C12G LABS

OpenNebula 1.4 Reference Guide

C12G Labs S.L.

Rev20100611

C12G LABS

Contents

Chapter 1

Configuration

1.1 Daemon Configuration File

The OpenNebula daemon `oned` manages the cluster nodes, virtual networks, virtual machines and users. The configuration file for the daemon is called `oned.conf` and it is placed inside the `$ONE_LOCATION/etc` directory. In this reference document we describe all the format and options that can be specified in `oned.conf`.

Note: If OpenNebula was installed in **system wide** mode this directory becomes `/etc/one/`. 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.

1.1.1 Daemon Configuration Attributes

- `HOST_MONITORING_INTERVAL` : Time in seconds between host monitorization
- `VM_POLLING_INTERVAL` : Time in seconds between virtual machine monitorization
- `VM_DIR` : Remote path to store the VM images, it should be shared between all the cluster nodes to perform live migrations. This path will be used for all the cluster nodes.
- `MAC_PREFIX`: Default MAC prefix to generate virtual network MAC addresses
- `NETWORK_SIZE`: Default size for virtual networks
- `PORT` : Port where oned will listen for xml-rpc calls
- `DEBUG_LEVEL` : Sets the level of verbosity of `$ONE_LOCATION/var/oned.log` log file. Possible values are:

DEBUG_LEVEL	Meaning
0	**ERROR**
1	**WARNING**
2	**INFO**
3	**DEBUG**

Example of this section:

```
#-------------------------------------------------------------------------------
# Daemon configuration attributes
#-------------------------------------------------------------------------------
HOST_MONITORING_INTERVAL = 10
VM_POLLING_INTERVAL      = 10

VM_DIR       = /local/images
```

```
MAC_PREFIX   = "00:01"
NETWORK_SIZE = 254

PORT         =  2633
DEBUG_LEVEL  = 3
```

1.1.2 Information Drivers

The information drivers are used to gather information from the cluster nodes, and they depend on the virtualizer you are using. You can define more than one information manager but make sure it has different names. To define it, the following needs to be set:

- **name**: name for this information driver.
- **executable**: path of the information driver executable, can be an absolute path or relative to `$ONE_LOCATION/lib/mads` (or `/usr/lib/one/mads/` in a system wide installation)
- **arguments**: for the driver executable, usually a probe configuration file, can be an absolute path or relative to `$ONE_LOCATION/etc` (or `/etc/one/` in a system wide installation).
- **default**: default values and configuration parameters for the driver, can be an absolute path or relative to `$ONE_LOCATION/etc` (or `/etc/one/` in a system wide installation).

For more information on configuring the information and monitoring system and hints to extend it please check the information driver configuration guide.

Sample configuration:

```
#-------------------------------------------------------------------------------
# Information Driver Configuration
#-------------------------------------------------------------------------------

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

1.1.3 Transfer Drivers

The transfer drivers are used to transfer, clone, remove and create VM images. You will be using one transfer driver or another depending on the storage layout of your cluster. You can define more than one transfer manager (e.g. you have different configurations for several cluster nodes) but make sure it has different names. To define it, there needs to be set:

- **name**: name for this transfer driver.
- **executable**: path of the transfer driver executable, can be an absolute path or relative to `$ONE_LOCATION/lib/mads` (or `/usr/lib/one/mads/` in a system wide installation)
- **arguments**: for the driver executable, usually a commands configuration file, can be an absolute path or relative to `$ONE_LOCATION/etc` (or `/etc/one/` in a system wide installation)

for the driver executable

- **default**: default values and configuration parameters for the driver, can be an absolute path or relative to `$ONE_LOCATION/etc` (or `/etc/one/` in a system wide installation)

For more information on configuring different storage alternatives please check the storage configuration guide.
Sample configuration:

```
#-------------------------------------------------------------------------------
# Transfer Driver Configuration
#-------------------------------------------------------------------------------

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

1.1.4 Virtualization Drivers

The virtualization drivers are used create, control and monitor VMs on the cluster nodes. You can define more than one virtualization driver (e.g. you have different virtualizers in several cluster nodes) but make sure it has different names. To define it, the following needs to be set:

- **name**: name of the virtualization driver.
- **executable**: path of the virtualization driver executable, can be an absolute path or relative to `$ONE_LOCATION/lib/mads` (or `/usr/lib/one/mads/` in a system wide installation)
- **arguments**: for the driver executable
- **type**: driver type, supported drivers: xen, kvm or ec2
- **default**: default values and configuration parameters for the driver, can be an absolute path or relative to `$ONE_LOCATION/etc` (or `/etc/one/` in a system guide installation)

For more information on configuring and setting up the virtualizer please check the guide that suits you:

- Xen Adaptor
- KVM Adaptor
- VMware Adaptor
- VirtualBox (planned for 1.4.2)

Sample configuration:

```
#-------------------------------------------------------------------------------
# Virtualization Driver Configuration
#-------------------------------------------------------------------------------

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

1.1.5 Hook System

Hooks in OpenNebula are programs (usually scripts) which execution is triggered by a change in state in Virtual Machines. The hooks can be executed either locally or remotely in the node where the VM is running. To configure the Hook System the following needs to be set in the OpenNebula configuration file:

- **executable**: path of the hook driver executable, can be an absolute path or relative to $ONE_LOCATION/lib/mads (or /usr/lib/one/mads/ if OpenNebula was installed in /)
- **arguments** : for the driver executable, can be an absolute path or relative to $ONE_LOCATION/etc (or /etc/one/ if OpenNebula was installed in /)

Sample configuration:

```
HM_MAD = [
    executable = "one_hm" ]
```

Then each hook has to be configured, and for each one the following needs to be set:

- **name**: for the hook, useful to track the hook (OPTIONAL).
- **on**: when the hook should be executed,
  - **- CREATE**, when the VM is created (onevm create)
  - **- RUNNING**, after the VM is successfully booted
  - **- SHUTDOWN**, after the VM is shutdown
  - **- STOP**, after the VM is stopped (including VM image transfers)
  - **- DONE**, after the VM is deleted or shutdown
- **command**: use absolute path here
- **arguments**: for the hook. You can access to VM template variables with $
  - **- $ATTR**, the value of an attribute e.g. $NAME or $VMID
  - **- $ATTR[VAR]**, the value of a vector e.g. $NIC[MAC]
  - **- $ATTR[VAR, COND]**, same of previous but COND select between multiple ATTRs e.g. $NIC[MAC, NETWORK="Public"]
- **remote**: values,
  - **- YES**, The hook is executed in the host where the VM was allocated
  - **- NO**, The hook is executed in the OpenNebula server (default)

Sample configuration:

