OpenNebula.org

OpenNebula 4.4 Advanced Administration Guide

OpenNebula Project

December 17, 2013

CONTENTS

CHAPTER

ONE

APPLICATION FLOW 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

Starting with OpenNebula 4.2, OneFlow is included in the default installation. Check the *Installation guide* for details of what package you have to install depending on your distribution

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
Server Configuration	
:one_xmlrpc	OpenNebula daemon host and port
:lcm_interval	Time in seconds between Life Cycle Manager steps
:host	Host where OneFlow will listen
:port	Port where OneFlow will listen
Defaults	
:default_cooldown	Default cooldown period after a scale operation, in seconds
:shutdown_action	Default shutdown action. Values: 'shutdown', 'shutdown-hard'
:action_number :action_period	Default number of virtual machines (action_number) that will receive the given call in each interval defined by action_period, when an action is performed on a role.
Auth	
: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
Log	
:debug_level	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

################################################################
# 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 hidden by default. To enable them, edit '/etc/one/sunstone-views/admin.yaml' and '/etc/one/sunstone-views/user.yaml' and set oneflow tabs inside 'enabled_tabs' to true:

```
enabled_tabs:
    dashboard-tab: true

    ...

    oneflow-dashboard: true
    oneflow-services: true
    oneflow-templates: true
```

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

ACL Rule

By default this rule is defined in OpenNebula to enable the creation of new services by any user. If you want to limit this, you will have to delete this rule and generate new ones.

```
* DOCUMENT/* CREATE
```

If you only want a specific group to be able to use OneFlow, execute:

```
$ oneacl create "@1 DOCUMENT/* CREATE"
```

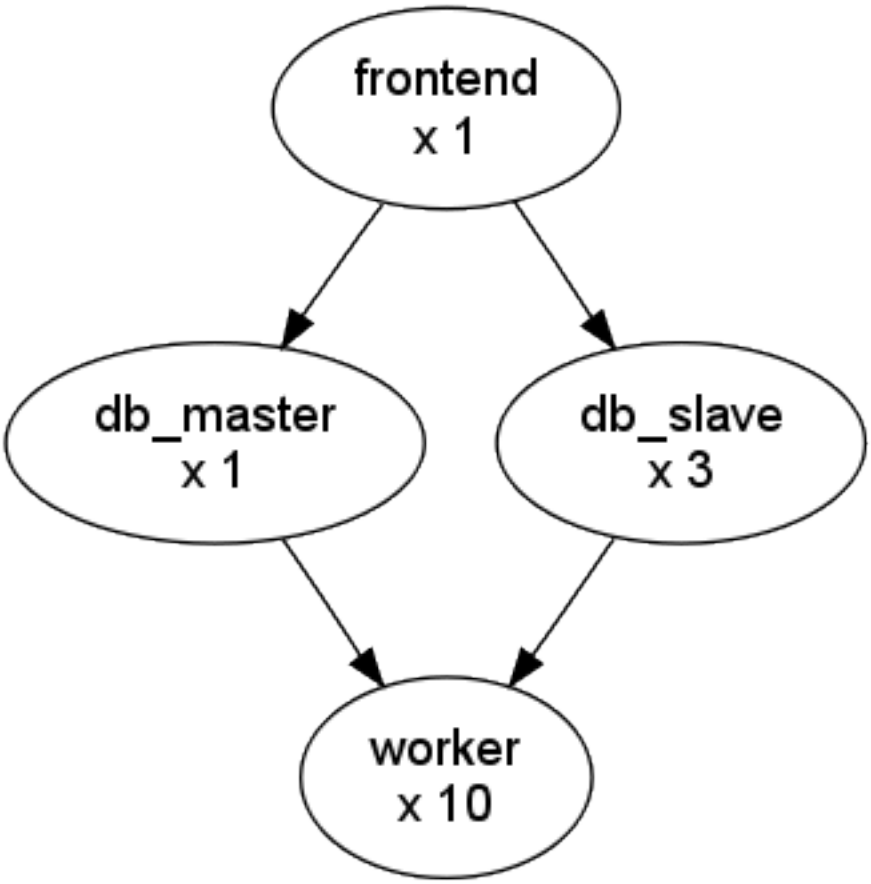
Read more about the *ACL Rules system here*.

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",
  "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.

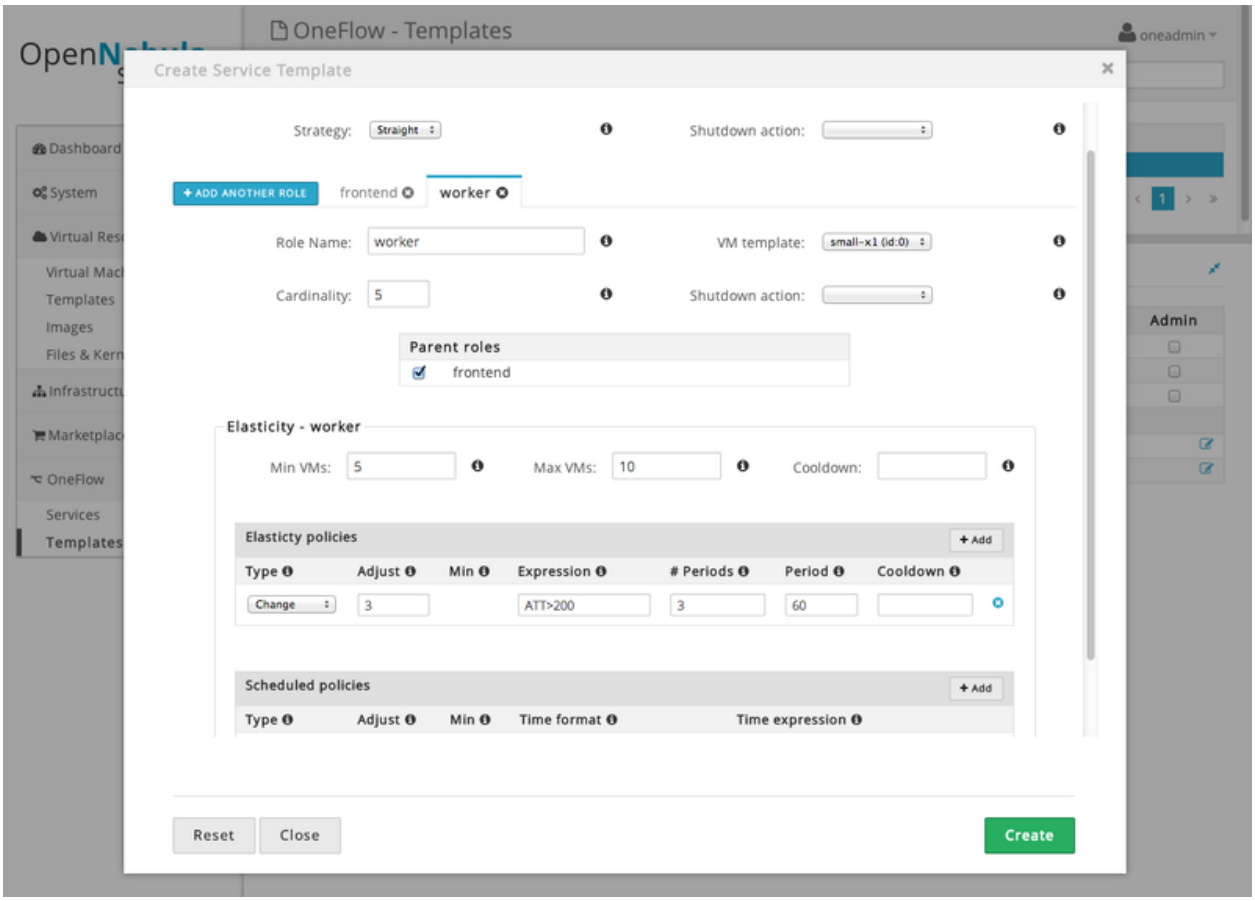
Users can manage the Service Templates using the command `oneflow-template`, or the graphical interface. For each user, the actual list of Service Templates available is determined by the ownership and permissions of the Templates.

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
```

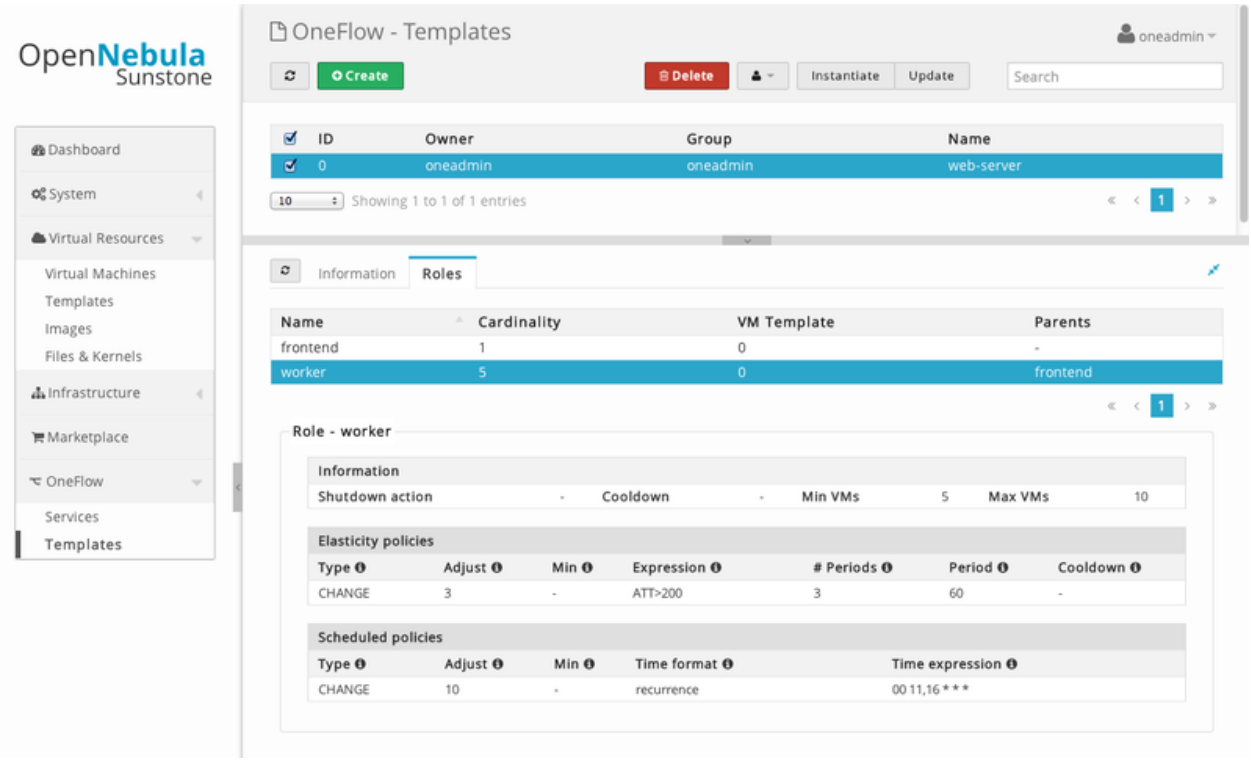
You can also create service template from Sunstone:



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

```
$ oneflow-template list
        ID USER            GROUP           NAME
         0 oneadmin        oneadmin        my_service

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

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

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

....
```

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



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.

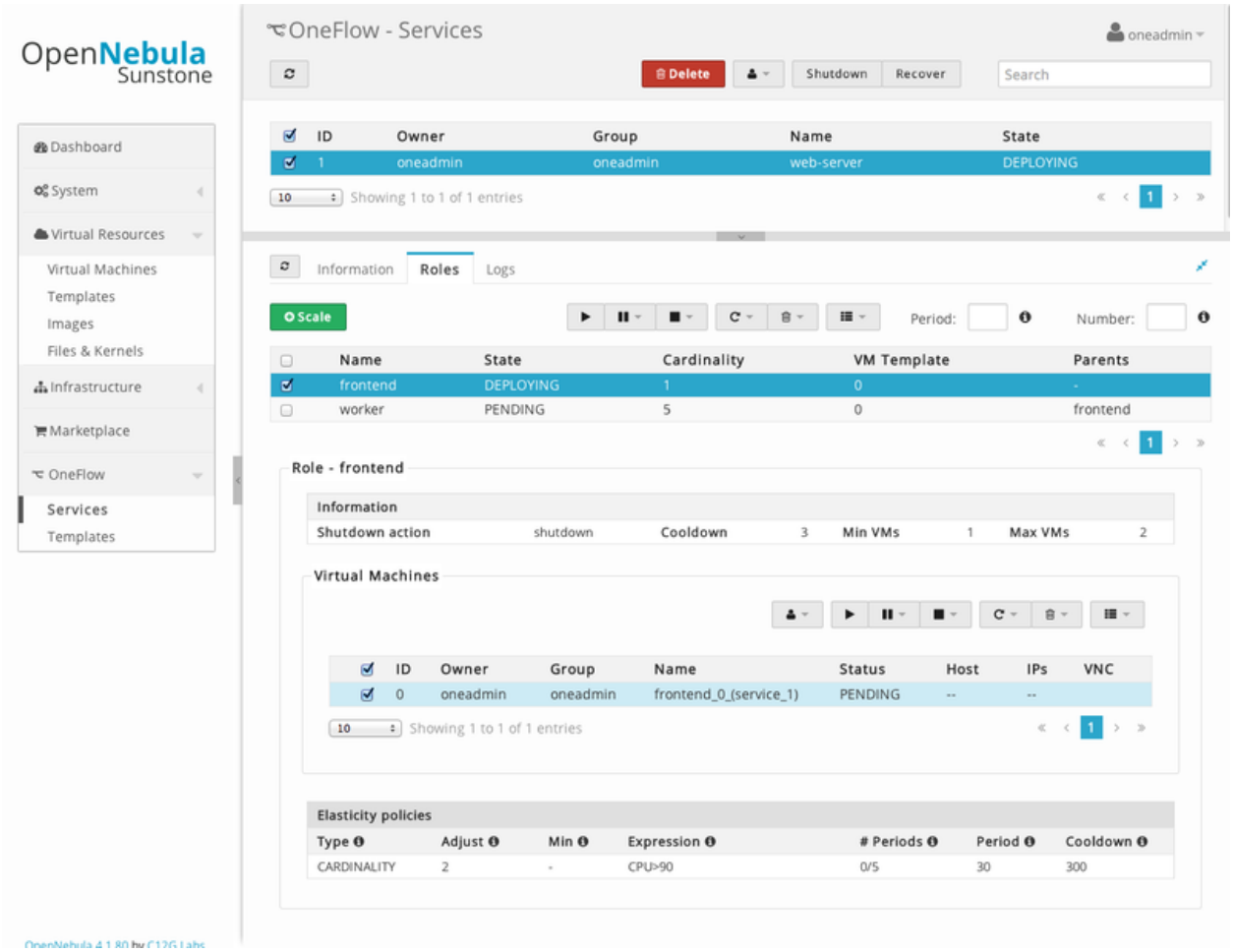
Create and List Existing Services

New Services are created from Service Templates, using the `oneflow-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 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
```

