

---

**OpenNebula.org**

**OpenNebula 4.14 Advanced  
Administration Guide**

*Release 4.14.2*

**OpenNebula Project**

November 25, 2015



<b>1</b>	<b>Multi-VM Applications and Auto-scaling</b>	<b>1</b>
1.1	OneFlow . . . . .	1
1.2	OneFlow Server Configuration . . . . .	1
1.3	Managing Multi-tier Applications . . . . .	5
1.4	Application Auto-scaling . . . . .	20
<b>2</b>	<b>Data Center Federation</b>	<b>29</b>
2.1	Data Center Federation . . . . .	29
2.2	OpenNebula Federation Configuration . . . . .	30
2.3	OpenNebula Federation Management . . . . .	35
<b>3</b>	<b>Scalability</b>	<b>37</b>
3.1	Configuring Sunstone for Large Deployments . . . . .	37
3.2	Configuring OpenNebula for Large Deployments . . . . .	41
<b>4</b>	<b>High Availability</b>	<b>43</b>
4.1	Virtual Machines High Availability . . . . .	43
4.2	OpenNebula High Availability . . . . .	46
<b>5</b>	<b>Cloud Bursting</b>	<b>53</b>
5.1	Cloud Bursting . . . . .	53
5.2	Amazon EC2 Driver . . . . .	54
5.3	SoftLayer Driver . . . . .	61
5.4	Azure Driver . . . . .	69
<b>6</b>	<b>Application Insight</b>	<b>79</b>
6.1	OneGate . . . . .	79
6.2	OneGate Server Configuration . . . . .	79
6.3	OneGate Usage . . . . .	83
<b>7</b>	<b>Public Cloud</b>	<b>93</b>
7.1	Building a Public Cloud . . . . .	93
7.2	EC2 Server Configuration . . . . .	94
7.3	OpenNebula EC2 User Guide . . . . .	101
7.4	EC2 Ecosystem . . . . .	104



## MULTI-VM APPLICATIONS AND AUTO-SCALING

### 1.1 OneFlow

OneFlow allows users and administrators to define, execute and manage multi-tiered applications, or services composed of interconnected Virtual Machines with deployment dependencies between them. Each group of Virtual Machines is deployed and managed as a single entity, and is completely integrated with the advanced OpenNebula user and group management.

#### 1.1.1 Benefits

- Define multi-tiered applications (services) as collection of applications
- Manage multi-tiered applications as a single entity
- Automatic execution of services with dependencies
- Provide configurable services from a catalog and self-service portal
- Enable tight, efficient administrative control
- Fine-grained access control for the secure sharing of services with other users
- Auto-scaling policies based on performance metrics and schedule

#### 1.1.2 Next Steps

- *OneFlow Server Configuration*
- *Multi-tier Applications*
- *Application Auto-scaling*

### 1.2 OneFlow Server Configuration

The OneFlow commands do not interact directly with the OpenNebula daemon, there is a server that takes the requests and manages the service (multi-tiered application) life-cycle. This guide shows how to start OneFlow, and the different options that can be configured.

## 1.2.1 Installation

OneFlow server is shipped with the main distribution. The oneflow server is usually contained in the ‘opennebula-flow’ package, and the commands in the specific CLI package. Check the Installation guide for details of what packages you have to install depending on your distribution.

Make sure you execute `install_gems` to install the required gems, in particular: `treetop`, `parse-cron`.

## 1.2.2 Configuration

The OneFlow configuration file can be found at `/etc/one/oneflow-server.conf`. It uses YAML syntax to define the following options:

Option	Description
<b>Server Configuration</b>	
<code>:one_xmlrpc</code>	OpenNebula daemon host and port
<code>:lcm_interval</code>	Time in seconds between Life Cycle Manager steps
<code>:host</code>	Host where OneFlow will listen
<code>:port</code>	Port where OneFlow will listen
<b>Defaults</b>	
<code>:default_cooldown</code>	Default cooldown period after a scale operation, in seconds
<code>:shutdown_action</code>	Default shutdown action. Values: ‘shutdown’, ‘shutdown-hard’
<code>:action_number :action_period</code>	Default number of virtual machines ( <code>action_number</code> ) that will receive the given call in each interval defined by <code>action_period</code> , when an action is performed on a role.
<code>:vm_name_template</code>	Default name for the Virtual Machines created by oneflow. You can use any of the following placeholders: <ul style="list-style-type: none"> <li>• <code>\$SERVICE_ID</code></li> <li>• <code>\$SERVICE_NAME</code></li> <li>• <code>\$ROLE_NAME</code></li> <li>• <code>\$VM_NUMBER</code></li> </ul>
<b>Auth</b>	
<code>:core_auth</code>	Authentication driver to communicate with OpenNebula core <code>cipher</code> : for symmetric cipher encryption of tokens <code>x509</code> : for x509 certificate encryption of tokens For more information, visit the OpenNebula Cloud Auth documentation
<b>Log</b>	
<code>:debug_level</code>	Log debug level. 0 = ERROR, 1 = WARNING, 2 = INFO, 3 = DEBUG

This is the default file

```
#####
# Server Configuration
#####

# OpenNebula daemon contact information
#
:one_xmlrpc: http://localhost:2633/RPC2

# Time in seconds between Life Cycle Manager steps
```

```

#
:lcm_interval: 30

# Host and port where OneFlow server will run
:host: 127.0.0.1
:port: 2474

#####
# Defaults
#####

# Default cooldown period after a scale operation, in seconds
:default_cooldown: 300

# Default shutdown action. Values: 'shutdown', 'shutdown-hard'
:shutdown_action: 'shutdown'

# Default oneflow action options when only one is supplied
:action_number: 1
:action_period: 60

# Default name for the Virtual Machines created by oneflow. You can use any
# of the following placeholders:
# $SERVICE_ID
# $SERVICE_NAME
# $ROLE_NAME
# $VM_NUMBER

:vm_name_template: '$ROLE_NAME_ $VM_NUMBER_(service_ $SERVICE_ID) '

#####
# Auth
#####

# Authentication driver to communicate with OpenNebula core
# - cipher, for symmetric cipher encryption of tokens
# - x509, for x509 certificate encryption of tokens
:core_auth: cipher

#####
# Log
#####

# Log debug level
# 0 = ERROR, 1 = WARNING, 2 = INFO, 3 = DEBUG
#
:debug_level: 2

```

### 1.2.3 Start OneFlow

To start and stop the server, use the `oneflow-server start/stop` command:

```

$ oneflow-server start
oneflow-server started

```

**Warning:** By default, the server will only listen to requests coming from localhost. Change the `:host` attribute in `/etc/one/oneflow-server.conf` to your server public IP, or `0.0.0.0` so oneflow will listen on any interface.

Inside `/var/log/one/` you will find new log files for the server, and individual ones for each service in `/var/log/one/oneflow/<id>.log`

```
/var/log/one/oneflow.error
/var/log/one/oneflow.log
```

## 1.2.4 Enable the Sunstone Tabs

The OneFlow tabs are enabled by default. To enable or disable them, edit `/etc/one/sunstone-views/admin.yaml` and `user.yaml` and comment out the oneflow tabs lines inside `enabled_tabs`:

```
enabled_tabs:
  - dashboard-tab

  ...

  - oneflow-dashboard
  - oneflow-services
  - oneflow-templates
```

Be sure to restart Sunstone for the changes to take effect.

For more information on how to customize the views based on the user/group interacting with Sunstone check the [sunstone views guide](#)

## 1.2.5 Advanced Setup


### Permission to Create Services


By default, new groups are allowed to create Document resources. Documents are a special tool used by OneFlow to store Service Templates and instances. When a new Group is created, you can decide if you want to allow or deny its users to create OneFlow resources (Documents).





## Create Group ✕

Name:

  
Views

  
Resources

  
Admin

  
Permissions

Allow users to view the VMs and Services of other users in the same group ?

Allow users in this group to create the following resources ?

	VMs	VNets	Images	Templates	Documents <span style="float: right;">?</span>
Users	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Admins	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Reset

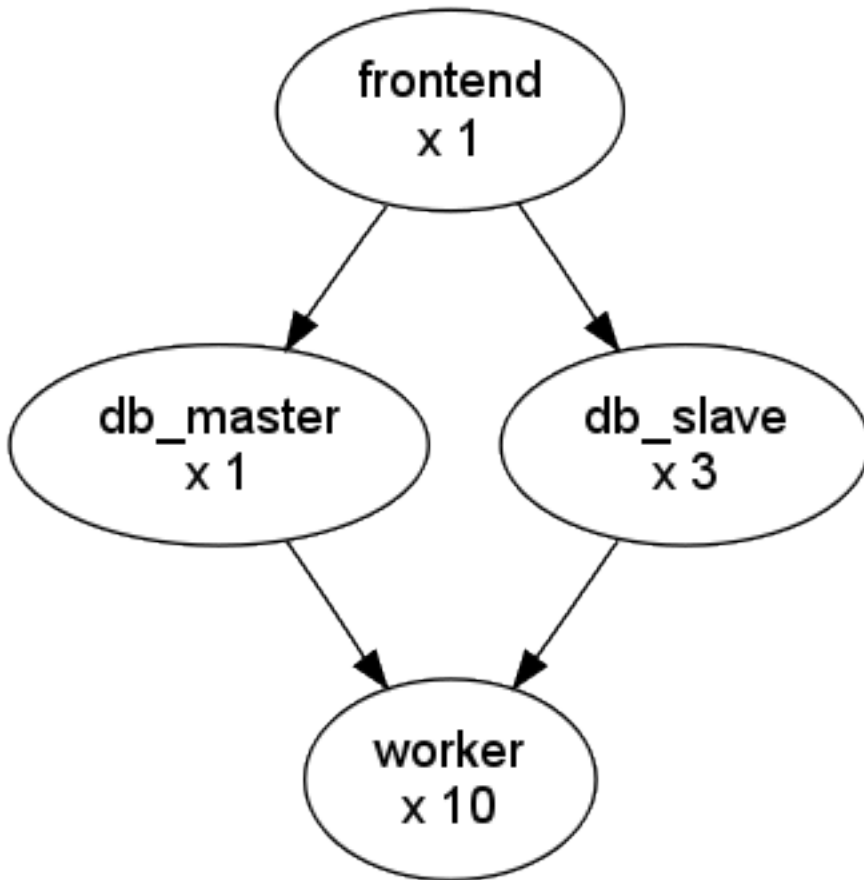
Create

## 1.3 Managing Multi-tier Applications

OneFlow allows users and administrators to define, execute and manage multi-tiered applications, or services composed of interconnected Virtual Machines with deployment dependencies between them. Each group of Virtual Machines is deployed and managed as a single entity, and is completely integrated with the advanced OpenNebula user and group management.

### 1.3.1 What Is a Service

The following diagram represents a multi-tier application. Each node represents a Role, and its cardinality (the number of VMs that will be deployed). The arrows indicate the deployment dependencies: each Role's VMs are deployed only when all its parent's VMs are running.



This Service can be represented with the following JSON template:

```
{
  "name": "my_service",
  "deployment": "straight",
  "ready_status_gate": true|false,
  "roles": [
    {
      "name": "frontend",
      "vm_template": 0
    },
    {
      "name": "db_master",
      "parents": [
        "frontend"
      ],
      "vm_template": 1
    },
    {
      "name": "db_slave",
      "parents": [
        "frontend"
      ],
      "cardinality": 3,
      "vm_template": 2
    },
    {

```

```
"name": "worker",
"parents": [
  "db_master",
  "db_slave"
],
"cardinality": 10,
"vm_template": 3
}
]
}
```

### 1.3.2 Managing Service Templates

OneFlow allows OpenNebula administrators and users to register Service Templates in OpenNebula, to be instantiated later as Services. These Templates can be instantiated several times, and also shared with other users.

Users can manage the Service Templates using the command `oneflow-template`, or the graphical interface. For each user, the actual list of Service Templates available is determined by the ownership and permissions of the Templates.

#### Create and List Existing Service Templates

The command `oneflow-template create` registers a JSON template file. For example, if the previous example template is saved in `/tmp/my_service.json`, you can execute:

```
$ oneflow-template create /tmp/my_service.json
ID: 0
```

You can also create service template from Sunstone:



## Create Service Template

Name <sup>?</sup>

Description <sup>?</sup>

^ Network Configuration

**Network Configuration**

Name	Description
Private	Private Network for internal communication
Public	Public IP Addresses

+ Add another Network

^ Advanced Service Parameters

### Roles

Master <sup>?</sup> Slave <sup>?</sup> + Add another role

Role Name <sup>?</sup>

VM template <sup>?</sup>  VMs <sup>?</sup>

**Network Interfaces** **Parent roles**

Private  Master

Public

^ Role Elasticity

^ Advanced Role Parameters

Reset Create

To list the available Service Templates, use `onflow-template list/show/top`:

```
$ onflow-template list
      ID USER          GROUP          NAME
      0 onadmin       onadmin       my_service

$ onflow-template show 0
SERVICE TEMPLATE 0 INFORMATION
ID                : 0
NAME              : my_service
USER              : onadmin
```

```

GROUP                : oneadmin

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

TEMPLATE CONTENTS
{
  "name": "my_service",
  "roles": [
    {
      ....

```

Templates can be deleted with `onflow-template delete`.

### Determining when a VM is READY

Depending on the deployment strategy, OneFlow will wait until all the VMs in a specific role are all in running state before deploying VMs that belong to a child role. How OneFlow determines the running state of the VMs can be specified with the checkbox `Wait for VMs to report that they are READY` available in the service creation dialog in Sunstone, or the attribute in `ready_status_gate` in the top-level of the service JSON.

If `ready_status_gate` is set to `true`, a VM will only be considered to be in running state the following points are true:

- VM is in running state for OpenNebula. Which specifically means that `LCM_STATE==3` and `STATE>=3`
- The VM has `READY=YES` in the user template.

The idea is to report via *OneGate* from inside the VM that it's running during the boot sequence:

```

curl -X "PUT" http://<onegate>/vm \
  --header "X-ONEGATE-TOKEN: ..." \

```

```
--header "X-ONEGATE-VMID: ..." \  
-d "READY = YES"
```



This can also be done directly using OpenNebula's interfaces: CLI, Sunstone or API.

If `ready_status_gate` is set to `false`, a VM will be considered to be in running state when it's in running state for OpenNebula (`LCM_STATE==3` and `STATE>=3`). Take into account that the VM will be considered `RUNNING` the very same moment the hypervisor boots the VM (before it loads the OS).

### Configure Dynamic Networks


A Service Role has a Virtual Machine Template assigned. The VM Template will define the capacity, disks, and network interfaces. But instead of using the Virtual Networks set in the VM Template, the Service Template can define a set of dynamic networks.


#### Network Configuration

Name	Description	
INTERNAL	Private network for the service traffic	
PUBLIC	Network with access to public IPs	
<a href="#">+ Add another Network</a>		

Each Role can be attached to the dynamic networks individually.

## Roles

  
master ✕

  
slave ✕

+ Add another role


Role Name ?

VM template ?

2: centos ▾

VMs ?

3

 Network Interfaces

---

INTERNAL

PUBLIC

Parent roles

---



master

▼ Role Elasticity

▼ Advanced Role Parameters

When a Service Template defines dynamic networks, the instantiate dialog will ask the user to select the networks to use for the new Service.

## Instantiate Service Template

Service Name   Number of Instances 

## Network

### Private network for the service traffic




ID	Owner	Group	Name	Reservation	Cluster	Leases
1	oneadmin	oneadmin	public	No	-	1 / 40
0	oneadmin	oneadmin	devs-private	No	-	2 / 100

« 1 »

You selected the following network: devs-private

### Network with access to public IPs



ID	Owner	Group	Name	Reservation	Cluster	Leases
1	oneadmin	oneadmin	public	No	-	1 / 40
0	oneadmin	oneadmin	devs-private	No	-	2 / 100

« 1 »

You selected the following network: public

This allows you to create more generic Service Templates. For example, the same Service Template can be used by users of different groups that may have access to different Virtual Networks.

### 1.3.3 Managing Services

A Service Template can be instantiated as a Service. Each newly created Service will be deployed by OneFlow following its deployment strategy.

Each Service Role creates Virtual Machines in OpenNebula from VM Templates, that must be created beforehand.

#### Create and List Existing Services

New Services are created from Service Templates, using the `onflow-template instantiate` command:



```
$ oneflow-template instantiate 0
ID: 1
```

To list the available Services, use `oneflow list/top`:

```
$ oneflow list
      ID USER          GROUP          NAME          STATE
      -- --          -
      1 oneadmin      oneadmin      my_service    PENDING
```

The screenshot displays the OpenNebula web interface for managing a service. The top navigation bar shows the user 'oneadmin' and the service name 'OneFlow - Service 5'. The left sidebar contains various system management options. The main area is divided into sections: 'Roles' and 'Role - DB'. The 'Roles' section features a table with columns for Name, State, Cardinality, VM Template, and Parents. The 'DB' role is highlighted. Below this, the 'Role - DB' section provides detailed information, including a 'Virtual Machines' table with columns for ID, Owner, Group, Name, Status, Host, IPs, and VNC. A single VM entry is shown with ID 13, owned by John, in the BlueVDC group, named DB\_0\_service\_5, and currently in a RUNNING state on host dev1 with IP 192.168.1.12.

The Service will eventually change to DEPLOYING. You can see information for each Role and individual Virtual Machine using `oneflow show`

```
$ oneflow show 1
SERVICE 1 INFORMATION
ID           : 1
NAME        : my_service
USER        : oneadmin
GROUP       : oneadmin
STRATEGY    : straight
SERVICE STATE : DEPLOYING

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

ROLE frontend
ROLE STATE  : RUNNING
CARNIDALITY : 1
VM TEMPLATE : 0
```

```

NODES INFORMATION
VM_ID NAME                STAT UCPU    UMEM HOST                TIME
  0 frontend_0_(service_1)  runn   67  120.3M localhost              0d 00h01

ROLE db_master
ROLE STATE                 : DEPLOYING
PARENTS                    : frontend
CARNIDALITY                : 1
VM TEMPLATE                : 1
NODES INFORMATION
VM_ID NAME                STAT UCPU    UMEM HOST                TIME
  1                        init      1      0K                       0d 00h00

ROLE db_slave
ROLE STATE                 : DEPLOYING
PARENTS                    : frontend
CARNIDALITY                : 3
VM TEMPLATE                : 2
NODES INFORMATION
VM_ID NAME                STAT UCPU    UMEM HOST                TIME
  2                        init      0      0K                       0d 00h00
  3                        init      0      0K                       0d 00h00
  4                        init      0      0K                       0d 00h00

ROLE worker
ROLE STATE                 : PENDING
PARENTS                    : db_master, db_slave
CARNIDALITY                : 10
VM TEMPLATE                : 3
NODES INFORMATION
VM_ID NAME                STAT UCPU    UMEM HOST                TIME

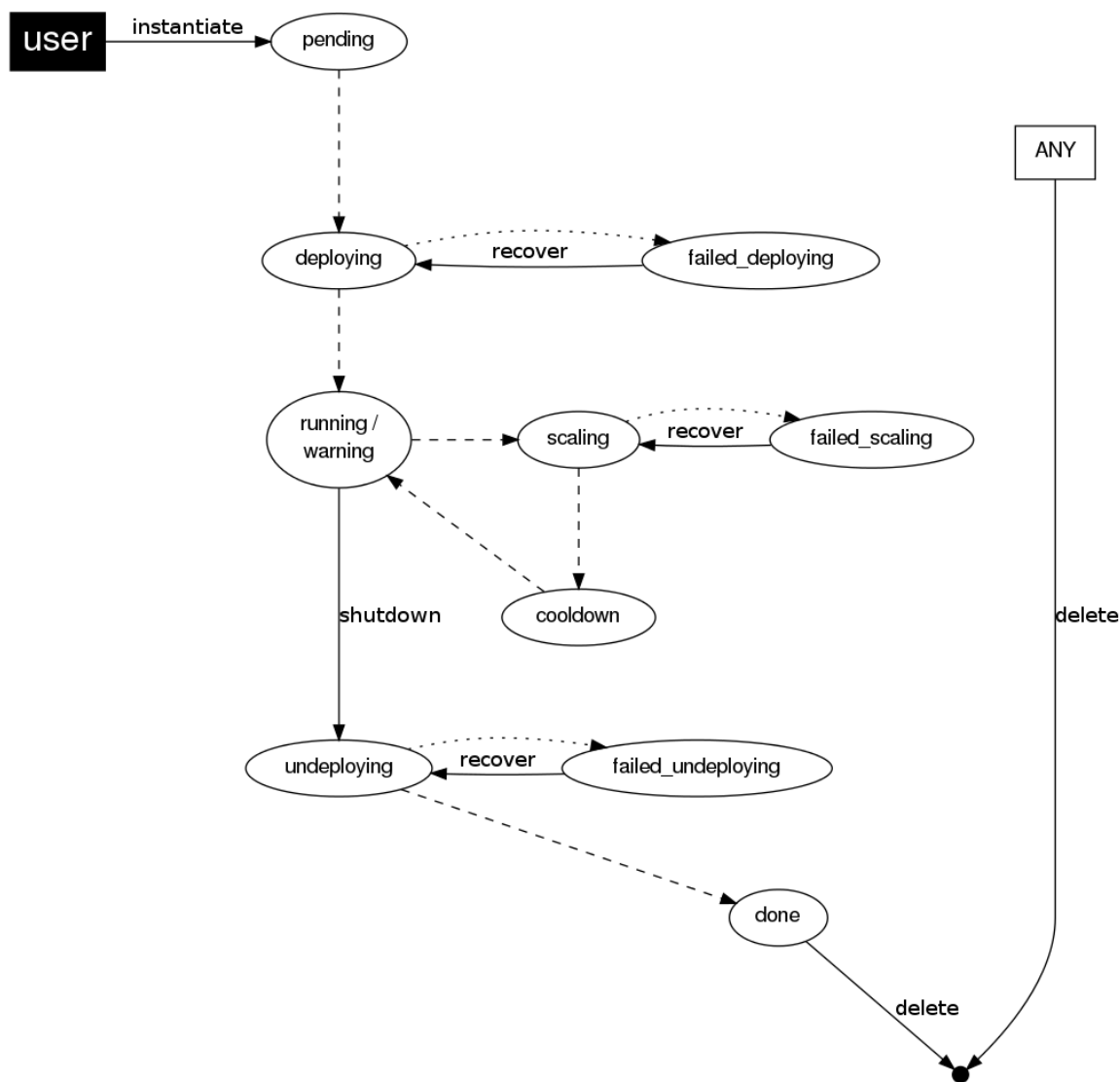
LOG MESSAGES
09/19/12 14:44 [I] New state: DEPLOYING
    
```

## Life-cycle

The deployment attribute defines the deployment strategy that the Life Cycle Manager (part of the *oneflow-server*) will use. These two values can be used:

- **none:** All roles are deployed at the same time.
- **straight:** Each Role is deployed when all its parent Roles are RUNNING.

Regardless of the strategy used, the Service will be RUNNING when all of the Roles are also RUNNING. Likewise, a Role will enter this state only when all the VMs are running.



This table describes the Service states:

Service State	Meaning
PENDING	The Service starts in this state, and will stay in it until the LCM decides to deploy it
DEPLOYING	Some Roles are being deployed
RUNNING	All Roles are deployed successfully
WARNING	A VM was found in a failure state
SCALING	A Role is scaling up or down
COOLDOWN	A Role is in the cooldown period after a scaling operation
UNDEPLOYING	Some Roles are being undeployed
DONE	The Service will stay in this state after a successful undeployment. It can be deleted
FAILED_DEPLOYING	An error occurred while deploying the Service
FAILED_UNDEPLOYING	An error occurred while undeploying the Service
FAILED_SCALING	An error occurred while scaling the Service

Each Role has an individual state, described in the following table:

Role State	Meaning
PENDING	The Role is waiting to be deployed
DEPLOYING	The VMs are being created, and will be monitored until all of them are running
RUNNING	All the VMs are running
WARNING	A VM was found in a failure state
SCALING	The Role is waiting for VMs to be deployed or to be shutdown
COOLDOWN	The Role is in the cooldown period after a scaling operation
UNDEPLOYING	The VMs are being shutdown. The role will stay in this state until all VMs are done
DONE	All the VMs are done
FAILED_DEPLOYING	An error occurred while deploying the VMs
FAILED_UNDEPLOYING	An error occurred while undeploying the VMs
FAILED_SCALING	An error occurred while scaling the Role

## Life-Cycle Operations

Services are deployed automatically by the Life Cycle Manager. To undeploy a running Service, users have the commands `onflow shutdown` and `onflow delete`.

The command `onflow shutdown` will perform a graceful shutdown of all the running VMs, and will delete any VM in a failed state (see `onevm shutdown` and `delete`). If the `straight` deployment strategy is used, the Roles will be shutdown in the reverse order of the deployment.

After a successful shutdown, the Service will remain in the `DONE` state. If any of the VM shutdown operations cannot be performed, the Service state will show `FAILED`, to indicate that manual intervention is required to complete the cleanup. In any case, the Service can be completely removed using the command `onflow delete`.

If a Service and its VMs must be immediately undeployed, the command `onflow delete` can be used from any Service state. This will execute a delete operation for each VM and delete the Service. Please be aware that **this is not recommended**, because VMs using persistent Images can leave them in an inconsistent state.

When a Service fails during a deployment, undeployment or scaling operation, the command `onflow recover` can be used to retry the previous action once the problem has been solved.

## Elasticity

A role's cardinality can be adjusted manually, based on metrics, or based on a schedule. To start the scalability immediately, use the command `onflow scale`:

```
$ onflow scale <serviceid> <role_name> <cardinality>
```

To define automatic elasticity policies, proceed to the [elasticity documentation guide](#).

## Sharing Information between VMs

The Virtual Machines of a Service can share information with each other, using the [OneGate server](#). OneGate allows Virtual Machine guests to push information to OpenNebula, and pull information about their own VM or Service.

From any VM, use the `PUT ${ONEGATE_ENDPOINT}/vm` action to store any information in the VM user template. This information will be in the form of `attribute=value`, e.g. `ACTIVE_TASK = 13`. Other VMs in the Service can request that information using the `GET ${ONEGATE_ENDPOINT}/service` action.

You can read more details in the [OneGate API documentation](#).

### 1.3.4 Managing Permissions

Both Services and Template resources are completely integrated with the OpenNebula user and group management. This means that each resource has an owner and group, and permissions. The VMs created by a Service are owned by the Service owner, so he can list and manage them.

For example, to change the owner and group of the Service 1, we can use `oneflow chown/chgrp`:

```
$ oneflow list
      ID USER          GROUP          NAME          STATE
      1 oneadmin       oneadmin       my_service     RUNNING

$ onevm list
      ID USER          GROUP          NAME          STAT UCPU    UMEM HOST          TIME
      0 oneadmin       oneadmin       frontend_0_(ser runn  17   43.5M localhost  0d 01h06
      1 oneadmin       oneadmin       db_master_0_(se runn  59  106.2M localhost  0d 01h06
...

$ oneflow chown my_service johndoe apptools

$ oneflow list
      ID USER          GROUP          NAME          STATE
      1 johndoe        apptools       my_service     RUNNING

$ onevm list
      ID USER          GROUP          NAME          STAT UCPU    UMEM HOST          TIME
      0 johndoe        apptools       frontend_0_(ser runn  62   83.2M localhost  0d 01h16
      1 johndoe        apptools       db_master_0_(se runn  74  115.2M localhost  0d 01h16
...
```

Note that the Service's VM ownership is also changed.

All Services and Templates have associated permissions for the **owner**, the users in its **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, and can be modified with the command `chmod`.