```
VM_HOOK = [
    name      = "dhcp",
    on        = "create",
    command   = "/bin/echo",
    arguments = "$NAME > /tmp/test.$VMID" ]
```

1.1.6 Example Configuration File

```
HOST_MONITORING_INTERVAL = 10
VM_POLLING_INTERVAL      = 10
VM_DIR=/local/one_images
PORT=2633
DEBUG_LEVEL=3
NETWORK_SIZE = 254
MAC_PREFIX   = "00:50"

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" ]

HM_MAD = [
    executable = "one_hm" ]
VM_HOOK = [
    name      = "mail",
    on        = "running",
    command   = "/usr/local/one/bin/send_mail",
    arguments = "$VMID $NAME",
    remote    = "no" ]
```

Chapter 2

Usage

2.1 Virtual Machine Definition File

A template file consists of a set of attributes that defines a Virtual Machine. The syntax of the template file is as follows:

- Anything behind the pound sign (#) is a **comment**.
- **Strings** are delimited with double quotes ("), if the a double quote is part of the string it needs to be escaped (\").
- **Single Attributes** are in the form:
- **Vector Attributes** that contain several values can be defined as follows:

2.1.1 Capacity Section

The following attributes can be defined to specified the capacity of a VM.

Attribute	Description
NAME	Name that the VM will get for description purposes. If **NAME** is not supplied a name generated by one will be in the form of `one-<VID>`.
MEMORY	Amount of RAM required for the VM, in Megabytes.
CPU	Percentage of CPU divided by 100 required for the Virtual Machine. Half a processor is written 0.5.
VCPU	Number of virtual cpus. This value is **optional**, the default hypervisor behavior is used, usually one virtual CPU

Example:

2.1.2 OS and Boot Options Section

The OS[1] system is defined with the `OS`[1] vector attribute. The following sub-attributes are supported:

Note the hypervisor column states that the attribute is **O**ptional, **M**andatory, or **-** not supported for that hypervisor

OS[1] Sub-Attribute	Description	XEN	KVM
KERNEL	path to the OS[1] kernel to boot the image	**M** see (*)	O
INITRD	path to the initrd image	O (for kernel)	O (for kernel)
ROOT	device to be mounted as root	O (for kernel)	O (for kernel)
KERNEL_CMD	arguments for the booting kernel	O (for kernel)	O (for kernel)
BOOTLOADER	path to the bootloader executable	**M** see (*)	O
BOOT	boot device type: `hd`,`fd`,`cdrom` ,`network`	-	**M**

[1]Operating System

(*) Xen needs a kernel or a bootloader to be specified. If both are set in the template, the kernel boot method will be used.

Example, a VM booting from `sda1` with kernel `/vmlinuz` :

2.1.3 Disks Section

The disks of a VM are defined with the `DISK` vector attribute. You can define as many `DISK` attributes as you need. There are two special disk types that are created on-the-fly in the target resource: `swap` and `fs`. The following sub-attributes for `DISK` are supported:

Note the hypervisor column states that the attribute is **O**ptional, **M**andatory, or `-` not supported for that hypervisor

DISK Sub-Attribute	Description	XEN	KVM
TYPE	disk type:`floppy`, `disk`, `cdrom`, `swap`, `fs`, `block`	O (only `swap`, `fs` and `block`) (if not present, `disk` will be assumed)	O
SOURCE	disk file location path or URL[2]	**M**	**M**
SIZE	size in Mb for **swap**, **fs** and **block** images	**M** (for `swap` and `fs`)	**M** (for `swap` and `fs`)
FORMAT	filesystem type for the **fs** images	**M** (for `fs`)	**M** (for `fs`)
TARGET	device to map disk	**M**	**M**
CLONE	clone this image `yes` (default), or `no`	O	O
SAVE	save this image after shutting down the VM `yes`, or `no` (default)	O	O
READONLY	`yes`, or `no` (default)	O	O
BUS	type of disk device to emulate: `ide`, `scsi`	-	O

Example, a VM with three disks: the base system attached to `sda1`, a clean filesystem attached to `sda2`, and a swap partition attached to `sda3`. Note that `fs` and `swap` are generated on-the-fly:

For more information on image management and moving please check the Storage guide.

2.1.4 Network Section

Each network interface of a VM is defined with the `NIC` vector attribute. You can define as many `NIC` attributes as you need. The following sub-attributes for `NIC` are supported:

Note the hypervisor column states that the attribute is **O**ptional, **M**andatory, or `-` not supported for that hypervisor

[2] Uniform Resource Locator

NIC Sub-Attribute	Description	XEN	KVM
NETWORK	Name of the network, as defined by `onevnet` to attach this device	O	O
IP	Request an specific IP from the `NETWORK`	O	O
MAC	HW address associated with the network interface	O	O
BRIDGE	Name of the bridge the network device is going to be attached to.	O	O
TARGET	name for the tun device created for the VM	-	O
SCRIPT	name of a shell script to be executed after creating the tun device for the VM	-	O
MODEL	hardware that will emulate this network interface	-	O

Example, a VM with two NIC attached to two different networks, one make use of the Virtual Network Manager lease feature:

For more information on setting up virtual networks please check the Managing Virtual Networks guide.

2.1.5 I/O Devices Section

The following I/O interfaces can be defined for a VM:

Note the hypervisor column states that the attribute is **O**ptional, **M**andatory, or **-** not supported for that hypervisor

Attribute	Description	XEN	KVM
INPUT	Define input devices, available sub-attributes: **- TYPE**: values are `mouse` or `tablet` **- BUS**: values are `usb`, `ps2` or `xen`	-	O
GRAPHICS	Wether the VM should export its graphical display and how, available sub-attributes: **- TYPE**: values: `vnc` `sdl` **- LISTEN**: IP to listen on. **- PORT**: port for the VNC server **- PASSWD**: password for the VNC server **- KEYMAP**: keyboard configuration locale to use in the VNC display	O	O

Example:

Note For KVM hypervisor the port number is a real one, not the VNC port. So for VNC port 0 you should specify 5900, for port 1 is 5901 and so on.

2.1.6 Placement Section

The following attributes placement constraints and preferences for the VM:

Note the hypervisor column states that the attribute is **O**ptional, **M**andatory, or **-** not supported for that hypervisor

Attribute	Description	XEN	KVM
REQUIREMENTS	Boolean expression that rules out provisioning hosts from list of machines suitable to run this VM.	O	O
RANK	This field sets which attribute will be used to sort the suitable hosts for this VM. Basically, it defines which hosts are *more suitable* than others.	O	O

Example:

2.1.7 Context Section

Context information is passed to the Virtual Machine via an ISO[3] mounted as a partition. This information can be defined in the VM template in the optional section called Context, with the following attributes:

Attribute	Description
VARIABLE	Variables that store values related to this virtual machine or others. The name of the variable is arbitrary (in the example, we use hostname).
FILES	space-separated list of paths to include in context device.
TARGET	device to attach the context ISO[3].

The values referred to by **VARIABLE** can be:

- **$<template_variable>**: any single value variable of the VM template, like for example $NAME
- **$<template_variable>[<attribute>]**: Any single value contained in a multiple value variable in the VM template, like for example $NIC[IP].
- **$<template_variable>[<attribute>, <attribute2>=<value2>]**: Any single value contained in a multiple value variable in the VM template, setting one atribute to discern between multiple variables called the same way, like for example $NIC[IP, NETWORK="Private LAN"].
- **$<vm_id>.<context_var>**: Any $<template_variable> (expressed in any of the previous ways) pertaining to VM with id=<vm_id>, like for example $4.$NAME, referring to the NAME of the VM with ID=4.

Example:

Requirement Expression Syntax

The syntax of the requirement expressions is defined as:

```
=' NUMBER | VARIABLE '>' NUMBER | VARIABLE '<' NUMBER | VARIABLE '=' STRING | VARIABLE
'!
```

Each expression is evaluated to 1 (TRUE) or 0 (FALSE). Only those hosts for which the requirement expression is evaluated to TRUE will be considered to run the VM.

Logical operators work as expected (less '<', greater '>', '&' AND, '|' OR, '!' NOT), '=' means equals with numbers (floats and integers). When you use '=' operator with strings, it performs a shell wildcard pattern matching.

! Any variable defined by the Information Manager driver can be used in the requirements. Check the configuration guide to find out how to extend the information model

! There are some predefined variables that can be used: `HOSTNAME`, `TOTALCPU`, `TOTALMEMORY`, `FREEMEMORY`, `FREECPU`, `USEDMEMORY`, `USEDCPU`, `HYPERVISOR`

! If using OpenNebula's default match-making scheduler in a hypervisor heterogeneous environment, it is a good idea to add an extra line like the following to the VM template to ensure its placement in a VMWare hypervisor enabled machine.

Examples:

[3]International Organization for Standardization

Rank Expression Syntax

The syntax of the rank expressions is defined as:

Rank expressions are evaluated using each host information. '+', '-', '*', '/' and '-' are arithmetic operators. The rank expression is calculated using floating point arithmetics, and then round to an integer value.

⚠ The rank expression is evaluated for each host, those hosts with a higher rank are used first to start the VM. The rank policy must be implemented by the scheduler. Check the configuration guide to configure the scheduler.

⚠ Similar to the requirements attribute, any number (integer or float) attribute defined for the host can be used in the rank attribute

Examples:

2.1.8 RAW Section

This optional section of the VM template is used whenever the need to pass special attributes to the underlying hypervisor arises. Anything placed in the data attribute gets passed straight to the hypervisor, unmodified.

RAW Sub-Attribute	Description	XEN	KVM
TYPE	Possible values are: `kvm`,`xen`	O	O
DATA	Raw data to be passed directly to the hypervisor	O	O

Example

2.2 Command Line Interface

OpenNebula provides four commands to interact with the system:

- `onevm`: to submit, control and monitor virtual machines
- `onehost`: to add, delete and monitor hosts
- `onevnet`: to add, delete and monitor virtual networks
- `oneuser`: to add, delete and monitor users

These commands share common options described below:

- **-l**, **list x,y,z**: Selects columns to display with list command.
- **list-columns**: Information about the columns available to display, order or filter.
- **-o**, **order x,y,z**: Order by these columns, column starting with - means decreasing order.
- **-f**, **filter x,y,z**: Filter data. An array is specified with column=value pairs.
- **-d**, **delay seconds**: Sets the delay in seconds for top command.
- **-h**, **help**: Shows help information.
- **version**: Shows version and copyright information.
- **-v**, **verbose**: Tells more information if the command is successful
- **-x**, **xml**: Returns xml instead of human readable text

Number ranges can also be specified this way:

- `[<start>-<end>]`: generates numbers from start to end
- `[<start>+<count>]`: generates a range that starts with the number provided and has count number of elements

If start first number is 0 then it will pad the numbers generated with 0 to the same size as the last element in the range.

Example:

```
[9-11]: 9 10 11
[09-11]: 09 10 11
[8+3]: 8 9 10
[08+3]: 08 09 10
```

2.2.1 onevm

This command enables the user to manage virtual machines in the ONE server. The user can allocate, deploy, migrate, suspend, resume and shutdown a virtual machine with the functionality present in **onevm**.

```
onevm [<options>] <command> [<parameters>]
```

Command Summary

create

Submits a new virtual machine, adding it to the ONE VM pool. It requires the filename of the VM template.

