OpenStack IceHouse 安装配置手册

Edition: 1.0

前提：本文档中涉及到的所有主机的 OS 均为 RHEL 6.5 x86_64 系统。

Control Node: controller.magedu.com，RAM：2G
　　管理接口：192.168.10.7
Compute Node:　compute1.magedu.com，RAM：2G
　　管理接口：192.168.10.8
　　隧道接口：10.0.10.8
Network node: network.magedu.com，RAM：512M
　　管理接口：192.168.10.9
　　隧道接口：10.0.10.9
　　外部接口：172.16.100.9
Block Storage node: stor1.magedu.com, RAM: 512M
　　管理接口：192.168.10.100

OpenStack 的版本为 icehouse，官方文档：http://docs.openstack.org/icehouse/install-guide/install/yum/content/。

前提：
（1）各节点禁用 NetworkManager 服务，启用 network 服务；
（2）各节点清空防火墙规则，并保存；
（3）Openstack 的 Networking（Neutron）需要一个控制节点、一个网络节点和至少一个计算节点。其中，控制节点仅需一个网络接口用于接入管理网络；而网络节点需三个接口，一个接入管理网络，一个用于 instance 隧道网络，一个用于外部网络；每个计算节点通常需要两个网络接口，一个用于管理网络，一个用于 instance 隧道网络。
（4）各节点之间如果期望以主机名的方式互相访问，需要编辑/etc/hosts 文件或定义 DNS 服务完成名称解析；
（5）Network node 的用于外部网络的接口不能用 IP 地址，建议使用类似如下配置，其中 INTERFACE_NAME 为实际的网络接口名，例如 eth2：

```
DEVICE=INTERFACE_NAME
TYPE=Ethernet
ONBOOT="yes"
BOOTPROTO="none"
```

（6） 各节点之间时间上要同步，可以通过使用 ntp 服务器实现；
（7） 准备好 mysql 服务器，并使其服务进程拥有如下配置信息：

```
[mysqld]
...
default-storage-engine = innodb
innodb_file_per_table = 1
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8
```

注意：文中配置中用到的 auth_uri 参数已经过期，必要时可以替换为 identity_uri。

一、安装配置 Keystone

OpenStack 的 Identify 服务(即 Keystone)有两个主要功能

用户管理：实现用户认证及授权；

服务目录(Service catalog)：所有可用服务的信息库，包含所有可用服务及其 API endpoint 路径；

1、安装 Keystone

1.1、安装并初始化 MySQL 服务器

```
# yum -y install mariadb-galera-server
```

编辑配置文件/etc/my.cnf，在[mysqld]段中添加如下内容

```
default-storage-engine = innodb
innodb_file_per_table = ON
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8
skip_name_resolve = ON
```

而后启动服务。

```
# service mysqld start
```

使用 mysql_secure_installation 命令进行安全实始化

```
# mysql_secure_installation
```

设置服务开机自动启动。

```
# chkconfig mysqld on
```

1.2、安装配置 Identity 服务

```
# yum install openstack-utils openstack-keystone python-keystoneclient
```

创建 keystone 数据库，其默认会创建一个 keystone 用户以访问此同名数据库，密码可以使用--pass 选项

指定。
openstack-db --init --service keystone --pass keystone
Please enter the password for the 'root' MySQL user:
Verified connectivity to MySQL.
Creating 'keystone' database.
Asking openstack-keystone to sync the database.
Complete!

如果本机尚未安装 mysql-server，此脚本会自动安装之，并且会提醒用户为 root 用户设定密码。

如果不想使用默认的 keystone 用户访问其数据，也可以使用下面的命令为 keystone 服务创建访问数据库的用户。这里使用 keystone 用户名及 keystone 密码。
mysql -uroot -p
mysql> GRANT ALL ON keystone.* TO 'keystone'@'%' IDENTIFIED BY 'keystone';
mysql> FLUSH PRIVILEGES;

1.3、编辑 keystone 的主配置文件，使得其使用 MySQL 做为数据存储池，并配置其使用正确的参数

openstack-config --set /etc/keystone/keystone.conf \
    database connection mysql://keystone:keystone@controller/keystone

vim /etc/keystone/keystone.conf
在[sql]段中，确保与 mysql 相关的内容类似如下，注意其中的密码为 keystone 用户访问 MySQL 服务器所使用的密码：
connection = mysql://keystone:keystone@controller/keystone

上面的参数指定连接时，其语法格式为“mysql://[user]:[pass]@[primary IP]/[db name]”。

而后初始化 keystone 数据库。
su -s /bin/sh -c "keystone-manage db_sync" keystone

1.4、配置 keystone 的管理 token

为了使用 admin 用户管理 keystone，可以通过配置 keystone 的客户端使用 SERVICE_TOKEN 和 SERVICE_ENDPOINT 环境变量来连接至 Keystone。

export ADMIN_TOKEN=$(openssl rand -hex 10)
export OS_SERVICE_TOKEN=$ADMIN_TOKEN
export OS_SERVICE_ENDPOINT=http://controller:35357/v2.0
echo $ADMIN_TOKEN > ~/ks_admin_token

openstack-config --set /etc/keystone/keystone.conf DEFAULT admin_token 'YOUR_ADMIN_TOKEN'
其中的‘YOUR_ADMIN_TOKEN’为一个字符串，建议使用一串随机数据，这可以使用 openssl 命令

生成。比如，上面的使用可以使用如下形式：
openstack-config --set /etc/keystone/keystone.conf DEFAULT admin_token $ADMIN_TOKEN

设定 openstack 用到的证书服务。
keystone-manage pki_setup --keystone-user keystone --keystone-group keystone
chown -R keystone:keystone /etc/keystone/ssl
chmod -R o-rwx /etc/keystone/ssl

接着启动 keystone 服务：
service openstack-keystone start
chkconfig openstack-keystone on

查看进程启动的相关信息：
ps auxf | grep -i keystone-all
root 27082 0.0 0.1 103252 844 pts/0 S+ 17:40 0:00 _ grep -i keystone-all
keystone 27063 1.0 6.4 293704 48772 ? S 17:38 0:00 /usr/bin/python /usr/bin/keystone-all

查看监听的端口：
ss -tnlp | grep keystone-all
LISTEN 0 128 *:35357 *:*
users:(("keystone-all",27063,4))
LISTEN 0 128 *:5000 *:*
users:(("keystone-all",27063,6))

检查日志文件中有无错误提示：
grep ERROR /var/log/keystone/keystone.log

1.6、设定 Keystone 为 API endpoint

在 Openstack 中，服务(service)指的是计算(nova)、对象存储(Swift)或映像(image)等，而 Horizon(web dashboard)依赖于 Keystone registry 中的 API endpoint(某网络资源或服务的访问路径，通常表现为 URL)来访问这些服务，包括 Keystone 自身。因此，这里需要将 Keystone 服务自身及访问路径(API endpoint)先加入到 Keystone 的 registry 中。

keystone service-create --name=keystone --type=identity --description="Keystone Identity Service"
+-------------+----------------------------------+
| Property | Value |
+-------------+----------------------------------+
description	Keystone Identity Service
enabled	True
id	01fa0fe3213d4410a60765f4d35cb5aa
name	keystone
type	identity

```
+-------------+----------------------------------+
```

为上面新建的 service 添加 endpoint。注意，其中的 service_id 的内容为上面 service-create 命令创建的 service 的 id。

```
# keystone endpoint-create \
    --service-id=$(keystone service-list | awk '/ identity / {print $2}') \
    --publicurl 'http://controller:5000/v2.0' \
    --adminurl 'http://controller:35357/v2.0' \
    --internalurl 'http://controller:5000/v2.0'
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
|   adminurl  |   http://controller:35357/v2.0   |
|      id     | d3eb1cbfdc5348809c9d557a5f78ef90 |
| internalurl |   http://controller:5000/v2.0    |
|  publicurl  |   http://controller:5000/v2.0    |
|    region   |            regionOne             |
|  service_id | 01fa0fe3213d4410a60765f4d35cb5aa |
+-------------+----------------------------------+
```

2 创建 user、role 及 tenant

tenant 是 OpenStack 的 Keystone 中的一个重要的术语，它相当于一个特定项目(project)或一个特定的组织(origaniztion)等，它实现了资源或 identity 对象的隔离。用户(user)在认证通过后即能访问资源，其通常会被直接关联至某 tenant，因此看起来就像用户是包含于 tenant 中的。角色(role)是权限的窗口，其可用于快速为一组用户完成相同的授权操作。

keystone 可通过两种方式完成用户认证，一种为 token 认证，一种为 credential(如用户名和密码等信息)认证。前面的配置中，为 Keystone 的管理用户 admin 提供了 token 的认证方式，并以之完成了如前所述的服务创建等工作。事实上，Keystone 的重要功能之一便是提供用户认证，但为了便于认证信息的管理，其通常基于 credential 的方式进行。风格统一起见，这里让 admin 用户基于后一种认证方式完成认证并执行管理工作，事实上，这也是 Keystone 服务配置的基本要求。

2.1 keystone 子命令

keystone 有许多子命令，分别用于实现 keystone 的各种对象管理，如用户创建、删除等。获取其帮助信息及子命令列表，可以使用-h 选项，获取某子命令的使用帮助，则可以使用如下格式：

```
# keystone help SUB_COMMAND
```

例如，要获取创建租客的使用 tenant-create 的使用帮助，可以使用如下命令：

```
# keystone help tenant-create
usage: keystone tenant-create --name <tenant-name>
                              [--description <tenant-description>]
```

[--enabled <true|false>]

Create new tenant.

Arguments:
  --name <tenant-name> New tenant name (must be unique).
  --description <tenant-description>
                        Description of new tenant. Default is none.
  --enabled <true|false>
                        Initial tenant enabled status. Default is true.

2.2 admin tenant

2.2.1 创建用于 keystone 管理的 tenant、角色和用户

创建管理 tenant，其名称为 admin，描述信息为“Admin Tenant”。

```
# keystone tenant-create --name admin --description "Admin Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |           Admin Tenant           |
|   enabled   |              True                |
|      id     | 3b655869dfcd409a9e7649769d5dfa57 |
|     name    |              admin               |
+-------------+----------------------------------+
```

查看创建的 admin 这个 tenant 的相关信息（可选步骤）。

```
# keystone tenant-list
+----------------------------------+-------+---------+
|                id                |  name | enabled |
+----------------------------------+-------+---------+
| 3b655869dfcd409a9e7649769d5dfa57 | admin |   True  |
+----------------------------------+-------+---------+
```

创建 admin 角色。

```
# keystone role-create --name admin
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|    id    | 9e09bd2e181e4556af2a8c6706c879ff |
|   name   |              admin               |
+----------+----------------------------------+
```

查看角色的相关信息（可选步骤）。

```
# keystone role-list
+----------------------------------+----------+
|                id                |   name   |
+----------------------------------+----------+
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_ |
| 9e09bd2e181e4556af2a8c6706c879ff |  admin   |
+----------------------------------+----------+
```

创建 admin 用户，并将其直接关联至前面创建的 Admin Tenant。事实上，这里的关联操作也可以不执行，而在为 admin 赋予角色时进行，如后面所示。

```
# keystone user-create --name admin --pass admin --tenant admin --email admin@magedu.com
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |        admin@magedu.com          |
| enabled  |              True                |
|    id    | 07002d00863742c7b02e2856587198a7 |
|   name   |              admin               |
| tenantId | 3b655869dfcd409a9e7649769d5dfa57 |
| username |              admin               |
+----------+----------------------------------+
```

查看创建的 admin 用户的相关信息。

```
# keystone user-list
+----------------------------------+-------+---------+-----------------+
|                id                |  name | enabled |      email      |
+----------------------------------+-------+---------+-----------------+
| 07002d00863742c7b02e2856587198a7 | admin |   True  | admin@magedu.com |
+----------------------------------+-------+---------+-----------------+
```

而后，将 admin 角色赋予 admin 用户，正常执行时，此命令没有信息输出。

```
# keystone user-role-add --user admin --role admin --tenant admin
```

user-role-list 命令可显示添加的相关结果。下面的显示可以看出，admin 用户已经成为 admin role 的一员

```
# keystone user-role-list --user admin --tenant admin
+----------------------------------+----------+----------------------------------+----------------------------------+
|                id                |   name   |             user_id              |            tenant_id             |
+----------------------------------+----------+----------------------------------+----------------------------------+
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_ | 07002d00863742c7b02e2856587198a7 | 3b655869dfcd409a9e7649769d5dfa57 |
| 9e09bd2e181e4556af2a8c6706c879ff |  admin   | 07002d00863742c7b02e2856587198a7 | 3b655869dfcd409a9e7649769d5dfa57 |
+----------------------------------+----------+----------------------------------+----------------------------------+
```

2.2.2 为 admin 用户启用基于 credential 的认证机制

定义如下环境变量，启用基于 credential 的用户认证。为了使用方便，下面的环境变量导出命令可以保存至一个配置文件中，如~/.keystonerc_admin。

```
# export OS_USERNAME=admin
# export OS_TENANT_NAME=admin
# export OS_PASSWORD=admin
# export OS_AUTH_URL=http://controller:35357/v2.0/
# export PS1='[\u@\h \W(keystone_admin)]\$ '
```

而后注销基于 token 认证时创建的环境变量从而禁止 admin 用户使用 token 认证。

```
# unset OS_SERVICE_TOKEN
# unset OS_SERVICE_ENDPOINT
```

验正新的认证机制是否已经生效。

```
[root@node1 ~(keystone_admin)]# keystone user-list
```

注意：icehouse 版本上，keystone 的默认验证方式是 PKI，这可能需要签名证书方能验正通过；如果没有如前面的步骤中实始化 PIK 机制，会遇到类似如下错误信息。此时需要执行与 PKI 设定相关的步骤。如果不想采用此用方式，使用下面描述的方案解决相关的问题亦可。

```
Authorization Failed: An unexpected error prevented the server from fulfilling your request. (HTTP 500)
```

Openstack 此前较老些的版本用的是 UUID 验正格式，因此，修改 keystone.conf 中 token_format 参数的值为 UUID，并重启 openstack-keystone 即可。

```
# keystone user-list
+----------------------------------+-------+---------+-----------------+
|                id                |  name | enabled |      email      |
+----------------------------------+-------+---------+-----------------+
| dac58f84fe5c44aaa38c0a500e04f6e3 | admin |   True  | admin@magedu.com |
+----------------------------------+-------+---------+-----------------+
```

2.3 创建服务 tenant

上面创建的 admin 用户用于 Keystone 的管理工作，而 OpenStack 的各服务(如 swift 等)通常需要定义在一个 service tenant 中，而且各 service 也都需要一个具有管理权限的用户，后续各服务的安装配置过程中会陆续创建此些用户。

```
# keystone tenant-create --name service --description "Service Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |          Service Tenant          |
```

```
|   enabled   |                True                |
|     id      | 15781d38b2bb4b088922a20d0c0311b5 |
|    name     |              service               |
+-------------+----------------------------------+
```

2.4 创建一个测试使用的 tenant 及相关的用户

提示：下面的步骤可以后面需要时再执行。

创建一个正常的用户及一个 tenant：

创建 demo 用户：

```
# keystone user-create --name=demo --pass=demo --email=demo@magedu.com
```

创建 demo tenant：

```
# keystone tenant-create --name=demo --description="Demo Tenant"
```

链接 demo 用户至_member_角色及 demo tenant：

```
# keystone user-role-add --user=demo --role=_member_ --tenant=demo
```

二、Openstack Image Service

Openstack Image 服务可用于发现、注册及检索虚拟机映像(image)，它提供了一个 RESTful 的 API，能够让用户查询 VM 映像的元数据及通过 HTTP 请求获取映像，并可以让 python 程序员通过客户端类在 python 代码中完成类似的所有任务。VM 映像文件可以存储在各类存储中，如普通的文件系统、对象存储系统(如 Openstack Object Storage)、S3 存储及 HTTP(仅作为获取映像文件之用，而不能写于此中创建映像文件)等。

Image Service 由以下个组件构成：

glance-api：glance 的 API 服务接口，负责接收对 Image Service API 中映像文件的查看、下载及存储请求；

glance-registry：存储、处理及获取映像文件的元数据，例如映像文件的大小及类型等；

database：存储映像文件元数据；

映像文件存储仓库：Image Service 支持多种类型的映像文件存储机制，包括使用普通的文件系统、对象存储、RADOS 块设备、HTTP 以及 Amazon 的 S3 等；

2.1 安装配置 Glance 服务

2.1.1 安装相关软件包

yum install openstack-glance python-glanceclient

初始化 glance 数据库，同时创建其服务同名的用户，并为其指定密码，这里选择使用与服务名同名的密码 glance。
openstack-db --init --service glance --password glance

如果不想使用默认的 glance 用户访问其数据，也可以使用下面的命令为 glance 创建数据库及相关的用户，注意要换需替换里面的 GLANCE_USER 和 GLANCE_PASS。
mysql -uroot -p
mysql> CREATE DATABASE glance;
mysql> GRANT ALL ON glance.* TO 'GLANCE_USER'@'%' IDENTIFIED BY 'GLANCE_PASS';
mysql> FLUSH PRIVILEGES;

生成 glance 依赖的数据库表。
su -s /bin/sh -c "glance-manage db_sync" glance

2.1.2 配置 glance-api 和 glance-registry 接入数据库

配置 Image Service 服务连入数据库的 url。
```
# openstack-config --set /etc/glance/glance-api.conf database \
   connection mysql://glance:glance@controller/glance
# openstack-config --set /etc/glance/glance-registry.conf database \
   connection mysql://glance:glance@controller/glance
```

