OpenNebula 5.0 Operation Guide

Release 5.0.2

OpenNebula Systems

Jul 21, 2016

CHAPTER

ONE

HOSTS AND CLUSTERS MANAGEMENT

1.1 Overview

A **Host** is a server that has the ability to run Virtual Machines and that is connected to OpenNebula's Front-end server. OpenNebula can work with Hosts with a heterogeneous configuration, i.e. you can connect Hosts to the same OpenNebula with different hypervisors or Linux distributions. To learn how to prepare the hosts you can read the Node Installation guide.

Clusters are pools of hosts that share datastores and virtual networks.

1.1.1 How Should I Read This Chapter

In this chapter there are four guides describing these objects.

- **Host Management**: Host management is achieved through the `onehost` CLI command or through the Sunstone GUI. You can read about Host Management in more detail in the *Managing Hosts* guide.
- **Cluster Management**: Hosts can be grouped in Clusters. These Clusters are managed with the `onecluster` CLI command, or through the Sunstone GUI. You can read about Cluster Management in more detail in the *Managing Clusters* guide.
- **Scheduler**: Where you'll learn how to change the scheduling configuration to suit your needs. For example changing the scheduling policies or the number of VMs that will be sent per host.
- **Datastore**: Where you'll learn about how to confgure and manage the different types of datastore types.

You should read all the guides in this chapter to familiarize with these objects. For small and homogeneous clouds you may not need to create new clusters.

1.1.2 Hypervisor Compatibility

These guides are compatible with both KVM and vCenter hypervisors.

1.2 Hosts

In order to use your existing physical nodes, you have to add them to the system as OpenNebula Hosts. To add a host only its hostname and type is needed. Hosts are usually organized in Clusters, you can read more about it in the *Managing Clusters* guide.

Warning: Before adding a KVM host check that you can ssh to it without being prompt for a password.

1.2.1 Create and Delete Hosts

Hosts are the servers managed by OpenNebula responsible for Virtual Machine execution. To use these hosts in OpenNebula you need to register them so they are monitored and made available to the scheduler.

Creating a host:

```
$ onehost create host01 --im kvm --vm kvm
ID: 0
```

The parameters are:

- `--im/-i`: Information Manager driver.
- `--vm/-v`: Virtual Machine Manager driver.

To remove a host, just like with other OpenNebula commands, you can either specify it by ID or by name. The following commands are equivalent:

```
$ onehost delete host01
$ onehost delete 0
```

1.2.2 Showing and Listing Hosts

To display information about a single host the `show` command is used:

```
HOST 0 INFORMATION
ID                    : 0
NAME                  : server
CLUSTER               : server
STATE                 : MONITORED
IM_MAD                : kvm
VM_MAD                : kvm
LAST MONITORING TIME  : 05/28 00:30:51

HOST SHARES
TOTAL MEM             : 7.3G
USED MEM (REAL)       : 4.4G
USED MEM (ALLOCATED)  : 1024M
TOTAL CPU             : 400
USED CPU (REAL)       : 28
USED CPU (ALLOCATED)  : 100
RUNNING VMS           : 1

LOCAL SYSTEM DATASTORE #0 CAPACITY
TOTAL:                : 468.4G
USED:                 : 150.7G
FREE:                 : 314.7G

MONITORING INFORMATION
ARCH="x86_64"
CPUSPEED="1599"
HOSTNAME="server"
```

```
HYPERVISOR="kvm"
IM_MAD="kvm"
MODELNAME="Intel(R) Core(TM) i7-4650U CPU @ 1.70GHz"
NETRX="0"
NETTX="0"
RESERVED_CPU=""
RESERVED_MEM=""
VERSION="5.00.0"
VM_MAD="kvm"

WILD VIRTUAL MACHINES

NAME                                                IMPORT_ID   CPU     MEMORY

VIRTUAL MACHINES

    ID USER     GROUP    NAME            STAT UCPU    UMEM HOST             TIME
    13 oneadmin oneadmin kvm1-13         runn  0.0   1024M server       8d 06h14
```

The information of a host contains:

- General information of the hosts including its name and the drivers used to interact with it.
- Capacity information (*Host Shares*) for CPU and memory.
- Local datastore information (*Local System Datastore*) if the Host is configured to use a local datastore (e.g. Filesystem in ssh transfer mode).
- Monitoring Information, including PCI devices
- Virtual Machines running on the hosts. *Wild* are virtual machines running on the host but not started by OpenNebula, they can be imported into OpenNebula.

To see a list of all the hosts:

```
$ onehost list
      ID NAME            CLUSTER    RVM      ALLOCATED_CPU      ALLOCATED_MEM STAT
       0 server          server       1    100 / 400 (25%) 1024M / 7.3G (13%) on
       1 kvm1            kvm          0                   -                   - off
       2 kvm2            kvm          0                   -                   - off
```

The above information can be also displayed in XML format using `-x`.

1.2.3 Host Life-cycle: Enable, Disable, Offline and Flush

In order to manage the life cycle of a host it can be set to different operation modes: enabled (on), disabled (dsbl) and offline (off). The different operation status for each mode is described by the following table:

OP. MODE	MONITORING	VM DEPLOYMENT		MEANING
		MAN-UAL	SCHED	
ENABLED (on)	Yes	Yes	Yes	The host is fully operational
UPDATE (update)	Yes	Yes	Yes	The host is being monitored
DISABLED (dsbl)	Yes	Yes	No	Disabled, e.g. to perform maintenance operations
OFFLINE (off)	No	No	No	Host is totally offline
ERROR (err)	Yes	Yes	No	Error while monitoring the host, use `onehost show` for the error description.
RETRY (retry)	Yes	Yes	No	Monitoring a host in error state

The `onehost` tool includes three commands to set the operation mode of a host: `disable`, `offline` and `enable`, for example:

```
$ onehost disable 0
```

To re-enable the host use the `enable` command:

```
$ onehost enable 0
```

Similarly to put the host offline:

```
$ onehost offline 0
```

The `flush` command will mark all the running VMs in the specified host as to be rescheduled, which means that they will be migrated to another server with enough capacity. At the same time, the specified host will be disabled, so no more Virtual Machines are deployed in it. This command is useful to clean a host of running VMs.

1.2.4 Custom Host Tags & Scheduling Policies

The Host attributes are inserted by the monitoring probes that run from time to time on the nodes to get information. The administrator can add custom attributes either creating a probe in the host, or updating the host information with: `onehost update`.

For example to label a host as *production* we can add a custom tag *TYPE*:

```
    $ onehost update
    ...
TYPE="production"
```

This tag can be used at a later time for scheduling purposes by adding the following section in a VM template:

```
SCHED_REQUIREMENTS="TYPE=\"production\""
```

That will restrict the Virtual Machine to be deployed in `TYPE=production` hosts. The scheduling requirements can be defined using any attribute reported by `onehost show`, see the *Scheduler Guide* for more information.

This feature is useful when we want to separate a series of hosts or marking some special features of different hosts. These values can then be used for scheduling the same as the ones added by the monitoring probes, as a *placement requirement*.

1.2.5 Update Host Drivers

When OpenNebula monitors a host, it copies driver files to `/var/tmp/one`. When these files are updated, they need to be copied again to the hosts with the `sync` command. To keep track of the probes version there's a file in `/var/lib/one/remotes/VERSION`. By default this holds the OpenNebula version (e.g. '5.0.0'). This version can be seen in he hosts with a `onehost show <host>`:

```
$ onehost show 0
HOST 0 INFORMATION
ID                    : 0
[...]
MONITORING INFORMATION
VERSION="5.0.0"
[...]
```

The command `onehost sync` only updates the hosts with `VERSION` lower than the one in the file `/var/lib/one/remotes/VERSION`. In case you modify the probes this `VERSION` file should be modified with a greater value, for example `5.0.0.01`.

In case you want to force upgrade, that is, no `VERSION` checking you can do that adding `--force` option:

```
$ onehost sync --force
```

You can also select which hosts you want to upgrade naming them or selecting a cluster:

```
$ onehost sync host01,host02,host03
$ onehost sync -c myCluster
```

`onehost sync` command can alternatively use `rsync` as the method of upgrade. To do this you need to have installed `rsync` command in the frontend and the nodes. This method is faster that the standard one and also has the benefit of deleting remote files no longer existing in the frontend. To use it add the parameter `--rsync`:

```
$ onehost sync --rsync
```

1.2.6 Host Information

Hosts include the following monitoring information. You can use this variables to create custom `RANK` and `REQUIREMENTS` expressions for scheduling. Note also that you can manually add any tag and use it also for `RANK` and `REQUIREMENTS`

Key	Description
HYPERVISOR	Name of the hypervisor of the host, useful for selecting the hosts with an specific technology.
ARCH	Architecture of the host CPU's, e.g. x86_64.
MODELNAME	Model name of the host CPU, e.g. Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz.
CPUSPEED	Speed in MHz of the CPU's.
HOSTNAME	As returned by the `hostname` command.
VERSION	This is the version of the monitoring probes. Used to control local changes and the update process
MAX_CPU	Number of CPU's multiplied by 100. For example, a 16 cores machine will have a value of 1600. The value of RESERVED_CPU will be subtracted from the information reported by the monitoring system. This value is displayed as `TOTAL CPU` by the `onehost show` command under `HOST SHARE` section.
MAX_MEM	Maximum memory that could be used for VMs. It is advised to take out the memory used by the hypervisor using RESERVED_MEM. This values is subtracted from the memory amount reported. This value is displayed as `TOTAL MEM` by the `onehost show` command under `HOST SHARE` section.
MAX_DISK	Total space in megabytes in the DATASTORE LOCATION.
USED_CPU	Percentage of used CPU multiplied by the number of cores. This value is displayed as `USED CPU (REAL)` by the `onehost show` command under `HOST SHARE` section.
USED_MEM	Memory used, in kilobytes. This value is displayed as `USED MEM (REAL)` by the `onehost show` command under `HOST SHARE` section.
USED_DISK	Used space in megabytes in the DATASTORE LOCATION.
FREE_CPU	Percentage of idling CPU multiplied by the number of cores. For example, if 50% of the CPU is idling in a 4 core machine the value will be 200.
FREE_MEM	Available memory for VMs at that moment, in kilobytes.
FREE_DISK	Free space in megabytes in the DATASTORE LOCATION
CPU_USAGE	Total CPU allocated to VMs running on the host as requested in `CPU` in each VM template. This value is displayed as `USED CPU (ALLOCATED)` by the `onehost show` command under `HOST SHARE` section.
MEM_USAGE	Total MEM allocated to VMs running on the host as requested in `MEMORY` in each VM template. This value is displayed as `USED MEM (ALLOCATED)` by the `onehost show` command under `HOST SHARE` section.
DISK_USAGE	Total size allocated to disk images of VMs running on the host computed using the `SIZE` attribute of each image and considering the datastore characteristics.
NETRX	Received bytes from the network
NETTX	Transferred bytes to the network
WILD	Comma separated list of VMs running in the host that were not launched and are not currently controlled by OpenNebula
ZOMBIES	Comma separated list of VMs running in the host that were launched by OpenNebula but are not currently controlled by it.

1.2.7 Importing Wild VMs

The monitoring mechanism in OpenNebula reports all VMs found in a hypervisor, even those not launched through OpenNebula. These VMs are referred to as Wild VMs, and can be imported to be managed through OpenNebula. This includes all supported hypervisors, even the hybrid ones.

The Wild VMs can be spotted through the `onehost show` command:

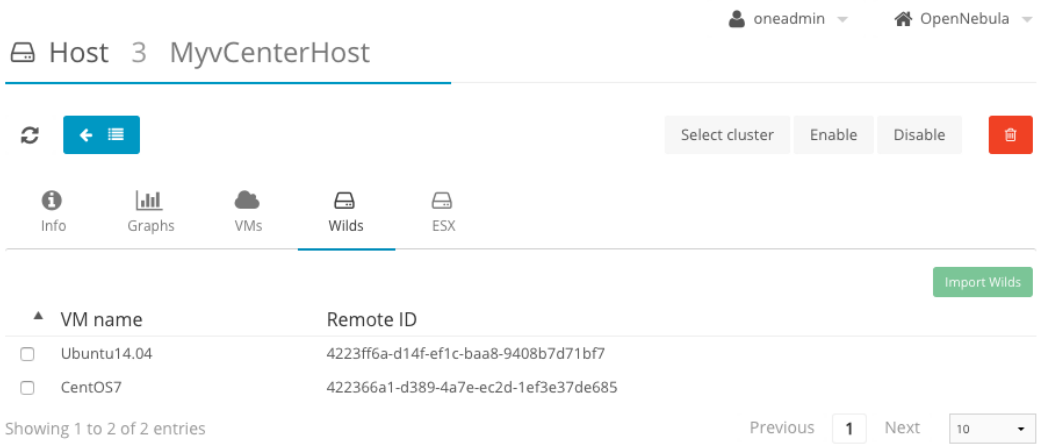
```
$ onehost show 3
HOST 3 INFORMATION
ID                    : 3
NAME                  : MyvCenterHost
CLUSTER               : -
STATE                 : MONITORED
[...]
WILD VIRTUAL MACHINES
                     NAME                                 IMPORT_ID  CPU      MEMORY
           Ubuntu14.04VM 4223f951-243a-b31a-018f-390a02ff5c96    1       2048
                 CentOS7 422375e7-7fc7-4ed1-e0f0-fb778fe6e6e0    1       2048
```

And imported through the `onehost importvm` command:

```
$ onehost importvm 0 CentOS7
$ onevm list
ID USER     GROUP    NAME            STAT UCPU    UMEM HOST               TIME
 3 oneadmin oneadmin CentOS7         runn    0    590M MyvCenterHost   0d 01h02
```

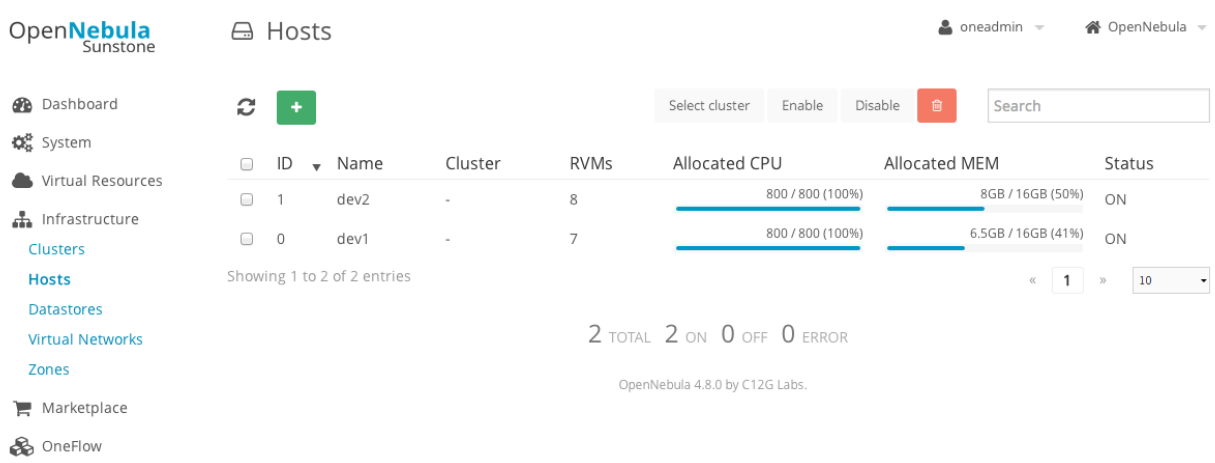
After a Virtual Machine is imported, their life-cycle (including creation of snapshots) can be controlled through OpenNebula. However, some operations *cannot* be performed on an imported VM, including: poweroff, undeploy, migrate or delete-recreate.

The same import mechanism is available graphically through Sunstone. Running and Powered Off VMs can be imported through the WILDS tab in the Host info tab.



1.2.8 Using Sunstone to Manage Hosts

You can also manage your hosts using Sunstone. Select the Host tab, and there, you will be able to create, enable, disable, delete and see information about your hosts in a user friendly way.



1.3 Clusters

A Cluster is a group of *Hosts*. Clusters can have associated Datastores and *Virtual Networks*, this is how the administrator sets which Hosts have the underlying requirements for each Datastore and Virtual Network configured.

1.3.1 Cluster Management

Clusters are managed with the *"onecluster" command*. To create new Clusters, use `onecluster create <name>`. Existing Clusters can be inspected with the `onecluster list` and `show` commands.

```
$ onecluster list
  ID NAME            HOSTS NETS  DATASTORES

$ onecluster create production
ID: 100

$ onecluster list
  ID NAME            HOSTS NETS  DATASTORES
 100 production      0     0     0

$ onecluster show production
CLUSTER 100 INFORMATION
ID             : 100
NAME           : production

HOSTS

VNETS

DATASTORES
```

Add Hosts to Clusters

Hosts can be created directly in a Cluster, using the `--cluster` option of `onehost create`, or be added at any moment using the command `onecluster addhost`. Hosts can be in only one Cluster at a time.

To delete a Host from a Cluster, the command `onecluster delhost` must be used. A Host needs to belong to a Cluster, so it will be moved to the `default` cluster.

In the following example, we will add Host 0 to the Cluster we created before. You will notice that the `onecluster show` command will list the Host ID 0 as part of the Cluster.

```
$ onehost list
  ID NAME          CLUSTER     RVM   TCPU   FCPU   ACPU   TMEM   FMEM   AMEM STAT
   0 host01        -             7    400    290    400   3.7G   2.2G   3.7G   on

$ onecluster addhost production host01

$ onehost list
  ID NAME          CLUSTER     RVM   TCPU   FCPU   ACPU   TMEM   FMEM   AMEM STAT
   0 host01        producti      7    400    290    400   3.7G   2.2G   3.7G   on

$ onecluster show production
CLUSTER 100 INFORMATION
ID             : 100
NAME           : production

HOSTS
0

VNETS

DATASTORES
```

Add Resources to Clusters

Datastores and Virtual Networks can be added to multiple Clusters. This means that any Host in those Clusters is properly configured to run VMs using Images from the Datastores, or is using leases from the Virtual Networks.

For instance, if you have several Hosts configured to use a given Open vSwitch network, you would group them in the same Cluster. The *Scheduler* will know that VMs using these resources can be deployed in any of the Hosts of the Cluster.

These operations can be done with the `onecluster addvnet/delvnet` and `adddatastore/deldatastore`:

```
$ onecluster addvnet production priv-ovswitch

$ onecluster adddatastore production iscsi

$ onecluster list
  ID NAME            HOSTS NETS  DATASTORES
 100 production      1     1     1

$ onecluster show 100
CLUSTER 100 INFORMATION
ID             : 100
NAME           : production

CLUSTER TEMPLATE

HOSTS
0
```

```
VNETS
1

DATASTORES
100
```

The System Datastore for a Cluster

In order to create a complete environment where the scheduler can deploy VMs, your Clusters need to have at least one System DS.

You can add the default System DS (ID: 0), or create a new one to improve its performance (e.g. balance VM I/O between different servers) or to use different system DS types (e.g. shared and ssh).

To use a specific System DS with your cluster, instead of the default one, just create it (with TYPE=SYSTEM_DS in its template), and associate it just like any other datastore (onecluster adddatastore).

Cluster Properties

Each cluster includes a generic template where cluster configuration properties or attributes can be defined. The following list of attributes are recognized by OpenNebula:

Attribute	Description
RESERVED_CPU	In percentage. Applies to all the Hosts in this cluster. It will be subtracted from the TOTAL CPU. See *scheduler*.
RESERVED_MEM	In KB. Applies to all the Hosts in this cluster. It will be subtracted from the TOTAL MEM. See *scheduler*.

You can easily update these values with the `onecluster update` command. Also, you can add as many variables as you want, following the standard template syntax. These variables will be used for now only for informational purposes.

1.3.2 Scheduling and Clusters

Automatic Requirements

When a Virtual Machine uses resources (Images or Virtual Networks) from a Cluster, OpenNebula adds the following *requirement* to the template:

```
$ onevm show 0
[...]
AUTOMATIC_REQUIREMENTS="CLUSTER_ID = 100"
```

Because of this, if you try to use resources that do not belong to the same Cluster, the Virtual Machine creation will fail with a message similar to this one:

```
$ onetemplate instantiate 0
[TemplateInstantiate] Error allocating a new virtual machine. Incompatible cluster␣
↪IDs.
DISK [0]: IMAGE [0] from DATASTORE [1] requires CLUSTER [101]
NIC [0]: NETWORK [1] requires CLUSTER [100]
```

Manual Requirements and Rank

The placement attributes *SCHED_REQUIREMENTS and SCHED_RANK* can use attributes from the Cluster template. Let's say you have the following scenario:

```
$ onehost list
  ID NAME            CLUSTER   RVM      ALLOCATED_CPU      ALLOCATED_MEM STAT
   1 host01          cluster_a   0       0 / 200 (0%)      0K / 3.6G (0%) on
   2 host02          cluster_a   0       0 / 200 (0%)      0K / 3.6G (0%) on
   3 host03          cluster_b   0       0 / 200 (0%)      0K / 3.6G (0%) on

$ onecluster show cluster_a
CLUSTER TEMPLATE
QOS="GOLD"

$ onecluster show cluster_b
CLUSTER TEMPLATE
QOS="SILVER"
```

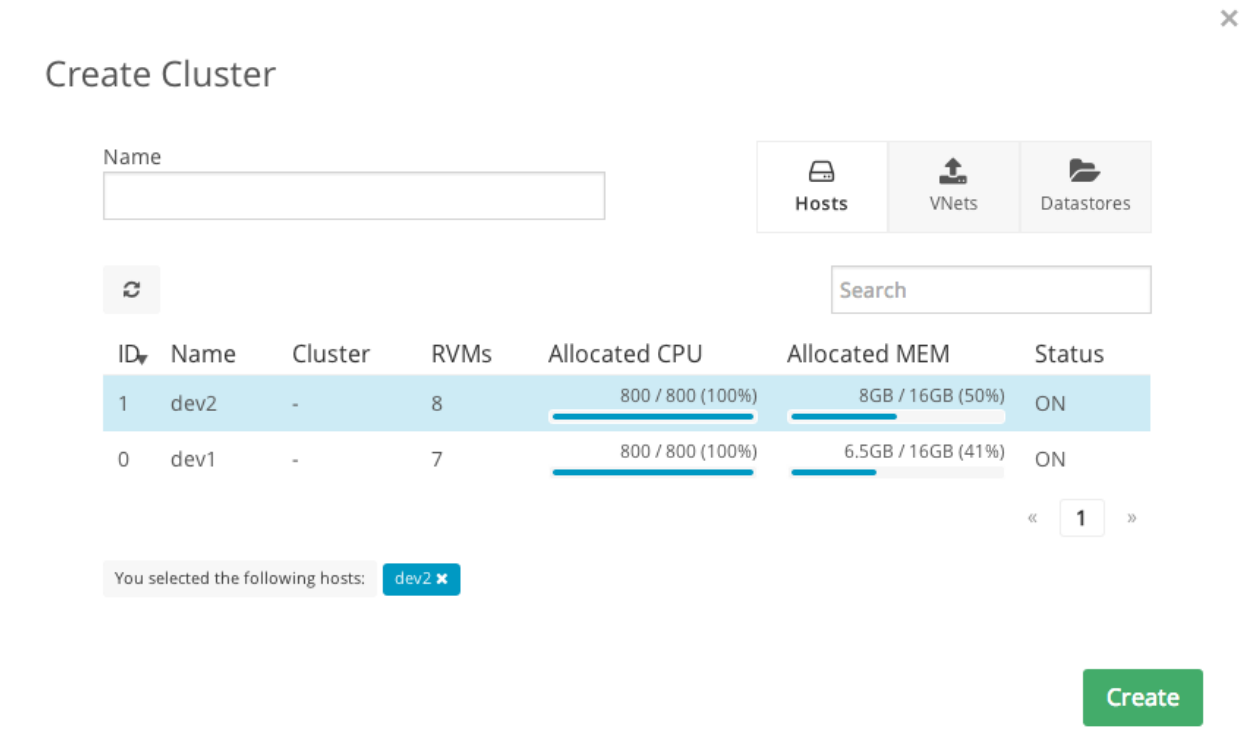
You can use these expressions:

```
SCHED_REQUIREMENTS = "QOS = GOLD"

SCHED_REQUIREMENTS = "QOS != GOLD & HYPERVISOR = kvm"
```

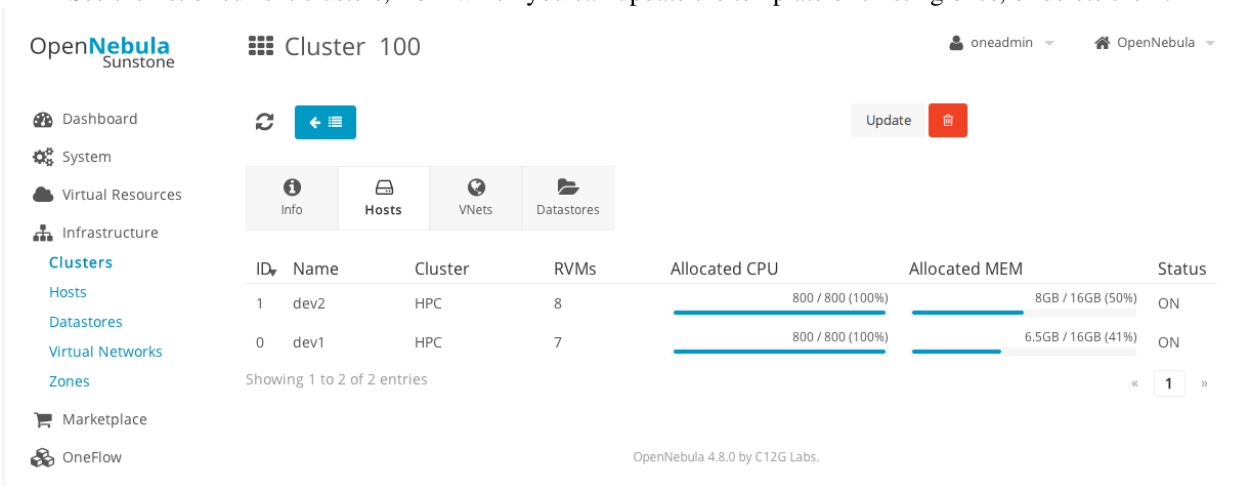
1.3.3 Managing Clusters in Sunstone

The Sunstone UI interface offers an easy way to manage clusters and the resources within them. You will find the cluster sub-menu under the infrastructure menu. From there, you will be able to:

- Create new clusters selecting the resources you want to include in this cluster:



- See the list of current clusters, from which you can update the template of existing ones, or delete them.



1.4 Scheduler

The Scheduler is in charge of the assignment between pending Virtual Machines and known Hosts. OpenNebula's architecture defines this module as a separate process that can be started independently of `oned` (it is however started automatically when you start the `opennebula` service).

1.4.1 Match-making

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

The match-making algorithm works as follows:

- Each disk of a running VM consumes storage from an Image Datastore. The VMs that require more storage than there is currently available are filtered out, and will remain in the `pending` state.
- Those hosts that do not meet the VM requirements (see the *SCHED_REQUIREMENTS attribute*) or do not have enough resources (available CPU and memory) to run the VM are filtered out (see below for more information).
- The same happens for System Datastores: the ones that do not meet the DS requirements (see the *SCHED_DS_REQUIREMENTS attribute*) or do not have enough free storage are filtered out.
- The *SCHED_RANK and SCHED_DS_RANK expressions* are evaluated upon the Host and Datastore list using the information gathered by the monitor drivers. Any variable reported by the monitor driver (or manually set in the Host or Datastore template) can be included in the rank expressions.
- Those resources with a higher rank are used first to allocate VMs.

This scheduler algorithm easily allows the implementation of several placement heuristics (see below) depending on the RANK expressions used.

Configuring the Scheduling Policies

The policy used to place a VM can be configured in two places:

- For each VM, as defined by the `SCHED_RANK` and `SCHED_DS_RANK` attributes in the VM template.
- Globally for all the VMs in the `sched.conf` file (OpenNebula restart required).

Re-Scheduling Virtual Machines

When a VM is in the running state it can be rescheduled. By issuing the `onevm resched` command the VM's recheduling flag is set. In a subsequent scheduling interval, the VM will be consider for rescheduling, if:

- There is a suitable host for the VM.
- The VM is not already running in it.

This feature can be used by other components to trigger rescheduling action when certain conditions are met.

Scheduling VM Actions

Users can schedule one or more VM actions to be executed at a certain date and time. The *onevm schedule* command will add a new SCHED_ACTION attribute to the Virtual Machine editable template. Visit *the VM guide* for more information.

1.4.2 Configuration

The behavior of the scheduler can be tuned to adapt it to your infrastructure with the following configuration parameters defined in /etc/one/sched.conf:

- `MESSAGE_SIZE`: Buffer size in bytes for XML-RPC responses.
- `ONE_XMLRPC`: URL to connect to the OpenNebula daemon (oned) (Default: http://localhost:2633/RPC2)

- SCHED_INTERVAL: Seconds between two scheduling actions (Default: 30)
- MAX_VM: Maximum number of Virtual Machines scheduled in each scheduling action (Default: 5000). Use 0 to schedule all pending VMs each time.
- MAX_DISPATCH: Maximum number of Virtual Machines actually dispatched to a host in each scheduling action (Default: 30)
- MAX_HOST: Maximum number of Virtual Machines dispatched to a given host in each scheduling action (Default: 1)
- LIVE_RESCHEDS: Perform live (1) or cold migrations (0) when rescheduling a VM
- DEFAULT_SCHED: Definition of the default scheduling algorithm.
  - RANK: Arithmetic expression to rank suitable **hosts** based on their attributes.
  - POLICY: A predefined policy, it can be set to:

POLICY	DESCRIPTION
0	**Packing**: Minimize the number of hosts in use by packing the VMs in the hosts to reduce VM fragmentation
1	**Striping**: Maximize resources available for the VMs by spreading the VMs in the hosts
2	**Load-aware**: Maximize resources available for the VMs by using those nodes with less load
3	**Custom**: Use a custom RANK
4	**Fixed**: Hosts will be ranked according to the PRIORITY attribute found in the Host or Cluster template

- DEFAULT_DS_SCHED: Definition of the default storage scheduling algorithm.
  - RANK: Arithmetic expression to rank suitable **datastores** based on their attributes.
  - POLICY: A predefined policy, it can be set to:

POLICY	DESCRIPTION
0	**Packing**:: Tries to optimize storage usage by selecting the DS with less free space
1	**Striping**: Tries to optimize I/O by distributing the VMs across datastores
2	**Custom**: Use a custom RANK
3	**Fixed**: Datastores will be ranked according to the PRIORITY attribute found in the Datastore template

The optimal values of the scheduler parameters depend on the hypervisor, storage subsystem and number of physical hosts. The values can be derived by finding out the max number of VMs that can be started in your set up with out getting hypervisor related errors.

Sample Configuration:

```
MESSAGE_SIZE = 1073741824

ONE_XMLRPC = "http://localhost:2633/RPC2"

SCHED_INTERVAL = 30

MAX_VM       = 5000
MAX_DISPATCH = 30
MAX_HOST     = 1

LIVE_RESCHEDS  = 0

DEFAULT_SCHED = [
   policy = 3,
   rank   = "- (RUNNING_VMS * 50  + FREE_CPU)"
```

```
]

DEFAULT_DS_SCHED = [
   policy = 1
]
```

Pre-defined Placement Policies

The following list describes the predefined policies (`DEFAULT_SCHED`) that can be configured through the `sched.conf` file.

Packing Policy

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

```
RANK = RUNNING_VMS
```

Striping Policy

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

```
RANK = "- RUNNING_VMS"
```

Load-aware Policy

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

```
RANK = FREE_CPU
```

Fixed Policy

- **Target**: Sort the hosts manually
- **Heuristic**: Use the PRIORITY attribute
- **Implementation**: Use those nodes with more PRIORITY first

```
RANK = PRIORITY
```

Pre-defined Storage Policies

The following list describes the predefined storage policies (`DEFAULT_DS_SCHED`) that can be configured through the `sched.conf` file.

Packing Policy

Tries to optimize storage usage by selecting the DS with less free space

- **Target**: Minimize the number of system datastores in use
- **Heuristic**: Pack the VMs in the system datastores to reduce VM fragmentation
- **Implementation**: Use those datastores with less free space first

```
RANK = "- FREE_MB"
```

Striping Policy

- **Target**: Maximize the I/O available to VMs
- **Heuristic**: Spread the VMs in the system datastores
- **Implementation**: Use those datastores with more free space first

```
RANK = "FREE_MB"
```

Fixed Policy

- **Target**: Sort the datastores manually
- **Heuristic**: Use the PRIORITY attribute
- **Implementation**: Use those datastores with more PRIORITY first

```
RANK = PRIORITY
```

1.4.3 Limiting the Resources Exposed by a Host

Prior to assigning a VM to a Host, the available capacity is checked to ensure that the VM fits in the host. The capacity is obtained by the monitor probes. You may alter this behavior by reserving an amount of capacity (MEMORY and CPU). You can reserve this capacity:

- Cluster-wise, by updating the cluster template (e.g. `onecluster update`). All the host of the cluster will reserve the same amount of capacity.
- Host-wise, by updating the host template (e.g. `onehost update`). This value will override those defined at cluster level.

In particular the following capacity attributes can be reserved:

- `RESERVED_CPU` in percentage. It will be subtracted from the `TOTAL CPU`
- `RESERVED_MEM` in KB. It will be subtracted from the `TOTAL MEM`

Note: These values can be negative, in that case you'll be actually increasing the overall capacity so overcommiting host capacity.

1.5 Datastores

OpenNebula features three different datastore types:

- The **Images Datastore**, stores the images repository.
- The **System Datastore** holds disk for running virtual machines, copied or cloned from the Images Datastore.
- The **Files & Kernels Datastore** to store plain files.

1.5.1 Datastore Management

Datastores are managed with the *''onedatastore" command*. In order to be operational an OpenNebula cloud needs at least one Image Datastore and one System Datastore.

Datastore Definition

A datastore definition includes specific attributes to configure its interaction with the storage system; and common attributes that define its generic behavior.

The specific attributes for System and Images Datastores depends on the storage:

- Define Filesystem Datastores.
- Define LVM Datastores.
- Define Ceph Datastores.
- Define Raw Device Mapping Datastores.
- Define iSCSI - Libvirt Datastores.

Also, there are a set of common attributes that can be used in any datastore and compliments the specific attributes for each datastore type described above for each datastore type.

Attribute	Description
`RESTRICTED_DIRS`	Paths that can not be used to register images. A space separated list of paths.
`SAFE_DIRS`	If you need to un-block a directory under one of the RESTRICTED_DIRS. A space separated list of paths.
`NO_DECOMPRESS`	Do not try to untar or decompress the file to be registered. Useful for specialized Transfer Managers
`LIMIT_TRANSFER_BW`	Specify the maximum transfer rate in bytes/second when downloading images from a http/https URL. Suffixes K, M or G can be used.
`DATASTORE_CAPACITY_CHECK`	If `yes`, the available capacity of the datastore is checked before creating a new image
`LIMIT_MB`	The maximum capacity allowed for the datastore in `MB`.
`BRIDGE_LIST`	Space separated list of hosts that have access to the storage to add new images to the datastore.
`STAGING_DIR`	Path in the storage bridge host to copy an Image before moving it to its final destination. Defaults to `/var/tmp`.

The Files & Kernels Datastore is an special datastore type to store plain files to be used as kernels, ram-disks or context files. See here to learn how to define them.

1.5.2 Multiple System Datastore Setup

In order to distribute efficiently the I/O of the Virtual Machines across different disks, LUNs or several storage backends, OpenNebula is able to define multiple System Datastores per cluster. Scheduling algorithms take into account disk requirements of a particular VM, so OpenNebula is able to pick the best execution host based on capacity and storage metrics.

Configuring Multiple Datastores

When more than one System Datastore is added to a cluster, all of them can be taken into account by the scheduler to place Virtual Machines into. System wide scheduling policies are defined in `/etc/one/sched.conf`. The storage scheduling policies are:

- **Packing**. Tries to optimize storage usage by selecting the Datastore with less free space.
- **Striping**. Tries to optimize I/O by distributing the Virtual Machines across Datastores.
- **Custom**. Based on any of the attributes present in the Datastore template.

To activate for instance the Stripping storage policy, `/etc/one/sched.conf` must contain:

```
DEFAULT_DS_SCHED = [
   policy = 1
]
```

These policies may be overriden in the Virtual Machine Template, and so apply specific storage policies to specific Virtual Machines:

Attribute	Description	Example

SCHED_DS_REQUIREMENTS	Boolean expression to select System Datastores (evaluates to true) to run a VM.	`SCHED_DS_REQUIREMENTS="ID=100"`
		`SCHED_DS_REQUIREMENTS="NAME=GoldenDS"`
		`SCHED_DS_REQUIREMENTS=FREE_MB > 250000`

SCHED_DS_RANK	Arithmetic expression to sort the suitable datastores for this VM.	`SCHED_DS_RANK= FREE_MB` `SCHED_DS_RANK=-FREE_MB`

After a VM is deployed in a System Datastore, the admin can migrate it to another System Datastore. To do that, the VM must be first *powered-off*. The command `onevm migrate` accepts both a new Host and Datastore id, that must have the same `TM_MAD` drivers as the source Datastore.

Warning: Any Host belonging to a given cluster **must** be able to access any System or Image Datastore defined in that cluster.

Warning: Admins rights grant permissions to deploy a virtual machine to a certain datastore, using 'onevm deploy' command.

1.5.3 Disable a System Datastore

System Datastores can be disabled to prevent the scheduler from deploying new Virtual Machines in them. Datastores in the `disabled` state and monitored as usual, and the existing Virtual Machines will continue to run in them.

```
$ onedatastore disable system -v
DATASTORE 0: disabled

$ onedatastore show system
DATASTORE 0 INFORMATION
ID             : 0
NAME           : system
...
STATE          : DISABLED
...
```

CHAPTER

TWO

USERS AND GROUPS

2.1 Overview

OpenNebula includes a complete user & group management system. Users in an OpenNebula installation are classified in four types:

- **Administrators**, an admin user belongs to an admin group (oneadmin or otherwise) and can perform manage operations
- **Regular users**, that may access most OpenNebula functionality.
- **Public users**, only basic functionality (and public interfaces) are open to public users.
- **Service users**, a service user account is used by the OpenNebula services (i.e. cloud APIs like EC2 or GUI's like Sunstone) to proxy auth requests.

The resources a user may access in OpenNebula are controlled by a permissions system that resembles the typical UNIX one. By default, only the owner of a resource (e.g. a VM or an image) can use and manage it. Users can easily share the resources by granting use or manage permissions to other users in her group or to any other user in the system.

Upon group creation, an associated admin user can be created. By default this user will be able to create users in the new group, and manage non owned resources for the regular group, through the CLI and/or a special Sunstone view. This group can also be assigned to VDC, what is basically a pool of OpenNebula physical resources (hosts, datastores and virtual networks).

Along with the users & groups the Auth Subsystem is responsible for the authentication and authorization of user's requests.

Any interface to OpenNebula (CLI, Sunstone, Ruby or Java OCA) communicates with the core using XML-RPC calls, that contain the user's session string, which is authenticated by the OpenNebula core comparing the username and password with the registered users.

Each operation generates an authorization request that is checked against the registered ACL rules. The core then can grant permission, or reject the request.

OpenNebula comes with a default set of ACL rules that enables a standard usage. You don't need to manage the ACL rules unless you need the level of permission customization if offers.

By default, the authentication and authorization is handled by the OpenNebula Core as described above. Optionally, you can delegate it to an external module, see the Authentication Guide for more information.

2.1.1 How Should I Read This Chapter

From these guides you should read at least the ones for **Users**, **Groups** and **Permissions** as are the basis for any cloud:

- *Managing Users*
- *Managing Groups*
- *Managing VDCs*
- *Managing Permissions*
- *Accounting Tool*
- *Showback*
- *Managing ACL Rules*
- *Quota Management*

2.1.2 Hypervisor Compatibility

These guides are compatible with both KVM and vCenter hypervisors.

2.2 Managing Users

OpenNebula supports user accounts and groups. This guide shows how to manage users, groups are explained in *their own guide*. To manage user rights, visit the *Managing ACL Rules* guide.

A user in OpenNebula is defined by a username and password. You don't need to create a new Unix account in the front-end for each OpenNebula user, they are completely different concepts. OpenNebula users are authenticated using a session string included in every operation, which is checked by the OpenNebula core.

Each user has a unique ID, and belongs to a group.

After the installation, you will have two administrative accounts, `oneadmin` and `serveradmin`; and two default groups. You can check it using the `oneuser list` and `onegroup list` commands.

There are different user types in the OpenNebula system:

- **Cloud Administrators**, the **oneadmin** account is created **the first time** OpenNebula is started using the ONE_AUTH data. `oneadmin` has enough privileges to perform any operation on any object. Any other user in the oneadmin group has the same privileges as `oneadmin`
- **Infrastructure User** accounts may access most of the functionality offered by OpenNebula to manage resources.
- **Group Administrators** accounts manage a limited set of resources and users.
- **Users** access a simplified Sunstone view with limited actions to create new VMs, and perform basic life cycle operations.
- **Public users** can only access OpenNebula through a public API (EC2), hence they can only use a limited set of functionality and can not access the xml-rpc API directly (nor any application using it like the CLI or Sunstone)
- User **serveradmin** is also created the first time OpenNebula is started. Its password is created randomly, and this account is used by the Sunstone and EC2 servers to interact with OpenNebula.

Note: The complete OpenNebula approach to user accounts, groups and VDC is explained in more detail in the Understanding OpenNebula guide.

2.2.1 Shell Environment

OpenNebula users should have the following environment variables set, you may want to place them in the .bashrc of the user's Unix account for convenience:

ONE_XMLRPC

URL where the OpenNebula daemon is listening. If it is not set, CLI tools will use the default: `http://localhost:2633/RPC2`. See the `PORT` attribute in the Daemon configuration file for more information.

ONE_AUTH

Needs to point to **a file containing a valid authentication key**, it can be:

- A password file with just a single line stating `username:password`.
- A token file with just a single line with `username:token`, where token is a valid token created with the `oneuser login` command or API call.

If `ONE_AUTH` is not defined, `$HOME/.one/one_auth` will be used instead. If no auth file is present, OpenNebula cannot work properly, as this is needed by the core, the CLI, and the cloud components as well.

ONE_POOL_PAGE_SIZE

By default the OpenNebula Cloud API (CLI and Sunstone make use of it) paginates some pool responses. By default this size is 2000 but it can be changed with this variable. A numeric value greater that 2 is the pool size. To disable it you can use a non numeric value.

```
$ export ONE_POOL_PAGE_SIZE=5000        # Sets the page size to 5000
$ export ONE_POOL_PAGE_SIZE=disabled    # Disables pool pagination
```

ONE_CERT_DIR and **ONE_DISABLE_SSL_VERIFY**

If OpenNebula XML-RPC endpoint is behind an SSL proxy you can specify an extra trusted certificates directory using `ONE_CERT_DIR`. Make sure that the certificate is named `<hash>.0`. You can get the hash of a certificate with this command:

```
$ openssl x509 -in <certificate.pem> -hash
```

Alternatively you can set the environment variable `ONE_DISABLE_SSL_VERIFY` to any value to disable certificate validation. You should only use this parameter for testing as it makes the connection insecure.

For instance, a user named `regularuser` may have the following environment:

```
$ tail ~/.bashrc

ONE_XMLRPC=http://localhost:2633/RPC2

export ONE_XMLRPC

$ cat ~/.one/one_auth
regularuser:password
```

Note: Please note that the example above is intended for a user interacting with OpenNebula from the front-end, but you can use it from any other computer. Just set the appropriate hostname and port in the `ONE_XMLRPC` variable.

Note: If you do not want passwords to be stored in plain files, protected with basic filesystem permissions, please refer to the token-based authentication mechanism described below.

An alternative method to specify credentials and OpenNebula endpoint is using command line parameters. Most of the commands can understand the following parameters:

`--user name`	User name used to connect to OpenNebula
`--password password`	Password to authenticate with OpenNebula
`--endpoint endpoint`	URL of OpenNebula XML-RPC Front-end

If `user` is specified but not `password` the user will be prompted for the password. `endpoint` has the same meaning and get the same value as `ONE_XMLRPC`. For example:

```
$ onevm list --user my_user --endpoint http://one.frontend.com:2633/RPC2
Password:
[...]
```

Warning: You should better not use `--password` parameter in a shared machine. Process parameters can be seen by any user with the command `ps` so it is highly insecure.

ONE_SUNSTONE

URL of the Sunstone portal, used for downloading MarketPlaceApps streamed through Sunstone. If this is not specified, it will be inferred from `ONE_XMLRPC` (by changing the port to 9869), and if that env variable is undefined as well, it will default to `http://localhost:9869`.

ONEFLOW_URL, **ONEFLOW_USER** and **ONEFLOW_PASSWORD**

These variables are used by the OneFlow command line tools. If not set, the default OneFlow URL will be `http://localhost:2474`. The user and password will be taken from the `ONE_AUTH` file if the environment variables are not found.

Shell Environment for Self-Contained Installations

If OpenNebula was installed from sources in **self-contained mode** (this is not the default, and not recommended), these two variables must be also set. These are not needed if you installed from packages, or performed a system-wide installation from sources.

ONE_LOCATION

It must point to the installation <destination_folder>.

PATH

The OpenNebula bin files must be added to the path

```
$ export PATH=$ONE_LOCATION/bin:$PATH
```

2.2.2 Adding and Deleting Users

User accounts within the OpenNebula system are managed by `oneadmin` with the `oneuser create` and `oneuser delete` commands. This section will show you how to create the different account types supported in OpenNebula

Administrators

Administrators can be easily added to the system like this:

```
$ oneuser create otheradmin password
ID: 2

$ oneuser chgrp otheradmin oneadmin

$ oneuser list
  ID GROUP    NAME            AUTH                                        PASSWORD
   0 oneadmin oneadmin        core      5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
   1 oneadmin serveradmin     server_c  1224ff12545a2e5dfeda4eddacdc682d719c26d5
   2 oneadmin otheradmin      core      5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8

$ oneuser show otheradmin
USER 2 INFORMATION
ID              : 2
NAME            : otheradmin
GROUP           : 0
PASSWORD        : 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
AUTH_DRIVER     : core
ENABLED         : Yes

USER TEMPLATE
```

Regular Users

Simply create the users with the create command:

```
$ oneuser create regularuser password
ID: 3
```

The enabled flag can be ignored as it doesn't provide any functionality. It may be used in future releases to temporarily disable users instead of deleting them.

Public Users

Public users needs to define a special authentication method that internally relies in the core auth method. First create the public user as it was a regular one:

```
$ oneuser create publicuser password
ID: 4
```

and then change its auth method (see below for more info) to the public authentication method.

```
$ oneuser chauth publicuser public
```

Server Users

Server user accounts are used mainly as proxy authentication accounts for OpenNebula services. Any account that uses the `server_cipher` or `server_x509` auth methods are a server user. You will never use this account directly. To create a user account just create a regular account

```
$ oneuser create serveruser password
ID: 5
```

and then change its auth method to `server_cipher` (for other auth methods please refer to the Authentication guide):

```
$ oneuser chauth serveruser server_cipher
```

2.2.3 Managing Users

User Authentication

In order to authenticate with OpenNebula you need a valid password or authentication token. Its meaning depends on the authentication driver, `AUTH_DRIVER`, set for the user. Note that you will be using this password or token to authenticate within the Sunstone portal or at the CLI/API level.

The default driver, `core`, is a simple user-password match mechanism. To configure a user account simply add to `$HOME/.one/one_auth` a single line with the format `<username>:<password>`. For example, for user `oneadmin` and password `opennebula` the file would be:

```
$ cat $HOME/.one/one_auth
oneadmin:opennebula
```

Once configured you will be able to access the OpenNebula API and use the CLI tools:

```
$ oneuser show
USER 0 INFORMATION
ID              : 0
NAME            : oneadmin
GROUP           : oneadmin
PASSWORD        : c24783ba96a35464632a624d9f829136edc0175e
```

Note: OpenNebula does not store the plain password but a hashed version in the database, as show by the oneuser example above.

Note that `$HOME/.one/one_auth` is just protected with the standard filesystem permissions. To improve the system security you can use authentication tokens. In this way there is no need to store plain passwords, OpenNebula can generate or use an authentication token with a given expiration time. By default, the tokens are also stored in `$HOME/.one/one_auth`.

The following example shows the token generation and usage for the previous `oneadmin` user:

```
$ ls $ONE_AUTH
ls: cannot access /home/oneadmin/.one/one_auth: No such file or directory

$ oneuser login oneadmin
Password:

$cat $HOME/.one/one_auth
oneadmin:800481374d8888805dd51dabd36ca50d77e2132e

$ oneuser show
USER 0 INFORMATION
ID              : 0
NAME            : oneadmin
GROUP           : oneadmin
PASSWORD        : c24783ba96a35464632a624d9f829136edc0175e
AUTH_DRIVER     : core
```

```
LOGIN_TOKEN     : 800481374d8888805dd51dabd36ca50d77e2132e
TOKEN VALIDITY  : not after 2014-09-15 14:27:19 +0200
```

By default tokens are generated with a valid period of ten hours. This can be adjusted when issuing the token with the `oneuser login` command. You can also use this token to authenticate through Sunstone.

Finally, you can configure multiple authentication drivers, read the External Auth guide for more information about, enabling LDAP/AD, SSH or x509 authentication.

Note: The purpose of the `$HOME/.one/one_auth` file and the token generation/usage are valid for any authentication mechanism.

User Templates

The `USER TEMPLATE` section can hold any arbitrary data. You can use the `oneuser update` command to open an editor and add, for instance, the following `DEPARTMENT` and `EMAIL` attributes:

```
$ oneuser show 2
USER 2 INFORMATION
ID              : 2
NAME            : regularuser
GROUP           : 1
PASSWORD        : 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
AUTH_DRIVER     : core
ENABLED         : Yes

USER TEMPLATE
DEPARTMENT=IT
EMAIL=user@company.com
```

These attributes can be later used in the *Virtual Machine Contextualization*. For example, using contextualization the user's public ssh key can be automatically installed in the VM:

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

2.2.4 Manage your Own User

Regular users can see their account information, and change their password.

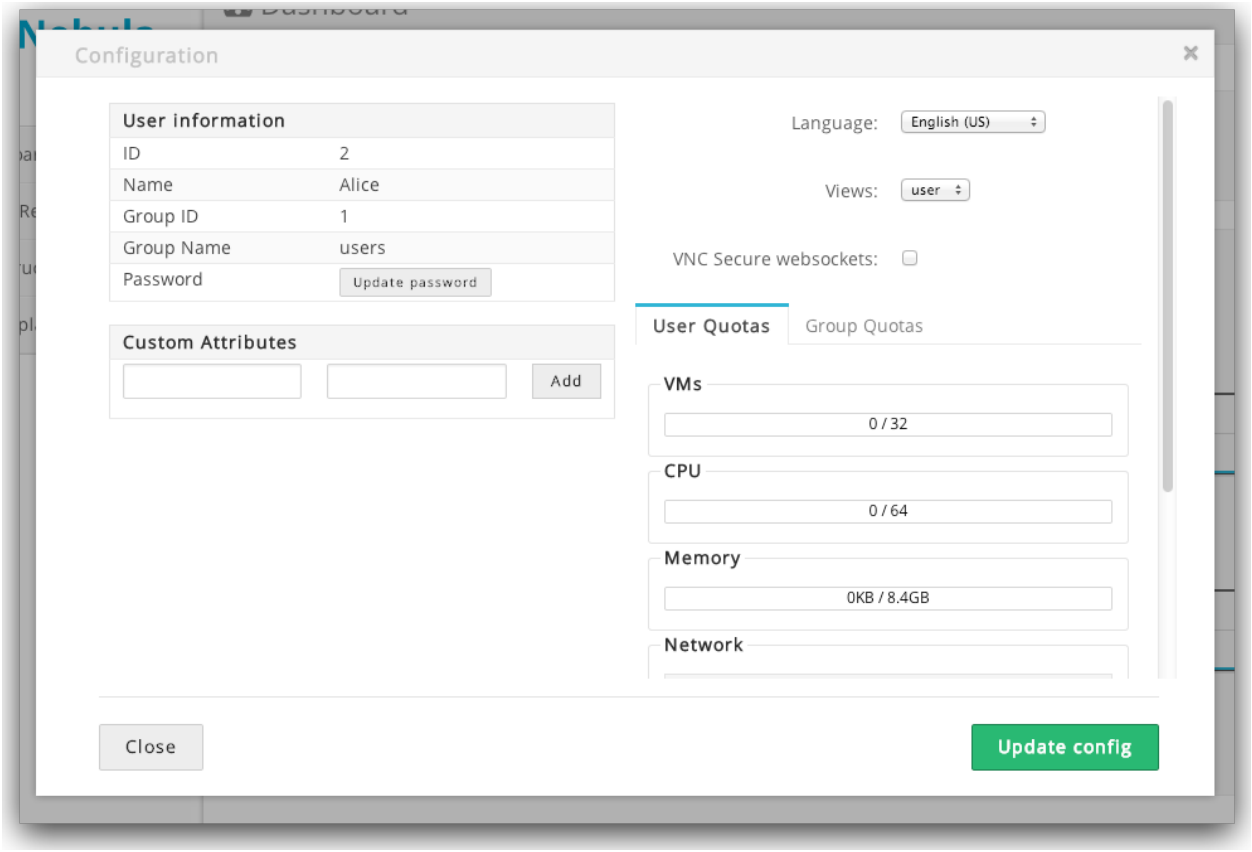
For instance, as `regularuser` you could do the following:

```
$ oneuser list
[UserPoolInfo] User [2] not authorized to perform action on user.

$ oneuser show
USER 2 INFORMATION
ID              : 2
NAME            : regularuser
GROUP           : 1
PASSWORD        : 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
AUTH_DRIVER     : core
ENABLED         : Yes
```

```
USER TEMPLATE
DEPARTMENT=IT
EMAIL=user@company.com

$ oneuser passwd 1 abcdpass
```

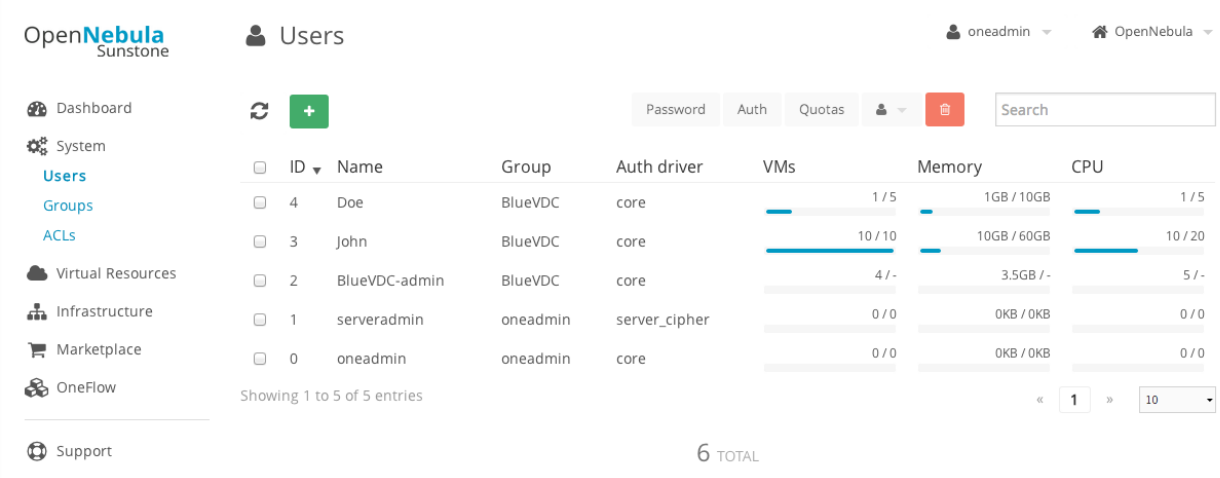
As you can see, any user can find out his ID using the `oneuser show` command without any arguments.

Regular users can retrieve their quota and user information in the settings section in the top right corner of the main screen:



2.2.5 Managing Users in Sunstone

All the described functionality is available graphically using Sunstone:



2.3 Managing Groups

A group in OpenNebula makes it possible to isolate users and resources. A user can see and use the *shared resources* from other users.

The group is an authorization boundary for the users, but you can also partition your cloud infrastructure and define what resources are available to each group using *Virtual Data Centers (VDC)*. You can read more about OpenNebula's approach to VDCs and the cloud from the perspective of different user roles in the Understanding OpenNebula guide.

2.3.1 Adding and Deleting Groups

There are two special groups created by default. The `oneadmin` group allows any user in it to perform any operation, allowing different users to act with the same privileges as the `oneadmin` user. The `users` group is the default group where new users are created.

Your can use the `onegroup` command line tool to manage groups in OpenNebula. There are two groups created by default, `oneadmin` and `users`.

To create new groups:

```
$ onegroup list
  ID NAME
   0 oneadmin
   1 users

$ onegroup create "new group"
ID: 100
```

The new group has ID 100 to differentiate the special groups from the user-defined ones.

Note: When a new group is created, an ACL rule is also created to provide the default behavior, allowing users to create basic resources. You can learn more about ACL rules in *this guide*; but you don't need any further configuration to start using the new group.

2.3.2 Adding Users to Groups

Use the `oneuser chgrp` command to assign users to groups.

```
$ oneuser chgrp -v regularuser "new group"
USER 1: Group changed

$ onegroup show 100
GROUP 100 INFORMATION
ID              : 100
NAME            : new group

USERS
ID               NAME
1                regularuser
```

To delete a user from a group, just move it again to the default `users` group.

2.3.3 Admin Users and Allowed Resources

Upon group creation, a special admin user account can be defined. This admin user will have administrative privileges only for the new group, not for all the resources in the OpenNebula cloud as the 'oneadmin' group users have.

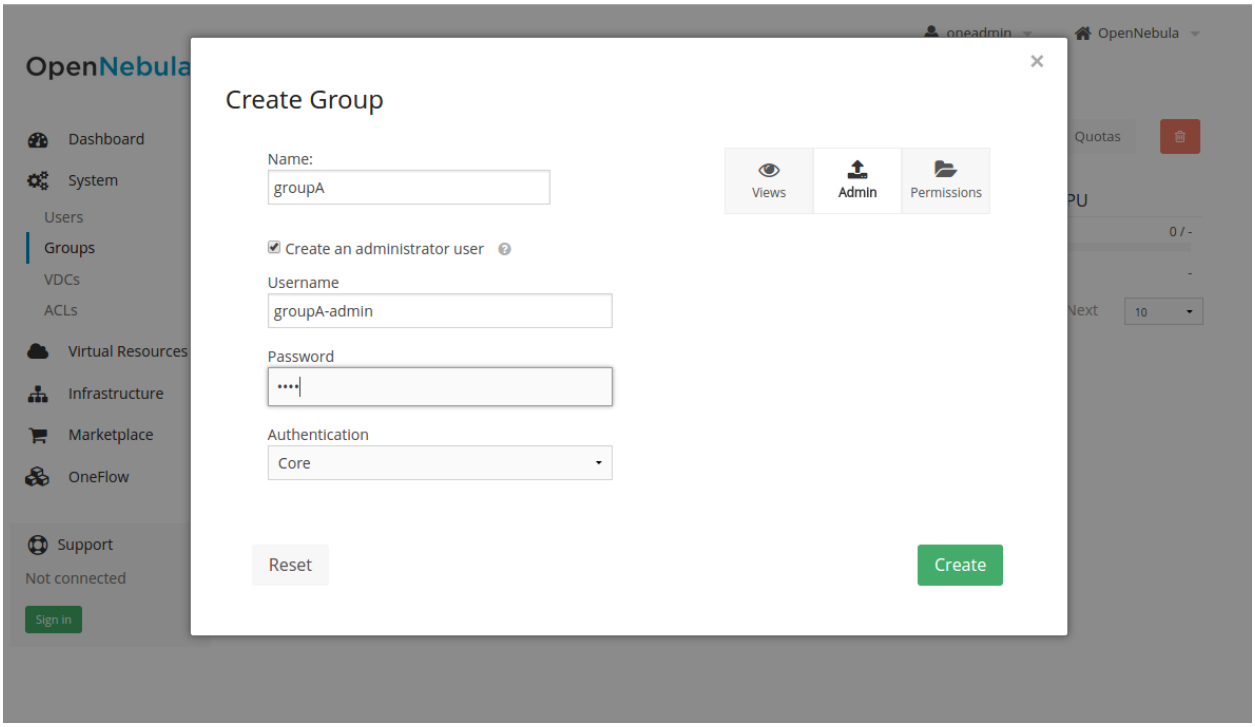
Another aspect that can be controlled on creation time is the type of resources that group users will be allowed to create.

This can be managed visually in Sunstone, and can also be managed through the CLI. In the latter, details of the group are passed to the `onegroup create` command as arguments. This table lists the description of said arguments.

