration

With respect to the **front-end**, the following steps need to be taken in order to configure properly the drivers:

- The VMWare VI SDK[4] following the Setup Guide. You should end up with a keystore containing VMWare certificates installed in the <oneadmin> home folder. Briefly you need to copy the `/etc/vmware/ssl/rui.crt` of all the hypervisors and add them to the OpenNebula front-end java keystore with a command like:

- Add all jars in $AXISHOME/lib and $SDKHOME/samples/Axis/java/lib/ to <oneadmin> CLASSPATH

### 2.3.3 Driver Installation

- Go to OpenNebula source code directory and navigate to src/vmm_mad/vmware. Run the install-vmware.sh script.

### 2.3.4 Driver Files

The drivers consists of the following files:

#### Virtual Machine Manager (VMM)

- `$ONE_LOCATION/lib/mads/*.class` : Driver libraries

- `$ONE_LOCATION/bin/one_vmm_vmware` : Wrapper for VMware Virtual Machine Manager driver

- `$ONE_LOCATION/etc/vmm_vmware/vmm_vmwarerc` : environment setup. Also useful for setting whether the driver should be verbose in the log entries.

#### Information Manager (IM)

- `$ONE_LOCATION/lib/mads/*.class` : Driver libraries

- `$ONE_LOCATION/bin/one_im_vmware` : Wrapper for VMware Information Manager driver

- `$ONE_LOCATION/etc/im_vmware/im_vmwarerc` : environment setup. Also useful for setting whether if the driver should be verbose in the log entries.

### 2.3.5 Configuration

#### OpenNebula Configuration

OpenNebula needs to be told how to run the drivers. The place to do so is the `$ONE_LOCATION/etc/oned.conf`, it needs to have the VMware transfer, information and virtualization drivers set like the following lines:

```
#-------------------------------------------------------------------------------
#  VMWare Information Driver Manager sample configuration
#-------------------------------------------------------------------------------
  IM_MAD = [
      name       = "im_vmware",
      executable = "one_im_vmware",
      arguments  = "--username <esxi_username> --password <esxi_password>"]
#-------------------------------------------------------------------------------
```

[4]Software Development Kit

```
#-------------------------------------------------------------------------------
#  VMWare Virtualization Driver Manager sample configuration
#-------------------------------------------------------------------------------
   VM_MAD = [
      name       = "vmm_vmware",
      executable = "one_vmm_vmware",
      arguments  = "--username <esxi_username> --password <esxi_password>",
      type       = "xml" ]
#-------------------------------------------------------------------------------
```

```
#-------------------------------------------------------------------------------
#  VMWare Transfer Driver Manager sample configuration
#-------------------------------------------------------------------------------
TM_MAD = [
    name       = "tm_vmware",
    executable = "one_tm",
    arguments  = "tm_vmware/tm_vmware.conf" ]
#-------------------------------------------------------------------------------
```

### Driver Configuration

Drivers rely on environmental variables to gather necessary information to access and manage the VMware hypervisors. This can be set in the shell session scope (i.e. in the .bashrc of oneadmin) or in the rc files of each driver. The needed variables are:

- `VMWARE_TRUSTORE` : Must point to the vmware.keystore file. (Needed by the information and virtualization drivers)
- `VMWARE_DATASTORE`: Name of the datastore shared by all the hypervisors. (Needed by the virtualization driver)
- `VMWARE_DATACENTER`: Name of the datacenter. This name has to be shared between all the hypervisor. By defaut it is `ha-datacenter`. (Needed by the virtualization driver)
- `DATASTORE_PATH`: Path to the exported `NFS` share that the hypervisors mount as their dastore. This is the local path in the OpenNebula front-end. (Needed by the transfer driver).

A sample rc configuration file for the virtualization driver (`$ONE_LOCATION/etc/vmm_vmware/vmm_vmwarerc`) follows:

```
# This must point to a java keystore with appropriate certificates for accessing
# all the ESXi hosts
VMWARE_TRUSTORE=~/.vmware-certs/vmware.keystore

# Uncomment the following line to active MAD debug
# ONE_MAD_DEBUG=1

# Datastore name
VMWARE_DATASTORE=datastore1

# Datacenter name
VMWARE_DATACENTER=ha-datacenter
```

Also, a Virtual Machine port group defined within a Virtual Switch attached to a physical device needs to be configured using the VIC (`Configuration` tab, `Networking` link in the left panel, `Add Networking` in the top right). Its name has to match the `BRIDGE` name of the OpenNebula Virtual Network defined to provide configuration to the VMs. More information about Virtual Networks in OpenNebula here

### 2.3.6 VMware Template Attributes

Relevant attributes in the Virtual Machine template for VMware are:

- **Name**: Name of the Virtual Machine.
- **Memory**: Expressed in Mb.
- **CPU**: Number of virtual CPUs that this VM will have assigned.
- **NIC**: Which Virtual Network to connect to, or direct set of the MAC address. All VM Virtual Ethernet Cards (defined when the VM was created using the VIC) will be erased, and one or more Virtual Ethernet Cards will be added (one per **NIC** section). If no `NIC` section is present, previously defined Virtual Ethernet Cards won't be erased.
- **DISKS**: Only one disk is needed (and compulsory). Its source MUST point to the folder containing the VM in the datastore. It has to be a local path in the OpenNebula front-end. Additionally, the CLONE and SAVE flags will have effect *on the whole of the Virtual Machine*.

Example:

```
NAME=VMwareVM
MEMORY=256
CPU=1

NIC=[NETWORK="VMWareNet"]

DISK=[ source="/images/vmware/myVM",
       clone="yes",
       save="no"]
```

### 2.3.7 Using the VMware Driver

In order to use a Virtual Machine with VMware hypervisors within OpenNebula it first needs to be created with the VIC in the shared datastore. To avoid security risks and enable Cloning and Saving capabilities, we recommend changing ownership of every file in the VM folder to the <oneadmin> user:

To avoid questions asked by the VMware hypervisors about the VM's uuid when they are moved around, the `vmx` file can be edited and the following added to it:

Also, to allow for any MAC address to be set for the VMs, the `vmx` has to include:

Alternatively to the consolidated shared filesystem approach, VMs can be stagged to VMware hypervisors using ssh. See here for details on the Vmware ssh transfer driver.

# Chapter 3

# Customizing your Private Cloud

## 3.1 Storage and Image Management

One key aspect of virtualization management is the process of dealing with Virtual Machines images. Allegedly, there are a number of possibly different configurations depending on the user needs. For example, the user may want all her images placed on a separate repository with only http access. Or images can be shared through NFS between all the hosts. OpenNebula aims to be flexible enough to support as many different image storage configurations as possible.

The image storage model upon which OpenNebula can organize the images uses the following concepts:

- **Image Repositories**, refer to any storage medium, local or remote, that hold the base images of the VMs. An image repository can be a dedicated file server or a remote URL[1] from an appliance provider, but they need to be accessible from the OpenNebula front-end.

- **Virtual Machine Directory**, is a directory on the cluster node where a VM is running and is of the form `<VM_DIR>/<VID>`, `<VM_DIR>` path is defined in the oned.conf file for the cluster. Deployment files for the hypervisor to boot the machine, checkpoints and images being used or saved, all of them specific to that VM will be placed into this directory. Note that the `<VM_DIR>` should be shared for most hypervisors to be able to perform live migrations.

### 3.1.1 The Image Life-Cycle

Any given VM image to be used crosses through the next steps:

- **Preparation**, implies all the necessary changes to be made to the machine's image so it is prepared to offer the service it is intended to. OpenNebula assumes that the image(s) that conform a particular VM are prepared and placed in the accessible image repository.

- **Cloning** the image means taking the image from the repository and placing it in the VM's directory **before** the VM is actually booted. If a VM image is to be cloned, means that the original image is not going to be used, and thus a copy will. There is a qualifier (`clone`) for the images that can mark them as to be cloned or not.

- **Save / Remove**, once the VM has been shutdown the images, and all the changes thereof, are to be disposed of. However, if the `save` qualifier is activated the image will be saved for later use under `$ONE_LOCATION/var/<VID>/images`.

**Note:** If OpenNebula was installed in **system wide** mode this directory becomes `/var/lib/one/images`. The rest of this guide refers to the `$ONE_LOCATION` paths (corresponding to **self contained** mode) and omits the equivalent **system wide** locations. More information on installation modes can be found here

[1] Uniform Resource Locator

### 3.1.2 Physical Cluster Configuration

The storage model assumed by OpenNebula does not require any special software to be installed. The following are two cluster configuration examples supported by OpenNebula out-of-the-box. They represent the choices of either sharing the <`VM_DIR`> among all the cluster nodes and the cluster front-end via `NFS`, or not sharing any folder and have the machines accessible using `SSH`[2]. Please note that the Transfer Manager was built using a modular architecture, where each action is associated with a small script that can be easily tuned to fit your cluster configuration. A third choice (share the image repository and not the <`VM_DIR`>) is explained in the Customizing \& Extending section.