2.1.3 创建 glance 管理用户

下面是一个示例：为服务 Tenant 创建用户 glance，此用户账号可用于后文中的 Glance 服务，密码同用户名。
```
# keystone user-create --tenant=admin --name=glance --pass=glance --email=glance@magedu.com
+----------+----------------------------------+
| Property |                Value             |
+----------+----------------------------------+
|   email  |          glance@magedu.com       |
| enabled  |                True              |
|    id    | ad425f2b0cb9463e86b73be2d48a6fe9 |
|   name   |               glance             |
| tenantId | 3b655869dfcd409a9e7649769d5dfa57 |
| username |               glance             |
+----------+----------------------------------+
```

而后，把 admin 角色赋予 nova 用户，正常执行时，此命令没有信息输出。

keystone user-role-add --user=glance --role=admin --tenant=service

查看添加的结果：

```
# keystone user-role-list --tenant service --user glance
+----------------------------------+-------+----------------------------------+----------------------------------+
|                id                |  name |             user_id              |            tenant_id             |
+----------------------------------+-------+----------------------------------+----------------------------------+
| 9e09bd2e181e4556af2a8c6706c879ff | admin | ad425f2b0cb9463e86b73be2d48a6fe9 | 15781d38b2bb4b088922a20d0c0311b5 |
+----------------------------------+-------+----------------------------------+----------------------------------+
```

2.1.4 配置 Glance 服务使用 Identity 服务认证

执行如下命令配置 glance-api:

```
openstack-config --set /etc/glance/glance-api.conf keystone_authtoken auth_uri http://controller:5000
openstack-config --set /etc/glance/glance-api.conf keystone_authtoken auth_host controller
openstack-config --set /etc/glance/glance-api.conf keystone_authtoken auth_port 35357
openstack-config --set /etc/glance/glance-api.conf keystone_authtoken auth_protocol http
openstack-config --set /etc/glance/glance-api.conf keystone_authtoken admin_tenant_name service
openstack-config --set /etc/glance/glance-api.conf keystone_authtoken admin_user glance
openstack-config --set /etc/glance/glance-api.conf keystone_authtoken admin_password glance
openstack-config --set /etc/glance/glance-api.conf paste_deploy flavor keystone
```

提示：glance 默认使用“File”做为其存储类型（default_store 参数），其也可以使用 swift 等也做为存储类型，但此时尚未配置 swift 服务，因此，此处先使用默认值即可。

执行如下命令配置 glance-registry：

```
openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken \
  auth_uri http://controller:5000
openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken \
  auth_host controller
openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken auth_port 35357
openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken auth_protocol http
openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken admin_tenant_name service
openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken admin_user glance
openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken admin_password glance
openstack-config --set /etc/glance/glance-registry.conf paste_deploy flavor keystone
```

2.1.5 在 keystone 注册 glance 服务

```
# keystone service-create --name=glance --type=image --description="Glance Image Service"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |        Glance Image Service      |
|   enabled   |               True               |
|      id     | 872902167e7b4d918f633b3fdc0c6b21 |
|     name    |              glance              |
|     type    |              image               |
+-------------+----------------------------------+

# keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ image / {print $2}') \
  --publicurl=http://controller:9292 \
  --internalurl=http://controller:9292 \
  --adminurl=http://controller:9292
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
|   adminurl  |      http://controller:9292      |
|      id     | ef739d1b5d5e436381ca967de181d1bf |
| internalurl |      http://controller:9292      |
|  publicurl  |      http://controller:9292      |
|    region   |             regionOne            |
|  service_id | 872902167e7b4d918f633b3fdc0c6b21 |
+-------------+----------------------------------+
```

2.1.6 启动服务

启动 glance-api 服务：

```
# service openstack-glance-api start
# chkconfig openstack-glance-api on
```

启动 glance-registry 服务：

```
# service openstack-glance-registry start
# chkconfig openstack-glance-registry on
```

2.1.7 测试

默认情况下，glance 中没有任何映像文件，因此下面的命令将没有任何返回值。如果其不能返回为空，原因可能是配置有问题，请自行检查前面的配置过程。

```
# glance image-list
```

2.2 Image 服务管理

2.2.1 glance image-create 命令

glance image-create 命令用于添加新的虚拟机映像至 glance 中，glance image-update 命令用于修改已经完成更新的映像的属性信息。

image-create 命令至少要接受三个参数：--name、--container_format 及--disk_format。其中--disk_format 用于指明磁盘映像文件的格式，包括 raw、qcow2、vhd、vmdk、iso、vdi、aki(amazon kernel image)、ari(amazon ramdisk image)及 ami(amazon machine image)等。--container_format 用于标明映像文件是否包含了虚拟机的元数据信息，然而，目前 Compute 服务并未使用此信息，因此，在不确定的情况可以将其指定为 bare，而合用的格式主要有 bare(没有 container 或元数据信息)、ovf、aki、ari 或 ami 几种。

2.2.2 映像元数据

glance image-create 或 glance image-update 命令的--property key=value 参数能够为映像文件附加元数据信息。而常用的属性主要有：

architecture：hypervisor 必须支持的 CPU 架构，如 x86_64、arm 等；
hypervisor_type：hypervisor 类型，其能够使用的值包括 xen、qemu、kvm、lxc、uml、vmware、hyperv 及 powervm；
vm_mode：定义虚拟机模式，即应用于虚拟机的 host/guest ABI；例如 hvm、xen、uml、exe 等；

xenAPI 还有其专用的两个属性可以定义：
auto_disk_config：布尔型属性值，true 表示在虚拟机实例启动前磁盘的根分区将被自动调整；
os_type：image 中安装的操作系统类型，如 linux、windows 等，XenAPI 能够根据此属性值来执行不同的操作，如为 windows 创建 fat32 格式的交换分区、限制其主机名称少于 16 个字符等；

VMware API 也有如下三个专用属性可以定义：
vmware_adaptertype：定义 hypervisor 使用的虚拟 SCSI 或 IDE 接口类型，其有效值为 lsiLogic、busLogic 及 ide；
vmware_ostype：VMware GuestID，用于描述映像文件中的操作系统类型；vmware 所支持的类型较多，可以参照 thinkvirt.com 获取详细信息；默认值为 otherGuest；
vmware_image_version：目前未使用，默认值为 1；

2.2.3 获取虚拟机映像

2.2.3.1 CirrOS(test)映像

由 Scott Moser 维护的一系列用于测试目的的微型虚拟机映像，登录名为 cirros，下载地址为 https://launchpad.net/cirros/+download。在 QEMU 或 KVM 中测试时，官方建议使用 QCOW2 格式的映像文件。

2.2.3.2 Ubuntu 映像

Canonical 官方提供了基于 Ubuntu 的系列映像，登录名为 ubuntu，下载地址为 http://uec-images.ubuntu.com/。在 QEMU 或 KVM 中部署时，建议使用 QCOW2 格式的映像文件。

2.2.3.3 Fedora 映像

Fedora 官方提供了预制的 Fedora JEOS 映像，下载地址为 http://berrange.fedorapeople.org/images，目前最新的是为 x86_64 平台提供为的 QCOW2 格式的映像 f17-x86_64-openstack-sda.qcow2。

2.2.3.4 OpenSUSE 和 SLES11 映像

通过 SUSE Studio(http://susestudio.com/)可以很方便地为 OpenSUSE 和 SLES11 构建与 OpenStack 兼容的虚拟应用，比如创建一个 OpenSUSE12.1 的 JEOS 映像。

2.2.3.5 Rackspace 云生成器映像

Rackspace 云生成器(https://github.com/rackerjoe/oz-image-build)提供了多种发行版预制的预制映像，如 RedHat、CentOS、Fedora 及 Ubuntu 等。

2.2.4 制作映像的专用工具

Oz(KVM)：Oz 是能够为常见 Linux 发行版创建映像文件的命令行工具，Rackspace 就是使用 Oz 创建的映像文件。在 Fedora Project 的 wiki 中，提供了使用 Oz 创建映像文件的案例，具体请参照 https://fedoraproject.org/wiki/Getting_started_with_OpenStack_Nova#Building_an_Image_With_Oz。

VMBuilder(KVM,Xen)：VMBuilder 能够为不同的 hypervisor 创建虚拟机映像文件，它是一个脚本，能够自动收集所需的资源为虚拟机创建映像文件。Ubuntu 为之提供了一个使用案例，具体请参照 https://help.ubuntu.com/12.04/serverguide/jeos-and-vmbuilder.html。

VeeWee(KVM)：VeeWee 通常用于创建 Vagrant 虚拟工作环境，一种基于 Virtualbox、VMware、AWS 等虚拟化技术的虚拟化工具。VeeWee 也可以用于创建 KVM 映像。

imagefactory：Aeolus 项目的一款工具，用于自动化创建、转换及为不同的云服务商上海映像文件，支持 Openstack 云。

2.2.5 为 Openstack 定制映像

映像文件要实现与 Openstack 兼容，需要顾及多方面的因素。

2.2.5.1 支持元数据服务或配置驱动(config drive)

Openstack 支持的映像文件必须能够由 Openstack 获取到其元数据信息，如 ssh 公钥以及用户在请求映像文件时提交的用户数据等。这些元数据信息可以通过元数据服务或配置驱动获取，最简单的方式莫过于在映像中安装 cloud-init 程序。cloud-init 用于为云实例提供配置及定制功能，项目的地址为

https://launchpad.net/cloud-init。

2.2.5.2 支持对磁盘映像大小进行调整

虚拟机映像文件的磁盘大小由创建映像时的初始大小决定，然而 Openstack 需要在启动实例时通过指定不同的 flavor 来使用不同大小的磁盘空间。例如，有着磁盘初始大小为 5G 的映像文件，在用于创建实例时使用了 m1.small 这个 flavor，虚拟机实例将需要一个大小为 10G 的主盘。调整实例的磁盘大小时，通过在其尾部填 0 来完成。

映像文件的分区大小也需要能够根据用户的需要在实例启动时进行调整，否则，在实例启动后，为了能够访问由 flavor 的配置指定的超出磁盘原始大小的其它空间就不得不手动去调整分区大小。因此，在实例启动时，映像文件需要运行一个脚本以修改分区表，并运行相应的程序(如 resize2fs 等)调整文件系统，使得其能够适应新的分区大小。

2.3 创建映像文件

为了使用方便，这里采用 CirrOS 项目制作的映像文件，其也经常被拿来测试 Openstack 的部署。其地址为 https://launchpad.net/cirros，可以按需下载所想尝试使用的版本，下载地址为 http://download.cirros-cloud.net/。这里以 0.3.3 版为例进行演示，请注意您的文件名及路径。

2.3.1 准备映像文件

```
# mkdir /stackimages
# cd /stackimages
# wget http://download.cirros-cloud.net/0.3.3/cirros-0.3.3-i386-disk.img
# wget http://download.cirros-cloud.net/0.3.3/cirros-0.3.3-x86_64-disk.img
```

使用 qemu-img 命令分别查看两个映像文件的格式信息。

提示：如果系统上没有 qemu-img 命令可用，需要事先安装 qemu-img 程序包。

```
qemu-img info cirros-0.3.3-i386-disk.img
image: cirros-0.3.3-i386-disk.img
file format: qcow2
virtual size: 39M (41126400 bytes)
disk size: 12M
cluster_size: 65536

# qemu-img info cirros-0.3.3-x86_64-disk.img
image: cirros-0.3.3-x86_64-disk.img
file format: qcow2
virtual size: 39M (41126400 bytes)
disk size: 13M
cluster_size: 65536
```

接下上传 cirros 两个映像文件

向 Image Service 请求存储映像文件的命令格式为：
glance image-create --name=IMAGELABEL --disk-format=FILEFORMAT \
  --container-format=CONTAINERFORMAT --is-public=ACCESSVALUE < IMAGEFILE

例如，下面导入 cirros-0.3.3-i386-disk.img 和 cirros-0.3.3-x86_64-disk.img 两个映像文件。

glance image-create --name=cirros-0.3.3-i386 --container-format=bare --disk-format=qcow2 --is-public=true < cirros-0.3.3-i386-disk.img
+------------------+--------------------------------------+
| Property | Value |
+------------------+--------------------------------------+
checksum	283c77db6fc79b2d47c585ec241e7edc
container_format	bare
created_at	2014-10-15T12:25:50
deleted	False
deleted_at	None
disk_format	qcow2
id	86f16760-0616-4c2e-899f-a1182260705a
is_public	True
min_disk	0
min_ram	0
name	cirros-0.3.3-i386
owner	3b655869dfcd409a9e7649769d5dfa57
protected	False
size	12268032
status	active
updated_at	2014-10-15T12:25:50
virtual_size	None
+------------------+--------------------------------------+

glance image-create --name=cirros-0.3.3-x86_64 --container-format=bare --disk-format=qcow2 --is-public=true < cirros-0.3.3-x86_64-disk.img
+------------------+--------------------------------------+
| Property | Value |
+------------------+--------------------------------------+
checksum	133eae9fb1c98f45894a4e60d8736619
container_format	bare
created_at	2014-10-15T12:26:21
deleted	False
deleted_at	None
disk_format	qcow2

id	b6c92772-f3d0-480d-ba90-0ad8c3669ea8
is_public	True
min_disk	0
min_ram	0
name	cirros-0.3.3-x86_64
owner	3b655869dfcd409a9e7649769d5dfa57
protected	False
size	13200896
status	active
updated_at	2014-10-15T12:26:21
virtual_size	None
+------------------+--------------------------------------+

上述命令示例中，container-format 选项用于指定映像容器的格式，其可接受的值有 bare、ovf、ami、ari 和 aki 等 5 个，其中后三个为 amazon 的专有格式。

列出上传的映像文件。

```
# glance image-list
+--------------------------------------+---------------------+-------------+------------------+----------+--------+
| ID                                   | Name                | Disk Format | Container Format | Size     | Status |
+--------------------------------------+---------------------+-------------+------------------+----------+--------+
| 86f16760-0616-4c2e-899f-a1182260705a | cirros-0.3.3-i386   | qcow2       | bare             | 12268032 | active |
| b6c92772-f3d0-480d-ba90-0ad8c3669ea8 | cirros-0.3.3-x86_64 | qcow2       | bare             | 13200896 | active |
+--------------------------------------+---------------------+-------------+------------------+----------+--------+
```

还可以使用 image-show 命令查看指定的 Image 映像文件的信息。

```
# glance image-show cirros-0.3.3-i386
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 283c77db6fc79b2d47c585ec241e7edc     |
| container_format | bare                                 |
| created_at       | 2014-10-15T12:25:50                  |
| deleted          | False                                |
| disk_format      | qcow2                                |
| id               | 86f16760-0616-4c2e-899f-a1182260705a |
| is_public        | True                                 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | cirros-0.3.3-i386                    |
| owner            | 3b655869dfcd409a9e7649769d5dfa57     |
```

protected	False
size	12268032
status	active
updated_at	2014-10-15T12:25:50
+------------------+--------------------------------------+

此外，可以使用 member-add 授权指定的映像文件为一个 tenant 所能够访问。member-delete 则能够从一个 tenant 中移除对指定 image 的访问权限。具体的用法请参见 glance 命令的使用帮助。

三、compute 服务

3.1 compute 概述

compute 服务由多个组件共同组成，这些组件用于提供 API、Compute 核心组件、网络功能、控制台接口、命令行客户端及其它组件。

3.1.1 API

nova-api 服务：接受并响应终端用户对 compute API 接口调用的服务。除了支持 Openstack compute API，还兼容 amazon ec2 API，以及专用于特权用户执行管理工作的 admin API。另外，nova-api 还用于管理 instance，以及强制生效指定的策略等。

nova-api-metadata 服务：接收并响应由 instance 发起的 metadata 调用请求，仅用于 nova-network 的 multi-host 模型中。

3.1.2 compute core

nova-compute 进程：借助于 hypervisor API 起动及终止虚拟机实例的后台工作进程；工作过程包括从队列中接受操作请求并执行一系列的操作，例如启动一个 vm instance 以及更新其在数据库中的状态信息等。

nova-scheduler 进程：compute 中最简单的组件，负责从队列中取出一个 instance 的请求并决定由哪个 compute server 来运行之。

nova-conductor 模块：nova-compute 与数据库的中间层，以避免 nova-compute 与数据直接进行交互。

3.1.3 network

nova-network 守护进程：类似 nova-compute，它负责从队列取出网络相关的任务并执行相关的操作，包括设定桥接接口及修改 iptables 规则等。不过，此功能已经由 openstack 中专用的组件 Neutron 来实现。

nova-dhcpbridge 脚本：通过 dnsmasq 的 dhcp-script 来跟踪 ip 地址租约并记录于数据库中，止前已经由专门的服务组件 Neutron 实现。

