Table of Contents

云计算折腾笔记

云计算平台研究、工具、解决方案、及其他。

版本历史

- 2015年9月22日 - 版本0.1

部署**CloudStack**

这里记录的是如何在基于CentOS7的虚拟机上快速部署CloudStack.

- 准备包
- 安装CloudStack

准备包

两个用于部署的仓库，其repo配置文件如下:

```
# cat mrepo7.repo
[0-base]
name=CentOS-local-base
baseurl=http://192.168.0.79/mrepo/centos7-x86_64/RPMS.os/
gpgcheck=0

[0-updates]
name=CentOS-local-updates
baseurl=http://192.168.0.79/mrepo/centos7-x86_64/RPMS.updates/
gpgcheck=0

[0-contrib]
name=CentOS-local-contrib
baseurl=http://192.168.0.79/mrepo/centos7-x86_64/RPMS.contrib/
gpgcheck=0

[0-extras]
name=CentOS-local-extras
baseurl=http://192.168.0.79/mrepo/centos7-x86_64/RPMS.extras/
gpgcheck=0

[0-fasttrack]
name=CentOS-local-fasttrack
baseurl=http://192.168.0.79/mrepo/centos7-x86_64/RPMS.fasttrack/
gpgcheck=0

[0-centosplus]
name=CentOS-local-centosplus
baseurl=http://192.168.0.79/mrepo/centos7-x86_64/RPMS.centosplus/
gpgcheck=0

[0-repo]
name=epel-7
baseurl=http://192.168.0.79/epelRepo/7/epel/
gpgcheck=0
```

```
# cat cloudstack7.repo
[cloudstack]
name=cloudstack
baseurl=http://192.168.0.79/4.5CentOS7
enabled=1
gpgcheck=0
```

安装CloudStack

基于Packer编译出来的镜像文件我们可以在其上构建出CloudStack运行环境，以下是详细步骤。

节点/网络拓扑

节点:

- Management节点: 192.168.139.2, CentOS6, 2-Core/3G Mem.
- Agent节点: 192.168.139.3, CentOS7, 4-Core/8G Mem.
- Gateway: 192.168.139.79

Gateway

Gateway本身是192.168.1.0/24网段的机器，需要添加一个192.168.139.79的地址:

```
# vim /etc/sysconfig/network-scripts/ifcfg-cloudbr0
    IPADDR=192.168.1.79
    NETMASK=255.255.255.0
    IPADDR1=192.168.0.79
    + IPADDR2=192.168.139.79
    NETMASK1=255.255.255.0
```

同样在Gateway端，需要添加到主机的路由，以使得外部机器都能顺利访问到192.168.139.2和192.168.139.3两台机器。

添加iptables:

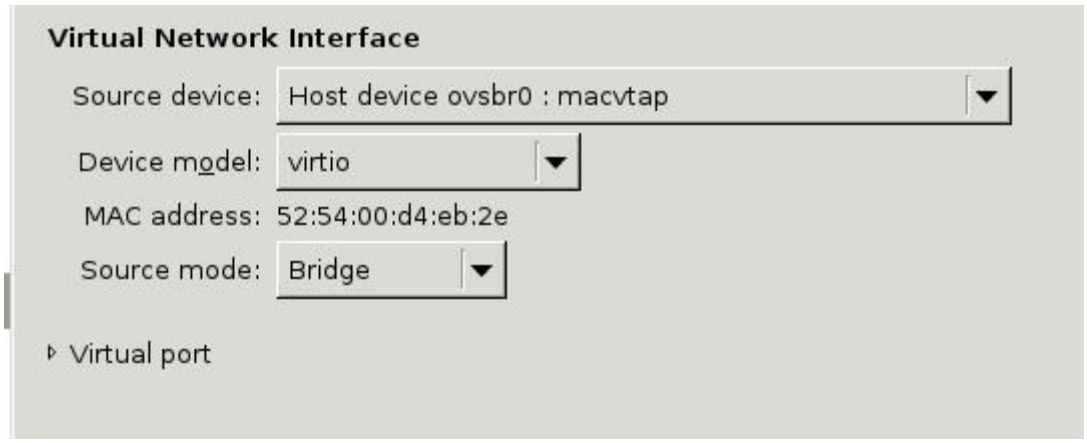
```
# iptables -A FORWARD -s 192.168.139.2 -j ACCEPT
# iptables -A FORWARD -s 192.168.139.3 -j ACCEPT
# iptables -t nat -A POSTROUTING -s 192.168.139.2 -j SNAT --to-source \
192.168.1.79
# iptables -t nat -A POSTROUTING -s 192.168.139.3 -j SNAT --to-source \
192.168.1.79
```

将添加的iptables规则加入到开机启动文件中:

```
# iptables-save>1.txt
# cp 1.txt /etc/sysconfig/iptables
```

Management节点网络

Mangement节点的网络需要桥接到主机节点，如下图, 桥接到了主机的ovsbr0接口:



配置网络:

```
# vim /etc/sysconfig/network-scripts/ifcfg-eth0
    DEVICE="eth0"
    BOOTPROTO="static"
    IPV6INIT="yes"
    NM_CONTROLLED="yes"
    ONBOOT="yes"
    TYPE="Ethernet"
    UUID="a5de6da4-9292-498e-9efd-2139d111dcb6"
    IPADDR=192.168.139.2
    NETMASK=255.255.255.0
    DNS1=223.5.5.5
    GATEWAY=192.168.139.79
```

配置Hostname:

```
# cat /etc/hosts
192.168.139.2           csmgmt
127.0.0.1               localhost
::1     localhost       ip6-localhost   ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
# cat /etc/hostname
csmgmt
```

检查hostname：

```
[root@csmgmt ~]# hostname
csmgmt
[root@csmgmt ~]# hostname --fqdn
csmgmt
```

Managment节点安装

0. 加载本地仓库:

```
# cd /etc/yum.respo.d/
# mkdir ./back
# mv *.repo .back/
# wget http://192.168.0.79/cloudstack6.repo
# wget http://192.168.0.79/mrepo6_64.repo
# yum clean all && yum makecache
```

1. 安装和配置 NTP:

```
# yum install -y ntp
# vim /etc/ntp.conf
    driftfile /var/lib/ntp/drift

    restrict default kod nomodify notrap nopeer noquery
    restrict -6 default kod nomodify notrap nopeer noquery

    restrict 127.0.0.1
    restrict -6 ::1

    server 0.uk.pool.ntp.org iburst
    server 1.uk.pool.ntp.org iburst
    server 2.uk.pool.ntp.org iburst
    server 3.uk.pool.ntp.org iburst

    includefile /etc/ntp/crypto/pw

    keys /etc/ntp/keys

    disable monitor
# service ntpd restart
# chkconfig ntpd on
```

2. SElinux的相关配置:

安装 libselinux-python:

```
# yum install -y libselinux-python
```

配置 SELinux:

```
# vim /etc/selinux/config
SELINUX=disabled
SELINUXTYPE=targeted
```

更改玩SElinux后，最好重启机器以使得规则生效。
After configurating SELinux, you'd better restart machine to let policy take effects.

3. MySQL

安装 MySQL:

```
# yum install -y mysql-server
```

安装 MySQL python模块:

```
# yum install -y MySQL-python
```

更改MYSQL的my.cnf文件，在'[mysqld_safe]'前添加以下条目:

```
# vim /etc/my.cnf
    + # CloudStack MySQL settings
    + innodb_rollback_on_timeout=1
    + innodb_lock_wait_timeout=600
    + max_connections=700
    + log-bin=mysql-bin
    + binlog-format = 'ROW'
    + bind-address=0.0.0.0

    [mysqld_safe]
```

启动服务，并保存到开机启动:

```
# service mysqld start
# chkconfig mysqld on
```

移除 anonymous 用户:

```
# mysql
mysql>  SELECT User, Host, Password FROM mysql.user;
+------+-----------+----------+
| User | Host      | Password |
+------+-----------+----------+
| root | localhost |          |
| root | csmgmt    |          |
| root | 127.0.0.1 |          |
|      | localhost |          |
|      | csmgmt    |          |
+------+-----------+----------+
mysql> DROP USER ''@'csmgmt';
mysql> DROP USER ''@'localhost';
mysql>  SELECT User, Host, Password FROM mysql.user;
+------+-----------+----------+
| User | Host      | Password |
```

```
+------+-----------+----------+
| root | localhost |          |
| root | csmgmt    |          |
| root | 127.0.0.1 |          |
+------+-----------+----------+
3 rows in set (0.00 sec)
```

Remove the testdb:

```
mysql> select * from mysql.db;
........
mysql> DELETE FROM mysql.db WHERE Db LIKE 'test%';
Query OK, 2 rows affected (0.00 sec)

mysql> select * from mysql.db;
Empty set (0.00 sec)
mysql> DROP DATABASE test;
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
mysql> \q
Bye
```

加密MYSQL安装并更改root用户密码，可以通过以下命令完成:

```
# mysql_secure_installation
```

开启 iptables:

```
# iptables -A INPUT -p tcp -m tcp --dport 3306 -j ACCEPT
# vim /etc/sysconfig/iptables
+       -A INPUT -p tcp -m tcp --dport 3306 -j ACCEPT
        -A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
# service ntpd restart
```

4. 安装 CloudStack-Management

安装 cloudstack management packages via:

```
# yum install -y cloudstack-management
```

5. Install Cloud-Monkey

Cloud-Monkey是用来方便的配置CloudStack的组件，用下列命令安装:

```
# wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-6.repo
# yum install -y python-pip
# pip install cloudmonkey
```

6. 配置CloudStack数据库

```
# cloudstack-setup-databases cloud:engine@localhost \
--deploy-as=root:engine123 -i 192.168.139.2>>/root/cs_dbinstall.out 2>&1
```

7. 配置 Management 服务器

```
# cloudstack-setup-management >> /root/cs_mgmtinstall.out 2>&1
```

到现在你可以通过浏览器访问Management节点:

```
# firefox http://192.168.139.2:8080/client/
```

Agent网络配置

Agent节点的网络需要配置Cloudbr0桥接，以下是详细的配置。

配置主机名:

```
# vim /etc/hostname
csagent
# vim /etc/hosts
192.168.139.3           csagent
127.0.0.1               localhost
::1     localhost       ip6-localhost   ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

检查hostname配置:

```
# hostname
csagent
# hostname --fqdn
csagent
```

配置桥接和固定IP地址:

```
# cat /etc/sysconfig/network-scripts/ifcfg-cloudbr0
DEVICE=cloudbr0
TYPE=Bridge
ONBOOT=yes
NM_CONTROLLED=yes
BOOTPROTO=static
IPADDR=192.168.139.3
NETMASK=255.255.255.0
DNS1=223.5.5.5
GATEWAY=192.168.139.79
# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
TYPE=Ethernet
ONBOOT=yes
NM_CONTROLLED=yes
BOOTPROTO=none
BRIDGE=cloudbr0
```

配置完毕后重启网络:

```
# systemctl restart network.service
```

Agent节点安装

0. 仓库配置

```
# cd /etc/yum.repos.d/
# mkdir ./back
# mv *.repo ./back
# wget http://192.168.0.79/mrepo7.repo
# wget http://192.168.0.79/cloudstack7.repo
# yum clean all && yum makecache
```

1. 安装包

安装CloudStack Agent包比较简单:

```
# yum install -y cloud-agent
```

2. 配置Agent

需要更改掉以下配置中的选项后重启libvirtd:

```
# sed -i 's/#vnc_listen = "0.0.0.0"/vnc_listen = "0.0.0.0"/g' \
 /etc/libvirt/qemu.conf && sed -i 's/cgroup_ \
 controllers=["cpu"]/#cgroup_controllers=["cpu"]/g' /etc/libvirt/qemu.conf
```

```
# sed -i 's/#listen_tls = 0/listen_tls = 0/g' \
  /etc/libvirt/libvirtd.conf && sed -i 's/#listen_tcp = 1/listen_tcp = 1/g' \
  /etc/libvirt/libvirtd.conf && sed -i \
 's/#tcp_port = "16509"/tcp_port = "16509"/g' \
 /etc/libvirt/libvirtd.conf && sed -i 's/#auth_tcp = "sasl"/auth_\
 tcp = "none"/g' /etc/libvirt/libvirtd.conf && \
 sed -i 's/#mdns_adv = 1/mdns_adv = 0/g' /etc/libvirt/libvirtd.conf
# sed -i 's/#LIBVIRTD_ARGS="--listen"/LIBVIRTD_ARGS="--listen"/g' \
 /etc/sysconfig/libvirtd
# sed -i '/cgroup_controllers/d' \
  /usr/lib64/python2.7/site-packages/cloudutils/serviceConfig.py
```

重启libvirtd:

```
# service libvirtd restart
```

二级存储

这里我们配置NFS服务作为CloudStack中用到的二级存储, 并在其上引入系统虚拟机模板。

NFS服务器

在Management节点上，默认的nfs-utils已经被安装，所以我们只需要按照以下步骤配置：

```
# vim /etc/exports
/home/exports *(rw,async,no_root_squash,no_subtree_check)
# mkdir -p /home/exports
# chmod 777 -R /home/exports/
# chkconfig nfs on
# chkconfig rpcbind on
# service nfs restart
# service rpcbind restart
# iptables -D INPUT -j REJECT --reject-with icmp-host-prohibited
# vim /etc/sysconfig/iptables
-D INPUT -j REJECT --reject-with icmp-host-prohibited
```

引入系统虚拟机模板

在Management机器上做如下操作:

```
# mount -t nfs 192.168.139.2:/home/exports/ /mnt
# /usr/share/cloudstack-common/scripts/storage/secondary/cloud-install-sys-tmplt \
 -m /mnt/ -u http://192.168.0.79/systemvm64template-4.5-kvm.qcow2.bz2 -h kvm -F
```

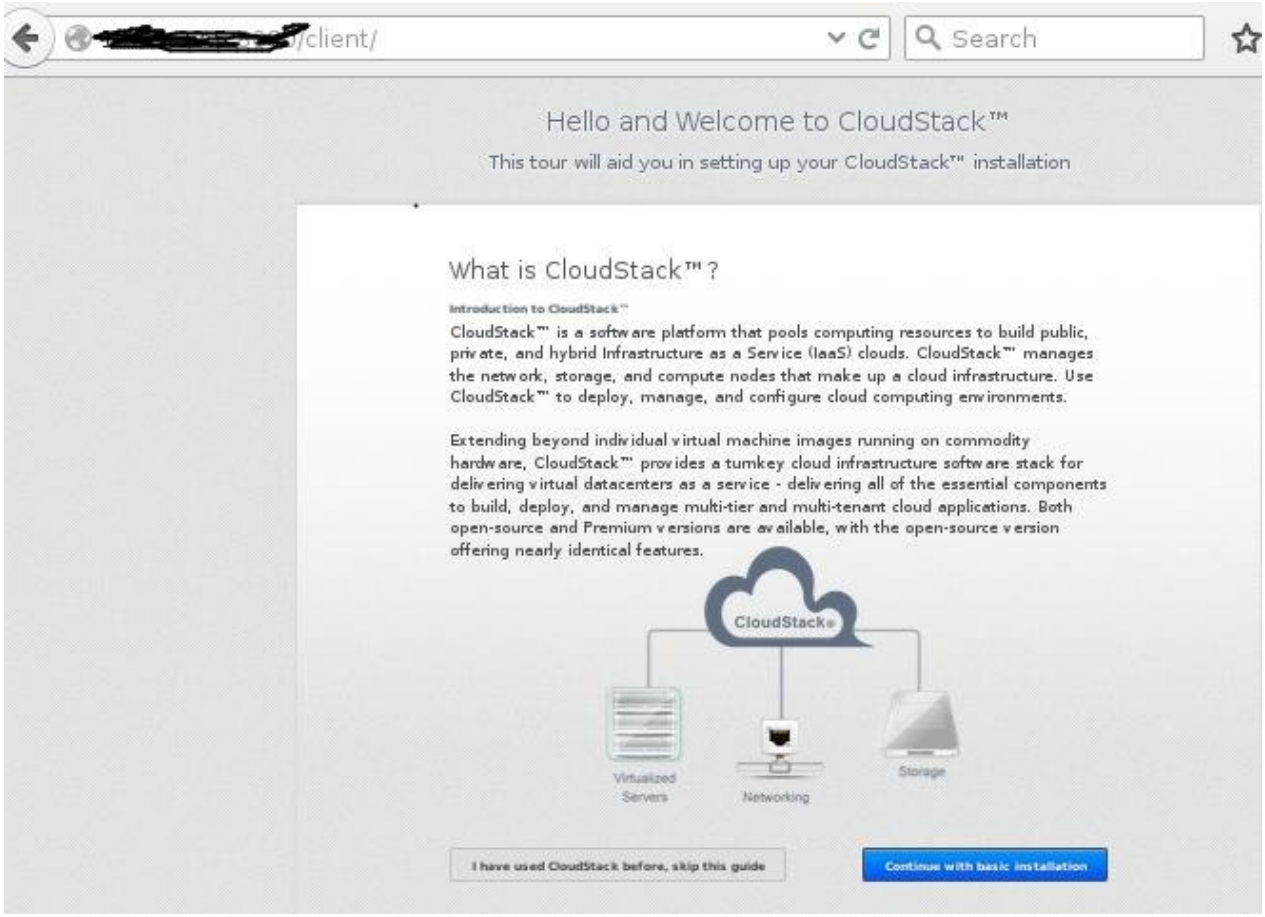
安装完模板后我们可以到/home/exports目录下检查：

```
[root@csmgmt ~]# ls /home/exports/
template
[root@csmgmt ~]# du -hs /home/exports/template/
290M    /home/exports/template/
```

配置CloudStack

安装好Management节点和Agent节点后，我们可以在页面里配置CloudStack。

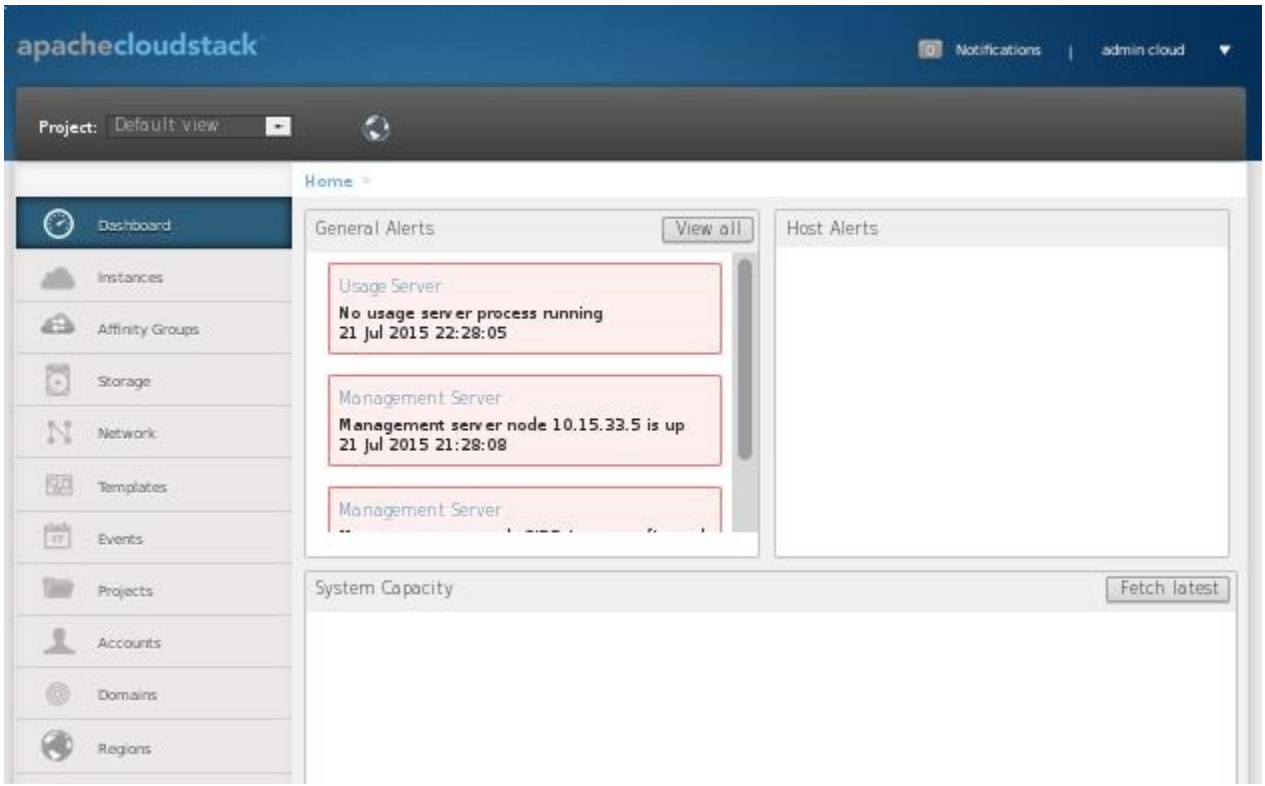
首次登录到CloudStack管理页面时，你看到如下网页:



点击 `Continue with basic installation` ,设置密码:

设置完毕后，刷新浏览器，用更改后的密码登录:

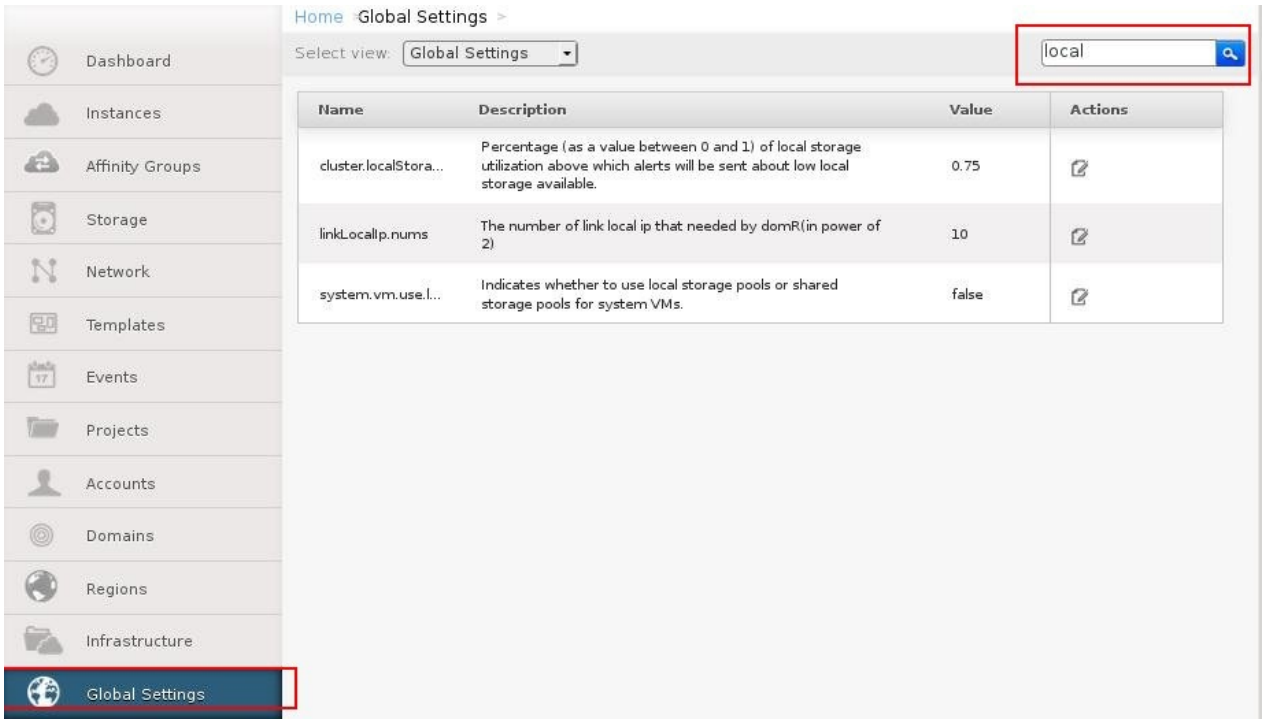
选择"I have used CloudStack before, skip this guide"按钮，到达这个页面:



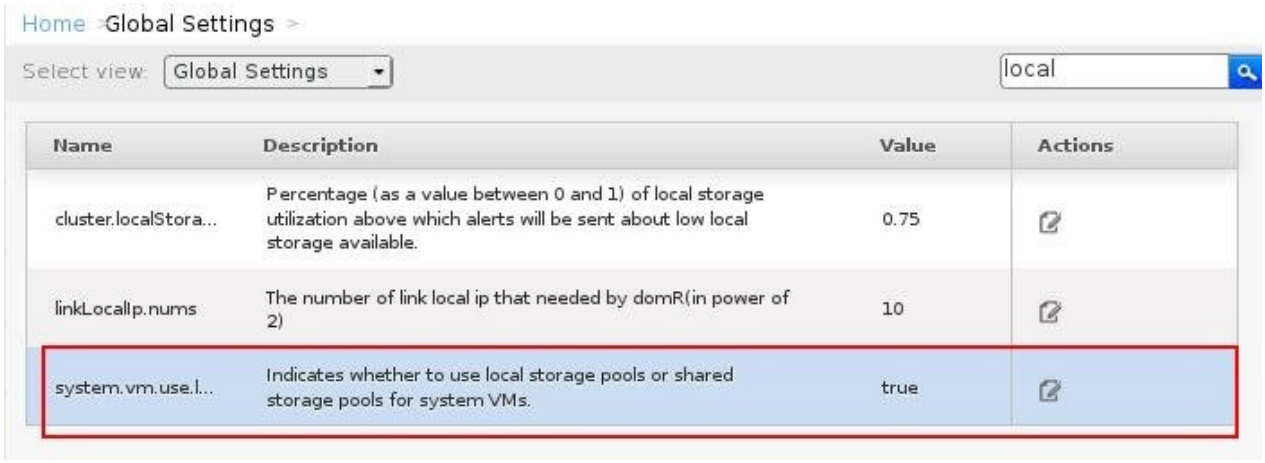
CloudStack Global Options

我们希望使用本地存储，所以通过下列设置来激活本地存储:

点击 `Global Options` 按钮， 输入 `local` ，搜索出相应结果:



更改 `system.vm.usel` 的值为 `true:

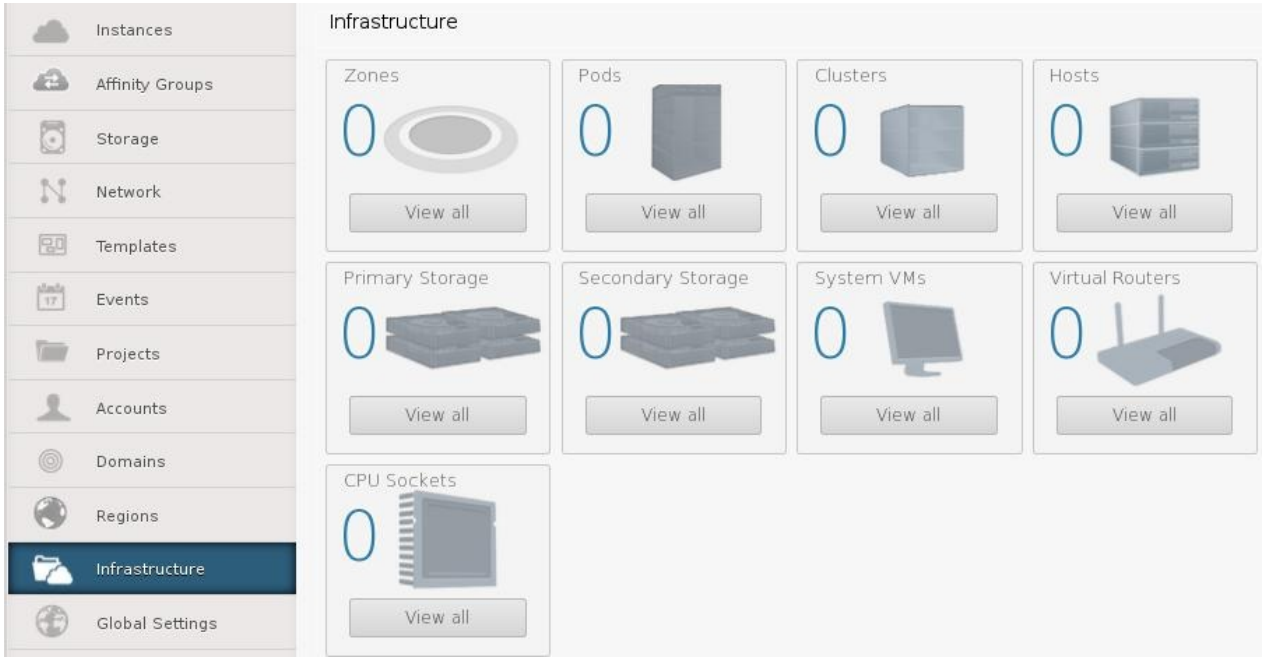


重启Cloudstack-management 服务:

```
# service cloudstack-management restart
```

Infrasturcture 配置

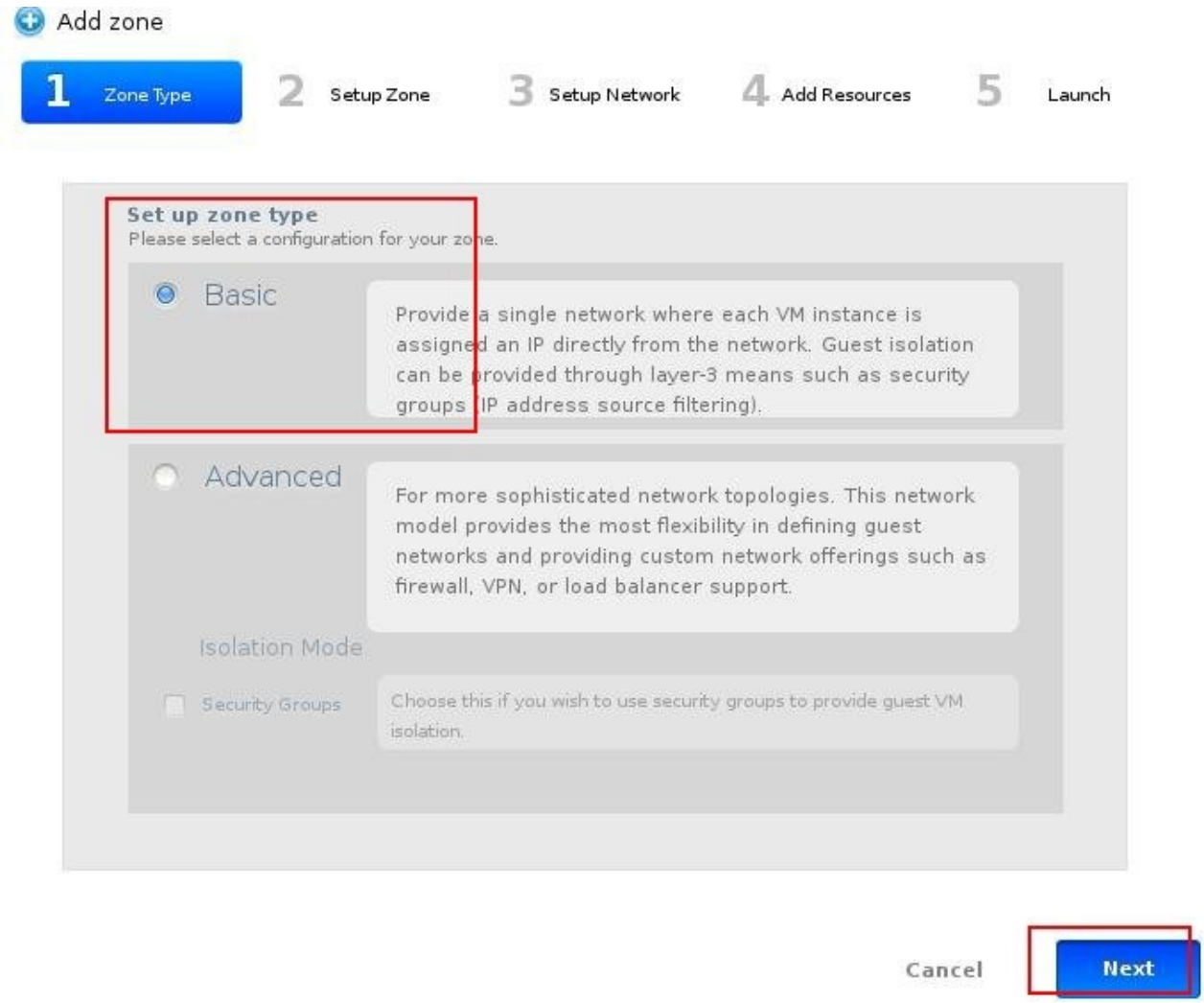
点击 `Infrasturcture`, 现在是什么也没有被配置的状态:



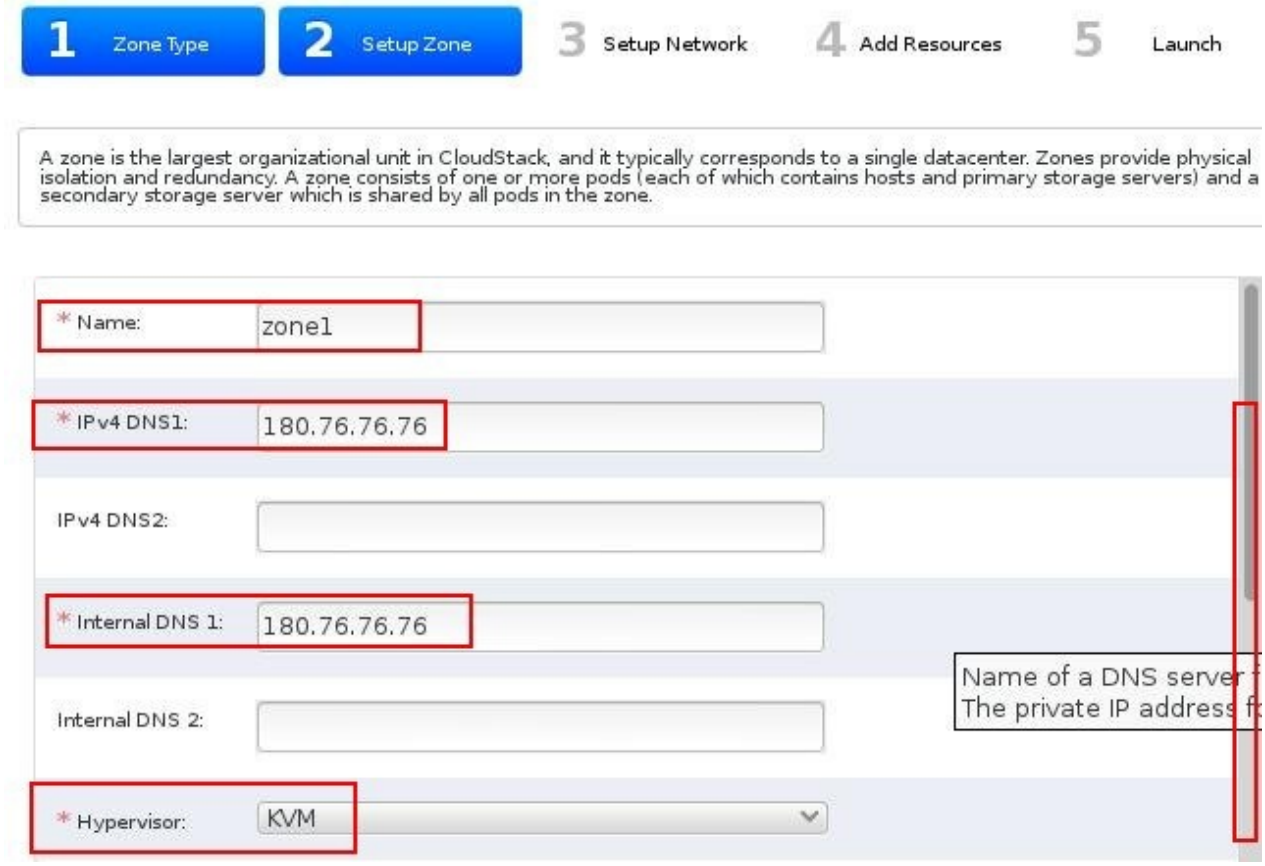
点击"Zone"下的"View All"按钮:

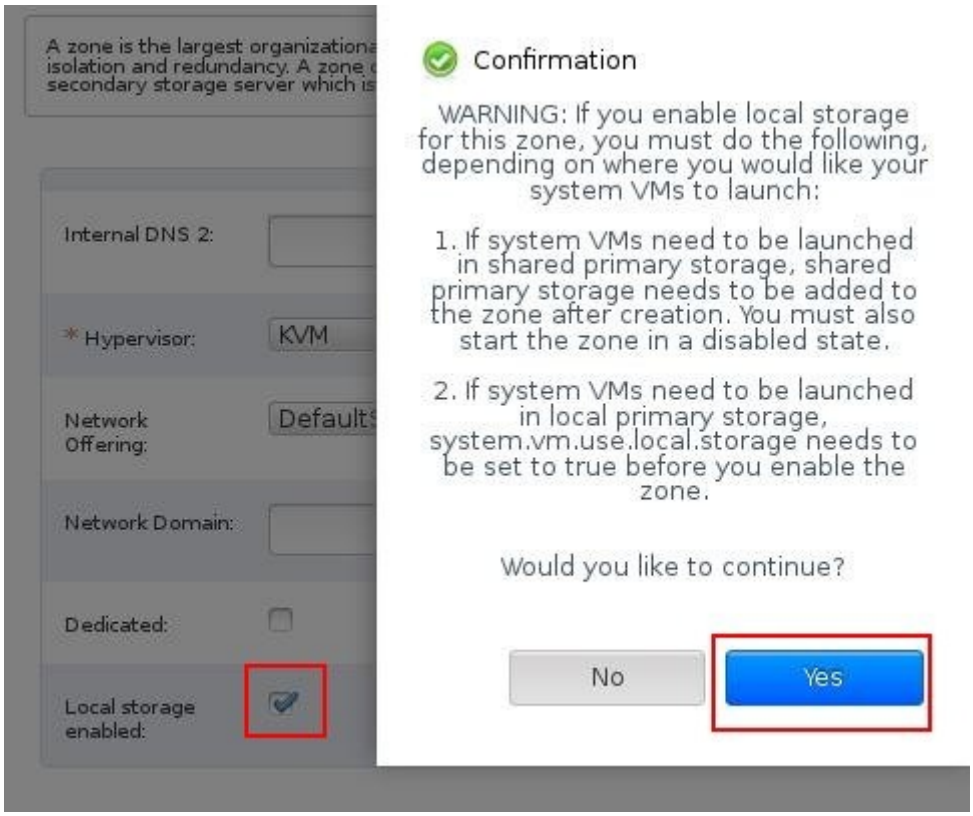
点击"Add Zone"按钮:

选择 Zone 类型:

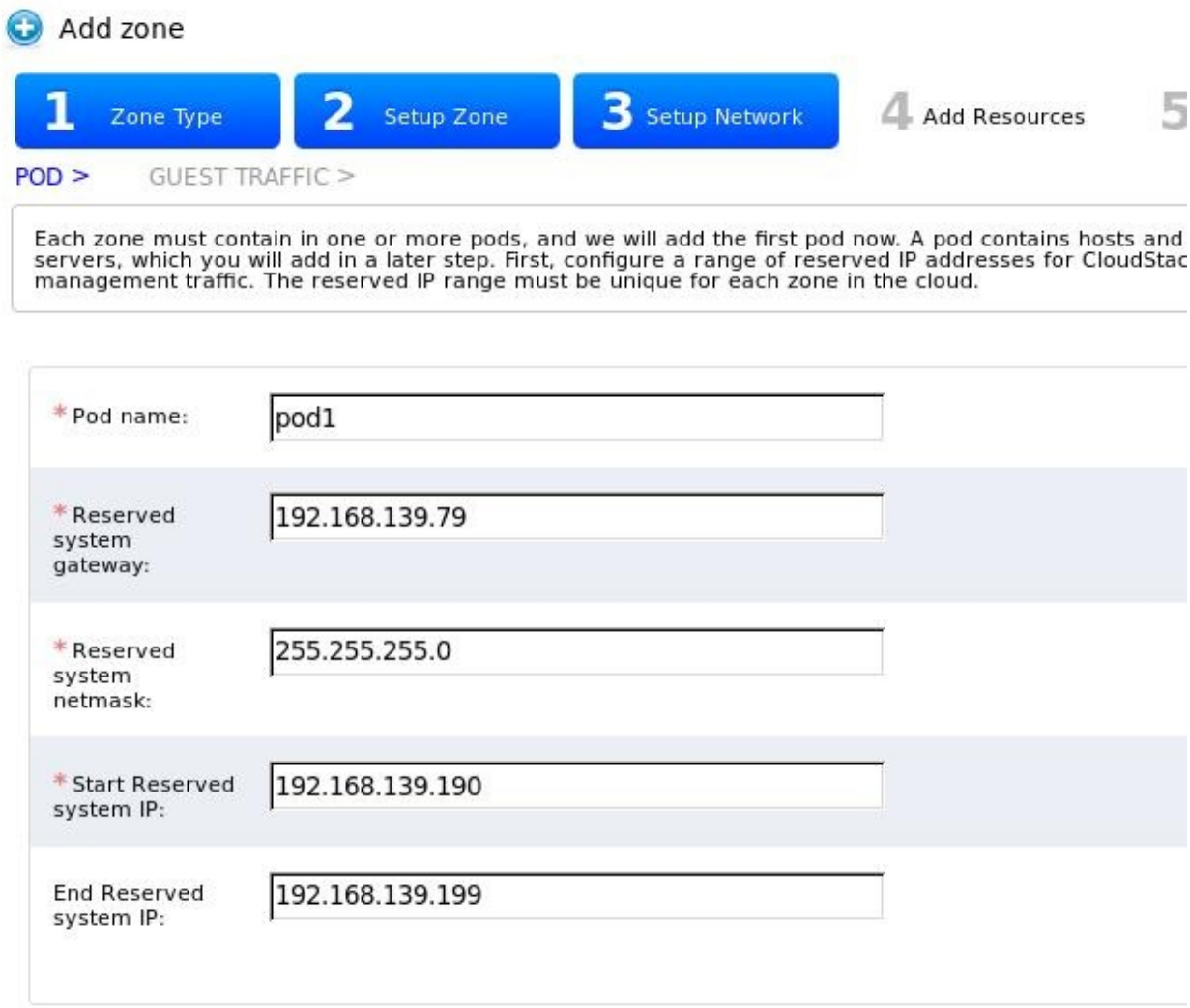

Add zone
1 Zone Type
2 Setup Zone
3 Setup Network
4 Add Resources
5 Launch
Set up zone type
Please select a configuration for your zone.
Basic
Provide a single network where each VM instance is assigned an IP directly from the network. Guest isolation can be provided through layer-3 means such as security groups (IP address source filtering).
Advanced
For more sophisticated network topologies. This network model provides the most flexibility in defining guest networks and providing custom network offerings such as firewall, VPN, or load balancer support.
Isolation Mode
Security Groups
Choose this if you wish to use security groups to provide guest VM isolation.
Cancel
Next

配置Zone信息:

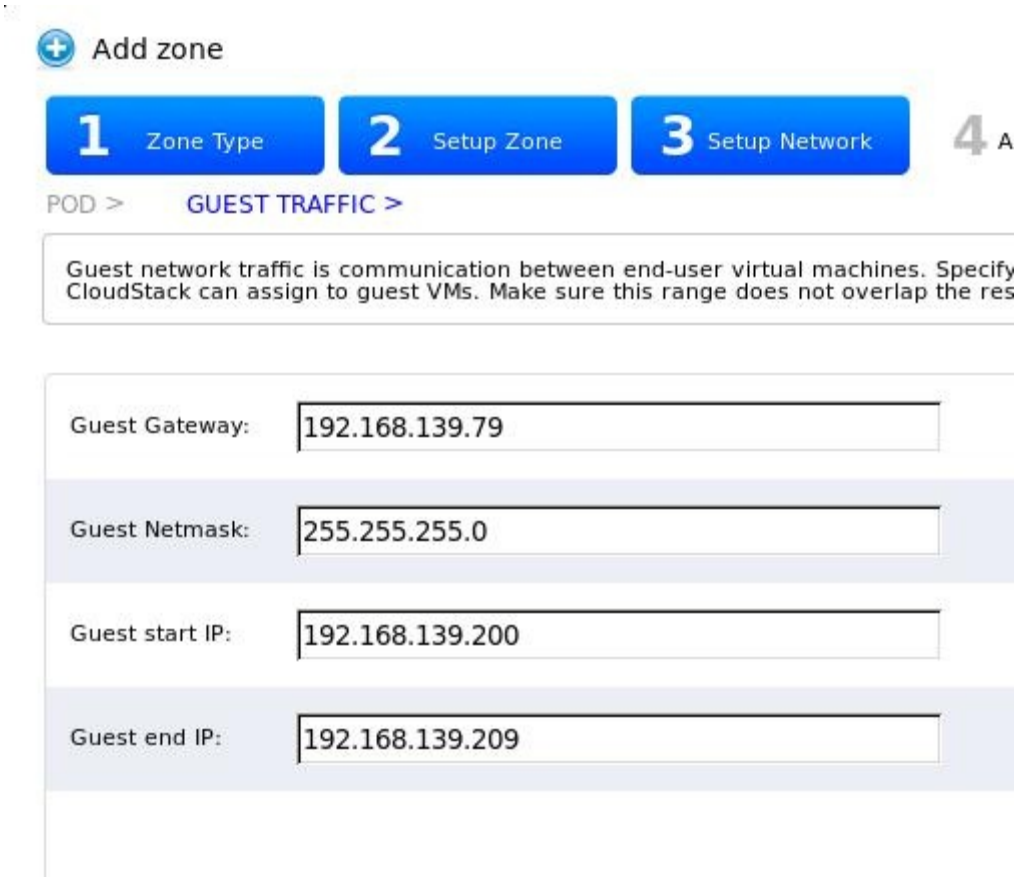




点击Next按钮，再点击一下，跳过第3步过程，进入到"4 Add Resources", 配置start/end IP范围:

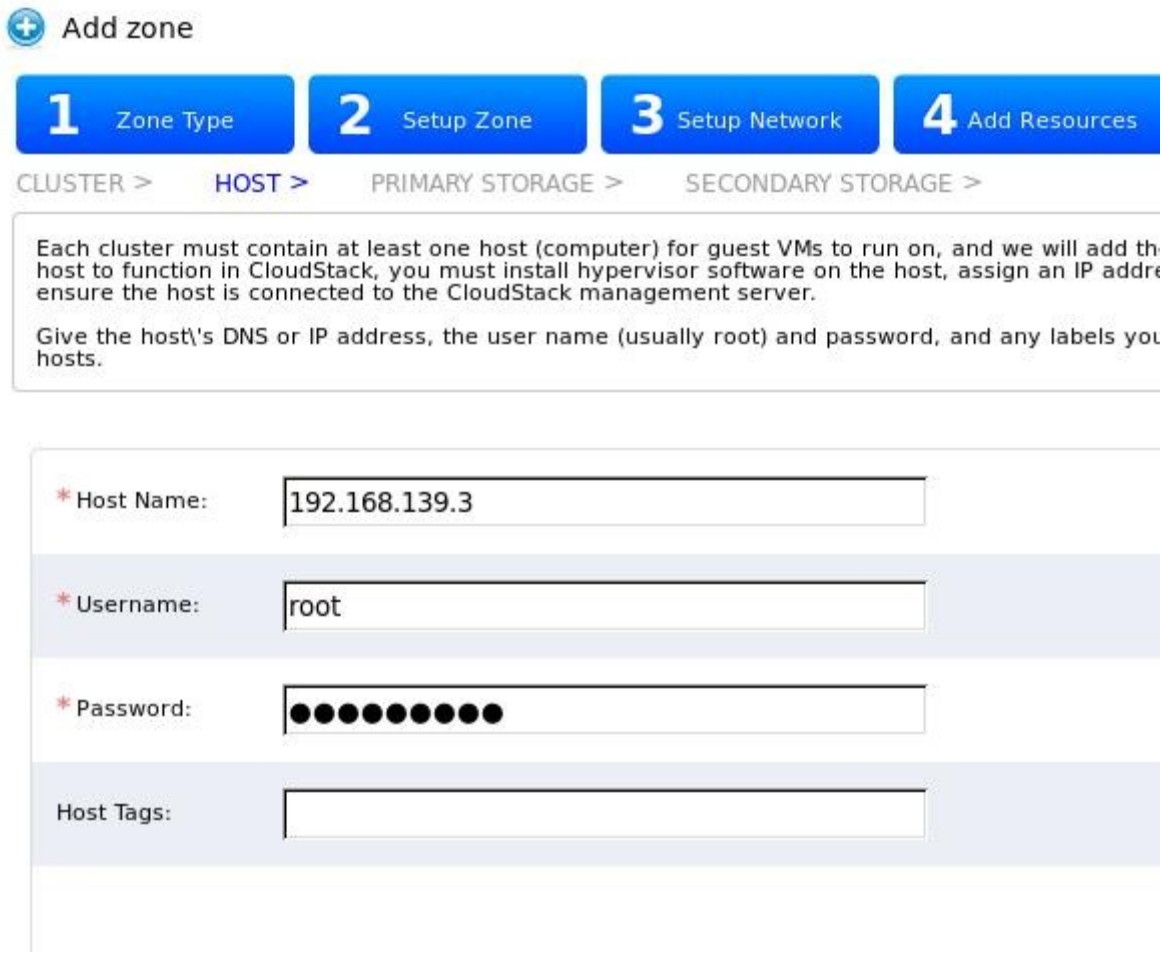


接着配置Guest Graffic:

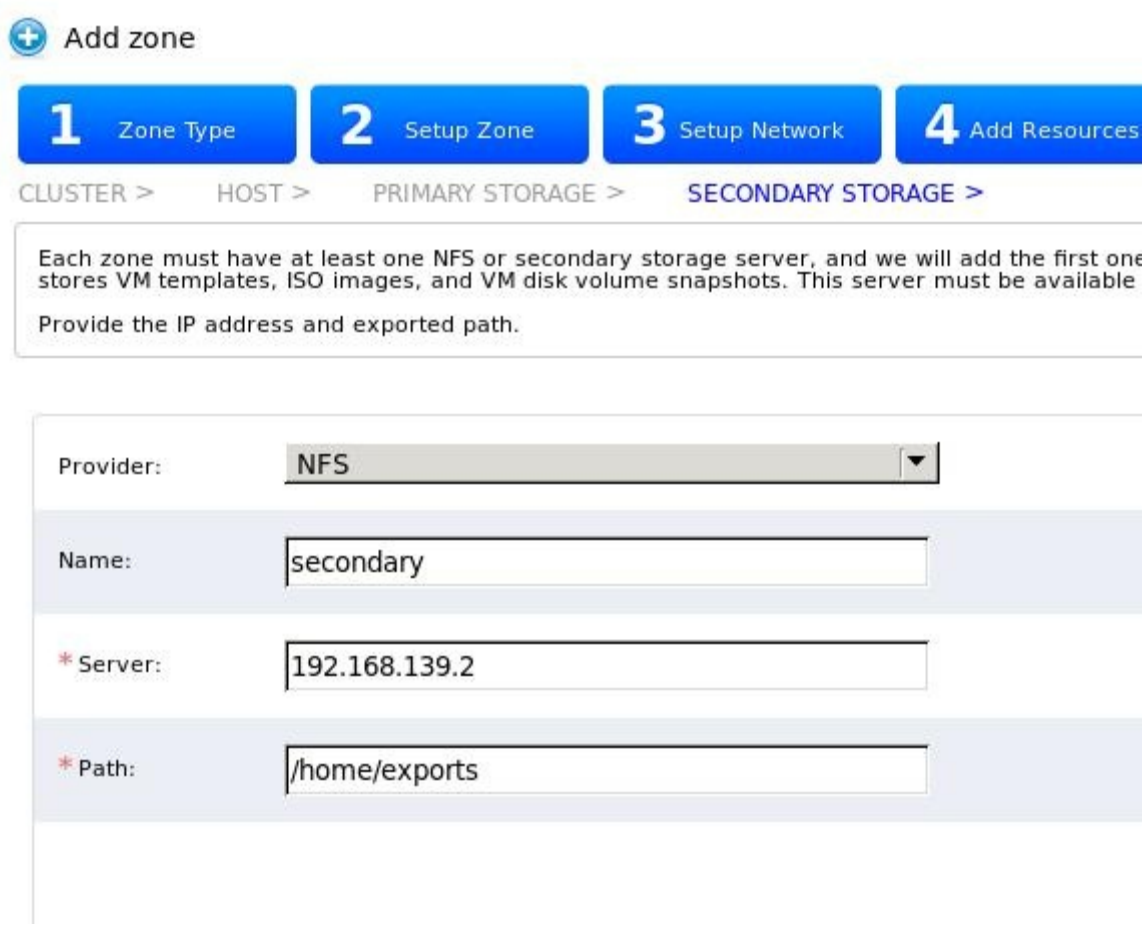


配置cluster名:

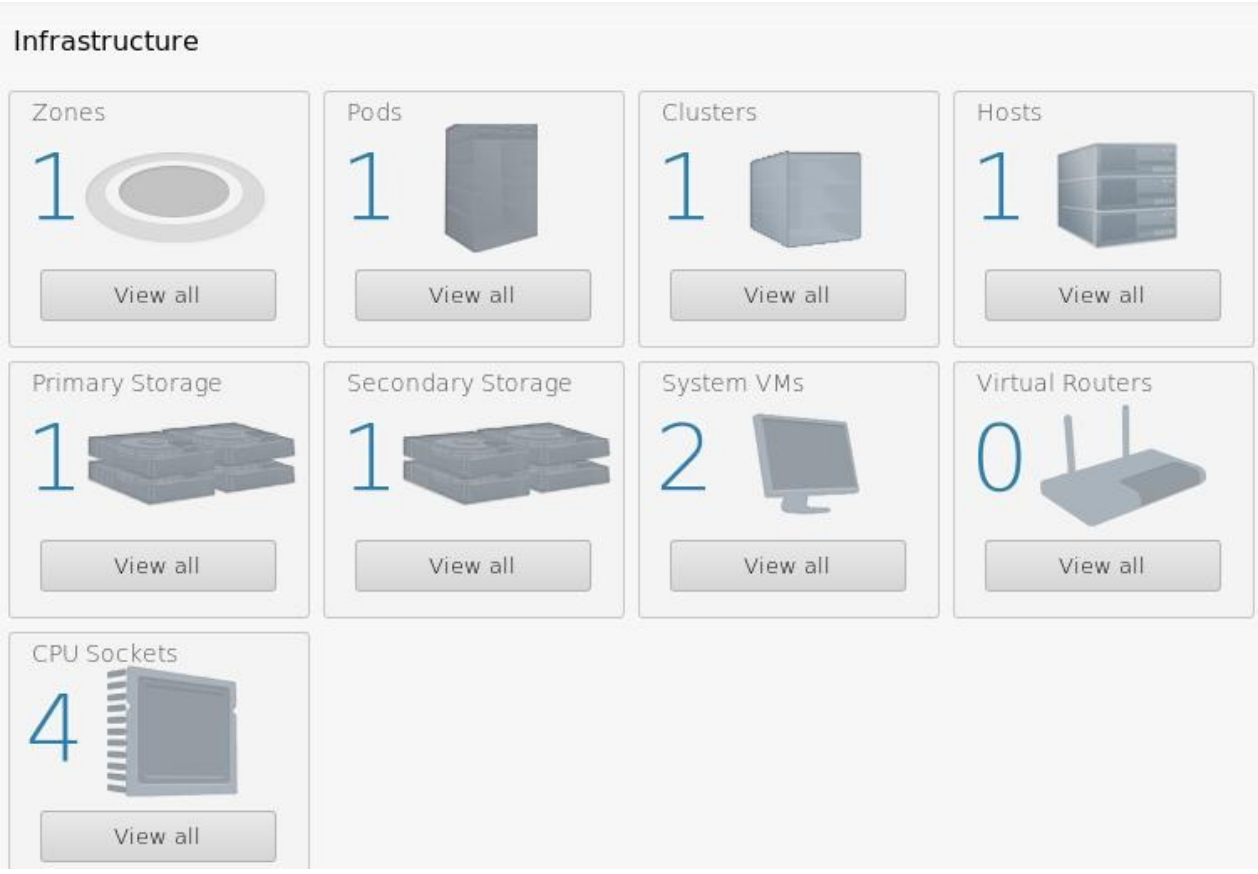
添加host:


Add zone
1 Zone Type
2 Setup Zone
3 Setup Network
4 Add Resources
CLUSTER >
HOST >
PRIMARY STORAGE >
SECONDARY STORAGE >
Each cluster must contain at least one host (computer) for guest VMs to run on, and we will add th
host to function in CloudStack, you must install hypervisor software on the host, assign an IP addre
ensure the host is connected to the CloudStack management server.
Give the host\'s DNS or IP address, the user name (usually root) and password, and any labels you
hosts.
* Host Name:
192.168.139.3
* Username:
root
* Password:
Host Tags:

添加二级存储:

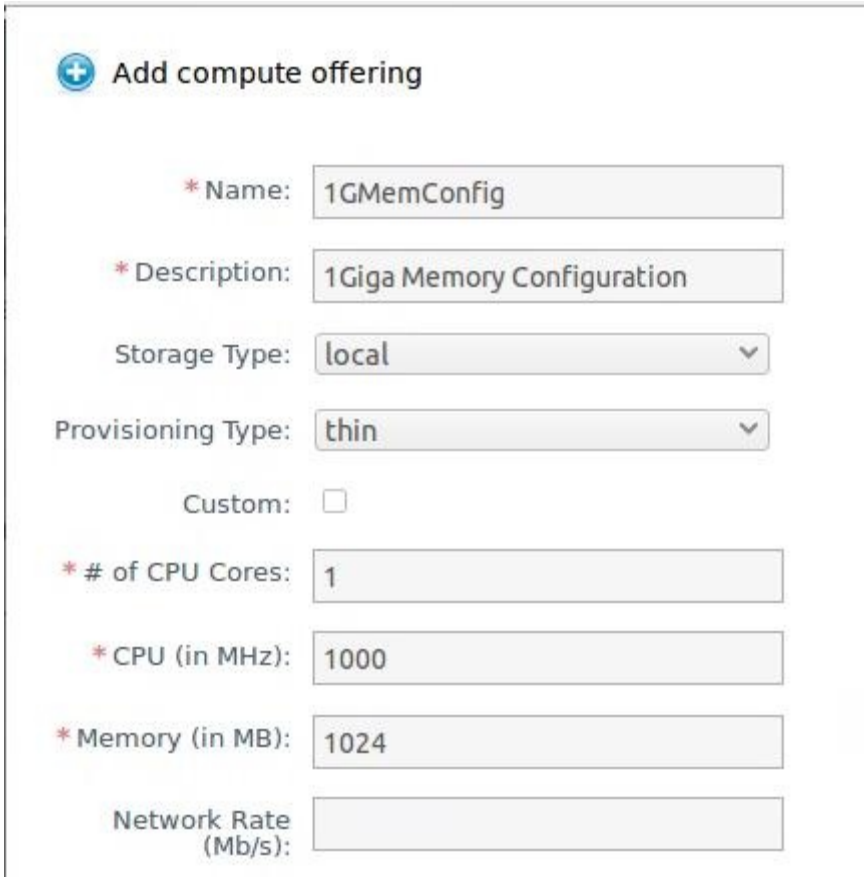


点击 Launch，你可以看到zone/pod被创建，host被添加进cluster.

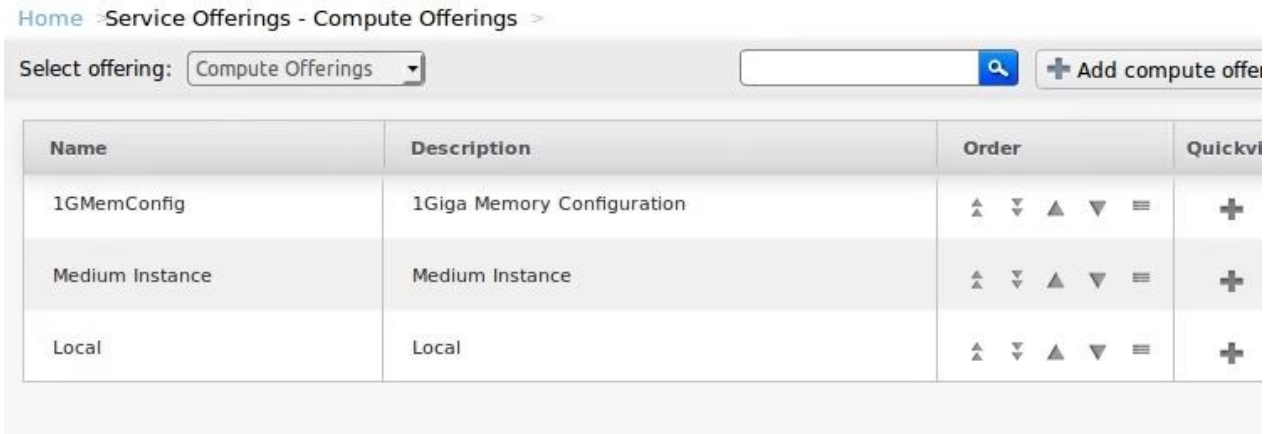


因为我们前面启用了本地存储，因而在Service Offering的选项中我们也需要提供出对应 本地存储的相关配置:

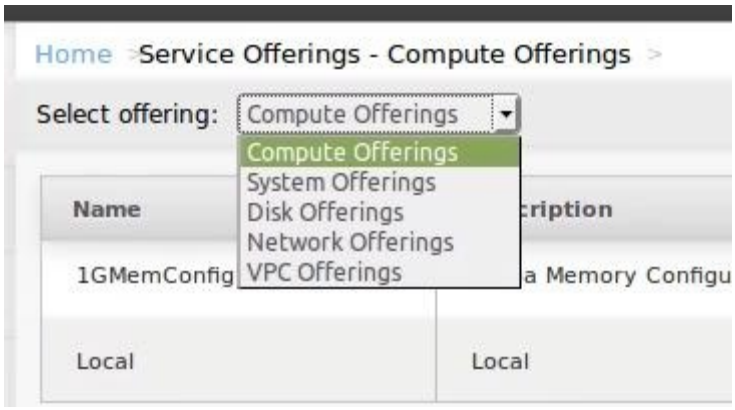
点击Service Offerings -> Add compute offering:



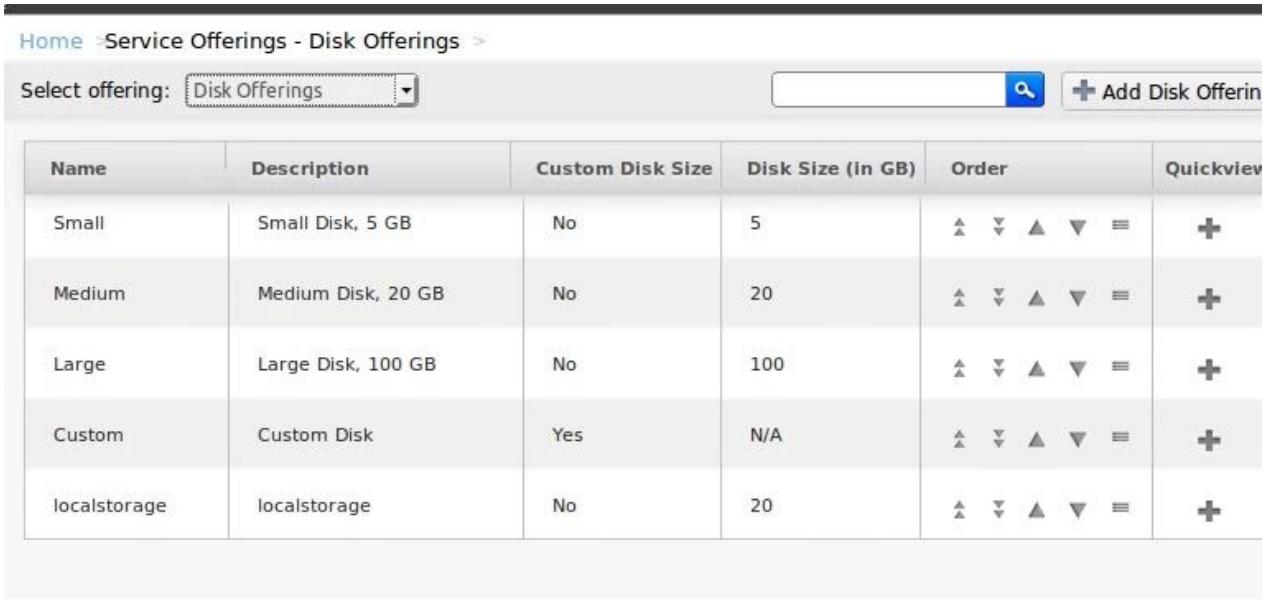
现在可以删除我们不需要的Compute Offering, 默认增加的Service Offering都是针对共 享存储的，可以直接删除掉。



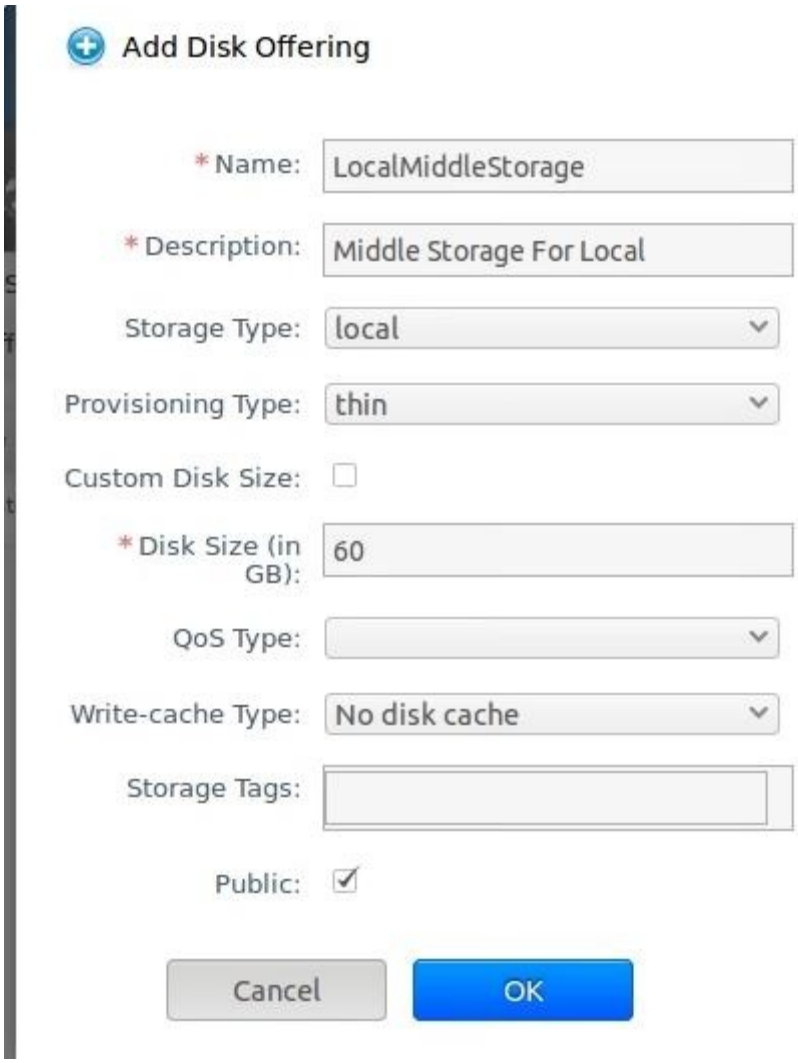
点击下拉列表里的Disk Offerings:



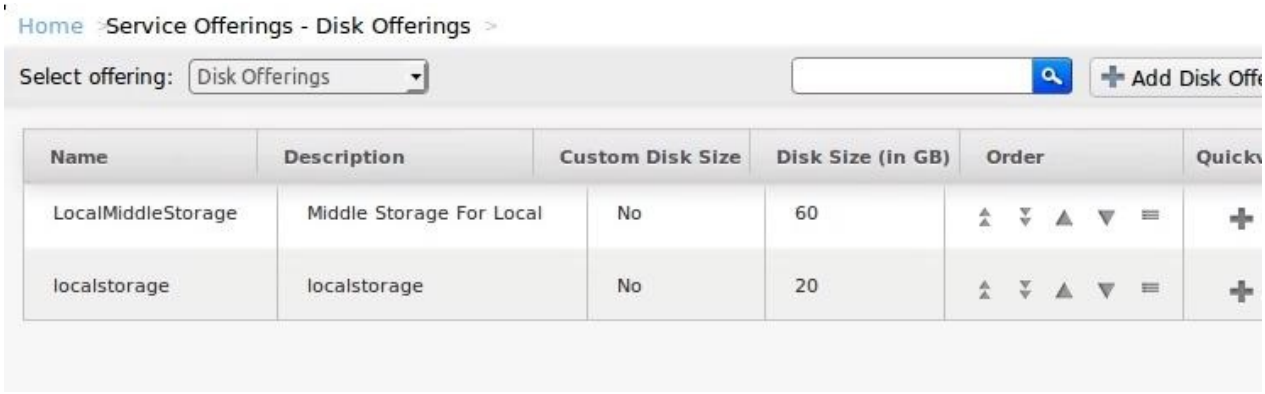
现在看到的套餐配置如下:



在这里添加针对本地存储的选项:



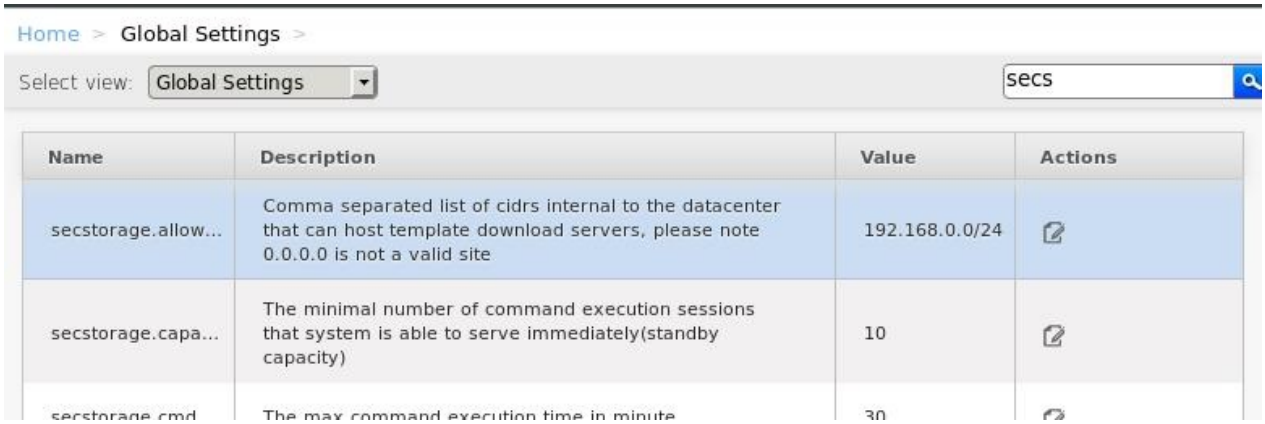
删除掉不需要的共享存储的选项后，最终的列表如下:



注册模板

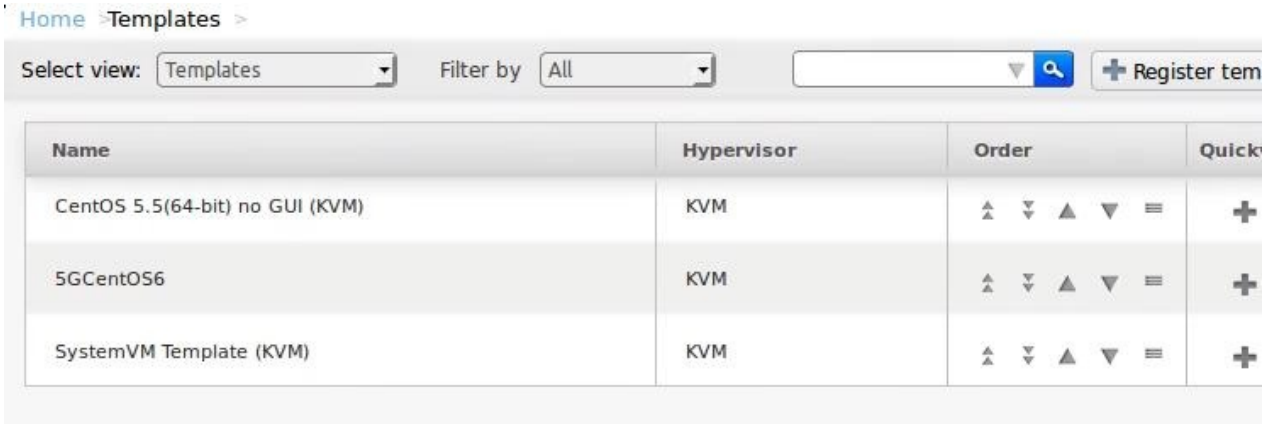
允许模板下载地址

更改模板下载允许服务器地址:



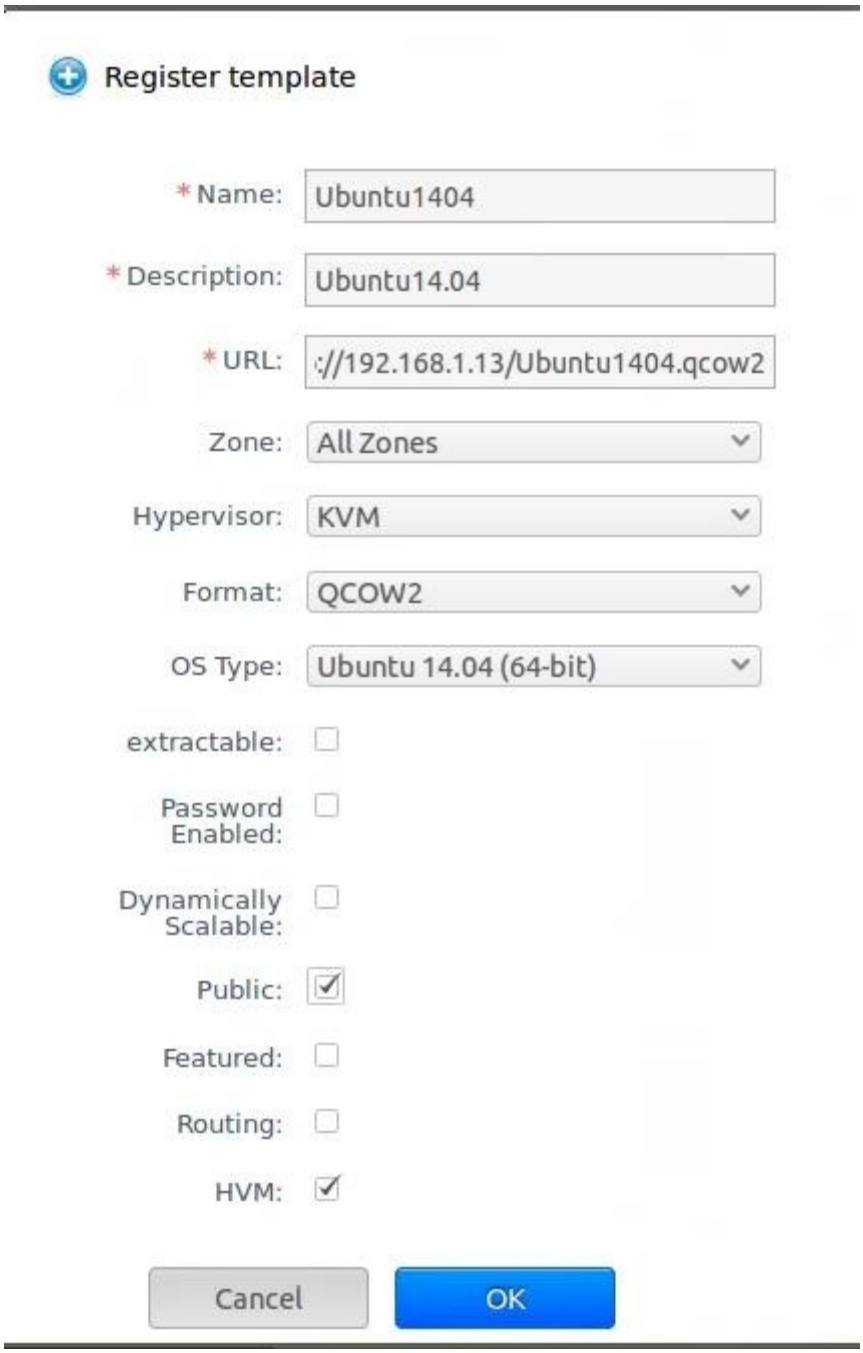
查看模板

CloudStack中已有的模板可以在CloudStack的Web页面中被查看。

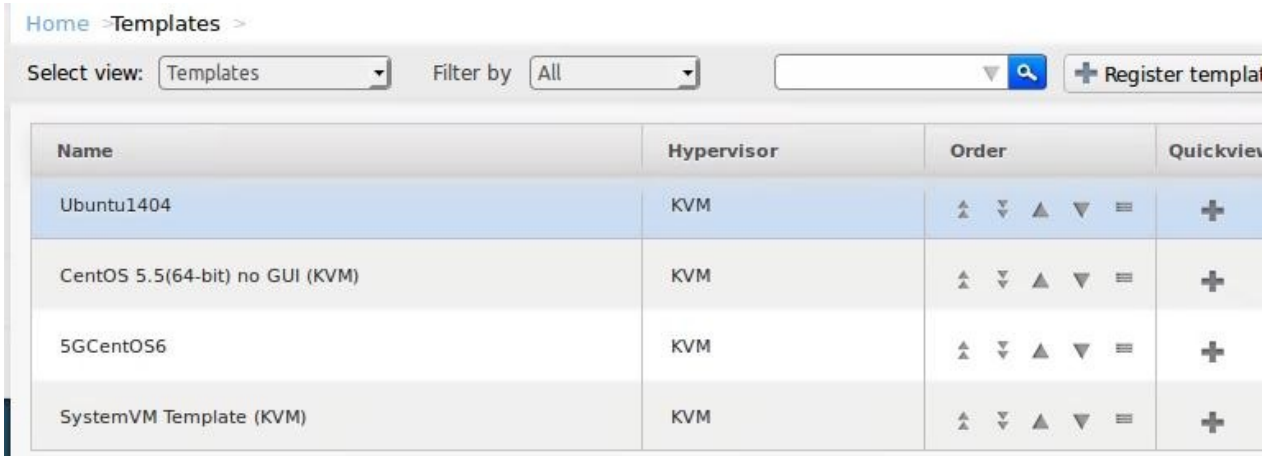


默认安装好以后，只有系统虚拟机模板是可用的，上图中的另外两个选项，"CentOS 5.5" 是安装CloudStack时自动带的，其实不可用，另一个5GCentOS6是我们手动添加的，以下 是手动添加的步骤.

