OpenNebula 5.2 Advanced Components Guide

Release 5.2.1

OpenNebula Systems

Jan 09, 2017

THE DOCUMENT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE DOCUMENT.

CONTENTS

1 Multi-VM Applications and Auto-scaling . . . 1
- 1.1 Overview . . . 1
- 1.2 OneFlow Server Configuration . . . 2
- 1.3 OneFlow Services Management . . . 5
- 1.4 OneFlow Services Auto-scaling . . . 20

2 Host and VM High Availability . . . 28
- 2.1 Overview . . . 28
- 2.2 Front-end HA Setup . . . 28
- 2.3 Virtual Machines High Availability . . . 38

3 Data Center Federation . . . 42
- 3.1 Overview . . . 42
- 3.2 OpenNebula Federation Configuration . . . 43
- 3.3 OpenNebula Federation Management . . . 51

4 Cloud Bursting . . . 54
- 4.1 Overview . . . 54
- 4.2 Amazon EC2 Driver . . . 55
- 4.3 Azure Driver . . . 62

5 Application Insight . . . 71
- 5.1 Overview . . . 71
- 5.2 OneGate Server Configuration . . . 72
- 5.3 OneGate Usage . . . 76

6 Public Cloud . . . 86
- 6.1 Overview . . . 86
- 6.2 EC2 Server Configuration . . . 86
- 6.3 OpenNebula EC2 User Guide . . . 93

7 MarketPlace . . . 97
- 7.1 Overview . . . 97
- 7.2 OpenNebula Systems MarketPlace . . . 99
- 7.3 HTTP MarketPlace . . . 100
- 7.4 S3 MarketPlace . . . 101
- 7.5 Managing MarketPlaceApps . . . 103

CHAPTER

ONE

MULTI-VM APPLICATIONS AND AUTO-SCALING

1.1 Overview

OneFlow is an OpenNebula component that allows users and administrators to define, execute and manage multi-tiered 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.

The benefits of OneFlow are:

- 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

1.1.1 How Should I Read This Chapter

This chapter should be read after the infrastructure is properly setup, and contains working Virtual Machine templates.

Proceed to each section following these links:

- *OneFlow Server Configuration*
- *OneFlow Services Management*
- *OneFlow Services Auto-scaling*

1.1.2 Hypervisor Compatibility

This chapter applies to all the hypervisors.

1.2 OneFlow Server Configuration

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

1.2.1 Installation

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

Note: Make sure you executed `install_gems` during the installation to install the required gems, in particular: `treetop`, `parse-cron`.

1.2.2 Configuration

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

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

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

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

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

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

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

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

1.2.3 Start OneFlow

To start and stop the server, use the `opennebula-flow` service:

```
# service opennebula-flow start
```

Note: 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 Set the Environment Variables

By default the command line tools will use the `one_auth` file and the `http://localhost:2474` OneFlow URL. To change it, set the shell environment variables as explained in the Managing Users documentation.

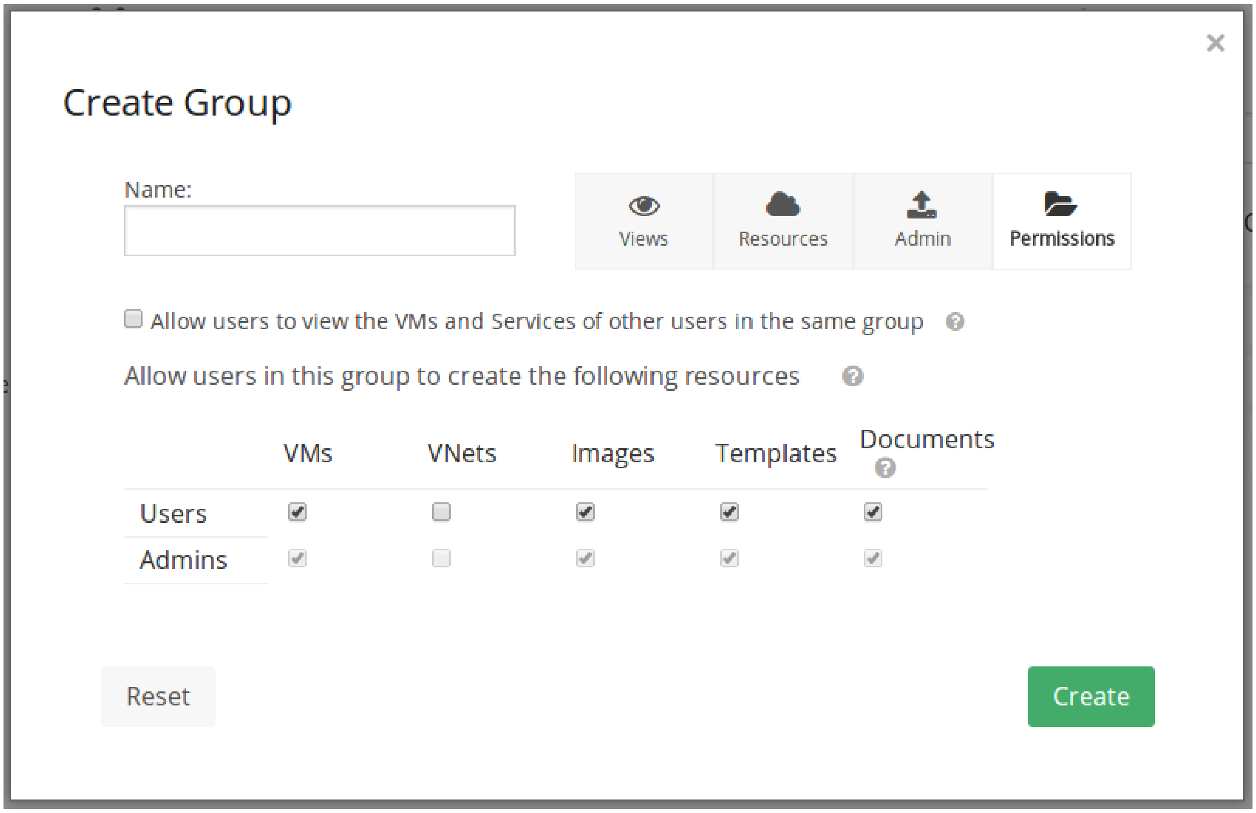
1.2.5 Enable the Sunstone Tabs

The OneFlow tabs (Services and Service Templates) are visible in Sunstone by default. To customize its visibility for each kind of user, visit the Sunstone views documentation

1.2.6 Advanced Setup

Permission to Create Services

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



Create Group

Name:

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

Allow users in this group to create the following resources

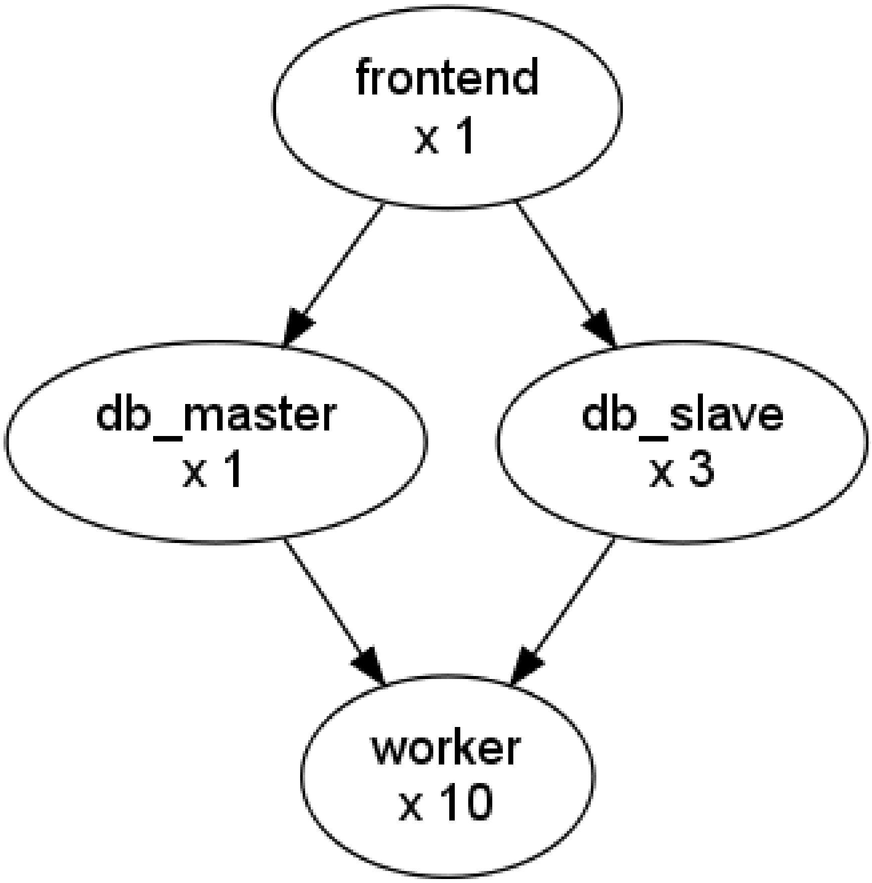
	VMs	VNets	Images	Templates	Documents
Users	☑	☐	☑	☑	☑
Admins	☑	☐	☑	☑	☑

1.3 OneFlow Services Management

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

1.3.1 What Is a Service

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



This Service can be represented with the following JSON template:

```
{
  "name": "my_service",
  "deployment": "straight",
  "ready_status_gate": true|false,
  "roles": [
    {
      "name": "frontend",
      "vm_template": 0
    },
    {
      "name": "db_master",
      "parents": [
        "frontend"
      ],
      "vm_template": 1
    },
    {
      "name": "db_slave",
      "parents": [
        "frontend"
      ],
      "cardinality": 3,
      "vm_template": 2
    },
    {
      "name": "worker",
      "parents": [
        "db_master",
        "db_slave"
      ],
      "cardinality": 10,
      "vm_template": 3
    }
  ]
}
```

1.3.2 Managing Service Templates

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

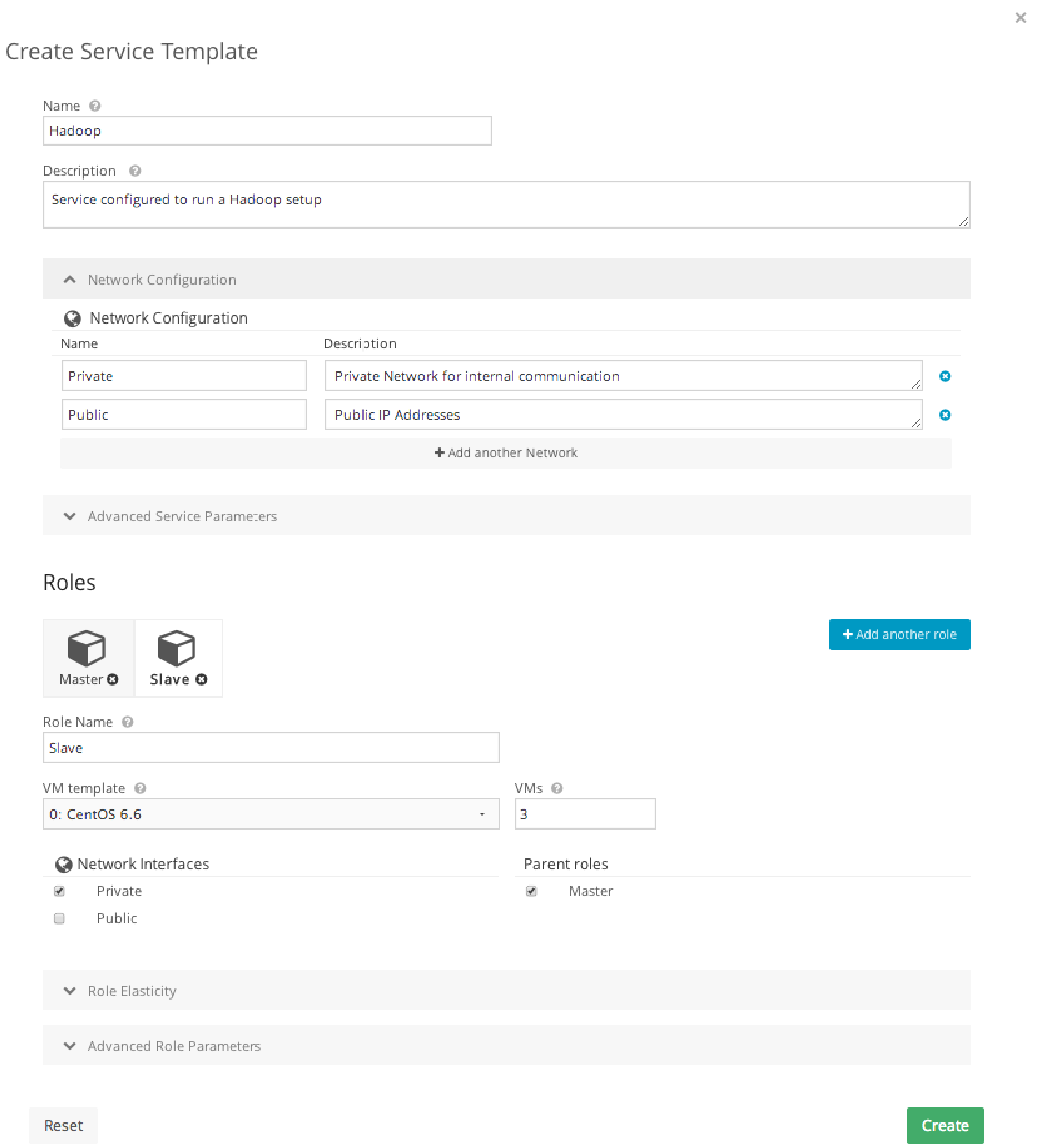
Users can manage the Service Templates using the command `oneflow-template`, or Sunstone. For each user, the actual list of Service Templates available is determined by the ownership and permissions of the Templates.

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 Templates from Sunstone:



Create Service Template

Name: Hadoop

Description: Service configured to run a Hadoop setup

Network Configuration

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

Roles

Role Name: Slave

VM template: 0: CentOS 6.6

VMs: 3

Network Interfaces

☑ Private

☐ Public

Parent roles

☑ Master

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

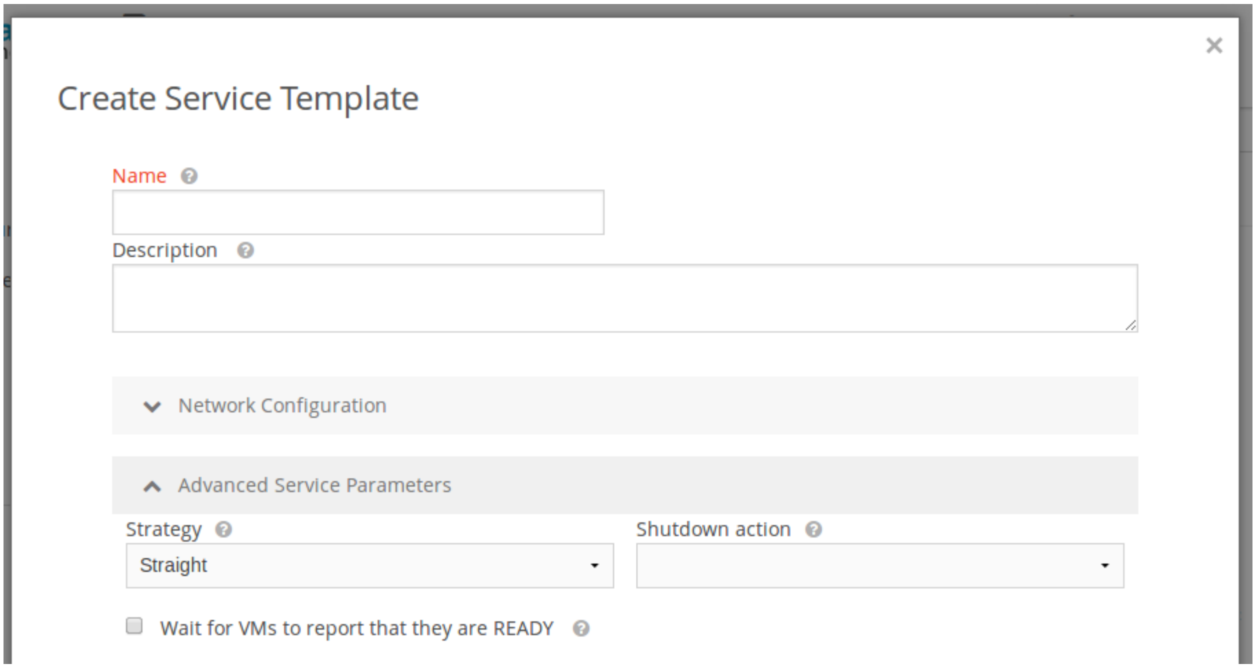
```
$ oneflow-template list
        ID USER            GROUP           NAME
         0 oneadmin        oneadmin        my_service

$ 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`.

Determining when a VM is READY

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



Create Service Template

Name

Description

Strategy: Straight

Shutdown action:

□ Wait for VMs to report that they are READY

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

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

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

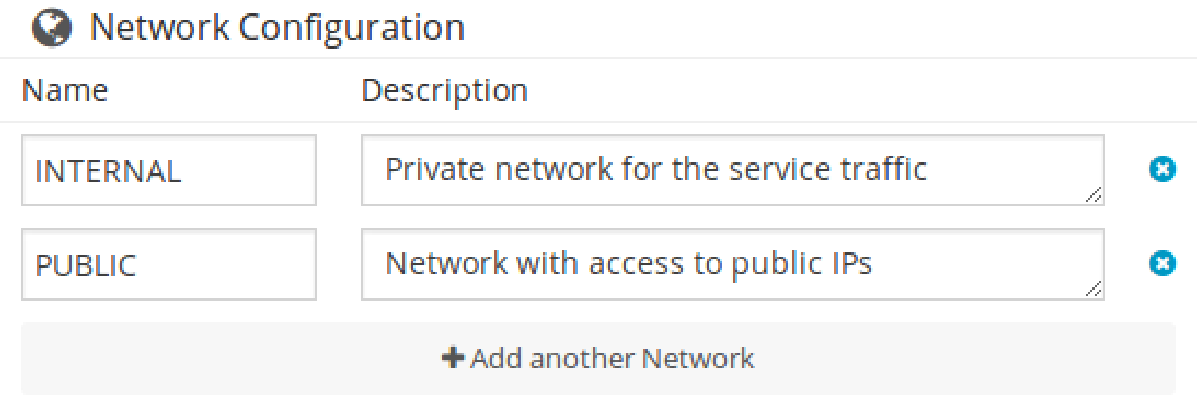
```
curl -X "PUT" http://<onegate>/vm \
  --header "X-ONEGATE-TOKEN: ..." \
  --header "X-ONEGATE-VMID: ..." \
  -d "READY = YES"
```

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

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

Configure Dynamic Networks

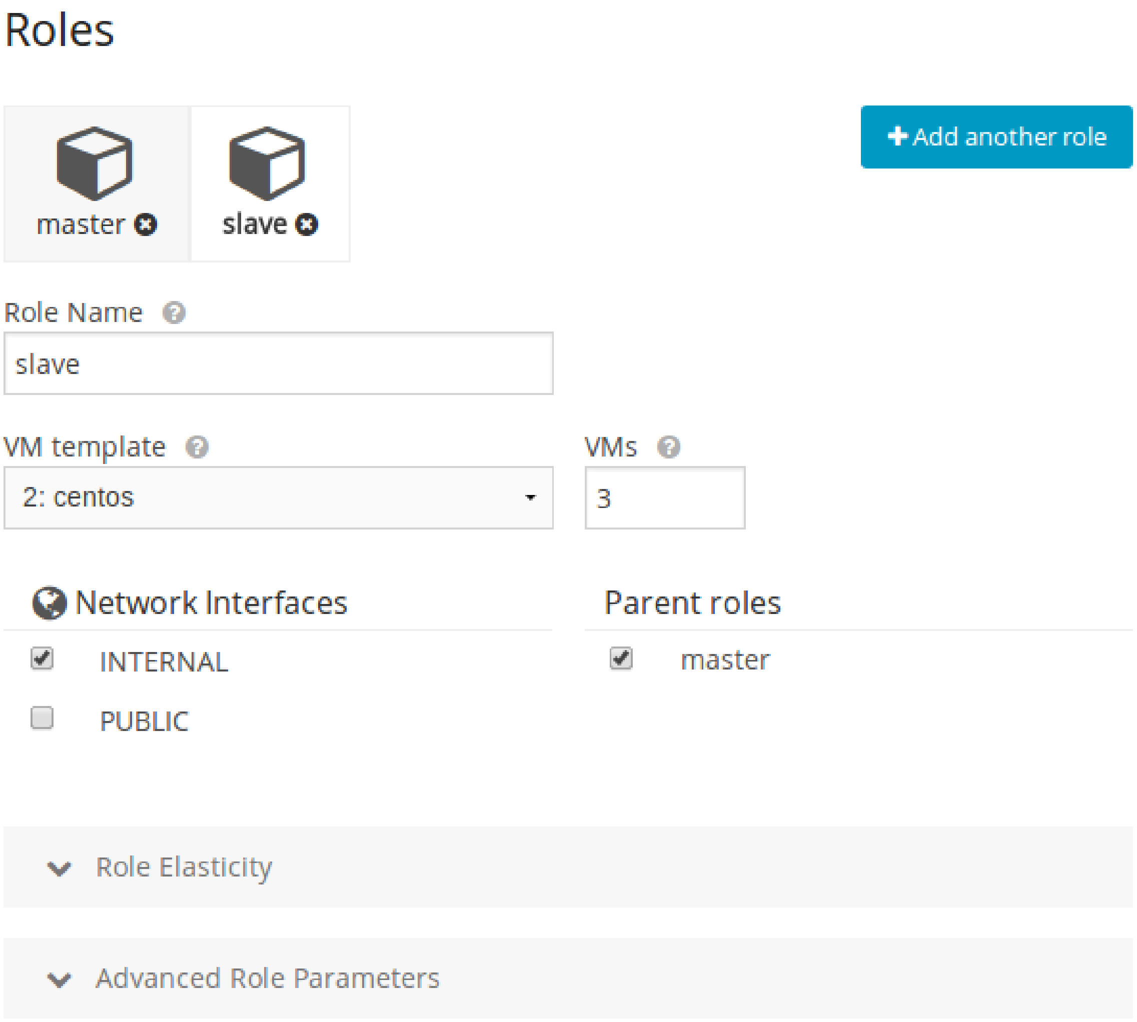
Each Service Role has a Virtual Machine Template assigned. The VM Template will define the capacity, disks, and network interfaces. But instead of using the Virtual Networks set in the VM Template, the Service Template can define a set of dynamic networks.



Network Configuration

Name	Description
INTERNAL	Private network for the service traffic
PUBLIC	Network with access to public IPs

Each Role can be attached to the dynamic networks individually.



Role Name: slave

VM template: 2: centos

VMs: 3

Network Interfaces

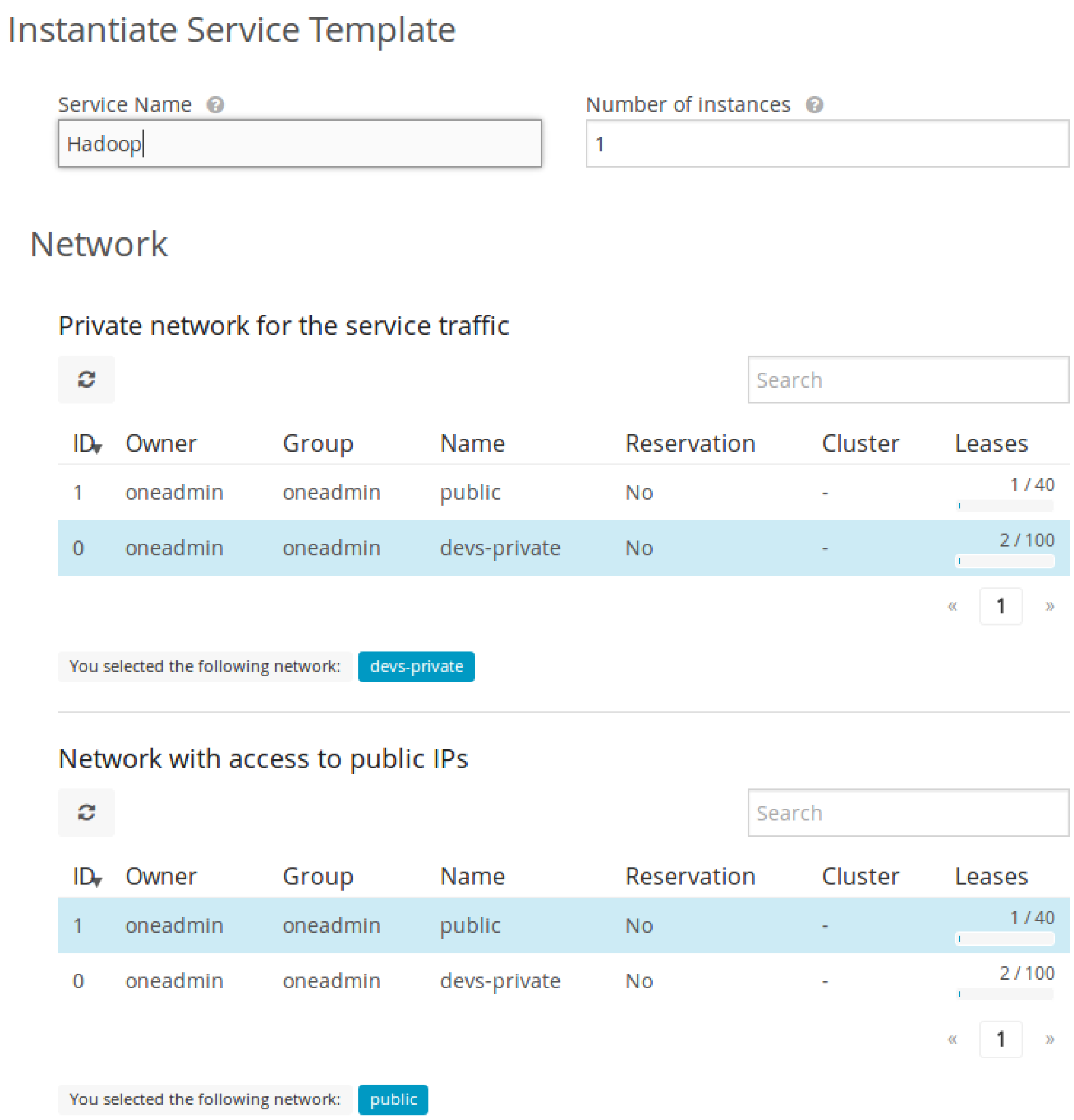
☑ INTERNAL

□ PUBLIC

Parent roles

☑ master

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



Instantiate Service Template

Service Name: Hadoop

Number of instances: 1

Network

Private network for the service traffic

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

You selected the following network: devs-private

Network with access to public IPs

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

You selected the following network: public

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

1.3.3 Managing Services

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

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

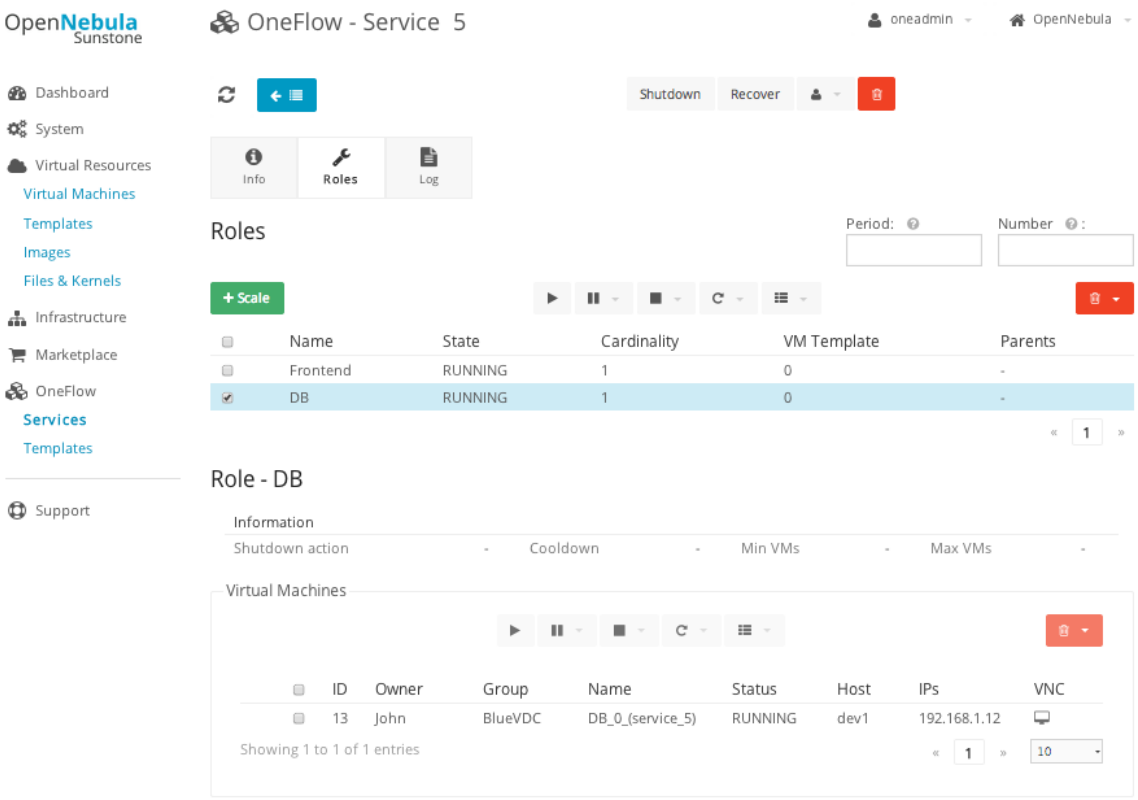
Create and List Existing Services

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



Roles

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

Role - DB

Information

Shutdown action	-	Cooldown	-	Min VMs	-	Max VMs	-

☐	ID	Owner	Group	Name	Status	Host	IPs	VNC
☐	13	John	BlueVDC	DB_0_(service_5)	RUNNING	dev1	192.168.1.12	🖵

Showing 1 to 1 of 1 entries

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
```

