OpenNebula.org

OpenNebula 4.6 User Guide

Release 4.6

OpenNebula Project

May 09, 2014

CONTENTS

1 Virtual Resource Management **1**

1.1 Introduction to Private Cloud Computing 1
1.2 Managing Virtual Networks 4
1.3 Managing Images 12
1.4 Creating Virtual Machines 22
1.5 Managing Virtual Machines 27

2 Virtual Machine Setup **43**

2.1 Contextualization Overview 43
2.2 Adding Content to Your Cloud 43
2.3 Basic Contextualization 44
2.4 Advanced Contextualization 45
2.5 Windows Contextualization 51
2.6 Cloud-init 53

3 OpenNebula Marketplace **55**

3.1 Interacting with the OpenNebula Marketplace 55
3.2 Howto Create Apps for the Marketplace 59

4 References **63**

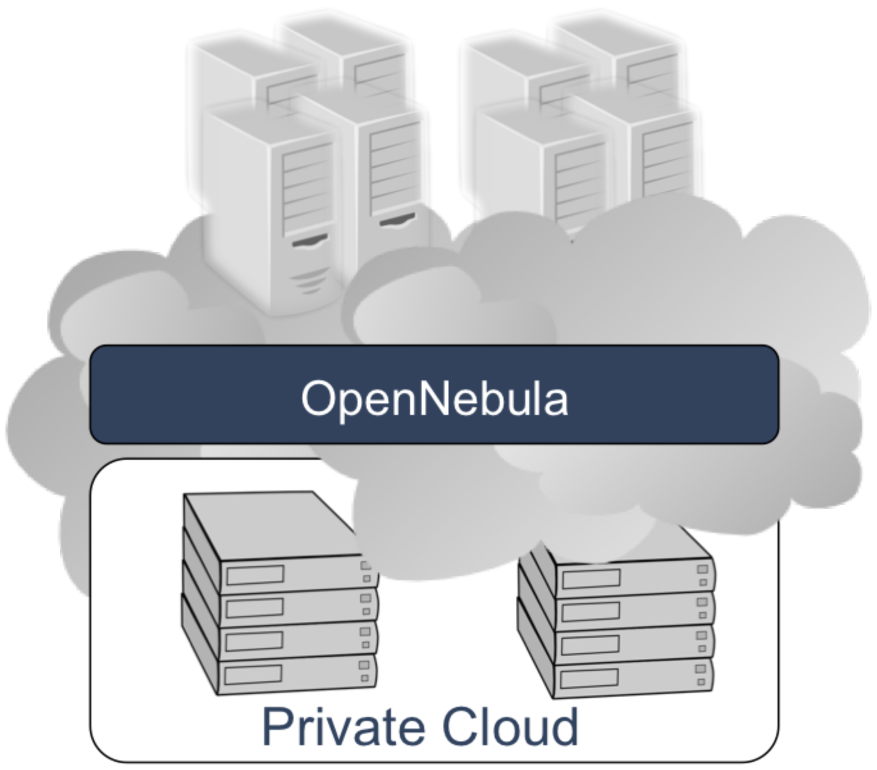
4.1 Virtual Machine Definition File 63
4.2 Image Definition Template 76
4.3 Virtual Network Definition File 79
4.4 Command Line Interface 81

CHAPTER

ONE

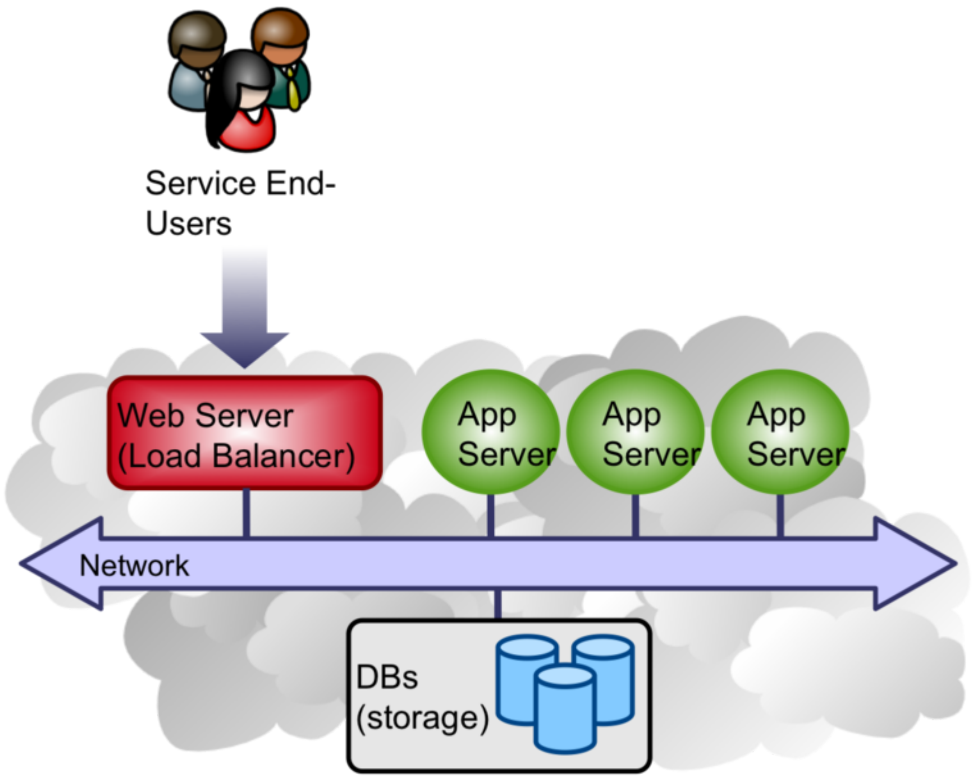
VIRTUAL RESOURCE MANAGEMENT

1.1 Introduction to Private Cloud Computing



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 cloud users and administrators 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**. This guide briefly describes how OpenNebula operates to build a Cloud infrastructure. After reading this guide you may be interested in reading the *guide describing how an hybrid cloud operates* and the *guide describing how a public cloud operates*.

1.1.1 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 *Sunstone* or any of the OpenNebula interfaces:

- *Command Line Interface (CLI)*
- *XML-RPC API*
- OpenNebula *Ruby* and *Java* Cloud APIs

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**:

```
$ onehost list
 ID NAME                  RVM    TCPU    FCPU    ACPU    TMEM    FMEM    AMEM    STAT
  0 host01                  0     800     800     800     16G     16G     16G      on
  1 host02                  0     800     800     800     16G     16G     16G      on
```

We can then **register an image** in OpenNebula, by using `oneimage`. We are going to build an *image template* to register the image file we had previously placed in the `/home/cloud/images` directory.

```
NAME          = Ubuntu
PATH          = /home/cloud/images/ubuntu-desktop/disk.0
PUBLIC        = YES
DESCRIPTION   = "Ubuntu 10.04 desktop for students."

$ oneimage create ubuntu.oneimg
ID: 0

$ oneimage list
```

```
  ID USER     GROUP    NAME            SIZE TYPE          REGTIME PUB PER STAT  RVMS
   1 oneadmin oneadmin Ubuntu           10G   OS   09/29 07:24:35 Yes  No  rdy     0
```

This image is now ready to be used in a virtual machine. We need to define a *virtual machine template* to be submitted using the `onetemplate` command.

```
NAME   = my_vm
CPU    = 1
MEMORY = 2056

DISK = [ IMAGE_ID  = 0 ]

DISK = [ type   = swap,
         size   = 1024 ]

NIC    = [ NETWORK_ID = 0 ]
```

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):

```
$ onetemplate create vm
ID: 0

$ onetemplate list
  ID USER     GROUP    NAME                         REGTIME PUB
   0 oneadmin oneadmin my_vm                 09/29 07:28:41  No
```

The listed template is just a VM definition. To execute an instance, we can use the onetemplate command again:

```
$ onetemplate instantiate 1
VM ID: 0
```

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

```
$ onevm list
    ID USER     GROUP    NAME         STAT CPU     MEM        HOSTNAME        TIME
     0 oneadmin oneadmin one-0        runn   0      0K          host01 00 00:00:06
```

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.

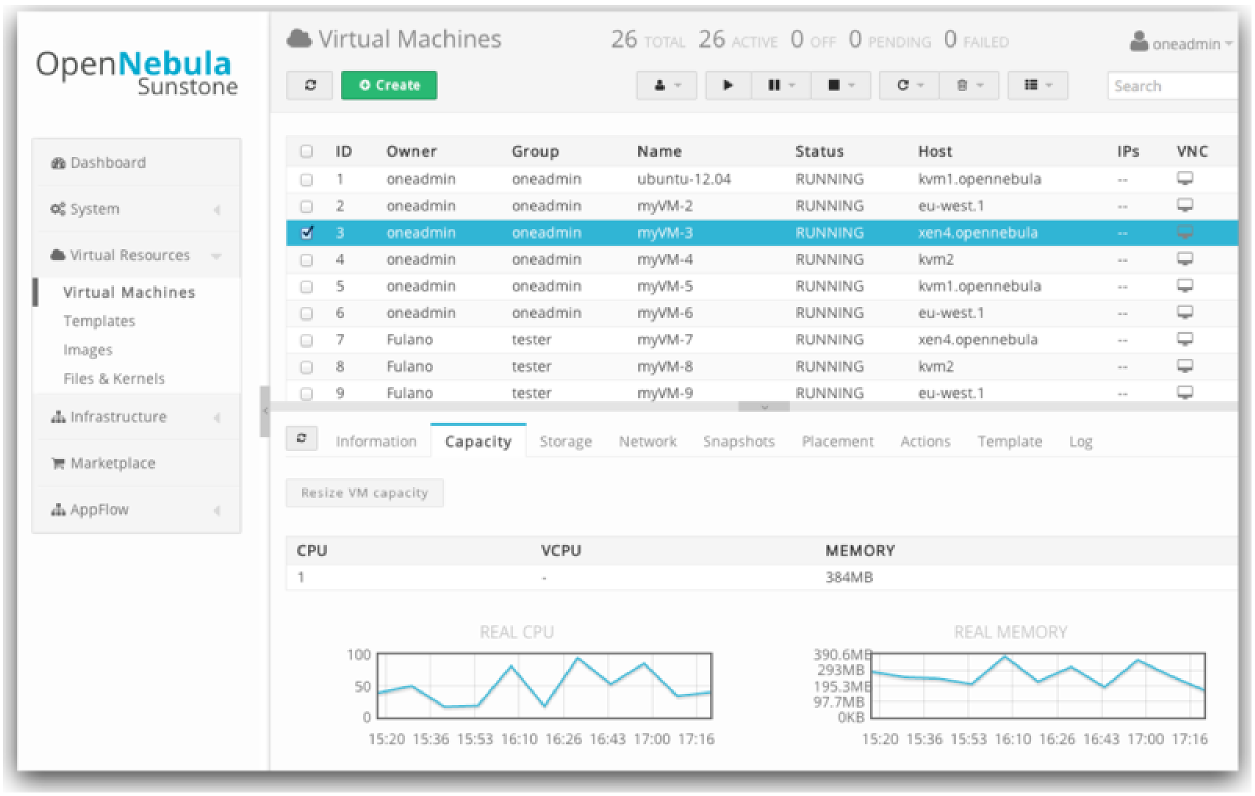
To **perform a migration**, we use yet again the `onevm` command. Let's move the VM (with VID=0) to *host02* (HID=1):

```
$ onevm migrate --live 0 1
```

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

```
$ onevm list
    ID USER     GROUP    NAME         STAT CPU     MEM        HOSTNAME        TIME
     0 oneadmin oneadmin one-0        runn   0      0K          host02 00 00:00:48
```

You can also reproduce this sample session using the graphical interface provided by *Sunstone*, that will simplify the typical management operations.



ID	Owner	Group	Name	Status	Host	IPs	VNC
1	oneadmin	oneadmin	ubuntu-12.04	RUNNING	kvm1.opennebula	--	
2	oneadmin	oneadmin	myVM-2	RUNNING	eu-west.1	--	
3	oneadmin	oneadmin	myVM-3	RUNNING	xen4.opennebula	--	
4	oneadmin	oneadmin	myVM-4	RUNNING	kvm2	--	
5	oneadmin	oneadmin	myVM-5	RUNNING	kvm1.opennebula	--	
6	oneadmin	oneadmin	myVM-6	RUNNING	eu-west.1	--	
7	Fulano	tester	myVM-7	RUNNING	xen4.opennebula	--	
8	Fulano	tester	myVM-8	RUNNING	kvm2	--	
9	Fulano	tester	myVM-9	RUNNING	eu-west.1	--	

CPU	VCPU	MEMORY
1	-	384MB

1.1.2 Next Steps

You can now read the different guides describing how to define and manage virtual resources on your OpenNebula cloud:

- *Virtual Networks*
- *Virtual Machine Images*
- *Virtual Machine Templates*
- *Virtual Machine Instances*

You can also install *OneFlow* to allows users and administrators to define, execute and manage multi-tiered applications composed of interconnected Virtual Machines with auto-scaling.

1.2 Managing Virtual Networks

A host is connected to one or more networks that are available to the virtual machines through the corresponding bridges. OpenNebula allows the creation of Virtual Networks by mapping them on top of the physical ones

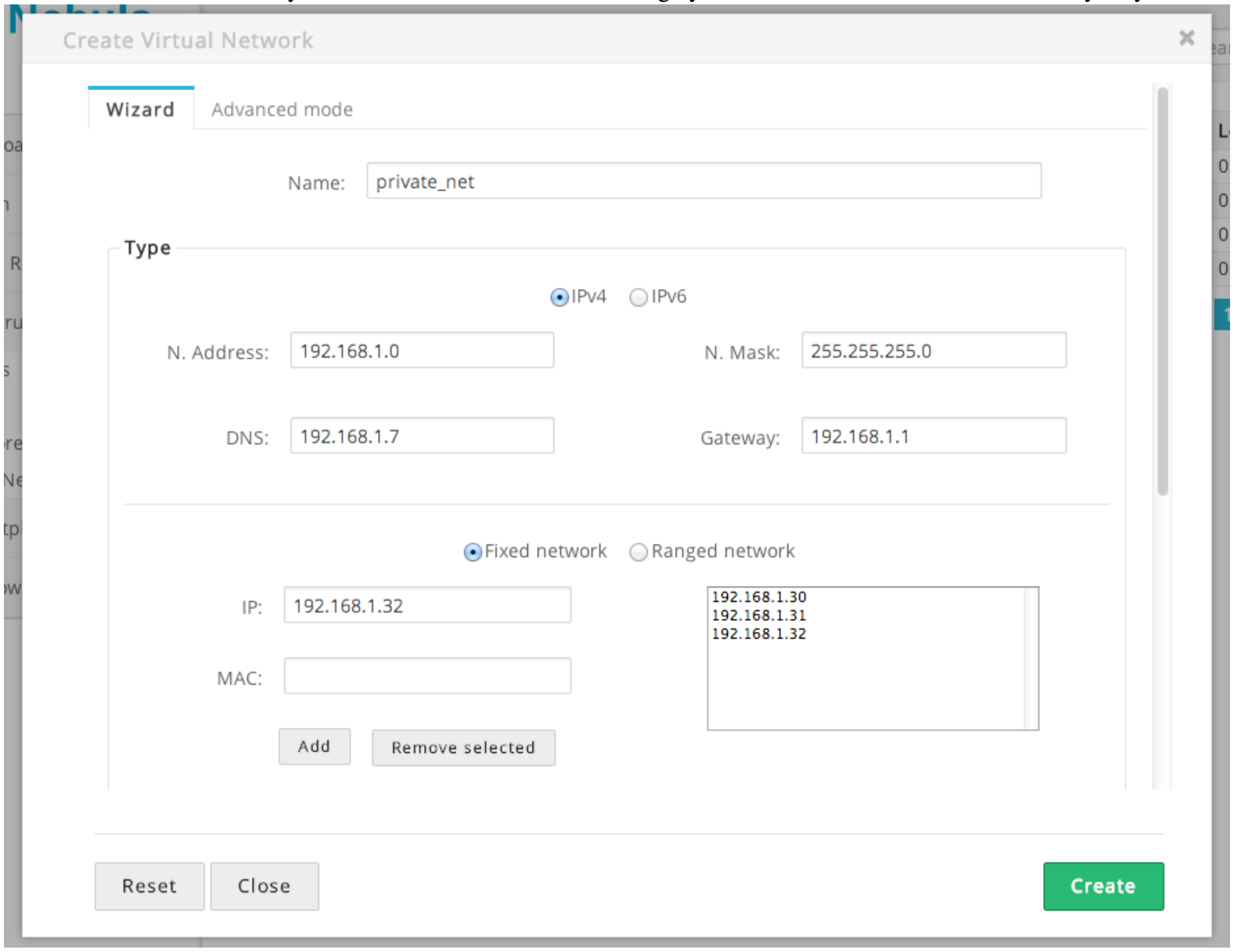
1.2.1 Overview

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

- A private network, through the virtual bridge vbr0

- A network with Internet connectivity, through vbr1

This guide uses the CLI command `onevnet`, but you can also manage your virtual networks using *Sunstone*. Select the Network tab, and there you will be able to create and manage your virtual networks in a user friendly way.



Name: private_net

Type

IPv4 [✓] IPv6

N. Address: 192.168.1.0

N. Mask: 255.255.255.0

DNS: 192.168.1.7

Gateway: 192.168.1.1

Fixed network [✓] Ranged network

IP: 192.168.1.32

MAC:

```
192.168.1.30
192.168.1.31
192.168.1.32
```

1.2.2 Adding, Deleting and Updating Virtual Networks

A virtual network is defined by two sets of options:

- The underlying networking parameters, e.g. BRIDGE, VLAN or PHY_DEV. These attributes depend on the networking technology (drivers) used by the hosts. Please refer to the specific networking guide.
- A set of Leases. A lease defines a MAC - IP pair, related as MAC = MAC_PREFFIX:IP. For IPv6 networks the only relevant part is the MAC address (see below).

Depending on how the lease set is defined the networks are:

- Fixed. A limited (possibly disjoint) set of leases, e.g: 10.0.0.1, 10.0.0.40 and 10.0.0.34
- Ranged. A continuous set of leases (like in a network way), e.g: 10.0.0.0/24

Please refer to the *Virtual Network template reference guide* for more information. The `onevnet` command is used to create a VNet from that template.

IPv4 Networks

IPv4 leases can be defined in several ways:

- Ranged. The ranged can be defined with:
  - A network address in CIDR format, e.g. NETWORK_ADDRESS=10.0.0.0/24.
  - A network address and a net mask, e.g. NETWORK_ADDRESS=10.0.0.0 NETWORK_MASK=255.255.255.0.
  - A network address and a size, e.g. NETWORK_ADDRESS=10.0.0.0, NETWORK_SIZE=C.
  - An arbitrary IP range, e.g. IP_START=10.0.0.1, IP_END=10.0.0.254.
- Fixed. Each lesae can be defined by:
  - An IP address, e.g. LEASE=[IP=10.0.0.1]
  - An IP address and a MAC to override the default MAC generation (MAC=PREFIX:IP), e.g. LEASE=[IP=10.0.0.1, MAC=e8:9d:87:8d:11:22]

As an example, we will create two new VNets, Blue and Red. Lets assume we have two files, `blue.net` and `red.net`.

Blue.net file:

```
NAME    = "Blue LAN"
TYPE    = FIXED

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

LEASES  = [IP=130.10.0.1]
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]

# Custom Attributes to be used in Context
GATEWAY = 130.10.0.1
DNS     = 130.10.0.1

LOAD_BALANCER = 130.10.0.4
```

And red.net file:

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

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

NETWORK_SIZE    = C
NETWORK_ADDRESS = 192.168.0.0

# Custom Attributes to be used in Context
GATEWAY = 192.168.0.1
DNS     = 192.168.0.1

LOAD_BALANCER = 192.168.0.3
```

Once the files have been created, we can create the VNets executing:

```
$ onevnet create blue.net
ID: 0
$ onevnet create red.net
ID: 1
```

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

```
$ onevnet list
  ID USER     GROUP    NAME            CLUSTER    TYPE BRIDGE  LEASES
   0 oneadmin oneadmin Blue LAN        -             F   vbr1       0
   1 oneadmin oneadmin Red LAN         -             R   vbr0       0
```

In the output above, `USER` is the owner of the network and `LEASES` the number of IP-MACs assigned to a VM from this network.

The following attributes can be changed after creating the network: `VLAN_ID`, `BRIDGE`, `VLAN` and `PHYDEV`. To update the network run `onevnet update <id>`.

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

```
$ onevnet delete 2
$ onevnet delete 'Red LAN'
```

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

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

IPv6 Networks

OpenNebula can generate three IPv6 addresses associated to each lease:

- Link local - fe80::/64 generated always for each lease as IP6_LINK
- Unique local address (ULA) - fd00::/8, generate if a local site prefix (SITE_PREFIX) is provided as part of the network template. The address is associated to the lease as IP6_SITE
- Global unicast address - if a global routing prefix (GLOBAL_PREFIX) is provided in the network template; available in the lease as IP6_GLOBAL

For all the previous addresses the lower 64 bits are populated with a 64-bit interface identifier in modified EUI-64 format. You do not need to define both SITE_PREFIX and GLOBAL_PREFIX , just the ones for the IP6 addresses needed by your VMs.

The IPv6 lease set can be generated as follows depending on the network type:

- Ranged. You will define a range of MAC addresses (that will be used to generate the EUI-64 host ID in the guest) with the first MAC and a size, e.g. MAC_START=e8:9d:87:8d:11:22 NETWORK_SIZE=254.
- Fixed. Just set the MACs for the network hosts as: LEASE=[MAC=e8:9d:87:8d:11:22] LEASE=[MAC=88:53:2e:08:7f:a0]

For example, the following template defines a ranged IPv6 network:

```
NAME = "Red LAN 6"
TYPE = RANGED

BRIDGE = vbr0

MAC_START    = 02:00:c0:a8:00:01
NETWORK_SIZE = C
```

```
SITE_PREFIX   = "fd12:33a:df34:1a::"
GLOBAL_PREFIX = "2004:a128::"
```

The IP leases are then in the form:

```
LEASE=[ MAC="02:00:c0:a8:00:01", IP="192.168.0.1", IP6_LINK="fe80::400:c0ff:fea8:1", IP6_SITE="fd12:3
```

Note that IPv4 addresses are generated from the MAC address in case you need to configure IPv4 and IPv6 addresses for the network.

1.2.3 Managing Virtual Networks

Adding and Removing Leases

You can add and remove leases to existing `FIXED` virtual networks (see the *template file reference* for more info on the network types). To do so, use the `onevnet addleases` and `onevnet rmleases` commands.

The new lease can be added specifying its IP and, optionally, its MAC. If the lease already exists, the action will fail.

```
$ onevnet addleases 0 130.10.0.10
$ onevnet addleases 0 130.10.0.11 50:20:20:20:20:31
$
$ onevnet addleases 0 130.10.0.1
[VirtualNetworkAddLeases] Error modifiying network leases. Error inserting lease,
IP 130.10.0.1 already exists
```

To remove existing leases from the network, they must be free (i.e., not used by any VM).