For example, to allow all users in the `apptools` group to **USE** (list, show) and **MANAGE** (shutdown, delete) the Service 1:

```
$ oneflow show 1
SERVICE 1 INFORMATION
..

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

$ oneflow chmod my_service 660

$ oneflow show 1
SERVICE 1 INFORMATION
..

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

Another common scenario is having Service Templates created by oneadmin that can be instantiated by any user. To implement this scenario, execute:

```
$ oneflow-template show 0
SERVICE TEMPLATE 0 INFORMATION
ID                : 0
NAME              : my_service
USER              : oneadmin
GROUP             : oneadmin

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

$ oneflow-template chmod 0 604

$ oneflow-template show 0
SERVICE TEMPLATE 0 INFORMATION
ID                : 0
NAME              : my_service
USER              : oneadmin
GROUP             : oneadmin

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

Please refer to the OpenNebula documentation for more information about users & groups, and resource permissions.

### 1.3.5 Scheduling Actions on the Virtual Machines of a Role

You can use the `action` command to perform a VM action on all the Virtual Machines belonging to a role. For example, if you want to suspend the Virtual Machines of the worker Role:

```
$ oneflow action <service_id> <role_name> <vm_action>
```

These are the commands that can be performed:

- shutdown
- shutdown-hard
- undeploy
- undeploy-hard
- hold
- release
- stop
- suspend
- resume
- boot

- delete
- delete-recreate
- reboot
- reboot-hard
- poweroff
- poweroff-hard
- snapshot-create

Instead of performing the action immediately on all the VMs, you can perform it on small groups of VMs with these options:

- `-p`, `-period x`: Seconds between each group of actions
- `-n`, `-number x`: Number of VMs to apply the action to each period

Let's say you need to reboot all the VMs of a Role, but you also need to avoid downtime. This command will reboot 2 VMs each 5 minutes:

```
$ oneflow action my-service my-role reboot --period 300 --number 2
```

The `oneflow-server.conf` file contains default values for `period` and `number` that are used if you omit one of them.

### 1.3.6 Recovering from Failures

Some common failures can be resolved without manual intervention, calling the `oneflow recover` command. This command has different effects depending on the Service state:

State	New State	Recover action
FAILED_DEPLOYING	DEPLOYING	<b>VMs in DONE or FAILED are deleted.</b> VMs in UNKNOWN are booted.
FAILED_UNDEPLOYING	UNDEPLOYING	The undeployment is resumed.
FAILED_SCALING	SCALING	<b>VMs in DONE or FAILED are deleted.</b> VMs in UNKNOWN are booted. For a scale-down, the shut-down actions are retried.
COOLDOWN	RUNNING	The Service is simply set to running before the cooldown period is over.
WARNING	WARNING	<b>VMs in DONE or FAILED are deleted.</b> VMs in UNKNOWN are booted. New VMs are instantiated to maintain the current cardinality.

### 1.3.7 Service Template Reference

For more information on the resource representation, please check the API guide

Read the *elasticity policies documentation* for more information.

## 1.4 Application Auto-scaling

A role's cardinality can be adjusted manually, based on metrics, or based on a schedule.

### 1.4.1 Overview

When a scaling action starts, the Role and Service enter the SCALING state. In this state, the Role will instantiate or shutdown a number of VMs to reach its new cardinality.

A role with elasticity policies must define a minimum and maximum number of VMs:

```
"roles": [  
  {  
    "name": "frontend",  
    "cardinality": 1,  
    "vm_template": 0,  
  
    "min_vms" : 1,  
    "max_vms" : 5,  
    ...  
  }  
]
```

After the scaling, the Role and Service are in the COOLDOWN state for the configured duration. During a scale operation and the cooldown period, other scaling actions for the same or for other Roles are delayed until the Service is RUNNING again.





## Create Service Template

Name

Description

^ Network Configuration

Network Configuration

Name	Description
<input type="text" value="Private"/>	<input type="text" value="Private Network for internal communication"/>
<input type="text" value="Public"/>	<input type="text" value="Public IP Addresses"/>

+ Add another Network

^ Advanced Service Parameters

### Roles

Master

Slave

+ Add another role

Role Name

VM template  -

VMs

Network Interfaces	Parent roles
<input checked="" type="checkbox"/> Private	<input checked="" type="checkbox"/> Master
<input type="checkbox"/> Public	

^ Role Elasticity

^ Advanced Role Parameters

Reset

Create

## 1.4.2 Set the Cardinality of a Role Manually

The command `onflow scale` starts the scalability immediately.

```
$ onflow scale <serviceid> <role_name> <cardinality>
```

You can force a cardinality outside the defined range with the `--force` option.

### 1.4.3 Maintain the Cardinality of a Role

The 'min\_vms' attribute is a hard limit, enforced by the elasticity module. If the cardinality drops below this minimum, a scale-up operation will be triggered.

### 1.4.4 Set the Cardinality of a Role Automatically

#### Auto-scaling Types

Both `elasticity_policies` and `scheduled_policies` elements define an automatic adjustment of the Role cardinality. Three different adjustment types are supported:

- **CHANGE**: Add/subtract the given number of VMs
- **CARDINALITY**: Set the cardinality to the given number
- **PERCENTAGE\_CHANGE**: Add/subtract the given percentage to the current cardinality

Attribute	Type	Mandatory	Description
type	string	Yes	Type of adjustment. Values: CHANGE, CARDINALITY, PERCENTAGE_CHANGE
adjust	integer	Yes	Positive or negative adjustment. Its meaning depends on 'type'
min_adjust_step	integer	No	Optional parameter for PERCENTAGE_CHANGE adjustment type. If present, the policy will change the cardinality by at least the number of VMs set in this attribute.

#### Auto-scaling Based on Metrics

Each role can have an array of `elasticity_policies`. These policies define an expression that will trigger a cardinality adjustment.

These expressions can use performance data from

- The VM guest. Using the *OneGate server*, applications can send custom monitoring metrics to OpenNebula.
- The VM, at hypervisor level. The Virtualization Drivers return information about the VM, such as CPU, MEMORY, NETTX and NETRX.

```
"elasticity_policies" : [
  {
    "expression" : "ATT > 50",
    "type" : "CHANGE",
    "adjust" : 2,

    "period_number" : 3,
    "period" : 10
  },
  ...
]
```

The **expression** can use VM attribute names, float numbers, and logical operators (!, &, |). When an attribute is found, it will take the **average** value for all the **running VMs** that contain that attribute in the Role. If none of the VMs contain the attribute, the expression will evaluate to false.

The attribute will be looked for in /VM/USER\_TEMPLATE, /VM/MONITORING, /VM/TEMPLATE and /VM, in that order. Logical operators have the usual precedence.

Attribute	Type	Mandatory	Description
expression	string	Yes	Expression to trigger the elasticity
period_number	integer	No	Number of periods that the expression must be true before the elasticity is triggered
period	integer	No	Duration, in seconds, of each period in period_number

### Auto-scaling Based on a Schedule

Combined with the elasticity policies, each role can have an array of `scheduled_policies`. These policies define a time, or a time recurrence, and a cardinality adjustment.

```
"scheduled_policies" : [
  {
    // Set cardinality to 2 each 10 minutes
    "recurrence" : "*/10 * * * *",

    "type" : "CARDINALITY",
    "adjust" : 2
  },
  {
    // +10 percent at the given date and time
    "start_time" : "2nd oct 2013 15:45",

    "type" : "PERCENTAGE_CHANGE",
    "adjust" : 10
  }
]
```

Attribute	Type	Mandatory	Description
recurrence	string	No	Time for recurring adjustments. Time is specified with the <a href="#">Unix cron syntax</a>
start_time	string	No	Exact time for the adjustment

### 1.4.5 Visualize in the CLI

The `onflow show / top` commands show the defined policies. When a service is scaling, the VMs being created or shutdown can be identified by an arrow next to their ID:

```
SERVICE 7 INFORMATION
...

ROLE frontend
ROLE STATE           : SCALING
CARNIDALITY         : 4
VM TEMPLATE         : 0
NODES INFORMATION
VM_ID NAME          STAT UCPU   UMEM HOST          TIME
  4 frontend_0_(service_7)  runn   0   74.2M host03          0d 00h04
  5 frontend_1_(service_7)  runn   0  112.6M host02          0d 00h04
  | 6                  init           0K                               0d 00h00
  | 7                  init           0K                               0d 00h00
```

```
ELASTICITY RULES
MIN VMS           : 1
MAX VMS           : 5

ADJUST           EXPRESSION                               EVALUATION PERIOD
+ 2              (ATT > 50) && !(OTHER_ATT = 5.5 || ABC <= 30) 0 / 3           10s
- 10 % (2)      ATT < 20                                     0 / 1           0s

ADJUST           TIME
= 6              0 9 * * mon,tue,wed,thu,fri
= 10             0 13 * * mon,tue,wed,thu,fri
= 2              30 22 * * mon,tue,wed,thu,fri

LOG MESSAGES
06/10/13 18:22 [I] New state: DEPLOYING
06/10/13 18:22 [I] New state: RUNNING
06/10/13 18:26 [I] Role frontend scaling up from 2 to 4 nodes
06/10/13 18:26 [I] New state: SCALING
```

## 1.4.6 Interaction with Individual VM Management

All the VMs created by a Service can be managed as regular VMs. When VMs are monitored in an unexpected state, this is what OneFlow interprets:

- VMs in a recoverable state ('suspend', 'poweroff', etc.) are considered are healthy machines. The user will eventually decide to resume these VMs, so OneFlow will keep monitoring them. For the elasticity module, these VMs are just like 'running' VMs.
- VMs in the final 'done' state are cleaned from the Role. They do not appear in the nodes information table, and the cardinality is updated to reflect the new number of VMs. This can be seen as an manual scale-down action.
- VMs in 'unknown' or 'failed' are in an anomalous state, and the user must be notified. The Role and Service are set to the 'WARNING' state.

OpenNebula Sunstone OneFlow - Service 5

oneadmin OpenNebula

Dashboard System Virtual Resources Virtual Machines Templates Images Files & Kernels Infrastructure Marketplace OneFlow Services Templates Support

Shutdown Recover

Info Roles Log

Roles

Period: Number:

+ Scale

Name	State	Cardinality	VM Template	Parents
Frontend	RUNNING	1	0	-
DB	RUNNING	1	0	-

Role - DB

Information

Shutdown action	Cooldown	Min VMs	Max VMs
-	-	-	-

Virtual Machines

ID	Owner	Group	Name	Status	Host	IPs	VNC
13	John	BlueVDC	DB_0_service_5	RUNNING	dev1	192.168.1.12	

Showing 1 to 1 of 1 entries

## 1.4.7 Examples

```

/*
Testing:

1) Update one VM template to contain
ATT = 40
and the other VM with
ATT = 60

Average will be 50, true evaluation periods will not increase in CLI output

2) Increase first VM ATT value to 45. True evaluations will increase each
10 seconds, the third time a new VM will be deployed.

3) True evaluations are reset. Since the new VM does not have ATT in its
template, the average will be still bigger than 50, and new VMs will be
deployed each 30s until the max of 5 is reached.

4) Update VM templates to trigger the scale down expression. The number of
VMs is adjusted -10 percent. Because  $5 * 0.10 < 1$ , the adjustment is rounded to 1;
but the min_adjust_step is set to 2, so the final adjustment is -2 VMs.
*/
{
  "name": "Scalability1",
  "deployment": "none",
  "roles": [

```

```

{
  "name": "frontend",
  "cardinality": 2,
  "vm_template": 0,

  "min_vms" : 1,
  "max_vms" : 5,

  "elasticity_policies" : [
    {
      // +2 VMs when the exp. is true for 3 times in a row,
      // separated by 10 seconds
      "expression" : "ATT > 50",

      "type" : "CHANGE",
      "adjust" : 2,

      "period_number" : 3,
      "period" : 10
    },
    {
      // -10 percent VMs when the exp. is true.
      // If 10 percent is less than 2, -2 VMs.
      "expression" : "ATT < 20",

      "type" : "PERCENTAGE_CHANGE",
      "adjust" : -10,
      "min_adjust_step" : 2
    }
  ]
}

```

```

{
  "name": "Time_windows",
  "deployment": "none",
  "roles": [
    {
      "name": "frontend",
      "cardinality": 1,
      "vm_template": 0,

      "min_vms" : 1,
      "max_vms" : 15,

      // These policies set the cardinality to:
      // 6 from 9:00 to 13:00
      // 10 from 13:00 to 22:30
      // 2 from 22:30 to 09:00, and the weekend

      "scheduled_policies" : [
        {
          "type" : "CARDINALITY",
          "recurrence" : "0 9 * * mon,tue,wed,thu,fri",
          "adjust" : 6
        },
        {

```

```
    "type" : "CARDINALITY",
    "recurrence" : "0 13 * * mon,tue,wed,thu,fri",
    "adjust" : 10
  },
  {
    "type" : "CARDINALITY",
    "recurrence" : "30 22 * * mon,tue,wed,thu,fri",
    "adjust" : 2
  }
]
}
]
```





## DATA CENTER FEDERATION

### 2.1 Data Center Federation

Several OpenNebula instances can be configured as a **Federation**. Each instance of the Federation is called a **Zone**, and they are configured as one master and several slaves.

An OpenNebula Federation is a tightly coupled integration. All the instances will share the same user accounts, groups, and permissions configuration. Of course, access can be restricted to certain Zones, and also to specific Clusters inside that Zone.

The typical scenario for an OpenNebula Federation is a company with several Data Centers, distributed in different geographic locations. This low-level integration does not rely on APIs, administrative employees of all Data Centers will collaborate on the maintenance of the infrastructure. If your use case requires a synergy with an external cloud infrastructure, that would fall into the cloudbursting scenario.

For the end users, a Federation allows them to use the resources allocated by the Federation Administrators no matter where they are. The integration is seamless, meaning that a user logged into the Sunstone web interface of a Zone will not have to log out and enter the address of the other Zone. Sunstone allows to change the active Zone at any time, and it will automatically redirect the requests to the right OpenNebula at the target Zone.

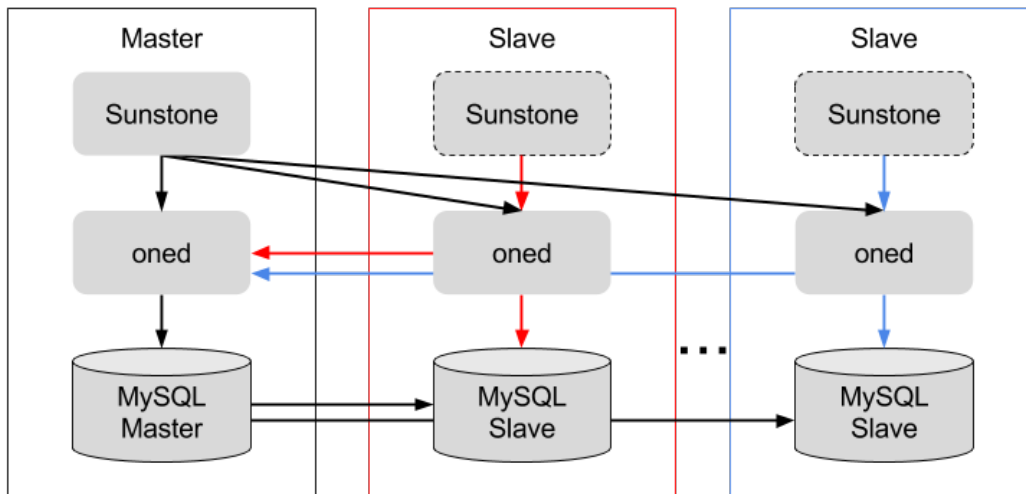
#### 2.1.1 Architecture

In a Federation, there is a master OpenNebula zone and several slaves sharing the database tables for users, groups, VDCs, ACL rules, and zones. The master OpenNebula is the only one that writes in the shared tables, while the slaves keep a read-only local copy, and proxy any writing actions to the master. This allows us to guarantee data consistency, without any impact on the speed of read-only actions.

The synchronization is achieved configuring MySQL to replicate certain tables only. MySQL's replication is able to perform over long-distance or unstable connections. Even if the master zone crashes and takes a long time to reboot, the slaves will be able to continue working normally except for a few actions such as new user creation or password updates.

New slaves can be added to an existing Federation at any moment. Moreover, the administrator can add a clean new OpenNebula, or import an existing deployment into the Federation keeping the current users, groups, configuration, and virtual resources.

Regarding the OpenNebula updates, we have designed the database in such a way that different OpenNebula versions will be able to be part of the same Federation. While an upgrade of the local tables (VM, Image, VNet objects) will be needed, new versions will keep compatibility with the shared tables. In practice, this means that when a new OpenNebula version comes out each zone can be updated at a different pace, and the Federation will not be affected.



To enable users to change zones, Sunstone server is connected to all the oned daemons in the Federation. You can have one Sunstone for all the Federation, or run one Sunstone for each Zone.

Regarding the administrator users, a Federation will have a unique oneadmin account. That is the Federation Administrator account. In a trusted environment, each Zone Administrator will log in with an account in the 'oneadmin' group. In other scenarios, the Federation Administrator can create a special administrative group with total permissions for one zone only.

The administrators can share appliances across Zones deploying a private OpenNebula Marketplace.

## 2.1.2 Other Services

Although a single Sunstone server can connect to different Zones, all the other OpenNebula services will only work with the local Zone resources. This includes the Scheduler, the *Public Cloud Servers*, *OneFlow*, and *OneGate*.

## 2.1.3 Next Steps

Continue to the following guides to learn how to configure and manage a Federation:

- *Federation Configuration*
- *Federation Management*

## 2.2 OpenNebula Federation Configuration

This section will explain how to configure two (or more) OpenNebula zones to work as federation master and slave. The process described here can be applied to new installations, or existing OpenNebula instances.

MySQL needs to be configured to enable the master-slave replication. Please read [the MySQL documentation for your version](#) for complete instructions. The required steps are summarized here, but it may happen that your MySQL version needs a different configuration.

**Warning:** If Sunstone is configured behind a proxy please make sure that the request headers are being *properly sent*.

## 2.2.1 1. Configure the OpenNebula Federation Master

- Start with an existing OpenNebula, or install OpenNebula as usual following the installation guide. For new installations, you may need to create a MySQL user for OpenNebula, read more in the MySQL configuration guide.

```
# mysql -u root -p
mysql> GRANT ALL PRIVILEGES ON opennebula.* TO 'oneadmin' IDENTIFIED BY 'oneadmin';
```

- Configure OpenNebula to use the **master MySQL**, and to act as a **federation master**.

```
# vi /etc/one/oned.conf
#DB = [ backend = "sqlite" ]

# Sample configuration for MySQL
DB = [ backend = "mysql",
        server = "<ip>",
        port   = 0,
        user   = "oneadmin",
        passwd = "oneadmin",
        db_name = "opennebula" ]

FEDERATION = [
    MODE = "MASTER",
    ZONE_ID = 0,
    MASTER_ONED = ""
]
```

- Restart OpenNebula
- Edit the local (master) Zone Endpoint. This can be done via Sunstone, or with the `onezone` command.

```
$ onezone update 0
ENDPOINT = http://<master-ip>:2633/RPC2
```

- Create a Zone for each one of the slaves, and write down the new Zone ID. This can be done via Sunstone, or with the `onezone` command.

```
$ vim /tmp/zone.tmpl
NAME      = slave-name
ENDPOINT  = http://<slave-ip>:2633/RPC2

$ onezone create /tmp/zone.tmpl
ID: 100

$ onezone list
  ID NAME
  0  OpenNebula
 100 slave-name
```

- Stop OpenNebula.

## 2.2.2 2. Import the Existing Slave OpenNebula

---

**Note:** If your slave OpenNebula is going to be installed from scratch, you can skip this step.

---

If the OpenNebula to be added as a Slave is an existing installation, and you need to preserve its database (users, groups, VMs, hosts...), you need to import the contents with the `onedb` command.

- Stop the slave OpenNebula. Make sure the master OpenNebula is also stopped.
- Run the `onedb import-slave` command. Use `-h` to get an explanation of each option.

```
$ onedb import-slave -h
## USAGE
import-slave
    Imports an existing federation slave into the federation master database

## OPTIONS
...

$ onedb import-slave -v \
--username oneadmin --password oneadmin \
--server 192.168.122.3 --dbname opennebula \
--slave-username oneadmin --slave-password oneadmin \
--slave-server 192.168.122.4 --slave-database opennebula
```

The tool will ask for the Zone ID you created in step 1.

```
Please enter the Zone ID that you created to represent the new Slave OpenNebula:
Zone ID:
```

You will also need to decide if the users, groups and VDCs will be merged.

If you had different people using the master and slave OpenNebula instances, then choose not to merge users. In case of name collision, the slave account will be renamed to `username-1`.

You will want to merge if your users were accessing both the master and slave OpenNebula instances before the federation. To put it more clearly, the same person had previous access to the `alice` user in master and `alice` user in the slave. This will be the case if, for example, you had more than one OpenNebula instances pointing to the same LDAP server for authentication.

When a user is merged, its user template is also copied, using the master contents in case of conflict. This means that if `alice` had a different password or `'SSH_KEY'` in her master and slave OpenNebula users, only the one in master will be preserved.

In any case, the ownership of existing resources and group membership is preserved.

```
The import process will move the users from the slave OpenNebula to the master
OpenNebula. In case of conflict, it can merge users with the same name.
For example:
```

Master	Slave	With merge	Without merge
5, alice	2, alice	5, alice	5, alice
6, bob	5, bob	6, bob	6, bob
			7, alice-1
			8, bob-1

In any case, the ownership of existing resources and group membership is preserved.

```
Do you want to merge USERS (Y/N): y
```

```
Do you want to merge GROUPS (Y/N): y
```

When the import process finishes, `onedb` will write in `/var/log/one/onedb-import.log` the new user IDs and names if they were renamed.

### 2.2.3 3. Configure the MySQL Replication Master

- In your **master MySQL**: enable the binary log for the opennebula database and set a server ID. Change the 'opennebula' database name to the one set in oned.conf.

```
# vi /etc/my.cnf
[mysqld]
log-bin                = mysql-bin
server-id              = 1
binlog-do-db          = opennebula

# service mysqld restart
```

- **Master MySQL**: You also need to create a special user that will be used by the MySQL replication slaves.

```
# mysql -u root -p
mysql> CREATE USER 'one-slave'@'%' IDENTIFIED BY 'one-slave-pass';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'one-slave'@'%';
```

**Warning:** In the previous example we are granting access to user one-replication from any host. You may want to restrict the hosts with the hostnames of the mysql slaves

- **Master MySQL**: Lock the tables and perform a dump.

First you need to lock the tables before dumping the federated tables.

```
mysql> FLUSH TABLES WITH READ LOCK;
```

Then you can safely execute the mysqldump command in another terminal. Please note the `--master-data` option, it must be present to allow the slaves to know the current position of the binary log.

```
mysqldump -u root -p --master-data opennebula user_pool group_pool vdc_pool zone_pool db_versioning a
```

Once you get the dump you can unlock the DB tables again.

```
mysql> UNLOCK TABLES;
```

- MySQL replication cannot use Unix socket files. You must be able to connect from the slaves to the master MySQL server using TCP/IP and port 3306 (default mysql port). Please update your firewall accordingly.
- You can start the master OpenNebula at this point.

### 2.2.4 4. Configure the MySQL Replication Slave

For each one of the slaves, configure the MySQL server as a replication slave. Pay attention to the `server-id` set in `my.cnf`, it must be unique for each one.

- Set a server ID for the **slave MySQL**, and configure these tables to be replicated. You may need to change 'opennebula' to the database name used in oned.conf. The database name must be the same for the master and slaves OpenNebulas.

```
# vi /etc/my.cnf
[mysqld]
server-id              = 100
replicate-do-table    = opennebula.user_pool
replicate-do-table    = opennebula.group_pool
replicate-do-table    = opennebula.vdc_pool
replicate-do-table    = opennebula.zone_pool
replicate-do-table    = opennebula.db_versioning
```

```
replicate-do-table = opennebula.acl  
  
# service mysqld restart
```

- Set the master configuration on the **slave MySQL**.

```
# mysql -u root -p  
mysql> CHANGE MASTER TO  
->     MASTER_HOST='master_host_name',  
->     MASTER_USER='one-slave',  
->     MASTER_PASSWORD='one-slave-pass';
```

- Copy the mysql dump file from the **master**, and import its contents to the **slave**.

```
mysql> CREATE DATABASE IF NOT EXISTS opennebula;  
mysql> USE opennebula;  
mysql> SOURCE /path/to/dump.sql;
```

- Start the **slave MySQL** process and check its status.

```
mysql> START SLAVE;  
mysql> SHOW SLAVE STATUS\G
```

The SHOW SLAVE STATUS output will provide detailed information, but to confirm that the slave is connected to the master MySQL, take a look at these columns:

```
Slave_IO_State: Waiting for master to send event  
Slave_IO_Running: Yes  
Slave_SQL_Running: Yes
```

## 2.2.5 5. Configure the OpenNebula Federation Slave

For each slave, follow these steps.

- If it is a new installation, install OpenNebula as usual following the installation guide.
- Configure OpenNebula to use MySQL, first you'll need to create a database user for OpenNebula and grant access to the OpenNebula database:

```
# mysql -u root -p  
mysql> GRANT ALL PRIVILEGES ON opennebula.* TO 'oneadmin' IDENTIFIED BY 'oneadmin';
```

and update oned.conf to use these values:

```
# vi /etc/one/oned.conf  
#DB = [ backend = "sqlite" ]  
  
# Sample configuration for MySQL  
DB = [ backend = "mysql",  
      server = "<ip>",  
      port   = 0,  
      user   = "oneadmin",  
      passwd = "oneadmin",  
      db_name = "opennebula" ]
```

- Configure OpenNebula to act as a **federation slave**. Remember to use the ID obtained when the zone was created.

```
FEDERATION = [
  MODE = "SLAVE",
  ZONE_ID = 100,
  MASTER_ONED = "http://<oned-master-ip>:2633/RPC2"
]
```

- Copy the directory `/var/lib/one/.one` from the **master** front-end to the **slave**. This directory and its contents must have **oneadmin as owner**. The directory should contain these files:

```
$ ls -l /var/lib/one/.one
ec2_auth
one_auth
oneflow_auth
onegate_auth
sunstone_auth
```

Make sure `one_auth` (the oneadmin credentials) is present. If it's not, copy it from **master** oneadmin's `$HOME/.one` to the **slave** oneadmin's `$HOME/.one`. For most configurations, oneadmin's home is `/var/lib/one` and this won't be necessary.

- Start the slave OpenNebula.

## 2.3 OpenNebula Federation Management

The administrator of a federation has the ability to add or remove Zones from the federation. See this guide for details on how to configure the federation in both the master and the slave of the OpenNebula federation.

A user will have access to all the Zones where at least one of her groups has VDC resources in. This access can be done through Sunstone or through the CLI

### 2.3.1 Adding a Zone

Adding a Zone through the CLI entails the creation of a Zone template.

Parameter	Description
Name	Name of the new Zone
Endpoint	XMLRPC endpoint of the OpenNebula

```
# vi zone.tmpl
NAME = ZoneB
ENDPOINT = http://zoneb.opennebula.front-end.server:2633/RPC2
```

This same operation can be performed through Sunstone (Zone tab -> Create).