3.1.4 console interface

nova-consoleauth 守护进程：认证由 console proxies 传入的用户认证 token。

nova-novncproxy 守护进程：向经由 vnc 接口连入各 instance 的请求提供代理服务，支持基于浏览器的 novnc 客户端。

nova-xvpnvncroxy 守护进程：向经由 vnc 接口连入各 instance 的请求提供代理服务，支持基于专为 openstack 设计的 java 客户端。

nova-cert 守护进程：管理 x509 格式的证书。

3.1.5 命令行客户端

nova：以 tenant 管理员或终端用户的身份通过命令行接口提交命令。

nova-manage：云管理员提交命令的接口。

3.1.6 其它组件

queue（AMQP）：在各进程间传递消息的消息队列服务，支持 RabbitMQ、Apache Qpid 以及 ZeroMQ。

SQL database：为云基础架构存储构建及运行时数据，包括可用的实例类型、运行中的实例、可用的网络等信息。理论上，Openstack 可支持任何 SQL 类型的数据库管理系统，但实践中较常用的为 SQLite、MySQL 或 PostgreSQL。

3.2 Compute 服务的安装与配置

compute 是一系列用于管理 vm instance 的服务的集合，这些服务可运行于同一节点，也可分别运行于不同的节点。中小型规模的部署实践中，其中多数服务部署于一个节点，即控制节点（control node），而负责实例运行的服务器则为一个或多个独立的专用节点，即计算节点(compute node)。

3.2.1 在控制节点安装启动 qpid

如前所述，Openstack 的各组件间通过消息队列传递消息以异步的方式进行协调，从而降低了各组件间的耦合度，使得高负载时的服务可用性大大提升。Openstack 支持使用的队列服务包括 RabbitMQ、Apache qpid 以及 zeroMQ。本文的示例以 qpid 为例。

yum install -y qpid-cpp-server

```
# sed -i -e 's/auth=.*/auth=no/g' /etc/qpidd.conf
# service qpidd start
# chkconfig qpidd on
```

3.2.2 安装配置 compute service 节点

```
# yum install openstack-nova-api openstack-nova-cert openstack-nova-conductor \
   openstack-nova-console openstack-nova-novncproxy openstack-nova-scheduler \
   python-novaclient
```

先在 keystone 中创建运行 nova 服务所需要的 nova 用户账号。

```
# keystone user-create --name=nova --pass=nova --email=nova@magedu.com
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |        nova@magedu.com           |
| enabled  |              True                |
|    id    | a069014ea0944763a588ef00baf6451f |
|   name   |              nova                |
| username |              nova                |
+----------+----------------------------------+
```

```
# keystone user-role-add --user=nova --tenant=service --role=admin
```

列出创建的用户的相关信息：

```
# keystone user-list --tenant=service
+----------------------------------+--------+---------+------------------+
|                id                |  name  | enabled |      email       |
+----------------------------------+--------+---------+------------------+
| ad425f2b0cb9463e86b73be2d48a6fe9 | glance |   True  | glance@magedu.com |
| a069014ea0944763a588ef00baf6451f |  nova  |   True  |  nova@magedu.com  |
+----------------------------------+--------+---------+------------------+
```

3.2.3 配置 nova 服务

配置过程可以直接编辑/etc/nova/nova.conf，也可以使用 openstack 的专用配置命令 openstack-config 来进行。这里通过 openstack-config 命令进行。

首先需要在数据库服务器上创建 nova 连接数据库服务器所用到的账号信息，而后为 nova 设定连接数据库服务器的 url。

初始化 nova 数据库，同时创建其服务同名的用户，并为其指定密码，这里选择使用与服务名同名的密码 nova。

```
# openstack-db --init --service nova --password nova
```

提示：如果不想使用默认的 nova 用户访问其数据，也可以使用下面的命令为 nova 创建数据库及相关的用户，请将下列命令中的 NOVA_USER 和 NOVA_PASS 替换成实际的用户名及密码。

```
# mysql -uroot -p
mysql> CREATE DATABASE nova;
mysql> GRANT ALL ON nova.* TO 'NOVA_USER'@'%' IDENTIFIED BY 'NOVA_PASS';
mysql> GRANT ALL ON nova.* TO 'NOVA_USER'@'localhost' IDENTIFIED BY 'NOVA_PASS';
mysql> FLUSH PRIVILEGES;
```

使用下面的命令生成 nova 服务用到的各表。

```
# su -s /bin/sh -c "nova-manage db sync" nova
```

配置 nova 连入数据库服务器的相关信息。

```
# openstack-config --set /etc/nova/nova.conf \
   database connection mysql://nova:nova@controller/nova
```

而后为 nova 指定连接队列服务 qpid 的相关信息。

```
# openstack-config --set /etc/nova/nova.conf DEFAULT rpc_backend qpid
# openstack-config --set /etc/nova/nova.conf DEFAULT qpid_hostname controller
```

接着将 my_ip、vncserver_listen 和 vncserver_proxyclient_address 参数的值设定为所属“管理网络”的接口地址。

```
# openstack-config --set /etc/nova/nova.conf DEFAULT my_ip 192.168.10.7
# openstack-config --set /etc/nova/nova.conf DEFAULT vncserver_listen 192.168.10.7
# openstack-config --set /etc/nova/nova.conf DEFAULT vncserver_proxyclient_address 192.168.10.7
```

运行如下各命令设定 nova 调用 keystone API 的相关配置。

```
 openstack-config --set /etc/nova/nova.conf DEFAULT auth_strategy keystone
 openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_uri http://controller:5000
 openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_host controller
 openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_protocol http
 openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_port 35357
 openstack-config --set /etc/nova/nova.conf keystone_authtoken admin_user nova
 openstack-config --set /etc/nova/nova.conf keystone_authtoken admin_tenant_name service
 openstack-config --set /etc/nova/nova.conf keystone_authtoken admin_password nova
```

设置虚拟网络接口插件的超时时长：

```
 openstack-config --set /etc/nova/nova.conf DEFAULT vif_plugging_timeout 0
 openstack-config --set /etc/nova/nova.conf DEFAULT vif_plugging_is_fatal False
```

3.2.4 在 KeyStone 中注册 Nova compute API

接下来需要将 Nova Compute API 在 KeyStone 中注册，dashboard 需要基于此与 Nova 交互。下面的部分

要以 keystone 的管理员 admin 身份执行，如果当前未为连接 KeyStone 设置环境变量，可以使用“source ~/.keystonerc_admin”进行。

```
# keystone service-create --name=nova --type=compute --description="OpenStack Compute"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |         OpenStack Compute        |
|   enabled   |               True               |
|      id     | 3cf89a7fe1e14fdb8f5db271e9cae54e |
|     name    |               nova               |
|     type    |              compute             |
+-------------+----------------------------------+

# keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ compute / {print $2}') \
  --publicurl=http://controller:8774/v2/%\(tenant_id\)s \
  --internalurl=http://controller:8774/v2/%\(tenant_id\)s \
  --adminurl=http://controller:8774/v2/%\(tenant_id\)s
+-------------+-----------------------------------------+
|   Property  |                  Value                  |
+-------------+-----------------------------------------+
|   adminurl  | http://controller:8774/v2/%(tenant_id)s |
|      id     |         1c5e582dd2f14332bd38f4bf66f8fc28        |
| internalurl | http://controller:8774/v2/%(tenant_id)s |
|  publicurl  | http://controller:8774/v2/%(tenant_id)s |
|    region   |                regionOne                |
|  service_id |         3cf89a7fe1e14fdb8f5db271e9cae54e        |
+-------------+-----------------------------------------+
```

3.2.5 启动 Nova service

```
# service openstack-nova-api start
# service openstack-nova-cert start
# service openstack-nova-consoleauth start
# service openstack-nova-scheduler start
# service openstack-nova-conductor start
# service openstack-nova-novncproxy start
# chkconfig openstack-nova-api on
# chkconfig openstack-nova-cert on
# chkconfig openstack-nova-consoleauth on
# chkconfig openstack-nova-scheduler on
# chkconfig openstack-nova-conductor on
# chkconfig openstack-nova-novncproxy on
```

提示：也可使用如下命令启动并设定各服务开机自动启动。

for svc in api cert consoleauth scheduler conductor novncproxy; do service openstack-nova-${svc} start; chkconfig openstack-nova-${svc} on; done

重启各服务：
for svc in api cert consoleauth scheduler conductor novncproxy; do service openstack-nova-${svc} restart; done

检验服务启动时是否有错误产生：
grep -i ERROR /var/log/nova/*

各服务启动后，检测 nova 服务是否已经正常工作。

```
# nova service-list
+------------------+------------------+----------+---------+-------+----------------------------+-----------------+
| Binary           | Host             | Zone     | Status  | State | Updated_at                 | Disabled Reason |
+------------------+------------------+----------+---------+-------+----------------------------+-----------------+
| nova-cert        | node1.magedu.com | internal | enabled | up    | 2014-10-15T12:40:23.000000 | -               |
| nova-consoleauth | node1.magedu.com | internal | enabled | up    | 2014-10-15T12:40:24.000000 | -               |
| nova-conductor   | node1.magedu.com | internal | enabled | up    | 2014-10-15T12:40:27.000000 | -               |
| nova-scheduler   | node1.magedu.com | internal | enabled | up    | 2014-10-15T12:40:25.000000 | -               |
+------------------+------------------+----------+---------+-------+----------------------------+-----------------+

# nova image-list
+--------------------------------------+---------------------+--------+--------+
| ID                                   | Name                | Status | Server |
+--------------------------------------+---------------------+--------+--------+
| 86f16760-0616-4c2e-899f-a1182260705a | cirros-0.3.3-i386   | ACTIVE |        |
| b6c92772-f3d0-480d-ba90-0ad8c3669ea8 | cirros-0.3.3-x86_64 | ACTIVE |        |
+--------------------------------------+---------------------+--------+--------+
```

3.3 compute 节点的安装与配置

计算节点负责接收来自控制节点的启动或关闭 vm instance 的指示并执行相应的操作。计算节点是用于运行一个或多个 vm instance 实例的主机，因此，其 CPU、内存及磁盘资源等都有着较高的要求，但实

际配置还需要根据项目的实际需求确定。实验性的配置中，完全可以把计算节点等所有服务运行于同一个节点。不过，本演示示例尽量以生产环境可用的方式进行，因此，这里将计算节点部署在一个独立的主机 172.16.100.8 上。实际使用中，可以使用类似下面的配置过程扩展使用多个计算节点。

3.3.1 安装 compute 节点需要程序包

```
# yum install openstack-nova-compute openstack-utils
```

3.3.2 配置 compute 节点

配置 nova 连接数据库服务器的相关信息。

```
# openstack-config --set /etc/nova/nova.conf database connection mysql://nova:nova@controller/nova
```

为 nova 指定连接队列服务 qpid 的相关信息。

```
# openstack-config --set /etc/nova/nova.conf DEFAULT rpc_backend qpid
# openstack-config --set /etc/nova/nova.conf DEFAULT qpid_hostname controller
```

为 nova 指定 glance-api 和 glance-registry 主机：

```
# openstack-config --set /etc/nova/nova.conf DEFAULT glance_host controller
```

将 my_ip、vncserver_listen 和 vncserver_proxyclient_address 参数的值设定为所属“管理网络”的接口地址。

```
 openstack-config --set /etc/nova/nova.conf DEFAULT my_ip 192.168.10.8
 openstack-config --set /etc/nova/nova.conf DEFAULT vnc_enabled True
 openstack-config --set /etc/nova/nova.conf DEFAULT vncserver_listen 0.0.0.0
 openstack-config --set /etc/nova/nova.conf DEFAULT vncserver_proxyclient_address 192.168.10.8
```

设置 novncproxy 的 base_url 为控制节点的地址。

```
# openstack-config --set /etc/nova/nova.conf \
   DEFAULT novncproxy_base_url http://controller:6080/vnc_auto.html
```

运行如下命令，设定 nova 调用 keystone API 的相关配置。

```
 openstack-config --set /etc/nova/nova.conf DEFAULT auth_strategy keystone
 openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_uri http://controller:5000
 openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_host controller
 openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_protocol http
 openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_port 35357
 openstack-config --set /etc/nova/nova.conf keystone_authtoken admin_user nova
 openstack-config --set /etc/nova/nova.conf keystone_authtoken admin_tenant_name service
 openstack-config --set /etc/nova/nova.conf keystone_authtoken admin_password nova
```

设置虚拟网络接口插件的超时时长：

```
 openstack-config --set /etc/nova/nova.conf DEFAULT vif_plugging_timeout 0
 openstack-config --set /etc/nova/nova.conf DEFAULT vif_plugging_is_fatal False
```

设置本机支持的 hypervisor。这里建议使用 kvm 虚拟化技术，但其要求计算节点的 CPU 支持硬件辅助的虚拟化技术。如果正在配置的测试节点不支持三件辅助的虚拟化，则需要将其指定为使用 qemu 类型的 hypervisor。测试计算节点是否支持硬件虚拟化技术的命令为“egrep -c '(vmx|svm)' /proc/cpuinfo”，如果命令返回值不为 0，则说明支持，否则则为不支持。这里使用的计算节点 172.16.100.8 的测试结果如下所示。

```
# egrep -c '(vmx|svm)' /proc/cpuinfo
4
```

上述测试结果表明，其为支持硬件辅助虚拟化，因此，这里使用如下命令设置 nova 使用的虚拟化技术。

```
# openstack-config --set /etc/nova/nova.conf libvirt virt_type kvm
```

3.3.3 启动 nova-compute 及相关的服务

```
# service libvirtd start
# service messagebus start
# service openstack-nova-compute start
# chkconfig libvirtd on
# chkconfig messagebus on
# chkconfig openstack-nova-compute on
```

提示：也可以使用如下命令启动相关的服务，并设置其开机自动启动。

```
# for svc in libvirtd messagebus openstack-nova-compute; do service $svc start; chkconfig $svc on; done
```

通过控制节点验正添加的 compute 节点是否已经能够使用：

```
# nova hypervisor-list
+----+---------------------+
| ID | Hypervisor hostname |
+----+---------------------+
| 1  | compute1.magedu.com |
+----+---------------------+
```

注：这里显示的是 172.16.100.8 的节点名称。

四、网络服务

Openstack 的 Networking service 代码名称为 neutron，它通过软件的方式提供了定义及接入网络的功能。Neutorn 为使用者提供了整合多个不同的网络技术来最大化发挥网络功能的机制，并提供了通过三层转发及 NAT 配置和管理网络功能的 API 以实现负载均衡、边缘防火墙及 IPsec VPN 等。

4.1 Networking 的基础概念

4.1.1 Networking API

Networking API 通过软件的方式定义了虚拟网络（network）、子网(subnet)及端口(port)等抽象层以描述网络资源。

Network：隔离的 2 层网络，类似于物理网络中的 VLAN；
Subnet：有着关联的配置状态的 ipv4 或 ipv6 的地址块；
Port：将主机连入单个网络设备的连接接口，类似于主机的一块物理网卡上的网线接口，可用于描述附加的网络配置，如 MAC 及 IP 地址等。

用 neutron 完全可以定义出功能丰富的网络拓扑，且其支持每个 tenant 在其内部创建多个 private network 并允许其配置使用任何可用的地址。多个 tenant 使用相同的私有地址配置也不会导致地址冲突。

4.1.2 Networking architecture

Neutron 在 Openstack 的模块化架构中是一个独立的组件，它独立于 Compute、Image、Identity 等服务，这意味着，部署 Neutron 可能需要在不同的节点上部署多个依赖的服务。Networing 服务器端依赖于 neutron-server 守护进程输出 Networking API 以及 Networking plug-in 的配置管理接口，由于是独立的，因此它可以被部署于一个专用的节点上，当然，也可以根据需要部署于所谓的“控制节点”上。Neutron 根据配置的需要，还包含多个插件：

plug-in agent：由 neutron-*-agent 程序包提供，运行于每个 hypervisor 节点上以实现本地的 vSwitch 配置功能。不同的插件依赖于不同的 agent，有些插件甚至于无须 agent。

dhcp agent：由 neutron-dhcp-agent 程序包提供，用于为 tenant 网络提供 DHCP 服务。

l3 agent：由 neutron-l3-agent 程序包提供，用于提供三层转发及 NAT 功能，帮助 tenant 网络中的 vm instance 访问外部网络。

metering agent：由 neutron-metering-agent 程序包提供，用于为 tenant 网络提供三层的流量计量。

4.1.3 Openstack 中物理机网络连接架构

一个标准的网络部署可能需要包含不止一个物理网络：

管理网络(management network)：为 Openstack 各组件间的通信提供的内部信道，此网络中的各地址应该仅能够在数据中心内部被访问到。

数据网络(data network)：为云环境内的 vm instance 提供数据访问信道。此网络中使用的 IP 地址取决于用到的网络插件。

外部网络(external network)：用于云环境中的 vm instance 需要与 Internet 通信时的信道。互联网上的地址可通过此网络访问到这个网络中的地址。

API 网络(API network)：将 Openstack 的 API（包括网络 API）提供给 tenant 的通信信道，此网络中的地址应该可以被 Internat 上的任何人访问到，因此，API 网络可以设计为使用与外部网络相同的物理网络。

4.1.4 Tenant and provider networks

简单来讲，供各 tenant 内部使用的网络即为 tenant netowrk；而为各 tenant 的网络提供通信承载的，不专属于某 tenant 的网络即称作 provider network。

4.1.5 tenant networks 和 provider networks

在 project 内部由 tenant 内部使用，且与其它 project 互相隔离的网络，即为 tenant network。Neutron 支持 4 种类型的 tenant network。

Flat：所有的 instance 位于同一个网络中，可被多个主机共享，不支持 VLAN 标签及其它网络隔离机制；

Local：各 instance 位于本地 compute 主机上，并且与 external networks 隔离；

VLAN：Networking 允许用户通过使用 VLAN IDs(802.1Q tagged)创建多个 provider 或 tenant 网络，用以映射至物理网络中真正的 VLAN。此机制使得 instance 可以跨环境实现彼此间的通信，以及与指定的 server、firewall、load balancer 以及其它位于同一个 2 层 VLAN 中的网络基础设施通信。