```
$ onevnet rmleases 0 130.10.0.3
```

Hold and Release Leases

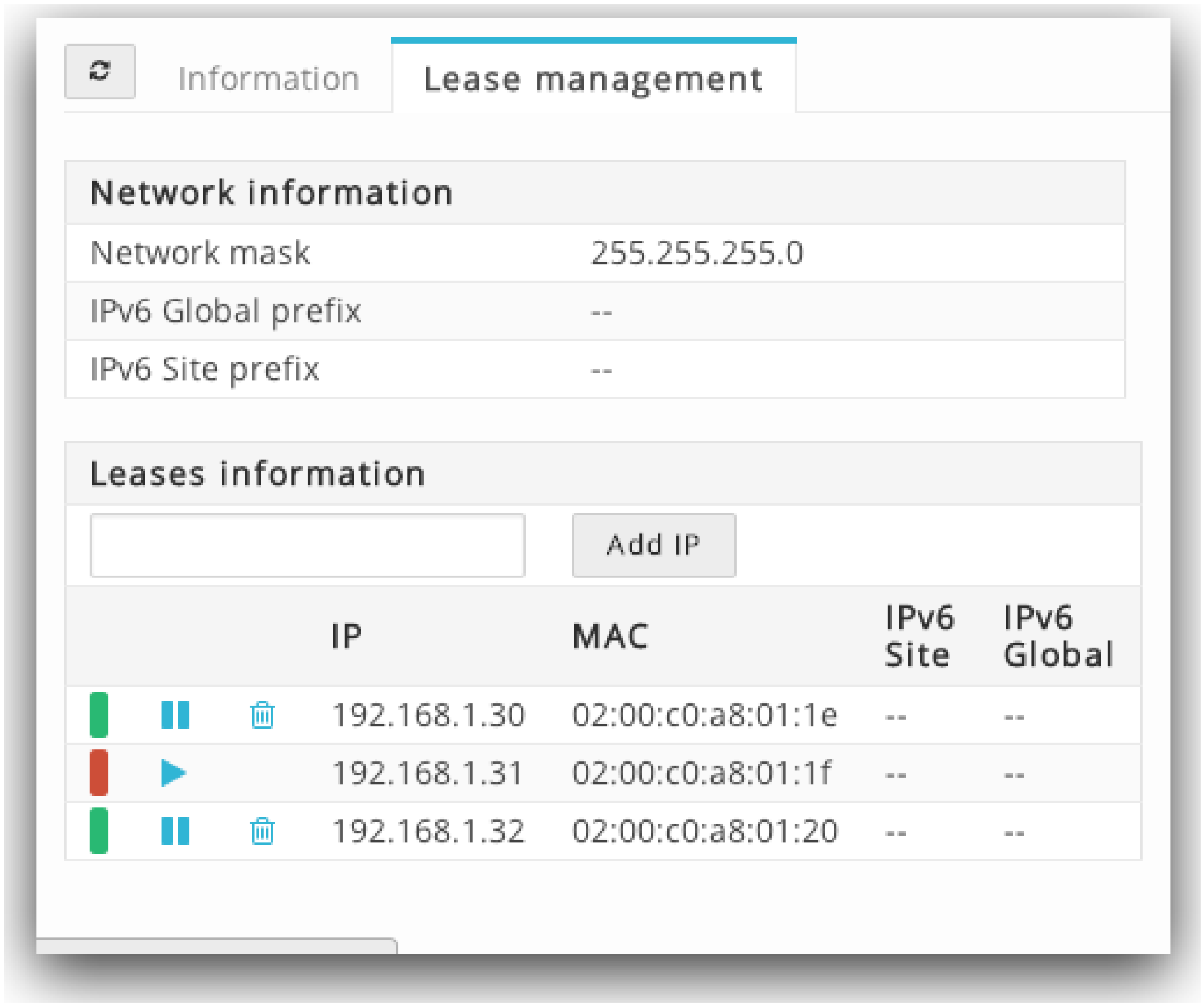
Leases can be temporarily be marked `on hold` state. These leases are reserved, they are part of the network, but they will not be assigned to any VM.

To do so, use the ‘onevnet hold’ and ‘onevnet release’ commands. You see the list of leases on hold with the ‘onevnet show’ command.

```
$ onevnet hold "Blue LAN" 130.10.0.1
$ onevnet hold 0 130.10.0.4
```

Lease Management in Sunstone

If you are using the Sunstone GUI, you can then easily add, remove, hold and release leases from the dialog of extended information of a Virtual Network. You can open this dialog by clicking the desired element on the Virtual Network table, as you can see in this picture:



Network information

Network mask	255.255.255.0
IPv6 Global prefix	--
IPv6 Site prefix	--

Leases information

			IP	MAC	IPv6 Site	IPv6 Global
🟩	❚❚	🗑	192.168.1.30	02:00:c0:a8:01:1e	--	--
🟥	▶		192.168.1.31	02:00:c0:a8:01:1f	--	--
🟩	❚❚	🗑	192.168.1.32	02:00:c0:a8:01:20	--	--

Update the Virtual Network Template

The `TEMPLATE` section can hold any arbitrary data. You can use the `onevnet update` command to open an editor and edit or add new template attributes. These attributes can be later used in the *Virtual Machine Contextualization*. For example:

```
dns = "$NETWORK[DNS, NETWORK_ID=3]"
```

Publishing Virtual Networks

The users can share their virtual networks with other users in their group, or with all the users in OpenNebula. See the *Managing Permissions documentation* for more information.

Let's see a quick example. To share the virtual network 0 with users in the group, the **USE** right bit for **GROUP** must be set with the **chmod** command:

```
$ onevnet show 0
...
PERMISSIONS
OWNER          : um-
```

```
GROUP          : ---
OTHER          : ---

$ onevnet chmod 0 640

$ onevnet show 0
...
PERMISSIONS
OWNER          : um-
GROUP          : u--
OTHER          : ---
```

The following command allows users in the same group **USE** and **MANAGE** the virtual network, and the rest of the users **USE** it:

```
$ onevnet chmod 0 664

$ onevnet show 0
...
PERMISSIONS
OWNER          : um-
GROUP          : um-
OTHER          : u--
```

The commands `onevnet publish` and `onevnet unpublish` are still present for compatibility with previous versions. These commands set/unset

1.2.4 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 `Blue LAN` just include in the template:

```
NIC = [ NETWORK_ID = 0 ]
NIC = [ NETWORK    = "Red LAN" ]
```

Networks can be referred in a NIC in two different ways, see the *Simplified Virtual Machine Definition File documentation* for more information:

- NETWORK_ID, using its ID as returned by the create operation
- NETWORK, using its name. In this case the name refers to one of the virtual networks owned by the user (names can not be repeated for the same user). If you want to refer to an NETWORK of other user you can specify that with NETWORK_UID (by the uid of the user) or NETWORK_UNAME (by the name of the user).

You can also request a specific address just by adding the `IP` attributes to `NIC` (or `MAC` address, specially in a IPv6):

```
NIC = [ NETWORK_ID = 1, IP = 192.168.0.3 ]
```

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

```
$ onevm show 0
VIRTUAL MACHINE 0 INFORMATION
ID                  : 0
NAME                : server
USER                : oneadmin
```

```
GROUP               : oneadmin
STATE               : PENDING
LCM_STATE           : LCM_INIT
START TIME          : 12/13 06:59:07
END TIME            : -
DEPLOY ID           : -

PERMISSIONS
OWNER          : um-
GROUP          : ---
OTHER          : ---

VIRTUAL MACHINE MONITORING
NET_TX              : 0
NET_RX              : 0
USED MEMORY         : 0
USED CPU            : 0

VIRTUAL MACHINE TEMPLATE
NAME=server
NIC=[
  BRIDGE=vbr1,
  IP=130.10.0.2,
  MAC=02:00:87:8d:11:25,
  IP6_LINK=fe80::400:87ff:fe8d:1125
  NETWORK="Blue LAN",
  NETWORK_ID=0,
  VLAN=NO ]
NIC=[
  BRIDGE=vbr0,
  IP=192.168.0.2,
  IP6_LINK=fe80::400:c0ff:fea8:2,
  MAC=00:03:c0:a8:00:02,
  NETWORK="Red LAN",
  NETWORK_ID=1,
  VLAN=NO ]
VMID=0
```

Warning: 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 VNet information:

```
$ onevnet list
  ID USER     GROUP    NAME             CLUSTER    TYPE BRIDGE  LEASES
   0 oneadmin oneadmin Blue LAN         -             F   vbr1       3
   1 oneadmin oneadmin Red LAN          -             R   vbr0       3
```

Note that there are two LEASES on hold, and one LEASE used in each network

```
$ onevnet show 1
VIRTUAL NETWORK 1 INFORMATION
ID             : 1
NAME           : Red LAN
USER           : oneadmin
GROUP          : oneadmin
TYPE           : RANGED
BRIDGE         : vbr0
VLAN           : No
```

```
PHYSICAL DEVICE:
VLAN ID        :
USED LEASES    : 3

PERMISSIONS
OWNER          : um-
GROUP          : ---
OTHER          : ---

VIRTUAL NETWORK TEMPLATE
DNS=192.168.0.1
GATEWAY=192.168.0.1
LOAD_BALANCER=192.168.0.3
NETWORK_MASK=255.255.255.0

RANGE
IP_START       : 192.168.0.1
IP_END         : 192.168.0.254

LEASES ON HOLD
LEASE=[ MAC="02:00:c0:a8:00:01", IP="192.168.0.1", IP6_LINK="fe80::400:c0ff:fea8:1", USED="1", VID="-
LEASE=[ MAC="02:00:c0:a8:00:03", IP="192.168.0.3", IP6_LINK="fe80::400:c0ff:fea8:3", USED="1", VID="-

USED LEASES

LEASE=[ MAC="02:00:c0:a8:00:02", IP="192.168.0.2", IP6_LINK="fe80::400:c0ff:fea8:2", USED="1", VID="4
```

Warning: IP 192.168.0.2 is in use by Virtual Machine 4

Apply Firewall Rules to VMs

You can apply firewall rules on your VMs, to filter TCP and UDP ports, and to define a policy for ICMP connections.

Read more about this feature *here*.

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.

In order to configure the IP inside the guest, you need to use one of the two available methods:

- Instantiate a *Virtual Router* inside each Virtual Network. The Virtual Router appliance contains a DHCP server that knows the IP assigned to each VM.
- Contextualize the VM. Please visit the *contextualization guide* to learn how to configure your Virtual Machines to automatically obtain an IP derived from the MAC.

1.3 Managing Images

The *Storage system* allows OpenNebula administrators and users to set up images, which can be operative systems or data, to be used in Virtual Machines easily. These images can be used by several Virtual Machines simultaneously, and also shared with other users.

If you want to customize the Storage in your system, visit the *Storage subsystem guide*.

1.3.1 Image Types

There are six different types of images. Using the command `oneimage chtype`, you can change the type of an existing Image.

- **OS**: An OS image contains a working operative system. Every *VM template* must define one DISK referring to an image of this type.
- **CDROM**: These images are readonly data. Only one image of this type can be used in each *VM template*. These type of images are not cloned when using shared storage.
- **DATABLOCK**: A datablock image is a storage for data, which can be accessed and modified from different Virtual Machines. These images can be created from previous existing data, or as an empty drive.
- **KERNEL**: A plain file to be used as kernel (VM attribute OS/KERNEL_DS). Note that KERNEL file images can be registered only in File Datastores.
- **RAMDISK**: A plain file to be used as ramdisk (VM attribute OS/INITRD_DS). Note that RAMDISK file images can be registered only in File Datastores.
- **CONTEXT**: A plain file to be included in the context CD-ROM (VM attribute CONTEXT/FILES_DS). Note that CONTEXT file images can be registered only in File Datastores.

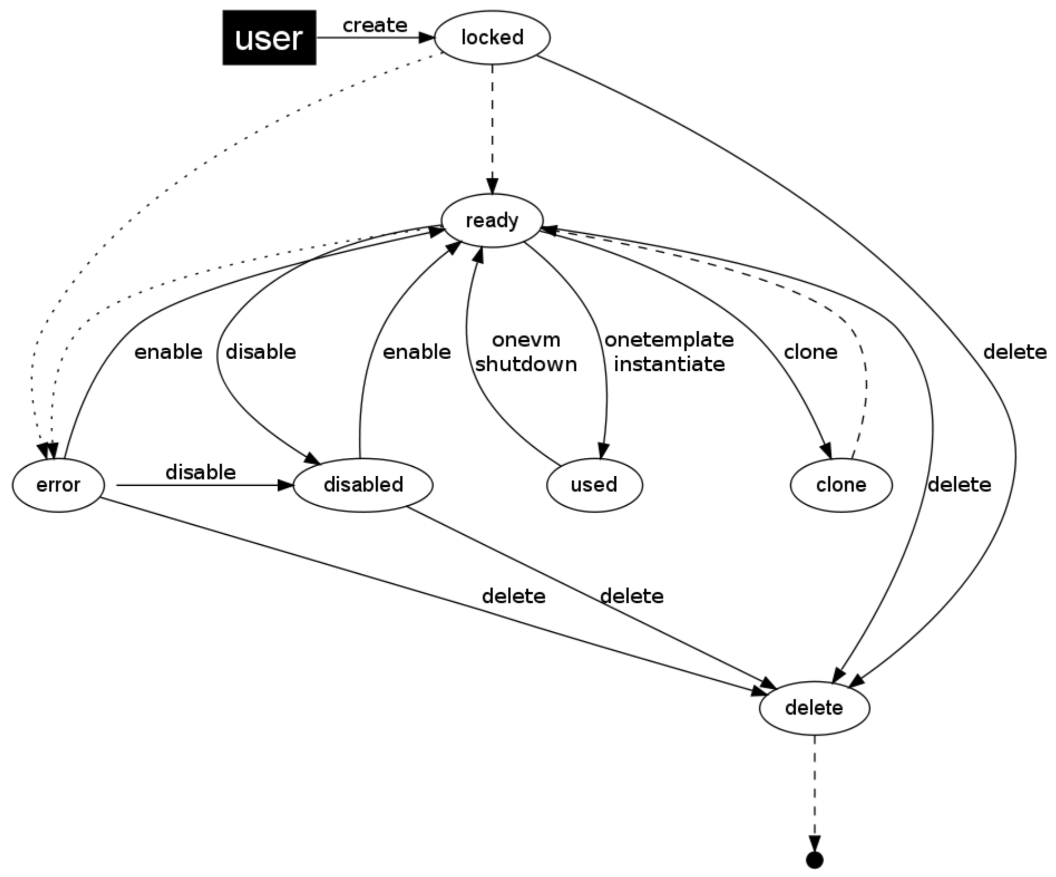
The Virtual Machines can use as many datablocks as needed. Refer to the *VM template* documentation for further information.

Warning: Note that some of the operations described below do not apply to KERNEL, RAMDISK and CONTEXT images, in particular: clone and persistent.

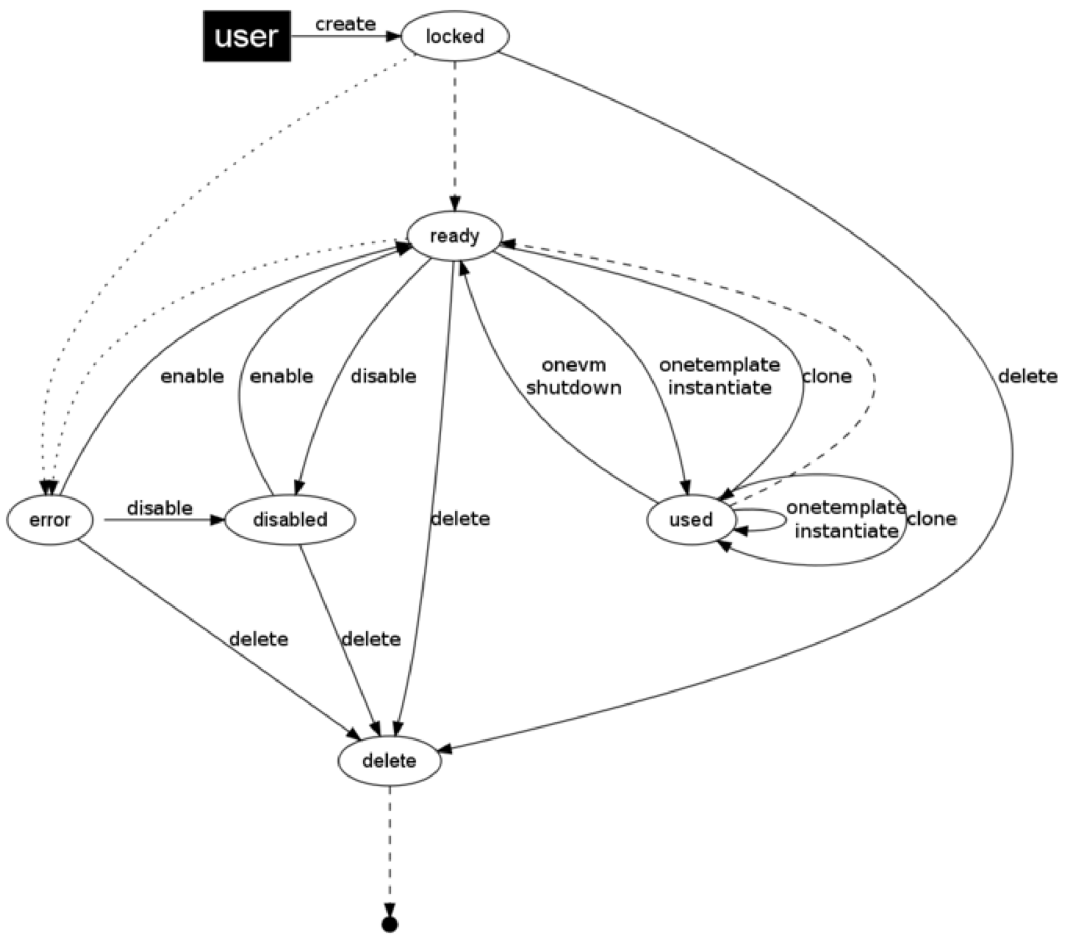
1.3.2 Image Life-cycle

Short state	State	Meaning
`lock`	`LOCKED`	The image file is being copied or created in the Datastore.
`rdy`	`READY`	Image ready to be used.
`used`	`USED`	Non-persistent Image used by at least one VM. It can still be used by other VMs.
`used`	`USED_PERS`	Persistent Image is use by a VM. It cannot be used by new VMs.
`disa`	`DISABLED`	Image disabled by the owner, it cannot be used by new VMs.
`err`	`ERROR`	Error state, a FS operation failed. See the Image information with `oneimage show` for an error message.
`dele`	`DELETE`	The image is being deleted from the Datastore.

This is the state diagram for **persistent** images:



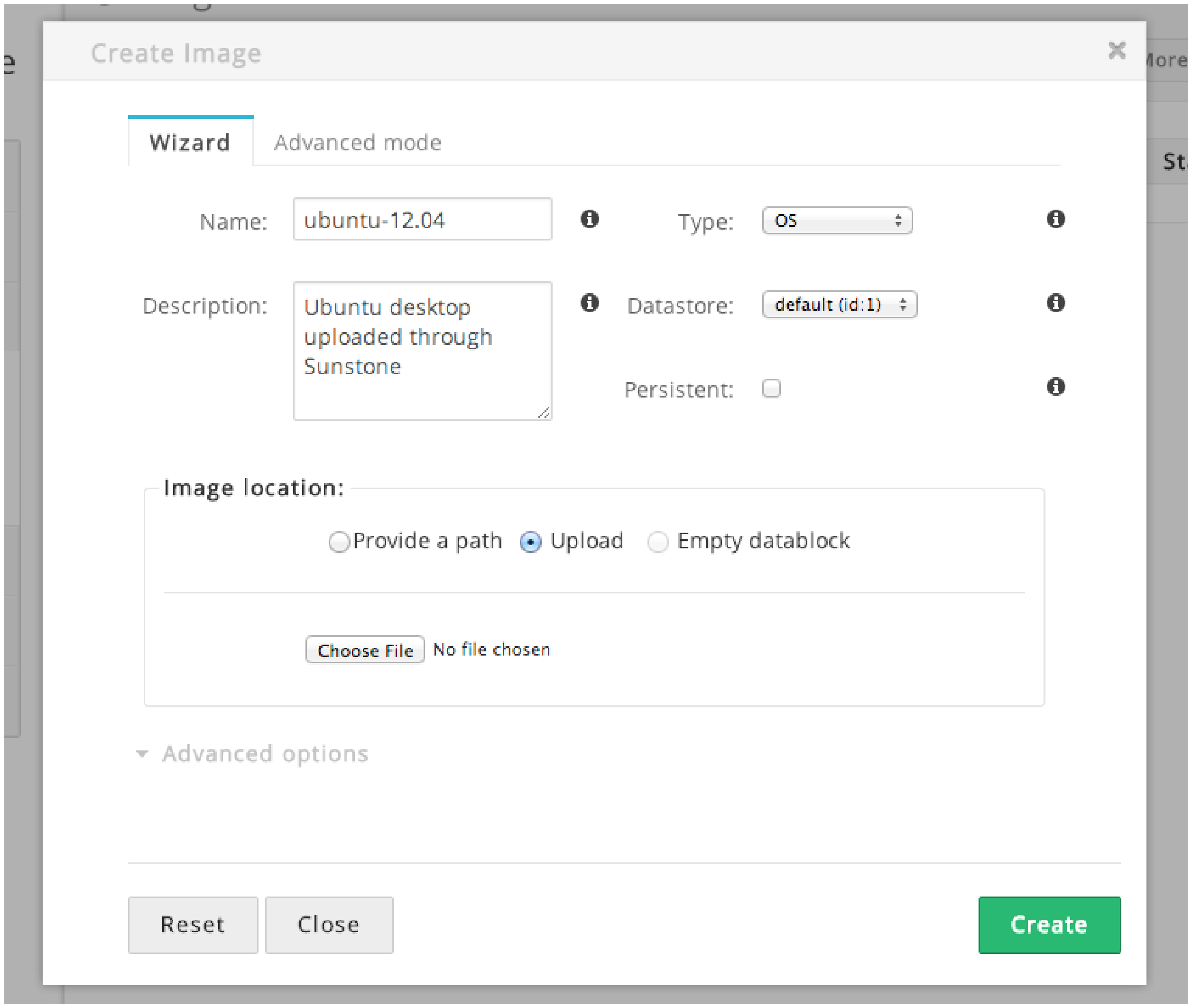
And the following one is the state diagram for **non-persistent** images:



1.3.3 Managing Images

Users can manage their images using the command line interface command `oneimage`. The complete reference is *here*.

You can also manage your images using *Sunstone*. Select the Images tab, and there you will be able to create, enable, disable, delete your images and even manage their persistence and publicity in a user friendly way. From Sunstone 3.4, you can also upload images directly from the web UI.



Name: ubuntu-12.04

Type: OS

Description: Ubuntu desktop uploaded through Sunstone

Datastore: default (id:1)

Persistent: ☐

Image location: Provide a path [✓ Upload] Empty datablock

Choose File No file chosen

Create Images

Warning: For VMWare images, please read **also** the *VMware Drivers guide*.

The three types of images can be created from an existing file, but for **datablock** images you can specify a size and filesystem type and let OpenNebula create an empty image in the datastore.

If you want to create an **OS image**, you need to prepare a contextualized virtual machine, and extract its disk.

Please read first the documentation about the MAC to IP mechanism in the *virtual network management documentation*, and how to use contextualization *here*.

Once you have a disk you want to upload, you need to create a new *image template*, and submit it using the `oneimage create` command.

The complete reference for the image template is *here*. This is how a sample template looks like:

```
$ cat ubuntu_img.one
NAME          = "Ubuntu"
PATH          = /home/cloud/images/ubuntu-desktop/disk.0
```

```
TYPE          = OS
DESCRIPTION   = "Ubuntu 10.04 desktop for students."
```

You need to choose the Datastore where to register the new Image. To know the available datastores, use the `onedatastore list` command. In this case, only the 'default' one is listed:

```
$ onedatastore list
  ID NAME            CLUSTER  IMAGES TYPE   TM
   1 default         -        1      fs     shared
```

To submit the template, you just have to issue the command

```
$ oneimage create ubuntu_img.one --datastore default
ID: 0
```

You can also create images using just parameters in the `oneimage create` call. The parameters to generate the image are as follows:

Parameter	Description
`-name name`	Name of the new image
`-description description`	Description for the new Image
`-type type`	Type of the new Image: OS, CDROM or DATABLOCK, FILE
`-persistent`	Tells if the image will be persistent
`-prefix prefix`	Device prefix for the disk (eg. hd, sd, xvd or vd)
`-target target`	Device the disk will be attached to
`-path path`	Path of the image file
`-driver driver`	Driver to use image (raw, qcow2, tap:aio:...)
`-disk_type disk_type`	Type of the image (BLOCK, CDROM or FILE)
`-source source`	Source to be used. Useful for not file-based images
`-size size`	Size in MB. Used for DATABLOCK type
`-fstype fstype`	Type of file system to be built: ext2, ext3, ext4, ntfs, reiserfs, jfs, swap, qcow2

To create the previous example image you can do it like this:

```
$ oneimage create --datastore default --name Ubuntu --path /home/cloud/images/ubuntu-desktop/disk.0
  --description "Ubuntu 10.04 desktop for students."
```

Warning: You can use **gz** compressed image files (i.e. as specified in path) when registering them in OpenNebula.

Uploading Images from Sunstone

Image file upload to the server via the client browser is possible with the help of a vendor library. The process is as follow:

- Step 1: The client uploads the whole image to the server in a temporal file in the `tpmdir` folder specified in the configuration.
- Step 2: OpenNebula registers an image setting the PATH to that temporal file.
- Step 3: OpenNebula copies the images to the datastore.
- Step 4: The temporal file is deleted and the request returns successfully to the user (a message pops up indicating that image was uploaded correctly).

Note that when file sizes become big (normally over 1GB), and depending on your hardware, it may take long to complete the copying in step 3. Since the upload request needs to stay pending until copying is sucessful (so it can delete the temp file safely), there might be Ajax timeouts and/or lack of response from the server. This may cause errors, or trigger re-uploads (which reinitiate the loading progress bar).

As of Firefox 11 and previous versions, uploads seem to be limited to 2GB. Chrome seems to work well with images > 4 GB.

Clone Images

Existing images can be cloned to a new one. This is useful to make a backup of an Image before you modify it, or to get a private persistent copy of an image shared by other user.

To clone an image, execute

```
$ oneimage clone Ubuntu new_image
```

Listing Available Images

You can use the `oneimage list` command to check the available images in the repository.

