CENTRE DE SERVEIS CIENTÍFICS
I ACADÈMICS DE CATALUNYA

CESCA | 20 ANYS 1991·2011

Guía Básica instalación OpenNebula 3.0 – Servicio de Cloud Pública (Beta)

Para la instalación del OpenNebula, lo haremos sobre las siguientes máquinas: una virtual con Ubuntu Server 10.04 en la que se instalará el OpenNebula (máquina ONE) y dos máquinas físicas con capacidad de alojar máquinas virtuales sobre un Ubuntu Server 10.04 sobre las que pondremos el KVM, donde serán lanzadas las maquinas virtuales (máquinas trabajadoras).

Tendremos 3 redes diferentes:

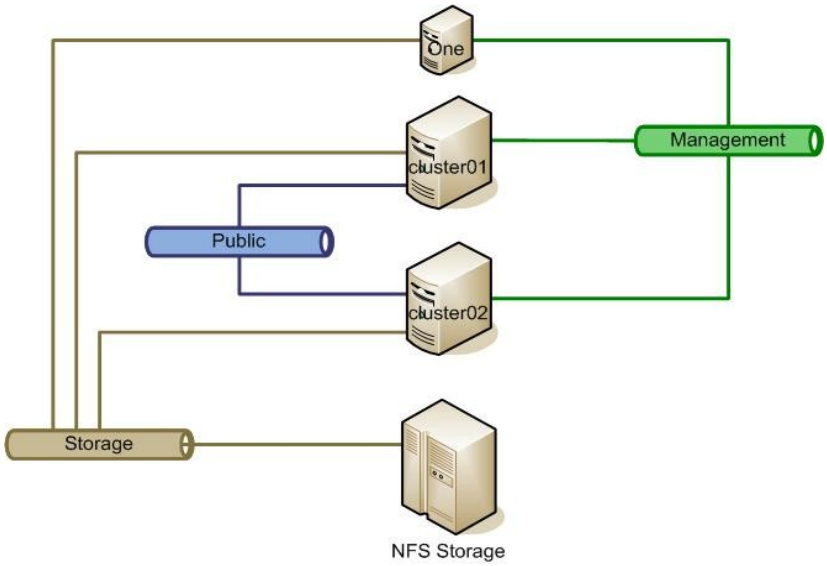
Storage: rango para llegar al storage NFS. (192.168.1.0/24)

Management: rango para management de las máquinas i comunicación entre ellas (192.168.2.0/24)

Pública: rango para las máquina virtuales. (8.8.8.0/24)

No entraremos en detalle en la configuración del almacenamiento, dónde supondremos que disponemos de un servidor NAS que exporta un volumen de NFS des de la IP 192.168.1.100.

El esquema es el siguiente:



Configuración de red, NFS

Máquina OpenNebula

En esta disponemos de dos interfícies de red, una para management (192.168.2.10) y otra para acceso al almacenamiento (192.168.1.10).

```
# cat /etc/network/interfaces

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
            address 192.168.2.10
            netmask 255.255.255.0
            gateway 192.168.2.1
            dns-nameservers 192.168.2.2 192.168.2.3

auto eth1
iface eth1 inet static
            address 192.168.1.10
            netmask 255.255.255.0
```

Configuración del NFS y hosts:

```
# echo “192.168.1.100 nfscloud ” >> /etc/hosts
# echo “192.168.2.11 cluster01 ” >> /etc/hosts
# echo “192.168.2.12 cluster02 ” >> /etc/hosts
# echo “nfscloud:/vol/cloud_proves /var/lib/one nfs defaults 0 0” >> /etc/fstab
# apt-get install nfs-common
# mount -a
```

Máquinas Container

Para las dos máquinas de tipo container, tenemos cuatro interfícies de red, red pública que entraremos con VLAN tagging y le haremos un bridge (eth0 – sin Ips, VLAN 888 sólo para las maquinas virtuales, bridge br888) , una de management (eth2 - 192.168.2.11 y 192.168.2.12) y dos interfícies con bonding para el storage (bonding VLAN 001, eth1+eth3 - 192.168.1.11 y 192.168.1.12).