首先需要准备好Web服务器，将需要导入的模板文件(一般是qcow2文件)放入Web服务器的 根目录下. 而后点击Register template按钮, 在弹出的对话框中输入以下内容:



注册完毕后的模板列表如下:

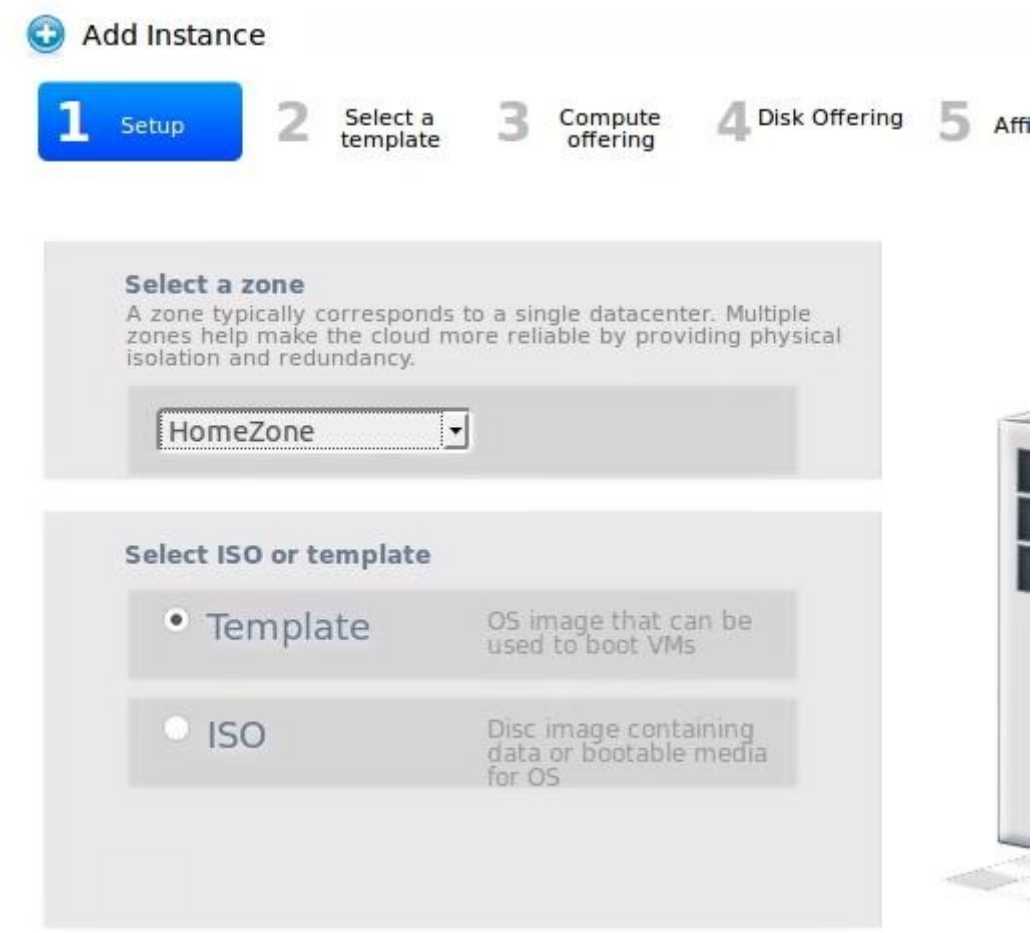


使用模板创建虚拟机

现在我们使用上面导入的Ubuntu14.04的模板用来创建一台虚拟机。

点击Instances, 在弹出的窗口中选择"Add Instance":

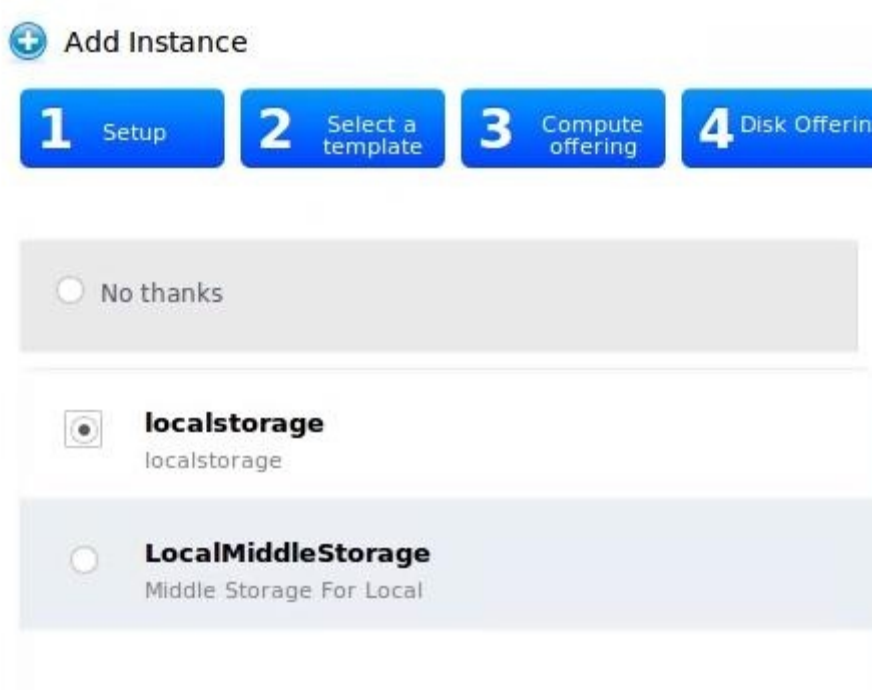
选择Zone:



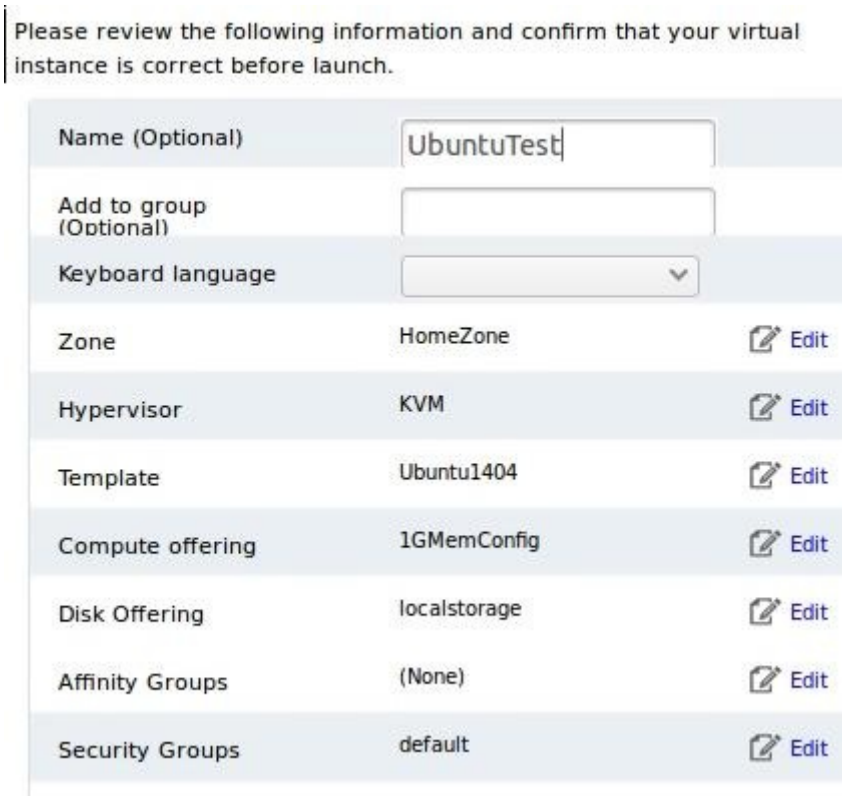
选择模板, community下的Ubuntu1404即我们需要创建虚拟机的模板:

选择内存和CPU配置:

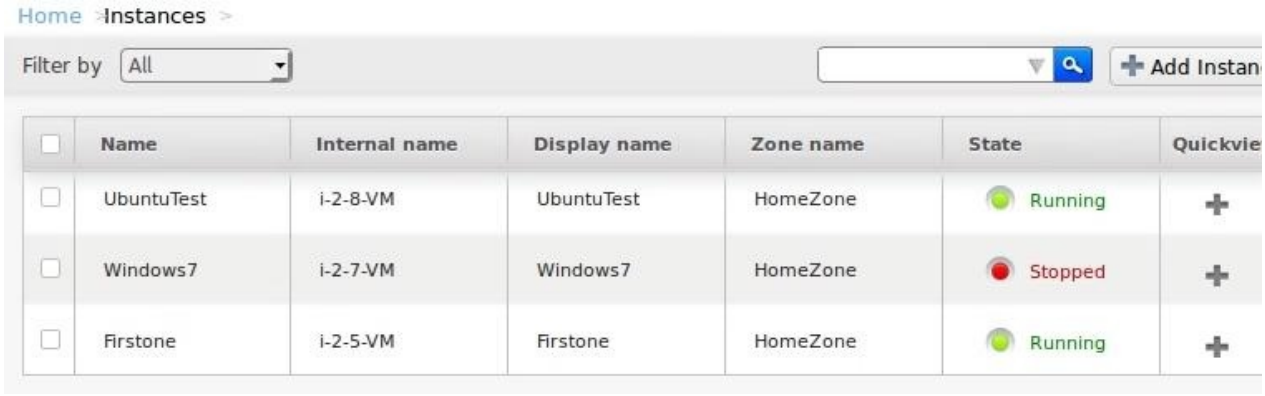
选择磁盘配置:



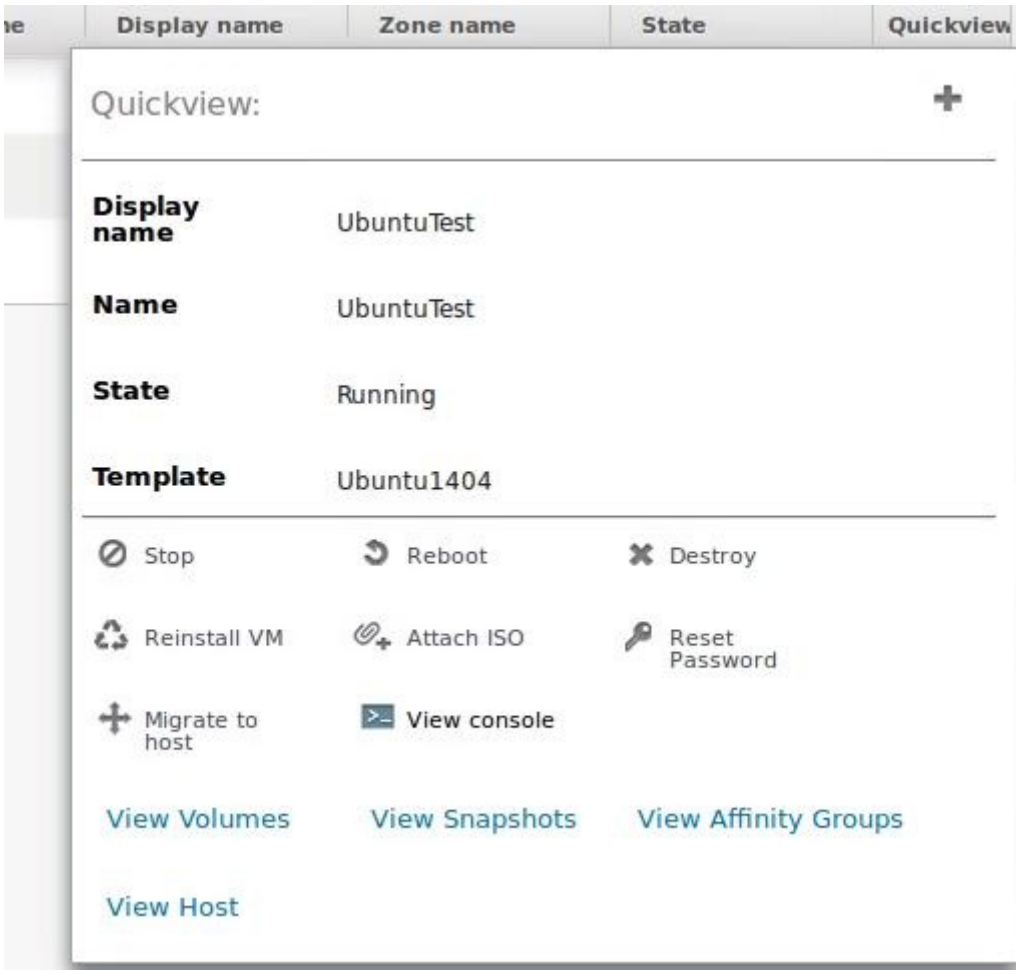
保持默认的Affinity和Network不变，在最后一步的Review中，定义该虚拟机的名字, 最 后点击Launch VM:



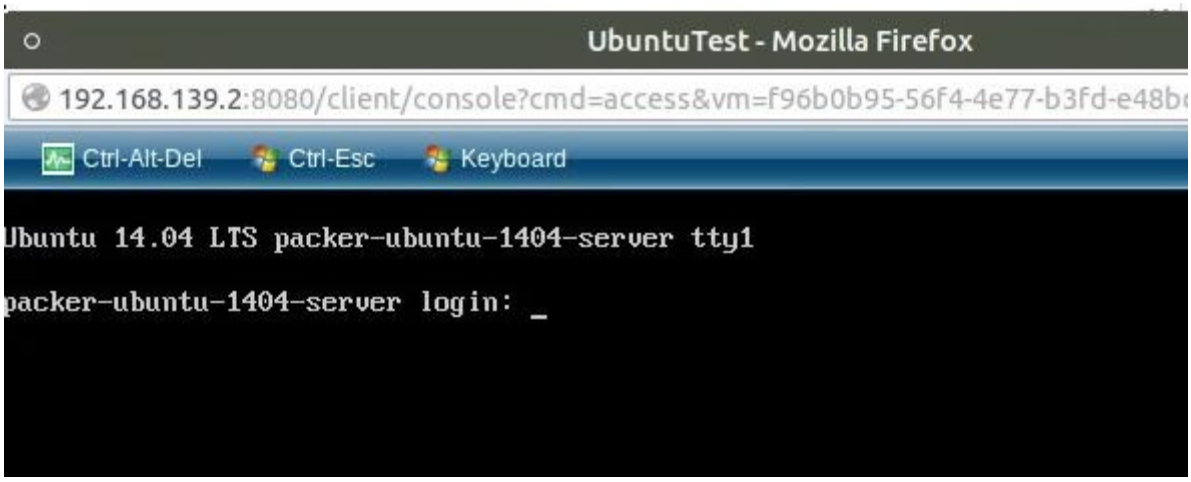
创建完毕的虚拟机实例如下:



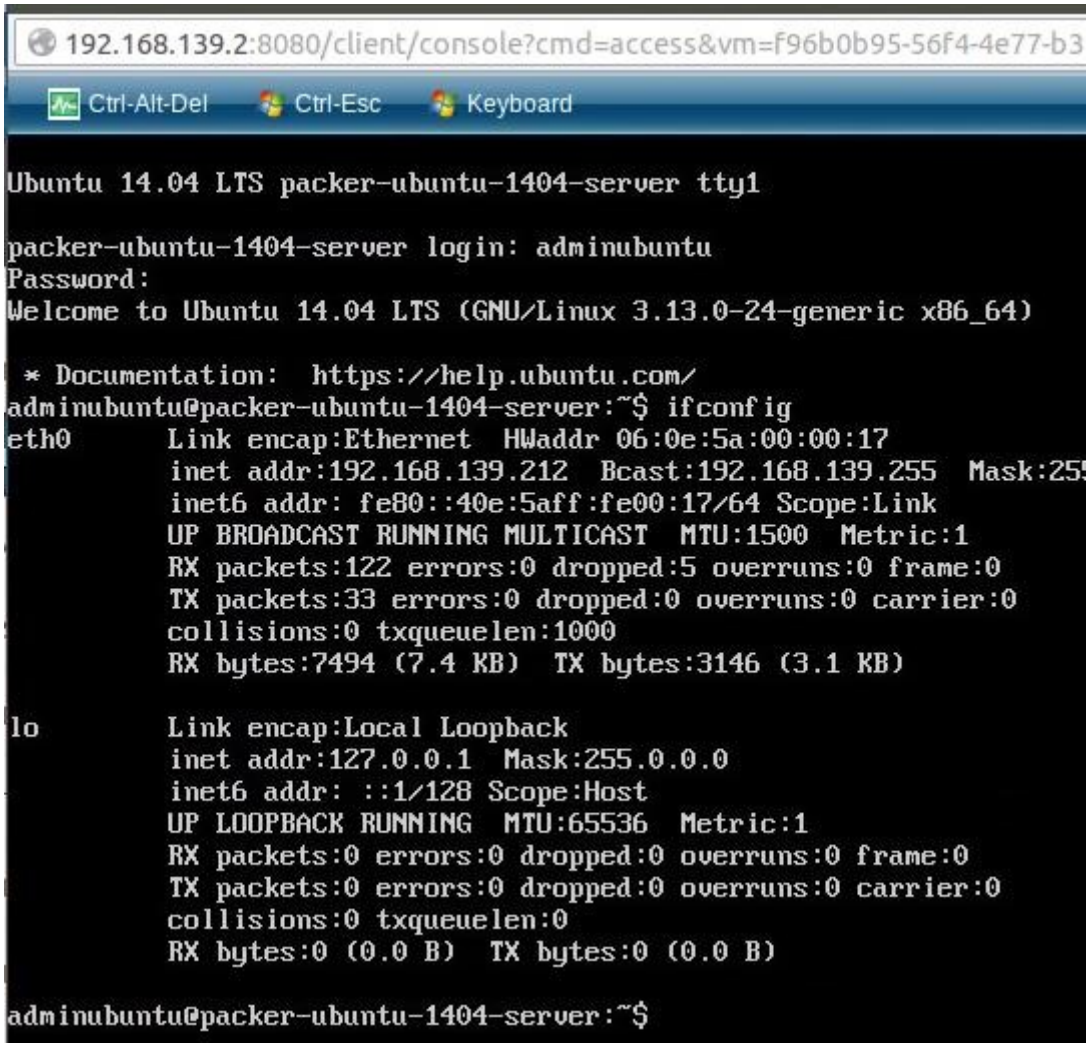
在quick view下点击终端窗口可以看到虚拟机的VNC页面:



该虚拟机的VNC页面如下:


UbuntuTest - Mozilla Firefox
192.168.139.2:8080/client/console?cmd=access&vm=f96b0b95-56f4-4e77-b3fd-e48b
Ctrl-Alt-Del
Ctrl-Esc
Keyboard
Ubuntu 14.04 LTS packer-ubuntu-1404-server tty1
packer-ubuntu-1404-server login: _

终端即可登录入该机器:

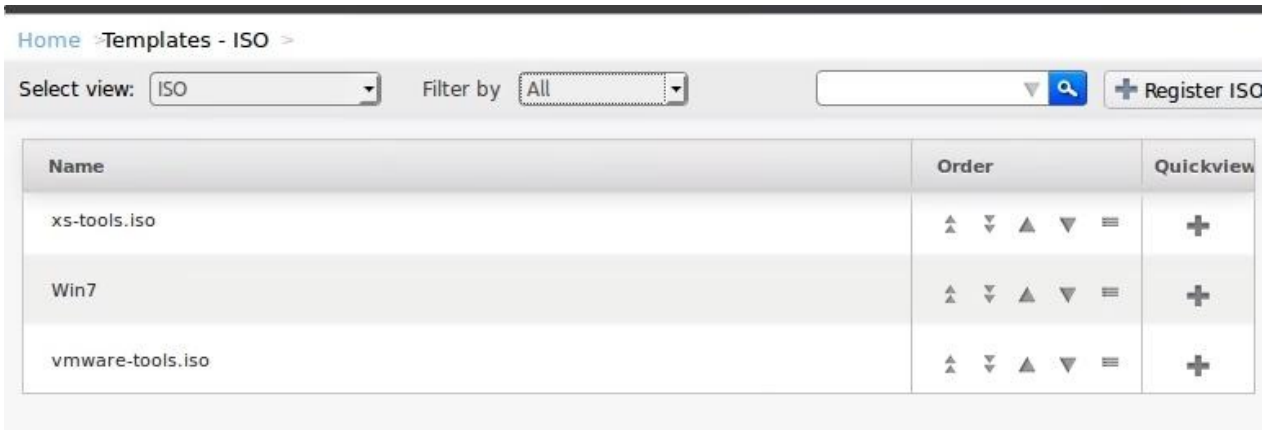

192.168.139.2:8080/client/console?cmd=access&vm=f96b0b95-56f4-4e77-b3
Ctrl-Alt-Del
Ctrl-Esc
Keyboard
Ubuntu 14.04 LTS packer-ubuntu-1404-server tty1
packer-ubuntu-1404-server login: adminubuntu
Password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)
* Documentation: https://help.ubuntu.com/
adminubuntu@packer-ubuntu-1404-server:~$ ifconfig
eth0 Link encap:Ethernet HWaddr 06:0e:5a:00:00:17
inet addr:192.168.139.212 Bcast:192.168.139.255 Mask:255
inet6 addr: fe80::40e:5aff:fe00:17/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:122 errors:0 dropped:5 overruns:0 frame:0
TX packets:33 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:7494 (7.4 KB) TX bytes:3146 (3.1 KB)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
adminubuntu@packer-ubuntu-1404-server:~$

注册ISO

除了通过模板创建虚拟机外，我们也可以采用从ISO安装创建虚拟机，以下是详细步骤。

查看ISO

在Template下点击ISO，可以看到已经安装的ISO列表， 如下图:

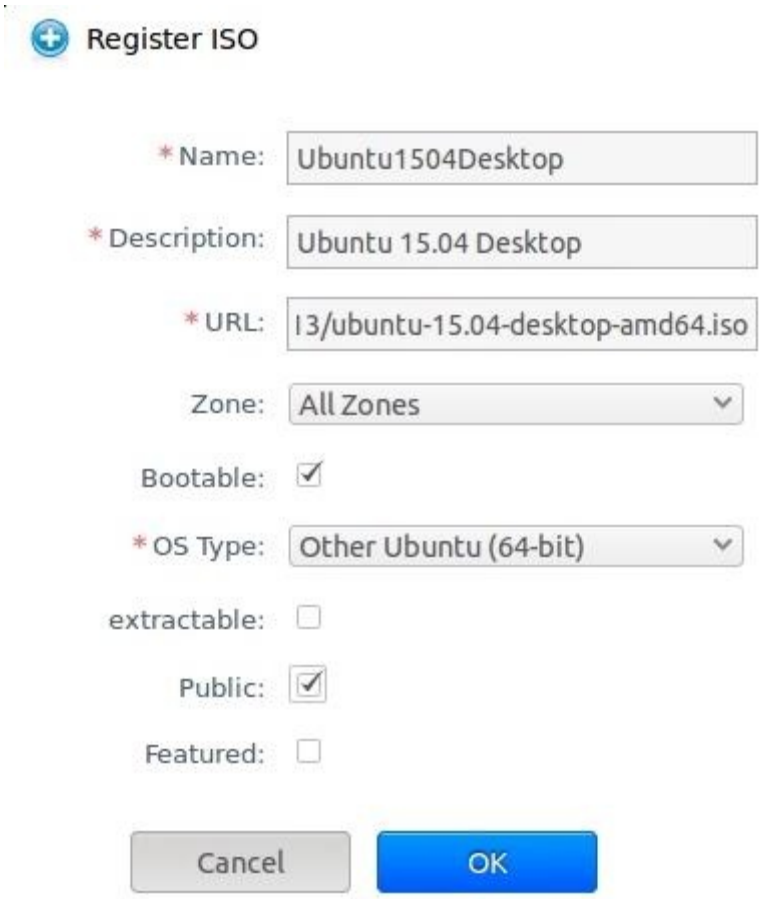


其中Win7是我们手工导入的ISO， xs-tools.iso和vmware-tools.iso是Cloudstack自带的 ISO，用于在生成的XenServer的虚拟机和VMware的虚拟机中安装工具。接下来我们将手工 导入一个ISO用于安装虚拟机。

导入ISO

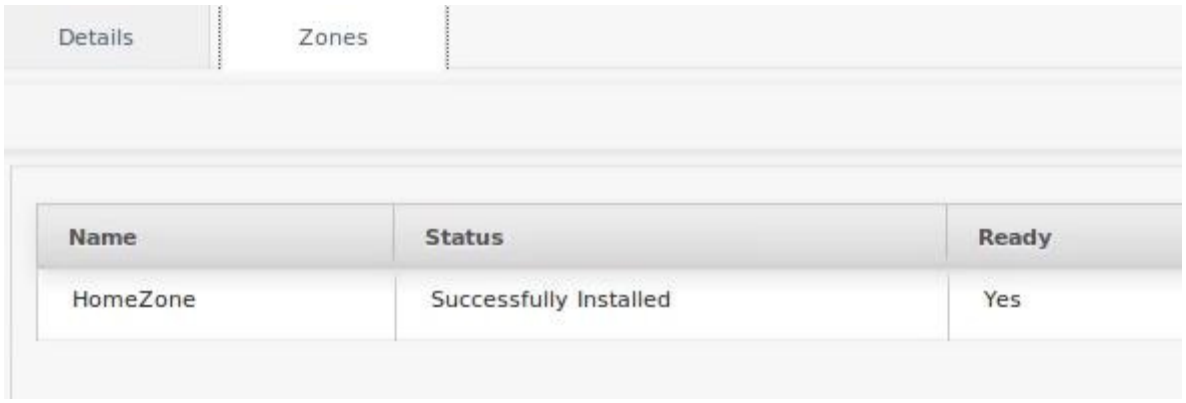
首先在Global Settings里添加internal sites的允许下载IP地址， 如下图所示，更改完 毕以后，重启Coudstack-management服务:

首先准备好Ubuntu15.04的ISO，放到Web服务器的根目录下, 而后点击Registe ISO,在弹 出的页面中输入以下内容:



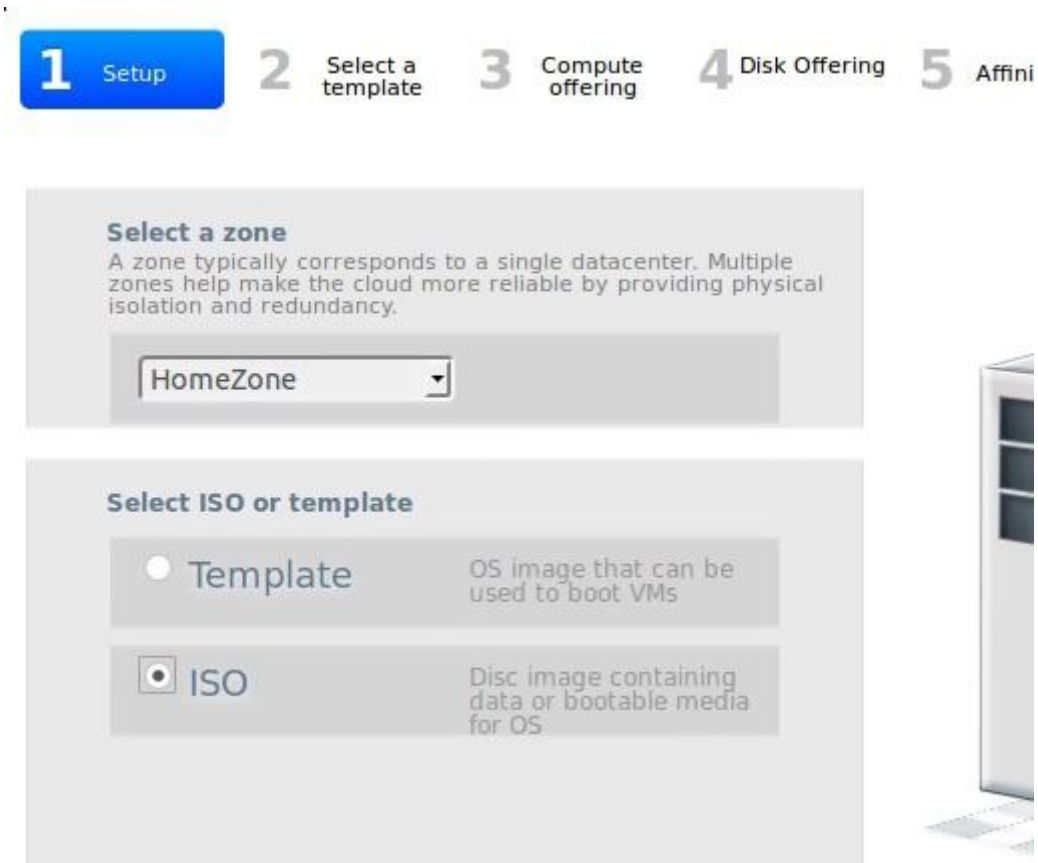
等待一段时间后，点击查看是否被完整导入。安装ISO中的状态如下：

可用状态如下:

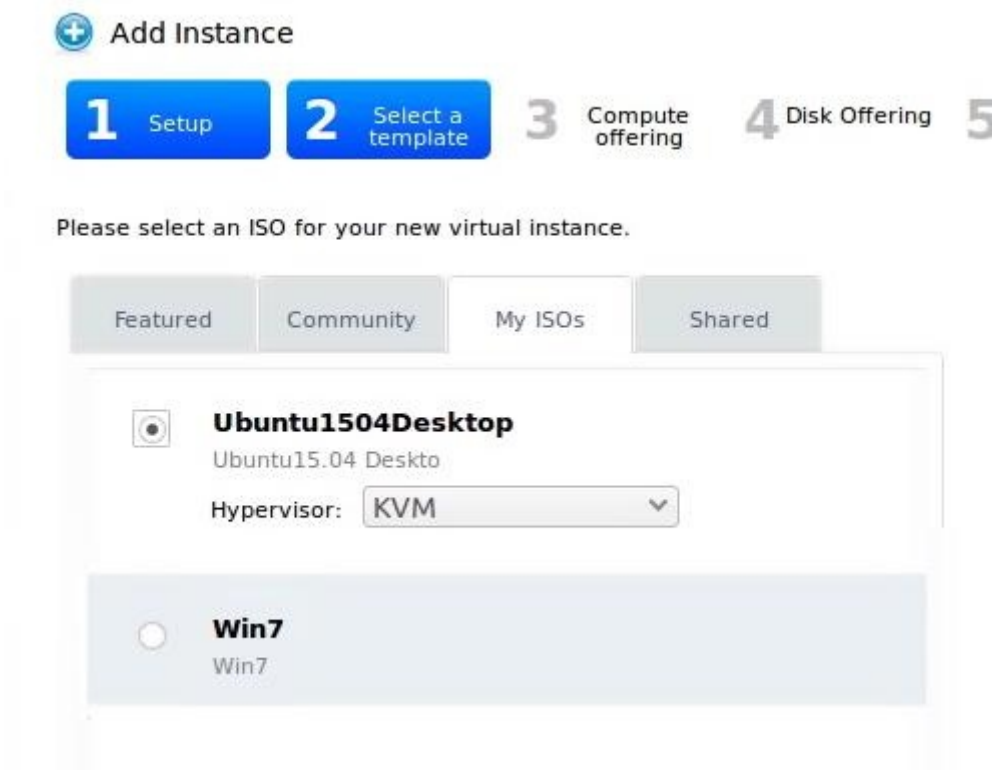


从ISO创建虚拟机

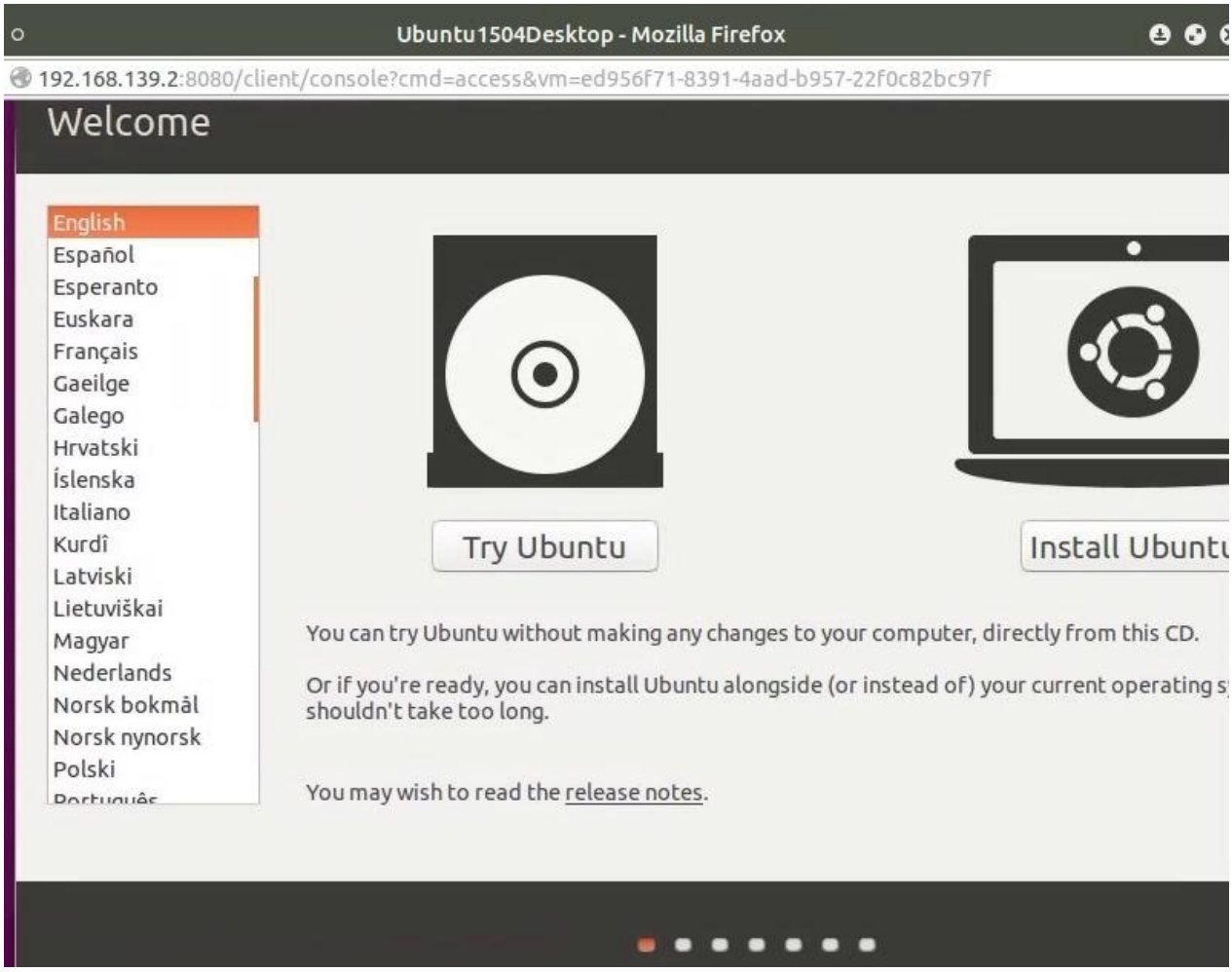
点击Instance -> Add Instance， 在选择zone时，选择ISO安装:



在选择模板时，选择对应的ISO：



剩下的过程和从模板创建的过程相同， 点击View Console后，我们将看到系统安装的页 面:

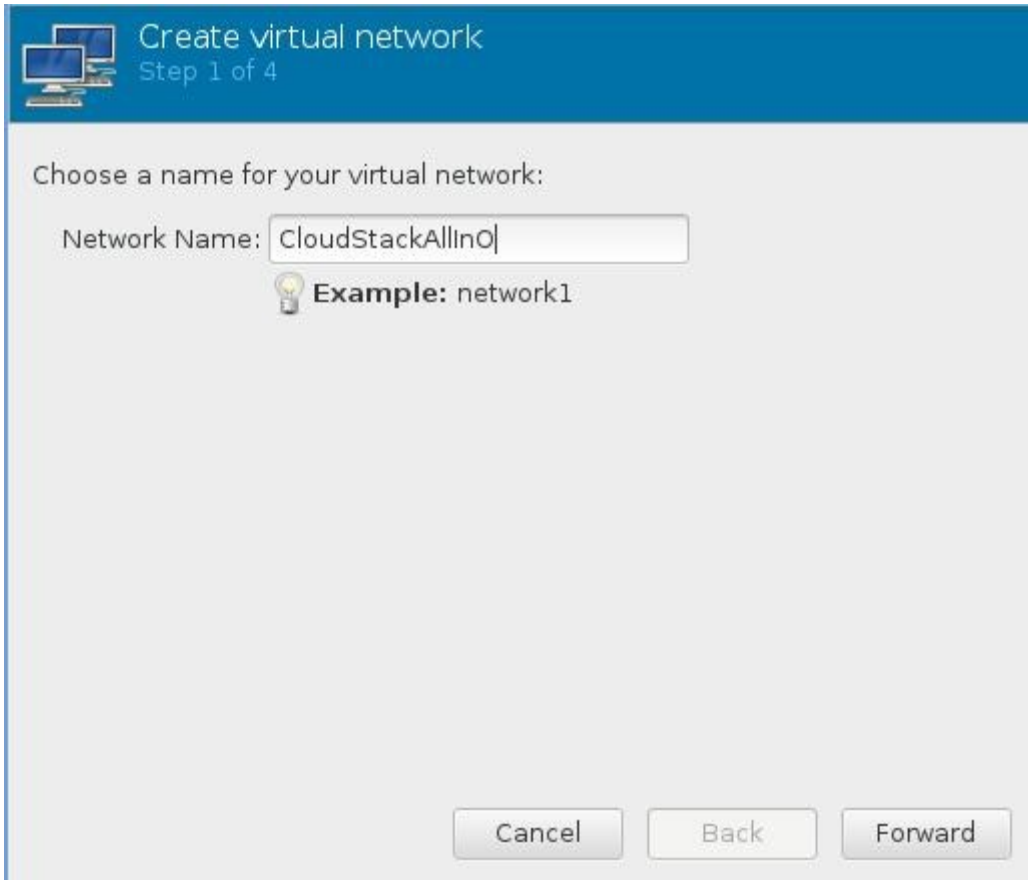


安装完后的VM与从模板创建的VM无异。

CloudStackAllInOne

本节将建立一个AllInOne的CloudStack环境，在此基础上将引入高级网络模式。

网络准备



安装步骤

配置IP地址:

```
# yum install -y bridge-utils
# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
ONBOOT="yes"
BOOTPROTO=none
NM_CONTROLLED="no"
BRIDGE="br0"
TYPE=Ethernet
# cat /etc/sysconfig/network-scripts/ifcfg-br0
DEVICE="br0"
ONBOOT="yes"
NM_CONTROLLED=yes
TYPE="Bridge"
BOOTPROTO=static
IPADDR=10.168.100.2
NETMASK=255.255.255.0
GATEWAY=10.168.100.1
DNS1=223.5.5.5
DEFROUTE=yes
```

接着配置CloudStack Management服务器，类似于上面提到的。

CloudStackAllInOneCentOS7

这里记载在CentOS7上搭建AllInOne环境。

网络配置

如上所示，加入到某隔绝网络中，我们选择 `10.168.100.1/24` 这个网络作为该节点所处的网络，设 置地址如下:

```
# vim /etc/sysconfig/network-scripts/ifcfg-eth0
    # Generated by dracut initrd
    NAME="eth0"
    ONBOOT=yes
    NETBOOT=yes
    UUID="51916db1-8f30-458d-a958-3f4adff662ea"
    IPV6INIT=yes
    BOOTPROTO=static
    IPADDR=10.168.100.20
    GATEWAY=10.168.100.1
    NETMASK=255.255.255.0
    DNS1=223.5.5.5
    TYPE=Ethernet
$ yum install -y net-tools vim bridge-utils
```

现在创建网桥cloudbr0:

```
# cat /etc/sysconfig/network-scripts/ifcfg-eth0
    DEVICE=eth0
    TYPE=Ethernet
    ONBOOT=yes
    NM_CONTROLLED=yes
    BOOTPROTO=none
    BRIDGE=cloudbr0
    IPV6INIT=no
# cat /etc/sysconfig/network-scripts/ifcfg-cloudbr0
    DEVICE=cloudbr0
    TYPE=Bridge
    BOOTPROTO=static
    IPADDR=10.168.100.20
    GATEWAY=10.168.100.1
    NETMASK=255.255.255.0
    DNS1=223.5.5.5
    IPV6INIT=no
    ONBOOT=yes
    DELAY=0
```

FQDN

主机名及FQDN配置如下:

```
# vim /etc/hosts
    10.168.100.20           cloudstackc7
    127.0.0.1               localhost
    ::1     localhost       ip6-localhost   ip6-loopback
    fe00::0 ip6-localnet
    ff00::0 ip6-mcastprefix
    ff02::1 ip6-allnodes
    ff02::2 ip6-allrouters
# vim /etc/hostname
    cloudstackc7
```

检查主机名如下:

```
# hostname
    cloudstackc7
# hostname --fqdn
    cloudstackc7
```

准备仓库

用预先准备好的cloudstack centos7仓库:

```
# yum install -y wget
# wget http://192.168.0.79/cloudstack7.repo
# yum makecache
# yum search cloudstack
```

CloudStack Management服务器

NTP

安装和配置ntp服务:

```
# yum install -y ntp

# vim /etc/ntp.conf
    driftfile /var/lib/ntp/drift

    restrict default kod nomodify notrap nopeer noquery
    restrict -6 default kod nomodify notrap nopeer noquery

    restrict 127.0.0.1
    restrict -6 ::1

    server 0.uk.pool.ntp.org iburst
    server 1.uk.pool.ntp.org iburst
```

```
    server 2.uk.pool.ntp.org iburst
    server 3.uk.pool.ntp.org iburst

    includefile /etc/ntp/crypto/pw

    keys /etc/ntp/keys

    disable monitor
# service ntpd restart
# chkconfig ntpd on
```

SELinux

禁止SELinux:

```
# vim /etc/selinux/config
SELINUX=disabled
SELINUXTYPE=targeted
```

安装CloudStack-Management

CentOS7 使用mariadb代替了MySQL:

```
# yum install -y mariadb-server
# yum install -y MySQL-python
```

配置数据库, 在/etc/my.cnf的[mysqld_safe]条目前添加如下定义后，重新启动mariadb并确保其在 开机时自动加载:

```
# vim /etc/my.cnf
  + # CloudStack MySQL settings
    + innodb_rollback_on_timeout=1
    + innodb_lock_wait_timeout=600
    + max_connections=700
    + log-bin=mysql-bin
    + binlog-format = 'ROW'
    + bind-address=0.0.0.0

    [mysqld_safe]
# service mariadb status
# service mariadb start
# chkconfig mariadb on
```

移除 anonymous 用户:

```
# mysql
mysql>  SELECT User, Host, Password FROM mysql.user;
+------+-----------+----------+
```

```
| User | Host      | Password |
+------+-----------+----------+
| root | localhost |          |
| root | csmgmt    |          |
| root | 127.0.0.1 |          |
|      | localhost |          |
|      | csmgmt    |          |
+------+-----------+----------+
mysql> DROP USER ''@'cloudstackc7';
mysql> DROP USER ''@'localhost';
mysql>  SELECT User, Host, Password FROM mysql.user;
+------+-----------+----------+
| User | Host      | Password |
+------+-----------+----------+
| root | localhost |          |
| root | csmgmt    |          |
| root | 127.0.0.1 |          |
+------+-----------+----------+
3 rows in set (0.00 sec)
```

删除testdb:

```
mysql> select * from mysql.db;
........
mysql> DELETE FROM mysql.db WHERE Db LIKE 'test%';
Query OK, 2 rows affected (0.00 sec)

mysql> select * from mysql.db;
Empty set (0.00 sec)
mysql> DROP DATABASE test;
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
mysql> \q
Bye
```

加密MYSQL安装并更改root用户密码，可以通过以下命令完成, 第一次输入空密码，而后可以设置root用户的访问密码:

```
# mysql_secure_installation
```

配置防火墙，关闭firewalld, 开启iptables:

```
# systemctl mask firewalld
# yum -y install iptables-services
# systemctl enable iptables
```

开启 iptables规则:

```
# iptables -A INPUT -p tcp -m tcp --dport 3306 -j ACCEPT
# iptables-save>/etc/sysconfig/iptables
# reboot
```

CloudStack-Management安装包

安装:

```
#  yum install -y cloudstack-management
```

Cloudmonkey

从epel仓库安装pip后，用pip安装cloudmonkey:

```
# wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo
# yum install -y python-pip
# pip install cloudmonkey
```

配置数据库

这里用到的root密码即我们在`mysql_secure_installation`中设置的root密码:

```
# cloudstack-setup-databases cloud:engine@localhost --deploy-as=root:xxxxx -i
10.168.100.20
```

配置CloudStack Management

配置命令:

```
# cloudstack-setup-management
```

现在打开浏览器访问`http://10.168.100.20:8080/client`则可以访问到CloudStack的配置页面， 使用admin/password登录即可

NFS存储配置

配置NFS共享存储目录:

```
# mkdir -p /export/primary /export/secondary
```

编辑导出目录定义:

```
$ cat >>/etc/exports <<EOM
/export  *(rw,async,no_root_squash,no_subtree_check)
EOM
```

开启并使能rpcbind及nfs-server服务：

```
# systemctl start rpcbind nfs-server
# systemctl enable rpcbind nfs-server
```

加载到/etc/fstab条目:

```
$ IP=10.168.100.10
$ mkdir -p /mnt/primary /mnt/secondary
$ cat >>/etc/fstab <<EOM
$IP:/export/primary   /mnt/primary    nfs
rsize=8192,wsize=8192,timeo=14,intr,vers=3,noauto  0   2
$IP:/export/secondary /mnt/secondary  nfs
rsize=8192,wsize=8192,timeo=14,intr,vers=3,noauto  0   2
EOM
$ mount /mnt/primary
$ mount /mnt/secondary
```

系统虚拟机模板

使用下列命令引入系统虚拟机模板到二级存储中:

```
# /usr/share/cloudstack-common/scripts/storage/secondary/cloud-install-sys-tmplt -m
 /mnt/secondary/ -u http://192.168.0.79/systemvm64template-4.5-kvm.qcow2.bz2 -h kvm -F
```

CloudStack Agent

安装包:

```
# yum install -y cloudstack-agent
```

需要更改掉以下配置中的选项后重启libvirtd:

```
# sed -i 's/#vnc_listen = "0.0.0.0"/vnc_listen = "0.0.0.0"/g' \
 /etc/libvirt/qemu.conf && sed -i 's/cgroup_ \
 controllers=["cpu"]/#cgroup_controllers=["cpu"]/g' /etc/libvirt/qemu.conf
# sed -i 's/#listen_tls = 0/listen_tls = 0/g' \
  /etc/libvirt/libvirtd.conf && sed -i 's/#listen_tcp = 1/listen_tcp = 1/g' \
  /etc/libvirt/libvirtd.conf && sed -i \
```

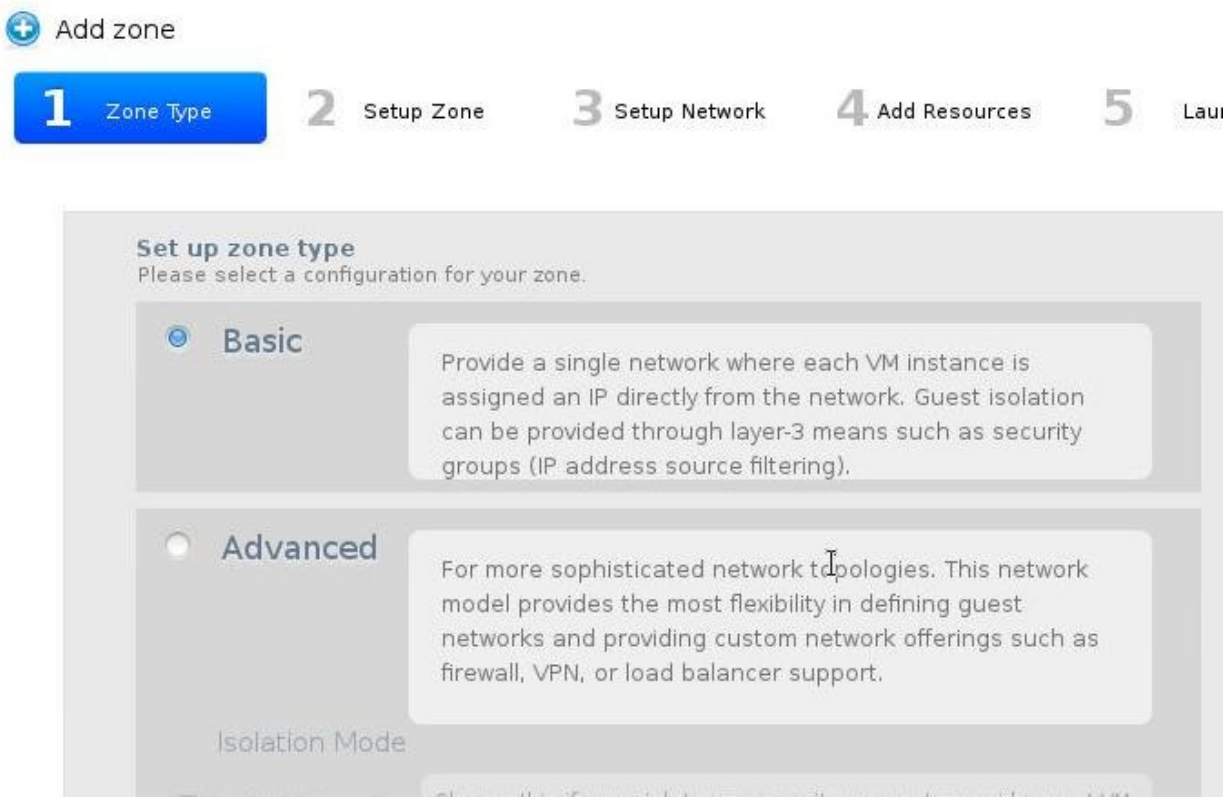
```
 's/#tcp_port = "16509"/tcp_port = "16509"/g' \
 /etc/libvirt/libvirtd.conf && sed -i 's/#auth_tcp = "sasl"/auth_\
 tcp = "none"/g' /etc/libvirt/libvirtd.conf && \
 sed -i 's/#mdns_adv = 1/mdns_adv = 0/g' /etc/libvirt/libvirtd.conf
# sed -i 's/#LIBVIRTD_ARGS="--listen"/LIBVIRTD_ARGS="--listen"/g' \
 /etc/sysconfig/libvirtd
# sed -i '/cgroup_controllers/d' \
  /usr/lib64/python2.7/site-packages/cloudutils/serviceConfig.py
```

重启libvirtd:

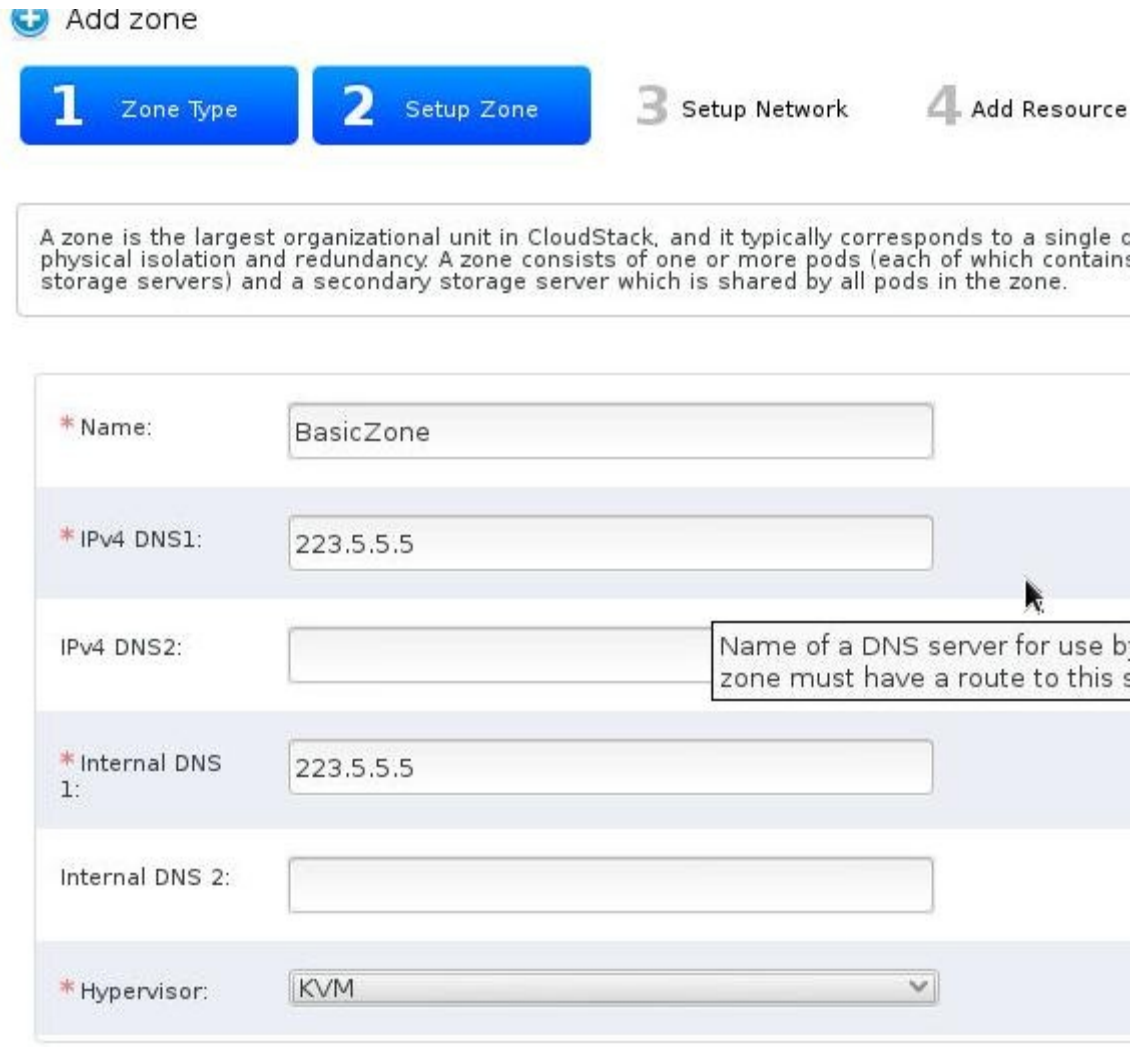
```
# service libvirtd restart
```

配置

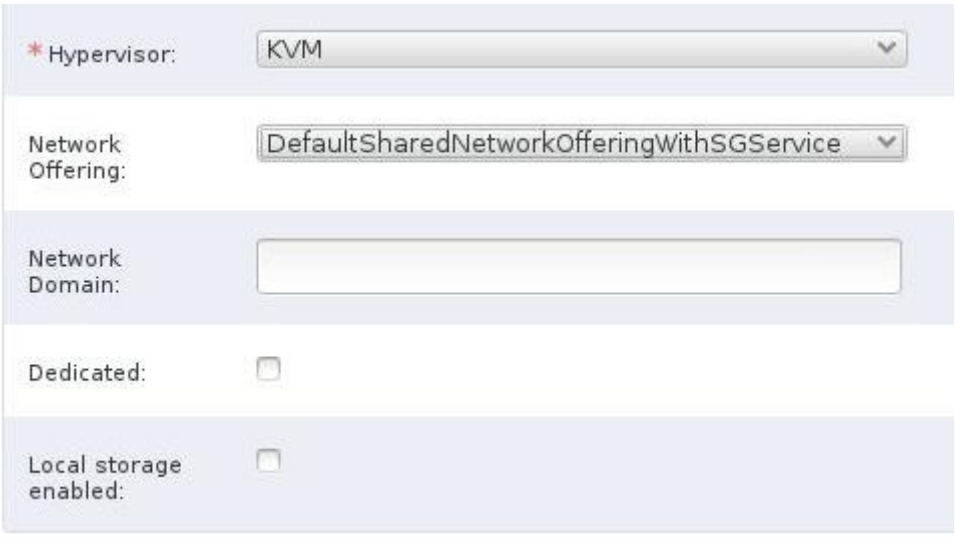
添加一个基本的Zone:



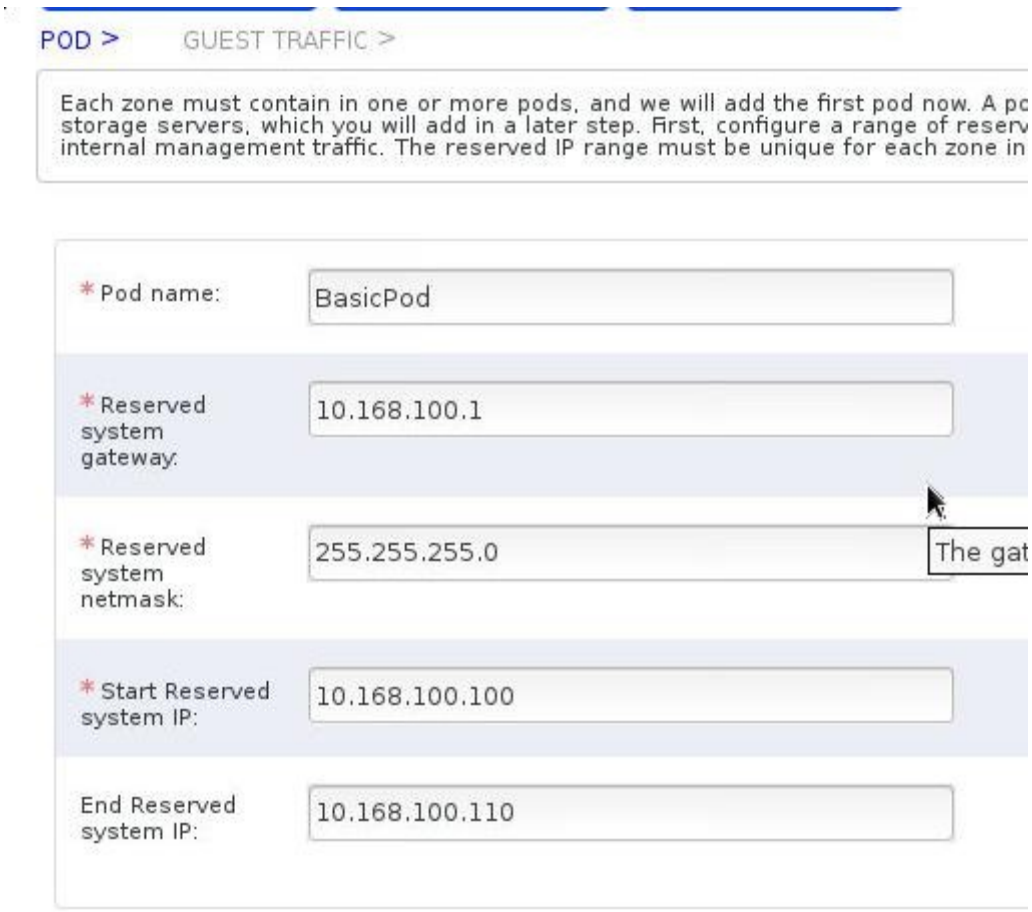
设置DNS Server并选择kvm:



网络类型选择如下:



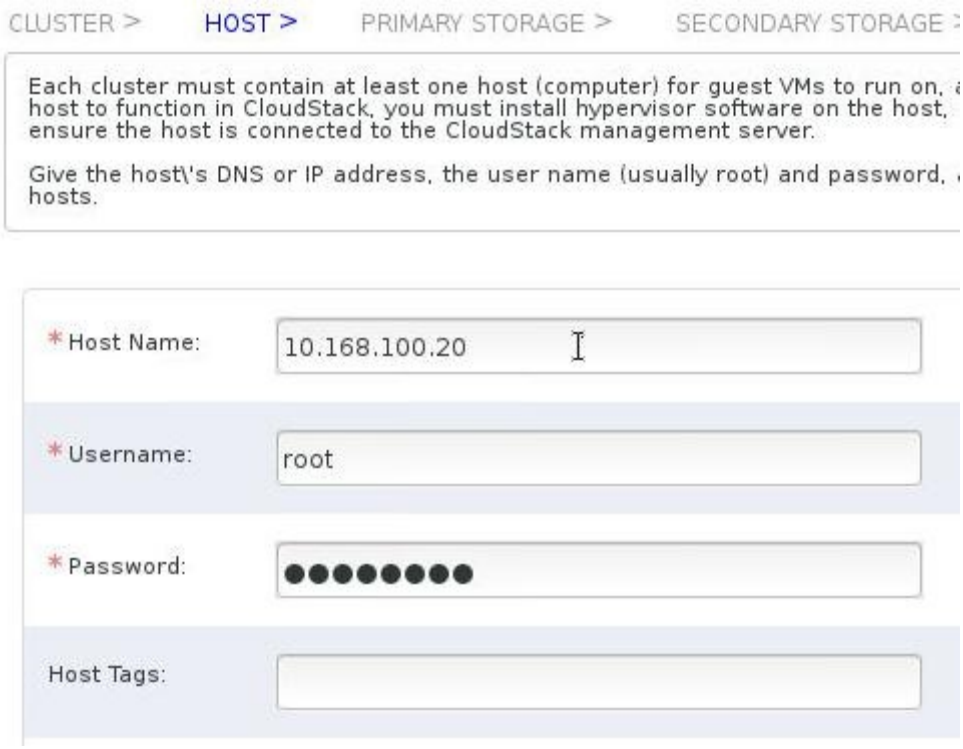
直接点击下一步，进入到POD配置:



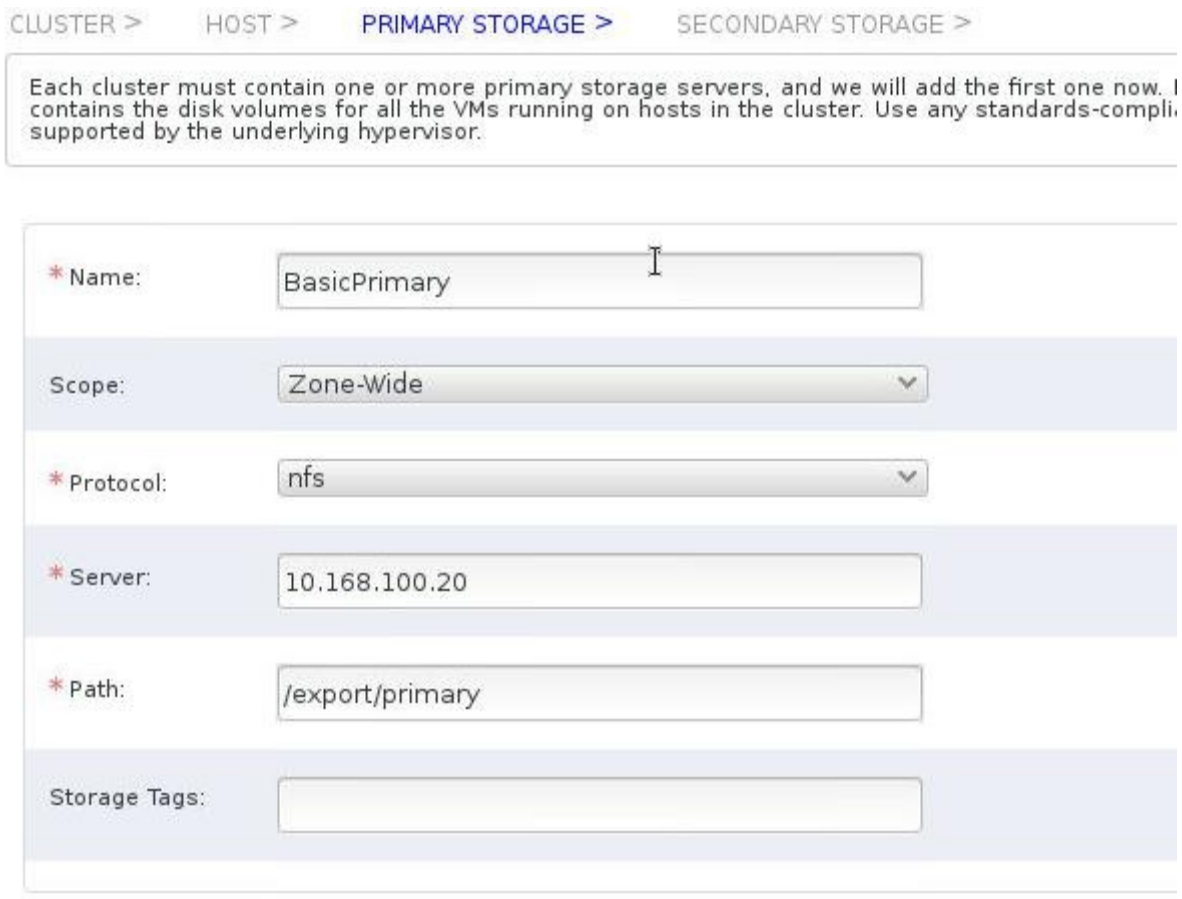
Guest Traffic配置:

配置Cluster:

添加host:



添加主存储:



添加二级存储:

最后一步点击Launch Zone。CloudStack All In One的环境在CentOS 7上就搭建完毕了。

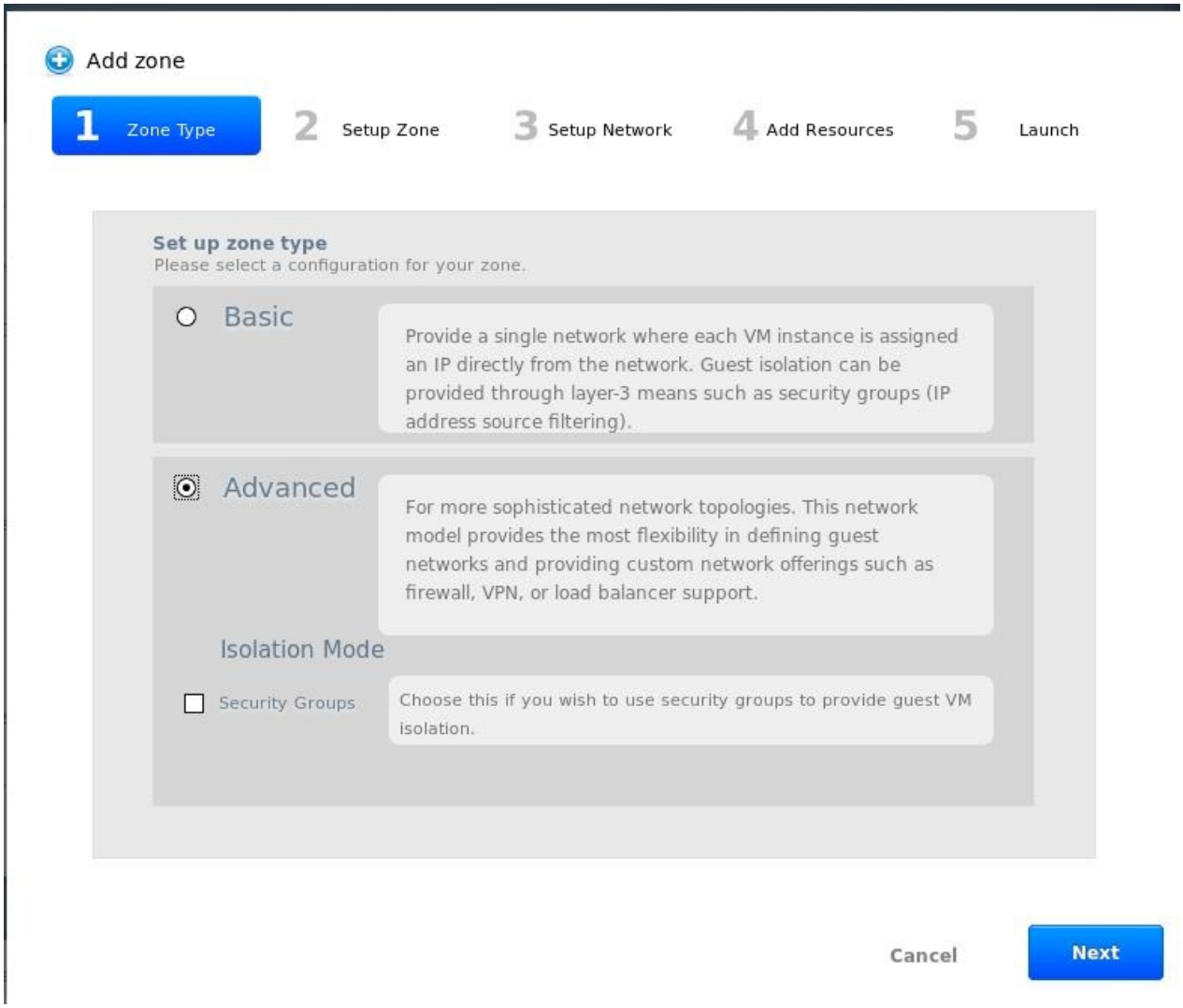
CloudStack高级网络模式

这里讲述如何配置CloudStack高级网络模式.

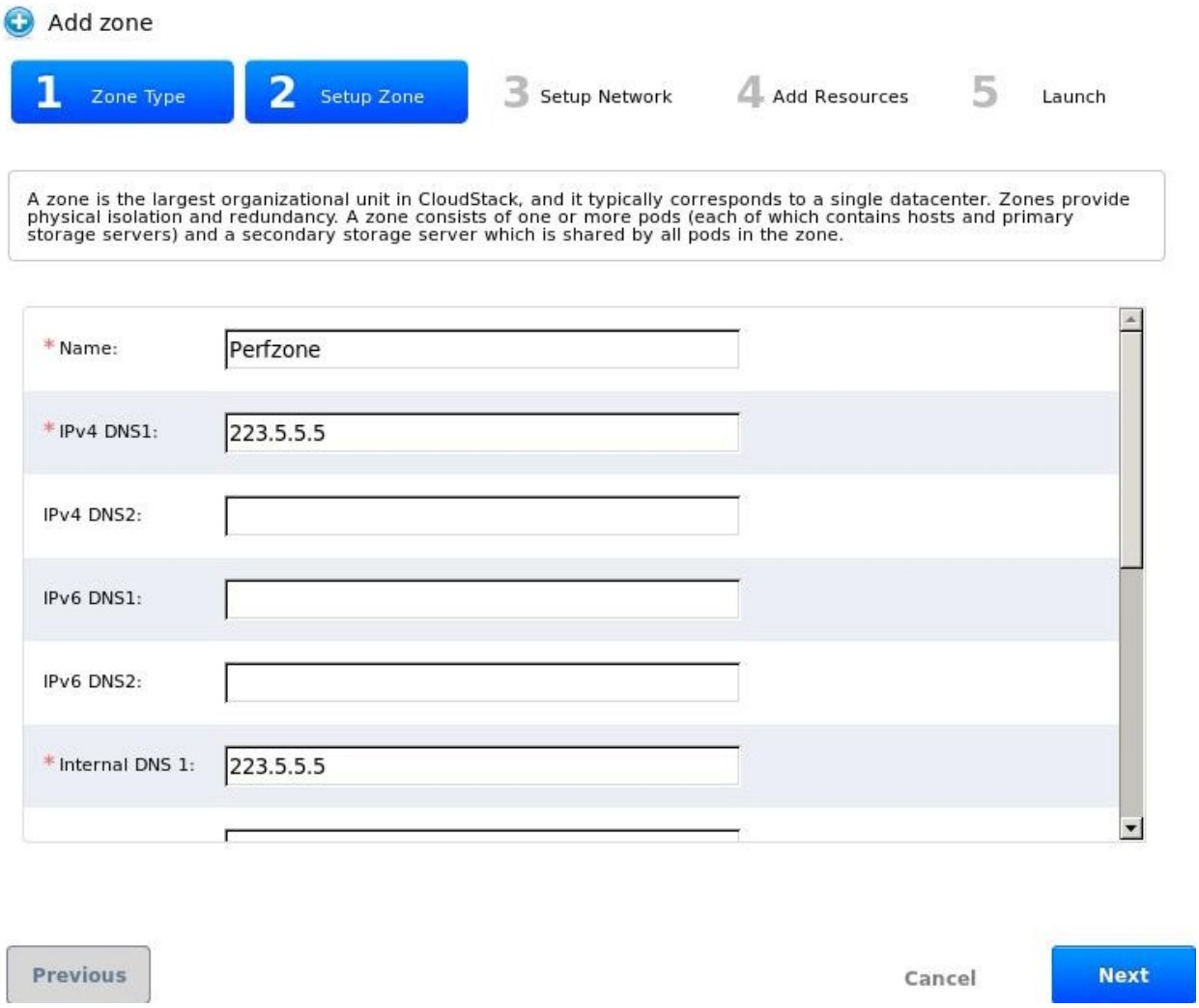
配置高级Zone

配置过程

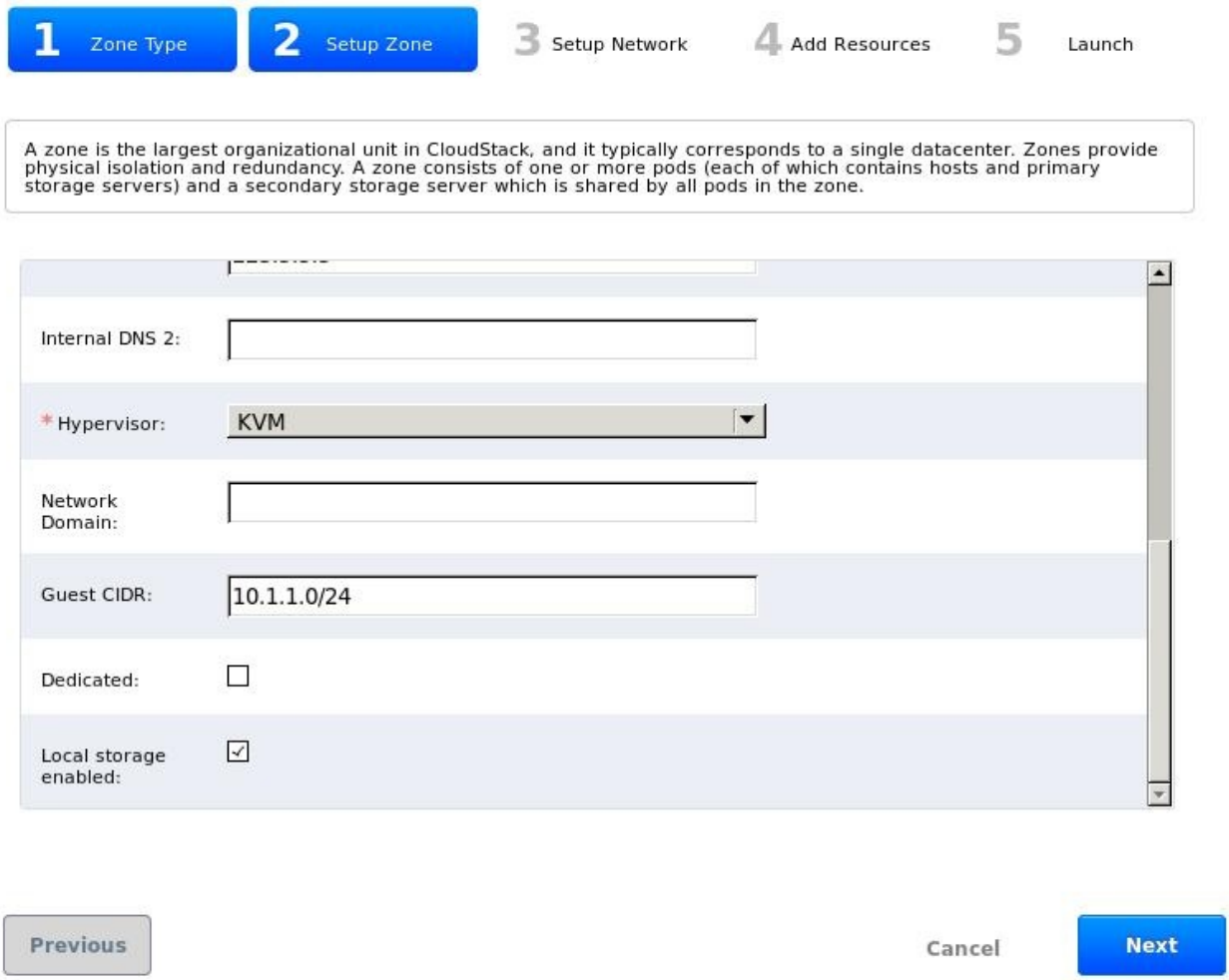
添加一个类型为 `Advanced` 的Zone:



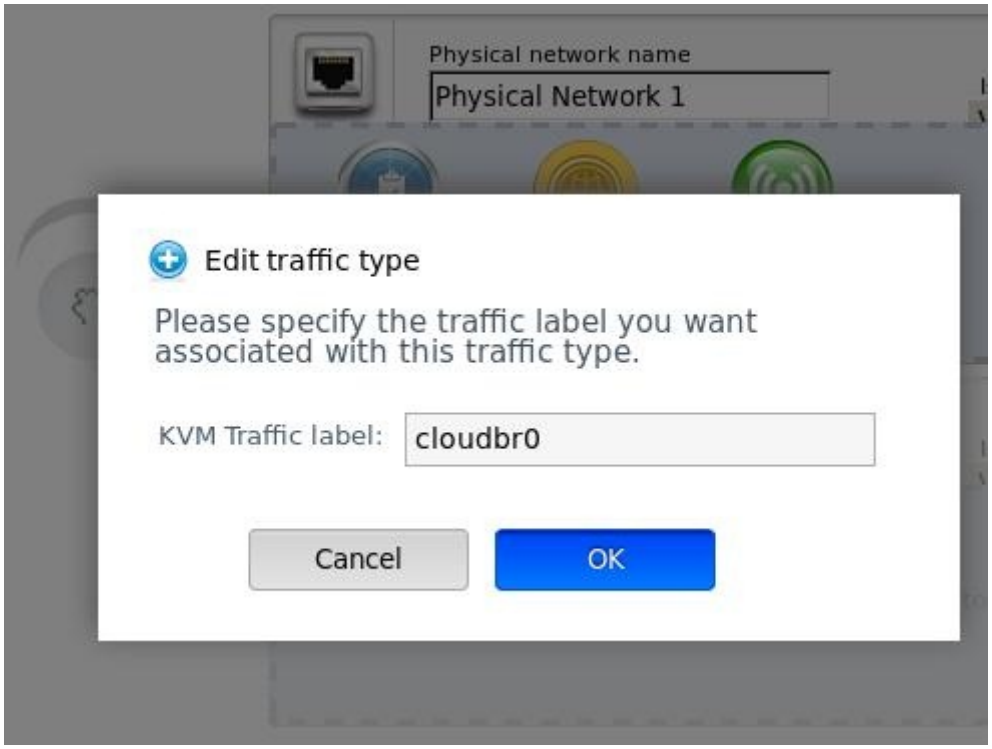
配置Zone信息:



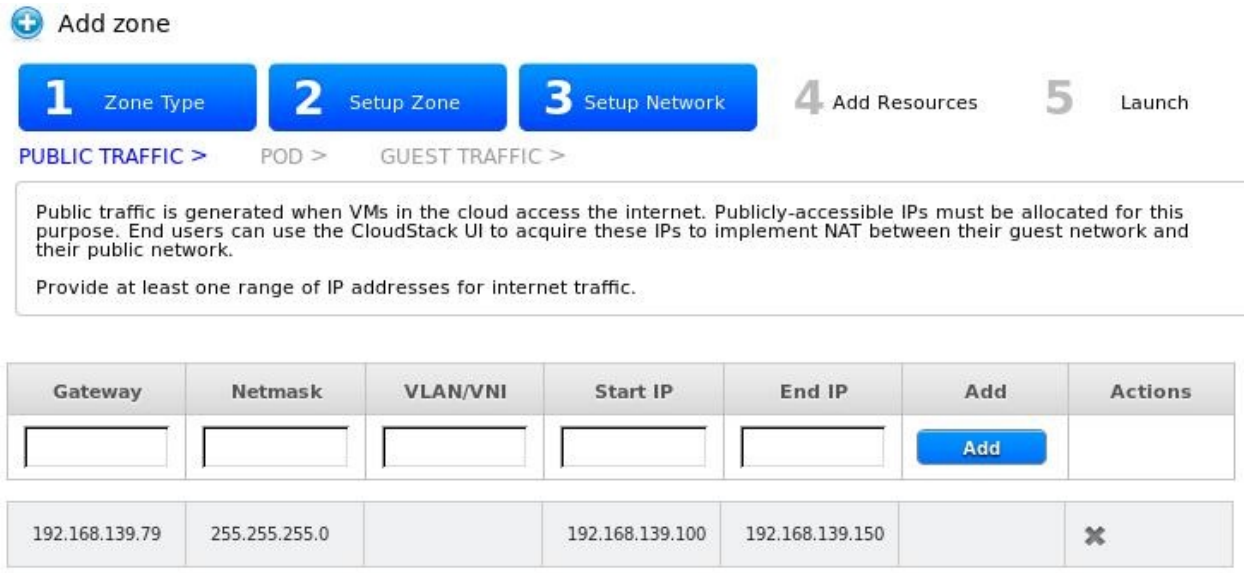
选择HyperVisor等信息:



在下一步的物理网络中，分别设置三个网段的 `KVM Traffic Label` 为cloudbr0, 这和我们 Cloudstack Agent主机上的网桥名称保持一致。

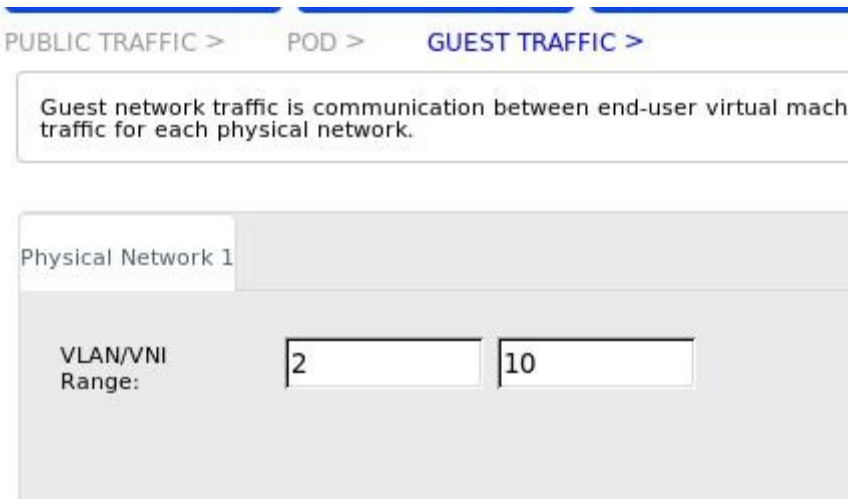


Guest Traffice配置:

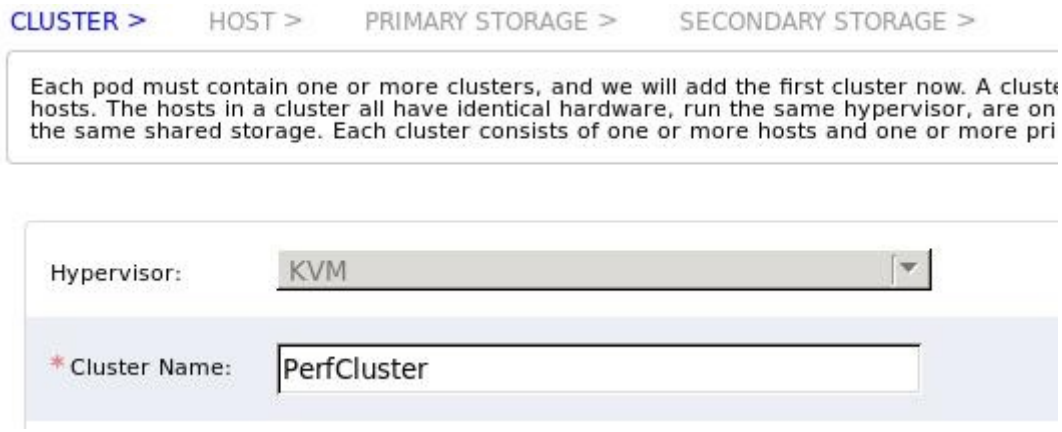


Pod配置:

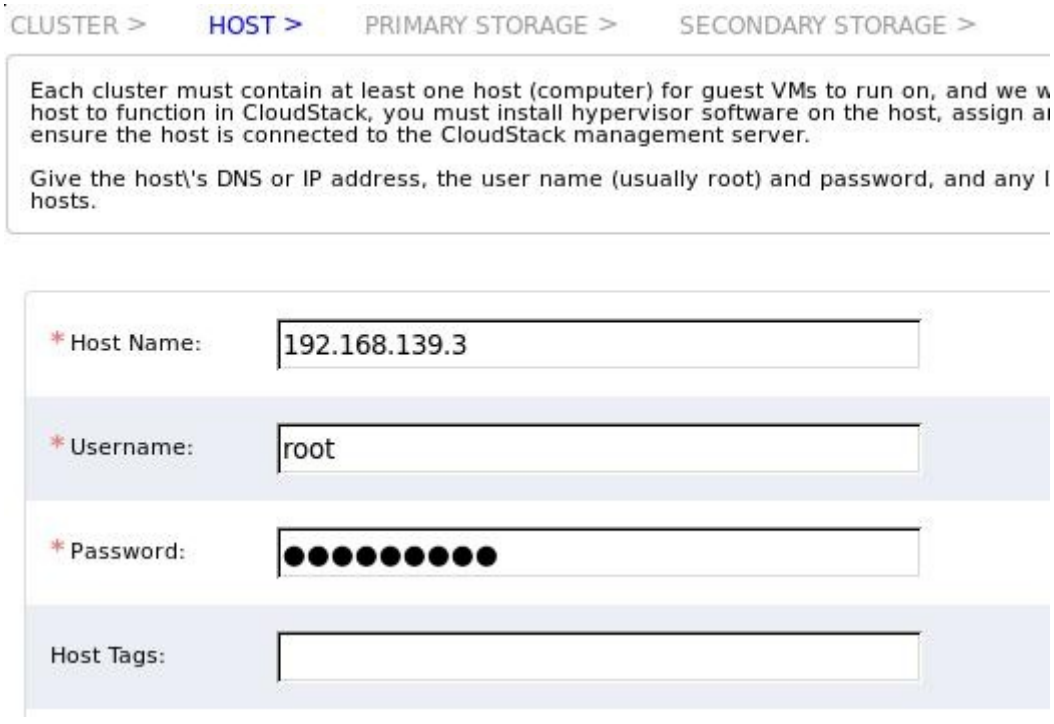
VLAN,随便填写：



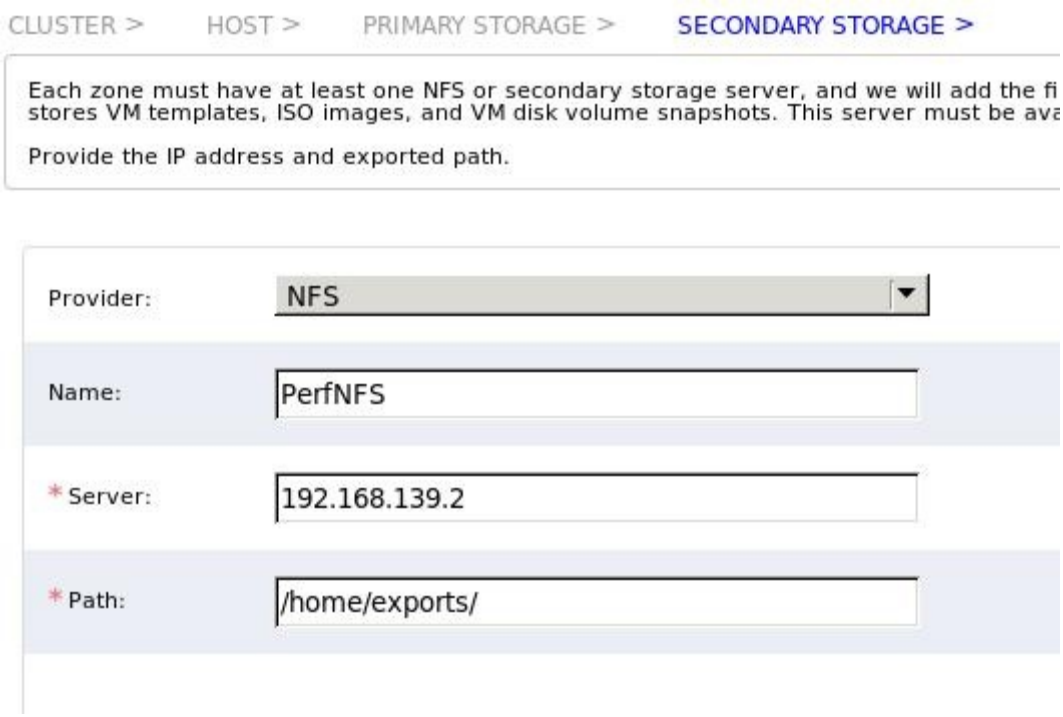
Cluster名字：



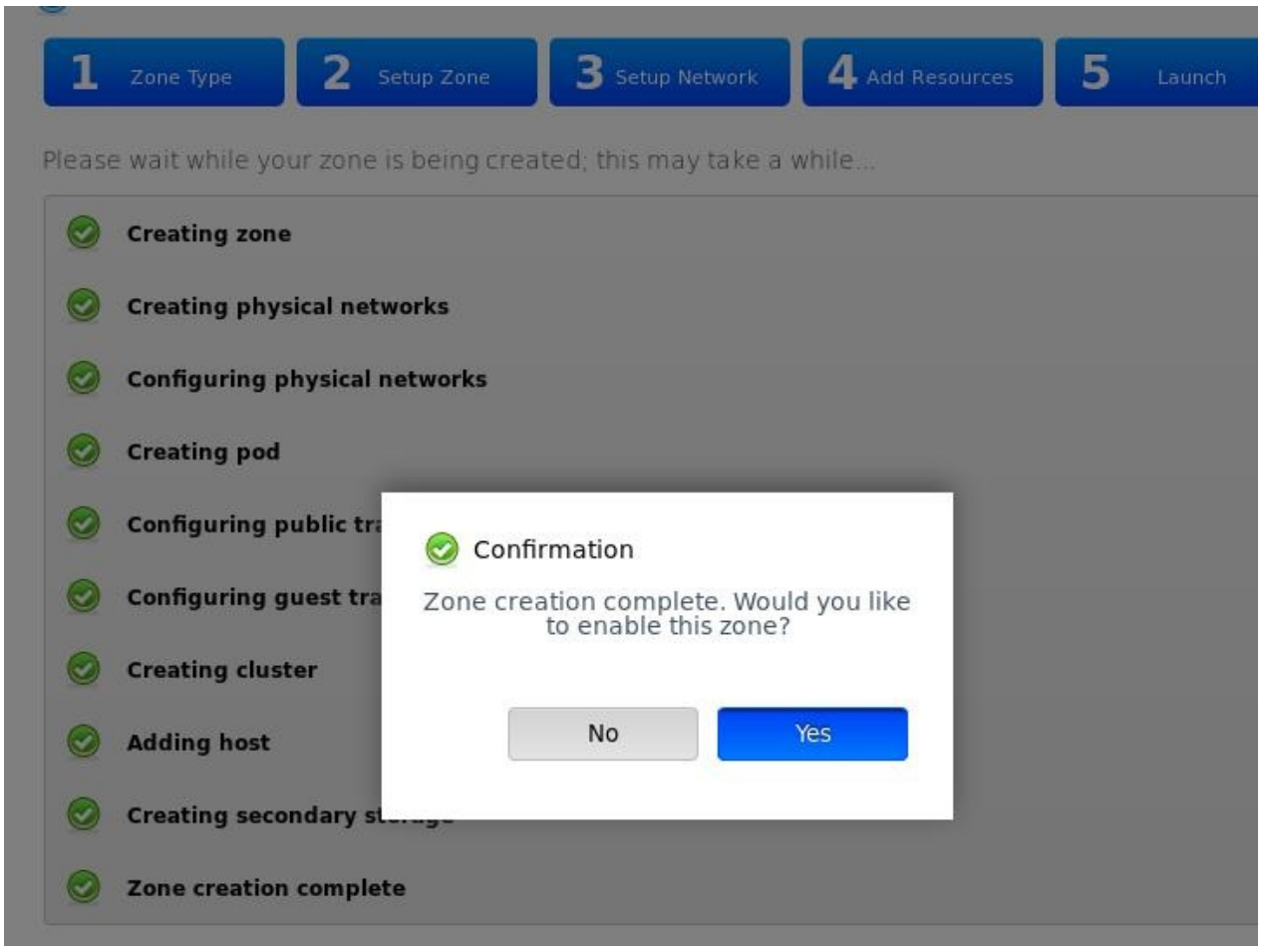
添加主机:



二级存储的添加:



所有配置完毕后，点击 `Launch Zone` 来配置Zone, 若配置成功会弹出以下窗口:

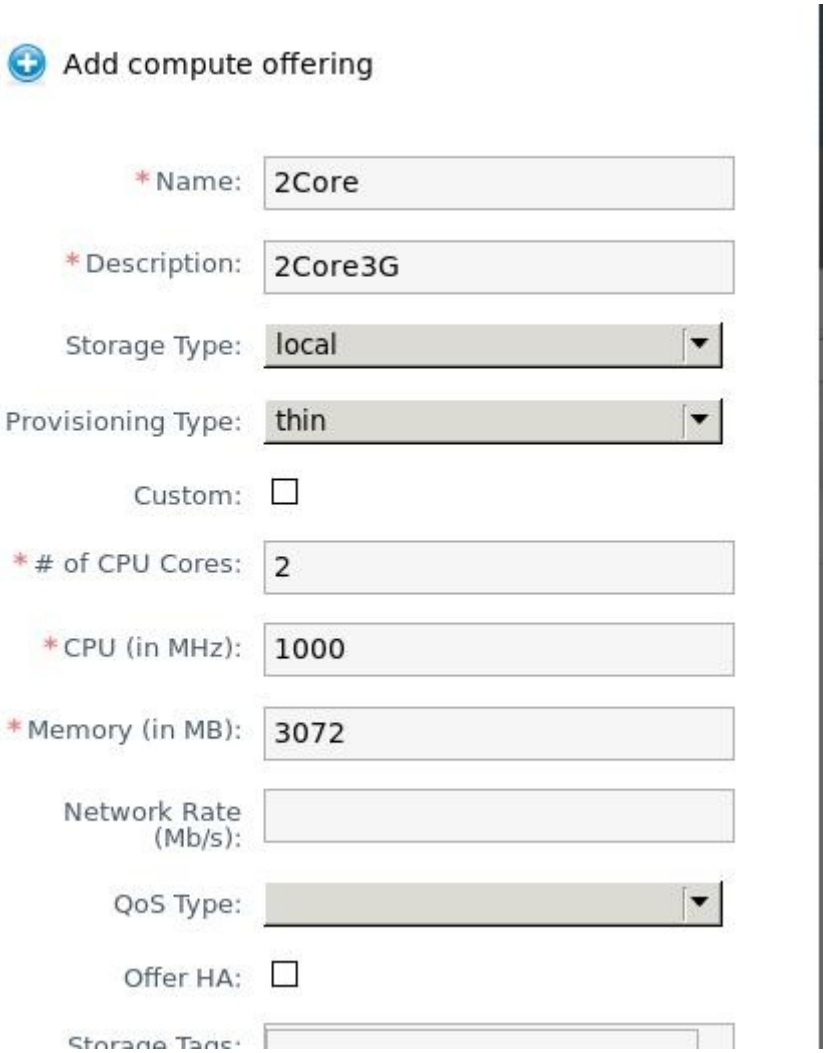


点击 `yes` 后会激活Zone并开始启动该Zone.

配置VM

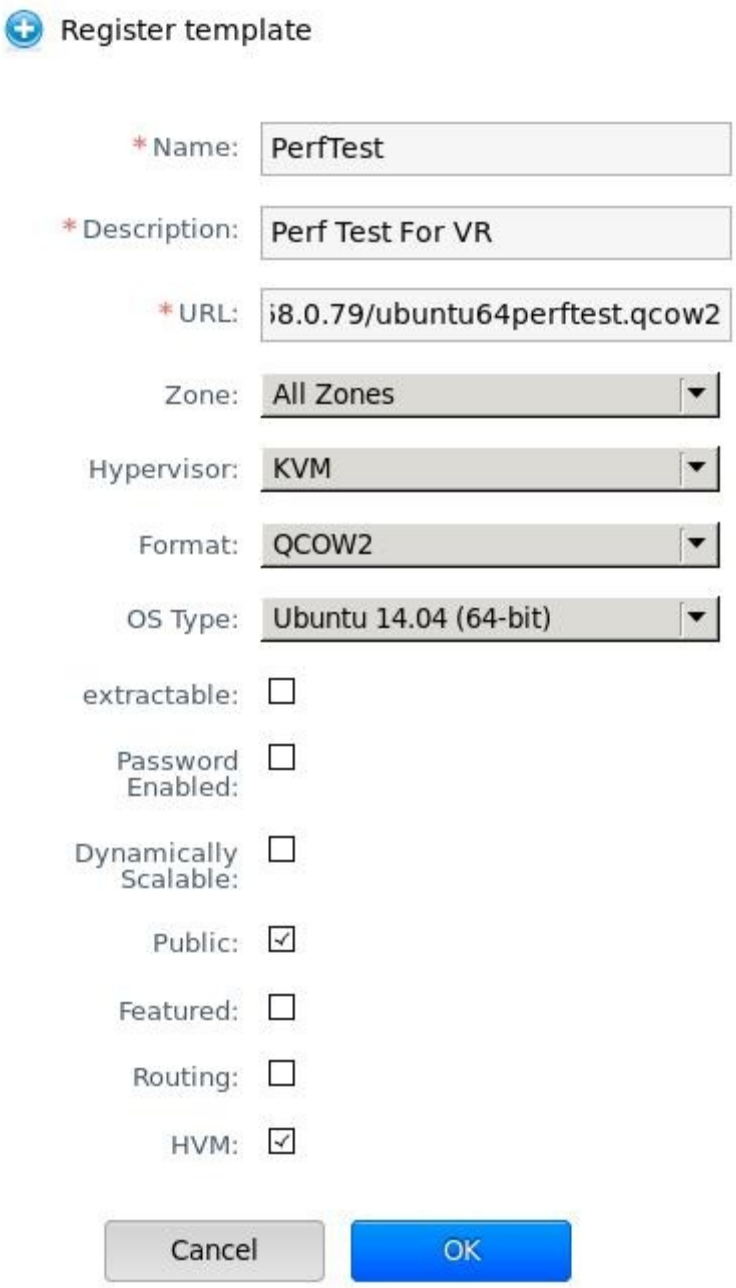
套餐添加

添加一个2核3G的套餐配置，在Service Offering中添加:



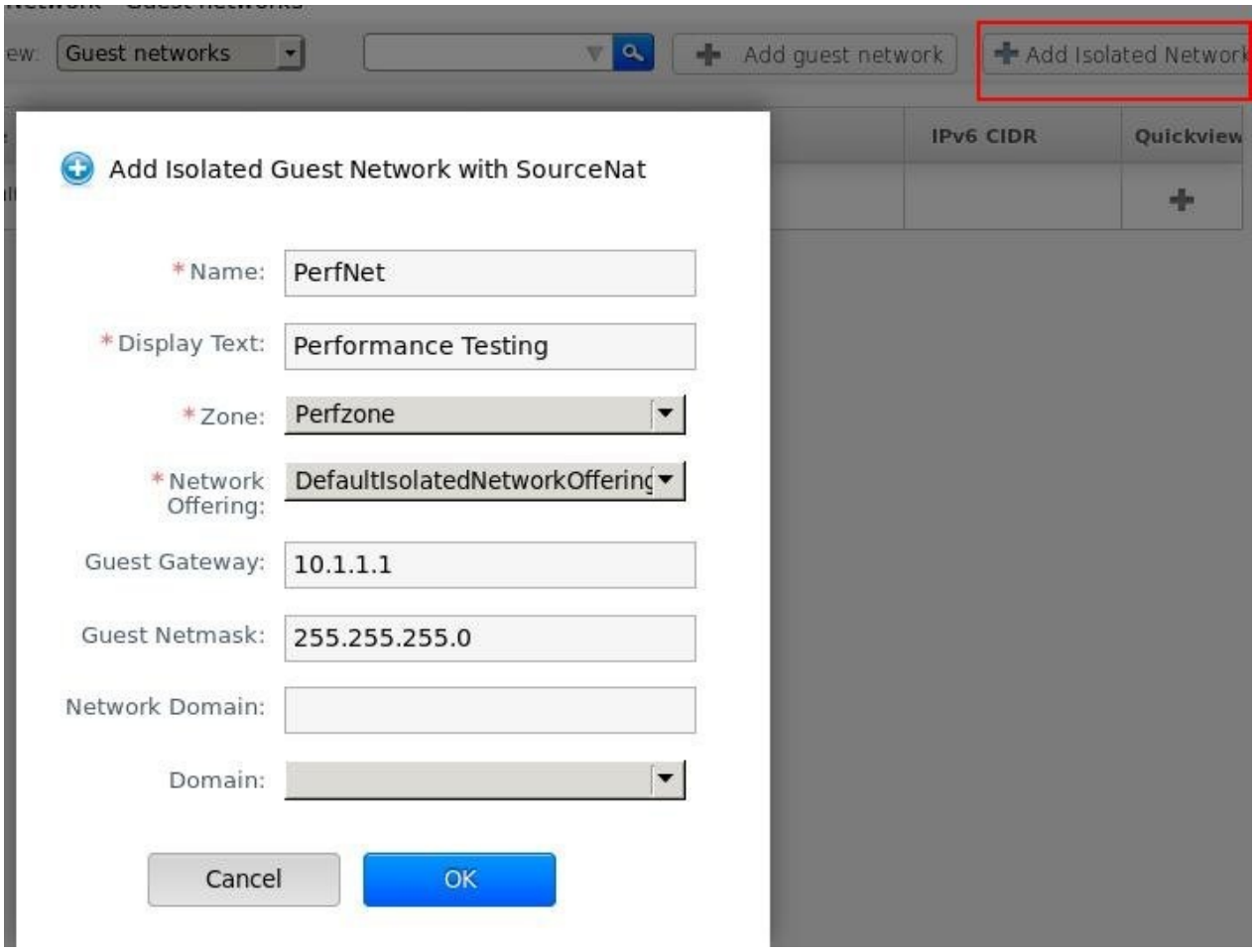
模板下载

添加我们预先定制好的系统镜像:



网络添加

添加以下网络, 注意类型为isolated:

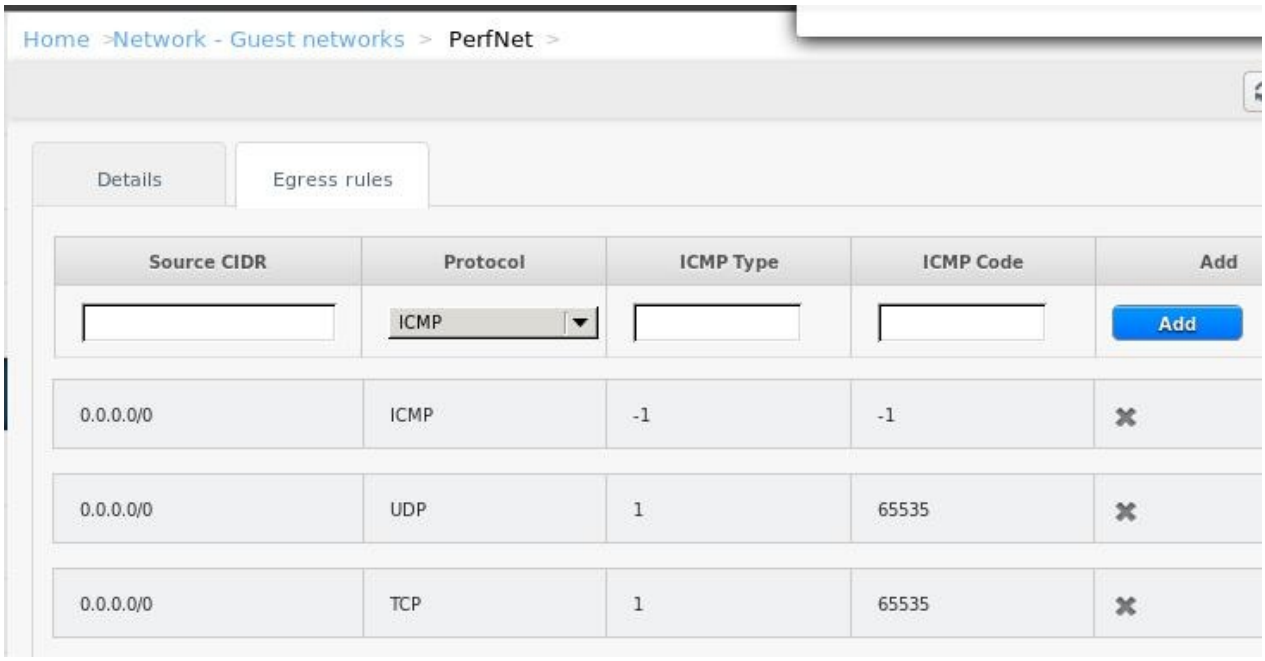


添加虚拟机实例

用上述定义的套餐， 网络， 以及模板， 创建出来一个虚拟机运行实例。具体过程略过。

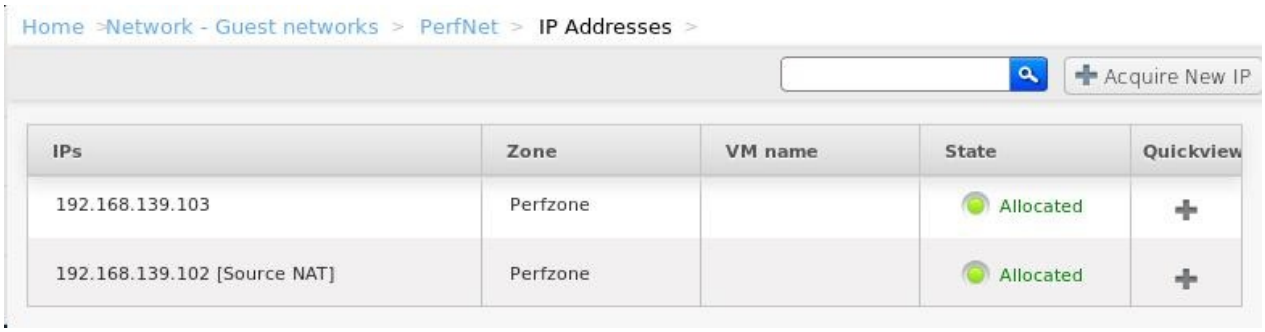
添加NAT

点击Network-> PerfNet, 首先添加Egress规则:



Details -> View All IP Address,可以看到以下：

点击 `Acquire New IP` 后得到一个新的IP地址:

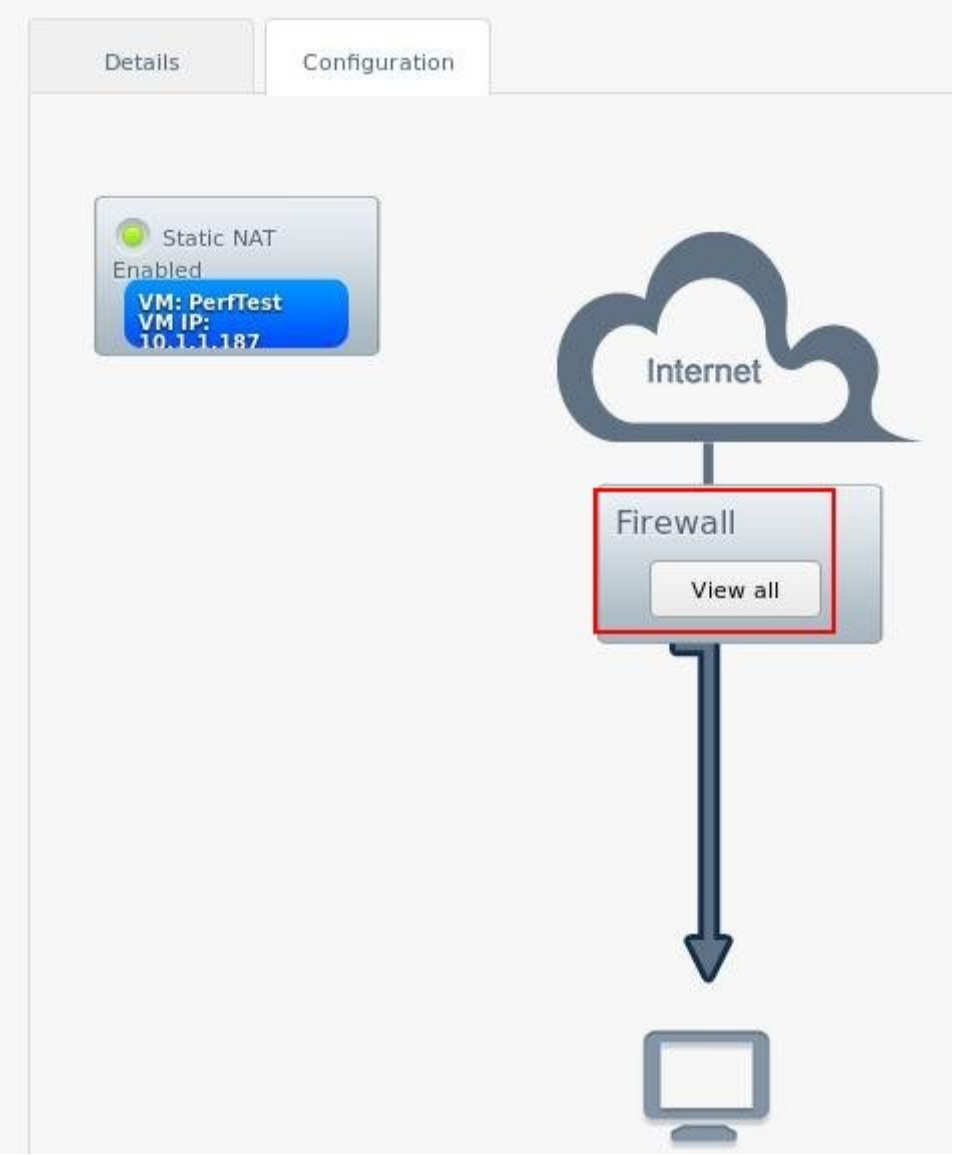


点击 `192.168.139.103` , 进入到配置:

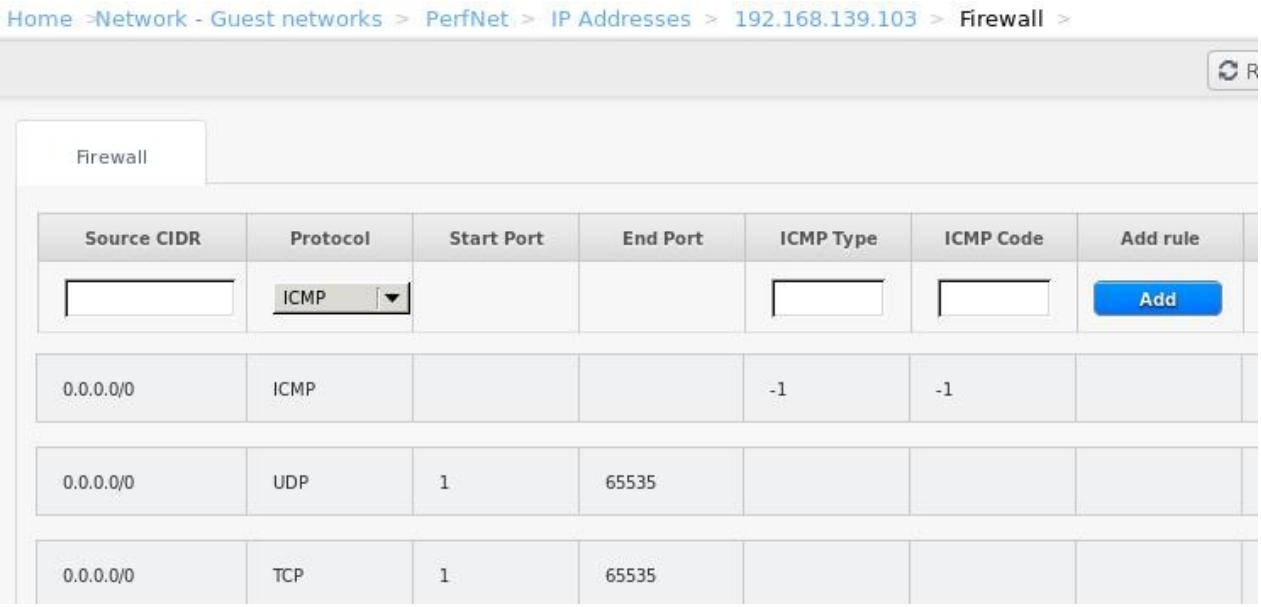
选择虚拟机后点击 `Apply`，设置完毕NAT:

接下来将自动提示是否刷新IP地址列表，点击 `Yes` 刷新之。

点击IP地址后，接着选择 `Configure`，看到如下画面:



点击上图中标注为红色的 `View All` 后配置防火墙:



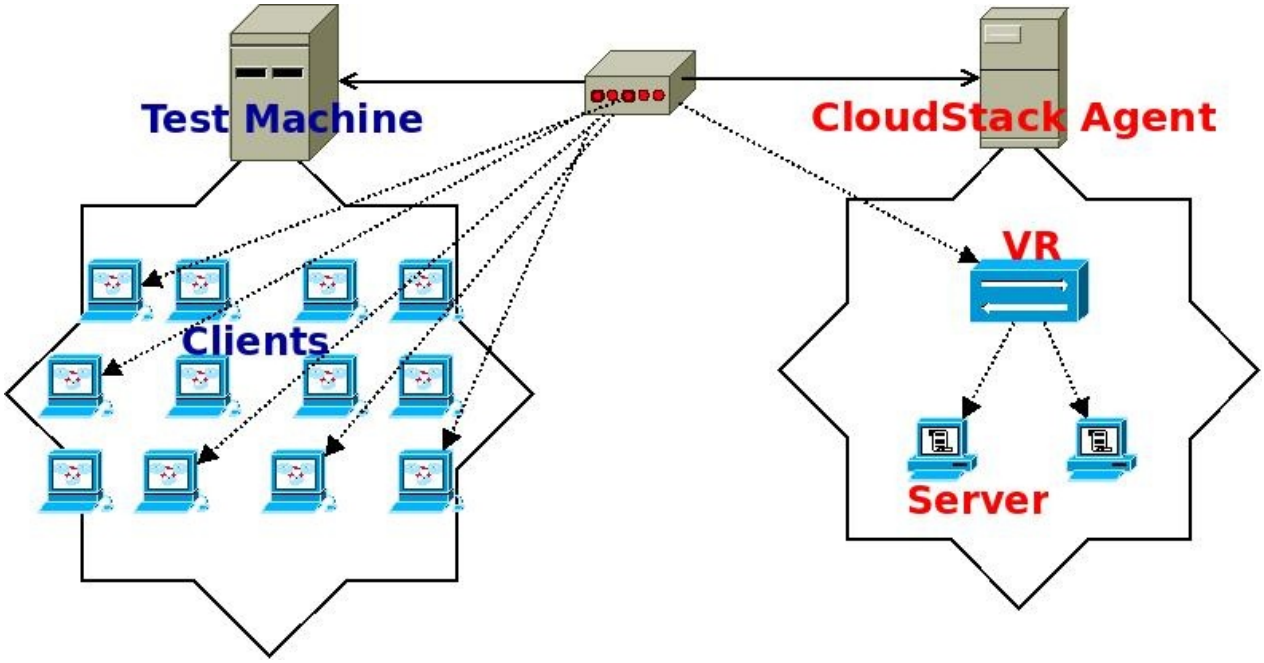
配置完毕后，在外网的机器上，可以通过ssh登录到 `192.168.139.103` ，即登录到我们刚才创建的

PerfTest实例了。

轰炸VR

环境准备

网络拓扑如下:



Test Machine:
硬件: I7 2620M 4核8线程， 12G内存.
操作系统: ArchLinux.
配置: 10+ KVM虚拟机，桥接模式，地址范围分别为192.168.1.2 ~ 192.168.1.20.
物理机**IP**: 192.168.1.11

CloudStack Agent:
硬件: i5-4460 4核4线程，32G 内存.
操作系统: Ubuntu14.04.
配置: CloudStack All-In-One, 版本4.5.2.
物理机**IP**: 192.168.1.13

Clients/Server:
操作系统: Ubuntu14.04.
Server: KVM虚拟机,双核/4G内存, 通过VR提供NAT获得公网地址.
Clients: KVM虚拟机，单核/768M内存.
IP地址: 均使用192.168.1.1/24段地址.

网络:
设备: OpenWRT路由器，1000M局域网连接.
DHCP范围: 192.168.1.2 ~ 192.168.1.20.

测试原理

突破限制

系统默认对

文件句柄限制

Linux默认的单个进程打开的文件限制为1024, 可以通过 `ulimit -n` 来查看, 永久改变需要更改以 下文件, 在文件最后添加两行:

```
# vim /etc/security/limits.conf
* soft nofile 1048576
* hard nofile 1048576
```

手动调节:

```
$ ulimit -HSn 1048576
```

最大文件数限制

当连接条目超过500000的时候有可能出现 `[warn] socket: Too many open files in system` , 我们将修改以下变量, 来确保 `/proc/sys/fs/file-max` 的值突破这个限制:

```
$ sudo vim /etc/sysctl.conf
.....
fs.file-max=1048576
$ sudo sysctl -p
```

端口限制

Linux系统默认的端口配置为:

```
# cat /proc/sys/net/ipv4/ip_local_port_range
32768 61000
```

这导致在单机上我们的程序可以使用的地址范围只有三万多个，更改如下，获得六万多个端口范围，从而可 以对外发起六万个连接。

```
# echo "1024 65535"> /proc/sys/net/ipv4/ip_local_port_range
```

永久更改方法:

```
# vim /etc/sysctl.conf
    net.ipv4.ip_local_port_range= 1024 65535
# sysctl -p
```

nf_conntrack限制

Netfilter模块对系统能track的connection做了限制，默认为65535, 查看如下:

```
$ sudo sysctl -a | grep -i conntrack_max
net.netfilter.nf_conntrack_max = 65536
net.nf_conntrack_max = 65536
```

Ubuntu 14.04/ArchLinux下更改如下:

```
# vim /etc/sysctl.d/99-sysctl.conf
net.netfilter.nf_conntrack_max = 1048576
# sysctl --system
```

或者(对Ubuntu14.04生效):

```
# vim /etc/sysctl.conf
net.netfilter.nf_conntrack_max = 6553500
# sysctl -p /etc/sysctl.conf
# sysctl -a | grep -i conntrack_max
net.netfilter.nf_conntrack_max = 6553500
net.nf_conntrack_max = 6553500
```

CloudStack Agent机器配置

Test Machine配置

单机多网卡

由下面的命令启动qemu实例,则可以得到8块网卡的虚拟机,每个上面6万多个连接,可以超 过50万.