#### Shared - NFS

This arrangement of the Storage Model assumes that the <`VM_DIR`> is shared between all the cluster nodes and the OpenNebula server. In this case, the semantics of the clone and save actions described above are:

- `Cloning`: If an image is clonable, it will be copied from the image repository to <`VM_DIR`>`/`<`VID`>`/images`, form where it will be used for the VM. If not, a symbolic link will be created from the image repository also to <`VM_DIR`>`/`<`VID`>`/images`, so effectively the VM is going to use the original image.

* `Saving`: This will have only effect if the image is not clonable, if it is then saving comes for free. Therefore, if the image is not clonable and savable, the image will be moved from <`VM_RDIR`>`/`<`VID`>`/images` to `$ONE_LOCATION/var/`<`VID`>`/images`.

**Please note that by default** <`VM_DIR`> is set to `$ONE_LOCATION/var`.

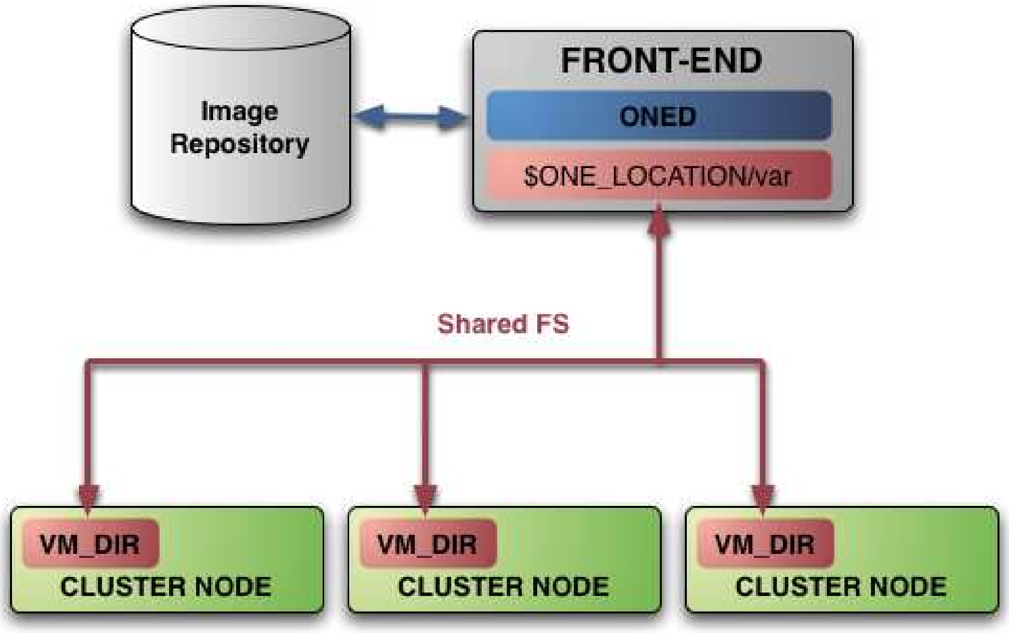


Figure 3.1: Storage Model : NFS

#### Non-Shared - SSH

In this scenario, the <`VM_DIR`> is not shared between the cluster front-end and the nodes. **Note**, that <`VM_DIR`> can be shared between the cluster nodes to perform live migrations. The semantics of clone and save are:

[2]Secure Shell

- `Cloning`: This attribute is ignored in this configuration, since images will always be cloned from the image repository to the `<VM_DIR>/<VID>/images`.

* `Saving`: If enabled, the image will be transferred back from `<VM_DIR>/<VID>/images` to `$ONE_LOCATION/var/<VID`
If not enabled, the image will be simply erased. It is therefore the users responsability to reuse the image from `$ONE_LOCATION/var/<VID>/images/` in subsequent uses of the VM in order to use any configuration done or data stored in it.

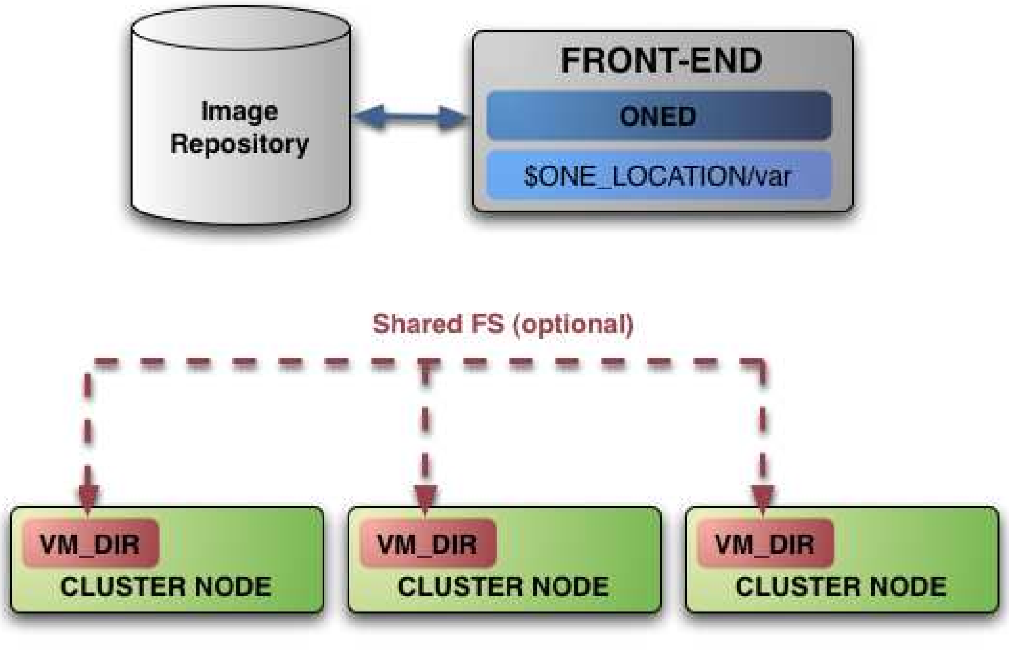


Figure 3.2: Storage Model : SSH

### LVM

There are many possible scenarios in which we can take advantage of OpenNebula's support of block devices, especially if we use LVM. The most powerful advantage of using LVM is snapshotting, which results in an immediate creation of disk devices.

OpenNebula ships with a set of Transfer Manager scripts which support LVM. The idea behind these scripts is not to provide a full-blown LVM solution, but a basic example which can be tailored to fit a more specific scenario.

The Transfer Manager makes the assumption that the block devices defined in the VM template are available in all the nodes i.e. if we have

```
source   = "/dev/default/ttylinux",
```

then /dev/default/ttylinux must exist in the node where the VM will be deployed. This can be achieved either by creating the device by hand or by using more complicated techniques like exporting a LVM to a cluster.

- `Cloning`: A new snapshot will be created and assigned to the VM. This process is almost instantaneous.
- `Saving`: Saving disk images is supported by dumping the device to a file and scp'ing the disk image back to the frontend.
- `Stop/Migration`: These features have not been implemented for this Transfer Manager, since they depend strongly on the scenario.

### 3.1.3 Configuration Interface

The Transfer Manager is configured in the `$ONE_LOCATION/etc/oned.conf` file, see the Daemon Configuration file. Being flexible, the TM is always the same program, and different configurations are achieved by changing the configuration file. This file regulates the assignment between actions, like `CLONE` or `LN`, and scripts, effectively changing the semantics of the actions understood by the TM.

```
TM_MAD = [
    name       = "tm_nfs",
    executable = "one_tm",
    arguments  = "<tm-configuration-file>",
    default    = "<default-tm-configuration-file" ]
```

Current OpenNebula release contains two set of scripts for the two scenarios described above, `Shared - NFS` (`$ONE_LOCATION/etc/tm_nfs/tm_nfs.conf`) or the `Non Shared SSH`[2] TM (`$ONE_LOCATION/etc/tm_ssh/tm_ssh.co`
Each different TM will have their own directory inside `$ONE_LOCATION/etc`.

Lets see a sample line from the `Shared - NFS` configuration file:

```
...
CLONE   = nfs/tm_clone.sh
...
```

Basically, the TM here is being told that whenever it receives a clone action it should call the `tm_clone.sh` script with the received parameters. For more information on modifying and extending these scripts see Customizing and Extending.