VXLAN 和 GRE：VXLAN 和 GRE 使用网络叠加机制支持 instance 间的私有通信，只不过此种场景中要求有 Networking router 以进行 GRE 或 VXLAN 与外部流量间的转换。此外，要实现 tenant network 与外部网络包包括 Internet 网络进行通信时也需要用到 router。事实上，router 提供了基于 floating IP 的机制从外部网络访问 instance 的能力。

Provider network 是由 Openstack 管理员创建的直接映射至数据中心的物理网络的网络，其支持的网络类型有 flat 和 VLAN。

4.1.6 Load-Balancer-as-a-Service (LBaaS)

Load-Balancer-as-a-Service (LBaaS)机制给 Networking 添加了将入站请求均衡于多个 instance 的能力。其支持的负载均衡算法有“Round robin”、“Source IP”以及“Least connections”。此外，LBaaS 还支持“Monitors”、“Management”、“Connection limits”和“Session persistence”等特性。

4.1.7 Firewall-as-a-Service (FWaaS)

Firewall-as-a-Service (FWaaS)是一个插件，它允许 Networking 实现的用户实现防火墙功能。FWaaS 通过 iptables 实现防火墙策略。在单个 project 中支持一个防火墙策略及逻辑防火墙实例。尽管 security group 是在 instance 级别实现，但 FWaaS 却是在 neutron 的 router 上实现流量过滤。

4.2 neutron server 节点

在实际部署的架构中，neutron 的部署架构可以分为三个角色，即 Neutron Server（Neutron 服务器）、network node（网络节点）和 compute node（计算节点）。这里先部署 Neutron 服务器。

4.2.1 安装相关的包

此处配置的为 neutron server 服务，根据此前的规划，这里将其部署在控制节点上。首先使用如下命令安装相关的程序包。

yum install openstack-neutron openstack-neutron-ml2 python-neutronclient

4.2.2 为 neutron 准备依赖的数据库

创建 neutron 数据库，同时创建其服务同名的用户，并为其指定密码，这里选择使用与服务名同名的密码 neutron。
openstack-db --init --service neutron --password neutron

不同于 nova 等服务的是 neutron 无需事先导入数据库表，因为其服务启动时会自动创建，但需要事先为其创建访问相关数据库的用户账号及密码等相关信息。

也可以使用如下命令创建相关的数据库及用户。
mysql -uroot -p
mysql> CREATE DATABASE neutron;
mysql> GRANT ALL ON neutron.* TO 'neutron'@'%' IDENTIFIED BY 'neutron';
mysql> GRANT ALL ON neutron.* TO 'neutron'@'localhost' IDENTIFIED BY 'neutron';
mysql> FLUSH PRIVILEGES;

4.2.3 在 keystone 中创建 neutron 用户

首先创建 neutron 用户。
keystone user-create --name neutron --pass neutron --email neutron@mageedu.com
+----------+----------------------------------+
| Property | Value |
+----------+----------------------------------+
email	neutron@mageedu.com
enabled	True
id	be63aab578f54eea829260a6e6d59b55
name	neutron
username	neutron

```
+----------+----------------------------------+
```

还需要将 neutron 用户链接至 service tenant 及 admin role。

```
# keystone user-role-add --user neutron --tenant service --role admin
```

而后创建 neutron 服务。

```
# keystone service-create --name neutron --type network --description "OpenStack Networking"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |       OpenStack Networking       |
|   enabled   |               True               |
|      id     | 70304acabb2c404cbdc03f6c3b9647ad |
|     name    |              neutron             |
|     type    |              network             |
+-------------+----------------------------------+
```

为 neutron 服务创建访问端点。

```
$ keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ network / {print $2}') \
  --publicurl http://controller:9696 \
  --adminurl http://controller:9696 \
  --internalurl http://controller:9696
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
|   adminurl  |      http://controller:9696      |
|      id     | 2f89da58050c4b89b25ad8e2b1c55d07 |
| internalurl |      http://controller:9696      |
|  publicurl  |      http://controller:9696      |
|    region   |             regionOne            |
|  service_id | 70304acabb2c404cbdc03f6c3b9647ad |
+-------------+----------------------------------+
```

4.2.4 配置 neutron server

首先，配置 neutron 连接数据库服务器的 URL。

```
# openstack-config --set /etc/neutron/neutron.conf database connection \
  mysql://neutron:neutron@controller/neutron
```

配置 Neutron Server 使用的消息队列服务。

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT rpc_backend
neutron.openstack.common.rpc.impl_qpid
```

openstack-config --set /etc/neutron/neutron.conf DEFAULT qpid_hostname controller

运行如下命令配置 Neutron Server 连入 keystone。

```
openstack-config --set /etc/neutron/neutron.conf DEFAULT auth_strategy keystone
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_uri http://controller:5000
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_host controller
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_protocol http
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_port 35357
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken admin_tenant_name service
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken admin_user neutron
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken admin_password neutron
```

运行如下命令配置 Neutron Server 通知 Compute 节点相关的网络定义的改变。

```
openstack-config --set /etc/neutron/neutron.conf DEFAULT notify_nova_on_port_status_changes True
openstack-config --set /etc/neutron/neutron.conf DEFAULT notify_nova_on_port_data_changes True
openstack-config --set /etc/neutron/neutron.conf DEFAULT nova_url http://controller:8774/v2
openstack-config --set /etc/neutron/neutron.conf DEFAULT nova_admin_username nova
openstack-config --set /etc/neutron/neutron.conf DEFAULT nova_admin_tenant_id $(keystone tenant-list | awk '/ service / { print $2 }')
openstack-config --set /etc/neutron/neutron.conf DEFAULT nova_admin_password nova
openstack-config --set /etc/neutron/neutron.conf DEFAULT nova_admin_auth_url http://controller:35357/v2.0
```

配置 Neutron Server 使用 Modular Layer 2 (ML2)插件及相关的服务

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT core_plugin ml2
# openstack-config --set /etc/neutron/neutron.conf DEFAULT service_plugins router
```

运行如下命令配置 ML2（Modular Layer 2）插件。ML2 插件使用 Open vSwitch(ovs)机制（agent)构建虚拟网络。不过，控制节点无需 ovs agent 或服务，因为其并不处理 vm instance 的网络流量。

```
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 type_drivers gre
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 tenant_network_types gre
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 mechanism_drivers openvswitch
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2_type_gre tunnel_id_ranges 1:1000
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini securitygroup \
  firewall_driver neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini securitygroup \
  enable_security_group True
```

注意：如果需要 ml2 支持更多的驱动类型，可将上面一组中的命令的第一个和第二个分别更换为：

```
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 type_drivers local,flat,vlan,gre,vxlan
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 tenant_network_types vlan,gre,vxlan
```

Networking 服务初始化脚本需要通过符号链接文件/etc/neutron/plugin.ini 链接至选择使用的插件，例如，

本案例中的/etc/neutron/plugins/ml2/mls_config.ini 文件。这可以使用如下命令进行。

```
# cd    /etc/neutron
# ln -s plugins/ml2/ml2_conf.ini /etc/neutron/plugin.ini
```

4.2.5 配置 Compute 服务

默认情况下，大多数发行版中的 compute 默认使用传统的网络服务功能，此时，必须配置 Compute service 使用 Networking。在 Neutron Server 运行如下命令即可。

```
 openstack-config --set /etc/nova/nova.conf DEFAULT network_api_class nova.network.neutronv2.api.API
 openstack-config --set /etc/nova/nova.conf DEFAULT neutron_url http://controller:9696
 openstack-config --set /etc/nova/nova.conf DEFAULT neutron_auth_strategy keystone
 openstack-config --set /etc/nova/nova.conf DEFAULT neutron_admin_tenant_name service
 openstack-config --set /etc/nova/nova.conf DEFAULT neutron_admin_username neutron
 openstack-config --set /etc/nova/nova.conf DEFAULT neutron_admin_password neutron
 openstack-config --set /etc/nova/nova.conf DEFAULT neutron_admin_auth_url http://controller:35357/v2.0
 openstack-config --set /etc/nova/nova.conf DEFAULT \
   linuxnet_interface_driver nova.network.linux_net.LinuxOVSInterfaceDriver
 openstack-config --set /etc/nova/nova.conf DEFAULT \
   firewall_driver nova.virt.firewall.NoopFirewallDriver
 openstack-config --set /etc/nova/nova.conf DEFAULT security_group_api neutron
```

注意：运行如下命令，设定虚拟网络接口插件的工作属性。

```
 openstack-config --set /etc/nova/nova.conf DEFAULT vif_plugging_timeout 10
 openstack-config --set /etc/nova/nova.conf DEFAULT vif_plugging_is_fatal False
```

而后重启 Compute 相关的服务，使得配置生效。

```
# service openstack-nova-api restart
# service openstack-nova-scheduler restart
# service openstack-nova-conductor restart
```

或：

```
# for svc in api scheduler conductor; do service openstack-nova-${svc} restart; done
```

4.2.6 启动 Neutron Server

```
# service neutron-server start
# chkconfig neutron-server on
```

注意（非必要步骤）：openstack 的有些发行版有时需要在启动 neutron server 之前运行"neutron-db-manage"命令进行初始配置。其步骤如下。

首先，配置 Networking 使用长格式的插件名称。

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
```

```
    core_plugin neutron.plugins.ml2.plugin.Ml2Plugin
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
    service_plugins neutron.services.l3_router.l3_router_plugin.L3RouterPlugin
```

其次，导入数据库。

```
# su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/neutron.conf \
    --config-file /etc/neutron/plugin.ini upgrade head" neutron
```

最后启动或重启 neutron-server 服务即可。

4.3 network 节点

在本配置示例中，192.168.10.9 为 network node，因此，下面的配置过程均在此节点进行。

4.3.1 配置内核中的网络功能

编辑/etc/sysctl.conf，确定如下参数的值为其所示。

```
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

让配置生效。

```
# sysctl -p
```

4.3.2 安装配置 network node

安装相关的程序包。

```
# yum install openstack-neutron openstack-neutron-ml2 openstack-neutron-openvswitch
```

4.3.2.1 基本配置

配置其使用的消息队列服务。

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT rpc_backend
neutron.openstack.common.rpc.impl_qpid
# openstack-config --set /etc/neutron/neutron.conf DEFAULT qpid_hostname controller
```

运行如下命令配置 Networking node 连入 keystone。

```
  openstack-config --set /etc/neutron/neutron.conf DEFAULT auth_strategy keystone
  openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_uri http://controller:5000
  openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_host controller
  openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_protocol http
```

```
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_port 35357
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken admin_tenant_name service
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken admin_user neutron
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken admin_password neutron
```

配置 Neutron Server 使用 Modular Layer 2 (ML2)插件及相关的服务

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT core_plugin ml2
# openstack-config --set /etc/neutron/neutron.conf DEFAULT service_plugins router
```

配置 Layer-3 (L3) agent

```
# openstack-config --set /etc/neutron/l3_agent.ini DEFAULT interface_driver neutron.agent.linux.interface.OVSInterfaceDriver
# openstack-config --set /etc/neutron/l3_agent.ini DEFAULT use_namespaces True
```

4.3.2.2 配置 DHCP agent

运行如下命令，生成相应的配置信息

```
openstack-config --set /etc/neutron/dhcp_agent.ini DEFAULT \
  interface_driver neutron.agent.linux.interface.OVSInterfaceDriver
openstack-config --set /etc/neutron/dhcp_agent.ini DEFAULT \
  dhcp_driver neutron.agent.linux.dhcp.Dnsmasq
openstack-config --set /etc/neutron/dhcp_agent.ini DEFAULT use_namespaces True
```

类似于 GRE 类的隧道协议为报文添加了额外的首部而导致报文体积增大。传统以太网中的 MTU 为 1500 字节，但 IP 协议能够通过 PMTUD（path MTU discovery）机制来动态判断通信双方的 MTU 大小。不过，那些缺少 PMTUD 支持的 OS 可能会因 MTU 过大而导致网络不通或引起网络性能问题。

理想情况下，可以通过在包含了 tenant 虚拟网络的物理网络上使用 jumbo 帧（现代以太网的特性，支持 9000 字节的帧）来规避此问题。然而依然有相当不少的 OS 依然不支持 jumbo 帧。此时，就能通过在各 instance 上缩减可用的 MTU 大小来解决此问题了。实践中，可以通过配置 DHCP 服务在分配 IP 地址时通知 instance 调整其 MTU 的大小。

(1) 运行如下命令配置 neutron 中 dhcp 服务使用自定义的配置文件。

```
# openstack-config --set /etc/neutron/dhcp_agent.ini DEFAULT \
  dnsmasq_config_file /etc/neutron/dnsmasq-neutron.conf
```

(2) 创建/etc/neutron/dnsmasq-neutron.conf 文件，并添加如下内容。

```
dhcp-option-force=26,1454
```

(3) 关闭所有的 dnsmasq 进程，等其重启后生效新配置。

```
# killall dnsmasq
```

4.3.2.3 配置 metadata agent

metadata agent 提供了类似于实例远程访问时的认证信息等配置。

在 network node 上运行如下命令，其中的 METADATA_SECRET 按需修改即可。

```
openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT auth_url http://controller:5000/v2.0
openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT auth_region regionOne
openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT admin_tenant_name service
openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT admin_user neutron
openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT admin_password neutron
openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT nova_metadata_ip controller
openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT metadata_proxy_shared_secret METADATA_SECRET
```

在控制节点上运行如下两步，其中的 METADATA_SECRET 要替换成与前面选择的相关的密码。

第一步：

```
# openstack-config --set /etc/nova/nova.conf DEFAULT service_neutron_metadata_proxy true
# openstack-config --set /etc/nova/nova.conf DEFAULT neutron_metadata_proxy_shared_secret METADATA_SECRET
```

第二步：重启控制节点上的 openstack-nova-api 服务。

```
# service openstack-nova-api restart
```

4.3.2.3 配置 ML2 插件

运行如下命令配置 ML2 插件，其中 10.0.1.9 为隧道接口的地址。

```
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 type_drivers gre
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 tenant_network_types gre
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2    mechanism_drivers openvswitch
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2_type_gre tunnel_id_ranges 1:1000
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs local_ip 10.0.1.9
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs tunnel_type gre
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs enable_tunneling True
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini securitygroup \
  firewall_driver neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini securitygroup enable_security_group True
```

注意：如果需要 ml2 支持更多的驱动类型，可将上面一组中的命令的第一个和第二个分别更换为：

```
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 type_driverslocal,flat,vlan,gre,vxlan
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 tenant_network_types vlan,gre,vxlan
```

4.3.2.4 配置 Open vSwitch (OVS) service

OVS 服务为 vm instances 提供了底层虚拟网络帧，其 br-int 桥用于处理各 instance 内部通信流量，而 br-ex 桥用于处理外部通信流量。

(1) 启动 OVS 服务
service openvswitch start
chkconfig openvswitch on

(2) 添加桥设备
ovs-vsctl add-br br-int

(3) 添加外部桥
ovs-vsctl add-br br-ex

(4) 为外部桥添加一下 port 以接入物理的外部网络接口，其中的 INTERFACE_NAME 为实际的物理接口
ovs-vsctl add-port br-ex INTERFACE_NAME

(5) 修改桥设备 br-ex 的 bridge-id 的属性值为 br-ex
ovs-vsctl br-set-external-id br-ex bridge-id br-ex

注意：此步骤为了保证后面的操作中添加路由器后其路由接口为 Active 状态，否则，由于 bug，其很可能为 DOWN。

4.3.3 配置并启动服务

Networking 服务初始化脚本需要通过符号链接文件/etc/neutron/plugin.ini 链接至选择使用的插件，例如，本案例中的/etc/neutron/plugins/ml2/mls_config.ini 文件。这可以使用如下命令进行。
cd /etc/neutron
ln -s plugins/ml2/ml2_conf.ini /etc/neutron/plugin.ini

由于包处理的 bug，Open vSwitch agent 初始化脚本会去查看 Open vSwitch 插件配置文件而不是指向 ML2 插件配置文件的符号链接文件/etc/neutron/plugin.ini，以下命令可修改这个问题。
cp /etc/init.d/neutron-openvswitch-agent /etc/init.d/neutron-openvswitch-agent.orig
sed -i 's,plugins/openvswitch/ovs_neutron_plugin.ini,plugin.ini,g' /etc/init.d/neutron-openvswitch-agent

下面启动相关的服务即可。
service neutron-openvswitch-agent start
service neutron-l3-agent start
service neutron-dhcp-agent start
service neutron-metadata-agent start
chkconfig neutron-openvswitch-agent on
chkconfig neutron-l3-agent on
chkconfig neutron-dhcp-agent on

chkconfig neutron-metadata-agent on

或者使用如下命令：
for svc in openvswitch-agent l3-agent dhcp-agent metadata-agent; do service neutron-${svc} start; chkconfig neutron-${svc} on; done

for svc in openvswitch-agent l3-agent dhcp-agent metadata-agent; do service neutron-${svc} restart; done

4.4 Compute 节点

在本配置示例中，192.168.10.8 为 Compute node，因此，下面的配置过程均在此节点进行。

4.4.1 配置内核中的网络功能

编辑/etc/sysctl.conf，确定如下参数的值为其所示。
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0