Argument	M / O	Value	Description
-n, –name name	Mandatory	Any string	Name for the new group
-u, –admin_user	Optional	Any string	Creates an admin user for the group with the given name
-p, –admin_password	Optional	Any string	Password for the admin user of the group
-d, –admin_driver	Optional	Any string	Auth driver for the admin user of the group
-r, –resources	Optional	"+" separated list	Which resources can be created by group users (VM+IMAGE+TEMPLATE by default)

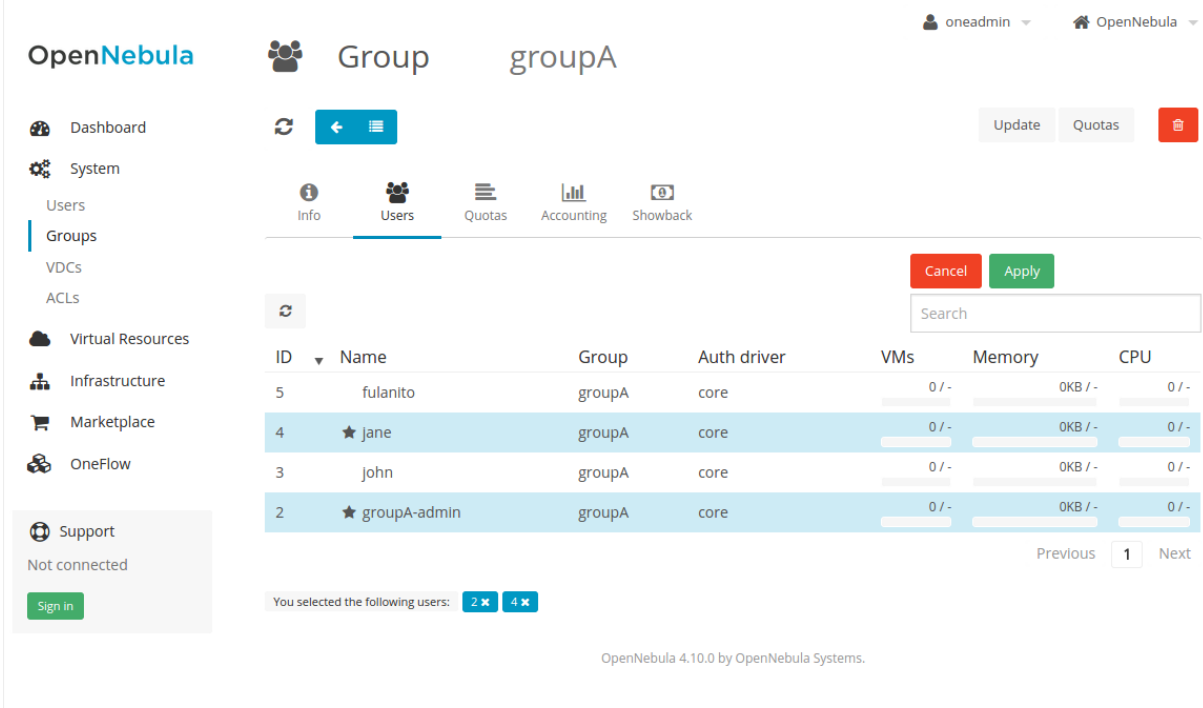
An example:

```
$ onegroup create --name groupA \
--admin_user admin_userA --admin_password somestr \
--resources TEMPLATE+VM
```



2.3.4 Add Admin Users to an Existing Group

Any user can be configured to be Admin of a group with the commands `onegroup addadmin` and `deladmin`.



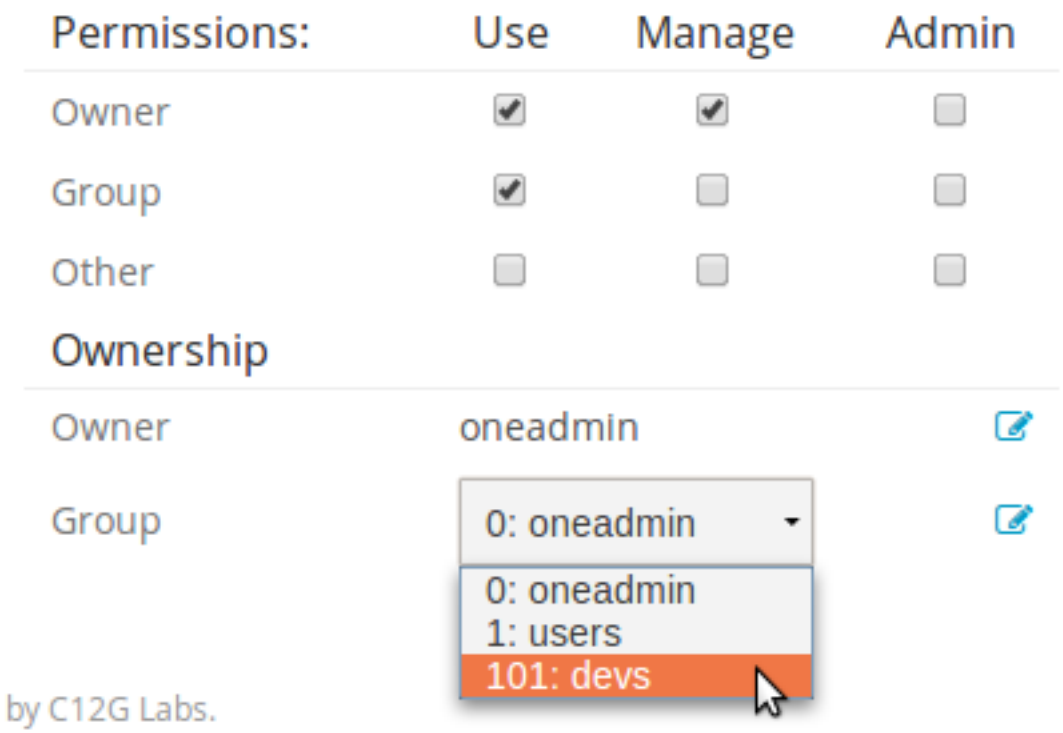
2.3.5 Managing Groups and Virtual Resources

You can make the following virtual resources available to group users:

- *Virtual Machine Templates*
- Service Templates
- *Images*
- *Files & Kernels*

To make a virtual resource owned by oneadmin available to users of the new group, you have two options:

- Change the resource's group, and give it `GROUP USE` permissions. This will make the resource only available to users in that group. The recommended practice to assign golden resources to groups is to first clone the resource and then assign it to the desired group for their users' consumption.
- Leave the resource in the oneadmin group, and give it `OTHER USE` permissions. This will make the resource available to every user in OpenNebula.



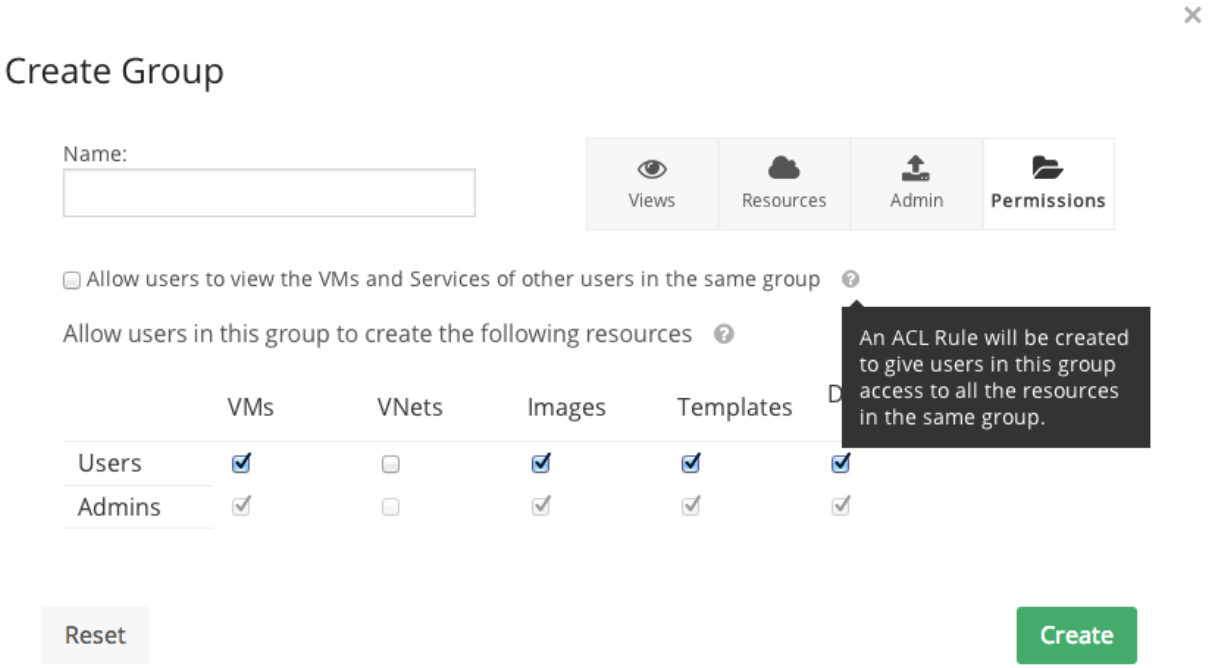
The Virtual Machine and Service Templates are visible to the group users when they want to create a new VM or Service. The Images (including File Images) used by those Templates are not visible to the users, but must be also made available, otherwise the VM creation will fail with an error message similar to this one:

```
[TemplateInstantiate] User [6] : Not authorized to perform USE IMAGE [0].
```

You can read more about OpenNebula permissions in the *Managing Permissions* and *Managing ACL Rules* guides.

2.3.6 Resource Sharing

When a new group is created the cloud administrator can define if the users of this view will be allowed to view the VMs and Services of other users in the same group. If this option is checked a new ACL rule will be created to give users in this group access to the VMS and Services in the same group. Users will not able to manage these resources but they will be included in the list views of each resource.



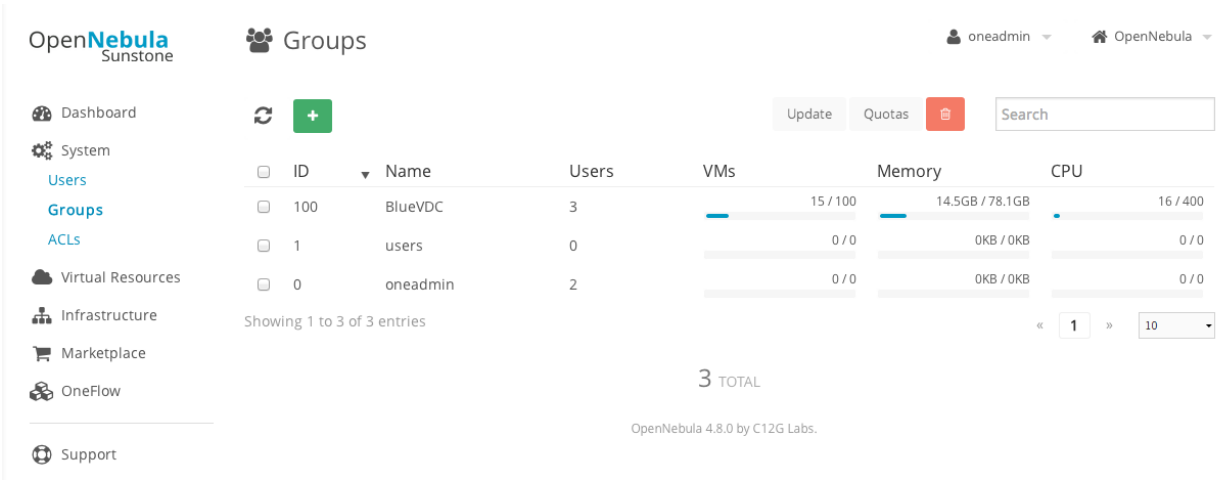
2.3.7 Primary and Secondary Groups

With the commands `oneuser addgroup` and `delgroup` the administrator can add or delete secondary groups. Users assigned to more than one group will see the resources from all their groups. e.g. a user in the groups testing and production will see VMs from both groups.

The group set with `chgrp` is the primary group, and resources (Images, VMs, etc) created by a user will belong to this primary group. Users can change their primary group to any of their secondary group without the intervention of an administrator, using `chgrp` again.

2.3.8 Managing Groups in Sunstone

All the described functionality is available graphically using Sunstone:



2.4 Managing VDCs

A VDC (Virtual Data Center) defines an assignment of one or several *groups* to a pool of physical resources. This pool of Physical Resources consists of resources from one or several Clusters that could belong to different Zones or public external clouds for hybrid cloud computing. You can read more about OpenNebula's approach to VDCs and the cloud from the perspective of different user roles in the Understanding OpenNebula guide.

2.4.1 The Default VDC

There is a special `default` VDC created during the installation that allows the use of `ALL` the physical resources.

The `users` group belongs to this VDC, and any new group is automatically added to the `default` VDC. You can modify the VDC physical resources, even remove all of them, but it can't be deleted.

Note: Before adding a new group to a specific VDC, you may want to remove it from the `default` one, since it allows the use of `ALL` the physical resources.

2.4.2 Adding and Deleting VDCs

Your can use the `onevdc` command line tool to manage VDCs in OpenNebula.

To create new VDCs:

```
$ onevdc list
  ID NAME
   0 default

$ onevdc create "high-performance"
ID: 100
```

The new VDC has ID 100 to differentiate the default VDC from the user-defined ones.

2.4.3 Adding Groups to VDCs

By default a group doesn't belong to any VDC, so users won't be entitled to use any resource until explicitly added to one.

To add groups to a VDC:

```
$ onevdc addgroup <vdc_id> <group_id>
```

2.4.4 Adding Physical Resources to VDCs

Physical resources (hosts, virtual networks, and datastores) can be added to the VDC. Internally, the VDC will create ACL Rules that will allow the VDC groups to use this pool of resources.

Typically, you will want to add Clusters to the VDC. For instance, Cluster 7 from Zone 0:

```
$ onevdc addcluster <vdc_id > 0 7
```

But you can also add individual hosts, virtual networks, and datastores:

```
$ onevdc addhost <vdc_id> 0 3
$ onevdc addvnet <vdc_id> 0 9
$ onevdc adddatastore <vdc_id> 0 102
```

The special id `ALL` can be used to add all clusters/hosts/vnets/datastores from the given zone:

```
$ onevdc addcluser <group_id> 0 ALL
```

To remove physical resources from the VDC, use the symmetric operations `delcluster`, `delhost`, `delvnet`, `deldatastore`.

When you assign physical resources to a VDC, users in that VDC's groups will be able to use the Datastores and Virtual Networks. The scheduler will also deploy VMs from that group to any of the VDC Hosts.

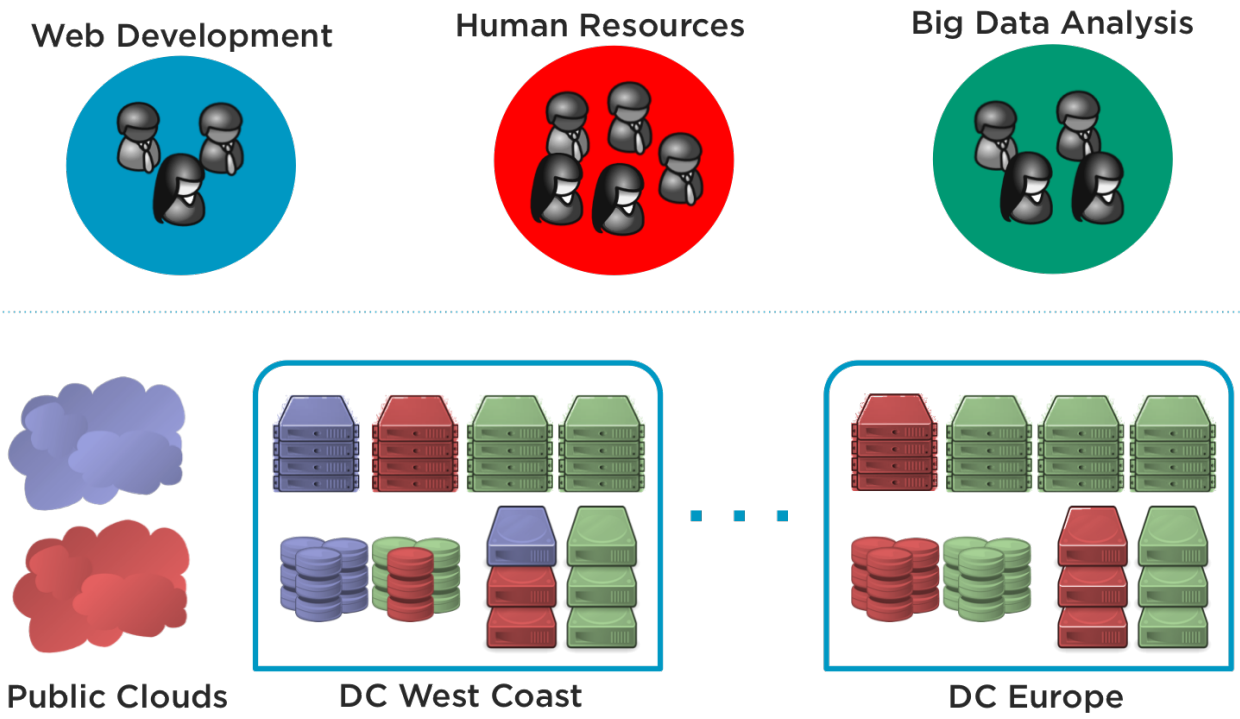
If you are familiar with *ACL rules*, you can take a look at the rules that are created with `oneacl list`. These rules are automatically added, and should not be manually edited. They will be removed by the `onevdc del*` commands.

2.4.5 Examples

The VDC management offers plenty of flexibility to suit many different scenarios. This section lists a few of them to help to visualize the possibilities of your OpenNebula infrastructure.

For example, let's say Web Development, Human Resources, and Big Data Analysis as business units represented by Groups in a private OpenNebula cloud, with resources allocated from your zones and public clouds in order to create three different VDCs.

- **VDC BLUE**: VDC that allocates resources from DC_West_Coast + Cloudbursting to Web Development
- **VDC RED**: VDC that allocates resources from DC_West_Coast + DC_Europe + Cloudbursting to Human Resources
- **VDC GREEN**: VDC that allocates resources from DC_West_Coast + DC_Europe to Big Data Analysis



Flexible Groups

If you plan to have a small infrastructure, the VDC management may seem like an unnecessary extra step to assign physical resources to each Group. But having an independent VDC object allows it to have more than one Group, and at the same time a Group can be part of more than one VDC.

In practical terms, this means that once you organize your Users into Groups, and then your physical resources into VDCs, you can easily assign more or less resources to those Groups.

Using the previous scenario as an example, the Cloud Admin can add the Group Web Development to the VDCs RED and GREEN if their workload increases, and then remove it again a few days later.

Create Super-Clusters

A VDC can have more than one physical resource of each type (Cluster, Hosts, VNets, Datastores), and a physical resource can be in more than one VDC. In contrast a Host can be part of only one Cluster. This means that you can decide to create a VDC that encompasses resources that may not be part of the same physical Cluster.

For example, a VDC called 'high-performance' may contain Hosts from two incompatible Clusters, let's say 'kvm-ceph' and 'kvm-qcow2'. These Hosts may be part of the same VDC, but from the deployment point of view, the important factor is their Cluster. The scheduler will decide the deployment target based on each Host's Cluster, and this guarantees that the VMs are always deployed in a compatible Host.

Partition a Cluster

Since a VDC can contain individual Hosts, VNets and Datastores, you can use VDCs to partition a Cluster into "sub-clusters" that contain a few Hosts.

Following the previous example, you may have a big "kvm-ceph" Cluster. A VDC with one or two Hosts can be created to isolate a small portion of the Cluster. In this case, remember to add the necessary Datastores and VNets to the VDC, otherwise the Users won't be able to instantiate the VM Templates.

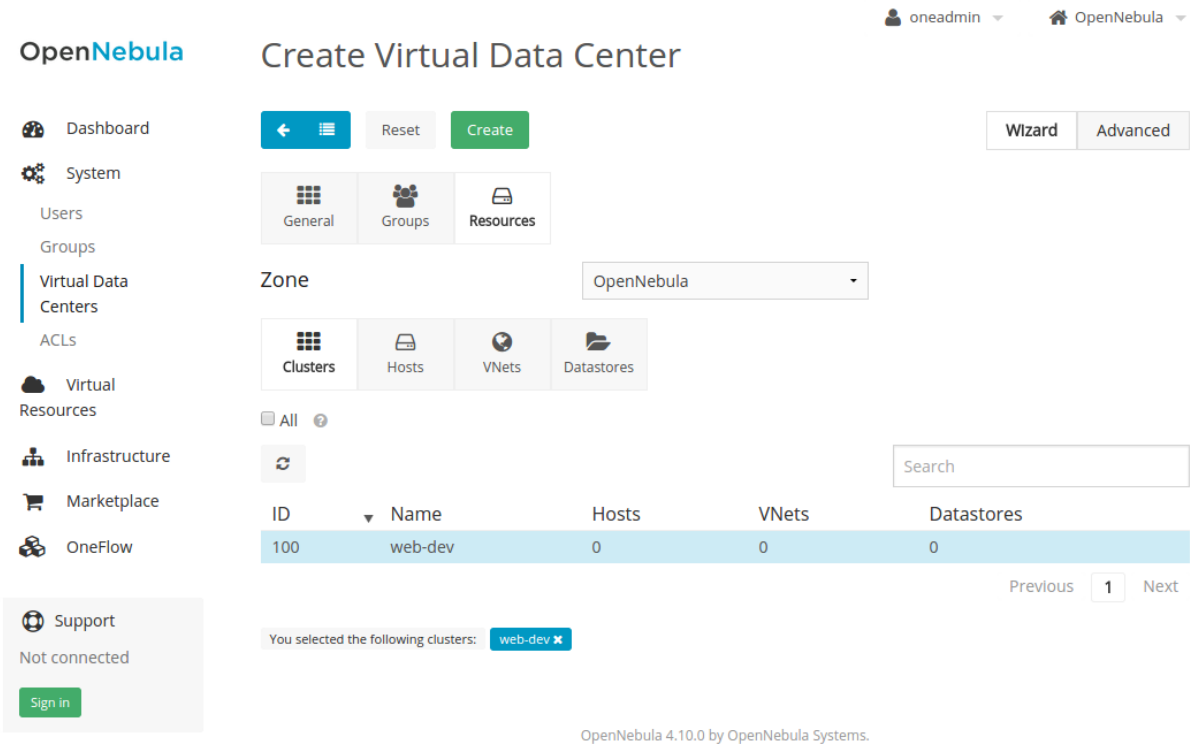
Share Physical Resources

You may have two Groups with a similar workload, but want to keep their users and virtual resources isolated. In this case, both can be added to the same VDC. In a similar way, a physical resource (such as a host) can be part of two different VDCs.

The Groups will share the physical resources, but without being aware of it. If the physical resources are not exclusively assigned to a Group, you may want to set *usage quotas*.

2.4.6 Managing VDCs in Sunstone

All the described functionality is available graphically using Sunstone:



2.5 Managing Permissions

Most OpenNebula resources have associated permissions for the **owner**, the users in her **group**, and **others**. For each one of these groups, there are three rights that can be set: **USE**, **MANAGE** and **ADMIN**. These permissions are very similar to those of UNIX file system.

The resources with associated permissions are *Templates*, *VMs*, *Images* and *Virtual Networks*. The exceptions are *Users*, *Groups* and *Hosts*.

2.5.1 Managing Permission through the CLI

This is how the permissions look in the terminal:

```
$ onetemplate show 0
TEMPLATE 0 INFORMATION
ID              : 0
NAME            : vm-example
USER            : oneuser1
GROUP           : users
REGISTER TIME   : 01/13 05:40:28

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

[...]
```

The previous output shows that for the Template 0, the owner user `oneuser1` has **USE** and **MANAGE** rights. Users in the group `users` have **USE** rights, and users that are not the owner or in the `users` group don't have any rights over this Template.

You can check what operations are allowed with each of the **USE**, **MANAGE** and **ADMIN** rights in the xml-rpc reference documentation. In general these rights are associated with the following operations:

- **USE**: Operations that do not modify the resource like listing it or using it (e.g. using an image or a virtual network). Typically you will grant `USE` rights to share your resources with other users of your group or with the rest of the users.
- **MANAGE**: Operations that modify the resource like stopping a virtual machine, changing the persistent attribute of an image or removing a lease from a network. Typically you will grant `MANAGE` rights to users that will manage your own resources.
- **ADMIN**: Special operations that are typically limited to administrators, like updating the data of a host or deleting an user group. Typically you will grant `ADMIN` permissions to those users with an administrator role.

Warning: By default every user can update any permission group (owner, group or other) with the exception of the admin bit. There are some scenarios where it would be advisable to limit the other set (e.g. OpenNebula Zones so users can not break the group limits). In these situations the `ENABLE_OTHER_PERMISSIONS` attribute can be set to `NO` in `/etc/one/oned.conf` file

Changing Permissions with chmod

The previous permissions can be updated with the chmod command. This command takes an octet as a parameter, following the octal notation of the Unix chmod command. The octet must be a three-digit base-8 number. Each digit, with a value between 0 and 7, represents the rights for the **owner**, **group** and **other**, respectively. The rights are represented by these values:

- The **USE** bit adds 4 to its total (in binary 100)
- The **MANAGE** bit adds 2 to its total (in binary 010)

- The **ADMIN** bit adds 1 to its total (in binary 001)

Let's see some examples:

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

$ onetemplate chmod 0 664 -v
VMTEMPLATE 0: Permissions changed

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

$ onetemplate chmod 0 644 -v
VMTEMPLATE 0: Permissions changed

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

$ onetemplate chmod 0 607 -v
VMTEMPLATE 0: Permissions changed

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

Setting Default Permissions with umask

The default permissions given to newly created resources can be set:

- Globally, with the **DEFAULT_UMASK** attribute in oned.conf
- Individually for each User, using the *oneuser umask command*.

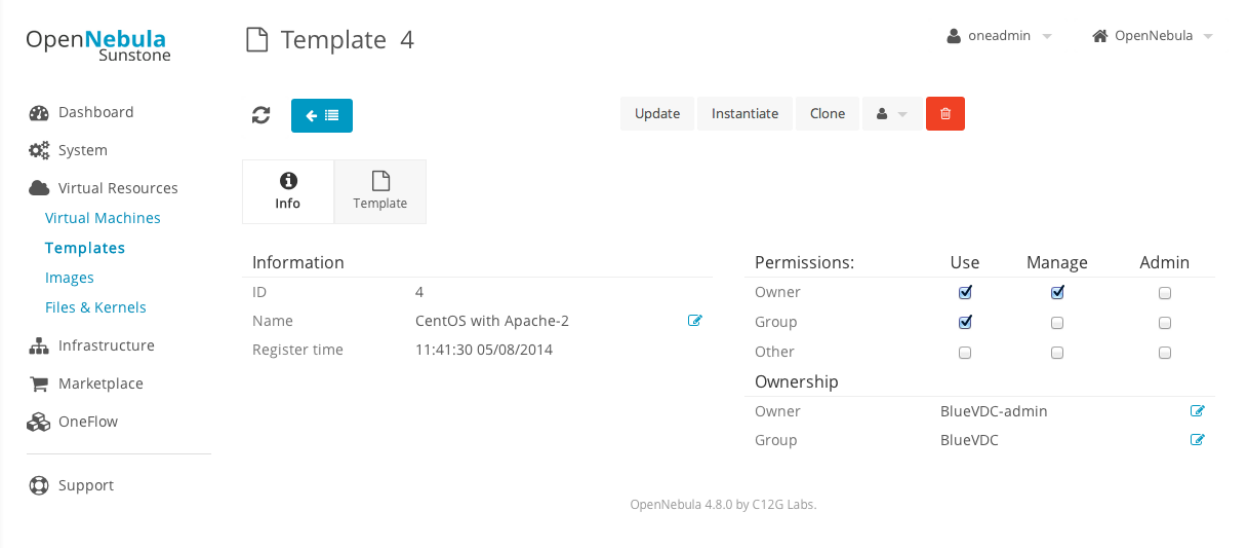
These mask attributes work in a similar way to the Unix umask command. The expected value is a three-digit base-8 number. Each digit is a mask that **disables** permissions for the **owner**, **group** and **other**, respectively.

This table shows some examples:

umask	permissions (octal)	permissions
177	600	`um- --- ---`
137	640	`um- u-- ---`
113	664	`um- um- u--`

2.5.2 Managing Permissions in Sunstone

Sunstone offers a convenient way to manage resources permissions. This can be done by selecting resources from a view (for example the templates view) and clicking on the `update properties` button. The update dialog lets the user conveniently set the resource's permissions.



2.6 Managing ACL Rules

The ACL authorization system enables fine-tuning of the allowed operations for any user, or group of users. Each operation generates an authorization request that is checked against the registered set of ACL rules. The core then can grant permission, or reject the request.

This allows administrators to tailor the user roles according to their infrastructure needs. For instance, using ACL rules you could create a group of users that can see and use existing virtual resources, but not create any new ones. Or grant permissions to a specific user to manage Virtual Networks for some of the existing groups, but not to perform any other operation in your cloud. Some examples are provided at the end of this guide.

Please note: the ACL rules is an advanced mechanism. For most use cases, you should be able to rely on the built-in *resource permissions* and the ACL Rules created automatically when a *group is created*, and when *physical resources are added to a VDC*.

2.6.1 Understanding ACL Rules

Lets start with an example:

```
#5 IMAGE+TEMPLATE/@103 USE+MANAGE #0
```

This rule grants the user with ID 5 the right to perform USE and MANAGE operations over all Images and Templates in the group with id 103.

The rule is split in four components, separated by a space:

- **User** component is composed only by an **ID definition**.
- **Resources** is composed by a list of **'+'** separated resource types, **'/'** and an **ID definition**.
- **Rights** is a list of Operations separated by the **'+'** character.

- **Zone** is an **ID definition** of the zones where the rule applies. This last part is optional, and can be ignored unless OpenNebula is configured in a federation.

The **ID definition** for User in a rule is written as:

- `#<id>` : for individual IDs
- `@<id>` : for a group ID
- `*` : for All

The **ID definition** for a Resource has the same syntax as the ones for Users, but adding:

- `%<id>` : for cluster IDs

Some more examples:

This rule allows all users in group 105 to create new virtual resources:

```
@105 VM+NET+IMAGE+TEMPLATE/* CREATE
```

The next one allows all users in the group 106 to use the Virtual Network 47. That means that they can instantiate VM templates that use this network.

```
@106 NET/#47 USE
```

Note: Note the difference between `* NET/#47 USE"` **vs** `* NET/@47 USE`

All Users can use NETWORK with ID 47 **vs** All Users can use NETWORKS belonging to the Group whose ID is 47

The following one allows users in group 106 to deploy VMs in Hosts assigned to the cluster 100

```
@106 HOST/%100 MANAGE
```

2.6.2 Managing ACL Rules via Console

The ACL rules are managed using the *oneacl command*. The 'oneacl list' output looks like this:

```
$ oneacl list
   ID     USER RES_VHNIUTGDCOZSvRMA    RID OPE_UMAC  ZONE
    0       @1     V--I-T---O-S----      *      ---c     *
    1        *     ----------Z-----      *      u---     *
    2        *     --------------MA      *      u---     *
    3       @1     -H--------------      *      -m--    #0
    4       @1     --N----D--------      *      u---    #0
    5     @106     ---I------------    #31      u---    #0
```

The rules shown correspond to the following ones:

```
@1      VM+IMAGE+TEMPLATE+DOCUMENT+SECGROUP/*   CREATE  *
*       ZONE/*                                  USE     *
*       MARKETPLACE+MARKETPLACEAPP/*            USE     *
@1      HOST/*                                  MANAGE  #0
@1      NET+DATASTORE/*                         USE     #0
@106    IMAGE/#31                               USE     #0
```

The first five were created on bootstrap by OpenNebula, and the last one was created using oneacl:

```
$ oneacl create "@106 IMAGE/#31 USE"
ID: 5
```

The **ID** column identifies each rule's ID. This ID is needed to delete rules, using **'oneacl delete <id>'**.

Next column is **USER**, which can be an individual user (#) or group (@) id; or all (*) users.

The **Resources** column lists the existing Resource types initials. Each rule fills the initials of the resource types it applies to.

- V : VM
- H : HOST
- N : NET
- I : IMAGE
- U : USER
- T : TEMPLATE
- G : GROUP
- D : DATASTORE
- C : CLUSTER
- O : DOCUMENT
- Z : ZONE
- S : SECURITY GROUP
- v : VDC
- R : VROUTER
- M : MARKETPLACE
- A : MARKETPLACEAPP

RID stands for Resource ID, it can be an individual object (#), group (@) or cluster (%) id; or all (*) objects.

The next **Operations** column lists the allowed operations initials.

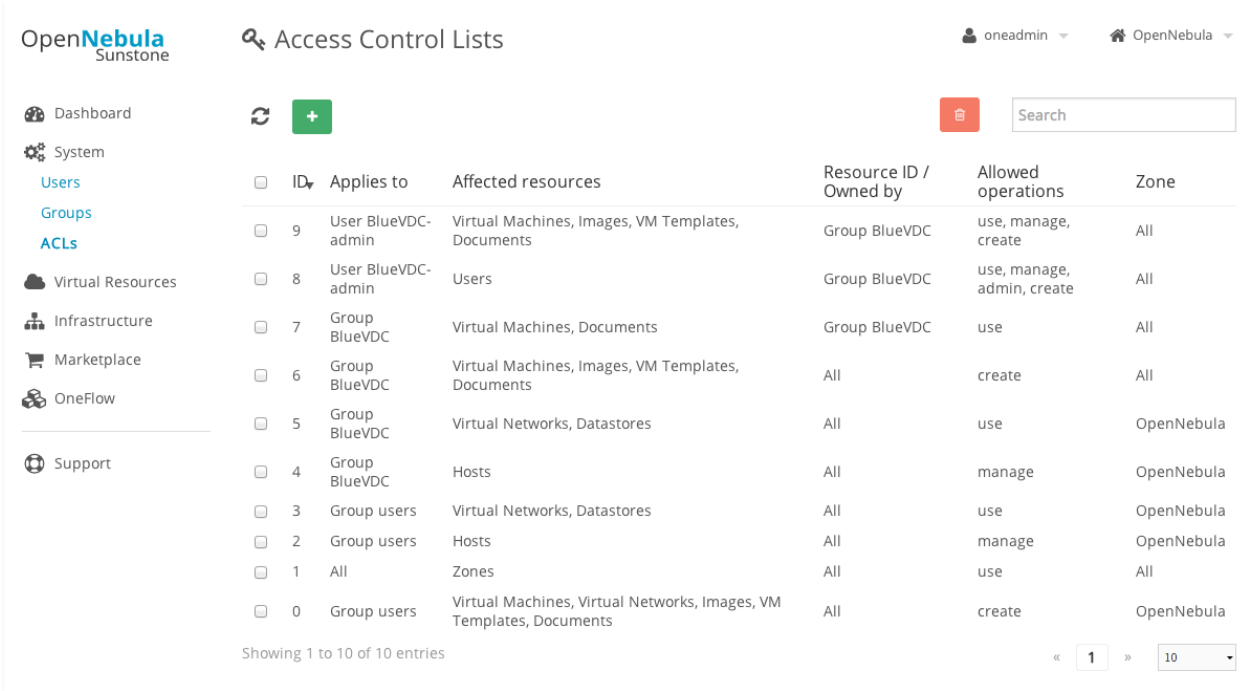
- U : USE
- M : MANAGE
- A : ADMIN
- C : CREATE

And the last column, **Zone**, shows the zone(s) where the rule applies. It can be an individual zone id (#), or all (*) zone.

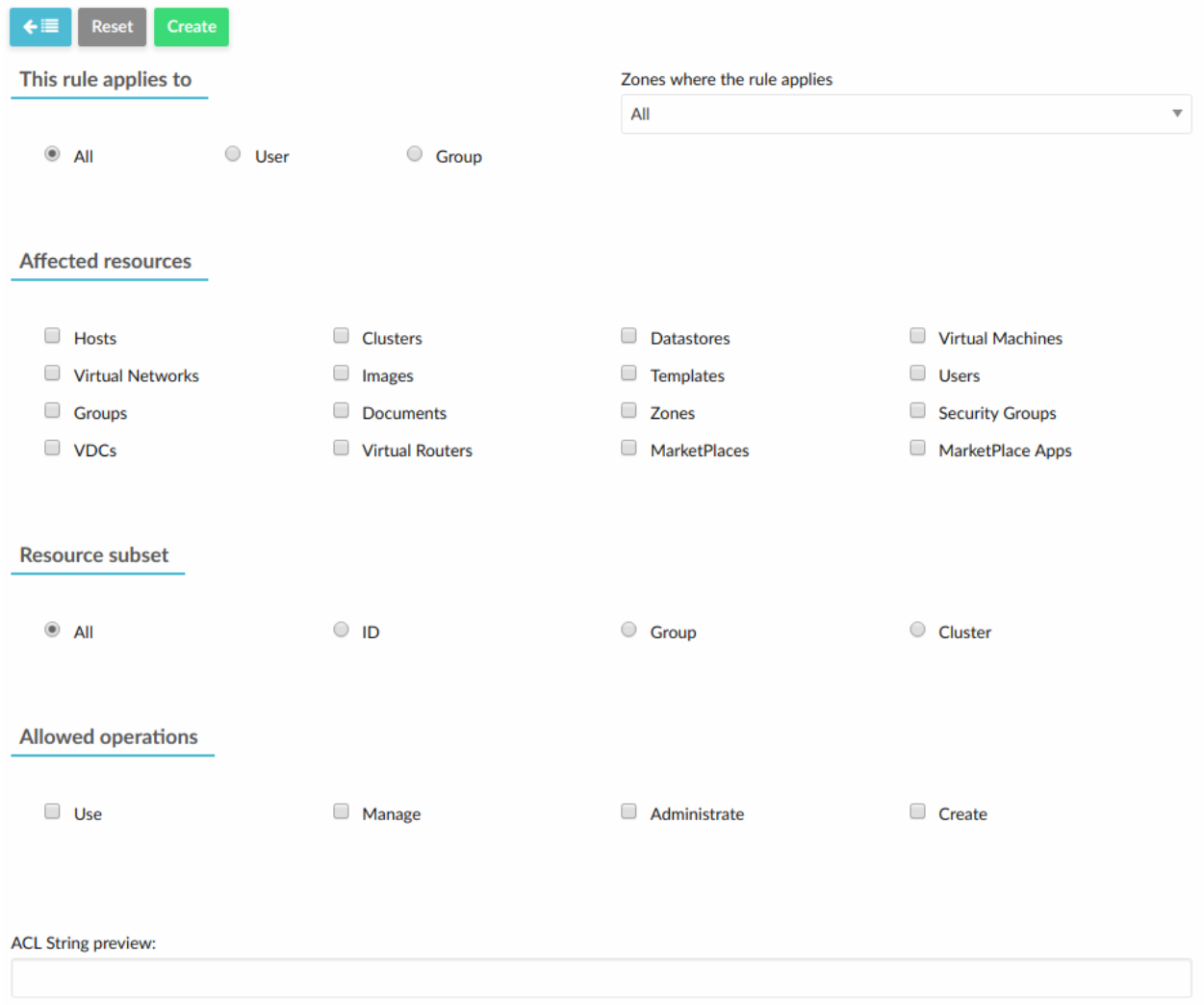
2.6.3 Managing ACLs via Sunstone

Sunstone offers a very intuitive and easy way of managing ACLs.

Select ACLs in the left-side menu to access a view of the current ACLs defined in OpenNebula:



This view is designed to easily understand what the purpose of each ACL is. You can create new ACLs by clicking on the `New` button at the top. A dialog will pop up:



In the creation dialog you can easily define the resources affected by the rule and the permissions that are granted upon them.

2.6.4 How Permission is Granted or Denied

Note: Visit the XML-RPC API reference documentation for a complete list of the permissions needed by each OpenNebula command.

For the internal Authorization in OpenNebula, there is an implicit rule:

- The oneadmin user, or users in the oneadmin group are authorized to perform any operation.

If the resource is one of type `VM`, `NET`, `IMAGE`, `TEMPLATE`, or `DOCUMENT` the object's permissions are checked. For instance, this is an example of the oneimage show output:

```
$ oneimage show 2
IMAGE 2 INFORMATION
ID             : 2
[...]
```

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

The output above shows that the owner of the image has `USE` and `MANAGE` rights.

If none of the above conditions are true, then the set of ACL rules is iterated until one of the rules allows the operation.

An important concept about the ACL set is that each rule adds new permissions, and they can't restrict existing ones: if any rule grants permission, the operation is allowed.

This is important because you have to be aware of the rules that apply to a user and his group. Consider the following example: if a user **#7** is in the group **@108**, with the following existing rule:

```
@108 IMAGE/#45 USE+MANAGE
```

Then the following rule won't have any effect:

```
#7 IMAGE/#45 USE
```

Special Authorization for Virtual Network Reservations

There is a special sub-type of Virtual Network: *reservations*. For these virtual networks the ACL system makes the following exceptions:

- ACL rules that apply to ALL (*) are ignored
- ACL rules that apply to a cluster (%) are ignored

The other ACL rules are enforced: individual (#) and group (@). The Virtual Network object's permissions are also enforced as usual.

2.7 Managing Quotas

This guide will show you how to set the usage quotas for users and groups.

2.7.1 Overview

The quota system tracks user and group usage of system resources, and allows the system administrator to set limits on the usage of these resources. Quota limits can be set for:

- **users**, to individually limit the usage made by a given user.
- **groups**, to limit the overall usage made by all the users in a given group. This can be of special interest for the OpenNebula Zones and Virtual Data Center (VDC) components.

2.7.2 Which Resource can be limited?

The quota system allows you to track and limit usage on:

- **Datastores**, to control the amount of storage capacity allocated to each user/group for each datastore.
- **Compute**, to limit the overall memory, cpu or VM instances.

- **Network**, to limit the number of IPs a user/group can get from a given network. This is specially interesting for networks with public IPs, which usually are a limited resource.
- **Images**, you can limit the how many VM instances from a given user/group are using a given image. You can take advantage of this quota when the image contains consumable resources (e.g. software licenses).

2.7.3 Defining User/Group Quotas

Usage quotas are set in a traditional template syntax (either plain text or XML). The following table explains the attributes needed to set each quota:

Datastore Quotas. Attribute name: DATASTORE

DATASTORE Attribute	Description
ID	ID of the Datastore to set the quota for
SIZE	Maximum size in MB that can be used in the datastore
IMAGE	Maximum number of images that can be created in the datastore

Compute Quotas. Attribute name: VM

VM Attribute	Description
VMS	Maximum number of VMs that can be created
MEMORY	Maximum memory in MB that can be requested by user/group VMs
CPU	Maximum CPU capacity that can be requested by user/group VMs
SYSTEM_DISK_SIZE	Maximum size (in MB) of system disks that can be requested by user/group VMs

Network Quotas. Attribute name: NETWORK

NETWORK Attribute	Description
ID	ID of the Network to set the quota for
LEASES	Maximum IPs that can be leased from the Network

Image Quotas. Attribute name: IMAGE

IMAGE Attribute	Description
ID	ID of the Image to set the quota for
RVMS	Maximum VMs that can used this image at the same time

For each quota, there are two special limits:

- **-1** means that the **default quota** will be used
- **-2** means **unlimited**

Warning: Each quota has an usage counter associated named `<QUOTA_NAME>_USED`. For example `MEMORY_USED` means the total memory used by user/group VMs, and its associated quota is `MEMORY`.

The following template shows a quota example for a user in plain text. It limits the overall usage in Datastore 0 to 20Gb (for an unlimited number of images); the number of VMs that can be created to 4 with a maximum memory to 2G and 5 CPUs; the number of leases from network 1 to 4; and image 1 can only be used by 3 VMs at the same time:

```
DATASTORE=[
  ID="1",
  IMAGES="-2",
  SIZE="20480"
]

VM=[
  CPU="5",
  MEMORY="2048",
  VMS="4",
  SYSTEM_DISK_SIZE="-1"
]

NETWORK=[
  ID="1",
  LEASES="4"
]

IMAGE=[
  ID="1",
  RVMS="3"
]

IMAGE=[
  ID="2",
  RVMS="-2"
]
```

Warning: Note that whenever a network, image, datastore or VM is used the corresponding quota counters are created for the user with an unlimited value. This allows to track the usage of each user/group even when quotas are not used.

2.7.4 Setting User/Group Quotas

User/group quotas can be easily set up either trough the command line interface or Sunstone. Note that you need `MANAGE` permissions to set a quota of user, and `ADMIN` permissions to set the quota of a group. In this way, by default, only oneadmin can set quotas for a group, but if you define a group manager she can set specific usage quotas for the users on her group (so distributing resources as required). You can always change this behavior setting the appropriate ACL rules.

To set the quota for a user, e.g. userA, just type:

```
$ oneuser quota userA
```

This will open an editor session to edit a quota template (with some tips about the syntax).

Warning: Usage metrics are included for information purposes (e.g. CPU_USED, MEMORY_USED, LEASES_USED...) you cannot modify them

Warning: You can add as many resource quotas as needed even if they have not been automatically initialized.

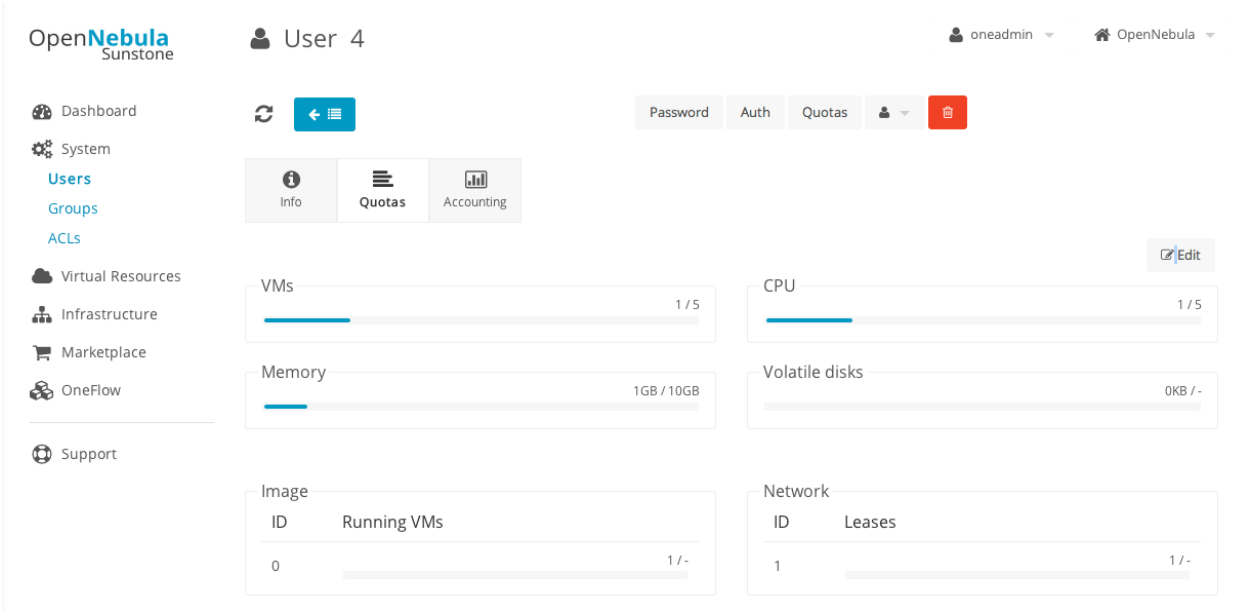
Similarly, you can set the quotas for group A with:

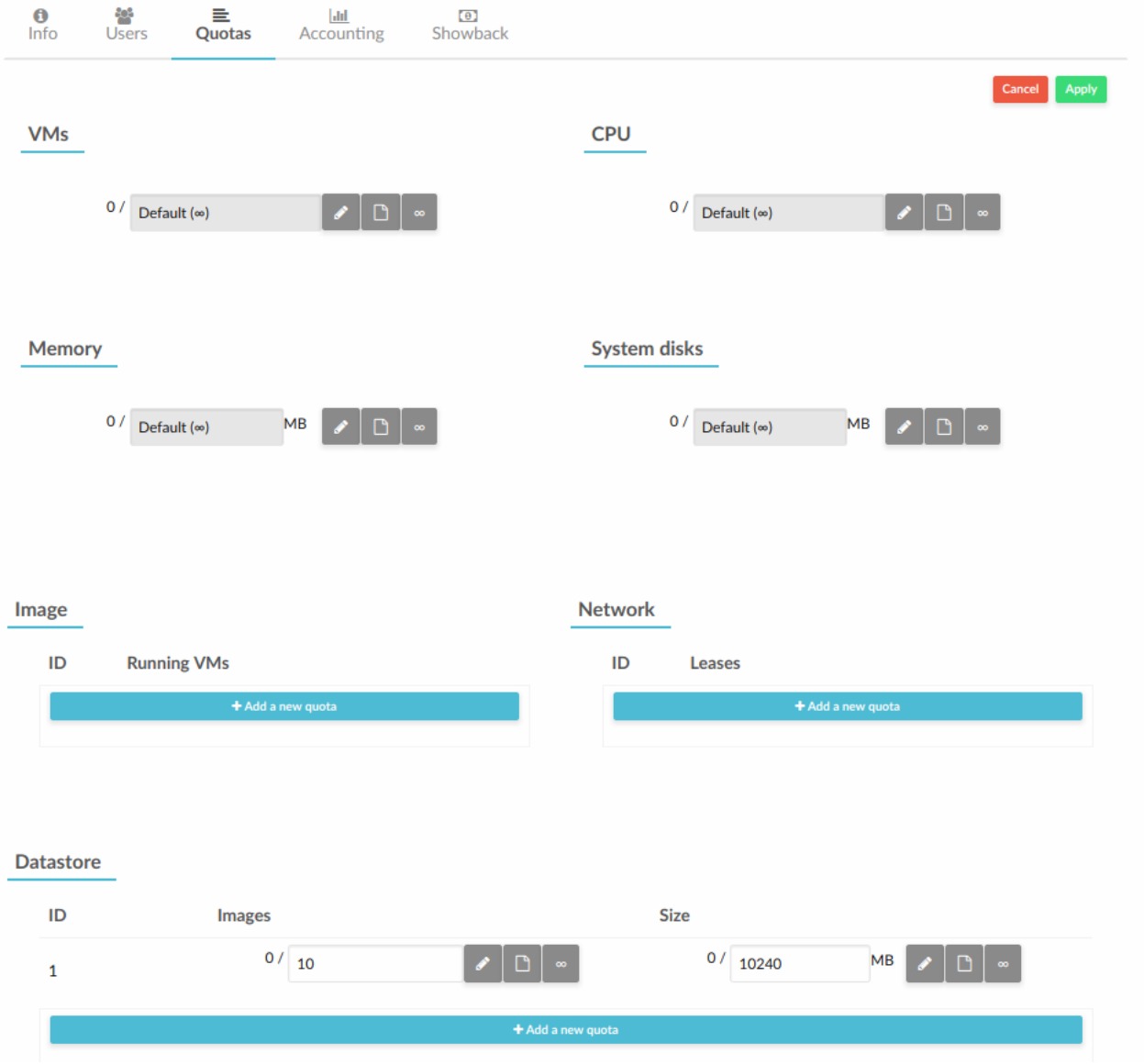
```
$ onegroup quota groupA
```

There is a `batchquota` command that allows you to set the same quotas for several users or groups:

```
$ oneuser batchquota userA,userB,35

$ onegroup batchquota 100..104
```

You can also set the user/group quotas in Sunstone through the user/group tab.





2.7.5 Setting Default Quotas

There are two default quota limit templates, one for users and another for groups. This template applies to all users/groups, unless they have an individual limit set.

Use the `oneuser/onegroup defaultquota` command.

```
$ oneuser defaultquota
```

2.7.6 Checking User/Group Quotas

Quota limits and usage for each user/group is included as part of its standard information, so it can be easily check with the usual commands. Check the following examples:

```
$ oneuser show uA
USER 2 INFORMATION
ID              : 2
NAME            : uA
GROUP           : gA
PASSWORD        : a9993e364706816aba3e25717850c26c9cd0d89d
AUTH_DRIVER     : core
ENABLED         : Yes

USER TEMPLATE


RESOURCE USAGE & QUOTAS

DATASTORE ID  IMAGES (used) IMAGES (limit)    SIZE (used)   SIZE (limit)
1                        1              0           1024              0

VMS           MEMORY (used) MEMORY (limit)     CPU (used)    CPU (limit)
0                      1024              0              1              0

NETWORK ID    LEASES (used) LEASES (limit)
1                        1              0

IMAGE ID        RVMS (used)   RVMS (limit)
1                        0              0
2                        0              0
```

And for the group:

```
$ onegroup show gA
GROUP 100 INFORMATION
ID              : 100
NAME            : gA

USERS
ID
2
3

RESOURCE USAGE & QUOTAS

DATASTORE ID  IMAGES (used) IMAGES (limit)    SIZE (used)   SIZE (limit)
1                        2              0           2048              0

VMS           MEMORY (used) MEMORY (limit)     CPU (used)    CPU (limit)
0                      2048              0              2              0

NETWORK ID    LEASES (used) LEASES (limit)
1                        1              0
2                        1              0

IMAGE ID        RVMS (used)   RVMS (limit)
1                        0              0
2                        0              0
5                        1              0
6                        1              0
```

This information is also available through Sunstone as part of the user/group information.

2.8 Accounting Client

The accounting toolset visualizes and reports resource usage data. This accounting tool addresses the accounting of the virtual resources. It includes resource consumption of the virtual machines as reported from the hypervisor.

2.8.1 Usage

`oneacct` - prints accounting information for virtual machines

```
Usage: oneacct [options]
 -s, --start TIME          First day of the data to retrieve
 -e, --end TIME            Last day of the data to retrieve
 -u, --userfilter user     User name or id to filter the results
 -g, --group group         Group name or id to filter the results
 -H, --host HOST           Host name or id to filter the results
 --xpath XPATH_EXPRESSION  Xpath expression to filter the results. For
                           example: oneacct --xpath 'HISTORY[ETIME>0]'
 -x, --xml                 Show the resource in xml format
 -j, --json                Show the resource in json format
 --split                   Split the output in a table for each VM
 -v, --verbose             Verbose mode
 -h, --help                Show this message
 -V, --version             Show version and copyright information
 --describe                Describe list columns
 -l, --list x,y,z          Selects columns to display with list command
 --csv                     Write table in csv format
 --user name               User name used to connect to OpenNebula
 --password password       Password to authenticate with OpenNebula
 --endpoint endpoint       URL of OpenNebula XML-RPC front-end
```

The time can be written as `month/day/year hour:minute:second`, or any other similar format, e.g `month/day hour:minute`.

To integrate this tool with other systems you can use `-j`, `-x` or `--csv` flags to get all the information in an easy computer readable format.

2.8.2 Accounting Output

The oneacct command shows individual Virtual Machine history records. This means that for a single VM you may get several accounting entries, one for each migration or stop/suspend action. A resize or disk/nic attachment will also create a new entry.

Each entry contains the complete information of the Virtual Machine, including the Virtual Machine monitoring information. By default, only network consumption is reported, see the *Tuning & Extending* section for more information.

When the results are filtered with the `-s` and/or `-e` options, all the history records that were active during that time interval are shown, but they may start or end outside that interval.

For example, if you have a VM that was running from May 1st to June 1st, and you request the accounting information with this command:

```
$ oneacct -s 05/01 -e 06/01
Showing active history records from 2016-05-01 00:00:00 +0200 to 2016-06-02 00:00:00␣
↪+0200

# User 0
```

```
 VID HOSTNAME        ACTION            REAS     START_TIME       END_TIME MEMORY CPU
↪NETRX  NETTX   DISK
  28 host01          terminate         user 05/27 16:40:47 05/27 17:09:20  1024M 0.1
↪   0K     0K  10.4G
  29 host02          none              none 05/27 17:09:28              -   256M   1
↪2.4M   1.3K    10G
```

The record shows the complete history record, and total network consumption. It will not reflect the consumption made only during the month of May.

Other important thing to pay attention to is that active history records, those with `END_TIME` **'-'**, refresh their monitoring information each time the VM is monitored. Once the VM is shut down, migrated or stopped, the `END_TIME` is set and the monitoring information stored is frozen. The final values reflect the total for accumulative attributes, like `NETRX`/`NETTX`.