**Note**: Remember that if OpenNebula is installed in root, the configuration files are placed in `/etc/one`.

#### Example Shared - NFS

To configure OpenNebula to be able to handle images with this arrangement of the storage model, add the following in `$ONE_LOCATION/etc/oned.conf`, so the TM knows what set of scripts it needs to use:

```
TM_MAD = [
    name       = "tm_nfs",
    executable = "one_tm",
    arguments  = "tm_nfs/tm_nfs.conf",
    default    = "tm_nfs/tm_nfs.conf" ]
```

#### Example Non-shared - SSH

To configure OpenNebula to be able to handle images with non-shared arrangement of the storage model, add the following in `$ONE_LOCATION/etc/oned.conf`, so the TM knows what set of scripts it needs to use:

```
TM_MAD = [
    name       = "tm_nfs",
    executable = "one_tm",
    arguments  = "tm_ssh/tm_ssh.conf",
    default    = "tm_ssh/tm_ssh.conf" ]
```

#### Example LVM

To configure OpenNebula to be able to handle images with the LVM storage model, add the following in `$ONE_LOCATION/etc/oned.conf`, so the TM knows what set of scripts it needs to use:

```
TM_MAD = [
    name       = "tm_lvm",
    executable = "one_tm",
    arguments  = "tm_lvm/tm_lvm.conf"]
```

### 3.1.4 Customizing and Extending

**Structure: The Action Scripts**

Transfer Manager (TM) architecture is highly modular. There are high level actions that OpenNebula asks the TM to perform, which in turn the TM translates into different tasks depending on the storage model specific configuration. Current release of OpenNebula comes with two different set of action scripts that conforms two different TMs for different storage models (see Configuration Interface). There is information available on how to build custom sets of action scripts to tackle custom configuration of the storage model.

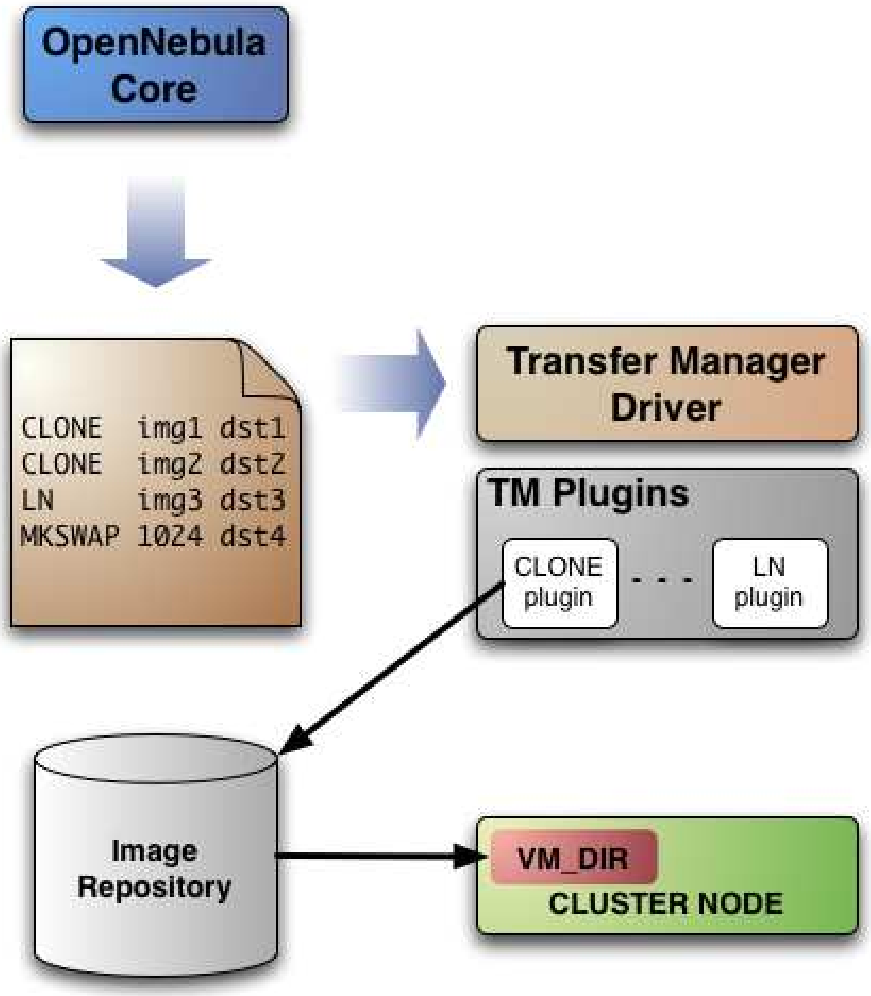


Figure 3.3: TM Structure

TM is all about moving images around and performing file system operations on (possibly) remote hosts, so all the actions receive as arguments at least one ORIGIN or one DESTINATION, or both. These have the form:

```
<host>:<path>
```

The different available actions that the TM understands are:

- **CLONE**: This action will basically make a copy of the image from `ORIGIN` to `DESTINATION`. This can mean different things depending on the storage model configuration (see Sample Configurations)
- **LN**: Creates a symbolic link in `DESTINATION` that points to `ORIGIN`
- **MKSWAP**: Generates a swap image in `DESTINATION`. The size is given in `ORIGIN` in MB[3].
- **MKIMAGE**: Creates a disk image in `DESTINATION` and populates it with the files inside `ORIGIN` directory.
- **DELETE**: Deletes `ORIGIN` file or directory.
- **MV**: Moves `ORIGIN` to `DESTINATION`.

Action scripts must conform to some rules to work. These are the general rules:

- The script should exit with code 0 on success or any other value to indicate failure.
- Everything written to STDOUT is logged to the corresponding VM log file.
- In case of failure the error message is written to STDERR surrounding the message with these separators:

```
ERROR MESSAGE --8<------
error message here
ERROR MESSAGE ------>8--
```

There is a helper shell script with some functions defined to do some common tasks. It is located in `$ONE_LOCATION/lib/mads/tm_common.sh` and can be loaded adding this line at the start of your script:

```
. $ONE_LOCATION/lib/mads/tm_common.sh
```

Here are the description of those functions.

- **log**: Takes one parameter that is a message that will be logged into the VM log file.

```
log "Creating directory $DST_DIR"
```

- **error_message**: sends an exit message to oned surrounding it by separators, use to send the error message when a command fails.

```
error_message "File '$FILE' not found"
```

- **arg_host**: gets the hostname part from a parameter

```
SRC_HOST=`arg_host $SRC`
```

- **arg_path**: gets the path part from a parameter

```
SRC_PATH=`arg_path $SRC`
```

- **exec_and_log**: executes a command and logs its execution. If the command fails the error message is sent to oned and the script ends

```
exec_and_log "chmod g+w $DST_PATH"
```

- **timeout_exec_and_log**: like `exec_and_log` but takes as first parameter the max number of seconds the command can run

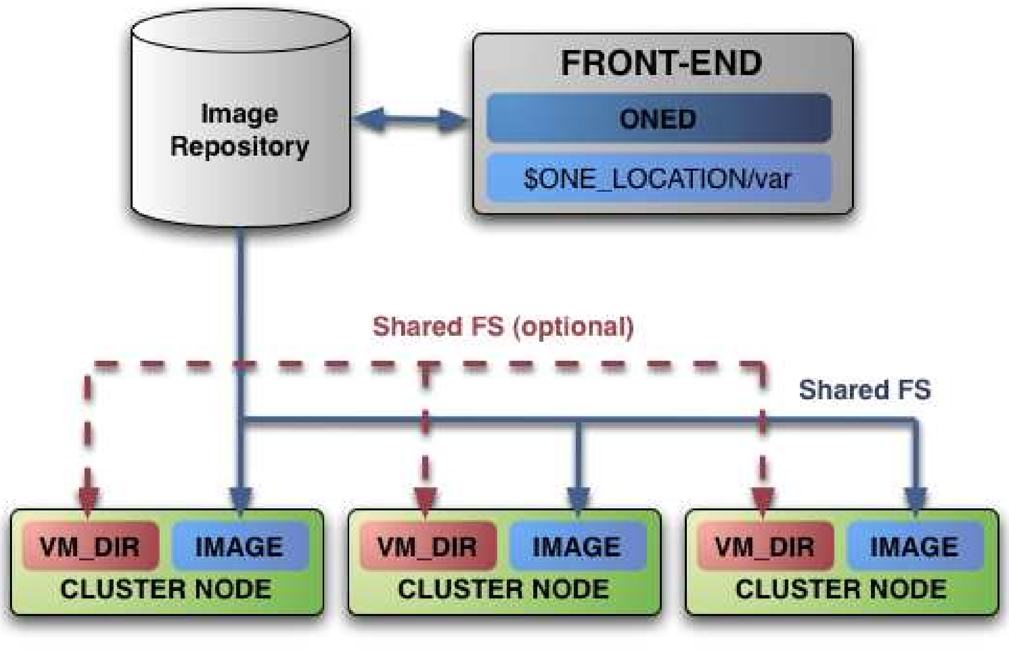
```
timeout_exec_and_log 15 "cp $SRC_PATH $DST_PATH"
```

**Note**: This script is placed in `/usr/lib/one/mads` if you installed OpenNebula in root.

[3] Megabyte

### Sample: A Shared Image Repository

As an example here is described how to modify NFS scripts for a different configuration. In this configuration we will have images directory shared but not <VM_DIR>.