让配置生效。
sysctl -p

4.4.2 安装并配置 neutron 相关的组件

yum install openstack-neutron-ml2 openstack-neutron-openvswitch

4.4.2.1 基本配置

配置其使用的消息队列服务。
openstack-config --set /etc/neutron/neutron.conf DEFAULT rpc_backend neutron.openstack.common.rpc.impl_qpid
openstack-config --set /etc/neutron/neutron.conf DEFAULT qpid_hostname controller

运行如下命令配置 Networking node 连入 keystone。
```
openstack-config --set /etc/neutron/neutron.conf DEFAULT auth_strategy keystone
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_uri http://controller:5000
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_host controller
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_protocol http
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken auth_port 35357
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken admin_tenant_name service
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken admin_user neutron
openstack-config --set /etc/neutron/neutron.conf keystone_authtoken admin_password neutron
```

指定 Image Service 主机的地址，本示例中为 controller。

```
openstack-config --set /etc/nova/nova.conf DEFAULT glance_host controller
```

配置 Neutron Server 使用 Modular Layer 2 (ML2)插件及相关的服务

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT core_plugin ml2
# openstack-config --set /etc/neutron/neutron.conf DEFAULT service_plugins router
```

4.4.2.2 配置 ML2 插件

运行如下命令配置 ML2 插件，其中 10.0.10.8 为本节点用于“隧道接口”的地址。

```
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 type_drivers gre
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 tenant_network_types gre
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 mechanism_drivers openvswitch
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2_type_gre tunnel_id_ranges 1:1000
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs local_ip 10.0.1.8
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs tunnel_type gre
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs enable_tunneling True
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini securitygroup \
  firewall_driver neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini securitygroup \
  enable_security_group True
```

注意：如果需要 ml2 支持更多的驱动类型，可将上面一组中的命令的第一个和第二个分别更换为：

```
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 type_driverslocal,flat,vlan,gre,vxlan
openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 tenant_network_types vlan,gre,vxlan
```

4.4.2.3 配置 Open vSwitch (OVS) service

OVS 服务为 vm instances 提供了底层虚拟网络帧，其 br-int 桥用于处理各 instance 内部通信流量。

(1) 启动 OVS 服务

```
# service openvswitch start
# chkconfig openvswitch on
```

(2) 添加桥设备

```
# ovs-vsctl add-br br-int
```

4.4.2.4 配置 Compute 使用 Networking 服务

运行如下命令，配置 Compute 使用 Networking 服务

```
openstack-config --set /etc/nova/nova.conf DEFAULT \
  network_api_class nova.network.neutronv2.api.API
openstack-config --set /etc/nova/nova.conf DEFAULT neutron_url http://controller:9696
openstack-config --set /etc/nova/nova.conf DEFAULT neutron_auth_strategy keystone
```

```
openstack-config --set /etc/nova/nova.conf DEFAULT neutron_admin_tenant_name service
openstack-config --set /etc/nova/nova.conf DEFAULT neutron_admin_username neutron
openstack-config --set /etc/nova/nova.conf DEFAULT neutron_admin_password neutron
openstack-config --set /etc/nova/nova.conf DEFAULT neutron_admin_auth_url http://controller:35357/v2.0
openstack-config --set /etc/nova/nova.conf DEFAULT \
  linuxnet_interface_driver nova.network.linux_net.LinuxOVSInterfaceDriver
openstack-config --set /etc/nova/nova.conf DEFAULT \
  firewall_driver nova.virt.firewall.NoopFirewallDriver
openstack-config --set /etc/nova/nova.conf DEFAULT security_group_api neutron
```

4.4.3 配置并启动服务

Networking 服务初始化脚本需要通过符号链接文件/etc/neutron/plugin.ini 链接至选择使用的插件，例如，本案例中的/etc/neutron/plugins/ml2/mls_config.ini 文件。这可以使用如下命令进行。

```
# cd    /etc/neutron
# ln -s plugins/ml2/ml2_conf.ini /etc/neutron/plugin.ini
```

由于包处理的 bug，Open vSwitch agent 初始化脚本会去查看 Open vSwitch 插件配置文件而不是指向 ML2 插件配置文件的符号链接文件/etc/neutron/plugin.ini，以下命令可修改这个问题。

```
# cp /etc/init.d/neutron-openvswitch-agent /etc/init.d/neutron-openvswitch-agent.orig
# sed -i 's,plugins/openvswitch/ovs_neutron_plugin.ini,plugin.ini,g' /etc/init.d/neutron-openvswitch-agent
```

接下来重启 openstack-nova-compute 服务。

```
# service openstack-nova-compute restart
```

而后启动 OVS agent 并配置其开机自动启动。

```
# service neutron-openvswitch-agent start
# chkconfig neutron-openvswitch-agent on
```

4.5 创建初始网络

外部网络用于为各 instance 提供访问互联网的能力。默认情况下，此网络仅允许通过 NAT 机制访问 Internet，但也可以通过为单个实现配置一个 floating IP 以及适当的 security group 从而让互联网上的主机能访问到此 instance。admin tenant 要为多个 tenant 提供访问 Internet 的能力，因此外部网络位于 admin tenant 中。

4.5.1 创建外部网络

在 Contoller 上运行如下命令。

创建一个外部网络：

```
# neutron net-create ext-net --shared --router:external=True
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 8a26872a-3135-43ec-9946-b13b58015124 |
| name                      | ext-net                              |
| provider:network_type     | gre                                  |
| provider:physical_network |                                      |
| provider:segmentation_id  | 1                                    |
| router:external           | True                                 |
| shared                    | True                                 |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | 3b655869dfcd409a9e7649769d5dfa57     |
+---------------------------+--------------------------------------+
```

Like a physical network, a virtual network requires a subnet assigned to it. The external network shares the same subnet and gateway associated with the physical network connected to the external interface on the network node. You should specify an exclusive slice of this subnet for router and floating IP addresses to prevent interference with other devices on the external network.

Replace FLOATING_IP_START and FLOATING_IP_END with the first and last IP addresses of the range that you want to allocate for floating IP addresses. Replace EXTERNAL_NETWORK_CIDR with the subnet associated with the physical network. Replace EXTERNAL_NETWORK_GATEWAY with the gateway associated with the physical network, typically the ".1" IP address. You should disable DHCP on this subnet because instances do not connect directly to the external network and floating IP addresses require manual assignment.

在外部网络中创建一个子网

创建 subnet 使用类似如下的命令格式：

```
# neutron subnet-create ext-net --name ext-subnet \
  --allocation-pool start=FLOATING_IP_START,end=FLOATING_IP_END \
  --disable-dhcp --gateway EXTERNAL_NETWORK_GATEWAY EXTERNAL_NETWORK_CIDR
```

例如下面的命令就在外部网络上创建了一个子网。

```
# neutron subnet-create ext-net --name ext-subnet \
  --allocation-pool start=172.16.200.151,end=172.16.200.180 \
  --disable-dhcp --gateway 172.16.0.1 172.16.0.0/16
Created a new subnet:
+------------------+------------------------------------------------------+
```

| Field | Value |
+-----------------+---+
allocation_pools	{"start": "172.16.200.151", "end": "172.16.200.180"}
cidr	172.16.0.0/16
dns_nameservers	
enable_dhcp	False
gateway_ip	172.16.0.1
host_routes	
id	2cee43ae-d1e8-4fdb-8523-e0f01c593a91
ip_version	4
name	ext-subnet
network_id	8a26872a-3135-43ec-9946-b13b58015124
tenant_id	3b655869dfcd409a9e7649769d5dfa57
+-----------------+---+

4.5.2 Tenant network

tenant network 为各 instance 之间提供了内部互访的通道，此机制用于实现各 tenant 网络之间的隔离。例如为前文中创建的 demo tenant 创建一个内部网络 demo net。

在 Controller 节点上运行如下命令。

```
# neutron net-create demo-net
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 066090a7-285a-4d2c-9f9a-981f4edc1014 |
| name                      | demo-net                             |
| provider:network_type     | gre                                  |
| provider:physical_network |                                      |
| provider:segmentation_id  | 2                                    |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | 3b655869dfcd409a9e7649769d5dfa57     |
+---------------------------+--------------------------------------+
```

Like the external network, your tenant network also requires a subnet attached to it. You can specify any valid subnet because the architecture isolates tenant networks. Replace TENANT_NETWORK_CIDR with the subnet you want to associate with the tenant network. Replace TENANT_NETWORK_GATEWAY with the gateway you want to associate with this network, typically the ".1" IP address. By default, this subnet will use DHCP so your instances can obtain IP addresses.

接下来为 tenant network 创建一个 subnet，其语法格式如下：
neutron subnet-create demo-net --name demo-subnet --gateway TENANT_NETWORK_GATEWAY TENANT_NETWORK_CIDR

例如下面的部分为 demo-net 网络创建了一个子网。
neutron subnet-create demo-net --name demo-subnet --gateway 192.168.27.1 192.168.27.0/24
Created a new subnet:
+-------------------+--+
| Field | Value |
+-------------------+--+
allocation_pools	{"start": "192.168.27.2", "end": "192.168.27.254"}
cidr	192.168.27.0/24
dns_nameservers	
enable_dhcp	True
gateway_ip	192.168.27.1
host_routes	
id	4f742d27-948d-452f-b467-783968b6ee34
ip_version	4
name	demo-subnet
network_id	066090a7-285a-4d2c-9f9a-981f4edc1014
tenant_id	3b655869dfcd409a9e7649769d5dfa57
+-------------------+--+

A virtual router passes network traffic between two or more virtual networks. Each router requires one or more interfaces and/or gateways that provide access to specific networks. In this case, you will create a router and attach your tenant and external networks to it.

接下来为 demo net 创建一个 router，并将其附加至外部网络和 demo net。
neutron router-create demo-router
Created a new router:
+-----------------------+--------------------------------------+
| Field | Value |
+-----------------------+--------------------------------------+
admin_state_up	True
external_gateway_info	
id	62230691-4147-4126-a5bd-1570f1737b43
name	demo-router
status	ACTIVE
tenant_id	3b655869dfcd409a9e7649769d5dfa57
+-----------------------+--------------------------------------+

将新创建的 router 关联至 tenant network(demo net)。
neutron router-interface-add demo-router demo-subnet

Added interface 8a32211d-5cb3-47cc-b5e3-00a7da9ccac9 to router demo-router.

通过将其设置为网关的方式把新建的 router 附加至外部网络。

```
# neutron router-gateway-set demo-router ext-net
Set gateway for router demo-router
```

列出已经创建的网络：

```
# neutron net-list
+--------------------------------------+----------+------------------------------------------------------+
| id                                   | name     | subnets                                              |
+--------------------------------------+----------+------------------------------------------------------+
| 8a26872a-3135-43ec-9946-b13b58015124 | ext-net  | 2cee43ae-d1e8-4fdb-8523-e0f01c593a91 172.16.0.0/16   |
| 066090a7-285a-4d2c-9f9a-981f4edc1014 | demo-net | 4f742d27-948d-452f-b467-783968b6ee34 192.168.27.0/24 |
+--------------------------------------+----------+------------------------------------------------------+
```

在 network 节点上查看刚创建的 router 的名称空间的接口及地址配置情况：

```
# ip netns exec qrouter-62230691-4147-4126-a5bd-1570f1737b43 ip addr show
17: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
18: qr-8a32211d-5c: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether fa:16:3e:d7:3f:7f brd ff:ff:ff:ff:ff:ff
    inet 192.168.27.1/24 brd 192.168.27.255 scope global qr-8a32211d-5c
    inet6 fe80::f816:3eff:fed7:3f7f/64 scope link
       valid_lft forever preferred_lft forever
19: qg-2eb2a806-e8: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether fa:16:3e:61:81:d9 brd ff:ff:ff:ff:ff:ff
    inet 172.16.200.151/16 brd 172.16.255.255 scope global qg-2eb2a806-e8
    inet6 fe80::f816:3eff:fe61:81d9/64 scope link
       valid_lft forever preferred_lft forever
```

补充：叠加子网（Overlapping subnets）和网络名称空间（network namespace）

基于主机的定义路由的一个潜在风险是 Openstack Networking 中定义的子网可能会使用与物理主机所在的物理网络相同的子网，例如物理机的 eth1 碰巧与其中的一个虚拟的子网使用了相同的网络 10.0.10.0/24，其结果是必将引起路由故障。因此，如果允许用户在 tenant 中自定义网络时必须使用技术

手段规避这个潜在的问题。

网络名称空间拥有其自己的接口、路由和 iptables 规则，因此可以把它想象成具有独立网络栈的与外界隔离的独立网络环境。这甚至有点类似于文件系统的 chroot，只不过，这里独立的不是文件系统而网络罢了。事实上，LXC 就使用网络名称空间进行网络功能的虚拟化。而 OpenStack 也正是使用网络名称空间来避免子网冲突的。

详细信息请参考官方文档：http://docs.openstack.org/grizzly/openstack-network/admin/content/under_the_hood_openvswitch.html 和 http://docs.openstack.org/grizzly/openstack-network/admin/content/under_the_hood_linuxbridge.html。

五、dashboard

5.1 安装

```
# yum install openstack-dashboard httpd mod_wsgi memcached python-memcached
```

5.2 配置

编辑/etc/openstack-dashboard/local_settings

(1) 指定 controller 节点
```
OPENSTACK_HOST = "controller"
```

(2) 配置访问权限
```
ALLOWED_HOSTS = ['*']
```

(3) 配置使用本机上的 memcached 作为会话缓存
```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

(4) 设置好时区
```
TIME_ZONE = "TIME_ZONE"
```

5.3 访问

启动 memcached 和 httpd 服务。
```
service httpd start
```

service memcached start
chkconfig httpd on
chkconfig memcached on

http://controller/dashboard

六、启动实例

6.1 SSH 公钥注入

只要对应的 VM 实例支持使用 SSH 服务，Compute 服务可以注入 SSH 公钥信息至此实例的某帐号中。“nova keypair-add”命令能够生成一对密钥(不保存至文件中)，并将其公钥添加至 Compute 服务；当然，也可以只用于将现有的某密钥对儿的公钥添加至 Compute 服务中。为了使用上的方便，这里先使用 ssh-keygen 命令生成一对密钥文件，而后将其公钥上传至 Compute 服务中。

```
# ssh-keygen -t rsa -P ''
# nova keypair-add --pub-key ~/.ssh/id_rsa.pub demokey
# nova keypair-list
+----------+-------------------------------------------------+
| Name     | Fingerprint                                     |
+----------+-------------------------------------------------+
| demotkey | 3b:df:f0:57:d0:ba:7d:02:f4:32:47:00:d3:53:e8:1f |
+----------+-------------------------------------------------+
```

6.2 启动一个 instance

6.2.1 flavor

在 OpenStack 中启动实例需要指定一个 VM 配置模板，即 flavor。“nova flavor-list”命令可用于查看所有可用的 flavor。

```
# nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
| 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0         | True      |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0         | True      |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0         | True      |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0         | True      |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0         | True      |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
```

创建一个拥有较小的内存设置的 flavor，供启动 cirror 测试使用：

```
# nova flavor-create --is-public true m1.cirros 6 256 1 1
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
| 6  | m1.cirros | 256       | 1    | 0         |      | 1     | 1.0         | True      |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+

# nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
| 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0         | True      |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0         | True      |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0         | True      |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0         | True      |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0         | True      |
| 6  | m1.cirros | 256       | 1    | 0         |      | 1     | 1.0         | True      |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
```

6.2.2 获取所有可用的 image 文件列表

启动一个 instance 时需要为其指定一个虚拟映像文件，并以之做为启动时用到的硬盘设备。

```
# nova image-list
+--------------------------------------+---------------------+--------+--------+
| ID                                   | Name                | Status | Server |
+--------------------------------------+---------------------+--------+--------+
| 86f16760-0616-4c2e-899f-a1182260705a | cirros-0.3.3-i386   | ACTIVE |        |
| b6c92772-f3d0-480d-ba90-0ad8c3669ea8 | cirros-0.3.3-x86_64 | ACTIVE |        |
+--------------------------------------+---------------------+--------+--------+
```

6.3.3 获取所有可用的网络列表

启动一个 instance 时，通常需要为其指定一个网络接口，并以之做为此 instance 的网络接口。

```
# neutron net-list
+--------------------------------------+----------+------------------------------------------------------+
| id                                   | name     | subnets                                              |
+--------------------------------------+----------+------------------------------------------------------+
| 8a26872a-3135-43ec-9946-b13b58015124 | ext-net  | 2cee43ae-d1e8-4fdb-8523-e0f01c593a91 172.16.0.0/16   |
| 066090a7-285a-4d2c-9f9a-981f4edc1014 | demo-net | 4f742d27-948d-452f-b467-783968b6ee34                 |
```

192.168.27.0/24 |
+--------------------------------------+----------+--+

6.3.4 获取所有可用的安全组列表

nova 通过安全组为 vm 部署安全访问策略。nova 安装完成后，会自动创建一个名为 default 的默认的安全组。启动实例时，通常需要为其指定一个可用的安全组。

```
# nova secgroup-list
+--------------------------------------+---------+-------------+
| Id                                   | Name    | Description |
+--------------------------------------+---------+-------------+
| 1524cd4e-0d2a-424a-8b8b-b0766d6f212c | default | default     |
+--------------------------------------+---------+-------------+
```