```
$ oneimage list
  ID USER     GROUP    NAME         DATASTORE     SIZE TYPE PER STAT  RVMS
   0 oneuser1 users    Ubuntu       default         8M   OS  No  rdy     0
```

To get complete information about an image, use `oneimage show`, or list images continuously with `oneimage top`.

Publishing Images

The users can share their images with other users in their group, or with all the users in OpenNebula. See the *Managing Permissions documentation* for more information.

Let's see a quick example. To share the image 0 with users in the group, the **USE** right bit for **GROUP** must be set with the **chmod** command:

```
$ oneimage show 0
...
PERMISSIONS
OWNER          : um-
GROUP          : ---
OTHER          : ---

$ oneimage chmod 0 640

$ oneimage show 0
...
PERMISSIONS
OWNER          : um-
GROUP          : u--
OTHER          : ---
```

The following command allows users in the same group **USE** and **MANAGE** the image, and the rest of the users **USE** it:

```
$ oneimage chmod 0 664

$ oneimage show 0
```

```
...
PERMISSIONS
OWNER           : um-
GROUP           : um-
OTHER           : u--
```

The commands `oneimage publish` and `oneimage unpublish` are still present for compatibility with previous versions. These commands set/unset the GROUP USE bit.

Making Images Persistent

Use the `oneimage persistent` and `oneimage nonpersistent` commands to make your images persistent or not.

A persistent image saves back to the datastore the changes made inside the VM after it is shut down. More specifically, the changes are correctly preserved only if the VM is ended with the `onevm shutdown` or `onevm shutdown --hard` commands. Note that depending on the Datastore type a persistent image can be a link to the original image, so any modification is directly made on the image.

```
$ oneimage list
  ID USER     GROUP    NAME          DATASTORE     SIZE TYPE PER STAT  RVMS
   0 oneadmin oneadmin Ubuntu        default        10G   OS  No  rdy     0
$ oneimage persistent Ubuntu
$ oneimage list
  ID USER     GROUP    NAME          DATASTORE     SIZE TYPE PER STAT  RVMS
   0 oneadmin oneadmin Ubuntu        default        10G   OS Yes  rdy     0
$ oneimage nonpersistent 0
$ oneimage list
  ID USER     GROUP    NAME          DATASTORE     SIZE TYPE PER STAT  RVMS
   0 oneadmin oneadmin Ubuntu        default        10G   OS  No  rdy     0
```

Warning: When images are public (GROUP or OTHER USE bit set) they are always cloned, and persistent images are never cloned. Therefore, an image cannot be public and persistent at the same time. To manage a public image that won't be cloned, unpublish it first and make it persistent.

1.3.4 How to Use Images in Virtual Machines

This a simple example on how to specify images as virtual machine disks. Please visit the *virtual machine user guide* and the *virtual machine template* documentation for a more thorough explanation.

Assuming you have an OS image called *Ubuntu desktop* with ID 1, you can use it in your *virtual machine template* as a DISK. When this machine is deployed, the first disk will be taken from the image repository.

Images can be referred in a DISK in two different ways:

- IMAGE_ID, using its ID as returned by the create operation
- IMAGE, using its name. In this case the name refers to one of the images owned by the user (names can not be repeated for the same user). If you want to refer to an IMAGE of other user you can specify that with IMAGE_UID (by the uid of the user) or IMAGE_UNAME (by the name of the user).

```
CPU    = 1
MEMORY = 3.08

DISK = [ IMAGE_ID   = 1 ]
DISK = [ type   = swap,
         size   = 1024  ]

NIC    = [ NETWORK_ID = 1 ]
NIC    = [ NETWORK_ID = 0 ]

# FEATURES=[ acpi="no" ]

GRAPHICS = [
  type    = "vnc",
  listen  = "1.2.3.4",
  port    = "5902"  ]


CONTEXT = [
    files      = "/home/cloud/images/ubuntu-desktop/init.sh"  ]
```

Save Changes

Once the VM is deployed you can snapshot a disk, i.e. save the changes made to the disk as a new image. There are two types of disk snapshots in OpenNebula:

- **Deferred snapshots** (disk-snapshot), changes to a disk will be saved as a new Image in the associated datastore when the VM is shutdown.
- **Hot snapshots** (hot disk-snapshot), just as the deferred snapshots, but the disk is copied to the datastore the moment the operation is triggered. Therefore, you must guarantee that the disk is in a consistent state during the save_as operation (e.g. by umounting the disk from the VM).

To save a disk, use the `onevm disk-snapshot` command. This command takes three arguments: The VM name (or ID), the disk ID to save and the name of the new image to register. And optionally the –live argument to not defer the disk-snapshot operation.

To know the ID of the disk you want to save, just take a look at the `onevm show` output for your VM, you are interested in the ID column in the VM DISK section.

```
$ onevm show 11
VIRTUAL MACHINE 11 INFORMATION
ID                  : 11
NAME                : ttylinux-11
USER                : ruben
GROUP               : oneadmin
STATE               : PENDING
LCM_STATE           : LCM_INIT
RESCHED             : No
START TIME          : 03/08 22:24:57
END TIME            : -
DEPLOY ID           : -

VIRTUAL MACHINE MONITORING
USED MEMORY         : 0K
USED CPU            : 0
NET_TX              : 0K
NET_RX              : 0K

PERMISSIONS
OWNER               : um-
GROUP               : ---
```

```
OTHER                : ---

VM DISKS
 ID TARGET IMAGE                               TYPE SAVE SAVE_AS
  0    hda ttylinux                            file   NO       -
  1    hdb raw - 100M                          fs     NO       -

VM NICS
ID NETWORK      VLAN BRIDGE   IP               MAC
 0 net_172        no vbr0     172.16.0.201     02:00:ac:10:00:c9
                              fe80::400:acff:fe10:c9

VIRTUAL MACHINE TEMPLATE
CPU="1"
GRAPHICS=[
  LISTEN="0.0.0.0",
  PORT="5911",
  TYPE="vnc" ]
MEMORY="512"
OS=[
  ARCH="x86_64" ]
TEMPLATE_ID="0"
VCPU="1"
```

The IDs are assigned in the same order the disks were defined in the *VM template*.

The next command will register a new image called *SO upgrade*, that will be ready as soon as the VM is shut down. Till then the image will be locked, and so you cannot use it.

```
$ onevm disk-snapshot ttylinux-11 0 "SO upgraded"
```

This command copies disk 1 to the datastore with name *Backup of DB volume*, the image will be available once the image copy end:

```
$ onevm disk-snapshot --live ttylinux-11 1 "Backup of DB volume"
```

1.3.5 How to Use File Images in Virtual Machines

KERNEL and RAMDISK

KERNEL and RAMDISK type Images can be used in the OS/KERNEL_DS and OS/INITRD_DS attributes of the VM template. See the *complete reference* for more information.

Example:

```
OS = [ KERNEL_DS  = "$FILE[IMAGE=kernel3.6]",
       INITRD_DS  = "$FILE[IMAGE_ID=23]",
       ROOT       = "sda1",
       KERNEL_CMD = "ro xencons=tty console=tty1" ]
```

CONTEXT

The *contextualization cdrom* can include CONTEXT type Images. Visit the *complete reference* for more information.

```
CONTEXT = [
  FILES_DS    = "$FILE[IMAGE_ID=34] $FILE[IMAGE=kernel]",
]
```

1.4 Creating Virtual Machines

In OpenNebula the Virtual Machines are defined with Template files. This guide explains **how to describe the wanted-to-be-ran Virtual Machine, and how users typically interact with the system**.

The Template Repository system allows OpenNebula administrators and users to register Virtual Machine definitions in the system, to be instantiated later as Virtual Machine instances. These Templates can be instantiated several times, and also shared with other users.

1.4.1 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
- 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 kernel and context information to be used inside the VM, are specified in a template file.

1.4.2 Defining a VM in 3 Steps

Virtual Machines are defined in an OpenNebula Template. Templates are stored in a repository to easily browse and instantiate VMs from them. To create a new Template you have to define 3 things

- **Capacity & Name**, how big will the VM be?

Attribute	Description	Mandatory	Default
`NAME`	Name that the VM will get for description purposes.	Yes	one-<vmid>
`MEMORY`	Amount of RAM required for the VM, in Megabytes.	Yes	
`CPU`	CPU ratio (e..g half a physical CPU is 0.5).	Yes	
`VCPU`	Number of virtual cpus.	No	1

- **Disks**. Each disk is defined with a DISK attribute. A VM can use three types of disk:
  - **Use a persistent Image** changes to the disk image will persist after the VM is shutdown.
  - **Use a non-persistent Image** images are cloned, changes to the image will be lost.
  - **Volatile** disks are created on the fly on the target host. After the VM is shutdown the disk is disposed.
- **Persistent and Clone Disks**

Attribute	Description	Mandatory	Default
`IMAGE_ID` and `IMAGE`	The ID or Name of the image in the datastore	Yes	
`IMAGE_UID`	Select the IMAGE of a given user by her ID	No	self
`IMAGE_UNAME`	Select the IMAGE of a given user by her NAME	No	self

- **Volatile**

Attribute	Description	Mandatory	Default
TYPE	Type of the disk: swap, fs. swap type will set the label to swap so it is easier to mount and the context packages will automatically mount it.	Yes	
SIZE	size in MB	Yes	
FORMAT	filesystem for **fs** images: ext2, ext3, etc. raw will not format the image. For VMs to run on vmfs or vmware shared configurations, the valid values are: vmdk_thin, vmdk_zeroedthick, vmdk_eagerzeroedthick	Yes	

- **Network Interfaces**. Each network interface of a VM is defined with the NIC attribute.

Attribute	Description	Mandatory	Default
NETWORK_ID and NETWORK	The ID or Name of the image in the datastore	Yes	
NETWORK_UID	Select the IMAGE of a given user by her ID	No	self
NETWORK_UNAME	Select the IMAGE of a given user by her NAME	No	self

The following example shows a VM Template file with a couple of disks and a network interface, also a VNC section was added.

```
NAME   = test-vm
MEMORY = 128
CPU    = 1

DISK = [ IMAGE  = "Arch Linux" ]
DISK = [ TYPE     = swap,
         SIZE     = 1024 ]

NIC = [ NETWORK = "Public", NETWORK_UNAME="oneadmin" ]

GRAPHICS = [
  TYPE    = "vnc",
  LISTEN  = "0.0.0.0"]
```

Simple templates can be also created using the command line instead of creating a template file. The parameters to do this for onetemplate are:

Parameter	Description
-name name	Name for the VM
-cpu cpu	CPU percentage reserved for the VM (1=100% one CPU)
-vcpu vcpu	Number of virtualized CPUs
-arch arch	Architecture of the VM, e.g.: i386 or x86_64
-memory memory	Memory ammount given to the VM
-disk disk0,disk1	Disks to attach. To use a disk owned by other user use user[disk]
-nic vnet0,vnet1	Networks to attach. To use a network owned by other user use user[network]
-raw string	Raw string to add to the template. Not to be confused with the RAW attribute. If you want to provide more than one element, just include an enter inside quotes, instead of using more than one -raw option
-vnc	Add VNC server to the VM
-ssh [file]	Add an ssh public key to the context. If the file is omited then the user variable SSH_PUBLIC_KEY will be used.
-net_context	Add network contextualization parameters
-context line1,line2	Lines to add to the context section
-boot device	Select boot device (hd, fd, cdrom or network)

A similar template as the previous example can be created with the following command:

```
$ onetemplate create --name test-vm --memory 128 --cpu 1 --disk "Arch Linux" --nic Public
```

> **Warning:** You may want to add VNC access, input hw or change the default targets of the disks. Check the *VM definition file for a complete reference*

> **Warning:** OpenNebula Templates are designed to be hypervisor-agnostic, but there are additional attributes that are supported for each hypervisor. Check the *Xen*, *KVM* and *VMware* configuration guides for more details

> **Warning:** Volatile disks can not be saved as. Pre-register a DataBlock image if you need to attach arbitrary volumes to the VM

1.4.3 Managing Templates

Users can manage the Template Repository using the command `onetemplate`, or the graphical interface *Sunstone*. For each user, the actual list of templates available are determined by the ownership and permissions of the templates.

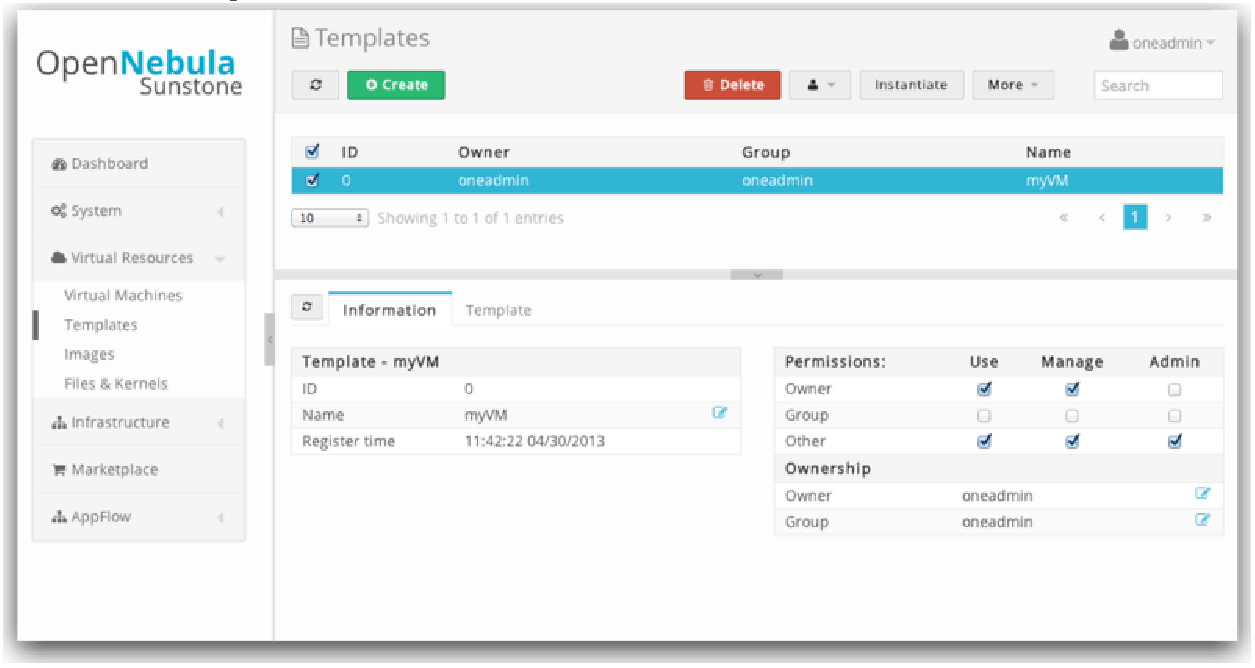
Listing Available Templates

You can use the `onetemplate list` command to check the available Templates in the system.

```
$ onetemplate list a
  ID USER     GROUP    NAME                        REGTIME
   0 oneadmin oneadmin template-0             09/27 09:37:00
   1 oneuser  users    template-1             09/27 09:37:19
   2 oneadmin oneadmin Ubuntu_server          09/27 09:37:42
```

To get complete information about a Template, use `onetemplate show`.

Here is a view of templates tab in Sunstone:



ID	Owner	Group	Name
0	oneadmin	oneadmin	myVM

10 Showing 1 to 1 of 1 entries

Template - myVM	
ID	0
Name	myVM
Register time	11:42:22 04/30/2013

Permissions:	Use	Manage	Admin
Owner	✓	✓	☐
Group	☐	☐	☐
Other	✓	✓	✓

Ownership	
Owner	oneadmin
Group	oneadmin

Adding and Deleting Templates

Using `onetemplate create`, users can create new Templates for private or shared use. The `onetemplate delete` command allows the Template owner -or the OpenNebula administrator- to delete it from the repository.

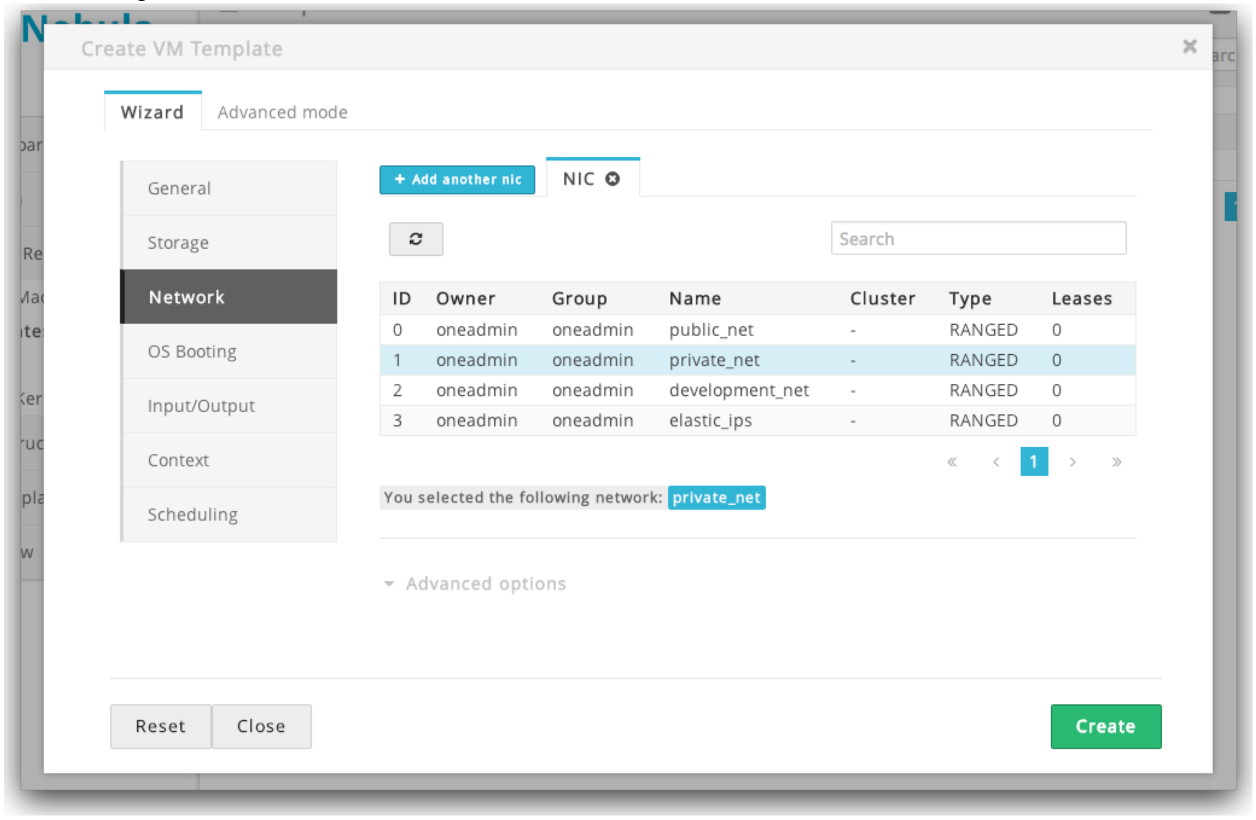
For instance, if the previous example template is written in the vm-example.txt file:

```
$ onetemplate create vm-example.txt
ID: 6
```

You can also clone an existing Template, with the `onetemplate clone` command:

```
$ onetemplate clone 6 new_template
ID: 7
```

Via Sunstone, you can easily add templates using the provided wizards (or copy/pasting a template file) and delete them clicking on the delete button:



ID	Owner	Group	Name	Cluster	Type	Leases
0	oneadmin	oneadmin	public_net	-	RANGED	0
1	oneadmin	oneadmin	private_net	-	RANGED	0
2	oneadmin	oneadmin	development_net	-	RANGED	0
3	oneadmin	oneadmin	elastic_ips	-	RANGED	0

You selected the following network: private_net

Updating a Template

It is possible to update a template by using the `onetemplate update`. This will launch the editor defined in the variable `EDITOR` and let you edit the template.

```
$ onetemplate update 3
```

Publishing Templates

The users can share their Templates with other users in their group, or with all the users in OpenNebula. See the *Managing Permissions documentation* for more information.

Let's see a quick example. To share the Template 0 with users in the group, the **USE** right bit for **GROUP** must be set with the **chmod** command:

```
$ onetemplate show 0
...
PERMISSIONS
OWNER          : um-
GROUP          : ---
OTHER          : ---

$ onetemplate chmod 0 640

$ onetemplate show 0
...
PERMISSIONS
OWNER          : um-
GROUP          : u--
OTHER          : ---
```

The following command allows users in the same group **USE** and **MANAGE** the Template, and the rest of the users **USE** it:

```
$ onetemplate chmod 0 664

$ onetemplate show 0
...
PERMISSIONS
OWNER          : um-
GROUP          : um-
OTHER          : u--
```

The commands `onetemplate publish` and `onetemplate unpublish` are still present for compatibility with previous versions. These commands set/unset the `GROUP USE` bit.

1.4.4 Instantiating Templates

The `onetemplate instantiate` command accepts a Template ID or name, and creates a VM instance (you can define the number of instances using the `-multiple num_of_instances` option) from the given template.

```
$ onetemplate instantiate 6
VM ID: 0

$ onevm list
    ID USER     GROUP    NAME         STAT CPU     MEM        HOSTNAME        TIME
     0 oneuser1 users    one-0        pend   0      0K                 00 00:00:16
```

You can also merge another template to the one being instantiated. The new attributes will be added, or will replace the ones fom the source template. This can be more convinient that cloning an existing template and updating it.

```
$ cat /tmp/file
MEMORY = 512
COMMENT = "This is a bigger instance"

$ onetemplate instantiate 6 /tmp/file
VM ID: 1
```

The same options to create new templates can be used to be merged with an existing one. See the above table, or execute 'onetemplate instantiate -help' for a complete reference.

```
$ onetemplate instantiate 6 --cpu 2 --memory 1024
VM ID: 2
```

Merge Use Case

The template merge functionality, combined with the restricted attibutes, can be used to allow users some degree of customization for predefined templates.

Let's say the administrator wants to provide base templates that the users can customize, but with some restrictions. Having the following *restricted attributes in oned.conf*:

```
VM_RESTRICTED_ATTR = "CPU"
VM_RESTRICTED_ATTR = "VPU"
VM_RESTRICTED_ATTR = "NIC"
```

And the following template:

```
CPU     = "1"
VCPU    = "1"
MEMORY  = "512"
DISK=[
  IMAGE_ID = "0" ]
NIC=[
  NETWORK_ID = "0" ]
```

Users can instantiate it customizing anything except the CPU, VCPU and NIC. To create a VM with different memory and disks:

```
$ onetemplate instantiate 0 --memory 1G --disk "Ubuntu 12.10"
```

Warning: The merged attributes replace the existing ones. To add a new disk, the current one needs to be added also.