Sample Output

Obtaining all the available accounting information:

```
$ oneacct
# User 0

 VID HOSTNAME        ACTION            REAS     START_TIME       END_TIME MEMORY CPU
↪NETRX  NETTX   DISK

  13 host01          nic-attach        user 05/17 17:10:57 05/17 17:12:48   256M 0.1
↪19.2K  15.4K     8G
  13 host01          nic-detach        user 05/17 17:12:48 05/17 17:13:48   256M 0.1
↪36.9K    25K     8G
  13 host01          nic-attach        user 05/17 17:13:48 05/17 17:14:54   256M 0.1
↪51.2K  36.4K     8G
  13 host01          nic-detach        user 05/17 17:14:54 05/17 17:17:19   256M 0.1
↪79.8K  61.7K     8G
  13 host01          nic-attach        user 05/17 17:17:19 05/17 17:17:27   256M 0.1
↪79.8K  61.7K     8G
  13 host01          terminate-hard    user 05/17 17:17:27 05/17 17:37:52   256M 0.1
↪124.6K  85.9K     8G
  14 host02          nic-attach        user 05/17 17:38:16 05/17 17:40:00   256M 0.1
↪16.5K  13.2K     8G
  14 host02          poweroff          user 05/17 17:40:00 05/17 17:53:40   256M 0.1
↪38.3K  18.8K     8G
  14 host02          terminate-hard    user 05/17 17:55:55 05/18 14:54:19   256M 0.1
↪   1M  27.3K     8G
```

The columns are:

Column	Meaning
VID	Virtual Machine ID
HOSTNAME	Host name
ACTION	Virtual Machine action that created a new history record
REASON	VM state change reason: • **none**: Virtual Machine still running • **erro**: The VM ended in error • **user**: VM action started by the user
START_TIME	Start time
END_TIME	End time
MEMORY	Assigned memory. This is the requested memory, not the monitored memory consumption
CPU	Number of CPUs. This is the requested number of Host CPU share, not the monitored cpu usage
NETRX	Data received from the network
NETTX	Data sent to the network

Obtaining the accounting information for a given user

```
$ oneacct -u 0 --split
# User 0

 VID HOSTNAME        ACTION           REAS     START_TIME       END_TIME MEMORY CPU
↪NETRX  NETTX   DISK
  12 host01          none             user 05/09 19:20:42 05/09 19:35:23  1024M   1
↪29.8M 638.8K     0K

 VID HOSTNAME        ACTION           REAS     START_TIME       END_TIME MEMORY CPU
↪NETRX  NETTX   DISK
  13 host01          nic-attach       user 05/17 17:10:57 05/17 17:12:48   256M 0.1
↪19.2K  15.4K     8G
  13 host01          nic-detach       user 05/17 17:12:48 05/17 17:13:48   256M 0.1
↪36.9K    25K     8G
  13 host01          nic-attach       user 05/17 17:13:48 05/17 17:14:54   256M 0.1
↪51.2K  36.4K     8G
  13 host01          nic-detach       user 05/17 17:14:54 05/17 17:17:19   256M 0.1
↪79.8K  61.7K     8G
  13 host01          nic-attach       user 05/17 17:17:19 05/17 17:17:27   256M 0.1
↪79.8K  61.7K     8G
  13 host01          terminate-hard   user 05/17 17:17:27 05/17 17:37:52   256M 0.1
↪124.6K  85.9K     8G

 VID HOSTNAME        ACTION           REAS     START_TIME       END_TIME MEMORY CPU
↪NETRX  NETTX   DISK
  14 host02          nic-attach       user 05/17 17:38:16 05/17 17:40:00   256M 0.1
↪16.5K  13.2K     8G
  14 host02          poweroff         user 05/17 17:40:00 05/17 17:53:40   256M 0.1
↪38.3K  18.8K     8G
  14 host02          terminate-hard   user 05/17 17:55:55 05/18 14:54:19   256M 0.1
↪   1M  27.3K     8G

 VID HOSTNAME        ACTION           REAS     START_TIME       END_TIME MEMORY CPU
↪NETRX  NETTX   DISK
  29 host02          none             none 05/27 17:09:28             -   256M   1
↪2.4M   1.3K    10G
```

In case you use CSV output (`--csv`) you will het a header with the name of each column and then the data. For example:

```
$ oneacct --csv
UID,VID,HOSTNAME,ACTION,REASON,START_TIME,END_TIME,MEMORY,CPU,NETRX,NETTX,DISK
0,12,host01,none,user,05/09 19:20:42,05/09 19:35:23,1024M,1,29.8M,638.8K,0K
0,13,host01,nic-attach,user,05/17 17:10:57,05/17 17:12:48,256M,0.1,19.2K,15.4K,8G
0,13,host01,nic-detach,user,05/17 17:12:48,05/17 17:13:48,256M,0.1,36.9K,25K,8G
0,13,host01,nic-attach,user,05/17 17:13:48,05/17 17:14:54,256M,0.1,51.2K,36.4K,8G
0,13,host01,nic-detach,user,05/17 17:14:54,05/17 17:17:19,256M,0.1,79.8K,61.7K,8G
0,13,host01,nic-attach,user,05/17 17:17:19,05/17 17:17:27,256M,0.1,79.8K,61.7K,8G
0,13,host01,terminate-hard,user,05/17 17:17:27,05/17 17:37:52,256M,0.1,124.6K,85.9K,8G
0,14,host02,nic-attach,user,05/17 17:38:16,05/17 17:40:00,256M,0.1,16.5K,13.2K,8G
0,14,host01,poweroff,user,05/17 17:40:00,05/17 17:53:40,256M,0.1,38.3K,18.8K,8G
0,14,host02,terminate-hard,user,05/17 17:55:55,05/18 14:54:19,256M,0.1,1M,27.3K,8G
0,29,host02,none,none,05/27 17:09:28,-,256M,1,2.4M,1.3K,10G
```

Output Reference

If you execute oneacct with the `-x` option, you will get an XML output defined by the following xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  targetNamespace="http://opennebula.org/XMLSchema" xmlns="http://opennebula.org/
↪XMLSchema">

  <xs:element name="HISTORY_RECORDS">
        <xs:complexType>
          <xs:sequence maxOccurs="1" minOccurs="1">
                <xs:element ref="HISTORY" maxOccurs="unbounded" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
  </xs:element>

  <xs:element name="HISTORY">
        <xs:complexType>
          <xs:sequence>
                <xs:element name="OID" type="xs:integer"/>
                <xs:element name="SEQ" type="xs:integer"/>
                <xs:element name="HOSTNAME" type="xs:string"/>
                <xs:element name="HID" type="xs:integer"/>
                <xs:element name="CID" type="xs:integer"/>
                <xs:element name="STIME" type="xs:integer"/>
                <xs:element name="ETIME" type="xs:integer"/>
                <xs:element name="VM_MAD" type="xs:string"/>
                <xs:element name="TM_MAD" type="xs:string"/>
                <xs:element name="DS_ID" type="xs:integer"/>
                <xs:element name="PSTIME" type="xs:integer"/>
                <xs:element name="PETIME" type="xs:integer"/>
                <xs:element name="RSTIME" type="xs:integer"/>
                <xs:element name="RETIME" type="xs:integer"/>
                <xs:element name="ESTIME" type="xs:integer"/>
                <xs:element name="EETIME" type="xs:integer"/>

                <!-- REASON values:
                  NONE  = 0 History record is not closed yet
                  ERROR = 1 History record was closed because of an error
```

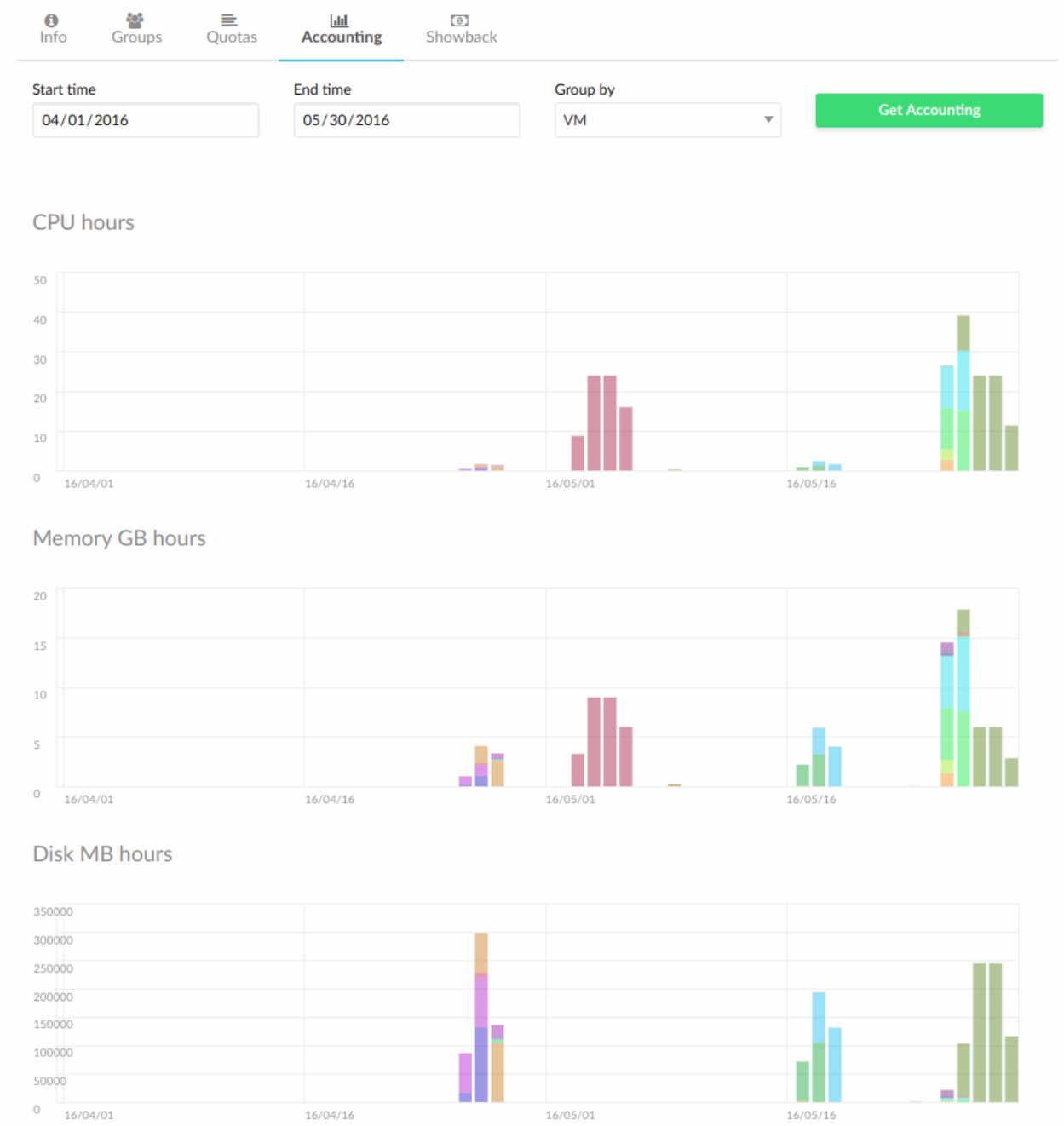
```
    USER  = 2 History record was closed because of a user action
  -->
  <xs:element name="REASON" type="xs:integer"/>

  <!-- ACTION values:
    NONE_ACTION             = 0
    MIGRATE_ACTION          = 1
    LIVE_MIGRATE_ACTION     = 2
    SHUTDOWN_ACTION         = 3
    SHUTDOWN_HARD_ACTION    = 4
    UNDEPLOY_ACTION         = 5
    UNDEPLOY_HARD_ACTION    = 6
    HOLD_ACTION             = 7
    RELEASE_ACTION          = 8
    STOP_ACTION             = 9
    SUSPEND_ACTION          = 10
    RESUME_ACTION           = 11
    BOOT_ACTION             = 12
    DELETE_ACTION           = 13
    DELETE_RECREATE_ACTION  = 14
    REBOOT_ACTION           = 15
    REBOOT_HARD_ACTION      = 16
    RESCHED_ACTION          = 17
    UNRESCHED_ACTION        = 18
    POWEROFF_ACTION         = 19
    POWEROFF_HARD_ACTION    = 20
    DISK_ATTACH_ACTION      = 21
    DISK_DETACH_ACTION      = 22
    NIC_ATTACH_ACTION       = 23
    NIC_DETACH_ACTION       = 24
    DISK_SNAPSHOT_CREATE_ACTION = 25
    DISK_SNAPSHOT_DELETE_ACTION = 26
    TERMINATE_ACTION        = 27
    TERMINATE_HARD_ACTION   = 28
  -->
  <xs:element name="ACTION" type="xs:integer"/>

  <xs:element name="VM">
    <xs:complexType>
          <xs:sequence>
            <xs:element name="ID" type="xs:integer"/>
            <xs:element name="UID" type="xs:integer"/>
            <xs:element name="GID" type="xs:integer"/>
            <xs:element name="UNAME" type="xs:string"/>
            <xs:element name="GNAME" type="xs:string"/>
            <xs:element name="NAME" type="xs:string"/>
            <xs:element name="PERMISSIONS" minOccurs="0" maxOccurs="1">
                  <xs:complexType>
                    <xs:sequence>
                          <xs:element name="OWNER_U" type="xs:integer"/>
                          <xs:element name="OWNER_M" type="xs:integer"/>
                          <xs:element name="OWNER_A" type="xs:integer"/>
                          <xs:element name="GROUP_U" type="xs:integer"/>
                          <xs:element name="GROUP_M" type="xs:integer"/>
                          <xs:element name="GROUP_A" type="xs:integer"/>
                          <xs:element name="OTHER_U" type="xs:integer"/>
                          <xs:element name="OTHER_M" type="xs:integer"/>
                          <xs:element name="OTHER_A" type="xs:integer"/>
```

```
                                </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="LAST_POLL" type="xs:integer"/>

                        <!-- STATE values,
                        see http://docs.opennebula.org/stable/user/references/vm_
↪states.html
                        -->
                        <xs:element name="STATE" type="xs:integer"/>

                        <!-- LCM_STATE values, this sub-state is relevant only when␣
↪STATE is
                                ACTIVE (4)
                        see http://docs.opennebula.org/stable/user/references/vm_
↪states.html
                        -->
                        <xs:element name="LCM_STATE" type="xs:integer"/>
                        <xs:element name="PREV_STATE" type="xs:integer"/>
                        <xs:element name="PREV_LCM_STATE" type="xs:integer"/>
                        <xs:element name="RESCHED" type="xs:integer"/>
                        <xs:element name="STIME" type="xs:integer"/>
                        <xs:element name="ETIME" type="xs:integer"/>
                        <xs:element name="DEPLOY_ID" type="xs:string"/>
                        <xs:element name="MONITORING">
                        <!--
                              <xs:complexType>
                                <xs:all>
                                      <- Percentage of 1 CPU consumed (two fully␣
↪consumed cpu is 200) ->
                                      <xs:element name="CPU" type="xs:decimal"␣
↪minOccurs="0" maxOccurs="1"/>

                                      <- MEMORY consumption in kilobytes ->
                                      <xs:element name="MEMORY" type="xs:integer"␣
↪minOccurs="0" maxOccurs="1"/>

                                      <- NETTX: Sent bytes to the network ->
                                      <xs:element name="NETTX" type="xs:integer"␣
↪minOccurs="0" maxOccurs="1"/>

                                      <- NETRX: Received bytes from the network ->
                                      <xs:element name="NETRX" type="xs:integer"␣
↪minOccurs="0" maxOccurs="1"/>
                                </xs:all>
                              </xs:complexType>
                        -->
                        </xs:element>
                        <xs:element name="TEMPLATE" type="xs:anyType"/>
                        <xs:element name="USER_TEMPLATE" type="xs:anyType"/>
                        <xs:element name="HISTORY_RECORDS">
                        </xs:element>
                        <xs:element name="SNAPSHOTS" minOccurs="0" maxOccurs=
↪"unbounded">
                              <xs:complexType>
                                <xs:sequence>
                                      <xs:element name="DISK_ID" type="xs:integer"/>
                                      <xs:element name="SNAPSHOT" minOccurs="0"␣
↪maxOccurs="unbounded">
```

```
                                        <xs:complexType>
                                            <xs:sequence>
                                              <xs:element name="ACTIVE" type="xs:
↪string" minOccurs="0" maxOccurs="1"/>
                                              <xs:element name="CHILDREN" type=
↪"xs:string" minOccurs="0" maxOccurs="1"/>
                                              <xs:element name="DATE" type="xs:
↪integer"/>
                                              <xs:element name="ID" type="xs:
↪integer"/>
                                              <xs:element name="NAME" type="xs:
↪string" minOccurs="0" maxOccurs="1"/>
                                              <xs:element name="PARENT" type="xs:
↪integer"/>
                                              <xs:element name="SIZE" type="xs:
↪integer"/>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

2.8.3 Sunstone

Sunstone also displays information about accounting. Information is accessible via the User dialogs for the user and admin views. The cloud view can access the metering information in the dashboard, whereas the group admin user can access them under the users section.



2.8.4 Tuning & Extending

There are two kinds of monitoring values:

- Instantaneous values: For example, `VM/CPU` or `VM/MEMORY` show the memory consumption last reported by the monitoring probes.
- Accumulative values: For example, `VM/NETRX` and `VM/NETTX` show the total network consumption since the history record started.

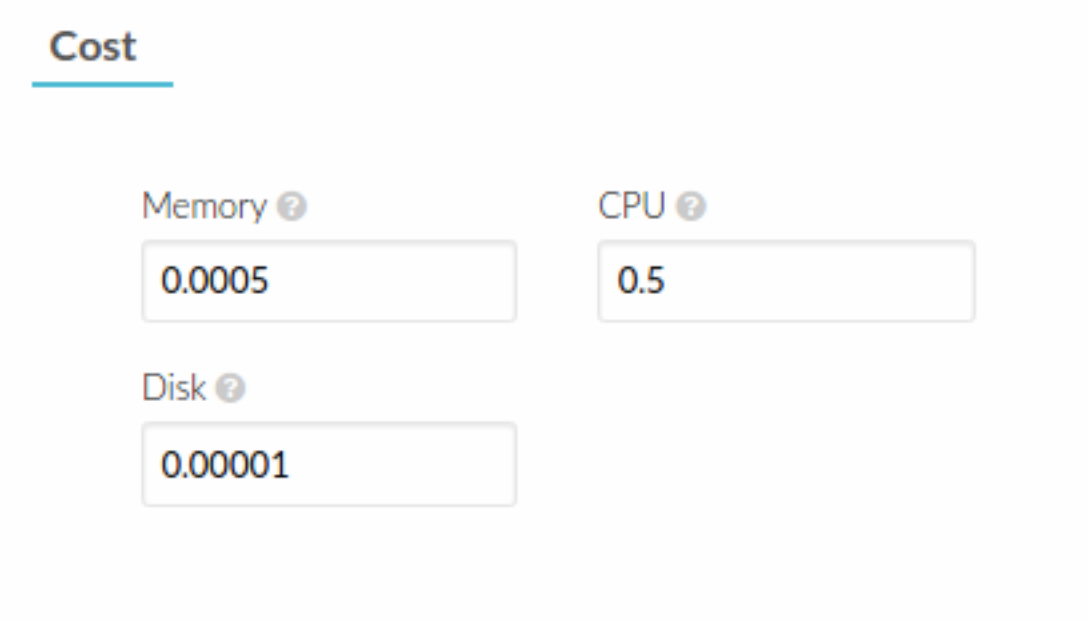
Developers interacting with OpenNebula using the Ruby bindings can use the VirtualMachinePool.accounting method to retrieve accounting information filtering and ordering by multiple parameters.

2.9 Showback

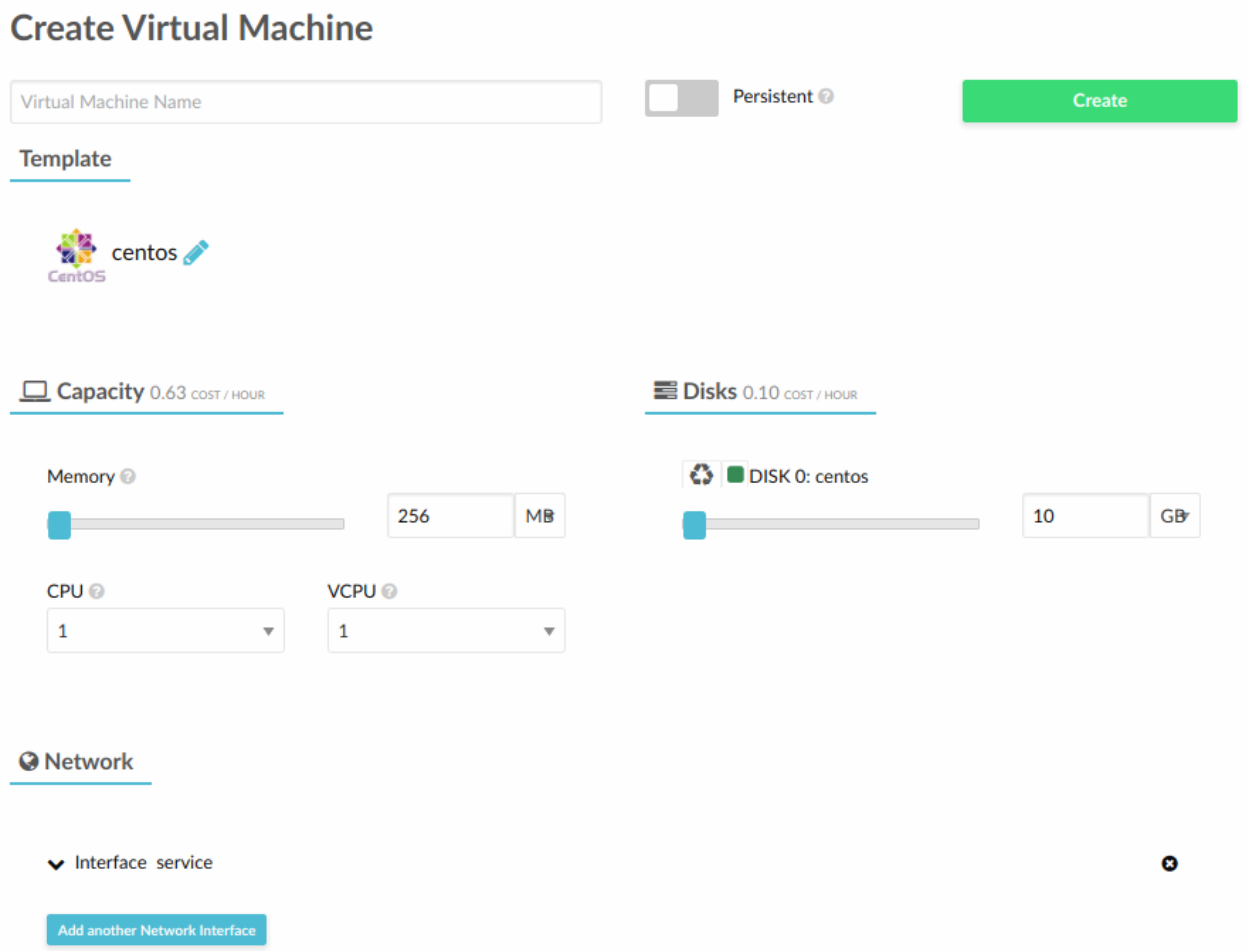
The showback toolset reports resource usage cost, and allows the integration with chargeback and billing platforms. The toolset generates showback reports using the information retrieved from OpenNebula.

2.9.1 Set the VM Cost

Each VM Template can optionally define a cost (see the *syntax here*). The cost is defined as **cost per cpu per hour**, and **cost per memory MB per hour**. The cost units are abstract and their equivalent to monetary or other cost metrics have to be defined in each deployment.



Using this cost schema allows the users to resize the Virtual Machine instances.



There is a default cost that will be applied to VM Templates without a cost defined. It can be set in the oned.conf file.

Warning: If your users can access the Sunstone 'user' view, it's important to set a default cost. These users can manage their own Templates, which won't have a specific cost assigned.

2.9.2 Calculate Monthly Reports

Before the cost reports can be seen by the users, the administrator has to generate them. To create the monthly cost reports, use the `oneshowback` command:

```
$ oneshowback calculate -h
Usage: oneshowback [options]
    -s, --start TIME                 First month of the data
    -e, --end TIME                   Last month of the data
```

When this command is executed, the OpenNebula core reads all the accounting records, and calculates the total cost for each month. The records include the total cost of the month, and basic information about the VM and its owner. This information is then stored in the database, to be consumed with the `oneshowback list` command.

The monthly cost of each VM is calculated as the sum of:

- `CPU_COST` * `CPU` * `HOURS`

- MEMORY_COST * MEMORY * HOURS
- DISK_COST * DISK_SIZE * HOURS

The number of hours is calculated as the total number of hours that a VM has been `running`. The time a VM is in other states, such as `pending`, `poweroff`, or `stopped` does not count towards the cost.

If the time range includes the current month, OpenNebula will calculate the cost up to today's date. It is up to the administrators to leave the current month out of the showback records, to update it daily, or hourly. In any case, it is important to re-calculate it when the month ends. This operation can be easily automated by a cron job.

The `oneshowback` command can only be executed by the oneadmin user.

Some examples:

To calculate all records, starting from March up to today:

```
$ oneshowback calculate --start "03/2016"
```

To calculate only September:

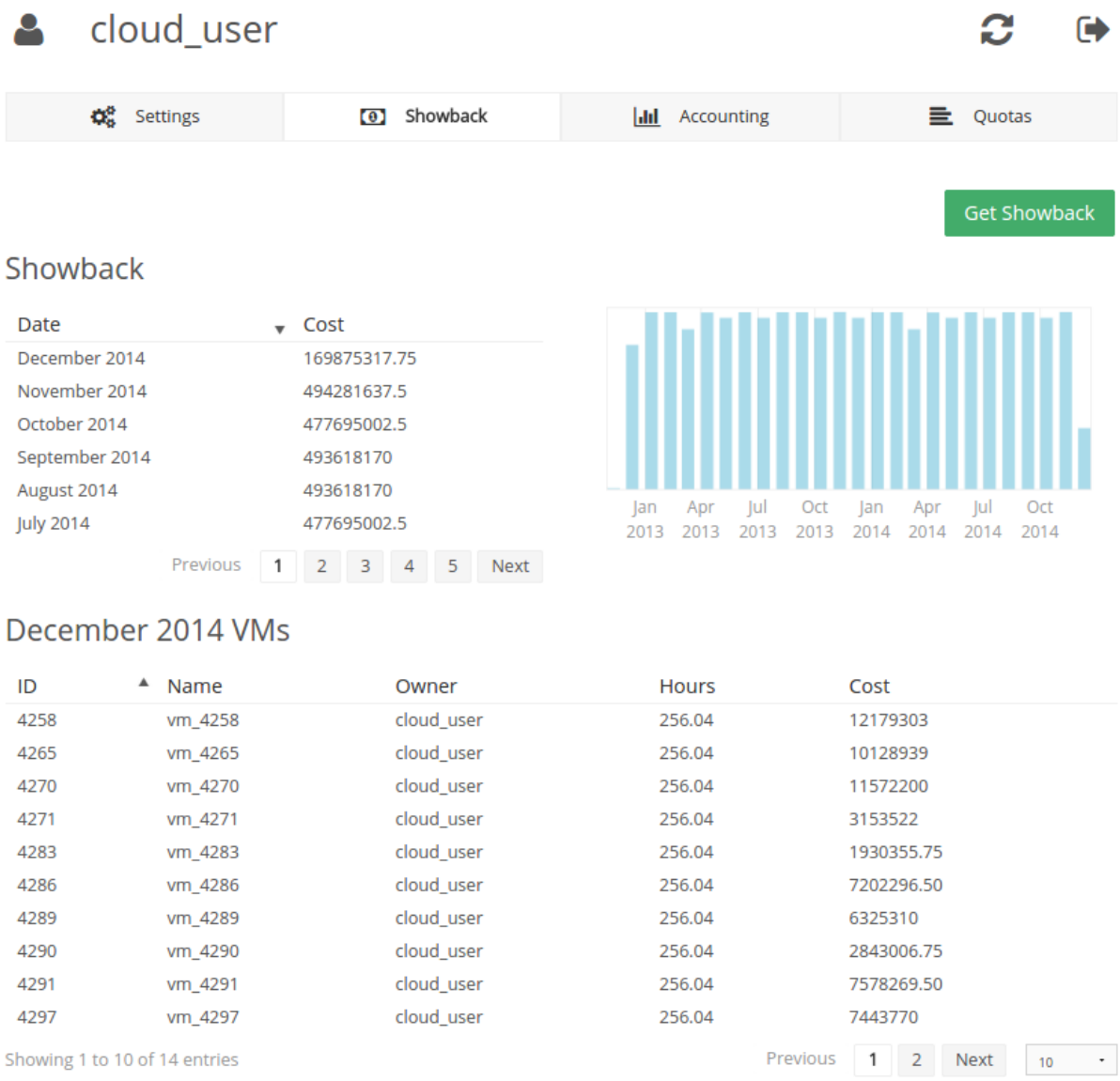
```
$ oneshowback calculate --start "09/2016" --end "09/2016"
```

Note: This is a resource intensive operation. For big deployments, it is recommended to use the `--start` option to process only the last missing months.

Note: Existing records can be re-calculated. This can be useful to update old records when a VM is renamed, or the owner is changed. In this case, the cost of previous months will be also assigned to the new user.

2.9.3 Retrieve Monthly Reports

Any user or administrator can see their monthly showback reports from the CLI or Sunstone:




```
## USAGE
list
        Returns the showback records
        valid options: start_time, end_time, userfilter, group, xml, json, verbose,␣
↪help, version, describe, list, csv, user, password, endpoint

## OPTIONS
     -s, --start TIME          First month of the data
     -e, --end TIME            Last month of the data
     -u, --userfilter user     User name or id to filter the results
     -g, --group group         Group name or id to filter the results
     -x, --xml                 Show the resource in xml format
     -j, --json                Show the resource in json format
     -v, --verbose             Verbose mode
     -h, --help                Show this message
     -V, --version             Show version and copyright information
     --describe                Describe list columns
```

```
    -l, --list x,y,z          Selects columns to display with list command
    --csv                     Write table in csv format
    --user name               User name used to connect to OpenNebula
    --password password       Password to authenticate with OpenNebula
    --endpoint endpoint       URL of OpenNebula xmlrpc frontend
```

2.9.4 Disable Showback in Sunstone

Showback reports can be disabled in any of the Sunstone views modifying the yaml file of those views. These files can be found in `/etc/one/sunstone-views`

```
...
features:
    showback: false
```

2.9.5 Tuning & Extending

To integrate the showback reports with external tools, you can get the CLI output as **xml**, **json**, or **csv** data.

```
$ oneshowback list -u cloud_user --list YEAR,MONTH,VM_ID,COST --csv
YEAR,MONTH,VM_ID,COST
2015,10,4258,1824279.62
2015,10,4265,433749.03
2015,11,4258,34248600
```

Developers interacting with OpenNebula using the Ruby bindings can use the VirtualMachinePool.showback method to retrieve showback information filtering and ordering by multiple parameters.

CHAPTER

THREE

VIRTUAL NETWORK MANAGEMENT

3.1 Overview

This chapter contains documentation on how to create and manage *Virtual Networks*, how to define and manage *Security Groups*, which will allow users and administrators to define firewall rules and apply them to the Virtual Machines, and how to create and manage *Virtual Routers* which are an OpenNebula resource that provide routing across Virtual Networks.

3.1.1 How Should I Read This Chapter

Before reading this chapter, you should have already installed your Frontend, the KVM Hosts or vCenter node and have an OpenNebula cloud up and running with at least one virtualization node.

3.1.2 Hypervisor Compatibility

Section	Compatibility
Virtual Networks	This Section applies to both KVM and vCenter.
Security Groups	This Section applies to KVM.
Virtual Routers	This Section applies to both KVM and vCenter.

3.2 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.

3.2.1 Virtual Network Definition

A Virtual Network definition consists of three different parts:

- The **underlying physical network infrastructure** that will support it, including the network driver.
- The **logical address space** available. Addresses associated to a Virtual Network can be IPv4, IPv6, dual stack IPv4-IPv6 or Ethernet.
- The **guest configuration attributes** to setup the Virtual Machine network, that may include for example network masks, DNS servers or gateways.

Physical Network Attributes

To define a Virtual Network include:

- NAME to refer this Virtual Network.
- VN_MAD the driver to implement this Virtual Network. Depending on the driver you may need to set additional attributes, check the following to get more details:
  - Define a bridged network
  - Define a 802.1Q network
  - Define a VXLAN network
  - Define a OpenvSwitch network

For example, to define a 802.1Q Virtual Network you would add:

```
NAME    = "Private Network"
VN_MAD  = "802.1Q"
PHYDEV  = "eth0"
```

Address Space

The addresses available in a Virtual Network are defined by one or more Address Ranges (AR). Each AR defines a continuous address range and optionally, configuration attributes that will override the first level attributes defined in the Virtual Network. There are four types of ARs:

- **IPv4**, to define a contiguous IPv4 address set (classless), *see more details here*
- **IPv6**, to define global and ULA IPv6 networks, *see full details here*
- **Dual stack**, each NIC in the network will get both a IPv4 and a IPv6 address, *see more here*
- **Ethernet**, just MAC addresses are generated for the VMs. You should use this AR when an external service is providing the IP addresses, such a DHCP server, *see more details here*

For example, to define the IPv4 address range 10.0.0.150 - 10.0.0.200

```
AR=[
    TYPE = "IP4",
    IP   = "10.0.0.150",
    SIZE = "51",
]
```

Guest Configuration Attributes (Context)

To setup the guest network, the Virtual Network may include additional information to be injected into the VM at boot time. These contextualization attributes may include for example network masks, DNS servers or gateways. For example, to define a gateway and DNS server for the virtual machines in the Virtual Network, simply add:

```
DNS = "10.0.0.23"
GATEWAY = "10.0.0.1"
```

These attributes are automatically added to the VM and processed by the context packages. Virtual Machines just need to add:

```
CONTEXT = [
  NETWORK="yes"
]
```

See here for a full list of supported attributes

Virtual Network Definition Example

Getting all the three pieces together we get:

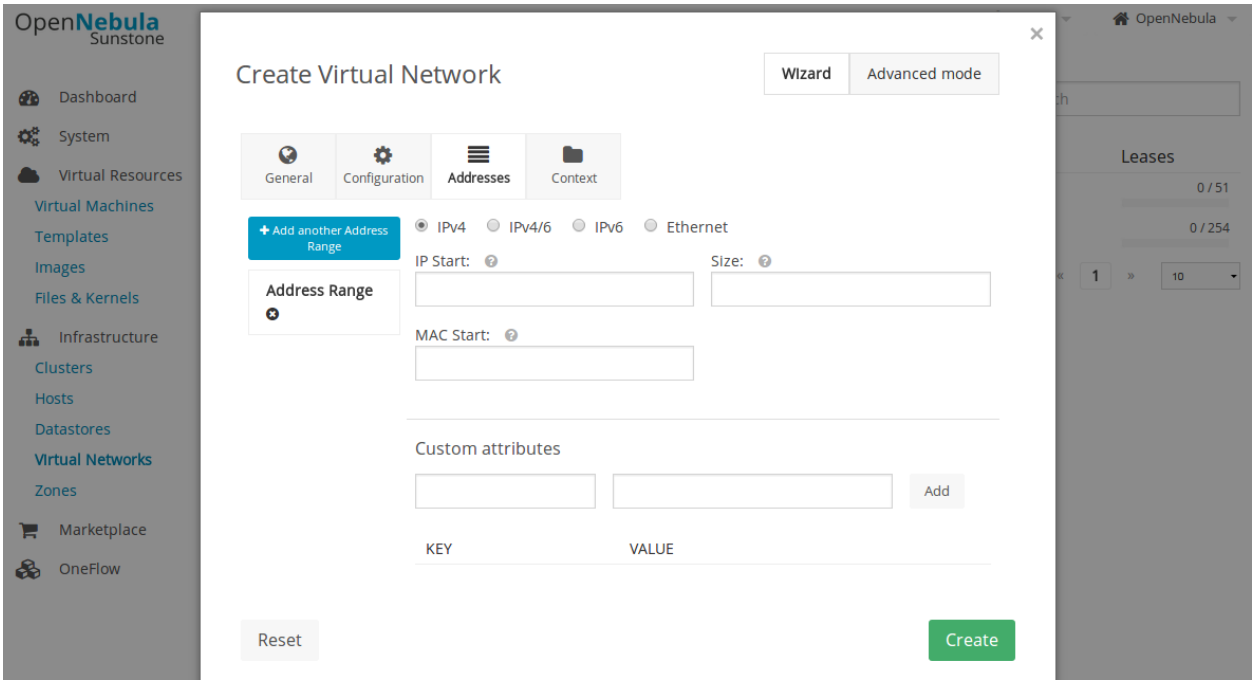
```
NAME    = "Private"
VN_MAD  = "802.1Q"
PHYDEV  = "eth0"

AR=[
    TYPE = "IP4",
    IP   = "10.0.0.150",
    SIZE = "51"
]

DNS     = "10.0.0.23"
GATEWAY = "10.0.0.1"

DESCRIPTION = "A private network for VM inter-communication"
```

This file will create a IPv4 network using VLAN tagging, the VLAN ID in this case is assigned by OpenNebula. The network will lease IPs in the range 10.0.0.150 - 10.0.0.200. Virtual Machines in this network will get a lease in the range and configure DNS servers to 10.0.0.23 and 10.0.0.1 as default gateway.

See here for more examples

3.2.2 Adding and Deleting Virtual Networks

Note: 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.



To create a new network put its configuration in a file, for example using the contents above, and then execute:

```
$ onevnet create priv.net
ID: 4
```

You can delete a virtual network using its ID or name:

```
$ onevnet delete 0
$ onevnet delete "Private"
```

To list the virtual networks in the system use `onevnet list`:

```
$ onevnet list
ID USER         GROUP        NAME            CLUSTER     BRIDGE    LEASES
 0 admin        oneadmin     Private         0,100       onebr.10       0
 1 admin        oneadmin     Public          0,101       vbr0           0
```

In the output above, `USER` is the owner of the network and `LEASES` the number of addresses assigned to a virtual machine or reserved.

You can check the details of a Virtual Network with the `onevnet show` command:

```
$ onevnet show 1
  VIRTUAL NETWORK 4 INFORMATION
  ID              : 4
  NAME            : Private
  USER            : ruben
  GROUP           : oneadmin
  CLUSTERS        : 0
  BRIDGE          : onebr4
  VN_MAD          : 802.1Q
  PHYSICAL DEVICE: eth0
  VLAN ID         : 6
  USED LEASES     : 0
```

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

VIRTUAL NETWORK TEMPLATE
BRIDGE="onebr4"
DESCRIPTION="A private network for VM inter-communication"
DNS="10.0.0.23"
GATEWAY="10.0.0.1"
PHYDEV="eth0"
SECURITY_GROUPS="0"
VN_MAD="802.1Q"

ADDRESS RANGE POOL
AR 0
SIZE            : 51
LEASES          : 0

RANGE                                   FIRST                                   LAST
MAC                           02:00:0a:00:00:96              02:00:0a:00:00:c8
IP                                   10.0.0.150                      10.0.0.200
```

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

Virtual Network Tips

- You may have some used IPs in a VNET so you do not want them to be assigned. You can add as many ARs as you need to implement these address gaps. Alternatively you can put address on hold to prevent them to be assigned.
- ARs can be of SIZE = 1 to define single addresses lease scheme.
- ARs does not need to be of the same type or belong to the same IP network. To accommodate this use case you can overwrite context attributes in the AR, for example adding attributes like NETWORK_MASK or DNS in the AR definition.
- *Super-netting*, you can even combine ARs overwriting the physical attributes, e.g. `BRIDGE` or `VLAN_ID`. This way a Virtual Network can be a logical super-net, e.g. DMZ, that can be implemented through multiple VLANs each using a different hypervisor bridge.
- There are no need to plan all your IP assignment plan beforehand, ARs can be added and modified after the Virtual Network is created, see below.

3.2.3 Updating a Virtual Network

After creating a Virtual Network, you can use the `onevnet update` command to update the following attributes:

- Any attribute corresponding to the context or description.
- Physical network configuration attributes, e.g. `PHYDEV` or `VLAN_ID`.
- Any custom tag.

Also the name of the Virtual Network can be changed with `onevnet rename` command.

3.2.4 Managing Address Ranges

Addresses are structured in Address Ranges (AR). Address Ranges can be dynamically added or removed from a Virtual Network. In this way, you can easily add new addresses to an existing Virtual Network if the current addresses are exhausted.

Adding and Removing Address Ranges

A new AR can be added using exactly the same definition parameters as described above. For example the following command will add a new AR of 20 IP addresses:

```
onevnet addar Private --ip 10.0.0.200 --size 20
```

In the same way you can remove an AR:

```
onevnet rmar Private 2
```

Updating Address Ranges

You can update the following attributes of an AR:

- `SIZE`, assigned addresses cannot fall outside of the range.
- IPv6 prefix: `GLOBAL_PREFIX` and `ULA_PREFIX`
- Any custom attribute that may override the Virtual Network defaults.

The following command shows how to update an AR using the CLI, an interactive editor session will be stated:

```
onevnet updatear Private 0
```

Hold and Release Leases

Addresses can be temporarily be marked as `hold`. They are still part of the network, but they will not be assigned to any virtual machine.

To do so, use the 'onevnet hold' and 'onevnet release' commands. By default, the address will be put on hold in all ARs containing it; if you need to hold the IP of a specific AR you can specified it with the '-a <AR_ID>' option.

```
#Hold IP 10.0.0.120 in all ARs
$ onevnet hold "Private Network" 10.0.0.120

#Hold IP 10.0.0.123 in AR 0
$ onevnet hold 0 10.0.0.123 -a 0
```

You see the list of leases on hold with the 'onevnet show' command, they'll show up as used by virtual machine -1, 'V: -1'

3.2.5 Using a Virtual Network

Once the Virtual Networks are setup, they can be made available to users based on access rights and ownership. The preferred way to do so is through *Virtual Data Center abstraction*. By default, all Virtual Networks are automatically available to the group `users`.

Attach a Virtual Machine to a Virtual Network

To attach a Virtual Machine to a Virtual Network simply specify its name or ID in the `NIC` attribute. For example, to define VM with a network interface connected to the `Private` Virtual Network just include in the template:

```
NIC = [ NETWORK = "Private" ]
```

Equivalently you can use the network ID as:

```
NIC = [ NETWORK_ID = 0 ]
```

The Virtual Machine will also get a free address from any of the address ranges of the network. You can also request a specific address just by adding the `IP` or `MAC` to `NIC`. For example to put a Virtual Machine in the network `Private` and request 10.0.0.153 use:

```
NIC = [ NETWORK = "Network", IP = 10.0.0.153 ]
```

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

Warning: Users can only attach VMs or make reservations from Virtual Networks with **USE** rights on it. See the *Managing Permissions documentation* for more information.

Configuring the Virtual Machine Network

Hypervisors will set the MAC address for the NIC of the Virtual Machines, but not the IP address. The IP configuration inside the guest is performed by the contextualization process, check the *contextualization guide* to learn how to prepare your Virtual Machines to automatically configure the network

Note: Altenatively a custom external service can configure the Virtual Machine network (e.g. your own DHCP server in a separate virtual machine)

3.2.6 Virtual Network Self-Provisioning: Reservations

Virtual Networks implement a simple self-provisioning scheme, that allows users to create their own networks consisting of portions of an existing Virtual Network. Each portion is called a Reservation. To implement this you need to:

- **Define a VNET**, with the desired ARs and configuration attributes. These attributes will be inherited by any Reservation, so the final users do not need to deal with low-level networking details.
- **Setting up access**. In order to make a Reservation, users needs USE rights on the Virtual Network, just as if they would use it to directly to provision IPs from it.
- **Make Reservations**. Users can easily request specific addresses or just a number of addresses from a network. Reservations are placed in a new Virtual Network for the user.
- **Use Reservations**. Reservations are Virtual Networks and offer the same interface, so simply point any Virtual Machine to them. The number of addresses and usage stats are shown also in the same way.

Make and delete Reservations

To make a reservations just choose the source Virtual Network, the number of addresses and the name of the reservation. For example to reserve 10 addresses from Private and place it on MyVNET just:

```
$ onevnet reserve Private -n MyVNET -s 10
Reservation VNET ID: 7
```

As a result a new VNET has been created:

```
$ onevnet list
ID USER         GROUP        NAME            CLUSTER    BRIDGE   LEASES
 0 admin        oneadmin     Private         -          vbr1         10
 7 helen        users        MyVNET          -          vbr1          0
```

Note that Private shows 10 address leases in use, those reserved by Virtual Network 7. Also note that both networks share the same configuration, e.g. `BRIDGE`.

Reservations can include advanced options such as:

- The AR where you want to make the reservation from in the source Virtual Network
- The starting IP or MAC to make the reservation from

A reservation can be remove just as a regular Virtual Network:

```
$ onevnet delete MyVNET
```

Using Reservations

To use a reservation you can use it as any other Virtual Network; as they expose the same interface. For example, to attach a virtual machine to the previous Reservation:

```
NIC = [ NETWORK = "MyVNET"]
```

Updating Reservations

A Reservation can be also extended with new addresses. This is, you can add a new reservation to an existing one. This way a user can refer to its own network with a controlled and deterministic address space.

Note: Reservation increase leases counters on the user and group, and they can be limited through a quota.

Note: The reservation interface is exposed by Sunstone in a very convenient way.

3.3 Virtual Routers

Virtual Routers provide routing across Virtual Networks. The administrators can easily connect Virtual Networks from Sunstone and the CLI. The routing itself is implemented with a Virtual Machine appliance available though the market place. This Virtual Machine can be seamlessly deployed in high availability mode.

3.3.1 Download the Virtual Router Appliance

OpenNebula provides a light weight Alpine-based virtual router. The virtual router image is prepared to run in a HA mode, and process the context information from OpenNebula. So its base capabilities can be easily extended.

- Download the appliance from the market place. For example to put the virtual router image in the default datastore and create a Virtual Router template named vrotuer_apline use:

```
$onemarketapp export  'alpine-vrouter (KVM)' vrouter_alpine --datastore default --
↪vmname vrouter_alpine
IMAGE
    ID: 9
VMTEMPLATE
    ID: 8
```

- Check that the resources are properly created, an update them to your OpenNebula installation if needed.

```
$ oneimage show 9  # 9 is the IMAGE ID from the previous onemarketapp command
$ onetemplate show 8 # 8 is for the VMTEMPLATE ID
```

Note: For vCenter infrastructures an ova with the preconfigured image can be imported from the following URL:

https://s3-eu-west-1.amazonaws.com/opennebula-marketplace/alpine-quagga.ova

After that you'll only need to import new templates from vcenter and set the template as Virtual Router at the bottom of the General tab of the template update wizard.

3.3.2 Creating a new Virtual Router

New Virtual Routers are created from a special type of VM Template, as the one created automatically when downloading the market app.

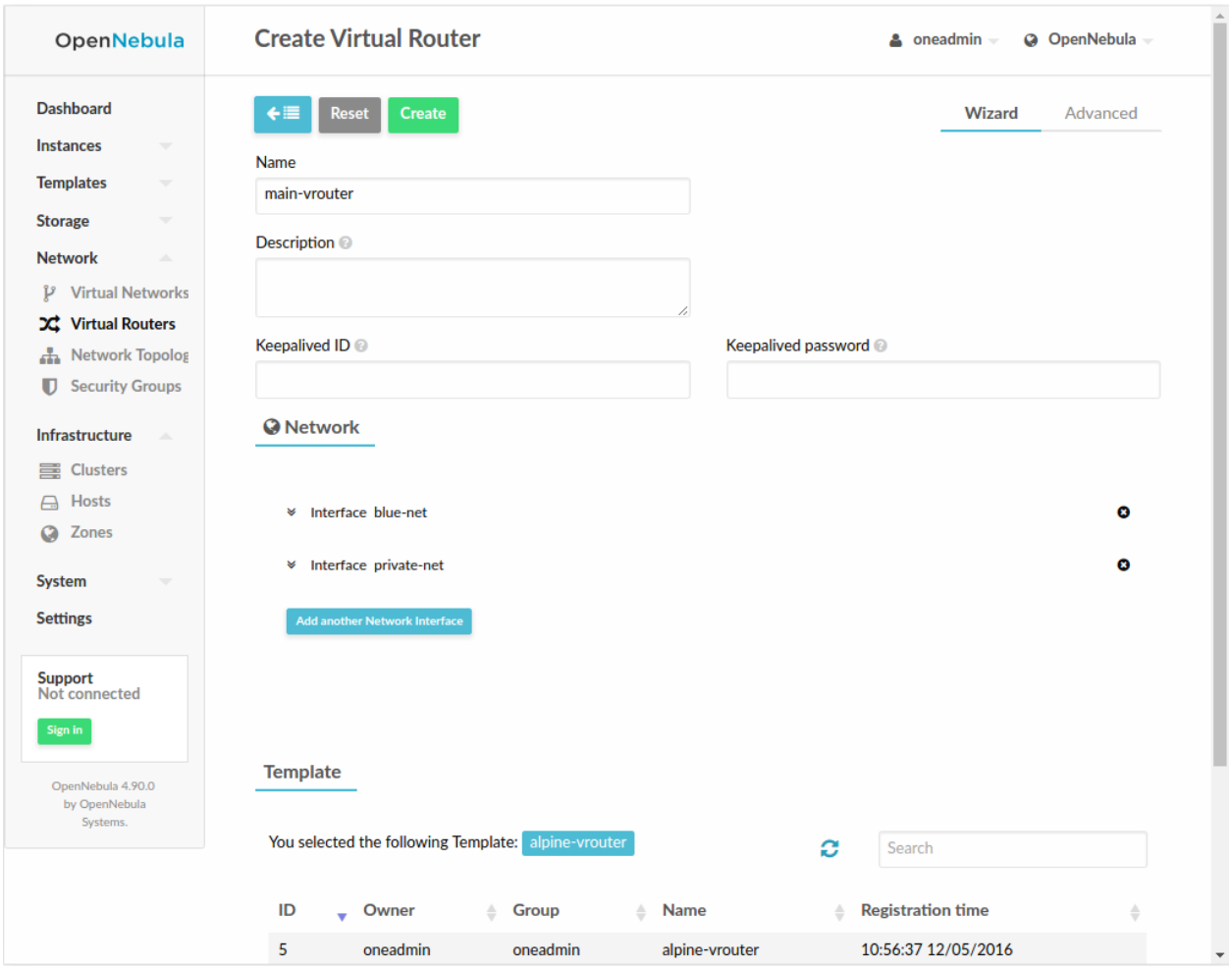
Sunstone

To create a new Virtual Router from Sunstone, follow the wizard to select the Virtual Networks that will get logically linked to it. This connection takes effect when the Virtual Machine containing the VR Appliance is automatically deployed, with a network interface attached to each Virtual Network.

For each Virtual Network, the following options can be defined:

- **Floating IP**. Only used in High Availability, explained bellow.
- **Force IPv4**. You can force the IP assigned to the network interface. When the VR is not configured in High Availability, this will be the IP requested for the Virtual Machine appliance.
- **Management interface**. If checked, this network interface will be a Virtual Router management interface. Traffic will not be forwarded to it.

Once ready, click the "create" button to finish. OpenNebula will create the Virtual Router and the Virtual Machines automatically.



CLI

Virtual Routers can be managed with the `onevrouter` command.

To create a new Virtual Router from the CLI, first you need to create a VR Template file, with the following attributes:

Then use the `onevrouter create` command:

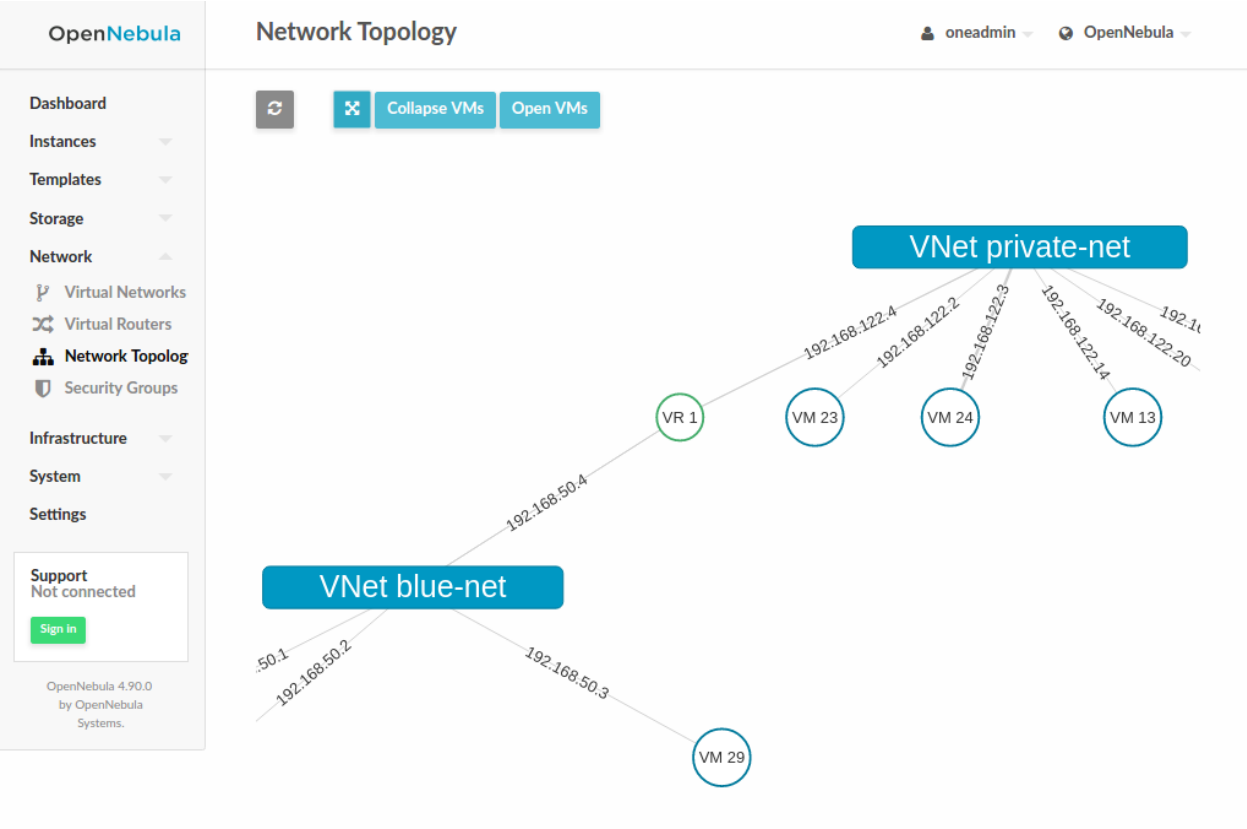
```
$ cat myvr.txt
NAME = my-vr
NIC = [
  NETWORK="blue-net",
  IP="192.168.30.5" ]
NIC = [
  NETWORK="red-net" ]

$ onevrouter create myvr.txt
ID: 1
```

At this point the Virtual Router resource is created, but it does not have any Virtual Machines. A second step is needed to create one (or more, if High Availability is used):

```
$ onevrouter instantiate <vrouterid> <templateid>
```

3.3.3 Managing Virtual Routers

Using the Virtual Routers tab in Sunstone, or the `onevrouter show` command, you can retrieve the generic resource information such as owner and group, the list of Virtual Networks interconnected by this router, and the Virtual Machines that are actually providing the routing.



The Virtual Networks connected to the VR machines can be modified with the attach/detach actions.

In Sunstone the actions can be found in the Virtual Router's main information panel, in the networks table. The options to add a new Virtual Network are the same that were explained for the creation wizard, see previous section.

The `onevrouter nic-attach` command takes a file containing a single NIC attribute. Alternatively, you can provide the new virtual network settings with command options, see `onevrouter nic-attach -h` for more information.

After a NIC is attached or detached, the Virtual Machine appliances are automatically reconfigured to start routing to the new interface. No other action, like a reboot, is required.

Managing Virtual Router VMs

The Virtual Machines that are associated to a Virtual Router have a limited set of actions. They can be terminated and new Virtual Machines can be added to an existing Virtual Router.

All the Virtual Machines associated with a Virtual Router are terminated automatically when the Virtual Router is deleted. Each VM can however be terminated individually at any time.

To create new VMs use the `onevrouter instantiate` command, or the "Instantiate VMs" dialog in Sunstone.

3.3.4 High Availability

More than one Virtual Machines can be associated to a Virtual Router in order to implement a high availability scenario. In this case, OpenNebula will also assign a floating IP to the group of Virtual Machines, that will coordinate to manage the traffic directed to that IP.

To enable a high availability scenario, you need to choose 2 or more number of instances when the Virtual Router is created in Sunstone. In the CLI, the number of VM instances is given with the `-m` option

```
$ onevrouter instantiate -h
[...]
-m, --multiple x          Instance multiple VMs
```

In this scenario, the following Virtual Router options became relevant:

- **Keepalived ID**: Optional. Sets keepalived configuration parameter `virtual_router_id`.
- **Keepalived password**: Optional. Sets keepalived configuration parameter `authentication/auth_pass`.

And for each Virtual Network Interface:

- **Floating IP**. Check it to enable the floating IP.
- **Force IPv4**. Optional. With the floating IP option selected, this field requests a fixed IP for that floating IP, not the individual VM IPs.

The floating IP assignment is managed in a similar way to normal VM IPs. If you open the information of the Virtual Network, it will contain a lease assigned to the Virtual Router (not a VM). Besides the floating IP, each VM will get their own individual IP.

Other Virtual Machines in the network will use the floating IP to contact the Virtual Router VMs. At any given time, only one VM is using that floating IP address. If the active VM crashes, the other VMs will coordinate to assign the floating IP to a new Virtual Router VM.

3.3.5 Customization

You can provide two optional parameters in the context to configure the keepalived service started in the Virtual Router VM:

- `VROUTER_KEEPALIVED_PASSWORD`: Password used for the service to protect the service from packages of rogue machines. By default the service is configured without password.
- `VROUTER_KEEPALIVED_ID`: Number identifier of the service (0-255). This is useful when you have several virtual routers or other keepalived services in the same network. By default it is generated from the Virtual Router ID (`$vrouter_id & 255`) but you can specify it manually if needed.

These parameters can also be provided in the Virtual Router creation wizzard of sunstone.

3.4 Security Groups

Security Groups define firewall rules to be applied them to the Virtual Machines.

Note: By default, the *default* security group is applied to new VMs, which allows all OUTBOUND traffic and all INBOUND traffic. You **must** Modify the *default* security group to make it more restrictive, if you leave as is everything will be always allowed.

Warning: Security groups is not supported for OpenvSwitch and vCenter networks, and IPv6 addressing.

3.4.1 Defining a Security Group

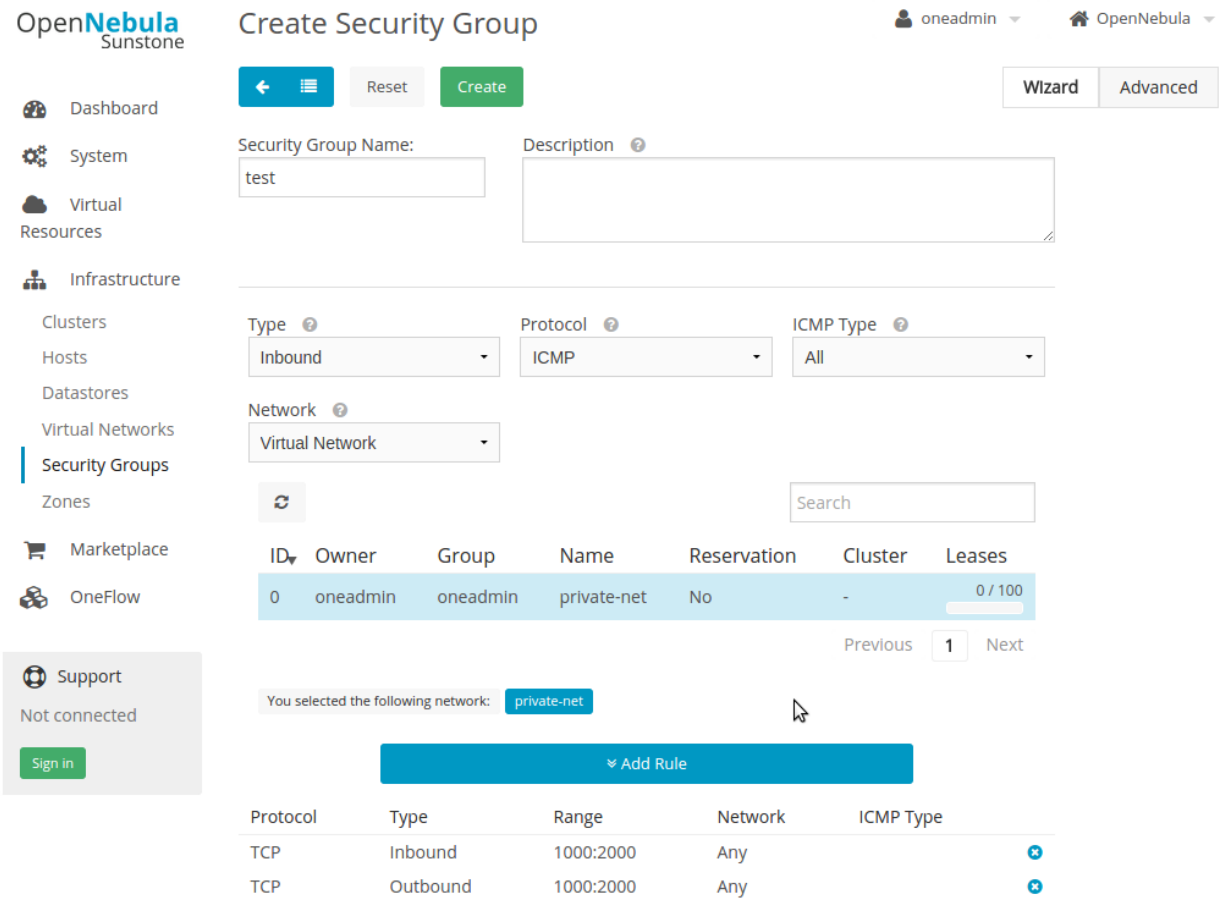
A Security Group is composed of several Rules. Each Rule is defined with the following attributes:

Attribute	Type	Meaning	Values
PROTOCOL	Mandatory	Defines the protocol of the rule	ALL, TCP, UDP, ICMP, IPSEC
RULE_TYPE	Mandatory	Defines the direction of the rule	INBOUND, OUTBOUND
IP	Optional	If the rule only applies to a specific net. This is the first **IP** of the consecutive set of **IPs**. Must be used with **SIZE**.	A valid IP
SIZE	Optional	If the rule only applies to a net. The number of total consecutive IPs of the network. Use always with **IP**.	An integer >= 1
RANGE	Optional	A Port Range to filter specific ports. Only works with **TCP** and **UDP**.	(iptables syntax) multiple ports or port ranges are separated using a comma, and a port range is specified using a colon. Example: `22,53,80:90,110,1024:65535`
ICMP_TYPE	Optional	Specific ICMP type of the rule. If a type has multiple codes, it includes all the codes within. This can only be used with **ICMP**. If omitted the rule will affect the whole **ICMP** protocol.	0,3,4,5,8,9,10,11,12,13,14,17,18

To create a Security Group, use the Sunstone web interface, or create a template file following this example:

```
$ cat ./sg.txt

NAME = test

RULE = [
    PROTOCOL = TCP,
    RULE_TYPE = inbound,
    RANGE = 1000:2000
]

RULE = [
    PROTOCOL= TCP,
    RULE_TYPE = outbound,
    RANGE = 1000:2000
]

RULE = [
    PROTOCOL = ICMP,
    RULE_TYPE = inbound,
    NETWORK_ID = 0
]

$ onesecgroup create ./sg.txt
ID: 102
```

```
.. note:: This guide focuses on the CLI command ``onesecgroup``, but you can also␣
↪manage Security Groups using :ref:`Sunstone <sunstone>`, mainly through the␣
↪Security Group tab in a user friendly way.
```

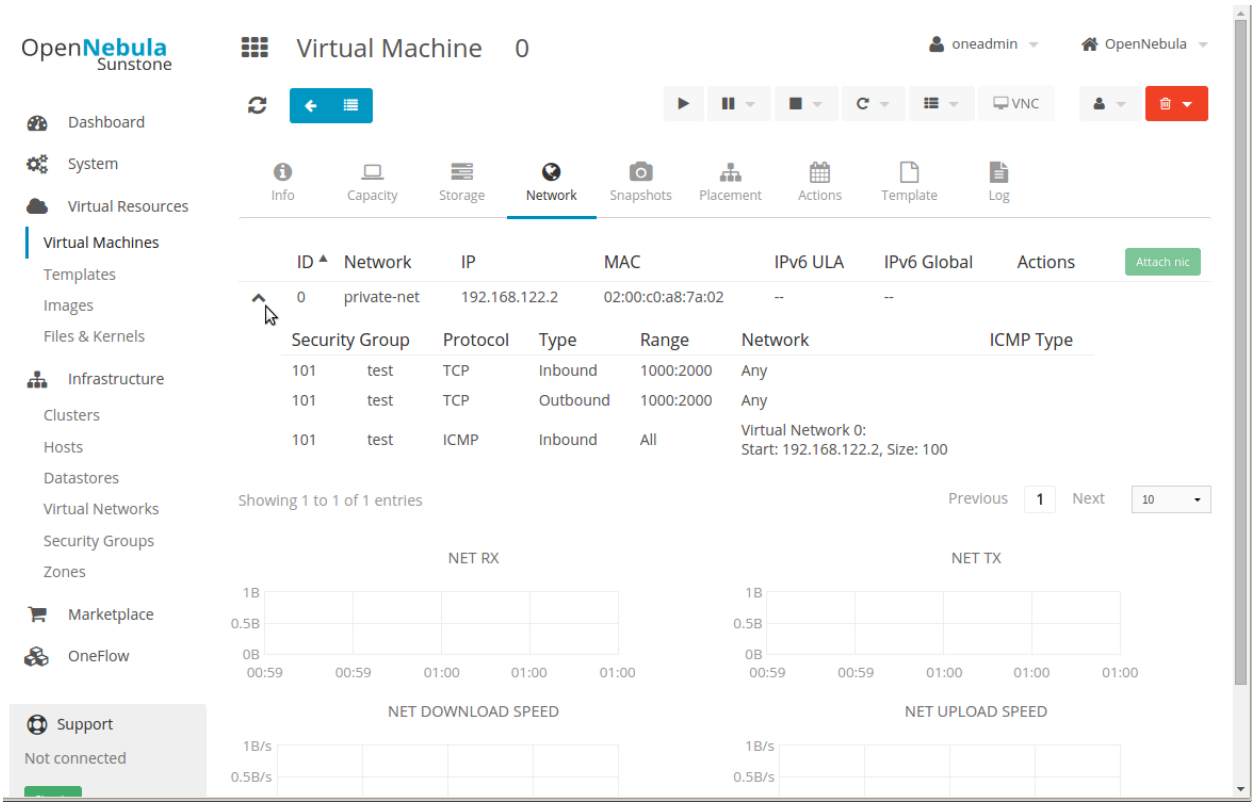


3.4.2 Using a Security Group

To apply a Security Group to your Virtual Machines, you can assign them to the Virtual Networks. Either use the Sunstone wizard, or set the SECURITY_GROUPS attribute:

```
$ onevnet update 0

SECURITY_GROUPS = "100, 102, 110"
```

When a Virtual Machine is instantiated, the rules are copied to the VM resource and can be seen in the CLI and Sunstone.



Advanced Usage

To accommodate more complex scenarios, you can also set Security Groups to each Address Range of a Virtual Network.