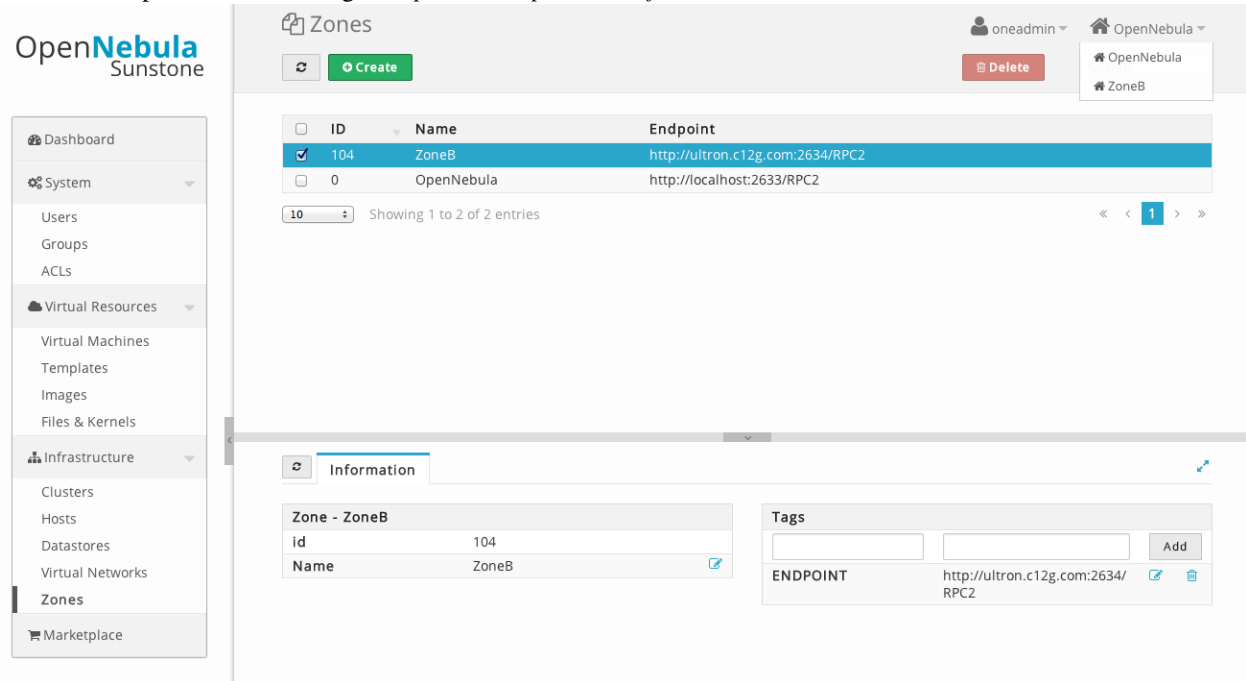
**Warning:** The ENDPOINT has to be reachable from the Sunstone server machine, or the computer running the CLI in order for the user to access the Zone.

### 2.3.2 Using a Zone

#### Through Sunstone

In the upper right position of Sunstone page, users will see a house icon next to the name of the Zone you are currently using. If the user clicks on that, she will get a dropdown with all the Zones she has access to. Clicking on any of the Zones in the dropdown will get the user to that Zone.

What’s happening behind the scenes is that the Sunstone server you are connecting to is redirecting its requests to the OpenNebula oned process present in the other Zone. In the example above, if the user clicks on ZoneB, Sunstone will contact the OpenNebula listening at `http://zoneb.opennebula.front-end.server:2633/RPC2`.



**Note:** Uploading an image functionality is limited to the zone where the Sunstone instance the user is connecting to, even if it can switch to other federated zones.

### Through CLI

Users can switch Zones through the command line using the `onezone` command. The following session can be examined to understand the Zone management through the CLI.

```
$ onezone list
C   ID NAME                               ENDPOINT
*   0  OpenNebula                          http://localhost:2633/RPC2
    104 ZoneB                               http://ultron.c12g.com:2634/RPC2
```

We can see in the above command output that the user has access to both “OpenNebula” and “ZoneB”, and it is currently in the “OpenNebula” Zone. To change the active Zone can be changed using the ‘set’ command of `onezone`:

```
$ onezone set 104
Endpoint changed to "http://ultron.c12g.com:2634/RPC2" in /home/<username>/.one/one_endpoint

$ onezone list
C   ID NAME                               ENDPOINT
    0  OpenNebula                          http://localhost:2633/RPC2
*   104 ZoneB                               http://ultron.c12g.com:2634/RPC2
```

All the subsequent CLI commands executed would connect to the OpenNebula listening at “`http://zoneb.opennebula.front-end.server:2633/RPC2`”.



## SCALABILITY

### 3.1 Configuring Sunstone for Large Deployments

Low to medium enterprise clouds will typically deploy Sunstone in a single machine along with the OpenNebula daemons. However this simple deployment can be improved by:

- Isolating the access from Web clients to the Sunstone server. This can be achieved by deploying the Sunstone server in a separated machine.
- Improve the scalability of the server for large user pools. Usually deploying sunstone in a separate application container in one or more hosts.

Check also the *api scalability guide* as these of the tips also have an impact on Sunstone performance.

#### 3.1.1 Deploying Sunstone in a Different Machine

By default the Sunstone server is configured to run in the frontend, but you are able to install the Sunstone server in a machine different from the frontend.

- You will need to install only the sunstone server packages in the machine that will be running the server. If you are installing from source use the `-s` option for the `install.sh` script.
- Make sure `:one_xmlprc:` variable in `sunstone-server.conf` points to the right place where OpenNebula frontend is running, You can also leave it undefined and export `ONE_XMLRPC` environment variable.
- Provide the serveradmin credentials in the following file `/var/lib/one/.one/sunstone_auth`. If you changed the serveradmin password please check the Cloud Servers Authentication guide.
- If you want to upload files to OpenNebula, you will have to share the upload directory (`/var/tmp` by default) between sunstone and oned. Some server do not take into account the `TMPDIR` env var and this directory must be defined in the configuration file, for example in Passenger (`client_body_temp_path`)

```
$ cat /var/lib/one/.one/sunstone_auth
serveradmin:1612b78a4843647a4b541346f678f9e1b43bbcf9
```

Using this setup the VirtualMachine logs will not be available. If you need to retrieve this information you must deploy the server in the frontend

#### 3.1.2 Running Sunstone Inside Another Webserver

Self contained deployment of Sunstone (using `sunstone-server` script) is ok for small to medium installations. This is no longer true when the service has lots of concurrent users and the number of objects in the system is high (for example, more than 2000 simultaneous virtual machines).

Sunstone server was modified to be able to run as a `rack` server. This makes it suitable to run in any web server that supports this protocol. In ruby world this is the standard supported by most web servers. We now can select web servers that support spawning multiple processes like `unicorn` or embedding the service inside `apache` or `nginx` web servers using the `Passenger` module. Another benefit will be the ability to run Sunstone in several servers and balance the load between them.

**Warning:** Deploying Sunstone behind a proxy in a federated environment requires some specific configuration to properly handle the Sunstone headers required by the Federation.

- **nginx:** enable `underscores_in_headers on;` and `proxy_pass_request_headers on;`

### Configuring memcached

When using one on these web servers the use of a `memcached` server is necessary. Sunstone needs to store user sessions so it does not ask for user/password for every action. By default Sunstone is configured to use memory sessions, that is, the sessions are stored in the process memory. Thin and webrick web servers do not spawn new processes but new threads and all of them have access to that session pool. When using more than one process to server Sunstone there must be a service that stores this information and can be accessed by all the processes. In this case we will need to install `memcached`. It comes with most distributions and its default configuration should be ok. We will also need to install ruby libraries to be able to access it. The rubygem library needed is `memcache-client`. If there is no package for your distribution with this ruby library you can install it using `rubygems`:

```
$ sudo gem install memcache-client
```

Then you will have to change in sunstone configuration (`/etc/one/sunstone-server.conf`) the value of `:sessions` to `memcache`.

If you want to use `novnc` you need to have it running. You can start this service with the command:

```
$ novnc-server start
```

Another thing you have to take into account is the user on which the server will run. The installation sets the permissions for `oneadmin` user and group and files like the Sunstone configuration and credentials can not be read by other users. Apache usually runs as `www-data` user and group so to let the server run as this user the group of these files must be changed, for example:

```
$ chgrp www-data /etc/one/sunstone-server.conf
$ chgrp www-data /etc/one/sunstone-plugins.yaml
$ chgrp www-data /var/lib/one/.one/sunstone_auth
$ chmod a+x /var/lib/one
$ chmod a+x /var/lib/one/.one
$ chgrp www-data /var/log/one/sunstone*
$ chmod g+w /var/log/one/sunstone*
```

We advise to use `Passenger` in your installation but we will show you how to run Sunstone inside `unicorn` web server as an example.

For more information on web servers that support `rack` and more information about it you can check the [rack documentation](#) page. You can alternatively check a [list of ruby web servers](#).

### Running Sunstone with Unicorn

To get more information about this web server you can go to its [web page](#). It is a multi process web server that spawns new processes to deal with requests.

The installation is done using `rubygems` (or with your package manager if it is available):

```
$ sudo gem install unicorn
```

In the directory where Sunstone files reside (`/usr/lib/one/sunstone` or `/usr/share/opennebula/sunstone`) there is a file called `config.ru`. This file is specific for rack applications and tells how to run the application. To start a new server using `unicorn` you can run this command from that directory:

```
$ unicorn -p 9869
```

Default unicorn configuration should be ok for most installations but a configuration file can be created to tune it. For example, to tell unicorn to spawn 4 processes and write `stderr` to `/tmp/unicorn.log` we can create a file called `unicorn.conf` that contains:

```
worker_processes 4
logger debug
stderr_path '/tmp/unicorn.log'
```

and start the server and daemonize it using:

```
$ unicorn -d -p 9869 -c unicorn.conf
```

You can find more information about the configuration options in the [unicorn documentation](#).

## Running Sunstone with Passenger in Apache

[Phusion Passenger](#) is a module for [Apache](#) and [Nginx](#) web servers that runs ruby rack applications. This can be used to run Sunstone server and will manage all its life cycle. If you are already using one of these servers or just feel comfortable with one of them we encourage you to use this method. This kind of deployment adds better concurrency and lets us add an `https` endpoint.

We will provide the instructions for Apache web server but the steps will be similar for `nginx` following [Passenger documentation](#).

First thing you have to do is install Phusion Passenger. For this you can use pre-made packages for your distribution or follow the [installation instructions](#) from their web page. The installation is self explanatory and will guide you in all the process, follow them and you will be ready to run Sunstone.

Next thing we have to do is configure the virtual host that will run our Sunstone server. We have to point to the `public` directory from the Sunstone installation, here is an example:

```
<VirtualHost *:80>
  ServerName sunstone-server
  PassengerUser oneadmin
  # !!! Be sure to point DocumentRoot to 'public'!
  DocumentRoot /usr/lib/one/sunstone/public
  <Directory /usr/lib/one/sunstone/public>
    # This relaxes Apache security settings.
    AllowOverride all
    # MultiViews must be turned off.
    Options -MultiViews
  </Directory>
</VirtualHost>
```

**Note:** When you're experiencing login problems you might want to set `PassengerMaxInstancesPerApp 1` in your passenger configuration or try `memcached` since Sunstone does not support sessions across multiple server instances.

Now the configuration should be ready, restart -or reload apache configuration- to start the application and point to the virtual host to check if everything is running.

### Running Sunstone behind nginx SSL Proxy

How to set things up with nginx ssl proxy for sunstone and encrypted vnc.

```
# No squealing.
server_tokens off;

# OpenNebula Sunstone upstream
upstream sunstone {
    server 127.0.0.1:9869;
}

# HTTP virtual host, redirect to HTTPS
server {
    listen 80 default_server;
    return 301 https://$server_name:443;
}

# HTTPS virtual host, proxy to Sunstone
server {
    listen 443 ssl default_server;
    ssl_certificate /etc/ssl/certs/opennebula-certchain.pem;
    ssl_certificate_key /etc/ssl/private/opennebula-key.pem;
    ssl_stapling on;
}
```

And this is the changes that have to be made to sunstone-server.conf:

```
UI Settings

:vnc_proxy_port: 29876
:vnc_proxy_support_wss: only
:vnc_proxy_cert: /etc/one/ssl/opennebula-certchain.pem
:vnc_proxy_key: /etc/one/ssl/opennebula-key.pem
:vnc_proxy_ipv6: false
```

If using a selfsigned cert, the connection to VNC window in Sunstone will fail, either get a real cert, or manually accept the selfsigned cert in your browser before trying it with Sunstone. Now, VNC sessions should show “encrypted” in the title.

### Running Sunstone with Passenger using FreeIPA/Kerberos auth in Apache

It is also possible to use Sunstone `remote` authentication with Apache and Passenger. The configuration in this case is quite similar to Passenger configuration but we must include the Apache auth module line. How to configure freeIPA server and Kerberos is outside of the scope of this document, you can get more info in [FreeIPA Apache setup example](#)

As example to include Kerberos authentication we can use two different modules: `mod_auth_gssapi` or `mod_authnz_pam` And generate the keytab for http service, here is an example with Passenger:

```
LoadModule auth_gssapi_module modules/mod_auth_gssapi.so

<VirtualHost *:80>
    ServerName sunstone-server
    PassengerUser oneadmin
```

```
# !!! Be sure to point DocumentRoot to 'public'!
DocumentRoot /usr/lib/one/sunstone/public
<Directory /usr/lib/one/sunstone/public>
  # Only is possible to access to this dir using a valid ticket
  AuthType GSSAPI
  AuthName "EXAMPLE.COM login"
  GssapiCredStore keytab:/etc/http.keytab
  Require valid-user
  ErrorDocument 401 '<html><meta http-equiv="refresh" content="0; URL=https://yourdomain"><body>K
  AllowOverride all
  # MultiViews must be turned off.
  Options -MultiViews
</Directory>
</VirtualHost>
```

**Note:** User must generate a valid ticket running `kinit` to get access to Sunstone service. You can also set a custom 401 document to warn users about any authentication failure.

Now our configuration is ready to use Passenger and Kerberos, restart -or reload apache configuration- and point to the virtual host using a valid ticket to check if everything is running.

### Running Sunstone in Multiple Servers

You can run Sunstone in several servers and use a load balancer that connects to them. Make sure you are using memcache for sessions and both Sunstone servers connect to the same memcached server. To do this change the parameter `:memcache_host` in the configuration file. Also make sure that both Sunstone instances connect to the same OpenNebula server.

## 3.2 Configuring OpenNebula for Large Deployments

### 3.2.1 Monitoring

OpenNebula supports two native monitoring systems: `ssh-pull` and `udp-push`. The former one, `ssh-pull` is the default monitoring system for OpenNebula  $\leq 4.2$ , however from OpenNebula 4.4 onwards, the default monitoring system is the `udp-push` system. This model is highly scalable and its limit (in terms of number of VMs monitored per second) is bounded to the performance of the server running `oned` and the database server. Our scalability testing achieves the monitoring of tens of thousands of VMs in a few minutes.

Read more in the Monitoring guide.

### 3.2.2 Core Tuning

OpenNebula keeps the monitorization history for a defined time in a database table. These values are then used to draw the plots in Sunstone.

These monitorization entries can take quite a bit of storage in your database. The amount of storage used will depend on the size of your cloud, and the following configuration attributes in `oned.conf`:

- `MONITORING_INTERVAL` (VMware only): Time in seconds between each monitorization. Default: 60.
- `collectd IM_MAD -i` argument (KVM & Xen only): Time in seconds of the monitorization push cycle. Default: 20.

- `HOST_MONITORING_EXPIRATION_TIME`: Time, in seconds, to expire monitoring information. Default: 12h.
- `VM_MONITORING_EXPIRATION_TIME`: Time, in seconds, to expire monitoring information. Default: 4h.

If you don't use Sunstone, you may want to disable the monitoring history, setting both expiration times to 0.

Each monitoring entry will be around 2 KB for each Host, and 4 KB for each VM. To give you an idea of how much database storage you will need to prepare, these some examples:

Monitoring interval	Host expiration	# Hosts	Storage
20s	12h	200	850 MB
20s	24h	1000	8.2 GB

Monitoring interval	VM expiration	# VMs	Storage
20s	4h	2000	1.8 GB
20s	24h	10000	7 GB

### 3.2.3 API Tuning

For large deployments with lots of `xmlrpc` calls the default values for the `xmlrpc` server are too conservative. The values you can modify and its meaning are explained in the `oned.conf` guide and the [xmlrpc-c library documentation](#). From our experience these values improve the server behaviour with a high amount of client calls:

```
MAX_CONN = 240
MAX_CONN_BACKLOG = 480
```

OpenNebula Cloud API (OCA) is able to use the library `Ox` for XML parsing. This library is makes the parsing of pools much faster. It is used by both the CLI and Sunstone so both will benefit from it.

The core is able to paginate some pool answers. This makes the memory consumption decrease and in some cases the parsing faster. By default the pagination value is 2000 objects but can be changed using the environment variable `ONE_POOL_PAGE_SIZE`. It should be bigger than 2. For example, to list VMs with a page size of 5000 we can use:

```
$ ONE_POOL_PAGE_SIZE=5000 onevm list
```

To disable pagination we can use a non numeric value:

```
$ ONE_POOL_PAGE_SIZE=disabled onevm list
```

This environment variable can be also used for Sunstone.

### 3.2.4 Driver Tuning

OpenNebula drivers have by default 15 threads. This is the maximum number of actions a driver can perform at the same time, the next actions will be queued. You can make this value in `oned.conf`, the driver parameter is `-t`.

### 3.2.5 Database Tuning

For non test installations use MySQL database. `sqlite` is too slow for more than a couple hosts and a few VMs.

### 3.2.6 Sunstone Tuning

Please refer to guide about *Configuring Sunstone for Large Deployments*.

## HIGH AVAILABILITY

### 4.1 Virtual Machines High Availability

OpenNebula delivers the availability required by most applications running in virtual machines. This guide's objective is to provide information in order to prepare for failures in the virtual machines or physical nodes, and recover from them. These failures are categorized depending on whether they come from the physical infrastructure (Host failures) or from the virtualized infrastructure (VM crashes). In both scenarios, OpenNebula provides a cost-effective failover solution to minimize downtime from server and OS failures.

If you are interested in setting up a high available cluster for OpenNebula, check the *High OpenNebula Availability Guide*.

#### 4.1.1 Host Failures

When OpenNebula detects that a host is down, a hook can be triggered to deal with the situation. OpenNebula comes with a script out-of-the-box that can act as a hook to be triggered when a host enters the ERROR state. This can be very useful to limit the downtime of a service due to a hardware failure, since it can redeploy the VMs on another host.

Let's see how to configure `/etc/one/oned.conf` to set up this Host hook, to be triggered in the ERROR state. The following should be uncommented in the mentioned configuration file:

```
#-----  
HOST_HOOK = [  
    name      = "error",  
    on        = "ERROR",  
    command   = "ft/host_error.rb",  
    arguments = "$ID -m -p 5",  
    remote    = "no" ]  
#-----
```

We are defining a host hook, named `error`, that will execute the script `'ft/host_error.rb'` locally with the following arguments:

Argument	Description
<b>Host ID</b>	ID of the host containing the VMs to treat. It is compulsory and better left to <b>\$ID</b> , that will be automatically filled by OpenNebula with the Host ID of the host that went down.
<b>Action</b>	This defines the action to be performed upon the VMs that were running in the host that went down. This can be: <ul style="list-style-type: none"> <li>• <b>-m</b> migrate VMs to another host. Only for images in shared storage</li> <li>• <b>-r</b> delete+recreate VMs running in the host. State will be lost.</li> <li>• <b>-d</b> delete VMs running in the host</li> </ul>
<b>ForceSuspended</b>	[-f] force resubmission of suspended VMs
<b>AvoidTransient</b>	[-p <n>] avoid resubmission if host comes back after <n> monitoring cycles

More information on hooks here.

**Warning:** Note that spurious network errors may lead to a VM started twice in different hosts and possibly contend on shared resources. The previous script needs to fence the error host to prevent split brain VMs. You may use any fencing mechanism for the host and invoke it within the error hook.

## 4.1.2 Virtual Machine Failures

The overall state of a virtual machine in a failure condition will show as `failure` (or `fail` in the CLI). To find out the specific failure situation you need to check the `LCM_STATE` of the VM in the VM info tab (or `onevm show` in the CLI). Moreover, a VM can be stuck in a transition (e.g. `boot` or `save`) because of a host or network failure. Typically these operations will eventually timeout and lead to a VM failure state.

The administrator has the ability to force a recovery action from Sunstone or from the CLI, with the `onevm recover` command. This command has the following options:

- `--success`: If the operation has been confirmed to succeed. For example, the administrator can see the VM properly running in the hypervisor, but the driver failed to inform OpenNebula of the successful boot.
- `--failure`: This will have the same effect as a driver reporting a failure. It is intended for VMs that get stuck in transient states. As an example, if a storage problem occurs and the administrator knows that a VM stuck in `prolog` is not going to finish the pending transfer, this action will manually move the VM to `prolog_failure`.
- `--retry`: To retry the previously failed action. Can be used, for instance, in case a VM is in `boot_failure` because the hypervisor crashed. The administrator can tell OpenNebula to retry the boot after the hypervisor is started again.
- `--retry --interactive`: In some scenarios where the failure was caused by an error in the Transfer Manager actions, each action can be rerun and debugged until it works. Once the commands are successful, a `success` should be sent. See the specific section below for more details.

Note also that OpenNebula will try to automatically recover some failure situations using the monitor information. A specific example is that a VM in the `boot_failure` state will become `running` if the monitoring reports that the VM was found running in the hypervisor.



## Hypervisor Problems

The following list details failures states caused by errors related to the hypervisor.

- `BOOT_FAILURE`, The VM failed to boot but all the files needed by the VM are already in the host. Check the hypervisor logs to find out the problem, and once fixed recover the VM with the retry option.
- `BOOT_MIGRATE_FAILURE`, same as above but during a migration. Check the target hypervisor and retry the operation.
- `BOOT_UNDEPLOY_FAILURE`, same as above but during a resume after an undeploy. Check the target hypervisor and retry the operation.
- `BOOT_STOPPED_FAILURE`, same as above but during a resume after a stop. Check the target hypervisor and retry the operation.

## Transfer Manager / Storage Problems

The following list details failure states caused by errors in the Transfer Manager driver. These states can be recovered by checking the `vm.log` and looking for the specific error (disk space, permissions, mis-configured datastore, etc). You can execute `--retry` to relaunch the Transfer Manager actions after fixing the problem (freeing disk space, etc). You can execute `--retry --interactive` to launch a Transfer Manager Interactive Debug environment that will allow you to: (1) see all the TM actions in detail (2) relaunch each action until its successful (3) skip TM actions.

- `PROLOG_FAILURE`, there was a problem setting up the disk images needed by the VM.
- `PROLOG_MIGRATE_FAILURE`, problem setting up the disks in the target host.
- `EPILOG_FAILURE`, there was a problem processing the disk images (may be discard or save) after the VM execution.
- `EPILOG_STOP_FAILURE`, there was a problem moving the disk images after a stop.
- `EPILOG_UNDEPLOY_FAILURE`, there was a problem moving the disk images after an undeploy.
- `PROLOG_MIGRATE_POWEROFF_FAILURE`, problem restoring the disk images after a migration in a poweroff state.
- `PROLOG_MIGRATE_SUSPEND_FAILURE`, problem restoring the disk images after a migration in a suspend state.
- `PROLOG_RESUME_FAILURE`, problem restoring the disk images after a stop.
- `PROLOG_UNDEPLOY_FAILURE`, problem restoring the disk images after an undeploy.

Example of a Transfer Manager Interactive Debug environment (`onevm recover <id> --retry --interactive`):

```
$ onevm show 2|grep LCM_STATE
LCM_STATE           : PROLOG_UNDEPLOY_FAILURE

$ onevm recover 2 --retry --interactive
TM Debug Interactive Environment.

TM Action list:
(1) MV shared haddock:/var/lib/one//datastores/0/2/disk.0 localhost:/var/lib/one//datastores/0/2/disk.0
(2) MV shared haddock:/var/lib/one//datastores/0/2 localhost:/var/lib/one//datastores/0/2 2 0

Current action (1):
MV shared haddock:/var/lib/one//datastores/0/2/disk.0 localhost:/var/lib/one//datastores/0/2/disk.0 2
```

```
Choose action:
(r) Run action
(n) Skip to next action
(a) Show all actions
(q) Quit
> r

LOG I Command execution fail: /var/lib/one/remotes/tm/shared/mv haddock:/var/lib/one//datastores/0/2
LOG I ExitCode: 1

FAILURE. Repeat command.

Current action (1):
MV shared haddock:/var/lib/one//datastores/0/2/disk.0 localhost:/var/lib/one//datastores/0/2/disk.0 2

Choose action:
(r) Run action
(n) Skip to next action
(a) Show all actions
(q) Quit
> # FIX THE PROBLEM...

> r

SUCCESS

Current action (2):
MV shared haddock:/var/lib/one//datastores/0/2 localhost:/var/lib/one//datastores/0/2 2 0

Choose action:
(r) Run action
(n) Skip to next action
(a) Show all actions
(q) Quit
> r

SUCCESS

If all the TM actions have been successful and you want to
recover the Virtual Machine to the RUNNING state execute this command:
$ onevm recover 2 --success

$ onevm recover 2 --success

$ onevm show 2|grep LCM_STATE
LCM_STATE          : RUNNING
```

## 4.2 OpenNebula High Availability

This guide walks you through the process of setting a high available cluster for OpenNebula. The ultimate goal is to reduce downtime of core OpenNebula services: core (oned), scheduler (mm\_sched) and Sunstone interface (sunstone-server).

We will be using the classical active-passive cluster architecture which is the recommended solution for OpenNebula. In this solution two (or more) nodes will be part of a cluster where the OpenNebula daemon, scheduler and Sunstone (web UI) are cluster resources. When the active node fails, the passive one takes control.

If you are interested in failover protection against hardware and operating system outages within your virtualized IT environment, check the *Virtual Machines High Availability Guide*.

This guide is structured in a *how-to* form using Pacemaker tested in a CentOS 7 installation; but generic considerations and requirements for this setup are discussed to easily implement this solution with other systems.