```
$ onetemplate instantiate 0 --disk 0,"Ubuntu 12.10"
```

1.4.5 Deployment

The OpenNebula Scheduler will deploy automatically the VMs in one of the available Hosts, if they meet the requirements. The deployment can be forced by an administrator using the `onevm deploy` command.

Use `onevm shutdown` to shutdown a running VM.

Continue to the *Managing Virtual Machine Instances Guide* to learn more about the VM Life Cycle, and the available operations that can be performed.

1.5 Managing Virtual Machines

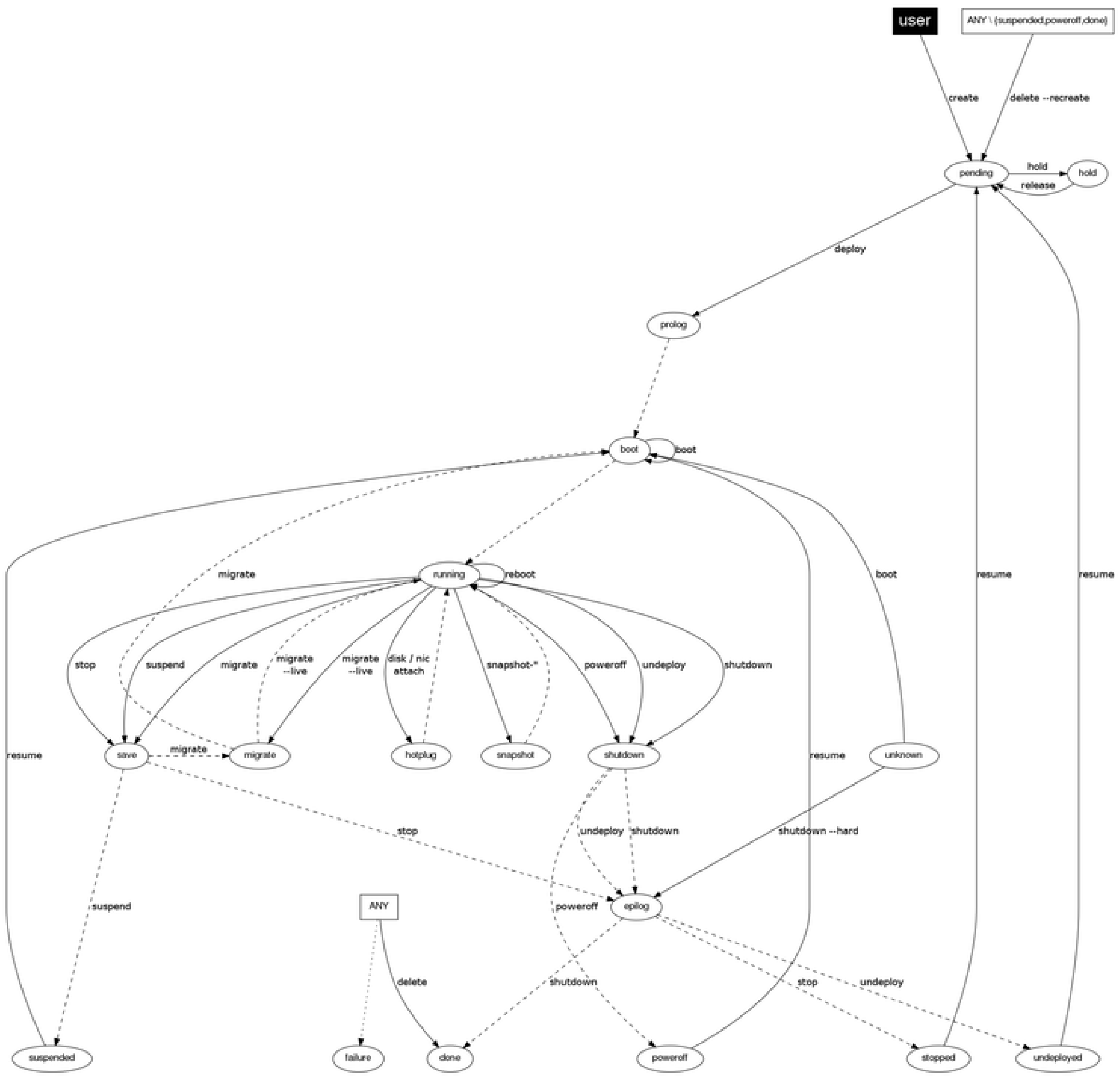
This guide follows the *Creating Virtual Machines guide*. Once a Template is instantiated to a Virtual Machine, there are a number of operations that can be performed using the `onevm` command.

1.5.1 Virtual Machine Life-cycle

The life-cycle of a Virtual Machine within OpenNebula includes the following stages:

Warning: Note that this is a simplified version. If you are a developer you may want to take a look at the complete diagram referenced in the *xml-rpc api page*):

Short state	State	Meaning
`pend`	`Pending`	By default a VM starts in the pending state, waiting for a resource to run on. It will stay in this state until the scheduler decides to deploy it, or the user deploys it using the `onevm deploy` command.
`hold`	`Hold`	The owner has held the VM and it will not be scheduled until it is released. It can be, however, deployed manually.
`prol`	`Prolog`	The system is transferring the VM files (disk images and the recovery file) to the host in which the virtual machine will be running.
`boot`	`Boot`	OpenNebula is waiting for the hypervisor to create the VM.
`runn`	`Running`	The VM is running (note that this stage includes the internal virtualized machine booting and shutting down phases). In this state, the virtualization driver will periodically monitor it.
`migr`	`Migrate`	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).
`hotp`	`Hotplug`	A disk attach/detach, nic attach/detach operation is in process.
`snap`	`Snapshot`	A system snapshot is being taken.
`save`	`Save`	The system is saving the VM files after a migration, stop or suspend operation.
`epil`	`Epilog`	In this phase the system cleans up the Host used to virtualize the VM, and additionally disk images to be saved are copied back to the system datastore.
`shut`	`Shutdown`	OpenNebula has sent the VM the shutdown ACPI signal, and is waiting for it to complete the shutdown process. If after a timeout period the VM does not disappear, OpenNebula will assume that the guest OS ignored the ACPI signal and the VM state will be changed to **running**, instead of **done**.
`stop`	`Stopped`	The VM is stopped. VM state has been saved and it has been transferred back along with the disk images to the system datastore.
`susp`	`Suspended`	Same as stopped, but the files are left in the host to later resume the VM there (i.e. there is no need to re-schedule the VM).
`poff`	`PowerOff`	Same as suspended, but no checkpoint file is generated. Note that the files are left in the host to later boot the VM there.
`unde`	`Undeployed`	The VM is shut down. The VM disks are transfered to the system datastore. The VM can be resumed later.
`fail`	`Failed`	The VM failed.
`unkn`	`Unknown`	The VM couldn't be reached, it is in an unknown state.
`done`	`Done`	The VM is done. VMs in this state won't be shown with `onevm list` but are kept in the database for accounting purposes. You can still get their information with the `onevm show` command.



1.5.2 Managing Virtual Machines

The following sections show the basics of the `onevm` command with simple usage examples. A complete reference for these commands can be found *here*.

Create and List Existing VMs

Warning: Read the *Creating Virtual Machines guide* for more information on how to manage and instantiate VM Templates.

Warning: Read the complete reference for *Virtual Machine templates*.

Assuming we have a VM Template registered called *vm-example* with ID 6, then we can instantiate the VM issuing a:

```
$ onetemplate list
  ID USER     GROUP    NAME                         REGTIME
   6 oneadmin oneadmin vm_example            09/28 06:44:07

$ onetemplate instantiate vm-example --name my_vm
VM ID: 0
```

afterwards, the VM can be listed with the `onevm list` command. You can also use the `onevm top` command to list VMs continuously.

```
$ onevm list
    ID USER     GROUP    NAME          STAT CPU     MEM        HOSTNAME        TIME
     0 oneadmin oneadmin my_vm         pend   0      0K                 00 00:00:03
```

After a Scheduling cycle, the VM will be automatically deployed. But the deployment can also be forced by oneadmin using `onevm deploy`:

```
$ onehost list
  ID NAME               RVM   TCPU   FCPU   ACPU   TMEM   FMEM   AMEM   STAT
   2 testbed              0    800    800    800    16G    16G    16G     on

$ onevm deploy 0 2

$ onevm list
    ID USER     GROUP    NAME          STAT CPU     MEM        HOSTNAME        TIME
     0 oneadmin oneadmin my_vm         runn   0      0K         testbed 00 00:02:40
```

and details about it can be obtained with `show`:

```
$ onevm show 0
VIRTUAL MACHINE 0 INFORMATION
ID                    : 0
NAME                  : my_vm
USER                  : oneadmin
GROUP                 : oneadmin
STATE                 : ACTIVE
LCM_STATE             : RUNNING
START TIME            : 04/14 09:00:24
END TIME              : -
DEPLOY ID:            : one-0

PERMISSIONS
OWNER          : um-
GROUP          : ---
OTHER          : ---

VIRTUAL MACHINE MONITORING
NET_TX                : 13.05
NET_RX                : 0
USED MEMORY           : 512
USED CPU              : 0

VIRTUAL MACHINE TEMPLATE
...
```

```
VIRTUAL MACHINE HISTORY
 SEQ        HOSTNAME REASON           START         TIME       PTIME
   0         testbed   none  09/28 06:48:18 00 00:07:23 00 00:00:00
```

Terminating VM Instances...

You can terminate a running instance with the following operations (either as `onevm` commands or through Sunstone):

- `shutdown`: Gracefully shuts down a running VM, sending the ACPI signal. Once the VM is shutdown the host is cleaned, and persistent and deferred-snapshot disk will be moved to the associated datastore. If after a given time the VM is still running (e.g. guest ignoring ACPI signals), OpenNebula will returned the VM to the `RUNNING` state.
- `shutdown --hard`: Same as above but the VM is immediately destroyed. Use this action instead of `shutdown` when the VM doesn't have ACPI support.

If you need to terminate an instance in any state use:

- `delete`: The VM is immediately destroyed no matter its state. Hosts are cleaned as needed but no images are moved to the repository, leaving then in error. Think of delete as kill -9 for a process, an so it should be only used when the VM is not responding to other actions.

All the above operations free the resources used by the VM

Pausing VM Instances...

There are two different ways to temporarily stop the execution of a VM: short and long term pauses. A **short term** pause keeps all the VM resources allocated to the hosts so its resume its operation in the same hosts quickly. Use the following `onevm` commands or Sunstone actions:

- `suspend`: the VM state is saved in the running Host. When a suspended VM is resumed, it is immediately deployed in the same Host by restoring its saved state.
- `poweroff`: Gracefully powers off a running VM by sending the ACPI signal. It is similar to suspend but without saving the VM state. When the VM is resumed it will boot immediately in the same Host.
- `poweroff --hard`: Same as above but the VM is immediately powered off. Use this action when the VM doesn't have ACPI support.

You can also plan a **long term pause**. The Host resources used by the VM are freed and the Host is cleaned. Any needed disk is saved in the system datastore. The following actions are useful if you want to preserve network and storage allocations (e.g. IPs, persistent disk images):

- `undeploy`: Gracefully shuts down a running VM, sending the ACPI signal. The Virtual Machine disks are transferred back to the system datastore. When an undeployed VM is resumed, it is be moved to the pending state, and the scheduler will choose where to re-deploy it.
- `undeploy --hard`: Same as above but the running VM is immediately destroyed.
- `stop`: Same as `undeploy` but also the VM state is saved to later resume it.

When the VM is successfully paused you can resume its execution with:

- `resume`: Resumes the execution of VMs in the stopped, suspended, undeployed and poweroff states.

Resetting VM Instances...

There are two ways of resetting a VM: in-host and full reset. The first one does not frees any resources and reset a RUNNING VM instance at the hypervisor level:

- `reboot`: Gracefully reboots a running VM, sending the ACPI signal.
- `reboot --hard`: Performs a 'hard' reboot.

A VM instance can be reset in any state with:

- `delete --recreate`: Deletes the VM as described above, but instead of disposing it the VM is moving again to PENDING state. As the delete operation this action should be used when the VM is not responding to other actions. Try undeploy or undeploy –hard first.

Delaying VM Instances...

The deployment of a PENDING VM (e.g. after creating or resuming it) can be delayed with:

- `hold`: Sets the VM to hold state. The scheduler will not deploy VMs in the `hold` state. Please note that VMs can be created directly on hold, using 'onetemplate instantiate –hold' or 'onevm create –hold'.

Then you can resume it with:

- `release`: Releases a VM from hold state, setting it to pending. Note that you can automatically release a VM by scheduling the operation as explained below

Life-Cycle Operations for Administrators

There are some `onevm` commands operations meant for the cloud administrators:

Scheduling:

- `resched`: Sets the reschedule flag for the VM. The Scheduler will migrate (or migrate –live, depending on the *Scheduler configuration*) the VM in the next monitorization cycle to a Host that better matches the requirements and rank restrictions. Read more in the *Scheduler documentation*.
- `unresched`: Clears the reschedule flag for the VM, canceling the rescheduling operation.

Deployment:

- `deploy`: Starts an existing VM in a specific Host.
- `migrate --live`: The Virtual Machine is transferred between Hosts with no noticeable downtime. This action requires a *shared file system storage*.
- `migrate`: The VM gets stopped and resumed in the target host.

Note: By default, the above operations do not check the target host capacity. You can use the -e (-enforce) option to be sure that the host capacity is not overcommitted.

Troubleshooting:

- `boot`: Forces the hypervisor boot action of a VM stuck in UNKNOWN or BOOT state.
- `recover`: If the VM is stuck in any other state (or the boot operation does not work), you can recover the VM by simulating the failure or success of the missing action. You **have to check the VM state on the host** to decide if the missing action was successful or not.

Disk Snapshoting

You can take a snapshot of a VM disk to preserve or backup its state at a given point of time. There are two types of disk snapshots in OpenNebula:

- **Deferred snapshots**, changes to a disk will be saved as a new Image in the associated datastore when the VM is shutdown. The new image will be locked till the VM is properly shutdown and the transferred from the host to the datastore.
- **Live snapshots**, just as the deferred snapshots, but the disk is copied to the datastore the moment the operation is triggered. Therefore, you must guarantee that the disk is in a consistent state during the copy operation (e.g. by umounting the disk from the VM). While the disk is copied to the datastore the VM will be in the HOTPLUG state.

The `onevm disk-snapshot` command can be run while the VM is RUNNING, POWEROFF or SUSPENDED. See the *Image guide* for specific examples of the disk-snapshot command.

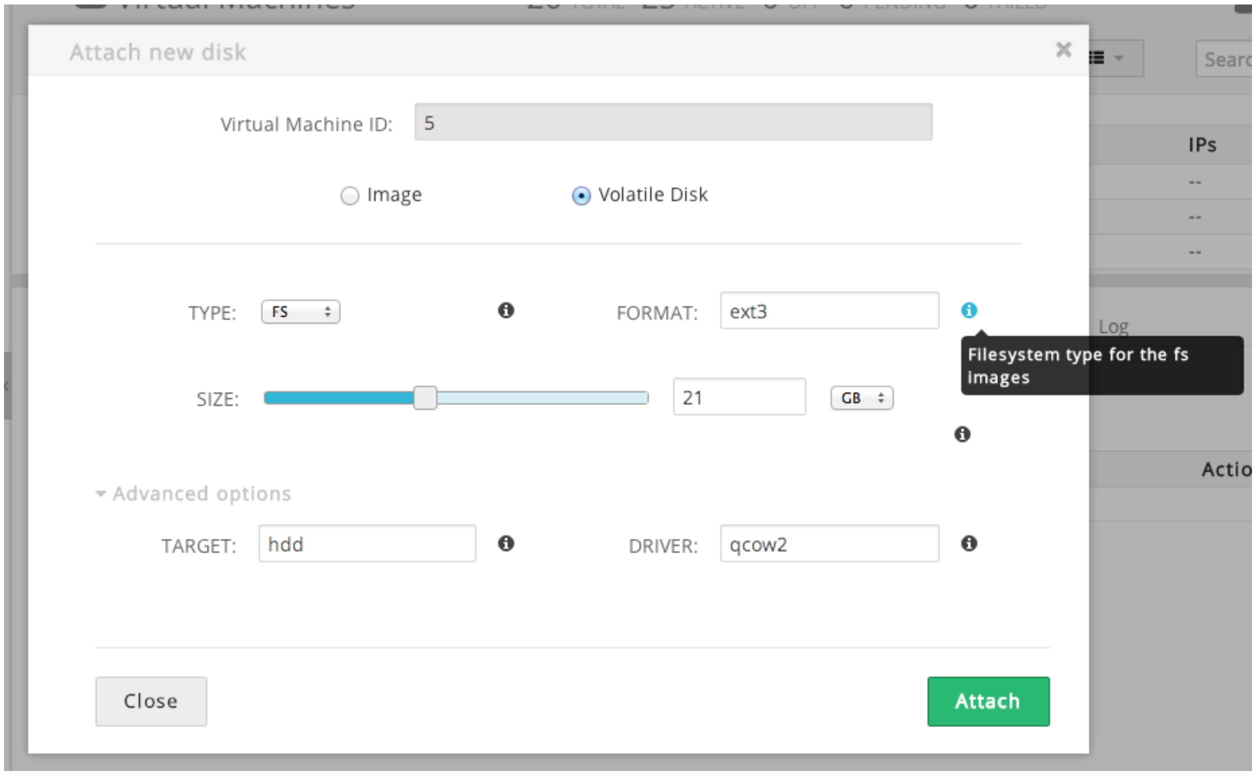
Disk Hotpluging

New disks can be hot-plugged to running VMs with the `onevm` `disk-attach` and `disk-detach` commands. For example, to attach to a running VM the Image named **storage**:

```
$ onevm disk-attach one-5 --image storage
```

To detach a disk from a running VM, find the disk ID of the Image you want to detach using the `onevm show` command, and then simply execute `onevm detach vm_id disk_id`:

```
$ onevm show one-5
...
DISK=[
  DISK_ID="1",
...
  ]
...

$ onevm disk-detach one-5 1
```



Attach new disk

Virtual Machine ID: 5

○ Image [✓ Volatile Disk]

TYPE: FS

FORMAT: ext3

Filesystem type for the fs images

SIZE: 21 GB

Advanced options

TARGET: hdd

DRIVER: qcow2

NIC Hotpluging

You can also hotplug network interfaces to a RUNNING VM. Simply, specify the network where the new interface should be attach to, for example:

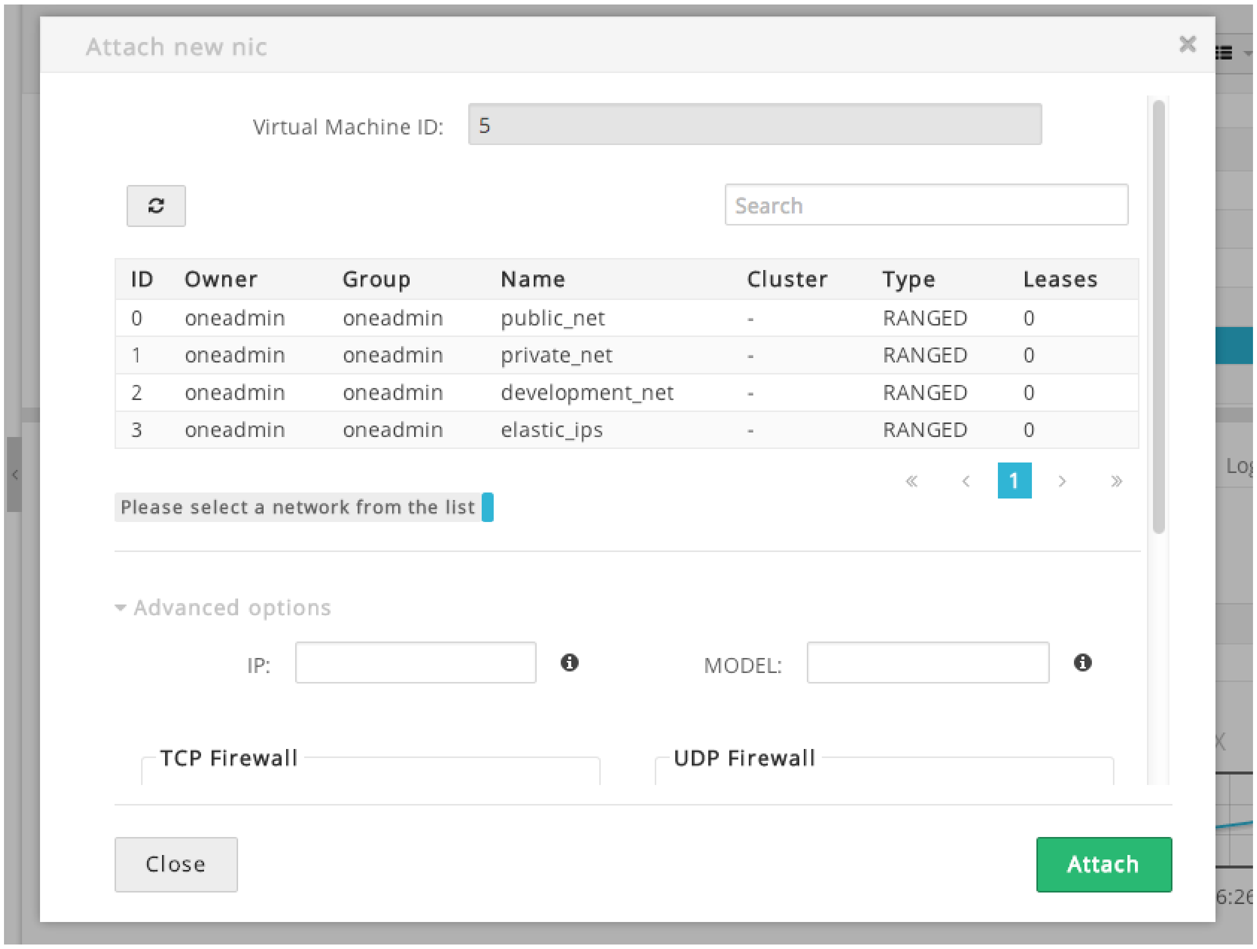
```
$ onevm show 2

VIRTUAL MACHINE 2 INFORMATION
ID                    : 2
NAME                  : centos-server
USER                  : ruben
GROUP                 : oneadmin
STATE                 : ACTIVE
LCM_STATE             : RUNNING
RESCHED               : No
HOST                  : cloud01

...

VM NICS
ID NETWORK       VLAN BRIDGE    IP               MAC
 0 net_172         no vbr0      172.16.0.201     02:00:ac:10:0

VIRTUAL MACHINE HISTORY
 SEQ HOST             REASON           START              TIME      PROLOG_TIME
   0 cloud01          none     03/07 11:37:40    0d 00h02m14s     0d 00h00m00s
...

$ onevm attachnic 2 --network net_172
```

After the operation you should see two NICs 0 and 1:

```
$ onevm show 2
VIRTUAL MACHINE 2 INFORMATION
ID                  : 2
NAME                : centos-server
USER                : ruben
GROUP               : oneadmin

...


VM NICS
ID NETWORK      VLAN BRIDGE   IP              MAC
 0 net_172        no vbr0     172.16.0.201    02:00:ac:10:00:c9
                              fe80::400:acff:fe10:c9

 1 net_172        no vbr0     172.16.0.202    02:00:ac:10:00:ca
                              fe80::400:acff:fe10:ca
...
```

Also, you can detach a NIC by its ID. If you want to detach interface 1 (MAC=02:00:ac:10:00:ca), just:

```
> onevm detachnic 2 1
```



Virtual Machine ID: 5

ID	Owner	Group	Name	Cluster	Type	Leases
0	oneadmin	oneadmin	public_net	-	RANGED	0
1	oneadmin	oneadmin	private_net	-	RANGED	0
2	oneadmin	oneadmin	development_net	-	RANGED	0
3	oneadmin	oneadmin	elastic_ips	-	RANGED	0

Please select a network from the list

IP:

MODEL:

Snapshotting

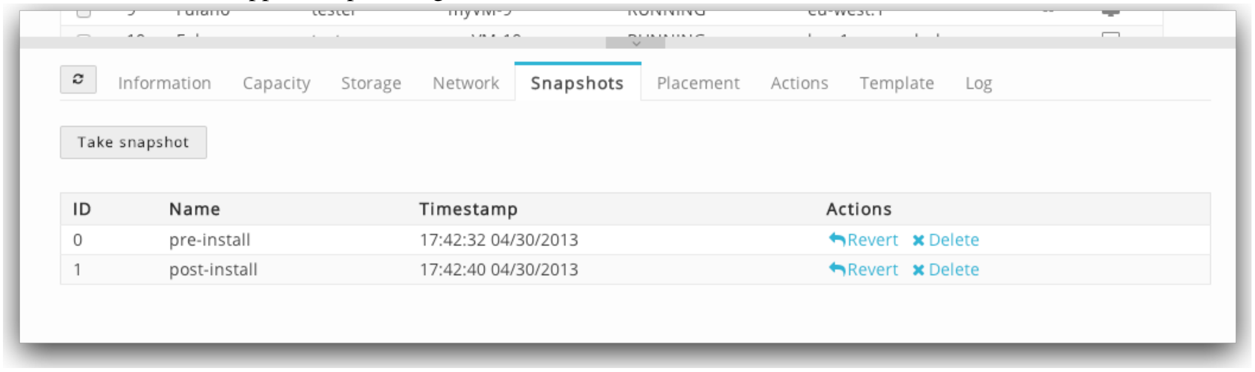
You can create, delete and restore snapshots for running VMs. A snapshot will contain the current disks and memory state.

> **Warning:** The snapshots will only be available during the `RUNNING` state. If the state changes (stop, migrate, etc...) the snapshots **will** be lost.