```
$ onevnet updatear 0 1

SECURITY_GROUPS = "100, 102, 110"
```

Moreover, each Virtual Machine Template NIC can define a list of Security Groups:

```
NIC = [
  NETWORK = "private-net",
  NETWORK_UNAME = "oneadmin",
  SECURITY_GROUPS = "103, 125"
]
```

If the Address Range or the Template NIC define SECURITY_GROUPS, the IDs do not overwrite the ones defined in the Virtual Network. All the Security Group IDs are combined, and applied to the Virtual Machine instance.

3.4.3 The Default Security Group

Warning: If you don't modify the default Security Group you will not be able to filter any connections.

There is a default Security Group with ID 0. This Security Group, unless modified, will allow all traffic, both outbound and inbound. You **must** modify this *default* Security Group if you want to restrict connections. Consider this Security

Group to be the bare minimum for all VMs. For example, it may make sense to define it as TCP port 22 inbound for SSH, and port 80 and 443 outbout to be able to install packages.

This special Security Group is added to all the Virtual Networks when they are created, but you can remove it later updating the network's properties.

3.4.4 Security Group Update

Security Groups can be updated to edit or add new rules. These changes are propagated to all VMs in the security group, so it may take some time till the changes are applied. The particular status of a VM can be checked in the security group properties, where outdated and up-to-date VMs are listed.

If the update process needs to be reset, i.e. apply again the rules, you can use the `onesecgroup commit` command.

CHAPTER

FOUR

VIRTUAL MACHINE MANAGEMENT

4.1 Overview

This chapter contains documentation on how to create and manage Virtual Machine *templates*, *instances*, and *Images* (VM disks).

4.1.1 How Should I Read This Chapter

Before reading this chapter, you should have already installed your Frontend, the KVM Hosts or vCenter node and have an OpenNebula cloud up and running with at least one virtualization node.

For vCenter based infrastructures read first the *vCenter Specifics* Section.

4.1.2 Hypervisor Compatibility

Section	Compatibility
Virtual Machine Images	This Section applies to both KVM and vCenter.
Virtual Machine Templates	This Section applies to both KVM and vCenter.
Virtual Machine Instances	This Section applies to both KVM and vCenter.
vCenter Specifics	This Section applies to vCenter.

4.2 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 documentation.

4.2.1 Image Types

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

For Virtual Machine disks:

- `OS`: An bootable disk Image. Every *VM template* must define one DISK referring to an Image of this type.
- `CDROM`: These Images are read-only data. Only one Image of this type can be used in each *VM template*.

- `DATABLOCK`: A datablock Image is a storage for data. These Images can be created from previous existing data, or as an empty drive.

"File" types. Images of these types cannot be used as VM disks, and are listed in Sunstone under the Files tab:

- `KERNEL`: A plain file to be used as kernel (VM attribute OS/KERNEL_DS).
- `RAMDISK`: A plain file to be used as ramdisk (VM attribute OS/INITRD_DS).
- `CONTEXT`: A plain file to be included in the context CD-ROM (VM attribute CONTEXT/FILES_DS).

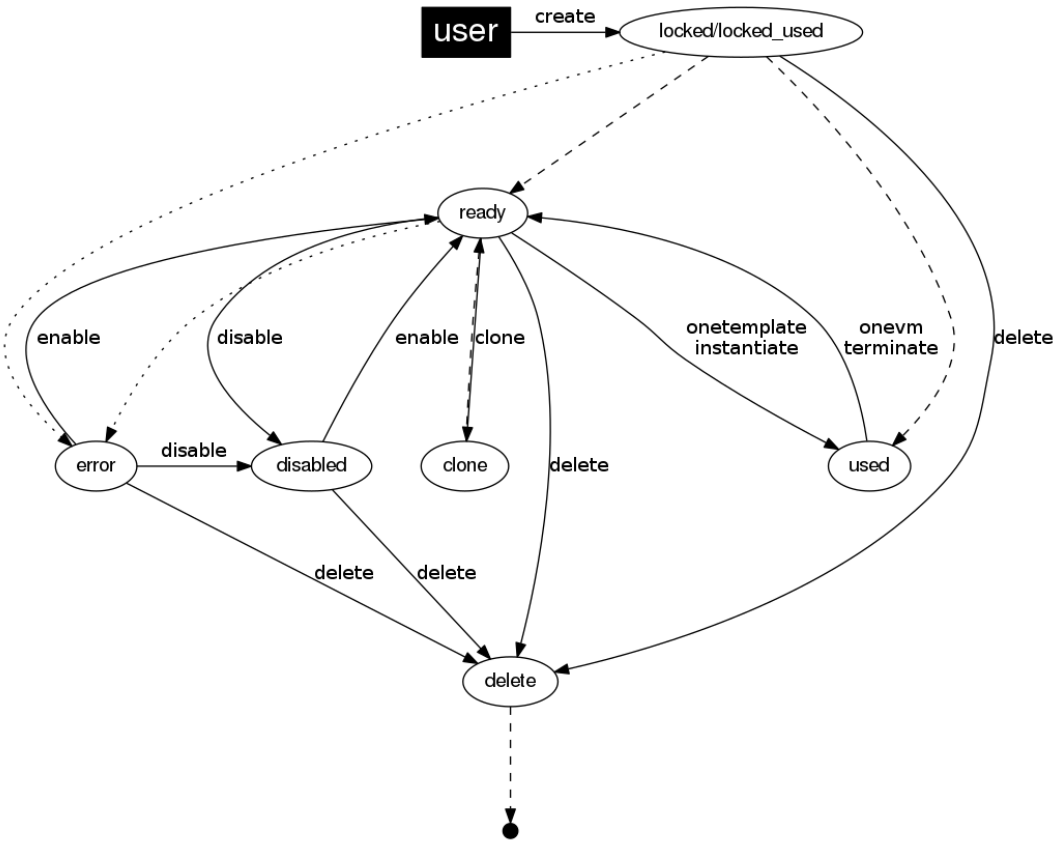
Note: KERNEL, RAMDISK and CONTEXT file Images can be registered only in File Datastores.

Note: Some of the operations described below do not apply to KERNEL, RAMDISK and CONTEXT Images, in particular: clone and persistent.

4.2.2 Image Life-cycle

Short state	State	Meaning
`lock`	`LOCKED`	The Image file is being copied or created in the Datastore.
`lock`	`LOCKED_USED`	Image file is being copied or created in the Datastore, with VMs waiting for the operation to finish.
`lock`	`LOCKED_USED_PERS`	Same as `LOCKED_USED`, for Persistent Images
`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.
`clon`	`CLONE`	The Image is being cloned.

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



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

4.2.3 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, selecting the Images tab. By default this tab is available in the `admin` view, but not in the `cloud` or `groupadmin` views.

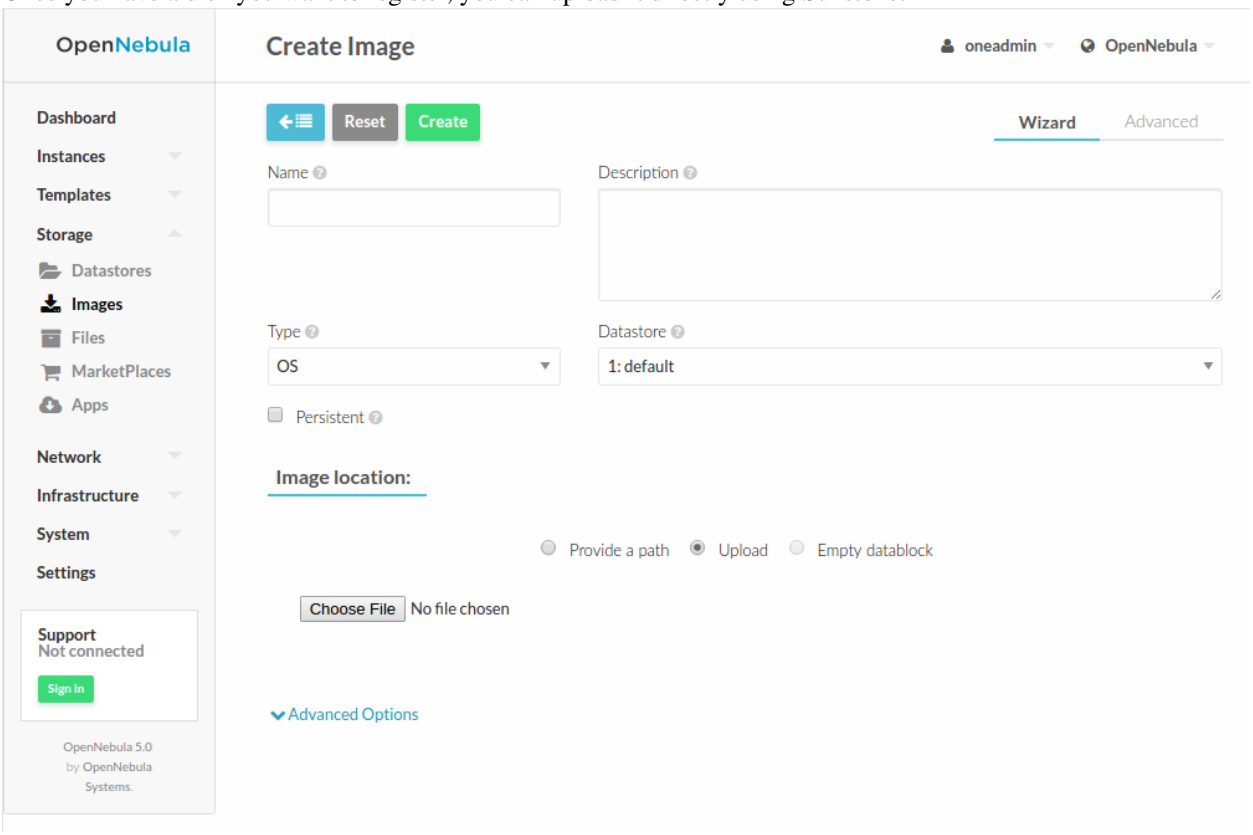
Create Images

The three types of Images can be created from an existing file, but for **datablock** Images you can specify a size 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 *VM contextualization here*.

Once you have a disk you want to register, you can upload it directly using Sunstone:



To register it from the command line 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 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 a clean installation you will only have one datastores with type `img`, default.

```
$ onedatastore list
  ID NAME                SIZE AVAIL CLUSTERS     IMAGES TYPE DS      TM      STAT
   0 system            145.2G 56%   0                0 sys  -       shared  on
   1 default           145.2G 56%   0                3 img  fs      shared  on
   2 files             145.2G 56%   0                0 fil  fs      ssh     on
```

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, DATABLOCK, KERNEL, RAMDISK, CONTEXT
`--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 (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

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 desktop for students."
```

Note: You can use **gz** compressed image files when registering them in OpenNebula.

Limitations when Uploading Images from Sunstone

Image file upload to the server via the client browser is possible. The process is as follow:

- Step 1: The client uploads the whole image file 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 image file 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 successful (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 re-initiate the loading progress bar).

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. Note that persistent Images with snapshots cannot be cloned. In order to do so, the user would need to flatten it first, see the *snapshots* section for more information.

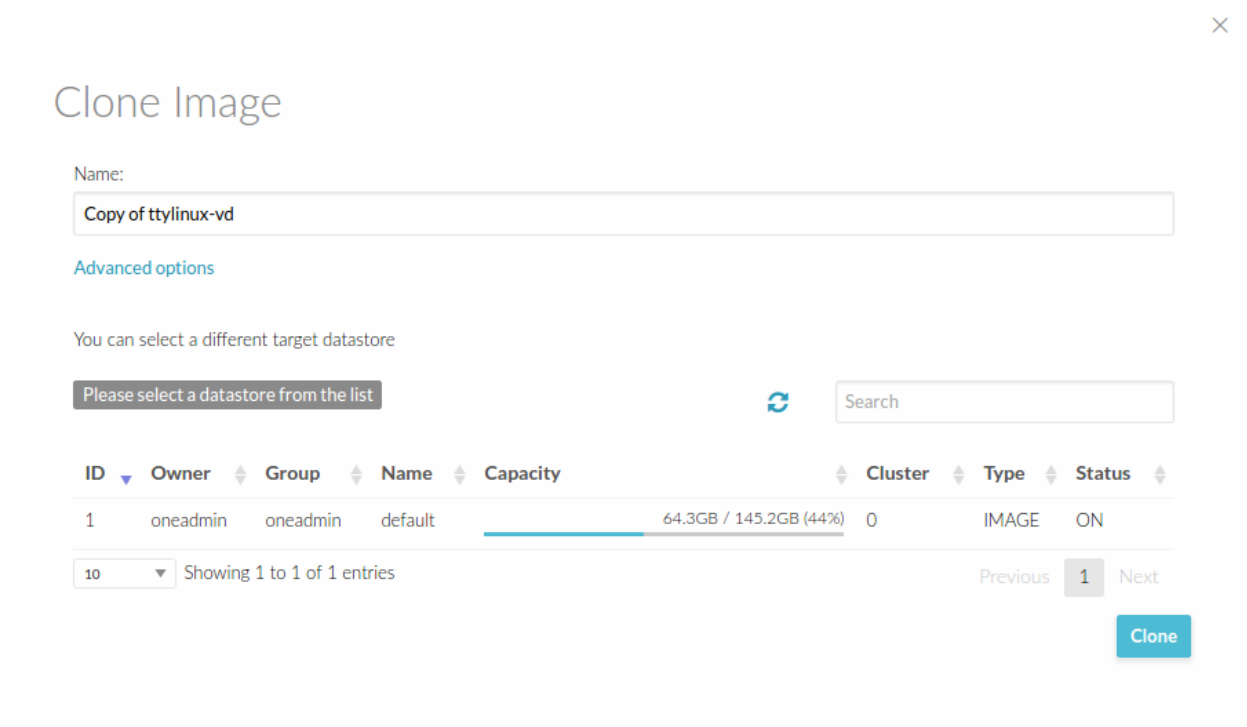
To clone an Image, execute

```
$ oneimage clone Ubuntu new_image
```

You can optionally clone the Image to a different Datastore. The new Datastore must be compatible with the current one, i.e. have the same DS_MAD drivers.

```
$ oneimage clone Ubuntu new_image --datastore new_img_ds
```

The Sunstone Images tab also contains a dialog for the clone operation:



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 oneadmin   oneadmin   ttylinux-vd     default       200M OS    No used    8
   1 johndoe    users      my-ubuntu-disk- default       200M OS   Yes used    1
   2 alice      testgroup  customized-ubun default       200M OS   Yes used    1
```

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

Sharing 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--
```

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.

```
$ 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
```

Note that persistent Images with snapshots cannot be made non-persistent. In order to do so, the user would need to flatten it first, see the *snapshots* section for more information.

Managing Snapshots in Persistent Images

Persistent Images can have associated snapshots if the user *created them* during the life-cycle of VM that used the persistent Image. The following are operations that allow the user to manage these snapshots directly:

- `oneimage snapshot-revert <image_id> <snapshot_id>`: The active state of the Image is overwritten by the specified snapshot. Note that this operation discards any unsaved data of the disk state.

- `oneimage snapshot-delete <image_id> <snapshot_id>`: Deletes a snapshot. This operation is only allowed if the snapshot is not the active snapshot and if it has no children.
- `oneimage snapshot-flatten <image_id> <snapshot_id>`: This operation effectively converts the Image to an Image without snapshots. The saved disk state of the Image is the state of the specified snapshot. It's an operation similar to running `snapshot-revert` and then deleting all the snapshots.

Images with snapshots **cannot** be cloned or made non-persistent. To run either of these operations the user would need to flatten the Image first.

4.2.4 How to Use Images in Virtual Machines

This is 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 Datastore.

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   = 7 ]

DISK = [ IMAGE       = "Ubuntu",
         IMAGE_UNAME = "oneadmin" ]

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"  ]
```

Save Changes

Once the VM is deployed you can and changes are made to its disk, you can save those changes in two different ways:

- **Disk snapshots**, a snapshot of the disk state is saved, you can later revert to this saved state.
- **Disk save_as**, the disk is copied to a new Image in the datastore. A new virtual machine can be started from it. The disk must be in a consistent state during the save_as operation (e.g. by unmounting the disk from the VM).

A detailed description of this process is *described in section Vritual Machine Instances*

4.2.5 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 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]",
]
```

4.3 Managing Virtual Machine Templates

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

The VM Template Pool 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.

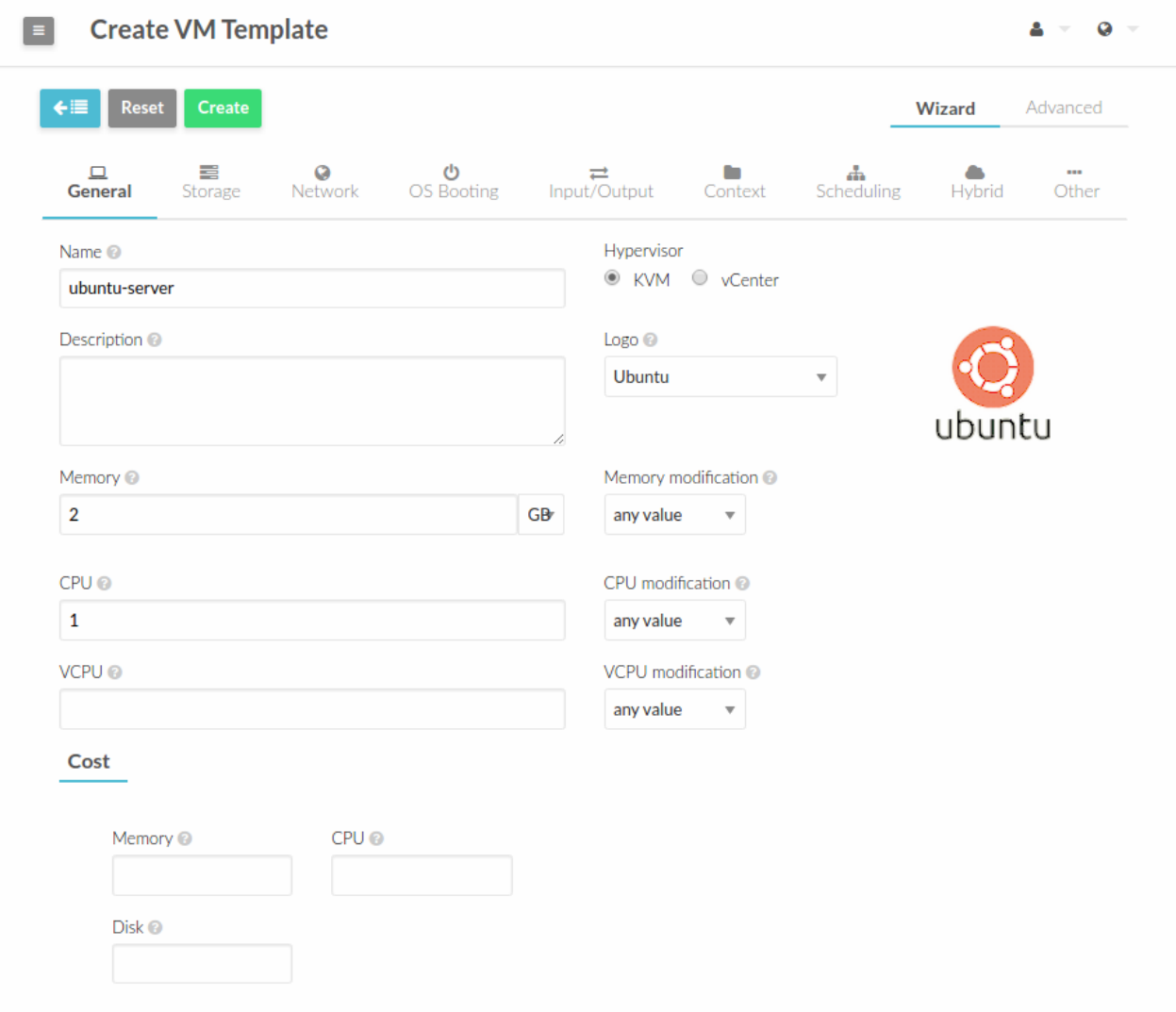
4.3.1 Defining a VM

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
- Optional attributes like VNC graphics, the booting order, context information, etc.

Virtual Machines are defined in an OpenNebula Template. Templates are stored in the system to easily browse and instantiate VMs from them.

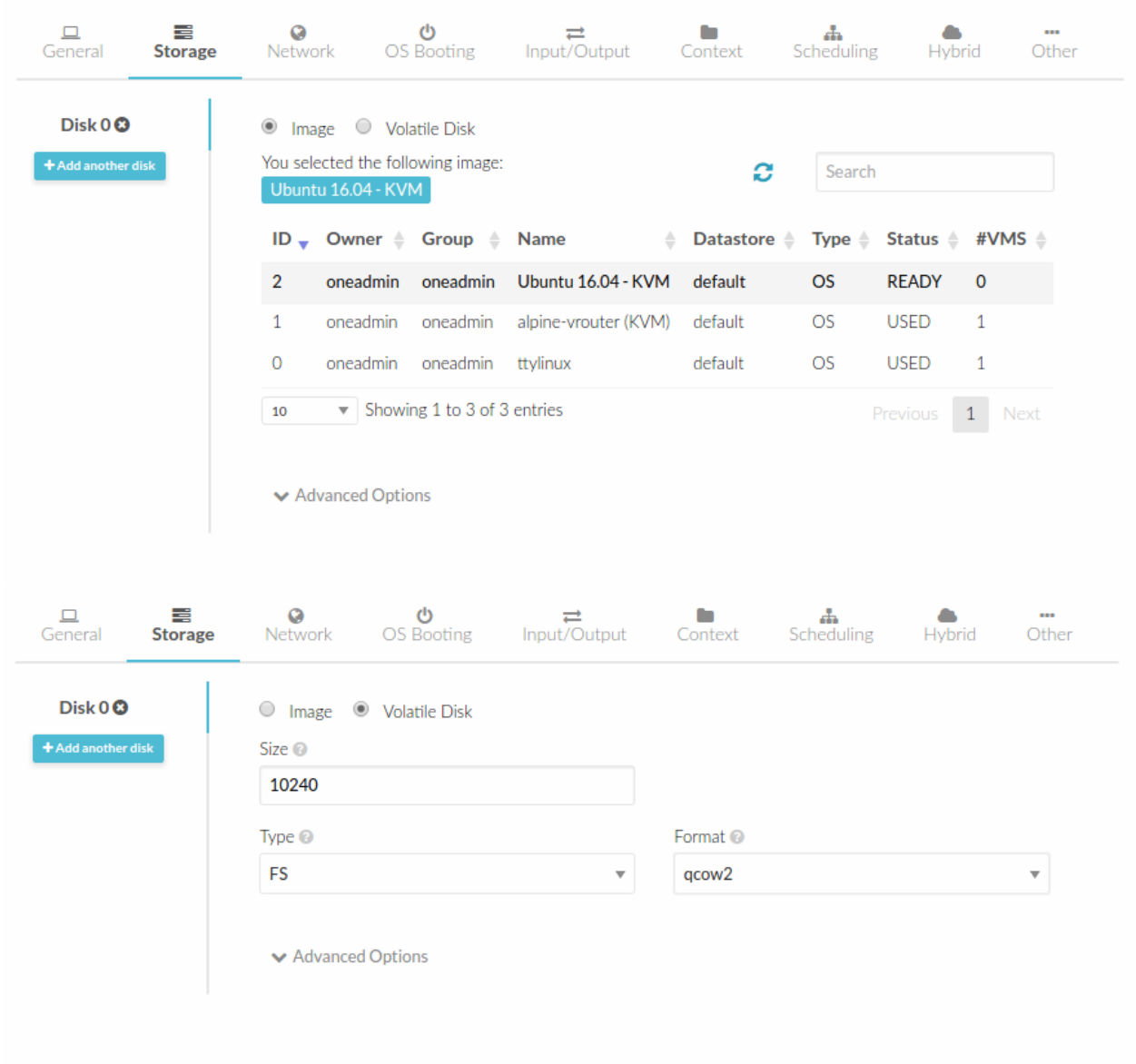
Capacity & Name



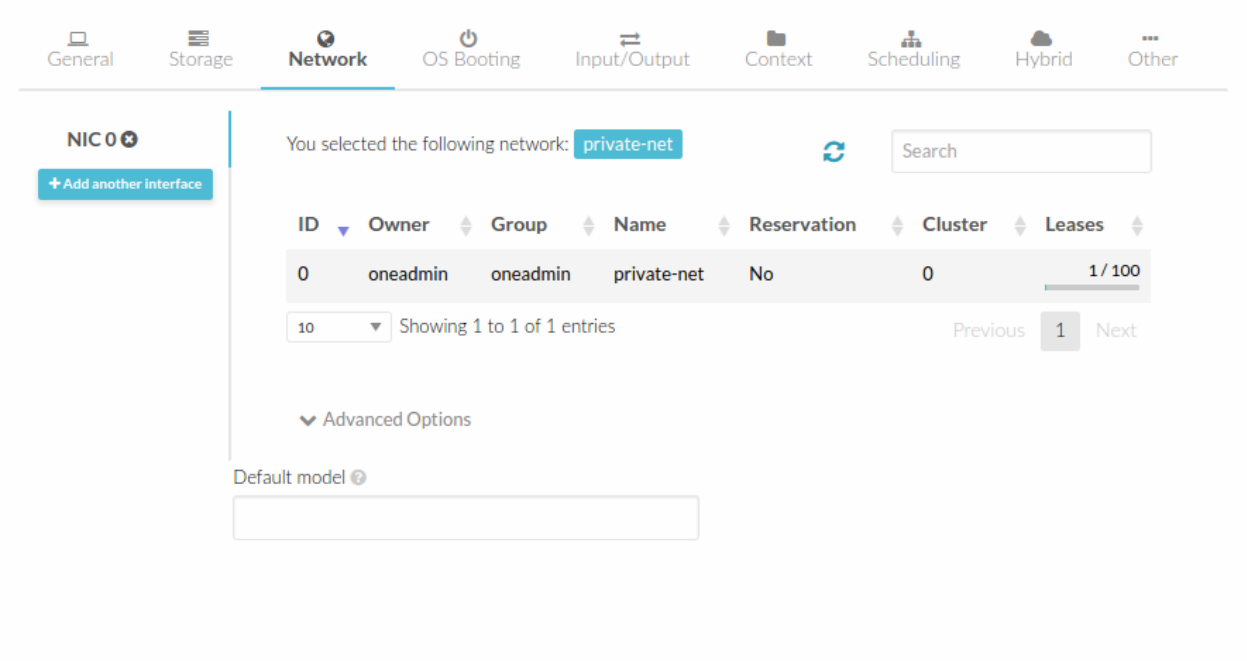
Disks

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 terminated.
- **Use a non-persistent Image**: a copy of the source Image is used, changes made to the VM disk will be lost.
- **Volatile**: disks are created on the fly on the target host. After the VM is terminated the disk is disposed.


General
Storage
Network
OS Booting
Input/Output
Context
Scheduling
Hybrid
Other
Disk 0
+ Add another disk
Image
Volatile Disk
You selected the following image:
Ubuntu 16.04 - KVM
Search
ID
Owner
Group
Name
Datastore
Type
Status
#VMS
2 oneadmin oneadmin Ubuntu 16.04 - KVM default OS READY 0
1 oneadmin oneadmin alpine-vrouter (KVM) default OS USED 1
0 oneadmin oneadmin ttylinux default OS USED 1
10
Showing 1 to 3 of 3 entries
Previous
1
Next
Advanced Options
General
Storage
Network
OS Booting
Input/Output
Context
Scheduling
Hybrid
Other
Disk 0
+ Add another disk
Image
Volatile Disk
Size
10240
Type
FS
Format
qcow2
Advanced Options

Network Interfaces



Example

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"]
```

Note: Check the *VM definition file for a complete reference*

Simple templates can be also created using the command line instead of creating a template file. For example, 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
```

For a complete reference of all the available options for `onetemplate create`, go to the *CLI reference*, or run `onetemplate create -h`.

Note: OpenNebula Templates are designed to be hypervisor-agnostic, but there are additional attributes that are sup-

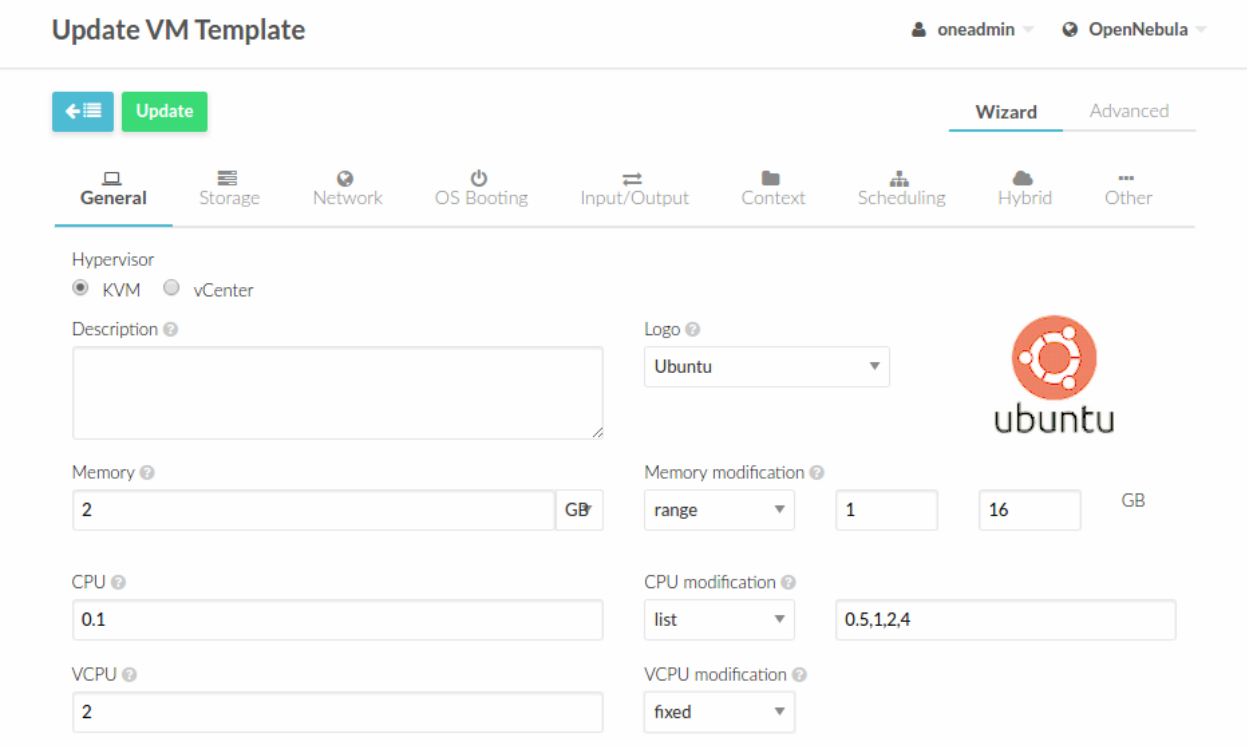
ported for each hypervisor. Check the KVM configuration and vCenter configuration for more details.

4.3.2 Preparing Templates for End-Users

Besides the basic VM definition attributes, you can setup extra options in your VM Template.

Customizable Capacity

The capacity attributes (CPU, MEMORY, VCPU) can be modified each time a VM Template is instantiated. The Template owner can decide *if* and *how* each attribute can be customized.



The modification options available in the drop-down are:

- **fixed**: The value cannot be modified.
- **any value**: The value can be changed to any number by the user instantiating the Template.
- **range**: Users will be offered a range slider between the given minimum and maximum values.
- **list**: Users will be offered a drop-down menu to select one of the given options.

If you are using a template file instead of Sunstone, the modification is defined with user input attributes (*see below*). The absence of user input is an implicit "any value". For example:

```
CPU = "1"
MEMORY = "2048"
VCPU = "2"
USER_INPUTS = [
  CPU = "M|list||0.5,1,2,4|1",
  MEMORY = "M|range||512..8192|2048" ]
```

Note: Use float types for CPU, and integer types for MEMORY and VCPU. More information in *the Template reference documentation*.

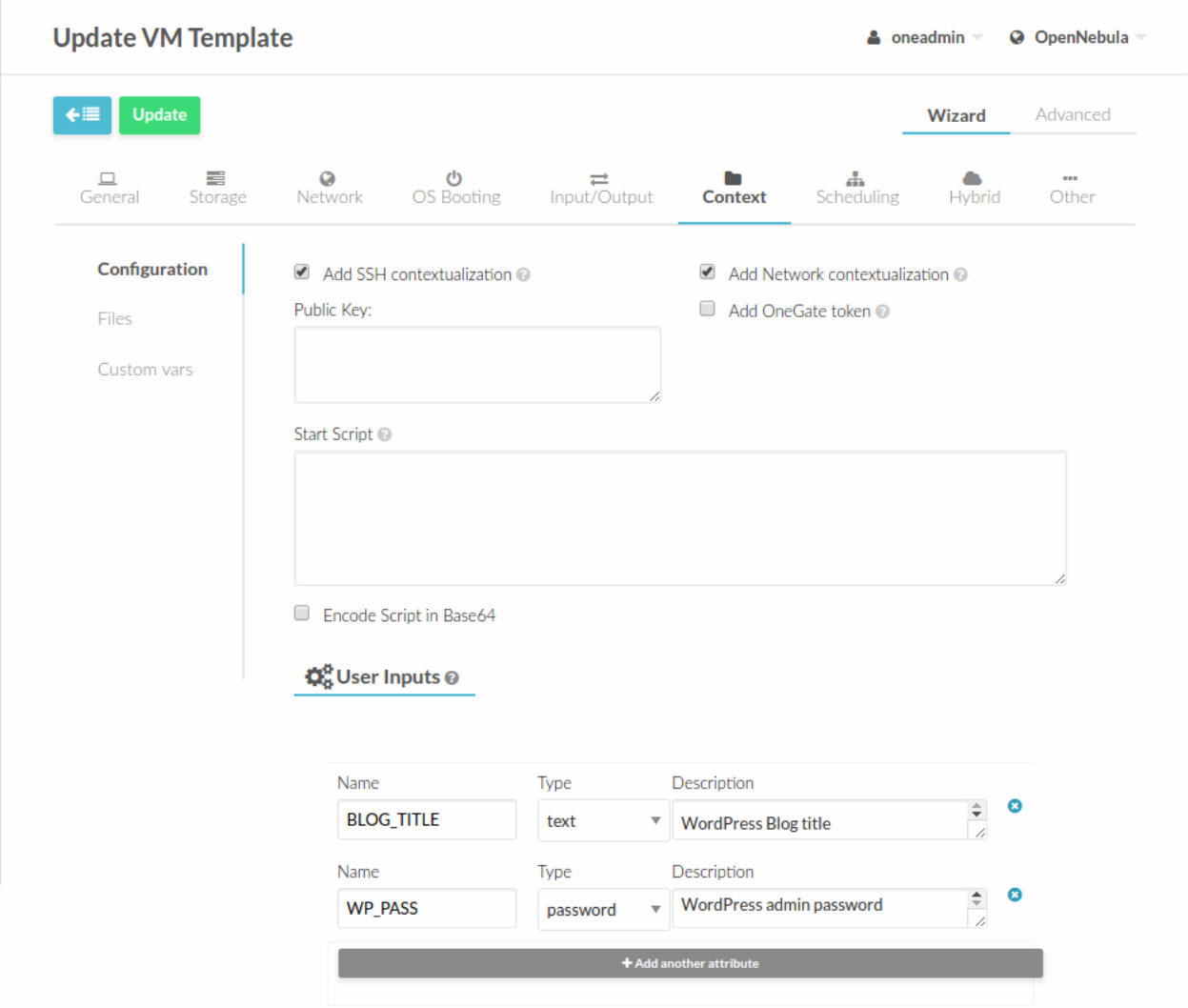
Note: This capacity customization can be forced to be disabled for any Template in the cloud view. Read more in the Cloud View Customization documentation.

Ask for User Inputs

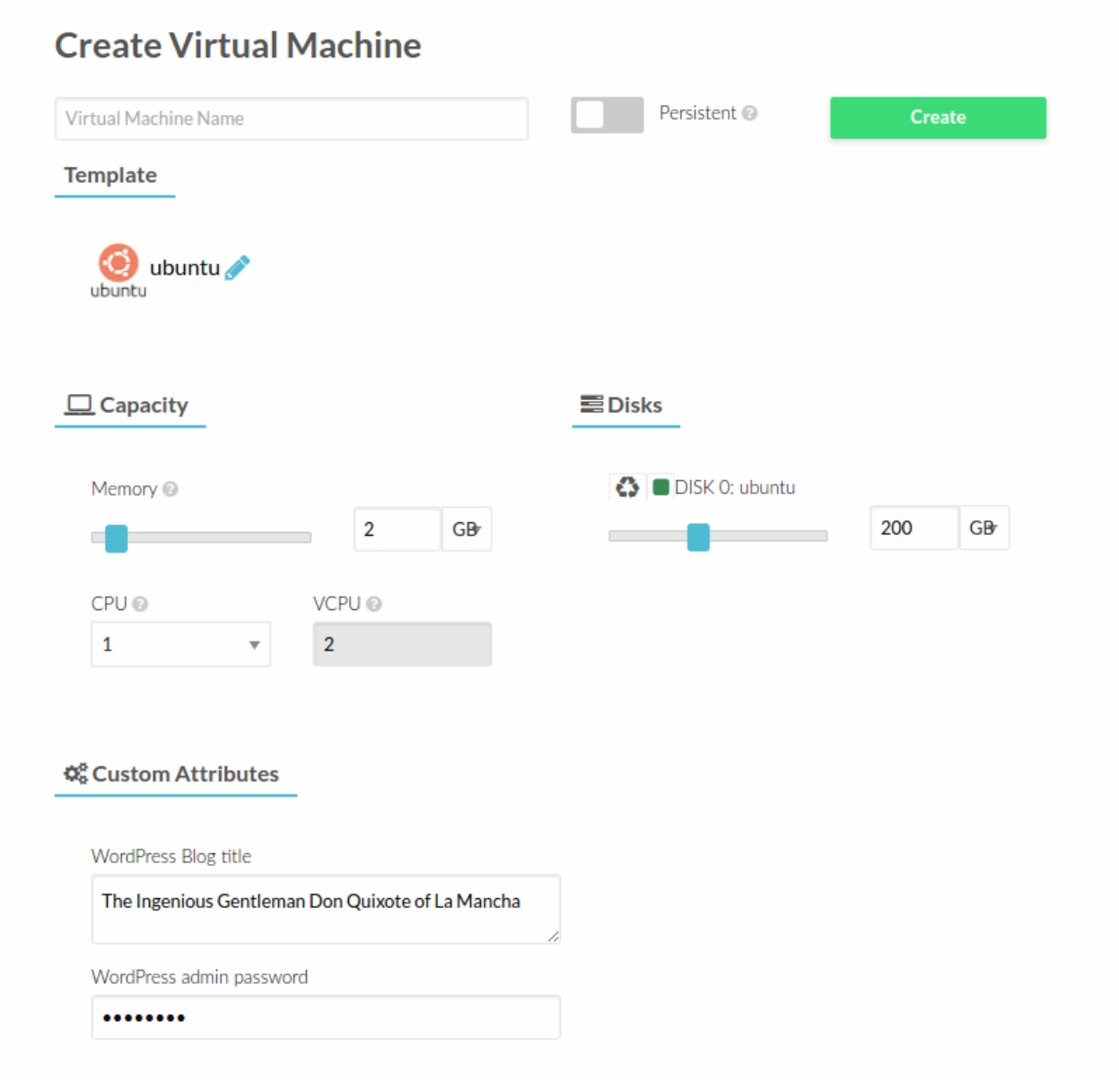
The User Inputs functionality provides the Template creator the possibility to dynamically ask the user instantiating the Template dynamic values that must be defined.

A user input can be one of the following types:

- **text**: any text value
- **password**: any text value. The interface will block the input visually, but the value will be stored as plain text.
- **text64**: will be encoded in base64 before the value is passed to the VM.
- **number**: any integer number.
- **number-float**: any number.
- **range**: any integer number within the defined min..max range.
- **range-float**: any number within the defined min..max range
- **list**: the user will select from a pre-defined list of values



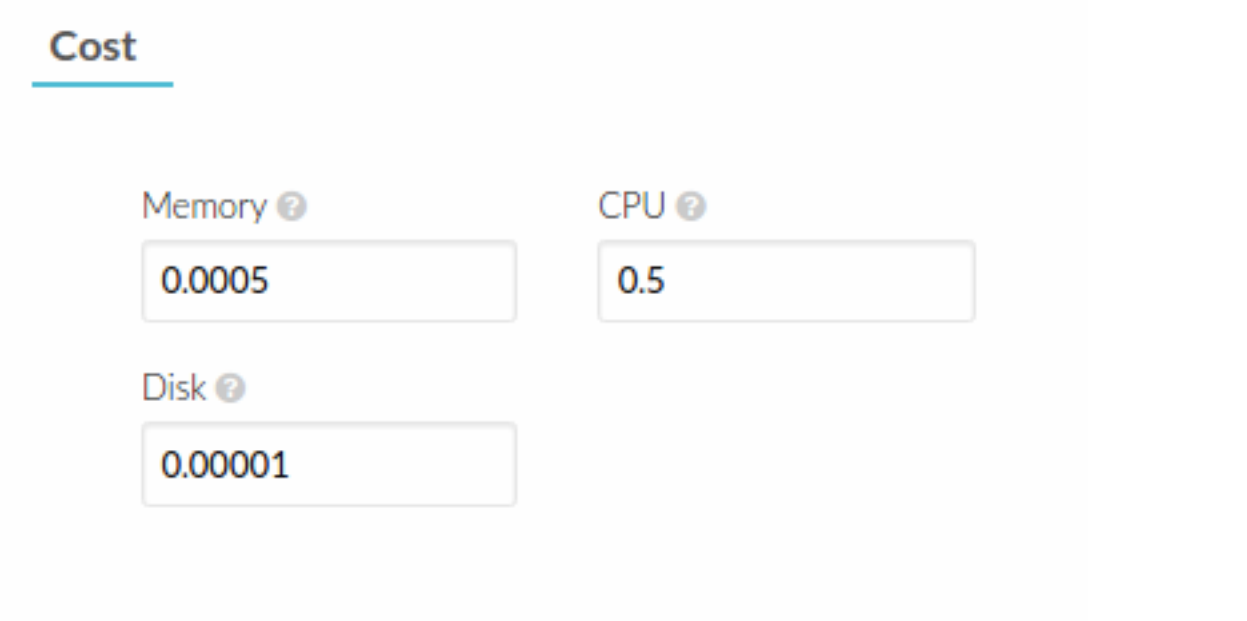
These inputs will be presented to the user when the Template is instantiated. The VM guest needs to be *contextualized* to make use of the values provided by the user.



Note: If a VM Template with user inputs is used by a Service Template Role, the user will be also asked for these inputs when the Service is created.

Set a Cost

Each VM Template can have a cost per hour. This cost is set by CPU and MEMORY MB, to allow users to change the capacity and see the cost updated accordingly. VMs with a cost will appear in the *showback reports*.

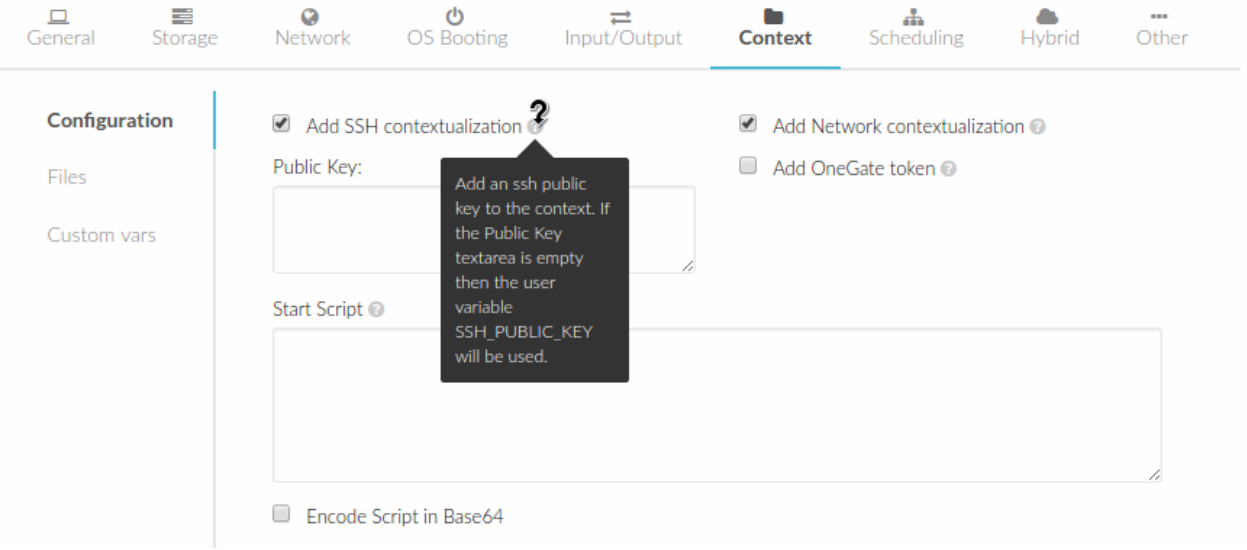


See the *template file syntax here*.

Enable End User Features

There are a few features of the Cloud View that will work if you configure the Template to make use of them:

- Users will see the Template logo and description, something that is not so visible in the normal admin view.
- The Cloud View gives access to the VM's VNC, but only if it is configured in the Template.
- End users can upload their public ssh key. This requires the VM guest to be *contextualized*, and the Template must have the ssh contextualization enabled.



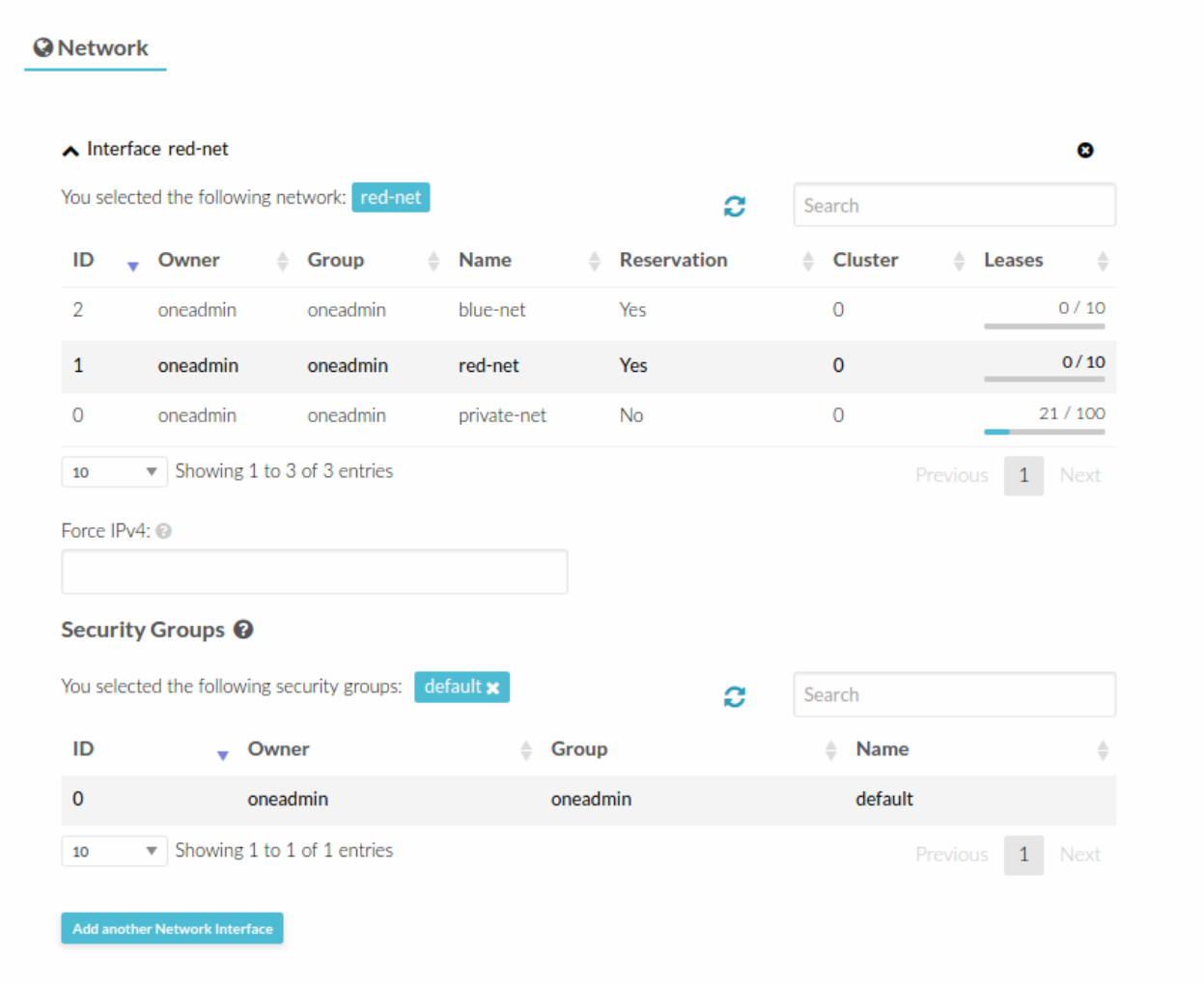
Make the Images Non-Persistent

If a Template is meant to be consumed by end-users, its Images should not be persistent. A *persistent Image* can only be used by one VM simultaneously, and the next user will find the changes made by the previous user.

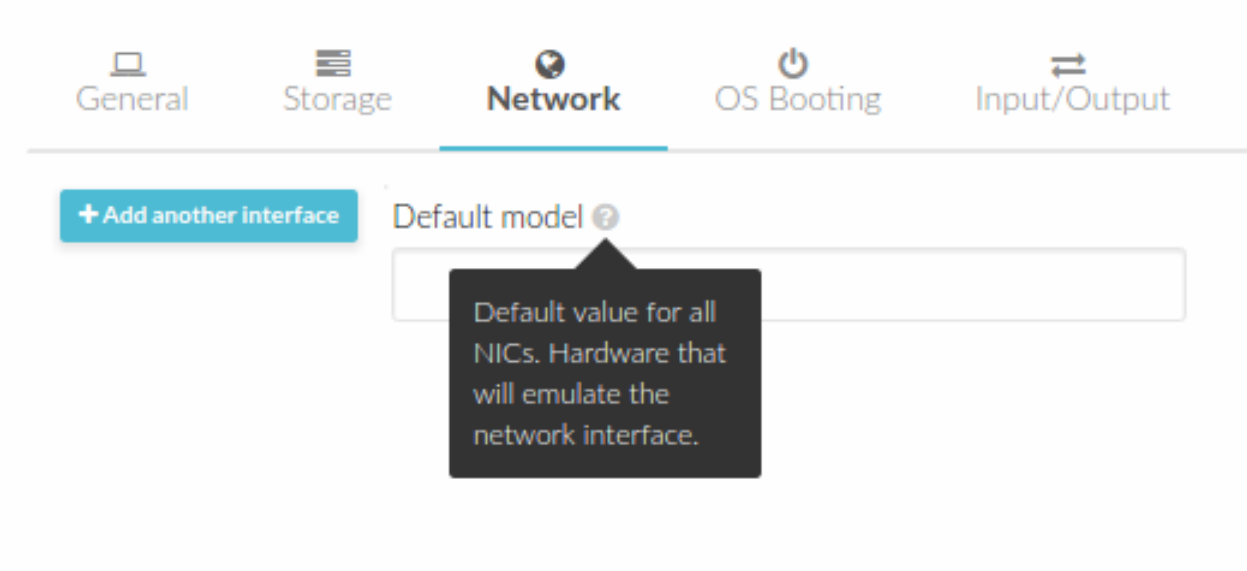
If the users need persistent storage, they can use the *"instantiate to persistent" functionality*.

Prepare the Network Interfaces

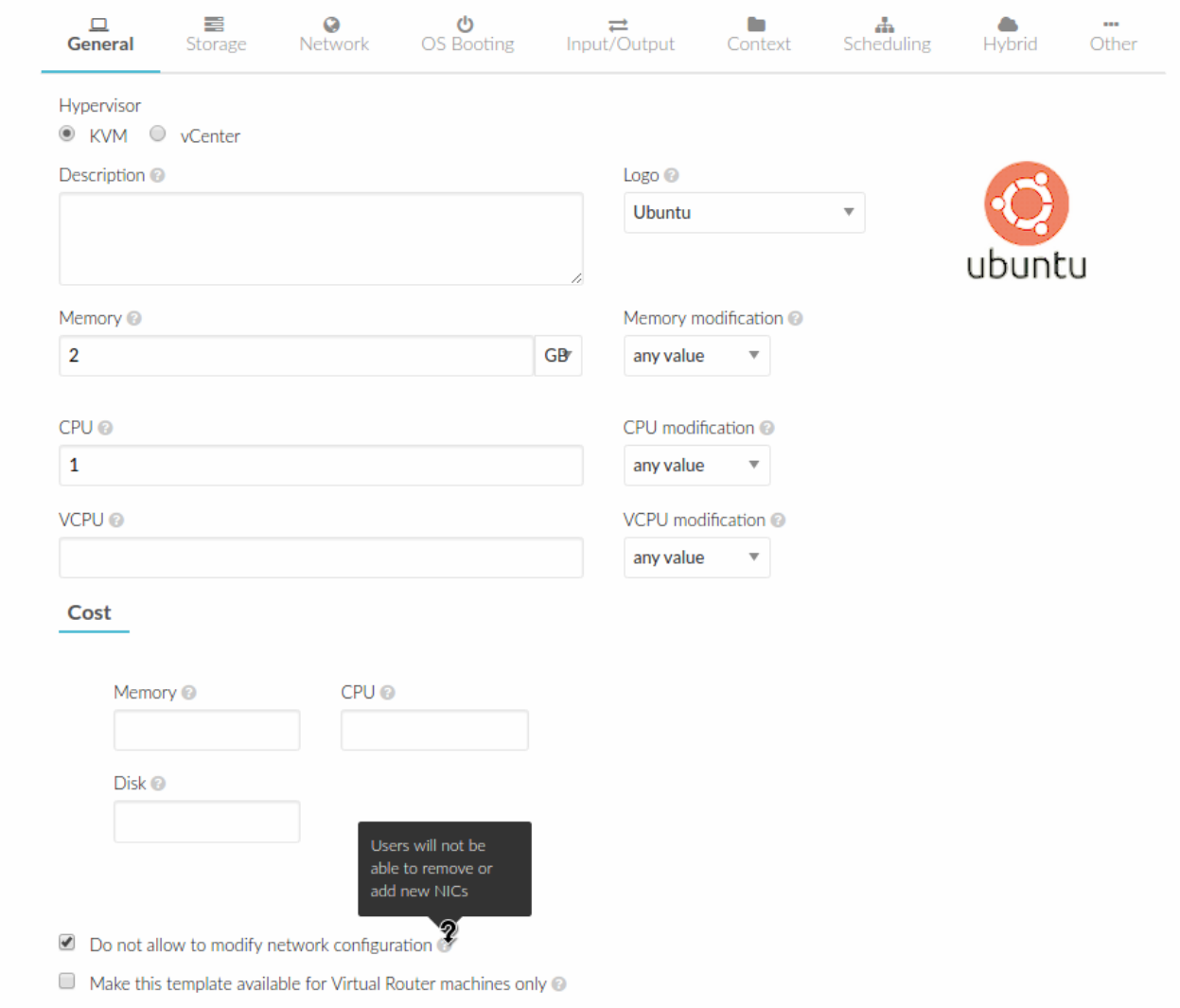
End-users can select the VM network interfaces when launching new VMs. You can create templates without any NIC, or set the default ones. If the template contains any NIC, users will still be able to remove them and select new ones.



Because users will add network interfaces, you need to define a default NIC model in case the VM guest needs a specific one (e.g. virtio for KVM). This can be done with the *NIC_DEFAULT* attribute, or through the Template wizard. Alternatively, you could change the default value for all VMs in the driver configuration file (see the KVM one for example).



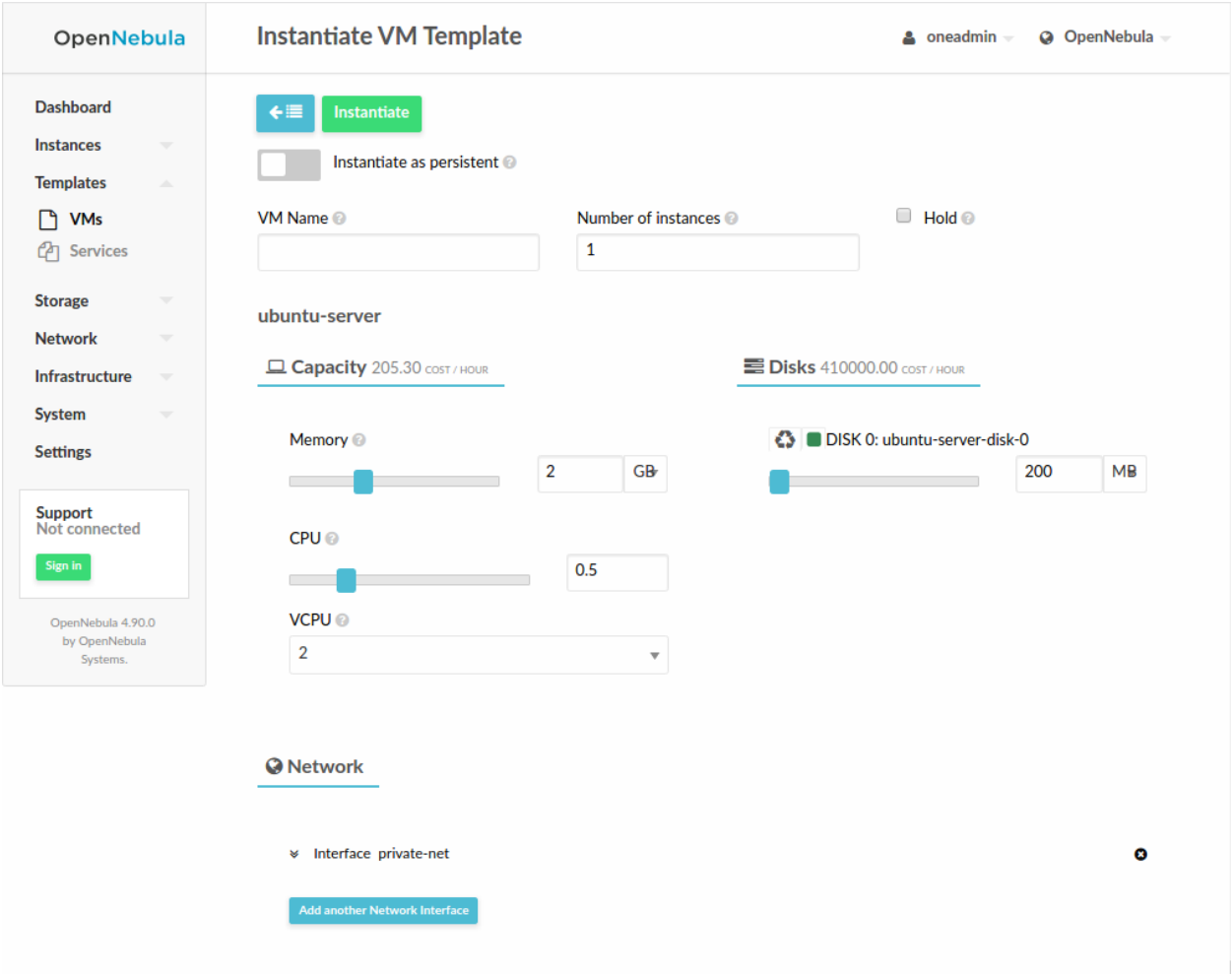
This networking customization can be disabled for each Template. The users instantiating the Template will not be able to add, remove, or customize set NICs set by the Template owner.



Note: This networking customization can be forced to be disabled for any Template in the cloud view. Read more in the Cloud View Customization documentation.

4.3.3 Instantiating Templates

From Sunstone:



From the CLI: the `onetemplate instantiate` command accepts a Template ID or name, and creates a VM instance from the given template. You can create more than one instance simultaneously with the `--multiple num_of_instances` option.

```
$ 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
```

Merge Use Case

The template merge functionality, combined with the restricted attributes, 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 16.04"
```

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 16.04"
```

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

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

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 terminate` to shutdown and delete 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.

4.3.4 Managing Templates

Users can manage the VM Templates 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.