### 4.2.1 Overview

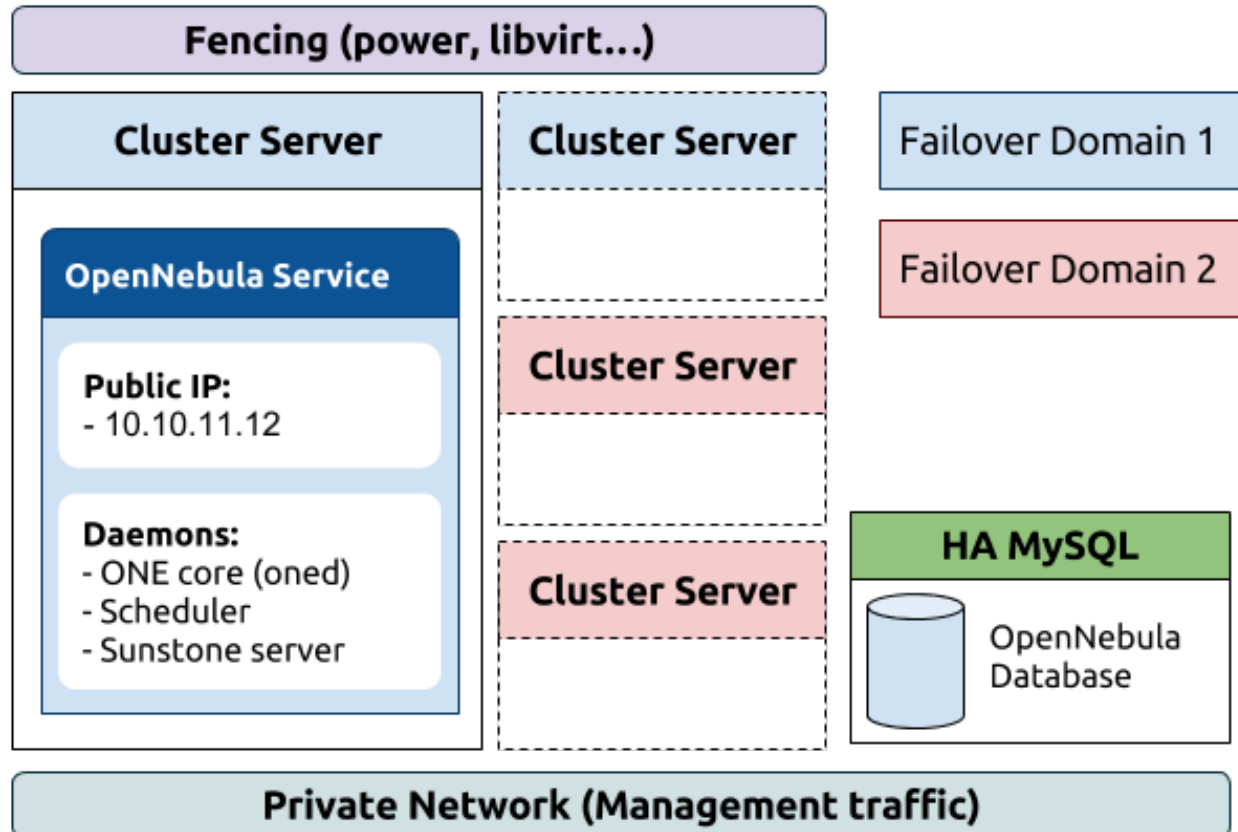
In terms of high-availability, OpenNebula consists in three different basic services, namely:

- **OpenNebula Core:** It is the main orchestration component, supervises the life-cycle of each resources (e.g. hosts, VMs or networks) and operates on the physical infrastructure to deploy and manage virtualized resources.
- **Scheduler:** The scheduler performs a matching between the virtual requests and the available resources using different scheduling policies. It basically assigns a physical host, and a storage area to each VM.
- **Sunstone:** The GUI for advanced and cloud users as well as system administrators. The GUI is accessed through a well-known end-point (IP/URL). Sunstone has been architected as a scalable web application supporting multiple application servers or processes.

The state of the system is stored in a database for persistency and managed by OpenNebula core. In order to improve the response time of the core daemon, it caches the most recently used data so it reduces the number of queries to the DB. Note that this prevents an active-active HA configuration for OpenNebula. However an active-passive configuration, given the lightweight and negligible start times of the core services, is a very adequate solution for the HA problem.

In this guide we assume that the DB backing OpenNebula core state is also configured in an HA mode. The database service should be configured as part of the cluster (with clustered storage), or a master-master setup running in the frontend Virtual Machines (very easy to setup), or in an external dedicated HA database cluster. The procedure to setup an HA environment for MySQL is beyond the scope of this document. Sqlite should not be used and is not recommended in any case for production setups.

This guide assumes we are not using Apache to serve the Sunstone GUI. Note that the recommendation for production environment is to use Apache. Instead of registering `opennebula-sunstone` as a cluster resource, we would need to register `httpd` (and configuring it first).



## 4.2.2 HA Cluster Components & Services

As shown in the previous figure, we will use just one fail-over domain (blue) with two hosts. All OpenNebula services will be co-located and run on the same server in this case. You can however easily modify this configuration to split them and assign each service to a different host and define different fail-over domains for each one (e.g. blue for oned and scheduler, red for sunstone).

The following components will be installed:

- opennebula services
- corosync+pacemaker
- fencing agents

## 4.2.3 Installation and Configuration

In the following, we assume that the cluster consists on two servers:

- one-server1
- one-server2

**Warning:** While setting and testing the installation it is recommended to disable any firewall. Also watch out for SELinux.

## Step 1: OpenNebula

You should have two servers (they may be VMs, as discussed below) ready to install OpenNebula. These servers will have the same requirements as regular OpenNebula front-end (e.g. network connection to hosts, ssh passwordless access, shared filesystems if required...). Remember to use a HA MySQL backend.

It is important to use a twin installation (i.e. same configuration files) so probably it is better to start and configure a server, and once it is tested rsync the configuration to the other one.

## Step 2: Install Cluster Software

In **both** cluster servers install the cluster components:

```
$ yum install pcs fence-agents-all
$ passwd hacluster
```

**Warning:** Set the same password for user hacluster in all the servers

Pacemaker bundles the Sinatra and Rack gems, and if the wrong version of the gems is installed in the host, it will fail to start. Therefore ensure you manually install the specific versions:

```
$ gem uninstall rack sinatra
$ gem install --no-ri --no-rdoc rack --version=1.5.2
$ gem install --no-ri --no-rdoc rack-protection --version=1.5.3
$ gem install --no-ri --no-rdoc rack-test --version=0.6.2
$ gem install --no-ri --no-rdoc sinatra --version=1.4.5
$ gem install --no-ri --no-rdoc sinatra-contrib --version=1.4.2
$ gem install --no-ri --no-rdoc sinatra-sugar --version=0.5.1
```

Maybe the versions listed above do not coincide with the `sinatra*` and `rack*` gems vendored in the `/usr/lib/pcsd/vendor/bundle/ruby/gems/` path (owned by the `pcsd` package). If they don't update the previous commands with the appropriate versions.

Start/enable the cluster services:

```
$ systemctl start pcsd.service
$ systemctl enable pcsd.service
$ systemctl enable corosync.service
$ systemctl enable pacemaker.service
```

At this point make sure the firewall allows the necessary ports for the cluster services. Remember to disable it if you hit any errors to determine if the error comes from the firewall. If you are using `firewalld` use the following snippet to allow the cluster services traffic:

```
$ firewall-cmd --permanent --add-service=high-availability
$ firewall-cmd --reload
```

## Step 3: Create the Cluster and Failover Domain

The following commands must be executed **only in one node**, for example `one-server1`.

Authorize the nodes:

```
$ pcs cluster auth one-server1 one-server2
Username: hacluster
```

```
one-server1: Authorized
one-server2: Authorized
```

Now we need to create the cluster:

```
$ pcs cluster setup --name opennebula one-server1 one-server2
```

Now we can start the cluster:

```
$ pcs cluster start --all
```

As we only have two nodes, we can't reach a majority quorum, we must disable it:

```
pcs property set no-quorum-policy=ignore
```

### Step 4: Define the OpenNebula Service

We need to enable a fencing agent. To query the available ones you can execute:

```
$ pcs stonith list
$ pcs stonith describe <fencing_agent>
```

In this case we will exemplify the `fence_ilo_ssh` command:

```
$ pcs stonith create fence_server1 fence_ilo_ssh pcmk_host_list=one-server1 ipaddr=<ilo_hypervisor_one-server1>
$ pcs stonith create fence_server2 fence_ilo_ssh pcmk_host_list=one-server2 ipaddr=<ilo_hypervisor_one-server2>
```

Not that the delay is different to get protection from stonith battles. With the above configuration, in a split brain event `one-server2` would be killed before it can kill `one-server1`, ensure we keep at least one node.

You can try out fencing manually by running these commands:

```
$ fence_ilo_ssh -o status -x -a "<ilo_hypervisor_one-server1>" -l "..." -p "..." -v
$ fence_ilo_ssh -o reboot -x -a "<ilo_hypervisor_one-server1>" -l "..." -p "..." -v
```

Or even by calling the cluster:

```
$ pcs stonith fence one-server2
```

Next, we can add the HA IP where users will be able to connect to:

```
$ pcs resource create Cluster_VIP ocf:heartbeat:IPAddr2 ip=<HA_ip> cidr_netmask=24 op monitor interval=...
```

The nic is inferred from the routing table, but it can be passed explicitly. Note that you can check all the possible arguments by running:

```
$ pcs resource describe ocf:heartbeat:IPAddr2
```

We are now ready to add the OpenNebula resources:

```
$ pcs resource create opennebula systemd:opennebula
$ pcs resource create opennebula-sunstone systemd:opennebula-sunstone
$ pcs resource create opennebula-gate systemd:opennebula-gate
$ pcs resource create opennebula-flow systemd:opennebula-flow
```

**Warning:** Make sure you run `systemctl disable <service>` for all the `systemd` services you add to the cluster.

You will notice that at this point the services are not started in the same host. If you want them to be in the same host, you can configure the colocation:

```
$ pcs constraint colocation add opennebula Cluster_VIP INFINITY
$ pcs constraint colocation add opennebula-sunstone Cluster_VIP INFINITY
$ pcs constraint colocation add opennebula-novnc Cluster_VIP INFINITY
$ pcs constraint colocation add opennebula-gate Cluster_VIP INFINITY
$ pcs constraint colocation add opennebula-flow Cluster_VIP INFINITY
```

At this point, the cluster should be properly configured:

```
$ pcs status
Cluster name: opennebula
Last updated: [...]
Stack: corosync
Current DC: one-server1 (version [...]) - partition with quorum
2 nodes and 8 resources configured

Online: [ one-server1 one-server2 ]

Full list of resources:

fence_server1 (stonith:fence_ilo_ssh): Started one-server1
fence_server2 (stonith:fence_ilo_ssh): Started one-server2
Cluster_VIP (ocf::heartbeat:IPaddr2): Started one-server1
opennebula (systemd:opennebula): Started one-server1
opennebula-sunstone (systemd:opennebula-sunstone): Started one-server1
opennebula-novnc (systemd:opennebula-novnc): Started one-server1
opennebula-gate (systemd:opennebula-gate): Started one-server1
opennebula-flow (systemd:opennebula-flow): Started one-server1

Failed Actions:

PCSD Status:
  one-server1: Online
  one-server2: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

#### 4.2.4 What to Do After a Fail-over Event

When the active node fails and the passive one takes control, it will start OpenNebula again. This OpenNebula will see the resources in the exact same way as the one in the server that crashed. However, there might be a set of Virtual Machines which could be stuck in transient states. For example when a Virtual Machine is deployed and it starts copying the disks to the target hosts it enters one of this transient states (in this case 'PROLOG'). OpenNebula will wait for the storage driver to return the 'PROLOG' exit status. This will never happen since the driver fails during the crash, therefore the Virtual Machine will get stuck in the state.

In these cases it's important to review the states of all the Virtual Machines and let OpenNebula know if the driver exited successfully or not. There is a command specific for this: `onevm recover`. You can read more about this command in the Managing Virtual Machines guide.

In our example we would need to manually check if the disk files have been properly deployed to our host and execute:

```
$ onevm recover <id> --success # or --failure
```

The transient states to watch out for are:

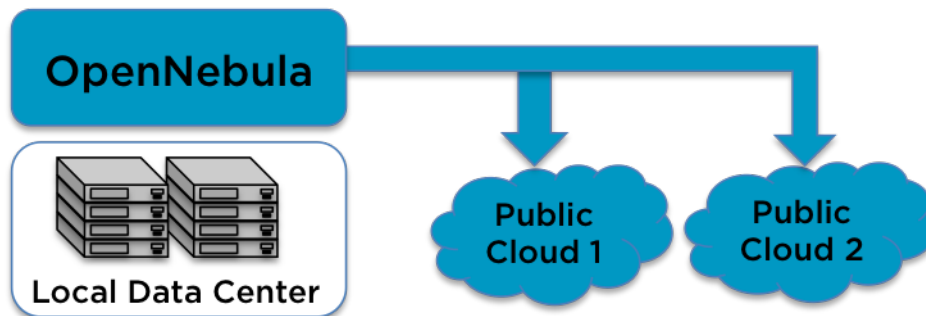
- BOOT
- CLEAN
- EPILOG
- FAIL
- HOTPLUG
- MIGRARTE
- PROLOG
- SAVE
- SHUTDOWN
- SNAPSHOT
- UNKNOWN



## CLOUD BURSTING

### 5.1 Cloud Bursting

Cloud bursting is a model in which the local resources of a Private Cloud are combined with resources from remote Cloud providers. The remote provider could be a commercial Cloud service, such as Amazon EC2, IBM SoftLayer or Microsoft Azure. Such support for cloud bursting enables highly scalable hosting environments.



As you may know, OpenNebula's approach to cloud bursting is quite unique. The reason behind this uniqueness is the transparency to both end users and cloud administrators to use and maintain the cloud bursting functionality. The **transparency to cloud administrators** comes from the fact that a AWS EC2 region, SoftLayer datacenter or an Azure location is modelled as any other host (albeit of potentially a much bigger capacity), so the scheduler can place VMs in the external cloud as it will do in any other local host. For instance, in EC2:

```
$ onehost list
```

ID	NAME	CLUSTER	RVM	ALLOCATED_CPU	ALLOCATED_MEM	STAT
2	kvm-	-	0	0 / 800 (0%)	0K / 16G (0%)	on
3	kvm-1	-	0	0 / 100 (0%)	0K / 1.8G (0%)	on
4	us-east-1	ec2	0	0 / 500 (0%)	0K / 8.5G (0%)	on

On the other hand, the **transparency to end users** is offered through the hybrid template functionality: the same VM template in OpenNebula can describe the VM if it is deployed locally and also if it gets deployed in EC2, SoftLayer or Azure. So users just have to instantiate the template, and OpenNebula will transparently choose if that is executed locally or remotely. A simple template like the following is enough to launch Virtual Machines in EC2:

```
NAME=ec2template
CPU=1
MEMORY=1700

EC2=[
  AMI="ami-6f5f1206",
  BLOCKDEVICEMAPPING="/dev/sdh=:20",
  INSTANCETYPE="m1.small",
```

```
KEYPAIR="gsg-keypair" ]  
SCHED_REQUIREMENTS="PUBLIC_CLOUD=YES"
```

```
$ onetemplate create ec2template.one  
ID: 112  
$ onetemplate instantiate 112  
VM ID: 234
```

For more information on how to configure an Amazon EC2 host see the following guide:

- [Amazon EC2 driver](#)

For more information on how to configure a SoftLayer host see the following guide:

- [SoftLayer driver](#)

For more information on how to configure an Azure host see the following guide:

- [Azure driver](#)

## 5.2 Amazon EC2 Driver

### 5.2.1 Considerations & Limitations

You should take into account the following technical considerations when using the EC2 cloud with OpenNebula:

- There is no direct access to the dom0, so it cannot be monitored (we don't know where the VM is running on the EC2 cloud).
- The usual OpenNebula functionality for snapshotting, hot-plugging, or migration is not available with EC2.
- By default OpenNebula will always launch m1.small instances, unless otherwise specified.
- Monitoring of VMs in EC2 is done through CloudWatch. Only information related to the consumption of CPU and Networking (both inbound and outbound) is collected, since CloudWatch does not offer information of guest memory consumption.

Please refer to the EC2 documentation to obtain more information about Amazon instance types and image management:

- [General information of instances](#)

### 5.2.2 Prerequisites

- You must have a working account for [AWS](#) and signup for EC2 and S3 services.
- The [aws-sdk ruby gem](#) needs to be installed. This gem is automatically installed as part of the installation process. In case you need to deploy it manually, please take into account that v2 of the gem only works for ruby versions > 1.9.

### 5.2.3 OpenNebula Configuration

Uncomment the EC2 IM and VMM drivers from `/etc/one/oned.conf` file in order to use the driver.

```

IM_MAD = [
    name      = "ec2",
    executable = "one_im_sh",
    arguments  = "-c -t 1 -r 0 ec2" ]

VM_MAD = [
    name      = "ec2",
    executable = "one_vmm_sh",
    arguments  = "-t 15 -r 0 ec2",
    type      = "xml" ]

```

Driver flags are the same as other drivers:

FLAG	SETs
-t	Number of threads
-r	Number of retries

Additionally you must define the AWS credentials and AWS region to be used and the maximum capacity that you want OpenNebula to deploy on the EC2, for this edit the file `/etc/one/ec2_driver.conf`:

```

regions:
  default:
    region_name: us-east-1
    access_key_id: YOUR_ACCESS_KEY
    secret_access_key: YOUR_SECRET_ACCESS_KEY
    capacity:
      m1.small: 5
      m1.large: 0
      m1.xlarge: 0

```

You can define an http proxy if the OpenNebula Frontend does not have access to the internet, in `/etc/one/ec2_driver.conf`:

```
proxy_uri: http://...
```

After OpenNebula is restarted, create a new Host that uses the ec2 drivers:

```
$ onehost create ec2 --im ec2 --vm ec2 --net dummy
```

## 5.2.4 EC2 Specific Template Attributes

In order to deploy an instance in EC2 through OpenNebula you must include an EC2 section in the virtual machine template. This is an example of a virtual machine template that can be deployed in our local resources or in EC2.

```

CPU      = 0.5
MEMORY   = 128

# Xen or KVM template machine, this will be use when submitting this VM to local resources
DISK     = [ IMAGE_ID = 3 ]
NIC      = [ NETWORK_ID = 7 ]

# EC2 template machine, this will be use wen submitting this VM to EC2
EC2 = [ AMI="ami-00bafcb5",
        KEYPAIR="gsg-keypair",
        INSTANCETYPE=m1.small]

#Add this if you want to use only EC2 cloud
#SCHED_REQUIREMENTS = 'HOSTNAME = "ec2"'

```

These are the attributes that can be used in the EC2 section of the template:

ATTRIBUTES	DESCRIPTION
AMI	Unique ID of a machine image, returned by a call to <code>ec2-describe-images</code> .
AKI	The ID of the kernel with which to launch the instance.
CLIENTTOKEN	Unique, case-sensitive identifier you provide to ensure idempotency of the request.
INSTANCETYPE	Specifies the instance type.
KEYPAIR	The name of the key pair, later will be used to execute commands like <code>ssh -i id_keypair</code> or <code>scp -i id_keypair</code>
LICENSEPOOL	<code>-license-pool</code>
BLOCKDEVICEMAPPING	The block device mapping for the instance. More than one can be specified in a space-separated list. Check the <code>-block-device-mapping</code> option of the <a href="#">EC2 CLI Reference</a> for the syntax
PLACEMENTGROUP	Name of the placement group.
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.
RAMDISK	The ID of the RAM disk to select.
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 <code>ec2-associate-address -i i-0041230 -a elasticip</code> .
TENANCY	The tenancy of the instance you want to launch.
USERDATA	Specifies Base64-encoded MIME user data to be made available to the instance(s) in this reservation.
SECURITYGROUPS	Name of the security group. You can specify more than one security group (comma separated).
SECURITYGROUPIDS	Ids of the security group. You can specify more than one security group (comma separated).
ELASTICIP	EC2 Elastic IP address to assign to the instance. This parameter is passed to the command <code>ec2-associate-address -i i-0041230 elasticip</code> .
TAGS	Key and optional value of the tag, separated by an equals sign (=). You can specify more than one tag (comma separated).
AVAILABILITYZONE	The Availability Zone in which to run the instance.
HOST	Defines which OpenNebula host will use this template
EBS_OPTIMIZED	Obtain a better I/O throughput for VMs with EBS provisioned volumes

Default values for all these attributes can be defined in the `/etc/one/ec2_driver.default` file.

```
<!--
Default configuration attributes for the EC2 driver
(all domains will use these values as defaults)
Valid attributes are: AKI AMI CLIENTTOKEN INSTANCETYPE KEYPAIR LICENSEPOOL
    PLACEMENTGROUP PRIVATEIP RAMDISK SUBNETID TENANCY USERDATA SECURITYGROUPS
    AVAILABILITYZONE EBS_OPTIMIZED ELASTICIP TAGS
Use XML syntax to specify defaults, note elements are UPPERCASE
Example:
<TEMPLATE>
  <EC2>
    <KEYPAIR>gsg-keypair</KEYPAIR>
    <INSTANCETYPE>m1.small</INSTANCETYPE>
  </EC2>
</TEMPLATE>
-->

<TEMPLATE>
  <EC2>
    <INSTANCETYPE>m1.small</INSTANCETYPE>
  </EC2>
</TEMPLATE>
```

**Note:** The EC2 and PUBLIC\_CLOUD sections allow for substitutions from template and virtual network variables, the same way as the CONTEXT section allows.

These values can furthermore be asked to the user using user inputs. A common scenario is to delegate the User Data to the end user. For that, a new User Input named USERDATA can be created of text64 (the User Data needs to be encoded on base64) and a placeholder added to the EC2 section:

```
EC2 = [ AMI="ami-00bafcb5",
        KEYPAIR="gsg-keypair",
        INSTANCETYPE=m1.small,
        USERDATA="$USERDATA" ]
```

## Context Support

If a CONTEXT section is defined in the template, it will be available as USERDATA inside the VM and can be retrieved by running the following command:

```
$ curl http://169.254.169.254/latest/user-data
ONEGATE_ENDPOINT="https://onegate..."
SSH_PUBLIC_KEY="ssh-rsa ABAABeqzaCly..."
```

If the linux context packages for EC2 are installed in the VM, these parameters will be used to configure the VM. These is the list of the supported parameters for EC2.

For example, if you want to enable SSH access to the VM, an existing EC2 keypair name can be provided in the EC2 template section or the SSH public key of the user can be included in the CONTEXT section of the template.

**Note:** If a value for the USERDATA attribute is provided in the EC2 section of the template, the CONTEXT section will be ignored and the value provided as USERDATA will be available instead of the CONTEXT information.

## 5.2.5 Multi EC2 Site/Region/Account Support

It is possible to define various EC2 hosts to allow opennebula the managing of different EC2 regions or different EC2 accounts.

When you create a new host the credentials and endpoint for that host are retrieved from the `/etc/one/ec2_driver.conf` file using the host name. Therefore, if you want to add a new host to manage a different region, i.e. `eu-west-1`, just add your credentials and the capacity limits to the `eu-west-1` section in the conf file, and specify that name (`eu-west-1`) when creating the new host.

```
regions:
  ...
  eu-west-1:
    region_name: us-east-1
    access_key_id: YOUR_ACCESS_KEY
    secret_access_key: YOUR_SECRET_ACCESS_KEY
    capacity:
      m1.small: 5
      m1.large: 0
      m1.xlarge: 0
```

After that, create a new Host with the `eu-west-1` name:

```
$ onehost create eu-west-1 --im ec2 --vm ec2 --net dummy
```

If the Host name does not match any regions key, the `default` will be used.

You can define a different EC2 section in your template for each EC2 host, so with one template you can define different AMIs depending on which host it is scheduled, just include a `HOST` attribute in each EC2 section:

```
EC2 = [ HOST="ec2",
        AMI="ami-0022c769" ]
EC2 = [ HOST="eu-west-1",
        AMI="ami-03324cc9" ]
```

You will have `ami-0022c769` launched when this VM template is sent to host `ec2` and `ami-03324cc9` whenever the VM template is sent to host `eu-west-1`.

**Warning:** If only one EC2 site is defined, the EC2 driver will deploy all EC2 templates onto it, not paying attention to the **HOST** attribute.

The availability zone inside a region, can be specified using the `AVAILABILITYZONE` attribute in the EC2 section of the template

### 5.2.6 Hybrid VM Templates

A powerful use of cloud bursting in OpenNebula is the ability to use hybrid templates, defining a VM if OpenNebula decides to launch it locally, and also defining it if it is going to be outsourced to Amazon EC2. The idea behind this is to reference the same kind of VM even if it is incarnated by different images (the local image and the remote AMI).

An example of a hybrid template:

```
## Local Template section
NAME=MNyWebServer

CPU=1
MEMORY=256

DISK=[ IMAGE="nginx-golden" ]
NIC=[ NETWORK="public" ]

EC2=[
  AMI="ami-xxxxx" ]
```

OpenNebula will use the first portion (from `NAME` to `NIC`) in the above template when the VM is scheduled to a local virtualization node, and the `EC2` section when the VM is scheduled to an EC2 node (ie, when the VM is going to be launched in Amazon EC2).

### 5.2.7 Testing

You must create a template file containing the information of the AMIs you want to launch. Additionally if you have an elastic IP address you want to use with your EC2 instances, you can specify it as an optional parameter.

```
CPU      = 1
MEMORY   = 1700

#Xen or KVM template machine, this will be use when submitting this VM to local resources
DISK     = [ IMAGE_ID = 3 ]
NIC      = [ NETWORK_ID = 7 ]

#EC2 template machine, this will be use wen submitting this VM to EC2
```

```
EC2 = [ AMI="ami-00bafcb5",
        KEYPAIR="gsg-keypair",
        INSTANCETYPE=m1.small]

#Add this if you want to use only EC2 cloud
#SCHED_REQUIREMENTS = 'HOSTNAME = "ec2"'
```

You only can submit and control the template using the OpenNebula interface:

```
$ onetemplate create ec2template
$ onetemplate instantiate ec2template
```

Now you can monitor the state of the VM with

```
$ onevm list
```

ID	USER	GROUP	NAME	STAT	CPU	MEM	HOSTNAME	TIME
0	oneadmin	oneadmin	one-0	runn	0	0K	ec2	0d 07:03

Also you can see information (like IP address) related to the amazon instance launched via the command. The attributes available are:

- AWS\_DNS\_NAME
- AWS\_PRIVATE\_DNS\_NAME
- AWS\_KEY\_NAME
- AWS\_AVAILABILITY\_ZONE
- AWS\_PLATFORM
- AWS\_VPC\_ID
- AWS\_PRIVATE\_IP\_ADDRESS
- AWS\_IP\_ADDRESS
- AWS\_SUBNET\_ID
- AWS\_SECURITY\_GROUPS
- AWS\_INSTANCE\_TYPE

```
$ onevm show 0
VIRTUAL MACHINE 0 INFORMATION
ID : 0
NAME : pepe
USER : oneadmin
GROUP : oneadmin
STATE : ACTIVE
LCM_STATE : RUNNING
RESCHED : No
HOST : ec2
CLUSTER ID : -1
START TIME : 11/15 14:15:16
END TIME : -
DEPLOY ID : i-a0c5a2dd

VIRTUAL MACHINE MONITORING
USED MEMORY : 0K
NET_RX : 208K
NET_TX : 4K
USED CPU : 0.2
```

```

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

VIRTUAL MACHINE HISTORY
SEQ HOST        ACTION          DS          START          TIME          PROLOG
 0 ec2          none             0 11/15 14:15:37 2d 21h48m    0h00m00s

USER TEMPLATE
EC2=[
  AMI="ami-6f5f1206",
  INSTANCETYPE="m1.small",
  KEYPAIR="gsg-keypair" ]
SCHED_REQUIREMENTS="ID=4"

VIRTUAL MACHINE TEMPLATE
AWS_AVAILABILITY_ZONE="us-east-1d"
AWS_DNS_NAME="ec2-54-205-155-229.compute-1.amazonaws.com"
AWS_INSTANCE_TYPE="m1.small"
AWS_IP_ADDRESS="54.205.155.229"
AWS_KEY_NAME="gsg-keypair"
AWS_PRIVATE_DNS_NAME="ip-10-12-101-169.ec2.internal"
AWS_PRIVATE_IP_ADDRESS="10.12.101.169"
AWS_SECURITY_GROUPS="sg-8e45a3e7"
    
```

## 5.2.8 Scheduler Configuration

Since ec2 Hosts are treated by the scheduler like any other host, VMs will be automatically deployed in them. But you probably want to lower their priority and start using them only when the local infrastructure is full.

### Configure the Priority

The ec2 drivers return a probe with the value `PRIORITY = -1`. This can be used by the scheduler, configuring the ‘fixed’ policy in `sched.conf`:

```

DEFAULT_SCHED = [
  policy = 4
]
    
```

The local hosts will have a priority of 0 by default, but you could set any value manually with the ‘onehost/onecluster update’ command.

There are two other parameters that you may want to adjust in `sched.conf`:

```

- ``MAX_DISPATCH``: Maximum number of Virtual Machines actually dispatched to a host in each scheduling cycle
- ``MAX_HOST``: Maximum number of Virtual Machines dispatched to a given host in each scheduling cycle
    
```

In a scheduling cycle, when `MAX_HOST` number of VMs have been deployed to a host, it is discarded for the next pending VMs.

For example, having this configuration:

- `MAX_HOST = 1`
- `MAX_DISPATCH = 30`



- 2 Hosts: 1 in the local infrastructure, and 1 using the ec2 drivers
- 2 pending VMs

The first VM will be deployed in the local host. The second VM will have also sort the local host with higher priority, but because 1 VMs was already deployed, the second VM will be launched in ec2.

A quick way to ensure that your local infrastructure will be always used before the ec2 hosts is to **set MAX\_DISPATCH to the number of local hosts**.

### Force a Local or Remote Deployment

The ec2 drivers report the host attribute PUBLIC\_CLOUD = YES. Knowing this, you can use that attribute in your VM requirements.

To force a VM deployment in a local host, use:

```
SCHED_REQUIREMENTS = "!(PUBLIC_CLOUD = YES)"
```

To force a VM deployment in an ec2 host, use:

```
SCHED_REQUIREMENTS = "PUBLIC_CLOUD = YES"
```

## 5.2.9 Importing VMs

VMs running on EC2 that were not launched through OpenNebula can be imported in OpenNebula.

## 5.3 SoftLayer Driver

### 5.3.1 Considerations & Limitations

You should take into account the following technical considerations when using the SoftLayer (SL) cloud with OpenNebula:

- There is no direct access to the hypervisor, so it cannot be monitored (we don't know where the VM is running on the SoftLayer cloud).
- This driver does not support proxy to connect to SoftLayer server.
- The usual OpenNebula functionality for snapshotting, hot-plugging, or migration is not available with SoftLayer (currently).
- By default OpenNebula will always launch slcci.small (1 CPU, 1024MB RAM) instances, unless otherwise specified.

Name	CPU Capacity	Memory Capacity
slcci.small	1 Core	1024 MB
slcci.medium	2 Cores	4096 MB
slcci.large	4 Cores	8192 MB

### 5.3.2 Prerequisites

**Warning:** ruby  $\geq$  1.9.3 is required, and it is not packaged in all distros supported by OpenNebula. If you are running on an older supported distro (like Centos 6.x) please update ruby or use `rvm` to run a newer ( $\geq$  1.9.3) version (remember to run `install_gems` after the ruby upgrade is done to reinstall all gems)

- You must have a working account for [SoftLayer](#)
- You need your username and API authentication key, that can be achieved in the [user profile page](#)
- The following gems are required `softlayer_api` and `configparser`. Otherwise, run the `install_gem` script as root:

```
# /usr/share/one/install_gems hybrid
```

### 5.3.3 OpenNebula Configuration

Uncomment the SoftLayer SL IM and VMM drivers from `/etc/one/oned.conf` file in order to use the driver.