```
$ onevm snapshot-create 4 "just in case"

$ onevm show 4
...
SNAPSHOTS
  ID         TIME NAME                                           HYPERVISOR_ID
   0  02/21 16:05 just in case                                   onesnap-0

$ onevm snapshot-revert 4 0 --verbose
VM 4: snapshot reverted
```

Please take into consideration the following limitations:

- **The snapshots are lost if any life-cycle operation is performed, e.g. a suspend, migrate, delete request.**
- KVM: Snapshots are only available if all the VM disks use the *qcow2 driver*.
- VMware: the snapshots will persist in the hypervisor after any life-cycle operation is performed, but they will not be available to be used with OpenNebula.
- Xen: does not support snapshotting



ID	Name	Timestamp	Actions
0	pre-install	17:42:32 04/30/2013	Revert Delete
1	post-install	17:42:40 04/30/2013	Revert Delete

Resizing a VM

You may re-size the capacity assigned to a Virtual Machine in terms of the virtual CPUs, memory and CPU allocated. VM re-sizing can be done when the VM is not ACTIVE, an so in any of the following states: PENDING, HOLD, FAILED and specially in POWEROFF.

If you have created a Virtual Machine and you need more resources, the following procedure is recommended:

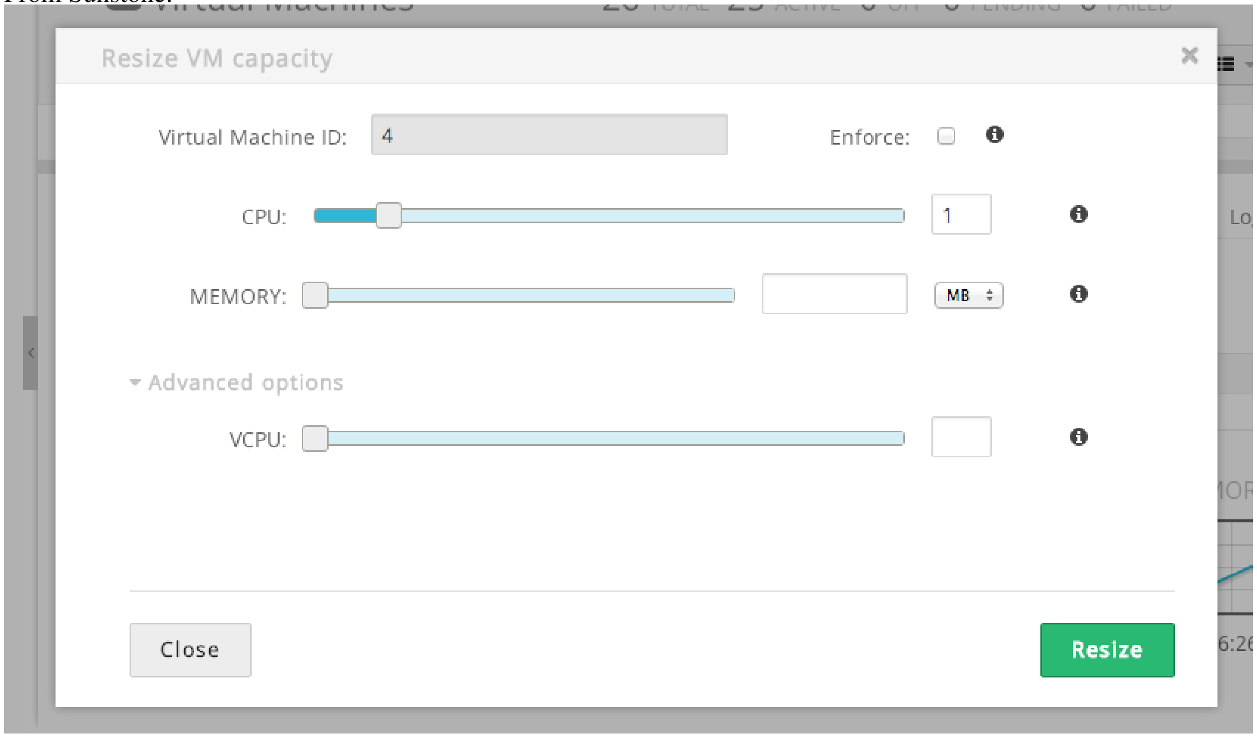
- Perform any operation needed to prepare your Virtual Machine for shutting down, e.g. you may want to manually stop some services...
- Poweroff the Virtual Machine
- Re-size the VM
- Resume the Virtual Machine using the new capacity

Note that using this procedure the VM will preserve any resource assigned by OpenNebula (e.g. IP leases)

The following is an example of the previous procedure from the command line (the Sunstone equivalent is straight forward):

```
> onevm poweroff web_vm
> onevm resize web_vm --memory 2G --vcpu 2
> onevm resume web_vm
```

From Sunstone:



Scheduling Actions

Most of the onevm commands accept the '–schedule' option, allowing users to delay the actions until the given date and time.

Here is an usage example:

```
$ onevm suspend 0 --schedule "09/20"
VM 0: suspend scheduled at 2013-09-20 00:00:00 +0200

$ onevm resume 0 --schedule "09/23 14:15"
VM 0: resume scheduled at 2013-09-23 14:15:00 +0200

$ onevm show 0
VIRTUAL MACHINE 0 INFORMATION
ID                  : 0
NAME                : one-0

[...]

SCHEDULED ACTIONS
ID ACTION        SCHEDULED         DONE MESSAGE
```

```
 0 suspend     09/20 00:00            -
 1 resume      09/23 14:15            -
```

These actions can be deleted or edited using the 'onevm update' command. The time attributes use Unix time internally.

```
$ onevm update 0

SCHED_ACTION=[
  ACTION="suspend",
  ID="0",
  TIME="1379628000" ]
SCHED_ACTION=[
  ACTION="resume",
  ID="1",
  TIME="1379938500" ]
```

These are the commands that can be scheduled:

- `shutdown`
- `shutdown --hard`
- `undeploy`
- `undeploy --hard`
- `hold`
- `release`
- `stop`
- `suspend`
- `resume`
- `boot`
- `delete`
- `delete-recreate`
- `reboot`
- `reboot --hard`
- `poweroff`
- `poweroff --hard`
- `snapshot-create`

User Defined Data

Custom tags can be associated to a VM to store metadata related to this specific VM instance. To add custom attributes simply use the `onevm update` command.

```
$ onevm show 0
...

VIRTUAL MACHINE TEMPLATE
...
VMID="0"
```

```
$ onevm update 0
ROOT_GENERATED_PASSWORD="1234"
~
~

$onevm show 0
...

VIRTUAL MACHINE TEMPLATE
...
VMID="0"

USER TEMPLATE
ROOT_GENERATED_PASSWORD="1234"
```

Manage VM Permissions

OpenNebula comes with an advanced *ACL rules permission mechanism* intended for administrators, but each VM object has also *implicit permissions* that can be managed by the VM owner. To share a VM instance with other users, to allow them to list and show its information, use the `onevm chmod` command:

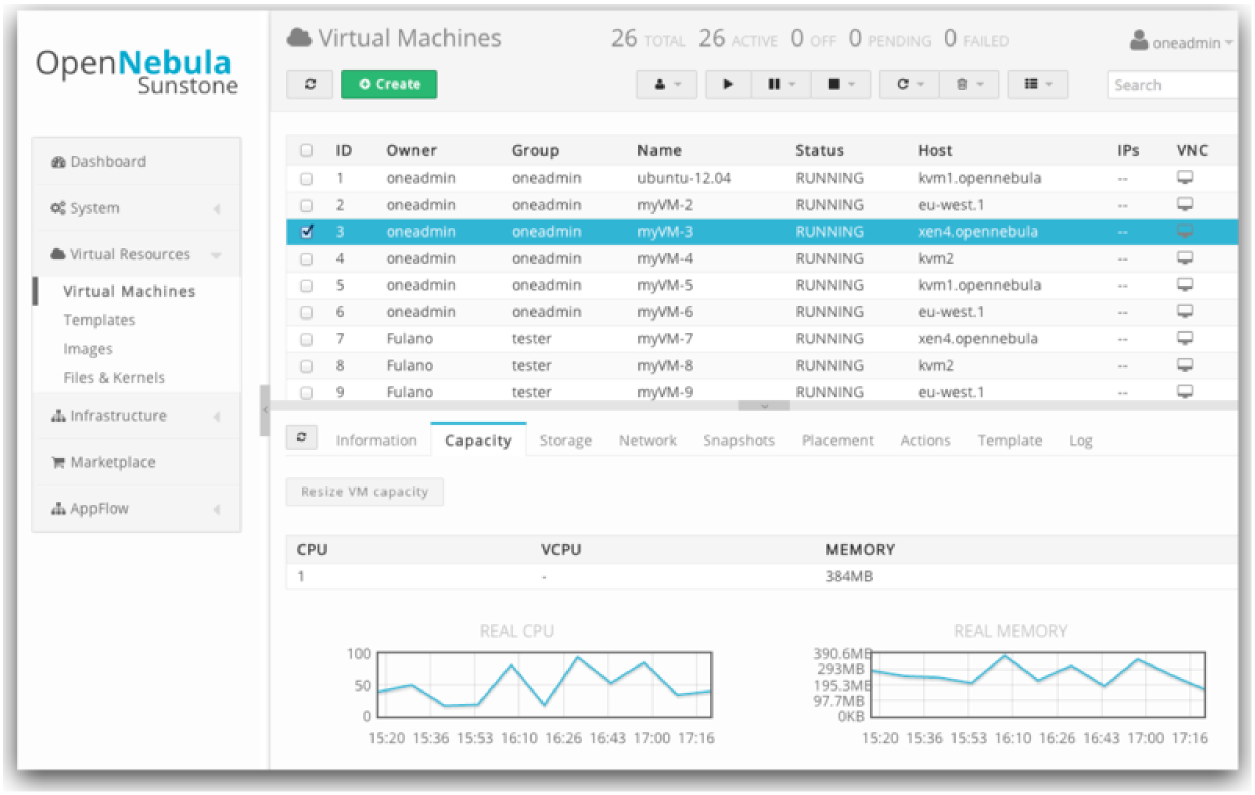
```
$ onevm show 0
...
PERMISSIONS
OWNER          : um-
GROUP          : ---
OTHER          : ---

$ onevm chmod 0 640

$ onevm show 0
...
PERMISSIONS
OWNER          : um-
GROUP          : u--
OTHER          : ---
```

Administrators can also change the VM's group and owner with the `chgrp` and `chown` commands.

1.5.3 Sunstone

You can manage your virtual machines using the *onevm command* or *Sunstone*.

In Sunstone, you can easily instantiate currently defined *templates* by clicking `New` on the Virtual Machines tab and manage the life cycle of the new instances



	ID	Owner	Group	Name	Status	Host	IPs	VNC
☐	1	oneadmin	oneadmin	ubuntu-12.04	RUNNING	kvm1.opennebula	--	🖵
☐	2	oneadmin	oneadmin	myVM-2	RUNNING	eu-west.1	--	🖵
☑	3	oneadmin	oneadmin	myVM-3	RUNNING	xen4.opennebula	--	🖵
☐	4	oneadmin	oneadmin	myVM-4	RUNNING	kvm2	--	🖵
☐	5	oneadmin	oneadmin	myVM-5	RUNNING	kvm1.opennebula	--	🖵
☐	6	oneadmin	oneadmin	myVM-6	RUNNING	eu-west.1	--	🖵
☐	7	Fulano	tester	myVM-7	RUNNING	xen4.opennebula	--	🖵
☐	8	Fulano	tester	myVM-8	RUNNING	kvm2	--	🖵
☐	9	Fulano	tester	myVM-9	RUNNING	eu-west.1	--	🖵

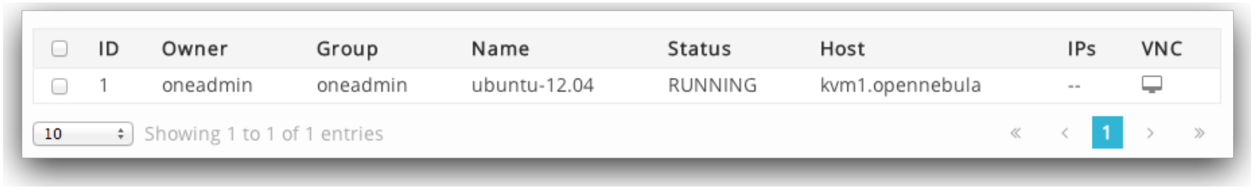
CPU	VCPU	MEMORY
1	-	384MB

Using the noVNC Console

In order to use this feature, make sure that:

- The VM template has a `GRAPHICS` section defined, that the `TYPE` attribute in it is set to `VNC`.
- The specified VNC port on the host on which the VM is deployed is accessible from the Sunstone server host.
- The VM is in `running` state.

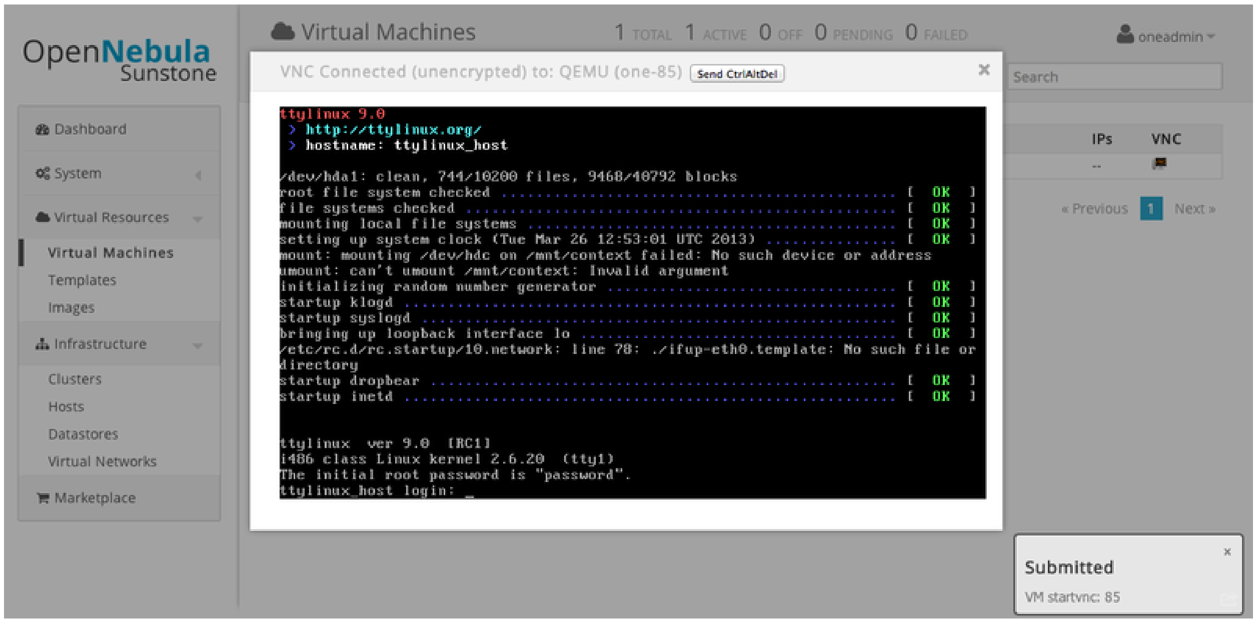
If the VM supports VNC and is `running`, then the VNC icon on the Virtual Machines view should be visible and clickable:



	ID	Owner	Group	Name	Status	Host	IPs	VNC
☐	1	oneadmin	oneadmin	ubuntu-12.04	RUNNING	kvm1.opennebula	--	🖵

When clicking the VNC icon, the process of starting a session begins:

- A request is made and if a VNC session is possible, Sunstone server will add the VM Host to the list of allowed vnc session targets and create a random token associated to it.
- The server responds with the session token, then a `noVNC` dialog pops up.
- The VNC console embedded in this dialog will try to connect to the proxy either using websockets (default) or emulating them using `Flash`. Only connections providing the right token will be successful. Websockets are supported from Firefox 4.0 (manual activation required in this version) and Chrome. The token expires and cannot be reused.



```
ttylinux 9.0
 > http://ttylinux.org/
 > hostname: ttylinux_host

/dev/hda1: clean, 744/10200 files, 9468/40792 blocks
root file system checked ........................................... [  OK  ]
file systems checked ............................................... [  OK  ]
mounting local file systems ........................................ [  OK  ]
setting up system clock (Tue Mar 26 12:53:01 UTC 2013) ............. [  OK  ]
mount: mounting /dev/hdc on /mnt/context failed: No such device or address
umount: can't umount /mnt/context: Invalid argument
initializing random number generator ............................... [  OK  ]
startup klogd ...................................................... [  OK  ]
startup syslogd .................................................... [  OK  ]
bringing up loopback interface lo .................................. [  OK  ]
/etc/rc.d/rc.startup/10.network: line 78: ./ifup-eth0.template: No such file or
directory
startup dropbear ................................................... [  OK  ]
startup inetd ...................................................... [  OK  ]


ttylinux  ver 9.0  [RC1]
i486 class Linux kernel 2.6.20  (tty1)
The initial root password is "password".
ttylinux_host login: _
```

In order to close the VNC session just close the console dialog.

Note: From Sunstone 3.8, a single instance of the VNC proxy is launched when Sunstone server starts. This instance will listen on a single port and proxy all connections from there.

1.5.4 Information for Developers and Integrators

- Although the default way to create a VM instance is to register a Template and then instantiate it, VMs can be created directly from a template file using the `onevm create` command.
- When a VM reaches the `done` state, it disappears from the `onevm list` output, but the VM is still in the database and can be retrieved with the `onevm show` command.
- OpenNebula comes with an *accounting tool* that reports resource usage data.
- The monitoring information, shown with nice graphs in *Sunstone*, can be retrieved using the XML-RPC methods *one.vm.monitoring and one.vmpool.monitoring*.

CHAPTER

TWO

VIRTUAL MACHINE SETUP

2.1 Contextualization Overview

OpenNebula provides different methods to pass information to a newly created Virtual Machine. This information can be the network configuration of the VM, user credentials, init scripts and free form data.

- *Basic Contextualization*: If you only want to configure networking and root ssh keys read this guide.
- *Advanced Contextualization*: For additional topics in contextualization like adding custom init scripts and variables also read this guide.
- *Cloud-init*: To know how to use the cloud-init functionality with OpenNebula check this guide.
- *Winwdows Contextualization*: Contextualization guide specific for Windows guests. From provisioning to contextualization.

2.2 Adding Content to Your Cloud

Once you have setup your OpenNebula cloud you'll have ready the infrastructure (clusters, hosts, virtual networks and datastores) but you need to add contents to it for your users. This basically means two different things:

- Add base disk images with OS installations of your choice. Including any software package of interest.
- Define virtual servers in the form of VM Templates. We recommend that VM definitions are made by the admins as it may require fine or advanced tunning. For example you may want to define a LAMP server with the capacity to be instantiated in a remote AWS cloud.

When you have basic virtual server definitions the users of your cloud can use them to easily provision VMs, adjusting basic parameters, like capacity or network connectivity.

There are three basic methods to bootstratp the contents of your cloud, namely:

- **External Images**. If you already have disk images in any supported format (raw, qcow2, vmdk...) you can just add them to a datastore. Alternatively you can use any virtualization tool (e.g. virt-manager) to install an image and then add it to a OpenNebula datastore.
- **Install within OpenNebula**. You can also use OpenNebula to prepare the images for your cloud. The process will be as follows:
  - Add the installation medium to a OpenNebula datastore. Usually it will be a OS installation CD-ROM/DVD.
  - Create a DATABLOCK image of the desired capacity to install the OS. Once created change its type to OS and make it persistent.

  - Create a new template using the previous two images. Make sure to set the OS/BOOT parameter to cdrom and enable the VNC console.
  - Instantiate the template and install the OS and any additional software
  - Once you are done, shutdown the VM
- **Use the OpenNebula Marketplace**. Go to the marketplace tab in Sunstone, and simply pick a disk image with the OS and Hypervisor of your choice.

Once the images are ready, just create VM templates with the relevant configuration attributes, including default capacity, networking or any other preset needed by your infrastructure.

You are done, make sure that your cloud users can access the images and templates you have just created.

2.3 Basic Contextualization

This guide shows how to automatically configure networking in the initialization process of the VM. Following are the instructions to contextualize your images to configure the network. For more in depth information and information on how to use this information for other duties head to the *Advanced Contextualization* guide.

2.3.1 Preparing the Virtual Machine Image

To enable the Virtual Machine images to use the contextualization information written by OpenNebula we need to add to it a series of scripts that will trigger the contextualization.

You can use the images available in the Marketplace, that are already prepared, or prepare your own images. To make your life easier you can use a couple of Linux packages that do the work for you.

The contextualization package will also mount any partition labeled `swap` as swap. OpenNebula sets this label for volatile swap disks.

- Start a image (or finish its installation)
- Install context packages with one of these methods:
  - Install from our repositories package **one-context** in Ubuntu/Debian or **opennebula-context** in CentOS/RedHat. Instructions to add the repository at the *installation guide*.
  - Download and install the package for your distribution:
    - DEB: Compatible with Ubuntu 11.10 to 14.04 and Debian 6/7
    - RPM: Compatible with CentOS and RHEL 6.x
- Shutdown the VM

2.3.2 Preparing the Template

We will also need to add the gateway information to the Virtual Networks that need it. This is an example of a Virtual Network with gateway information:

```
NAME=public
NETWORK_ADDRESS=80.0.0.0
NETWORK_MASK=255.255.255.0
GATEWAY=80.0.0.1
DNS="8.8.8.8 8.8.4.4"
```

And then in the VM template contextualization we set NETWORK to `yes`:

```
CONTEXT=[
  NETWORK=YES ]
```

When the template is instantiated, those parameters for `eth0` are automatically set in the VM as:

```
CONTEXT=[
  DISK_ID="0",
  ETH0_DNS="8.8.8.8 8.8.4.4",
  ETH0_GATEWAY="80.0.0.1",
  ETH0_IP="80.0.0.2",
  ETH0_MASK="255.255.255.0",
  ETH0_NETWORK="80.0.0.0",
  NETWORK="YES",
  TARGET="hda" ]
```

If you add more that one interface to a Virtual Machine you will end with same parameters changing ETH0 to ETH1, ETH2, etc.

You can also add `SSH_PUBLIC_KEY` parameter to the context to add a SSH public key to the `authorized_keys` file of root.

```
CONTEXT=[
  SSH_PUBLIC_KEY = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC+vPFFwem49zcepQxsyO51YMSpuywwt6GazgpJe9vQ
]
```

If you want to known more in deep the contextualization options head to the *Advanced Contextualization guide*.

2.4 Advanced Contextualization

There are two contextualization mechanisms available in OpenNebula: the automatic IP assignment, and a more generic way to give any file and configuration parameters. You can use any of them individually, or both.

You can use already made packages that install context scripts and prepare udev configuration in your appliances. This is described in *Contextualization Packages for VM Images* section.

2.4.1 Automatic IP Assignment

With OpenNebula you can derive the IP address assigned to the VM from the MAC address using the MAC_PREFFIX:IP rule. In order to achieve this we provide context scripts for Debian, Ubuntu, CentOS and openSUSE based systems. These scripts can be easily adapted for other distributions, check dev.opennebula.org.

To configure the Virtual Machine follow these steps:

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

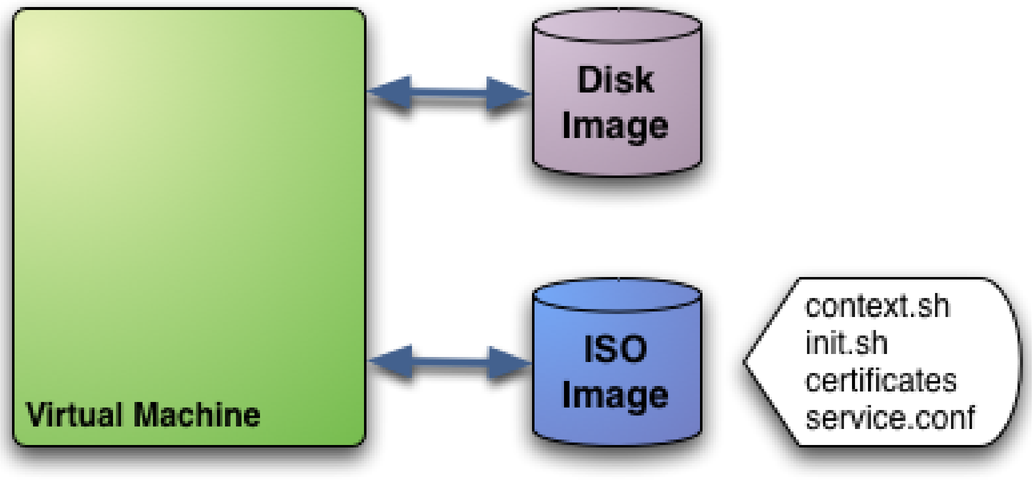
- Copy the script `$ONE_SRC_CODE_PATH/share/scripts/vmcontext.sh` into the `/etc/init.d` directory in the VM root file system.
- Execute the script at boot time before starting any network service, usually runlevel 2 should work.