Instalamos herramientas de red:

```
apt-get install bridge-utils vlan ethtool ifenslave
```

Esta es la configuración para una máquina trabajadora 192.168.2.11 (la otra será lo mismo cambiando las IPs correspondientes):

```
# cat /etc/network/interfaces

auto lo
```

```
iface lo inet loopback

auto eth0.888
iface eth0.888 inet manual
        up ifconfig eth0.888 up

auto br888
iface br888 inet manual
        bridge_ports eth0.888
        bridge_maxwait 5
        bridge_fd 1
        bridge_stp on

auto eth2
iface eth2 inet static
        address 192.168.2.11
        netmask 255.255.255.0
        gateway 192.168.2.1

auto bond0
iface bond0 inet manual
        post-up ifconfig $IFACE up
        pre-down ifconfig $IFACE down
        bond-slaves none
        bond-mode active-backup
        bond-miimon 100
        bond-downdelay 200
        bond-updelay 100

allow-bond0 eth1
iface eth1 inet manual
        bond-master bond0

auto eth3
allow-bond0 eth3
iface eth3 inet manual
        bond-master bond0

auto bond0.001
iface bond0.001 inet manual
   post-up ifconfig $IFACE up
   pre-down ifconfig $IFACE down
   vlan-raw-device bond0

auto br001
iface br001 inet static
   post-up ifconfig $IFACE up
   pre-down ifconfig $IFACE down
   bridge_ports bond0.001
   bridge_stp off
   bridge_fd 0
   bridge_maxwait 0
```

```
        address 192.168.1.11
        netmask 255.255.255.0
```

Configuración del NFS y hosts:

```
# echo “192.168.1.100 nfscloud ” >> /etc/hosts
# echo “192.168.2.11 cluster01 ” >> /etc/hosts
# echo “192.168.2.12 cluster02 ” >> /etc/hosts
# echo “nfscloud:/vol/cloud_proves /var/lib/one nfs defaults 0 0” >> /etc/fstab
# apt-get install nfs-common
# mount -a
```

Instalación KVM

Instalamos KVM en las máquinas de tipo container:

```
apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils
```

En los nodos trabajadores, tenemos que activar en la BIOS la virtualización hardware, para ver si la tenemos activada, ejecutamos:

```
# kvm-ok
```

Tendría que darnos una salida parecida a lo siguiente:

```
INFO: Your CPU supports KVM extensions
INFO: /dev/kvm exists
KVM acceleration can be used
```

Instalación OpenNebula

Máquina OpenNebula

Instalamos ruby:

```
# apt-get install ruby rubygems libopenssl-ruby ruby-dev
```

Instalamos One:

```
# wget "http://dev.opennebula.org/packages/opennebula-3.0.0/Ubuntu\
-10.04/opennebula_3.0.0-1_amd64.deb"
# dpkg -i opennebula_3.0.0-1_amd64.deb
# apt-get -f install
# gem install rubygems-update
# cd /var/lib/gems/1.8/bin
# ./update_rubygems
# /usr/share/one/install_gems
```

Instalamos MySQL:

```
# apt-get install mysql-server libmysql++-dev libxml2-dev
# mysql -u root -p
> GRANT ALL PRIVILEGES ON opennebula.* TO 'oneadmin' IDENTIFIED BY 'pwd_one';
```

Añadimos configuración para mysql a /etc/one/oned.conf:

```
DB = [ backend = "mysql",
       server  = "localhost",
       port    = 0,
       user    = "oneadmin",
       passwd  = "pwd_one",
       db_name = "opennebula" ]
```

Configuración usuario y contraseña en usuario de sistema operativo oneadmin:

```
mkdir -p /var/lib/one/.one
echo "oneadmin:pwd_one" > .one/one_auth
cp /var/lib/one/.one/one_auth /var/lib/one/auth
echo "export ONE_LOCATION=/var/lib/one
export ONE_AUTH=$HOME/.one/one_auth">>/var/lib/one/.bashrc
```