```
IM_MAD = [
  name      = "sl",
  executable = "one_im_sh",
  arguments  = "-c -t 1 -r 0 sl" ]

VM_MAD = [
  name      = "sl",
  executable = "one_vmm_sh",
  arguments  = "-t 15 -r 0 sl",
  type      = "xml" ]
```

Driver flags are the same as other drivers:

FLAG	SETs
-t	Number of threads, i.e. number of actions performed at the same time
-r	Number of retries when contacting SoftLayer service

Additionally you must define your credentials, the SoftLayer datacenter to be used and the maximum capacity that you want OpenNebula to deploy on SoftLayer. In order to do this, edit the file `/etc/one/sl_driver.conf`:

```
regions:
  default:
    region_name: ams01
    username: <your_username_here>
    api_key: <your_api_key_here>
    capacity:
      slcci.small: 5
      slcci.large: 0
      slcci.large: 0
  ams01:
    region_name: ams01
    username: <your_username_here>
    api_key: <your_api_key_here>
    capacity:
      slcci.small: 5
      slcci.large: 0
      slcci.large: 0
      m1.xlarge: 0
```

In the above file, each region represents a SoftLayer datacenter. (see the *multi site region account section* for more information).

Once the file is saved, OpenNebula needs to be restarted (as oenadmin, do a 'onevm restart'), create a new Host that uses the SL drivers:

```
$ onehost create ams01 --im sl --vm sl --net dummy
```

### 5.3.4 SoftLayer Specific Template Attributes

In order to deploy an instance in SoftLayer through OpenNebula you must include an PUBLIC\_CLOUD section in the virtual machine template. This is an example of a virtual machine template that can be deployed in our local resources or in SoftLayer.

```
CPU          = 0.5
MEMORY       = 128

# Xen or KVM template machine, this will be use when submitting this VM to local resources
DISK         = [ IMAGE_ID = 3 ]
NIC          = [ NETWORK_ID = 7 ]

# SoftLayer template machine, this will be use wen submitting this VM to SoftLayer
PUBLIC_CLOUD=[
  TYPE="SOFTLAYER",
  HOSTNAME="MySLVM",
  DOMAIN="c12g.com",
  INSTANCE_TYPE="slcci.medium",
  OPERATINGSYSTEM="UBUNTU_LATEST"
]

#Add this if you want this VM to only go to the SL cloud
#SCHED_REQUIREMENTS = 'HOSTNAME = "asm01"'
```

These are the attributes that can be used in the PUBLIC\_CLOUD section of the template for TYPE SoftLayer:

ATTRIBUTES	DESCRIPTION
HOSTNAME	Hostname for the computing instance
DOMAIN	Domain for the computing instance
INSTANCE_TYPE	Specifies the capacity of the VM in terms of CPU and memory. If both STARTCPUS and MAXMEMORY are used, then this parameter is disregarded
STARTCPUS	The number of CPU cores to allocate to the VM
MAXMEMORY	The amount of memory to allocate in megabytes
HOURLYBILLING	Specifies the billing type for the instance . When true the computing instance will be billed on hourly usage, otherwise it will be billed on a monthly basis
LOCALDISK	Name of the placement group. When true the disks for the computing instance will be provisioned on the host which it runs, otherwise SAN disks will be provisioned
DEDICATEDHOST	Specifies whether or not the instance must only run on hosts with instances from the same account
DATACENTER	Specifies which datacenter the instance is to be provisioned in
OPERATINGSYSTEM	An identifier for the operating system to provision the computing instance with. A non exhaustive list of identifiers can be found <a href="#">here</a>
BLOCKDEVICETEMPLATE	A global identifier for the template to be used to provision the computing instance
BLOCKDEVICE	Size of the block device size to be presented to the VM
NETWORKCOMPONENTS	Specifies the connection speed for the instance's network components
PRIVATENETWORKON	Specifies whether or not the instance only has access to the private network (ie, if it is going to have a public IP interface or not)
PRIMARYNETWORKVLAN	Specifies the network vlan which is to be used for the frontend interface of the computing instance
PRIMARYBACKENDNETWORKVLAN	Specifies the network vlan which is to be used for the backend interface of the computing instance
USERDATA	Arbitrary data to be made available to the computing instance
SSHKEYS	SSH keys to install on the computing instance upon provisioning
POSTSCRIPT	Specifies the uri location of the script to be downloaded and run after installation is complete

**Note:** PUBLIC\_CLOUD section allows for substitutions from template and virtual network variables, the same way as the CONTEXT section allows.

Default values for all these attributes can be defined in the `/etc/one/sl_driver.default` file.

```
<!--
Default configuration attributes for the SoftLayer driver
(all domains will use these values as defaults)

Use XML syntax to specify defaults, note elements are UPPERCASE
Example:
<TEMPLATE>
  <SOFTLAYER>
    <INSTANCETYPE>slcci.small</INSTANCETYPE>
  </SOFTLAYER>
</TEMPLATE>
-->

<TEMPLATE>
  <SOFTLAYER>
    <DOMAIN>c12g.com</DOMAIN>
    <INSTANCE_TYPE>slcci.small</INSTANCE_TYPE>
    <HOURLYBILLINGFLAG>true</HOURLYBILLINGFLAG>
    <LOCALDISKFLAG>true</LOCALDISKFLAG>
  </SOFTLAYER>
</TEMPLATE>
```

### 5.3.5 Multi SoftLayer Site/Account Support

It is possible to define various SoftLayer hosts to allow OpenNebula the managing of different SoftLayer datacenters or different SoftLayer accounts. OpenNebula choses the datacenter in which to launch the VM in the following way:

- if the VM description contains the `DATACENTER` attribute, then OpenNebula knows that the VM needs to be launch in this SoftLayer datacenter
- if the name of the host matches the region name (remember, this is the same as a SL datacenter), then OpenNebula knows that the VMs sent to this host needs to be launch in that SL datacenter
- if the VM doesn't have a `DATACENTER` attribute, and the host name doesn't match any of the defined regions, then the default region is picked.

When you create a new host the credentials and endpoint for that host are retrieved from the `/etc/one/sl_driver.conf` file using the host name. Therefore, if you want to add a new host to manage a different datacenter, i.e. `sjc01`, just add your credentials and the capacity limits to the `sjc01` section in the conf file, and specify that name (`sjc01`) when creating the new host.

```
regions:
  ...
  sjc01:
    region_name: sjc01
    username:
    api_key:
    capacity:
      slcci.small: 5
      slcci.medium: 0
      slcci.large: 0
```

After that, create a new Host with the `sjc01` name:

```
$ onehost create sjc01 --im sl --vm sl --net dummy
```

If the Host name does not match any regions key, the default will be used.

You can define a different SoftLayer section in your template for each SoftLayer host, so with one template you can define different VMs depending on which host it is scheduled, just include a `HOSTNAME` attribute in each `PUBLIC_CLOUD` section:

```
PUBLIC_CLOUD = [ TYPE="SOFTLAYER",
                HOSTNAME="sjc01",
                OPERATINGSYSTEM="UBUNTU_LATEST",
                INSTANCE_TYPE="sclcci.small" ]

PUBLIC_CLOUD = [ TYPE="SOFTLAYER",
                HOSTNAME="ams01",
                OPERATINGSYSTEM="REDHAT_LATEST",
                INSTANCE_TYPE="sclcci.medium" ]
```

You will have a small Ubuntu VM launched when this VM template is sent to host `sjc01` and a medium RedHat VM launched whenever the VM template is sent to host `ams01`.

**Warning:** If only one SoftLayer site is defined, the SoftLayer driver will deploy all SoftLayer templates onto it, not paying attention to the `HOSTNAME` attribute.

### 5.3.6 Hybrid VM Templates

A powerful use of cloud bursting in OpenNebula is the ability to use hybrid templates, defining a VM if OpenNebula decides to launch it locally, and also defining it if it is going to be outsourced to SoftLayer. The idea behind this is to reference the same kind of VM even if it is incarnated by different images (the local image and the SoftLayer image).

An example of a hybrid template:

```
## Local Template section
NAME=MNyWebServer

CPU=1
MEMORY=256

DISK=[IMAGE="nginx-golden"]
NIC=[NETWORK="public"]

PUBLIC_CLOUD = [ TYPE="SOFTLAYER",
                 HOSTNAME="sjc01",
                 OPERATINGSYSTEM="UBUNTU_LATEST",
                 INSTANCE_TYPE="sclcci.small" ]
```

OpenNebula will use the first portion (from NAME to NIC) in the above template when the VM is scheduled to a local virtualization node, and the PUBLIC\_CLOUD section of TYPE="SOFTLAYER" when the VM is scheduled to an SoftLayer node (ie, when the VM is going to be launched in SoftLayer).

### 5.3.7 Testing

You must create a template file containing the information of the VMs you want to launch.

```
CPU      = 1
MEMORY   = 1700

#Xen or KVM template machine, this will be use when submitting this VM to local resources
DISK     = [ IMAGE_ID = 3 ]
NIC      = [ NETWORK_ID = 7 ]

#SoftLayer template machine, this will be use wen submitting this VM to SoftLayer

PUBLIC_CLOUD = [ TYPE="SOFTLAYER",
                 HOSTNAME="sjc01",
                 OPERATINGSYSTEM="UBUNTU_LATEST",
                 INSTANCE_TYPE="sclcci.small" ]

#Add this if you want to use only SoftLayer cloud
#SCHED_REQUIREMENTS = 'HYPERVISOR = "SOFTLAYER"'
```

You can submit and control the template using the OpenNebula interface:

```
$ onetemplate create sltemplate
$ ontemplate instantiate sltemplate
```

Now you can monitor the state of the VM with

```
$ onevm list
  ID USER      GROUP      NAME          STAT CPU    MEM      HOSTNAME      TIME
  0  oneadmin  oneadmin  one-0         runn  0      0K          sjc01         0d 07:03
```

Also you can see information (like IP address) related to the SoftLayer instance launched via the command. The attributes available are:

- SL\_CRED\_PASSWORD
- SL\_CRED\_USER
- SL\_DOMAIN
- SL\_FULLYQUALIFIEDDOMAINNAME
- SL\_GLOBALIDENTIFIER
- SL\_HOSTNAME
- SL\_ID
- SL\_MAXCPU
- SL\_MAXMEMORY
- SL\_PRIMARYBACKENDIPADDRESS
- SL\_PRIMARYIPADDRESS
- SL\_STARTCPUS
- SL\_UUID

```
$ onevm show 0
VIRTUAL MACHINE 0 INFORMATION
ID                : 32
NAME              : one-32
USER              : oneadmin
GROUP             : oneadmin
STATE             : ACTIVE
LCM_STATE         : RUNNING
RESCHED           : No
HOST              : sjc01
CLUSTER ID        : -1
START TIME        : 06/05 20:01:46
END TIME          : -
DEPLOY ID         : 4978604

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

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

VIRTUAL MACHINE HISTORY
SEQ HOST          ACTION          DS          START          TIME          PROLOG
  0 sjc01         none          -1 06/05 20:01:59 3d 16h53m 0h00m00s

USER TEMPLATE
PUBLIC_CLOUD = [ TYPE="SOFTLAYER",
                 HOSTNAME="sjc01",
                 OPERATINGSYSTEM="UBUNTU_LATEST",
```

```
INSTANCE_TYPE="sclcci.small" ]

VIRTUAL MACHINE TEMPLATE
AUTOMATIC_REQUIREMENTS="!(PUBLIC_CLOUD = YES) | (PUBLIC_CLOUD = YES & (HYPERVISOR = SOFTLAYER | HYPERVISOR = KVM))"
CPU="1"
MEMORY="1024"
SL_CRED_PASSWORD="xxxxxx"
SL_CRED_USER="root"
SL_DOMAIN="c12g.com"
SL_FULLYQUALIFIEDDOMAINNAME="MySLVM.c12g.com"
SL_GLOBALIDENTIFIER="xx299e80-96a0-434f-b228-430689c45ffb"
SL_HOSTNAME="MySLVM"
SL_ID="4978604"
SL_MAXCPU="2"
SL_MAXMEMORY="4096"
SL_PRIMARYBACKENDIPADDRESS="10.104.201.xxx"
SL_PRIMARYIPADDRESS="5.153.45.xx"
SL_STARTCPUS="2"
SL_UUID="xxxxxxxx-a0cc-e648-2ebd-e5fb2a500965"
```

### 5.3.8 Scheduler Configuration

Since SoftLayer Hosts are treated by the scheduler like any other host, VMs will be automatically deployed in them. But you probably want to lower their priority and start using them only when the local infrastructure is full.

#### Configure the Priority

The SoftLayer drivers return a probe with the value `PRIORITY = -1`. This can be used by the scheduler, configuring the ‘fixed’ policy in `sched.conf`:

```
DEFAULT_SCHED = [
    policy = 4
]
```

The local hosts will have a priority of 0 by default, but you could set any value manually with the ‘onehost/onecluster update’ command.

There are two other parameters that you may want to adjust in `sched.conf`:

```
- MAX_DISPATCH: Maximum number of Virtual Machines actually dispatched to a host in each scheduling action
- MAX_HOST: Maximum number of Virtual Machines dispatched to a given host in each scheduling action
```

In a scheduling cycle, when `MAX_HOST` number of VMs have been deployed to a host, it is discarded for the next pending VMs.

For example, having this configuration:

- `MAX_HOST = 1`
- `MAX_DISPATCH = 30`
- 2 Hosts: 1 in the local infrastructure, and 1 using the SoftLayer drivers
- 2 pending VMs

The first VM will be deployed in the local host. The second VM will have also sort the local host with higher priority, but because 1 VMs was already deployed, the second VM will be launched in SoftLayer.



A quick way to ensure that your local infrastructure will be always used before the SoftLayer hosts is to set **MAX\_DISPATH to the number of local hosts**.

### Force a Local or Remote Deployment

The SoftLayer drivers report the host attribute `PUBLIC_CLOUD = YES`. Knowing this, you can use that attribute in your VM requirements.

To force a VM deployment in a local host, use:

```
SCHED_REQUIREMENTS = "! (PUBLIC_CLOUD = YES) "
```

To force a VM deployment in a SoftLayer host, use:

```
SCHED_REQUIREMENTS = "PUBLIC_CLOUD = YES"
```

## 5.3.9 Importing VMs

VMs running on SoftLayer that were not launched through OpenNebula can be imported in OpenNebula.

## 5.4 Azure Driver

### 5.4.1 Considerations & Limitations

You should take into account the following technical considerations when using the Microsoft Azure (AZ) cloud with OpenNebula:

- There is no direct access to the hypervisor, so it cannot be monitored (we don't know where the VM is running on the Azure cloud).
- The usual OpenNebula functionality for snapshotting, hot-plugging, or migration is not available with Azure (currently).
- By default OpenNebula will always launch Small (1 CPU, 1792 MB RAM) instances, unless otherwise specified.

Name	CPU Capacity	Memory Capacity
ExtraSmall	0.1 Cores	768 MB
Small	1 Cores	1792 MB
Medium	2 Cores	3584 MB
Large	4 Cores	7168 MB
ExtraLarge	8 Cores	14336 MB
A5	2 Cores	14336 MB
A6	4 Cores	28672 MB
A7	8 Cores	57344 MB
A8	8 Cores	57344 MB
A9	16 Cores	114688 MB

### 5.4.2 Prerequisites

**Warning:** ruby  $\geq$  1.9.3 is required, and it is not packaged in all distros supported by OpenNebula. If you are running on an older supported distro (like Centos 6.x) please update ruby or use `rvm` to run a newer ( $\geq$  1.9.3) version (remember to run `install_gems` after the ruby upgrade is done to reinstall all gems)

- You must have a working account for [Azure](#)
- You need your Azure credentials (Information on how to manage Azure certificates can be found [here](#) ). The information can be obtained from the [Management Azure page](#)
- First, the Subscription ID, that can be uploaded and retrieved from Settings -> Subscriptions
- Second, the Management Certificate file, that can be created with the following steps- We need the .pem file (for the ruby gem) and the .cer file (to upload to Azure):

```
## Install openssl
## CentOS
$ sudo yum install openssl
## Ubuntu
$ sudo apt-get install openssl

## Create certificate
$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout myPrivateKey.key -out myCert.pem
$ chmod 600 myPrivateKey.key

## Concatenate key and pem certificate
$ cat myCert.pem myPrivateKey.key > vOneCloud.pem

## Generate .cer file for Azure
$ openssl x509 -outform der -in myCert.pem -out myCert.cer
```

- Third, the certificate file (.cer) has to be uploaded to Settings -> Management Certificates
- The following gem is required: azure. Otherwise, run the install\_gems script as root:

```
# /usr/share/one/install_gems cloud
```

### 5.4.3 OpenNebula Configuration

Uncomment the Azure AZ IM and VMM drivers from `/etc/one/oned.conf` file in order to use the driver.

```
IM_MAD = [
  name          = "az",
  executable    = "one_im_sh",
  arguments     = "-c -t 1 -r 0 az" ]

VM_MAD = [
  name          = "az",
  executable    = "one_vmm_sh",
  arguments     = "-t 15 -r 0 az",
  type          = "xml" ]
```

Driver flags are the same as other drivers:

FLAG	SETs
-t	Number of threads, i.e. number of actions performed at the same time
-r	Number of retries when contacting Azure service

Additionally you must define your credentials, the Azure location to be used and the maximum capacity that you want OpenNebula to deploy on Azure. In order to do this, edit the file `/etc/one/az_driver.conf`:

```
default:
  region_name: "West Europe"
  pem_management_cert: <path-to-your-vonecloud-pem-certificate-here>
  subscription_id: <your-subscription-id-here>
```

```

management_endpoint:
  capacity:
    Small: 5
    Medium: 1
    Large: 0
west-europe:
  region_name: "West Europe"
  pem_management_cert: <path-to-your-vonecloud-pem-certificate-here>
  subscription_id: <your-subscription-id-here>
  management_endpoint:
  capacity:
    Small: 5
    Medium: 1
    Large: 0

```

In the above file, each region represents an [Azure datacenter](#) (Microsoft doesn't provide an official list). (see the [multi site region account section](#) for more information).

If the OpenNebula frontend needs to use a proxy to connect to internet you also need to configure the proxy in that file. The parameter is called `proxy_uri`. Authenticated proxies are not supported, that is, the ones that require user name and password. For example, if the proxy is in `10.0.0.1` and its port is `8080` the configuration line should read:

```
proxy_uri: http://10.0.0.1:8080
```

Once the file is saved, OpenNebula needs to be restarted (as `oneadmin`, do a `'onevm restart'`), create a new Host that uses the AZ drivers:

```
$ onehost create west-europe -i az -v az -n dummy
```

## 5.4.4 Azure Specific Template Attributes

In order to deploy an instance in Azure through OpenNebula you must include an `PUBLIC_CLOUD` section in the virtual machine template. This is an example of a virtual machine template that can be deployed in our local resources or in Azure.

```

CPU      = 0.5
MEMORY   = 128

# Xen or KVM template machine, this will be use when submitting this VM to local resources
DISK     = [ IMAGE_ID = 3 ]
NIC      = [ NETWORK_ID = 7 ]

# Azure template machine, this will be use wen submitting this VM to Azure
PUBLIC_CLOUD = [
  TYPE=AZURE,
  INSTANCE_TYPE=ExtraSmall,
  IMAGE=b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-14_04-LTS-amd64-server-20140606.1-en-us-30GB,
  VM_USER="azuser",
  VM_PASSWORD="mypassword",
  WIN_RM="https",
  TCP_ENDPOINTS="80",
  SSHPORT=2222
]

#Add this if you want this VM to only go to the West EuropeAzure cloud
#SCHED_REQUIREMENTS = 'HOSTNAME = "west-europe"'

```

These are the attributes that can be used in the PUBLIC\_CLOUD section of the template for TYPE “AZURE”:

ATTRIBUTES	DESCRIPTION
INSTANCE_TYPE	Specifies the capacity of the VM in terms of CPU and memory
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 <a href="#">here</a>
VM_USER	If the selected IMAGE is prepared for Azure provisioning, a username can be specified here to access the VM once booted
VM_PASSWORD	Password for VM_USER
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.
STORAGE_ACCOUNT	Specify the storage account where this VM will belong
WIN_RM	Comma-separated list of possible protocols to access this Windows VM
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”.
TCP_ENDPOINTS	Comma-separated list of TCP ports to be accesible from the public internet to this VM
SSHPORT	Port where the VMs ssh server will listen on
VIRTUAL_NETWORK_NAME	Name of the virtual network to which this VM will be connected
SUBNET	Name of the particular Subnet where this VM will be connected to
AVAILABILITY_SET	Name of the availability set to which this VM will belong
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

**Note:** The PUBLIC\_CLOUD sections allow for substitutions from template and virtual network variables, the same way as the CONTEXT section allows.

Default values for all these attributes can be defined in the /etc/one/az\_driver.default file.

```
<!--
Default configuration attributes for the Azure driver
(all domains will use these values as defaults)
Valid attributes are: INSTANCE_TYPE, IMAGE, VM_USER, VM_PASSWORD, LOCATION,
STORAGE_ACCOUNT, WIN_RM, CLOUD_SERVICE, TCP_ENDPOINTS, SSHPORT, AFFINITY_GROUP,
VIRTUAL_NETWORK_NAME, SUBNET and AVAILABILITY_SET
Use XML syntax to specify defaults, note elements are UPGCASE
Example:
<TEMPLATE>
  <AZURE>
    <LOCATION>west-europe</LOCATION>
    <INSTANCE_TYPE>Small</INSTANCE_TYPE>
    <CLOUD_SERVICE>MyDefaultCloudService</CLOUD_SERVICE>
    <IMAGE>0b11de9248dd4d87b18621318e037d37__RightImage-Ubuntu-12.04-x64-v13.4</IMAGE>
    <VM_USER>MyUser</VM_USER>
    <VM_PASSWORD>MyPassword</VM_PASSWORD>
    <STORAGE_ACCOUNT>MyStorageAccountName</STORAGE_ACCOUNT>
    <WIN_RM>http</WIN_RM>
    <CLOUD_SERVICE>MyCloudServiceName</CLOUD_SERVICE>
    <TCP_ENDPOINTS>80,3389:3390</TCP_ENDPOINTS>
    <SSHPORT>2222</SSHPORT>
    <AFFINITY_GROUP>MyAffinityGroup</AFFINITY_GROUP>
    <VIRTUAL_NETWORK_NAME>MyVirtualNetwork</VIRTUAL_NETWORK_NAME>
    <SUBNET>MySubNet<SUBNET>
    <AVAILABILITY_SET>MyAvailabilitySetName<AVAILABILITY_SET>
  </AZURE>
</TEMPLATE>
-->
```

```
<TEMPLATE>
  <AZURE>
    <LOCATION>west-europe</LOCATION>
    <INSTANCE_TYPE>Small</INSTANCE_TYPE>
  </AZURE>
</TEMPLATE>
```

### 5.4.5 Multi Azure Location/Account Support

It is possible to define various Azure hosts to allow OpenNebula the managing of different Azure locations or different Azure accounts. OpenNebula choses the datacenter in which to launch the VM in the following way:

- if the VM description contains the LOCATION attribute, then OpenNebula knows that the VM needs to be launch in this Azure location
- if the name of the host matches the region name (remember, this is the same as an Azure location), then OpenNebula knows that the VMs sent to this host needs to be launched in that Azure datacenter
- if the VM doesn't have a LOCATION attribute, and the host name doesn't match any of the defined regions, then the default region is picked.

When you create a new host the credentials and endpoint for that host are retrieved from the `/etc/one/az_driver.conf` file using the host name. Therefore, if you want to add a new host to manage a different datacenter, i.e. `west-europe`, just add your credentials and the capacity limits to the `west-europe` section in the conf file, and specify that name (`west-europe`) when creating the new host.