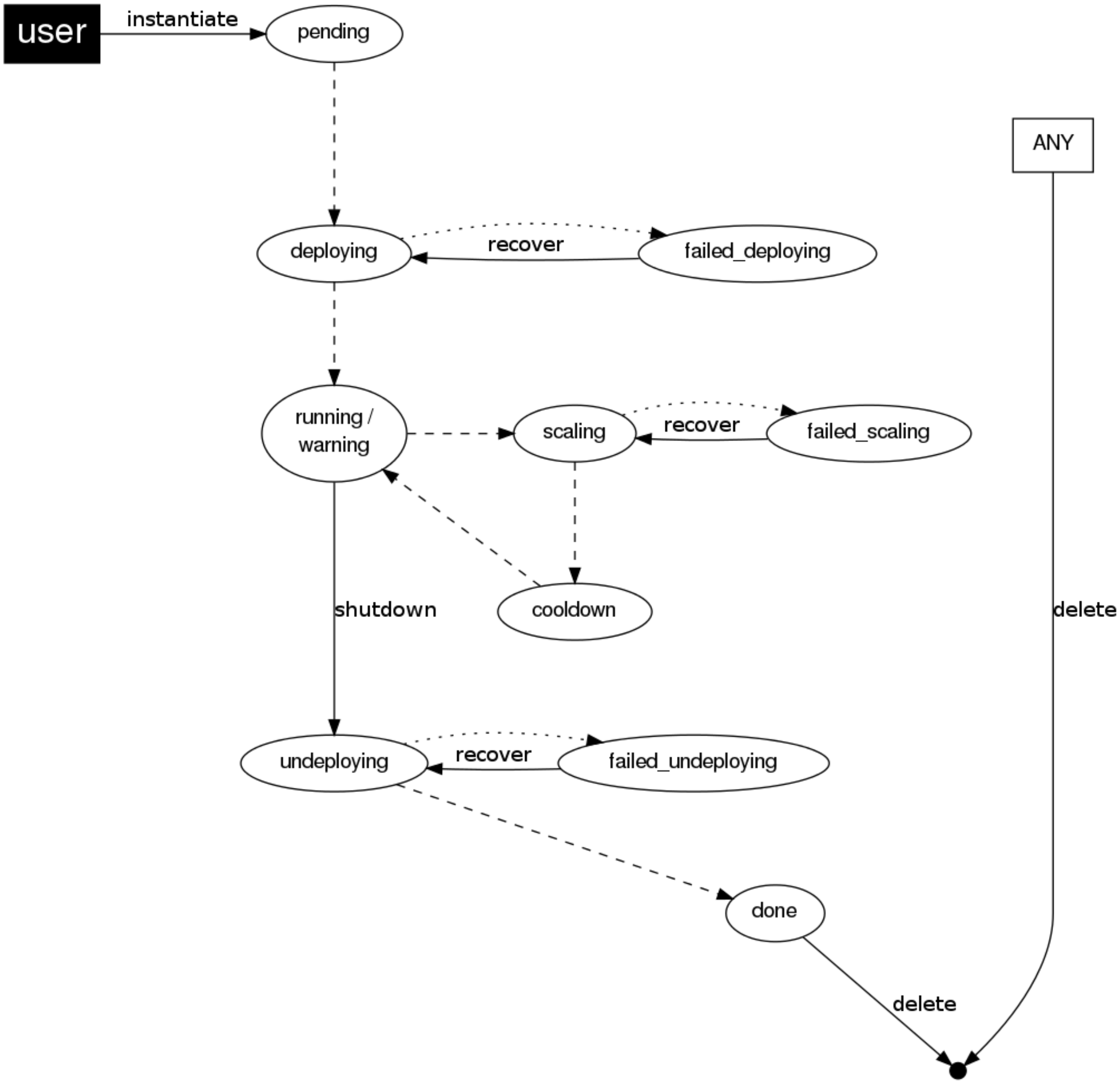
```
VM TEMPLATE         : 0
NODES INFORMATION
 VM_ID NAME                    STAT UCPU    UMEM HOST                       TIME
     0 frontend_0_(service_1)  runn   67  120.3M localhost              0d 00h01

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 can use the commands oneflow shutdown and oneflow delete.

The command oneflow shutdown will perform a graceful a terminate on all the running VMs (see onevm terminate). 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 terminate 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 terminate operation for each VM and delete the Service. Please be aware that **this is not recommended**, because failed terminate actions may leave VMs in the system.

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*.

Sharing Information between VMs

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

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

You can read more details in the *OneGate API documentation*.

1.3.4 Managing Permissions

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

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

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

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

$ oneflow chown my_service johndoe apptools

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

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

Note: 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:

- `terminate`
- `terminate-hard`
- `undeploy`
- `undeploy-hard`
- `hold`
- `release`
- `stop`

- `suspend`
- `resume`
- `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 `/etc/one/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 terminated. VMs in `UNKNOWN` are booted.
`FAILED_UNDEPLOYING`	`UNDEPLOYING`	The undeployment is resumed.
`FAILED_SCALING`	`SCALING`	VMs in `DONE` or `FAILED` are terminated. 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 terminated. 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 OneFlow Services Auto-scaling

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

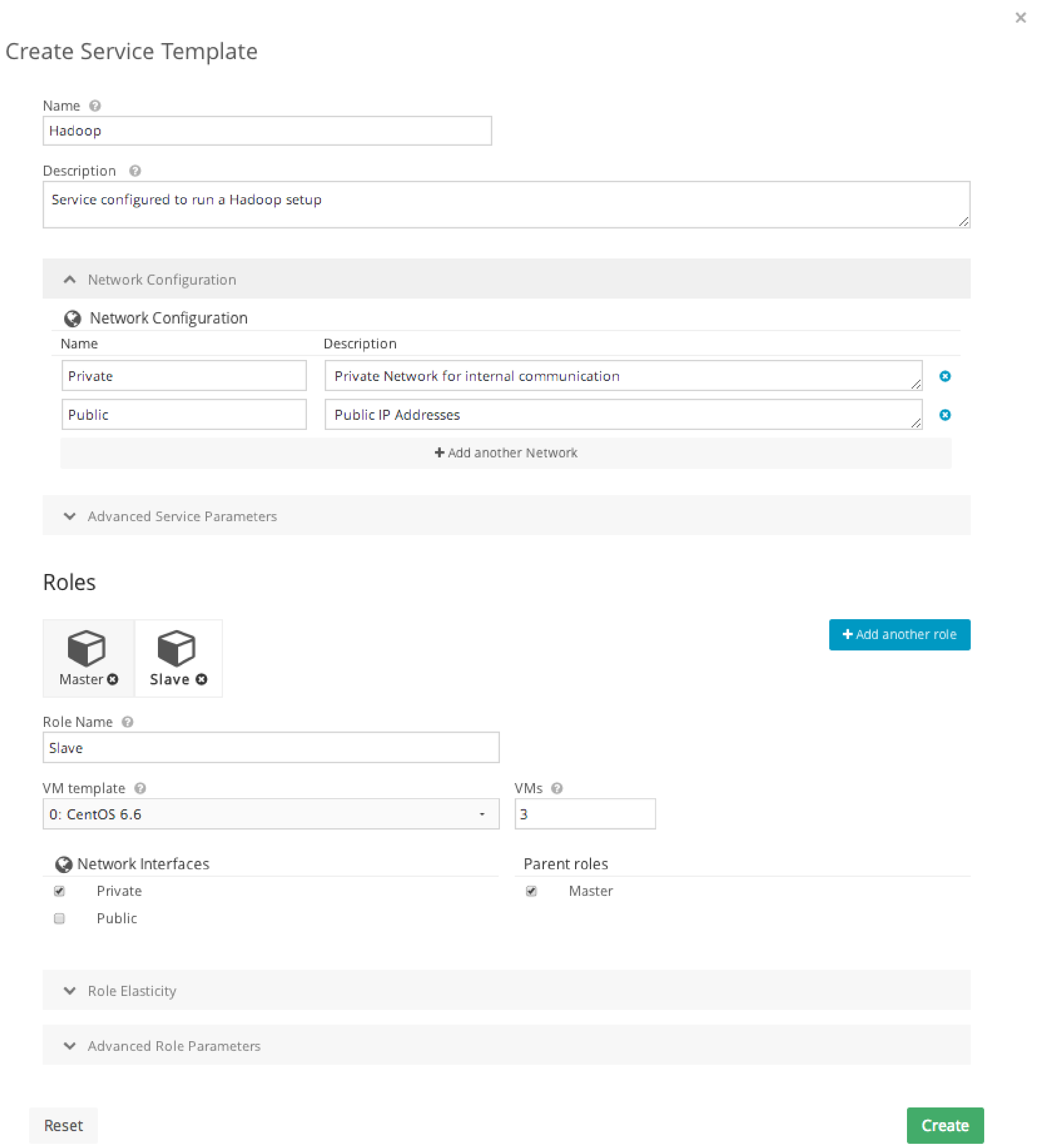
1.4.1 Overview

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

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

```
"roles": [
    {
      "name": "frontend",
      "cardinality": 1,
      "vm_template": 0,

      "min_vms" : 1,
      "max_vms" : 5,
...
```

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



Create Service Template

Name: Hadoop

Description: Service configured to run a Hadoop setup

Network Configuration

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

Role Name: Slave

VM template: 0: CentOS 6.6

VMs: 3

Network Interfaces

☑ Private

□ Public

Parent roles

☑ Master

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/subtract the given number of VMs
- **CARDINALITY**: Set the cardinality to the given number
- **PERCENTAGE_CHANGE**: Add/subtract the given percentage to the current cardinality

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

Auto-scaling Based on Metrics

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

These expressions can use performance data from

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

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

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

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

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

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

Auto-scaling Based on a Schedule

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

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

    "type" : "CARDINALITY",
    "adjust" : 2
  },
  {
    // +10 percent at the given date and time
    "start_time" : "2nd oct 2017 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 terminated can be identified by an arrow next to their ID:

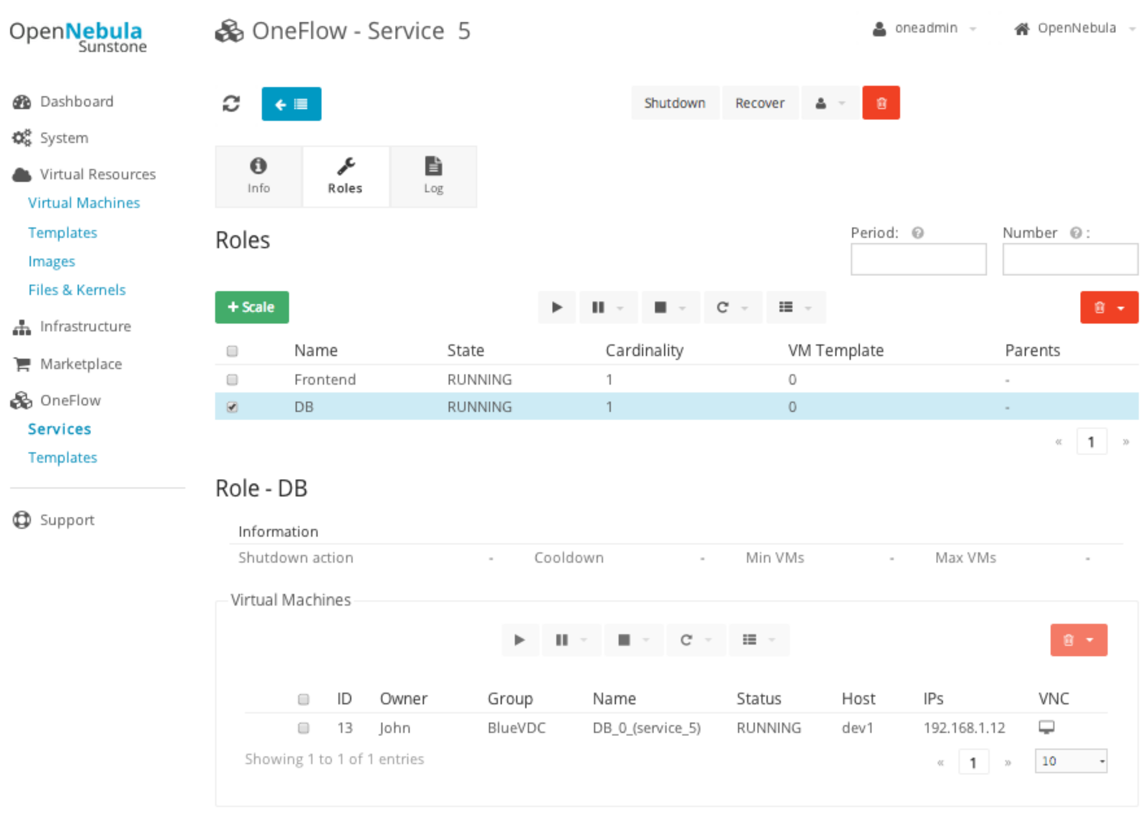
```
SERVICE 7 INFORMATION
...

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 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.



Roles

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

Role - DB

Information

Shutdown action	-	Cooldown	-	Min VMs	-	Max VMs	-

Virtual Machines

	ID	Owner	Group	Name	Status	Host	IPs	VNC
□	13	John	BlueVDC	DB_0_(service_5)	RUNNING	dev1	192.168.1.12	🖥

Showing 1 to 1 of 1 entries

1.4.7 Examples

```
/*
Testing:

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

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

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

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

4) Update VM templates to trigger the scale down expression. The number of
VMs is adjusted -10 percent. Because 5 * 0.10 < 1, the adjustment is rounded to 1;
but the min_adjust_step is set to 2, so the final adjustment is -2 VMs.
*/
{
  "name": "Scalability1",
  "deployment": "none",
  "roles": [
    {
      "name": "frontend",
      "cardinality": 2,
      "vm_template": 0,

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

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

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

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

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

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

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

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

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

CHAPTER

TWO

HOST AND VM HIGH AVAILABILITY

2.1 Overview

2.1.1 How Should I Read This Chapter

The *Front-end HA Setup* section will guide you through the process of setting up two front-ends in an active/passive cluster mode. This setup requires the use of a MySQL database.

If you want to enable an automatic Virtual Machine recovery in case of a Host failure, or if you want to learn how to manually recover a Virtual Machine in a failed state, please read the *Virtual Machines High Availability* section.

2.1.2 Hypervisor Compatibility

Section	Compatibility
Front-end HA Setup	This Section applies to both KVM and vCenter.
Virtual Machines High Availability	This Section applies only to KVM.

2.2 Front-end HA Setup

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

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

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

This guide is structured in a *how-to* form using Pacemaker tested in a CentOS 7 installation and in Ubuntu 14.04; but generic considerations and requirements for this setup are discussed to easily implement this solution with other systems.

2.2.1 Overview

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

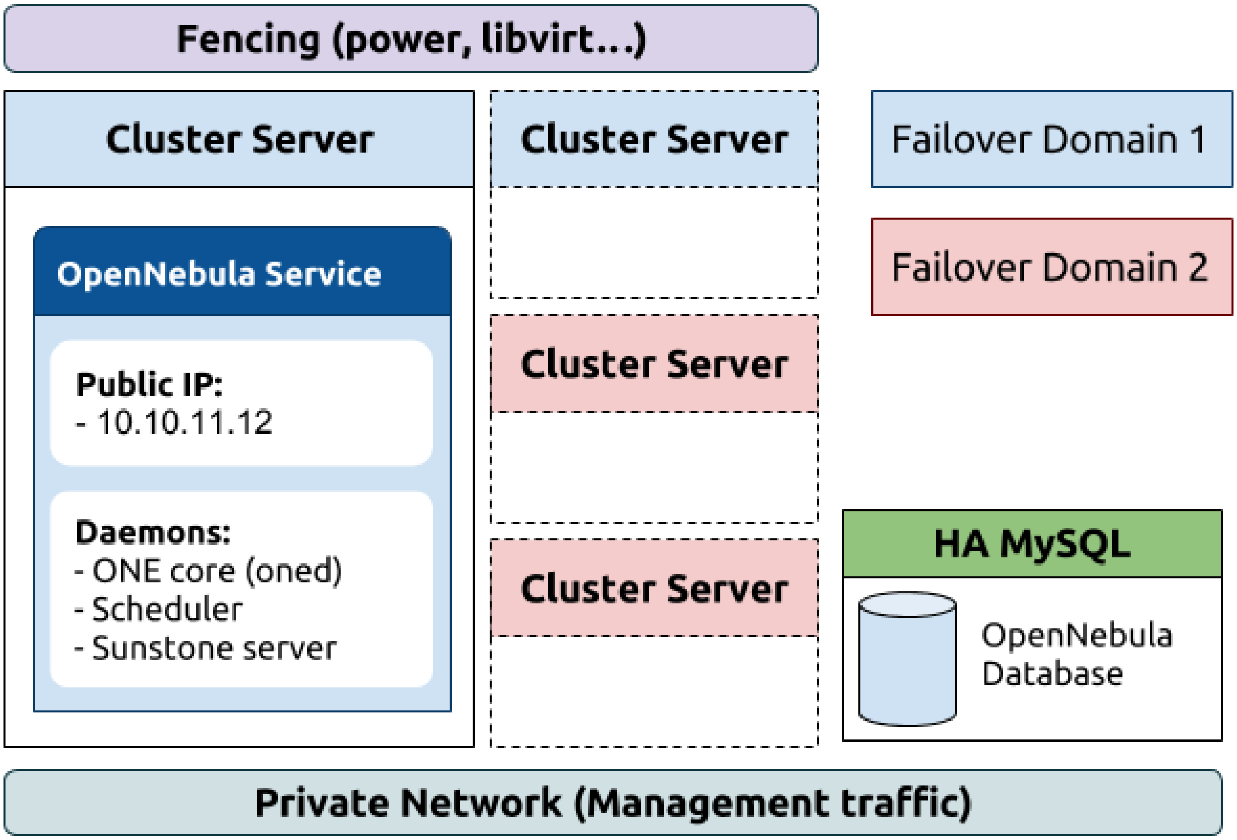
- **OpenNebula Core**: It is the main orchestration component, supervises the life-cycle of each resources (e.g. hosts, VMs or networks) and operates on the physical infrastructure to deploy and manage virtualized resources.

- **Scheduler**: The scheduler performs a matching between the virtual requests and the available resources using different scheduling policies. It basically assigns a physical host, and a storage area to each VM.
- **Sunstone**: The GUI for advanced and cloud users as well as system administrators. The GUI is accessed through a well-known end-point (IP/URL). Sunstone has been architected as a scalable web application supporting multiple application servers or processes.

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

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

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



2.2.2 HA Cluster Components & Services

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

The following components will be installed:

- opennebula services
- corosync+pacemaker
- fencing agents

2.2.3 Installation and Configuration in CentOS 7

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

- one-server1
- one-server2

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

Step 1: OpenNebula

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

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

Step 2: Install Cluster Software

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

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

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

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

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

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

Start/enable the cluster services:

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

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

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

Step 3: Create the Cluster and Failover Domain

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

Authorize the nodes:

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

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

Now we need to create the cluster:

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

Now we can start the cluster:

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

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

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

Step 4: Define the OpenNebula Service

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

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

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