```
onevm create <template>
```

deploy

Starts a previously submitted VM on a specific host

```
onevm deploy <vm_id> <host_id>
```

shutdown

Shutdown an already deployed VM

```
onevm shutdown <vm_id>
```

livemigrate

Migrates a running VM to another host without downtime

```
onevm livemigrate <vm_id> <host_id>
```

migrate

Saves a running VM and starts it again in the specified host

```
onevm migrate <vm_id> <host_id>
```

hold

Sets a VM to hold state, scheduler will not deploy it

```
onevm hold <vm_id>
```

release
Releases a VM from hold state

```
onevm release <vm_id>
```

stop
Stops a running VM

```
onevm stop <vm_id>
```

suspend
Saves a running VM

```
onevm suspend <vm_id>
```

resume
Resumes the execution of a saved VM

```
onevm resume <vm_id>
```

delete
Deletes a VM from the pool

```
onevm delete <vm_id>
```

restart
Resubmits the VM after failure

```
onevm restart <vm_id>
```

list
Shows VMs in the pool

```
onevm list <filter_flag>
    where filter_flag can be
        a, all   --> all the known VMs
        m, mine  --> the VMs belonging to the user in ONE_AUTH
        uid      --> VMs of the user identified by this uid
        username --> VMs of the user identified by the username
```

show
Gets information about a specific VM

```
onevm show <vm_id>
```

top
Lists VMs continuously

```
onevm top
```

history
Gets history from VMs, if no vm_id is provided it will list history for all known VMs

```
onevm history [<vm_id> <vm_id> ...]
```

Information Columns

ID	ONE VM identifier
USER	Username of the VM owner
NAME	Name of the ONE
STAT	Status of the VM
CPU	CPU percentage used by the VM
MEM	Memory used by the VM
HOSTNAME	Host where the VM is being or was run
TIME	Time since the submission of the VM (days hours:minutes:seconds)

VM States

pend	pending
hold	VM on hold (not runnable)
stop	stopped
susp	suspended
done	finished
prol	prolog
boot	booting
runn	running
migr	migrating
save	saving the VM to disk
epil	epilog
shut	shutting down
fail	failed

2.2.2 onehost

This command enables the user to manage hosts in the Open Nebula server. It provides functionality to allocate, get information and delete a particular host or to list all the available hosts.

```
onehost [<options>] <command> [<parameters>]
```

Command Summary

create
Adds a new machine to the pool

```
onehost create <hostname> <im_mad> <vmm_mad>
```

im_mad and *vmm_mad* as written in oned.conf.

show
Gets info from a single host

```
onehost show <host_id>
```

delete
Removes a machine from the pool

```
onehost delete <host_id>
```

list
Lists machines in the pool

```
onehost list
```

enable
Enables host

```
onehost enable <host_id>
```

disable
Disables host

```
onehost disable <host_id>
```

top
Lists hosts continuously

```
onehost top
```

Information Columns

HID	Host ID
NAME	Host name
RVM	Number of running VMs
TCPU	Total CPU (percentage)
FCPU	Free CPU (percentage)
ACPU	Available CPU (not allocated by VMs)
TMEM	Total memory
FMEM	Free memory
STAT	Host status

2.2.3 onevnet

This command enables the user to manage virtual networks in the OpenNebula server. It provides functionality to create, get information and delete a particular network or to list available and used IP's.

```
onevnet <command> [<parameters>]
```

Command Summary

create
Adds a new virtual network to the pool

```
onevnet create <network_configuration_file>
```

show
Gets info from a single virtual network

```
onevnet show <network_id>
```

delete
Removes a virtual network

```
onevnet delete <network_id>
```

list
Lists virtual networks in the pool

```
onevnet list <filter_flag>
    where filter_flag can be
        a, all   --> all the known VNs
        m, mine  --> the VNs belonging to the user in ONE_AUTH
        uid      --> VNs of the user identified by this uid
        username --> VNs of the user identified by the username
```

Information Columns

NID	Network ID
NAME	Name of the virtual network
TYPE	Type of virtual network (0=ranged, 1=fixed)
BRIDGE	Bridge associated to the virtual network
#LEASES	Number of leases used from this virtual network

2.2.4 oneuser

This command enables the OpenNebula administrator to manage users, adding, listing and deleting them.

```
oneuser [<options>] <command> [<parameters>]
```

Command Summary

create
Creates a new user

```
oneuser create username password
```

delete
Removes a user

```
onevnet delete <uid>
```

list
Lists all the user prenset in OpenNebula

```
onevnet list
```

Information Columns

UID	User ID
NAME	Name of the user
PASSWORD	SHA1 encrypted password
ENABLE	Whether the user is enabled or not

Chapter 3

Programming

3.1 Libvirt API

The OpenNebula libvirt implementation lets you use any libvirt application at a distributed level. In a nutshell, you'll be able to use your libvirt XML[1] description files and any libvirt tool, like virsh or virt-manager to connect to OpenNebula. In this way, you can manage and monitor your VMs in a distributed environment using the current libvirt tools. This is, a whole cluster can be managed as any other libvirt node.

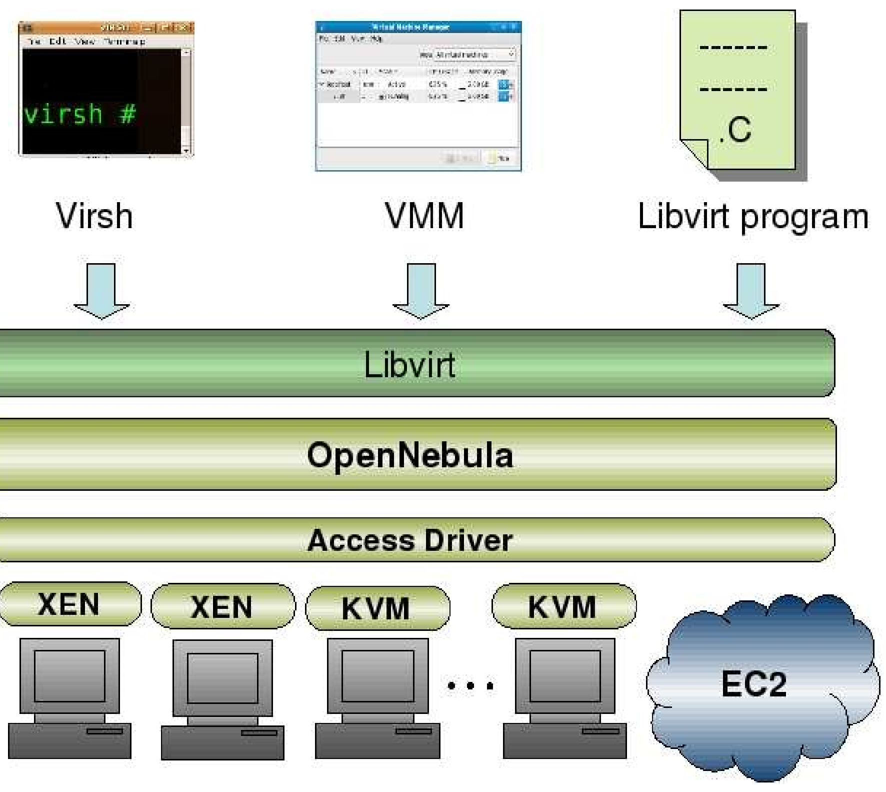

Figure 3.1:

For example, you can create your domain with virsh create, then OpenNebula will look for a suitable resource, transfer the VM images and boot your VM using any of the supported hypervisors. The distributed management is completely transparent to the libvirt application.

[1]Extensible Markup Language

3.1.1 Requirements

The OpenNebula driver for libvirt has been incorporated in libvirt main distribution, so in order to use it you will need in the OpenNebula front-end:

- libvirt >= 0.6.5
- OpenNebula >= 1.2

3.1.2 Configuration

Libvirt needs to be configured (look here for configuration options of libvirt). Afterwards, the **libvirtd** deaemon has to be started. Also, the `oned` daemon has to be up and running.

3.1.3 Checking the installation

To verify the installation of ”ONE driver” for libvirt, execute ”virsh” with the following URI[2]: ”one:///” to connect to the local ONE instance. With the installation done and both daemons running (libvirt’s and OpenNebula’s), you should be able to connect to OpenNebula:
///

3.1.4 Usage Examples

The driver allows libvirt to interact with OpenNebula as a virtualization hypervisor, so you can use libvirt applications to manage a distributed infrastructure.

Using virsh

The virsh program is the main interface for managing libvirt guest domains.

Connect to the hypervisor ///

Creating and Monitoring a new VM

Resume a suspended Domain

Shutdown Domains

Programming Examples

The following examples illustrate how to use the C API[3] >= libvirt-0.6.5 with OpenNebula. This will allow you to develop applications that interacts with OpenNebula as it was a single hypervisor.

Set connection to OpenNebula