```
$ ln /etc/init.d/vmcontext.sh /etc/rc2.d/S01vmcontext.sh
```

Having done so, whenever the VM 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.

2.4.2 Generic Contextualization

The method we provide to give configuration parameters to a newly started virtual machine is using an ISO image (OVF recommendation). This method is network agnostic so it can be used also to configure network interfaces. In the VM description file you can specify the contents of the iso file (files and directories), tell the device the ISO image will be accessible and specify the configuration parameters that will be written to a file for later use inside the virtual machine.



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 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

Warning: 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.

Warning: To prevent regular users to copy system/secure files, the `FILES` attribute within `CONTEXT` is only allowed to OpenNebula users within the oneadmin group. `FILES_DS` can be used to include arbitrary files from Files Datastores.

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   = "MAINHOST",
  ip_private = "$NIC[IP, NETWORK=\"public net\"]",
  dns        = "$NETWORK[DNS, NETWORK_ID=0]",
  root_pass   = "$IMAGE[ROOT_PASS, IMAGE_ID=3]",
  ip_gen      = "10.0.0.$VMID",
  files_ds    = "$FILE[IMAGE=\"certificate\"] $FILE[IMAGE=\"server_license\"]"
]
```

Variables inside CONTEXT section will be added to `context.sh` file inside the contextualization image. These variables can be specified in three different ways:

Hardcoded variables

```
hostname   = "MAINHOST"
```

Using template variables

`$<template_variable>`: any single value variable of the VM template, like for example:

```
ip_gen     = "10.0.0.$VMID"
```

`$<template_variable>[<attribute>]`: Any single value contained in a multiple value variable in the VM template, like for example:

```
ip_private = $NIC[IP]
```

`$<template_variable>[<attribute>, <attribute2>=<value2>]`: Any single value contained in a multiple value variable in the VM template, setting one attribute to discern between multiple variables called the same way, like for example:

```
ip_public = "$NIC[IP, NETWORK=\"Public\"]"
```

You can use any of the attributes defined in the variable, NIC in the previous example.

Using Virtual Network template variables

`$NETWORK[<vnet_attribute>, <NETWORK_ID|NETWORK>=<vnet_id|vnet_name>]`: Any single value variable in the Virtual Network template, like for example:

```
dns = "$NETWORK[DNS, NETWORK_ID=3]"
```

Using Image template variables

`$IMAGE[<image_attribute>, <IMAGE_ID|IMAGE>=<img_id|img_name>]`: Any single value variable in the Image template, like for example:

```
    root = "$IMAGE[ROOT_PASS, IMAGE_ID=0]"
```

```
Note that the image MUST be in used by any of the DISKs defined in the template. The image\_attribute
```

Using User template variables

`$USER[<user_attribute>]`: Any single value variable in the user (owner of the VM) template, like for example:

```
ssh_key = "$USER[SSH_KEY]"
```

The user_attribute can be `TEMPLATE` to include the whole user template in XML (base64 encoded).

Pre-defined variables, apart from those defined in the template you can use:

- `$UID`, the uid of the VM owner
- `$UNAME`, the VM owner user name
- `$GID`, the id of the VM group
- `$GNAME`, the VM group name
- `$TEMPLATE`, the whole template in XML format and encoded in base64

The file generated will be something like this:

```
# Context variables generated by OpenNebula
hostname="MAINHOST"
ip_private="192.168.0.5"
dns="192.168.4.9"
ip_gen="10.0.0.85"
files_ds="/home/cloud/var/datastores/2/3fae86a862b7539b41de350e8fa56100 /home/cloud/var/datastores/2/
target="sdb"
root="13.0"
```

Some of the variables have special meanings, but none of them are mandatory:

Attribute	Description
files_ds	Files that will be included in the contextualization image. Each file must be stored in a FILE_DS Datastore and must be of type CONTEXT
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, e.g. ubuntu VMs should use hd* as the target device
file	Files and directories that will be included in the contextualization image. Specified as absolute paths, by default this **can be used only by oneadmin**.
init_scripts	If you want the VM to execute an script that is not called init.sh (or if you want to call more than just one script),this list contains the scripts to run and their order. Ex. `init.sh users.sh mysql.sh` will force the VM to execute init.sh , then users.sh and lastly mysql.sh at boot time'
TOKEN	`YES` to create a token.txt file for *OneGate monitorization*
NET-WORK	`YES` to fill automatically the networking parameters for each NIC, used by the *Contextualization packages*
SET_HOSTNAME	This parameter value will be the hostname of the VM.
DNS_HOSTNAME	`YES` to set the VM hostname to the reverse dns name (from the first IP)

Warning: A default target attribute is *generated automatically* by OpenNebula, based on the default device prefix set at *oned.conf*.

Contextualization Packages for VM Images

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 easily source it inside a shellscript to get the variables that it contains.

Contextualization packages are available to several distributions so you can prepare them to work with OpenNebula without much effort. These are the changes they do to your VM:

- Disables udev net and cd persistent rules
- Deletes udev net and cd persistent rules
- Unconfigures the network
- Adds OpenNebula contextualization scripts to startup

Warning: These packages are destructive. The configuration for networking will be deleted. Make sure to use this script on copies of your images.

Instructions on how to install the contextualization packages are located in the *contextualization overview documentation*.

After the installation of these packages the images on start will configure the network using the mac address generated by OpenNebula. They will also try to mount the cdrom context image from `/dev/cdrom` and if `init.sh` is found it will be executed.

Network Configuration

These packages also install a generic network configuration script that will get network information from some contextualization parameters and also root SSH key. This way we don't have to supply an `init.sh` script to do this work. The parameters that these scripts will use are as follows:

Attribute	Description
`<DEV>_MAC`	MAC address of the interface
`<DEV>_IP`	IP assigned to the interface
`<DEV>_NETWORK`	Interface network
`<DEV>_MASK`	Interface net mask
`<DEV>_GATEWAY`	Interface gateway
`<DEV>_DNS`	DNS servers for the network
`<DEV>_SEARCH_DOMAIN`	DNS domain search path
`<DEV>_IPV6`	Global IPv6 assigned to the interface
`<DEV>_GATEWAY6`	IPv6 gateway for this interface
`<DEV>_CONTEXT_FORCE_IPV4`	Configure IPv4 even if IPv6 values are present
`DNS`	main DNS server for the machine
`SSH_PUBLIC_KEY`	public ssh key added to root authorized_keys

We can have the networks defined with those parameters and use them to configure the interfaces. Given these two networks (excerpt):

Public:

```
NAME = public
TYPE = RANGED
NETWORK_ADDRESS = 130.10.0.0
NETWORK_MASK = 255.255.255.0
GATEWAY = 130.10.0.1
DNS = "8.8.8.8 8.8.4.4"
```

Private:

```
NAME = private
TYPE = RANGED
NETWORK_ADDRESS = 10.0.0.0
NETWORK_MASK = 255.255.0.0
```

We can configure both networks adding this context to the VM template:

```
CONTEXT=[
  NETWORK="YES",
  SSH_PUBLIC_KEY="$USER[SSH_PUBLIC_KEY]" ]

NIC=[
  NETWORK="public" ]
NIC=[
  NETWORK="private" ]
```

Please note that SSH_PUBLIC_KEY was added as a user attribute, this way the templates can be generic.

When this template is instantiated, the context section will contain all the relevant networking attributes:

```
CONTEXT=[
  DISK_ID="0",

  ETH0_DNS="8.8.8.8 8.8.4.4",
  ETH0_GATEWAY="130.10.0.1",
  ETH0_IP="130.10.0.1",
  ETH0_MASK="255.255.255.0",
  ETH0_NETWORK="130.10.0.0",

  ETH1_IP="10.0.0.1",
  ETH1_MASK="255.255.0.0",
  ETH1_NETWORK="10.0.0.0",

  NETWORK="YES",
  SSH_PUBLIC_KEY="ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC+vPFFwem49zcepQxsyO51YMSpuywwt6GazgpJe9vQzw
  TARGET="hda" ]
```

2.4.3 Generating Custom Contextualization Packages

Network configuration is a script located in `/etc/one-context.d/00-network`. Any file located in that directory will be executed on start, in alphabetical order. This way we can add any script to configure or start processes on boot. For example, we can have a script that populates authorized_keys file using a variable in the context.sh. Remember that those variables are exported to the environment and will be easily accessible by the scripts:

```
#!/bin/bash
echo "$SSH_PUBLIC_KEY" > /root/.ssh/authorized_keys
```

OpenNebula source code comes with the scripts and the files needed to generate contextualization packages. This way you can also generate custom packages tweaking the scripts that will go inside your images or adding new scripts that will perform other duties.

The files are located in `share/scripts/context-packages`:

- `base`: files that will be in all the packages. Right now it contains empty `udev` rules and the init script that will be executed on startup.
- `base_<type>`: files specific for linux distributions. It contains the contextualization scripts for the network and comes in `rpm` and `deb` flavors. You can add here your own contextualization scripts and they will be added to the package when you run the generation script.
- `generate.sh`: The script that generates the packages.
- `postinstall`: This script will be executed after the package installation and will clean the network and `udev` configuration. It will also add the init script to the started services on boot.

To generate the packages you will need:

- Ruby >= 1.8.7
- gem fpm
- dpkg utils for deb package creation
- rpm utils for rpm package creation

You can also give to the generation script some parameters using env variables to generate the packages. For example, to generate an `rpm` package you will execute:

```
$ PACKAGE_TYPE=rpm ./generate.sh
```

These are the default values of the parameters, but you can change any of them the same way we did for `PACKAGE_TYPE`:

```
VERSION=4.4.0
MAINTAINER=C12G Labs <support@c12g.com>
LICENSE=Apache
PACKAGE_NAME=one-context
VENDOR=C12G Labs
DESCRIPTION="
This package prepares a VM image for OpenNebula:
  * Disables udev net and cd persistent rules
  * Deletes udev net and cd persistent rules
  * Unconfigures the network
  * Adds OpenNebula contextualization scripts to startup

To get support use the OpenNebula mailing list:
  http://opennebula.org/community:mailinglists
"
PACKAGE_TYPE=deb
URL=http://opennebula.org
```

For more information check the `README.md` file from that directory.

2.5 Windows Contextualization

This guide describes the standard process of provisioning and contextualizing a Windows guest.

Note: This guide has been tested for Windows 2008 R2, however it should work with Windows systems >= Windows 7.

2.5.1 Provisioning

Installation

Provisioning a Windows VM is performed the standard way in OpenNebula:

1. Register the Installation media (typically a DVD) into a Datastore
2. Create an empty datablock with an appropriate size, at least 10GB. Change the type to `OS`. If you are using a `qcow2` image, don't forget to add `DRIVER=qcow2` and `FORMAT=qcow2`.

3. Create a template that boots from CDROM, enables VNC, and references the Installation media and the Image created in step 2.
4. Follow the typical installation procedure over VNC.
5. Perform a deferred disk-snapshot of the OS disk, which will be saved upon `shutdown`.
6. Shutdown the VM.

The resulting image will boot under any OpenNebula cloud that uses KVM or VMware, and for any storage subsystem. However it hasn't been contextualized, therefore it will only obtain its IP via DHCP. To apply contextualization please follow the *Contextualization* section.

Sysprep

If you are adapting a pre-existing Windows VM to run in an OpenNebula environment, and you want to remove all the pre-existing senstitive data in order to be able to clone and deliver it to third party users, it's highly recommended to run Sysprep on the image. To do so simply run `c:\Windows\System32\sysprep\sysprep.exe`. Select `OOBE` and `Generalize`.

2.5.2 Contextualization

Enabling Contextualization

The official addon-opennebula-context provides all the necessary files to run the contextualization in Windows 2008 R2.

The contextualization procedure is as follows:

1. Download `startup.vbs` to the Windows VM (you can also send it via Context files) and write it to a path under `C:`.
2. Open the Local Group Policy Dialog by running `gpedit.msc`. Under: Computer Configuration -> Windows Settings -> Scripts -> startup (right click); browse to the `startup.vbs` file and enable it as a startup script.

Save the image by performing a deferred disk-snapshot of the OS disk, which will be saved upon `shutdown`.

To use the Windows contextualization script you need to use the previously prepared Windows image and include into the CONTEXT files the `context.ps1` script available here.

Warning: The `context.ps1` name matters. If changed, the script will not run.

Features

The `context.ps1` script will:

- Add a new user (using `USERNAME` and `PASSWORD`).
- Rename the server (using `SET_HOSTNAME`).
- Enable Remote Desktop.
- Enable Ping.
- Configure the Network, using the automatically generated networking variables in the CONTEXT CD-ROM.
- Run arbitrary PowerShell scripts available in the CONTEXT CD-ROM and referenced by the `INIT_SCRIPTS` variable.

Variables

The contextualization variables supported by the Windows context script are very similar to the ones in *Linux* except for a few Windows-specific exceptions.

This is the list of supported variables:

- `<DEV>_MAC`: MAC address of the interface.
- `<DEV>_IP`: IP assigned to the interface.
- `<DEV>_NETWORK`: Interface network.
- `<DEV>_MASK`: Interface net mask.
- `<DEV>_GATEWAY`: Interface gateway.
- `<DEV>_DNS`: DNS servers for the network.
- `<DEV>_SEARCH_DOMAIN`: DNS domain search path.
- `DNS`: main DNS server for the machine.
- `SET_HOSTNAME`: Set the hostname of the machine.
- `INIT_SCRIPTS`: List of PowerShell scripts to be executed. Must be available in the CONTEXT CD-ROM.
- `USERNAME`: Create a new user.
- `PASSWORD`: Password for the new user.

Customization

The `context.ps1` script has been designed to be easily hacked and modified. Perform any changes to that script and use it locally.

2.6 Cloud-init

Since version 0.7.3 of cloud-init packages the OpenNebula context CD is supported. It is able to get and configure networking, hostname, ssh key for root and cloud-init user data. Here are the options in a table:

Option	Description
standard network options	OpenNebula network parameters in the context added by `NETWORK=yes`
HOSTNAME	VM hostname
SSH_PUBLIC_KEY	ssh public key added to root's authorized keys
USER_DATA	Specific user data for cloud-init
DSMODE	Can be set to local, net or disabled to change cloud-init datasource mode

You have more information on how to use it at the cloud-init documentation page.

There are plenty of examples on what can go in the USER_DATA string at the cloud-init examples page.

Warning: The current version of cloud-init configures the network before running cloud-init configuration. This makes the network configuration not reliable. Until a new version that fixes this is released you can add OpenNebula context packages or this user data to reboot the machine so the network is properly configured.

```
CONTEXT=[
  USER_DATA="#cloud-config
power_state:
 mode: reboot
" ]
```

2.6.1 Platform Specific Notes

CentOS

Works correctly for `cloud-init >= 0.7.4`.

Ubuntu/Debian

To make it configure the network correctly it needs to be down so the network configuration part makes its work:

```
CONTEXT=[
  NETWORK="YES",
  SSH_PUBLIC_KEY="$USER[SSH_PUBLIC_KEY]",
  USER_DATA="#cloud-config
bootcmd:
  - ifdown -a
runcmd:
  - curl http://10.0.1.1:8999/I_am_alive
write_files:
-   encoding: b64
    content: RG9lcyBpdCB3b3JrPwo=
    owner: root:root
    path: /etc/test_file
    permissions: '0644'
packages:
  - ruby2.0" ]
```

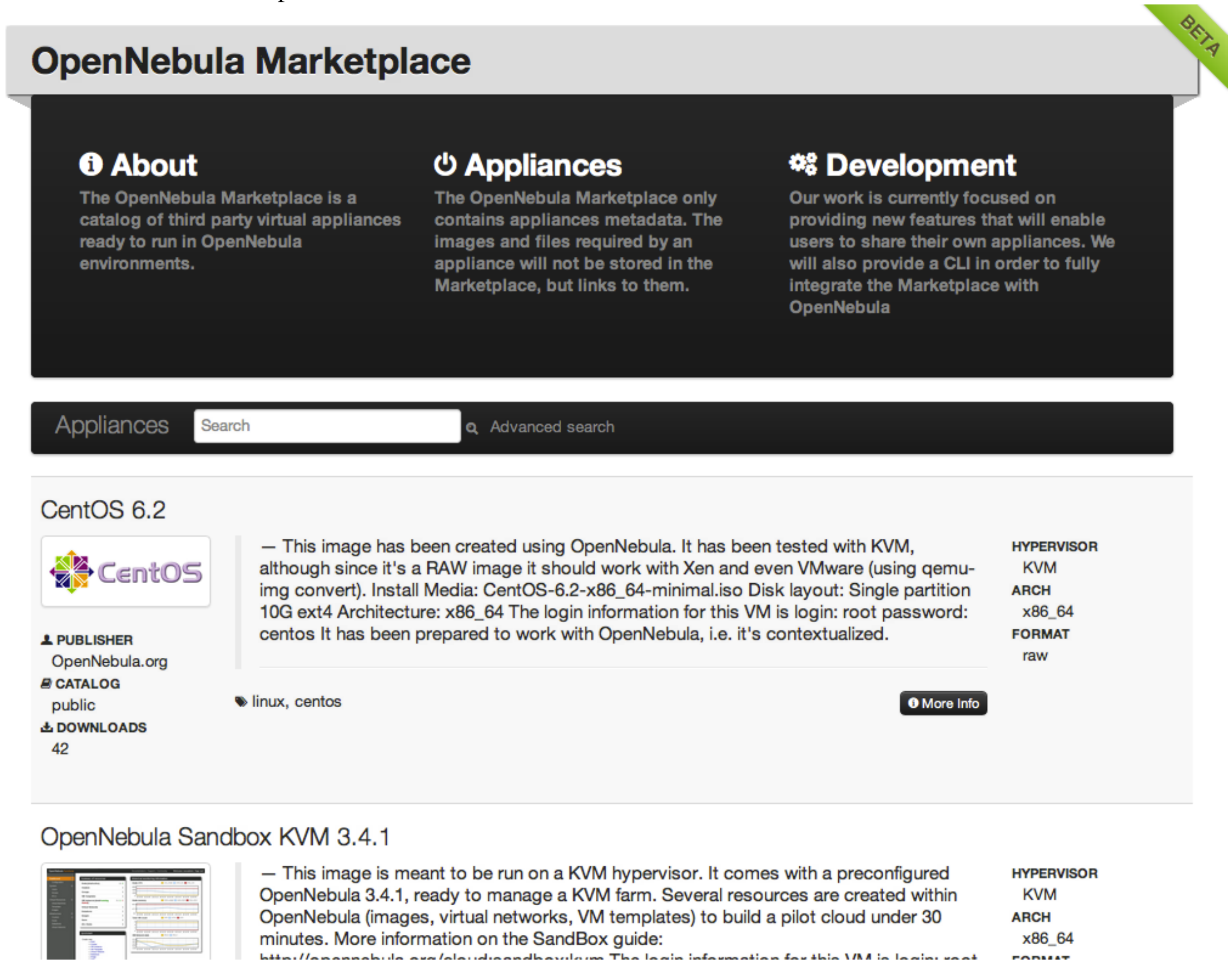
CHAPTER

THREE

OPENNEBULA MARKETPLACE

3.1 Interacting with the OpenNebula Marketplace

The OpenNebula Marketplace is a catalog of third party virtual appliances ready to run in OpenNebula environments. The OpenNebula Marketplace only contains appliances metadata. The images and files required by an appliance will not be stored in the Marketplace, but links to them.



The OpenNebula Marketplace is a catalog of third party virtual appliances ready to run in OpenNebula environments.

The OpenNebula Marketplace only contains appliances metadata. The images and files required by an appliance will not be stored in the Marketplace, but links to them.

Our work is currently focused on providing new features that will enable users to share their own appliances. We will also provide a CLI in order to fully integrate the Marketplace with OpenNebula

CentOS 6.2

— This image has been created using OpenNebula. It has been tested with KVM, although since it's a RAW image it should work with Xen and even VMware (using qemu-img convert). Install Media: CentOS-6.2-x86_64-minimal.iso Disk layout: Single partition 10G ext4 Architecture: x86_64 The login information for this VM is login: root password: centos It has been prepared to work with OpenNebula, i.e. it's contextualized.