```
$ pcs stonith create fence_server1 fence_ilo_ssh pcmk_host_list=one-server1 ipaddr=
↪<ilo_hypervisor_one-server1> login="..." passwd="..." action="reboot" secure=yes␣
↪delay=30 op monitor interval=20s
$ pcs stonith create fence_server2 fence_ilo_ssh pcmk_host_list=one-server2 ipaddr=
↪<ilo_hypervisor_one-server2> login="..." passwd="..." action="reboot" secure=yes␣
↪delay=15 op monitor interval=20s
```

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

You can try out fencing manually by running these commands:

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

Or even by calling the cluster:

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

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

```
$ pcs resource create Cluster_VIP ocf:heartbeat:IPaddr2 ip=<HA_ip> cidr_netmask=24 op␣
↪monitor interval=20s
```

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

```
$ pcs resource describe ocf:heartbeat:IPaddr2
```

We are now ready to add the OpenNebula resources:

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

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

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

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

At this point, the cluster should be properly configured:

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

Online: [ one-server1 one-server2 ]

Full list of resources:

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

Failed Actions:

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

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

2.2.4 Installation and Configuration in Ubuntu 14.04

We assume that the cluster consists on two servers:

- one-server1
- one-server2

Step 1: OpenNebula

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

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

Step 2: Install Cluster Software

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

```
$ apt-get install pacemaker fence-agents
```

Now disable the automatic start-up of the OpenNebula resources and enable the HA services on boot time:

```
$ update-rc.d opennebula disable
$ update-rc.d opennebula-gate disable
$ update-rc.d opennebula-flow disable
$ update-rc.d pacemaker defaults 20 01
```

Step 3: Configure the Cluster

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

We need to generate a corosync key, in order to do so we will need an entropy dameon which we will uninstall right afterwards:

```
$ apt-get install haveged
$ corosync-keygen
$ apt-get remove --purge haveged
```

Let's define the corosync.conf (based on `/etc/corosync/corosync.conf.example.udpu` which installs as part of the package):

```
$ cat /etc/corosync/corosync.conf
totem {
  version: 2
  cluster_name: opennebula

  crypto_cipher: none
  crypto_hash: none

  interface {
    ringnumber: 0
    bindnetaddr: 10.3.4.0
    mcastport: 5405
    ttl: 1
  }
  transport: udpu
}

logging {
  fileline: off
  to_logfile: yes
  to_syslog: yes
  logfile: /var/log/corosync/corosync.log
  debug: off
  timestamp: on
  logger_subsys {
    subsys: QUORUM
    debug: off
  }
}

nodelist {
  node {
    ring0_addr: 10.3.4.20
    nodeid: 1
  }

  node {
    ring0_addr: 10.3.4.21
    nodeid: 2
  }
}

quorum {
  provider: corosync_votequorum
  two_node: 1
}
```

Substitute the IPs with the proper ones, and make sure that in `totem -> interface -> bindnetaddr` you type in the **network address**, not the IP.

Allow the pacemaker service to use corosync:

```
$ cat /etc/corosync/service.d/pcmk
service {
  name: pacemaker
  ver: 1
}
```

And enable the automatic startup of corosync:

```
$ cat /etc/default/corosync
# start corosync at boot [yes|no]
START=yes
```

Distribute these files to the other node:

```
$ scp /etc/corosync/authkey one-server2:/etc/corosync
$ scp /etc/corosync/corosync.conf one-server2:/etc/corosync/corosync.conf
$ scp /etc/corosync/service.d/pcmk one-server2:/etc/corosync/service.d/pcmk
$ scp /etc/default/corosync one-server2:/etc/default/corosync
```

Now start corosync and pacemaker in **both** nodes:

```
$ service corosync start
$ service pacemaker start
```

You can now check that everything is correct:

Check that everything is correct (be patient, it might take a minute or so to get a similar output):

```
$ crm configure show
Last updated: Thu Jul 21 12:05:50 2016
Last change: Thu Jul 21 12:05:18 2016 via cibadmin on one-server2
Stack: corosync
Current DC: one-server2 (2) - partition with quorum
Version: 1.1.10-42f2063
2 Nodes configured
0 Resources configured

Online: [ one-server1 one-server2 ]
```

Step 4: Add resources to the Cluster

Now that the cluster is set up, we need to add the resources to it. However, first we need to determine what fencing (stonith) mechanism we are going to use.

Note: Fencing is not mandatory, but recommended. If you don't want to use fencing you can add the property: `property stonith-enabled=false` to the cib transaction (see below) and skip all the stonith commands below and the stonith primitives in the cib transaction.

To get the supported stonith mechanisms:

```
$ crm ra list stonith
apcmaster                   apcmastersnmp               apcsmart                   ␣
↪baytech                      bladehpi
```

```
cyclades                    drac3                       external/drac5              ␣
 ↪external/dracmc-telnet      external/hetzner
external/hmchttp            external/ibmrsa             external/ibmrsa-telnet      ␣
 ↪external/ipmi               external/ippower9258
external/kdumpcheck         external/libvirt            external/nut                ␣
 ↪external/rackpdu            external/riloe
external/ssh                external/vcenter            external/vmware             ␣
 ↪external/xen0               external/xen0-ha
fence_ack_manual            fence_alom                  fence_apc                   ␣
 ↪fence_apc_snmp              fence_baytech
fence_bladecenter           fence_brocade               fence_bullpap               ␣
 ↪fence_cdu                   fence_cisco_mds
fence_cisco_ucs             fence_cpint                 fence_drac                  ␣
 ↪fence_drac5                 fence_eaton_snmp
fence_egenera               fence_eps                   fence_ibmblade              ␣
 ↪fence_ifmib                 fence_ilo
fence_ilo_mp                fence_intelmodular          fence_ipmilan               ␣
 ↪fence_ldom                  fence_legacy
fence_lpar                  fence_mcdata                fence_na                    ␣
 ↪fence_nss_wrapper           fence_pcmk
fence_rackswitch            fence_rhevm                 fence_rsa                   ␣
 ↪fence_rsb                   fence_sanbox2
fence_scsi                  fence_virsh                 fence_vixel                 ␣
 ↪fence_vmware                fence_vmware_helper
fence_vmware_soap           fence_wti                   fence_xcat                  ␣
 ↪fence_xenapi                fence_zvm
ibmhmc                      ipmilan                     meatware                    ␣
 ↪null                        nw_rpc100s
rcd_serial                  rps10                       ssh                         ␣
 ↪suicide                     wti_mpc
wti_nps
```

To access the documentation of these mechanisms, including how to configure them, use these commands:

```
$ crm ra meta stonith:fence_ipmilan
$ stonith_admin --metadata -a ipmilan
```

From that documentation we can infer that we will need to prepare a stonith primitive such as:

```
primitive ipmi-fencing stonith::fence_ipmilan \
 params pcmk_host_list="one-server1 one-server2" ipaddr=10.0.0.1 login=testuser␣
 ↪passwd=abc123 \
 op monitor interval="60s"
```

Now we are ready to create a new cib transaction and commit it to pacemaker.

```
$ crm
crm(live)# cib new conf1
crm(live)# cib new conf11
INFO: conf11 shadow CIB created
crm(conf1)# configure
crm(conf1)configure# property no-quorum-policy=ignore
crm(conf1)configure# primitive ipmi-fencing stonith::fence_ipmilan \
> params pcmk_host_list="one-server1 one-server2" ipaddr=10.0.0.1 login=testuser␣
 ↪passwd=abc123 \
> op monitor interval="60s"
crm(conf1)configure# primitive VIP ocf:IPaddr2 params ip=10.3.4.2 op monitor␣
 ↪interval=10s
```

```
crm(conf1)configure# primitive opennebula lsb::opennebula op monitor interval=15s
crm(conf1)configure# primitive opennebula-sunstone lsb::opennebula-sunstone op␣
↪monitor interval=15s
crm(conf1)configure# primitive opennebula-flow lsb::opennebula-flow op monitor␣
↪interval=15s
crm(conf1)configure# primitive opennebula-gate lsb::opennebula-gate op monitor␣
↪interval=15s
crm(conf1)configure# group opennebula-cluster VIP opennebula opennebula-gate␣
↪opennebula-sunstone opennebula-flow
crm(conf1)configure# commit
crm(conf1)configure# end
crm(conf1)# cib commit conf1
INFO: committed 'conf1' shadow CIB to the cluster
crm(conf1)# quit
```

At this point, the cluster should be properly configured:

```
$ crm status
Last updated: Thu Jul 21 16:10:38 2016
Last change: Thu Jul 21 16:10:34 2016 via cibadmin on one-server1
Stack: corosync
Current DC: one-server1 (1) - partition with quorum
Version: 1.1.10-42f2063
2 Nodes configured
6 Resources configured


Online: [ one-server1 one-server2 ]

 Resource Group: opennebula-cluster
     VIP  (ocf::heartbeat:IPaddr2): Started one-server2
     opennebula (lsb:opennebula): Started one-server2
     opennebula-gate  (lsb:opennebula-gate):  Started one-server2
     opennebula-sunstone  (lsb:opennebula-sunstone):  Started one-server2
     opennebula-flow  (lsb:opennebula-flow):  Started one-server2
 ipmi-fencing (stonith:ipmilan): Started one-server1
```

2.2.5 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 might be a set of Virtual Machines which could be stuck in transient states. For example when a Virtual Machine is deployed and it starts copying the disks to the target hosts it enters one of this transient states (in this case 'PROLOG'). OpenNebula will wait for the storage driver to return the 'PROLOG' exit status. This will never happen since the driver fails during the crash, therefore the Virtual Machine will get stuck in the state.

In these cases it's important to review the states of all the Virtual Machines and let OpenNebula know if the driver exited 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

2.3 Virtual Machines High Availability

This section's objective is to provide information in order to prepare for failures in the Virtual Machines or Hosts, 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.

2.3.1 Host Failures

Todo

needs to be updated with new flags

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   = "ft/host_error.rb",
    arguments = "$ID -m -p 5",
    remote    = "no" ]
#-------------------------------------------------------------------------------
```

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

Argument	Description
Host ID	ID of the host containing the VMs to treat. It is compulsory and better left to **$ID**, that will be automatically filled by OpenNebula with the Host ID of the host that went down.
Action	This defines the action to be performed upon the VMs that were running in the host that went down. This can be: • **-m** migrate VMs to another host. Only for images in shared storage • **-r** delete+recreate VMs running in the host. State will be lost. • **-d** delete VMs running in the host
ForceSuspended	[-f] force resubmission of suspended VMs
AvoidTransient	[-p <n>] avoid resubmission if host comes back after <n> monitoring cycles

More information on hooks here.

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

2.3.2 Virtual Machine Failures

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

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

- `--success`: If the operation has been confirmed to succeed. For example, the administrator can see the VM properly running in the hypervisor, but the driver failed to inform OpenNebula of the successful boot.
- `--failure`: This will have the same effect as a driver reporting a failure. It is intended for VMs that get stuck in transient states. As an example, if a storage problem occurs and the administrator knows that a VM stuck in `prolog` is not going to finish the pending transfer, this action will manually move the VM to `prolog_failure`.
- `--retry`: To retry the previously failed action. Can be used, for instance, in case a VM is in `boot_failure` because the hypervisor crashed. The administrator can tell OpenNebula to retry the boot after the hypervisor is started again.
- `--retry --interactive`: In some scenarios where the failure was caused by an error in the Transfer Manager actions, each action can be rerun and debugged until it works. Once the commands are successful, a `success` should be sent. See the specific section below for more details.
- `--delete`: No recover action possible, delete the VM. This is equivalent to the deprecated OpenNebula < 5.0 command: `onevm delete`.
- `--recreate`: No recover action possible, delete and recreate the VM. This is equivalent to the deprecated OpenNebula < 5.0 command: `onevm delete --recreate`.

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

Hypervisor Problems

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

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

Transfer Manager / Storage Problems

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

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

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

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

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

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

```
(2) MV shared haddock:/var/lib/one//datastores/0/2 localhost:/var/lib/one//datastores/
↪0/2 2 0

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

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

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

FAILURE. Repeat command.

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

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

> r

SUCCESS

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

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

SUCCESS

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

$ onevm recover 2 --success

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

CHAPTER

THREE

DATA CENTER FEDERATION

3.1 Overview

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

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

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

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

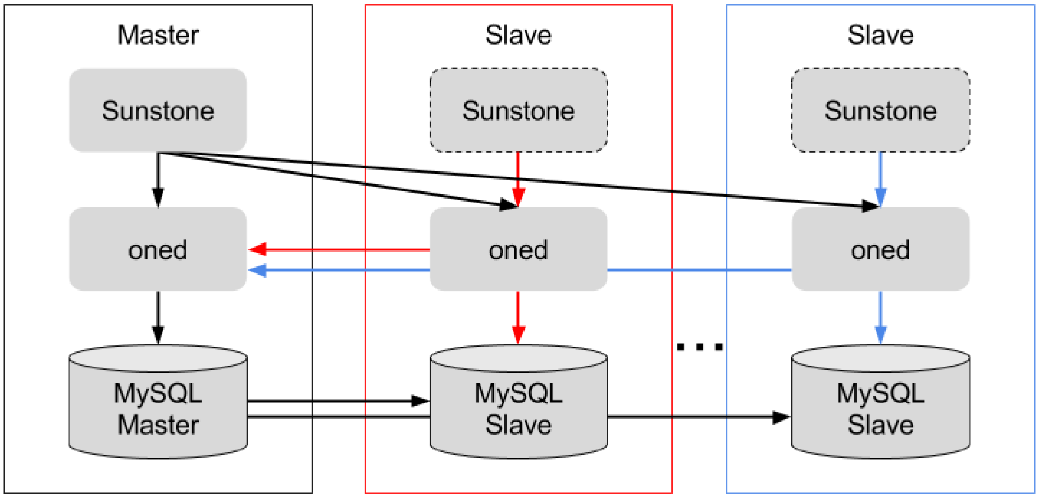
3.1.1 Architecture

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

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

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

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



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

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

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

3.1.2 Other Services

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

3.1.3 How Should I Read This Chapter

Before reading this chapter make sure you have read the Deployment Guide.

Read the *Federation Configuration* section to learn how to setup a federation, and the *Federation Management* section to learn how to manage zones in OpenNebula.

After reading this chapter you can continue configuring more *Advanced Components*.

3.1.4 Hypervisor Compatibility

This chapter applies both to KVM and vCenter.

3.2 OpenNebula Federation Configuration

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

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

3.2.1 Step 1. Configure the OpenNebula Federation Master

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

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

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

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

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

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

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

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

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

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

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

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

- Stop OpenNebula.

3.2.2 2. Import the Existing Slave OpenNebula

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

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

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

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

## OPTIONS
...

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

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

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

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

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

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

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

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

```
The import process will move the users from the slave OpeNenbula to the master
OpenNebula. In case of conflict, it can merge users with the same name.
For example:
+----------+-------------++------------+---------------+
| Master   | Slave       || With merge | Without merge |
+----------+-------------++------------+---------------+
| 5, alice | 2, alice    || 5, alice   | 5, alice      |
| 6, bob   | 5, bob      || 6, bob     | 6, bob        |
|          |             ||            | 7, alice-1    |
|          |             ||            | 8, bob-1      |
+----------+-------------++------------+---------------+
In any case, the ownership of existing resources and group membership
is preserved.

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

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

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

3.2.3 3. Configure the MySQL Replication Master

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

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

$ sudo service mysqld restart
```

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

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

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

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

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

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

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

```
mysqldump -u root -p --master-data opennebula user_pool group_pool marketplace_pool␣
↪marketplaceapp_pool vdc_pool zone_pool db_versioning acl > dump.sql
```

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

```
mysql> UNLOCK TABLES;
```

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

3.2.4 4. Configure the MySQL Replication Slave

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

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

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

$ sudo service mysqld restart
```

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

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

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

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

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

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

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

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

3.2.5 5. Configure the OpenNebula Federation Slave

For each slave, follow these steps.

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

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

and update oned.conf to use these values:

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

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

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

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

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

```
$ ls -1 /var/lib/one/.one
ec2_auth
one_auth
oneflow_auth
onegate_auth
sunstone_auth
```

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

- Start the slave OpenNebula.

3.2.6 6. Configure Sunstone for a Federation

If Sunstone is behind a proxy, make sure you follow these instructions in order to enable the proper header support required by a federation.