```
regions:
  ...
  west-europe:
    region_name: "West Europe"
    pem_management_cert: "<path-to-your-vonecloud-pem-certificate-here>"
    subscription_id: "your-subscription-id"
    management_endpoint:
    capacity:
      Small: 5
      Medium: 1
      Large: 0
```

After that, create a new Host with the `west-europe` name:

```
$ onehost create west-europe -i az -v az -n dummy
```

If the Host name does not match any regions key, the default will be used.

You can define a different Azure section in your template for each Azure host, so with one template you can define different VMs depending on which host it is scheduled, just include a LOCATION attribute in each PUBLIC\_CLOUD section:

```
PUBLIC_CLOUD = [ TYPE=AZURE,
                INSTANCE_TYPE=Small,
                IMAGE=b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-14_04-LTS-amd64-server-20140606.1-en-
                VM_USER="MyUserName",
                VM_PASSWORD="MyPassword",
                LOCATION="brazil-south"
              ]

PUBLIC_CLOUD = [ TYPE=AZURE,
                INSTANCE_TYPE=Medium,
```

```
        IMAGE=0b11de9248dd4d87b18621318e037d37__RightImage-Ubuntu-12.04-x64-v13.4,  
        VM_USER="MyUserName",  
        VM_PASSWORD="MyPassword",  
        LOCATION="west-europe"  
    ]
```

You will have a small Ubuntu 14.04 VM launched when this VM template is sent to host *brazil-south* and a medium Ubuntu 13.04 VM launched whenever the VM template is sent to host *west-europe*.

**Warning:** If only one Azure host is defined, the Azure driver will deploy all Azure templates onto it, not paying attention to the **LOCATION** attribute.

### 5.4.6 Hybrid VM Templates

A powerful use of cloud bursting in OpenNebula is the ability to use hybrid templates, defining a VM if OpenNebula decides to launch it locally, and also defining it if it is going to be outsourced to Azure. The idea behind this is to reference the same kind of VM even if it is incarnated by different images (the local image and the Azure image).

An example of a hybrid template:

```
## Local Template section  
NAME=MNyWebServer  
  
CPU=1  
MEMORY=256  
  
DISK=[ IMAGE="nginx-golden" ]  
NIC=[ NETWORK="public" ]  
  
PUBLIC_CLOUD = [ TYPE=AZURE,  
                 INSTANCE_TYPE=Medium,  
                 IMAGE=0b11de9248dd4d87b18621318e037d37__RightImage-Ubuntu-12.04-x64-v13.4,  
                 VM_USER="MyUserName",  
                 VM_PASSWORD="MyPassword",  
                 LOCATION="west-europe"  
    ]
```

OpenNebula will use the first portion (from NAME to NIC) in the above template when the VM is scheduled to a local virtualization node, and the PUBLIC\_CLOUD section of TYPE="AZURE" when the VM is scheduled to an Azure node (ie, when the VM is going to be launched in Azure).

### 5.4.7 Testing

You must create a template file containing the information of the VMs you want to launch.

```
CPU      = 1  
MEMORY   = 1700  
  
# Xen or KVM template machine, this will be use when submitting this VM to local resources  
DISK     = [ IMAGE_ID = 3 ]  
NIC      = [ NETWORK_ID = 7 ]  
  
# Azure template machine, this will be use when submitting this VM to Azure  
PUBLIC_CLOUD = [ TYPE=AZURE,
```

```

        INSTANCE_TYPE=Medium,
        IMAGE=0b11de9248dd4d87b18621318e037d37__RightImage-Ubuntu-12.04-x64-v13.4,
        VM_USER="MyUserName",
        VM_PASSWORD="MyPassword",
        LOCATION="west-europe"
    ]

# Add this if you want to use only Azure cloud
#SCHED_REQUIREMENTS = 'HYPERVISOR = "AZURE"'

```

You can submit and control the template using the OpenNebula interface:

```

$ onetemplate create aztemplate
$ onetemplate instantiate aztemplate

```

Now you can monitor the state of the VM with

```

$ onevm list
  ID USER      GROUP      NAME          STAT CPU    MEM      HOSTNAME      TIME
   0 oneadmin  oneadmin  one-0         runn  0      0K        west-europe   0d 07:03

```

Also you can see information (like IP address) related to the Azure instance launched via the command. The attributes available are:

- AZ\_AVAILABILITY\_SET\_NAME
- AZ\_CLOUD\_SERVICE\_NAME,
- AZ\_DATA\_DISKS,
- AZ\_DEPLOYMENT\_NAME,
- AZ\_DISK\_NAME,
- AZ\_HOSTNAME,
- AZ\_IMAGE,
- AZ\_IPADDRESS,
- AZ\_MEDIA\_LINK,
- AZ\_OS\_TYPE,
- AZ\_ROLE\_SIZE,
- AZ\_TCP\_ENDPOINTS,
- AZ\_UDP\_ENDPOINTS,
- AZ\_VIRTUAL\_NETWORK\_NAME

```

$ onevm show 0
VIRTUAL MACHINE 0 INFORMATION
ID                : 0
NAME              : one-0
USER              : oneadmin
GROUP            : oneadmin
STATE             : ACTIVE
LCM_STATE         : RUNNING
RESCHED           : No
START TIME       : 06/25 13:05:29
END TIME         : -
HOST              : west-europe

```

```

CLUSTER ID          : -1
DEPLOY ID           : one-0_opennebuladefaultcloudservicename-0

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

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

VIRTUAL MACHINE HISTORY
SEQ HOST            ACTION                DS          START          TIME          PROLOG
  0 west-europe     none                    -1  06/25 13:06:25  0d 00h06m    0h00m00s

USER TEMPLATE
PUBLIC_CLOUD=[
  IMAGE="b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-14_04-LTS-amd64-server-20140606.1-en-us-30GB",
  INSTANCE_TYPE="ExtraSmall",
  SSH_PORT="2222",
  TCP_ENDPOINTS="80",
  TYPE="AZURE",
  VM_PASSWORD="MyVMPassWord",
  VM_USER="MyUserName",
  WIN_RM="https" ]
VIRTUAL MACHINE TEMPLATE
AUTOMATIC_REQUIREMENTS="!(PUBLIC_CLOUD = YES) | (PUBLIC_CLOUD = YES & (HYPERVISOR = AZURE | HYPERVISOR = VMWARE))"
AZ_CLOUD_SERVICE_NAME="opennebuladefaultcloudservicename-0"
AZ_DEPLOYMENT_NAME="OpenNebulaDefaultCloudServiceName-0"
AZ_DISK_NAME="OpenNebulaDefaultCloudServiceName-0-one-0_OpenNebulaDefaultCloudServiceName-0-0-20140606.1-en-us-30GB"
AZ_HOSTNAME="ubuntu"
AZ_IMAGE="b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-14_04-LTS-amd64-server-20140606.1-en-us-30GB"
AZ_IPADDRESS="191.233.70.93"
AZ_MEDIA_LINK="http://one0opennebuladefaultclo.blob.core.windows.net/vhds/disk_2014_06_25_13_07.vhd"
AZ_OS_TYPE="Linux"
AZ_ROLE_SIZE="ExtraSmall"
AZ_TCP_ENDPOINTS="name=SSH,vip=23.97.101.202,publicport=2222,local_port=22,local_port=tcp;name=TCP-PORT"
CPU="1"
MEMORY="1024"
VMID="0"

```

## 5.4.8 Scheduler Configuration

Since Azure Hosts are treated by the scheduler like any other host, VMs will be automatically deployed in them. But you probably want to lower their priority and start using them only when the local infrastructure is full.

### Configure the Priority

The Azure drivers return a probe with the value `PRIORITY = -1`. This can be used by the scheduler, configuring the 'fixed' policy in `sched.conf`:



```
DEFAULT_SCHED = [
    policy = 4
]
```

The local hosts will have a priority of 0 by default, but you could set any value manually with the ‘onehost/onecluster update’ command.

There are two other parameters that you may want to adjust in sched.conf:

```
- MAX_DISPATCH: Maximum number of Virtual Machines actually dispatched to a host in each scheduling
- MAX_HOST: Maximum number of Virtual Machines dispatched to a given host in each scheduling action
```

In a scheduling cycle, when MAX\_HOST number of VMs have been deployed to a host, it is discarded for the next pending VMs.

For example, having this configuration:

- MAX\_HOST = 1
- MAX\_DISPATCH = 30
- 2 Hosts: 1 in the local infrastructure, and 1 using the Azure drivers
- 2 pending VMs

The first VM will be deployed in the local host. The second VM will have also sort the local host with higher priority, but because 1 VMs was already deployed, the second VM will be launched in Azure.

A quick way to ensure that your local infrastructure will be always used before the Azure hosts is to **set MAX\_DISPATCH to the number of local hosts**.

### Force a Local or Remote Deployment

The Azure drivers report the host attribute PUBLIC\_CLOUD = YES. Knowing this, you can use that attribute in your VM requirements.

To force a VM deployment in a local host, use:

```
SCHED_REQUIREMENTS = "! (PUBLIC_CLOUD = YES) "
```

To force a VM deployment in a Azure host, use:

```
SCHED_REQUIREMENTS = "PUBLIC_CLOUD = YES"
```

### 5.4.9 Importing VMs

VMs running on Azure that were not launched through OpenNebula can be imported in OpenNebula.

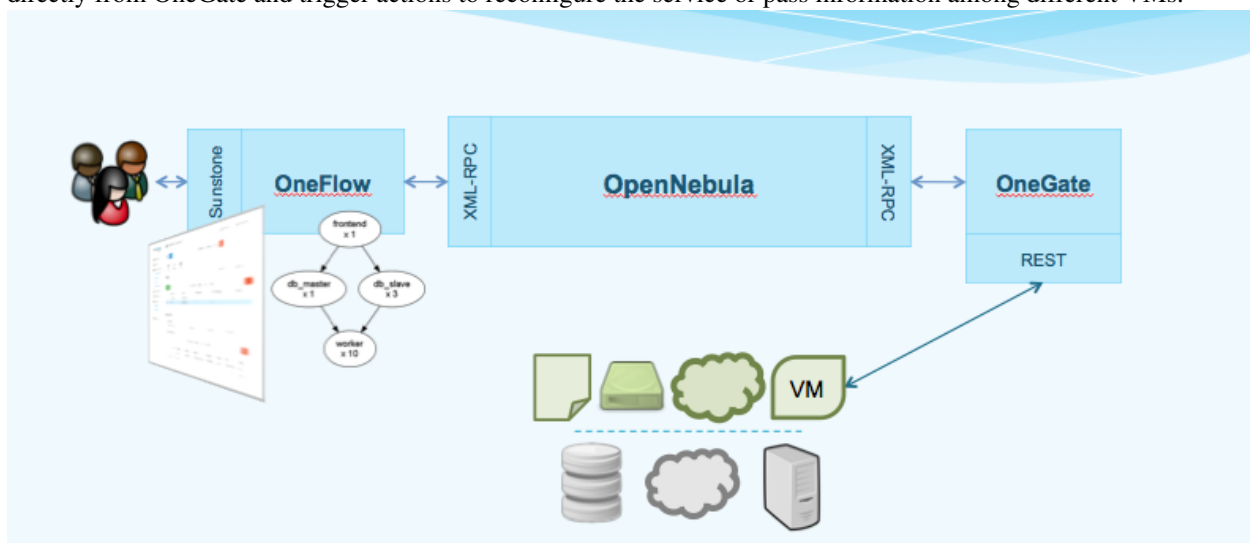


## APPLICATION INSIGHT

### 6.1 OneGate

OneGate allows Virtual Machine guests to pull and push VM information from OpenNebula. Users and administrators can use it to gather metrics, detect problems in their applications, and trigger OneFlow elasticity rules from inside the VM.

For Virtual Machines that are part of a Multi-VM Application (Service), they can also retrieve the Service information directly from OneGate and trigger actions to reconfigure the service or pass information among different VMs.



#### 6.1.1 Next Steps

- *OneGate Server Configuration*
- *Application Monitoring*

### 6.2 OneGate Server Configuration

The OneGate service allows Virtual Machines guests to pull and push VM information from OpenNebula. Although it is installed by default, its use is completely optional.

## 6.2.1 Requirements

Check the Installation guide for details of what package you have to install depending on your distribution

Currently, OneGate is not supported for VMs instantiated in Softlayer and Azure, since the authentication token is not available inside these VMs. OneGate support for these drivers will be include in upcoming releases. Since OpenNebula 4.14.2 the authentication token is available for :ref: ‘ instances deployed in EC2 <context\_ec2> ‘.

## 6.2.2 Configuration

The OneGate configuration file can be found at `/etc/one/onegate-server.conf`. It uses YAML syntax to define the following options:

### Server Configuration

- `one_xmlrpc`: OpenNebula daemon host and port
- `host`: Host where OneGate will listen
- `port`: Port where OneGate will listen
- `ssl_server`: SSL proxy URL that serves the API (set if is being used)

### Log

- `debug_level`: Log debug level. 0 = ERROR, 1 = WARNING, 2 = INFO, 3 = DEBUG

### Auth

- `auth`: Authentication driver for incomming requests.
  - `onegate`: based on token provided in the context
- `core_auth`: Authentication driver to communicate with OpenNebula core.
  - `cipher` for symmetric cipher encryption of tokens
  - `x509` for x509 certificate encryption of tokens. For more information, visit the OpenNebula Cloud Auth documentation.
- `onflow_server` Endpoint where the OneFlow server is listening.
- `permissions` By default OneGate exposes all the available API calls, each of the actions can be enabled/disabled in the server configuration.

This is the default file

```
#####
# Server Configuration
#####

# OpenNebula sever contact information
#
:one_xmlrpc: http://localhost:2633/RPC2

# Server Configuration
#
:host: 127.0.0.1
:port: 5030

# SSL proxy URL that serves the API (set if is being used)
#:ssl_server: https://service.endpoint.fqdn:port/
```

```
#####
# Log
#####

# Log debug level
# 0 = ERROR, 1 = WARNING, 2 = INFO, 3 = DEBUG
#
:debug_level: 3

#####
# Auth
#####

# Authentication driver for incoming requests
# onegate, based on token provided in the context
#
:auth: onegate

# Authentication driver to communicate with OpenNebula core
# cipher, for symmetric cipher encryption of tokens
# x509, for x509 certificate encryption of tokens
#
:core_auth: cipher

#####
# OneFlow Endpoint
#####

:oneflow_server: http://localhost:2474

#####
# Permissions
#####

:permissions:
  :vm:
    :show: true
    :show_by_id: true
    :update: true
    :update_by_id: true
    :action_by_id: true
  :service:
    :show: true
    :change_cardinality: true
```

### 6.2.3 Start OneGate

To start and stop the server, use the `onegate-server start/stop` command:

```
$ onegate-server start
onegate-server started
```

**Warning:** By default, the server will only listen to requests coming from localhost. Change the `:host` attribute in `/etc/one/onegate-server.conf` to your server public IP, or `0.0.0.0` so onegate will listen on any interface.

Inside `/var/log/one/` you will find new log files for the server:

```
/var/log/one/onegate.error
/var/log/one/onegate.log
```

## 6.2.4 Use OneGate

Before your VMs can communicate with OneGate, you need to edit `/etc/one/oned.conf` and set the OneGate endpoint. This IP must be reachable from your VMs.

```
ONEGATE_ENDPOINT = "http://192.168.0.5:5030"
```

Continue to the *OneGate usage guide*.

## 6.2.5 Configuring a SSL Proxy

This is an example on how to configure Nginx as a ssl proxy for Onegate in Ubuntu.

Update your package lists and install Nginx:

```
sudo apt-get update
sudo apt-get install nginx
```

You should get an official signed certificate, but for the purpose of this example we will generate a self-signed SSL certificate:

```
cd /etc/one
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/one/cert.key -out /etc/one/cert
```

Next you will need to edit the default Nginx configuration file or generate a new one. Change the `ONEGATE_ENDPOINT` variable with your own domain name.

```
server {
    listen 80;
    return 301 https://$host$request_uri;
}

server {
    listen 443;
    server_name ONEGATE_ENDPOINT;

    ssl_certificate      /etc/one/cert.crt;
    ssl_certificate_key  /etc/one/cert.key;

    ssl on;
    ssl_session_cache  builtin:1000  shared:SSL:10m;
    ssl_protocols     TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers       HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
    ssl_prefer_server_ciphers on;

    access_log        /var/log/nginx/onegate.access.log;

    location / {

        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
```

```

proxy_set_header    X-Forwarded-Proto $scheme;

# Fix the "It appears that your reverse proxy set up is broken" error.
proxy_pass          http://localhost:5030;
proxy_read_timeout  90;

proxy_redirect      http://localhost:5030 https://ONEGATE_ENDPOINT;
}
}

```

Update `/etc/one/oned.conf` with the new OneGate endpoint

```
ONEGATE_ENDPOINT = "https://ONEGATE_ENDPOINT"
```

Update `/etc/one/onegate-server.conf` with the new OneGate endpoint and uncomment the `ssl_server` parameter

```
:ssl_server: https://ONEGATE_ENDPOINT
```

Then restart `oned`, `onegate-server` and `Nginx`:

```

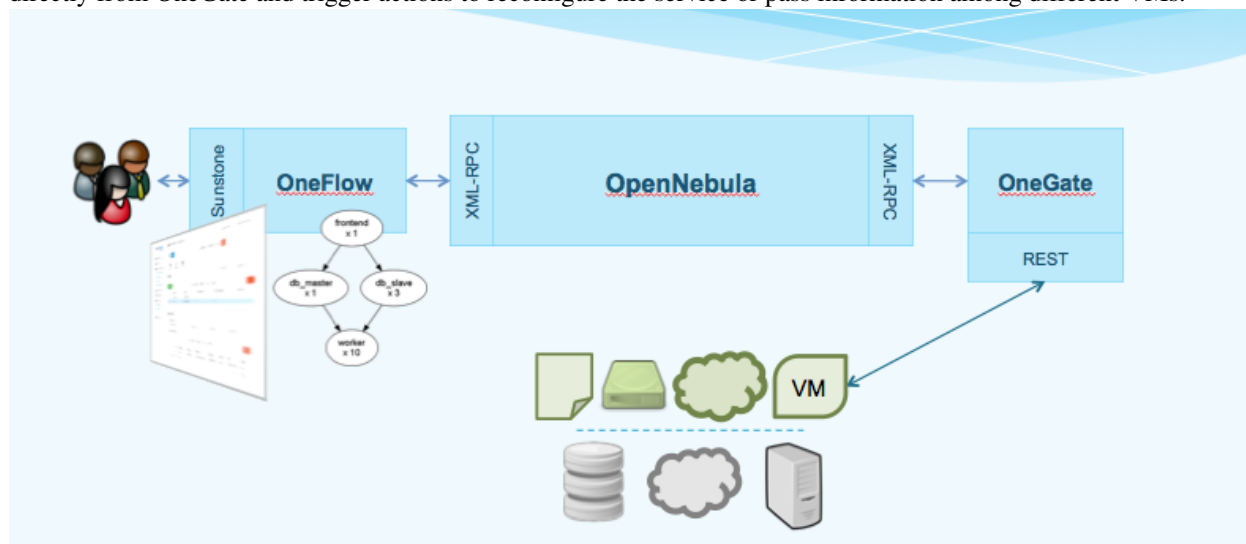
sudo service nginx restart
sudo service opennebula restart
sudo service opennebula-gate restart

```

## 6.3 OneGate Usage

OneGate allows Virtual Machine guests to pull and push VM information from OpenNebula. Users and administrators can use it to gather metrics, detect problems in their applications, and trigger OneFlow elasticity rules from inside the VM.

For Virtual Machines that are part of a Multi-VM Application (Service), they can also retrieve the Service information directly from OneGate and trigger actions to reconfigure the service or pass information among different VMs.



### 6.3.1 OneGate Workflow Explained

OneGate is a server that listens to http connections from the Virtual Machines. OpenNebula assigns an individual token to each VM instance, and Applications running inside the VM use this token to interact with the OneGate API. This token is generated using VM information and signed with the user `TOKEN_PASSWORD`. This password can be changed updating the user template, but tokens from existing vms will not work anymore.

### 6.3.2 OneGate Usage

First, the cloud administrator must configure and start the *OneGate server*.

#### Setup the VM Template

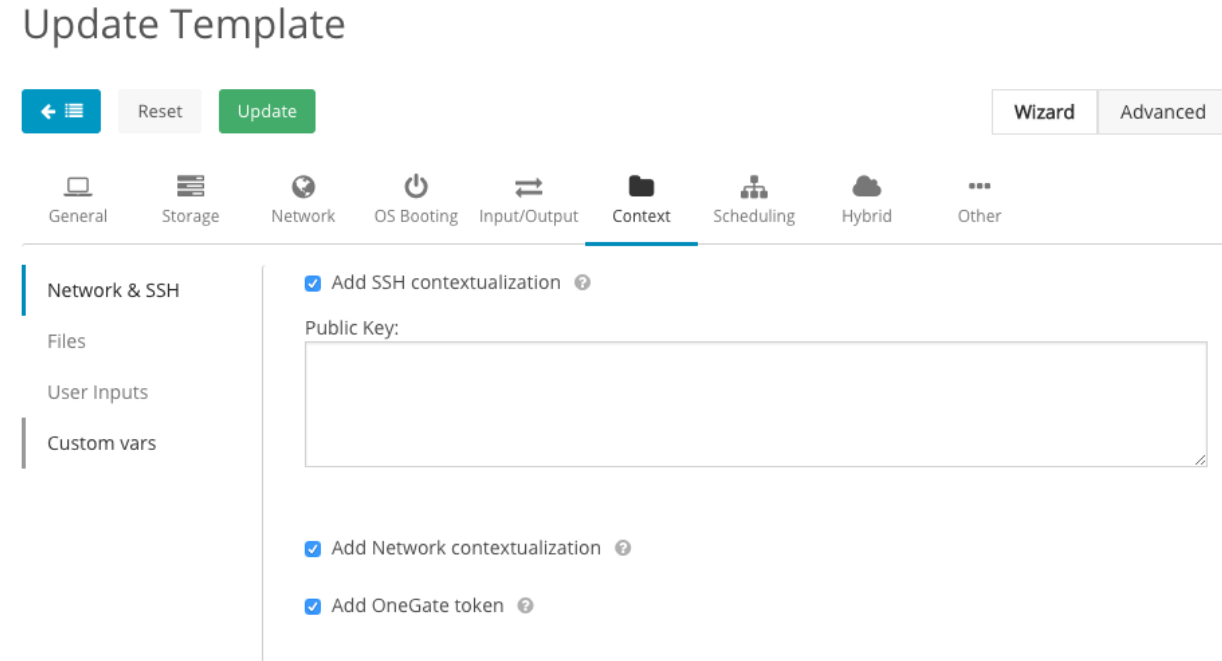
Your VM Template must set the `CONTEXT/TOKEN` attribute to *yes*.

```
CPU      = "0.5"
MEMORY  = "128"

DISK = [
  IMAGE_ID = "0" ]
NIC = [
  NETWORK_ID = "0" ]

CONTEXT = [
  TOKEN = "YES" ]
```

or check the OneGate checkbox in Sunstone:



When this Template is instantiated, OpenNebula will automatically add the `ONEGATE_ENDPOINT` context variable, and a `token.txt` will be placed in the context `cdrom`. This `token.txt` file is only accessible from inside the VM.



```
...
CONTEXT=[
  DISK_ID="1",
  ONEGATE_ENDPOINT="http://192.168.0.1:5030",
  TARGET="hdb",
  TOKEN="YES" ]
```

In vCenter this information is available in the extraConfig section of the VM metadata, available in the guest OS through the VMware tools as explained in the contextualization guide.

### Using the OneGate Client inside the Guest VM

A ruby client that implements the OneGate API is included in the official [OpenNebula context packages](#). This is a simple command line interface to interact with the OneGate server, it will handle the authentication and requests complexity.

### OneGate Client Usage

Available commands and usage are shown with *onegate -h*:

```
$ onegate -h
Available commands
  $ onegate vm show [VMID] [--json]

  $ onegate vm update [VMID] --data KEY=VALUE[\nKEY2=VALUE2]

$ onegate vm ACTION VMID
  $ onegate vm resume [VMID]
  $ onegate vm stop [VMID]
  $ onegate vm suspend [VMID]
  $ onegate vm delete [VMID] [--hard]
  $ onegate vm shutdown [VMID] [--hard]
  $ onegate vm reboot [VMID] [--hard]
  $ onegate vm poweroff [VMID] [--hard]
  $ onegate vm resubmit [VMID]
  $ onegate vm resched [VMID]
  $ onegate vm unresched [VMID]
  $ onegate vm hold [VMID]
  $ onegate vm release [VMID]

$ onegate service show [--json]

$ onegate service scale --role ROLE --cardinality CARDINALITY
```

With the appropriate policies implemented in the service, these mechanisms allow services to be self-managed, enabling self-configuration, self-healing, self-optimization and self-protection.

- **Self-Awareness**

There are several actions available to retrieve information of the Virtual Machine and the Service it belongs to. A Virtual Machine can also retrieve information of other Virtual Machines that are part of the Service.

### Retrieving information of the VM

Using the `onegate vm show` command the information of the Virtual Machine will be retrieved. For a detailed version use the `-json` option and all the information will be returned in JSON format.

If no argument is provided, the information of the current Virtual Machine will be retrieved. Alternatively, a VM ID can be provided to retrieve the information of a specific Virtual Machine.

```
$ onegate vm show
VM 8
NAME           : master_0_(service_1)
STATE          : RUNNING
IP             : 192.168.122.23
```

### Retrieving information of the Service

Using the `onegate service show` command the information of the Service will be retrieved. For a detailed version use the `-json` option and all the information will be returned in JSON format.

```
$ onegate service show
SERVICE 1
NAME           : PANACEA service
STATE          : RUNNING

ROLE master
VM 8
NAME           : master_0_(service_1)
STATE          : RUNNING
IP             : 192.168.122.23

ROLE slave
VM 9
NAME           : slave_0_(service_1)
STATE          : RUNNING
```

### Updating the VM Information

The Virtual Machine can update the information of itself or other Virtual Machine of the Service. This information can be retrieved from any of the Virtual Machines.

For example, the master Virtual Machine can change the *ACTIVE* attribute from one Virtual Machine to another one. Then, this information can be used to trigger any kind of action in the other Virtual Machine.

```
$ onegate vm update 9 --data ACTIVE=YES
$ onegate vm show 9 --json
{
  "VM": {
    "NAME": "slave_0_(service_1)",
    "ID": "9",
    "STATE": "3",
    "LCM_STATE": "3",
    "USER_TEMPLATE": {
      "ACTIVE": "YES",
      "FROM_APP": "4fc76a938fb81d3517000003",
      "FROM_APP_NAME": "ttylinux - kvm",
      "LOGO": "images/logos/linux.png",
```

```

    "ROLE_NAME": "slave",
    "SERVICE_ID": "1"
  },
  "TEMPLATE": {
    "NIC": [

    ]
  }
}
}
}

```

- **Self-Configuration**

There are several actions to adapt the Service to a given situation. Actions on any of the Virtual Machines can be performed individually. Also, the size of the service can be customized just specifying a cardinality for each of the roles.

### Performing actions on a VM

One of the following actions can be performed in any of the Virtual Machines of the Service. Again, if no VM ID is provided as argument, the action will be performed on the current Virtual Machine.

- `onegate vm resume`: Resumes the execution of the a saved VM. Valid states: STOPPED, SUSPENDED, UNDEPLOYED, POWEROFF
- `onegate vm stop`: Stops a running VM. The VM state is saved and transferred back to the front-end along with the disk files. Valid states: RUNNING
- `onegate vm suspend`: Saves a running VM. It is the same as `onegate vm stop`, but the files are left in the remote machine to later restart the VM there (i.e. the resources are not freed and there is no need to re-schedule the VM). Valid states: RUNNING
- `onegate vm delete`: Deletes the given VM. Using `--recreate` resubmits the VM. With `--hard` it unplugs the VM. Valid states: ANY
- `onegate vm shutdown`: Shuts down the given VM. The VM life cycle will end. With `--hard` it unplugs the VM. Valid states: RUNNING, UNKNOWN (with `--hard`)
- `onegate vm reboot`: Reboots the given VM, this is equivalent to execute the `reboot` command from the VM console. The VM will be ungracefully rebooted if `--hard` is used. Valid states: RUNNING
- `onegate vm poweroff`: Powers off the given VM. The VM will remain in the poweroff state, and can be powered on with the `onegate vm resume` command. Valid states: RUNNING
- `onegate vm resubmit`: Resubmits the VM to PENDING state to re-deploy a fresh copy of the same VM. Valid states: ANY
- `onegate vm resched`: Sets the rescheduling flag for the VM. The VM will be moved to a different host based on the scheduling policies. Valid states: RUNNING
- `onegate vm unresched`: Unsets the rescheduling flag for the VM. Valid states: RUNNING
- `onegate vm hold`: Sets the given VM on hold. A VM on hold is not scheduled until it is released. Valid states: PENDING
- `onegate vm release`: Releases a VM on hold. See `onegate vm hold` Valid states: HOLD

```
$ onegate vm shutdown --hard 9
```

### Change Service cardinality

The number of Virtual Machines of a Service can be also modified from any of the Virtual Machines that have access to the OneGate Server. The Virtual Machines of Services are grouped in roles and each role has a cardinality (number of Virtual Machines). This cardinality can be increased or decreased, in case the given cardinality is lower than the current one, Virtual Machines will be shut down to meet the given number. If the cardinality is greater than the current one, new Virtual Machines will be instantiated using the template associated to the role.