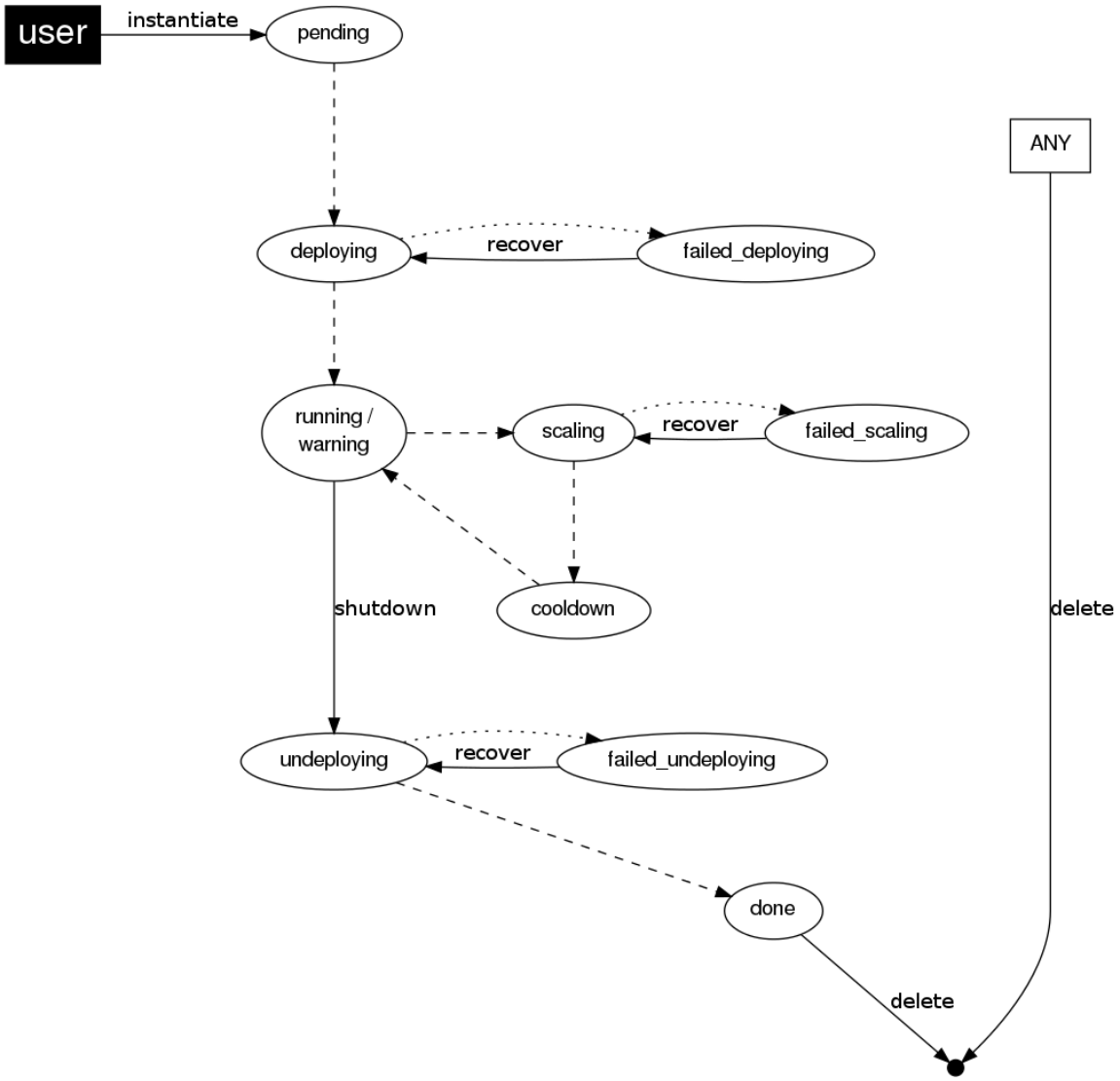
```
ROLE db_master
ROLE STATE           : DEPLOYING
PARENTS              : frontend
CARNIDALITY          : 1
VM TEMPLATE          : 1
NODES INFORMATION
 VM_ID NAME                    STAT UCPU    UMEM HOST                       TIME
     1                         init           0K                      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           0K                      0d 00h00
     3                         init           0K                      0d 00h00
     4                         init           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 `oneflow shutdown` and `oneflow delete`.

The command `oneflow 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 `oneflow delete`.

If a Service and its VMs must be immediately undeployed, the command `oneflow 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 `oneflow 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 `oneflow scale`:

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

To define automatic elasticity policies, proceed to the *elasticity documentation guide*.

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	**VMs in `DONE` or `FAILED` are deleted.** VMs in `UNKNOWN` are booted.
FAILED_UNDEPLOYING	UNDEPLOYING	The undeployment is resumed.
FAILED_SCALING	SCALING	**VMs in `DONE` or `FAILED` are deleted.** VMs in `UNKNOWN` are booted. For a scale-down, the shutdown actions are retried.
COOLDOWN	RUNNING	The Service is simply set to running before the cooldown period is over.
WARNING	WARNING	**VMs in `DONE` or `FAILED` are deleted.** 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.

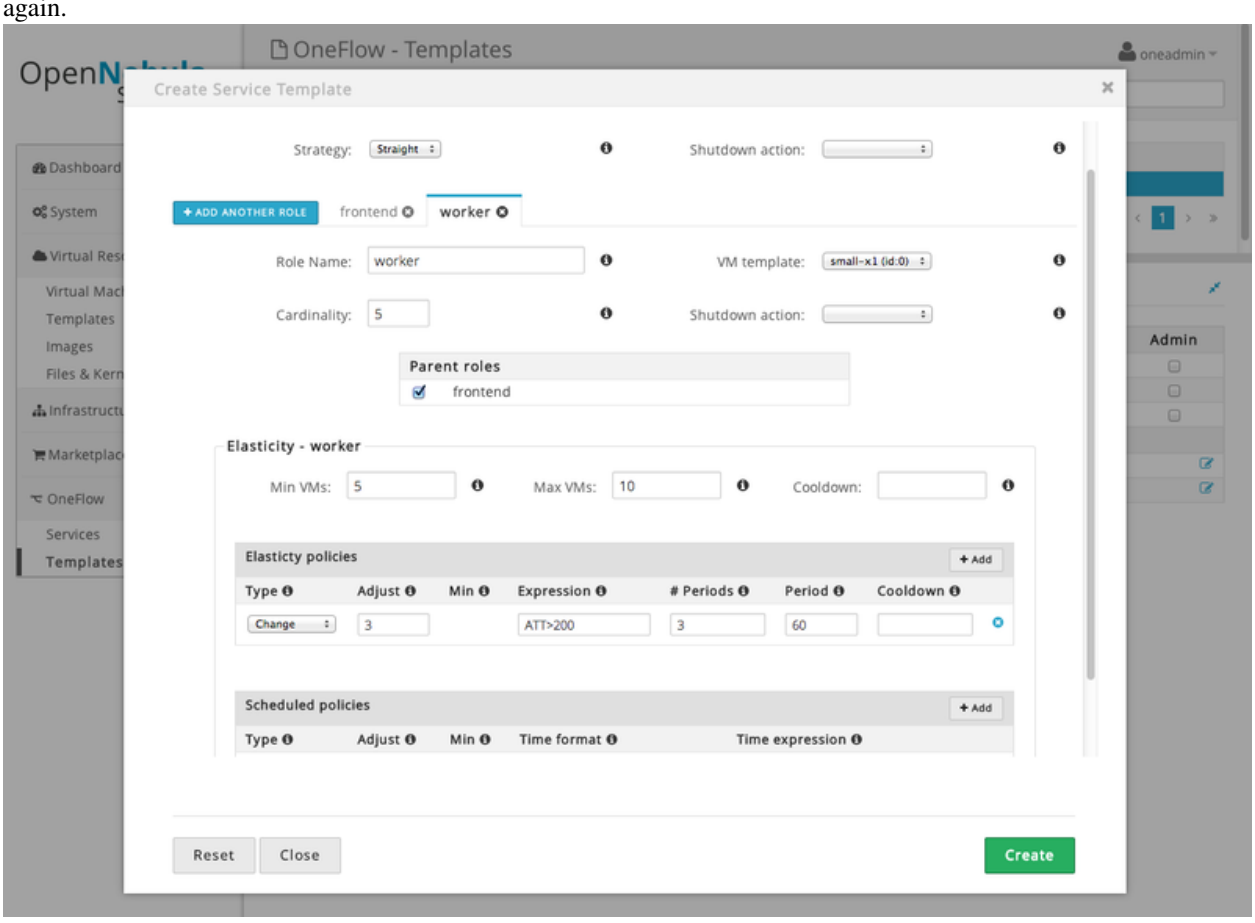
1.4.1 Overview

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.



1.4.2 Set the Cardinality of a Role Manually

The command `oneflow scale` starts the scalability immediately.

```
$ oneflow 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/substract the given number of VMs
- **CARDINALITY**: Set the cardinality to the given number
- **PERCENTAGE_CHANGE**: Add/substract 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, NET_TX and NET_RX.

```
"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, and /VM/TEMPLATE, 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 adjustements. Time is specified with the Unix cron sytax
start_time	string	No	Exact time for the adjustement

1.4.5 Visualize in the CLI

The `oneflow 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:

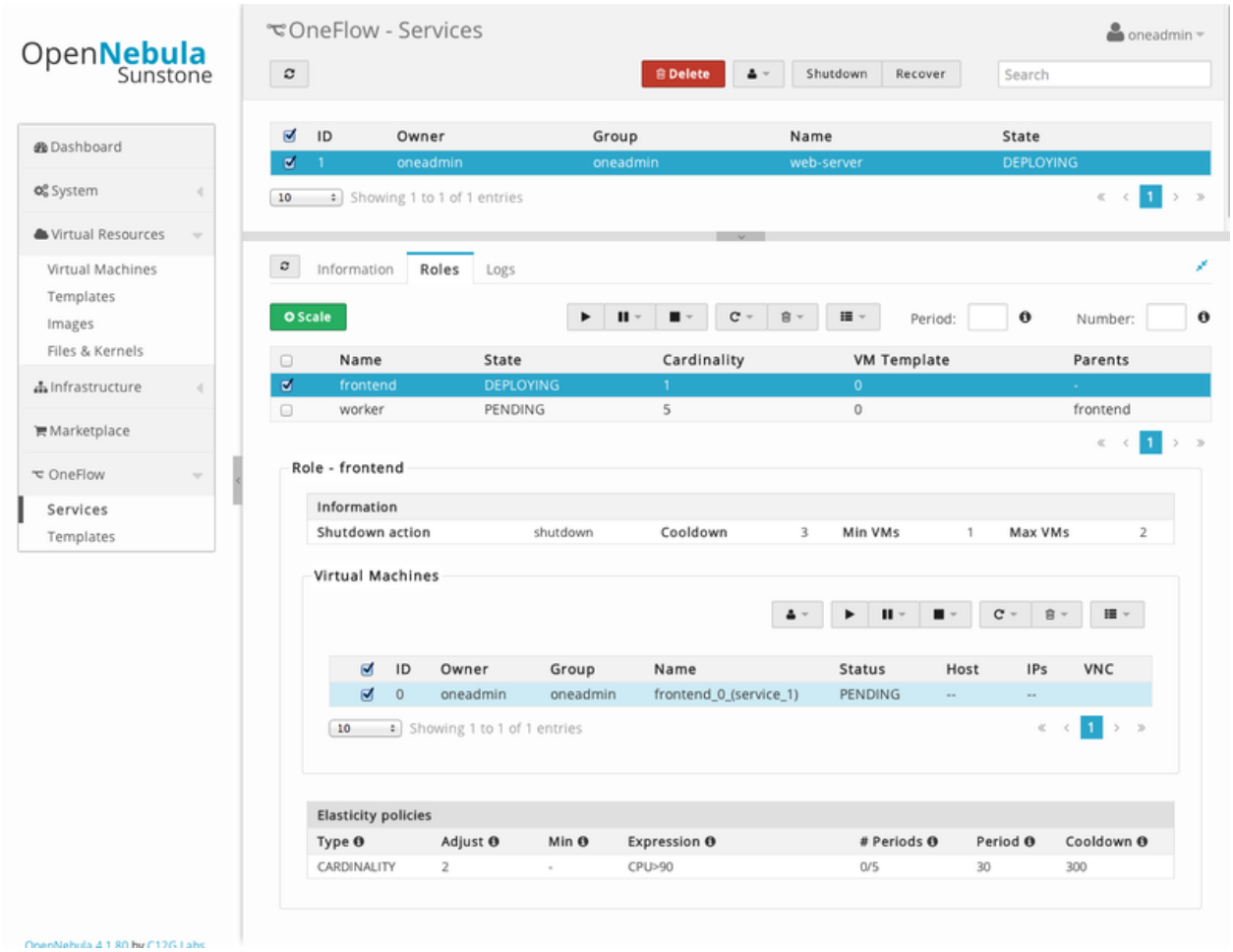
```
SERVICE 7 INFORMATION
...

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.



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",
```

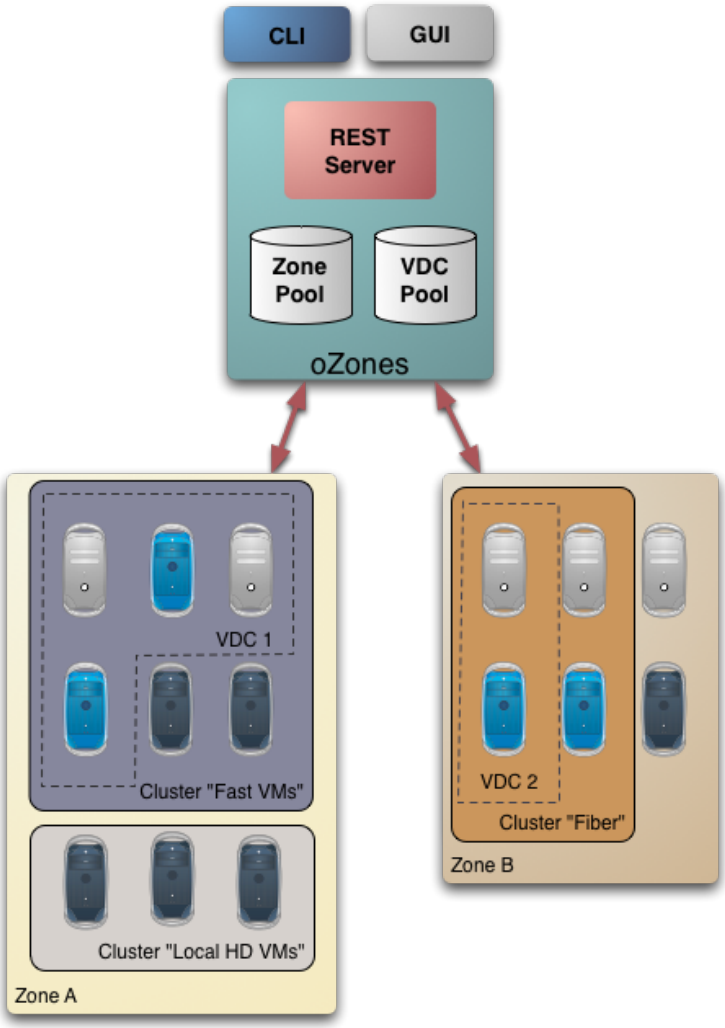
```
          "recurrence" : "0 9 * * mon,tue,wed,thu,fri",
          "adjust" : 6
        },
        {
          "type" : "CARDINALITY",
          "recurrence" : "0 13 * * mon,tue,wed,thu,fri",
          "adjust" : 10
        },
        {
          "type" : "CARDINALITY",
          "recurrence" : "30 22 * * mon,tue,wed,thu,fri",
          "adjust" : 2
        }
      ]
    }
  ]
}
```

CHAPTER

TWO

MULTIPLE ZONE AND VIRTUAL DATA CENTERS

2.1 OpenNebula Zones Overview



The **OpenNebula Zones** (oZones) component allows for the centralized management of multiple instances of OpenNebula (zones), managing in turn potentially different administrative domains. The module is run by the oZones administrator, with capacity to grant access to the different zones to particular users.

These zones can be effectively shared through the Virtual DataCenter (VDC) abstraction. A VDC is a set of virtual resources (images, VM templates, virtual networks and virtual machines) and users that manage those virtual resources, all sustained by infrastructure resources offered by OpenNebula. A VDC is supported by the resources of one zone, and it is associated to one *cluster* of the zone. The resources that the VDC can dispose of are a subset of that cluster. There is a special user (the VDC administrator) that can create new users inside the VDC, as well as manage all the virtual resources (but can not access other resources in the zone or even the see the physical hosts used for the VDC). VDC admin and users access the zone through a reverse proxy, so they don't need to know the endpoint of the zone, but rather the address of the oZones module and the VDC where they belong to.

The bird's-eye view of the oZones component can be sketched with a simple scenario. Let's take the point of view of the oZones manager that has access to two OpenNebula instances, managing resources in two different administrative domains. She can add those two instances as OpenNebula Zones in the oZones manager (provided she has the `oneadmin` credentials of both OpenNebula instances), and afterwards take a look at an aggregated view of resources combined from both zones. Also, she may want to give just a portion of the physical resources to a set of users, so she will create a VDC in one of the given zones, selecting a subset of the available hosts, and creating an account for the VDC admin. Once this is in place, she will be able to provide with access URL for the OpenNebula CLI and Sunstone GUI to the users, an url that will mask the location of the OpenNebula zone by using a reverse proxy. An example of such a URL can be:

```
http://ozones-server/MyVDC
```

2.1.1 Benefits

This new **Zones** functionality addresses many common requirements in enterprise use cases, like for instance:

- Complete **isolation** of users, organizations or workloads in different Zones with different levels of security or high availability
- Optimal **performance** with the execution of different workload profiles in different physical clusters with specific architecture and software/hardware execution environments
- Massive **scalability** of cloud infrastructures beyond a single cloud instance
- **Multiple site** support with **centralized management** and access to clouds hosted in different data centers to build a geographically distributed cloud

Moreover, the **VDC** mechanism allows advanced on-demand provisioning scenarios like:

- On-premise Private Clouds Serving **Multiple Projects, Departments, Units or Organizations**. On-premise private clouds in large organizations require powerful and flexible mechanisms to manage the access privileges to the virtual and physical infrastructure and to dynamically allocate the available resources. In these scenarios, the cloud administrator would create a VDC for each Department, dynamically allocation physical hosts according to their needs, and delegating the internal administration of the VDC to the Department IT administrator.
- Cloud Providers Offering **Virtual Private Cloud Computing**. There is a growing number of cloud providers, especially Telecom Operators, that are offering Virtual Private Cloud environments to extend the Private Clouds of their customers over virtual private networks, thus offering a more reliable and secure alternative to traditional Public Cloud providers. In this new cloud offering scenario, the cloud provider provides customers with a fully-configurable and isolated VDC where they have full control and capacity to administer its users and resources. This combines a public cloud with the protection and control usually seen in a personal private cloud system. Users can themselves create and configure servers via the SunStone portal or any of the supported cloud APIs. The total amount of physical resources allocated to the virtual private cloud can also be adjusted.

2.1.2 Next Steps

- *Configure the server*
- *Manage Zones*
- *Manage VDCs*

2.2 OpenNebula Zones Server Setup

This guide intends to give a walk through the steps needed to correctly configure the oZones Server to start managing Zones and VDCs. Also, it provides steps to configure a reverse proxy based on the Apache web server to hide the VDC details from end users.

2.2.1 Requirements

- **Ruby Gems**
  - Rubygems needs to be installed
  - gem install json thin rack sinatra libopenssl-ruby
  - gem install sequel
- **Apache**
  - Version should be >=2.2
  - apt-get install libopenssl-ruby apache2
- **Zones**
  - There should be at least one Zone based on an OpenNebula 3.4+ installation, properly configured and running

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

2.2.2 Configuration

Configure Apache

Apache needs to be configured to act as a reverse proxy, using the mod_proxy module. To correctly configure it, the following steps need to be taken:

> **Warning:** The following details are valid for Ubuntu installations, but it should be fairly easy to extrapolate to any other linux flavor.

- Enable these modules:

```
$ sudo a2enmod rewrite
$ sudo a2enmod proxy_http
```

- Edit /etc/apache2/apache2.conf and add the following at the end

```
ServerName <hostname-of-ozones-front-end>
```

- Edit /etc/apache2/mods-available/proxy.conf and change `Deny from all` line to `Allow from all`

- Then edit /etc/apache2/sites-available/default. Change the following

```
<Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
```

To this:

```
<Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride all
```

- Restart apache

```
$ sudo /etc/init.d/apache2 restart
```

Configure oZones Server

Before starting the oZones server be sure to:

- If you are planning to use a DB backend , make sure you have at hand the credentials of a DB user with access to a pre-created database called `ozones`, and also the DNS or IP adress of the DB server.
- Edit `/etc/one/ozones-server.conf` and change any of the following parameters accordingly:

Attribute	Description
databse-type	This can to be set to 'sqlite' or 'mysql'. For the latter, a *ozones* named database needs to be created manually.
databas-eserver	Only needed for mysql and postgres backends. Syntax is <dbusername>:<dbuserpassword>@<DBserver_hostname>.
htaccess	Location of the root .htaccess file for the apache reverse proxying configuration, if not sure leave the default /var/www/.htaccess. This file needs to be writable by *oneadmin* (or the user executing the ozones-server), one option is to precreate the .htaccess file and change its owner to oneadmin.
dbdebug	Wether the DB related events are going to be logged or not.
host	Hostname of the server running the oZones server.
port	Port of the server where the oZones server will listen.

- Set `OZONES_AUTH` the first time the oZones server is started, it will add to the DB the credentials of the zones administrator (which is the user entitled to add new zones and created VDCs). This credentials will be retrieved from the file pointed out by the environment variable `$OZONES_AUTH`, which should contain the credentials separated by a colon, like 'username:password'. The same credentials will be needed to be used to access the oZones server using the CLI or the GUI.

Then start simply start the server that will be listening in the target URL with:

```
> ozones-server start
ozones-server listening on 127.0.0.1:6121
```

Configure oZones Client

You will need to set the following environment variables in order to use the CLI:

Variables	Description
OZONES_URL	Should point to the HTTP URL of the oZones server (defaults to `http://localhost:6121`).
OZONES_AUTH	Should point to a file containing the oZones administrator credentials separated by a colon, like 'username:password'.

2.3 Managing Multiple Zones

A Zone is essentially a group of interconnected physical hosts with hypervisors controlled by OpenNebula. A Zone can be added to the oZones server, providing valid oneadmin credentials, so it can contribute to the list of aggregated resources presented by the oZones component. A Zone can be further *compartmentalized* in *Virtual Data Center or VDCs*.

2.3.1 Adding a New Zone

This guide assumes that the oZones command line interface is correctly *installed and configured*.

Zones resource can be managed using the **onezone** command. In order to create a zone, we will need a Zone template, with the necessary information to setup such resource:

```
NAME=MyZone
ONENAME=oneadmin
ONEPASS=opennebula
ENDPOINT=http://zone.domain.name:2633/RPC2
SUNSENDPOINT=http://zone.domain.name:9869
SELFENDPOINT=http://zone.domain.name:4567/ui
```

The *oneadmin* credentials are checked against the OpenNebula present in the Zone by requesting the list of available hosts. No changes are made to the OpenNebula whatsoever. Let's create a Zone:

```
$ onezone create myzone.template
ID: 1
```

Now we can list the available zones:

```
$ onezone list
  ID            NAME                                    ENDPOINT
   1          MyZone        http://zone.domain.name:2633/RPC2
```

2.3.2 Examining a Zone

We can further examine the contents of the zone with the **onezone show** command:

```
$ onezone show 1
ZONE zone0 INFORMATION
ID             : 1
NAME           : zone0
ZONE ADMIN     : oneadmin
ZONE PASS      : 4478db59d30855454ece114e8ccfa5563d21c9bd
ENDPOINT       : http://zone.domain.name:2633/RPC2
# VDCS         : 0
```

Zone resources can be specifically queried with **onezone show <id>** plus one of the following flags:

- `vmtemplate` for VM templates
- `image` for Images
- `user` for Users
- `vm` for Virtual Machines
- `vnet` for Virtual Networks

- `host` for Hosts

Let's query the hosts of the newly created zone:

```
$ onezone show 1 host
ZONE zone0 INFORMATION
ID             : 1
NAME           : MyZone
ZONE ADMIN     : tinova
ZONE PASS      : 4478db59d30855454ece114e8ccfa5563d21c9bd
ENDPOINT       : http://zone.domain.name:2633/RPC2
# VDCS         : 0

  ID NAME                RVM   TCPU   FCPU   ACPU   TMEM   FMEM   AMEM   STAT
   0 MyHost                0      0      0    100     0K     0K     0K     on
   1 EC2Host               0      0      0    100     0K     0K     0K     on
   2 64BitsComp            0      0      0    100     0K     0K     0K     on
```

2.3.3 Deleting a Zone

We can delete a zone with **onezone delete**, providing the Zone ID:

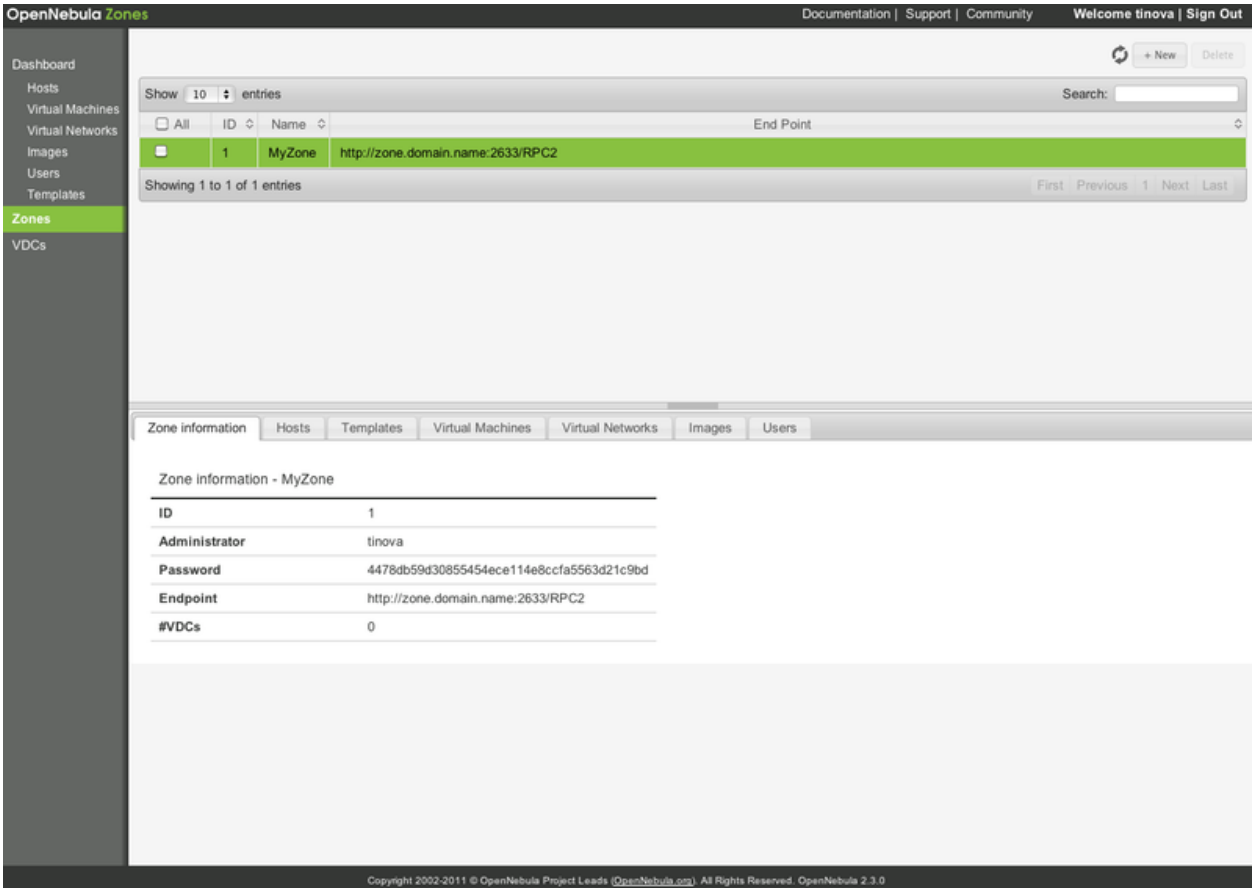
```
$ onezone delete 1
Resource zone with id 1 successfully deleted
```

2.3.4 Using the oZones GUI

Pointing the browser to

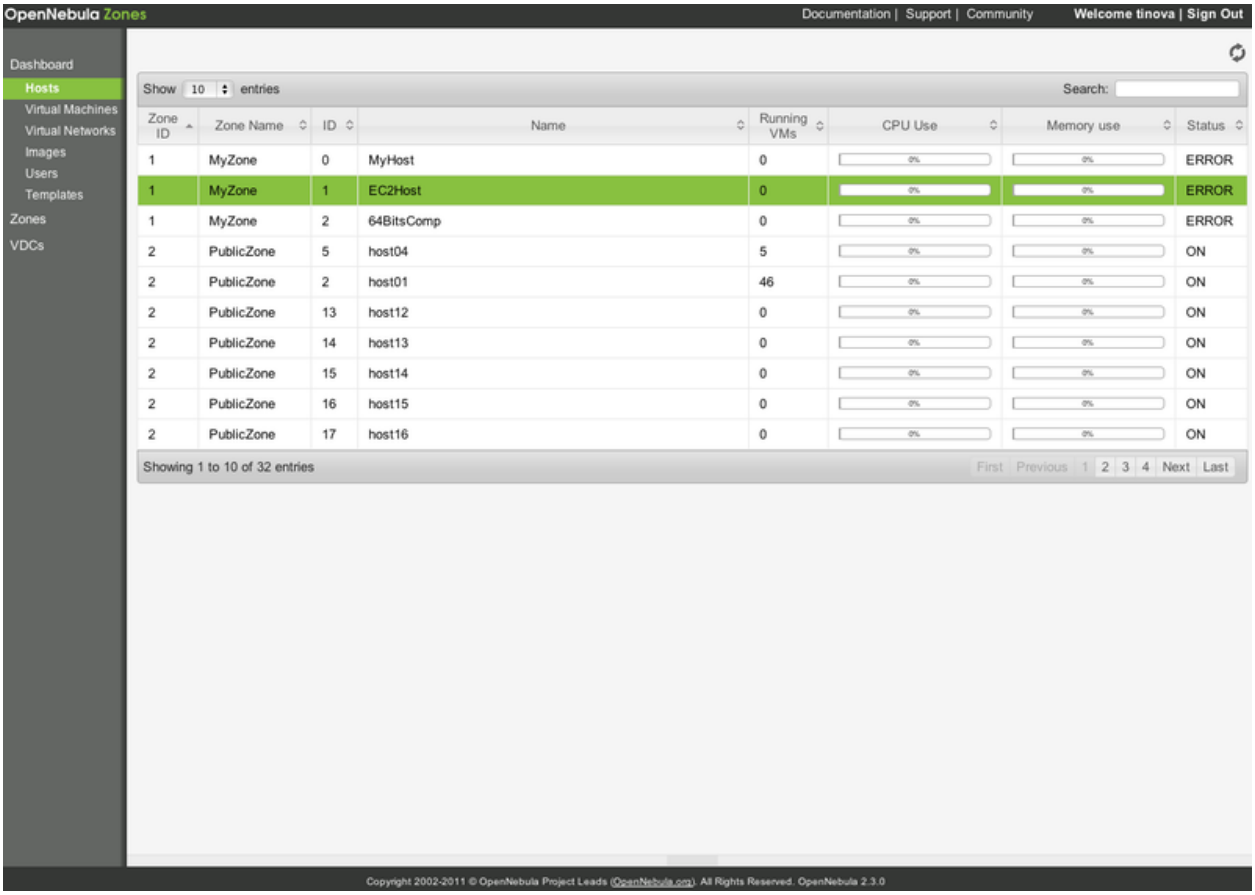
```
http://ozones.server.domainname:6121
```

Will give access to the oZones GUI, where all the functionality of the CLI is offered.



Examining Aggregated Resources

Also, in the GUI there is the ability to see the aggregated resources from multiple zones: Templates, Images, Users, Virtual Machines, Virtual Networks and Hosts.



2.4 Managing Multiple Virtual Data Centers

Virtual Data Centers (VDCs) are fully-isolated virtual infrastructure environments where a group of users, under the control of the VDC administrator, can create and manage compute, storage and networking capacity. The VDC administrator can create new users inside the VDC. Both VDC admins and users access the zone through a reverse proxy, so they don't need to know the endpoint of the zone, but rather the address of the oZones module and the VDC where they belong to.

2.4.1 Adding a New VDC

This guide assumes that the oZones command line interface is correctly *installed and configured*.

VDCs resources can be managed using the **onevdc** command. In order to create a VDC, we will need a VDC template, with the necessary information to setup such resource:

```
NAME=MyVDC
VDCADMINNAME=vdcadmin
VDCADMINPASS=vdcpass
CLUSTER_ID=100
ZONE_ID=1
HOSTS=4,7,9,10
DATASTORES=100,101
NETWORKS=0
```

Once created, the above VDC template will mean the following in the OpenNebula managing the Zone with ID 1:

- New group called `MyVDC`
- New user called `vdcadmin`, using `vdcpass` as password
- A set of ACLs to allow users from group `MyVDC` to:
  - deploy in Hosts 4,7,9 and 10
  - allow `vdcadmin` to create new users
  - manage newly created resources in the group.
  - use previously available resources. This resources must belong to the cluster with 100, such as virtual network with id 0 and datastores 100 and 101 (with their respective images)

Let's create the VDC:

```
$ onevdc create vdctest.ozo
ID: 4
```

Now it's time to see if it appears in the listing:

```
$ onevdc list
  ID            NAME                                    ZONEID
   4         MyVDC                                         1
```

If we have access to the Zone 3, we can see that the following has just been created:

Group

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

User

```
$ oneuser list
  ID GROUP    NAME            AUTH                                                PASSWORD
   0 oneadmin oneadmin        core                4478db59d30855454ece114e8ccfa5563d21c9bd
   1 oneadmin serveradmin     server_c            7fa9d527c7690405aa639f3280aaef81d13cff5c
   2 MyZone   myvdcadmin      core                c5802acd106f0dfb65c506d50f0b7d5abdcb4494
```

ACLs

```
$ oneacl list
   ID     USER RES_VHNIUTGDC   RID OPE_UMAC
    0       @1     V-NI-T---     *     ---c
    1       @1     -H-------     *     -m--
    2     @100     V--I-T---     *     ---c
    3       #2     ----U----     *     ---c
    4       #2     ----U----  @100     uma-
    5       #2     V--I-T---  @100     um--
    6     @100     -H-------    #2     -m--
    7     @100     -------D-    #1     u---
    8     @100     --N------    #0     u---
```

VDC Resource Sharing

By default, the oZones server will check that no resources such as Virtual Networks, Datastores and Hosts are shared between VDCs. This behavior can be overriden by setting the following in the VDC template

```
FORCE=yes
```

or, more intuitively, through the oZones GUI.

2.4.2 Examining a VDC

Once created, a VDC can be asked for details with **onevdc show**, passing the VDC ID:

```
$ onevdc show 4
VDC  INFORMATION
ID           : 4
NAME         : MyZone
ZONE_ID      : 4
CLUSTER_ID   : 100
GROUP_ID     : 100
VDCADMIN     : myvdcadmin
HOSTS        : 2
DATASTORES   : 1
NETWORKS     : 0
```

2.4.3 Deleting a VDC

A VDC can be deleted if the VDC ID is known, using **onevdc delete**

```
$ onevdc delete 4
Resource vdc with id 4 successfully deleted
```

2.4.4 Adding or Removing Resources to/from VDC

Resources such as Datastores, hosts and Virtual Networks pertaining to the cluster associated to the VDC can be updated, using the CLI and the oZones GUI.

The CLI offers the functionality through the `onevdc` command:

```
* add <vdcid>
     Adds the set of resources to the VDC
     valid options: force, hosts, datastores, networks
* del <vdcid>
     Deletes the set of resources from the VDC
     valid options: hosts, datastores, networks
```

In the oZones GUI the VDC can be updated graphically.

2.4.5 Using VDCs

After *creating a Zone*, and a *VDC* inside it, users can start to be added to the VDC in order to allow them to use the VDC resources. This can be done through the command line interface or the Sunstone GUI.

Accessing through the Command Line Interface

There are two needed environment variable to access the VDC:

- **ONE_XMLRPC** This is an environment variable that tells OpenNebula CLI where to look for the OpenNebula server. It is going to be the address of the reverse proxy, with a reference to the VDC that the user is trying to access. The proxy will redirect the requests to the appropriate Zone. If the VDC has **MyVCD** as name, the variable would look like

```
ONE_XMLRPC=http://ozones.server/MyVDC
```

- **ONE_AUTH** It should point to a file containing valid credentials for the VDC.

For example, let's say we created the VDC used above on a oZones server running at server *ozones.server*.

The variables should be:

- ONE_XMLRPC=http://ozones.server/MyVDC
- ONE_AUTH=~/.one/one_auth

where ~/.one/one_auth contains:

```
vdcadmin:vdcpass
```

Once this is in place, the VDC admin can start adding new users to the VDC. This works pretty much as a normal `oneadmin` session (although with no ability to change the host pool):

```
$ oneuser create vdcuser1 password
```

Now, the VDC admin or the user can start defining other resources, such as Virtual Networks, Templates, Images, etc.

Accessing through Sunstone

The reverse proxy is set to redirect requests from /sunstone_MyVDC, so just pointing a browser to

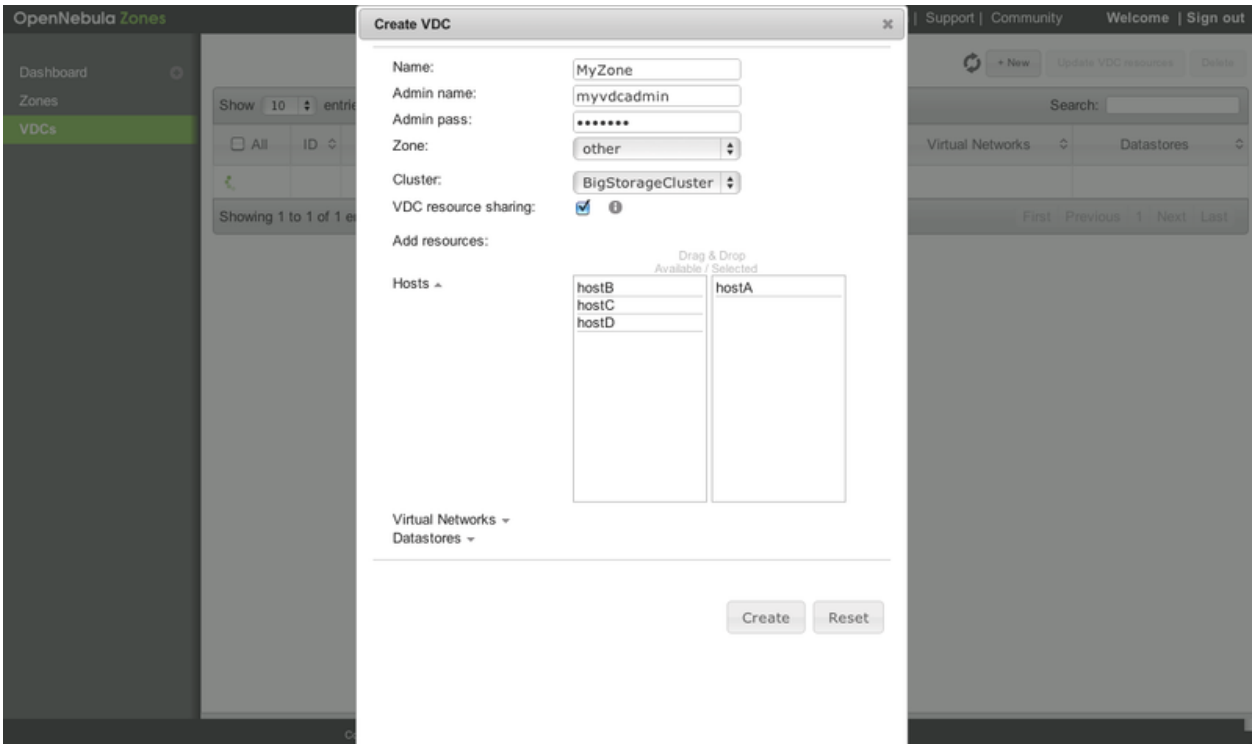
```
http://ozones.server/sunstone_MyVDC/
```

should get you to the VDC. Please note the trailing back slash, otherwise the proxy rules won't properly apply.

Now just log in with the VDCAdmin credentials and start creating users for the VDC.

2.4.6 Using the oZones GUI

All the VDC functionality can be accessed using the CLI. The creation of VDCs using the GUI is specially useful, as the Zone resources can be easily picked from a list:


OpenNebula Zones
Dashboard
Zones
VDCs
Create VDC
Name:
MyZone
Admin name:
myvdcadmin
Admin pass:
Zone:
other
Cluster:
BigStorageCluster
VDC resource sharing:
Add resources:
Drag & Drop
Available / Selected
Hosts
hostB
hostC
hostD
hostA
Virtual Networks
Datastores
Create
Reset
Support | Community
Welcome | Sign out
+ New
Search:
Virtual Networks
Datastores

CHAPTER

THREE

SCALABILITY

3.1 Configuring Sunstone for Large Deployments

Low to medium enterprise clouds will typically deploy Sunstone in a single machine a long 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.

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*.

```
$ 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.

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 an 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 novcn 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 fun 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 an 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
  # !!! Be sure to point DocumentRoot to 'public'!
  DocumentRoot /usr/lib/one/sunstone/public
  <Directory /usr/lib/one/sunstone/publicc>
     # This relaxes Apache security settings.
     AllowOverride all
     # MultiViews must be turned off.
     Options -MultiViews
  </Directory>
</VirtualHost>
```

Make sure you change the directory to `/usr/share/opennebula/sunstone/public` if you are using Debian or Ubuntu.

Now the configuration should be ready, restart of reload apache configuration to start the application and point to the virtual host 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 <= 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 xmlprc calls the default values for the xmlprc 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
```

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*.

CHAPTER

FOUR

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 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   = "host_error.rb",
    arguments = "$HID -r n",
    remote    = no ]
#-------------------------------------------------------------------------------
```

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

Argument	Description
Host ID	ID of the host containing the VMs to treat. It is compulsory and better left to **$HID**, that will be automatically filled by OpenNebula with the Host ID of the host that went down.
Action	This defined the action to be performed upon the VMs that were running in the host that went down. This can be **-r** (recreate) or **-d** (delete).
DoSuspended	This argument tells the hook to perform Action to suspended VMs belonging to the host that went down (**y**), or not to perform Action to them (**n**) .

More information on hooks *here*.

Additionally, there is a corner case that in critical production environments should be taken into account. OpenNebula also has become tolerant to network errors (up to a limit). This means that a spurious network error won't trigger the hook. But if this network error stretches in time, the hook may be triggered and the VMs deleted and recreated. When (and if) the network comes back, there will be a potential clash between the old and the reincarnated VMs. In order to prevent this, a script can be placed in the cron of every host, that will detect the network error and shutdown the host completely (or delete the VMs).

4.1.2 Virtual Machine Failures

The Virtual Machine lifecycle management can fail in several points. The following two cases should cover them:

- **VM fails**: This may be due to a network error that prevents the image to be staged into the node, a hypervisor related issue, a migration problem, etc. The common symptom is that the VM enters the FAILED state. In order to deal with these errors, a Virtual Machine hook can be set to `recreate` the failed VM (or, depending the production scenario, delete it). This can be achieved by uncommenting the following (for recreating, the deletion hook is also present in the same file) in `/etc/one/oned.conf` (and restarting `oned`):

```
#-------------------------------------------------------------------------------
VM_HOOK = [
   name      = "on_failure_recreate",
   on        = "FAILURE",
   command   = "onevm delete --recreate",
   arguments = "$VMID" ]
#-------------------------------------------------------------------------------
```

- **VM crash**: This point is concerned with crashes that can happen to a VM **after** it has been successfully booted (note that here boot doesn't refer to the actual VM boot process, but to the OpenNebula boot process, that comprises staging and hypervisor deployment). OpenNebula is able to detect such crashes, and report it as the VM being in an UNKNOWN state. This failure can be recovered from using the `onevm boot` functionality.

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 the Red Hat HA Cluster suite tested in a CentOS installation; but generic considerations and requirements for this setup are discussed to easily implement this solution with other systems.

4.2.1 Overview

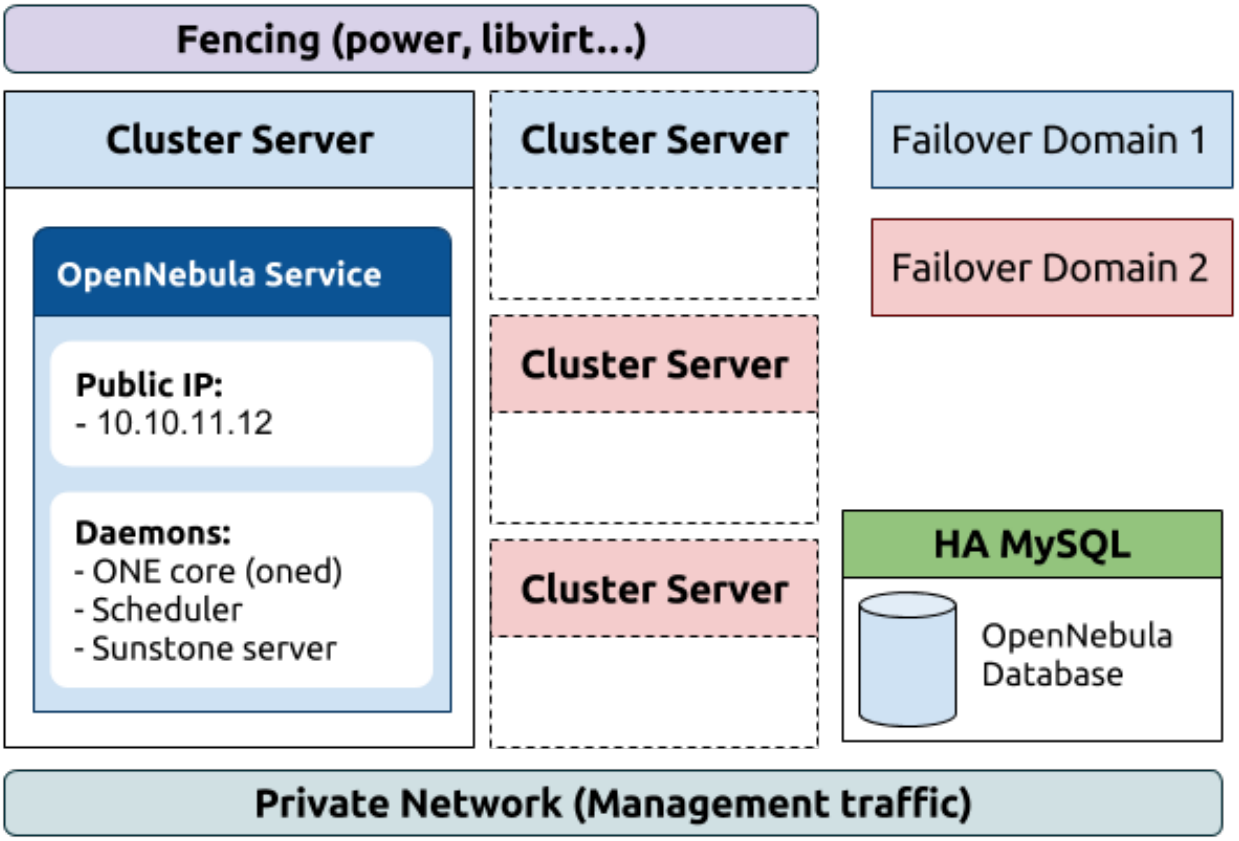
In terms of high-availability, OpenNebula consists in three different basic services, namely:

- **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 such a configuration, given the lightweight and negligible start times of the core services, does not suppose any advantage.

In this guide we assume that the DB backing OpenNebula core state is also configured in a HA mode. The procedure for MySQL is well documented elsewhere. Although Sqlite could also be used it is not recommended for a HA deployment.



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 collocated and run on the same server in this case. You can however easily modify this configuration to split them and allocate 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 and configured based on the RedHat Cluster suite:

* **Cluster management**, CMAN (cluster manager) and corosync. These components manage cluster membership and quorum. It prevents service corruption in a distributed setting because of a *split-brain* condition (e.g. two opennebulas updating the DB).

* **Cluster configuration system**, CCS. It keeps and synchronizes the cluster configuration information. There are other windows-based configuration systems.

* **Service Management**, rgmanager. This module checks service status and start/stop them as needed in case of failure.

* **Fencing**, in order to prevent OpenNebula DB corruption it is important to configure a suitable fencing mechanism.

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 se_linux.

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 all the cluster servers install the cluster components:

```
# yum install ricci
# passwd ricci
```

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

```
# yum install cman rgmanager
# yum install ccs
```

Finally enable the daemons and start ricci.