3.2.7 High-Availability and Federation

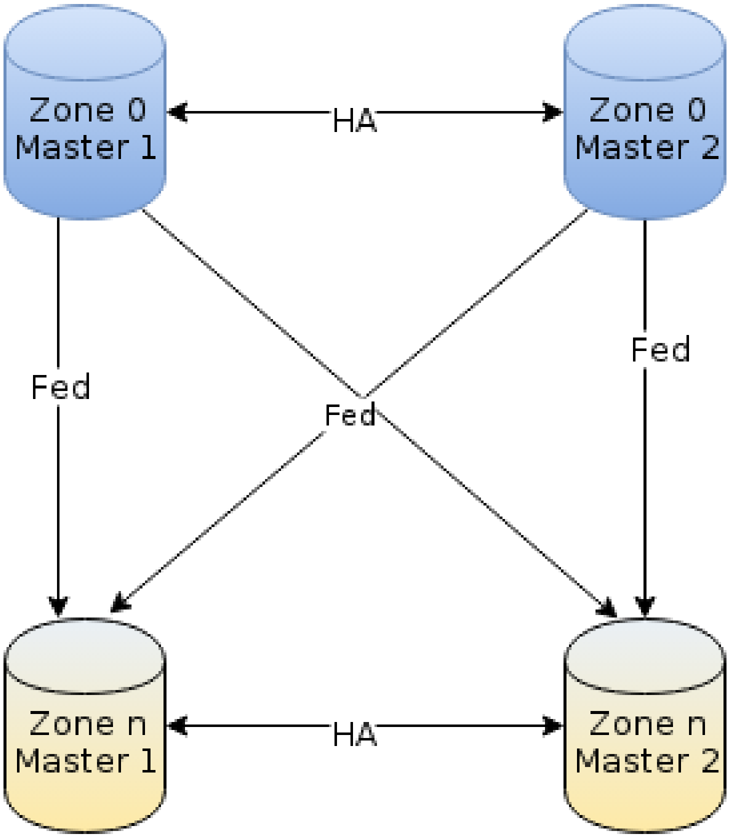
In order to add *federation* to an HA set-up you will need to use MariaDB >= 10.0.2. If this version is not available in your distribution, please use the repositories provided by MariaDB.

The procedure to enable both HA and Federation uses the multi source replication capabilities of MariaDB.

- Every zone must have a 2-node master-master MariaDB cluster.

- Every zone except for the master zone should configure DB replcation for the federated tables from **both** MariaDB nodes of the master zone.

This is illustrated by the following diagram:



The **HA** arrows represent a full master-master replication of all the OpenNebula tables. The **Fed** arrows represent a replication of only the federation tables.

Each replication arrow is implemented in MariaDB by a slave configured with the CHANGE MASTER directive. Note that we will be using a *connection_name* in order to identify each slave.

Note: The HA cluster can must be composed of at least 2 nodes, but you can scale up to as many nodes as you need. In order to so, you should set up a circular replication for HA: A->B->C->...->A and pull the federated tables from all the nodes of the master zone in the rest of the zones.

Configuration

To set-up the HA replication in each cluster enable the following in the MariaDB configuration file, e.g. */etc/my.cnf.d/server.cnf* of both nodes:

```
[mysqld]
server-id     = 1 # Use a different ID for all the servers
log-bin       = mysql-bin
binlog-do-db = opennebula
```

Additionally, in all the zones but the master zone, configure the federation replication. This is how */etc/my.cnf.d/server.cnf* looks like for these nodes

```
[mysqld]
server-id    = 100 # Use a different ID for all the servers
log-bin      = mysql-bin
binlog-do-db = opennebula

zone0-master1.replicate-do-table  = opennebula.user_pool
zone0-master1.replicate-do-table  = opennebula.group_pool
zone0-master1.replicate-do-table  = opennebula.marketplace_pool
zone0-master1.replicate-do-table  = opennebula.marketplaceapp_pool
zone0-master1.replicate-do-table  = opennebula.vdc_pool
zone0-master1.replicate-do-table  = opennebula.zone_pool
zone0-master1.replicate-do-table  = opennebula.db_versioning
zone0-master1.replicate-do-table  = opennebula.acl

zone0-master2.replicate-do-table  = opennebula.user_pool
zone0-master2.replicate-do-table  = opennebula.group_pool
zone0-master2.replicate-do-table  = opennebula.marketplace_pool
zone0-master2.replicate-do-table  = opennebula.marketplaceapp_pool
zone0-master2.replicate-do-table  = opennebula.vdc_pool
zone0-master2.replicate-do-table  = opennebula.zone_pool
zone0-master2.replicate-do-table  = opennebula.db_versioning
zone0-master2.replicate-do-table  = opennebula.acl
```

Restart the MariaDB service in all the nodes, e.g.:

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

Create the replication users as explained in *this section*.

HA Replication

Follow these steps in all the zones, including the master zone.

Obtain the master position in the first node:

```
> SHOW MASTER STATUS;
+------------------+-----------+--------------+------------------+
| File             | Position  | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+-----------+--------------+------------------+
| <LOG_FILE>       | <LOG_POS> | opennebula   |                  |
+------------------+-----------+--------------+------------------+
```

Configure the second node to replicate using this data:

```
CHANGE MASTER 'zone<ZONE_ID>-master' TO  MASTER_HOST='<NODE1>', \
                                MASTER_USER='<REPLICATION_USER>', \
                                MASTER_PASSWORD='<REPLICATION_PASS>', \
                                MASTER_LOG_FILE = '<LOG_FILE>', \
                                MASTER_LOG_POS = <LOG_POS>;
START SLAVE 'zone<ZONE_ID>-master';
```

Repeat the reverse process by running *SHOW MASTER STATUS* in the second node, and establishing it as the master in the first node:

```
CHANGE MASTER 'zone<ZONE_ID>-slave' TO  MASTER_HOST = '<NODE2>', \
                                MASTER_USER = '<REPLICATION_USER>', \
                                MASTER_PASSWORD ='<REPLICATION_PASS>', \
                                MASTER_LOG_FILE = '<LOG_FILE>', \
                                MASTER_LOG_POS = <LOG_POS>;
START SLAVE 'zone<ZONE_ID>-slave';
```

Federation

In all the nodes, except the nodes in the master zone, you will to set up the replication of the federated tables from both nodes in the master zone.

Repeat the following commands in both nodes of each zone:

```
CHANGE MASTER 'zone0-master1' TO    MASTER_HOST = '<ZONE0_MASTER1_IP>', \
                                    MASTER_USER = '<REPLICATION_USER>', \
                                    MASTER_PASSWORD = '<REPLICATION_PASS>';
START SLAVE 'zone0-master1';

CHANGE MASTER 'zone0-master2' TO    MASTER_HOST = '<ZONE0_MASTER2_IP>', \
                                    MASTER_USER = '<REPLICATION_USER>', \
                                    MASTER_PASSWORD = '<REPLICATION_PASS>';
START SLAVE 'zone0-master2';
```

Verify

Verify in all the nodes that the replication is up and running both for HA and for Federation:

```
$ mysql -u root -p -e "SHOW ALL SLAVES STATUS \G" | grep -E 'Connection_name|_Running'
              Connection_name: zone0-master1
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Connection_name: zone0-master2
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Connection_name: zone<ZONE_ID>-<master|slave>
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
```

If *Slave_IO_Running* or *Slave_SQL_Running* is not *Yes*, then the replication is not running.

Failover Scenario

Should a failover event take place, the OpenNebula service will balance normally and everything will work. However, when the fenced node is brought up again, it should **not** be configured to enter the cluster automatically. It is very important that the node only enters the cluster again only if the replication is up to date, that is, only if *Slave_IO_Running* or *Slave_SQL_Running* are set to *Yes*.

3.3 OpenNebula Federation Management

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

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

3.3.1 Adding a Zone

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

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

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

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

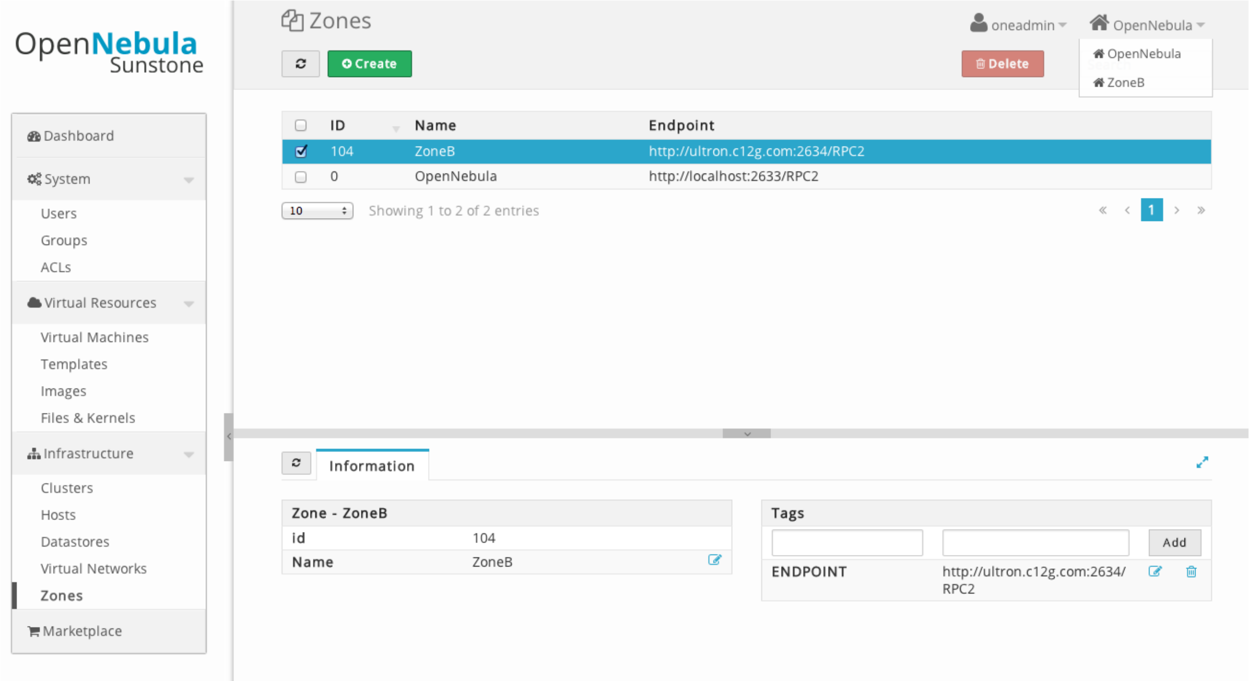
Note: The ENDPOINT has to be reachable from the Sunstone server machine, or the computer running the CLI in order for the user to access the Zone.

3.3.2 Using a Zone

Through Sunstone

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

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



	ID	Name	Endpoint
☑	104	ZoneB	http://ultron.c12g.com:2634/RPC2
□	0	OpenNebula	http://localhost:2633/RPC2

Showing 1 to 2 of 2 entries

Zone - ZoneB	
id	104
Name	ZoneB

ENDPOINT http://ultron.c12g.com:2634/RPC2

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

Through CLI

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

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

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

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

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

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

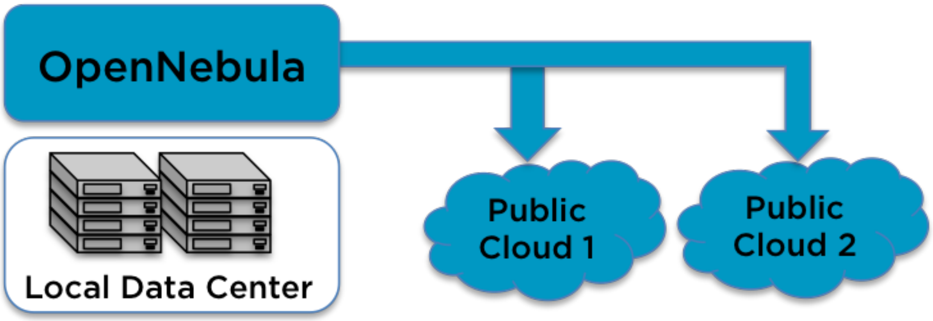
CHAPTER

FOUR

CLOUD BURSTING

4.1 Overview

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 Microsoft Azure. Such support for cloud bursting enables highly scalable hosting environments.



OpenNebula's approach to cloud bursting is based on the transparency to both end users and cloud administrators to use and maintain the cloud bursting functionality. The **transparency to cloud administrators** comes from the fact that a AWS EC2 region or an Azure location is modeled as any other host (albeit of potentially a much bigger capacity), so the scheduler can place VMs in the external cloud as it will do in any other local host.

On the other hand, the **transparency to end users** is offered through the hybrid template functionality: the same VM template in OpenNebula can describe the VM if it is deployed locally and also if it gets deployed in EC2, or Azure. Therefore users just have to instantiate the template, and OpenNebula will transparently choose if that is executed locally or remotely.

4.1.1 How Should I Read This Chapter

You should be reading this Chapter as part of the *Advanced Components Guide* review of that OpenNebula advanced functionality that you are interested in enabling and configuring.

Within this Chapter you can find a guide to configure and use the *Amazon EC2 driver* and the *Azure driver*. Additionally, there is support of IBM SoftLayer available as add-on (ie, not contained in the OpenNebula main distribution) here.

Once the cloud architecture has been designed the next step would be to learn how to install the OpenNebula front-end.

4.1.2 Hypervisor Compatibility

All the Sections in this Chapter applies to both KVM and vCentre based OpenNebula clouds.

4.2 Amazon EC2 Driver

4.2.1 Considerations & Limitations

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

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

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

- General information of instances

4.2.2 Prerequisites

- You must have a working account for AWS and signup for EC2 and S3 services.
- The aws-sdk ruby gem needs to be installed, version 1.66. This gem is automatically installed as part of the installation process.

4.2.3 OpenNebula Configuration

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

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

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

Driver flags are the same as other drivers:

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

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

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

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

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

Also, you can modify in the same file the default 300 seconds timeout that is waited for the VM to be in the EC2 running state in case you also want to attach to the instance a elastic ip:

```
state_wait_timeout_seconds: 300
```

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

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

4.2.4 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

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

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

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

Check an exhaustive list of attributes in the Virtual Machine Definition File Reference Section.

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

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

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

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

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

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

Context Support

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

```
$ curl http://169.254.169.254/latest/user-data
ONEGATE_ENDPOINT="https://onegate...
SSH_PUBLIC_KEY="ssh-rsa ABAABeqzaC1y...
```

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

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

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

4.2.5 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
```

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

You can define a different PUBLIC_CLOUD 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:

```
PUBLIC_CLOUD = [ TYPE="EC2",
                 HOST="ec2",
                 AMI="ami-0022c769" ]
PUBLIC_CLOUD = [ TYPE="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

4.2.6 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).

4.2.7 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

# 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

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

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

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

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

Now you can monitor the state of the VM with

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

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

- AWS_DNS_NAME
- AWS_PRIVATE_DNS_NAME
- AWS_KEY_NAME
- AWS_AVAILABILITY_ZONE
- AWS_PLATFORM
- AWS_VPC_ID
- AWS_PRIVATE_IP_ADDRESS

- AWS_IP_ADDRESS
- AWS_SUBNET_ID
- AWS_SECURITY_GROUPS
- AWS_INSTANCE_TYPE

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

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

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

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

USER TEMPLATE
PUBLIC_CLOUD=[
  TYPE="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"
```

4.2.8 Scheduler Configuration

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

Configure the Priority

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

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

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

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

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

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

For example, having this configuration:

- MAX_HOST = 1
- MAX_DISPATCH = 30
- 2 Hosts: 1 in the local infrastructure, and 1 using the 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"
```

4.2.9 Importing VMs

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

4.3 Azure Driver

4.3.1 Considerations & Limitations

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

- There is no direct access to the hypervisor, so it cannot be monitored (we don't know where the VM is running on the Azure cloud).
- The usual OpenNebula functionality for snapshotting, hot-plugging, or migration is not available with Azure.
- By default OpenNebula will always launch Small (1 CPU, 1792 MB RAM) instances, unless otherwise specified. The following table is an excerpt of all the instance types available in Azure, a more exhaustive list can be found (and edited) in `/etc/one/az_driver.conf`.