PUBLISHER OpenNebula.org
CATALOG public
DOWNLOADS 42

linux, centos

HYPERVISOR KVM
ARCH x86_64
FORMAT raw

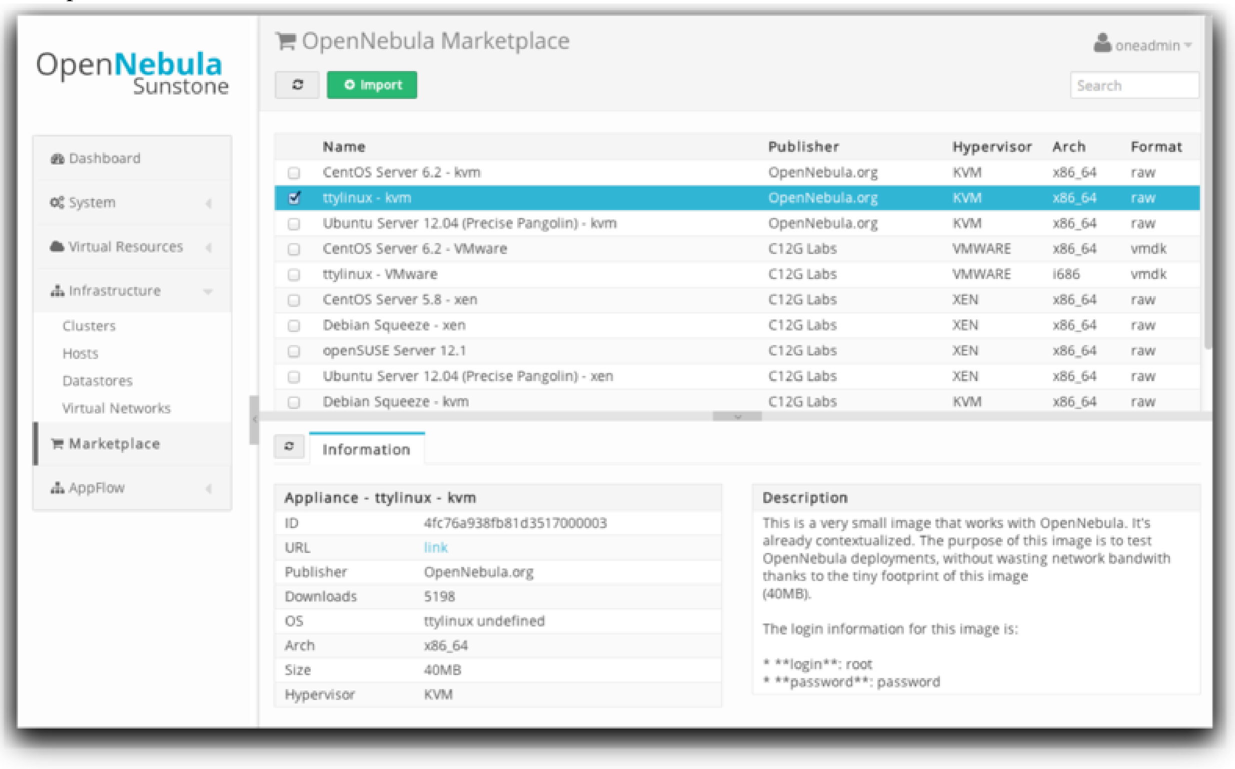
OpenNebula Sandbox KVM 3.4.1

— This image is meant to be run on a KVM hypervisor. It comes with a preconfigured OpenNebula 3.4.1, ready to manage a KVM farm. Several resources are created within OpenNebula (images, virtual networks, VM templates) to build a pilot cloud under 30 minutes. More information on the SandBox guide:

HYPERVISOR KVM
ARCH x86_64

3.1.1 Using Sunstone to Interact with the OpenNebula Marketplace

Since the release 3.6, Sunstone includes a new tab that allows OpenNebula users to interact with the OpenNebula Marketplace:



	Name	Publisher	Hypervisor	Arch	Format
☐	CentOS Server 6.2 - kvm	OpenNebula.org	KVM	x86_64	raw
✓	ttylinux - kvm	OpenNebula.org	KVM	x86_64	raw
☐	Ubuntu Server 12.04 (Precise Pangolin) - kvm	OpenNebula.org	KVM	x86_64	raw
☐	CentOS Server 6.2 - VMware	C12G Labs	VMWARE	x86_64	vmdk
☐	ttylinux - VMware	C12G Labs	VMWARE	i686	vmdk
☐	CentOS Server 5.8 - xen	C12G Labs	XEN	x86_64	raw
☐	Debian Squeeze - xen	C12G Labs	XEN	x86_64	raw
☐	openSUSE Server 12.1	C12G Labs	XEN	x86_64	raw
☐	Ubuntu Server 12.04 (Precise Pangolin) - xen	C12G Labs	XEN	x86_64	raw
☐	Debian Squeeze - kvm	C12G Labs	KVM	x86_64	raw

Appliance - ttylinux - kvm	
ID	4fc76a938fb81d3517000003
URL	link
Publisher	OpenNebula.org
Downloads	5198
OS	ttylinux undefined
Arch	x86_64
Size	40MB
Hypervisor	KVM

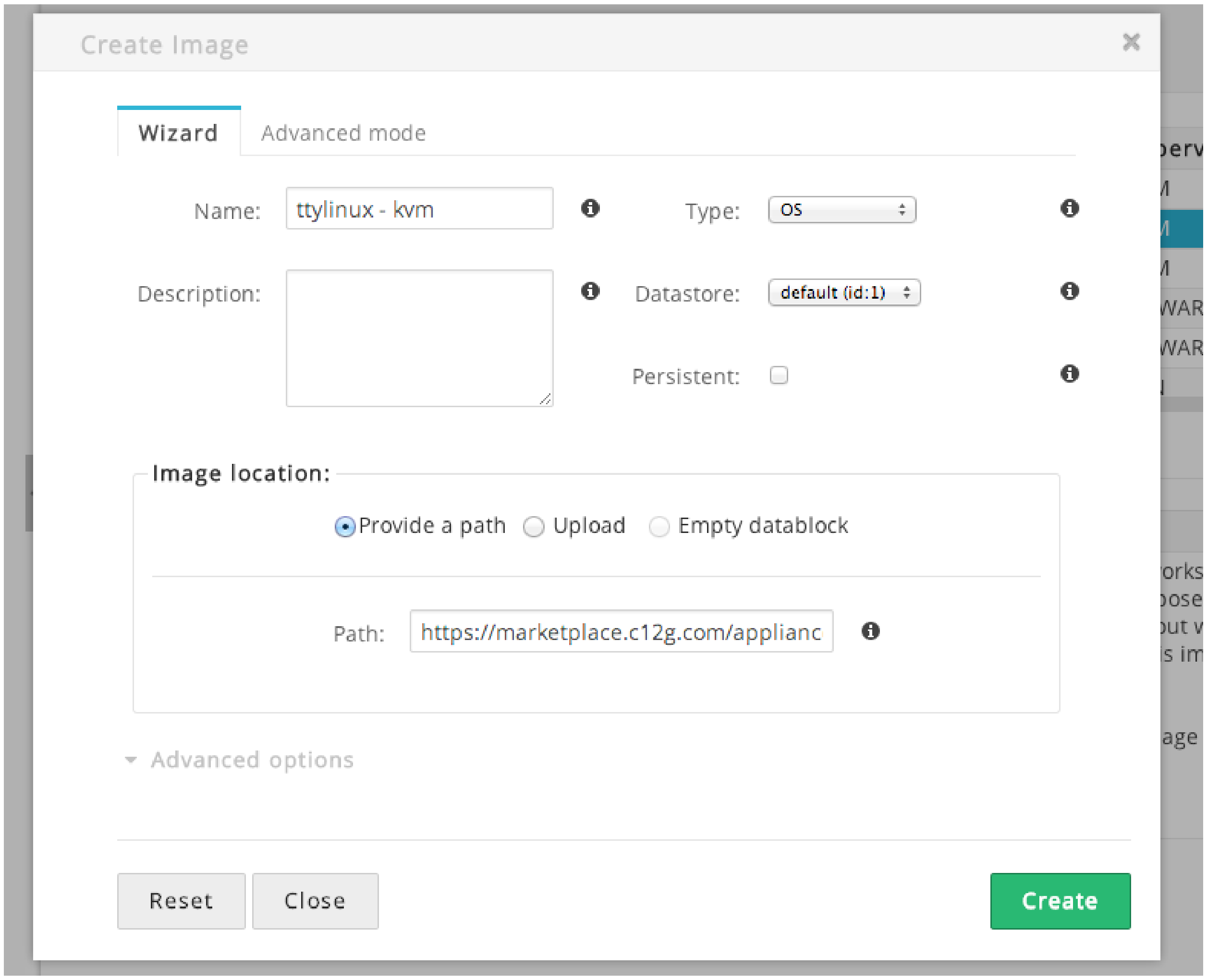
Description

This is a very small image that works with OpenNebula. It's already contextualized. The purpose of this image is to test OpenNebula deployments, without wasting network bandwith thanks to the tiny footprint of this image (40MB).

The login information for this image is:

* **login**: root
* **password**: password

If you want to import a new appliance into your local infrastructure, you just have to select an image and click the button `import`. A new dialog box will prompt you to create a new image.



Name: ttylinux - kvm
Type: OS
Description:
Datastore: default (id:1)
Persistent: ☐
Image location:
[✓ Provide a path] Upload Empty datablock
Path: https://marketplace.c12g.com/applianc

After that you will be able use that image in a template in order to create a new instance.

3.1.2 Using the CLI to Interact with the OpenNebula Marketplace

You can also use the CLI to interact with the OpenNebula Marketplace:

- List appliances:

```
$ onemarket list --server http://marketplace.c12g.com
                       ID                                            NAME        PUBLISHER
 4fc76a938fb81d3517000001        Ubuntu Server 12.04 LTS (Precise Pangolin)  OpenNebula.org
 4fc76a938fb81d3517000002                                        CentOS 6.2  OpenNebula.org
 4fc76a938fb81d3517000003                                          ttylinux  OpenNebula.org
 4fc76a938fb81d3517000004                    OpenNebula Sandbox VMware 3.4.1       C12G Labs
 4fcf5d0a8fb81d1bb8000001                       OpenNebula Sandbox KVM 3.4.1       C12G Labs
```

- Show an appliance:

```
$ onemarket show 4fc76a938fb81d3517000004 --server http://marketplace.c12g.com
{
  "_id": {"$oid": "4fc76a938fb81d3517000004"},
  "catalog": "public",
  "description": "This image is meant to be run on a ESX hypervisor, and comes with a preconfigured
  "downloads": 90,
  "files": [
    {
      "type": "OS",
      "hypervisor": "ESX",
      "format": "VMDK",
      "size": 693729120,
      "compression": "gzip",
      "os-id": "CentOS",
      "os-release": "6.2",
      "os-arch": "x86_64",
      "checksum": {
        "md5": "2dba351902bffb4716168f3693e932e2"
      }
    }
  ],
  "logo": "/img/logos/view_dashboard.png",
  "name": "OpenNebula Sandbox VMware 3.4.1",
  "opennebula_template": "",
  "opennebula_version": "",
  "publisher": "C12G Labs",
  "tags": [
    "linux",
    "vmware",
    "sandbox",
    "esx",
    "frontend"
  ],
  "links": {
    "download": {
      "href": "http://marketplace.c12g.com/appliance/4fc76a938fb81d3517000004/download"
    }
  }
}
```

- Create a new image: You can use the download link as PATH in a new Image template to create am Image.

```
$ onemarket show 4fc76a938fb81d3517000004 --server http://marketplace.c12g.com
{
  ...
  "links": {
    "download": {
      "href": "http://marketplace.c12g.com/appliance/4fc76a938fb81d3517000004/download"
    }
  }
}

$ cat marketplace_image.one
NAME          = "OpenNebula Sandbox VMware 3.4.1"
PATH          = http://marketplace.c12g.com/appliance/4fc76a938fb81d3517000004/download
TYPE          = OS

$ oneimage create marketplace_image.one
ID: 1231
```

3.2 Howto Create Apps for the Marketplace

In this section some general guidelines on creating OpenNebula compatible images for the marketplace are described. Following this you will find a tutorial showing how to create an Ubuntu 12.04 image ready to distribute it in the marketplace.

3.2.1 Image Creation Guidelines

Images in the marketplace are just direct installation of OS, prepared to run with OpenNebula. There are two basic things you need to do (apart from the standard OS installation):

- Add OpenNebula contextualization script, so the image is able to receive and use context information
- Disable udev network rule writing, usually images are cloned multiple times, using different MAC addresses each time. In this case, you'll need to disable udev to prevent getting a new interface each time.

These both steps can be automated in some distributions (Debian, Ubuntu, CentOS and RHEL) using preparation packages. You can find the packages and more information about them at the *Contextualization Packages for VM Images* section.

Add OpenNebula Contextualization Script

The contextualization scripts configure the VM on startup. You can find the scripts for different distributions at the OpenNebula repository. Depending on the distribution the method of installation is different so refer to the distribution documentation to do so. Make sure that these scripts are executed before the network is initialized.

You can find more information about contextualization in the *Contextualizing Virtual Machines* guide.

Disable udev Network Rule Writing

Most linux distribution upon start search for new devices and write the configuration for them. This fixes the network device for each different network mac address. This is a bad behavir in VM images as they will be used to run with very different mac addresses. You need to disable this udev configuration saving and also delete any udev network rule that could be already saved.

3.2.2 Tutorial: Preparing an Ubuntu 12.04 Xen for the Marketplace

The installation is based on the Ubuntu documentation.

You will need a machine where xen is correctly configured, a bridge with internet connection and a public IP or a private IP with access to a router that can connecto the internet.

First we create an empty disk, in this case it will be 8 Gb:

```
$ dd if=/dev/zero of=ubuntu.img bs=1 count=1 seek=8G
```

Then we download netboot kernel and initrd compatible with Xen. We are using a mirror near to us but you can select one from the Ubuntu mirrors list:

```
$ wget http://ftp.dat.etsit.upm.es/ubuntu/dists/precise/main/installer-amd64/current/images/netboot/x
$ wget http://ftp.dat.etsit.upm.es/ubuntu/dists/precise/main/installer-amd64/current/images/netboot/x
```

Now we can create a file describing the VM where the ubuntu will be installed:

```
name = "ubuntu"

memory = 256

disk = ['file:PATH/ubuntu.img,xvda,w']
vif = ['bridge=BRIDGE']

kernel = "PATH/vmlinuz"
ramdisk = "PATH/initrd.gz"
```

Change `PATH` to the path where the VM files are located and `BRIDGE` to the name of the network bridge you are going to use. After this we can start the VM:

```
$ sudo xm create ubuntu.xen
```

To connect to the VM console and proceed with the installation you can use xm console command:

```
$ sudo xm console ubuntu
```

Use the menus to configure your VM. Make sure that you configure the network correctly as this installation will use it to download packages.

After the installation is done it will reboot again into the installation. You can exit the console pressing `<CTRL>+<]>`. Now you should shutdown the machine:

```
$ sudo xm shutdown ubuntu
```

The system is now installed in the disk image and now we must start it to configure it so it plays nice with OpenNebula. The configuratio we are going to do is:

- Disable udev network generation rules and clean any that could be saved
- Add contextualization scripts

To start the VM we will need a new xen description file:

```
name = "ubuntu1204"

memory = 512

disk = ['file:PATH/ubuntu.img,xvda,w']
vif = ['bridge=BRIDGE']

bootloader = "pygrub"
```

It is pretty similar to the other one but notice that we no longer specify kernel nor initrd and we also add the bootloader option. This will make out VM use the kernel and initrd that reside inside out VM image.

We can start it using the same command as before:

```
$ sudo xm create ubuntu-new.xen
```

And the console also works the same as before:

```
$ sudo xm console ubuntu
```

We log and become `root`. To disable udev network rule generation we should edit the file `/lib/udev/rules.d/75-persistent-net-generator.rules` and comment the line that says:

```
DRIVERS=="?*", IMPORT{program}="write_net_rules"
```

Now to make sure that no network rules are saved we can empty the rules file:

```
# echo '' > /etc/udev/rules.d/70-persistent-net.rules
```

Copy the contextualiza located at the OpenNebula repository to `/etc/init.d` and give it write permissions. This is the script that will contextualize the VM on start.

Now we modify the file `/etc/init/networking.conf` and change the line:

```
pre-start exec mkdir -p /run/network
```

with

```
pre-start script
  mkdir -p /run/network
  /etc/init.d/vmcontext
end script
```

and also in `/etc/init/network-interface.conf` we add the line:

```
/etc/init.d/vmcontext
```

so it looks similar to:

```
pre-start script
    /etc/init.d/vmcontext
    if [ "$INTERFACE" = lo ]; then
        # bring this up even if /etc/network/interfaces is broken
        ifconfig lo 127.0.0.1 up || true
        initctl emit -n net-device-up \
            IFACE=lo LOGICAL=lo ADDRFAM=inet METHOD=loopback || true
    fi
    mkdir -p /run/network
    exec ifup --allow auto $INTERFACE
end script
```

CHAPTER

FOUR

REFERENCES

4.1 Virtual Machine Definition File

A template file consists of a set of attributes that defines a Virtual Machine. Using the command `onetemplate create`, a template can be registered in OpenNebula to be later instantiated. For compatibility with previous versions, you can also create a new Virtual Machine directly from a template file, using the `onevm create` command.

Warning: There are some template attributes that can compromise the security of the system or the security of other VMs, and can be used **only** by users in the oneadmin group. These attributes can be configured in *oned.conf*, the default ones are labeled with `*` in the following tables. See the complete list in the *Restricted Attributes* section.

4.1.1 Syntax

The syntax of the template file is as follows:

- Anything behind the pound or hash sign `#` is a **comment**.
- **Strings** are delimited with double quotes `"`, if a double quote is part of the string it needs to be escaped `\\"`.
- **Single Attributes** are in the form:

```
NAME=VALUE
```

- **Vector Attributes** that contain several values can be defined as follows:

```
NAME=[NAME1=VALUE1,NAME2=VALUE2]
```

- **Vector Attributes** must contain at least one value.
- Attribute names are case insensitive, in fact the names are converted to uppercase internally.

4.1.2 XML Syntax

Since OpenNebula 3.4, template files can be in XML, with the following syntax:

- The root element must be `TEMPLATE`
- **Single Attributes** are in the form:

```
<NAME>VALUE</NAME>
```

- **Vector Attributes** that contain several values can be defined as follows:

```
<NAME>
  <NAME1>VALUE1</NAME1>
  <NAME2>VALUE2</NAME2>
</NAME>
```

A simple example:

```
<TEMPLATE>
  <NAME>test_vm</NAME>
  <CPU>2</CPU>
  <MEMORY>1024</MEMORY>
  <DISK>
    <IMAGE_ID>2</IMAGE_ID>
  </DISK>
  <DISK>
    <IMAGE>Data</IMAGE>
    <IMAGE_UNAME>oneadmin</IMAGE_UNAME>
  </DISK>
</TEMPLATE>
```

4.1.3 Capacity Section

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

Attribute	Description	Mandatory
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>`. **NOTE**: When defining a Template it is the name of the VM Template. The actual name of the VM will be set when the VM Template is instantiated.	**YES For Templates** **NO** For VMs - will be set to one-<vmid> if omitted
MEMORY	Amount of RAM required for the VM, in Megabytes.	**YES**
CPU	Percentage of CPU divided by 100 required for the Virtual Machine, half a processor is written 0.5. This value is used by OpenNebula and the scheduler to guide the host overcommitment.	**YES**
VCPU	Number of virtual cpus. This value is **optional**, the default hypervisor behavior is used, usually one virtual CPU.	**YES** - will be set to 1 if omitted, this can be changed in the driver configuration

Example:

```
NAME   = test-vm
MEMORY = 128
CPU    = 1
```

4.1.4 OS and Boot Options Section

The OS system is defined with the `OS` 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 Sub-Attribute	Description	XEN	KVM	VMWARE
ARCH	CPU architecture to virtualize	•	**M** (default i686)	**M** (default i686)
MACHINE	libvirt machine type. Check libvirt capabilities for the list of available machine types.	•	O	•
KERNEL	path to the OS kernel to boot the image in the host	O see (*)	O	•
KERNEL_DS	image to be used as kernel (see !!)	O see (*)	O	•
INITRD	path to the initrd image in the host	O (for kernel)	O (for kernel)	•
INITRD_DS	image to be used as ramdisk (see !!)	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	O see (*)	O	•
BOOT	comma separated list of boot devices types, by order of preference (first device in the list is the first device used for boot). Possible values: `hd`,`fd`,`cdrom` ,`network`	O (only HVM)	**M**	•

(*) If no `kernel`/`initrd` or `bootloader` are specified a Xen HVM will be created.

(!!) Use one of KERNEL_DS or KERNEL (and INITRD or INITRD_DS).

KERNEL_DS and INITRD_DS refer to and image registered in a File Datastore and must be of type KERNEL and RAMDISK, respectively. The image should be refer using one of the following:

- `$FILE[IMAGE=<image name>]`, to select own files
- `$FILE[IMAGE=<image name>, <IMAGE_UNAME|IMAGE_UID>=<owner name|owner id>]`, to select images owned by other users, by user name or uid.
- `$FILE[IMAGE_ID=<image id>]`, global file selection

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

```
OS = [ KERNEL     = /vmlinuz,
       INITRD     = /initrd.img,
       ROOT       = sda1,
       KERNEL_CMD = "ro xencons=tty console=tty1"]

OS = [ KERNEL_DS  = "$FILE[IMAGE=\"kernel 3.6\"]",
       INITRD_DS  = "$FILE[IMAGE=\"initrd 3.6\"]",
       ROOT       = sda1,
       KERNEL_CMD = "ro xencons=tty console=tty1"]
```

4.1.5 Features Section

This section configures the features enabled for the VM.

Note the hypervisor column states that the attribute is **O**ptional or − not supported for that hypervisor

Sub-Attribute	Description	XEN HVM	KVM
PAE	Physical address extension mode allows 32-bit guests to address more than 4 GB of memory	O	O
ACPI	Useful for power management, for example, with KVM guests it is required for graceful shutdown to work	O	O
APIC	Enables the advanced programmable IRQ management. Useful for SMP machines.	O	O
LOCALTIME	The guest clock will be synchronized to the host's configured timezone when booted. Useful for Windows VMs	•	O
HYPERV	Add hyperv extensions to the VM. The options can be configured in the driver configuration, HYPERV_OPTIONS	•	O
DEVICE_MODE	Used to change the IO emulator in Xen HVM.	O	•

```
FEATURE = [
    PAE = "yes",
    ACPI = "yes",
    APIC = "no",
    DEVICE_MODE = "qemu-dm"
]
```

4.1.6 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 three types of disks:

- Persistent disks, uses an Image registered in a Datastore mark as persistent.
- Clone disks, uses an Image registered in a Datastore. Changes to the images will be discarded. A clone disk can be saved as other image.
- Volatile disks, created on-the-fly on the target hosts. Disks are disposed when the VM is shutdown and cannot be saved_as

Persistent and Clone Disks

DISK Sub-Attribute	Description	Xen	KVM	VMware
IMAGE_ID	ID of the Image to use	**Mandatory** (no IMAGE)	**Mandatory** (no IMAGE)	**Mandatory** (no IMAGE)
IMAGE	Name of the Image to use	**Mandatory** (no IMAGE_ID)	**Mandatory** (no IMAGE_ID)	**Mandatory** (no IMAGE_ID)
IMAGE_UID	To select the IMAGE of a given user by her ID	Optional	Optional	Optional
IMAGE_UNAME	To select the IMAGE of a given user by her NAME	Optional	Optional	Optional
DEV_PREFIX	Prefix for the emulated device this image will be mounted at. For instance, `hd`, `sd`, or `vd` for KVM virtio. If omitted, the dev_prefix attribute of the Image will be used	Optional	Optional	Optional
TARGET	Device to map image disk. If set, it will overwrite the default device mapping.	Optional	Optional	Optional
DRIVER	Specific image mapping driver	Optional e.g.: `tap:aio:`,`file:`	Optional e.g.: `raw`, `qcow2`	•
CACHE	Selects the cache mechanism for the disk. Values are `default`, `none`, `writethrough`, `writeback`, `directsync` and `unsafe`. More info in the libvirt documentation	•	Optional	•
READONLY	Set how the image is exposed by the hypervisor	Optional e.g.: `yes`, `no`. This attribute should only be used for special storage configurations	Optional e.g.: `yes`, `no`. This attribute should only be used for special storage configurations	Optional e.g.: `yes`, `no`. This attribute should only be used for special storage configurations
IO	Set IO policy. Values are `threads`, `native`	•	Optional	•

Volatile DISKS

DISK Sub-Attribute	Description	XEN	KVM	VMWARE
TYPE	Type of the disk:`swap`, `fs`	Optional	Optional	Optional
SIZE	size in MB	Optional	Optional	Optional
FORMAT	filesystem for **fs** images: `ext2`, `ext3`... `raw` will not format the image.	**Mandatory** (for fs)	**Mandatory** (for fs)	**Mandatory** (for fs)
DEV_PREFIX	Prefix for the emulated device this image will be mounted at. For instance, `hd`, `sd`. If omitted, the default dev_prefix set in oned.conf will be used	Optional	Optional	Optional
TARGET	device to map disk	Optional	Optional	Optional
DRIVER	special disk mapping options. KVM: `raw`,`qcow2`. Xen: `tap:aio:`, `file:`	Optional	Optional	Optional
CACHE	Selects the cache mechanism for the disk. Values are `default`, `none`, `writethrough`, `writeback`, `directsync` and `unsafe`. More info in the libvirt documentation	•	Optional	•
READONLY	Set how the image is exposed by the hypervisor	Optional e.g.: `yes`, `no`. This attribute should only be used for special storage configurations	Optional e.g.: `yes`, `no`. This attribute should only be used for special storage configurations	Optional e.g.: `yes`, `no`. This attribute should only be used for special storage configurations
IO	Set IO policy. Values are `threads`, `native`	•	Optional	•

Disks Device Mapping

If the TARGET attribute is not set for a disk, OpenNebula will automatically assign it using the following precedence, starting with `dev_prefix + a`:

- First **OS** type Image.
- Contextualization CDROM.
- **CDROM** type Images.
- The rest of **DATABLOCK** and **OS** Images, and **Volatile** disks.

Please visit the guide for *managing images* and the *image template reference* to learn more about the different image types.

You can find a complete description of the contextualization features in the *contextualization guide*.

The default device prefix `sd` can be changed to `hd` or other prefix that suits your virtualization hypervisor requirements. You can find more information in the *daemon configuration guide*.

An Example

This a sample section for disks. There are four disks using the image repository, and two volatile ones. Note that `fs` and `swap` are generated on-the-fly:

```
# First OS image, will be mapped to sda. Use image with ID 2
DISK = [ IMAGE_ID  = 2 ]

# First DATABLOCK image, mapped to sdb.
# Use the Image named Data, owned by the user named oneadmin.
DISK = [ IMAGE        = "Data",
         IMAGE_UNAME  = "oneadmin" ]

# Second DATABLOCK image, mapped to sdc
# Use the Image named Results owned by user with ID 7.
DISK = [ IMAGE        = "Results",
         IMAGE_UID    = 7 ]

# Third DATABLOCK image, mapped to sdd
# Use the Image named Experiments owned by user instantiating the VM.
DISK = [ IMAGE        = "Experiments" ]

# Volatile filesystem disk, sde
DISK = [ TYPE   = fs,
         SIZE   = 4096,
         FORMAT = ext3 ]

# swap, sdf
DISK = [ TYPE     = swap,
         SIZE     = 1024 ]
```

Because this VM did not declare a CONTEXT or any disk using a CDROM Image, the first DATABLOCK found is placed right after the OS Image, in `sdb`. For more information on image management and moving please check the *Storage guide*.