```
$ sudo qemu-system-x86_64 -net nic,model=virtio,macaddr=52:54:00:12:34:56,vlan=1
-net tap,vlan=1 -net nic,model=virtio,macaddr=52:54:00:12:34:57,vlan=2 -net
tap,vlan=2 -net nic,model=virtio,macaddr=52:54:00:12:34:58,vlan=3 -net
tap,vlan=3 -net nic,model=virtio,macaddr=52:54:00:12:34:59,vlan=4 -net
tap,vlan=4 -net nic,model=virtio,macaddr=52:54:00:12:34:60,vlan=5 -net
tap,vlan=5 -net nic,model=virtio,macaddr=52:54:00:12:34:61,vlan=6 -net
tap,vlan=6 -net nic,model=virtio,macaddr=52:54:00:12:34:62,vlan=7 -net
tap,vlan=7 -net nic,model=virtio,macaddr=52:54:00:12:34:63,vlan=8 -net
tap,vlan=8 -hda ./ubuntu64perftest.qcow2 -m 5120 --enable-kvm
```

启动虚拟机后,设置地址:

```
# cat ./ethernet.sh
ifconfig eth1 up
ifconfig eth1 192.168.1.254
ifconfig eth2 up
ifconfig eth2 192.168.1.253
ifconfig eth3 up
ifconfig eth3 192.168.1.252
ifconfig eth4 up
ifconfig eth4 192.168.1.251
ifconfig eth5 up
ifconfig eth5 192.168.1.250
ifconfig eth6 up
ifconfig eth6 192.168.1.249
ifconfig eth7 up
ifconfig eth7 192.168.1.248
#@ifconfig eth8 up
#@ifconfig eth8 192.168.1.247
```

调用客户端大量连接的脚本:

```
 cat startbomb.sh
#!/bin/sh
./client2 -h 192.168.1.109 -p 8000 -m 64000 -o  192.168.1.16,192.168.1.254,192.168.1.253
```

`client2.c` 下载地址为

https://gist.github.com/yongboy/5324779/raw/f29c964fcd67fefc3ce66e487a44298ced611cdc/client2.c

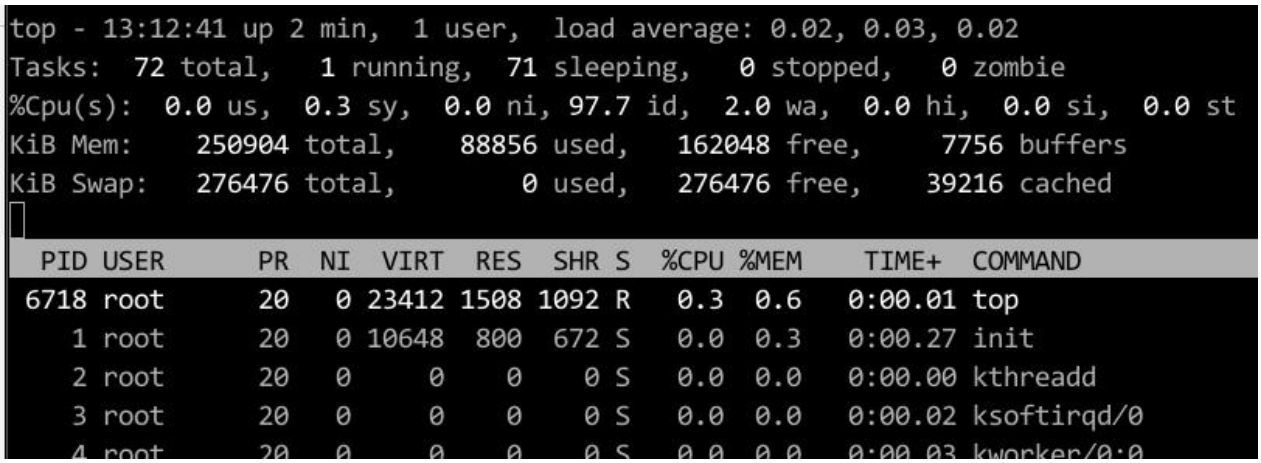
`client2.c` 中需要修改的行:

```
static char ip_array[300] =
"192.168.1.16,192.168.1.254,192.168.1.253,192.168.1.252,192.168.1.251,192.168.1.250,192.
//"192.168.190.134,192.168.190.143,192.168.190.144,192.168.190.145,192.168.190.146,192.1
static char server_ip[16] =  "192.168.1.109";
```

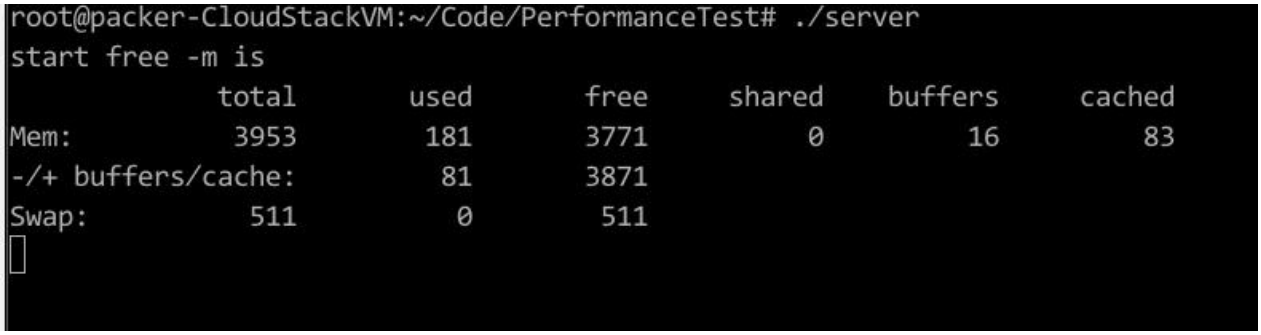
轰炸开始

在CloudStack的实例上运行server端程序, 此时的输出如下:

VR的 `top` 输出:

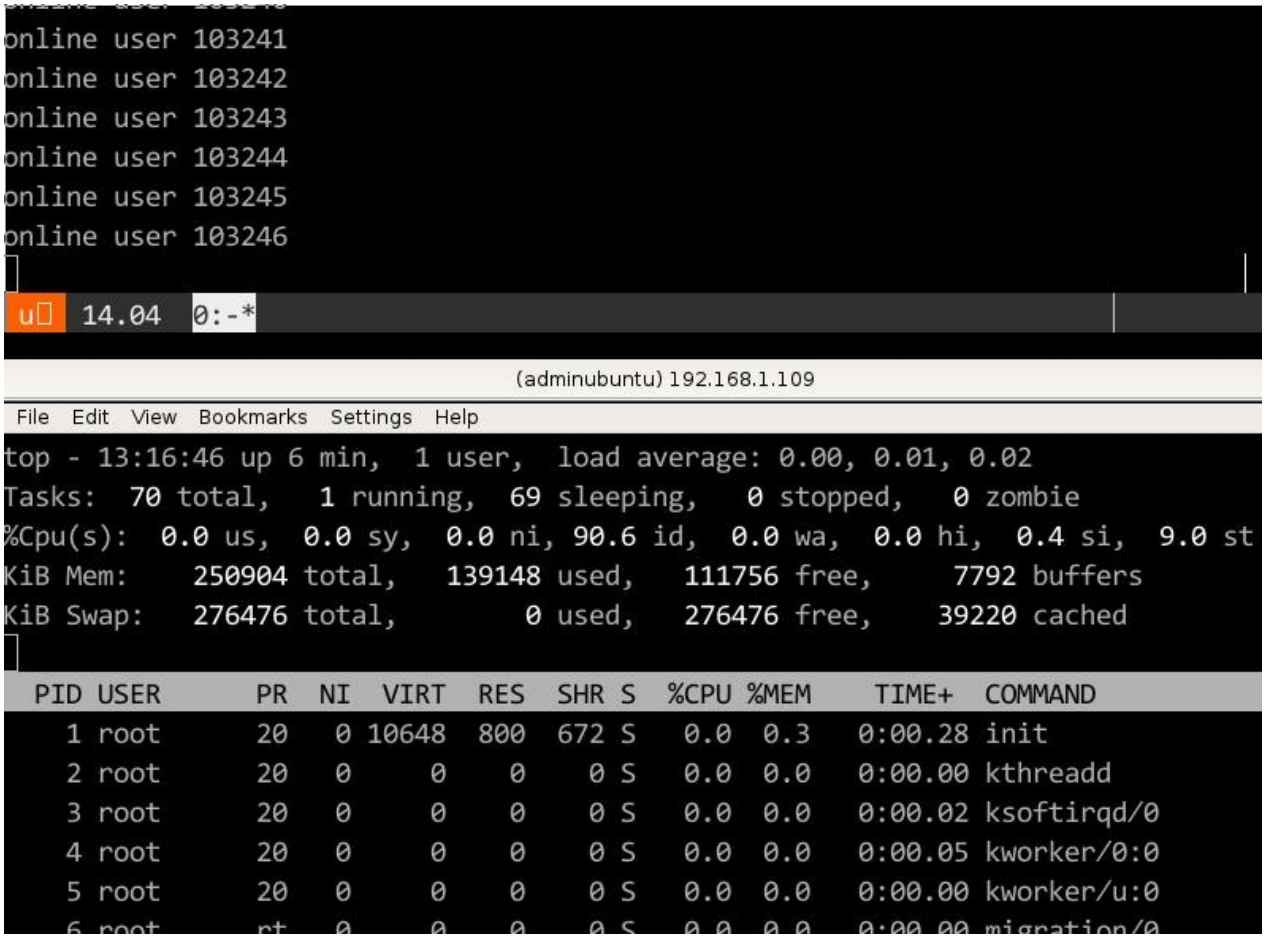


Server输出:

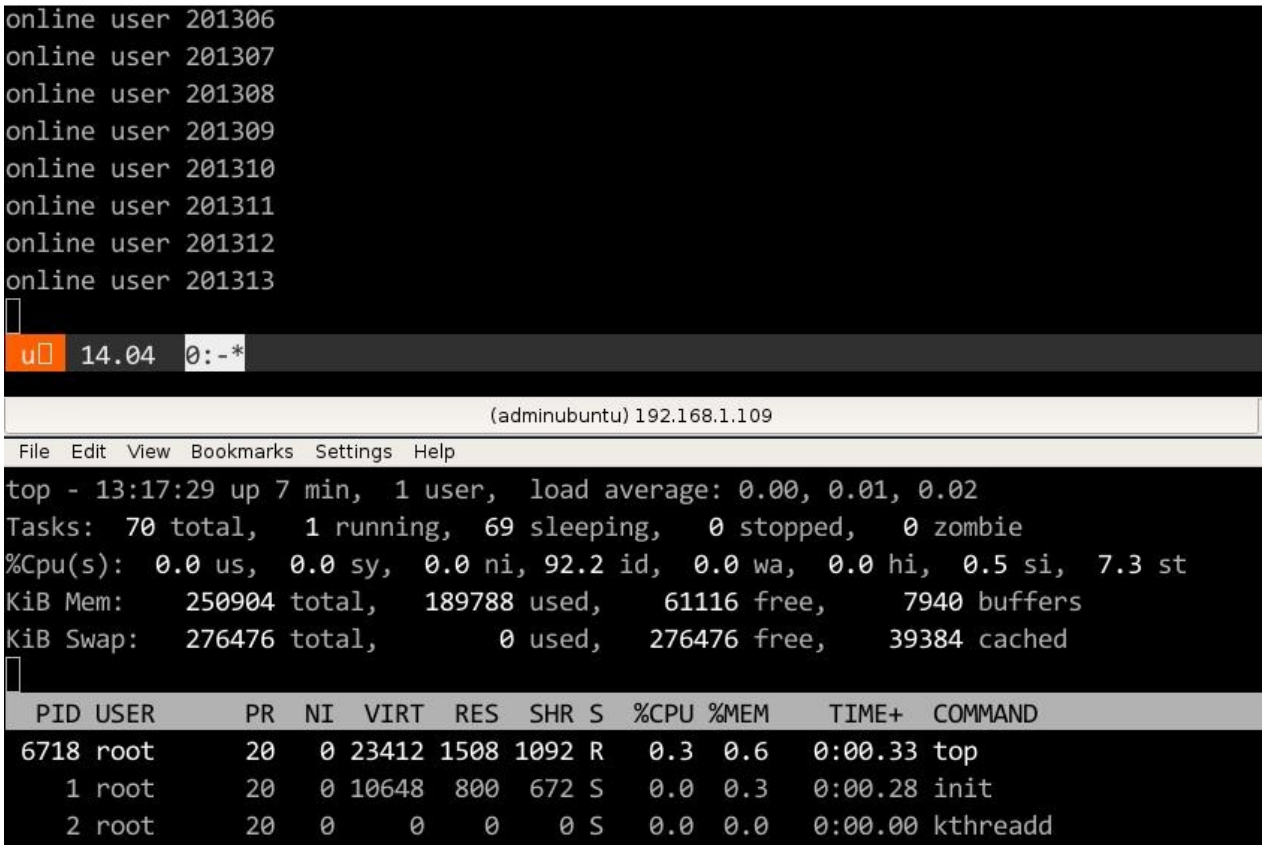


客户端开始发包连接的过程中:

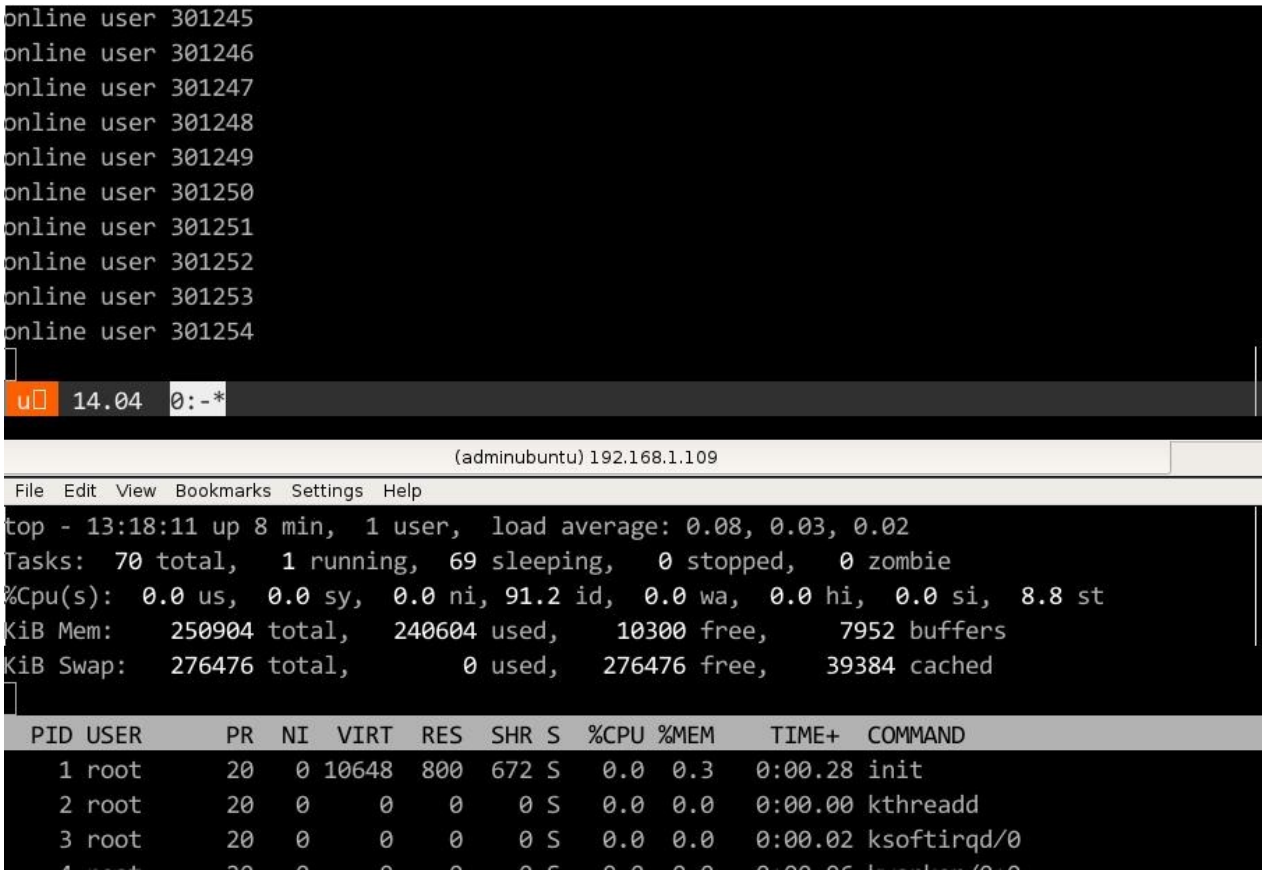
10W:


online user 103241
online user 103242
online user 103243
online user 103244
online user 103245
online user 103246
14.04 0:-*
(adminubuntu) 192.168.1.109
File Edit View Bookmarks Settings Help
top - 13:16:46 up 6 min, 1 user, load average: 0.00, 0.01, 0.02
Tasks: 70 total, 1 running, 69 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 90.6 id, 0.0 wa, 0.0 hi, 0.4 si, 9.0 st
KiB Mem: 250904 total, 139148 used, 111756 free, 7792 buffers
KiB Swap: 276476 total, 0 used, 276476 free, 39220 cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 20 0 10648 800 672 S 0.0 0.3 0:00.28 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:00.02 ksoftirqd/0
4 root 20 0 0 0 0 S 0.0 0.0 0:00.05 kworker/0:0
5 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kworker/u:0

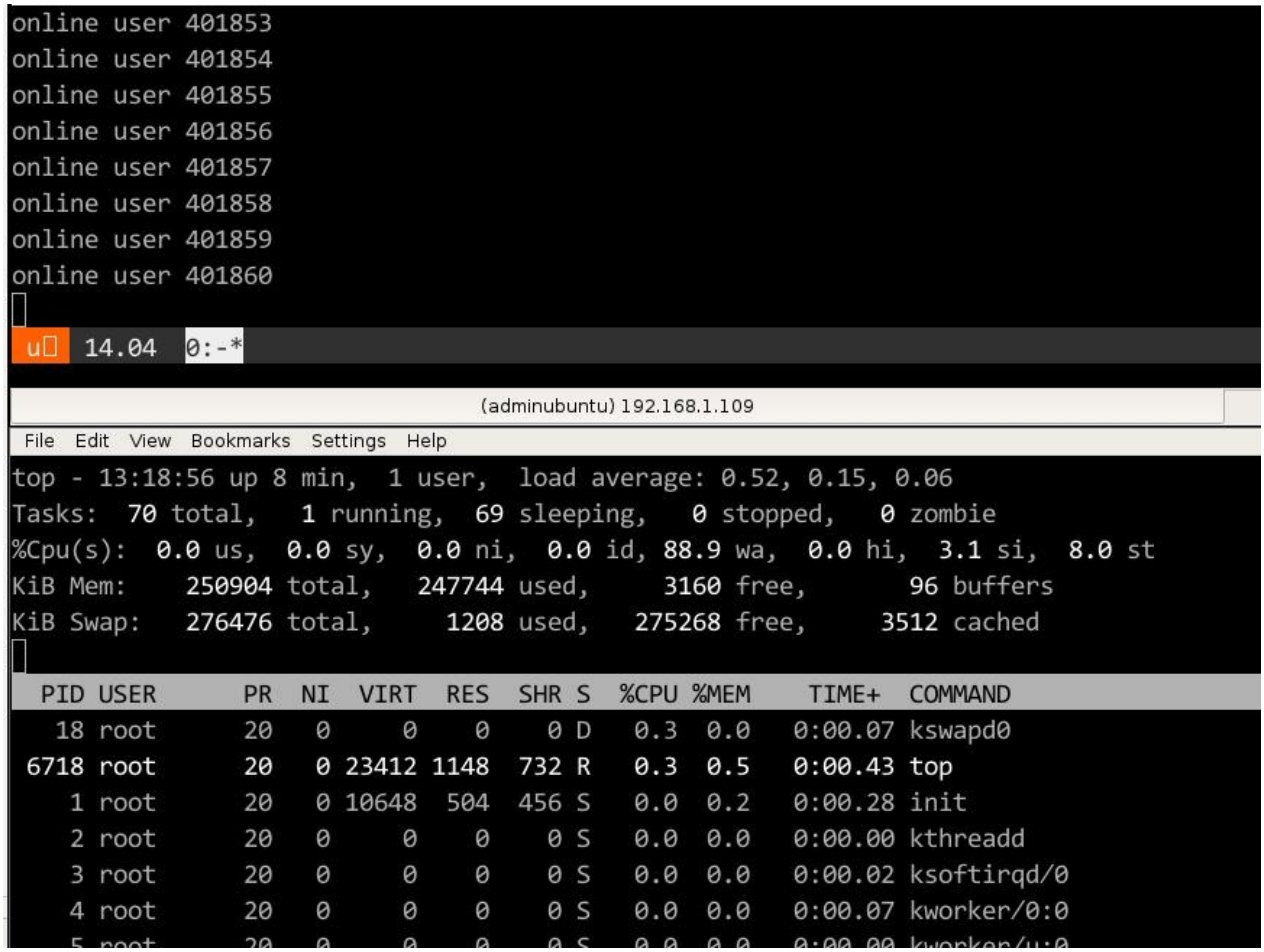
20W:


online user 201306
online user 201307
online user 201308
online user 201309
online user 201310
online user 201311
online user 201312
online user 201313
14.04 0:-*
(adminubuntu) 192.168.1.109
File Edit View Bookmarks Settings Help
top - 13:17:29 up 7 min, 1 user, load average: 0.00, 0.01, 0.02
Tasks: 70 total, 1 running, 69 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 92.2 id, 0.0 wa, 0.0 hi, 0.5 si, 7.3 st
KiB Mem: 250904 total, 189788 used, 61116 free, 7940 buffers
KiB Swap: 276476 total, 0 used, 276476 free, 39384 cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
6718 root 20 0 23412 1508 1092 R 0.3 0.6 0:00.33 top
1 root 20 0 10648 800 672 S 0.0 0.3 0:00.28 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd

30W:


online user 301245
online user 301246
online user 301247
online user 301248
online user 301249
online user 301250
online user 301251
online user 301252
online user 301253
online user 301254
14.04 0:-*
(adminubuntu) 192.168.1.109
File Edit View Bookmarks Settings Help
top - 13:18:11 up 8 min, 1 user, load average: 0.08, 0.03, 0.02
Tasks: 70 total, 1 running, 69 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 91.2 id, 0.0 wa, 0.0 hi, 0.0 si, 8.8 st
KiB Mem: 250904 total, 240604 used, 10300 free, 7952 buffers
KiB Swap: 276476 total, 0 used, 276476 free, 39384 cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 20 0 10648 800 672 S 0.0 0.3 0:00.28 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:00.02 ksoftirqd/0

40W:


online user 401853
online user 401854
online user 401855
online user 401856
online user 401857
online user 401858
online user 401859
online user 401860
u 14.04 0:-*
(adminubuntu) 192.168.1.109
File Edit View Bookmarks Settings Help
top - 13:18:56 up 8 min, 1 user, load average: 0.52, 0.15, 0.06
Tasks: 70 total, 1 running, 69 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 0.0 id, 88.9 wa, 0.0 hi, 3.1 si, 8.0 st
KiB Mem: 250904 total, 247744 used, 3160 free, 96 buffers
KiB Swap: 276476 total, 1208 used, 275268 free, 3512 cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
18 root 20 0 0 0 0 D 0.3 0.0 0:00.07 kswapd0
6718 root 20 0 23412 1148 732 R 0.3 0.5 0:00.43 top
1 root 20 0 10648 504 456 S 0.0 0.2 0:00.28 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:00.02 ksoftirqd/0
4 root 20 0 0 0 0 S 0.0 0.0 0:00.07 kworker/0:0

44W, VR轰挂:

```
online user 440016
online user 440017
online user 440018
online user 440019
online user 440020
online user 440021
online user 440022
online user 440023
online user 440024
online user 440025
```

```
0:-*
```

```
(adminubuntu) 192.168.1.109
File  Edit  View  Bookmarks  Settings  Help
top - 13:19:11 up 9 min,  1 user,  load average: 0.55, 0.18, 0.07
Tasks:  70 total,   1 running,  69 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.6 sy,  0.0 ni,  0.0 id, 85.4 wa,  0.0 hi,  3.7 si, 10.4 st
KiB Mem:    250904 total,   247864 used,     3040 free,       96 buffers
KiB Swap:   276476 total,    16924 used,   259552 free,     1436 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
   18 root      20   0     0    0    0 D   0.3  0.0   0:00.12 kswapd0
 6710 root      20   0 69184   96   56 S   0.3  0.0   0:00.22 sshd
 6718 root      20   0 23412  364  232 R   0.3  0.1   0:00.45 top
    1 root      20   0 10648   64   44 S   0.0  0.0   0:00.28 init
    2 root      20   0     0    0    0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S   0.0  0.0   0:00.02 ksoftirqd/0
```

此时查看VR的重启时间:

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@r-4-VM:~# uptime
 13:20:27 up 1 min,  1 user,  load average: 0.14, 0.06, 0.02
root@r-4-VM:~#
```

All-In-One VR性能测试

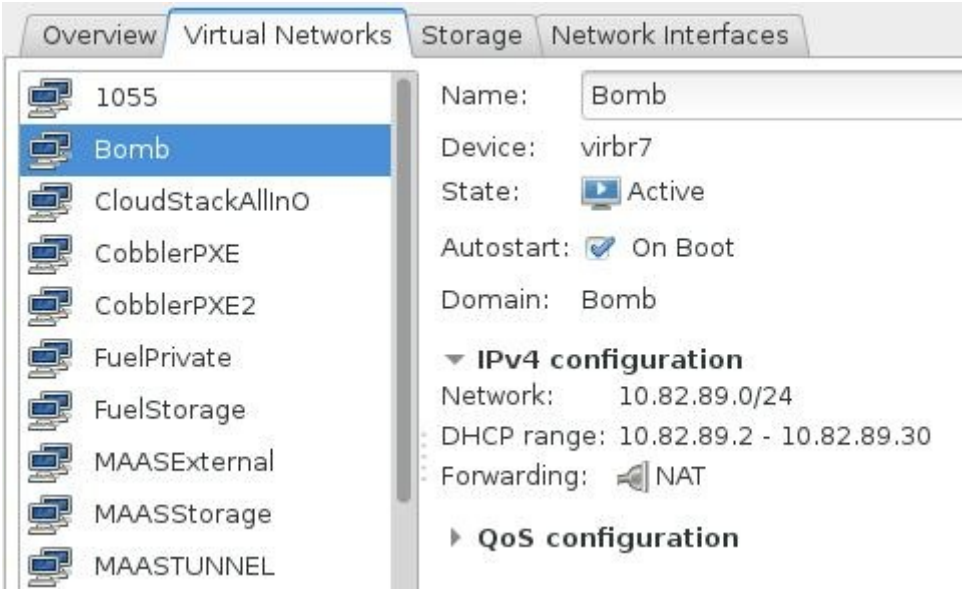
测试环境

硬件

CPU: Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 四核八线程。
内存: 24 G。

虚拟机网络

虚拟机网络: 新建网段为10.82.89.0/24, DHCP范围设置如下，NAT转发外网:



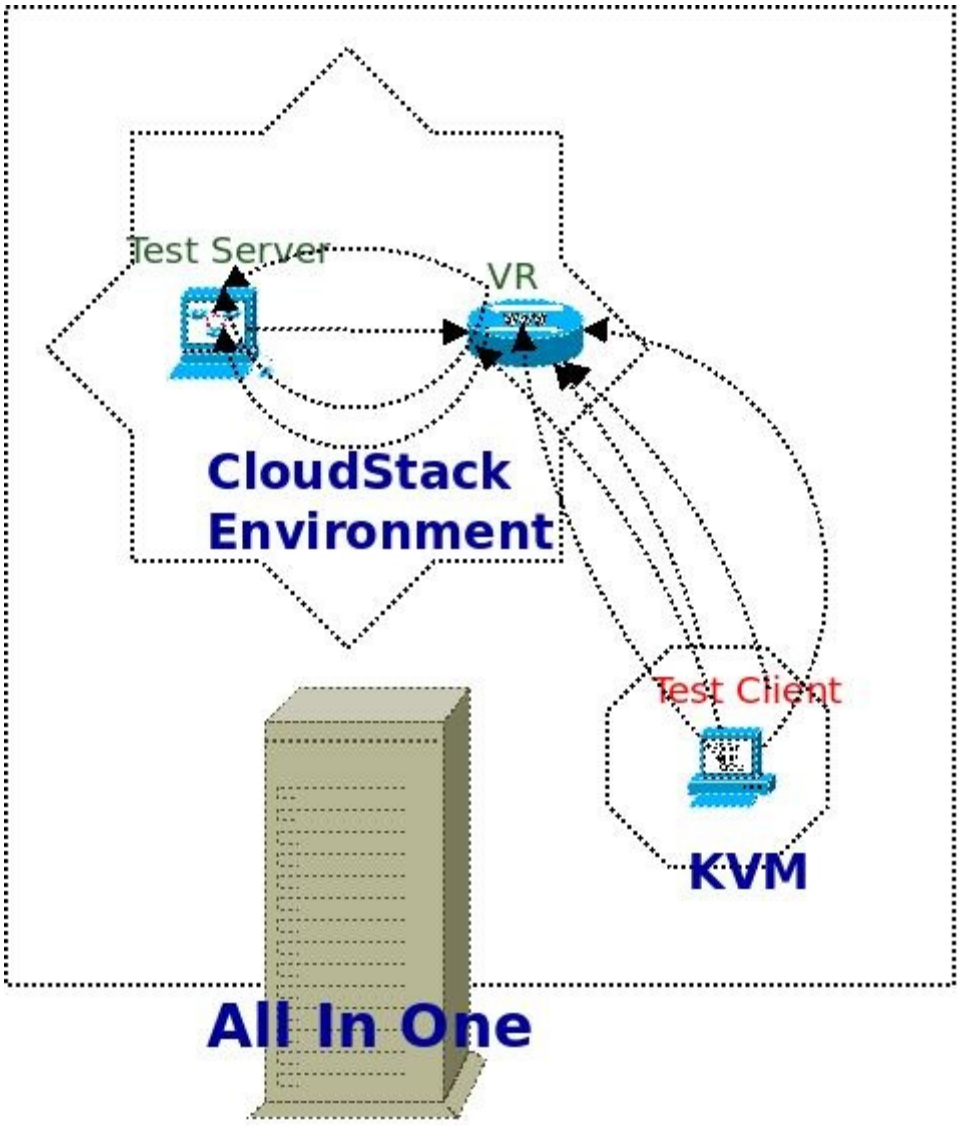
All-In-One 虚拟机

CPU: 6 Core, Copy Host CPU Configuration。
内存: 8 G。 硬盘: 200 G。 系统: Ubuntu 14.04 x86_64。
CloudStack版本: 4.5.2。

CloudStack环境

Zone: Advance Zone(PerfZone)。
IP: 取10.82.89.0/24中DHCP尚未使用的IP地址。
测试机模板: ubuntu64perftest.qcow2(http://192.168.0.79/ubuntu64perftest.qcow2)
测试机套餐: 2 Core, 内存3G。

拓扑图及说明



IP地址说明:

All-In-One: 10.82.89.89
Test Server: CloudStack实例，NAT后的10.82.89.0/24地址
VR: CloudStack自动分配的10.82.89.0/24地址
KVM: All-In-One上运行的虚拟机，绑定8个IP地址, 10.82.89.11/10.82.89.254~10.82.89.247

测试方法

Test Server上运行一个可支持多用户长时间在线的服务端程序，并可统计同时在线人数，接受外部 客户端发来的服务请求。

Test Client上运行一个快速发起并保持连接的客户端程序，服务端地址指向Test Server, 服务端 口即Test Server上服务端程序监听的端口。

单个IP最多支持6万多个活动连接，为了提升单台机器能支持的同时在线人数，需要在Test Client 上同时绑定多个网卡，kvm最多支持8个网卡，60000*8=500000, 单台机可以发起并保持的连接数超 过50万。

Server端代码下载地址:
https://gist.github.com/yongboy/5318930/raw/ccf8dc236da30fcf4f89567d567eaf295b363d47/server.c

Client端代码下载地址:

https://gist.github.com/yongboy/5324779/raw/f29c964fcd67fefc3ce66e487a44298ced611cdc/client2.c

打开各节点限制

在All-In-One机器/Test Server/Test Client上，均需要打开以下限制:

最大文件句柄数

```
# vim /etc/security/limits.conf

* soft nofile 104857
* hard nofile 104857
```

最大文件数限制

```
# vim /etc/sysctl.conf
fs.file-max=1048576
# sudo sysctl -p /etc/sysctl.conf
```

conntrack最大连接数

```
# vim /etc/sysctl.conf
net.netfilter.nf_conntrack_max = 6553500
# sysctl -p /etc/sysctl.conf
```

可用端口数

```
# vim /etc/sysctl.conf
    net.ipv4.ip_local_port_range= 1024 65535
# sysctl -p /etc/sysctl.conf
```

更改完毕后，需要重新启动机器生效，为了使sysctl的配置每次都生效，可以考虑将命令加到启动 项中。

测试过程

启动**Server**端：

Server端启动后将监听服务器端的8000端口：

```
root@packer-CloudStackVM:~/Code/PerformanceTest# ./server
start free -m is
             total       used       free     shared    buffers     cached
Mem:          3009        188       2821          0         17         89
-/+ buffers/cache:         81       2928
Swap:          511          0        511

```

启动**Client**端：

单机多网卡

在All-In-One机器上，由下面的命令启动qemu实例,则可以得到8块网卡的虚拟机, 内存大小`-m 5120`可以调小，一般1024～2048就够:

```
$ sudo qemu-system-x86_64 -net nic,model=virtio,macaddr=52:54:00:12:34:56,vlan=1
-net tap,vlan=1 -net nic,model=virtio,macaddr=52:54:00:12:34:57,vlan=2 -net
tap,vlan=2 -net nic,model=virtio,macaddr=52:54:00:12:34:58,vlan=3 -net
tap,vlan=3 -net nic,model=virtio,macaddr=52:54:00:12:34:59,vlan=4 -net
tap,vlan=4 -net nic,model=virtio,macaddr=52:54:00:12:34:60,vlan=5 -net
tap,vlan=5 -net nic,model=virtio,macaddr=52:54:00:12:34:61,vlan=6 -net
tap,vlan=6 -net nic,model=virtio,macaddr=52:54:00:12:34:62,vlan=7 -net
tap,vlan=7 -net nic,model=virtio,macaddr=52:54:00:12:34:63,vlan=8 -net
tap,vlan=8 -hda ./ubuntu64perftest.qcow2 -m 5120 --enable-kvm
```

启动虚拟机后, 在虚拟机里设置地址(默认只从eth0得到dhcp地址):

```
# cat ./ethernet.sh
ifconfig eth1 up
ifconfig eth1 192.168.1.254
ifconfig eth2 up
ifconfig eth2 192.168.1.253
ifconfig eth3 up
ifconfig eth3 192.168.1.252
ifconfig eth4 up
ifconfig eth4 192.168.1.251
ifconfig eth5 up
ifconfig eth5 192.168.1.250
ifconfig eth6 up
ifconfig eth6 192.168.1.249
ifconfig eth7 up
ifconfig eth7 192.168.1.248
```

客户端中，发起海量连接的脚本:

```
 cat startbomb.sh
#!/bin/sh
```

```
./client2 -h 192.168.1.109 -p 8000 -m 64000 -o
192.168.1.16,192.168.1.254,192.168.1.253,192.168.1.252,
192.168.1.251,192.168.1.250,192.168.1.249,192.168.1.248
```

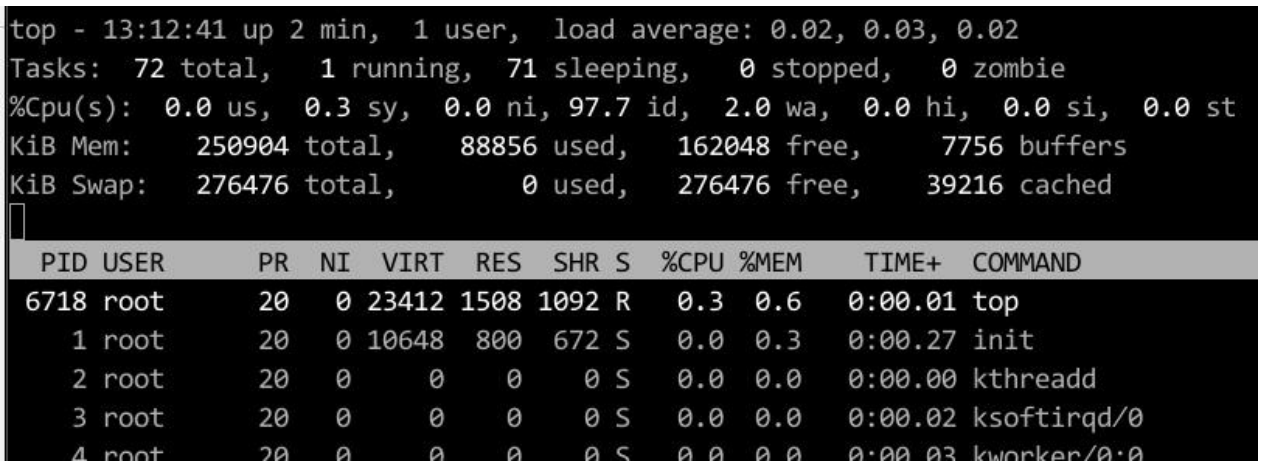
调用 `./startbomb.sh` 即可开始对Server端发起大量在线连接