**tm_ln.sh** This script is responsible of creating a link in <VM_DIR> that points to the original image when it has clone set to "off". As we are dealing with non shared <VM_DIR> it has to be modified so it creates the remote directory and also makes the link in the destination host.

```
#!/bin/bash

SRC=$1
DST=$2

. $ONE_LOCATION/lib/mads/tm_common.sh

SRC_PATH=`arg_path $SRC`
DST_PATH=`arg_path $DST`
DST_HOST=`arg_host $DST`

DST_DIR=`dirname $DST_PATH`

log "Creating directory $DST_DIR"
exec_and_log "ssh $DST_HOST mkdir -p $DST_DIR"

log "Link $SRC_PATH"
exec_and_log "ssh $DST_HOST ln -s $SRC_PATH $DST_PATH"
```

We added the `mkdir` command and changed link creation to be executed in the remote machine.

**tm_clone.sh** Here the modifications are similar to LN, changed the commands so they are executed in the destination host.

```
#!/bin/bash

SRC=$1
DST=$2

. $ONE_LOCATION/lib/mads/tm_common.sh

SRC_PATH=`arg_path $SRC`
DST_PATH=`arg_path $DST`
DST_HOST=`arg_host $DST`

```

```
log "$1 $2"
log "DST: $DST_PATH"

DST_DIR=`dirname $DST_PATH`

log "Creating directory $DST_DIR"
exec_and_log "ssh $DST_HOST mkdir -p $DST_DIR"
exec_and_log "ssh $DST_HOST chmod a+w $DST_DIR"

case $SRC in
http://*)
    log "Downloading $SRC"
    exec_and_log "ssh $DST_HOST wget -O $DST_PATH $SRC"
    ;;

*)
    log "Cloning $SRC_PATH"
    exec_and_log "ssh $DST_HOST cp $SRC_PATH $DST_PATH"
    ;;
esac

exec_and_log "ssh $DST_HOST chmod a+w $DST_PATH"
```

Note the ssh command appended to each of commands.

The rest of the the commands follow similar modifications, executing the commands using ssh on the remote machine.

#### Adding Custom Remote URLs

When you specify a disk that has ":" in its path is not treated in any way and it is passed to TM action script directly. This lets us pass things like `<hostname>:<path>` so you can specify an image that is located in a remote machine that is accessible using ssh or customize clone scripts so it accepts different kinds of URLS. This modification should be done in `tm_clone.sh` script adding a new option in the case statement. For example, if we want to add ftp support using wget we can add this code:

```
case $SRC in
ftp://*)
    log "Downloading $SRC"
    exec_and_log "wget -O $DST_PATH $SRC"
    ;;
```

Note that we are using `-O` option of `wget`, that tells where to download the file. This is needed as the file should be in an specific directory and also needs to be correctly named.

## 3.2 Networking

### 3.2.1 Overview

The physical hosts that will conform the fabric of our virtual infrastructures will need to have some constraints in order to be able to deliver virtual networks effectively to our virtual machines. Therefore, we can define our physical cluster under the point of view of networking as a set of hosts with one or more network interfaces, each of them connected to a different physical network.

### 3.2.2 Isolating the Virtual Networks

#### Description

In the proposed architecture several virtual networks will share the same physical network. Doing this makes this configuration very flexible as you do not need to setup new physical networks or configure vLANs (IEEE 802.1Q) in the switch each time you need a new virtual network.

However sharing the same physical network also makes the virtual networks potentially vulnerable to a compromised VM. Here we will explain a method of isolating networks at the ethernet level so the traffic generated in different virtual networks can not be seen in others.

OpenNebula provides you with the infrastructure to dynamically create new LANs for you virtualized services as you need them without the need of reconfigure your switches or any other special network component.

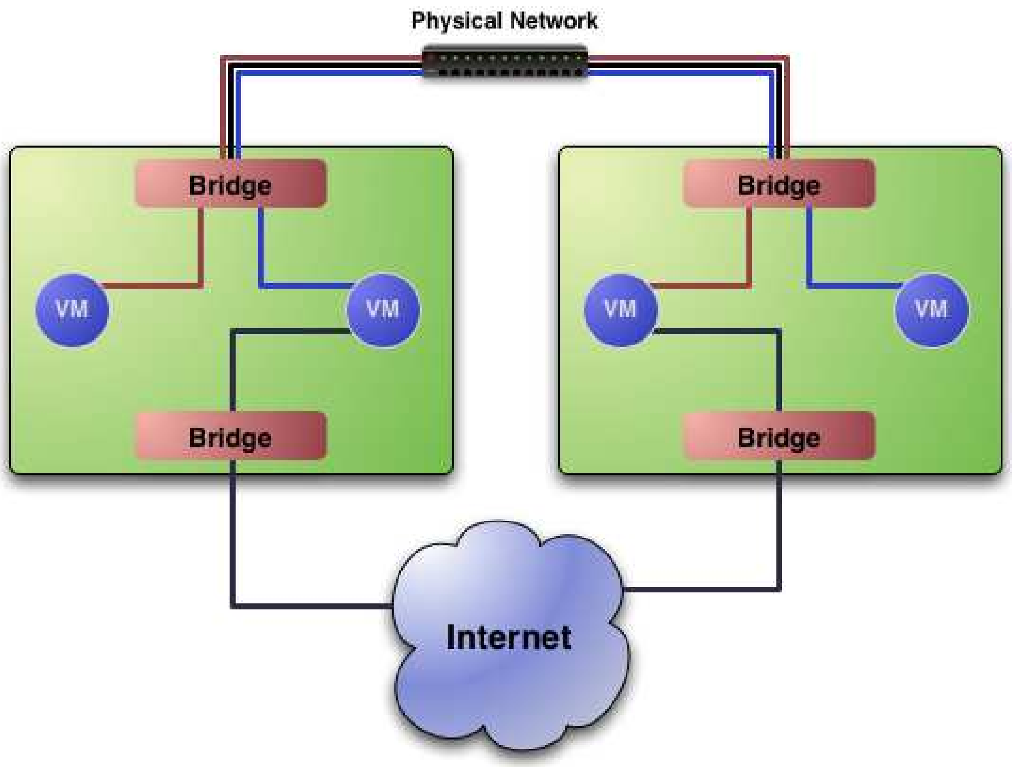


Figure 3.4:

Network isolation allows you to put any network service within your VMs. For example you can put a VM with a DHCP[4] server in each virtual network to handle the VM's IP in that virtual network.

### Configuration

The virtual networks are isolated by filtering the ethernet traffic in the device where the VM is attached to. So we let the VM to send packets only to machines in the same network (those with mac addresses in the virtual network, as defined with `onevnet`. This is achieved using `ebtables` that lets you change bridge tables and the OpenNebula Hooks.

All the network configuration will be done in the cluster nodes, these are the additional requisites:

- `ebtables` package installed
- `sudoers` configured so `oneadmin` can execute `ebtables`

**Sudoers Configuragtion** Add this line to sudoers on each cluster node:

```
oneadmin    ALL=(ALL) NOPASSWD: /sbin/ebtables *
```

If you happen to have `ebtables` binary installed in other path change the line for sudoers file accordingly.

**OpenNebula Configuration** You have to configure OpenNebula so the script is called each time a new VM is created and also for VM shutdown (to delete ebtables rules). This is the configuration needed in `oned.conf`:

[4]Dynamic Host Configuration Protocol

```
VM_HOOK = [
    name      = "ebtables-start",
    on        = "running",
    command   = "/srv/cloud/one/share/hooks/ebtables-kvm", # or ebtables-xen
    arguments = "one-$VMID",
    remote    = "yes" ]

VM_HOOK = [
    name      = "ebtables-flush",
    on        = "done",
    command   = "/srv/cloud/one/share/hooks/ebtables-flush",
    arguments = "",
    remote    = "yes" ]
```

The first script takes one parameter: the VM name. The second script removes all the ebtables rules which refer to a nonexistent VM, therefore no argument is needed. These hooks are executed when the VM enters the running (`on = "running"`), shutdown or stop states. Note also that they are executed in the cluster nodes (`remote = yes`). Check the oned configuration reference for more information.