```
# chkconfig ricci on
# chkconfig cman rgmanager on
# chkconfig rgmanager on
# service ricci start
```

Step 3: Create the Cluster and Failover Domain

Cluster configuration is stored in `/etc/cluster/cluster.conf` file. You can either edit this file directly or use the ccs tool. It is important, however to synchronize and activate the configurtion on all nodes after a change.

To define the cluster using ccs:

```
# ccs -h one-server1 --createcluster opennebula
# ccs -h one-server1 --setcman two_node=1 expected_votes=1
# ccs -h one-server1 --addnode one-server1
# ccs -h one-server1 --addnode one-server2
# ccs -h one-server1 --startall
```

Warning: You can use the -p option in the previous commands with the password set for user ricci.

Now you should have a cluster with two nodes, note the specific quorum options for cman, running. Let's create one failover domain for OpenNebula services consisting of both servers:

```
# ccs -h one-server1 --addfailoverdomain opennebula ordered
# ccs -h one-server1 --addfailoverdomainnode opennebula one-server1 1
# ccs -h one-server1 --addfailoverdomainnode opennebula one-server2 2
# ccs -h one-server1 --sync --activate
```

Step 4: Define the OpenNebula Service

As pointed out previously we'll use just one fail-over domain with all the OpenNebula services co-allocated in the same server. You can easily split the services in different servers and failover domains if needed (e.g. for security reasons you want Sunstone in other server).

First create the resources of the service: A IP address to reach Sunstone, the one init.d script (it starts oned and scheduler) and the sunstone init.d script

```
# ccs --addresource ip address=10.10.11.12 sleeptime=10 monitor_link=1
# ccs --addresource script name=opennebula file=/etc/init.d/opennebula
# ccs --addresource script name=sunstone file=/etc/init.d/opennebula-sunstone
```

Finally compose the service with these resources and start it:

```
# ccs --addservice opennebula domain=opennebula recovery=restart autostart=1
# ccs --addsubservice opennebula ip ref=10.10.11.12
# ccs --addsubservice opennebula script ref=opennebula
# ccs --addsubservice opennebula script ref=sunstone
# ccs -h one-server1 --sync --activate
```

As a reference the /etc/cluster/cluster.conf should look like:

```
<?xml version="1.0"?>
<cluster config_version="17" name="opennebula">
  <fence_daemon/>
  <clusternodes>
    <clusternode name="one-server1" nodeid="1"/>
    <clusternode name="one-server2" nodeid="2"/>
  </clusternodes>
  <cman expected_votes="1" two_node="1"/>
  <fencedevices/>
  <rm>
    <failoverdomains>
      <failoverdomain name="opennebula" nofailback="0" ordered="1" restricted="0">
        <failoverdomainnode name="one-server1" priority="1"/>
        <failoverdomainnode name="one-server2" priority="2"/>
      </failoverdomain>
    </failoverdomains>
    <resources>
      <ip address="10.10.11.12" sleeptime="10"/>
```

```
      <script file="/etc/init.d/opennebula" name="opennebula"/>
      <script file="/etc/init.d/opennebula-sunstone" name="sunstone"/>
    </resources>
    <service domain="opennebula" name="opennebula" recovery="restart">
      <ip ref="10.10.11.12"/>
      <script ref="opennebula"/>
      <script ref="sunstone"/>
    </service>
  </rm>
</cluster>
```

4.2.4 Fencing and Virtual Clusters

Fencing is an essential component when setting up a HA cluster. You should install and test a proper fencing device before moving to production. In this section we show how to setup a special fencing device for virtual machines.

OpenNebula can be (and it is usually) installed in a virtual machine. Therefore the previous one-server1 and one-server2 can be in fact virtual machines running in the same physical host (you can run them in different hosts, requiring a different fencing plugin).

In this case, a virtual HA cluster running in the same host, you could control misbehaving VMs and restart OpenNebula in other virtual server. However, if you need a to control also host failures you need to fencing mechanism for the physical host (typically based on power).

Let's assume then that one-server1 and one-server2 are VMs using KVM and libvirt, and running on a physical server.

Step 1: Configuration of the Physical Server

Install the fence agents:

```
yum install fence-virt fence-virtd fence-virtd-multicast fence-virtd-libvirt
```

Now we need to generate a random key, for the virtual servers to communicate with the dencing agent in the physical server. You can use any convinient method, for example: generate key ro access xvm

```
# mkdir /etc/cluster
# date +%s | sha256sum | base64 | head -c 32  > /etc/cluster/fence_xvm.key
# chmod 400 /etc/cluster/fence_xvm.key
```

Finally configure the fence-virtd agent

```
# fence-virtd -c
```

The configuration file should be similar to:

```
=== Begin Configuration ===
backends {
  libvirt {
    uri = "qemu:///system";
  }

}

listeners {
  multicast {
    interface = "eth0";
    port = "1229";
```

```
    family = "ipv4";
    address = "225.0.0.12";
    key_file = "/etc/cluster/fence_xvm.key";
  }

}

fence_virtd {
  module_path = "/usr/lib64/fence-virt";
  backend = "libvirt";
  listener = "multicast";
}

=== End Configuration ===
```

Warning: Interface (eth0 in the example) is the interface used to communicate the virtual and physical servers.

Now you can start and test the fencing agent:

```
# chkconfig fence_virtd on
# service fence_virtd start
# fence_xvm -o list
```

Step 2: Configuration of the Virtual Servers

You need to copy the key to each virtual server:

```
scp /etc/cluster/fence_xvm.key one-server1:/etc/cluster/
scp /etc/cluster/fence_xvm.key one-server2:/etc/cluster/
```

Now you should be able to test the fencing agent in the virtual nodes:

```
# fence_xvm -o list
```

Step 3: Configure the Cluster to Use Fencing

Finally we need to add the fencing device to the cluster:

```
ccs --addfencedev libvirt-kvm agent=fence_xvm key_file="/etc/cluster/fence_xvm.key" multicast_address
```

And let the servers use it:

```
# ccs --addmethod libvirt-kvm one-server1
# ccs --addmethod libvirt-kvm one-server2
# ccs --addfenceinst libvirt-kvm one-server1 libvirt-kvm port=one1
# ccs --addfenceinst libvirt-kvm one-server2 libvirt-kvm port=one2
```

Finally synchronize and activate the configuration:

```
# ccs -h one-server1 --sync --activate
```

4.2.5 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 will be a set of Virtual

Machines which will 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 succesfully 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

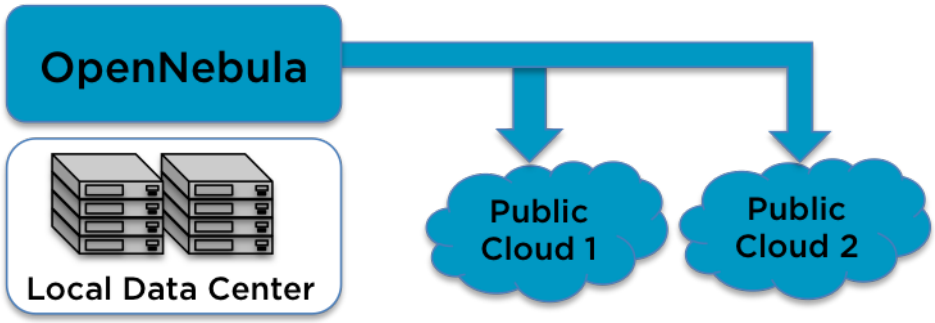
CHAPTER

FIVE

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, or a partner infrastructure running a different OpenNebula instance. 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 an AWS EC2 region is modelled as any other host (albeit of potentially a much bigger capacity), so the scheduler can place VMs in EC2 as it will do in any other local host.

```
$ 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 Amazon EC2. 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 Amazon 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*

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.

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.

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
```

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 ec2-describe-images.
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 *ssh -i id_keypair* or *scp -i id_keypair*
LICENSEPOOL	–license-pool
BLOCKDEVICEMAPPING	The block device mapping for the instance. More than one can be specified in a space-separated list. Check the –block-device-mapping option of the EC2 CLI Reference for the syntax
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 *ec2-associate-address -i i-0041230 -a elasticip*.
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).
ELASTICIP	EC2 Elastic IP address to assign to the instance. This parameter is passed to the command *ec2-associate-address -i i-0041230 elasticip*.
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 atributes 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 UPCASE
 Example:
 <TEMPLATE>
   <EC2>
     <KEYPAIR>gsg-keypair</KEYPAIR>
     <INSTANCETYPE>m1.small</INSTANCETYPE>
   </EC2>
 </TEMPLATE>
-->

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

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 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
$ ontemplate 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              : 0K
NET_TX              : 0K
USED CPU            : 0

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 schedul
-  ``MAX_HOST``: Maximum number of Virtual Machines dispatched to a given host in each scheduling act
```

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_DISPATH 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"
```

CHAPTER

SIX

APPLICATION INSIGHT

6.1 OneGate

OneGate allows Virtual Machine guests to push monitoring information to OpenNebula. Users and administrators can use it to gather metrics, detect problems in their applications, and trigger OneFlow auto-scaling rules

6.1.1 Next Steps

- *OneGate Server Configuration*
- *Application Monitoring*

6.2 OneGate Server Configuration

The OneGate service allows Virtual Machines guests to push monitoring information to OpenNebula. Althouth 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

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

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*.

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

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

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

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

# Authentication driver for incomming 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
```

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*.

6.3 Application Monitoring

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

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 send monitoring metrics to OneGate.

When OneGate checks the VM ID and the token sent, 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.

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" ]
```

When this Template is instantiated, OpenNebula will automatically add the ONEGATE_URL 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_URL="http://192.168.0.1:5030/vm/0",
  TARGET="hdb",
  TOKEN="YES" ]
```

Push Metrics from the VM Guest

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_URL='http://192.168.0.1:5030/vm/0'
TARGET='hdb'
TOKEN='yes'

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

With that data, you can perform this http request message:

- **Request**: `PUT ONEGATE\_URL`.
- **Headers**: `X-ONEGATE-TOKEN: token.txt contents`.
- **Body**: Monitoring values, in the usual `ATTRIBUTE = VALUE` OpenNebula syntax.

For example, using the curl command:

```
curl -X "PUT" http://192.168.0.1:5030/vm/0 --header "X-ONEGATE-TOKEN: yCxieDUS7kra7Vn9ILA0+g==" -d "A
```

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

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

6.3.3 Sample 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_URL ]; then
    echo "ONEGATE_URL 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
################################################################################

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

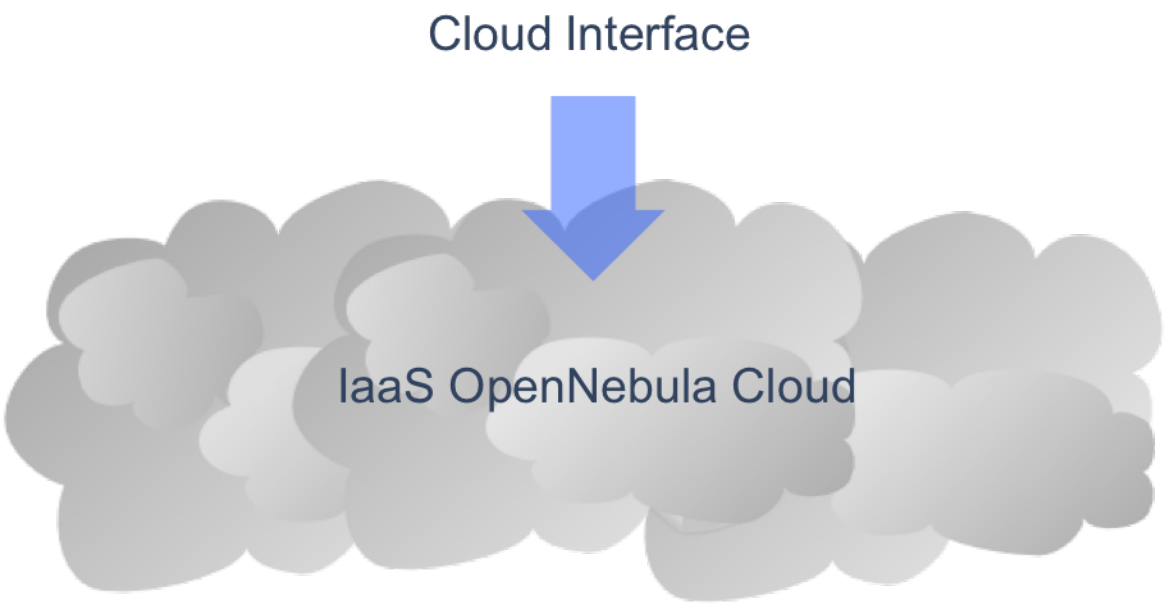
CHAPTER

SEVEN

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*
- *OGF OCCI*

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
```

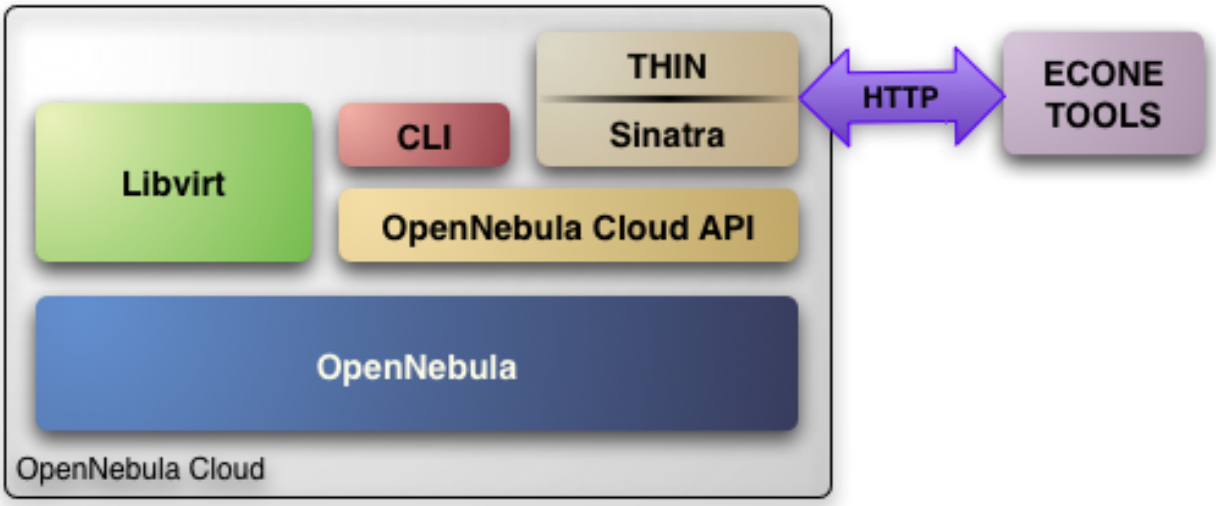
7.1.3 How the System Operates

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 dependencias 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 incomming requests
- core_auth: Authentication driver to communicate with OpenNebula core. Check *this guide* for more information about the core_auth syste

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 descibe_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 eached
- 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

- ebs_fstype: 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
...

$ ontemplate create /tmp/m1.small
$ ontemplate chgrp m1.small users
$ ontemplate 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 easliy 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 FIXED VirtualNetwork containing the public IPs that will be controlled by the EC2 users:

```
NAME    = "ElasticIPs"
TYPE    = FIXED

PHYDEV  = "eth0"
VLAN    = "YES"
VLAN_ID = 50
BRIDGE  = "brhm"

LEASES  = [IP=10.0.0.1]
LEASES  = [IP=10.0.0.2]
LEASES  = [IP=10.0.0.3]
LEASES  = [IP=10.0.0.4]

# 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 were 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 milleage 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/lighttp
```

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:

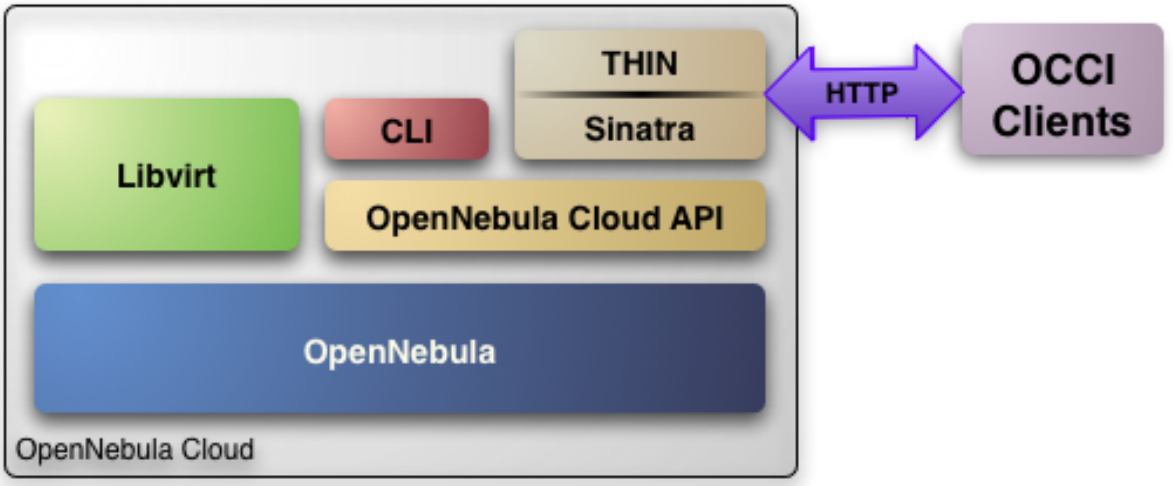
```
# Host and port where econe server will run
:host: localhost
:port: 4567

#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 OCCI Server Configuration

The OpenNebula OCCI (Open Cloud Computing Interface) server is a web service that enables you to launch and manage virtual machines in your OpenNebula installation using an implementation of the OGF OCCI API specification based on the draft 0.8. This implementation also includes some extensions, requested by the community, to support OpenNebula specific functionality. The OpenNebula OCCI 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 following sections explain how to install and configure the OCCI service on top of a running OpenNebula cloud.

Warning: The OpenNebula OCCI service provides an OCCI interface to your cloud instance, that can be used alongside the native OpenNebula CLI, Sunstone or even the EC2 Query API

Warning: The OpenNebula distribution includes the tools needed to use the OpenNebula OCCI service

7.3.1 Requirements

You must have an OpenNebula site properly configured and running to install the OpenNebula OCCI service, 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 OCCI 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 OCCI dependencias as explained at the end of the *Building from Source Code guide*

7.3.2 Considerations & Limitations

The OCCI Server included in the OpenNebula distribution does not implement the latest OCCI specification, it is based on the draft 0.8 of the OFG OCCI specification. The implementation of the latest specification is being developed by TU-Dortmund in a ecosystem project. You can check the documentation of this project in the following link

7.3.3 Configuration

occi-server.conf

The service is configured through the `/etc/one/occi-server.conf` file, where you can set up the basic operational parameters for the OCCI service, namely:

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 OCCI server will run. `port`: Port where OCCI server will run. `ssl_server`: SSL proxy 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 incoming requests `core_auth`: Authentication driver to communicate with OpenNebula core

Resources

`instance_types`: The Computes types for your cloud `datastore_id`: Datastore in which the Images uploaded through OCCI will be allocated, by default 1 `cluster_id`: Cluster associated with the OCCI resources, by default no Cluster is defined

> **Warning:** The `SERVER` **must** be a FQDN, do not use IP's here

> **Warning:** Preserve YAML syntax in the `occi-server.conf` file

Example:

```
################################################################
# Server configuration
################################################################

# Directory to store temp files when uploading images
:tmpdir: /var/tmp/one

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

# Host and port where OCCI server will run
:host: 127.0.0.1
:port: 4567

# SSL proxy that serves the API (set if is being used)
#:ssl_server: fqdm.of.the.server

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

# Authentication driver for incomming requests
#   occi, for OpenNebula's user-password scheme
#   x509, for x509 certificates based authentication
#   opennebula, use the driver defined for the user in OpenNebula
:auth: occi

# 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: 3

################################################################
# Resources
################################################################

# Cluster associated with the OCCI resources, by default no Cluster is defined
#:cluster_id:

# Datastore in which the Images uploaded through OCCI will be allocated, by default 1
#:datastore_id:

# VM types allowed and its template file (inside templates directory)
:instance_types:
  :small:
    :template: small.erb
    :cpu: 1
    :memory: 1024
  :medium:
    :template: medium.erb
```

```
    :cpu: 4
    :memory: 4096
  :large:
    :template: large.erb
    :cpu: 8
    :memory: 8192
```

Configuring OCCI Virtual Networks

You have to adapt the `/etc/one/occi_templates/network.erb` file to the configuration that the Virtual Networks created through the OCCI interface will use. For more information about the Virtual Network configuration check the following *guide*.

```
NAME = "<%= @vnet_info['NAME'] %>"
TYPE = RANGED

NETWORK_ADDRESS = <%= @vnet_info['ADDRESS'] %>
<% if @vnet_info['SIZE'] != nil %>
NETWORK_SIZE    = <%= @vnet_info['SIZE']%>
<% end %>

<% if @vnet_info['DESCRIPTION'] != nil %>
DESCRIPTION = "<%= @vnet_info['DESCRIPTION'] %>"
<% end %>

<% if @vnet_info['PUBLIC'] != nil %>
PUBLIC = "<%= @vnet_info['PUBLIC'] %>"
<% end %>

#BRIDGE = NAME_OF_DEFAULT_BRIDGE
#PHYDEV = NAME_OF_PHYSICAL_DEVICE
#VLAN   = YES|NO
```

Defining Compute Types

You can define as many Compute types as you want, just:

- Create a template (new_type.erb) for the new type and place it in `/etc/one/occi_templates`. This template will be *completed* with the data for each *occi-compute create* request and the content of the `/etc/one/occi_templates/common.erb` file, and then submitted to OpenNebula.

```
# This is the content of the new /etc/one/occi_templates/new_type.erb file
CPU    = 1
MEMORY = 512

OS = [ kernel     = /vmlinuz,
       initrd     = /initrd.img,
       root       = sda1,
       kernel_cmd = "ro xencons=tty console=tty1"]
```

- Add a new type in the instance_types section of the occi-server.conf

```
:new_type:
  :template: new_type.erb
  :cpu: 1
  :memory: 512
```

- You can add common attributes for your cloud templates modifying the `/etc/one/occi_templates/common.erb` file.

Warning: The templates are processed by the OCCI service to include specific data for the instance, you should not need to modify the <%= ... %> compounds inside the `common.erb` file.

7.3.4 Usage

Starting the Cloud Service

To start the OCCI service just issue the following command

```
occi-server start
```

You can find the OCCI server log file in `/var/log/one/occi-server.log`.

To stop the OCCI service:

```
occi-server stop
```

Warning: In order to start the OCCI server the `/var/lib/one/.one/occi_auth` file should be readable by the user that is starting the server and the serveradmin user must exist in OpenNebula

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 HTTP basic authentication with `user-ID` their OpenNebula's username and `password` their OpenNebula's 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
```

7.3.5 Tuning & Extending

Authorization Methods

OpenNebula OCCI Server supports two authorization methods in order to log in. The method can be set in the *occi-server.conf*, as explained above. These two methods are:

Basic Auth

In the basic mode, username and password(sha1) are matched to those in OpenNebula's database in order to authenticate the user in each request.

x509 Auth

This method performs the login to OpenNebula based on a x509 certificate DN (Distinguished Name). The DN is extracted from the certificate and matched to the password value in the user database (remember, spaces are removed from DNs).

The user password has to be changed running one of the following commands

```
oneuser chauth new_user x509 "/C=ES/O=ONE/OU=DEV/CN=clouduser"
oneuser chauth new_user --x509 --cert /tmp/my_cert.pem
```

or create a new user:

```
oneuser create new_user "/C=ES/O=ONE/OU=DEV/CN=clouduser" --driver x509
oneuser create new_user --x509 --cert /tmp/my_cert.pem
```

To enable this login method, set the `:auth:` option of `/etc/one/sunstone-server.conf` to `x509`:

```
:auth: x509
```

Note that OpenNebula will not verify that the user is holding a valid certificate at the time of login: this is expected to be done by the external container of the OCCI server (normally Apache), whose job is to tell the user's client that the site requires a user certificate and to check that the certificate is consistently signed by the chosen Certificate Authority (CA).

Configuring a SSL Proxy

OpenNebula OCCI 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 OCCI 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
- OCCI 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 OCCI Service.

Let's assume the server were 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 milleage 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/lighttp
```

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. OCCI Service Configuration

The `occi.conf` needs to define the following:

```
# Host and port where the occi server will run
:server: <FQDN OF OCCI SERVER>
:port: 4567

# SSL proxy that serves the API (set if is being used)
:ssl_server: https://localhost:443
```

Once the lighttpd server is started, OCCI 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.

7.4 OpenNebula OCCI User Guide

The OpenNebula OCCI API is a RESTful service to create, control and monitor cloud resources using an implementation of the OGF OCCI API specification based on the draft 0.8. This implementation also includes some extensions, requested by the community, to support OpenNebula specific functionality. Interactions with the resources are done through HTTP verbs (**GET**, **POST**, **PUT** and **DELETE**).

7.4.1 Commands

There are four kind of resources, listed below with their implemented actions:

- **Storage**:
  - `occi-storage list [-verbose]`
  - `occi-storage create xml_template`
  - `occi-storage update xml_template`
  - `occi-storage show resource_id`
  - `occi-storage delete resource_id`
- **Network**:
  - `occi-network list [-verbose]`
  - `occi-network create xml_template`
  - `occi-network update xml_template`
  - `occi-network show resource_id`
  - `occi-network delete resource_id`
- **Compute**:
  - `occi-compute list [-verbose]`
  - `occi-compute create xml_template`
  - `occi-compute update xml_template`
  - `occi-compute show resource_id`
  - `occi-compute delete resource_id`
  - `occi-compute attachdisk resource_id storage_id`
  - `occi-compute detachdisk resource_id storage_id`
- **Instance_type**:
  - `occi-instance-type list [-verbose]`
  - `occi-instance-type show resource_id`

7.4.2 User Account Configuration

An account is needed in order to use the OpenNebula OCCI 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 *managing users guide* to setup accounts, and automatically the OpenNebula OCCI 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 her environment, particularly the following aspects:

- **Authentication**: This can be achieved in two different ways, listed here in order of priority (i.e. values specified in the argument line supersede environmental variables)
  - Using the **commands arguments**. All the commands accept a **username** (as the OpenNebula username) and a **password** (as the OpenNebula password)
  - If the above is not available, the **ONE_AUTH** variable will be checked for authentication (with the same used for OpenNebula CLI, pointing to a file containing a single line: `username:password`).
- **Server location**: The command need to know where the OpenNebula OCCI service is running. You can pass the OCCI service endpoint using the `-url` flag in the commands. If that is not present, the **OCCI_URL** environment variable is used (in the form of a http URL, including the port if it is not the standard 80). Again, if the **OCCI_URL** variable is not present, it will default to `http://localhost:4567`

Warning: The **OCCI_URL** has to use the FQDN of the OCCI Service

7.4.3 Create Resources

Lets take a walk through a typical usage scenario. In this brief scenario it will be shown how to upload an image to the OCCI OpenNebula Storage repository, how to create a Network in the OpenNebula OCCI cloud and how to create Compute resource using the image and the network previously created.

- **Storage**

Assuming we have a working Ubuntu installation residing in an **.img** file, we can upload it into the OpenNebula OCCI cloud using the following OCCI representation of the image:

```
<STORAGE>
    <NAME>Ubuntu Desktop</NAME>
    <DESCRIPTION>Ubuntu 10.04 desktop for students.</DESCRIPTION>
    <TYPE>OS</TYPE>
    <URL>file:///images/ubuntu/jaunty.img</URL>
</STORAGE>
```

Next, using the **occi-storage** command we will create the Storage resource:

```
$ occi-storage --url http://cloud.server:4567 --username oneadmin --password opennebula create image
<STORAGE href='http://cloud.server:4567/storage/0'>
    <ID>3</ID>
    <NAME>Ubuntu Desktop</NAME>
    <TYPE>OS</TYPE>
    <DESCRIPTION>Ubuntu 10.04 desktop for students.</DESCRIPTION>
    <PUBLIC>NO</PUBLIC>
    <PERSISTENT>NO</PERSISTENT>
    <SIZE>41943040</SIZE>
</STORAGE>
```

The user should take note of this **ID**, as it will be needed to add it to the Compute resource.