```
virConnectPtr conn;
conn = virConnectOpen("one:%%///%%");
```

[2]Uniform Resource Identifier
[3]Application Programming Interface

Creating a Domain and getting info

```
 virDomainPtr dom = NULL;
 virDomainInfo info;
 virDomainCreateLinux(conn,XML_template,NULL);
 ...
/* Find the domain of the given id */
 dom = virDomainLookupByID(conn, id);

/* Get information: */
 ret = virDomainGetInfo(dom, &info);
 printf("Domain %d CPUs\n",info.nrVirtCpu);
 printf("Domains %d Memory\n", id, info.memory);
 printf("Domains %s Status\n", id, info.status);
```

Suspending a Virtual Machine

```
 virDomainPtr dom;
 virDomainInfo ret;
 dom =  virDomainLookupByName(conn, vm_name);
 ret = virDomainGetInfo(dom, &info);

/* Check if the current State allows to shutdown the VM */
 if(info.state == VIR_DOMAIN_RUNNING)
   virDomainShutdown(dom);
 else
  fprintf(stderr, "Failed to Shutdown %s\n", vm_name);
```

XML Domain Definition

Libvirt make use of XML[1] format to represent domains, there are some limitations on the attributes that may be specified when interfacing OpenNebula. The following list details the attributes and options supported by the libvirt driver:

General Metadata

- **Name**

OS Booting The booting method should be configured using a ”Direct kernel boot” method, as:

```
<os>
  <type>hvm</type>
  <kernel>/boot/vmlinuz-2.6.24-17-xen</kernel>
  <initrd>/boot/initrd.img-2.6.24-17-xen</initrd>
  <cmdline></cmdline>
  <root>sda1</root>
</os>
```

The following options can be define within an `os` section

- **type**, should be set to hvm
- **kernel**
- **initrd**
- **root**
- **cmdline**

Basic Resources

- **memory**
- **vcpu**

3.1.5 Devices

Sample configuration of a VM disk

```
<disk type='file' device='disk'>
  <source file='/images/sgehosts/01/swap.img'/>
  <target dev='sda2'/>
</disk>
```

Attributes for a disk section:

- **disk**: the type option should be set to file and device to disk
- **source**: option file is supported
- **target**: the option bus is no supported

Network The network of the VMs should be configured following the semantics of Bridged networking, or Virtual Network for example:

```
...
<interface type='bridge'>
  <source bridge=.../>
  <target dev=.../>
  <mac address='00:16:3e:01:01:01'/>
</interface>
...
```

```
...
<interface type='network'>
  <source network='mynetwork'/>
</interface>
...
```

The following options can be define within an interface section:

- **type**: should be set to bridge or network
- **source** : bridge,network
- **target**: optional at bridge configuration, not used at network
- **mac** optional at bridge configuration, not used at network

XML description example

Following, there is an example ”libvirt XML[1] Description” file defining a guest Domain in OpenNebula:

```
<domain type='one'>
  <name>vm01</name>
  <memory>32768</memory>
  <vcpu>1</vcpu>
  <os>
    <type>hvm</type>
    <cmdline></cmdline>
    <kernel>/boot/vmlinuz-2.6.24-17-xen</kernel>
    <initrd>/boot/initrd.img-2.6.24-17-xen</initrd>
    <root>sda1</root>
  </os>
  <devices>
    <disk type='file' device='disk'>
      <source file='/images/sgehosts/01/disk.img'/>
      <target dev='sda1'/>
    </disk>
    <disk type='file' device='disk'>
      <source file='/images/sgehosts/01/swap.img'/>
      <target dev='sda2'/>
    </disk>
    <interface type='bridge'>
      <source bridge='eth0'/>
      <target dev='tap0'/>
      <mac address='00:16:3e:01:01:01'/>
    </interface>
  </devices>
</domain>
```

Driver Support

These are the main functions to manage VM supported by ONE driver:

- **virDomainCreate**
- **virDomainCreateLinux**
- **virDomainGetInfo**
- **virDomainSuspend**
- **virDomainResume**
- **virDomainShutDown**

Libvirt API[3] details can be scouted here.
Libvirt calls supported on the OpenNebula driver are listed in the table below:

API[3] Call	API[3] call
virClose	virDomainGetConnect
virConnectGetVersion	virDomainGetUUID
virConnectNumOfDefinedDomains	virDomainLookupByName
virDomainCreate	virDomainSuspend
virDomainDestroy	virInitialize
virDomainGetAutostart	virConnectGetURI
virDomainGetInfo	virConnectListDomains
virDomainGetName	virConnectOpen
virDomainGetVcpus	virDomainDefineXML
virDomainLookupByID	virDomainFree
virDomainLookupByUUIDString	virDomainGetID
virDomainShutdown	virDomainGetOSType
virGetVersion	virDomainGetUUIDString
virConnectListDefinedDomains	virDomainLookupByUUID
virConnectNumOfDomains	virDomainResume
virDomainCreateLinux	virDomainUndefine

3.2 EC2 Query API

The EC2 Query API offers the functionality exposed by Amazon EC2: upload images, register them, run, monitor and terminate instances, etc. In short, Query requests are HTTP[4] or HTTPS requests that use the HTTP[4] verb GET or POST and a Query parameter.

OpenNebula implements a subset of the EC2 Query interface, enabling the creation of public clouds managed by OpenNebula. In this first release of the API[3] implementation, the methods implemented are:

- **upload image**: Uploads an image to the repository manager
- **register image**: Registers an image (previously uploaded in the repository manager) in order to be launched, check this link for the method description.
- **describe images**: Lists all registered images belonging to one particular user.

* **run instances**: Runs an instance of a particular image (that needs to be referenced), check this link for the method