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

4.3.2 Prerequisites

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

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

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

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

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

- Third, the certificate file (.cer) has to be uploaded to Settings -> Management Certificates
- The following gem is required: `azure`. This gem is automatically installed as part of the installation process. Otherwise, run the `install_gems` script as root:

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

4.3.3 OpenNebula Configuration

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

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

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

Driver flags are the same as other drivers:

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

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

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

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

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

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

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

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

4.3.4 Azure Specific Template Attributes

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

```
CPU      = 0.5
MEMORY   = 128

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

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

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

These are the attributes that can be used in the PUBLIC_CLOUD section of the template for TYPE "AZURE":

Check an exhaustive list of attributes in the Virtual Machine Definition File Reference Section.

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

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

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

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

4.3.5 Multi Azure Location/Account Support

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

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

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

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

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

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

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

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

```
PUBLIC_CLOUD = [ TYPE=AZURE,
                 INSTANCE_TYPE=Small,
                 IMAGE=b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-14_04-LTS-amd64-
↪server-20140606.1-en-us-30GB,
                 VM_USER="MyUserName",
                 VM_PASSWORD="MyPassword",
                 LOCATION="brazil-south"
]

PUBLIC_CLOUD = [ TYPE=AZURE,
                 INSTANCE_TYPE=Medium,
                 IMAGE=0b11de9248dd4d87b18621318e037d37__RightImage-Ubuntu-12.04-x64-
↪v13.4,
                 VM_USER="MyUserName",
                 VM_PASSWORD="MyPassword",
                 LOCATION="west-europe"
]
```

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

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

4.3.6 Hybrid VM Templates

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

An example of a hybrid template:

```
## Local Template section
NAME=MNyWebServer

CPU=1
```

```
MEMORY=256

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

PUBLIC_CLOUD = [ TYPE=AZURE,
                 INSTANCE_TYPE=Medium,
                 IMAGE=0b11de9248dd4d87b18621318e037d37__RightImage-Ubuntu-12.04-x64-
↪v13.4,
                 VM_USER="MyUserName",
                 VM_PASSWORD="MyPassword",
                 LOCATION="west-europe"
]
```

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

4.3.7 Testing

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

```
CPU      = 1
MEMORY   = 1700

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

# Azure template machine, this will be use when submitting this VM to Azure

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

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

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

```
$ onetemplate create aztemplate
$ onetemplate instantiate aztemplate
```

Now you can monitor the state of the VM with

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

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

- AZ_AVAILABILITY_SET_NAME
- AZ_CLOUD_SERVICE_NAME,
- AZ_DATA_DISKS,
- AZ_DEPLOYMENT_NAME,
- AZ_DISK_NAME,
- AZ_HOSTNAME,
- AZ_IMAGE,
- AZ_IPADDRESS,
- AZ_MEDIA_LINK,
- AZ_OS_TYPE,
- AZ_ROLE_SIZE,
- AZ_TCP_ENDPOINTS,
- AZ_UDP_ENDPOINTS,
- AZ_VIRTUAL_NETWORK_NAME

```
$ onevm show 0
VIRTUAL MACHINE 0 INFORMATION
ID                  : 0
NAME                : one-0
USER                : oneadmin
GROUP               : oneadmin
STATE               : ACTIVE
LCM_STATE           : RUNNING
RESCHED             : No
START TIME          : 06/25 13:05:29
END TIME            : -
HOST                : west-europe
CLUSTER ID          : -1
DEPLOY ID           : one-0_opennebuladefaultcloudservicename-0


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

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

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


USER TEMPLATE
PUBLIC_CLOUD=[
  IMAGE="b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-14_04-LTS-amd64-server-20140606.1-
↪en-us-30GB",
  INSTANCE_TYPE="ExtraSmall",
  SSH_PORT="2222",
  TCP_ENDPOINTS="80",
  TYPE="AZURE",
  VM_PASSWORD="MyVMPassword",
  VM_USER="MyUserName",
  WIN_RM="https" ]
VIRTUAL MACHINE TEMPLATE
AUTOMATIC_REQUIREMENTS="!(PUBLIC_CLOUD = YES) | (PUBLIC_CLOUD = YES & (HYPERVISOR =␣
↪AZURE | HYPERVISOR = AZURE))"
AZ_CLOUD_SERVICE_NAME="opennebuladefaultcloudservicename-0"
AZ_DEPLOYMENT_NAME="OpenNebulaDefaultCloudServiceName-0"
AZ_DISK_NAME="OpenNebulaDefaultCloudServiceName-0-one-0_
↪OpenNebulaDefaultCloudServiceName-0-0-201406251107210062"
AZ_HOSTNAME="ubuntu"
AZ_IMAGE="b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-14_04-LTS-amd64-server-20140606.1-
↪en-us-30GB"
AZ_IPADDRESS="191.233.70.93"
AZ_MEDIA_LINK="http://one0opennebuladefaultclo.blob.core.windows.net/vhds/disk_2014_
↪06_25_13_07.vhd"
AZ_OS_TYPE="Linux"
AZ_ROLE_SIZE="ExtraSmall"
AZ_TCP_ENDPOINTS="name=SSH,vip=23.97.101.202,publicport=2222,local_port=22,local_
↪port=tcp;name=TCP-PORT-80,vip=23.97.101.202,publicport=80,local_port=80,local_
↪port=tcp"
CPU="1"
MEMORY="1024"
VMID="0"
```

4.3.8 Scheduler Configuration

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

Configure the Priority

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

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

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

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

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

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

For example, having this configuration:

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

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

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

Force a Local or Remote Deployment

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

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

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

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

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

4.3.9 Importing VMs

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

CHAPTER

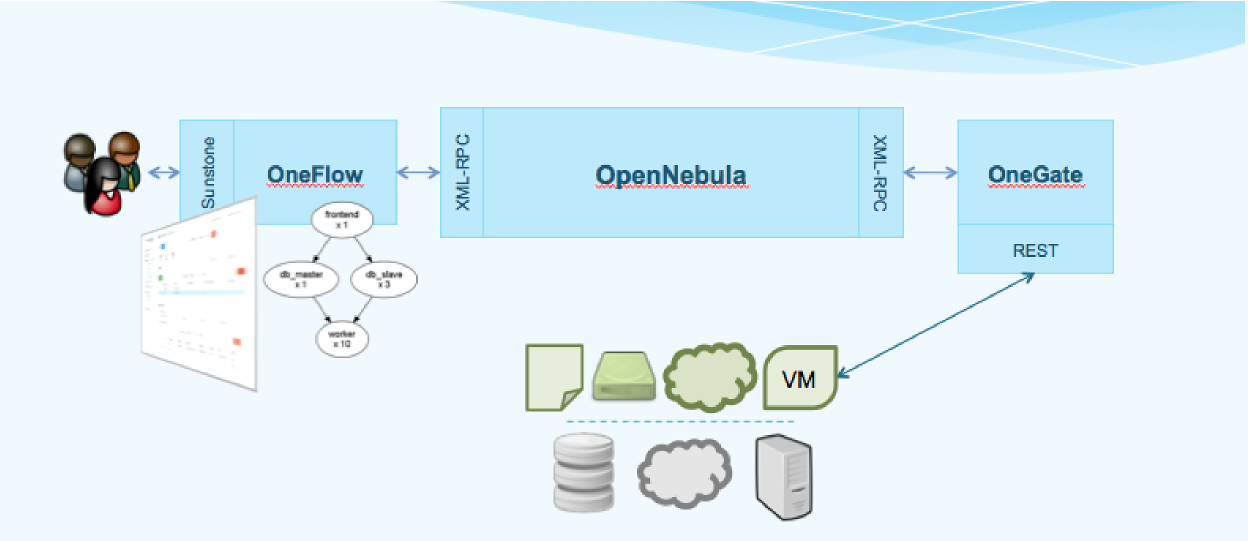
FIVE

APPLICATION INSIGHT

5.1 Overview

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

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



5.1.1 How Should I Read This Chapter

This chapter should be read after the infrastructure is properly setup, and contains working Virtual Machine templates.

Proceed to each section following these links:

- *OneGate Server Configuration*
- *Application Monitoring*

5.1.2 Hypervisor Compatibility

This chapter applies to all the hypervisors.

5.2 OneGate Server Configuration

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

5.2.1 Requirements

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

OneGate can be used with VMs running on KVM, vCenter, and *EC2*.

Currently, OneGate is not supported for VMs instantiated in Azure, since the authentication token is not available inside these VMs. OneGate support for these drivers will be include in upcoming releases.

5.2.2 Configuration

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

Server Configuration

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

Log

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

Auth

- `auth`: Authentication driver for incomming requests.
  - `onegate`: based on token provided in the context
- `core_auth`: Authentication driver to communicate with OpenNebula core.
  - `cipher` for symmetric cipher encryption of tokens
  - `x509` for x509 certificate encryption of tokens. For more information, visit the OpenNebula Cloud Auth documentation.
- `oneflow_server` Endpoint where the OneFlow server is listening.
- `permissions` By default OneGate exposes all the available API calls, each of the actions can be enabled/disabled in the server configuration.
- `restricted_attrs` Attrs that cannot be modified when updating a VM template
- `restricted_actions` Actions that cannot be performed on a VM

This is the default file

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

# OpenNebula sever contact information
```

```
#
:one_xmlrpc: http://localhost:2633/RPC2

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

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

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

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

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

# Authentication driver for 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

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

:oneflow_server: http://localhost:2474

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

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

# Attrs that cannot be modified when updating a VM template
:restricted_attrs
```

```
  - SCHED_REQUIREMENTS
  - SERVICE_ID
  - ROLE_NAME

# Actions that cannot be performed on a VM
:restricted_actions
  #- deploy
  #- delete
  #- hold
  ...
```

5.2.3 Start OneGate

To start and stop the server, use the `opennebula-gate` service:

```
# systemctl start opennebula-gate
```

Or use service in older Linux systems:

```
# service opennebula-gate start
```

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
```

5.2.4 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"
```

At this point the service is ready, you can continue to the *OneGate usage documentation*.

5.2.5 Configuring a SSL Proxy

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

Update your package lists and install Nginx:

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

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

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

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

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

server {
  listen 443;
  server_name ONEGATE_ENDPOINT;

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

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

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

  location / {

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

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

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

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

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

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

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

Then restart oned, onegate-server and Nginx:

```
$ sudo service nginx restart
$ sudo service opennebula restart
$ sudo service opennebula-gate restart
```

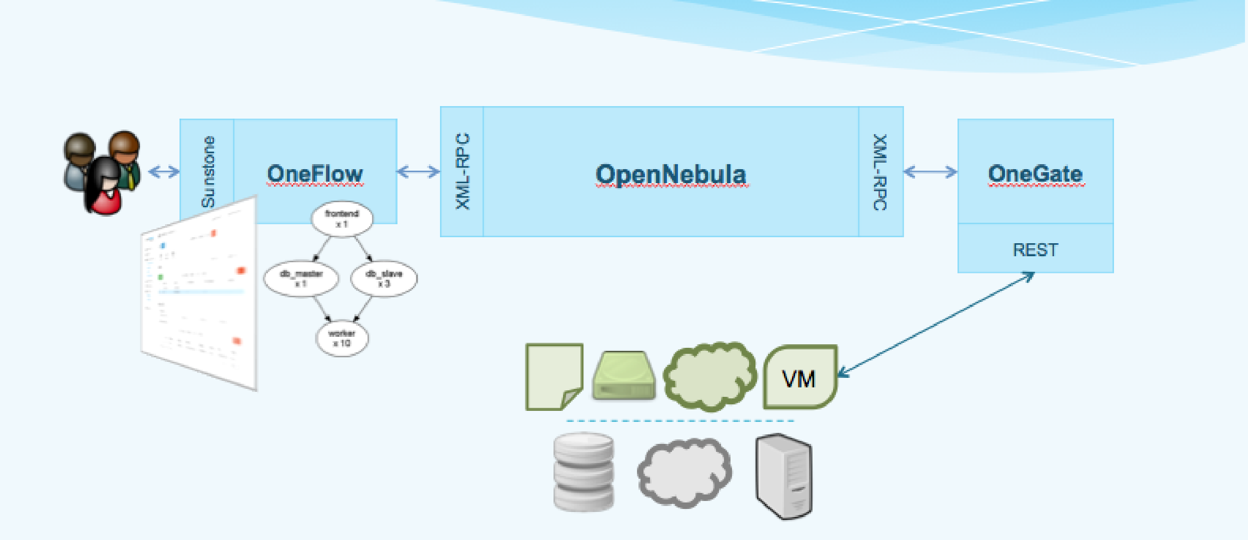
5.3 OneGate Usage

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

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

5.3.1 OneGate Workflow Explained

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



5.3.2 OneGate Usage

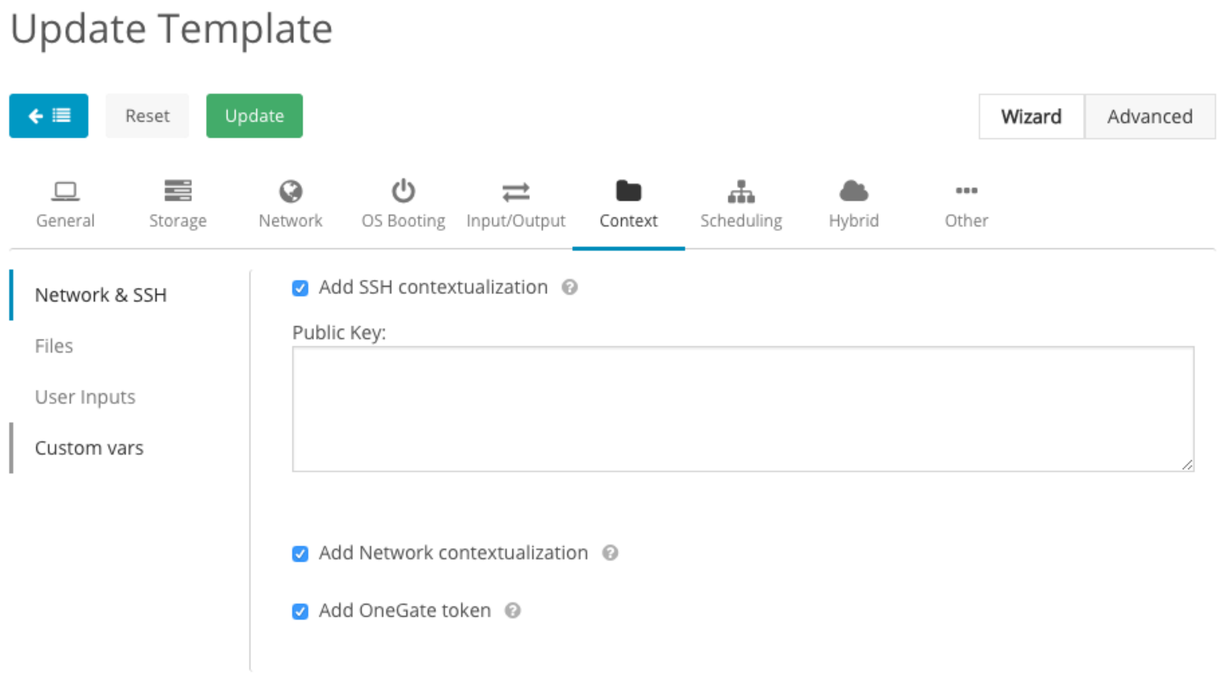
First, the cloud administrator must configure and start the *OneGate server*.

Setup the VM Template

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

```
CPU     = "0.5"
MEMORY  = "1024"

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

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

or check the OneGate checkbox in Sunstone:



☑ Add SSH contextualization

Public Key:

☑ Add Network contextualization

☑ Add OneGate token

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

```
...

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

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

Using the OneGate Client inside the Guest VM

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

OneGate Client Usage

Available commands and usage are shown with `onegate -h`.

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

Self-Awareness

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

Retrieving Information of the VM

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

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

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

Retrieving information of the Service

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

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

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

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

Updating the VM Information

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

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

```
$ onegate vm update 9 --data ACTIVE=YES
$ onegate vm show 9 --json
{
  "VM": {
    "NAME": "slave_0_(service_1)",
    "ID": "9",
    "STATE": "3",
    "LCM_STATE": "3",
    "USER_TEMPLATE": {
      "ACTIVE": "YES",
      "FROM_APP": "4fc76a938fb81d3517000003",
      "FROM_APP_NAME": "ttylinux - kvm",
      "LOGO": "images/logos/linux.png",
      "ROLE_NAME": "slave",
      "SERVICE_ID": "1"
    },
    "TEMPLATE": {
      "NIC": [

      ]
    }
  }
}
```

Self-Configuration

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

Performing actions on a VM

The following actions can be performed in any of the Virtual Machines of the Service.

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

```
$ onegate vm terminate --hard 9
```

Change Service cardinality

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

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

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

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

5.3.3 OneGate API

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

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

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

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

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

- **Headers**:

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

OneGate supports these actions:

Self-awareness

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

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

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

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

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

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

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

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

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

                    },
                    // more nodes ...
                ]
            },
            // more roles ...
        ]
    }
}
```

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

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

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

Self-configuration

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

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

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

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

5.3.4 Sample Application Monitoring Script

```
#!/bin/bash

# -------------------------------------------------------------------------- #
# Copyright 2002-2016, OpenNebula Project, OpenNebula Systems                #
#                                                                            #
# Licensed under the Apache License, Version 2.0 (the "License"); you may    #
# not use this file except in compliance with the License. You may obtain    #
# a copy of the License at                                                   #
#                                                                            #
# http://www.apache.org/licenses/LICENSE-2.0                                 #
#                                                                            #
# Unless required by applicable law or agreed to in writing, software        #
# distributed under the License is distributed on an "AS IS" BASIS,          #
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.   #
# See the License for the specific language governing permissions and        #
# limitations under the License.                                             #
#--------------------------------------------------------------------------- #

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

ERROR=0

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

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

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

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

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

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

```
MEM_USED=$(($MEM_TOTAL-$MEM_FREE))

MEM_USED_PERC="0"

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

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

SWAP_USED_PERC="0"

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


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

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

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

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

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

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

    DISK_USED_PERC="0"

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

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

################################################################################
```

```
# PUT command
################################################################################

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

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

CHAPTER

SIX

PUBLIC CLOUD

6.1 Overview

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.