- **Network**

The next step would be to create a Network resource

```
<NETWORK>
    <NAME>MyServiceNetwork</NAME>
    <ADDRESS>192.168.1.1</ADDRESS>
    <SIZE>200</SIZE>
    <PUBLIC>NO</PUBLIC>
</NETWORK>
```

Next, using the **occi-network** command we will create the Network resource:

```
$ occi-network --url http://cloud.server:4567 --username oneadmin --password opennebula create vnet.x
<NETWORK href='http://cloud.server:4567/network/0'>
    <ID>0</ID>
    <NAME>MyServiceNetwork</NAME>
    <ADDRESS>192.168.1.1/ADDRESS>
    <SIZE>200/SIZE>
    <PUBLIC>NO</PUBLIC>
</NETWORK>
```

- **Compute**

The last step would be to create a Compute resource referencing the Storage and Networks resource previously created by means of their ID, using a representation like the following:

```
<COMPUTE>
    <NAME>MyCompute</NAME>
    <INSTANCE_TYPE href="http://www.opennebula.org/instance_type/small"/>
    <DISK>
        <STORAGE href="http://www.opennebula.org/storage/0"/>
    </DISK>
    <NIC>
        <NETWORK href="http://www.opennebula.org/network/0"/>
        <IP>192.168.1.12</IP>
    </NIC>
    <CONTEXT>
        <HOSTNAME>MAINHOST</HOSTNAME>
        <DATA>DATA1</DATA>
    </CONTEXT>
</COMPUTE>
```

Next, using the **occi-compute** command we will create the Compute resource:

```
$ occi-compute --url http://cloud.server:4567 --username oneadmin --password opennebula create vm.xml
<COMPUTE href='http://cloud.server:4567/compute/0'>
  <ID>0</ID>
  <CPU>1</CPU>
  <MEMORY>1024</MEMORY>
  <NAME>MyCompute</NAME>
  <INSTANCE_TYPE href="http://www.opennebula.org/instance_type/small"/>
  <STATE>PENDING</STATE>
  <DISK id='0'>
    <STORAGE href='http://cloud.server:4567/storage/3' name='Ubuntu Desktop'/>
    <TYPE>DISK</TYPE>
    <TARGET>hda</TARGET>
  </DISK>
  <NIC>
    <NETWORK href='http://cloud.server:4567/network/0' name='MyServiceNetwork'/>
    <IP>192.168.1.12</IP>
```

```
    <MAC>02:00:c0:a8:01:0c</MAC>
  </NIC>
  <CONTEXT>
    <DATA>DATA1</DATA>
    <HOSTNAME>MAINHOST</HOSTNAME>
    <TARGET>hdb</TARGET>
  </CONTEXT>
</COMPUTE>
```

7.4.4 Updating Resources

Storage

Some of the characteristics of an storage entity can be modified using the occi-storage update command:

> **Warning:** Only one characteristic can be updated per request

Storage Persistence

In order to make a storage entity persistent we can update the resource using the following xml:

```
<STORAGE href='http://cloud.server:4567/storage/0'>
    <ID>3</ID>
    <PERSISTENT>YES</PERSISTENT>
</STORAGE>
```

Next, using the **occi-storage** command we will create the Storage resource:

```
$ occi-storage --url http://cloud.server:4567 --username oneadmin --password opennebula update image
<STORAGE href='http://cloud.server:4567/storage/0'>
    <ID>3</ID>
    <NAME>Ubuntu Desktop</NAME>
    <TYPE>OS</TYPE>
    <DESCRIPTION>Ubuntu 10.04 desktop for students.</DESCRIPTION>
    <PUBLIC>NO</PUBLIC>
    <PERSISTENT>YES</PERSISTENT>
    <SIZE>41943040</SIZE>
</STORAGE>
```

Publish a Storage

In order to publish a storage entity so that other users can use it, we can update the resource using the following xml:

```
<STORAGE href='http://cloud.server:4567/storage/0'>
    <ID>3</ID>
    <PUBLIC>YES</PUBLIC>
</STORAGE>
```

Next, using the **occi-storage** command we will create the Storage resource:

```
$ occi-storage --url http://cloud.server:4567 --username oneadmin --password opennebula update image
<STORAGE href='http://cloud.server:4567/storage/0'>
    <ID>3</ID>
```

```
    <NAME>Ubuntu Desktop</NAME>
    <TYPE>OS</TYPE>
    <DESCRIPTION>Ubuntu 10.04 desktop for students.</DESCRIPTION>
    <PUBLIC>YES</PUBLIC>
    <PERSISTENT>YES</PERSISTENT>
    <SIZE>41943040</SIZE>
</STORAGE>
```

Network

Some of the characteristics of an network entity can be modified using the occi-network update command:

Warning: Only one characteristic can be updated per request

Publish a Network

In order to publish a network entity so that other users can use it, we can update the resource using the following xml:

```
<NETWORK href='http://cloud.server:4567/network/0'>
    <ID>0</ID>
    <PUBLIC>YES</PUBLIC>
</NETWORK>
```

Next, using the **occi-network** command we will update the Network resource:

```
$ occi-network --url http://cloud.server:4567 --username oneadmin --password opennebula update vnet.x
<NETWORK href='http://cloud.server:4567/network/0'>
    <ID>0</ID>
    <NAME>MyServiceNetwork</NAME>
    <ADDRESS>192.168.1.1/ADDRESS>
    <SIZE>200/SIZE>
    <PUBLIC>YES</PUBLIC>
</NETWORK>
```

Compute

Some of the characteristics of a compute entity can be modified using the occi-compute update command:

Warning: Only one characteristic can be updated per request

Change the Compute State

In order to change the Compute state, we can update the resource using the following xml:

```
<COMPUTE href='http://cloud.server:4567/compute/0'>
  <ID>0</ID>
  <STATE>STOPPED</STATE>
</COMPUTE>
```

Next, using the **occi-compute** command we will update the Compute resource:

The available states to update a Compute resource are:

- STOPPED
- SUSPENDED
- RESUME
- CANCEL
- SHUTDOWN
- REBOOT
- RESET
- DONE

Save a Compute Disk in a New Storage

In order to save a Compute disk in a new image, we can update the resource using the following xml. The disk will be saved after shutting down the Compute.

```
<COMPUTE href='http://cloud.server:4567/compute/0'>
  <ID>0</ID>
  <DISK id="0">
    <STORAGE href="http://cloud.server:4567/storage/0" name="first_image"/>
    <SAVE_AS name="save_as1"/>
  </DISK>
</COMPUTE>
```

Next, using the **occi-compute** command we will update the Compute resource:

```
$ occi-compute --url http://cloud.server:4567 --username oneadmin --password opennebula update vm.xml
<COMPUTE href='http://cloud.server:4567/compute/0'>
  <ID>0</ID>
  <CPU>1</CPU>
  <MEMORY>1024</MEMORY>
  <NAME>MyCompute</NAME>
  <INSTANCE_TYPE>small</INSTANCE_TYPE>
  <STATE>STOPPED</STATE>
  <DISK id='0'>
    <STORAGE href='http://cloud.server:4567/storage/3' name='Ubuntu Desktop'/>
    <SAVE_AS href="http://cloud.server:4567/storage/7"/>
    <TYPE>DISK</TYPE>
    <TARGET>hda</TARGET>
  </DISK>
  <NIC>
    <NETWORK href='http://cloud.server:4567/network/0' name='MyServiceNetwork'/>
    <IP>192.168.1.12</IP>
    <MAC>02:00:c0:a8:01:0c</MAC>
  </NIC>
  <CONTEXT>
    <DATA>DATA1</DATA>
    <HOSTNAME>MAINHOST</HOSTNAME>
    <TARGET>hdb</TARGET>
  </CONTEXT>
</COMPUTE>
```

Create a Volume and Attach It to a Running VM

In this example we will show how to create a new volume using the following template and attach it to a running compute resource.

```
<STORAGE>
  <NAME>Volume1</NAME>
  <TYPE>DATABLOCK</TYPE>
  <DESCRIPTION>Volume to be hotplugged</DESCRIPTION>
  <PUBLIC>NO</PUBLIC>
  <PERSISTENT>NO</PERSISTENT>
  <FSTYPE>ext3</FSTYPE>
  <SIZE>10</SIZE>
</STORAGE>

$ cat /tmp/storage
<STORAGE>
    <NAME>Volume1</NAME>
    <TYPE>DATABLOCK</TYPE>
    <DESCRIPTION>Volume to be hotplugged</DESCRIPTION>
    <PUBLIC>NO</PUBLIC>
    <PERSISTENT>NO</PERSISTENT>
    <FSTYPE>ext3</FSTYPE>
    <SIZE>10</SIZE>
</STORAGE>

$ occi-storage create /tmp/storage
<STORAGE href='http://127.0.0.1:4567/storage/5'>
  <ID>5</ID>
  <NAME>Volume1</NAME>
  <USER href='http://127.0.0.1:4567/user/0' name='oneadmin'/>
  <GROUP>oneadmin</GROUP>
  <STATE>READY</STATE>
  <TYPE>DATABLOCK</TYPE>
  <DESCRIPTION>Volume to be hotplugged</DESCRIPTION>
  <SIZE>10</SIZE>
  <FSTYPE>ext3</FSTYPE>
  <PUBLIC>NO</PUBLIC>
  <PERSISTENT>NO</PERSISTENT>
</STORAGE>

$ occi-compute list
<COMPUTE_COLLECTION>
  <COMPUTE href='http://127.0.0.1:4567/compute/4' name='one-4'/>
  <COMPUTE href='http://127.0.0.1:4567/compute/6' name='one-6'/>
</COMPUTE_COLLECTION>

$ occi-storage list
<STORAGE_COLLECTION>
  <STORAGE name='ttylinux - kvm' href='http://127.0.0.1:4567/storage/1'/>
  <STORAGE name='Ubuntu Server 12.04 (Precise Pangolin) - kvm' href='http://127.0.0.1:4567/storage/2'
  <STORAGE name='Volume1' href='http://127.0.0.1:4567/storage/5'/>
</STORAGE_COLLECTION>

$ occi-compute attachdisk 6 5
<COMPUTE href='http://127.0.0.1:4567/compute/6'>
  <ID>6</ID>
  <USER name='oneadmin' href='http://127.0.0.1:4567/user/0'/>
```

```
  <GROUP>oneadmin</GROUP>
  <CPU>1</CPU>
  <MEMORY>512</MEMORY>
  <NAME>one-6</NAME>
  <STATE>ACTIVE</STATE>
  <DISK id='0'>
    <STORAGE name='Ubuntu Server 12.04 (Precise Pangolin) - kvm' href='http://127.0.0.1:4567/storage/
    <TYPE>FILE</TYPE>
    <TARGET>hda</TARGET>
  </DISK>
  <DISK id='1'>
    <STORAGE name='Volume1' href='http://127.0.0.1:4567/storage/5'/>
    <TYPE>FILE</TYPE>
    <TARGET>sda</TARGET>
  </DISK>
  <NIC>
    <NETWORK name='local-net' href='http://127.0.0.1:4567/network/0'/>
    <IP>192.168.122.6</IP>
    <MAC>02:00:c0:a8:7a:06</MAC>
  </NIC>
</COMPUTE>
```

Warning: You can obtain more information on how to use the above commands accessing their Usage help passing them the **-h** flag. For instance, a -T option is available to set a connection timeout.

Warning: In platforms where 'curl' is not available or buggy (i.e. CentOS), a '-M' option is available to perform upload using the native ruby Net::HTTP using http multipart

7.5 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.

7.5.1 AMIs

- **upload image**: Uploads an image to OpenNebula
- **describe images**: Lists all registered images belonging to one particular user.

7.5.2 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 ofvirtual machines (or cancel, depending on its state).
- **reboot instances**: Reboots a set ofvirtual machines.

- **start instances**: Starts a set ofvirtual machines.
- **stop instances**: Stops a set ofvirtual machines.

7.5.3 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 snpahost**:

7.5.4 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**: Disasociate a publicIP of the user currently associated with an instance

7.5.5 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

7.5.6 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.

Todo

What configuration guide

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).