Configuramos SSH, teniendo en cuenta que /var/lib/one es compartido entre todos los nodos, sólo hará falta hacerlo en el nodo ONE, y todos los usuarios tendrán la misma clave pública:

```
mkdir -p /var/lib/one/.ssh
cd /var/lib/one/.ssh
cat id_rsa.pub >> authorized_keys
echo "Host *
   StrictHostKeyChecking no " >> config
touch /var/lib/one/.hushlogin
```

Configuramos los drivers Information, Virtualization y transfer, aunque lo que se explica a continuación ya es como viene por defecto con opennebula, es interesante repasarlo ya que puede ser útil si por ejemplo queremos añadir soporte para Xen o otros storages no compartidos,...

Information driver:

El information manager es el driver para monitorizar los hosts que añadiremos como trabajadores, para soporte con **KVM** añadimos al fichero /etc/one/oned.conf:

```
IM_MAD = [
      name       = "im_kvm",
      executable = "one_im_ssh",
      arguments  = "-r 0 -t 15 kvm" ]
```

Soporte **XEN**:

```
IM_MAD = [
   name      = "im_xen",
   executable = "one_im_ssh",
   arguments  = "xen" ]
```

Soporte **EC2**:

```
IM_MAD = [
     name      = "im_ec2",
     executable = "one_im_ec2",
     arguments  = "im_ec2/im_ec2.conf" ]
```

Virtualization Driver:

El virtualization driver es el que se encarga de comunicarse con el hypervisor de las máquinas, para tener soporte con hosts **KVM**:

```
VM_MAD = [
   name      = "vmm_kvm",
   executable = "one_vmm_exec",
   arguments  = "-t 15 -r 0 kvm",
   default    = "vmm_exec/vmm_exec_kvm.conf",
   type       = "kvm" ]
```

Para tener soporte con **Xen**:

```
VM_MAD = [
   name      = "vmm_xen",
   executable = "one_vmm_exec",
   arguments  = "-t 15 -r 0 xen",
   default    = "vmm_exec/vmm_exec_xen.conf",
   type       = "xen" ]
```

Para soporte **EC2**:

```
VM_MAD = [
   name      = "vmm_ec2",
   executable = "one_vmm_ec2",
   arguments  = "vmm_ec2/vmm_ec2.conf",
   type       = "xml" ]
```

Transfer Driver (Shared Storage):

El transfer driver es el modulo para mover las imágenes entre nodos, en nuestro caso, como tenemos NFS, será el **shared**:

```
TM_MAD = [
   name      = "tm_shared",
   executable = "one_tm",
   arguments  = "tm_shared/tm_shared.conf" ]
```

Si tenemos máquinas sin storage compartido, podemos añadir el driver para copia por

SSH:

```
TM_MAD = [
    name       = "tm_ssh",
    executable = "one_tm",
    arguments  = "tm_ssh/tm_ssh.conf" ]
```

Para terminar ejecutamos el siguiente comando y nos apuntamos la salida, ya que la necesitaremos en el siguiente paso.

```
#id oneadmin
uid=107(oneadmin) gid=113(oneadmin) groups=113(oneadmin)
```

Máquinas de tipo Container

Teniendo en cuenta, la salida del último comando, añadimos el usuario a la máquinas trabajadoras con el mismo id y grupos con el mismo gid:

```
root@oned:~# groupadd –gid 113 oneadmin
root@oned:~# useradd –uid 107 -g 113 -d /var/lib/one oneadmin
root@oned:~# usermod -a -G libvirtd oneadmin
root@oned:~# usermod -a -G kvm oneadmin
```

Una vez configuradas las máquinas trabajadoras, probamos des de la máquina One, si podemos acceder con el usuario oneadmin a las trabajadoras sin contraseña:

```
oneadmin@one:~$ ssh cluster01
oneadmin@cluster01:~$ exit
oneadmin@one:~$ ssh cluster02
oneadmin@cluster02:~$ exit
```

Añadimos las máquinas trabajadoras como hosts de One, des de la máquina One ejecutamos:

```
oneadmin@one:~$ onehost create cluster01 im_kvm vmm_kvm tm_shared
oneadmin@one:~$ onehost create cluster02 im_kvm vmm_kvm tm_shared
```

Para crear un host se llama al create y como parámetros, el host, el information driver, el virtualization driver y el transfer_driver a utilizar.