Here is the script that configures ebtables for kvm machines. A version for Xen is located in `$ONE_LOCATION/share/hooks`.

```ruby
#!/usr/bin/env ruby

require 'pp'
require 'rexml/document'

VM_NAME=ARGV[0]

# Uncomment to act only on the listed bridges.
#FILTERED_BRIDGES = ['beth0']

def activate(rule)
    system "sudo ebtables -A #{rule}"
end

def get_bridges
    bridges = Hash.new
    brctl_exit=`brctl show`
    cur_bridge = ""
    brctl_exit.split("\n")[1..-1].each do |l|
        l = l.split
        if l.length > 1
            cur_bridge = l[0]
            bridges[cur_bridge] = Array.new
            bridges[cur_bridge] << l[3]
        else
            bridges[cur_bridge] << l[0]
        end
    end
    bridges
end

def get_interfaces
    bridges = get_bridges
    if defined? FILTERED_BRIDGES
        FILTERED_BRIDGES.collect {|k,v| bridges[k]}.flatten
    else
        bridges.values.flatten
    end
end

nets=`virsh -c qemu:///system dumpxml #{VM_NAME}`

doc=REXML::Document.new(nets).root

interfaces = get_interfaces()

doc.elements.each('/domain/devices/interface') {|net|
    tap=net.elements['target'].attributes['dev']
    if interfaces.include? tap
        iface_mac=net.elements['mac'].attributes['address']

        mac=iface_mac.split(':')
        mac[-1]='00'
```

```
            net_mac=mac.join(':')


            in_rule="FORWARD -s ! #{net_mac}/ff:ff:ff:ff:ff:00 -o #{tap} -j DROP"
            out_rule="FORWARD -s ! #{iface_mac} -i #{tap} -j DROP"

            activate(in_rule)
            activate(out_rule)
        end
}
```

## 3.3 Information and Monitoring

The Information Manager (IM) is in charge of monitoring the cluster nodes. It comes with various sensors, each one responsible of a different aspects of the computer to be monitored (CPU, memory, hostname...). Also, there are sensors prepared to gather information from different hypervisors (currently KVM and XEN).

### 3.3.1 Requirements

Depending on the sensors that are going to conform the IM driver there are different requirements, mainly the availability of the hypervisor corresponding to the sensor if one of the KVM sensor or XEN sensor are used at all. Also, as for all the OpenNebula configurations, `SSH`[2] access to the hosts without password has to be possible.

### 3.3.2 Driver Files

This section details the files used by the Information Drivers to monitor the cluster nodes. This files are placed in the following directories:

- `$ONE_LOCATION/lib/mads/`: The drivers executable files
- `$ONE_LOCATION/lib/im_probes/`: The cluster nodes probe to gather each monitoring metric
- `$ONE_LOCATION/etc/`: The drivers configuration files, usually in subdirectories in the form `im_<virtualizer>`.

**Note:** If OpenNebula was installed in **system wide** mode these directories become `/usr/lib/one/mads`, `/usr/lib/one/im_probes` and `/etc/one`, respectively. The rest of this guide refers to the `$ONE_LOCATION` paths (corresponding to **self contained** mode) and omits the equivalent **system wide** locations. More information on installation modes can be found here

#### Common files

These files are used by the IM regardless of the hypervisor present on the machine to be monitored:

- $ONE_LOCATION/lib/mads/one_im_ssh : shell script wrapper to the driver itself. Sets the environment and other bootstrap tasks.
- $ONE_LOCATION/lib/mads/one_im_ssh.rb : The actual Information driver.
- $ONE_LOCATION/lib/im_probes/* : sensors home. Little scripts or binaries that extract information from the remote hosts. Let's see a simple one to understand how they work:

This uses the uname command to get the hostname of the remote cluster host, and then outputs the information as:

```
NAME=host1.mydomain.org
```

#### Hypervisor specific

The following files contain pre-made configuration files to get the IM working with different hypervisors. These files are not fixed, they can be modified, and even the IM can be set to work with different files (this is set in OpenNebula configuration file, more details in next section).

❗ There is always one obligatory attribute set by all the Information Drivers, **HYPERVISOR** set to the kind of hypervisor (XEN, KVM, VMWare, EC2, etc) present on the host this particular Information Driver is monitoring.

#### XEN Hypervisor

- $ONE_LOCATION/etc/im_xen/im_xenrc : environment setup and bootstrap instructions
- $ONE_LOCATION/etc/im_xen/im_xen.conf : This file defines the REMOTE_DIR, the path where the sensors will be uploaded in the remote physical to perform the monitoring. It also defines which sensors will be used (you can add and remove probes as you wish), for XEN the defaults are:

```
cpuarchitecture=architecture.sh
nodename=name.sh
cpu=cpu.sh
xen=xen.rb
```

- $ONE_LOCATION/lib/im_probes/xen.rb : xen specific sensor.

#### KVM Hypervisor

- $ONE_LOCATION/etc/im_kvm/im_kvmrc : environment setup and bootstrap instructions
- $ONE_LOCATION/etc/im_kvm/im_kvm.conf : This file defines the REMOTE_DIR, the path where the sensors will be uploaded in the remote physical to perform the monitoring. It also defines which sensors will be used (you can add and remove probes as you wish), for KVM the defaults are:

```
cpuarchitecture=architecture.sh
nodename=name.sh
cpu=cpu.sh
kvm=kvm.rb
```

- $ONE_LOCATION/lib/im_probes/kvm.rb : kvm specific sensor.

### 3.3.3 OpenNebula Configuration

The OpenNebula daemon loads its drivers whenever it starts. Inside `$ONE_LOCATION/etc/oned.conf` there are definitions for the drivers. The following lines, will configure OpenNebula to use the Xen probes:

```
IM_MAD = [
    name       = "im_xen",
    executable = "bin/one_im_ssh",
    arguments  = "im_xen/im_xen.conf",
    default    = "im_xen/im_xen.conf" ]
```

Equivalently for KVM, you'd put the following in `oned.conf`:

```
IM_MAD = [
    name       = "im_kvm",
    executable = "one_im_ssh",
    arguments  = "im_kvm/im_kvm.conf",
    default    = "im_kvm/im_kvm.conf" ]
```

And finally for EC2:

```
IM_MAD = [
    name       = "im_ec2",
    executable = "one_im_ec2",
    arguments  = "im_ec2/im_ec2.conf",
    default    = "im_ec2/im_ec2.conf" ]
```

Please remember that you can add your custom probes for later use by other OpenNebula modules like the scheduler.

### 3.3.4 Testing

In order to test the driver, add a host to OpenNebula using **onehost**, specifying the defined IM driver:

Now give it time to monitor the host (this time is determined by the value of HOST_MONITORING_INTERVAL in `$ONE_LOCATION/etc/oned.conf`). After one interval, check the output of **onehost list**, it should look like the following:

## 3.4 Scheduling Policies

### 3.4.1 Overview

The Scheduler module is in charge of the assignment between pending Virtual Machines and known Hosts. OpenNebula's architecture defines this module as a separate process that can be started independently of `oned`. The OpenNebula scheduling framework is designed in a generic way, so it is highly modifiable and can be easily replace by third-party developments.

### 3.4.2 The Match-making Algorithm

OpenNebula comes with a `match making` scheduler (*mm_sched*) that implements the *Rank Scheduling Policy*. The goal of this policy is to prioritize those resources more suitable for the VM.

The algorithm works as follows:

- First those hosts that do not meet the VM requirements (see the ”REQUIREMENTS” attribute) and do not have enough resources (available CPU and memory) to run the VM are filtered out.
- The ”RANK” expression is evaluated upon this list using the information gathered by the monitor drivers. Any variable reported by the monitor driver can be included in the rank expression.
- Those resources with a higher rank are used first to allocate VMs.

### 3.4.3 Placement Policies

You can implement several placement heuristics by carefully choosing the `RANK` expression. Note that each VM has its own `RANK` and so its own policy. This is specially relevant when configuring a Cloud Interface as you can apply different policies to different instance types.

**Packing Policy**

- **Target**: Minimize the number of cluster nodes in use
- **Heuristic**: Pack the VMs in the cluster nodes to reduce VM fragmentation
- **Implementation**: Use those nodes with more VMs running first

```
RANK = RUNNING_VMS
```

**Striping Policy**

- **Target**: Maximize the resources available to VMs in a node
- **Heuristic**: Spread the VMs in the cluster nodes
- **Implementation**: Use those nodes with less VMs running first

```
RANK = - RUNNING_VMS
```

**Load-aware Policy**

- **Target**: Maximize the resources available to VMs in a node
- **Heuristic**: Use those nodes with less load
- **Implementation**: Use those nodes with more FREECPU first

```
RANK = FREECPU
```

### 3.4.4 The Haizea Scheduler

The  Haizea lease manager can also be used as a scheduling module in OpenNebula. Haizea allows OpenNebula to support advance reservation of resources and queuing of best effort requests (more generally, it allows you to lease your resources as VMs, with a variety of lease terms). The Haizea documentation includes a guide on  how to use OpenNebula and Haizea to manage VMs on a cluster

C12G LABS

# Chapter 4

# Using your Private Cloud

## 4.1 Managing Virtual Machines

OpenNebula is a VM manager that is executed and configured by a cluster administrator. This cluster administrator would be also the ONE administrator, and therefore he should be the holder of the <`oneadmin`> account. This guide assumes that the cluster administrator is the only user for `oned`, although the system has been designed to support several users.