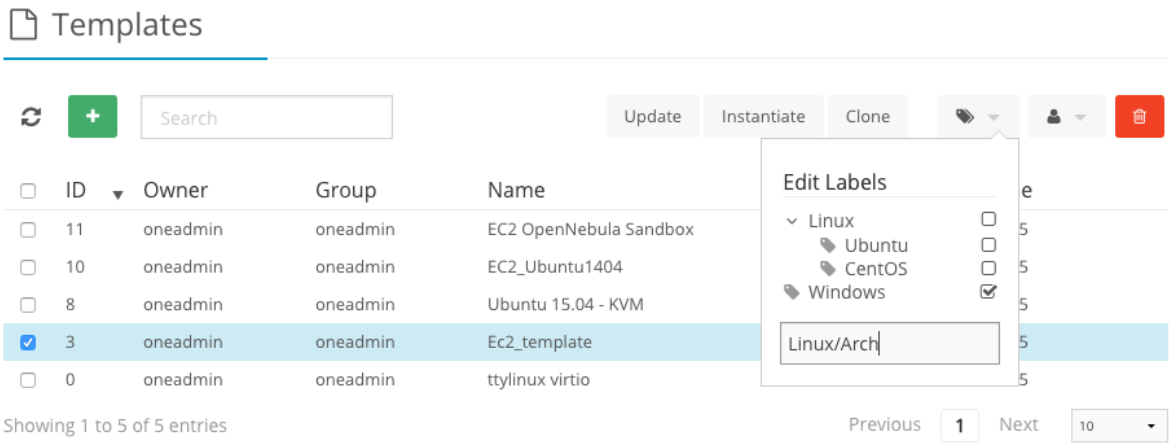
Listing Available Templates

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

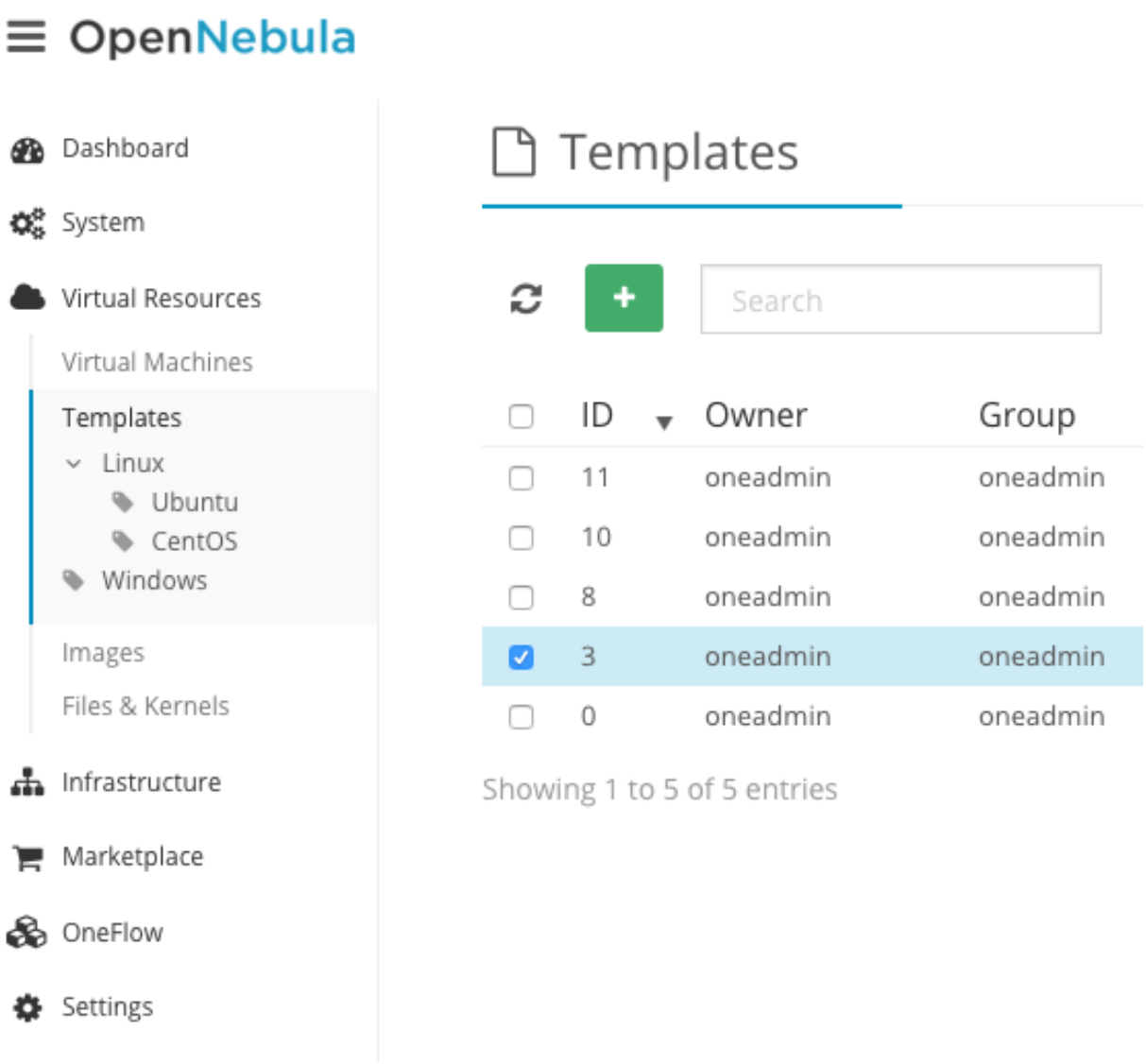
```
$ onetemplate list a
  ID USER     GROUP    NAME                          REGTIME
   0 oneadmin oneadmin template-0            09/27 09:37:00
   1 oneuser  users    template-1            09/27 09:37:19
   2 oneadmin oneadmin Ubuntu_server         09/27 09:37:42
```

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

Here is a view of templates tab in Sunstone:



Labels can be defined for most of the OpenNebula resources from the admin view. Each resource will store the labels information in its own template, thus it can be easily edited from the CLI or Sunstone. This feature enables the possibility to group the different resources under a given label and filter them in the admin and cloud views. The user will be able to easily find the template she wants to instantiate or select a set of resources to apply a given action.



The list of labels defined for each pool will be shown in the left navigation menu. After clicking on one of these labels only the resources with this label will be shown in the table. This filter is also available in the cloud view inside the virtual machine creation form to easily select a specific template.

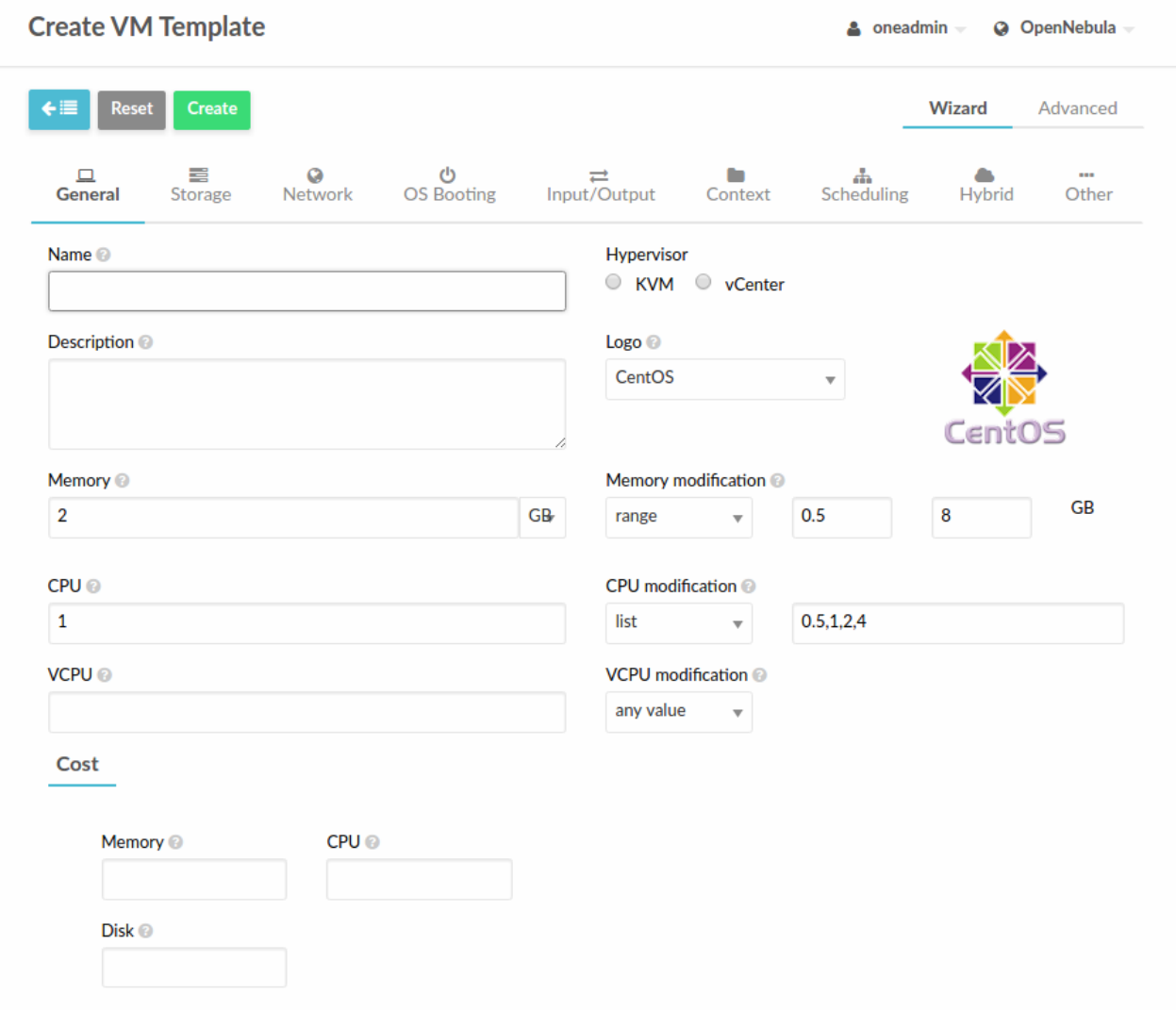
Adding and Deleting Templates

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
```

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:

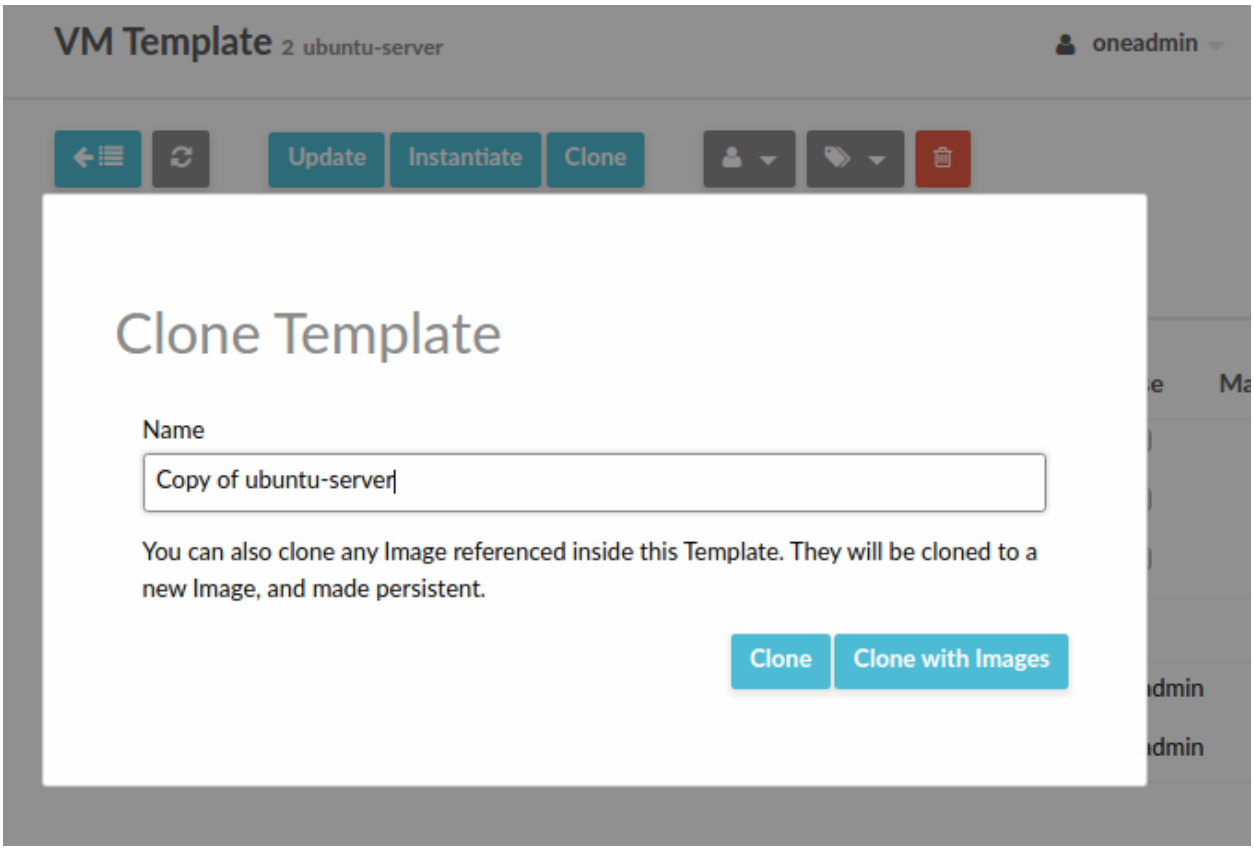


Cloning Templates

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

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

If you use the `onetemplate clone --recursive` option, OpenNebula will clone each one of the Images used in the Template Disks. These Images are made persistent, and the cloned template DISK/IMAGE_ID attributes are replaced to point to them.



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
```

Sharing 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
...
```

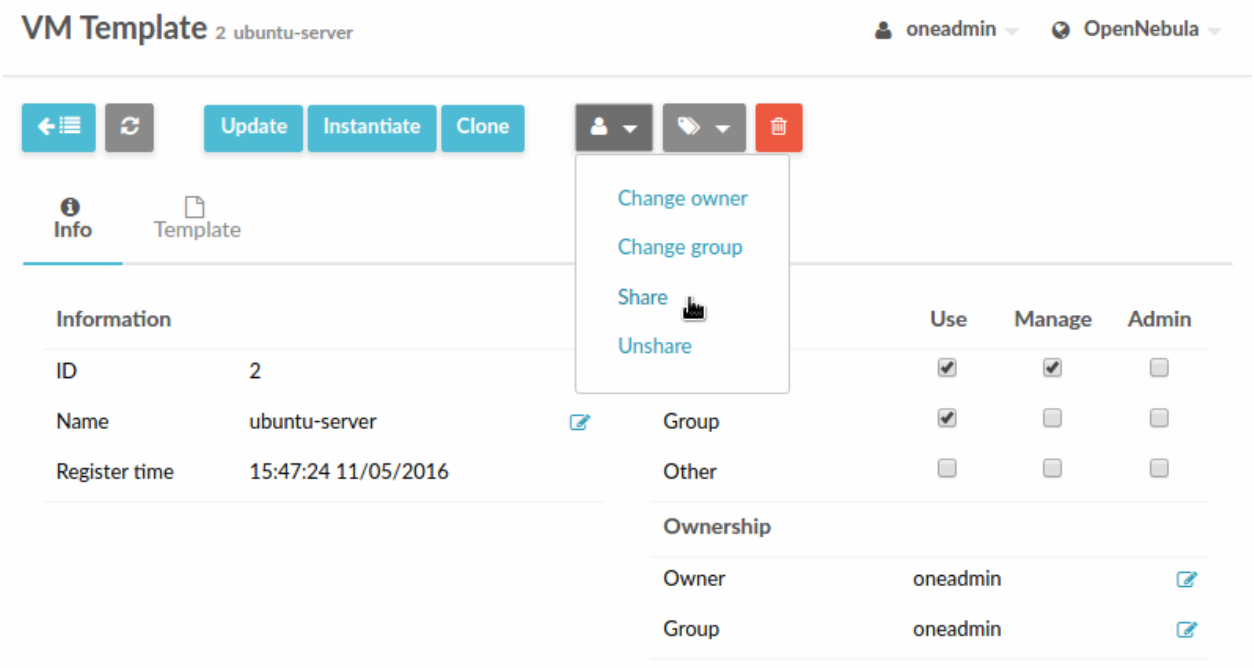
```
PERMISSIONS
OWNER           : um-
GROUP           : u--
OTHER           : ---
```

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 `onetemplate chmod --recursive` option will perform the chmod action also on each one of the Images used in the Template disks.

Sunstone offers an "alias" for `onetemplate chmod --recursive 640`, the share action:



4.4 Managing Virtual Machines Instances

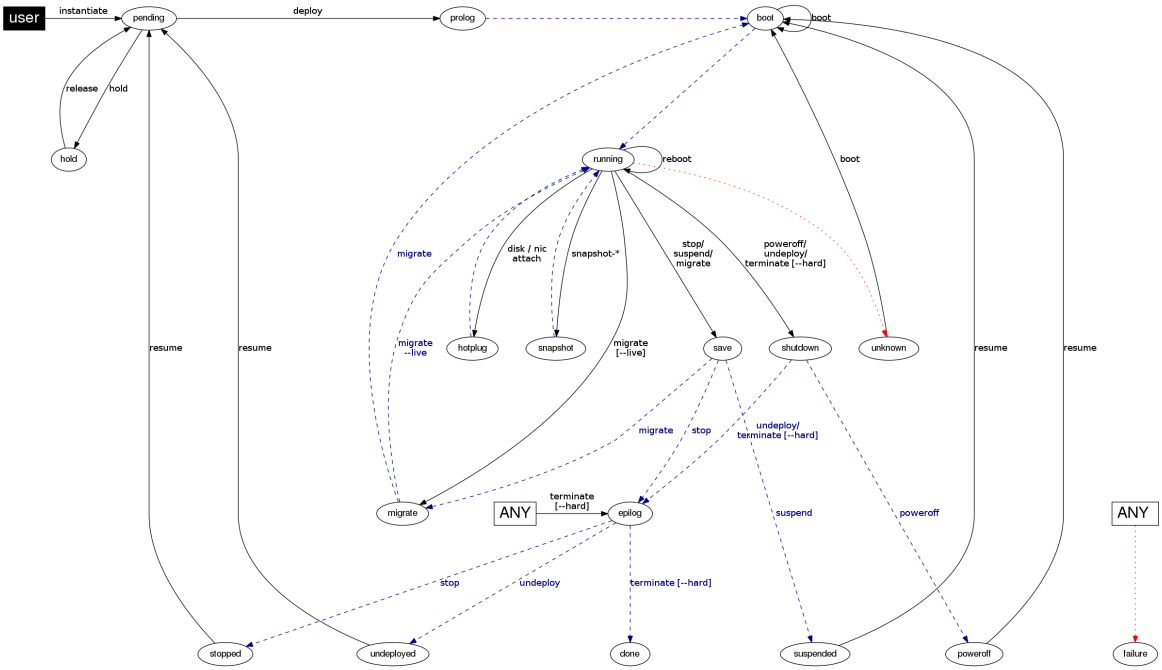
This guide follows the *Creating Virtual Machines guide*. Once a Template is instantiated to a Virtual Machine, there are a number of operations that can be performed using the `onevm` command.

4.4.1 Virtual Machine Life-cycle

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

Note: 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 *Virtual Machines States Reference guide*):

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.
clon	Cloning	The VM is waiting for one or more disk images to finish the initial copy to the repository (image state still in `lock`)
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 reschedule the VM).
poff	PowerOff	Same as suspended, but no check-



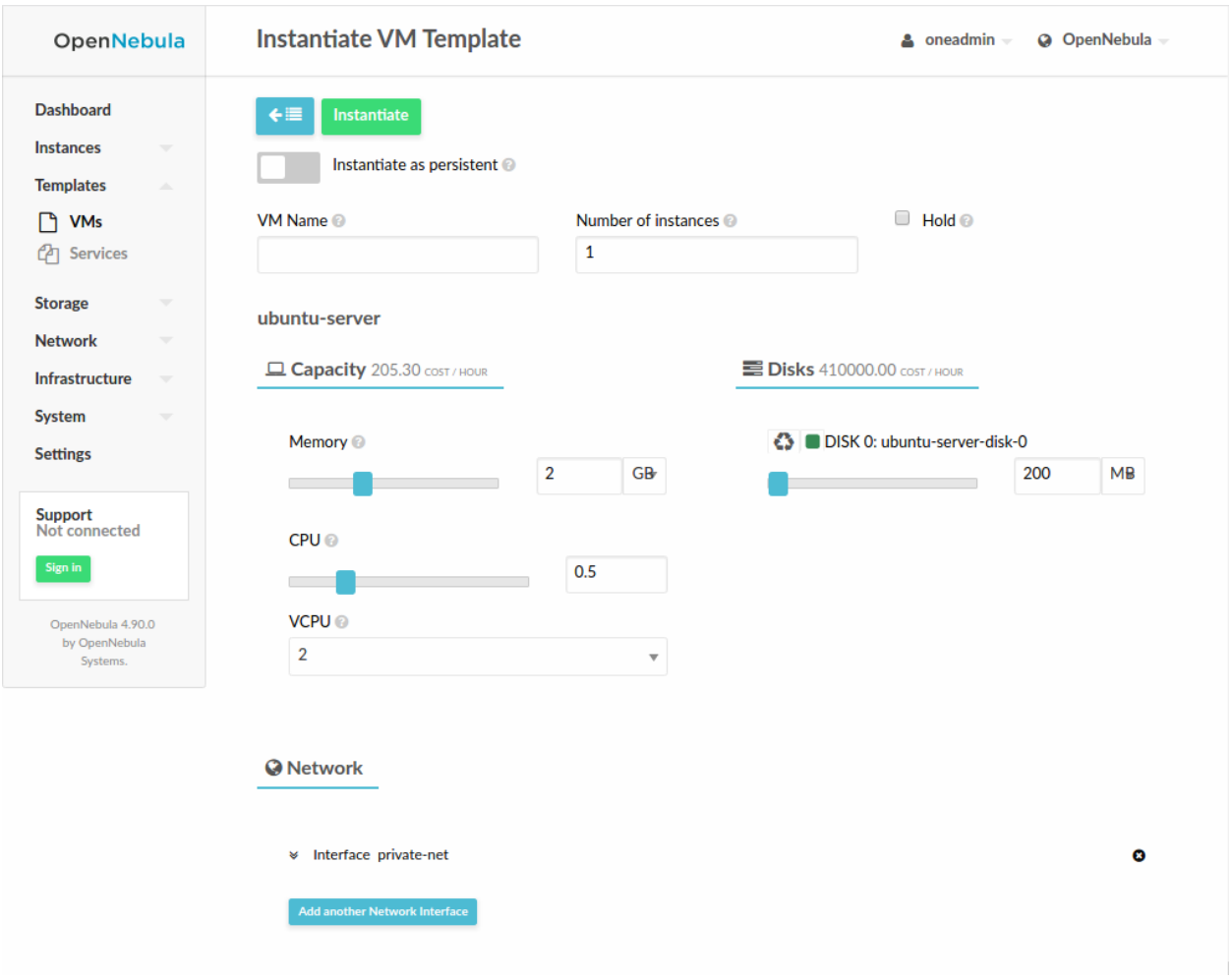
4.4.2 Managing Virtual Machines

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

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

Note: 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
```

If the template has *USER INPUTS* defined the CLI will prompt the user for these values:

```
$ onetemplate instantiate vm-example --name my_vm
There are some parameters that require user input.
  * (BLOG_TITLE) Blog Title: <my_title>
  * (DB_PASSWORD) Database Password:
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 an instance with the `onevm terminate` command, from any state. It will shutdown (if needed) and delete the VM. This operation will free the resources (images, networks, etc) used by the VM.

If the instance is running, there is a `--hard` option that has the following meaning:

- `terminate`: Gracefully shuts down and deletes 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.

- `terminate --hard`: Same as above but the VM is immediately destroyed. Use this action instead of `terminate` when the VM doesn't have ACPI support.

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.

Note: When the guest is shutdown from within the VM, OpenNebula will put the VM in the `poweroff` state.

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.

Rebooting VM Instances...

Use the following commands to reboot a VM:

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

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

Disk Snapshots

There are two kinds of operations related to disk snapshots:

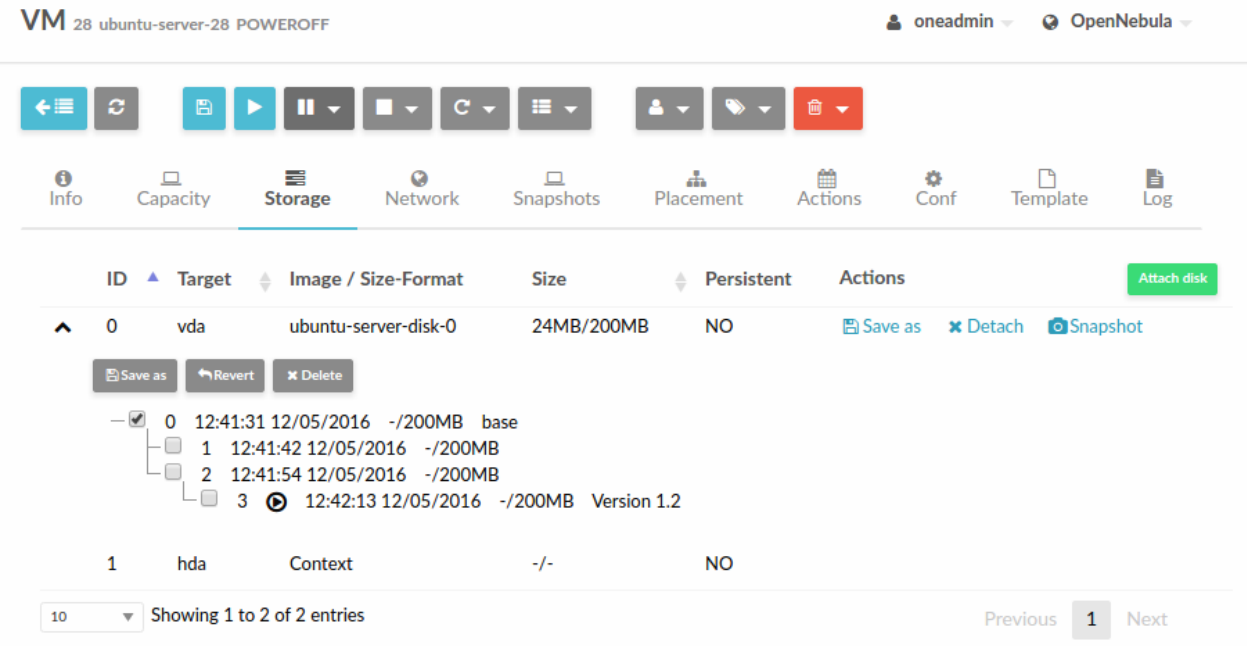
- `disk-snapshot-create`, `disk-snapshot-revert`, `disk-snapshot-delete`: Allows the user to take snapshots of the disk states and return to them during the VM life-cycle. It is also possible to delete snapshots.
- `disk-saveas`: Exports VM disk (or a previously created snapshot) to an image. This is a live action.

Warning: Disk Snapshots are not supported in vCenter

Managing Disk Snapshots

A user can take snapshots of the disk states at any moment in time (if the VM is in `RUNNING`, `POWEROFF` or `SUSPENDED` states). These snapshots are organized in a tree-like structure, meaning that every snapshot has a parent, except for the first snapshot whose parent is `-1`. At any given time a user can revert the disk state to a previously taken snapshot. The active snapshot, the one the user has last reverted to, or taken, will act as the parent of the next snapshot. In addition, it's possible to delete snapshots that are not active and that have no children.

- `disk-snapshot-create <vmid> <diskid> <name>`: Creates a new snapshot of the specified disk.
- `disk-snapshot-revert <vmid> <diskid> <snapshot_id>`: Reverts to the specified snapshot. The snapshots are immutable, therefore the user can revert to the same snapshot as many times as he wants, the disk will return always to the state of the snapshot at the time it was taken.
- `disk-snapshot-delete <vmid> <diskid> <snapshot_id>`: Deletes a snapshot if it has no children and is not active.



`disk-snapshot-create` can take place when the VM is in `RUNNING` state, provided that the drivers support it, while `disk-snapshot-revert` requires the VM to be `POWEROFF` or `SUSPENDED`. Live snapshots are only supported for some drivers:

- Hypervisor `VM_MAD=kvm` combined with `TM_MAD=qcow2` datastores. In this case OpenNebula will request that the hypervisor executes `virsh snapshot-create`.
- Hypervisor `VM_MAD=kvm` with Ceph datastores (`TM_MAD=ceph`). In this case OpenNebula will initially create the snapshots as Ceph snapshots in the current volume.

With CEPH and qcow2 datastores and KVM hypervisor you can enable QEMU Guest Agent. With this agent enabled the filesystem will be frozen while the snapshot is being done.

OpenNebula will not automatically handle non-live `disk-snapshot-create` and `disk-snapshot-revert` operations for VMs in `RUNNING` if the drivers do not support it. In this case the user needs to suspend or poweroff the VM before creating the snapshot.

See the Storage Driver guide for a reference on the driver actions invoked to perform live and non-live snapshost.

Persistent Image Snapshots

These actions are available for both persistent and non-persistent images. In the case of persistent images the snapshots **will** be preserved upon VM termination and will be able to be used by other VMs using that image. See the *snapshots* section in the Images guide for more information.

Back-end Implementations

The snapshot operations are implemented differently depending on the storage back-end:

Operation/TM_MAD	Ceph	Shared and SSH	Qcow2	Dev, FS_LVM, LVM
Snap Create	Creates a protected snapshot	Copies the file.	Creates a new qcow2 image with the previous disk as the backing file.	*Not Supported*
Snap Create (live)	*Not Supported*	*Not Supported*	(For KVM only) Launches `virsh snapshot-create`.	*Not Supported*
Snap Revert	Overwrites the active disk by creating a new snapshot of an existing protected snapshot	Overwrites the file with a previously copied one.	Creates a new qcow2 image with the selected snapshot as the backing file.	*Not Supported*
Snap Delete	Deletes a protected snapshot	Deletes the file.	Deletes the selected qcow2 snapshot.	*Not Supported*

Warning: Depending on the `DISK/CACHE` attribute the live snapshot may or may not work correctly. To be sure, you can use `CACHE=writethrough`, although this delivers the slowest performance.

Exporting Disk Images with `disk-saveas`

Any VM disk can be exported to a new image (if the VM is in `RUNNING`, `POWEROFF` or `SUSPENDED` states). This is a live operation that happens immediately. This operation accepts `--snapshot <snapshot_id>` as an optional argument, which specifies a disk snapshot to use as the source of the clone, instead of the current disk state (value by default).

Warning: This action is not in sync with the hypervisor. If the VM is in `RUNNING` state make sure the disk is unmounted (preferred), synced or quiesced in some way or another before taking the snapshot.

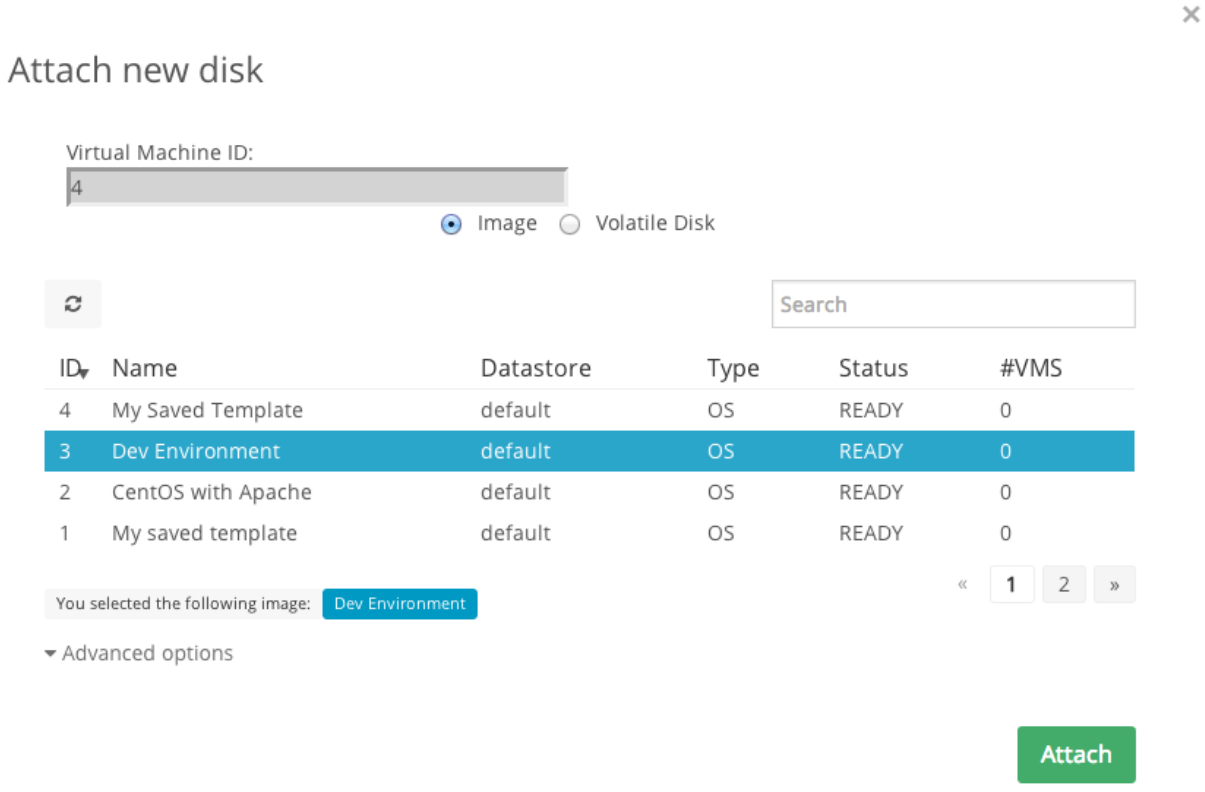
Disk Hot-plugging

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
```



NIC Hot-plugging

You can hot-plug network interfaces to VMs in the `RUNNING`, `POWEROFF` or `SUSPENDED` states. Simply specify the network where the new interface should be attached to, for example:

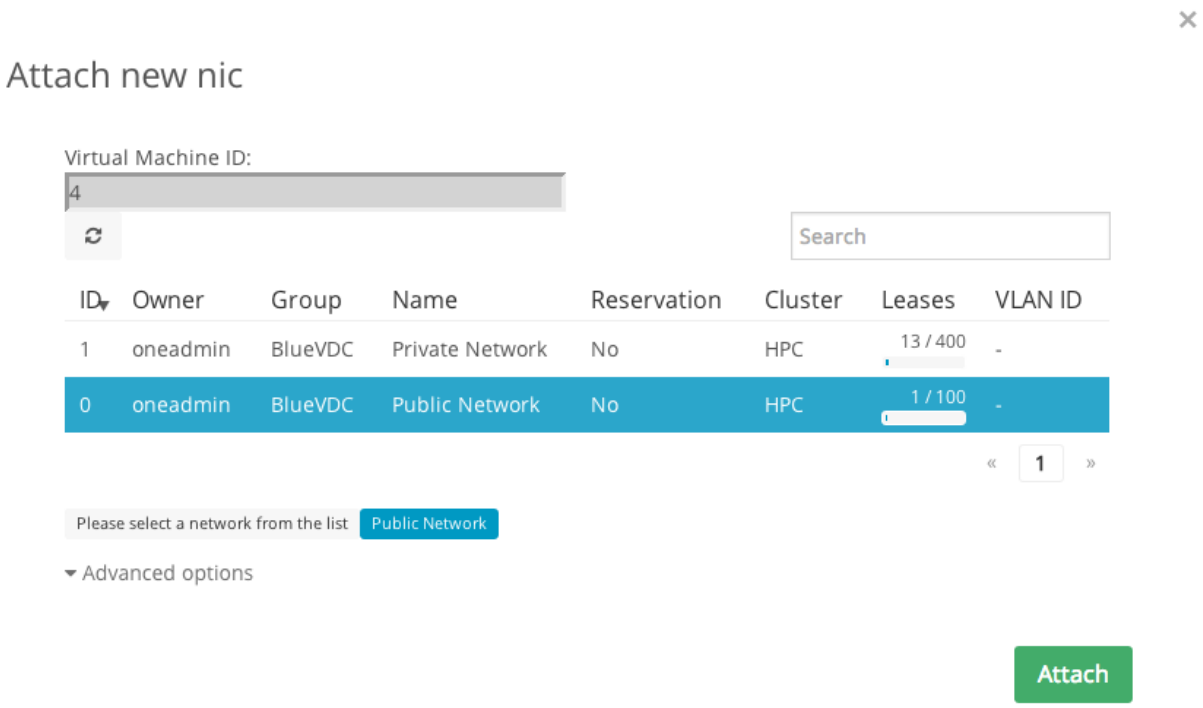
```
$ onevm show 2

VIRTUAL MACHINE 2 INFORMATION
ID                  : 2
NAME                : centos-server
STATE               : ACTIVE
LCM_STATE           : RUNNING

...

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

...

$ onevm nic-attach 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
STATE               : ACTIVE
LCM_STATE           : RUNNING

...


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
...
```

You can also detach a NIC by its ID. If you want to detach interface 1 (MAC `02:00:ac:10:00:ca`), execute:

```
$ onevm nic-detach 2 1
```



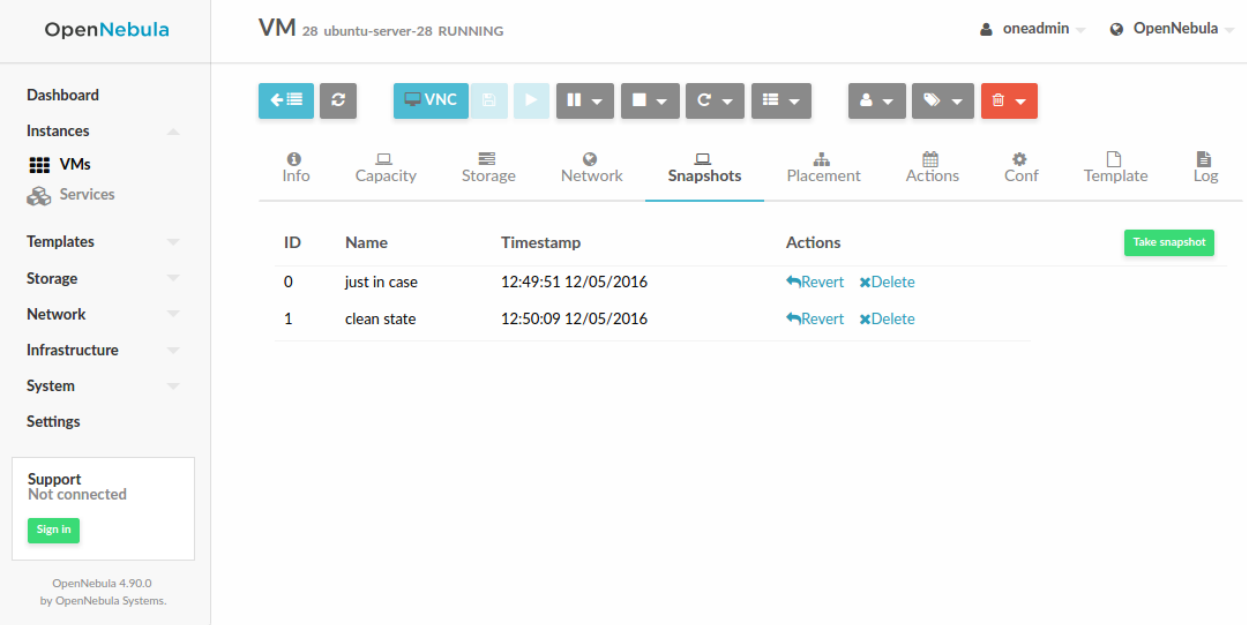
Snapshotting

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

```
$ 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
```

Warning: **For KVM only**. 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.
- Snapshots are only available if all the VM disks use the *qcow2 driver*.



Resizing VM Capacity

You may resize the capacity assigned to a Virtual Machine in terms of the virtual CPUs, memory and CPU allocated. VM resizing can be done in any of the following states: POWEROFF, UNDEPLOYED.

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

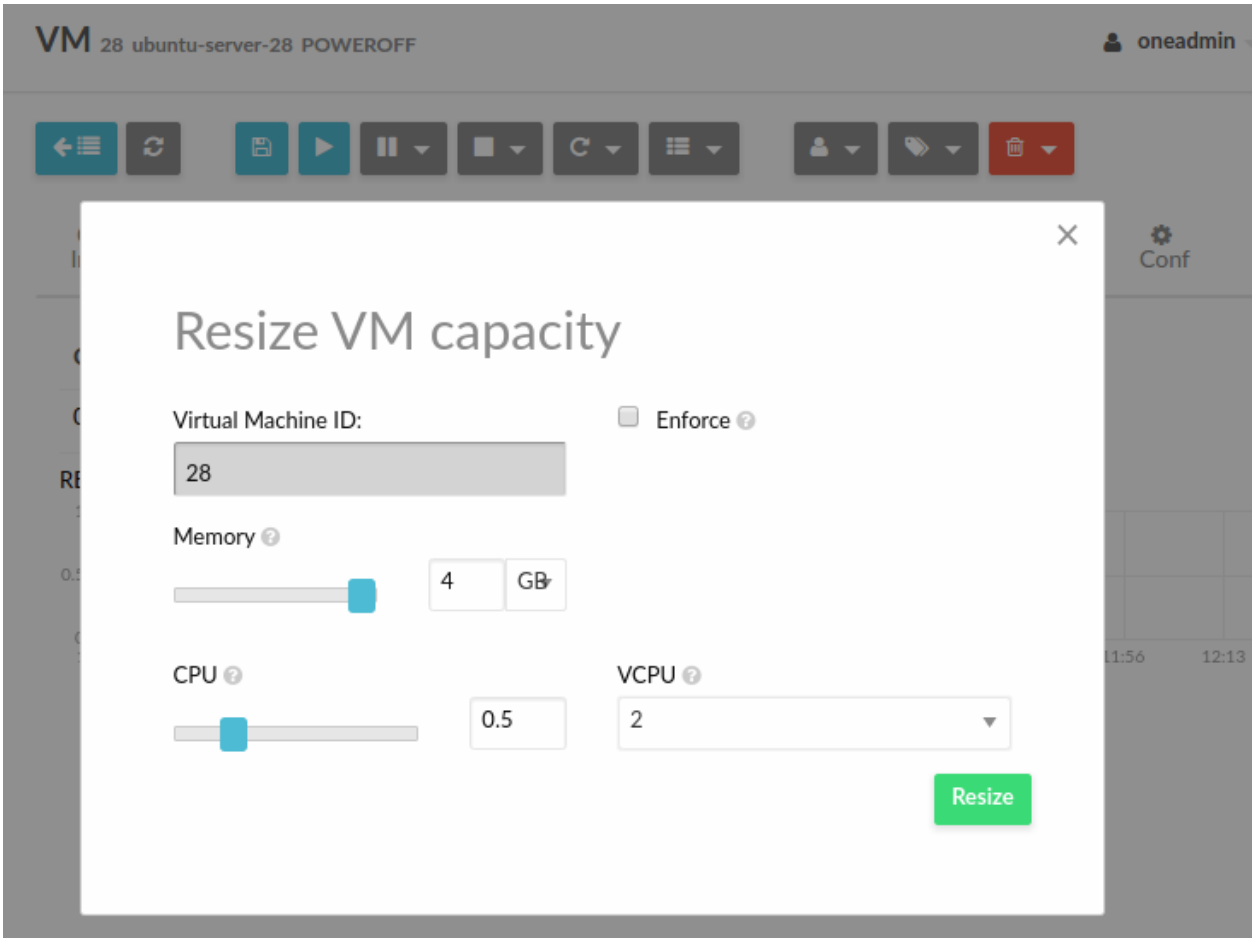
- Perform any operation needed to prepare your Virtual Machine for shutting down, e.g. you may want to manually stop some services
- Poweroff the Virtual Machine
- Resize the VM
- Resume the Virtual Machine using the new capacity

Note that using this procedure the VM will preserve any resource assigned by OpenNebula, such as IP leases.

The following is an example of the previous procedure from the command line:

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

From Sunstone:



Resizing VM Disks

If the disks assigned to a Virtual Machine need more size, this can achieved at instantiation time of the VM. The SIZE parameter of the disk can be adjusted and, if it is bigger than the original size of the image, OpenNebula will:

- Increase the size of the disk container prior to launching the VM
- Using the *contextualization packages*, at boot time the VM will grow the filesystem to adjust to the new size. **This is only available for Linux guests in KVM**.

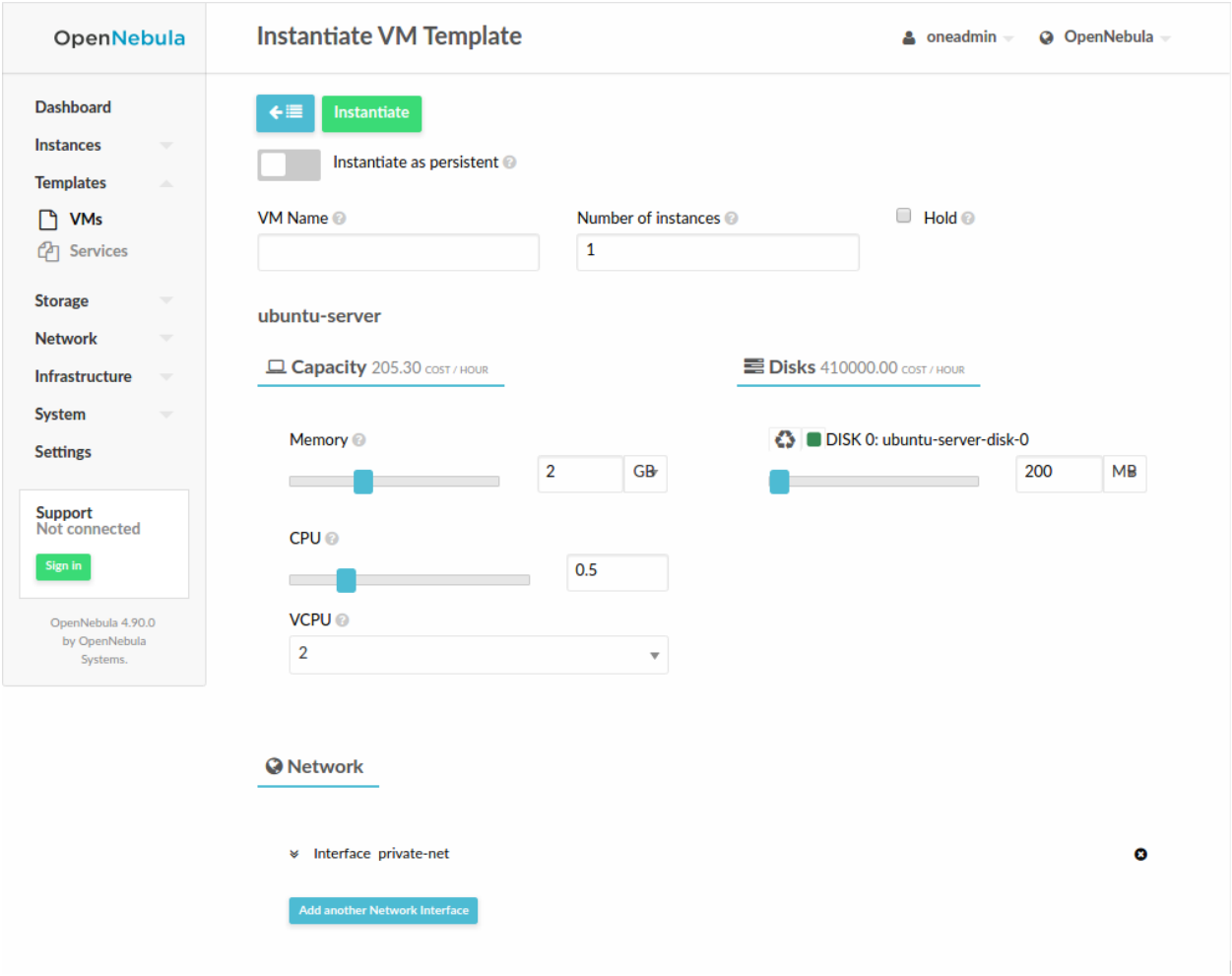
This can be done with an extra file given to the `instantiate` command:

```
$ cat /tmp/disk.txt
DISK = [ IMAGE_ID = 4,
         SIZE = 2000]    # If Image 4 is 1 GB, OpenNebula will resize it to 2 GB

$ onetemplate instantiate 7 /tmp/disk.txt
```

Or with CLI options:

```
$ onetemplate instantiate <template> --disk image0:size=20000
```

This can also be achieved from Sunstone, both in Cloud and Admin View, at the time of instantiating a VM Template:



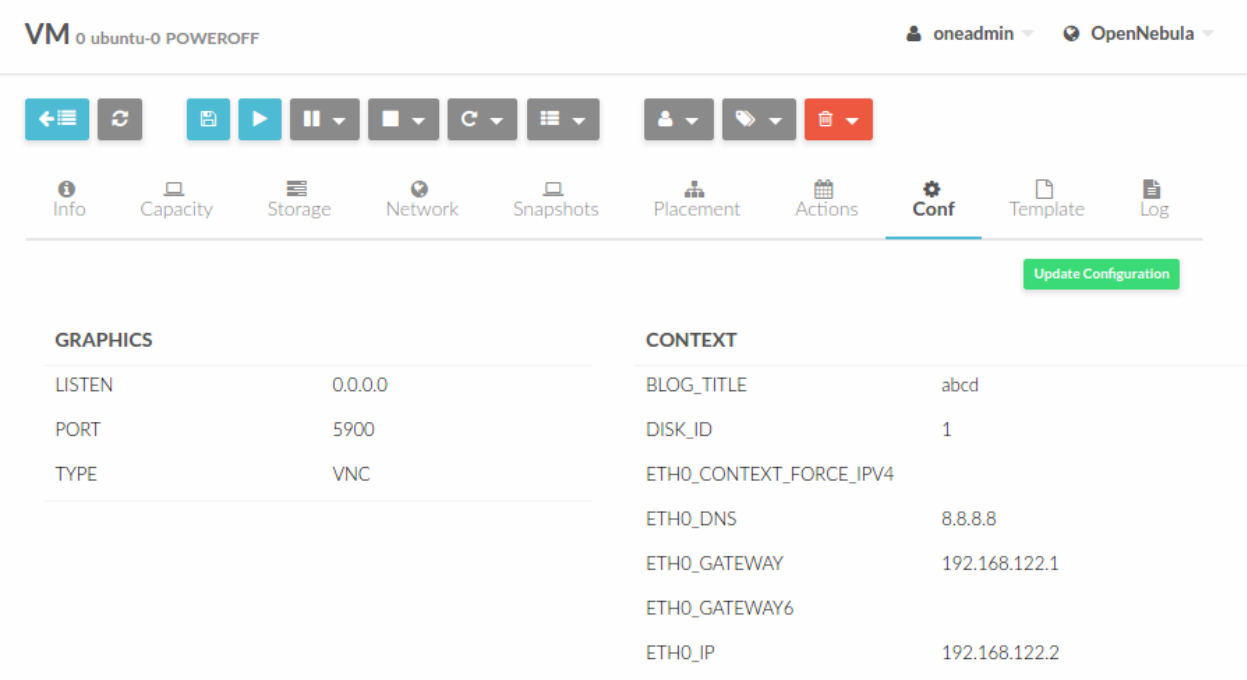
Updating VM Configuration

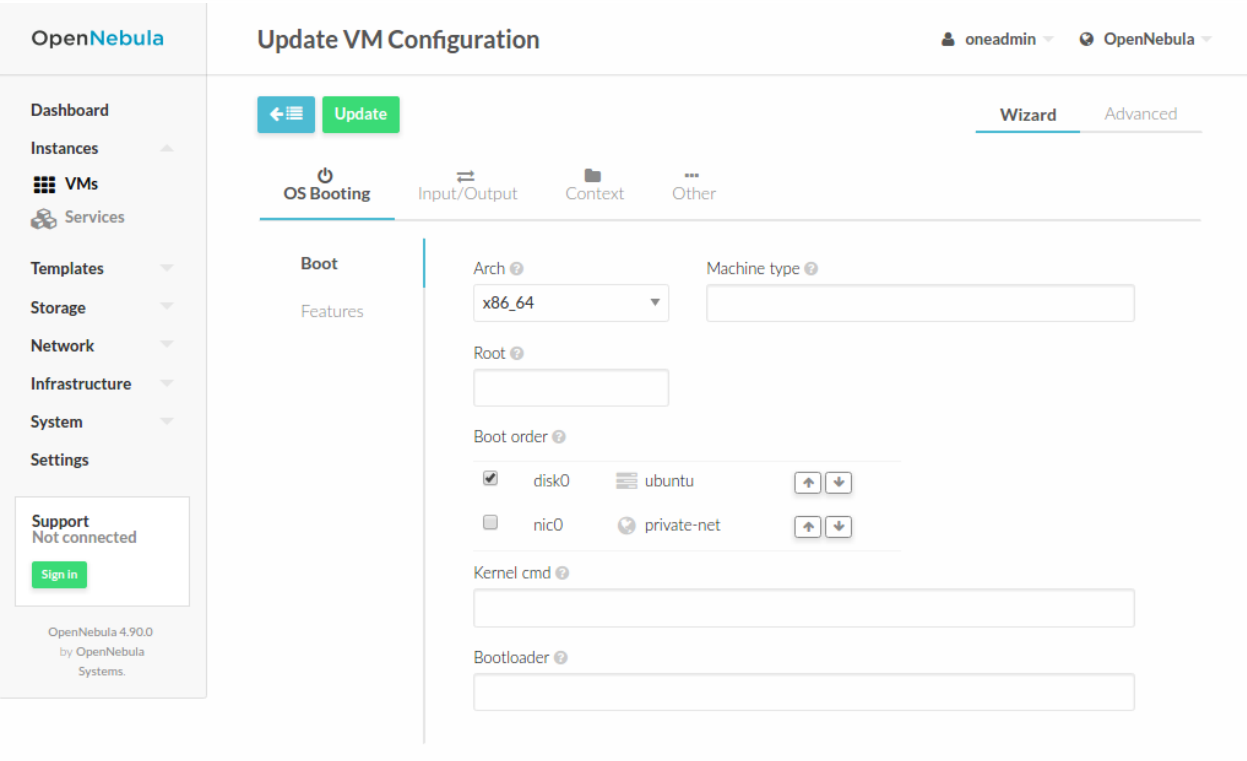
Some of the VM configuration attributes defined in the VM Template can be updated after the VM is created. If the VM is not running, the `onevm updateconf` command will allow you to change the following attributes:

Attribute	Sub-attributes
OS	ARCH, MACHINE, KERNEL, INITRD, BOOTLOADER, BOOT
FEATURES	ACPI, PAE, APIC, LOCALTIME, HYPERV, GUEST_AGENT
INPUT	TYPE, BUS
GRAPHICS	TYPE, LISTEN, PASSWD, KEYMAP
RAW	DATA, DATA_VMX, TYPE
CONTEXT	Any value. **Variable substitution will be made**

Note: Visit the *Virtual Machine Template reference* for a complete description of each attribute

In Sunstone this action is inside the 'Conf' VM panel:





Cloning a VM

A VM Template or VM instance can be copied to a new VM Template. This copy will preserve the changes made to the VM disks after the instance is terminated. The template is private, and will only be listed to the owner user.

There are two ways to create a persistent private copy of a VM:

- Instantiate a template 'to persistent'

- Save a existing VM instance with `onevm save`

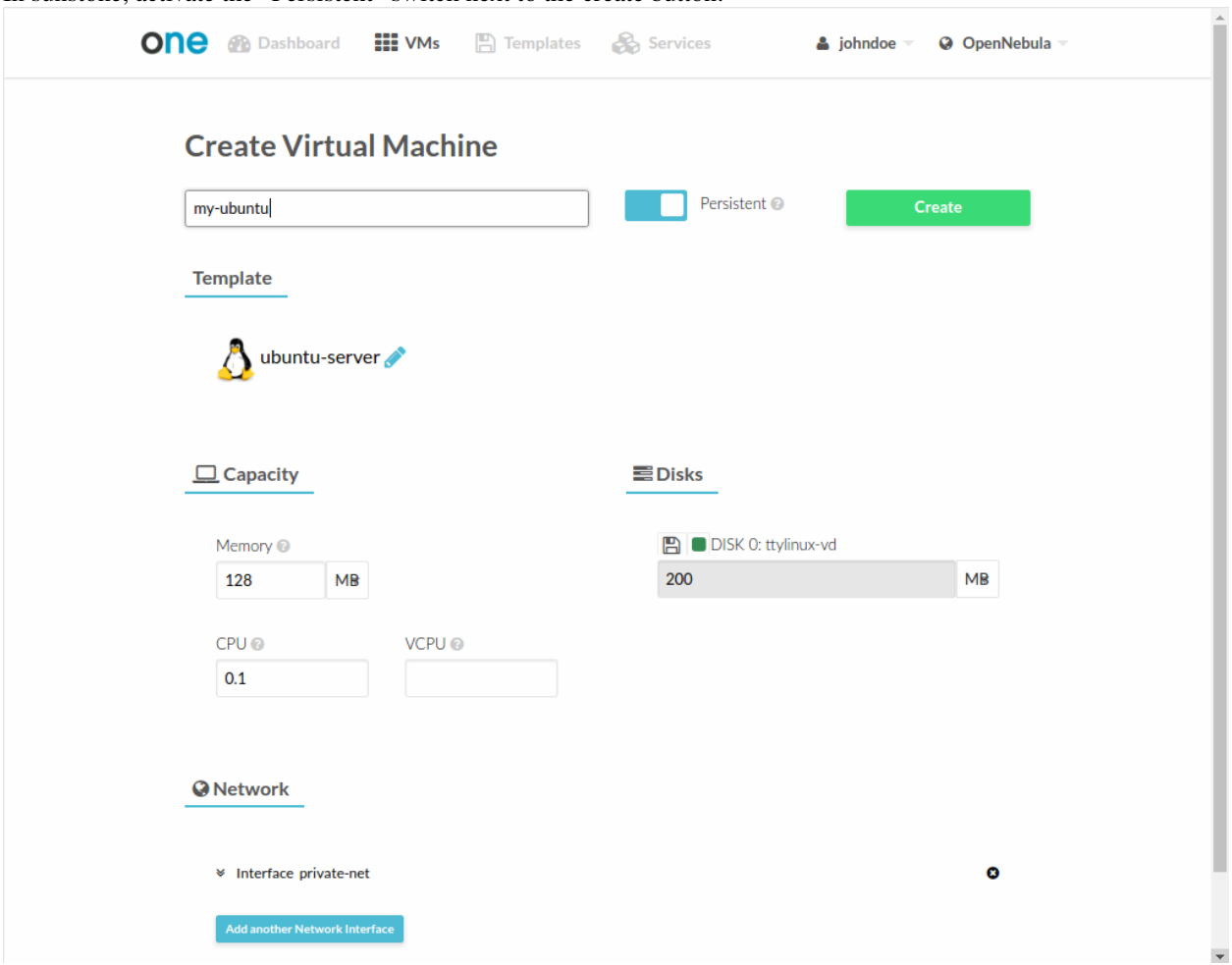
Instantiate to persistent

When **instantiating to persistent** the Template is cloned recursively (a private persistent clone of each disk Image is created), and that new Template is instantiated.

To "instantiate to persistent" use the `--persistent` option:

```
$ onetemplate instantiate web_vm --persistent --name my_vm
VM ID: 31

$ onetemplate list
  ID USER            GROUP           NAME                                REGTIME
   7 oneadmin        oneadmin        web_vm                           05/12 14:53:11
   8 oneadmin        oneadmin        my_vm                            05/12 14:53:38

$ oneimage list
  ID USER       GROUP      NAME            DATASTORE     SIZE TYPE PER STAT RVMS
   7 oneadmin   oneadmin   web-img         default       200M OS   Yes used    1
   8 oneadmin   oneadmin   my_vm-disk-0    default       200M OS   Yes used    1
```

In sunstone, activate the "Persistent" switch next to the create button:



Please bear in mind the following `ontemplate instantiate --persistent` limitation:

- Volatile disks cannot be persistent, and the contents will be lost when the VM is terminated. The cloned VM Template will contain the definition for an empty volatile disk.

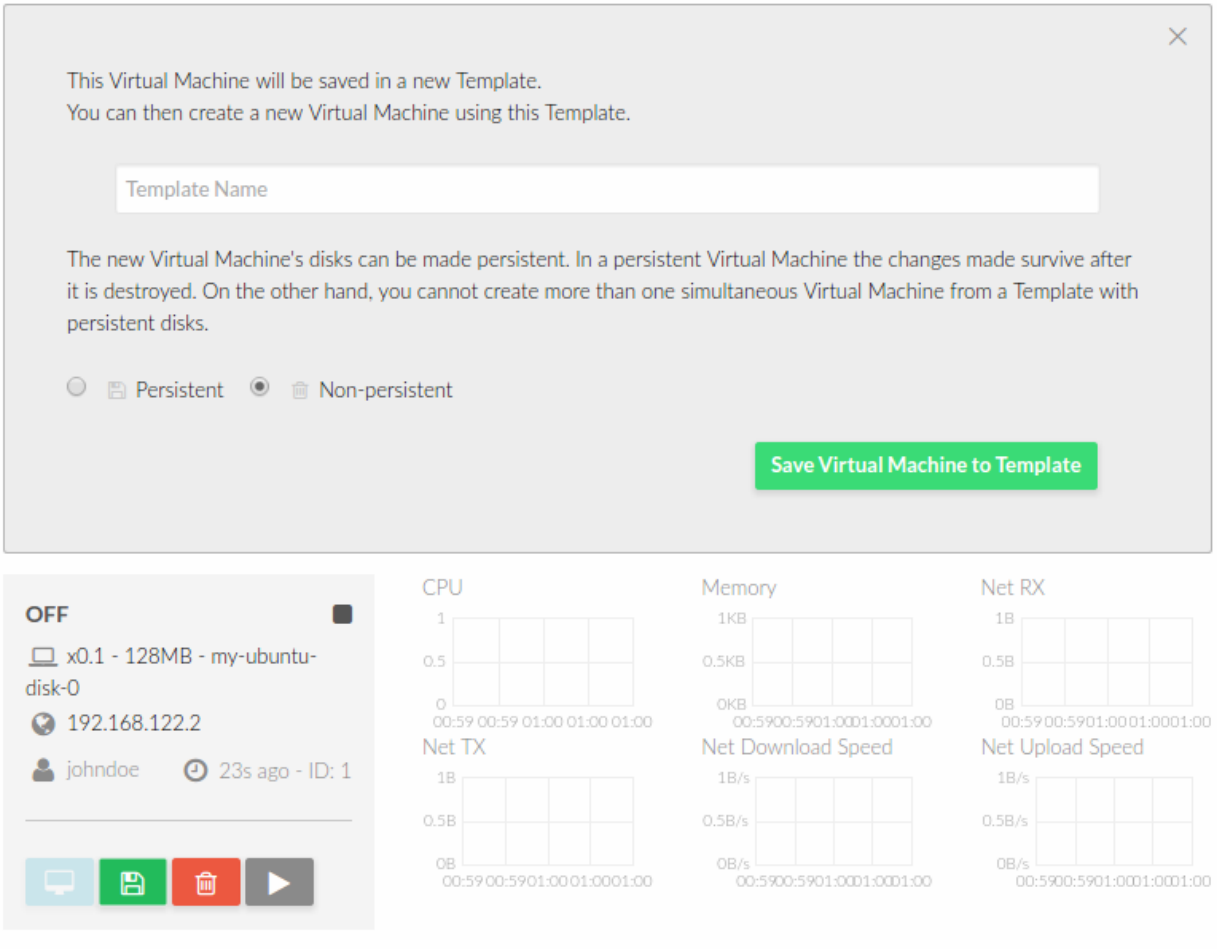
Save a VM Instance

Alternatively, a VM that was not created as persistent can be **saved** before it is destroyed. To do so, the user has to `poweroff` the VM first and then use the `save` operation.