```
*
**[[http://docs.amazonwebservices.com/AWSEC2/2009-04-04/APIReference/ApiReference-query-DescribeInstances.html|
instances]]**: Outputs a list of launched images belonging to one particular user,
[[http://docs.amazonwebservices.com/AWSEC2/2009-04-04/APIReference/ApiReference-query-RunInstances.html|check
this link for the method description]].
```

- **terminate instances**: Shutdown a virtual machine(or cancel, depending on its state), check this link for the metho

3.2.1 User Account Configuration

An account is needed in order to use the OpenNebula cloud. The cloud administrator will be responsible for assigning these accounts, which have a one to one correspondence with OpenNebula accounts, so all the cloud administrator has to do is check the configuration guide to setup accounts, and automatically the OpenNebula cloud account will be created.

In order to use such an account, the end user can make use of clients programmed to access the services described in the previous section. For this, she has to set up his environment, particularly the following aspects:

[4]Hyper Text Transfer Protocol

- **Authentication**: This can be achieved in three different ways, here listed in order of priority (i.e. values specified in the argument line supersede environmental variables)
  - Using the **commands arguments**. All the commands accept an **Access Key** (as the OpenNebula username) and a **Secret Key** (as the OpenNebula password)
  - Using **EC2_ACCESS_KEY** and **EC2_SECRET_KEY** environment variables the same way as the arguments
  - If none of the above is available, the **ONE_AUTH** variable will be checked for authentication (with the same used for OpenNebula CLI).
- **Server location**: The command need to know where the OpenNebula cloud service is running. That information needs to be stored within the **EC2_URL**[5] environment variable (in the form of a http URL[5], including the port if it is not the standard 80).

The `EC2_URL`[5] has to use the FQDN of the EC2-Query Server

3.2.2 Hello Cloud!

Lets take a walk through a typical usage scenario. In this brief scenario it will be shown how to upload an image to the OpenNebula image repository, how to register it in the OpenNebula cloud and perform operations upon it.

- **upload_image**

Assuming we have a working Gentoo installation residing in an **.img** file, we can upload it into the OpenNebula cloud using the **econe-upload** command:
The user should take note of this **ImageId**, as it will be needed to register the image.

- **register_image**

The next step should be registering the image to enable its instantiation. We can do this with the **econe-register** command:

- **describe_images**

We will need the **ImageId** to launch the image, so in case we forgotten we can list registered images using the **econe-describe-images** command:
* **run_instance**
Once we recall the ImageId, we will need to use the **econe-run-instances** command to launch an Virtual Machine instance of our image:
We will need the **InstanceId** to monitor and shutdown our instance, so we better write down that 15.

- **describe_instances**

If we have too many instances launched and we don't remember everyone of them, we can ask **econe-describe-instances** to show us which instances we have submitted.
We can see that the instances with Id 15 has been launched, but it is still pending, i.e., it still needs to be deployed into a physical host. If we try the same command again after a short while, we should be seeing it running as in the following excerpt:
* **terminate_instances**
After we put the Virtual Machine to a good use, it is time to shut it down to make space for other Virtual Machines (and, presumably, to stop being billed for it). For that we can use the **econe-terminate-instances** passing to it as an argument the **InstanceId** that identifies our Virtual Machine:
You can obtain more information on how to use the above commands accessing their Usage help passing them the **-h** flag

[5]Uniform Resource Locator

3.3 OpenNebula OFG OCCI API

3.3.1 Resources

The OpenNebula OCCI API[3] is a RESTful service to create, control and monitor cloud resources based on the latest draft of the OGF OCCI API specification. There are two types of resources that resemble the basic entities managed by the OpenNebula system, namely:

- **Pool Resources (PR)**: Represents a collection of elements owned by a given user. In particular three pool resources are defined: COMPUTES, NETWORKS and STORAGE.
- **Entry Resources (ER)**: Represents a single entry within a given collection: COMPUTE, NETWORK and DISK.

A `COMPUTE` entry resource can be linked to one or more `DISK` or `NETWORK` resources.

3.3.2 Methods

The methods associated with each resource type are as follows:

- **Pool Resources (PR)**
  - **GET**: to list all the entry resources in that pool resource owned by the user
  - **POST**: to create a new entry resource
- **Entry Resources (ER)**
  - **GET**: to list the information associated with that resource
  - **PUT**: to update the resource (only supported by the COMPUTE resource)
  - **DELETE**: to delete the resource

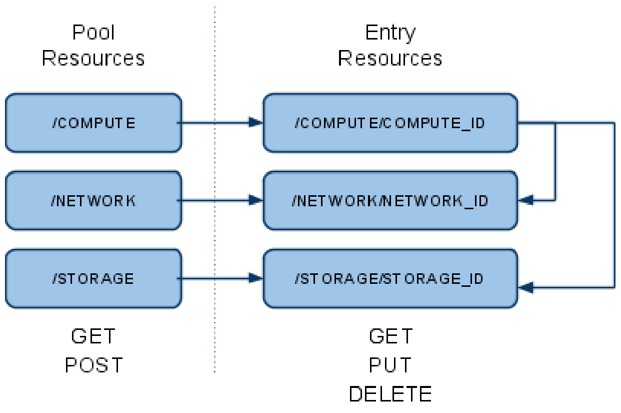

Figure 3.2:

3.3.3 Data Schema (XML Format)

This section describes the XML[1] format used to represent `COMPUTE`, `NETWORK` and `DISK` resources; as well as the collection of them (Pool Resources, PRs).

The Pool Resources

The root element required for all the PRs is named after the pool name, eg. `COMPUTES`, `NETWORKS` or `STORAGE` (note that XML[1] tags are upper case). No attributes can be defined for the root element.

Each one of ERs in the pool are described by an element (e.g. `COMPUTE`, `NETWORK` or `DISK`) with one attribute:

- `href`, a URI[2] for the ER

Example:

```
<COMPUTES>
     <COMPUTE href="http://www.opennebula.org/compute/234">
     <COMPUTE href="http://www.opennebula.org/compute/432">
     <COMPUTE href="http://www.opennebula.org/compute/123">
</COMPUTES>
```

The Network Resource

The `NETWORK` element defines a virtual network that interconnects those `COMPUTES` with a network interface card attached to that network. The traffic of each network is isolated from any other network, so it constitutes a broadcasting domain.

The following elements can be defined for a `NETWORK`:

- `ID`, the uuid of the network
- `NAME`, describing the network
- `ADDRESS`, of the network
- `SIZE`, of the network, defaults to C

Example:

```
<NETWORK>
      <ID>123</ID>
      <NAME>BlueNetwork</NAME>
      <ADDRESS>192.168.0.1</ADDRESS>
      <SIZE>C</SIZE>
</NETWORK>
```

The Disk Resource

The `DISK` element defines a virtual disk that supports a VM block device. The following elements can be defined:

- `ID`, the uuid of the image
- `NAME`, describing the image
- `SIZE`, of the image in MBs
- `URL`[5], pointer to the original image

Example:

```
<DISK>
     <ID>123</ID>
     <NAME>Ubuntu 9.04 LAMP</NAME>
     <SIZE>2048</SIZE>
     <URL>file:///images/ubuntu/jaunty.img</URL>
</DISK>
```

The Compute Resource

The `COMPUTE` element defines a virtual machine by specifying its basic configuration attributes such as `NIC` or `DISK`. The following elements can be defined:

- `ID`, the uuid of the virtual machine.
- `NAME`, describing the virtual machine.
- `TYPE`, a COMPUTE type specifies a CPU and memory capacity, valid types are small, medium and large.
- `STATE`, the state of the COMPUTE. This can be changed to
  - stopped
  - suspended
  - resume
  - cancel
  - shutdown
  - done
- `DISKS`, the block devices attached to the virtual machine. The following devices can be specified:
  - `DISK`, a block device supported by a previously registered image. The id attribute specifies the image, dev the device to attach the image to.
  - `SWAP`, a swap device attached to the specified device (dev) with the given size (in MBs).
  - `FS`, a plain filesystem attached to the specified device (dev) with the given size (in MBs) and format (ext3 and ext2).
- `NICS`, the network interfaces, defined with a list of NIC elements. Each NIC can have the following attributes:
  - network, the UUID of the network to bind the interface. Use 0 to make this interface attached to the internet.
  - ip, ask for a given IP of the network.

  Example:

```
<COMPUTE>
    <ID>123AF</ID>
    <NAME>Web Server</NAME>
    <INSTANCE_TYPE>small</INSTANCE_TYPE>
    <STATE>running</STATE>
    <DISKS>
        <DISK image="234" dev="sda1"/>
        <SWAP size="1024" dev="sda2"/>
        <FS size="1024" format="ext3" dev="sda3"/>
    </DISKS>
    <NETWORK>
        <NIC network="4567f" ip="19.12.1.1"/>
        <NIC network="0"/>
    </NETWORK>
</COMPUTE>
```

3.3.4 Authentication & Authorization

User authentication will be HTTP Basic access authentication to comply with REST philosophy. Authorization will be handled by OpenNebula's user management module, that currently works as:

- There are normal users and one privilege user (known as oneadmin)
- All users can access retrieve information of all PRs
- All users can perform operations over all PRs
- Normal users can perform operations over their ERs, but no over other users'
- Privilege user oneadmin can perform operations over all ERs

3.3.5 HTTP Headers

The following headers are compulsory:

- **Content-Length**: The size of the Entity Body in octets
- **Content-Type**: application/xml

Uploading images needs HTTP[4] multi part support, and also the following header

- **Content-Type**: multipart/form-data

3.3.6 Return Codes

The OpenNebula Cloud API[3] uses the following subset of HTTP[4] Status codes:

- **200 OK** : The request has succeeded. The information returned with the response is dependent on the method used in the request, as follows:
  - **GET** an entity corresponding to the requested resource is sent in the response
  - **POST** an entity containing the result of the action
- **201 Created** : Request was successful and a new resource has being created
- **202 Accepted** : The request has been accepted for processing, but the processing has not been completed
- **204 No Content** : The request has been accepted for processing, but no info in the response
- **400 Bad Request** : Malformed syntax
- **401 Unauthorized** : Bad authentication
- **403 Forbidden** : Bad authorization
- **404 Not Found** : Resource not found
- **500 Internal Server Error** : The server encountered an unexpected condition which prevented it from fulfilling the request.
- **501 Not Implemented** : The functionality requested is not supported

The methods specified below are described without taking into account **4xx** (can be inferred from authorization information in section above) and **5xx** errors (which are method independent). HTTP[4] verbs not defined for a particular entity will return a **501 Not Implemented**.

3.3.7 Pool Resource Methods

Computes

- Base URL[5] : http://www.opennebula.org/compute

Networks

- Base URL[5] : http://www.opennebula.org/network

Storage

- Base URL[5] : http://www.opennebula.org/storage

All the above resources share the same HTTP[4] verb semantics:

Method	Meaning / Entity Body	Response
GET	Request for the contents of the pool	**200 OK**: An XML[1] representation of the pool in the http body
POST	Request for the creation of an ER. An XML[1] representation of a VM without the ID element should be passed in the http body	**201 Created**: An XML[1] representation of a ER of type COMPUTE with the ID

3.3.8 Entity Resource Methods

Network

- **Base URL**[5] : http://www.opennebula.org/network/<net_id>

Method	Meaning / Entity Body	Response
GET	Request the representation of the network resource identified by <net_id>	**200 OK** : An XML[1] representation of the network in the http body
DELETE	Deletes the Network resource identified by <net-id>	**200 OK**: The Network has been successfully deleted

Storage

- **Base URL**[5] : http://www.opennebula.org/storage/<storage_id>

Method	Meaning / Entity Body	Response
GET	Request the representation of the image resource identified by <storage_id>	**200 OK** : An XML[1] representation of the image in the http body
DELETE	Deletes the Image resource identified by <storage_id>	**200 OK** : The image has been successfully deleted

Compute

- **Base URL**[5] : http://www.opennebula.org/compute/<compute_id>

Method	Meaning / Entity Body	Response
GET	Request the representation of the Compute resource identified by <compute_id>	**200 OK** : An XML[1] representation of the Compute in the http body
PUT	Update request for a Compute identified by <compute_id>	**202 Accepted** : The update request is being process, polling required to confirm update
DELETE	Deletes the Compute resource identified by <compute_id>	**200 OK** : The Compute has been successfully deleted

3.3.9 Implementation Notes

Authentication

It is recommended that the server-client communication is performed over HTTPS to avoid sending user authentication information in plain text.

Notifications

HTTP[4] protocol does not provide means for notification, so this API[3] relies on asynchronous polling to find whether a VM update is successful or not.

Chapter 4

Internals

4.1 Architecture

The OpenNebula internal architecture can be divided into three layers:

- *Tools*, management tools developed using the interfaces provided by the OpenNebula Core.
- *Core*, the main virtual machine, storage, virtual network and host management components.
- *Drivers*, to plug-in different virtualization, storage and monitoring technologies and Cloud services into the core.

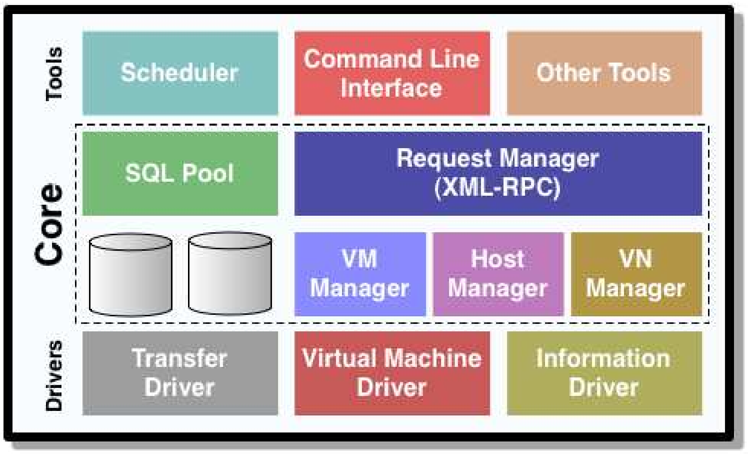

Figure 4.1: OpenNebula Architecture

4.1.1 Tools

This layer contains tools distributed with OpenNebula, such as the CLI, the scheduler, the libvirt API[1] implementation or the Cloud RESTful interfaces, and also 3rd party tools that can be easily created using the XML[2]-RPC[3] interface or the new OpenNebula Cloud API[1].

Command Line Interface

A CLI for infrastructure administrators and users is provided with OpenNebula to manually manipulate the virtual infrastructure. For more information about the CLI (command line interface) go **here**.

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

Scheduler

The Scheduler is an independent entity in the OpenNebula architecture, so it can be easily tailored or changed since it is decoupled from the rest of the components. It uses the XML[2]-RPC[3] interface provided by OpenNebula to invoke actions on virtual machines. The scheduler distributed with OpenNebula allows the definition of several resource and load aware policies.

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

4.1.2 OpenNebula Core

The core consists of a set of components to control and monitor virtual machines, virtual networks, storage and hosts. The core performs its actions (e.g. monitor a host, or cancel a VM) by invoking a suitable driver. The main functional components of OpenNebula core are:

- **Request Manager**, to handle client requests
- **Virtual Machine Manager**, to manage and monitor of VMs
- **Transfer Manager**, to manage VM images
- **Virtual Network Manager**, to manage virtual networks
- **Host Manager**, to manage and monitor physical resources
- **Database**, persistent storage for ONE data structures

Request Manager

The Request Manager exposes a XML[2]-RPC[3] Interface, and then depending on the invoked method a given component is called internally. The XML[2]-RPC[3] decouples most of the functionality in the OpenNebula core, from external components i.e. the Scheduler.

Virtual Machine Manager

This component is responsible for the management and monitoring of VMs. The operations of the VM Manager are abstracted from the underlying hypervisor the use of plugable drivers.

Transfer Manager

The Transfer Manager (TM) is in charge of all the files transfers needed for the correct deployment of virtual machines. This includes the transfer of images **to** the cluster node selected for running the images' virtual machine, the transfer of the image **from** the cluster node to the image repository, the transfer of checkpoint files between cluster nodes for cold migrations or to the cluster front-end when the virtual machine is stopped, etc.

Virtual Network Manager

The Virtual Network Manager (VNM) is responsible for the handling of IP and MAC addresses, allowing the creation of virtual networks by keeping track of leases (a set form by one IP and one MAC valid on a particular network) and their association with virtual machines and the physical bridges the VM are using.

Host Manager

This component manages and monitors the physical hosts. Monitor and management actions are performed also through a suitable driver. The host monitoring infrastructure is flexible and can be extended to include any host attribute.

Database

A persistent generic pool based on a SQLite3 backend is the core component of the OpenNebula internal data structures. This component provides OpenNebula with the scalability and reliability (in case of failure the state of OpenNebula is automatically recovered) needed in the management VMs.

Note that this information can be accessed through the SQLite3 interface to develop custom accounting applications.

4.1.3 Drivers

OpenNebula has a set of pluggable modules to interact with specific middleware (e.g. virtualization hypervisor, cloud services, file transfer mechanisms or information services), these adaptors are called Drivers.