OpenNebula is able to assign, deploy, monitor and control VMs. This guide explains **how to describe the wanted-to-be-ran Virtual Machine, and how users typically interact with the system**.

### 4.1.1 Virtual Machine Model

A Virtual Machine within the OpenNebula system consists of:

- A capacity in terms memory and CPU
- A set of NICs attached to one or more virtual networks
- A set of **disk images**. In general it could be necessary to transfer some of these image files to/from the execution host.
- A state file (optional) or **recovery file**, that contains the memory image of a running VM plus some hypervisor specific information.

The above items, plus some additional VM attributes like the OS[1] kernel and context information to be used inside the VM, are specified in a **VM template** file.

Each VM in OpenNebula is identified by an unique number, the <`VID`>. Also, the user can assign it a name in the VM template, the default name for each VM is `one-`<`VID`>.

#### Virtual Machine Life-cycle

The life-cycle of a virtual machine within the system includes the following stages (note that this is a simplified version):

- **Pending** (`pend`): By default a VM starts in the pending state, waiting for a resource to run on.
- **Hold** (`hold`): The owner has held the VM and it will not be scheduled until it is released.
- **Prolog** (`prol`): The system is transferring the VM files (disk images and the recovery file).
- **Running** (`runn`): The VM is running (note that this stage includes booting and shutting down phases). In this state, the virtualization driver will periodically monitor it.

[1]Operating System

- **Migrate** (`migr`): The VM is migrating from one resource to another. This can be a life migration or cold migration (the VM is saved and VM files are transferred to the new resource).
- **Epilog** (`epil`): In this phase the system clean ups the cluster node used to host the VM, and additionally disk images to be saved are copied back to the cluster front-end.
- **Stopped** (`stop`): The VM is stopped. VM state has been saved and it has been transferred back along with the disk images.
- **Suspended** (`susp`): Same as stopped, but the files are left in the remote resource to later restart the VM there (i.e. there is no need to re-schedule the VM).
- **Failed** (`fail`): The VM failed.
- **Unknown** (`unknown`): The VM couldn't be reached, it is in an unknown state.
- **Done** (`done`): The VM is done. VMs in this state wont be shown with "onevm list" but are kept in the database for accounting purposes.

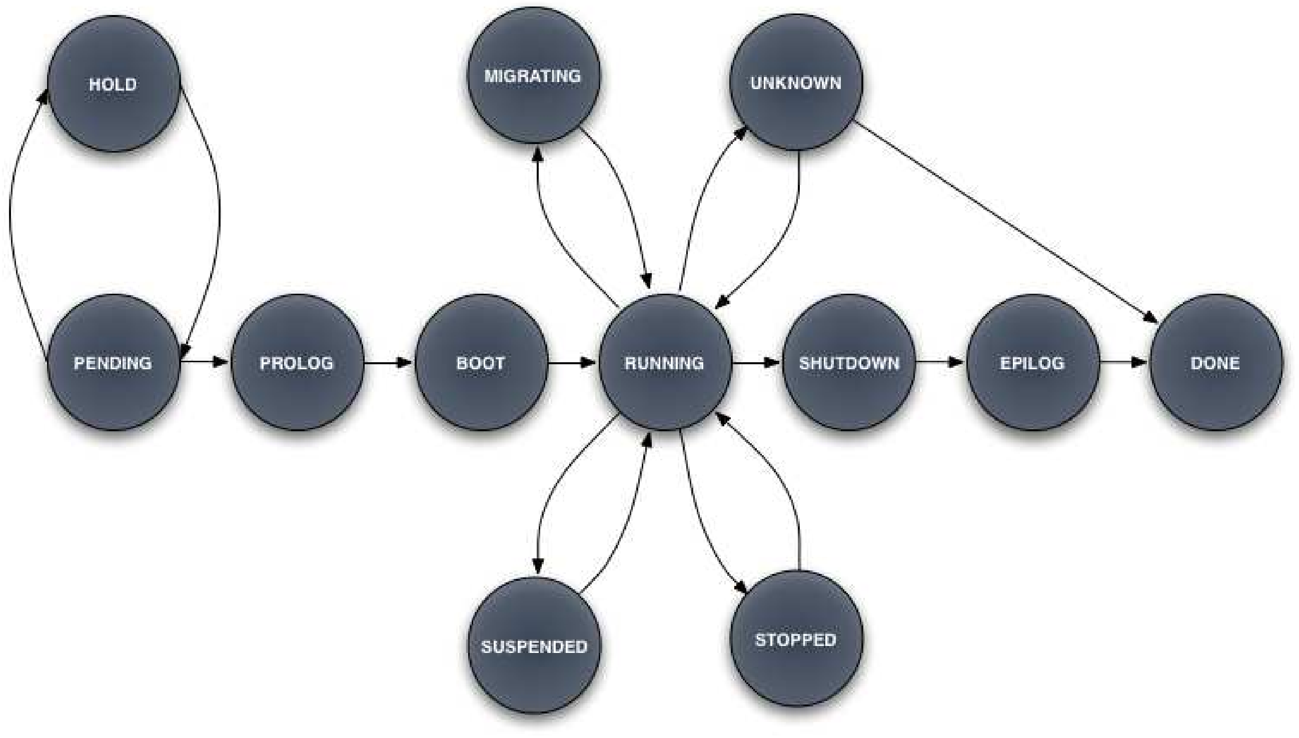


Figure 4.1: Virtual Machine States

### 4.1.2 User Environment

In order to use OpenNebula, you need to set the following variables in your environment. You may want to place them, in the *.bashrc* of the <`oneadmin`> account for commodity:

| | |
|---|---|
| **ONE_LOCATION** | If OpenNebula was installed in **self-contained** mode, this variable must be set to <destination_folder>. Otherwise, in **system wide** mode, this variable must be unset. More info on installation modes can be found here |
| **ONE_XMLRPC** | http://localhost:2633/RPC2 |
| **PATH** | $ONE_LOCATION/bin:$PATH if **self-contained**. Otherwise this is not needed. |
| **ONE_AUTH** | Needs to point to **a file containing just a single line stating "username:password"**. If ONE_AUTH is not defined, $HOME/.one/one_auth will be used instead. If no auth file is present, OpenNebula cannot work properly, as this is needed by the core, the CLI, and the cloud components as well. |

### 4.1.3 Virtual Machine Template

OpenNebula templates are designed to be hypervisor-agnostic, but still there are some peculiarities to be taken into account, and mandatory attributes change depending on the target hypervisor. Hypervisor specific information for this attributes can be found in the drivers configuration guides:

- Xen Adaptor
- KVM Adaptor
- VMware Adaptor

OpenNebula has been designed to be easily extended, so any attribute name can be defined for later use in any OpenNebula module. There are some pre-defined attributes, though.

Please check the Virtual Machine definition file reference for details on all the sections.

#### A VM Template Example

For example, the following template defines a VM with 512MB of memory and one CPU. The VM has two disks: sda supported by an image file; and sdb (a swap partition). Only one NIC is defined with a predefined MAC address. In this case, between all the suitable hosts (those that meets CPU and MEMORY requirements, and also CPUSPEED > 1000 requirement), OpenNebula will pick the host with more free CPU.

```
#---------------------------------------
# VM definition example
#---------------------------------------

NAME = vm-example

CPU    = 1
MEMORY = 512

# --- kernel & boot device ---

OS = [
  kernel   = "/vmlinuz",
  initrd   = "/initrd.img",
  root     = "sda" ]

# --- 2 disks ---
```

```
DISK = [
  source   = "/local/xen/domains/etch/disk.img",
  target   = "sda",
  readonly = "no" ]

DISK = [
  type     = swap,
  size     = 1024,
  target   = "sdb",
  readonly = "no" ]

# --- 1 NIC ---

NIC = [ mac="00:ff:72:17:20:27"]
```

Note that you can add as many `DISK` and `NIC` attributes as you need

### 4.1.4 Command Line Interface

OpenNebula comes with a rich command line interface intended for users fond of consoles. There are commands to manage three basic aspects of OpenNebula: **virtual machines**, **hosts** and **virtual networks**.

A complete reference for these three commands can be found here. The following subsections show the basics of the available commands with simple usage examples.