4.1.7 Network Section

NIC Sub-Attribute	Description	Mandatory
NETWORK_ID	ID of the network to attach this device, as defined by `onevnet`. Use if no NETWORK	**Mandatory** (No NETWORK)
NETWORK	Name of the network to use (of those owned by user). Use if no NETWORK_ID	**Mandatory** (No NETWORK_ID)
NETWORK_UID	To select the NETWORK of a given user by her ID	Optional
NETWORK_UNAME	To select the NETWORK of a given user by her NAME	Optional
IP	Request an specific IP from the `NETWORK`	Optional
MAC*	Request an specific HW address from the network interface	Optional
BRIDGE	Name of the bridge the network device is going to be attached to.	Optional
TARGET	name for the tun device created for the VM	Option for KVM and VMWare
SCRIPT	name of a shell script to be executed after creating the tun device for the VM	Optional
MODEL	hardware that will emulate this network interface. With Xen this is the type attribute of the vif. In KVM you can choose `virtio` to select its specific virtualization IO framework	Optional
WHITE_PORTS_TCP	**iptables_range``**: Permits access to the VM only through the specified ports in the TCP protocol. Supersedes BLACK_PORTS_TCP if defined.	Optional
BLACK_PORTS_TCP	**iptables_range``**: Doesn't permit access to the VM through the specified ports in the TCP protocol. Superseded by WHITE_PORTS_TCP if defined.	Optional
WHITE_PORTS_UDP	**iptables_range``**: Permits access to the VM only through the specified ports in the UDP protocol. Supersedes BLACK_PORTS_UDP if defined.	Optional
BLACK_PORTS_UDP	**iptables_range``**: Doesn't permit access to the VM through the specified ports in the UDP protocol. Superseded by WHITE_PORTS_UDP if defined.	Optional
ICMP	**drop**: Blocks ICMP connections to the VM. By default it's set to accept.	Optional

Warning: The PORTS and ICMP attributes require the firewalling functionality to be configured. Please read the *firewall configuration guide*.

Example, a VM with two NIC attached to two different networks:

```
NIC = [ NETWORK_ID = 1 ]

NIC = [ NETWORK     = "Blue",
        NETWORK_UID = 0 ]
```

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

4.1.8 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	VMWARE
INPUT	Define input devices, available sub-attributes: • **TYPE**: values are `mouse` or `tablet` • **BUS**: values are `usb`, `ps2` or `xen`	O (only usb tablet is supported)	O	•
GRAPHICS	Wether the VM should export its graphical display and how, available sub-attributes: • **TYPE**: values: `vnc`, `sdl`, `spice` • **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:

```
GRAPHICS = [
  TYPE    = "vnc",
  LISTEN  = "0.0.0.0",
  PORT    = "5"]
```

Warning: 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.

Warning: If the user does not specify the port variable, OpenNebula will automatically assign `$VNC_BASE_PORT + $VMID`, allowing to generate different ports for VMs so they do not collide. The `VNC_BASE_PORT` is specified inside the `oned.conf` file.

4.1.9 Context Section

Context information is passed to the Virtual Machine via an ISO 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	Mandatory
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).	Optional
FILES *	space-separated list of paths to include in context device.	Optional
FILES_DS	space-separated list of File images to include in context device.	Optional
TARGET	device to attach the context ISO.	Optional
TOKEN	`YES` to create a token.txt file for *OneGate monitorization*	Optional
NETWORK	`YES` to fill automatically the networking parameters for each NIC, used by the *Contextualization packages*	Optional

* only for users in oneadmin group

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

Hardcoded values:

```
HOSTNAME   = "MAINHOST"
```

Using template variables

`$<template_variable>`: any single value variable of the VM template, like for example:

```
IP_GEN     = "10.0.0.$VMID"
```

`$<template_variable>[<attribute>]`: Any single value contained in a multiple value variable in the VM template, like for example:

```
IP_PRIVATE = $NIC[IP]
```

`$<template_variable>[<attribute>, <attribute2>=<value2>]`: Any single value contained in the variable of the VM template, setting one attribute to discern between multiple variables called the same way, like for example:

```
IP_PUBLIC = "$NIC[IP, NETWORK=\"Public\"]"
```

Using Virtual Network template variables

`$NETWORK[<vnet_attribute>, <NETWORK_ID|NETWORK>=<vnet_id|vnet_name>]`: Any single value variable in the Virtual Network template, like for example:

```
dns = "$NETWORK[DNS, NETWORK_ID=3]"
```

Note: The network MUST be in used by any of the NICs defined in the template. The vnet_attribute can be `TEMPLATE` to include the whole vnet template in XML (base64 encoded).

Using Image template variables

`$IMAGE[<image_attribute>, <IMAGE_ID|IMAGE>=<img_id|img_name>]`: Any single value variable in the Image template, like for example:

```
root = "$IMAGE[ROOT_PASS, IMAGE_ID=0]"
```

Note: The image MUST be in used by any of the DISKs defined in the template. The image_attribute can be `TEMPLATE` to include the whole image template in XML (base64 encoded).

Using User template variables

`$USER[<user_attribute>]`: Any single value variable in the user (owner of the VM) template, like for example:

```
ssh_key = "$USER[SSH_KEY]"
```

Note: The user_attribute can be `TEMPLATE` to include the whole user template in XML (base64 encoded).

Pre-defined variables, apart from those defined in the template you can use:

- `$UID`, the uid of the VM owner
- `$UNAME`, the name of the VM owner
- `$GID`, the id of the VM owner's group
- `$GNAME`, the name of the VM owner's group
- `$TEMPLATE`, the whole template in XML format and encoded in base64

FILES_DS, each file must be registered in a FILE_DS datastore and has to be of type CONTEXT. Use the following to select files from Files Datastores:

- `$FILE[IMAGE=<image name>]`, to select own files
- `$FILE[IMAGE=<image name>, <IMAGE_UNAME|IMAGE_UID>=<owner name|owner id>]`, to select images owned by other users, by user name or uid.
- `$FILE[IMAGE_ID=<image id>]`, global file selection

Example:

```
CONTEXT = [
  HOSTNAME    = "MAINHOST",
  IP_PRIVATE = "$NIC[IP]",
  DNS         = "$NETWORK[DNS, NAME=\"Public\"]",
  IP_GEN      = "10.0.0.$VMID",
  FILES       = "/service/init.sh /service/certificates /service/service.conf",
  FILES_DS    = "$FILE[IMAGE_ID=34] $FILE[IMAGE=\"kernel\"]",
  TARGET      = "sdc"
]
```

4.1.10 Placement Section

The following attributes placement constraints and preferences for the VM:

Attribute	Description
SCHED_REQUIREMENTS	Boolean expression that rules out provisioning hosts from list of machines suitable to run this VM.
SCHED_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.
SCHED_DS_REQUIREMENTS	Boolean expression that rules out entries from the pool of datastores suitable to run this VM.
SCHED_DS_RANK	States which attribute will be used to sort the suitable datastores for this VM. Basically, it defines which datastores are more suitable than others.

Example:

```
SCHED_REQUIREMENTS    = "CPUSPEED > 1000"
SCHED_RANK            = "FREE_CPU"
```

```
SCHED_DS_REQUIREMENTS = "NAME=GoldenCephDS"
SCHED_DS_RANK         = FREE_MB
```

Requirement Expression Syntax

The syntax of the requirement expressions is defined as:

```
stmt::= expr';'
expr::= VARIABLE '=' NUMBER
      | VARIABLE '!=' NUMBER
      | VARIABLE '>' NUMBER
      | VARIABLE '<' NUMBER
      | VARIABLE '=' STRING
      | VARIABLE '!=' STRING
      | expr '&' expr
      | expr '|' expr
      | '!' expr
      | '(' expr ')'
```

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 included in the Host template or its Cluster template can be used in the requirements. You may also use an XPath expression to refer to the attribute.

There is a special variable, `CURRENT_VMS`, that can be used to deploy VMs in a Host where other VMs are (not) running. It can be used only with the operators '=' and '!='

Warning: Check the *Monitoring Subsystem* guide to find out how to extend the information model and add any information probe to the Hosts.

Warning: There are some predefined variables that can be used: `NAME`, `MAX_CPU`, `MAX_MEM`, `FREE_MEM`, `FREE_CPU`, `USED_MEM`, `USED_CPU`, `HYPERVISOR`

Examples:

```
# Only aquila hosts (aquila0, aquila1...), note the quotes
SCHED_REQUIREMENTS = "NAME = \"aquila*\""

# Only those resources with more than 60% of free CPU
SCHED_REQUIREMENTS = "FREE_CPU > 60"

# Deploy only in the Host where VM 5 is running
SCHED_REQUIREMENTS = "CURRENT_VMS = 5"

# Deploy in any Host, except the ones where VM 5 or VM 7 are running
SCHED_REQUIREMENTS = "(CURRENT_VMS != 5) & (CURRENT_VMS != 7)"
```

Warning: 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.

```
SCHED_REQUIREMENTS = "HYPERVISOR=\"vmware\""
```

Warning: Template variables can be used in the SCHED_REQUIREMENTS section.

- `$<template_variable>`: any single value variable of the VM template.
- `$<template_variable>[<attribute>]`: Any single value contained in a multiple value variable in the VM template.
- `$<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.

For example, if you have a custom probe that generates a MACS attribute for the hosts, you can do short of a MAC pinning, so only VMs with a given MAC runs in a given host.

```
SCHED_REQUIREMENTS = "MAC=\"$NIC[MAC]\""
```

Rank Expression Syntax

The syntax of the rank expressions is defined as:

```
stmt::= expr';'
expr::= VARIABLE
      | NUMBER
      | expr '+' expr
      | expr '-' expr
      | expr '*' expr
      | expr '/' expr
      | '-' expr
      | '(' expr ')'
```

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.

Warning: 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.

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

Examples:

```
# First those resources with a higher Free CPU
  SCHED_RANK = "FREE_CPU"

# Consider also the CPU temperature
  SCHED_RANK = "FREE_CPU * 100 - TEMPERATURE"
```

4.1.11 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	VMWARE
TYPE	Possible values are: `kvm`, `xen`, `vmware`	O	O	O
DATA	Raw data to be passed directly to the hypervisor	O	O	O
DATA_VMX	Raw data to be added directly to the .vmx file	•	•	O

Example:

Add a custom builder and bootloader to a Xen VM:

```
RAW     = [
      TYPE  = "xen",
      DATA  = "builder=\"linux\"
               bootloader=\"/usr/lib/xen/boot/domUloader.py\"
               bootargs=\"--entry=xvda2:/boot/vmlinuz-xenpae,/boot/vmlinuz-xenpae\"" ]
```

Add a guest type and a specific scsi controller to a vmware VM:

```
RAW = [
  TYPE     = "vmware",
  DATA     = "<devices><controller type='scsi' index='0' model='lsilogic'/></devices>",
  DATA_VMX = "pciBridge0.present = \"TRUE\"\nguestOS=\"windows7srv-64\""
]
```

4.1.12 Restricted Attributes

All the **default** restricted attributes to users in the oneadmin group are summarized in the following list:

- CONTEXT/FILES
- DISK/SOURCE
- NIC/MAC
- NIC/VLAN_ID
- SCHED_RANK

These attributes can be configured in *oned.conf*.

4.2 Image Definition Template

This page describes how to define a new image template. An image template follows the same syntax as the *VM template*.

If you want to learn more about the image repository, you can do so *here*.

Warning: There are some template attributes that can compromise the security of the system or the security of other VMs, and can be used **only** by users in the oneadmin group. These attributes can be configured in *oned.conf*, the default ones are labeled with `*` in the following tables. See the complete list in the *Restricted Attributes* section.

4.2.1 Template Attributes

The following attributes can be defined in the template.

Attribute	M / O	Value	Description
NAME	Mandatory	Any string	Name that the Image will get. Every image must have a unique name.
DESCRIPTION	Optional	Any string	Human readable description of the image for other users.
TYPE	Optional	`OS, CDROM, DATABLOCK, KERNEL, RAMDISK, CONTEXT`	Type of the image, explained in detail in the following section. If omitted, the default value is the one defined in *oned.conf* (install default is `OS`).
PERSISTENT	Optional	`YES, NO`	Persistence of the image. If omitted, the default value is `NO`.
PERSISTENT_TYPE	Optional	`IMMUTABLE`	An special persistent image, that will not be modified. This attribute should only be used for special storage configurations.
DEV_PREFIX	Optional	Any string	Prefix for the emulated device this image will be mounted at. For instance, `hd`, `sd`, or `vd` for KVM virtio. If omitted, the default value is the one defined in *oned.conf* (installation default is `hd`).
TARGET	Optional	Any string	Target for the emulated device this image will be mounted at. For instance, `hdb`, `sdc`. If omitted, it will be *assigned automatically*.
DRIVER	Optional	KVM: `raw`, `qcow2` Xen:`tap:aio:`, `file:`	Specific image mapping driver. VMware is unsupported
PATH	Mandatory (if no SOURCE)	Any string	Path to the original file that will be copied to the image repository. If not specified for a DATABLOCK type image, an empty image will be created. Note that gzipped files are supported and OpenNebula will automatically decompress them. Bzip2 compressed files is also supported, but it's strongly discouraged since OpenNebula will not calculate it's size properly.
SOURCE*	Mandatory (if no PATH)	Any string	Source to be used in the DISK attribute. Useful for not file-based images.
DISK_TYPE	Optional	`BLOCK`, `CDROM` or `FILE` (default).	This is the type of the supporting media for the image: a block device (`BLOCK`) an ISO-9660 file or readonly block device (`CDROM`) or a plain file (`FILE`).
READONLY	Optional	`YES`, `NO`.	This attribute should only be used for special storage configurations. It sets how the image is going to be exposed to the hypervisor. Images of type `CDROM` and those with PERSISTENT_TYPE set to `IMMUTABLE` will have `READONLY` set to `YES`. Otherwise, by default it is set to `NO`.
CLONE_FSTYPE	Optional	`thin,zeroedthick, eagerzeroedthick ,thick,thin`	Only for VMware images ion `vmfs` datastores. Sets the format of the target image when cloning within the datstore. More information on types.
MD5	Optional	An md5 hash	MD5 hash to check for image integrity
SHA1	Optional	An sha1 hash	SHA1 hash to check for image integrity

Warning: Be careful when `PATH` points to a compressed bz2 image, since although it will work, OpenNebula will not calculate its size correctly.

Mandatory attributes for `DATABLOCK` images with no `PATH` set:

Attribute	Value	Description
SIZE	An integer	Size in MB.
FSTYPE	String	Type of file system to be built. **Plain**. When the disk image is used directly by the hypervisor we can format the image, and so it is ready to be used by the guest OS. **Values**: `ext2`, `ext3`, `ext4`, `ntfs`, `reiserfs`, `jfs`, `swap`. Any other fs supported by mkfs will work if no special option is needed. **Formatted**. The disk image is stored in a hypervisor specific format VMDK or Qcow2. Then we cannot really make a filesystem on the image, just create the device and let the guest OS format the disk. Use `raw` to not to format the new image. **Values**: `raw`, `qcow2`, `vmdk_*`.

4.2.2 Template Examples

Example of an OS image:

```
NAME          = "Ubuntu Web Development"
PATH          = /home/one_user/images/ubuntu_desktop.img
DESCRIPTION   = "Ubuntu 10.04 desktop for Web Development students.
Contains the pdf lessons and exercises as well as all the necessary
programming tools and testing frameworks."
```

Example of a CDROM image:

```
NAME          = "MATLAB install CD"
TYPE          = CDROM
PATH          = /home/one_user/images/matlab.iso
DESCRIPTION   = "Contains the MATLAB installation files. Mount it to install MATLAB on new OS images
```

Example of a DATABLOCK image:

```
NAME          = "Experiment results"
TYPE          = DATABLOCK
# No PATH set, this image will start as a new empty disk
SIZE          = 3.08
FSTYPE        = ext3
DESCRIPTION   = "Storage for my Thesis experiments."
```

4.2.3 Restricted Attributes

All the **default** restricted attributes to users in the oneadmin group are summarized in the following list:

- SOURCE

4.3 Virtual Network Definition File

This page describes how to define a new Virtual Network template. A Virtual Network template follows the same syntax as the *VM template*.

If you want to learn more about the Virtual Network management, you can do so *here*.

4.3.1 Common Attributes

There are two types of Virtual Networks, ranged and fixed. Their only difference is how the leases are defined in the template.

These are the common attributes for both types of VNets:

Attribute	Value	Description	Mandatory
NAME	String	Name of the Virtual Network	YES
BRIDGE	String	Name of the physical bridge in the physical host where the VM should connect its network interface	YES if PHYDEV is not set
TYPE	RANGED/FIXED	Type of this VNet	YES
VLAN	YES/NO	Whether or not to isolate this virtual network using the *Virtual Network Manager drivers*. If omitted, the default value is NO.	NO
VLAN_ID	Integer	Optional VLAN id for the *802.1Q* and *Open vSwitch* networking drivers.	NO
PHYDEV	String	Name of the physical network device that will be attached to the bridge.	YES for *802.1Q* driver
SITE_PREFIX	String	IPv6 unicast local addresses (ULAs). Must be a valid IPv6	Optional
GLOBAL_PREFIX	String	IPv6 global unicast addresses. Must be a valid IPv6	Optional

Please note that any arbitrary value can be set in the Virtual Network template, and then used in the *contextualization* section of the VM. For instance, `NETWORK\_GATEWAY="x.x.x.x"` might be used to define the Virtual Network, and then used in the context section of the VM to configure its network to connect through the GATEWAY.

If you need OpenNebula to generate IPv6 addresses, that can be later used in context or for Virtual Router appliances, you can use the `GLOBAL_PREFIX` and `SITE_PREFIX` attributes

Attributes Used for Contextualization

Attribute	Description
NETWORK_ADDRESS	Base network address
NETWORK_MASK	Network mask
GATEWAY	Router for this network, do not set when the network is not routable
DNS	Specific DNS for this network
GATEWAY6	IPv6 router for this network
CONTEXT_FORCE_IPV4	When a vnet is IPv6 the IPv4 is not configured unless this attribute is set

4.3.2 Leases

A lease is a definition of an IP-MAC pair. From an IP address, OpenNebula generates an associated MAC using the following rule: `MAC = MAC_PREFFIX:IP`. All Virtual Networks share a default value for the MAC_PREFIX, set in the `oned.conf` file.

So, for example, from IP 10.0.0.1 and MAC_PREFFIX 02:00, we get 02:00:0a:00:00:01.

The available leases for new VNets are defined differently for each type.

Fixed Virtual Networks

Fixed VNets need a series of `LEASES` vector attributes, defined with the following sub-attributes:

Sub-Attribute	Value	Description	Mandatory
IP	IP address	IP for this lease	YES
MAC	MAC address	MAC associated to this IP	NO

Warning: The optional MAC attribute will overwrite the default MAC_PREFIX:IP rule. Be aware that this will break the default *contextualization mechanism*.

Ranged Virtual Networks

Instead of a list of `LEASES`, ranged Virtual Networks contain a range of IPs that can be defined in a flexible way using these attributes:

Attribute	Value	Description
NETWORK_ADDRESS	IP address, optionally in CIDR notation	Base network address to generate IP addresses.
NETWORK_SIZE	`A`, `B`, `C`, or Number	Number of VMs that can be connected using this network. It can be defined either using a number or a network class (A, B or C). The default value for the network size can be found in `oned.conf`.
NETWORK_MASK	Mask in dot-decimal notation	Network mask for this network.
IP_START	IP address	First IP of the range.
IP_END	IP address	Last IP of the range.
MAC_START	MAC address	First MAC of the range.

The following examples define the same network range, from 10.10.10.1 to 10.10.10.254:

```
NETWORK_ADDRESS = 10.10.10.0
NETWORK_SIZE    = C

NETWORK_ADDRESS = 10.10.10.0
NETWORK_SIZE    = 254

NETWORK_ADDRESS = 10.10.10.0/24

NETWORK_ADDRESS = 10.10.10.0
NETWORK_MASK    = 255.255.255.0
```

You can change the first and/or last IP of the range:

```
NETWORK_ADDRESS = 10.10.10.0/24
IP_START        = 10.10.10.17
```

Or define the range manually:

```
IP_START        = 10.10.10.17
IP_END          = 10.10.10.41
```

Finally, you can define the network by just specifying the MAC address set (specially in IPv6). The following is equivalent to the previous examples but with MACs:

```
MAC_START    = 02:00:0A:0A:0A:11
NETWORK_SIZE = 254
```

Warning: With either of the above procedures, no matter if you are defining the set using IPv4 networks, OpenNebula will generate IPv6 addresses if the GLOBAL_PREFIX and/or SITE_PREFIX is added to the network template. Note that the link local IPv6 address will be always generated.

4.3.3 Examples

Sample fixed VNet:

```
NAME    = "Blue LAN"
TYPE    = FIXED

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

LEASES  = [IP=130.10.0.1]
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]

# Custom Attributes to be used in Context
GATEWAY = 130.10.0.1
DNS     = 130.10.0.1

LOAD_BALANCER = 130.10.0.4
```

Sample ranged VNet:

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

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

NETWORK_ADDRESS = 192.168.0.0/24
IP_START        = 192.168.0.3

# Custom Attributes to be used in Context
GATEWAY = 192.168.0.1
DNS     = 192.168.0.1

LOAD_BALANCER = 192.168.0.2
```

4.4 Command Line Interface

OpenNebula provides a set commands to interact with the system:

4.4.1 CLI

- oneacct: gets accounting data from OpenNebula
- oneacl: manages OpenNebula ACLs
- onecluster: manages OpenNebula clusters
- onedatastore: manages OpenNebula datastores
- onedb: OpenNebula database migration tool
- onegroup: manages OpenNebula groups
- onehost: manages OpenNebula hosts
- oneimage: manages OpenNebula images
- onetemplate: manages OpenNebula templates
- oneuser: manages OpenNebula users
- onevdc: manages OpenNebula Virtual DataCenters
- onevm: manages OpenNebula virtual machines
- onevnet: manages OpenNebula networks
- onezone: manages OpenNebula zones

The output of these commands can be customized by modifying the configuration files that can be found in `/etc/one/cli/`. They also can be customized on a per-user basis, in this case the configuration files should be placed in `$HOME/.one/cli`.

4.4.2 OCCI Commands

- occi-compute: manages compute objects
- occi-network: manages network objects
- occi-storage: manages storage objects
- occi-instance-type: Retrieve instance types

4.4.3 ECONE Commands

- econe-upload: Uploads an image to OpenNebula
- econe-describe-images: Lists all registered images belonging to one particular user.
- econe-run-instances: Runs an instance of a particular image (that needs to be referenced).
- econe-describe-instances: Outputs a list of launched images belonging to one particular user.
- econe-terminate-instances: Shutdowns a set ofvirtual machines (or cancel, depending on its state).
- econe-reboot-instances: Reboots a set ofvirtual machines.
- econe-start-instances: Starts a set ofvirtual machines.
- econe-stop-instances: Stops a set ofvirtual machines.
- econe-create-volume: Creates a new DATABLOCK in OpenNebula
- econe-delete-volume: Deletes an existing DATABLOCK.

- econe-describe-volumes: Describe all available DATABLOCKs for this user
- econe-attach-volume: Attaches a DATABLOCK to an instance
- econe-detach-volume: Detaches a DATABLOCK from an instance
- econe-allocate-address: Allocates a new elastic IP address for the user
- econe-release-address: Releases a publicIP of the user
- econe-describe-addresses: Lists elastic IP addresses
- econe-associate-address: Associates a publicIP of the user with a given instance
- econe-disassociate-address: Disasociate a publicIP of the user currently associated with an instance
- econe-create-keypair: Creates the named keypair
- econe-delete-keypair: Deletes the named keypair, removes the associated keys
- econe-describe-keypairs: List and describe the key pairs available to the user
- econe-register: Registers an image

4.4.4 oneFlow Commands

- oneflow: oneFlow Service management
- oneflow-template: oneFlow Service Template management