Sunstone (GUI OpenNebula)

Para la instalación del sunstone, editamos el fichero de configuración /etc/one/sunstone-server.conf

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

:host: 127.0.0.1
:port: 9869

:auth: basic
```

```
:vnc_proxy_base_port: 29876
:novnc_path: /usr/share/one/noVNC
```

Así podremos editar el puerto por el que queremos que escuche el sunstone y el puerto VNC base, mas la configuración de VNC la cual no funcionará sin el siguiente paso.

NoVNC

Instalación NoVNC para poder ver por el Sunstone las máquinas virtuales:

```
root@oned:~# cd /usr/share/one
root@oned:~# sudo ./install_novnc.sh
```

Una vez configurado el sunstone, ya podemos levantarlo (siempre con usuario oneadmin):

```
oneadmin@oned:~$ sunstone-server start
```

Interfície OCCI

Para la instalación de la interfície OCCI, este es el fichero de configuración /etc/one/occi-server.conf

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

:server: localhost
:port: 4569

:bridge: br888

:auth: basic

:instance_types:
  :custom:
    :template: custom.erb
  :small:
    :template: small.erb
  :medium:
    :template: medium.erb
  :large:
    :template: large.erb
```

A tener en cuenta, en esta configuración, el puerto, el bridge, en nuestro caso el br888 es el bridge por defecto dónde se crearán las redes virtuales. En nuestro caso, las crearemos por defecto en el brdige br888.

Una vez configurado el sunstone, ya podemos levantarlo (siempre con usuario oneadmin):

```
oneadmin@oned:~$ occi-server start
```

Interfície EC2

Empezamos instalando las librerías de ruby necesarias:

```
root@oned:~# gem install amazon-ec2
root@oned:~# gem install sinatra
root@oned:~# gem install thin
root@oned:~# gem install uuid
root@oned:~# gem install curb
root@oned:~# gem install sqlite3-ruby
root@oned:~# apt-get install libsqlite3-ruby libcurl4-gnutls-dev
root@oned:~# apt-get install libopenssl-ruby1.8
```

Éste és el fichero de configuración /etc/one/econe.conf

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

:server: localhost
:port: 4568

:auth: ec2

:instance_types:
  :m1.small:
    :template: m1.small.erb
```

Añadimos a la plantilla small la red por defecto (fichero /etc/one/ec2query_templates/m1.small.erb):

```
NIC=[NETWORK_ID=1]
```

Vemos cómo por el momento sólo hay definido el template small, si quisiéramos definir de nuevos, por ejemplo medium, añadir al final de la configuración:

```
  :m1.medium:
    :template: m1.medium.erb
```

y creamos el fichero /etc/one/ec2query_templates/m1.medium.erb con la plantilla que deseemos.

Debido a un fallo en el One, a través del HybridFox nos da un error de “User not Authorized” cuándo esta bien configurado, para que no pase modificamos el fichero /usr/lib/one/ruby/cloud/CloudAuth/EC2CloudAuth.rb

Buscamos las siguientes lineas:

```
  def signature_v1(params, secret_key, digest='sha1')
      params.delete('Signature')
```

y añadimos:

```
        params.delete(:econe_host)
```

```
      params.delete(:econe_port)
```

Con lo que queda:

```
  def signature_v1(params, secret_key, digest='sha1')
     params.delete('Signature')
     params.delete(:econe_host)
     params.delete(:econe_port)
```

Una vez configurado, ya podemos levantarlo (siempre con usuario oneadmin):

```
oneadmin@oned:~$ econe-server start
```

Para comprobar que el EC2 está funcionando:

```
oneuser list
```

y copiamos la columna password del usuario que queramos:

```
export EC2_ACCESS_KEY=user
export EC2_SECRET_KEY=<passwd_copiado>
export EC2_URL=http://localhost:4568/
euca-describe-images
```

Compatibilidad para qcow

Para tener la posibilidad de crear imágenes con qcow2, modificamos los siguientes ficheros:

OCCI

Modificamos el fichero /etc/one/occi_templates/common.erb buscamos el siguiente texto:

```
   <% if disk.attr('STORAGE','href') %>
    DISK = [ IMAGE_ID = <%= disk.attr('STORAGE','href').split('/').last %>
        ]
    <%end%>
```

y le añadimos delante:

```
<% if disk.attr('DRIVER','type') %>
     DISK = [ IMAGE_ID = <%= disk.attr('STORAGE','href').split('/').last %>,
    DRIVER= <%= disk.attr('DRIVER','type') %>
        ]
   <%else %>
```

Y eso detrás:

```
   <% end %>
```

Con lo que todo el bloque entero queda:

```
<% @vm_info.each('DISK') do |disk| %>
   <% if disk.attr('DRIVER','type') %>
     DISK = [ IMAGE_ID = <%= disk.attr('STORAGE','href').split('/').last %>,
    DRIVER= <%= disk.attr('DRIVER','type') %>
         ]
   <%else %>
   <% if disk.attr('STORAGE','href') %>
     DISK = [ IMAGE_ID = <%= disk.attr('STORAGE','href').split('/').last %>
         ]
    <%end%>

   <% end %>
<% end %>
```

EC2

Para que las imágenes creadas con ec2 las cree con qcow, editamos la plantilla correspondiente, por ejemplo /etc/one/ec2query_templates/m1.small.erb, y buscamos la siguiente linea:

```
DISK = [ IMAGE_ID   = <%= erb_vm_info[:img_id] %> ]
```

y la substituimos por:

```
DISK = [ IMAGE_ID   = <%= erb_vm_info[:img_id] %>, driver=qcow2 ]
```

Otros

Init.d

Para que al arrancar la maquina virtual de One, arranque también el OCCI, EC2 y Sunstone, editamos el fichero /etc/init.d/opennebula y buscamos la función do_start, y le añadimos al final:

```
   su oneadmin -s /bin/sh -c 'occi-server start'
   su oneadmin -s /bin/sh -c 'sunstone-server start'
   su oneadmin -s /bin/sh -c 'econe-server start'
   su oneadmin -s /bin/sh -c 'oneacctd start'
```

Con lo que toda la función entera queda de la siguiente forma:

```
do_start()
{
   mkdir -p /var/run/one /var/lock/one
   chown oneadmin /var/run/one /var/lock/one
   ONE_AUTH_FILE=/var/lib/one/auth
   if [ ! -f $ONE_AUTH_FILE ]; then
      PASSWORD=$(cat /dev/urandom|tr -dc 'a-zA-Z0-9'|fold -w 10|head -n1)
```

```
        su oneadmin -s /bin/sh -c "echo oneadmin:$PASSWORD > $ONE_AUTH_FILE"
    fi
    ONE_AUTH=$ONE_AUTH_FILE su oneadmin -s /bin/sh -c 'one start'
    su oneadmin -s /bin/sh -c 'occi-server start'
    su oneadmin -s /bin/sh -c 'sunstone-server start'
    su oneadmin -s /bin/sh -c 'econe-server start'
    su oneadmin -s /bin/sh -c 'oneacctd start'
}
```

Buscamos también la función do_stop y le añadimos al principio:

```
    su oneadmin -s /bin/sh -c 'occi-server stop'
    su oneadmin -s /bin/sh -c 'sunstone-server stop'
    su oneadmin -s /bin/sh -c 'econe-server stop'
    su oneadmin -s /bin/sh -c 'oneacctd stop'
```

Con lo que la función entera queda de la siguiente forma:

```
do_stop()
{
    su oneadmin -s /bin/sh -c 'occi-server stop'
    su oneadmin -s /bin/sh -c 'sunstone-server stop'
    su oneadmin -s /bin/sh -c 'econe-server stop'
    su oneadmin -s /bin/sh -c 'oneacctd stop'
    su oneadmin -s /bin/sh -c 'one stop'
}
```