#### Managing Virtual Machines

This command enables virtual machine management. Actions offered are:

- `create` ( a VM in OpenNebula's VM pool )
- `deploy` (on a particular cluster node)
- `shutdown`
- `livemigrate` (the virtual machine is transferred between cluster nodes with no noticeable downtime)
- `migration` (machine gets stopped and resume elsewhere)
- `hold`
- `release` (from hold state)
- `stop` (virtual machine state is transferred back to OpenNebula for a possible reschedule)
- `cancel`
- `suspend` (virtual machine state is left in the cluster node for resuming)
- `resume`
- `delete`
- `restart`(resubmits the virtual machine after failure)
- `list` (outputs all the available VMs)
- `show` (outputs information for a specific VM)
- `top` (lists VMs continously)

- `history` (gets VMs history of execution on the cluster nodes)

Assuming we have a VM template called *myVM.one* describing a virtual machine. Then, we can allocate the VM in OpenNebula issuing a:
afterwards, the VM can be listed with the `list` option:
and details about it can be obtained with `show`:

**Exploring the Cluster**

This command enables cluster nodes scouting. Actions offered are:

- `show` (details of a particular cluster node)
- `list` (lists nodes in the host pool)
- `top` (lists host continuously)

Assuming a cluster node called *hosts01* running the `XEN` hypervisor and without shared folders, we can `list` the available nodes:
and retrieve more detailed information for this node using `show`:

**onevnet**

This command enables the management of virtual networks. Actions offered are:

- `create` ( a virtual network in OpenNebula's VN pool )
- `show` (details of a particular virtual network)
- `delete`
- `list` (lists all the VNs in the VN pool)

Assuming a virtual network template called *myVN.net*, we can create the virtual network within OpenNebula issuing:
and then `list` it using:
and obtain details about it with:

### 4.1.5 Usage Scenarios

OpenNebula can be used to deploy different **solutions for on-demand execution of services**. We have prepared the following use cases to illustrate its functionality in typical computing scenarios:

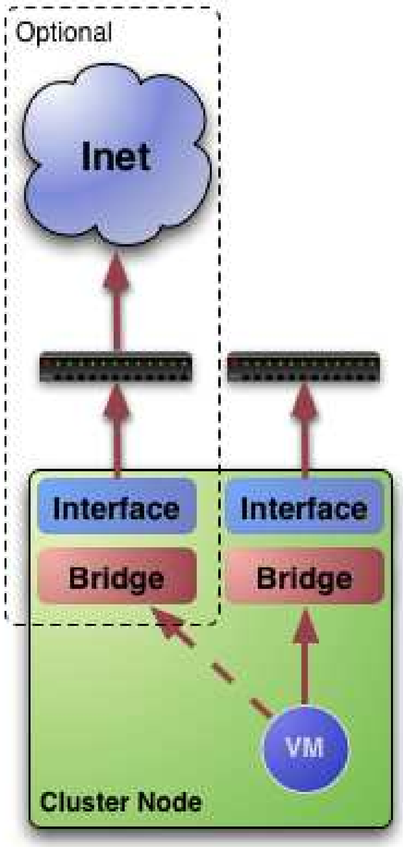
- On-demand Scaling of Computing Clusters
- On-demand Infrastructure for Training
- Scaling out Computing Clusters to Amazon EC2
- Scaling out Web Servers to Amazon EC2

## 4.2 Managing Virtual Networks

A cluster node is connected to one or more networks that are available to the virtual machines through the corresponding bridges. To set up a virtual networks you just need to know the name of the bridge to bind the virtual machines to.

In this guide you'll learn how to define and use virtual networks. For the sake of completeness the following examples assumes that the cluster nodes are attached to two **physical** networks:

- A private network, through the virtual bridge vbr0
- A network with Internet connectivity, through vbr1



### 4.2.1 Defining a Virtual Network

OpenNebula allows for the creation of Virtual Networks by mapping them on top of the physical ones. All Virtual Networks are going to share a default value for the MAC preffix, set in the `oned.conf` file.

There are two types of Virtual Networks in OpenNebula:

- **Fixed**, defines a fixed set of IP-MAC pair addresses
- **Ranged**, defines a class network.

! Virtual Networks created by `oneadmin` can be used by every other user.

#### Fixed Virtual Networks

A fixed network consists of a set of IP addresses and associated MACs, defined in a text file.

We need four pieces of information to define a fixed VN:

- **NAME**: Name of the Virtual Network.
- **TYPE**: Fixed, in this case.
- **BRIDGE**: Name of the physical bridge in the physical host where the VM should connect its network interface.
- **LEASES**: Definition of the IP-MAC pairs. If an IP is defined, and there is no associated MAC, OpenNebula will generate it using the following rule: `MAC = MAC_PREFFIX:IP`. So, for example, from IP 10.0.0.1 and MAC_PREFFIX 00:16, we get 00:16:0a:00:00:01. Defining only a MAC address with no associated IP is not allowed.

For example to create a Fixed Virtual Network, called `Public` with the set of public IPs to be used by the VMs, just create a file with the following contents:

```
NAME = "Public"
TYPE = FIXED

#We have to bind this network to ''virbr1'' for Internet Access
BRIDGE = vbr1

LEASES = [IP=130.10.0.1, MAC=50:20:20:20:20:20]
LEASES = [IP=130.10.0.2, MAC=50:20:20:20:20:21]
LEASES = [IP=130.10.0.3]
LEASES = [IP=130.10.0.4]
```

#### Ranged Virtual Networks

This type of VNs allows for a definition supported by a base network address and a size. So we need to define:

- **NAME**: Name of the Virtual Network.
- **TYPE**: Ranged, in this case.
- **BRIDGE**: Name of the physical bridge.
- **NETWORK_ADDRESS**: Base network address to generate IP addresses.
- **NETWORK_SIZE**: Number of hosts that can be connected using this network. It can be defined either using a number or a network class (B or C).

The following is an example of a Ranged Virtual Network template:

```
NAME = "Red LAN"
TYPE = RANGED

#Now we'll use the cluster private network (physical)
BRIDGE = vbr0

NETWORK_SIZE    = C
NETWORK_ADDRESS = 192.168.0.0
```

Default value for the network size can be found in `oned.conf`.

### 4.2.2 Adding and Deleting Virtual Networks

Once a template for a VN has been defined, the `onevnet` command can be used to create it.

To create the previous networks put their definitions in two different files, `public.net` and `red.net`, respectively. Then, execute:

Also, `onevnet` can be used to query OpenNebula about available VNs:

with `USER` the owner of the network and `#LEASES` the number of IP-MACs assigned to a VM from this network.

To delete a virtual network just use `onevnet delete`. For example to delete the previous networks:

❗ You can also check the IPs leased in a network with the `onevnet show` command

❗ Check the `onevnet` command help or reference guide for more options to list the virtual networks.

### 4.2.3 Getting a Lease

A lease from a virtual network can be obtained by simply specifying the virtual network name in the `NIC` attribute.

For example, to define VM with two network interfaces, one connected to `Red LAN` and other connected to `Public` just include in the template:

```
NIC=[NETWORK="Public"]
NIC=[NETWORK="Red LAN"]
```

You can also request an specific address just by adding the `IP` or `MAC` attributes to `NIC`:

```
NIC=[NETWORK="Red LAN", IP=192.168.0.3]
```

When the VM is submitted, OpenNebula will look for available IPs in the `Public` and `Red LAN` virtual networks. If successful, the `onevm show` command should return information about the machine, including network information.

⚠ Note that if OpenNebula is not able to obtain a lease from a network the submission will fail.

Now we can query OpenNebula with `onevnet show` to find out about given leases and other VN information:

⚠ Note that there is one LEASE active in each network

⚠ IP 192.168.0.1 is in use by Virtual Machine 12

### 4.2.4 Using the Leases within the Virtual Machine

Hypervisors can attach a specific MAC address to a virtual network interface, but Virtual Machines need to obtain an IP address. There are a variety of ways to achieve this within OpenNebula:

- Obtain the IP from the MAC address, using the default MAC assignment schema (**PREFERRED**)
- Use the `CONTEXT` attribute, check the Contextualization Guide

#### Configuring the Virtual Machine to use the Leases

With OpenNebula you can also derive the IP address from the MAC address using the MAC_PREFFIX:IP rule. In order to achieve this we provide a context script for Debian based systems. This script can be easily adapted for other distributions, check dev.opennebula.org.

To configure the Virtual Machine follow these steps:

⚠ These actions are to configure the VM, the commands refer to the VMs root file system

- Copy the script`$ONE_LOCATION/share/scripts/vmcontext.sh` into the `/etc/init.d` directory in the VM root file system.
- Execute the script at boot time before starting any network service, usually runlevel 2 should work.