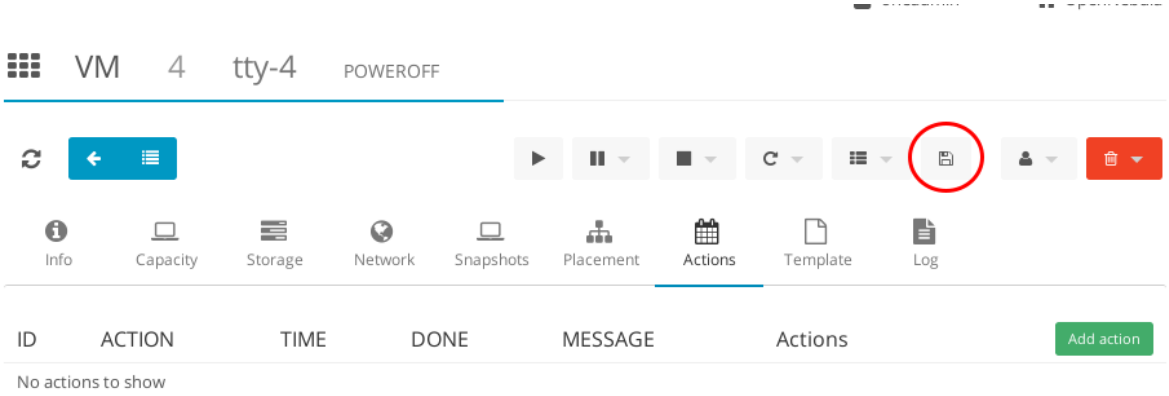
This action clones the VM source Template, replacing the disks with snapshots of the current disks (see the disk-snapshot action). If the VM instance was resized, the current capacity is also used. The new cloned Images can be made persistent with the `--persistent` option. NIC interfaces are also overwritten with the ones from the VM instance, to preserve any attach/detach action.

```
$ onevm save web_vm copy_of_web_vm --persistent
Template ID: 26
```

In the *Cloud View*:



From the Admin View:



Please bear in mind the following `onevm save` limitations:

- The VM's source Template will be used. If this Template was updated since the VM was instantiated, the new contents will be used.
- Volatile disks cannot be saved, and the current contents will be lost. The cloned VM Template will contain the definition for an empty volatile disk.
- Disks and NICs will only contain the target Image/Network ID. If your Template requires extra configuration (such as DISK/DEV_PREFIX), you will need to update the new Template.

Scheduling Actions

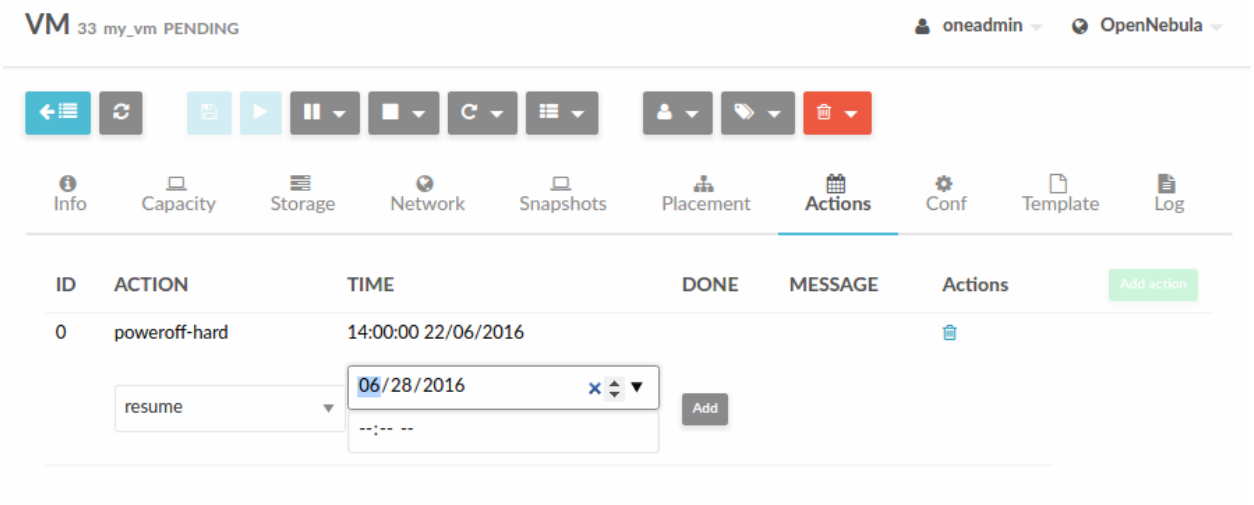
Most of the onevm commands accept the `--schedule` option, allowing users to delay the actions until the given date and time.

Here is an usage example:

```
$ onevm suspend 0 --schedule "09/20"
VM 0: suspend scheduled at 2016-09-20 00:00:00 +0200

$ onevm resume 0 --schedule "09/23 14:15"
VM 0: resume scheduled at 2016-09-23 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:

- `terminate [--hard]`
- `undeploy [--hard]`
- `hold`
- `release`
- `stop`
- `suspend`
- `resume`
- `delete`
- `delete-recreate`
- `reboot [--hard]`
- `poweroff [--hard]`
- `snapshot-create`

User Defined Data

Custom attributes can be added 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.

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. In an infrastructure with *multiple system datastores*, the VM storage can be also migrated (the datastore id can be specified).

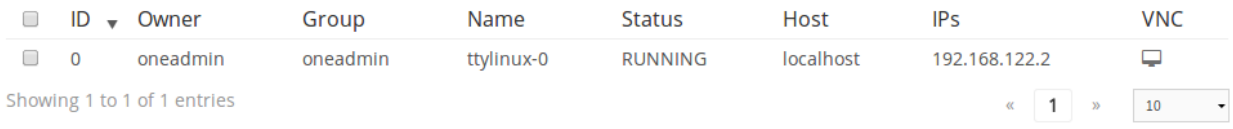
Note: By default, the above operations do not check the target host capacity. You can use the `--enforce` option to be sure that the host capacity is not overcommitted.

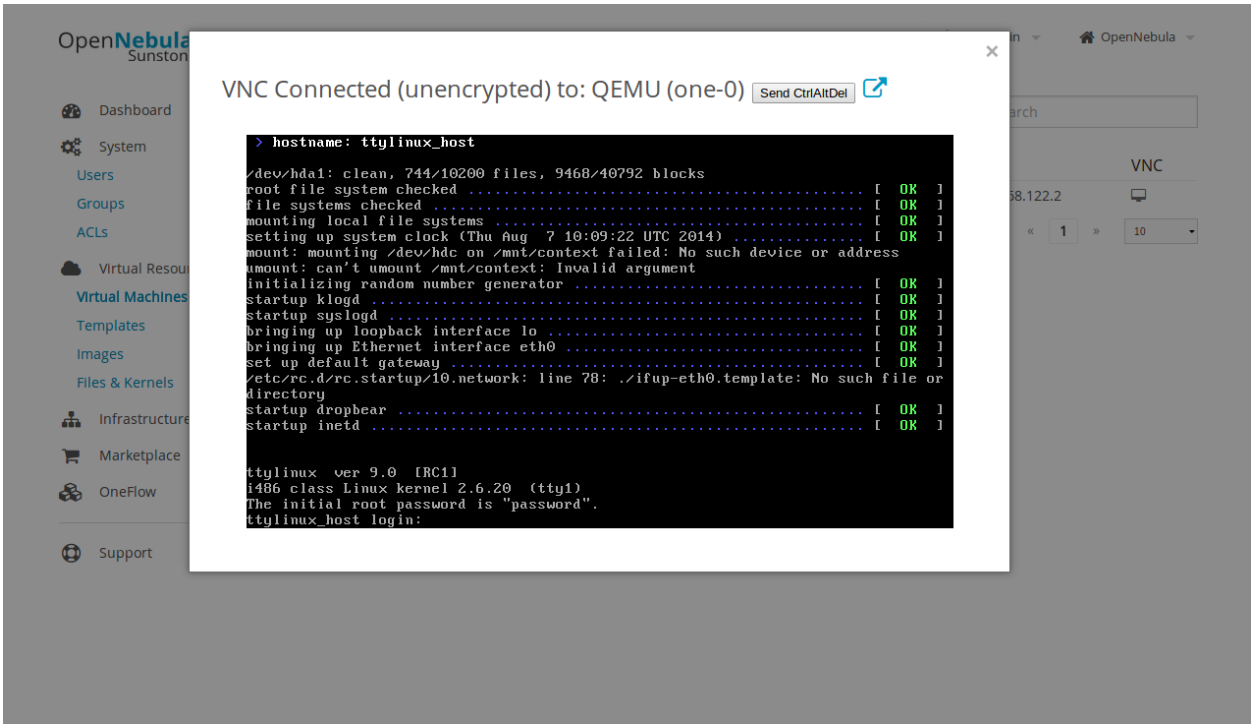
Troubleshooting:

- `recover`: If the VM is stuck in any other state (or the boot operation does not work), you can recover the VM with the following options. Read the Virtual Machine Failures guide for more information.
  - `--success`: simulates the success of the missing driver action
  - `--failure`: simulates the failure of the missing driver action
  - `--retry`: retries to perform the current driver action. Optionally the `--interactive` can be combined if its a Transfer Manager problem.
  - `--delete`: Deletes the VM, moving it to the DONE state immediately
  - `--recreate`: Deletes the VM, and moves it to the PENDING state
- `migrate` or `resched`: A VM in the UNKNOWN state can be booted in a different host manually (`migrate`) or automatically by the scheduler (`resched`). This action must be performed only if the storage is shared, or manually transfered by the administrator. OpenNebula will not perform any action on the storage for this migration.

4.4.3 VNC/Spice Access through Sunstone

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





The Sunstone documentation contains a section on VCN troubleshooting.

4.4.4 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.

4.5 vCenter Specifics

4.5.1 vCenter VM and VM Templates

To learn how to use VMs and VM Templates you can read the *Managing Virtual Machines Instances* and *Managing Virtual Machine Templates*, but first take into account the following considerations. In order to manually create a VM Template definition in OpenNebula that represents a vCenter VM Template, the following attributes are needed:

Operation	Note
CPU	Physical CPUs to be used by the VM. This does not have to relate to the CPUs used by the vCenter VM Template, OpenNebula will change the value accordingly
MEMORY	Physical Memory in MB to be used by the VM. This does not have to relate to the CPUs used by the vCenter VM Template, OpenNebula will change the value accordingly
NIC	Check *VM template reference*. Valid MODELs are: virtuale1000, virtuale1000e, virtualpcnet32, virtualsriovethernetcard, virtualvmxnetm, virtualvmxnet2, virtualvmxnet3.
DISK	Check *VM template reference*. Take into account that all images are persistent, as explained in vCenter Datastore Setup.
GRAPHICS	Multi-value - Only VNC supported, check the *VM template reference*.
PUBLIC_CLOUD	Multi-value. TYPE must be set to vcenter, and VM_TEMPLATE must point to the uuid of the vCenter VM that is being represented
SCHED_REQUIREMENTS	Example: "name of the vCenter cluster where this VM Template can instantiated into a VM". See *VM Scheduling section* for more details.
CONTEXT	All *sections* will be honored except FILES. You can find more information about contextualization in the *vcenter Contextualization* section.
KEEP_DISKS_ON_DONE	(Optional) Prevent OpenNebula from erasing the VM disks upon reaching the done state (either via shutdown or cancel)
VCENTER_DATASTORE	By default, the VM will be deployed to the datastore where the VM Template is bound to. This attribute allows to set the name of the datastore where this VM will be deployed. This can be overwritten explicitly at deployment time from the CLI or Sunstone. More information in the :ref:‘ vCenter Datastore Setup Section <vcenter_ds>‘
RESOURCE_POOL	By default, the VM will be deployed to the default resource pool. If this attribute is set, its value will be used to confine this the VM in the referred resource pool. Check this section for more information.

After a VM Template is instantiated, the life-cycle of the resulting virtual machine (including creation of snapshots) can be controlled through OpenNebula. Also, all the operations available in the vCenter Admin view can be performed, including:

- network management operations like the ability to attach/detach network interfaces
- capacity (CPU and MEMORY) resizing
- VNC connectivity
- Attach/detach VMDK images as disks

The following operations are not available for vCenter VMs:

- migrate
- livemigrate

The monitoring attributes retrieved from a vCenter VM are:

- ESX_HOST
- GUEST_IP
- GUEST_STATE
- VMWARETOOLS_RUNNING_STATUS
- VMWARETOOLS_VERSION
- VMWARETOOLS_VERSION_STATUS

VM Template Cloning Procedure

OpenNebula uses VMware cloning VM Template procedure to instantiate new Virtual Machines through vCenter. From the VMware documentation:

– Deploying a virtual machine from a template creates a virtual machine that is a copy of the template. The new virtual machine has the virtual hardware, installed software, and other properties that are configured for the template.

A VM Template is tied to the host where the VM was running, and also the datastore(s) where the VM disks where placed. By default, the VM will be deployed in that datastore where the VM Template is bound to, although another datastore can be selected at deployment time. Due to shared datastores, vCenter can instantiate a VM Template in any of the hosts belonging to the same cluster as the original one.

OpenNebula uses several assumptions to instantiate a VM Template in an automatic way:

- **diskMoveType**: OpenNebuls instructs vCenter to "move only the child-most disk backing. Any parent disk backings should be left in their current locations.". More information here
- Target **resource pool**: OpenNebula uses the default cluster resource pool to place the VM instantiated from the VM template, unless VCENTER_RESOURCE_POOL variable defined in the OpenNebula host template, or the tag RESOURCE_POOL is present in the VM Template inside the PUBLIC_CLOUD section.

Saving a VM Template: Instantiate to Persistent

At the time of deploying a VM Template, a flag can be used to create a new VM Template out of the VM.

```
$ onetemplate instantiate <tid> --persistent
```

Whenever the VM life-cycle ends, OpenNebula will instruct vCenter to create a new vCenter VM Template out of the VM, with the settings of the VM including any new disks or network interfaces added through OpenNebula. Any new disk added to the VM will be saved as part of the template, and when a new VM is spawned from this new VM Template the disk will be cloned by OpenNebula (ie, it will no longer be persistent).

A new OpenNebula VM Template will also be created pointing to this new VM Template, so it can be instantiated through OpenNebula. This new OpenNebula VM Template will be pointing to the original template until the VM is shutdown, at which point it will be converted to a vCenter VM Template and the OpenNebual VM Template updated to point to this new vCentre VM Template.

This functionality is very useful to create new VM Templates from a original VM Template, changing the VM configuration and/or installing new software, to create a complete VM Template catalog.

VM Scheduling

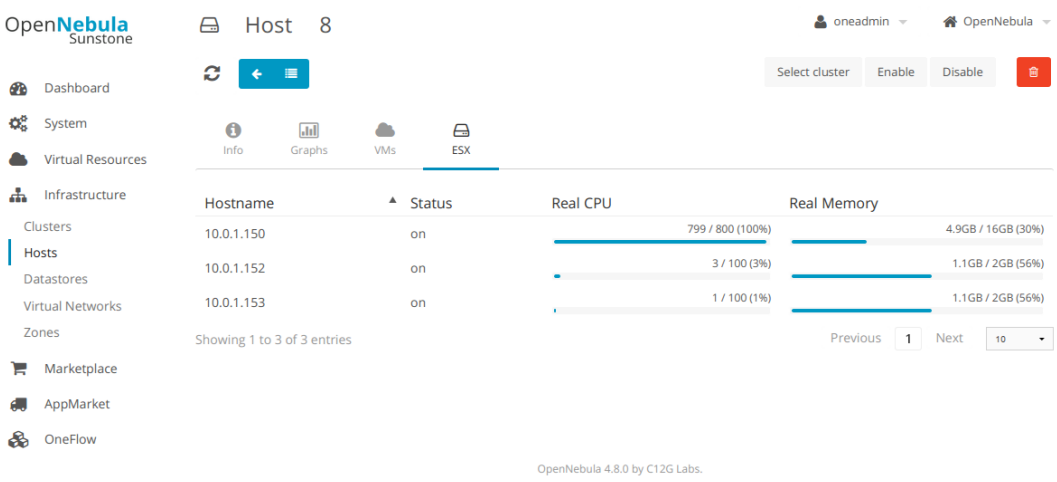
OpenNebula scheduler should only chose a particular OpenNebula host for a OpenNebula VM Template representing a vCenter VM Template, since it most likely only would be available in a particular vCenter cluster.

Since a vCenter cluster is an aggregation of ESX hosts, the ultimate placement of the VM on a particular ESX host would be managed by vCenter, in particular by the Distribute Resource Scheduler (DRS).

In order to enforce this compulsory match between a vCenter cluster and a OpenNebula/vCenter VM Template, add the following to the OpenNebula VM Template:

```
SCHED_REQUIREMENTS = "NAME=\"name of the vCenter cluster where this VM Template can␣
↪instantiated into a VM\""
```

In Sunstone, a host abstracting a vCenter cluster will have an extra tab showing the ESX hosts that conform the cluster.



4.5.2 vCenter Images

You can follow the *Managing Images Section* to learn how to manage images, considering that all images in vCenter are persistent and that VMDK snapshots are not supported as well as the following considerations.

vCenter VMDK images managed by OpenNebula are always persistent, ie, OpenNebula won't copy them for new VMs, but rather the originals will be used. This means that only one VM can use one image at the same time.

vCenter VM Templates with already defined disks will be imported without this information in OpenNebula. These disks will be invisible for OpenNebula, and therefore cannot be detached from the VMs. The imported Templates in OpenNebula can be updated to add new disks from VMDK images imported from vCenter (please note that these will always be persistent).

There are three ways of adding VMDK representations in OpenNebula:

- Upload a new VMDK from the local filesystem
- Register an existent VMDK image already in the datastore
- Create a new empty datablock

The following image template attributes need to be considered for vCenter VMDK image representation in OpenNebula:

Attribute	Description
PERSISTENT	Must be set to 'YES'
PATH	This can be either: • local filesystem path to a VMDK to be uploaded, which can be a single VMDK or tar.gz of vmdk descriptor and flat files (no OVAs supported) • path of an existing VMDK file in the vCenter datastore. In this case a ''vcenter://" prefix must be used (for instance, an image win10.vmdk in a Windows folder should be set to vcenter://Windows/win10.vmdk)
ADAPTER_TYPE	Possible values (careful with the case): lsiLogic, ide, busLogic. More information in the VMware documentation. Known as "Bus adapter controller" in Sunstone.
DISK_TYPE	The type of disk has implications on performance and occupied space. Values (careful with the case): delta,eagerZeroedThick,flatMonolithic,preallocated,raw,rdm,rdmp,seSparse More information in the VMware documentation

VMDK images in vCenter datastores can be:

- Cloned
- Deleted
- Hotplugged to VMs

Images can be imported from the vCenter datastore using the **onevcenter** tool:

```
$ onevcenter images datastore1 --vcenter <vcenter-host> --vuser <vcenter-username> --
↪vpass <vcenter-password>

Connecting to vCenter: vcenter.vcenter3...done!

Looking for Images...done!

  * Image found:
      - Name      : win-test-context-fixed2 - datastore1
      - Path      : win-test-context-fixed2/win-test-context-fixed2.vmdk
      - Type      : VmDiskFileInfo
    Import this Image [y/n]? n

  * Image found:
      - Name      : windows-2008R2 - datastore1
      - Path      : windows/windows-2008R2.vmdk
      - Type      : VmDiskFileInfo
    Import this Image [y/n]? y
    OpenNebula image 0 created!
```

CHAPTER

FIVE

VIRTUAL MACHINE SETUP

5.1 Overview

OpenNebula uses a method called contextualization to send information to the VM at boot time. Its most basic usage is to share networking configuration and login credentials with the VM so it can be configured. More advanced cases can be starting a custom script on VM boot or preparing configuration to use OpenNebula Gate.

5.1.1 How Should I Read This Chapter

Before reading this chapter, you should have already installed your Frontend, the KVM Hosts or vCenter node and have an OpenNebula cloud up and running with at least one virtualization node.

To enable the use of contextualization there are two steps that you need to perform:

- Installing contextualization packages in your images
- Set contextualization data in the VM template

Learn how to do that in the contextualization guide linked below for the hypervisor configured.

5.1.2 Hypervisor Compatibility

Section	Compatibility
KVM Contextualization	This Section applies to KVM.
vCenter Contextualization	This Section applies to vCenter.
Adding Content to your Cloud	This Section applies to both KVM and vCenter.

5.2 KVM Contextualization

5.2.1 Prepare the Virtual Machine Image

Step 1. Start a VM with the OS you want to Customize

Supported contextualization packages are available for the following OS's:

- **CentOS/RHEL** >= 6
- **Debian** >= 6
- **Ubuntu** >= 11.10

- **Windows** >= 7
- **Windows Server** >= 2008

Step 2. Download Contextualization Packages to the VM

CentOS/RHEL

```
# wget https://github.com/OpenNebula/addon-context-linux/releases/download/v5.0.1/
↪one-context_5.0.1.rpm
```

Debian/Ubuntu

```
# wget https://github.com/OpenNebula/addon-context-linux/releases/download/v5.0.1/
↪one-context_5.0.1.deb
```

Windows

Downloads these two files to `C:\`:

- https://raw.githubusercontent.com/OpenNebula/addon-context-windows/master/context.ps1
- https://raw.githubusercontent.com/OpenNebula/addon-context-windows/master/startup.vbs

Step 3. Install Contextualization Packages and Dependencies

CentOS/RHEL 6

```
# rpm -Uvh one-context*rpm
# yum install -y epel-release
# yum install ruby # only needed for onegate command
# yum install -i dracut-modules-growroot
# dracut -f
```

CentOS/RHEL 7

```
# rpm -Uvh one-context*rpm
# yum install -y epel-release
# yum install ruby # only needed for onegate command
# yum install -y cloud-utils-growpart
```

Debian/Ubuntu

```
# dpkg -i one-context*deb
# apt-get install ruby # only needed for onegate command
# apt-get install -y cloud-utils
```

Windows

- Open the Local Group Policy Dialog by running `gpedit.msc`.
- Go to *Computer Configuration -> Windows Settings -> Scripts -> startup* (right click).
- Browse to the `startup.vbs` file and enable it as a startup script.

Step 4. Power Off the Machine and Save it

After these configuration is done you should power off the machine, so it is in a consistent state the next time it boots. Then you will have to save the image.

If you are using OpenNebula to prepare the image you can use the command `onevm disk-saveas`, for example, to save the first disk of a Virtual Machine called "centos-installation" into an image called "centos-contextualized" you can issue this command:

```
$ onevm disk-saveas centos-installation 0 centos-contextualized
```

Using sunstone web interface you can find the option in the Virtual Machine storage tab.

5.2.2 Set Up the Virtual Machine Template

The Virtual Machine Template has a section called context where you can automate different configuration aspects. The most common attributes are network configuration, user credentials and startup scripts. These parameters can be both added using the CLI to the template or using Sunstone Template wizard. Here is an example of the context section using the CLI:

```
CONTEXT = [
    TOKEN = "YES",
    NETWORK = "YES",
    SSH_PUBLIC_KEY = "$USER[SSH_PUBLIC_KEY]",
    START_SCRIPT = "yum install -y ntpdate"
]
```

In the example we are telling OpenNebula to:

- Set OneGate token and onegate information in the context
- Add network configuration to the Virtual Machine
- Enable login into the Virtual Machine using ssh with the value of the user's parameter `SSH_PUBLIC_KEY`
- On Virtual Machine boot execute the command `yum install -y ntpdate`

OneGate Token

OpenNebula has a centralized service to share data between Virtual Machines and the main daemon, useful to set monitoring information that can be gathered inside the VM and configuration data. It also lets you send scaling actions when the Virtual Machine belongs to a Service.

To do so the client installed with the contextualization packages (`onegate`) needs some information:

- **Token**: it's the key specific to each VM used to authenticate with the service
- **OneGate endpoint**: the address where the OneGate daemon is reachable

To fill this information you have to specify `TOKEN = "YES"` in the contextualization section.

Network Configuration

OpenNebula does not rely on a DHCP server to configure networking in the Virtual Machines. To do this configuration it injects the network information in the contextualization section. This is done with option `NETWORK = "YES"`. When OpenNebula finds this option it adds the IP information for each of the network interfaces configured plus extra information that resides in the Virtual Network template, like DNS, gateway and network mask.

The parameters used from the Virtual Network template are explained in the *Managing Virtual Networks section*.

User Credentials

One of the other very important things you have to configure is user credentials to connect to the newly created Virtual Machine. For linux base images we recommend to use SSH public key authentication and using it with OpenNebula is very convenient.

The first thing the users should do its to add their SSH public key (or keys) to its OpenNebula user configuration. This can be done in the Settings section of the web interface or using the command line interface:

```
$ oneuser update myusername
# an editor is opened, add this line
SSH_PUBLIC_KEY="ssh-rsa MYPUBLICKEY..."
```

Then in the Virtual Machine Template we add the option:

```
CONTEXT = [
    SSH_PUBLIC_KEY = "$USER[SSH_PUBLIC_KEY]"
]
```

Using this system the new Virtual Machines will be configured with the SSH public key of the user that instantiated it.

For Windows machines SSH is not available but you can use the options `USERNAME` and `PASSWORD` to create and set the password of an initial administrator.

```
CONTEXT = [
    USERNAME = "Administrator",
    PASSWORD = "VeryComplexPassw0rd"
]
```

Execute Scripts on Boot

To be able to execute commands on boot, for example, to install some software, you can use the option `START_SCRIPT`. When this option is used a new file that contains the value of the option will be created and executed.

For Windows machines this is a PowerShell script. For linux machines this can be any scripting language as long as it is installed in the base image and the proper shebang line is set (shell scripts don't need shebang).

In this example some commands will be executed using `bash` shell that will install the package `ntpdate` and set the time.

```
CONTEXT = [
    START_SCRIPT = "#!/bin/bash
yum update
yum install -y ntpdate
ntpdate 0.pool.ntp.org"
]
```

To add more complex scripts you can also use the option `START_SCRIPT_BASE64`. This option gets a base64 encoded string that will be decoded before writing the temporary script file.

Advanced Contextualization

There are more options that can be set in the contextualization section. You can read about them in the *Virtual Machine Definition File reference section*

5.3 vCenter Contextualization

5.3.1 Prepare the Virtual Machine Image

Step 1. Start a VM with the OS you want to Customize

Supported contextualization packages are available for the following OS's:

- **CentOS/RHEL** >= 6
- **Debian** >= 6
- **Ubuntu** >= 11.10
- **Windows** >= 7
- **Windows Server** >= 2008

If you already happen to have a VM or Template in vCenter with the installed OS you can start it and prepare it to be used with OpenNebula. Alternatively you can start an installation process with the OS media.

Step 2. Download Contextualization Packages to the VM

CentOS/RHEL

```
# wget https://github.com/OpenNebula/addon-context-linux/releases/download/v5.0.1/
↪one-context_5.0.1.rpm
```

Debian/Ubuntu

```
# wget https://github.com/OpenNebula/addon-context-linux/releases/download/v5.0.1/
↪one-context_5.0.1.deb
```

Windows

Downloads these two files to `C:\`:

- https://raw.githubusercontent.com/OpenNebula/addon-context-windows/master/context.ps1
- https://raw.githubusercontent.com/OpenNebula/addon-context-windows/master/startup.vbs

Step 3. Install Contextualization Packages and Dependencies

CentOS/RHEL 6

```
# rpm -Uvh one-context*rpm
# yum install -y epel-release
# yum install ruby # only needed for onegate command
# yum install -i dracut-modules-growroot
# dracut -f
```

CentOS/RHEL 7

```
# rpm -Uvh one-context*rpm
# yum install -y epel-release
# yum install ruby # only needed for onegate command
# yum install -y cloud-utils-growpart
```

Debian/Ubuntu

```
# dpkg -i one-context*deb
# apt-get install ruby # only needed for onegate command
# apt-get install -y cloud-utils
```

Windows

- Open the Local Group Policy Dialog by running `gpedit.msc`.
- Go to *Computer Configuration -> Windows Settings -> Scripts -> startup* (right click).
- Browse to the `startup.vbs` file and enable it as a startup script.

Step 4. Install VMware Tools

CentOS

```
# yum install open-vm-tools
```

Debian/Ubuntu

```
# apt-get install open-vm-tools
```

Windows

In vCenter open the VM menu, go to "Guest OS" section, click in "Install VMware Tools..." and follow the instructions.

Step 5. Power Off the Machine and Save it

These are the steps needed to finish the preparation and import it to OpenNebula:

- Power off the machine so it is in a consistent state the next time it boots
- Make sure that you take out any installation media used in the previous steps
- Remove the network interfaces from the VM
- Convert the VM into a Template
- Import the template in OpenNebula

This last step can be done using Sunstone going to Templates -> VMs and pressing the Import button. Alternatively you can also do it using the CLI:

```
$ onevcenter templates --vcenter vcenter.host --vuser vcenter@user --password the_
↪password
```

5.3.2 Set Up the Virtual Machine Template

The Virtual Machine Template has a section called context where you can automate different configuration aspects. The most common attributes are network configuration, user credentials and startup scripts. These parameters can be both added using the CLI to the template or using Sunstone Template wizard. Here is an example of the context section using the CLI:

```
CONTEXT = [
    TOKEN = "YES",
    NETWORK = "YES",
    SSH_PUBLIC_KEY = "$USER[SSH_PUBLIC_KEY]",
    START_SCRIPT = "yum install -y ntpdate"
]
```

In the example we are telling OpenNebula to:

- Set OneGate token and onegate information in the context
- Add network configuration to the Virtual Machine
- Enable login into the Virtual Machine using ssh with the value of the user's parameter `SSH_PUBLIC_KEY`
- On Virtual Machine boot execute the command `yum install -y ntpdate`

OneGate Token

OpenNebula has a centralized service to share data between Virtual Machines and the main daemon, useful to set monitoring information that can be gathered inside the VM and configuration data. It also lets you send scaling actions when the Virtual Machine belongs to a Service.

To do so the client installed with the contextualization packages (`onegate`) needs some information:

- **Token**: it's the key specific to each VM used to authenticate with the service
- **OneGate endpoint**: the address where the OneGate daemon is reachable

To fill this information you have to specify `TOKEN = "YES"` in the contextualization section.

Network Configuration

OpenNebula does not rely on a DHCP server to configure networking in the Virtual Machines. To do this configuration it injects the network information in the contextualization section. This is done with option `NETWORK = "YES"`. When OpenNebula finds this option it adds the IP information for each of the network interfaces configured plus extra information that resides in the Virtual Network template, like DNS, gateway and network mask.

The parameters used from the Virtual Network template are explained in the *Managing Virtual Networks section*.

User Credentials

One of the other very important things you have to configure is user credentials to connect to the newly created Virtual Machine. For linux base images we recommend to use SSH public key authentication and using it with OpenNebula is very convenient.

The first thing the users should do its to add their SSH public key (or keys) to its OpenNebula user configuration. This can be done in the Settings section of the web interface or using the command line interface:

```
$ oneuser update myusername
# an editor is opened, add this line
SSH_PUBLIC_KEY="ssh-rsa MYPUBLICKEY..."
```

Then in the Virtual Machine Template we add the option:

```
CONTEXT = [
    SSH_PUBLIC_KEY = "$USER[SSH_PUBLIC_KEY]"
]
```

Using this system the new Virtual Machines will be configured with the SSH public key of the user that instantiated it.

For Windows machines SSH is not available but you can use the options `USERNAME` and `PASSWORD` to create and set the password of an initial administrator.

```
CONTEXT = [
    USERNAME = "Administrator",
    PASSWORD = "VeryComplexPassw0rd"
]
```

Execute Scripts on Boot

To be able to execute commands on boot, for example, to install some software, you can use the option `START_SCRIPT`. When this option is used a new file that contains the value of the option will be created and executed.

For Windows machines this is a PowerShell script. For linux machines this can be any scripting language as long as it is installed in the base image and the proper shebang line is set (shell scripts don't need shebang).

In this example some commands will be executed using `bash` shell that will install the package `ntpdate` and set the time.

```
CONTEXT = [
    START_SCRIPT = "#!/bin/bash
yum update
yum install -y ntpdate
ntpdate 0.pool.ntp.org"
]
```

To add more complex scripts you can also use the option `START_SCRIPT_BASE64`. This option gets a base64 encoded string that will be decoded before writing the temporary script file.

Advanced Contextualization

There are more options that can be set in the contextualization section. You can read about them in the *Virtual Machine Definition File reference section*

5.4 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 tuning. 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 bootstrap 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.
- **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.

5.4.1 Adding External Images

You can use as basis for your images the ones provided by the distributions. These images are usually prepared to be used with other clouds and won't behave correctly or will not have all the features provided by OpenNebula. You can do a customization of these images before importing them.

To do this modification we are going to use the software libguestfs in a Linux machine with kvm support. You should use a modern distribution to have a recent version of libguestfs (>= 1.26). To have the latest version you can use Arch Linux but a CentOS 7 is OK.

Step 1. Install Libguestfs

The package is available in most distributions. Here are the commands to do it in some of them.

CentOS

```
# yum install libguestfs-tools
```

Debian/Ubuntu

```
# apt-get install libguestfs-tools
```

Arch Linux

This package is available in aur repository. You can either download the `PKGBUILD` and compile it manually or use a pacman helper like yaourt:

```
# yaourt -S libguestfs
```

Step 2. Download the Image

You can find the images for distributions in these links. We are going to use the ones from CentOS but the others are here for reference:

- **CentOS 7**: http://cloud.centos.org/centos/7/images/
- **Debian 8**: http://cdimage.debian.org/cdimage/openstack/current/
- **Ubuntu**: https://cloud-images.ubuntu.com/

Step 3. Download Context Packages

The context packages can be downloaded from the release section of the project. Make sure you download the version you need. For example, for CentOS download the *rpm* version. Also, don't download the packages marked with *ec2* as they are specific for EC2 images.

You have to download them to a directory that we will later refer. In this example it's going to be called `packages`.

```
$ mkdir packages
$ cd packages
$ wget https://github.com/OpenNebula/addon-context-linux/releases/download/v5.0.1/one-
↪context_5.0.1.rpm
$ wget https://github.com/OpenNebula/addon-context-linux/releases/download/v5.0.1/one-
↪context_5.0.1.deb
$ cd ..
```

Step 4. Create a CDROM Image with Context Packages

We will use this image as the source to install the context package. The image will be created with an specific label so later is easier to mount it. The label chosen is `PACKAGES`.

```
$ genisoimage -o packages.iso -R -J -V PACKAGES packages/
```

Step 5. Create a Script to Prepare the Image

The script will be different depending on the distribution and any extra steps we want to do to the image. The script will be executed in a chroot of the image root filesystem.

Here are some versions of the script for several distributions. The script will be called `script.sh`.

CentOS 6

```
mkdir /tmp/mount
mount LABEL=PACKAGES /tmp/mount

# Install opennebula context package
rpm -Uvh /tmp/mount/one-context*rpm

# Remove cloud-init and NetworkManager
yum remove -y NetworkManager cloud-init

# Install growpart and upgrade util-linux
yum install -y epel-release --nogpgcheck
yum install -y cloud-utils-growpart --nogpgcheck
yum upgrade -y util-linux --nogpgcheck

# Install ruby and rubygem-json for onegate
yum install -y ruby rubygem-json

# Install VMware tools. You can skip this step for KVM images
yum install -y open-vm-tools
```

CentOS 7

```
mkdir /tmp/mount
mount LABEL=PACKAGES /tmp/mount

# Install opennebula context package
rpm -Uvh /tmp/mount/one-context*rpm

# Remove cloud-init and NetworkManager
yum remove -y NetworkManager cloud-init

# Install growpart and upgrade util-linux
yum install -y epel-release --nogpgcheck
yum install -y cloud-utils-growpart --nogpgcheck
yum upgrade -y util-linux --nogpgcheck

# Install ruby for onegate tool
yum install -y ruby

# Install VMware tools. You can skip this step for KVM images
yum install -y open-vm-tools
```

Debian 8

```
# mount cdrom with packages
mkdir /tmp/mount
mount LABEL=PACKAGES /tmp/mount

# remove cloud-init and add one-context
dpkg -i /tmp/mount/one-context*deb
apt-get remove -y cloud-init


# This package contains growpart
apt-get install -y cloud-utils

# Unconfigure serial console. OpenNebula does not configure a serial console
# and growpart in initrd tries to write to it. It panics in the first boot
# if it is configured in the kernel parameters.
sed -i 's/console=ttyS0,115200//' /extlinux.conf
cat /extlinux.conf

# Install ruby for onegate tool
apt-get install -y ruby

# Install VMware tools. You can skip this step for KVM images
apt-get install -y open-vm-tools
```

Ubuntu 14.04

```
# mount cdrom with packages
mkdir /tmp/mount
mount LABEL=PACKAGES /tmp/mount

apt-key update
apt-get update

# remove cloud-init and add one-context
dpkg -i /tmp/mount/one-context*deb
apt-get remove -y cloud-init

# This package contains partx. Some old versions can not do online partition
# resizing
apt-get install -y util-linux

# This package contains growpart
apt-get install -y cloud-utils

# Install ruby for onegate tool
apt-get install -y ruby

# Install VMware tools. You can skip this step for KVM images
apt-get install -y open-vm-tools
```

Ubuntu 16.04

```
# mount cdrom with packages
mkdir /tmp/mount
mount LABEL=PACKAGES /tmp/mount

apt-key update
apt-get update

# remove cloud-init and add one-context
dpkg -i /tmp/mount/one-context*deb
apt-get remove -y cloud-init

# This package contains partx. Some old versions can not do online partition
# resizing
apt-get install -y util-linux

# This package contains growpart
apt-get install -y cloud-utils

# Install ruby for onegate tool
apt-get install -y ruby

# Take out serial console from kernel configuration. It prevents the
# image from booting.
sed -i 's/console=ttyS0$//g' /boot/grub/grub.cfg

# Install VMware tools. You can skip this step for KVM images
apt-get install -y open-vm-tools
```

Step 6. Create an Overlay Image

It's always a good idea to not modify the original image in case you want to use it again or something goes wrong with the process. To do it we can use `qemu-img` command:

```
$ qemu-img create -f qcow2 -b <original image> modified.qcow2
```

Step 7. Apply Customizations to the Image

Now we are going to execute `virt-customize` (a tool of libguestfs) to modify the image. This is the meaning of the parameters:

- `-v`: verbose output, in case we want to debug problems
- `--attach packages.iso`: add the CDROM image previously created with the packages
- `--format qcow2`: the image format is qcow2
- `-a modified.qcow2`: the disk image we want to modify
- `--run script.sh`: script with the instructions to modify the image
- `--root-password disabled`: deletes root password. In case you want to set a password (for debugging) use `--root-password password:the-new-root-password`

```
$ virt-customize -v --attach packages.iso --format qcow2 -a modified.qcow2 --run␣
↪script.sh --root-password disabled
```

Step 8. Convert the Image to the Desired Format

After we are happy with the result we can convert the image to the preferred format to import to OpenNebula. Even if we want a `qcow2` image we have to convert it to consolidate all the layers in one file. For example, to create a `qcow2` image that can be imported to fs (ssh, shared and qcow2), ceph and fs_lvm datastores we can execute this command:

```
$ qemu-img convert -O qcow2 modified.qcow2 final.qcow2
```

To create a vmdk image, for vCenter hypervisors we can use this other command:

```
$ qemu-img convert -O vmdk modified.qcow2 final.vmdk
```

Step 9. Upload it to an OpenNebula Datastore

You can now use Sunstone to upload the final version of the image or copy it to the frontend and import it. If you are going to use the second option make sure that the image is in a directory that allows image imports (by default `/var/tmp`). For example:

```
$ oneimage create --name centos7 --path /var/tmp/final.qcow2 --driver qcow2 --prefix␣
↪vd --datastore default
```

5.4.2 Install within OpenNebula

If you are using KVM hypervisor you can do the installations using OpenNebula. Here are the steps to do it:

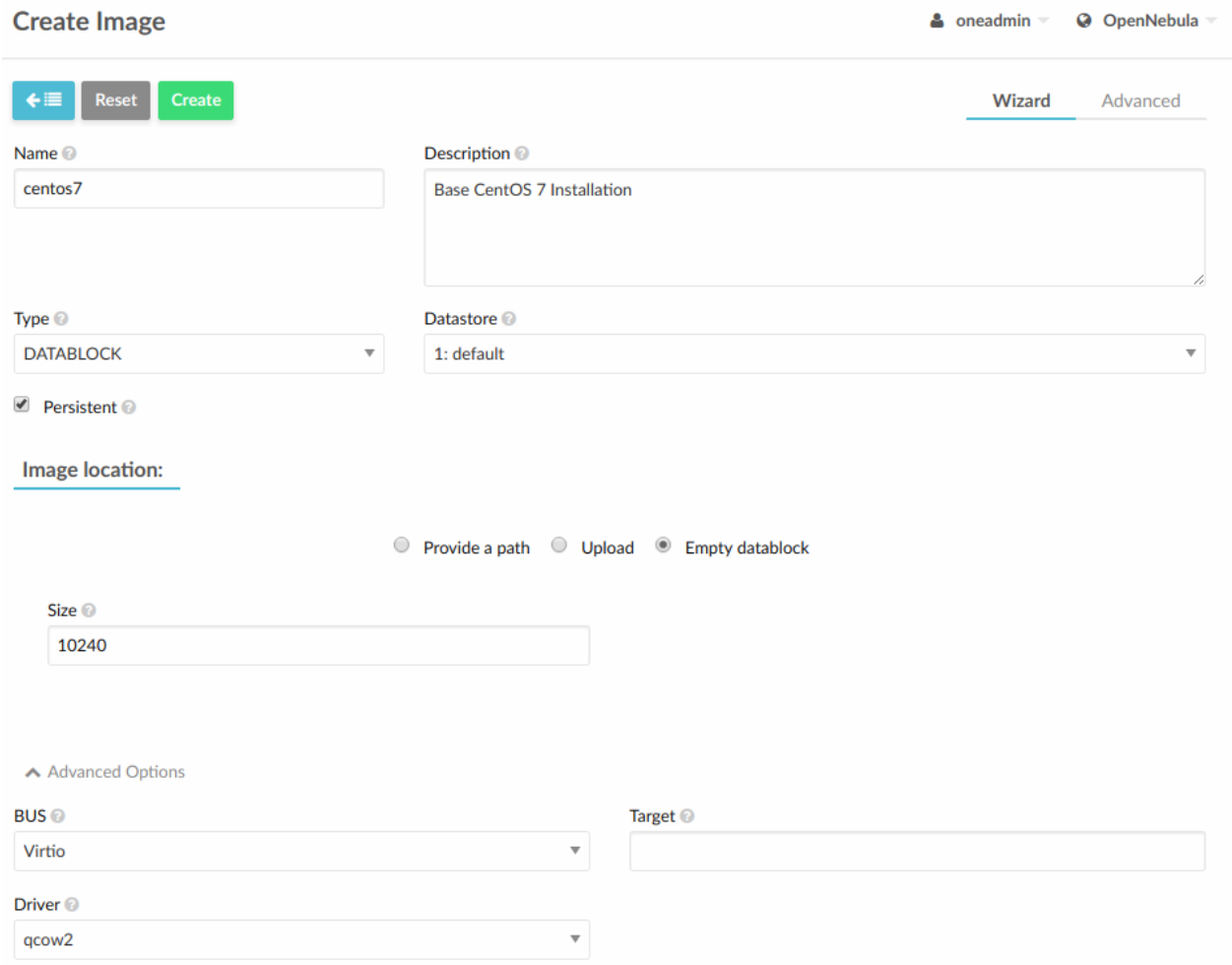
Step 1. Add the Installation Medium

You can add the installation CD to OpenNebula uploading the image using Sunstone and setting its type to CDROM or using the command line. For example, to add the CentOS ISO file you can use this command:

```
$ oneimage create --name centos7-install --path http://buildlogs.centos.org/rolling/7/
↪isos/x86_64/CentOS-7-x86_64-DVD.iso --type CDROM --datastore default
```

Step 2. Create Installation Disk

The disk where the OS will be installed needs to be created as a DATABLOCK. Don't make the image too big as it can be resized afterwards on VM instantiation. Also make sure to make it persistent so we don't lose the installation when the Virtual Machine terminates.



If you are using the CLI you can do the same with this command:

```
$ oneimage create --name centos7 --description "Base CentOS 7 Installation" --type␣
↪DATABLOCK --persistent --prefix vd --driver qcow2 --size 10240 --datastore default
```

Step 3. Create a Template to do the Installation

In this step you have to take the following into account:

- Add first the persistent datablock and second the installation media in the storage tab
- Add a network as it will be needed to download context packages
- On OS Booting tab enable both disks for booting. The first time it will use the CD and after installing the OS the DATABLOCK will be used
- In Input/Output tab enable VNC and add as input an USB Tablet. This will be useful in case the OS has a graphical installation

This can be done with the CLI using this command:

```
$ onetemplate create --name centos7-cli --cpu 1 --memory 1G --disk centos7,centos7-
↪install --nic network --boot disk0,disk1 --vnc --raw "INPUT=[TYPE=tablet,BUS=usb]"
```

Now instantiate the template and do the installation using the VNC viewer. Make sure that you configure the network manually as there are no context packages in the installation media. Upon completion tell the instanter to reboot the machine, log into the new OS and follow the instructions from the accompanying sections to install the contextualization.

As a tip, one of the latest things you should do when using this method is disabling `root` password and deleting any extra users that the install tool has created.

Step 4. Shutdown the Machine and Configure the Image

You can now shutdown the Virtual Machine from inside, that is, use the OS to shutdown itself. When the machine appears as poweroff in OpenNebula terminate it.

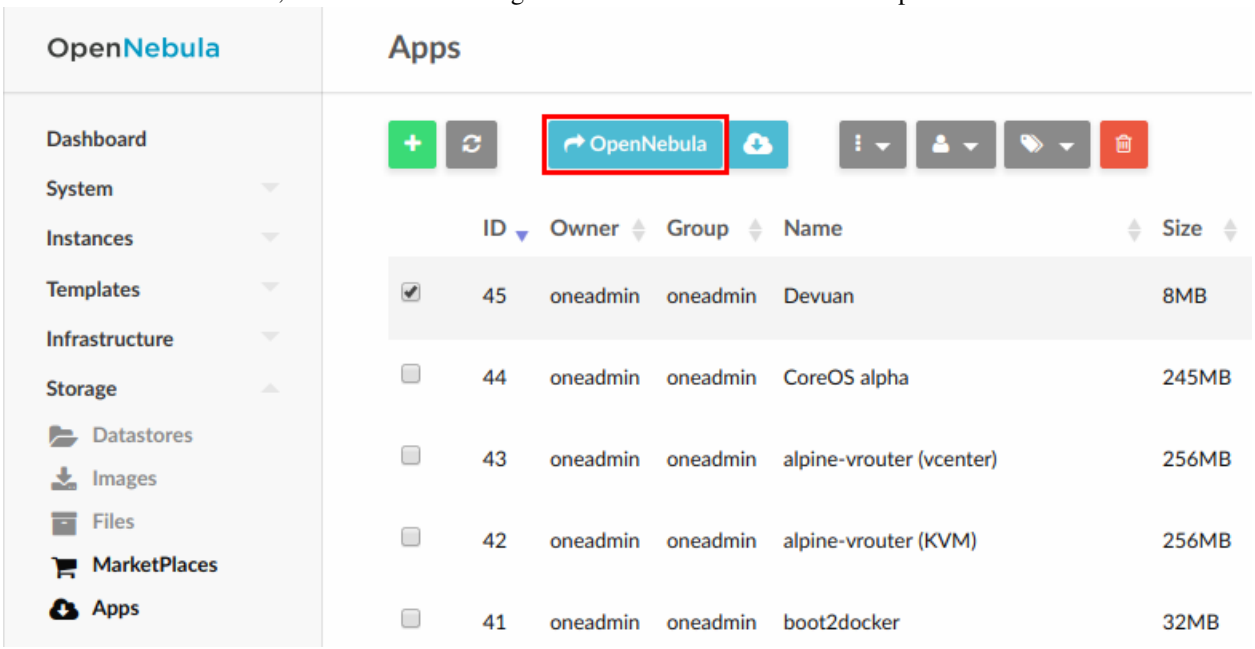
Make sure that you change the image to non persistent and you give access to other people.

Using the CLI you can do:

```
$ oneimage nonpersistent centos7
$ oneimage chmod centos7 744
```

5.4.3 Use the OpenNebula Marketplace

If your frontend is connected to the internet it should have access to the public OpenNebula Marketplace. In it there are several images prepared to run in an OpenNebula Cloud. To get images from it you can go to the Storage/Apps tab in Sunstone web interface, select one of the images and click the button "<arrow> OpenNebula":



Using the CLI we can list an import using these commands:

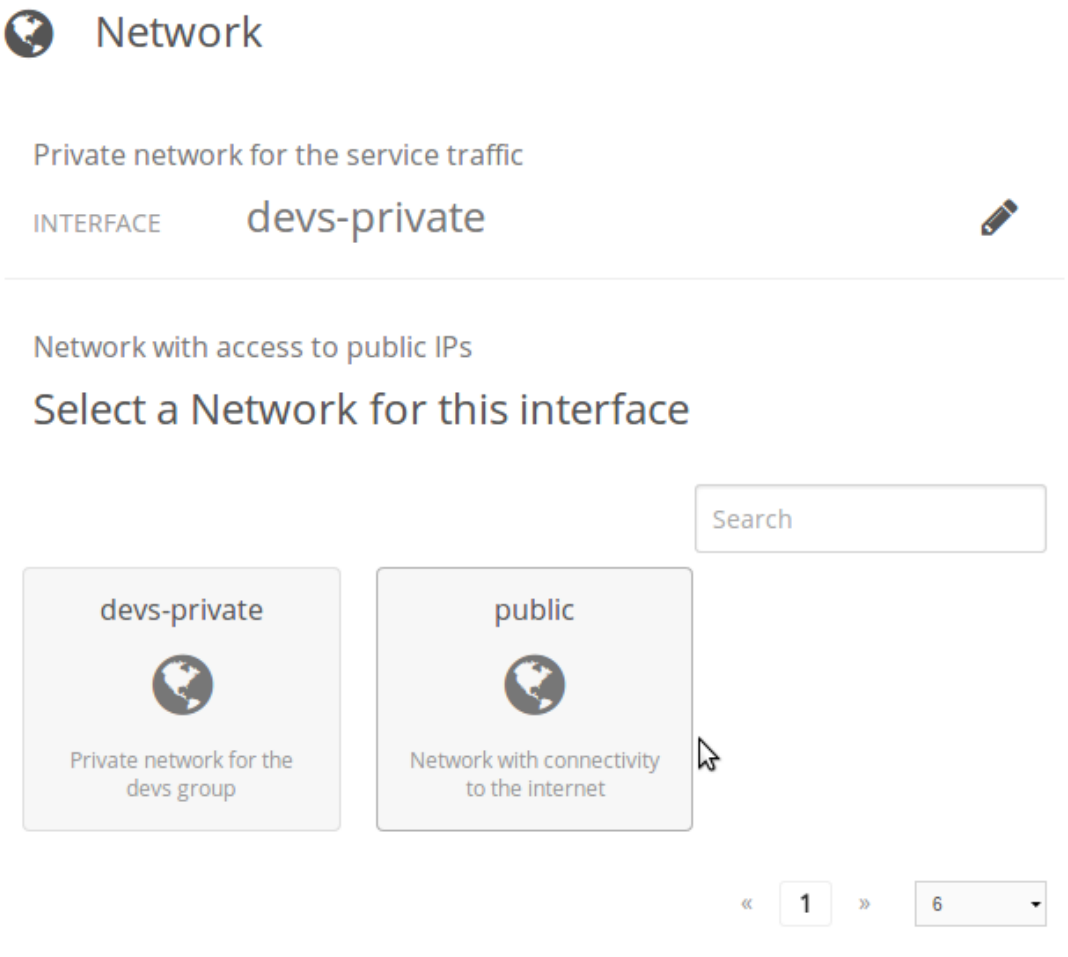
```
$ onemarketapp list
  ID NAME                            VERSION   SIZE STAT TYPE  REGTIME MARKET
↪   ZONE
[...]
  41 boot2docker                      1.10.2    32M  rdy  img 02/26/16 OpenNebula Public
↪      0
  42 alpine-vrouter (KVM)             1.0.3   256M  rdy  img 03/10/16 OpenNebula Public
↪      0
```

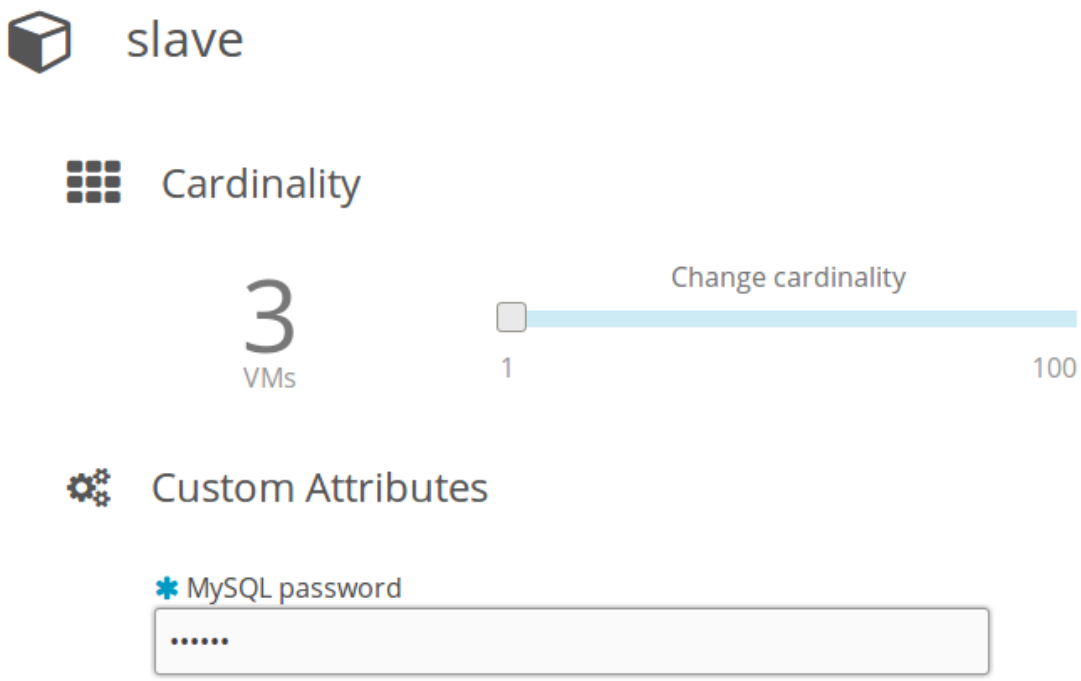
```
  43 alpine-vrouter (vcenter)          1.0  256M  rdy  img 03/10/16 OpenNebula Public
↪      0
  44 CoreOS alpha                 1000.0.0  245M  rdy  img 04/03/16 OpenNebula Public
↪      0
  45 Devuan                       1.0 Beta    8M  rdy  img 05/03/16 OpenNebula Public
↪      0
$ onemarketapp export Devuan Devuan --datastore default
IMAGE
        ID: 12
VMTEMPLATE
        ID: -1
```

5.4.4 How to Prepare the Service Templates

When you prepare a OneFlow Service Template to be used by the Cloud View users, take into account the following:

- You can define dynamic networks in the Service Template, to allow users to choose the virtual networks for the new Service instance.
- If any of the Virtual Machine Templates used by the Roles has User Inputs defined (see the section above), the user will be also asked to fill them when the Service Template is instantiated.
- Users will also have the option to change the Role cardinality before the Service is created.


Network
Private network for the service traffic
INTERFACE
devs-private
Network with access to public IPs
Select a Network for this interface
Search
devs-private
Private network for the devs group
public
Network with connectivity to the internet
«
1
»
6



To make a Service Template available to other users, you have two options:

- Change the Template's group, and give it `GROUP USE` permissions. This will make the Service Template only available to users in that group.
- Leave the Template in the oneadmin group, and give it `OTHER USE` permissions. This will make the Service Template available to every user in OpenNebula.

Please note that you will need to do the same for any VM Template used by the Roles, and any Image and Virtual Network referenced by those VM Templates, otherwise the Service deployment will fail.

CHAPTER

SIX

CLOUD END-USER

6.1 Overview

This chapter contains reference guides for Sunstone end-users.

6.1.1 How Should I Read This Chapter

The following sections are intended for the cloud consumers. They can skip most of the OpenNebula documentation and read these two guides only.

Procceed to the corresponding guide following these links:

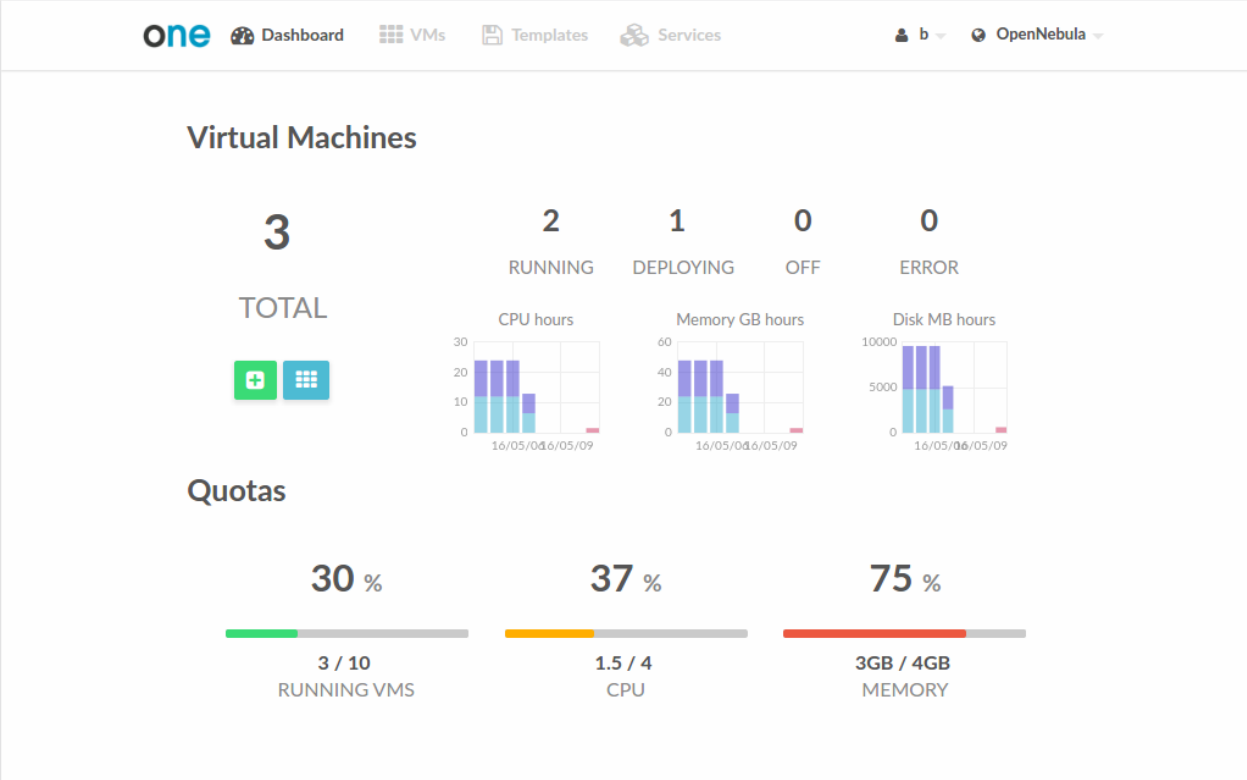
- *Self-service Cloud View*: For cloud consumers that just require a portal where they can provision new virtual machines and services easily.
- *Group Admin View*: For group administrators. This view allows the management of the group's resources, including the creation of new users.

6.1.2 Hypervisor Compatibility

Sunstone is available for all the hypervisors. When using vCenter, the cloud admin should enable the `groupadmin_vcenter` and `cloud_vcenter` Sunstone views.

6.2 Self-service Cloud View

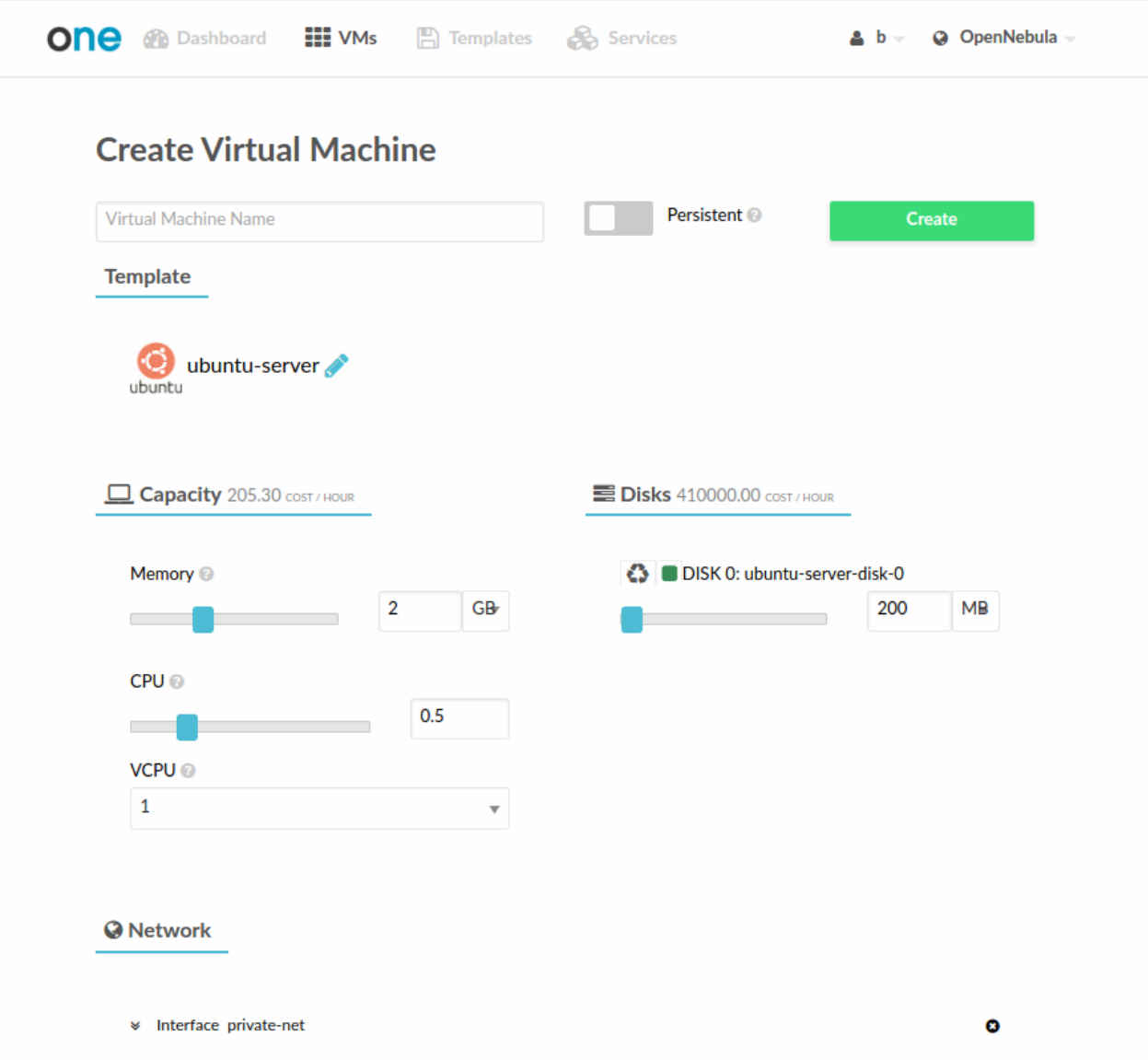
This is a simplified view intended for cloud consumers that just require a portal where they can provision new virtual machines easily. To create new VMs and Services, they just have to select one of the available templates prepared by the administrators.



6.2.1 Using the Cloud

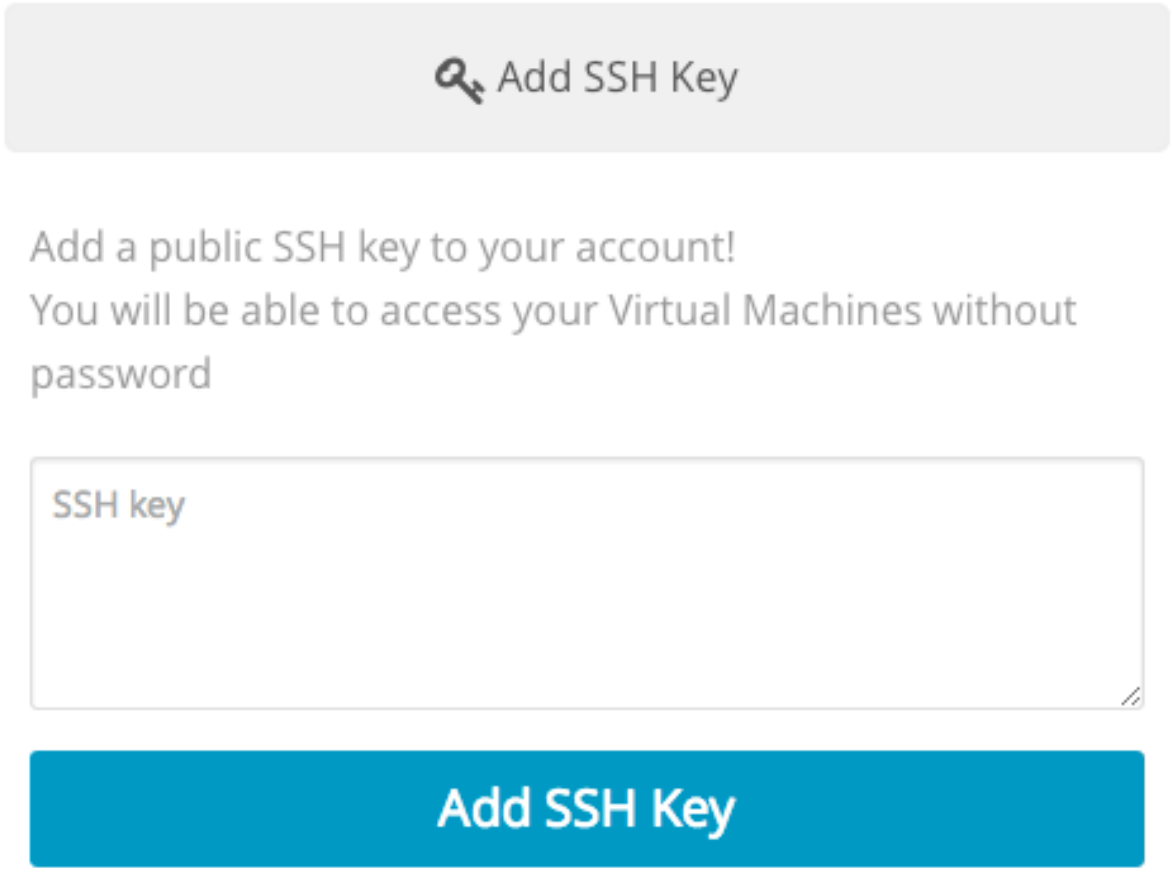
Create VM

In this scenario the cloud administrator must prepare a set of templates and images and make them available to the cloud users. These Templates must be ready to be instantiated, i.e. they define all the mandatory attributes. Before using them, users can optionally customize the VM capacity, resize disks, add new network interfaces and provide values required by the template. Read *Adding Content to Your Cloud* for more information.



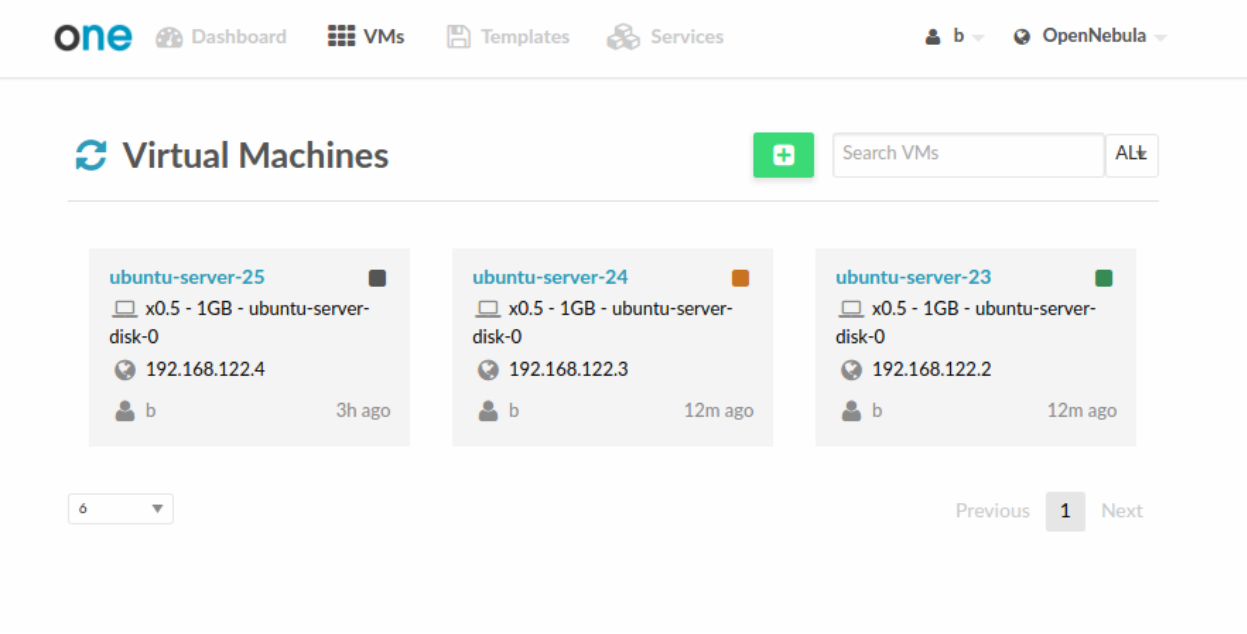
Access the VMs with SSH Keys

Any user can provide his own ssh public key to be included in the VMs created through this view. Note that the template has to be configured to include it.

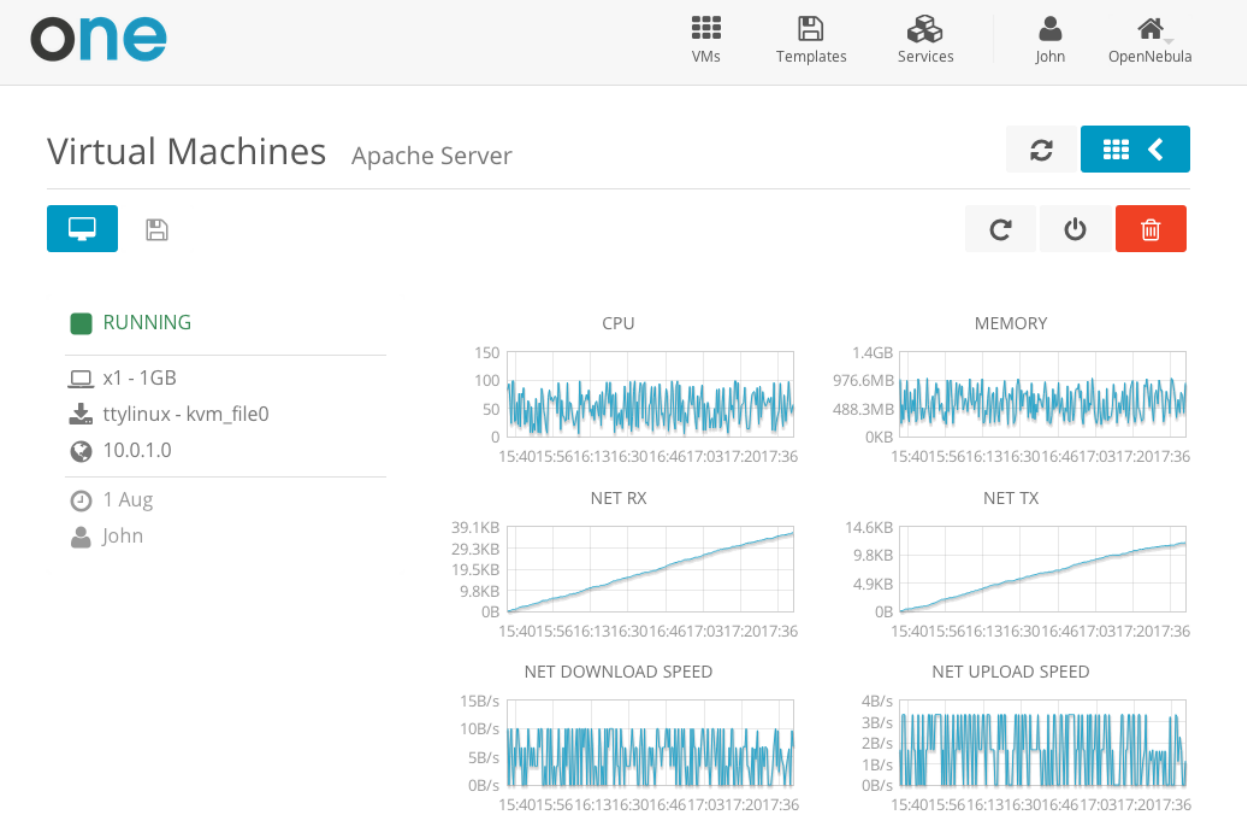


Manage VMs

The status of the VMs can be monitored from the VMs tab.



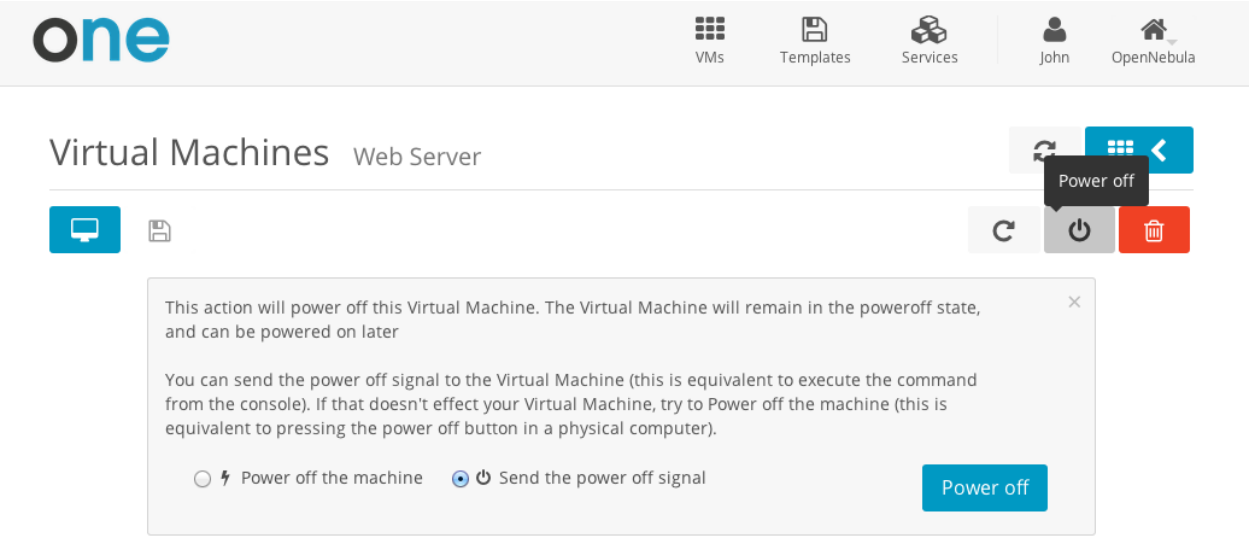
Information about the capacity, OS, IPs, creation time and monitoring graphs for a specific VM are available in the detailed view of the VM



A user can perform the following actions from this view:

- Access the VNC console, note that the Template has to be configured for this
- Reboot the VM, the user can send the reboot signal (reboot) or reboot the machine (reboot hard)

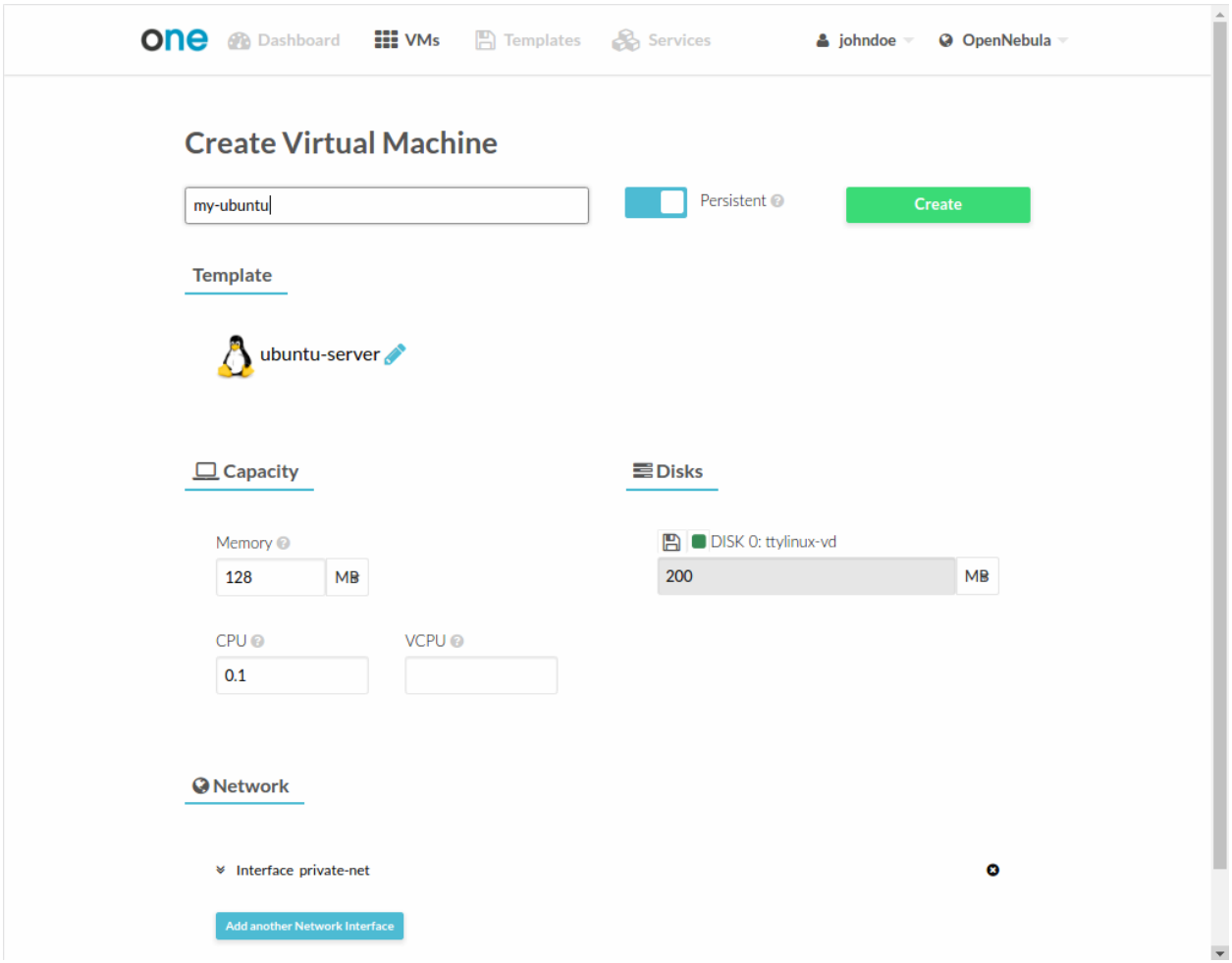
- Power off the VM, the user can send the power off signal (poweroff) or power off the machine (poweroff hard)
- Terminate the VM
- Save the VM into a new Template
- Power on the VM



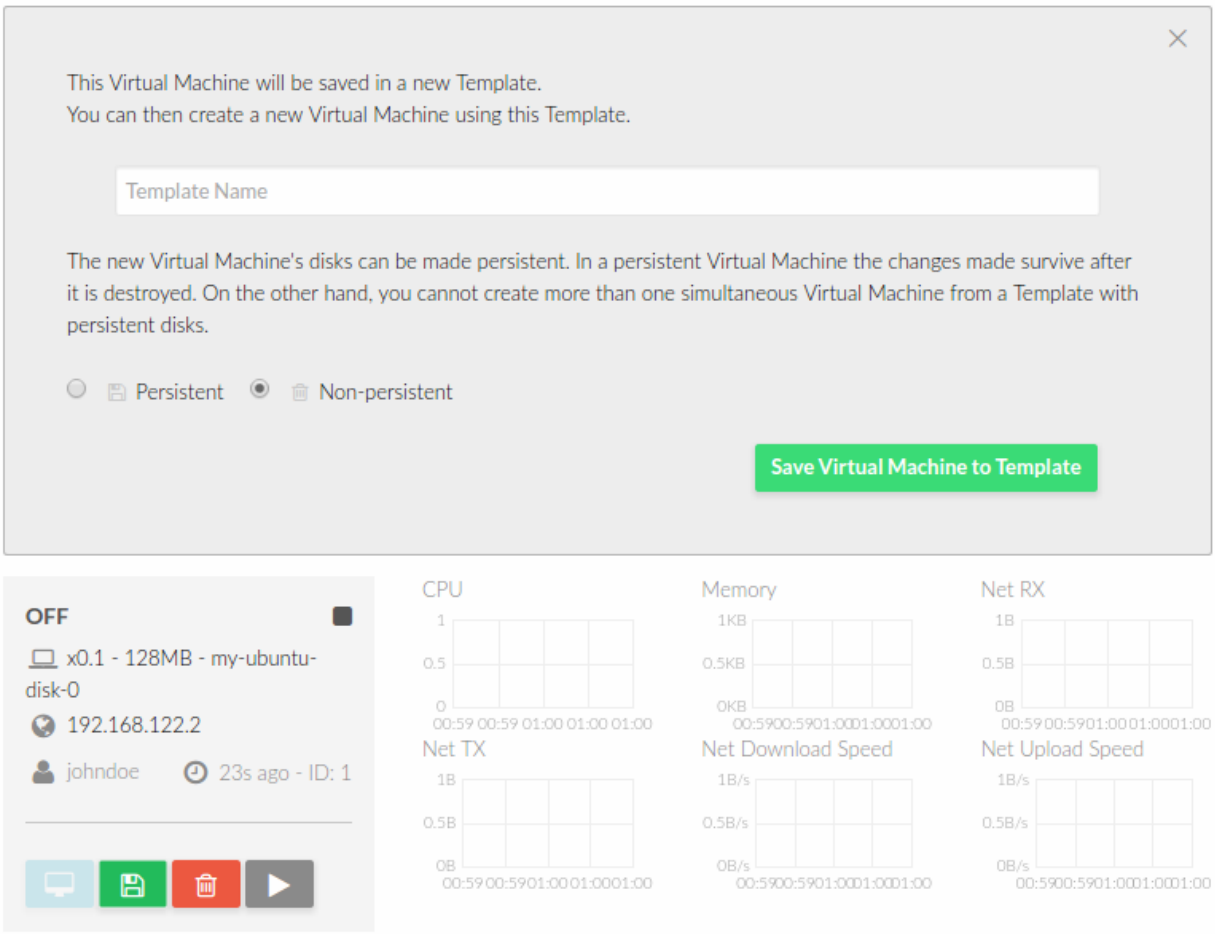
Make the VM Changes Persistent

Users can create a persistent private copy of the available templates. A persistent copy will preserve the changes made to the VM disks after the instance is terminated. This template is private, and will only be listed to the owner user.

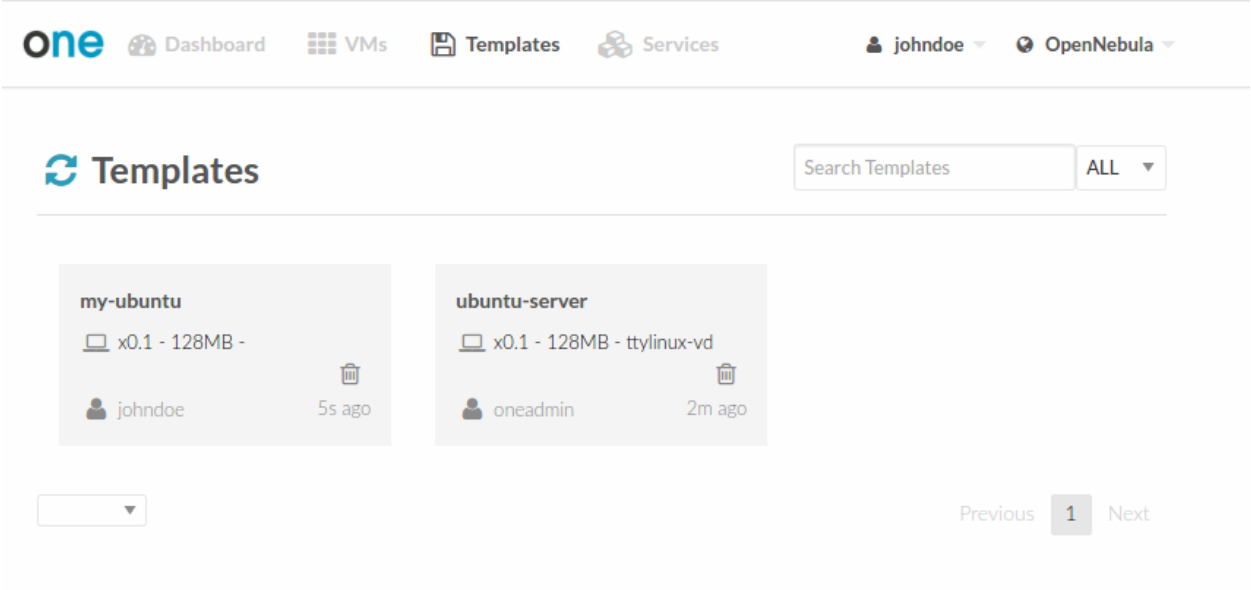
To create a persistent copy, use the “Persistent” switch next to the create button:



Alternatively, a VM that was not created as persistent can be saved before it is destroyed. To do so, the user has to power off the VM first and then use the save operation.



Any of the these two actions will create a new Template with the VM name. This template can be used in the "new VM wizard" to restore the VM after it is terminated. This template contains a copy of each one of the original disk images. If you delete this template, all the disk contents will be also lost.

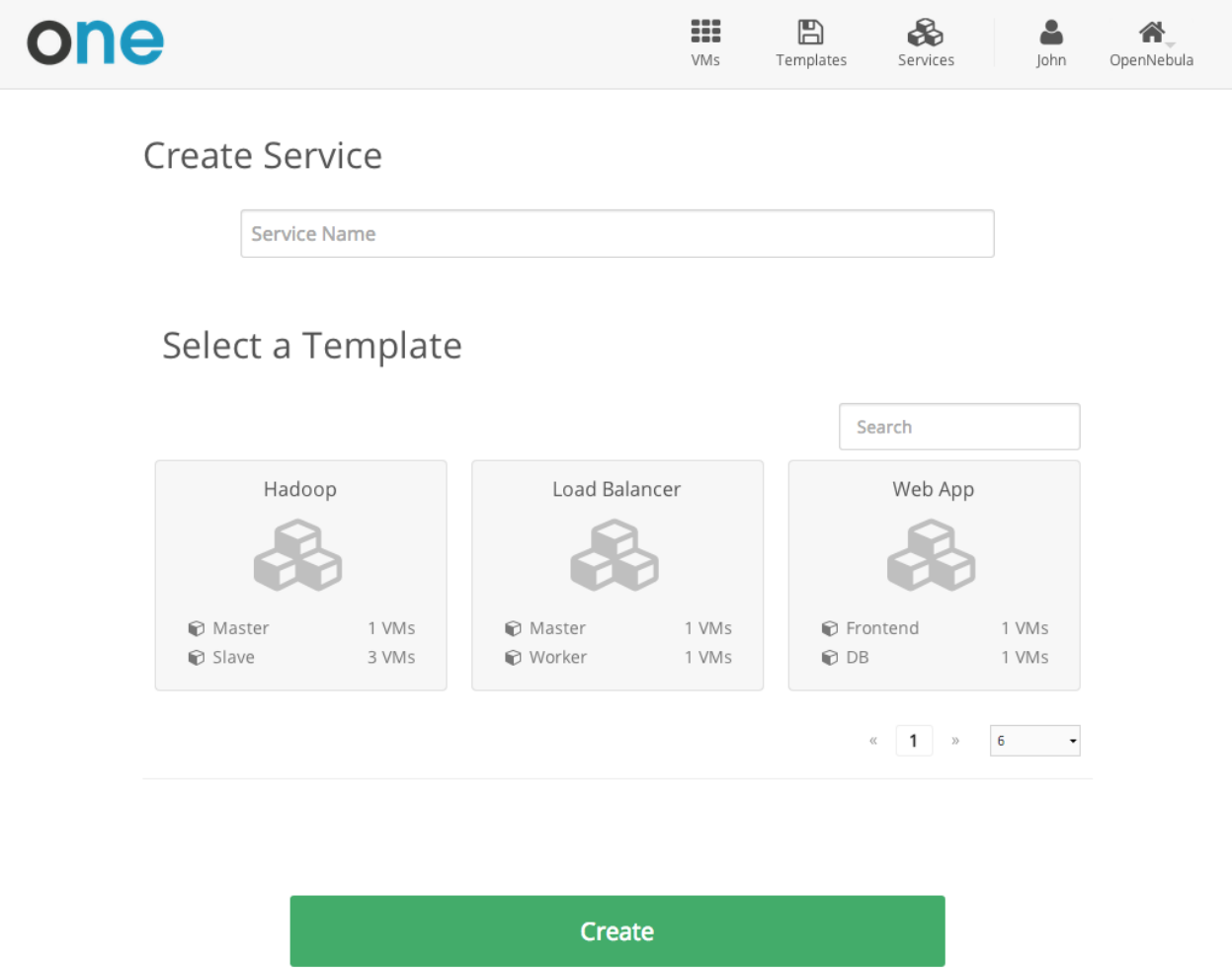


Note: Avoid making a persistent copy of a persistent copy! Although there are use cases where it is justified, you will end with a long list of Templates and the disk usage quota will decrease quickly.

For more details about the limitations of saved VM, continue to the *Managing Virtual Machines guide*.

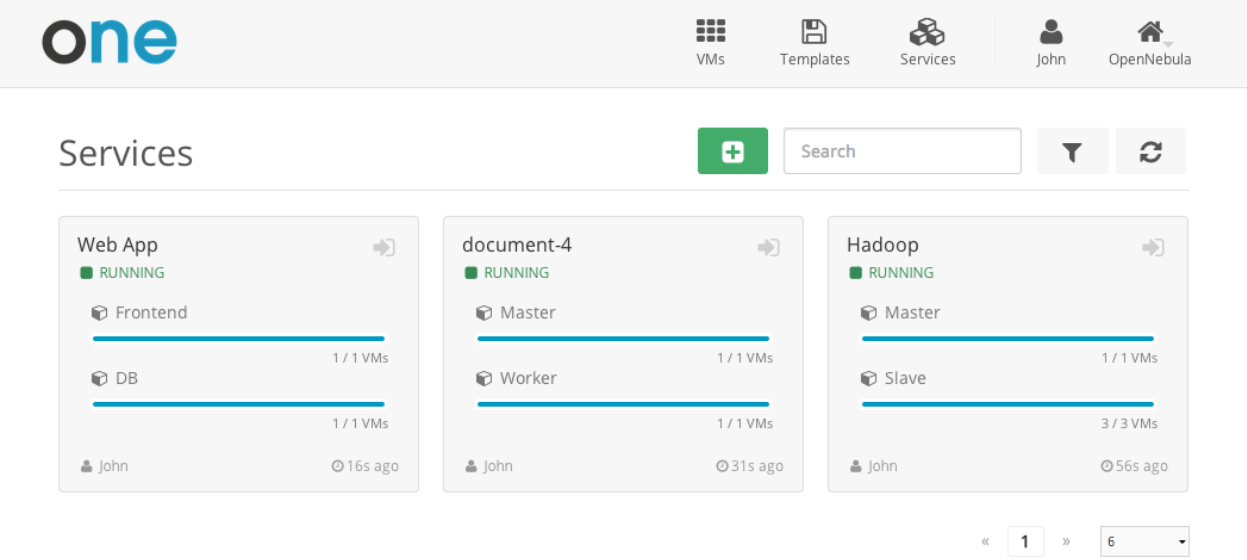
Create Service

In this scenario the cloud administrator must prepare a set of Service templates and make them available to the cloud users. These Service templates must be ready to be instantiated, i.e. they define all the mandatory attributes and the templates that are referenced are available for the user. Before using them, users can optionally customize the Service cardinality, define the network interfaces and provide values required by the template. Read *Adding Content to Your Cloud* for more information.

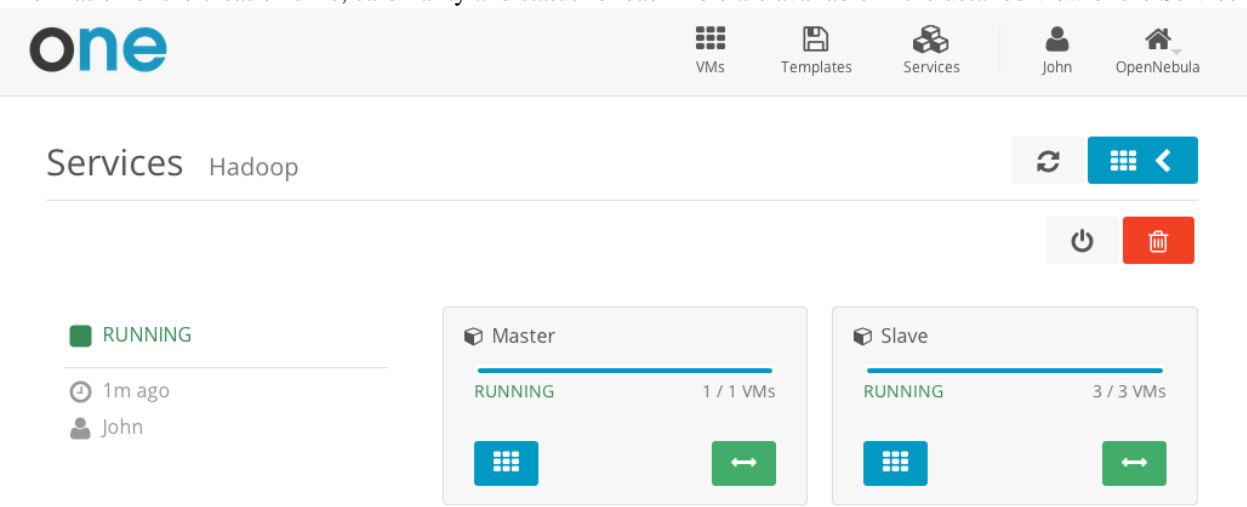


Manage Services

The status of the Services can be monitored from the Services tab



Information of the creation time, cardinality and status for each Role are available in the detailed view of the Service

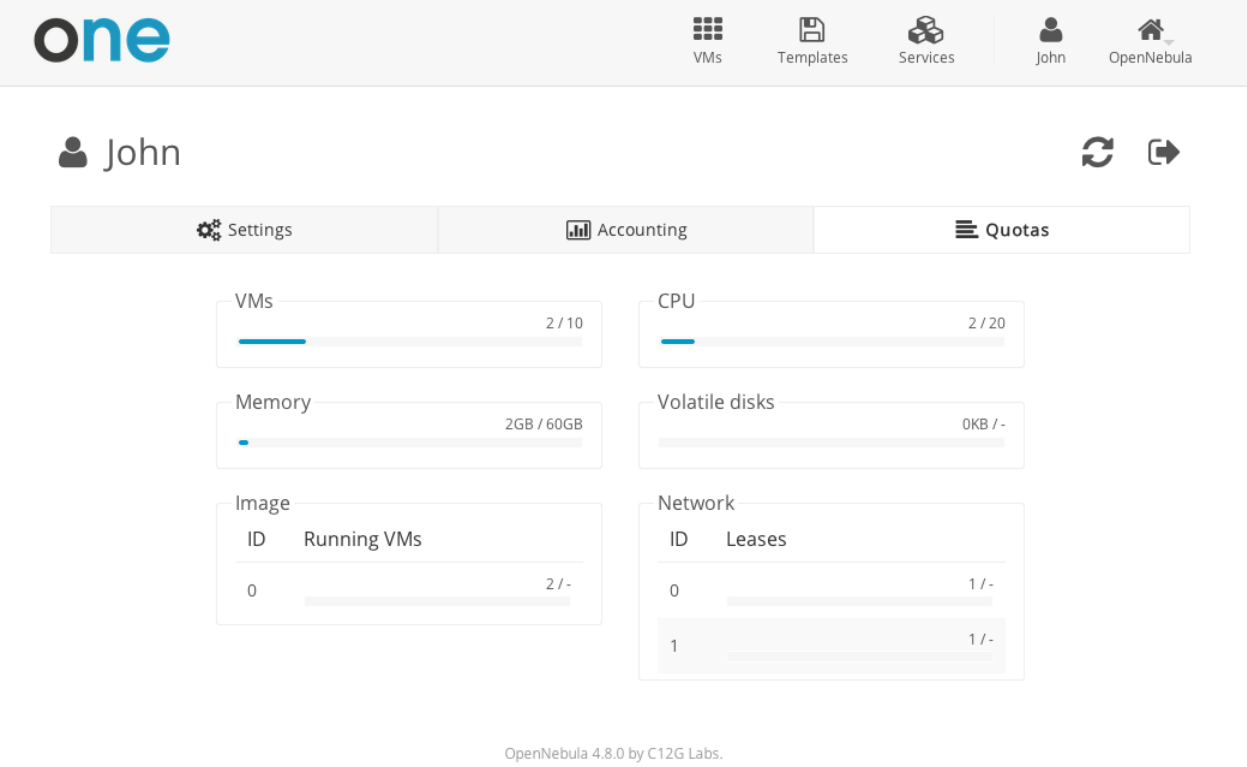


A user can perform the following actions from this view:

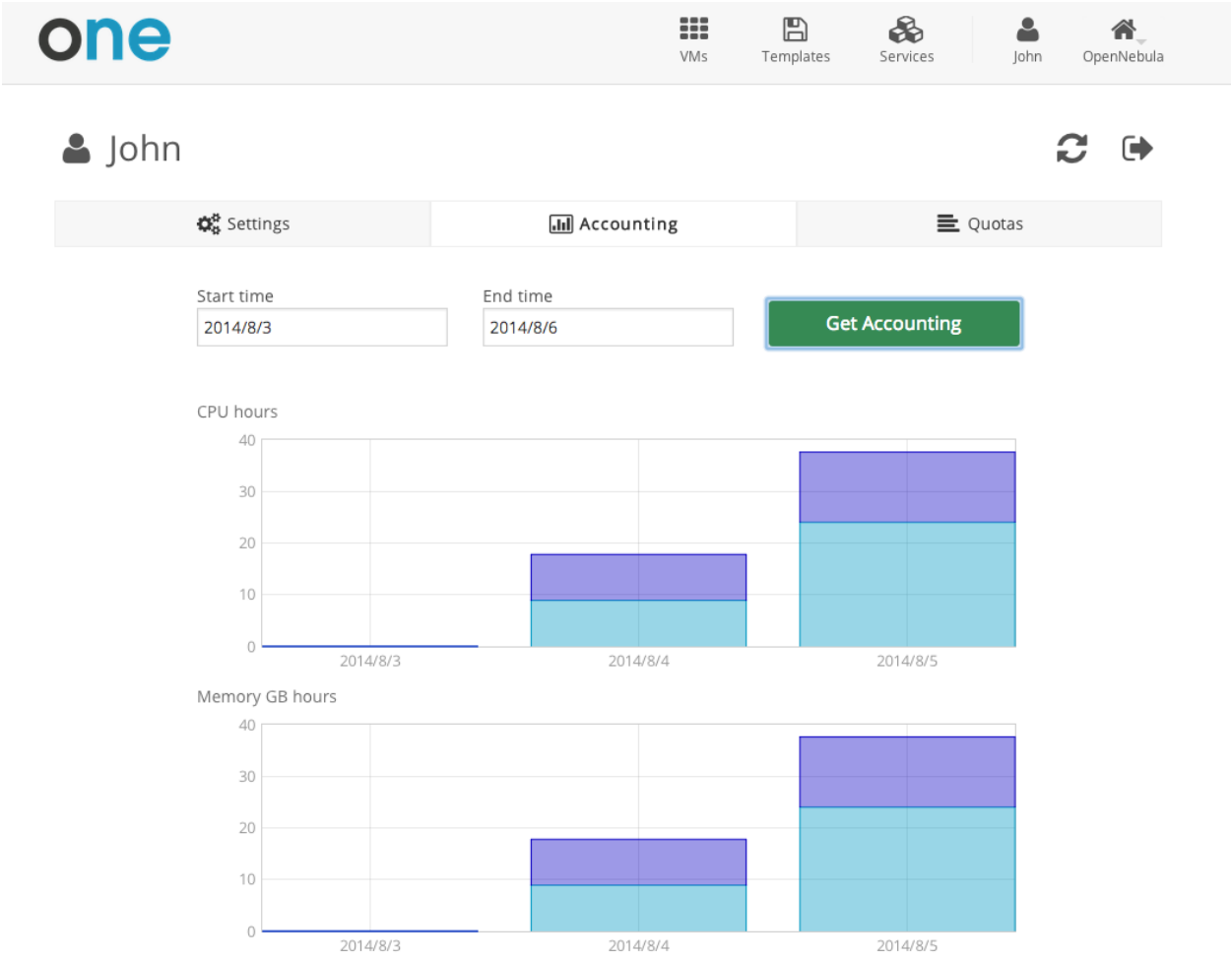
- Change the cardinality of each Role
- Retrieve the VMs of each Role
- Delete the Service
- Recover the Service from a fail status

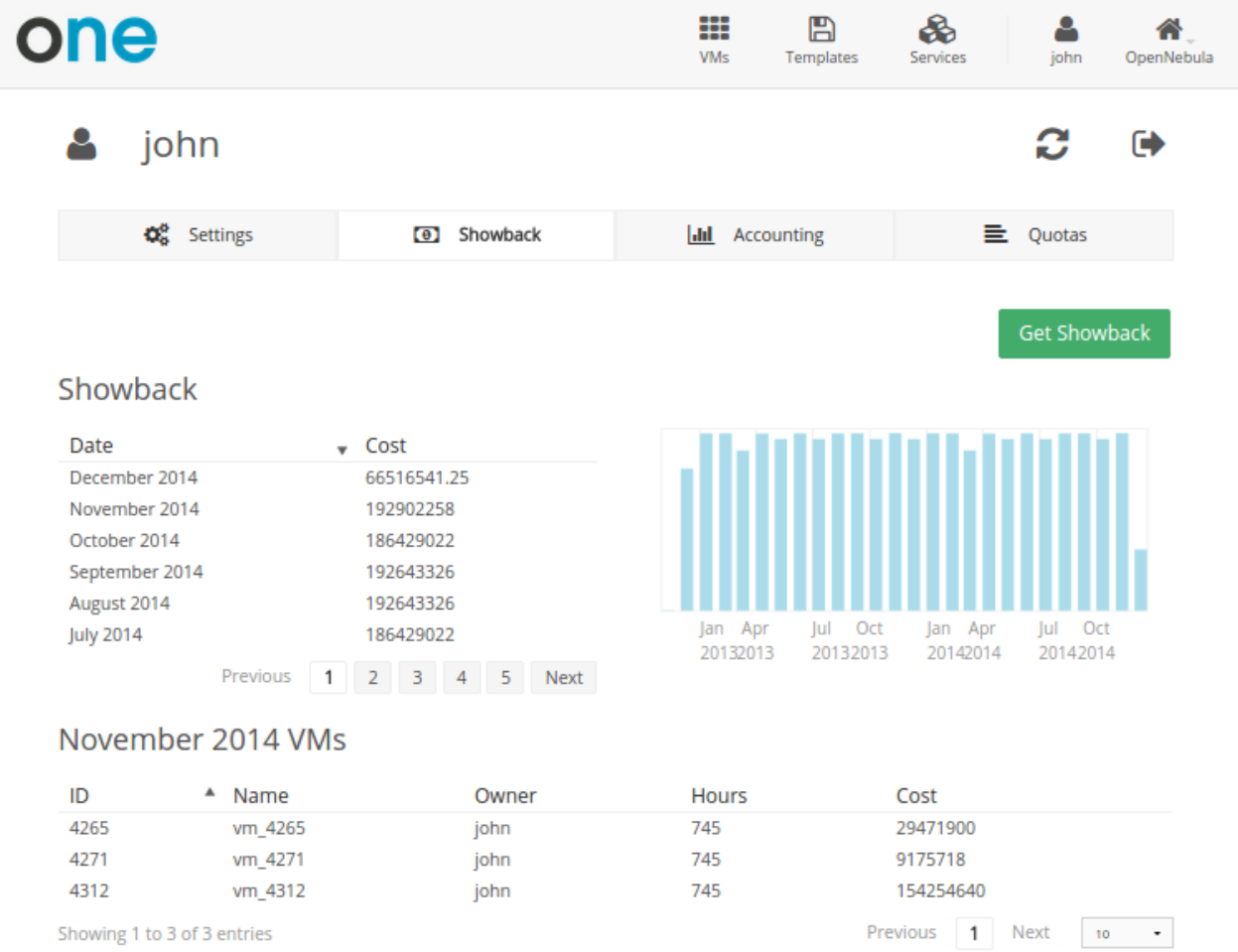
Usage, Accounting and Showback

The user can check his current usage and quotas



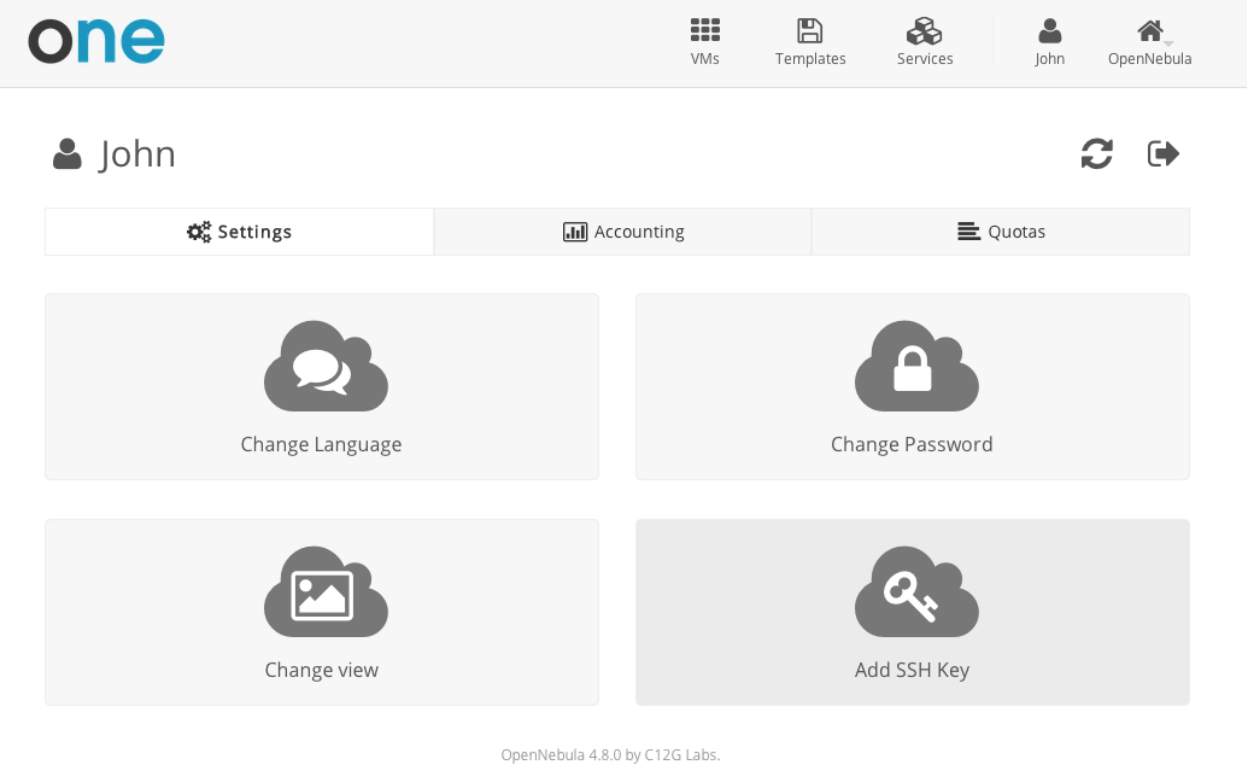
Also, the user can generate accounting reports for a given range of time


one
VMs
Templates
Services
John
OpenNebula
John
Settings
Accounting
Quotas
Start time
2014/8/3
End time
2014/8/6
Get Accounting
CPU hours
40
30
20
10
0
2014/8/3
2014/8/4
2014/8/5
Memory GB hours
40
30
20
10
0
2014/8/3
2014/8/4
2014/8/5



User Settings

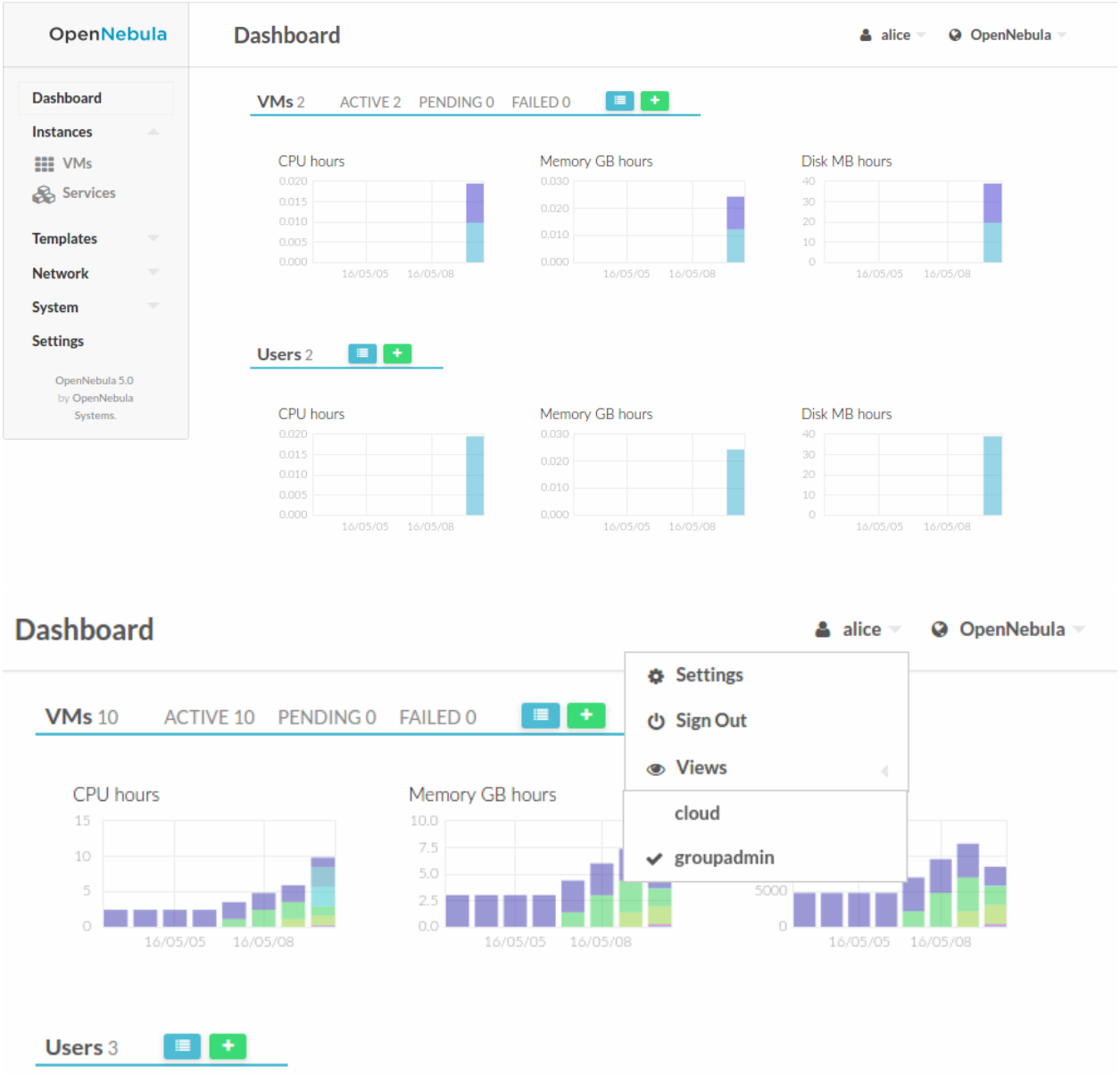
From the user settings tab, the user can change his password, language, ssh key and view



6.3 Group Admin View

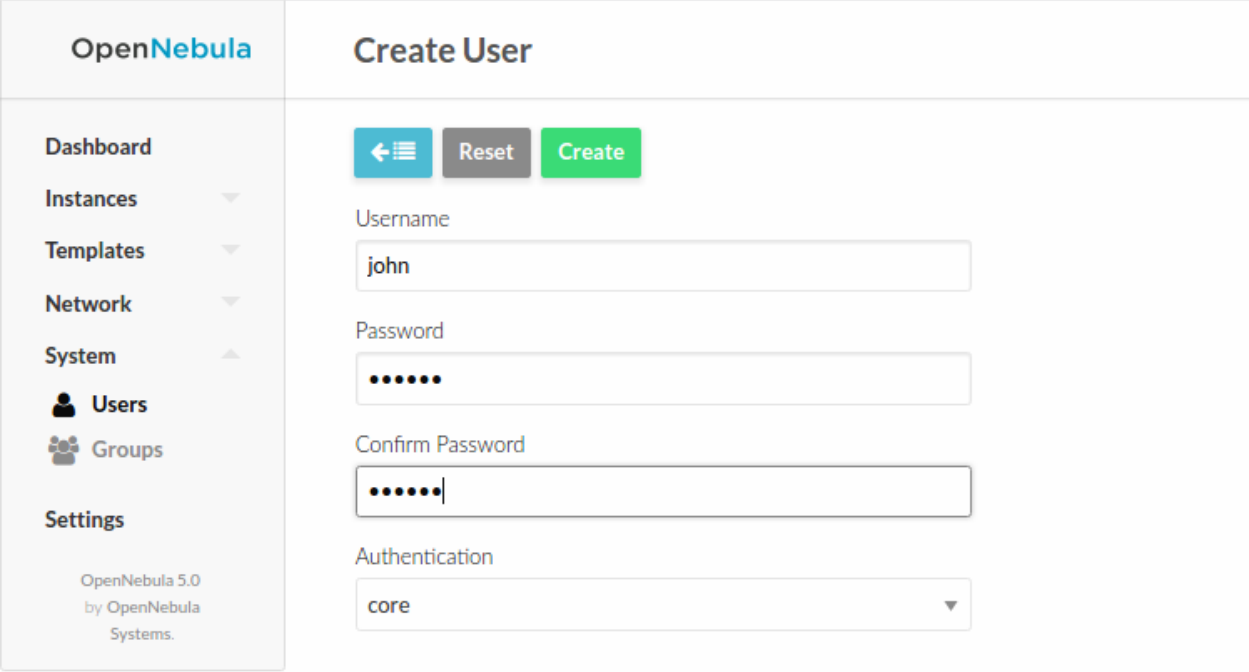
The role of a Group Admin is to manage all the virtual resources of the Group, including the creation of new users. When one of these Group Admin users access Sunstone, they get a limited version of the cloud administrator view. You can read more about OpenNebula's approach to Groups and VDC's from the perspective of different user roles in the Understanding OpenNebula guide.

Group administrators can also access the *simplified Cloud View* if they prefer to.

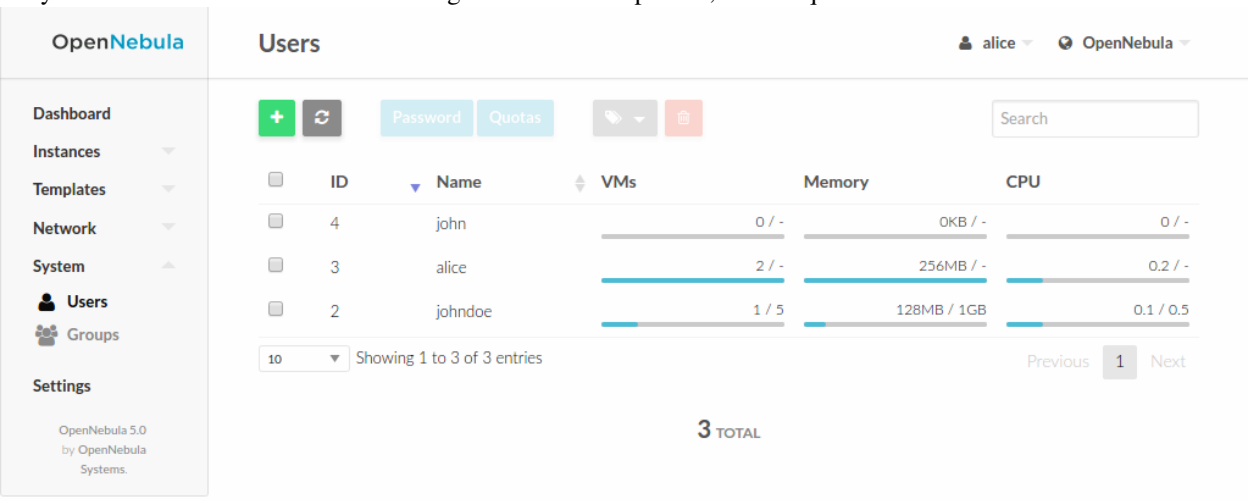


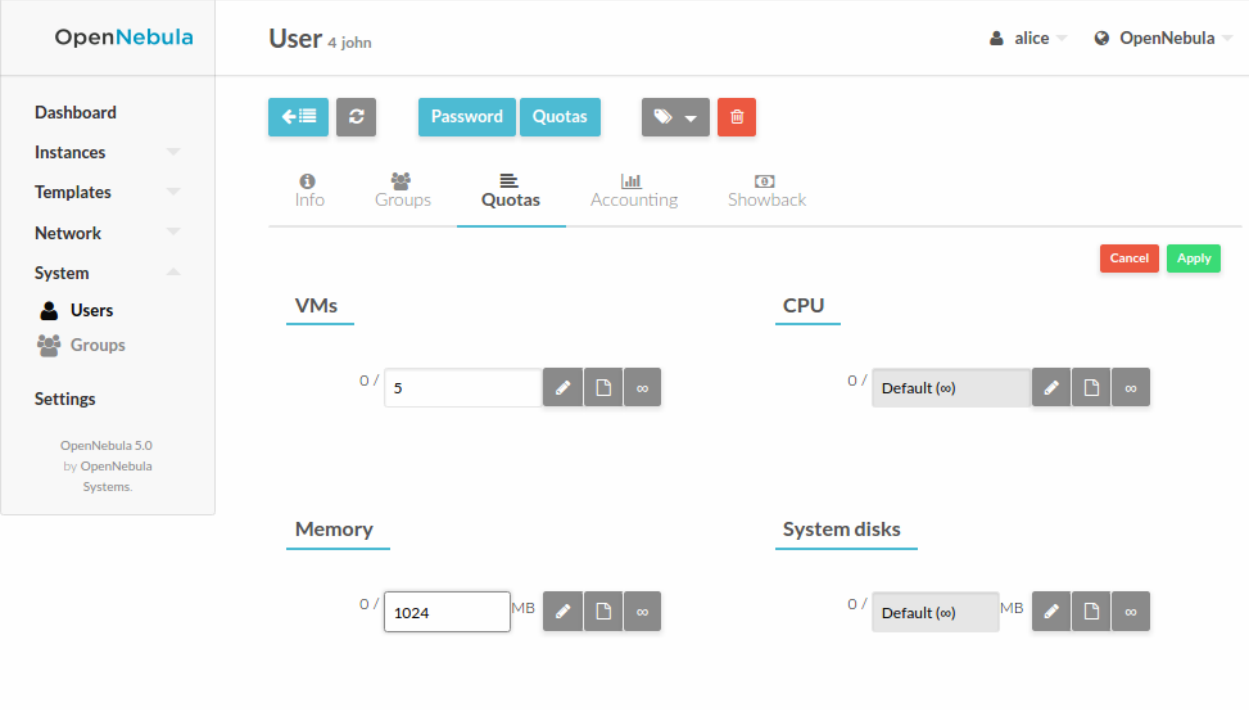
6.3.1 Manage Users

The Group Admin can create new user accounts, that will belong to the same Group.



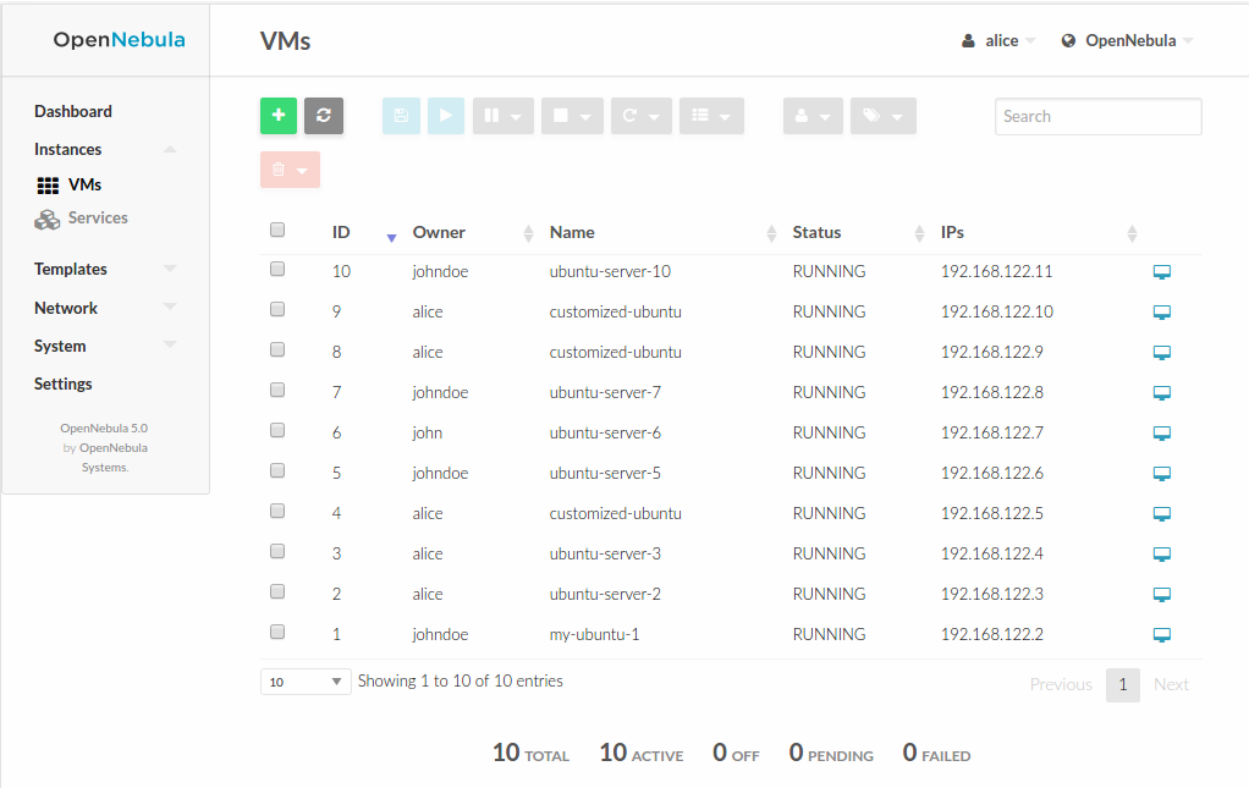
They can also see the current resource usage of all the Group users, and set quota limits for each one of them.





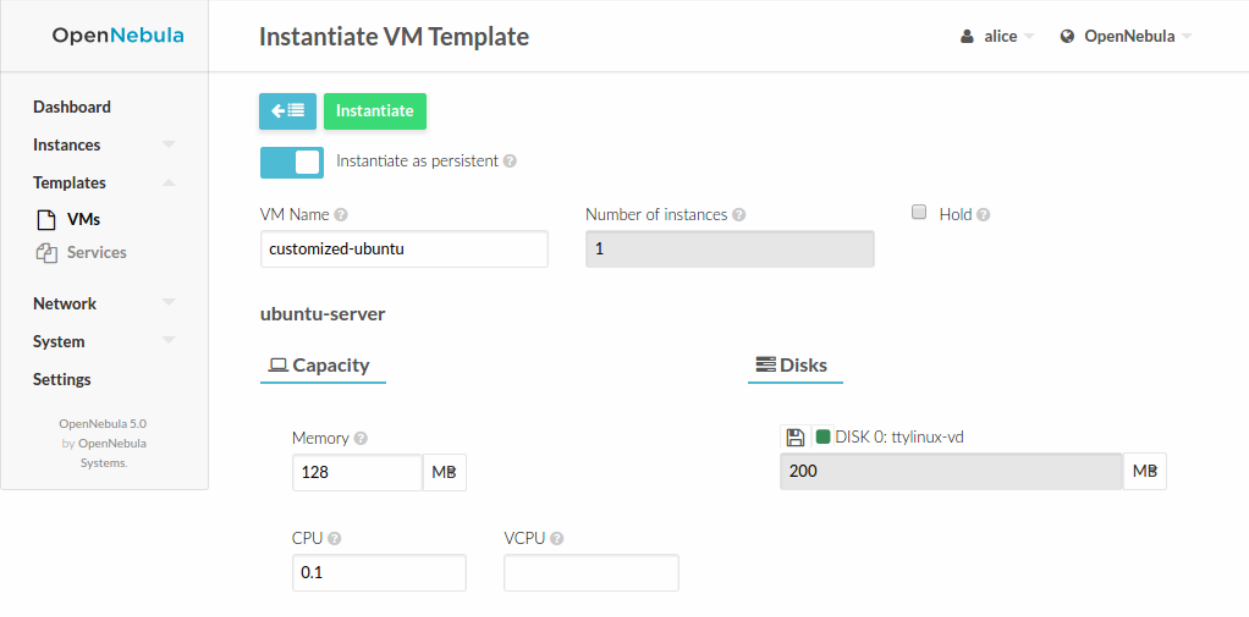
6.3.2 Manage Resources

The Group admin can manage the Services, VMs and Templates of other users in the Group.



6.3.3 Create Resources

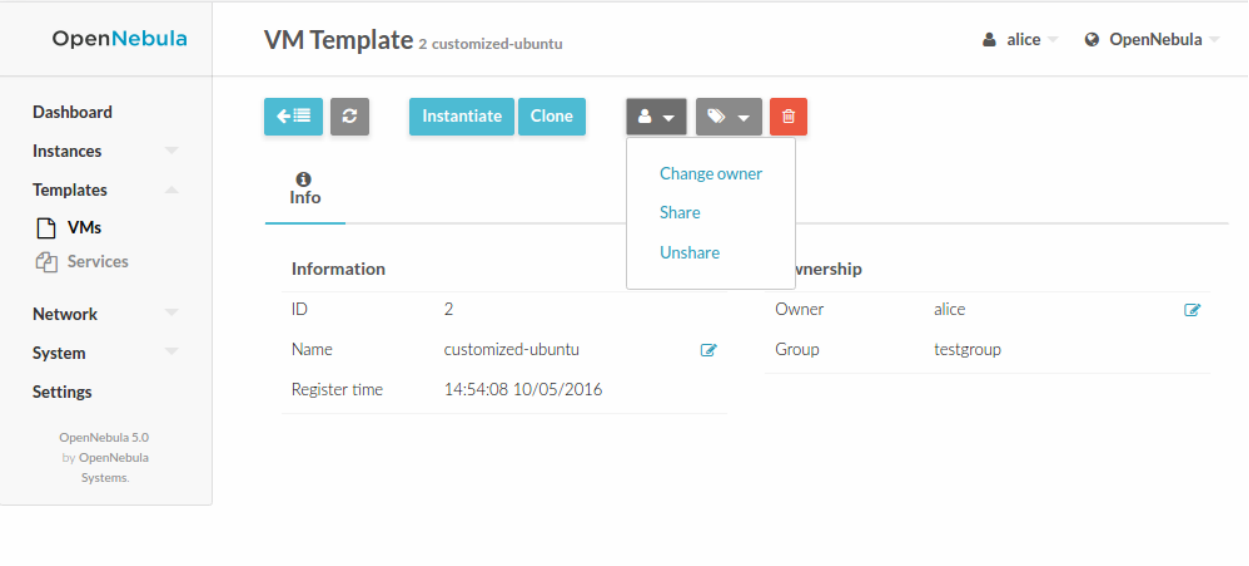
The Group admin can create new resources in the same way as a regular user does from the *Cloud view*. The creation wizard for the Virtual Machines and Services are similar in the `groupadmin` and `cloud` views.



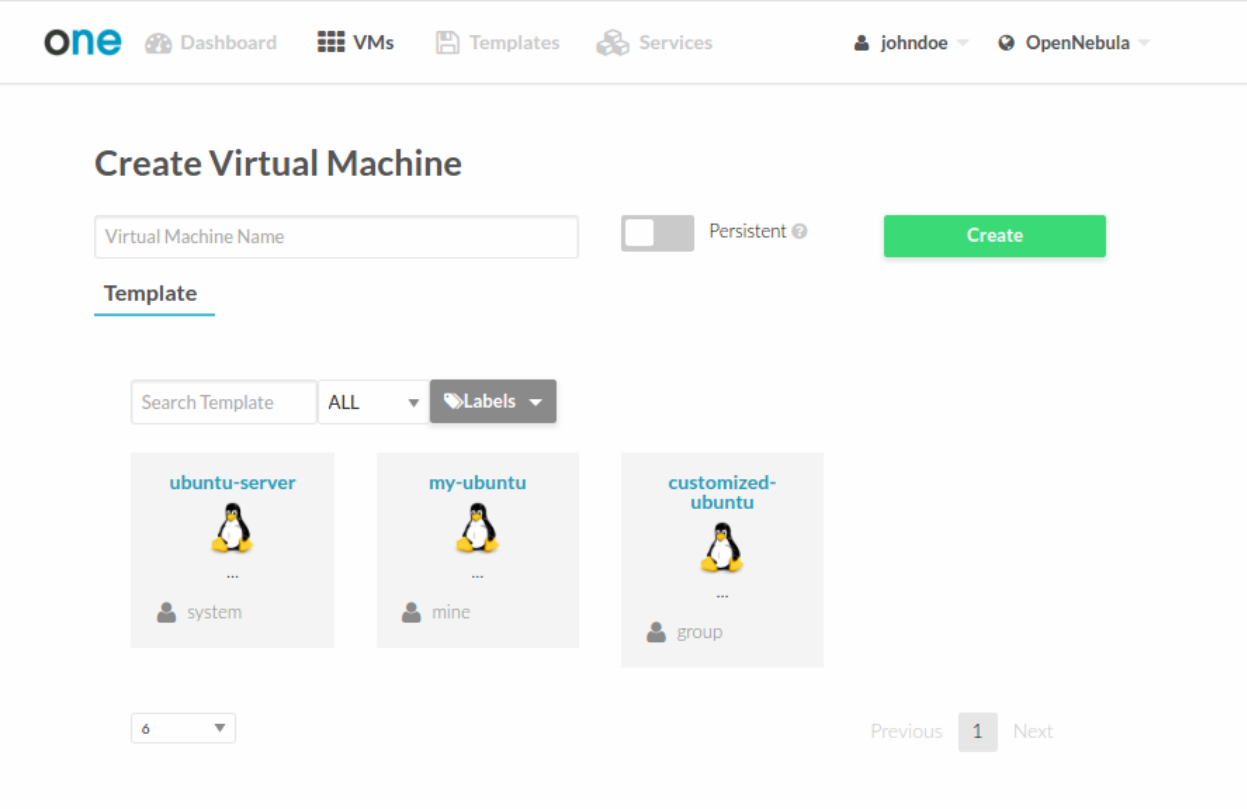
6.3.4 Prepare Resources for Other Users

Any user of the Cloud View or Group Admin View can save the changes made to a VM back to a new Template, and use this Template to instantiate new VMs later. See the *VM persistency options in the Cloud View* for more information.

The Group admin can also share his own Saved Templates with the rest of the group. For example the Group admin can instantiate a clean VM prepared by the cloud administrator, install software needed by other users in his Group, save it in a new Template and make it available for the rest of the group.



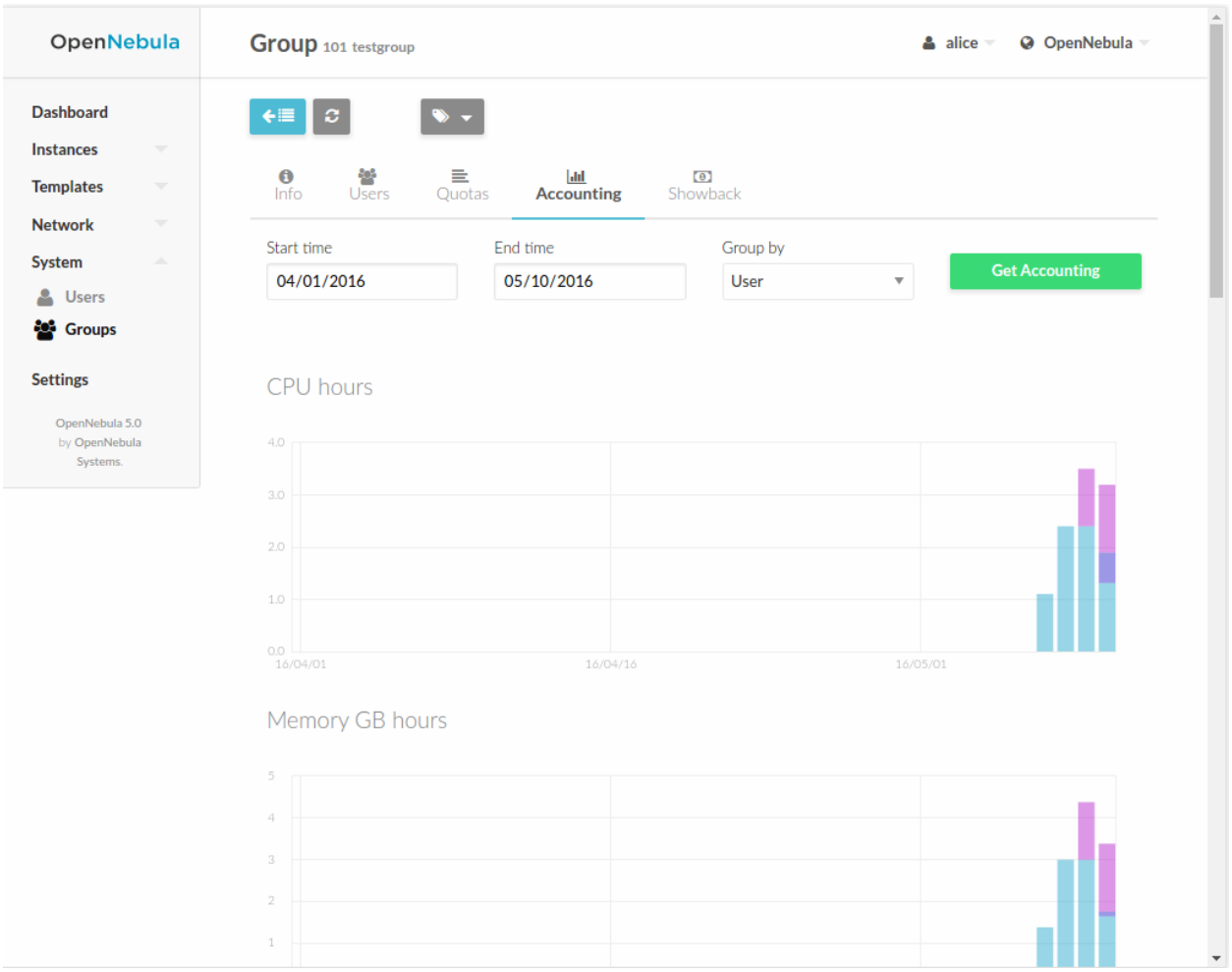
These shared templates will be listed to all the group users in the VM creation wizard, marked as ‘group’. A Saved Template created by a regular user is only available for that user and is marked as ‘mine’.

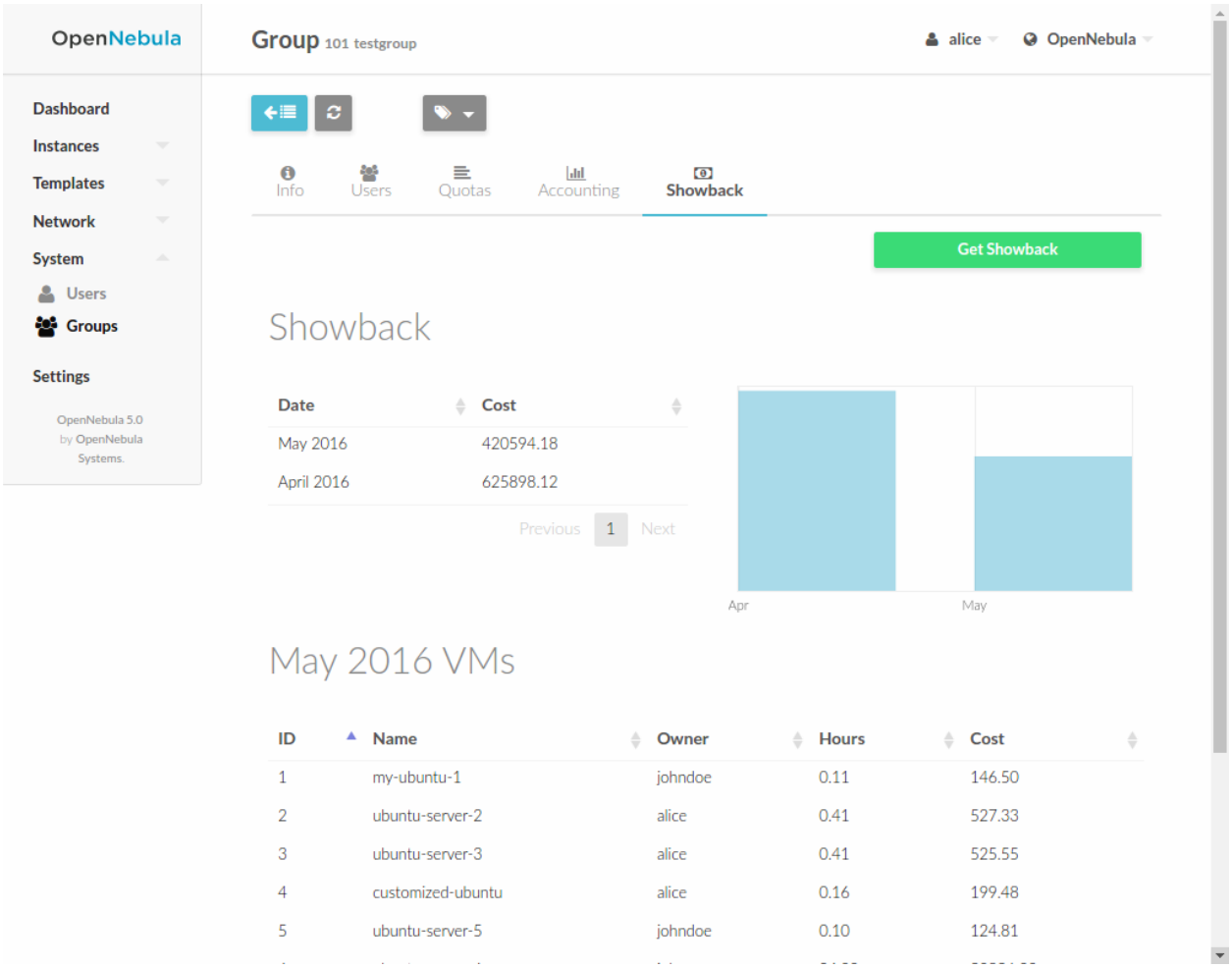


6.3.5 Accounting & Showback

Group Accounting & Showback

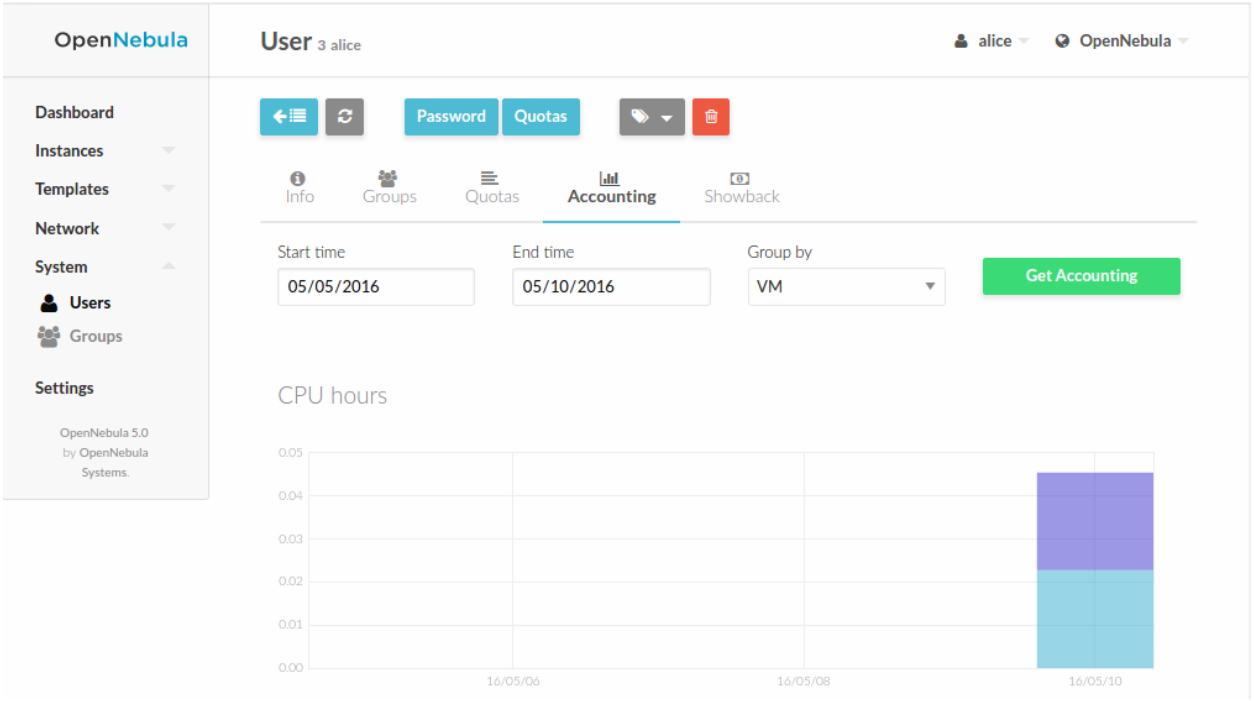
The Group info tab provides information of the usage of the Group and also accounting and showback reports can be generated. These reports can be configured to report the usage per VM or per user for a specific range of time.


OpenNebula
Group 101 testgroup
alice
OpenNebula
Dashboard
Instances
Templates
Network
System
Users
Groups
Settings
OpenNebula 5.0 by OpenNebula Systems.
Info
Users
Quotas
Accounting
Showback
Start time
04/01/2016
End time
05/10/2016
Group by
User
Get Accounting
CPU hours
4.0
3.0
2.0
1.0
0.0
16/04/01
16/04/16
16/05/01
Memory GB hours
5
4
3
2
1



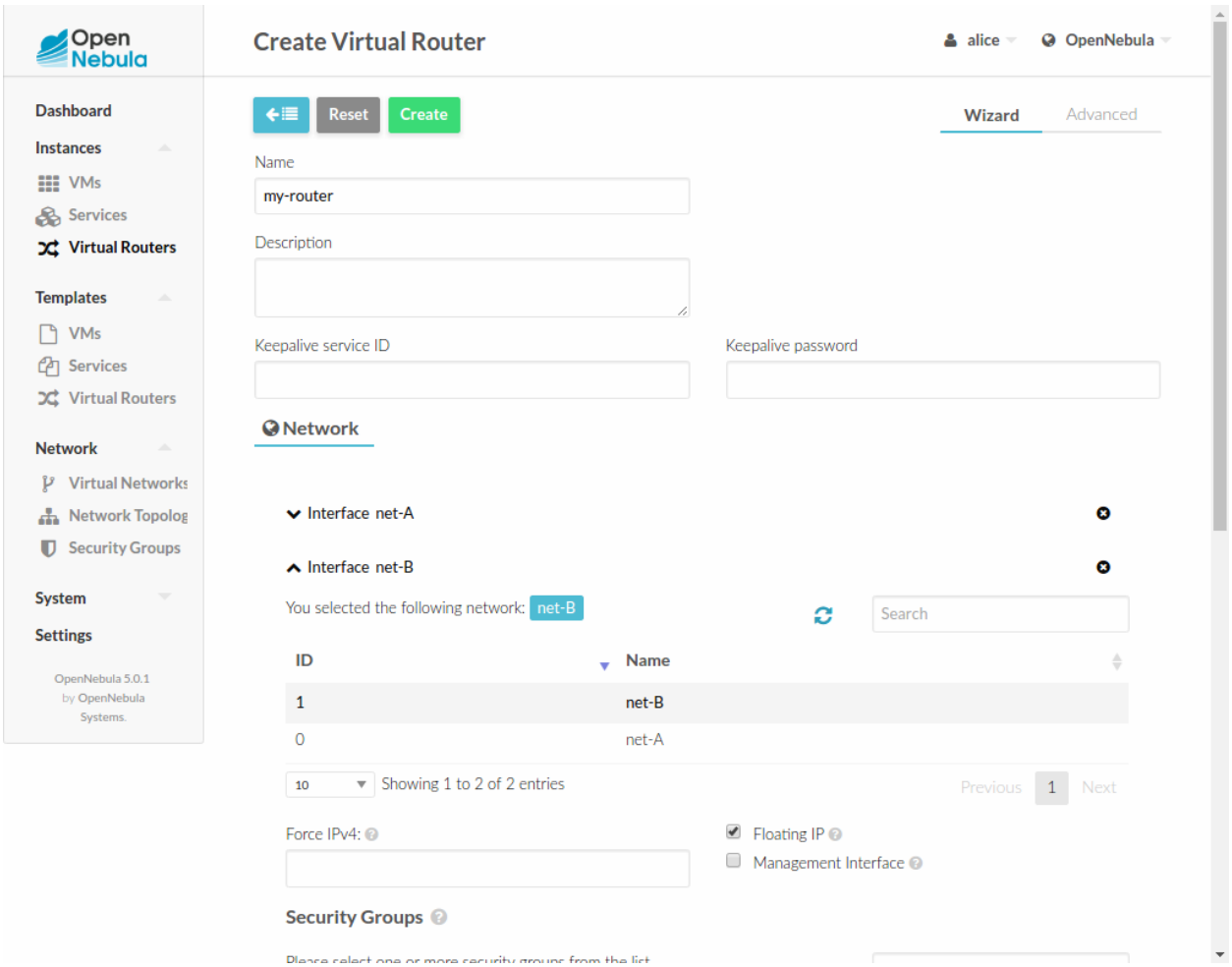
User Accounting & Showback

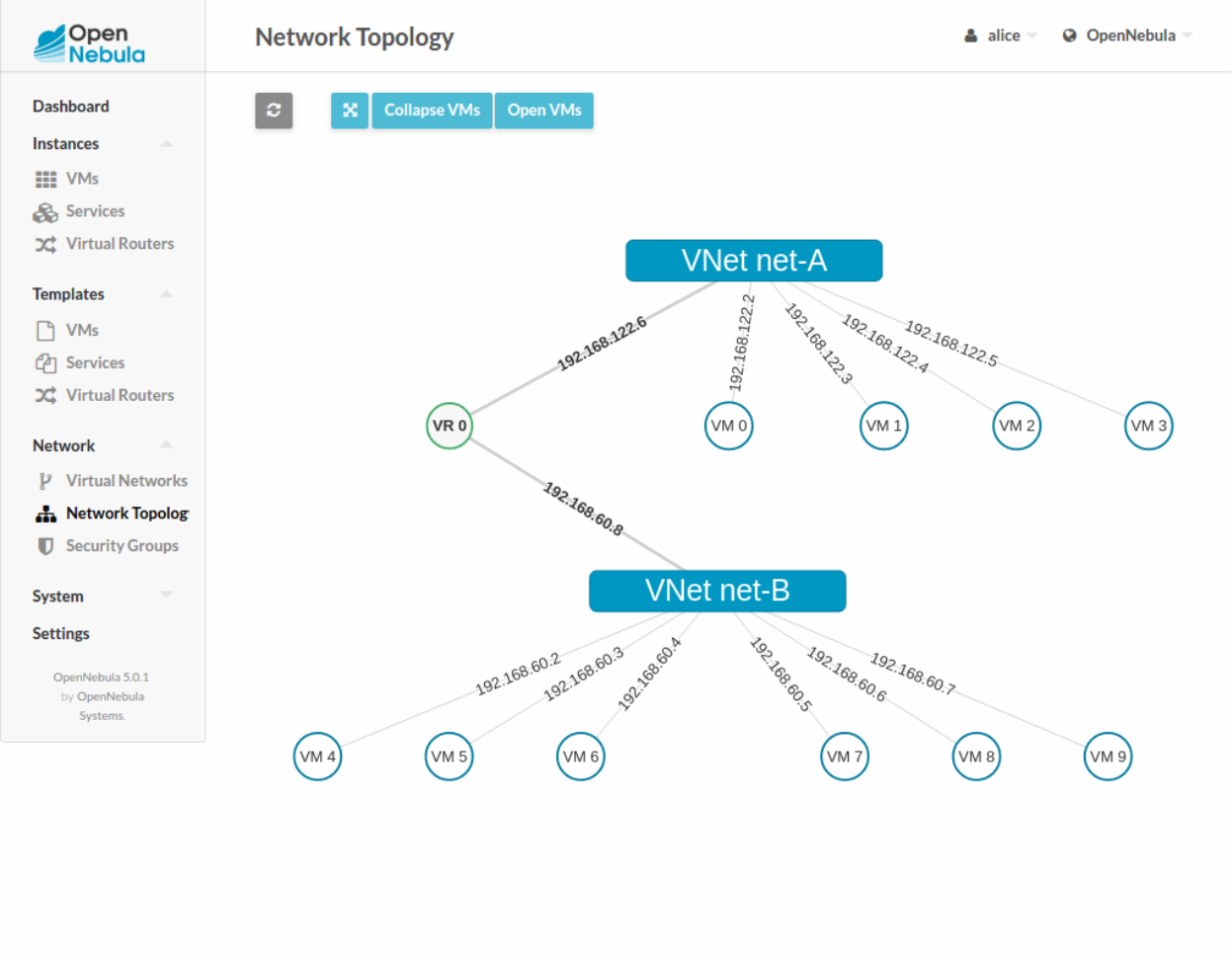
The detailed view of the user provides information of the usage of the user, from this view accounting reports can be also generated for this specific user



6.3.6 Networking

Group administrators can create *Virtual Routers* from Templates prepared by the cloud administrator. These Virtual Routers can be used to connect two or more of the Virtual Networks assigned to the Group.


Open Nebula
Create Virtual Router
alice
OpenNebula
Dashboard
Instances
VMs
Services
Virtual Routers
Templates
VMs
Services
Virtual Routers
Network
Virtual Networks
Network Topolog
Security Groups
System
Settings
OpenNebula 5.0.1 by OpenNebula Systems.
Reset
Create
Wizard
Advanced
Name
my-router
Description
Keepalive service ID
Keepalive password
Network
Interface net-A
Interface net-B
You selected the following network: net-B
Search
ID
Name
1
net-B
0
net-A
10
Showing 1 to 2 of 2 entries
Previous
1
Next
Force IPv4:
Floating IP
Management Interface
Security Groups


Open Nebula
Network Topology
alice
OpenNebula
Dashboard
Instances
VMs
Services
Virtual Routers
Templates
VMs
Services
Virtual Routers
Network
Virtual Networks
Network Topolog
Security Groups
System
Settings
OpenNebula 5.0.1 by OpenNebula Systems.
Collapse VMs
Open VMs
VNet net-A
192.168.122.6
192.168.122.2
192.168.122.3
192.168.122.4
192.168.122.5
VR 0
VM 0
VM 1
VM 2
VM 3
192.168.60.8
VNet net-B
192.168.60.2
192.168.60.3
192.168.60.4
192.168.60.5
192.168.60.6
192.168.60.7
VM 4
VM 5
VM 6
VM 7
VM 8
VM 9

CHAPTER

SEVEN

REFERENCES

7.1 Overview

Every resource in OpenNebula has its own Template, a collection of attributes that modify its behavior and their relationship with other cloud components. This Chapter contains an exhaustive reference of the templates of various resources.

Todo

To discuss: do we need host template reference? datastore reference? public_cloud reference vcenter, ec2, azure?

7.1.1 How Should I Read This Chapter

After reviewing and understanding the contents of the *operation guide* pertinent to your particular cloud infrastructure, you can use this reference Sections to look for the meaning of particular attributes that may be interesting to fine tune the behavior of different resources.

Within this Chapter, you can find references for the templates of *images*, *templates* and *virtual networks*.Also you can find references to all the commands of the *command line interface*, and a state machine describing all the *VM life-cycle states*.

You probably be coming back to these Chapter frequently, if you are in the process of deploying and configuring an OpenNebula cloud the next step would be to proceed to the Advanced Components Guide.

7.1.2 Hypervisor Compatibility

All the Sections of this Chapter applies to both KVM and vCenter hypervisors.

7.2 Virtual Machine Definition Template

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.

Note: If not explicitly stated, the described attributes are valid for all supported hypervisors.

7.2.1 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.

7.2.2 XML Syntax

Template files can be expressed 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>
```