```
$ onegate service scale --role slave --cardinality 2
$ onegate service show
SERVICE 1
NAME      : PANACEA service
STATE     : SCALING

ROLE master
VM 8
NAME      : master_0_(service_1)
STATE     : RUNNING
IP        : 192.168.122.23

ROLE slave
VM 9
NAME      : slave_0_(service_1)
STATE     : RUNNING
VM 10
NAME      : slave_1_(service_1)
STATE     : PENDING
```

### 6.3.3 OneGate API

OneGate provides a REST API. To use this API you will need to get some data from the CONTEXT file.

The contextualization cdrom should contain the `context.sh` and `token.txt` files.

```
# mkdir /mnt/context
# mount /dev/hdb /mnt/context
# cd /mnt/context
# ls
context.sh token.txt
# cat context.sh
# Context variables generated by OpenNebula
DISK_ID='1'
ONEGATE_ENDPOINT='http://192.168.0.1:5030'
VMID='0'
TARGET='hdb'
TOKEN='yes'

# cat token.txt
yCxieDUS7kra7Vn9ILA0+g==
```

With that data, you can obtain the headers required for all the ONEGATE API methods:

- **Headers:**
  - X-ONEGATE-TOKEN: token.txt contents
  - X-ONEGATE-VMID: <vmid>

OneGate supports these actions:

## Self-awareness

- GET `${ONEGATE_ENDPOINT}/vm`: To request information about the current Virtual Machine.
- GET `${ONEGATE_ENDPOINT}/vms/${VM_ID}`: To request information about a specific Virtual Machine of the Service. The information is returned in JSON format and is ready for public cloud usage:

```
$ curl -X "GET" "${ONEGATE_ENDPOINT}/vm" \
  --header "X-ONEGATE-TOKEN: `cat token.txt`" \
  --header "X-ONEGATE-VMID: $VMID"
{
  "VM": {
    "ID": ...,
    "NAME": ...,
    "TEMPLATE": {
      "NIC": [
        {
          "IP": ...,
          "IP6_LINK": ...,
          "MAC": ...,
          "NETWORK": ...,
        },
        // more nics ...
      ]
    },
    "USER_TEMPLATE": {
      "ROLE_NAME": ...,
      "SERVICE_ID": ...,
      // more user template attributes
    }
  }
}
```

- PUT `${ONEGATE_ENDPOINT}/vm`: To add information to the template of the current VM. The new information is placed inside the VM's user template section. This means that the application metrics are visible from the command line, Sunstone, or the APIs, and can be used to trigger OneFlow elasticity rules.
- PUT `${ONEGATE_ENDPOINT}/vms/${VM_ID}`: To add information to the template of a specific VM of the Service.

```
$ curl -X "PUT" "${ONEGATE_ENDPOINT}/vm" \
  --header "X-ONEGATE-TOKEN: `cat token.txt`" \
  --header "X-ONEGATE-VMID: $VMID" \
  -d "APP_LOAD = 9.7"
```

The new metric is stored in the user template section of the VM:

```
$ onevm show 0
...
USER TEMPLATE
APP_LOAD="9.7"
```

- GET `${ONEGATE_ENDPOINT}/service`: To request information about the Service. The information is returned in JSON format and is ready for public cloud usage. By pushing data PUT `/vm` from one VM and pulling the service data from another VM GET `/service`, nodes that are part of a OneFlow service can pass values from one to another.

```
$ curl -X "GET" "${ONEGATE_ENDPOINT}/service" \
  --header "X-ONEGATE-TOKEN: `cat token.txt`" \
  --header "X-ONEGATE-VMID: $VMID"
```

```

{
  "SERVICE": {
    "id": ...,
    "name": ...,
    "roles": [
      {
        "name": ...,
        "cardinality": ...,
        "state": ...,
        "nodes": [
          {
            "deploy_id": ...,
            "running": true|false,
            "vm_info": {
              // VM template as return by GET /VM
            }
          },
          // more nodes ...
        ]
      },
      // more roles ...
    ]
  }
}

```

- GET `${ONEGATE_ENDPOINT}/service`: returns information endpoints:

```

$ curl -X "GET" "${ONEGATE_ENDPOINT}/service" \
  --header "X-ONEGATE-TOKEN: `cat token.txt`" \
  --header "X-ONEGATE-VMID: $VMID"

{
  "vm_info": "http://<onagate_endpoint>/vm",
  "service_info": "http://<onagate_endpoint>/service"
}

```

## Self-configuration

- PUT `${ONEGATE_ENDPOINT}/service/role/${ROLE_NAME}`: To change the cardinality of a specific role of the Service:

```

$ curl -X "PUT" "${ONEGATE_ENDPOINT}/service/role/worker" \
  --header "X-ONEGATE-TOKEN: `cat token.txt`" \
  --header "X-ONEGATE-VMID: $VMID" \
  -d '{"cardinality' : 10}"

```

- POST `${ONEGATE_ENDPOINT}/vms/${VM_ID}/action`: To perform an action on a specific VM of the Service. Supported actions (resume, stop, suspend, delete, shutdown, reboot, poweroff, resubmit, resched, unresched, hold, release)

```

$ curl -X "POST" "${ONEGATE_ENDPOINT}/vms/18/action" \
  --header "X-ONEGATE-TOKEN: `cat token.txt`" \
  --header "X-ONEGATE-VMID: $VMID" \
  -d '{"action' : {'perform': 'resched'}}"

```

### 6.3.4 Sample Application Monitoring Script

```
#!/bin/bash

# -----#
# Copyright 2002-2013, OpenNebula Project (OpenNebula.org), C12G Labs      #
#                                                                           #
# Licensed under the Apache License, Version 2.0 (the "License"); you may  #
# not use this file except in compliance with the License. You may obtain  #
# a copy of the License at                                               #
#                                                                           #
# http://www.apache.org/licenses/LICENSE-2.0                             #
#                                                                           #
# Unless required by applicable law or agreed to in writing, software     #
# distributed under the License is distributed on an "AS IS" BASIS,      #
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.#
# See the License for the specific language governing permissions and     #
# limitations under the License.                                         #
#-----#

#####
# Initialization
#####

ERROR=0

if [ -z $ONEGATE_TOKEN ]; then
    echo "ONEGATE_TOKEN env variable must point to the token.txt file"
    ERROR=1
fi

if [ -z $ONEGATE_ENDPOINT ]; then
    echo "ONEGATE_ENDPOINT env variable must be set"
    ERROR=1
fi

if [ $ERROR = 1 ]; then
    exit -1
fi

TMP_DIR=`mktemp -d`
echo "" > $TMP_DIR/metrics

#####
# Memory metrics
#####

MEM_TOTAL=`grep MemTotal: /proc/meminfo | awk '{print $2}'`
MEM_FREE=`grep MemFree: /proc/meminfo | awk '{print $2}'`
MEM_USED=$((MEM_TOTAL-MEM_FREE))

MEM_USED_PERC="0"

if ! [ -z $MEM_TOTAL ] && [ $MEM_TOTAL -gt 0 ]; then
    MEM_USED_PERC=`echo "$MEM_USED $MEM_TOTAL" | \
        awk '{ printf "%.2f", 100 * $1 / $2 }'`
fi
```

```

SWAP_TOTAL=`grep SwapTotal: /proc/meminfo | awk '{print $2}'`
SWAP_FREE=`grep SwapFree: /proc/meminfo | awk '{print $2}'`
SWAP_USED=$(( $SWAP_TOTAL - $SWAP_FREE ))

SWAP_USED_PERC="0"

if ! [ -z $SWAP_TOTAL ] && [ $SWAP_TOTAL -gt 0 ]; then
    SWAP_USED_PERC=`echo "$SWAP_USED $SWAP_TOTAL" | \
        awk '{ printf "%.2f", 100 * $1 / $2 }'`
fi

#echo "MEM_TOTAL = $MEM_TOTAL" >> $TMP_DIR/metrics
#echo "MEM_FREE = $MEM_FREE" >> $TMP_DIR/metrics
#echo "MEM_USED = $MEM_USED" >> $TMP_DIR/metrics
echo "MEM_USED_PERC = $MEM_USED_PERC" >> $TMP_DIR/metrics

#echo "SWAP_TOTAL = $SWAP_TOTAL" >> $TMP_DIR/metrics
#echo "SWAP_FREE = $SWAP_FREE" >> $TMP_DIR/metrics
#echo "SWAP_USED = $SWAP_USED" >> $TMP_DIR/metrics
echo "SWAP_USED_PERC = $SWAP_USED_PERC" >> $TMP_DIR/metrics

#####
# Disk metrics
#####

/bin/df -k -P | grep '^/dev' > $TMP_DIR/df

cat $TMP_DIR/df | while read line; do
    NAME=`echo $line | awk '{print $1}' | awk -F '/' '{print $NF}'`

    DISK_TOTAL=`echo $line | awk '{print $2}'`
    DISK_USED=`echo $line | awk '{print $3}'`
    DISK_FREE=`echo $line | awk '{print $4}'`

    DISK_USED_PERC="0"

    if ! [ -z $DISK_TOTAL ] && [ $DISK_TOTAL -gt 0 ]; then
        DISK_USED_PERC=`echo "$DISK_USED $DISK_TOTAL" | \
            awk '{ printf "%.2f", 100 * $1 / $2 }'`
    fi

    #echo "DISK_TOTAL_$NAME = $DISK_TOTAL" >> $TMP_DIR/metrics
    #echo "DISK_FREE_$NAME = $DISK_FREE" >> $TMP_DIR/metrics
    #echo "DISK_USED_$NAME = $DISK_USED" >> $TMP_DIR/metrics
    echo "DISK_USED_PERC_$NAME = $DISK_USED_PERC" >> $TMP_DIR/metrics
done

#####
# PUT command
#####

VMID=(source /mnt/context.sh; echo $VMID)

curl -X "PUT" $ONEGATE_ENDPOINT/vm \
    --header "X-ONEGATE-TOKEN: `cat $ONEGATE_TOKEN`" \
    --header "X-ONEGATE-VMID: $VMID" \
    --data-binary @$TMP_DIR/metrics

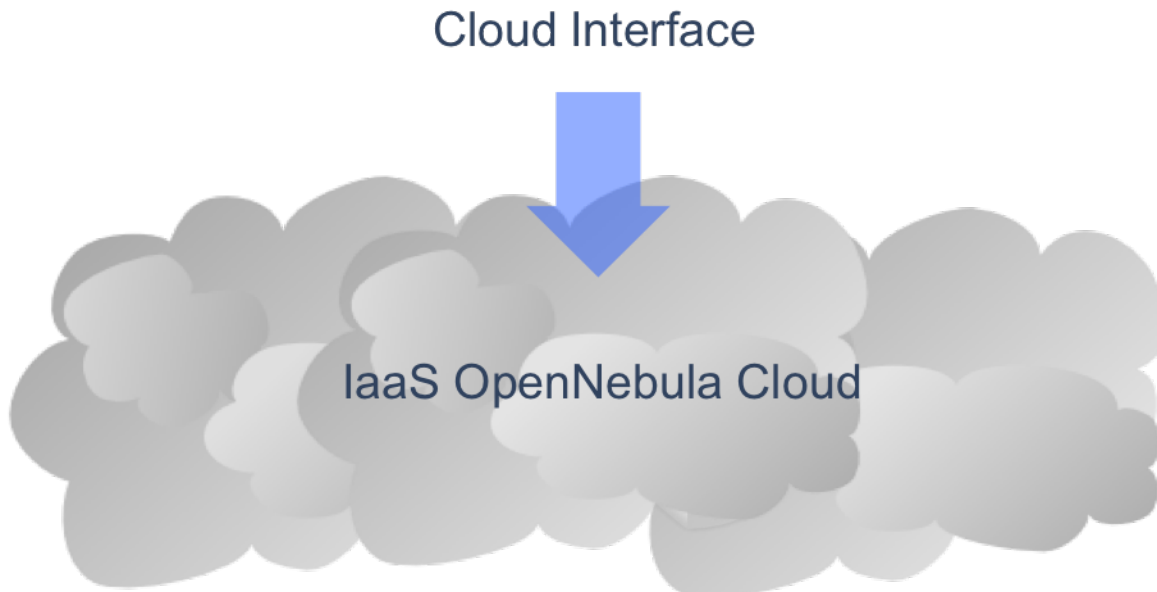
```



## PUBLIC CLOUD

### 7.1 Building a Public Cloud

#### 7.1.1 What is a Public Cloud?



A Public Cloud is an **extension of a Private Cloud to expose RESTful Cloud interfaces**. Cloud interfaces can be added to your Private or Hybrid Cloud if you want to provide partners or external users with access to your infrastructure, or to sell your overcapacity. Obviously, a local cloud solution is the natural back-end for any public cloud.

#### 7.1.2 The User View

The following interfaces provide a **simple and remote management of cloud (virtual) resources at a high abstraction level**:

- *EC2 Query subset*

Users will be able to use commands that **clone the functionality of the EC2 Cloud service**. Starting with a working installation of an OS residing on an **.img** file, with three simple steps a user can launch it in the cloud.

First, they will be able to **upload** it to the cloud using:

```
$ ./econe-upload /images/gentoo.img
Success: ImageId ami-00000001
```

After the image is uploaded in OpenNebula repository, it needs to be **registered** to be used in the cloud:

```
$ ./econe-register ami-00000001
Success: ImageId ami-00000001
```

Now the user can **launch** the registered image to be run in the cloud:

```
$ ./econe-run-instances -H ami-00000001
Owner      ImageId      InstanceId InstanceType
-----
helen      ami-00000001      i-15      m1.small
```

Additionally, the instance can be **monitored** with:

```
$ ./econe-describe-instances -H
Owner      Id      ImageId      State      IP      Type
-----
helen      i-15      ami-00000001 pending      147.96.80.33      m1.small
```

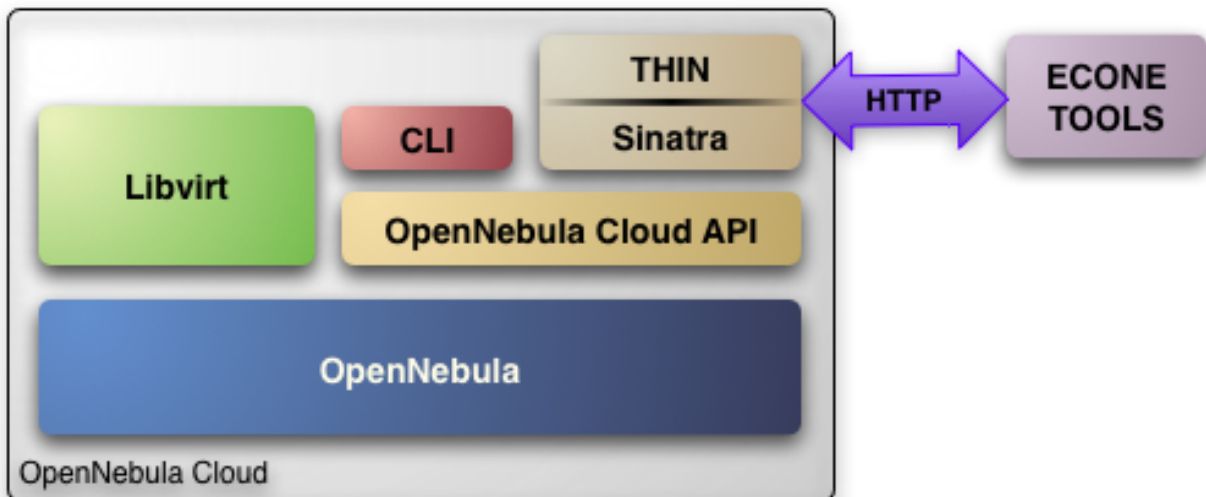
### 7.1.3 How the System Operates

There is **no modification in the operation of OpenNebula to expose Cloud interfaces**. Users can interface the infrastructure using any Private or Public Cloud interface.

## 7.2 EC2 Server Configuration

### 7.2.1 Overview

The OpenNebula EC2 Query is a web service that enables you to launch and manage virtual machines in your OpenNebula installation through the [Amazon EC2 Query Interface](#). In this way, you can use any EC2 Query tool or utility to access your Private Cloud. The EC2 Query web service is implemented upon the **OpenNebula Cloud API (OCA)** layer that exposes the full capabilities of an OpenNebula private cloud; and [Sinatra](#), a widely used light web framework.



The current implementation includes the basic routines to use a Cloud, namely: image upload and registration, and the VM run, describe and terminate operations. The following sections explain you how to install and configure the EC2 Query web service on top of a running OpenNebula cloud.

**Warning:** The OpenNebula EC2 Query service provides a Amazon EC2 Query API compatible interface to your cloud, that can be used alongside the native OpenNebula CLI or OpenNebula Sunstone.

**Warning:** The OpenNebula distribution includes the tools needed to use the EC2 Query service.

## 7.2.2 Requirements & Installation

You must have an OpenNebula site properly configured and running , be sure to check the OpenNebula Installation and Configuration Guides to set up your private cloud first. This guide also assumes that you are familiar with the configuration and use of OpenNebula.

The OpenNebula EC2 Query service was installed during the OpenNebula installation, and the dependencies of this service are installed when using the `install_gems` tool as explained in the installation guide

If you installed OpenNebula from source you can install the EC2 Query dependencies as explained at the end of the Building from Source Code guide

## 7.2.3 Configuration

The service is configured through the `/etc/one/econe.conf` file, where you can set up the basic operational parameters for the EC2 Query web service. The following table summarizes the available options:

### Server configuration

- `tmpdir`: Directory to store temp files when uploading images
- `one_xmlrpc`: oned xmlrpc service, `http://localhost:2633/RPC2`
- `host`: Host where econe server will run
- `port`: Port where econe server will run
- `ssl_server`: URL for the EC2 service endpoint, when configured through a proxy

### Log

- `debug_level`: Log debug level, 0 = ERROR, 1 = WARNING, 2 = INFO, 3 = DEBUG.

### Auth

- `auth`: Authentication driver for incoming requests
- `core_auth`: Authentication driver to communicate with OpenNebula core. Check this guide for more information about the `core_auth` system

### File based templates

- `use_file_templates`: Use former file based templates for instance types instead of OpenNebula templates
- `instance_types`: DEPRECATED The VM types for your cloud

### Resources

- `describe_with_terminated_instances`: Include terminated instances in the `describe_instances` xml. When this parameter is enabled all the VMs in DONE state will be retrieved in each `describe_instances` action and then filtered. This can cause performance issues when the pool of VMs in DONE state is huge

- `terminated_instances_expiration_time`: Terminated VMs will be included in the list till the termination date + `terminated_instances_expiration_time` is reached
- `datastore_id`: Datastore in which the Images uploaded through EC2 will be allocated, by default 1
- `cluster_id`: Cluster associated with the EC2 resources, by default no Cluster is defined

### Elastic IP

- `elasticips_vnet_id`: VirtualNetwork containing the elastic ips to be used with EC2. If no defined the Elastic IP functionality is disabled
- `associate_script`: Script to associate a public IP with a private IP arguments: `elastic_ip private_ip vnet_template(base64_encoded)`
- `disassociate_script`: Script to disassociate a public IP arguments: `elastic_ip`

### EBS

- `ebsfstype`: FSTYPE that will be used when creating new volumes (DATABLOCKS)

**Warning:** The `:host` must be a FQDN, do not use IP's here.

**Warning:** Preserve YAML syntax in the `econe.conf` file.

## Cloud Users

The cloud users have to be created in the OpenNebula system by `oneadmin` using the `oneuser` utility. Once a user is registered in the system, using the same procedure as to create private cloud users, they can start using the system.

The users will authenticate using the [Amazon EC2 procedure](#) with `AWSAccessKeyId` their OpenNebula's username and `AWSSecretAccessKey` their OpenNebula's hashed password.

The cloud administrator can limit the interfaces that these users can use to interact with OpenNebula by setting the driver `public` for them. Using that driver cloud users will not be able to interact with OpenNebula through Sunstone, CLI nor XML-RPC.

```
$ oneuser chauth cloud_user public
```

## Defining VM Types

You can define as many Virtual Machine types as you want, just:

- Create a new OpenNebula template for the new type and make it available for the users group. You can use restricted attributes and set permissions like any other opennebula resource. **You must include the `EC2_INSTANCE_TYPE` parameter inside the template definition**, otherwise the template will not be available to be used as an instance type in EC2.

```
# This is the content of the /tmp/m1.small file
NAME = "m1.small"
EC2_INSTANCE_TYPE = "m1.small"
CPU = 1
MEMORY = 1700
...
```

```
$ onetemplate create /tmp/m1.small
$ onetemplate chgrp m1.small users
$ onetemplate chmod m1.small 640
```

The template must include all the required information to instantiate a new virtual machine, such as network configuration, capacity, placement requirements, etc. This information will be used as a base template and will be merged with the information provided by the user.

The user will select an instance type along with the ami id, keypair and user data when creating a new instance. Therefore, **the template should not include the OS**, since it will be specified by the user with the selected AMI.

**Warning:** The templates are processed by the EC2 server to include specific data for the instance.

## 7.2.4 Starting the Cloud Service

To start the EC2 Query service just issue the following command

```
$ econe-server start
```

You can find the econe server log file in `/var/log/one/econe-server.log`.

To stop the EC2 Query service:

```
$ econe-server stop
```

## 7.2.5 Advanced Configuration

### Enabling Keypair

In order to benefit from the Keypair functionality, the images that will be used by the econe users must be prepared to read the `EC2_PUBLIC_KEY` and `EC2_USER_DATA` from the `CONTEXT` disk. This can be easily achieved with the new [contextualization packages](#), generating a new custom contextualization package like this one:

```
#!/bin/bash
echo "$EC2_PUBLIC_KEY" > /root/.ssh/authorized_keys
```

### Enabling Elastic IP Functionality

An Elastic IP address is associated with the user, not a particular instance, and the user controls that address until he chooses to release it. This way the user can programmatically remap his public IP addresses to any of his instances.

In order to enable this functionality you have to follow the following steps:

#### 1. Create a VNET Containing the Elastic IPS

- As oneadmin create a new VirtualNetwork containing the public IPs that will be controlled by the EC2 users. Each IP **must be placed in its own AR**:

```
NAME      = "ElasticIPs"
PHYDEV    = "eth0"
VLAN      = "YES"
VLAN_ID   = 50
BRIDGE    = "brhm"
```

```
AR = [IP=10.0.0.1, TYPE=IP4, SIZE=1]
AR = [IP=10.0.0.2, TYPE=IP4, SIZE=1]
AR = [IP=10.0.0.3, TYPE=IP4, SIZE=1]
AR = [IP=10.0.0.4, TYPE=IP4, SIZE=1]

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

```
$ onevnet create /tmp/fixed.vnet
ID: 8
```

This VNET will be managed by the oneadmin user, therefore USE permission for the ec2 users is not required

- Update the econe.conf file with the VNET ID:

```
:elastic_ips_vnet: 8
```

- Provide associate and disassociate scripts

The interaction with the infrastructure has been abstracted, therefore two scripts have to be provided by the cloud administrator in order to interact with each specific network configuration. This two scripts enable us to adapt this feature to different configurations and data centers.

These scripts are language agnostic and their path has to be specified in the econe configuration file:

```
:associate_script: /usr/bin/associate_ip.sh
:disassociate_script: /usr/bin/disassociate_ip.sh
```

The associate script will receive three arguments: **elastic\_ip** to be associated; **private\_ip** of the instance; **Virtual Network template** base64 encoded

The disassociate script will receive three arguments: **elastic\_ip** to be disassociated

Scripts to interact with OpenFlow can be found in the following [ecosystem project](#)

### Using a Specific Group for EC2

It is recommended to create a new group to handle the ec2 cloud users:

```
$ onegroup create ec2
ID: 100
```

Create and add the users to the ec2 group (ID:100):

```
$ oneuser create clouduser my_password
ID: 12
$ oneuser chgrp 12 100
```

Also, you will have to create ACL rules so that the cloud users are able to deploy their VMs in the allowed hosts.

```
$ onehost list
ID NAME          CLUSTER  RVM    ALLOCATED_CPU    ALLOCATED_MEM    STAT
  1 kvm1          -         2     110 / 200 (55%)  640M / 3.6G (17%)  on
  1 kvm2          -         2     110 / 200 (55%)  640M / 3.6G (17%)  on
  1 kvm3          -         2     110 / 200 (55%)  640M / 3.6G (17%)  on
```

These rules will allow users inside the ec2 group (ID:100) to deploy VMs in the hosts kvm01 (ID:0) and kvm03 (ID:3)

```
$ oneacl create "@100 HOST/#1 MANAGE"
$ oneacl create "@100 HOST/#3 MANAGE"
```

You **have to create a VNet network** using the `onevnet` utility with the IP's you want to lease to the VMs created with the EC2 Query service.

```
$ onevnet create /tmp/templates/vnet
ID: 12
```

Remember that you will have to add this VNet (ID:12) to the users group (ID:100) and give USE (640) permissions to the group in order to get leases from it.

```
$ onevnet chgrp 12 100
$ onevnet chmod 12 640
```

**Warning:** You will have to update the NIC template, inside the `/etc/one/ec2query_templates` directory, in order to use this VNet ID

## Configuring a SSL Proxy

OpenNebula EC2 Query Service runs natively just on normal HTTP connections. If the extra security provided by SSL is needed, a proxy can be set up to handle the SSL connection that forwards the petition to the EC2 Query Service and takes back the answer to the client.

This set up needs:

- A server certificate for the SSL connections
- An HTTP proxy that understands SSL
- EC2Query Service configuration to accept petitions from the proxy

If you want to try out the SSL setup easily, you can find in the following lines an example to set a self-signed certificate to be used by a `lighttpd` configured to act as an HTTP proxy to a correctly configured EC2 Query Service.

Let's assume the server where the `lighttpd` proxy is going to be started is called `cloudserver.org`. Therefore, the steps are:

### 1. Snakeoil Server Certificate

We are going to generate a snakeoil certificate. If using an Ubuntu system follow the next steps (otherwise your mileage may vary, but not a lot):

- Install the `ssl-cert` package

```
$ sudo apt-get install ssl-cert
```

- Generate the certificate

```
$ sudo /usr/sbin/make-ssl-cert generate-default-snakeoil
```

- As we are using `lighttpd`, we need to append the private key with the certificate to obtain a server certificate valid to `lighttpd`

```
$ sudo cat /etc/ssl/private/ssl-cert-snakeoil.key /etc/ssl/certs/ssl-cert-snakeoil.pem > /etc/lighttpd
```

### 2. lighttpd as a SSL HTTP Proxy

You will need to edit the `/etc/lighttpd/lighttpd.conf` configuration file and

- Add the following modules (if not present already)
  - mod\_access
  - mod\_alias
  - mod\_proxy
  - mod\_accesslog
  - mod\_compress
- Change the server port to 443 if you are going to run lighttpd as root, or any number above 1024 otherwise:

```
server.port = 8443
```

- Add the proxy module section:

```
#### proxy module
## read proxy.txt for more info
proxy.server = ( " " =>
                ( " " =>
                  (
                    "host" => "127.0.0.1",
                    "port" => 4567
                  )
                )
              )

#### SSL engine
ssl.engine = "enable"
ssl.pemfile = "/etc/lighttpd/server.pem"
```

The host must be the server hostname of the computer running the EC2Query Service, and the port the one that the EC2Query Service is running on.