Having done so, whenever the VN boots it will execute this script, which in turn would scan the available network interfaces, extract their MAC addresses, make the MAC to IP conversion and construct a `/etc/network/interfaces` that will ensure the correct IP assignment to the corresponding interface.

## 4.3 Contextualizing Virtual Machines

### 4.3.1 Overview

The method we provide to give configuration parameters to a newly started virtual machine is using an ISO[2] image (OVF recommendation). This method is network agnostic so it can be used also to configure

[2]International Organization for Standardization

network interfaces. In the VM description file you can specify the contents of the iso file (files and directories), tell the device the ISO[2] image will be accessible and specify the configuration parameters that will be written to a file for later use inside the virtual machine.

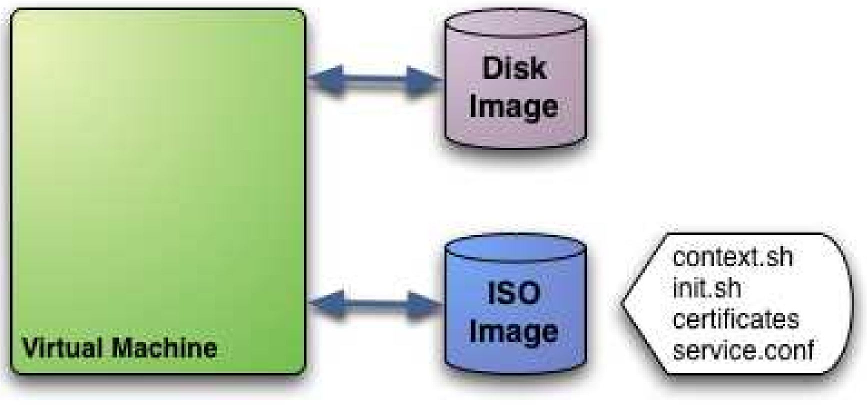


Figure 4.2:

In this example we see a Virtual Machine with two associated disks. The Disk Image holds the filesystem where the Operating System will run from. The ISO[2] image has the contextualization for that VM:

- `context.sh`: file that contains configuration variables, filled by OpenNebula with the parameters specified in the VM description file
- `init.sh`: script called by VM at start that will configure specific services for this VM instance
- `certificates`: directory that contains certificates for some service
- `service.conf`: service configuration

⚠ This is just an example of what a contextualization image may look like. Only `context.sh` is included by default. You have to specify the values that will be written inside `context.sh` and the files that will be included in the image.

### 4.3.2 Defining context

In VM description file you can tell OpenNebula to create a contextualization image and to fill it with values using `CONTEXT` parameter. For example:

```
CONTEXT = [
  hostname   = "$NAME",
  ip_private = "$NIC[IP, NETWORK=\"Private LAN\"]",
  ip_gen     = "10.0.0.$VM_ID",
  files      = "/service/init.sh /service/certificates /service/service.conf",
  target     = "sdc"
]
```

Variables inside CONTEXT section will be added to `context.sh` file inside the contextualization image. The variables starting with $ are substituted by the values that this specific VM instance has (you can check the values with `onevm show`). In this case `$NAME` gets its value from the `NAME` specified in the VM description file. `$NIC[IP, NETWORK="Private LAN"]` will get the IP assigned to the interface that associated to `Private LAN` network.

The file generated will be something like this:

```
# Context variables generated by OpenNebula
hostname="somename"
ip_private="192.168.0.5"
ip_gen="10.0.0.85"
files="/service/init.sh /service/certificates /service/service.conf"
target="sdc"
```

Some of the variables have special meanings:

| Attribute | Description |
|---|---|
| **files** | Files and directories that will be included in the contextualization image |
| **target** | device where the contextualization image will be available to the VM instance. Please note that the proper device mapping may depend on the guest OS[1], e.g. ubuntu VMs should use hd* as the target device |

### 4.3.3 Using Context

The VM should be prepared to use the contextualization image. First of all it needs to mount the contextualization image somewhere at boot time. Also a script that executes after boot will be useful to make use of the information provided.

The file `context.sh` is compatible with `bash` syntax so you can easilly source it inside a shellscript to get the variables that it contains.

### 4.3.4 EXAMPLE

Here we propose a way to use this contextualization data. Each unix has their own filesystem layout and way of handling init scripts, this examples assumes a debian-based virtual machine.

We are going to use contextualization data to set the hostname, the IP address and a user with known ssh keys.

First thing, lets outline the `CONTEXT` section of the VM template:

```
CONTEXT = [
  hostname  = "$NAME",
  ip_public = "$NIC[IP, NETWORK=\"Public\"]",
  username  = virtualuser
  files     = "/vms_configuration/id_rsa.pub /vms_configuration/init.sh",
  target    = "sdc"
]
```

The OpenNebula front-end will thus require a `/vms_configuration` folder with:

- `id_rsa.pub`: Public ssh key to be added to the trusted ssh keys of the new user
- `init.sh`: script that will perform the configuration. Explained below.

Now we will need to configure the VM to make use of this data. We are going to place in `/etc/rc.local` as:

```
#!/bin/sh -e

mount -t iso9660 /dev/sdc /mnt

if [ -f /mnt/context.sh ]
  . /mnt/init.sh
fi

umount /mnt

exit 0
```

We use an indirection (rc.local calls init.sh) so changing the script means editing a file locally rather that changing it inside the VMs.

The init.sh script will be the one actually doing the work:

```
#!/bin/bash

if [ -f /mnt/context.sh ]
  . /mnt/context.sh
fi

hostname $HOSTNAME
ifconfig eth0 $IP_PUBLIC

useradd -m $USERNAME

mkdir -p ~$USERNAME/.ssh
cat /mnt/id_rsa.pub >> ~$USERNAME/.ssh/authorized_keys

chown -R $USERNAME /home/$USERNAME
```

## 4.4 A VM sample with context & network

The purpose of this section is to demonstrate how to quickly deploy a VM with OpenNebula in a few easy steps. We will assume that you have properly configured OpenNebula and that you have at least one worker node running KVM (this guide does not work with Xen for the moment).

#### Downloading the pre-configured VM

We have prepared and contextualized a VM which is available for download here. The VM runs ttylinux.

$ cd $ mkdir one-templates $ cd one-templates $ wget http://dev.opennebula.org/attachments/download/61/ttylinux.t
$ tar xvzf ttylinux.tar.gz

#### Preparing the Network

For this example we are going to use the simplest possible network configuration. We provide a file with the package which is a network template example, called small_network.net. You should edit that file and change the LEASES entries to available IPs from your network.

You should also change the BRIDGE entry if you Hypervisor is configured to use a different bridge.

Once the file is prepared we can create the network:

#### Modifying the template

The VM template that you have just downloaded, ttylinux.one, has a few parameters that you should adjust to your convenience:

- source = "/path/to/ttylinux.img",
- files = "/path/to/init.sh /path/to/id_dsa.pub",
- username = "opennebula"
- ip_public = "x.x.x.x"

Do not uncomment the CONTEXT section just yet, though.

You should choose the ip_public field to be an IP which does not match any of the predefined LEASES of small_network.net (only for demonstration purposes).

Finally copy your public key to the directory where you downloaded the VM. Be sure to rename id_dsa.pub to id_rsa.pub in the VM template if you are using an RSA key.

### Running the VM

We are ready to deploy the VM. To do so simply do:

It will take a minute or so to copy the image to $ONE_LOCATION/var and to boot up the system. In the mean time we can figure out what IP the VM will have so that we can ssh into it.

By now, the VM should be up and running:

Note: If the **STAT** attribute is not **runn** you should read the logs to see why it did not boot. You can find these logs in $ONE_LOCATION/var/<id>/vm.log (vm specific log) and $ONE_LOCATION/var/oned.log.

We can ssh into the VM. The user is **root** and the password is **password**:

You might have been wondering how did the VM get automagically configured with an IP from the pool of IPs defined by the ONE Network associated to the VM template. Basically, we developed a script that runs during the bootup procedure which configures the IP address based on the MAC address of the VM. This is more thoroughly explained here.

### Running the VM again with a CONTEXT

We have not yet used the CONTEXT feature of OpenNebula which not only provides a simple way to configure the IP of the VM, but which also allows us to configure users, public keys, the host name, and any other thing we might think of. You can read a more detailed explanation on how to contextualize here.

Edit the ttylinux.one template, uncomment the CONTEXT section and redeploy the VM.

Now we can ssh to the VM without entering a password, since the id_dsa.pub has been copied to the authorized_keys of both root and the username account you have define in the template.

As you can see the IP specified in the CONTEXT section overrides the IP generated by OpenNebula.

### Debugging

Sometimes for unexpected reasons the VMs won't boot. In that case the best way to debug the booting process is by enabling VNC. To do so simply add the following to your VM template:

Read more on this here.