The *EC2 Query subset* interfaces provide a simple and remote management of cloud (virtual) resources at a high abstraction level. 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.

6.1.1 How Should I Read This Chapter

Before reading this chapter make sure you have read the Deployment Guide.

Read the *EC2 Server Configuration* to understand how to start the EC2 API for OpenNebula. *OpenNebula EC2 User Guide* contains a reference of the supported commands and their usage.

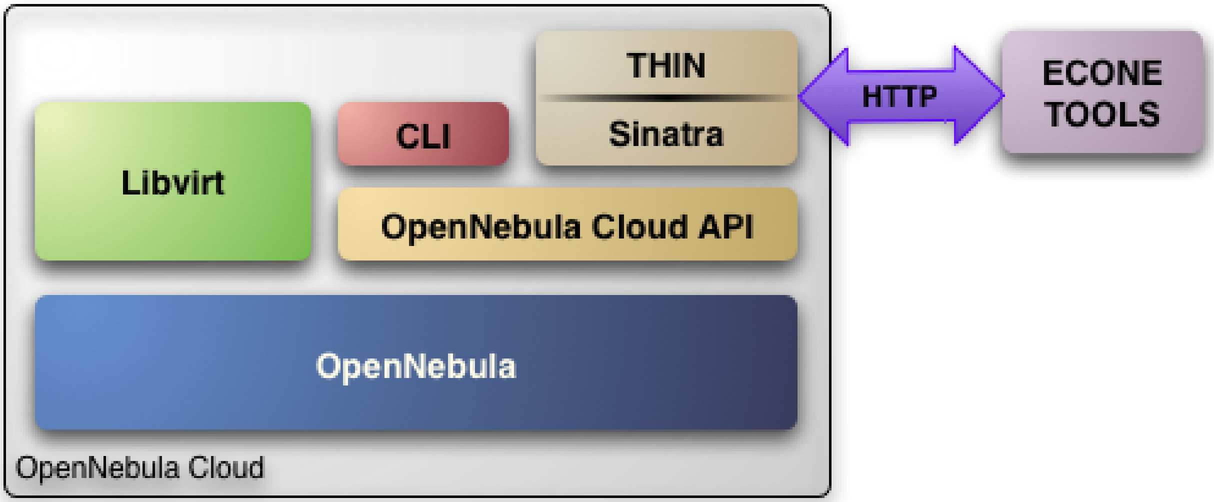
After reading this chapter you can continue configuring more *Advanced Components*.

6.1.2 Hypervisor Compatibility

This Chapter applies both to KVM and vCenter.

6.2 EC2 Server Configuration

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.

Note: 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. The OpenNebula distribution includes the tools needed to use the EC2 Query service.

6.2.1 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

6.2.2 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 available options are:

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 system

File based templates

- use_file_templates: Use former file based templates for instance types instead of OpenNebula templates
- instance_types: DEPRECATED The VM types for your cloud

Resources

- describe_with_terminated_instances: Include terminated instances in the describe_instances xml. When this parameter is enabled all the VMs in DONE state will be retrieved in each describe_instances action and then filtered. This can cause performance issues when the pool of VMs in DONE state is huge
- terminated_instances_expiration_time: Terminated VMs will be included in the list till the termination date + terminated_instances_expiration_time is reached
- datastore_id: Datastore in which the Images uploaded through EC2 will be allocated, by default 1
- cluster_id: Cluster associated with the EC2 resources, by default no Cluster is defined

Elastic IP

- elasticips_vnet_id: VirtualNetwork containing the elastic ips to be used with EC2. If no defined the Elastic IP functionality is disabled
- associate_script: Script to associate a public IP with a private IP arguments: elastic_ip private_ip vnet_template(base64_encoded)
- disassociate_script: Script to disassociate a public IP arguments: elastic_ip

EBS

- 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.

Cloud Users

The cloud users have to be created in OpenNebula 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 user name and AWSSecretAccessKey their OpenNebula's hashed password.

The cloud administrator can limit the interfaces that these users can use to interact with OpenNebula by setting the driver public for them. Using that driver cloud users will not be able to interact with OpenNebula through Sunstone, CLI nor XML-RPC.

```
$ oneuser chauth cloud_user public
```

Defining VM Types

You can define as many Virtual Machine types as you want, just:

- Create a new OpenNebula template for the new type and make it available for the users group. You can use restricted attributes and set permissions like any other OpenNebula resource. **You must include the EC2_INSTANCE_TYPE parameter inside the template definition**, otherwise the template will not be available to be used as an instance type in EC2.

```
# This is the content of the /tmp/m1.small file
NAME = "m1.small"
EC2_INSTANCE_TYPE = "m1.small"
CPU = 1
MEMORY = 1700
...
```

```
$ onetemplate create /tmp/m1.small
$ onetemplate chgrp m1.small users
$ onetemplate chmod m1.small 640
```

The template must include all the required information to instantiate a new virtual machine, such as network configuration, capacity, placement requirements, etc. This information will be used as a base template and will be merged with the information provided by the user.

The user will select an instance type along with the ami id, keypair and user data when creating a new instance. Therefore, **the template should not include the OS**, since it will be specified by the user with the selected AMI.

Note: The templates are processed by the EC2 server to include specific data for the instance.

6.2.3 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
```

6.2.4 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 easiliy 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 she chooses to release it. This way the user can remap his public IP addresses to any of his instances.

In order to enable this functionality you have to follow the following steps in order to create a VNET containing the elastic IPS

- Create a new Virtual Network as oneadmin containing the public IPs that will be controlled by the EC2 users. Each IP **must be placed in its own AR**:

```
NAME    = "ElasticIPs"

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

AR  = [IP=10.0.0.1, TYPE=IP4, SIZE=1]
AR  = [IP=10.0.0.2, TYPE=IP4, SIZE=1]
AR  = [IP=10.0.0.3, TYPE=IP4, SIZE=1]
AR  = [IP=10.0.0.4, TYPE=IP4, SIZE=1]

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

```
$ onevnet create /tmp/fixed.vnet
ID: 8
```

This VNET will be managed by the oneadmin user, therefore `USE` permission for the ec2 users is not required

- Update the econe.conf file with the VNET ID:

```
:elastic_ips_vnet: 8
```

- Provide associate and disassociate scripts

The interaction with the infrastructure has been abstracted, therefore two scripts have to be provided by the cloud administrator in order to interact with each specific network configuration. These 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 mileage may vary, but not a lot):

- Install the `ssl-cert` package

```
$ sudo apt-get install ssl-cert
```

- Generate the certificate

```
$ sudo /usr/sbin/make-ssl-cert generate-default-snakeoil
```

- As we are using lighttpd, we need to append the private key with the certificate to obtain a server certificate valid to lighttpd

```
$ sudo cat /etc/ssl/private/ssl-cert-snakeoil.key /etc/ssl/certs/ssl-cert-snakeoil.
↪pem > /etc/lighttpd/server.pem
```

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.

6.3 OpenNebula EC2 User Guide

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

OpenNebula implements a subset of the EC2 Query interface, enabling the creation of public clouds managed by OpenNebula.

6.3.1 AMIs

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

6.3.2 Instances

- **run instances**: Runs an instance of a particular image (that needs to be referenced).
- **describe instances**: Outputs a list of launched images belonging to one particular user.
- **terminate instances**: Shutdowns a set of virtual machines (or cancel, depending on its state).
- **reboot instances**: Reboots a set of virtual machines.
- **start instances**: Starts a set of virtual machines.
- **stop instances**: Stops a set of virtual machines.

6.3.3 EBS

- **create volume**: Creates a new DATABLOCK in OpenNebula
- **delete volume**: Deletes an existing DATABLOCK.

- **describe volumes**: Describe all available DATABLOCKs for this user
- **attach volume**: Attaches a DATABLOCK to an instance
- **detach volume**: Detaches a DATABLOCK from an instance
- **create snapshot**:
- **delete snapshot**:
- **describe snapshot**:

6.3.4 Elastic IPs

- **allocate address**: Allocates a new elastic IP address for the user
- **release address**: Releases a publicIP of the user
- **describe addresses**: Lists elastic IP addresses
- **associate address**: Associates a publicIP of the user with a given instance
- **disassociate address**: Disassociate a publicIP of the user currently associated with an instance

6.3.5 Keypairs

- **create keypair**: Creates the named keypair
- **delete keypair**: Deletes the named keypair, removes the associated keys
- **describe keypairs**: List and describe the key pairs available to the user

6.3.6 Tags

- **create-tags**
- **describe-tags**
- **remove-tags**

Commands description can be accessed from the Command Line Reference.

User Account Configuration

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

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

- **Authentication**: This can be achieved in three different ways, here listed in order of priority (i.e. values specified in the argument line supersede environmental variables)
  - Using the **commands arguments**. All the commands accept an **Access Key** (as the OpenNebula username) and a **Secret Key** (as the OpenNebula hashed password)
  - Using **EC2_ACCESS_KEY** and **EC2_SECRET_KEY** environment variables the same way as the arguments

   - If none of the above is available, the **ONE_AUTH** variable will be checked for authentication (with the same used for OpenNebula CLI).

- **Server location**: The command need to know where the OpenNebula cloud service is running. That information needs to be stored within the **EC2_URL** environment variable (in the form of a http URL, including the port if it is not the standard 80).

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
$ econe-register ami-00000001
Success: ImageId ami-00000001
```

- **describe_images**

We will need the **ImageId** to launch the image, so in case we forgotten we can list registered images using the **econe-describe-images** command:

```
$ econe-describe-images -H
Owner        ImageId       Status         Visibility   Location
------------------------------------------------------------------------------
helen        ami-00000001  available      private     ␣
 ↪19ead5de585f43282acab4060bfb7a07
```

- **run_instance**

Once we recall the ImageId, we will need to use the **econe-run-instances** command to launch an Virtual Machine instance of our image:

```
$ econe-run-instances -H ami-00000001
Owner       ImageId                InstanceId InstanceType
------------------------------------------------------------------------------
helen       ami-00000001           i-15       m1.small
```

We will need the **InstanceId** to monitor and shutdown our instance, so we better write down that `i-15`.

- **describe_instances**

If we have too many instances launched and we don't remember everyone of them, we can ask **econe-describe-instances** to show us which instances we have submitted.

```
$ econe-describe-instances  -H
Owner       Id    ImageId      State         IP              Type
--------------------------------------------------------------------------------------
 ↪----------------------
helen       i-15  ami-00000001 pending       147.96.80.33    m1.small
```

We can see that the instances with Id i-15 has been launched, but it is still pending, i.e., it still needs to be deployed into a physical host. If we try the same command again after a short while, we should be seeing it running as in the following excerpt:

```
$ econe-describe-instances  -H
Owner       Id    ImageId      State         IP              Type
--------------------------------------------------------------------------------------
↪----------------------
helen       i-15  ami-00000001 running      147.96.80.33     m1.small
```

- **terminate_instances**

After we put the Virtual Machine to a good use, it is time to shut it down to make space for other Virtual Machines (and, presumably, to stop being billed for it). For that we can use the **econe-terminate-instances** passing to it as an argument the **InstanceId** that identifies our Virtual Machine:

```
$ econe-terminate-instances i-15
Success: Terminating i-15 in running state
```

Note: You can obtain more information on how to use the above commands accessing their Usage help passing them the **-h** flag

CHAPTER

SEVEN

MARKETPLACE

7.1 Overview

Sharing, provisioning and consuming cloud images is one of the main concerns when using Cloud. OpenNebula provides a simple way to create and integrate with a cloud image provider, called MarketPlaces. Think of them as external datastores.

A MarketPlace can be:

- **Public**: accessible universally by all OpenNebula's.
- **Private**: local within an organization and specific for a single OpenNebula (a single zone) or shared by a federation (a collection of zones).

A MarketPlace is a repository of MarketPlaceApps. A MarketPlaceApp can be thought of as an external Image optionally associated to a Virtual Machine Template.

Using MarketPlaces is very convenient, as it will allow you to move images across different kinds of datastores (using the MarketPlace as an exchange point), it is a way to share OpenNebula images in a Federation, as these resources are federated. In an OpenNebula deployment where the different VDCs don't share any resources, a MarketPlace will act like a shared datastore for all the users.

7.1.1 Supported Actions

MarketPlaces support various actions:

Action	Description
`create`	Create a new MarketPlace.
`monitor`	This automatic action, discovers the available MarketPlaceApps and monitors the available space of the MarketPlace.
`delete`	Removes a MarketPlace from OpenNebula. For the Publick MarketPlace, it will also remove the MarketPlaceApps, but for any other type of MarketPlace this will not remove the MarketPlaceApps, and will only work if the MarketPlace is empty.
other	Generic actions common to OpenNebula resources are also available: update, chgrp, chown, chmod and rename.

As for the MarketPlaceApps, they support these actions:

Action	Description
`create`	Upload a local image into the the MarketPlace.
`export`	Export the MarketPlaceApp and download it into a local Datastore.
`delete`	Removes a MarketPlaceApp.
`download`	Downloads a MarketPlaceApp to a file.
other	Generic actions common to OpenNebula resources are also available: update, chgrp, chown, chmod, rename, enable and disable.

Warning: In order to use the `download` functionality make sure you read the Sunstone Advanced Guide.

7.1.2 Backends

MarketPlaces store the actual MarketPlaceApp images. How they do so depends on the backend driver. Currently these drivers are shipped with OpenNebula:

Driver	Upload	Description
one	No	This driver allows read access to the official public OpenNebula Systems Marketplace, as well as to the OpenNebula AppMarket Add-on.
http	Yes	When an image is uploaded to a MarketPlace of this kind, the image is written into a file in a specified folder, which is in turn available via a web-server.
S3	Yes	Images are stored into a S3 API-capable service. This means it can be stored in the official AWS S3 service , or in services that implement that API like Ceph Object Gateway S3

OpenNebula ships with the OpenNebula Systems MarketPlace pre-registered, so users can access it directly.

7.1.3 Use Cases

Using the MarketPlace is recommended in many scenarios, to name a few:

- When starting with an empty OpenNebula, the public OpenNebula Systems Marketplace contains a catalog of OpenNebula-ready cloud images, allowing you to get on your feet very quickly.
- You can upload an image into a MarketPlace, and download it later on to another Datastores even if the source and target Datastores are of a different type, thus enabling image cloning from any datastore to any other datastore.
- In a federation, it is almost essential to have a shared MarketPlace in order to share MarketPlaceApps across zones.
- MarketPlaces are a great way to provide content for the users in VDCs with no initial virtual resources.

7.1.4 How Should I Read This Chapter

Before reading this chapter make sure you have read the Deployment Guide.

Read the *OpenNebula Systems MarketPlace* as it's global for all the OpenNebula installations. Then read the specific guide for the MarketPlace flavor you are interested in. Finally, read the *Managing MarketPlaceApps* to understand what operations you can perform on MarketPlaceApps.

After reading this chapter you can continue configuring more *Advanced Components*.

7.1.5 Hypervisor Compatibility

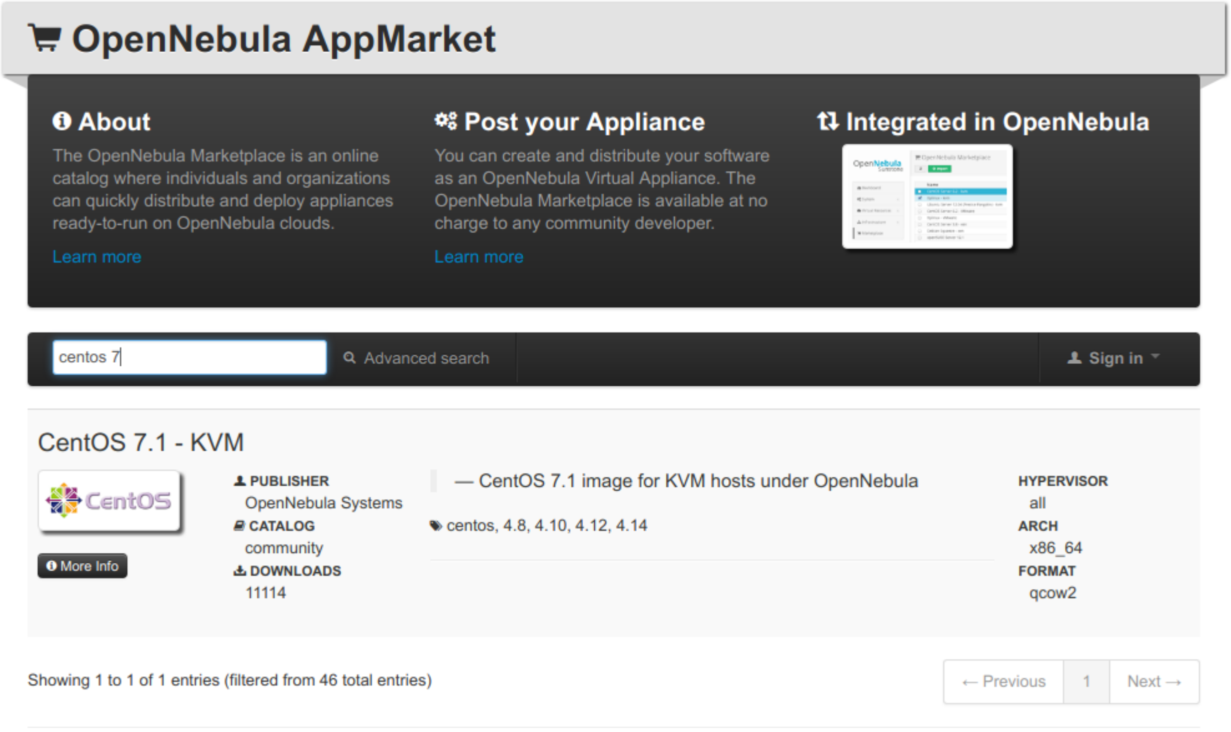
This chapter applies only to KVM.

7.2 OpenNebula Systems MarketPlace

7.2.1 Overview

OpenNebula Systems provides a public and official MarketPlace, universally available to all the OpenNebula's. The OpenNebula Marketplace is a catalog of third party virtual appliances ready to run in OpenNebula environments. This MarketPlace is available here: http://marketplace.opennebula.systems. Anyone can request an account and upload their appliances and share them with other OpenNebula's, however, as opposed to other MarketPlaces, MarketPlaceApp creation is not done through OpenNebula, but by following the instructions in http://marketplace.opennebula.systems. Deletion of MarketPlaceApps is likewise limited.

The MarketPlaceApps included in the official MarketPlace are third-party contributions of other OpenNebula users, meaning that they are not certificated by the OpenNebula project.



OpenNebula AppMarket

About

The OpenNebula Marketplace is an online catalog where individuals and organizations can quickly distribute and deploy appliances ready-to-run on OpenNebula clouds.

Learn more

Post your Appliance

You can create and distribute your software as an OpenNebula Virtual Appliance. The OpenNebula Marketplace is available at no charge to any community developer.

Learn more

Integrated in OpenNebula

centos 7

CentOS 7.1 - KVM

PUBLISHER
OpenNebula Systems
CATALOG
community
DOWNLOADS
11114

— CentOS 7.1 image for KVM hosts under OpenNebula

centos, 4.8, 4.10, 4.12, 4.14

HYPERVISOR
all
ARCH
x86_64
FORMAT
qcow2

Showing 1 to 1 of 1 entries (filtered from 46 total entries)

You can also connect to MarketPlaces deployed with the OpenNebula Add-on AppMarket. The already deployed AppMarkets can still be used, but they are now deprecated in favor of the *HTTP MarketPlaces*.

7.2.2 Requirements

The url http://marketplace.opennebula.systems must be reachable from the OpenNebula Frontend.

7.2.3 Configuration

The Official OpenNebula Systems Marketplace is pre-registered in OpenNebula:

```
$ onemarket list
ID NAME                                  SIZE AVAIL        APPS MAD     ZONE
 0 OpenNebula Public                       0M -              46 one        0
```

Therefore it does not require any additional action from the administrator.

However, to connect to OpenNebula Add-on AppMarkets, it is possible to do so by creating a new MarketPlace template with the following attributes:

Attribute	Description
NAME	Required
MARKET_MAD	Must be one.
ENDPOINT	(**Required to connect to AppMarket**) The URL of AppMarket.

For example, the following examples illustrates the creation of a MarketPlace:

```
$ cat market.conf
NAME = PrivateMarket
MARKET_MAD = one
ENDPOINT = "http://privatemarket.opennebula.org"

$ onemarket create market.conf
ID: 100
```

7.2.4 Tuning & Extending

System administrators and integrators are encouraged to modify these drivers in order to integrate them with their datacenter. Please refer to the Market Driver Development guide to learn about the driver details.

7.3 HTTP MarketPlace

7.3.1 Overview

MarketPlaces of *HTTP* make use of a conventional HTTP server to expose the images (MarketPlaceApps) uploaded to a MarketPlace of this kind. The image will be placed in a specific directory that must be configured to be exposed by HTTP.

This is a fully supported MarketPlace with all the implemented features.

7.3.2 Requirements

A web-server should be deployed either in the Frontend or in a node reachable by the Frontend. A directory that will be used to store the uploaded images (MarketPlaceApps) should be configured to have the desired available space, and the web-server must be configured in order to grant HTTP access to that directory.

It is recommended to use either Apache or NGINX as they are known to work properly with the potentially large size of the MarketPlaceApp files. However, other web servers may work as long as they're capable to handle the load.

The web-server should be deployed by the administrator before registering the MarketPlace.

7.3.3 Configuration

These are the configuration attributes of a MarketPlace template of the *HTTP* kind.

Attribute	Description
`NAME`	Required
`MARKET_MAD`	Must be `http`
`PUBLIC_DIR`	(**Required**) Absolute directory path to place images, the document root for http server, in the Frontend or in the hosts pointed at by the `BRIDGE_LIST` directive.
`BASE_URL`	(**Required**) URL base to generate MarketPlaceApp endpoints.
`BRIDGE_LIST`	(Optional) Comma separated list of servers to access the public directory. If not defined, public directory will be local to the Frontend.

For example, the following examples illustrates the creation of an MarketPlace:

```
$ cat market.conf
NAME = PrivateMarket
MARKET_MAD = http
BASE_URL = "http://frontend.opennebula.org/"
PUBLIC_DIR = "/var/loca/market-http"
BRIDGE_LIST = "web-server.opennebula.org"

$ onemarket create market.conf
ID: 100
```

7.3.4 Tuning & Extending

System administrators and integrators are encouraged to modify these drivers in order to integrate them with their datacenter. Please refer to the Market Driver Development guide to learn about the driver details.

7.4 S3 MarketPlace

7.4.1 Overview

This MarketPlace uses an S3 API-capable service as the backend. This means MarketPlaceApp images will be stored in the official AWS S3 service , or in services that implement that API like Ceph Object Gateway S3.

7.4.2 Limitations

Since the S3 API does not provide the available space, this space is hard-coded into the driver file, limiting it to 1TB. See below to learn how to change the default value.

7.4.3 Requirements

To use this driver you require access to an S3 API-capable service.

- If you want to use AWS Amazon S3, you can start with the Getting Started guide.
- For Ceph S3 you must follow the Configuring Ceph Object Gateway guide.

Make sure you obtain both an *access_key* and a *secret_key* of a user that has access to a bucket with the exclusive purpose of storing MarketPlaceApp images.

7.4.4 Configuration

These are the configuration attributes of a MarketPlace template of the *S3* kind:

Attribute	Description
`NAME`	Required
`MARKET_MAD`	Must be `s3`
`ACCESS_KEY_ID`	(**Required**) The access key of the S3 user.
`SECRET_ACCESS_KEY`	(**Required**) The secret key of the S3 user.
`BUCKET`	(**Required**) The bucket where the files will be stored.
`REGION`	(**Required**) The region to connect to. If you are using Ceph-S3 any value here will work.
`ENDPOINT`	(Optional) This is only required if you are connecting to a service other than Amazon AWS S3 service. Preferably don't use an endpoint that includes the bucket as leading part of the host's url.
`SIGNATURE_VERSION`	(Optional) Leave blank for Amazon AWS S3 service. If connecting to Ceph S3 it **must** be `s3`.
`FORCE_PATH_STYLE`	(Optional) Leave blank for Amazon AWS S3 service. If connecting to Ceph S3 it **must** be `YES`.
`TOTAL_MB`	(Optional) This parameter defines the Total size of the MarketPlace in MB. It defaults to `1024 GB`.
`READ_LENGTH`	(Optional) Split the file into chunks of this size (in MB). You should **never** user a quantity larger than *100*. Defaults to *32* (MB).

For example, the following examples illustrates the creation of an MarketPlace:

```
$ cat market.conf
NAME=S3CephMarket
ACCESS_KEY_ID="I0PJDPCIYZ665MW88W9R"
SECRET_ACCESS_KEY="dxaXZ8U90SXydYzyS5ivamEP20hkLSUViiaR"
BUCKET="opennebula-market"
ENDPOINT="http://ceph-gw.opennebula.org"
FORCE_PATH_STYLE="YES"
MARKET_MAD=s3
REGION="default"
SIGNATURE_VERSION=s3

$ onemarket create market.conf
ID: 100
```

7.4.5 Tuning & Extending

In order to change the available size of the MarketPlace from 1TB to your desired value, you can modify */var/lib/one/remotes/market/s3/monitor* and change:

```
TOTAL_MB_DEFAULT = 1048576 # Default maximum 1TB
```

System administrators and integrators are encouraged to modify these drivers in order to integrate them with their datacenter. Please refer to the Market Driver Development guide to learn about the driver details.

7.5 Managing MarketPlaceApps

7.5.1 Overview

As mentioned in the *MarketPlace Overview*, a MarketPlaceApp is a resource that can be either a single Image associated with a template, a template associated with one or more images, or a flow composed of one or more templates associated with images. In short, MarketPlaceApps are composed of metadata (a template) and binary data (the images). This guide addresses how you can manage these MarketPlaceApps in OpenNebula, considering both the metadata and the images.

MarketPlaceApps can be managed either using the CLI with the `onemarketapp` command or with the Sunstone GUI. In this section we will detail the actions available for MarketPlaceApps in both interfaces. MarketPlaceApps are common OpenNebula objects and respond the the common actions shared by all the OpenNebula objets: *list*, *show*, *create*, *delete*, etc, plus some specific ones.

7.5.2 Listing MarketPlaceApps

Using the CLI:

```
$ onemarketapp list
 ID NAME                         VERSION  SIZE STAT TYPE  REGTIME               MARKET
  0 ttylinux - kvm                   1.0   40M  rdy  img 06/08/46    OpenNebula Public
  1 ttylinux - VMware                1.1  102M  rdy  img 08/08/16    OpenNebula Public
  2 Carina Environment Manage        1.0  1.2G  rdy  img 05/14/26    OpenNebula Public
  3 Testing gUSE installation        1.0   16G  rdy  img 07/22/86    OpenNebula Public
  4 gUse v3.5.2                    3.5.2   16G  rdy  img 04/02/43    OpenNebula Public
  5 Vyatta Core 6.5R1 - kvm          1.0    2G  rdy  img 07/22/86    OpenNebula Public
  6 gUSE CloudBroker Wrapper         1.0   16G  rdy  img 04/08/43    OpenNebula Public
  7 debian-7.1-amd64-kvm             1.3    5G  rdy  img 07/22/86    OpenNebula Public
  8 Hadoop 1.2 Master                1.0  1.3G  rdy  img 04/07/43    OpenNebula Public
  9 Hadoop 1.2 Slave                 1.0  1.3G  rdy  img 05/18/14    OpenNebula Public
```

Todo

Sunstone

7.5.3 Show a MarketPlaceApp

Using the CLI:

```
$ onemarketapp show 0
MARKETPLACE APP 0 INFORMATION
ID             : 0
NAME           : ttylinux - kvm
TYPE           : IMAGE
USER           : oneadmin
GROUP          : oneadmin
MARKETPLACE    : OpenNebula Public
STATE          : rdy

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

```
OTHER          : u--

DETAILS
SOURCE         : http://marketplace.opennebula.systems//appliance/
↪4fc76a938fb81d3517000003/download/0
MD5            : 04c7d00e88fa66d9aaa34d9cf8ad6aaa
PUBLISHER      : OpenNebula.org
PUB. DATE      : Wed Jun  8 22:17:19 137435166546
VERSION        : 1.0
DESCRIPTION    : This is a very small image that works with OpenNebula. It's already␣
↪contextualized. The purpose of this image is to test OpenNebula deployments,␣
↪without wasting network bandwith thanks to the tiny footprint of this image
(40MB).
SIZE           : 40M
ORIGIN_ID      : -1
FORMAT         : raw

IMPORT TEMPLATE


MARKETPLACE APP TEMPLATE
IMPORTED="YES"
IMPORT_ID="4fc76a938fb81d3517000003"
TAGS="linux, ttylinux,  4.8,  4.10"
VMTEMPLATE64=
↪"Q09OVEVYVCA9IFsgTkVUV09SSyAgPSJZRVMiLFNTSF9QVUJMSUNfS0VZICA9IiRVU0VSW1NTSF9QVUJMSUNfS0VZXSJdCgpDU
↪"
```

Not that if we unpack that *VMTEMPLATE64* we obtain the following:

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

CPU = "0.1"
GRAPHICS = [ LISTEN  ="0.0.0.0",TYPE  ="vnc"]

MEMORY = "128"
LOGO = "images/logos/linux.png"
```

Which demonstrates the capability of including a template into the appliance's data.

Todo

sunstone

7.5.4 Create a New MarketPlaceApp

In order to create a MarketPlaceApp you will need to prepare a new template file with the following attributes:

Attribute	Description
NAME	Required
ORIGIN_ID	(**Required**) The ID of the source image. It must reference an available image and it must be in one of the supported datastores.
TYPE	(**Required**) Must be IMAGE.
MARKETPLACE_ID	(**Required**) The target marketplace ID. Alternatively you can specify the MARKETPLACE name.
MARKETPLACE	(**Required**) The target marketplace name. Alternatively you can specify the MARKETPLACE_ID name.
DESCRIPTION	(Optional) Text description of the MarketPlaceApp.
PUBLISHER	(Optional) If not provided, the username will be used.
VERSION	(Optional) A string indicating the MarketPlaceApp version.
VMTEMPLATE64	(Optional) Creates this template (encoded in base64) pointing to the base image.
APPTEMPLATE64	(Optional) This is the image template (encoded in base64) that will be added to the registered image. It is useful to include parameteres like DRIVER or DEV_PREFIX.

Example:

```
$ cat marketapp.tpl
NAME=TTYlinux
ORIGIN_ID=0
TYPE=image

$ onemarketapp create marketapp.tpl -m "OpenNebula Public"
ID: 40
```

Todo

sunstone

7.5.5 Exporting a MarketPlaceApp

Using the CLI:

The command that exports the MarketPlaceApp is *onemarketapp export* which will return the ID of the new Image **and** the ID of the new associated template. If no template has been defined, it will return *-1*.

```
$ onemarketapp export 40 from_t1app -d 1
IMAGE
    ID: 1
VMTEMPLATE
    ID: -1
```

Todo

Sunstone

7.5.6 Downloading a MarketPlaceApp

To downloadd a MarketPlaceApp to a file:

```
$ onemarketapp download 40 /path/to/app
```

Warning: This command requires that the *ONE_SUNSTONE* environment variable is set. Read here for more information.

Warning: Make sure the Sunstone is properly deployed to handle this feature. Read here for more information.

7.5.7 Additional Commands

Like any other OpenNebula Resource, MarketPlaceApps respond to the base actions, namely:

- delete
- update
- chgrp
- chown
- chmod
- enable
- disable

Please take a look at the CLI reference to see how to use these actions. In Sunstone this options are also available.

Tuning & Extending

System administrators and integrators are encouraged to modify these drivers in order to integrate them with their datacenter. Please refer to the Market Driver Development guide to learn about the driver details.