6.3.5 启动 instance

```
# nova boot --flavor m1.cirros --image cirros-0.3.3-i386 --nic net-id=$(nova net-list | awk '/demo-net/{print $2}') \
  --security-group default --key-name demokey demo-i1
+--------------------------------------+------------------------------------------------------------+
| Property                             | Value                                                      |
+--------------------------------------+------------------------------------------------------------+
| OS-DCF:diskConfig                    | MANUAL                                                     |
| OS-EXT-AZ:availability_zone          | nova                                                       |
| OS-EXT-SRV-ATTR:host                 | -                                                          |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | -                                                          |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000003                                          |
| OS-EXT-STS:power_state               | 0                                                          |
| OS-EXT-STS:task_state                | scheduling                                                 |
| OS-EXT-STS:vm_state                  | building                                                   |
| OS-SRV-USG:launched_at               | -                                                          |
| OS-SRV-USG:terminated_at             | -                                                          |
| accessIPv4                           |                                                            |
```

|
| accessIPv6 |
|
| adminPass | ThuRUGhU2H2s
|
| config_drive |
|
| created | 2014-10-19T06:50:29Z
|
| flavor | m1.cirros (6)
|
| hostId |
|
| id | 70af5516-b0ce-4908-823b-fdb5ee0ce22b
|
| image | cirros-0.3.3-i386 (86f16760-0616-4c2e-899f-a1182260705a) |
| key_name | demokey
|
| metadata | {}
|
| name | demo-i1
|
| os-extended-volumes:volumes_attached | []
|
| progress | 0
|
| security_groups | default
|
| status | BUILD
|
| tenant_id | 3b655869dfcd409a9e7649769d5dfa57
|
| updated | 2014-10-19T06:50:29Z
|
| user_id | 07002d00863742c7b02e2856587198a7
|
+--------------------------------------+--+

列出所有 instace：
nova list
+--------------------------------------+---------+--------+------------+-------------+----------------------+
| ID | Name | Status | Task State | Power State | Networks
|
+--------------------------------------+---------+--------+------------+-------------+----------------------+

| 70af5516-b0ce-4908-823b-fdb5ee0ce22b | demo-i1 | BUILD | spawning | NOSTATE | demo-net=192.168.27.2 |
+--------------------------------------+---------+--------+------------+-------------+----------------------+

nova list
+--------------------------------------+---------+--------+------------+-------------+----------------------+
| ID | Name | Status | Task State | Power State | Networks |
+--------------------------------------+---------+--------+------------+-------------+----------------------+
| 70af5516-b0ce-4908-823b-fdb5ee0ce22b | demo-i1 | ACTIVE | - | Running | demo-net=192.168.27.2 |
+--------------------------------------+---------+--------+------------+-------------+----------------------+

面的命令可以看到指定实例启动时的控制台日志输出。
nova console-log demo-i1

6.3.6 打开目标实例的控制台

(1) 使用本地的 vncviewer 程序

查看指定实例的详细信息可使用如下命令。
nova show demo-i1
+--------------------------------------+--+
| Property | Value |
+--------------------------------------+--+
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-SRV-ATTR:host	compute1.magedu.com
OS-EXT-SRV-ATTR:hypervisor_hostname	compute1.magedu.com
OS-EXT-SRV-ATTR:instance_name	instance-00000003
OS-EXT-STS:power_state	1
OS-EXT-STS:task_state	-
OS-EXT-STS:vm_state	active
OS-SRV-USG:launched_at	2014-10-19T06:51:08.000000

```
|      OS-SRV-USG:terminated_at                                                    |      -
|
|     accessIPv4                                                                   |
|
|     accessIPv6                                                                   |
|
|     config_drive                                                                 |
|
|    created                                                                       |    2014-10-19T06:50:29Z
|
|     demo-net     network                                                         |     192.168.27.2
|
|    flavor                                                                        |    m1.cirros    (6)
|
|     hostId                                                                       |
6f4f425f4f6a095d295e17dc5a5fe374c3efce2657400eb894afa9d5 |
|   id                                                                             |   70af5516-b0ce-4908-823b-fdb5ee0ce22b
|
| image                                    | cirros-0.3.3-i386 (86f16760-0616-4c2e-899f-a1182260705a) |
|    key_name                                                                      |    demokey
|
|    metadata                                                                      |    {}
|
|    name                                                                          |    demo-i1
|
|                          os-extended-volumes:volumes_attached                    |                    []
|
|    progress                                                                      |    0
|
|     security_groups                                                              |     default
|
|    status                                                                        |    ACTIVE
|
|   tenant_id                                                                      |   3b655869dfcd409a9e7649769d5dfa57
|
|    updated                                                                       |    2014-10-19T06:51:09Z
|
|   user_id                                                                        |   07002d00863742c7b02e2856587198a7
|
+----------------------------------+--------------------------------------------------------+
```

上面的输出结果可以看出，刚启动的实例 demo-i1 运行于 compute1.magedu.com 主机上。在 instace 运行的主机上运行下面所示的命令安装 tigervnc 程序后，可于本机通过 vncviewer 命令打开其 GUI 界面的登录控制台。

```
# yum install xauth tigervnc
# vncviewer :0
```

注：上述命令的成功运行需要在 compute 节点的图形界面，或者支持 x server 功能的 xshell 程序通过远程登录 compute 节点。

(2) novnc 控制台

首先去获取指定实例的 novnc 控制台接口地址。

```
# nova get-vnc-console demo-i1 novnc
+-------+-----------------------------------------------------------------------------+
| Type  | Url                                                                         |
+-------+-----------------------------------------------------------------------------+
| novnc | http://controller:6080/vnc_auto.html?token=96896ccc-09fc-4fb2-8545-0c2bc102e336 |
+-------+-----------------------------------------------------------------------------+
```

在能访问到 controller 的任何主机上打开浏览器输入相应的 url 即可访问。

6.3.7 测试网络的连通性

以 demo-i1 为例，其所在的网络为 demo-net 中的 demo-subnet，即 192.168.27.0/24。且前面的设定命令中可以看到，其网关为路由器 demo-router 上的接口 192.168.27.1，通往外部网络的路由器接口为 172.16.200.151，而外部网络的网关接口为 172.16.0.1，此时，即可在 demo-i1 上使用 ping 命令测试与这些接口之间的连通性了。

6.3.8 floating ip

虽然 demo-i1 已然可以与外部网络通信，但外部网络中的主机如需访问 demo-i1 上的服务，则还需要用到 floating ip 的机制。

简单来讲，floating IP 就是通过网络名称空间虚拟出一台路由器设备，其外部接口桥接至可通过物理接口与外部网络通信的网桥设备，而内部接口则做为内部网桥设备上关联的各虚拟机的网关接口，而后在外部网络接口上配置一个 IP 地址，并通过 DNAT 的方式转换至内部某指定的主机上，反过来，从内部某指定的主机上发出的报文则由路由器通过 SNAT 机制转发至外部接口上某特定的地址，从而实现了外部网络与内部 VM 的通信。

(1) 创建 floating ip

在支持 floating ip 的能与外部网络通信的网络中创建 floating ip。

```
# neutron floatingip-create ext-net
Created a new floatingip:
+---------------------+--------------------------------------+
| Field               | Value                                |
```

```
+--------------------+-------------------------------------+
| fixed_ip_address       |                                     |
| floating_ip_address | 172.16.200.152                      |
| floating_network_id | 8a26872a-3135-43ec-9946-b13b58015124 |
| id                          | 688a1c3c-2c31-451a-8eb8-0da490910160 |
| port_id                   |                                     |
| router_id                 |                                     |
| status                    | DOWN                                |
| tenant_id                 | 3b655869dfcd409a9e7649769d5dfa57    |
+--------------------+-------------------------------------+
```

(2) 将创建出的 floating ip 绑定至目标实例

使用“nova floating-ip-associate”进行绑定。

```
# nova floating-ip-associate demo-i1 172.16.200.152
```

而后检查 floating ip 的绑定情况

```
# nova list
+--------------------------------------+---------+--------+------------+-------------+-----------------------------------+
| ID                                   | Name    | Status | Task State | Power State | Networks                          |
+--------------------------------------+---------+--------+------------+-------------+-----------------------------------+
| 70af5516-b0ce-4908-823b-fdb5ee0ce22b | demo-i1 | ACTIVE | -          | Running     | demo-net=192.168.27.2, 172.16.200.152 |
+--------------------------------------+---------+--------+------------+-------------+-----------------------------------+
```

(3) 测试

通过外部网络中的主机即可对 demo-i1 通过 172.16.200.152 进行访问测试了。

注意：如果此时无法 ping 通 172.16.200.152，则有可能是 secgroup 中默认的策略所致。此时只需要在 controller 节点上执行如下命令即可。

```
# nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
```

如里还需要通过 ssh 的方式远程连接 172.16.200.152，还需要执行如下命令。

```
# nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

七、Block Storage Service

块存储服务(Block Storage Service)用以实现卷创建和删除、卷快照等与卷相关的功能，其主要有三个组件：

cinder-api：接收对卷服务的 API 请求，并将请求路由至 cinder-volume 执行相应的操作；

cinder-volume：负责从块存储数据库读取卷的相关状态以及将状态更新信息及时存储于存储数据库中，通过消息队列与其它进程进行交互以及直接访问块存储设备；它能够通过不同的驱动程序与多种不同的块存储提供者进行接驳。

cinder-sheculer 守护进程：类似于 nova-scheduler 的功能，负责从后端块存储设备中挑选合适的位置创建卷；

消息队列：在块存储的各进程间传递消息；

在 cinder 的架构中需要控制节点(Controller)和存储节点(Storage)，每个存储节点都有一个 Volume Service，若干个这样的存储节点联合起来可以构成一个存储资源池。为了支持不同类型和型号的存储，当前版本的 Cinder 为 Volume Service 如下 drivers。

本地存储：LVM, Sheepdog

网络存储： NFS, RBD (RADOS)

分布式存储：GlusterFS, Ceph

IBM: XIV, Storwize V7000， SVC storage systems

Netapp: NFS 存储；ISCSI 存储则需要 OnCommand 5.0 和 Data ONTAP 7-mode storage systems with installed iSCSI licenses

EMC: VNX, VMAX/VMAXe

Solidfire: Solidfire cluster

本示例演示以 LFS（本地文件系统）做为存储后端的实现方式，且以此前的 Controller 节点做为 Block Storage Service 的 Controller。

7.1 部署 Block Storage Service 的 Controller

7.1.1 安装及基本配置

安装相应的程序包

```
# yum install openstack-cinder
```

初始化 cinder 数据库，同时创建其服务同名的用户，并为其指定密码，这里选择使用与服务名同名的密码 cinder。

```
# openstack-db --init --service cinder --password cinder
```

如果不想使用默认的 glance 用户访问其数据，也可以使用下面的命令为 glance 创建数据库及相关的用户，注意要换需替换里面的 GLANCE_USER 和 GLANCE_PASS。

```
# mysql -uroot -p
mysql> CREATE DATABASE cinder;
mysql> GRANT ALL ON cinder.* TO 'cinder'@'%' IDENTIFIED BY 'cinder';
mysql> FLUSH PRIVILEGES;
```

如果是手动创建的 mysql 数据库及相关的用户，则还需要执行如下命令生成 glance 依赖的数据库表。

```
# su -s /bin/sh -c "cinder-manage db sync" cinder
```

配置 Block Storage Service 接入消息队列。

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT rpc_backend qpid
# openstack-config --set /etc/cinder/cinder.conf DEFAULT qpid_hostname controller
```

配置 Block Storage Service 连入数据库的 url。

```
# openstack-config --set /etc/cinder/cinder.conf \
    database connection mysql://cinder:cinder@controller/cinder
```

7.1.2 配置 cinder 服务

在 keystone 中创建 cinder 用户

首先创建 cinder 用户。

```
# keystone user-create --name cinder --pass cinder --email cinder@mageedu.com
```

还需要将 cinder 用户链接至 service tenant 及 admin role。

```
# keystone user-role-add --user=cinder --tenant=service --role=admin
```

配置 cinder 服务。

```
 openstack-config --set /etc/cinder/cinder.conf DEFAULT auth_strategy keystone
 openstack-config --set /etc/cinder/cinder.conf keystone_authtoken auth_uri http://controller:5000
 openstack-config --set /etc/cinder/cinder.conf keystone_authtoken auth_host controller
 openstack-config --set /etc/cinder/cinder.conf keystone_authtoken auth_protocol http
 openstack-config --set /etc/cinder/cinder.conf keystone_authtoken auth_port 35357
 openstack-config --set /etc/cinder/cinder.conf keystone_authtoken admin_user cinder
 openstack-config --set /etc/cinder/cinder.conf keystone_authtoken admin_tenant_name service
 openstack-config --set /etc/cinder/cinder.conf keystone_authtoken admin_password cinder
```

7.1.3 在 keystone 中注册 cinder 服务

而后创建 cinder 服务和对应的访问端点。

(1) 第一版 API 访问接口

```
# keystone service-create --name=cinder --type=volume --description="OpenStack Block Storage"
# keystone endpoint-create \
   --service-id=$(keystone service-list | awk '/ volume / {print $2}') \
   --publicurl=http://controller:8776/v1/%\(tenant_id\)s \
   --internalurl=http://controller:8776/v1/%\(tenant_id\)s \
   --adminurl=http://controller:8776/v1/%\(tenant_id\)s
```

(2) 第二版 API 访问接口

```
$ keystone service-create --name=cinderv2 --type=volumev2 --description="OpenStack Block Storage v2"
$ keystone endpoint-create \
   --service-id=$(keystone service-list | awk '/ volumev2 / {print $2}') \
```

```
  --publicurl=http://controller:8776/v2/%\(tenant_id\)s \
  --internalurl=http://controller:8776/v2/%\(tenant_id\)s \
  --adminurl=http://controller:8776/v2/%\(tenant_id\)s
```

7.1.4 启动服务

在 controller 节点运行如下命令启动服务并设置其开机自动启动。

```
service openstack-cinder-api start
service openstack-cinder-scheduler start
chkconfig openstack-cinder-api on
chkconfig openstack-cinder-scheduler on
```

7.2 配置存储节点

实践中，cinder 存储节点可能不止一台，它们都是 OpenStack 集群中的节点，因此基于 ntp 的时间同步等都需要事先做好配置。本示例中以 192.168.10.100 为例演示存储节点的实现。

7.2.1 准备逻辑卷

假设本地/dev/sdb 磁盘为 cinder 使用的逻辑卷存储盘。首先创建物理卷及卷组。

```
# pvcreate /dev/sdb
# vgcreate cinder-volumes /dev/sdb
```

在/etc/lvm/lvm.conf 文件中添加过滤器以保证 LVM 仅扫描由 VM 使用的逻辑卷，当然也应该包括系统本机使用的各逻辑卷。下面的命令中，a 表示接受扫描的逻辑卷所在的设备，而 r 表示阻止扫描的设备。配置中，本节点上所有需要由本机正常访问的物理卷都应该定义为 a 设备。

```
devices {
...
filter = [ "a/sda1/", "a/sdb/", "r/.*/"]
...
}
```

7.2.2 安装并配置 cinder 存储服务

安装相应的程序包。

```
# yum install openstack-cinder scsi-target-utils
```

配置 Block Storage 接入消息队列。

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT rpc_backend qpid
# openstack-config --set /etc/cinder/cinder.conf DEFAULT qpid_hostname controller
```

配置 Block Storage 连入数据库的 url。

```
# openstack-config --set /etc/cinder/cinder.conf \
```

database connection mysql://cinder:cinder@controller/cinder

配置 Block Storage 能够访问至 Glance 服务。
openstack-config --set /etc/cinder/cinder.conf DEFAULT glance_host controller

配置本节点提供 cinder-volume 服务里使用的接口（通常为管理网络接口的地址）。
openstack-config --set /etc/cinder/cinder.conf DEFAULT my_ip 192.168.10.100

配置 cinder 卷信息文件的存放位置（官方文档少此步骤）。
openstack-config --set /etc/cinder/cinder.conf DEFAULT volumes_dir /etc/cinder/volumes

配置 cinder 接入 keystone 认证服务。
openstack-config --set /etc/cinder/cinder.conf DEFAULT auth_strategy keystone
openstack-config --set /etc/cinder/cinder.conf keystone_authtoken auth_uri http://controller:5000
openstack-config --set /etc/cinder/cinder.conf keystone_authtoken auth_host controller
openstack-config --set /etc/cinder/cinder.conf keystone_authtoken auth_protocol http
openstack-config --set /etc/cinder/cinder.conf keystone_authtoken auth_port 35357
openstack-config --set /etc/cinder/cinder.conf keystone_authtoken admin_user cinder
openstack-config --set /etc/cinder/cinder.conf keystone_authtoken admin_tenant_name service
openstack-config --set /etc/cinder/cinder.conf keystone_authtoken admin_password cinder

7.2.3 配置 scsi-target

编辑 scsi-target 服务的配置文件/etc/tgt/targets.conf，在文件首部添加如下一行。
include /etc/cinder/volumes/*

7.2.4 启动服务

fedora 的 epel 源的中 icehouse 版本的 openstack-cinder 的服务 openstack-cinder-volume 默认为先读取 /usr/share/cinder/cinder-dist.conf 这个配置文件，而其内容是有错误的。直接启动会导致创建后的卷无法关联至 instace 上。有两种办法解决这个问题：
(1) 修改/etc/rc.d/init.d/openstack-cinder-volume 脚本，禁止服务启动时读取/usr/share/cinder/cinder-dist.conf 这个配置文件；
(2) 复制修改好的配置/etc/cinder/cinder.conf 覆盖/usr/share/cinder/cinder-dist.conf；
cp /etc/cinder/cinder.conf /usr/share/cinder/cinder-dist.conf

service openstack-cinder-volume start
service tgtd start
chkconfig openstack-cinder-volume on
chkconfig tgtd on

7.3 卷创建测试

在 cinder Controller 节点执行如下命令，创建一个 5G 大小名为 demoVolume 的逻辑卷。