7.2.3 Capacity Section

The following attributes can be defined to specify 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
```

7.2.4 Showback Section

The following attributes can be defined to set the cost of a VM. Read the *showback documentation* for more information.

Attribute	Description	Mandatory
MEMORY_COST	Cost of each memory MB per hour.	NO
CPU_COST	Cost of each CPU per hour.	NO
DISK_COST	Cost of each disk MB per hour.	NO

7.2.5 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	KVM	vCenter
ARCH	CPU architecture to virtualize	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	-
KERNEL_DS	image to be used as kernel (see !!)	O	-
INITRD	path to the initrd image in the host	O (for kernel)	-
INITRD_DS	image to be used as ramdisk (see !!)	O (for kernel)	-
ROOT	device to be mounted as root	O (for kernel)	-
KERNEL_CMD	arguments for the booting kernel	O (for kernel)	-
BOOTLOADER	path to the bootloader executable	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`	M	-

(!!) 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 console=tty1"]
```

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

7.2.6 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	KVM	vCenter
PAE	Physical address extension mode allows 32-bit guests to address more than 4 GB of memory	O	-
ACPI	Useful for power management, for example, with KVM guests it is required for graceful shutdown to work	O	-
APIC	Enables the advanced programmable IRQ management. Useful for SMP machines.	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	-
GUEST_AGENT	Enables the QEMU Guest Agent communication. This only creates the socket inside the VM, the Guest Agent itself must be installed and started in the VM.	O	-

```
FEATURES = [
    PAE = "yes",
    ACPI = "yes",
    APIC = "no",
    GUEST_AGENT = "yes"
]
```

7.2.7 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

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

DISK Sub-Attribute	Description	KVM	vCenter
IMAGE_ID	ID of the Image to use	M (no IMAGE)	M (no IMAGE)
IMAGE	Name of the Image to use	M(no IMAGE_ID)	M (no IMAGE_ID)
IMAGE_UID	To select the IMAGE of a given user by her ID	O	O
IMAGE_UNAME	To select the IMAGE of a given user by her NAME	O	O
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	O	O
TARGET	Device to map image disk. If set, it will overwrite the default device mapping.	O	-
DRIVER	Specific image mapping driver	O 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	O	-
READONLY	Set how the image is exposed by the hypervisor	O e.g.: `yes`, `no`. This attribute should only be used for special storage configurations	-
IO	Set IO policy. Values are `threads`, `native`	O (Needs qemu 1.1)	-
TOTAL_BYTES_SEC, **READ_BYTES_SEC**, **WRITE_BYTES_SEC** **TOTAL_IOPS_SEC**, **READ_IOPS_SEC**, **WRITE_IOPS_SEC**	IO throttling attributes for the disk. They are specified in bytes or IOPS (IO Operations) and can be specified for the total (read+write) or specific for read or write. Total and read or write can not be used at the same time. By default these parameters are only allowed to be used by oneadmin.	O (Needs qemu 1.1)	-
`ADAPTER_TYPE`	Possible values (careful with the case): lsiLogic, ide, busLogic. More information in the VMware documentation	-	M (can be inherited from Datastore)
`DISK_TYPE`	The type of disk has implications on performance and occupied space. Values (careful with the case): delta, eagerZeroedThick, flatMonolithic, preallocated, raw, rdm, rdmp, seSparse, sparse2Gb, sparseMonolithic, thick, thick2Gb, thin. More information in the VMware documentation	-	M (can be inherited from Datastore)

Volatile DISKS

DISK Sub-Attribute	Description	KVM	vCenter
TYPE	Type of the disk: `swap` or `fs`.	O	-
SIZE	size in MB	O	-
FORMAT	Format of the Image: `raw` or `qcow2`.	M(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	O	-
TARGET	device to map disk	O	-
DRIVER	special disk mapping options. KVM: `raw`, `qcow2`.	O	-
CACHE	Selects the cache mechanism for the disk. Values are `default`, `none`, `writethrough`, `writeback`, `directsync` and `unsafe`. More info in the libvirt documentation	O	-
READONLY	Set how the image is exposed by the hypervisor	O e.g.: `yes`, `no`. This attribute should only be used for special storage configurations	-
IO	Set IO policy. Values are `threads`, `native`	O	-
TOTAL_BYTES_SEC, **READ_BYTES_SEC**, **WRITE_BYTES_SEC**, **TOTAL_IOPS_SEC**, **READ_IOPS_SEC**, **WRITE_BYTES_SEC**	IO throttling attributes for the disk. They are specified in bytes or IOPS (IO Operations) and can be specified for the total (read+write) or specific for read or write. Total and read or write can not be used at the same time. By default these parameters are only allowed to be used by oneadmin.	O	-

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.

7.2.8 Network Section

NIC Sub-Attribute	Description	KVM	vCenter
NETWORK_ID	ID of the network to attach this device, as defined by `onevnet`. Use if no NETWORK	M (No NETWORK)	M (No NETWORK)
NETWORK	Name of the network to use (of those owned by user). Use if no NETWORK_ID	M (No NETWORK_ID)	M (No NETWORK_ID)
NETWORK_UID	To select the NETWORK of a given user by her ID	O	O
NETWORK_UNAME	To select the NETWORK of a given user by her NAME	O	O
IP	Request an specific IP from the `NETWORK`	O	O
MAC*	Request an specific HW address from the network interface	O	O
BRIDGE	Name of the bridge the network device is going to be attached to.	O	O
TARGET	name for the tun device created for the VM	O	O
SCRIPT	name of a shell script to be executed after creating the tun device for the VM	O	O
MODEL	hardware that will emulate this network interface. In KVM you can choose `virtio` to select its specific virtualization IO framework	O	O
SECURITY_GROUPS	Command separated list of the IDs of the security groups to be applied to this interface.	O	-

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*.

Network Defaults

You can define a `NIC_DEFAULT` attribute with values that will be copied to each new `NIC`. This is specially useful for an administrator to define configuration parameters, such as `MODEL`, that final users may not be aware of.

```
NIC_DEFAULT = [ MODEL = "virtio" ]
```

7.2.9 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	KVM	vCenter
INPUT	Define input devices, available sub-attributes: • **TYPE**: values are `mouse` or `tablet` • **BUS**: values are `usb`, `ps2`	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 • **RANDOM_PASSWD**: if "YES", generate a random password for each VM	O	O

Example:

```
GRAPHICS = [
  TYPE    = "vnc",
  LISTEN  = "0.0.0.0",
  PORT    = "5905"]
```

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: OpenNebula will prevent VNC port collision within a cluster to ensure that a VM can be deployed or migrated to any host in the selected cluster. If the selected port is in use the VM deployment will fail. If the user does not specify the port variable, OpenNebula will try to assign `VNC_PORTS[START] + VMID`, or the first lower available port. The `VNC_PORTS[START]` is specified inside the `oned.conf` file.

7.2.10 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:

Note the hypervisor column states that the attribute is **O**ptional, – not supported for that hypervisor or only valid for **Linux** guests.

Attribute	Description	KVM	vCenter	EC2
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).	O	O	O
FILES *	space-separated list of paths to include in context device.	O	-	-
FILES_DS	space-separated list of File images to include in context device. (Not supported for vCenter)	O	-	-
INIT_SCRIPTS	If the VM uses the OpenNebula contextualization package the init.sh file is executed by default. When the init script added is not called init.sh or more than one init script is added, this list contains the scripts to run and the order. Ex. "init.sh users.sh mysql.sh"	O	-	-
START_SCRIPT	Text of the script executed when the machine starts up. It can contain shebang in case it is not shell script. For example `START_SCRIPT="yum upgrade"`	O	O	O
START_SCRIPT_BASE64	The same as `START_SCRIPT` but encoded in Base64	O	O	O
TARGET	device to attach the context ISO.	O	-	-
TOKEN	`YES` to create a token.txt file for OneGate monitorization	O	O	O
NETWORK	`YES` to fill automatically the networking parameters for each NIC, used by the *Contextualization packages*	O	O	-
SET_HOSTNAME	This parameter value will be the hostname of the VM.	O	O	-
DNS_HOSTNAME	`YES` to set the VM hostname to the reverse dns name (from the first IP)	Linux	Linux	-
GATEWAY_IFACE	This variable can be set to the interface number you want to configure the gateway. It is useful when several networks have GATEWAY parameter and you want yo choose the one that configures it. For example to set the first interface to configure the gateway you use `GATEWAY_IFACE=0`	Linux	Linux	-
DNS	Specific DNS server for the Virtual Machine	Linux	Linux	-
ETHx_MAC	Used to find the correct interface	O	O	-
ETHx_IP	IPv4 address for the interface	O	O	-
ETHx_IPV6	IPv6 address for the interface	Linux	Linux	-
ETHx_NETWORK	Network address of the interface	O	O	-
ETHx_MASK	Network mask	O	O	-
ETHx_GATEWAY	Default IPv4 gateway for the interface	O	O	-
ETHx_GATEWAY6	Default IPv6 gateway for the interface	Linux	Linux	-
ETHx_MTU	`MTU` value for the interface	Linux	Linux	-
ETHx_DNS	DNS for the network	O	O	-
USERNAME	User to be created in the guest OS	O	O	-
PASSWORD	Password for the newly created user, used with **USERNAME**	O	O	-
CRYPTED_PASSWORD	Encrypted password for the new user, used with **USERNAME**	Linux	Linux	-
SSH_PUBLIC_KEY	Key to be added to **USERNAME** `authorized_keys` file or `root` in case **USERNAME** is not set.	Linux	Linux	O
EC2_PUBLIC_KEY	Same as **SSH_PUBLIC_KEY**	Linux	Linux	O

* 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|NIC_ID>=<vnet_id|vnet_name|nic_id>]`: 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"
]
```

7.2.11 Placement Section

The following attributes sets placement constraints and preferences for the VM, valid for all hypervisors:

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
```

7.2.12 vCenter Section

7.2.13 Public Cloud Section

To define a Virtual Machine in the supported cloud providers, the following attributes can be used in the PUBLIC_CLOUD section:

Amazon EC2 Attributes

More information in the Amazon EC2 Driver Section:

Attribute	Description	Mandatory
TYPE	Needs to be set to "EC2"	YES
AMI	Unique ID of a machine image, returned by a call to ec2-describe-images.	YES
AKI	The ID of the kernel with which to launch the instance.	NO
CLIENTTOKEN	Unique, case-sensitive identifier you provide to ensure idem-potency of the request.	NO
INSTANCETYPE	Specifies the instance type.	YES
KEYPAIR	The name of the key pair, later will be used to execute commands like *ssh -i id_keypair* or *scp -i id_keypair*	NO
LICENSEPOOL	–license-pool	NO
BLOCKDEVICEMAPPING	The block device mapping for the instance. More than one can be specified in a space-separated list. Check the –block-device-mapping option of the EC2 CLI Reference for the syntax	NO
PLACEMENTGROUP	Name of the placement group.	NO
PRIVATEIP	If you're using Amazon Virtual Private Cloud, you can optionally use this parameter to assign the instance a specific available IP address from the subnet.	NO
RAMDISK	The ID of the RAM disk to select.	NO
SUBNETID	If you're using Amazon Virtual Private Cloud, this specifies the ID of the subnet you want to launch the instance into. This parameter is also passed to the command *ec2-associate-address -i i-0041230 -a elasticip*.	NO
TENANCY	The tenancy of the instance you want to launch.	NO
USERDATA	Specifies Base64-encoded MIME user data to be made available to the instance(s) in this reservation.	NO
SECURITYGROUPS	Name of the security group. You can specify more than one security group (comma separated).	NO
SECURITYGROUPIDS	Ids of the security group. You can specify more than one security group (comma separated).	NO
ELASTICIP	EC2 Elastic IP address to assign to the instance. This parameter is passed to the command *ec2-associate-address -i i-0041230 elasticip*.	NO
TAGS	Key and optional value of the tag, separated by an equals sign (=).You can specify more than one tag (comma separated).	NO
AVAILABILITYZONE	The Availability Zone in which to run the instance.	NO
HOST	Defines which OpenNebula host will use this template	NO
EBS_OPTIMIZED	Obtain a better I/O throughput for VMs with EBS provisioned volumes	NO

Azure Attributes

More information in the Azure Driver Section:

Attribute	Description	Mandatory
TYPE	Needs to be set to "AZURE"	YES
INSTANCE_TYPE	Specifies the capacity of the VM in terms of CPU and memory	YES
IMAGE	Specifies the base OS of the VM. There are various ways to obtain the list of valid images for Azure, the simplest one is detailed here	YES
VM_USER	If the selected IMAGE is prepared for Azure provisioning, a username can be specified here to access the VM once booted	NO
VM_PASSWORD	Password for VM_USER	NO
LOCATION	Azure datacenter where the VM will be sent. See /etc/one/az_driver.conf for possible values (use the name of the section, not the region names). Spaces are not supported in this value.	NO
STORAGE_ACCOUNT	Specify the storage account where this VM will belong	NO
WIN_RM	Comma-separated list of possible protocols to access this Windows VM	NO
CLOUD_SERVICE	Specifies the name of the cloud service where this VM will be linked. Defaults to "csn<vid>, where vid is the id of the VM".	NO
TCP_ENDPOINTS	Comma-separated list of TCP ports to be accesible from the public internet to this VM	NO
SSHPORT	Port where the VMs ssh server will listen on	NO
VIRTUAL_NETWORK_NAME	Name of the virtual network to which this VM will be connected	NO
SUBNET	NAme of the particular Subnet where this VM will be connected to	NO
AVAILABILITY_SET	Name of the availability set to which this VM will belong	NO
AFFINITY_GROUP	Affinity groups allow you to group your Azure services to optimize performance. All services and VMs within an affinity group will be located in the same region belong	NO

Predefined Host Attributes

There are some predefined Host attributes that can be used in the requirements and rank expressions, valid for all hypervisors:

Attribute	Meaning
NAME	Hostname.
MAX_CPU	Total CPU in the host, in (# cores * 100).
CPU_USAGE	Allocated used CPU in (# cores * 100). This value is the sum of all the CPU requested by VMs running on the host, and is updated instantly each time a VM is deployed or undeployed.
FREE_CPU	Real free CPU in (# cores * 100), as returned by the probes. This value is updated each monitorization cycle.
USED_CPU	Real used CPU in (# cores * 100), as returned by the probes. USED_CPU = MAX_CPU - FREE_CPU. This value is updated each monitorization cycle.
MAX_MEM	Total memory in the host, in KB.
MEM_USAGE	Allocated used memory in KB. This value is the sum of all the memory requested by VMs running on the host, and is updated instantly each time a VM is deployed or undeployed.
FREE_MEM	Real free memory in KB, as returned by the probes. This value is updated each monitorization cycle.
USED_MEM	Real used memory in KB, as returned by the probes. USED_MEM = MAX_MEM - FREE_MEM. This value is updated each monitorization cycle.
RUNNING_VMS	Number of VMs deployed on this host.
HYPERVISOR	Hypervisor name.

You can execute `onehost show <id> -x` to see all the attributes and their values.

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

Requirement Expression Syntax

The syntax of the requirement expressions is defined as:

```
stmt::= expr';'
expr::= VARIABLE '=' NUMBER
      | VARIABLE '!=' NUMBER
      | VARIABLE '>' NUMBER
      | VARIABLE '<' NUMBER
      | VARIABLE '@>' NUMBER
      | VARIABLE '=' STRING
      | 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, 'I' OR, '!' NOT), '=' means equals with numbers (floats and integers). When you use '=' operator with strings, it performs a shell wildcard pattern matching. Additionally the '@>' operator means *contains*, if the variable evaluates to an array the expression will be true if that array contains the given number or string (or any string that matches the provided pattern).

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 '!='

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. Two different forms:
SCHED_REQUIREMENTS = "CURRENT_VMS = 5"
SCHED_REQUIREMENTS = "\"HOST/VMS/ID\" @> 5"

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

# Use any datastore that is in cluster 101 (it list of cluster IDs contains 101)
SCHED_DS_REQUIREMENTS = "\"CLUSTERS/ID\" @> 101"
```

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 specific hypervisor.

```
SCHED_REQUIREMENTS = "HYPERVISOR=\"vcenter\""
```

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"
```

7.2.14 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	KVM	vCenter
TYPE	Possible values are: `kvm`, `xen`, `vmware`	O	-
DATA	Raw data to be passed directly to the hypervisor	O	-
DATA_VMX	Raw data to be added directly to the .vmx file	-	-

7.2.15 Restricted Attributes

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

- `CONTEXT/FILES`
- `NIC/MAC`
- `NIC/VLAN_ID`
- `NIC/BRIDGE`
- `NIC_DEFAULT/MAC`
- `NIC_DEFAULT/VLAN_ID`
- `NIC_DEFAULT/BRIDGE`
- `DISK/TOTAL_BYTES_SEC`
- `DISK/READ_BYTES_SEC`
- `DISK/WRITE_BYTES_SEC`
- `DISK/TOTAL_IOPS_SEC`
- `DISK/READ_IOPS_SEC`
- `DISK/WRITE_IOPS_SEC`
- `CPU_COST`
- `MEMORY_COST`
- `DISK_COST`

These attributes can be configured in oned.conf.

7.2.16 User Inputs

Todo

update with new types, and options

`USER_INPUTS` provides the template creator with the possibility to dynamically ask the user instantiating the template for dynamic values that must be defined.

```
USER_INPUTS = [
  BLOG_TITLE="M|text|Blog Title",
  MYSQL_PASSWORD="M|password|MySQL Password",
  INIT_HOOK="M|text64|You can write a script that will be run on startup",
  <VAR>="M|<type>|<desc>"
]

CONTEXT=[
  BLOG_TITLE="$BLOG_TITLE",
  MYSQL_PASSWORD="$MYSQL_PASSWORD" ]
```

Note that the CONTEXT references the variables defined in the USER_INPUTS so the value is injected into the VM.

Valid `types` are `text`, `text64` and `password`. The type `text64` will encode the user's response in Base64.

7.3 Virtual Machines States Reference

This page is a complete reference of all the VM states that will be useful for administrators doing troubleshooting and developers.

The simplified life-cycle is explained in the *Managing Virtual Machines guide*. That simplified diagram uses a smaller number of state names. These names are the ones used by onevm list, e.g. `prolog`, `prolog_migrate` and `prolog_resume` are all presented as `prol`. It is intended as a reference for end-users. That section should be enough for end-users and every-day administration tasks.

7.3.1 List of States

In OpenNebula a Virtual Machine has 2 variables to define its state: `STATE` and `LCM_STATE`. The `LCM_STATE` is only relevant when the `STATE` is `ACTIVE`. Both states can be seen from the CLI (`onevm show`) and from Sunstone (Info panel for the VM).

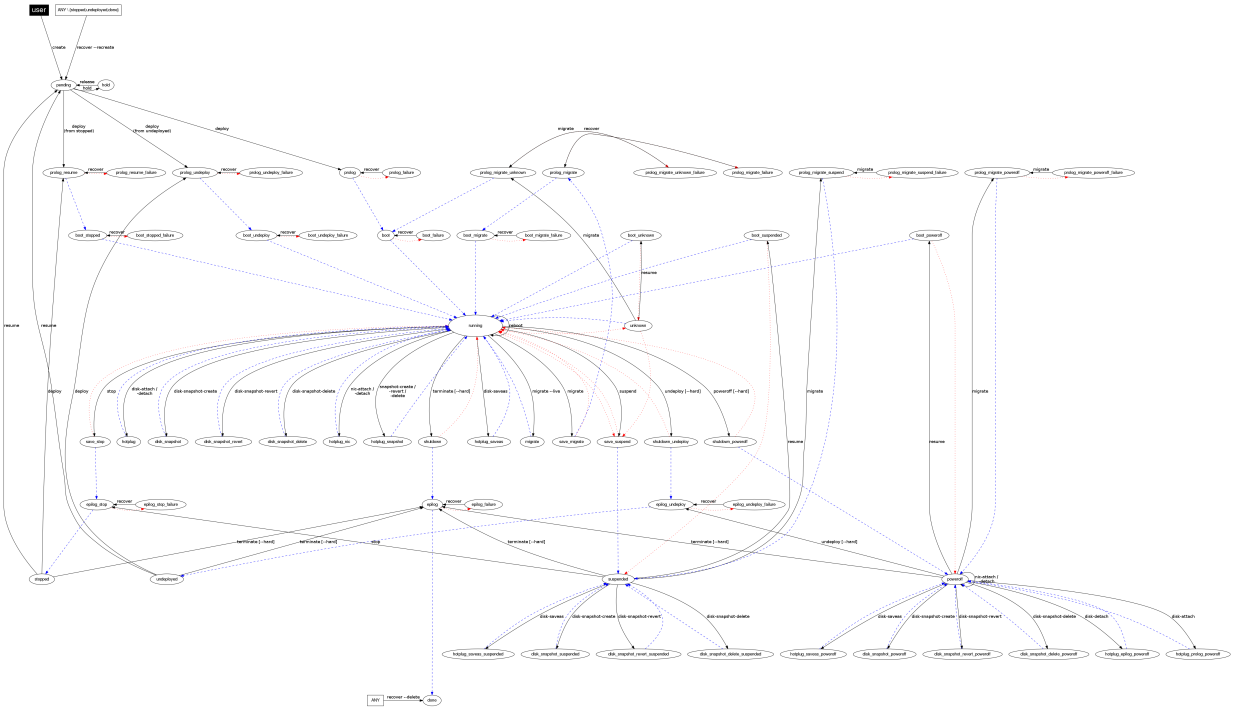
#	State	#	LCM State	Short State Alias	Meaning
0	INIT			`init`	Internal initialization s
1	PENDING			`pend`	By default a VM starts
2	HOLD			`hold`	The owner has held the
3	ACTIVE	0	LCM_INIT	`init`	Internal initialization s
		1	PROLOG	`prol`	The system is transferr
		2	BOOT	`boot`	OpenNebula is waiting
		3	RUNNING	`runn`	The VM is running (no
		4	MIGRATE	`migr`	The VM is migrating f
		5	SAVE_STOP	`save`	The system is saving th
		6	SAVE_SUSPEND	`save`	The system is saving th
		7	SAVE_MIGRATE	`save`	The system is saving th
		8	PROLOG_MIGRATE	`migr`	File transfers during a
		9	PROLOG_RESUME	`prol`	File transfers after a re
		10	EPILOG_STOP	`epil`	File transfers from the

#	State	#	LCM State	Short State Alias	Meaning
		11	EPILOG	epil	The system cleans up t
		12	SHUTDOWN	shut	OpenNebula has sent th
		15	CLEANUP_RESUBMIT	clea	Cleanup after a delete-
		16	UNKNOWN	unkn	The VM couldn't be m
		17	HOTPLUG	hotp	A disk attach/detach op
		18	SHUTDOWN_POWEROFF	shut	OpenNebula has sent th
		19	BOOT_UNKNOWN	boot	OpenNebula is waiting
		20	BOOT_POWEROFF	boot	OpenNebula is waiting
		21	BOOT_SUSPENDED	boot	OpenNebula is waiting
		22	BOOT_STOPPED	boot	OpenNebula is waiting
		23	CLEANUP_DELETE	clea	Cleanup after a delete a
		24	HOTPLUG_SNAPSHOT	snap	A system snapshot acti
		25	HOTPLUG_NIC	hotp	A NIC attach/detach op
		26	HOTPLUG_SAVEAS	hotp	A disk-saveas operation
		27	HOTPLUG_SAVEAS_POWEROFF	hotp	A disk-saveas operation
		28	HOTPLUG_SAVEAS_SUSPENDED	hotp	A disk-saveas operation
		29	SHUTDOWN_UNDEPLOY	shut	OpenNebula has sent th
		30	EPILOG_UNDEPLOY	epil	The system cleans up t
		31	PROLOG_UNDEPLOY	prol	File transfers after a re
		32	BOOT_UNDEPLOY	boot	OpenNebula is waiting
		33	HOTPLUG_PROLOG_POWEROFF	hotp	File transfers for a disk
		34	HOTPLUG_EPILOG_POWEROFF	hotp	File transfers for a disk
		35	BOOT_MIGRATE	boot	OpenNebula is waiting
		36	BOOT_FAILURE	fail	Failure during a BOOT
		37	BOOT_MIGRATE_FAILURE	fail	Failure during a BOOT
		38	PROLOG_MIGRATE_FAILURE	fail	Failure during a PROL
		39	PROLOG_FAILURE	fail	Failure during a PROL
		40	EPILOG_FAILURE	fail	Failure during an EPIL
		41	EPILOG_STOP_FAILURE	fail	Failure during an EPIL
		42	EPILOG_UNDEPLOY_FAILURE	fail	Failure during an EPIL
		43	PROLOG_MIGRATE_POWEROFF	migr	File transfers during a
		44	PROLOG_MIGRATE_POWEROFF_FAILURE	fail	Failure during a PROL
		45	PROLOG_MIGRATE_SUSPEND	migr	File transfers during a
		46	PROLOG_MIGRATE_SUSPEND_FAILURE	fail	Failure during a PROL
		47	BOOT_UNDEPLOY_FAILURE	fail	Failure during a BOOT
		48	BOOT_STOPPED_FAILURE	fail	Failure during a BOOT
		49	PROLOG_RESUME_FAILURE	fail	Failure during a PROL
		50	PROLOG_UNDEPLOY_FAILURE	fail	Failure during a PROL
		51	DISK_SNAPSHOT_POWEROFF	snap	A disk-snapshot-create
		52	DISK_SNAPSHOT_REVERT_POWEROFF	snap	A disk-snapshot-revert
		53	DISK_SNAPSHOT_DELETE_POWEROFF	snap	A disk-snapshot-delete
		54	DISK_SNAPSHOT_SUSPENDED	snap	A disk-snapshot-create
		55	DISK_SNAPSHOT_REVERT_SUSPENDED	snap	A disk-snapshot-revert
		56	DISK_SNAPSHOT_DELETE_SUSPENDED	snap	A disk-snapshot-delete
		57	DISK_SNAPSHOT	snap	A disk-snapshot-create
		59	DISK_SNAPSHOT_DELETE	snap	A disk-snapshot-delete
		60	PROLOG_MIGRATE_UNKNOWN	migr	File transfers during a
		61	PROLOG_MIGRATE_UNKNOWN_FAILURE	fail	Failure during a PROL
4	STOPPED			stop	The VM is stopped. VI

#	State	#	LCM State	Short State Alias	Meaning
5	SUSPENDED			`susp`	Same as stopped, but th
6	DONE			`done`	The VM is done. VMs
8	POWEROFF			`poff`	Same as suspended, bu
9	UNDEPLOYED			`unde`	The VM is shut down.
10	CLONING			`clon`	The VM is waiting for
11	CLONING_FAILURE			`fail`	Failure during a CLON

7.3.2 Diagram

You can click on the following image to open it in a new window. For a simplified version of this diagram, please visit the *Managing Virtual Machines guide*.



7.4 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.

7.4.1 Template Attributes

The following attributes can be defined in the template.

Attribute	KVM	vCenter	Value	Description
NAME	Mandatory	Mandatory	Any string	Name that the Image will get. Every image must have a unique name.
DESCRIPTION	Optional	Optional	Any string	Human readable description of the image for other users.
TYPE	Optional	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	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`	Specific image mapping driver.
PATH	Mandatory (if no SOURCE)	Mandatory	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	Optional	For KVM: `BLOCK`, `CDROM` or `FILE` (default). For vCenter (careful with the case): delta,eagerZeroedThick,flatMonolithic,preallocated,raw,rdm,rdmp,seSparse,sparse2Gb,sparseMonolithic,thick (`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
ADAPTER_TYPE	Optional	Optional	Possible values (careful with the case): lsiLogic, ide, busLogic.	Type of controller to be used with this disk. More information in the VMware documentation
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`.
MD5	Optional	Optional	An md5 hash	MD5 hash to check for image integrity
SHA1	Optional	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.

7.4.2 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
DESCRIPTION   = "Storage for my Thesis experiments."
```

7.4.3 Restricted Attributes

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

- SOURCE

7.5 Virtual Network Definition

This page describes how to define a new Virtual Network. A Virtual Network includes three different aspects:

- Physical network attributes
- Address Range
- Configuration attributes for the guests

Note: When writing a Virtual Network template in a file just follows the same syntax as the *VM template*.

7.5.1 Physical Network Attributes

It defines the **underlying networking infrastructure** that will support the Virtual Network, such as the VLAN ID or the hypervisor interface to bind the Virtual Network.

Attribute	Description	Value	Mandatory	Drivers
NAME	Name of the Virtual Network	String	YES	All
VN_MAD	The network driver to implement the network	802.1Q ebtables fw ovswtich vxlan vcenter dummy	YES	All
BRIDGE	Device to attach the virtual machines to, depending on the network driver it may refer to different technologies or require host setups.	String	YES for dummy ovswtich ebtables fw vcenter	dummy 802.1Q vxlan ovswicth ebtables fw vcenter
VLAN_ID	Identifier for the VLAN	Integer	NO	802.1Q vxlan ovswtich vcenter
PHYDEV	Name of the physical network device that will be attached to the bridge.	String	YES	802.1Q vxlan

7.5.2 The Address Range

IPv4 Address Range

Attribute	Description	Mandatory
TYPE	`IP4`	YES
IP	First IP in the range in dot notation.	YES
MAC	First MAC, if not provided it will be generated using the IP and the MAC_PREFIX in `oned.conf`.	NO
SIZE	Number of addresses in this range.	YES

IPv6 Address Range

Attribute	Description	Mandatory
TYPE	`IP6`	YES
MAC	First MAC, if not provided it will be generated.	YES
GLOBAL_PREFIX	A /64 globally routable prefix	NO
ULA_PREFIX	A /64 unique local address (ULA) prefix corresponding to the `fd00::/8` block	NO
SIZE	Number of addresses in this range.	YES

Dual IPv4-IPv6 Address Range

Attribute	Description	Mandatory
TYPE	`IP4_6`	YES
IP	First IPv4 in the range in dot notation.	YES
MAC	First MAC, if not provided it will be generated using the IP and the MAC_PREFIX in `oned.conf`.	NO
GLOBAL_PREFIX	A /64 globally routable prefix	NO
ULA_PREFIX	A /64 unique local address (ULA) prefix corresponding to the `fd00::/8` block	NO
SIZE	Number of addresses in this range.	YES

Ethernet Address Range

Attribute	Description	Mandatory
TYPE	`ETHER`	YES
MAC	First MAC, if not provided it will be generated randomly.	NO
SIZE	Number of addresses in this range.	YES

7.5.3 Contextualization Attributes

Attribute	Description
NETWORK_ADDRESS	Base network address
NETWORK_MASK	Network mask
GATEWAY	Default gateway for the network
GATEWAY6	IPv6 router for this network
DNS	DNS servers, a space separated list of servers
GUEST_MTU	Sets the MTU for the NICs in this network
CONTEXT_FORCE_IPV4	When a vnet is IPv6 the IPv4 is not configured unless this attribute is set
SEARCH_DOMAIN	Default search domains for DNS resolution

7.5.4 Virtual Network Definition Examples

Sample IPv4 VNet:

```
# Configuration attributes (dummy driver)
NAME        = "Private Network"
DESCRIPTION = "A private network for VM inter-communication"

BRIDGE = "bond-br0"

# Context attributes
NETWORK_ADDRESS = "10.0.0.0"
NETWORK_MASK    = "255.255.255.0"
DNS             = "10.0.0.1"
GATEWAY         = "10.0.0.1"

#Address Ranges, only these addresses will be assigned to the VMs
AR=[TYPE = "IP4", IP = "10.0.0.10", SIZE = "100" ]

AR=[TYPE = "IP4", IP = "10.0.0.200", SIZE = "10" ]
```

Sample IPv4 VNet, using AR of just one IP:

```
# Confgiuration attributes (OpenvSwtich driver)
NAME        = "Public"
DESCRIPTION = "Network with public IPs"

BRIDGE  = "br1"
VLAN    = "YES"
VLAN_ID = 12

DNS            = "8.8.8.8"
GATEWAY        = "130.56.23.1"
LOAD_BALANCER = 130.56.23.2

AR=[ TYPE = "IP4", IP = "130.56.23.2", SIZE = "1"]
AR=[ TYPE = "IP4", IP = "130.56.23.34", SIZE = "1"]
AR=[ TYPE = "IP4", IP = "130.56.23.24", SIZE = "1"]
AR=[ TYPE = "IP4", IP = "130.56.23.17", MAC= "50:20:20:20:20:21", SIZE = "1"]
AR=[ TYPE = "IP4", IP = "130.56.23.12", SIZE = "1"]
```

7.6 Command Line Interface

OpenNebula provides a set commands to interact with the system:

7.6.1 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
- onesecgroup: manages OpenNebula security groups
- onevcenter: handles vCenter resource import
- onevrouter: manages OpenNebula Virtual Routers
- oneshowback: OpenNebula Showback Tool

- onemarket: manages internal and external Marketplaces
- onemarketapp: manages appliances from Marketplaces

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`.

7.6.2 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 of virtual machines (or cancel, depending on its state).
- econe-reboot-instances: Reboots a set of virtual machines.
- econe-start-instances: Starts a set of virtual machines.
- econe-stop-instances: Stops a set of virtual 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

7.6.3 OneFlow Commands

- oneflow: OneFlow Service management
- oneflow-template: OneFlow Service Template management

7.6.4 OneFlow Commands

- *onegate </doc/5.0/cli/oneflow.1.html>*: OneGate Service management