Warning: The `EC2_URL` 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
```

- **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

7.6 EC2 Ecosystem

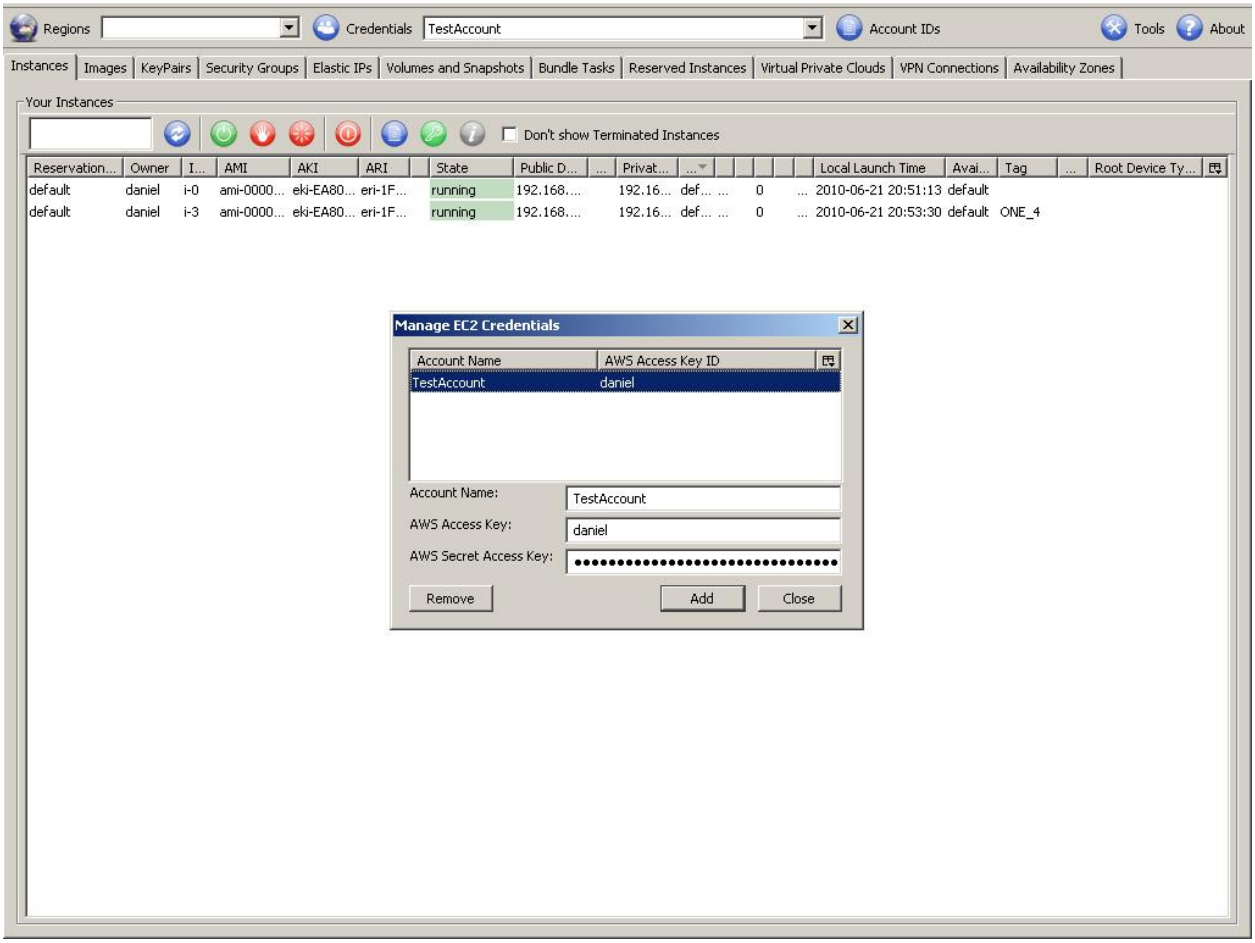
In order to interact with the EC2 Service that OpenNebula implements you can use the client included in the OpenNebula distribution, but also you can choose one of the well known tools that are supposed to interact with cloud servers through the EC2 Query API, like the Firefox extension HybridFox, or the command line tools, Euca2ools.

7.6.1 HybridFox

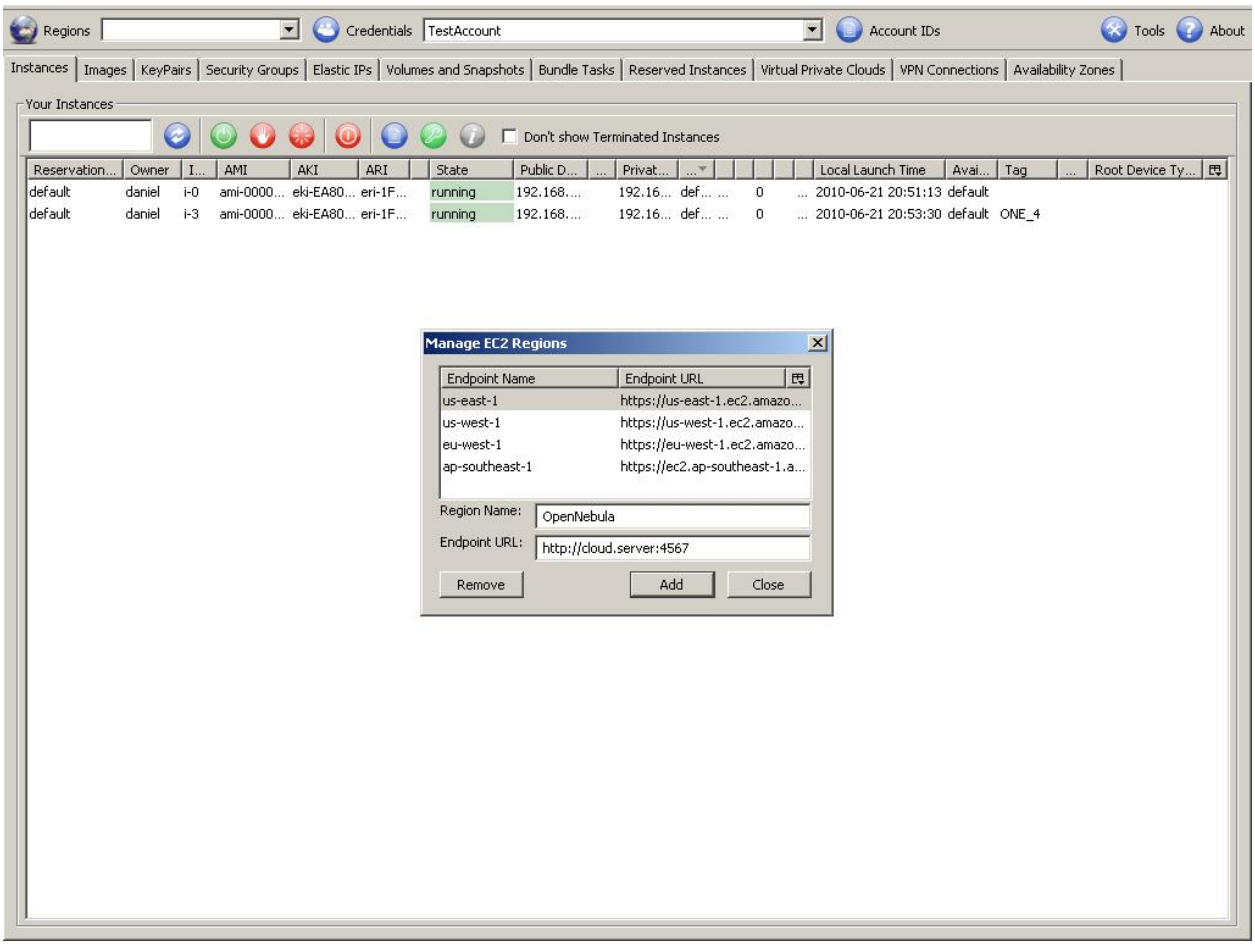
HybridFox is a Mozilla Firefox extension for managing your Amazon EC2 account. Launch new instances, mount Elastic Block Storage volumes, map Elastic IP addresses, and more.

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



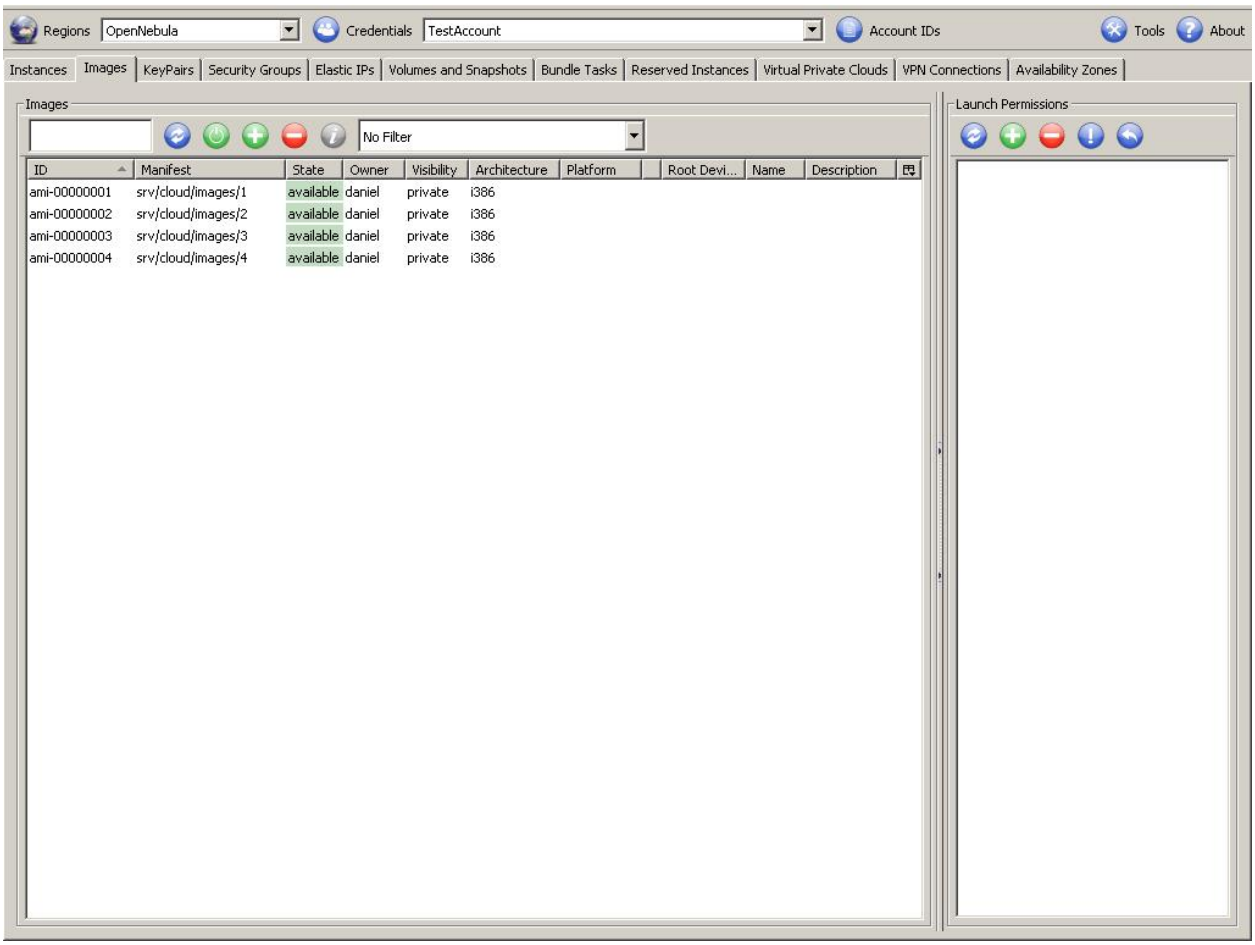
- Also you have to specify in a new `Region` the endpoint in which the EC2 Service is running, by pressing on the Regions button. Take care of using exactly the same url and port that is specified in the econe.conf file, otherwise you will get `AuthFailure`:



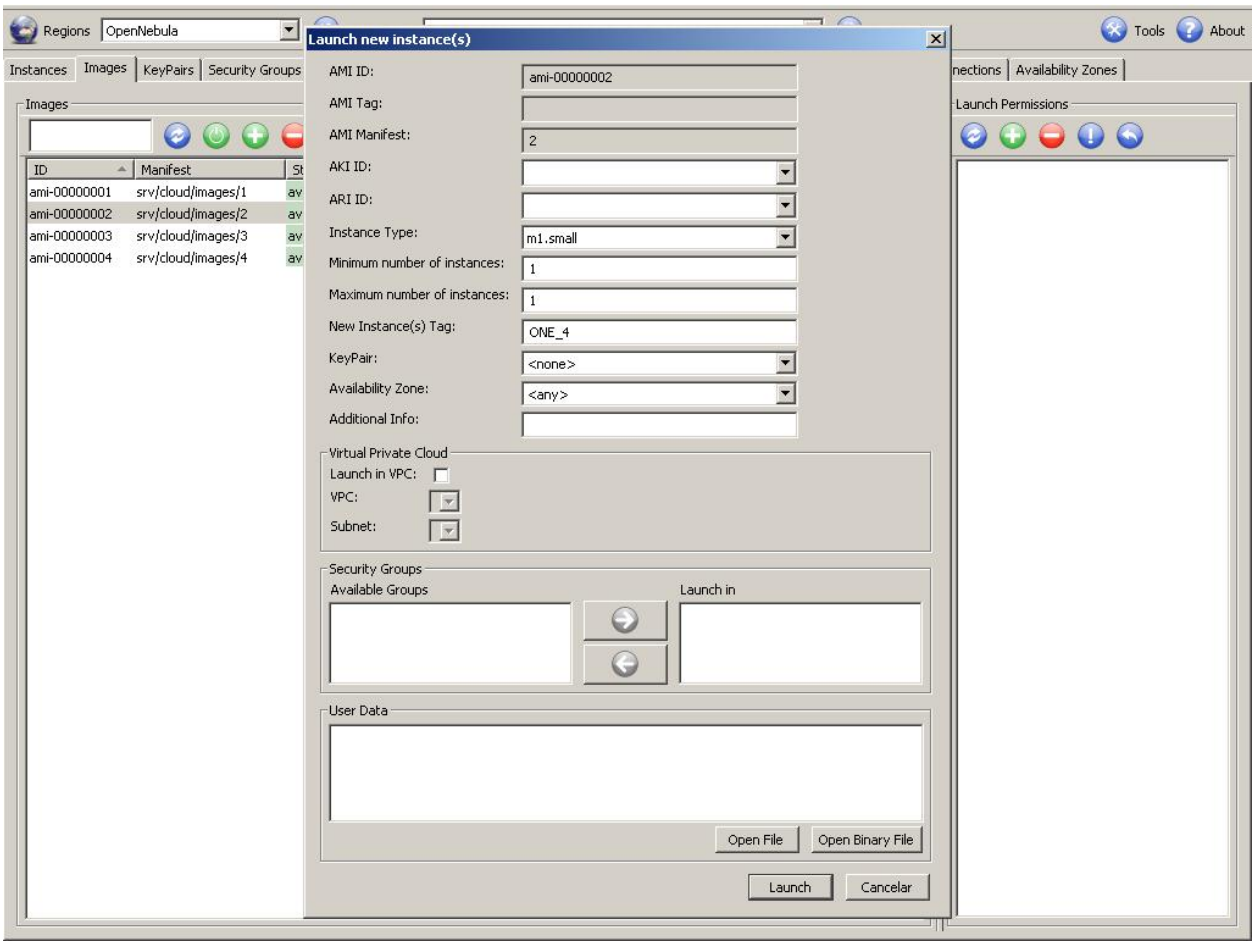
Warning: If you have problems adding a new region, try to add it manually in the ec2ui.endpoints variable inside the Firefox about:config

Typical usage scenarios

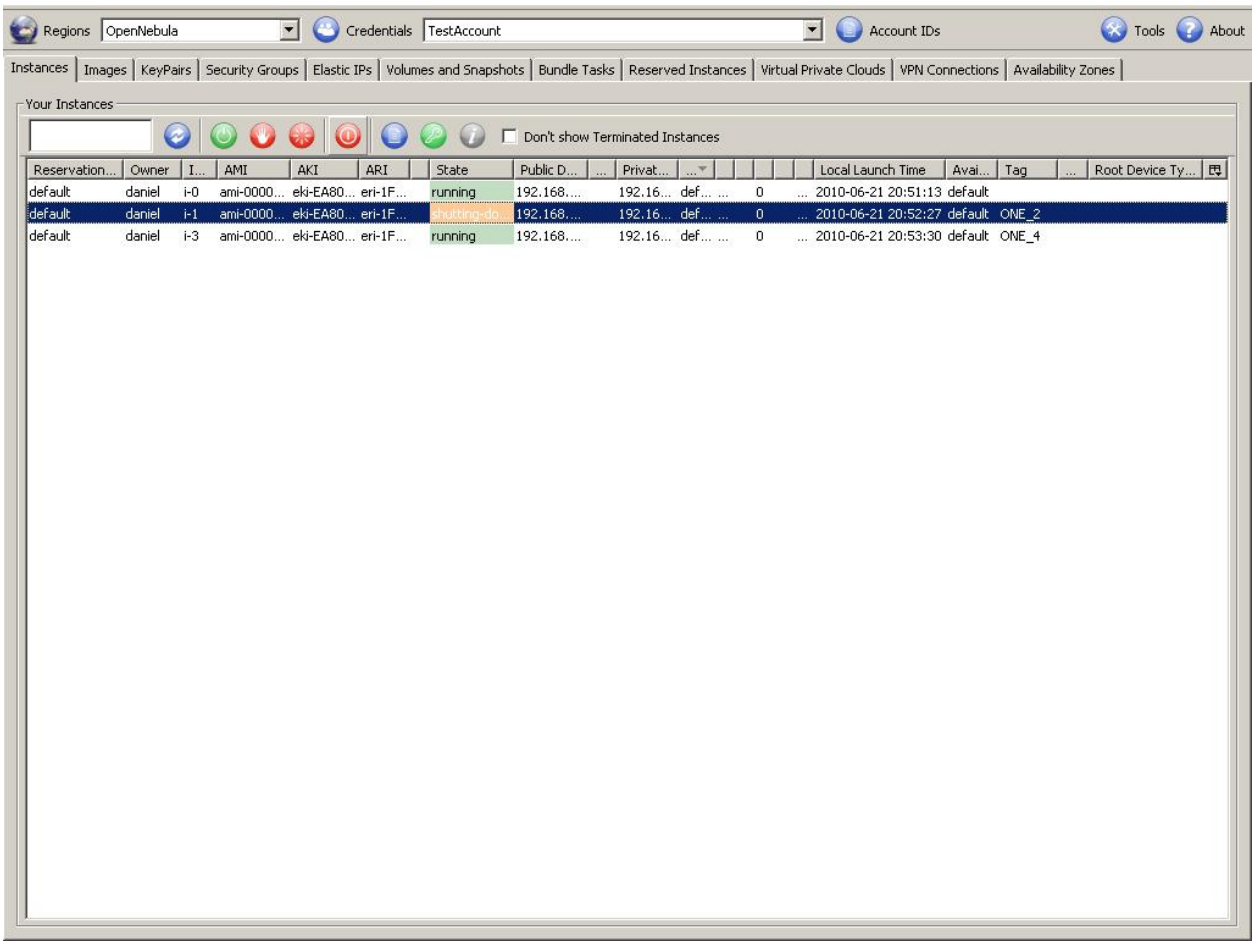
- **List images**



- **Run instances**



- **Control instances**



You can also use HybridFox a similar Mozilla Firefox extension to interact with cloud services through the EC2 Query API

7.6.2 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   N
```