```
# cinder create --display-name demoVolume 3
+---------------------+--------------------------------------+
|          Property          |                Value                 |
+---------------------+--------------------------------------+
|       attachments          |                  []                  |
|   availability_zone    |                 nova                 |
|          bootable          |                 false                |
|         created_at         |      2014-10-19T13:11:00.637837      |
| display_description |                 None                 |
|       display_name       |              demoVolume              |
|         encrypted          |                 False                |
|              id              | 32dec761-6096-481c-8fa5-1e6e4bfcb245 |
|          metadata          |                  {}                  |
|             size             |                  3                   |
|       snapshot_id        |                 None                 |
|       source_volid       |                 None                 |
|            status          |               creating               |
|       volume_type        |                 None                 |
+---------------------+--------------------------------------+
```

列出所有卷。

```
# cinder list
+--------------------------------------+-----------+--------------+------+-------------+----------+-------------+
|                  ID                  |   Status  | Display Name | Size | Volume Type | Bootable | Attached to |
+--------------------------------------+-----------+--------------+------+-------------+----------+-------------+
| 32dec761-6096-481c-8fa5-1e6e4bfcb245 | available |  demoVolume  |  3   |     None    |  false   |             |
+--------------------------------------+-----------+--------------+------+-------------+----------+-------------+
```

将此卷附加至指定的 instance 上。

```
# nova volume-attach demo-i1 32dec761-6096-481c-8fa5-1e6e4bfcb245
+----------+--------------------------------------+
| Property | Value                                |
+----------+--------------------------------------+
| device   | /dev/vdb                             |
| id       | 32dec761-6096-481c-8fa5-1e6e4bfcb245 |
| serverId | 70af5516-b0ce-4908-823b-fdb5ee0ce22b |
| volumeId | 32dec761-6096-481c-8fa5-1e6e4bfcb245 |
+----------+--------------------------------------+
```

再次执行卷查看命令，查看关联的结果。

```
# cinder list
```

```
+--------------------------------------+--------+--------------+------+-------------+----------+--------------------------------------+
|                  ID                  | Status | Display Name | Size | Volume Type | Bootable |             Attached to              |
+--------------------------------------+--------+--------------+------+-------------+----------+--------------------------------------+
| 32dec761-6096-481c-8fa5-1e6e4bfcb245 | in-use |  demoVolume  |  3   |     None    |  false   | 70af5516-b0ce-4908-823b-fdb5ee0ce22b |
+--------------------------------------+--------+--------------+------+-------------+----------+--------------------------------------+
```

接下来就可以打开对应 instance 的控制台，查看磁盘的附加状态，并对磁盘进行相应的操作了。

附录 1：Linux Network NameSpace

CentOS 6.5 的内核默认支持 Network NameSpace 的功能，但命令行工具 iproute 中的 ip 命令并不支持使用 netns 这个 object。因此，尝试使用 netns 命令之前，要先通过 rdo release 更新本机上的 iproute 为支持 netns 的版本。

1.1 ip netns 命令

ip 命令的 netns object 可用于创建、删除、监控 Network NameSpace，以及在指定的网络名称空间中执行命令等。获取命令使用帮助的方式如下所示。

```
# ip netns help
Usage: ip netns list
       ip netns add NAME
       ip netns delete NAME
       ip netns exec NAME cmd ...
       ip netns monitor
```

例如，添加一个名称空间：

```
# ip netns add ns0
```

显示所有的网络名称空间：
ip netns list

删除一个名称空间可使用类似如下方式：
ip netns delete ns0

1.2 将接口关联至名称空间

1.2.1 创建 vif 并将其添加至指定的网络名称空间

创建 vif 接口可使用“ip link”命令实现，可使用“ip link help”获取帮助信息。

例如，创建一个名为 veth0 的虚拟接口，且其对端为 veth1 接口的命令：
ip link add veth0 type veth peer name veth1

上面的命令会新添加两个接口 veth0 和 veth1，其功能类似于创建了两个直接通过网线连在一起的网卡设备：网线一端连接 veth0，另一端连接 veth1。接口的状态默认为 DOWN，“ip link show”可显示所有的接口，如下所示。
ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:96:c5:24 brd ff:ff:ff:ff:ff:ff
4: veth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether aa:11:73:26:cc:d2 brd ff:ff:ff:ff:ff:ff
5: veth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 36:96:a1:2f:23:b5 brd ff:ff:ff:ff:ff:ff

1.2.1 将接口关联至指定的网络名称空间

虚拟接口可以添加至指定的名称空间，而且之后此接口也仅为此名称空间可见。例如，如果要将虚拟接口 veth0 关联至网络名称空间 ns0：
ip link set veth0 netns veth0

此时，当前默认的网络名称空间内当不再有 veth0 接口，因为此接口已经属于网络名称空间 ns0。：
ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:96:c5:24 brd ff:ff:ff:ff:ff:ff
4: veth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether aa:11:73:26:cc:d2 brd ff:ff:ff:ff:ff:ff

可以使用“ip netns exec <NS_NAME> <CMD>”命令在指定的名称空间内执行命令，例如，想查看网络名称空间 ns0 中的接口：

```
# ip netns exec ns0 ip link show
3: lo: <LOOPBACK> mtu 16436 qdisc noop state DOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5: veth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 36:96:a1:2f:23:b5 brd ff:ff:ff:ff:ff:ff
```

从上述输出结果中可以看出，veth0 已经属于 ns0 网络名称空间。如果更习惯于其使用 eth0 为接口名称，还可以为 ns0 中的 veth0 重命名：

```
# ip netns exec ns0 ip link set name eth0 dev veth0
# ip netns exec ns0 ip link show
3: lo: <LOOPBACK> mtu 16436 qdisc noop state DOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 36:96:a1:2f:23:b5 brd ff:ff:ff:ff:ff:ff
```

上述输出结果可以看出，接口没有可用的地址，事实上，也可以使用常用的地址配置命令为些接口指定地址，例如：

```
# ip netns exec ns0 ip addr add 10.0.10.1/24 dev eth0
# ip netns exec ns0 ip addr show eth0
5: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 8a:ff:4b:19:db:03 brd ff:ff:ff:ff:ff:ff
    inet 10.0.10.1/24 scope global eth0
```

地址已经配置成功，接着可以使用如下命令激活此接口：

```
# ip netns exec ns0 ip addr show eth0
5: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 8a:ff:4b:19:db:03 brd ff:ff:ff:ff:ff:ff
    inet 10.0.10.1/24 scope global eth0
```

1.2.3 vif 接口两端互通

上述结果显示出接口已经激活（UP），但链接的状态依然为 DOWN，这是因为互联的对端接口 veth1 尚未激活。接着去激活 veth1 接口，且配置同网段地址。

```
# ip addr add 10.0.10.2/24 dev veth1
# ip link set veth1 up
# ip link show veth1
4: veth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 16:3b:e7:c9:00:85 brd ff:ff:ff:ff:ff:ff
# ip netns exec ns0 ip link show eth0
5: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
```

link/ether 8a:ff:4b:19:db:03 brd ff:ff:ff:ff:ff:ff

现在两边的接口都已经处于 UP 状态，且位于同一网段；可以通过链接连通性测试工具进行测试。例如：

```
# ping 10.0.10.2
# ip netns exec ns0 ping 10.0.10.1
```

1.2.4 名称空间互联

如前一节描述的内容中，veth0 被配置给了 ns0 网络名称空间，而 veth1 依然留在默认的网络名称空间中使用，二者可以实现互通。而事实上，也可以分别把一对 vif 接口的两端分别关联至不同的名称空间，并实现二者的通信。

```
# ip link set veth1 down
# ip netns add ns1
# ip link set veth1 netns ns1
# ip netns exec ns1 ip link set veth1 name eth0
# ip netns exec ns1 ip link show
4: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 16:3b:e7:c9:00:85 brd ff:ff:ff:ff:ff:ff
6: lo: <LOOPBACK> mtu 16436 qdisc noop state DOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

接着为 ns1 中的 eth0 配置与 ns0 的 eth0 同一网段的 IP 地址，并实现二者互通。

```
# ip netns exec ns1 ip addr add 10.0.10.3/24 dev eth0
# ip netns exec ns1 ip link set eth0 up
# ip netns exec ns1 ip addr show
4: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 16:3b:e7:c9:00:85 brd ff:ff:ff:ff:ff:ff
    inet 10.0.10.3/24 scope global eth0
    inet6 fe80::143b:e7ff:fec9:85/64 scope link
        valid_lft forever preferred_lft forever
```

探测连通性：

```
# ip netns exec ns0 ping 10.0.10.3
# ip netns exec ns1 ping 10.0.10.1
```

1.3 多名称空间互联

前面的演示过程中可以得知，虚拟网络接口 veth pair 仅可以连接两个设备，因此，仅能实现双名称空间的直接互通。如果实现多名称空间互通，得借助于 Linux 上的桥接设备实现。其实现的机制很简单，即用软件的方式模拟实现一个网桥设备，而后生成多个 veth pais 设备，每个设备一端关联至一个网络名

称空间，而另一端关联至桥设备即可。

1.3.1 brctl 命令

brctl 由 bridge-utils 程序包所提供，用于管理 Linux 系统上的桥设备。

```
# brctl help
never heard of command [(null)]
Usage: brctl [commands]
commands:
   addbr          <bridge>       add bridge
   delbr          <bridge>       delete bridge
   addif          <bridge> <device> add interface to bridge
   delif          <bridge> <device> delete interface from bridge
   …… …… ……
   …… …… ……
   show           [ <bridge> ]       show a list of bridges
   showmacs       <bridge>       show a list of mac addrs
   showstp        <bridge>       show bridge stp info
   stp            <bridge> {on|off} turn stp on/off
```

上述帮助信息显示出，“brctl addbr”可用于添加桥设备，而“brctl addif”则可向桥上关联接口设备；“brctl delbr”用于删除桥设备，而“brctl delif”则用于从桥上移除关联的接口设备。

1.3.2 多名称空间互联

首先创建一个桥设备 br-00，以及三组 veth pair 设备（veth0XX 和 veth1XX、veth0YY 和 veth1YY，以及 veth0ZZ 和 veth1ZZ）。

```
# brctl addbr br-00
# ip link set br-00 up
# brctl show
bridge name bridge id      STP enabled interfaces
br-00      8000.000000000000 no

# ip link add veth0XX type veth peer name veth1XX
# ip link add veth0YY type veth peer name veth1YY
# ip link add veth0ZZ type veth peer name veth1ZZ
```

而后创建三个名称空间 nsX、nsY 和 nsZ:

```
# ip netns add nsX
# ip netns add nsY
# ip netns add nsZ
```

接着，分别将三组 veth pair 设备的一端关联至网络名称空间并配置好 IP 地址，而将另一端关联至网桥 br-00：

```
# ip link set veth0XX netns nsX
# ip netns exec nsX ip addr add 10.0.20.1/24 dev veth0XX
# ip netns exec nsX ip link set veth0XX up

# ip link set veth0YY netns nsY
# ip netns exec nsY ip addr add 10.0.20.2/24 dev veth0YY
# ip netns exec nsY ip link set veth0YY up

# ip link set veth0ZZ netns nsZ
# ip netns exec nsZ ip addr add 10.0.20.3/24 dev veth0ZZ
# ip netns exec nsZ ip link set veth0ZZ up

# ip link set veth1XX up
# brctl addif br-00 veth1XX
# ip link set veth1YY up
# brctl addif br-00 veth1YY
# ip link set veth1ZZ up
# brctl addif br-00 veth1ZZ
# brctl show
bridge name bridge id      STP enabled interfaces
br-00      8000.0ede363d74f0 no        veth1XX
                             veth1YY
                             veth1ZZ
```

测试连通性：

```
# ip netns exec nsX ping 10.0.20.2
# ip netns exec nsY ping 10.0.20.3
# ip netns exec nsZ ping 10.0.20.1
```

1.3.3 自定义名称空间与外部通信

给桥接接口 br-00 配置同一网段的地址，即可实现默认网络名称空间与三个自定义名称空间中接口地址的通信。

```
# ip addr add 10.0.20.254/24 dev br-00
```

此时，在主机上启动 nat 机制，把来自于 10.0.20.0/24 网络中的报文地址转换至某物理接口的地址，即可实现自定义网络空间中的地址与外部地址通信。这里以 nsX 网络名称空间为例。

先去为名称空间 nsX 指定默认路由。

```
# ip netns exec nsX ip route add default via 10.0.20.254
# ip netns exec nsX ip route show
```

10.0.20.0/24 dev veth0XX proto kernel scope link src 10.0.20.1
default via 10.0.20.254 dev veth0XX

在主机上基于 iptables 配置 nat 规则（本机其中一块物理网卡的地址为 172.16.100.7/16）：
iptables -t nat -A POSTROUTING -s 10.0.20.0/24 -j SNAT --to-source 172.16.100.7

在本机外部有一个可以通信的主机 172.16.0.1，下面通过 nsX 进行连通性测试：
ip netns exec nsX ping -c 2 172.16.0.1
PING 172.16.0.1 (172.16.0.1) 56(84) bytes of data.
64 bytes from 172.16.0.1: icmp_seq=1 ttl=63 time=3.16 ms
64 bytes from 172.16.0.1: icmp_seq=2 ttl=63 time=1.80 ms

--- 172.16.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 1.806/2.485/3.164/0.679 ms

附录 2：OpenVSwitch

2.1 openvswitch

OpenvSwitch，简写为 OVS，是一款虚拟交换软件，基于 C 语言开发，其主要用于虚拟化环境以软件的方式构建虚拟交换机，支持 Xen/XenServer, KVM, and VirtualBox 等多种虚拟化技术。功能及特性如下面的描述信息。

Standard 802.1Q VLAN model with trunk and access ports
NIC bonding with or without LACP on upstream switch
NetFlow, sFlow(R), and mirroring for increased visibility
QoS (Quality of Service) configuration, plus policing
GRE, GRE over IPSEC, VXLAN, and LISP tunneling
802.1ag connectivity fault management
OpenFlow 1.0 plus numerous extensions
Transactional configuration database with C and Python bindings
High-performance forwarding using a Linux kernel module

OpenvSwitch 有如下几个组件共同组成：
ovs-vswitchd：OVS 守护进程，实现交换功能，和 Linux 内核兼容模块一起，实现基于流的交换 flow-based switching。
ovsdb-server：轻量级的数据库服务，主要保存了整个 OVS 的配置信息，例如接口、交换内容和 VLAN 等等，而且 ovs-vswitchd 会根据数据库中的配置信息完成数据交换等工作。
ovs-dpctl：用来配置交换机内核模块的工具程序，可以控制转发规则。
ovs-vsctl：主要是获取或者更改 ovs-vswitchd 的配置信息，其悠操作会更新 ovsdb-server 中的数据库。
ovs-appctl：用于向 OVS 守护进程发送命令。

ovsdbmonitor：GUI 工具来显示 ovsdb-server 中数据信息。
ovs-controller：一个简单的 OpenFlow 控制器。
ovs-ofctl：用来控制 OVS 作为 OpenFlow 交换机工作时候的流表内容。
ovs-pki：用于为 OpenFlow 创建和管理 PKI 的命令行工具。

2.2 安装启用 openvswitch

2.3 编辑文件的方式启动 openvswitch

编辑 /etc/sysconfig/network-scripts/ifcfg-br0 文件，如下：

```
DEVICE=br0
ONBOOT=yes
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPADDR=192.168.1.15
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
HOTPLUG=no
```

编辑 /etc/sysconfig/network-scripts/ifcfg-eth0 文件，如下：

```
DEVICE=eth0
ONBOOT=yes
DEVICETYPE=ovs
TYPE=OVSPort
OVS_BRIDGE=br0
BOOTPROTO=none
HOTPLUG=no
```

2.4 基本操作

添加桥设备：

```
# ovs-vsctl add-br br0
# ovs-vsctl list-br
br0
# ovs-vsctl show
f81c34ac-cc76-4e77-ae5e-34a75a576b12
```

```
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
    ovs_version: "1.11.0"
```

向桥设备关联 port：

```
# ovs-vsctl add-port br0 eth1
# ovs-vsctl show
f81c34ac-cc76-4e77-ae5e-34a75a576b12
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
        Port "eth1"
            Interface "eth1"
    ovs_version: "1.11.0"
```