### 3. EC2Query Service Configuration

The `econe.conf` needs to define the following:

```
# Host and port where econe server will run
:host: localhost
:port: 4567

#SSL proxy URL that serves the API (set if is being used)
:ssl_server: https://cloudserver.org:8443/
```

Once the lighttpd server is started, EC2Query petitions using HTTPS uris can be directed to `https://cloudserver.org:8443`, that will then be unencrypted, passed to localhost, port 4567, satisfied (hopefully), encrypted again and then passed back to the client.

**Warning:** Note that `:ssl_server` **must** be an URL that may contain a custom path.



## 7.3 OpenNebula EC2 User Guide

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

OpenNebula implements a subset of the EC2 Query interface, enabling the creation of public clouds managed by OpenNebula.

### 7.3.1 AMIs

- **upload image:** Uploads an image to OpenNebula
- **register image:** Register an image into OpenNebula
- **describe images:** Lists all registered images belonging to one particular user.

### 7.3.2 Instances

- **run instances:** Runs an instance of a particular image (that needs to be referenced).
- **describe instances:** Outputs a list of launched images belonging to one particular user.
- **terminate instances:** Shutdowns a set of virtual machines (or cancel, depending on its state).
- **reboot instances:** Reboots a set of virtual machines.
- **start instances:** Starts a set of virtual machines.
- **stop instances:** Stops a set of virtual machines.

### 7.3.3 EBS

- **create volume:** Creates a new DATABLOCK in OpenNebula
- **delete volume:** Deletes an existing DATABLOCK.
- **describe volumes:** Describe all available DATABLOCKS for this user
- **attach volume:** Attaches a DATABLOCK to an instance
- **detach volume:** Detaches a DATABLOCK from an instance
- **create snapshot:**
- **delete snapshot:**
- **describe snapshot:**

### 7.3.4 Elastic IPs

- **allocate address:** Allocates a new elastic IP address for the user
- **release address:** Releases a publicIP of the user
- **describe addresses:** Lists elastic IP addresses
- **associate address:** Associates a publicIP of the user with a given instance

- **disassociate address**: Disassociate a publicIP of the user currently associated with an instance

### 7.3.5 Keypairs

- **create keypair**: Creates the named keypair
- **delete keypair**: Deletes the named keypair, removes the associated keys
- **describe keypairs**: List and describe the key pairs available to the user

### 7.3.6 Tags

- **create-tags**
- **describe-tags**
- **remove-tags**

Commands description can be accessed from the Command Line Reference.

## User Account Configuration

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

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

- **Authentication**: This can be achieved in three different ways, here listed in order of priority (i.e. values specified in the argument line supersede environmental variables)
  - Using the **commands arguments**. All the commands accept an **Access Key** (as the OpenNebula username) and a **Secret Key** (as the OpenNebula hashed password)
  - Using **EC2\_ACCESS\_KEY** and **EC2\_SECRET\_KEY** environment variables the same way as the arguments
  - If none of the above is available, the **ONE\_AUTH** variable will be checked for authentication (with the same used for OpenNebula CLI).
- **Server location**: The command need to know where the OpenNebula cloud service is running. That information needs to be stored within the **EC2\_URL** environment variable (in the form of a http URL, including the port if it is not the standard 80).

<b>Warning:</b> The <b>EC2_URL</b> has to use the FQDN of the EC2-Query Server
--

## Hello Cloud!

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

- **upload\_image**

Assuming we have a working Gentoo installation residing in an **.img** file, we can upload it into the OpenNebula cloud using the **econe-upload** command:

```
$ econe-upload /images/gentoo.img
Success: ImageId ami-00000001
$ econe-register ami-00000001
Success: ImageId ami-00000001
```

- **describe\_images**

We will need the **ImageId** to launch the image, so in case we forgotten we can list registered images using the **econe-describe-images** command:

```
$ econe-describe-images -H
Owner      ImageId      Status      Visibility  Location
-----
helen      ami-00000001 available    private     19ead5de585f43282acab4060bfb7a07
```

- **run\_instance**

Once we recall the **ImageId**, we will need to use the **econe-run-instances** command to launch an Virtual Machine instance of our image:

```
$ econe-run-instances -H ami-00000001
Owner      ImageId      InstanceId InstanceType
-----
helen      ami-00000001      i-15      m1.small
```

We will need the **InstanceId** to monitor and shutdown our instance, so we better write down that **i-15**.

- **describe\_instances**

If we have too many instances launched and we don't remember everyone of them, we can ask **econe-describe-instances** to show us which instances we have submitted.

```
$ econe-describe-instances -H
Owner      Id      ImageId      State      IP      Type
-----
helen      i-15    ami-00000001 pending     147.96.80.33    m1.small
```

We can see that the instances with Id **i-15** has been launched, but it is still pending, i.e., it still needs to be deployed into a physical host. If we try the same command again after a short while, we should be seeing it running as in the following excerpt:

```
$ econe-describe-instances -H
Owner      Id      ImageId      State      IP      Type
-----
helen      i-15    ami-00000001 running     147.96.80.33    m1.small
```

- **terminate\_instances**

After we put the Virtual Machine to a good use, it is time to shut it down to make space for other Virtual Machines (and, presumably, to stop being billed for it). For that we can use the **econe-terminate-instances** passing to it as an argument the **InstanceId** that identifies our Virtual Machine:

```
$ econe-terminate-instances i-15
Success: Terminating i-15 in running state
```

**Warning:** You can obtain more information on how to use the above commands accessing their Usage help passing them the **-h** flag

## 7.4 EC2 Ecosystem

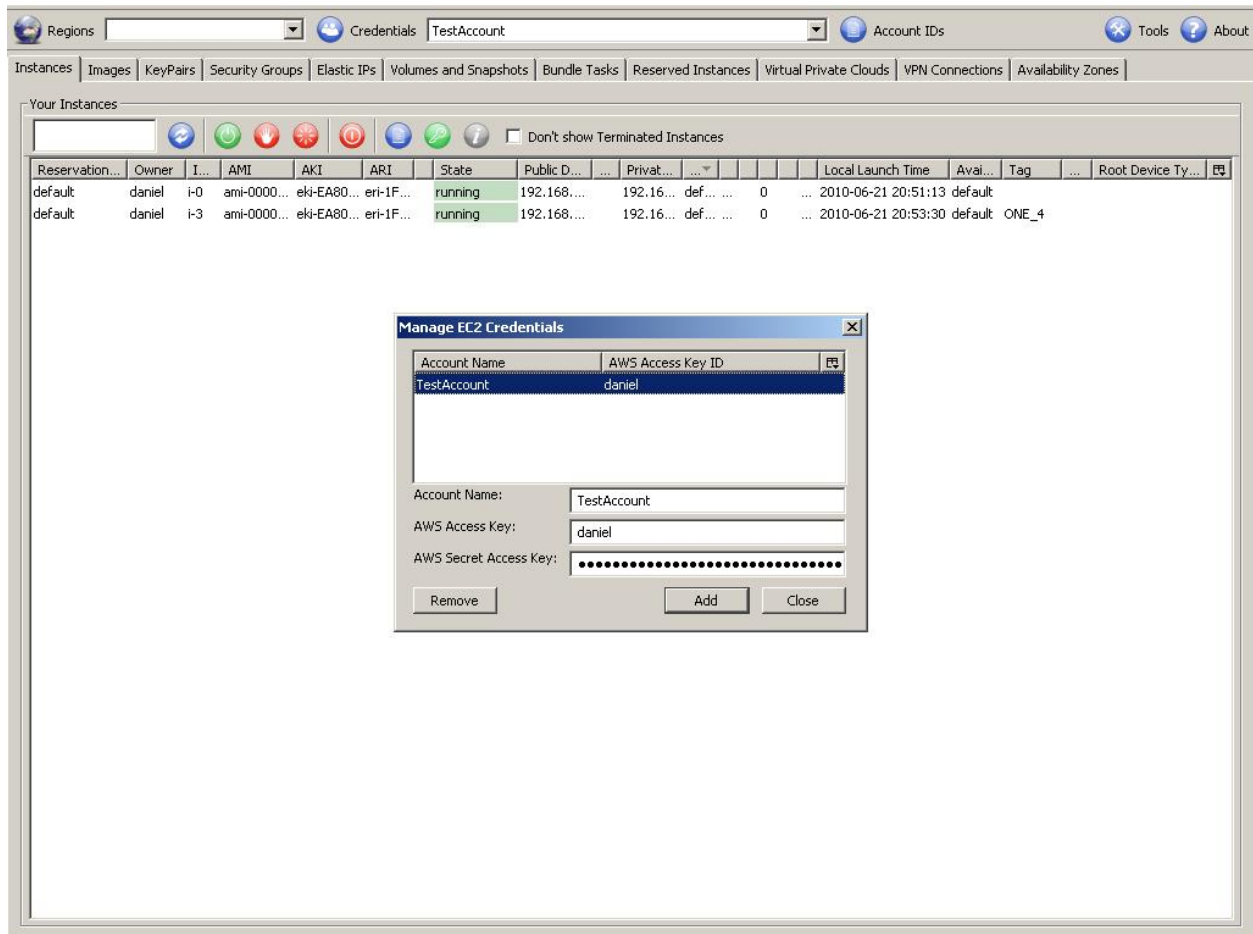
In order to interact with the EC2 Service that OpenNebula implements you can use the client included in the OpenNebula distribution, but also you can choose one of the well known tools that are supposed to interact with cloud servers through the EC2 Query API, like the Firefox extension [HybridFox](#), or the command line tools, [Euca2ools](#).

### 7.4.1 HybridFox

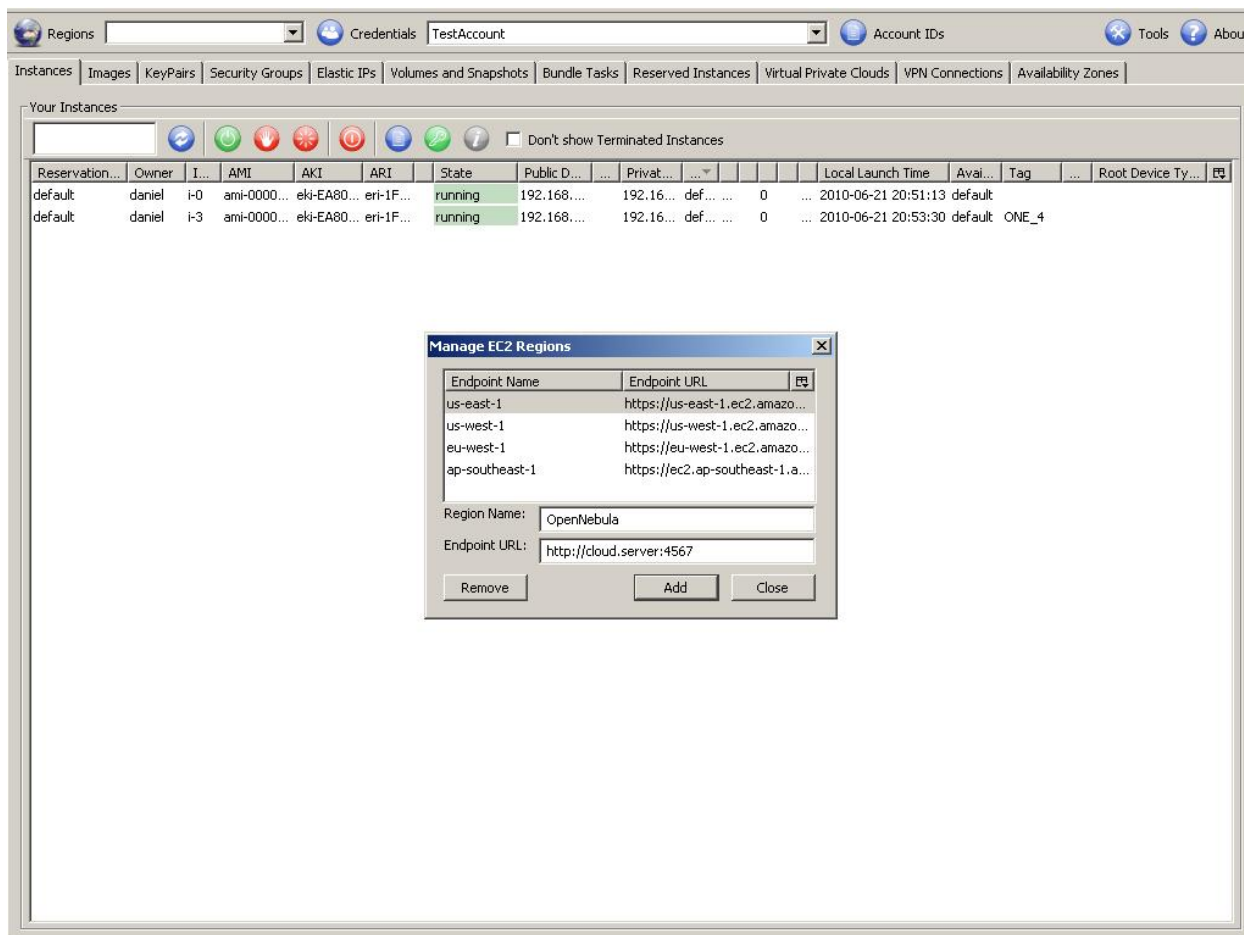
[HybridFox](#) is a Mozilla Firefox extension for managing your Amazon EC2 account. Launch new instances, mount Elastic Block Storage volumes, map Elastic IP addresses, and more.

#### Configuration

- You have to set up the credentials to interact with OpenNebula, by pressing the `Credentials` button:
  1. Account Name, add a name for this account
  2. AWS Access Key, add your OpenNebula username
  3. AWS Secret Access Key, add your OpenNebula SHA1 hashed password



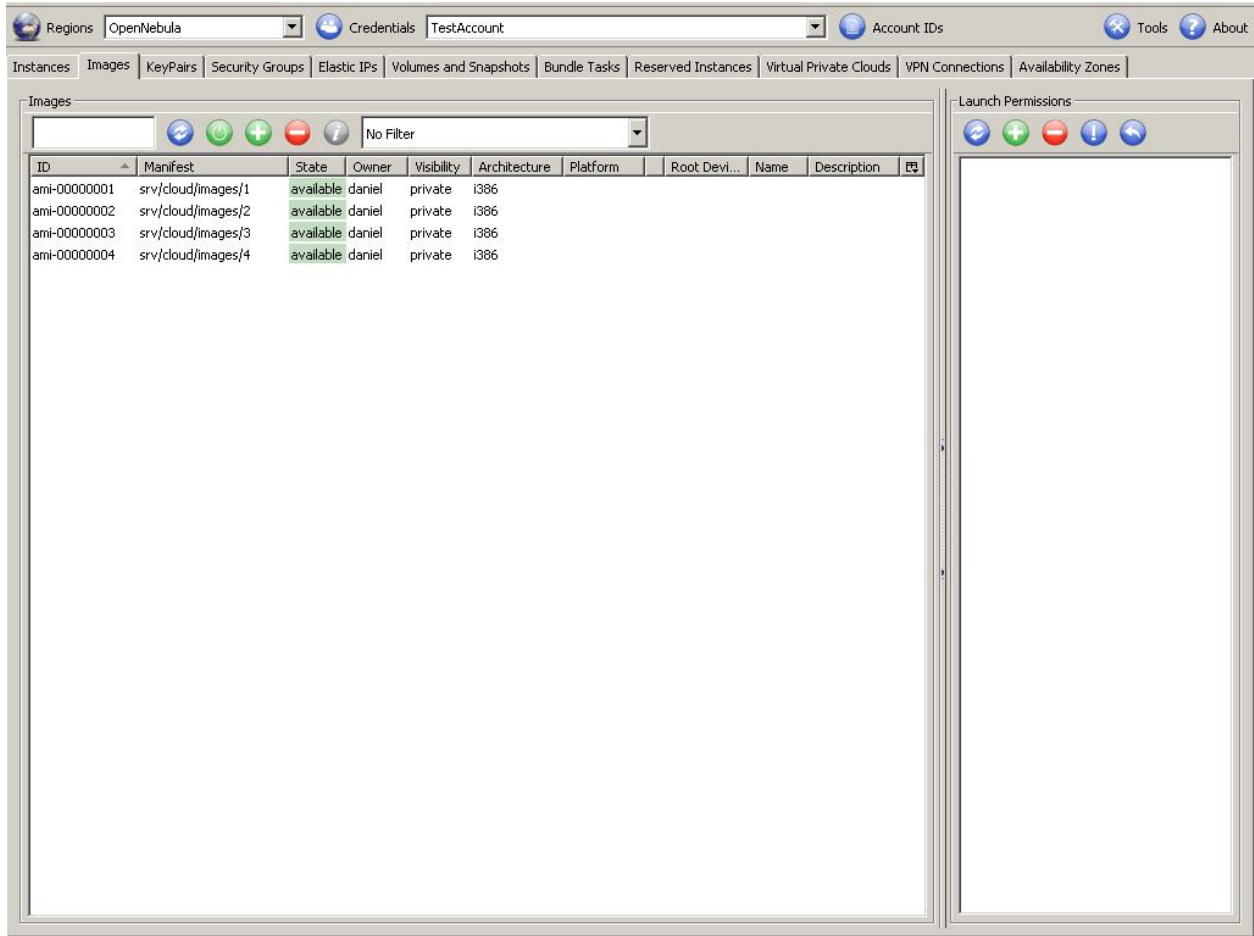
- Also you have to specify in a new Region the endpoint in which the EC2 Service is running, by pressing on the Regions button. Take care of using exactly the same url and port that is specified in the econe.conf file, otherwise you will get AuthFailure:



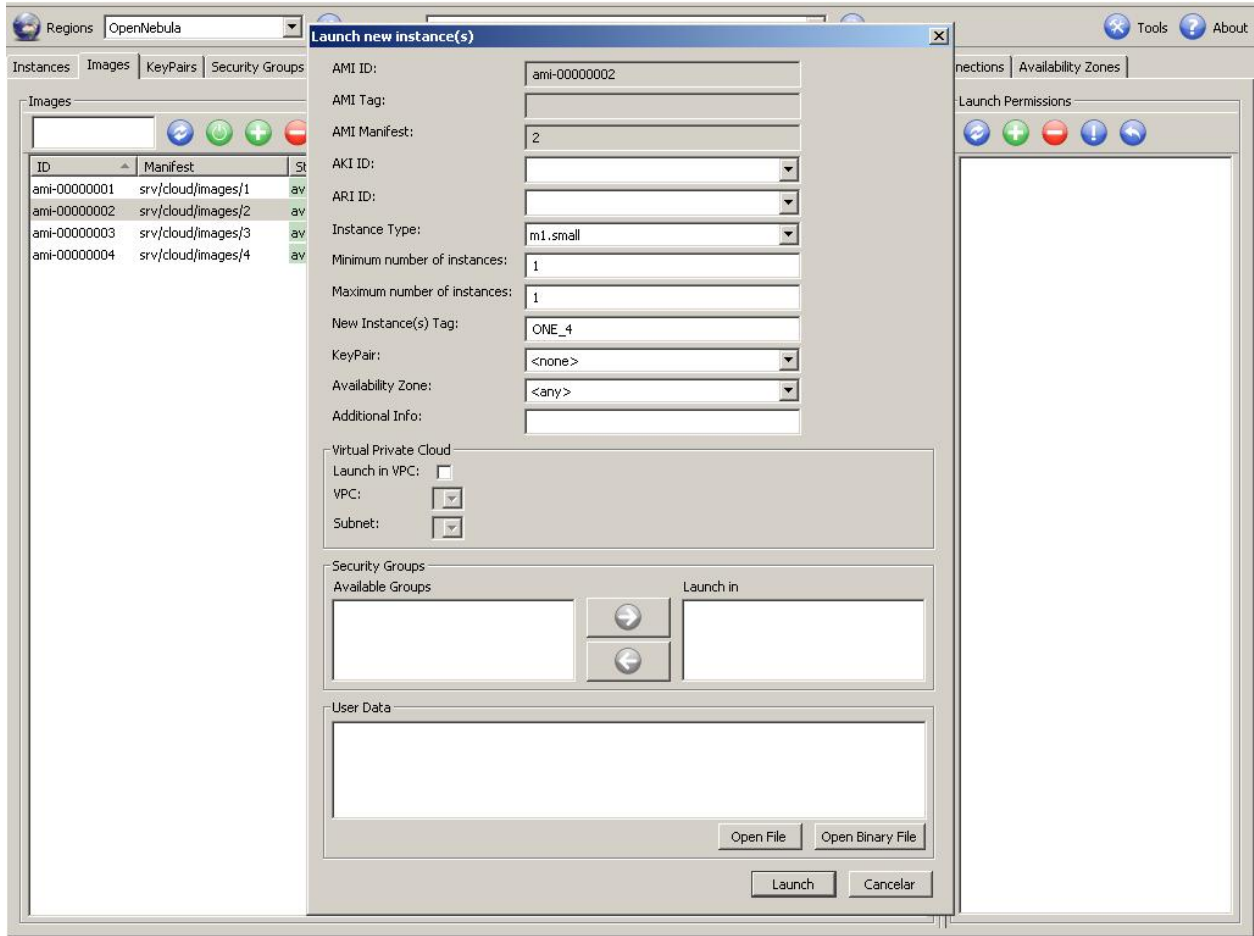
**Warning:** If you have problems adding a new region, try to add it manually in the ec2ui.endpoints variable inside the Firefox about:config

### Typical usage scenarios

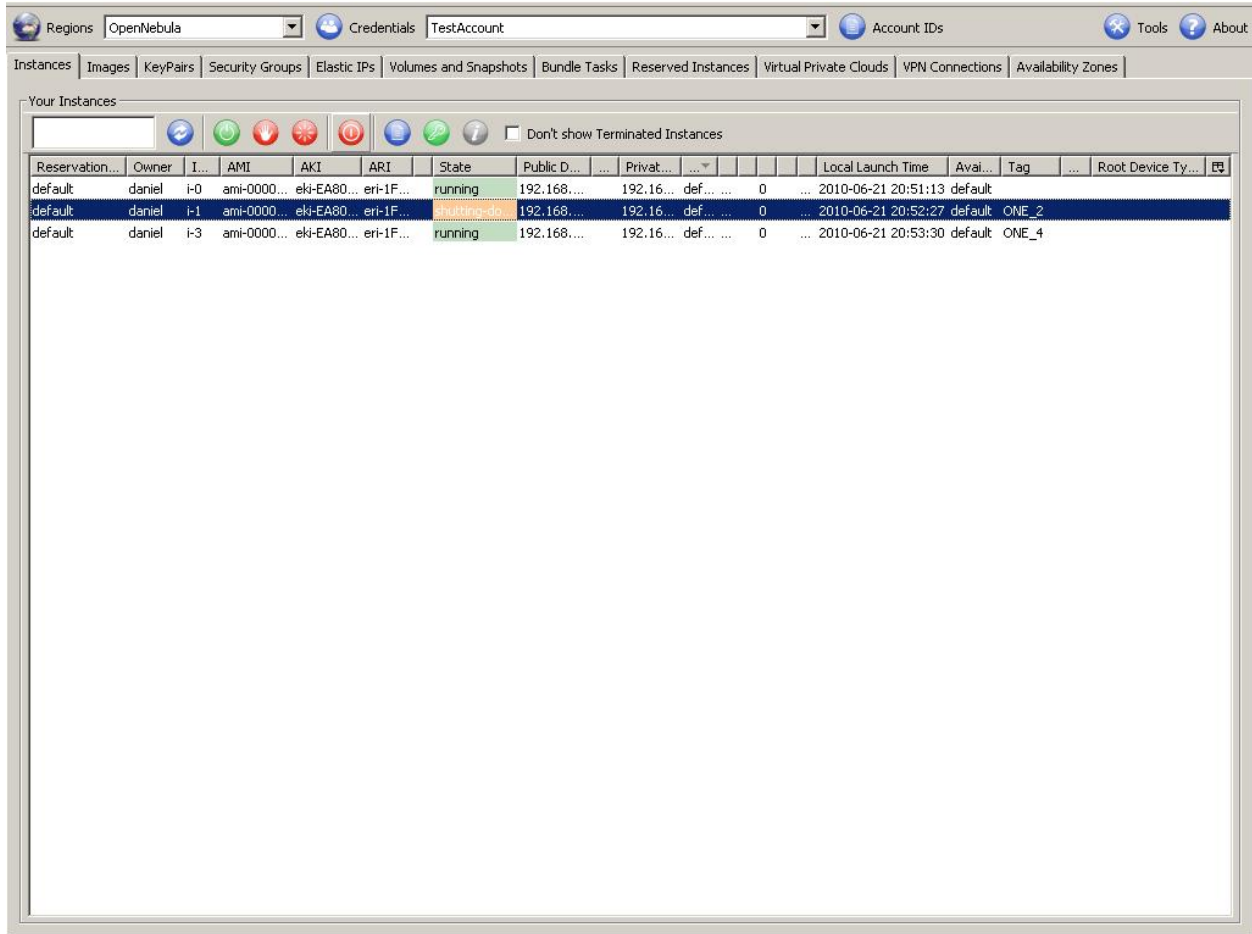
- List images



- **Run instances**



- Control instances



You can also use [HybridFox](#) a similar Mozilla Firefox extension to interact with cloud services through the EC2 Query API

## 7.4.2 Euca2ools

[Euca2ools](#) are command-line tools for interacting with Web services that export a REST/Query-based API compatible with Amazon EC2 and S3 services.

You have to set the following environment variables in order to interact with the OpenNebula EC2 Query Server. The `EC2_URL` will be the same endpoint as defined in the `/etc/one/econe.conf` file of Opennebula. The `EC2_ACCESS_KEY` will be the OpenNebula username and the `EC2_SECRET_KEY` the OpenNebula sha1 hashed user password

```
~$ env | grep EC2
EC2_SECRET_KEY=e17a13.0834936f71bb3242772d25150d40791e72
EC2_URL=http://localhost:4567
EC2_ACCESS_KEY=oneadmin
```



## Typical usage scenarios

### • List images

```

~$ euca-describe-images
IMAGE    ami-00000001    srv/cloud/images/1    daniel    available    private    i386    machine
IMAGE    ami-00000002    srv/cloud/images/2    daniel    available    private    i386    machine
IMAGE    ami-00000003    srv/cloud/images/3    daniel    available    private    i386    machine
IMAGE    ami-00000004    srv/cloud/images/4    daniel    available    private    i386    machine

```

### • List instances

```

~$ euca-describe-instances
RESERVATION default daniel default
INSTANCE    i-0    ami-00000002    192.168.0.1    192.168.0.1    running    default    0    m1.small    2010-
INSTANCE    i-3    ami-00000002    192.168.0.4    192.168.0.4    running    default    0    m1.small    2010-

```

### • Run instances

```

~$ euca-run-instances --instance-type m1.small ami-00000001
RESERVATION r-47a5402e daniel default
INSTANCE    i-4    ami-00000001    192.168.0.2    192.168.0.2    pending    default    2010-06-22T11:54:07+02:00    M

```