列出指定桥上的所有 port：

```
# ovs-vsctl list-ports br0
eth1
```

列出指定桥上的所有网络接口：

```
# ovs-vsctl list-ifaces br0
eth1
```

使用“ovs-ofctl”命令查看指定交换机的信息：

```
# ovs-ofctl show br0
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000c29fce3d8
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST
SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 1(eth1): addr:00:0c:29:fc:e3:d8
     config:     0
     state:      0
     current:    1GB-FD COPPER AUTO_NEG
     advertised: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG
     supported:  10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG
     speed: 1000 Mbps now, 1000 Mbps max
 LOCAL(br0): addr:00:0c:29:fc:e3:d8
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0

本示例中，eth1 接口接入的物理网络中有一台主机的 IP 为 192.168.0.254；下面通过 Network NameSpace 的方式验正本地名称空间中的 IP 可通过此桥与外部主机通信。

2.5 VLAN 应用示例

VLAN 全称为 Virtual Local Area Network），即虚拟局域网，它是一种将局域网设备从逻辑上划分成一个个网络，从而实现虚拟工作组的数据交换技术。。VLAN 是一个在物理网络上根据用途，工作组、应用等来逻辑划分的局域网络，是一个广播域，与用户的物理位置没有关系。实践中，VLAN 有多个划分标准，包含根据端口划分、根据 MAC 地址划分、根据网络层协议划分、根据 IP 组播划分、基于规则切分以及按用户划分的多种划分方式。现行的 VLAN 协议标准是 802.1q。

2.5.1 演示环境说明

本演示示例的思路是提供两个主机 Host1 和 Host2，各提供两个 Networing NameSpace ns10 和 ns20，各自提供 eth1 网卡实现主机间通信。

Host1:

桥设备：br0，关联的物理接口为 eth1，eth1 为 VLAN Trunk 接口，承载 10 和 20 两个 tag；

两个网络名称空间 ns1 和 ns2；

两组虚拟网络接口：

veth10 和 veth11

veth20 和 veth21

其中，veth10 添加至 br0，Vlan tag 为 10，veth11 关联至 ns1; veth20 添加至 br0，VLAN tag 为 20，veth21 关于至 ns2；因此，veth10 和 veth20 是 VLAN Access 接口；

ns1 中 veth11 地址为 192.168.0.81/24，ns2 中 veth21 接口的地址为 192.168.0.82/24；

Host2:

桥设备：br0，关联的物理接口为 eth1，eth1 为 VLAN Trunk 接口，承载 10 和 20 两个 tag；

两个网络名称空间 ns1 和 ns2；

两组虚拟网络接口：

veth10 和 veth11

veth20 和 veth21

其中，veth10 添加至 br0，Vlan tag 为 10，veth11 关联至 ns1; veth20 添加至 br0，VLAN tag 为 20，veth21 关于至 ns2；因此，veth10 和 veth20 是 VLAN Access 接口；

ns1 中 veth11 地址为 192.168.0.91/24，ns2 中 veth21 接口的地址为 192.168.0.92/24；

目标：Host1 上的 ns1 中的地址可以与 Host2 上的 ns1 中的地址通信；Host1 上的 ns2 中的地址可以 Host2 上的 ns2 中的地址通信；

2.5.2 配置过程

定义 OVS 交换机，并设置 Trunk 接口（以下操作在两个主机上分别进行）：

```
# ovs-vsctl add-br br0
# ovs-vsctl add-port br0 eth1
# ovs-vsctl set port eth1 trunks=10,20
```

注意：如果期望 eth1 接口既负责处理 trunk 内指定的 tagged 流量，也负责处理 untagged 流量，还需要为其做如下设定：

```
# ovs-vsctl set port eth1 vlan_mode=native-untagged
```

创建虚拟网络接口（以下操作在两个主机上分别进行）：

```
# ip link add veth10 type veth peer name veth11
# ip link add veth20 type veth peer name veth21
```

将相应的接口添加至网桥设备（以下操作在两个主机上分别进行）：

```
# ovs-vsctl add-port br0 veth10 tag=10
# ovs-vsctl add-port br0 veth20 tag=20
# ip link set veth10 up
# ip link set veth20 up
```

创建网络名称空间，关联相应的虚拟接口（以下操作在两个主机上分别进行）：

```
# ip netns add ns1
# ip netns add ns2
# ip link set veth11 netns ns1
# ip link set veth21 netns ns2
```

在 Host1 上执行如下命令，配置接口 IP：

```
# ip netns exec ns1 ifconfig veth11 192.168.0.81/24 up
# ip netns exec ns2 ifconfig veth21 192.168.0.82/24 up
```

在 Host2 上执行如下命令，配置接口 IP：

```
# ip netns exec ns1 ifconfig veth11 192.168.0.91/24 up
# ip netns exec ns2 ifconfig veth21 192.168.0.92/24 up
```

测试：

目标结果：Host1 上的 ns1 可以连通 Host2 上的 ns1，但无法连通 Host1 上的 ns2 以及 Host2 上的 ns2；

2.6 GRE 应用示例

GRE（Generic Routing Encapsulation，通用路由封装）协议是对某些网络层协议（如 IP 和 IPX）的数据报文进行封装，使这些被封装的数据报文能够在另一个网络层协议（如 IP）中传输。GRE 采用了 Tunnel（隧道）技术，Tunnel 是一个虚拟的点对点的连接，提供了一条通路使封装的数据报文能够在这个通路上传输，并且在一个 Tunnel 的两端分别对数据报进行封装及解封装。

一个被承载的协议的报文要想穿越 IP 网络在 Tunnel 中传输，必须要经过加封装与解封装两个过程，

2.6.1 演示环境说明

Host1:
桥设备：br0，关联的 gre 虚拟接口为 gre0；
用于 gre 隧道的网络接口 eth1，IP 地址为 10.0.10.1/24；
网络名称空间 ns0；
虚拟网络接口对儿：
veth0：关联至 br0
veth1：关联至 ns0，且在 ns0 中的地址为 192.168.22.1/24；

Host2:
桥设备：br0，关联的 gre 虚拟接口为 gre0；
用于 gre 隧道的网络接口 eth1，IP 地址为 10.0.10.2/24；
网络名称空间 ns0；
虚拟网络接口对儿：
veth0：关联至 br0
veth1：关联至 ns0，且在 ns0 中的地址为 192.168.22.2/24；

注意：由于隧道报文要经由 eth1 接口进行转发，因此，Host1 和 Host2 均应该开启路由转发功能。

2.6.2 配置过程

打开路由间转发（在两台主机上分别进行）：

编辑/etc/sysctl.conf，确定如下参数的值为其所示。
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0

让配置生效。
sysctl -p

创建虚拟网络接口（以下操作在两个主机上分别进行）：
ip link add veth0 type veth peer name veth1

将相应的接口添加至网桥设备（以下操作在两个主机上分别进行）：
ovs-vsctl add-port br0 veth0
ip link set veth0 up

创建网络名称空间，关联相应的虚拟接口（以下操作在两个主机上分别进行）：
ip netns add ns0
ip link set veth1 netns ns0

配置接口 IP：

在 Host1 上执行如下命令

```
# ip netns exec ns0 ifconfig veth1 192.168.22.1/24 up
```

在 Host2 上执行如下命令

```
# ip netns exec ns0 ifconfig veth1 192.168.22.2/24 up
```

设置 gre 隧道：

在 Host1 上执行如下命令

```
# ip addr add eth1 10.0.10.1/24
# ip link set eth1 up
# ovs-vsctl add-port br0 gre0
# ovs-vsctl set interface gre0 type=gre options:remote_ip=10.0.10.2
```

在 Host2 上执行如下命令

```
# ip addr add eth1 10.0.10.2/24
# ip link set eth1 up
# ovs-vsctl add-port br0 gre0
# ovs-vsctl set interface gre0 type=gre options:remote_ip=10.0.10.1
```

注意：各 gre 节点选项中“remote_ip”分别为对端节点的 IP。

测试目标：Host1 上的 ns0 中的地址 192.168.22.1 可与 Host2 上的 ns0 中的地址 192.168.22.2 直接通信。可使用 tcpdump 抓包查看流量。

例如，在 Host1 上的 ns0 向 Host2 上的 ns0 的 192.168.22.2 发起 ping 操作，而后在 Host2 上对 eth1 接口抓包的结果：

```
# tcpdump -i eth1 -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
16:33:46.459330 IP 10.0.10.1 > 10.0.10.2: GREv0, length 102: IP 192.168.22.1 > 192.168.22.2: ICMP echo request, id 53767, seq 64, length 64
16:33:46.459402 IP 10.0.10.2 > 10.0.10.1: GREv0, length 102: IP 192.168.22.2 > 192.168.22.1: ICMP echo reply, id 53767, seq 64, length 64
16:33:47.460510 IP 10.0.10.1 > 10.0.10.2: GREv0, length 102: IP 192.168.22.1 > 192.168.22.2: ICMP echo request, id 53767, seq 65, length 64
16:33:47.460573 IP 10.0.10.2 > 10.0.10.1: GREv0, length 102: IP 192.168.22.2 > 192.168.22.1: ICMP echo reply, id 53767, seq 65, length 64
16:33:48.462892 IP 10.0.10.1 > 10.0.10.2: GREv0, length 102: IP 192.168.22.1 > 192.168.22.2: ICMP echo request, id 53767, seq 66, length 64
16:33:48.462953 IP 10.0.10.2 > 10.0.10.1: GREv0, length 102: IP 192.168.22.2 > 192.168.22.1: ICMP echo reply, id 53767, seq 66, length 64
```

注意：启用 GRE 隧道之后，有可能需要重新设定接口的 MTU 属性。

提示：VLAN 也可以通过 GRE 进行。此外，OpenVSwitch 还支持 VXLAN（依赖较新版本的内核及 iproute 程序）。

2.7 floating IP

前面附录 2.6 中描述了基于跨物理节点（Host1 和 Host2）的两个虚拟机（以网络名称空间模块）之间通过隧道协议通信的方式，它模拟了一种虚拟局域网的实现。不过，虚拟机之间赖以通信的网桥并未关联任何能够与外部网络（包括 Internet）通信的接口，因此，其仅能用于虚拟机间的内容通信。如果某虚拟机需要与外部网络进行通信，其实现方式有将虚拟机的接口桥接至某关联到了能够与外部网络通信的物理接口上（即常见的 vmware workstation 或 virtualbox 中所谓的桥接模式的功能），以及 floating IP 机制。

简单来讲，floating IP 就是通过网络名称空间虚拟出一台路由器设备，其外部接口桥接至可通过物理接口与外部网络通信的网桥设备，而内部接口则做为内部网桥设备上关联的各虚拟机的网关接口，而后在外部网络接口上配置一个 IP 地址，并通过 DNAT 的方式转换至内部某指定的主机上，反过来，从内部某指定的主机上发出的报文则由路由器通过 SNAT 机制转发至外部接口上某特定的地址，从而实现了外部网络与内部 VM 的通信。

2.7.1 演示环境说明

此演示建立在附录 2.6 节所实现的环境的基础上，但额外需要 Host1 具有一个能与外部网络通信的物理接口 eth2。

此演示中，外部网络为 192.168.0.0/24，且有一台主机地址为 192.168.0.254。
而此前的演示中已经可以得知，虚拟机间通信的内部网络为 192.168.22.0/24，GRE 隧道网络为 10.0.10.0/24。

2.7.2 配置过程

在具有能够与外部网络通信的物理接口 eth2 的 Host1 主机上进行如下步骤：

开启 eth2 的混杂模式，且删除其上配置的任何 IP 地址：

```
# ip addr flush eth2
# ip link set eth2 up
# ip link set eth2 promisc on
```

创建关联至 eth2 的能够与外部网络通信的网桥设备，这里以 br-ex 为例：

ovs-vsctl add-br br-ex
ovs-vsctl add-port br-ex eth2
ip addr add 192.168.0.81/24 dev br-ex
ip link set br-ex up

为 br-ex 添加网络接口(port) reth0，并将为虚拟路由器 router0 的对外的通信接口：
ovs-vsctl add-port br-ex reth0
ovs-vsctl set interface reth0 type=internal

ip netns add router0
ip link set reth0 netns router0
ip netns exec router0 ip addr add 192.168.0.82/24 dev reth0
ip netns exec router0 ip link set reth0 up

为网桥 br0 添加网络接口(port) geth0，并将为虚拟路由器 router0 的对内的通信接口，且内部各虚拟机均以此地址为其网关：
ovs-vsctl add-port br0 geth0
ovs-vsctl set interface geth0 type=internal

ip link set geth0 netns router0
ip netns exec gouter0 ip addr add 192.168.22.254/24 dev geth0
ip netns exec router0 ip link set geth0 up

在虚拟路由器 router 内部的 reth0 接口上添加一个 IP 地址 192.168.0.83/24 作为 floating IP 分配给虚拟机 192.168.22.2（Host2 上的网络名称空间 ns0）使用：
ip netns exec router0 ip addr add 192.168.0.83/24 dev reth0
ip netns exec router0 iptables -t nat -A PREROUTING -d 192.168.0.83 -j DNAT --to-destination 192.168.22.2
ip netns exec router0 iptables -t nat -A POSTROUTING -s 192.168.22.2 -j SNAT --to-source 192.168.0.83

在虚拟机 192.168.22.2 所在的主机 Host2 上执行如下配置：

确保网络名称空间 ns0 的网关指定向 192.168.22.254：
ip netns exec ns0 ip route add default via 192.168.22.254

通过虚拟机 192.168.22.2 测试与外部网络接口 192.168.0.81 的通信：
ip netns exec ns0 ping 192.168.0.81

在 Host1 上对 geth0 接口抓包的结果显示如下：

```
# ip netns exec router0 tcpdump -i geth0 -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on geth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

18:17:53.032833 IP 192.168.22.2 > 192.168.0.81: ICMP echo request, id 61284, seq 55, length 64
18:17:53.032876 IP 192.168.0.81 > 192.168.22.2: ICMP echo reply, id 61284, seq 55, length 64
18:17:54.033897 IP 192.168.22.2 > 192.168.0.81: ICMP echo request, id 61284, seq 56, length 64
18:17:54.033942 IP 192.168.0.81 > 192.168.22.2: ICMP echo reply, id 61284, seq 56, length 64

而在 Host1 上对 reth0 接口抓包的结果显示如下：
ip netns exec router0 tcpdump -i reth0 -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on reth0, link-type EN10MB (Ethernet), capture size 65535 bytes
18:19:18.084453 IP 192.168.0.83 > 192.168.0.81: ICMP echo request, id 61284, seq 140, length 64
18:19:18.084479 IP 192.168.0.81 > 192.168.0.83: ICMP echo reply, id 61284, seq 140, length 64
18:19:19.085120 IP 192.168.0.83 > 192.168.0.81: ICMP echo request, id 61284, seq 141, length 64
18:19:19.085153 IP 192.168.0.81 > 192.168.0.83: ICMP echo reply, id 61284, seq 141, length 64
18:19:20.084952 IP 192.168.0.83 > 192.168.0.81: ICMP echo request, id 61284, seq 142, length 64
18:19:20.084981 IP 192.168.0.81 > 192.168.0.83: ICMP echo reply, id 61284, seq 142, length 64

附录 3：virsh 接口创建 kvm 或 qemu 虚拟实例的模板文件示例

```
<domain type='qemu' id='1'>
  <name>instance-00000003</name>
  <uuid>70af5516-b0ce-4908-823b-fdb5ee0ce22b</uuid>
  <memory unit='KiB'>262144</memory>
  <currentMemory unit='KiB'>262144</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <sysinfo type='smbios'>
    <system>
      <entry name='manufacturer'>Red Hat Inc.</entry>
      <entry name='product'>OpenStack Nova</entry>
      <entry name='version'>2014.1.2-1.el6</entry>
      <entry name='serial'>564dfd0f-a071-d1dc-307a-f7d7a700c67c</entry>
      <entry name='uuid'>70af5516-b0ce-4908-823b-fdb5ee0ce22b</entry>
    </system>
  </sysinfo>
  <os>
    <type arch='x86_64' machine='rhel6.5.0'>hvm</type>
    <boot dev='hd'/>
    <smbios mode='sysinfo'/>
  </os>
  <features>
    <acpi/>
    <apic/>
```

```
</features>
<cpu mode='host-model'>
  <model fallback='allow'/>
</cpu>
<clock offset='utc'/>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>destroy</on_crash>
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' cache='none'/>
    <source file='/var/lib/nova/instances/70af5516-b0ce-4908-823b-fdb5ee0ce22b/disk'/>
    <target dev='vda' bus='virtio'/>
    <alias name='virtio-disk0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
  </disk>
  <controller type='usb' index='0'>
    <alias name='usb0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x2'/>
  </controller>
  <interface type='bridge'>
    <mac address='fa:16:3e:a9:d4:6e'/>
    <source bridge='qbr73716d82-cb'/>
    <target dev='tap73716d82-cb'/>
    <model type='virtio'/>
    <driver name='qemu'/>
    <alias name='net0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  <serial type='file'>
    <source path='/var/lib/nova/instances/70af5516-b0ce-4908-823b-fdb5ee0ce22b/console.log'/>
    <target port='0'/>
    <alias name='serial0'/>
  </serial>
  <serial type='pty'>
    <source path='/dev/pts/0'/>
    <target port='1'/>
    <alias name='serial1'/>
  </serial>
  <console type='file'>
    <source path='/var/lib/nova/instances/70af5516-b0ce-4908-823b-fdb5ee0ce22b/console.log'/>
    <target type='serial' port='0'/>
    <alias name='serial0'/>
```

```
    </console>
    <input type='tablet' bus='usb'>
      <alias name='input0'/>
    </input>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='5900' autoport='yes' listen='0.0.0.0' keymap='en-us'>
      <listen type='address' address='0.0.0.0'/>
    </graphics>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <alias name='video0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <memballoon model='virtio'>
      <alias name='balloon0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </memballoon>
  </devices>
  <seclabel type='none'/>
</domain>
```