测试截图

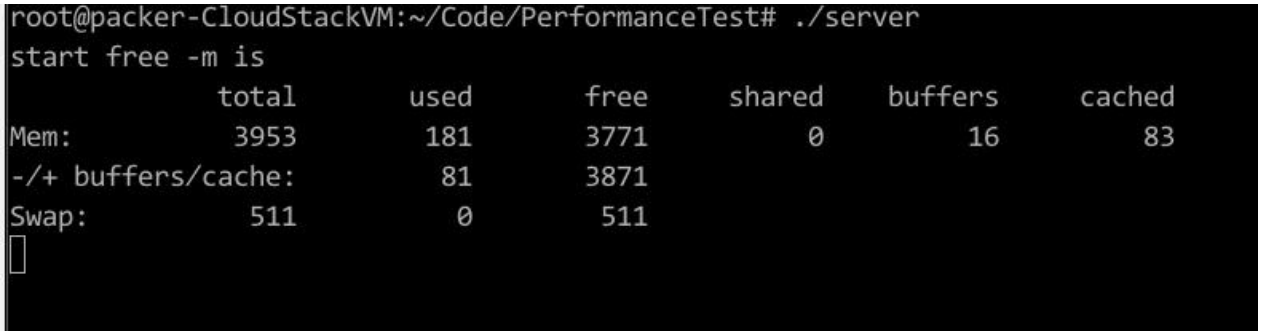
这里记录一次完整的VR因为过多连接数造成内存耗尽自动重启的过程。

在CloudStack的实例上运行server端程序, 此时的输出如下:

VR的 `top` 输出:

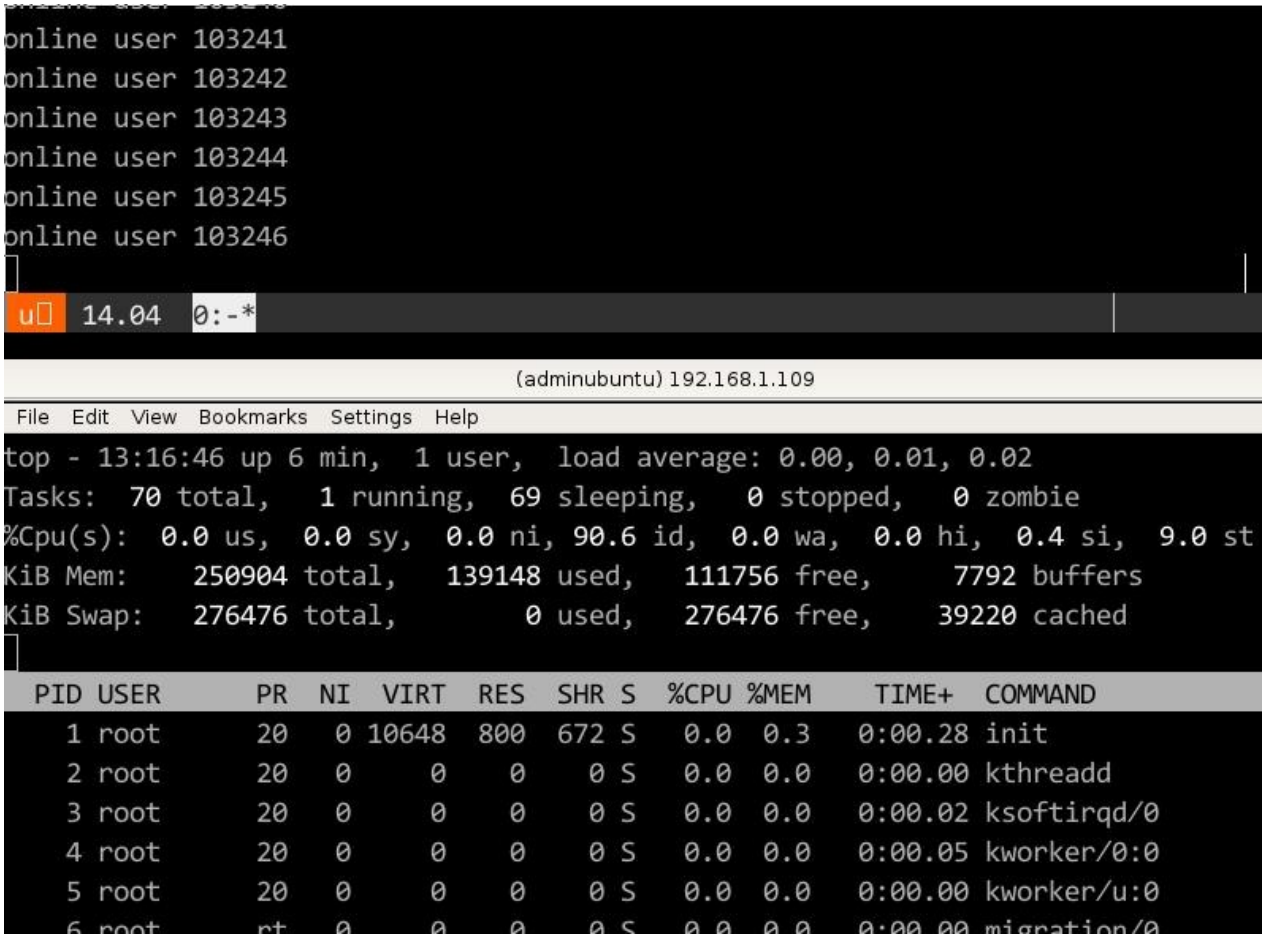


Server输出:

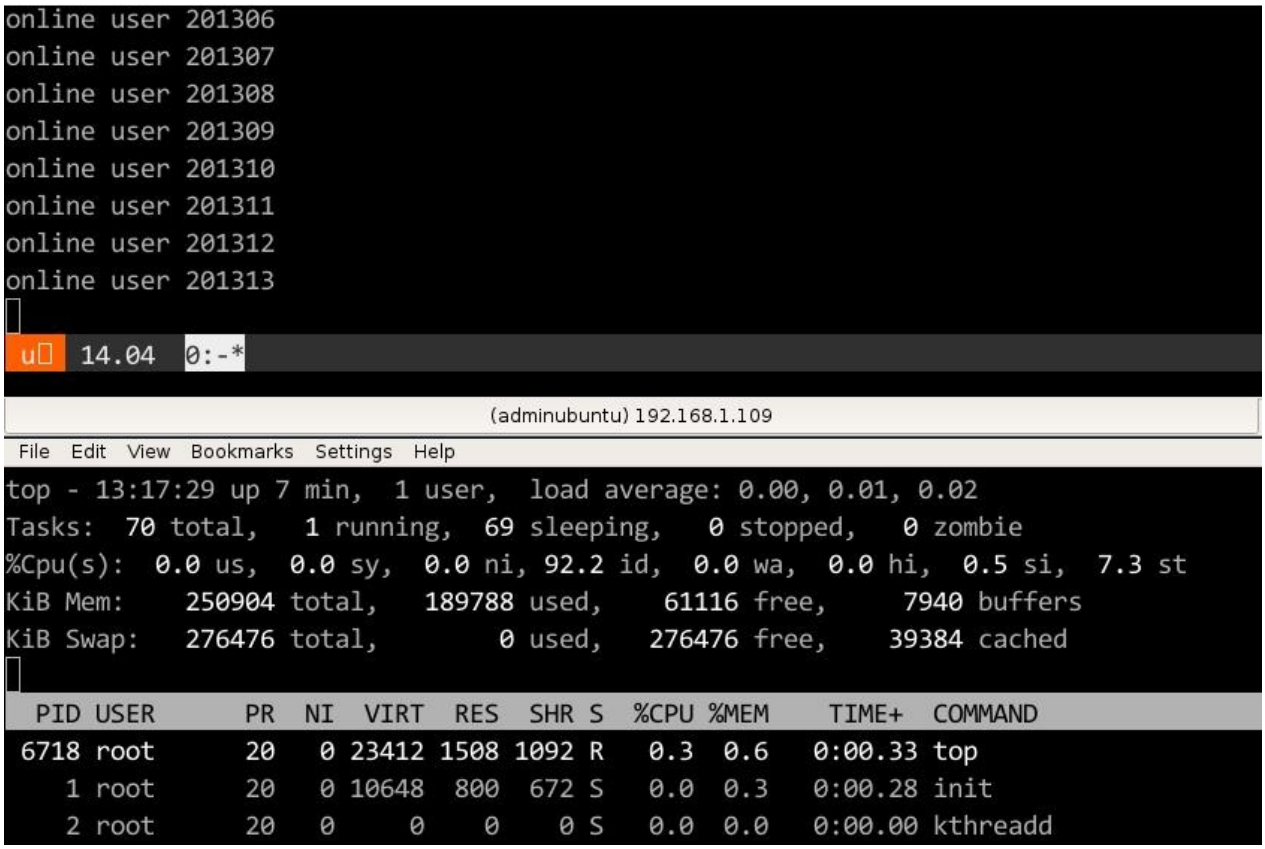


客户端开始发包连接的过程中:

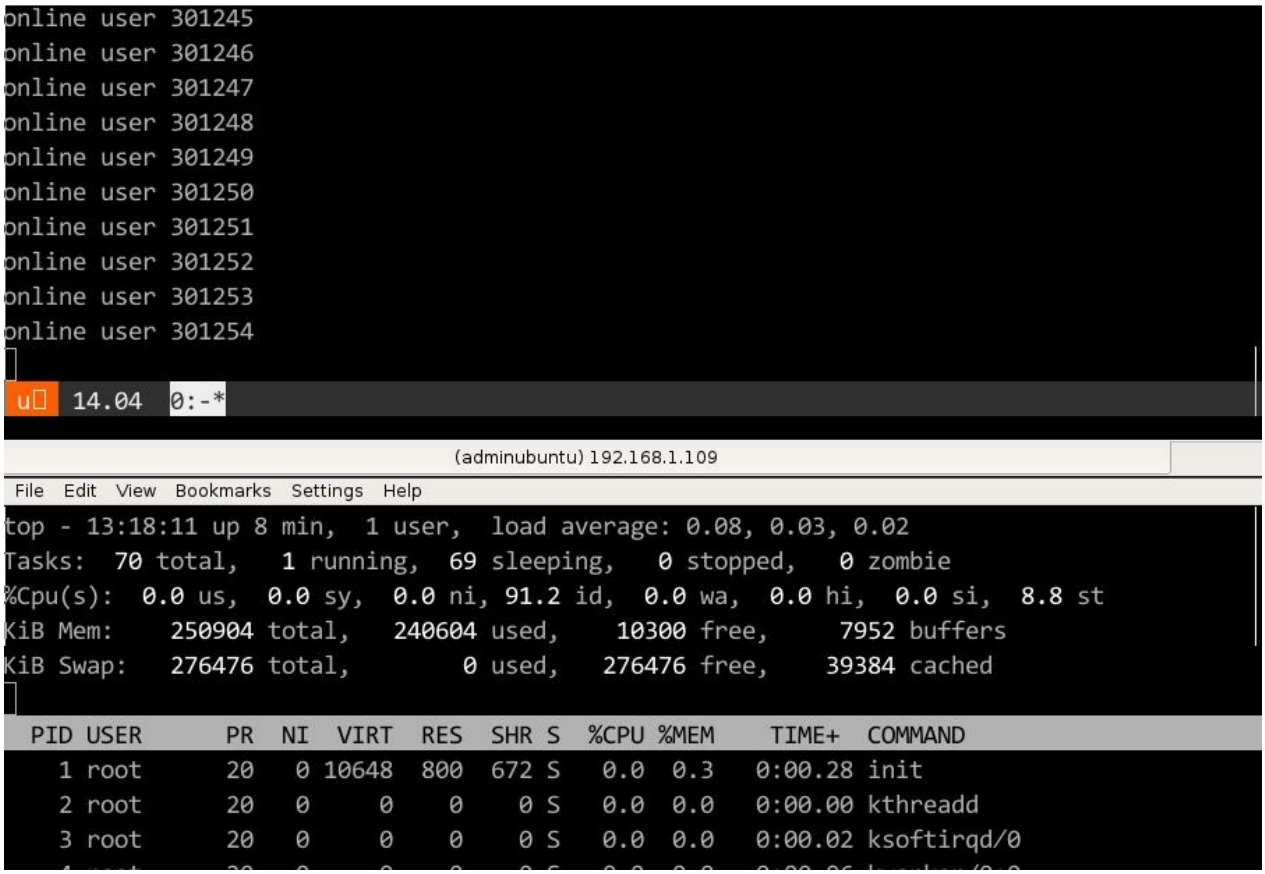
10W:


online user 103241
online user 103242
online user 103243
online user 103244
online user 103245
online user 103246
14.04 0:-*
(adminubuntu) 192.168.1.109
File Edit View Bookmarks Settings Help
top - 13:16:46 up 6 min, 1 user, load average: 0.00, 0.01, 0.02
Tasks: 70 total, 1 running, 69 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 90.6 id, 0.0 wa, 0.0 hi, 0.4 si, 9.0 st
KiB Mem: 250904 total, 139148 used, 111756 free, 7792 buffers
KiB Swap: 276476 total, 0 used, 276476 free, 39220 cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 20 0 10648 800 672 S 0.0 0.3 0:00.28 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:00.02 ksoftirqd/0
4 root 20 0 0 0 0 S 0.0 0.0 0:00.05 kworker/0:0
5 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kworker/u:0

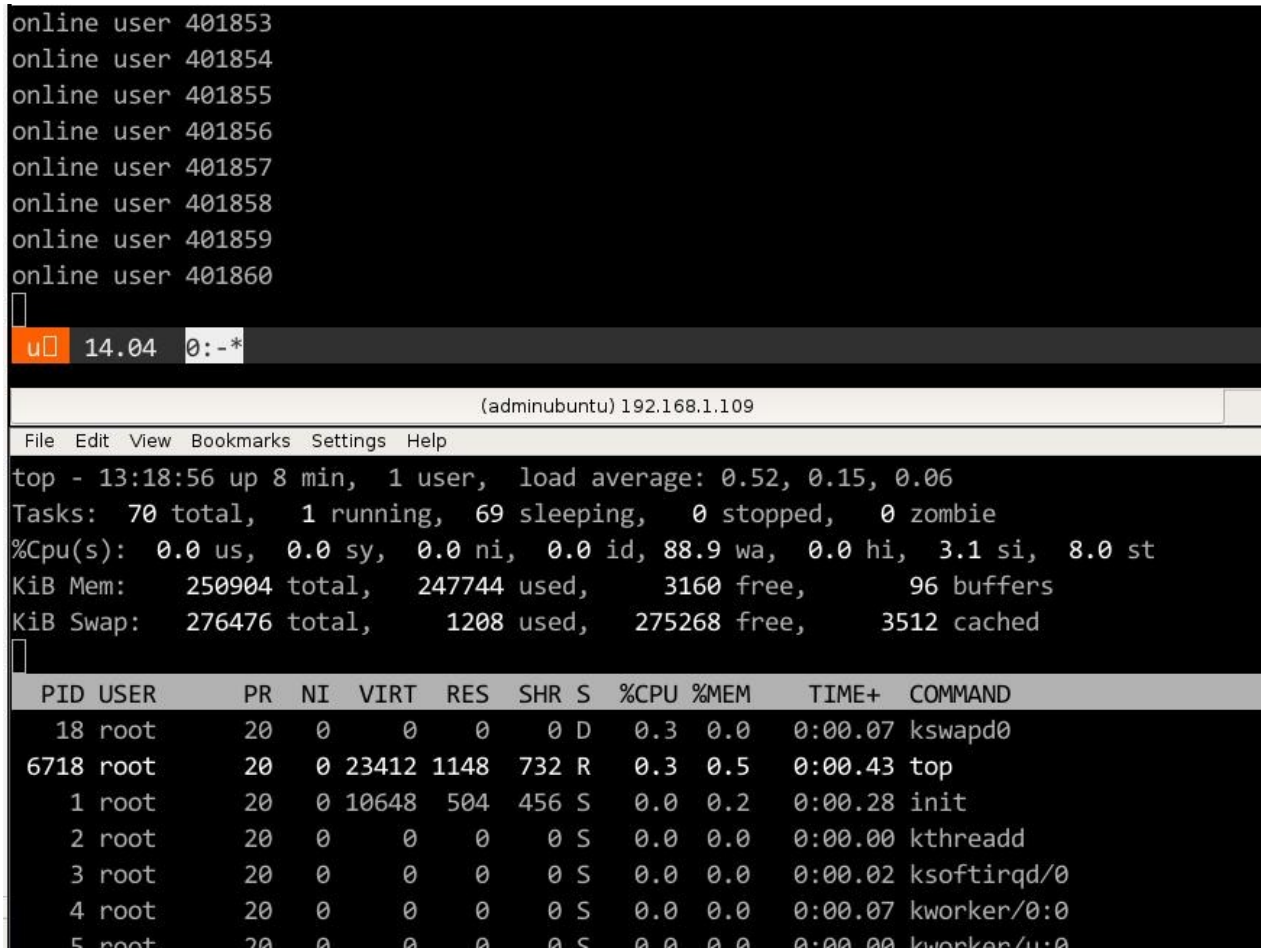
20W:


online user 201306
online user 201307
online user 201308
online user 201309
online user 201310
online user 201311
online user 201312
online user 201313
14.04 0:-*
(adminubuntu) 192.168.1.109
File Edit View Bookmarks Settings Help
top - 13:17:29 up 7 min, 1 user, load average: 0.00, 0.01, 0.02
Tasks: 70 total, 1 running, 69 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 92.2 id, 0.0 wa, 0.0 hi, 0.5 si, 7.3 st
KiB Mem: 250904 total, 189788 used, 61116 free, 7940 buffers
KiB Swap: 276476 total, 0 used, 276476 free, 39384 cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
6718 root 20 0 23412 1508 1092 R 0.3 0.6 0:00.33 top
1 root 20 0 10648 800 672 S 0.0 0.3 0:00.28 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd

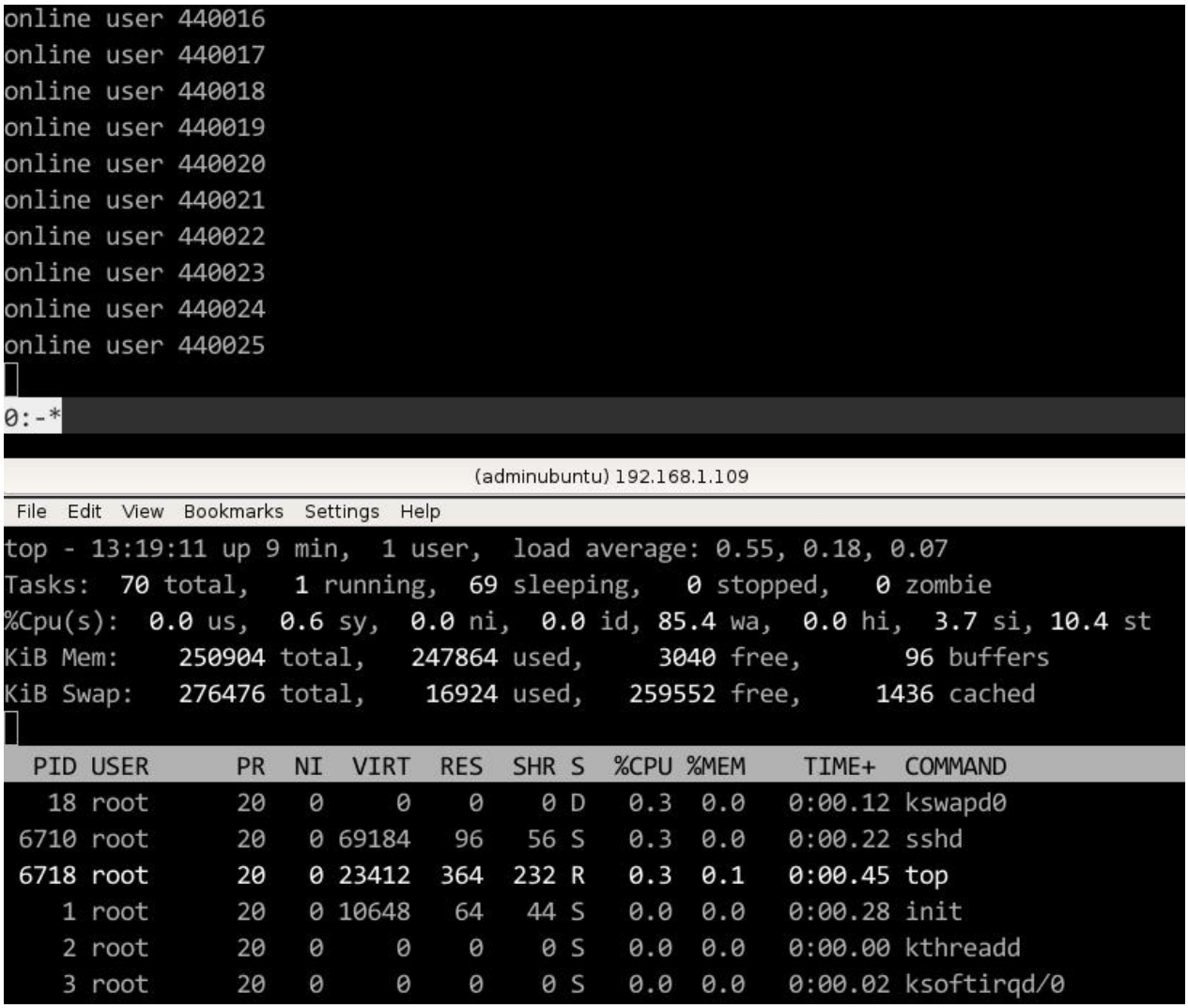
30W:


online user 301245
online user 301246
online user 301247
online user 301248
online user 301249
online user 301250
online user 301251
online user 301252
online user 301253
online user 301254
14.04 0:-*
(adminubuntu) 192.168.1.109
File Edit View Bookmarks Settings Help
top - 13:18:11 up 8 min, 1 user, load average: 0.08, 0.03, 0.02
Tasks: 70 total, 1 running, 69 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 91.2 id, 0.0 wa, 0.0 hi, 0.0 si, 8.8 st
KiB Mem: 250904 total, 240604 used, 10300 free, 7952 buffers
KiB Swap: 276476 total, 0 used, 276476 free, 39384 cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 20 0 10648 800 672 S 0.0 0.3 0:00.28 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:00.02 ksoftirqd/0

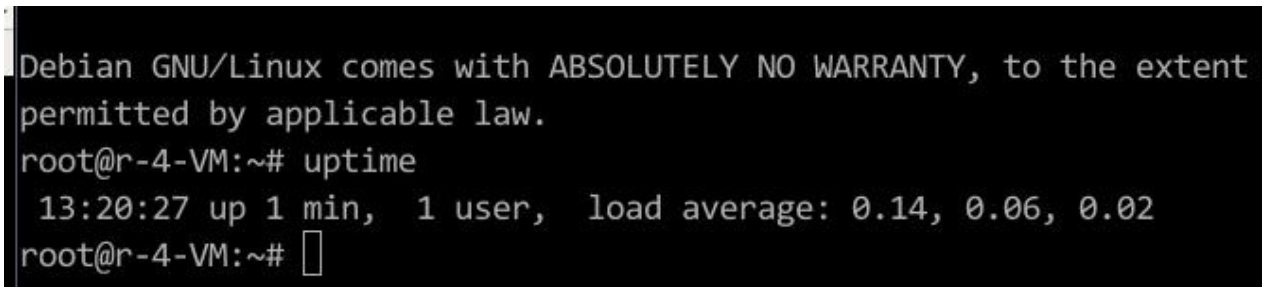
40W:


online user 401853
online user 401854
online user 401855
online user 401856
online user 401857
online user 401858
online user 401859
online user 401860
u 14.04 0:-*
(adminubuntu) 192.168.1.109
File Edit View Bookmarks Settings Help
top - 13:18:56 up 8 min, 1 user, load average: 0.52, 0.15, 0.06
Tasks: 70 total, 1 running, 69 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 0.0 id, 88.9 wa, 0.0 hi, 3.1 si, 8.0 st
KiB Mem: 250904 total, 247744 used, 3160 free, 96 buffers
KiB Swap: 276476 total, 1208 used, 275268 free, 3512 cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
18 root 20 0 0 0 0 D 0.3 0.0 0:00.07 kswapd0
6718 root 20 0 23412 1148 732 R 0.3 0.5 0:00.43 top
1 root 20 0 10648 504 456 S 0.0 0.2 0:00.28 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:00.02 ksoftirqd/0
4 root 20 0 0 0 0 S 0.0 0.0 0:00.07 kworker/0:0

44W, VR轰挂:



此时查看VR的重启时间:



可以看到VR启动时间很短，这证明VR由于内存耗尽已经自动重启了。

应对策略

暂时应对

在VR上对每个进/出 虚拟路由器的IP作连接数限制，TCP/UDP都需要设置。ICMP暂时未作限定:

Iptables规则如下:

```
-A PREROUTING -i eth0 -p tcp  -m connlimit --connlimit-above 1000 --connlimit-mask 32
--connlimit-saddr -j DROP
-A PREROUTING -i eth2 -p tcp  -m connlimit --connlimit-above 1000 --connlimit-mask 32
--connlimit-saddr -j DROP
-A PREROUTING -i eth0 -p udp  -m connlimit --connlimit-above 1000 --connlimit-mask 32
--connlimit-saddr -j DROP
-A PREROUTING -i eth2 -p udp  -m connlimit --connlimit-above 1000 --connlimit-mask 32
--connlimit-saddr -j DROP
```

添加完此规则后，Test Server最多可接受1000个同时在线连接。这时在Client端发起大量连接超过1000的都会被VR的iptables规则链丢弃。

潜在问题

对于过多过快的攻击性连接，VR匹配iptables会造成VR CPU占用率过高。建议采用硬件防火墙来阻挡这类 攻击。

部署CloudStack-Ubuntu

本章主要讲述如何在Ubuntu14.04上部署CloudStack，所有节点均在同一台机器上，All in One 模式。

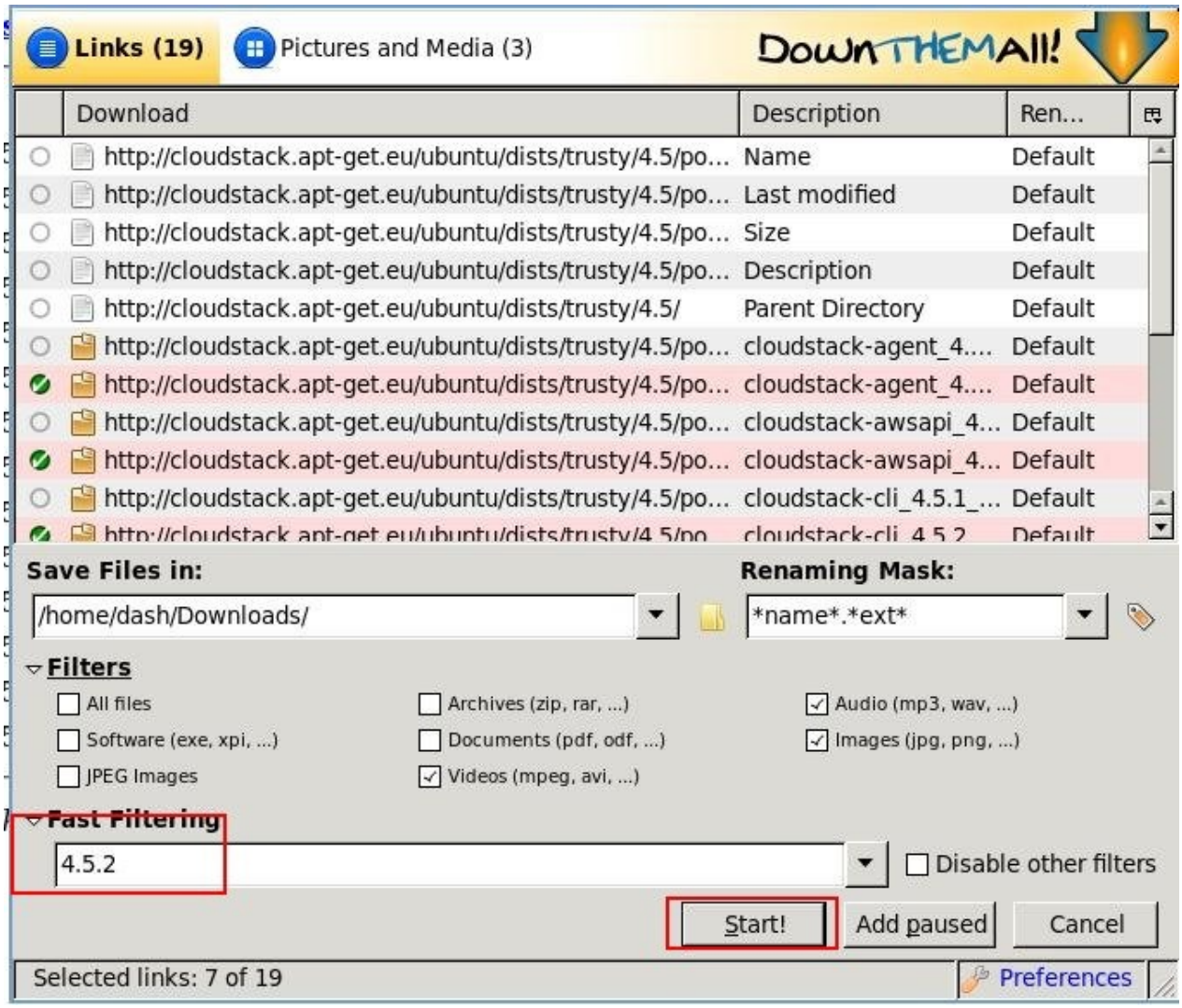
准备包

我们首先把CloudStack的DEB安装包下载到本地，构建出本地仓库，以便安装和部 署的流程。

批量下载包

可以用Firefox的Downthemall插件，批量从下面地址下载DEB包:

http://cloudstack.apt-get.eu/ubuntu/dists/trusty/4.5/pool/



构建仓库

在Debian系机器机器上，安装 `dpkg-dev` 包.

```
$ ls cloudstackdeb
cloudstack-agent_4.5.2_all.deb   cloudstack-cli_4.5.2_all.deb
cloudstack-docs_4.5.2_all.deb        cloudstack-usage_4.5.2_all.deb
cloudstack-awsapi_4.5.2_all.deb  cloudstack-common_4.5.2_all.deb
cloudstack-management_4.5.2_all.deb
```

```
$ dpkg-scanpackages cloudstackdeb | gzip -9c >
cloudstackdeb/Packages.gz
dpkg-scanpackages: info: Wrote 7 entries to output Packages file.
```

现在把cloudstackdeb目录上传到Web服务器根目录下.

以后要使用该仓库时，只需要编辑sources.list文件如下:

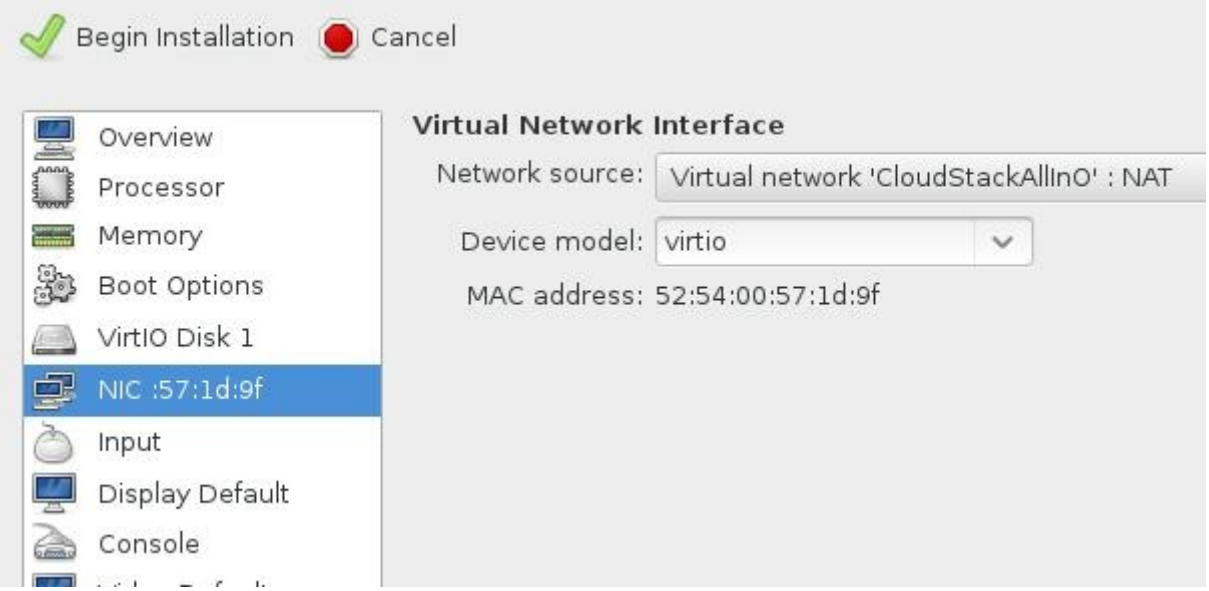
```
$ vim /etc/apt/sources.list
deb http://Your_Server_IP/    cloudstackdeb/
```

准备系统

我们使用Ubuntu 14.04来部署CloudStack。 以下是系统准备过程。

网络配置

首先把待配置的Ubuntu虚拟机加入到网络里:



配置IP地址为 `10.168.100.10` :

```
$ sudo vim /etc/network/interfaces
.....
# The primary network interface
auto eth0
iface eth0 inet static
address 10.168.100.10
netmask 255.255.255.0
gateway 10.168.100.1
dns-nameservers 223.5.5.5
```

主机名及**FQDN**配置

步骤如下:

```
# vim /etc/hostname
ubuntucloudstack
# vim /etc/hosts
....
- 127.0.1.1.    xxxxxx
+ 10.168.100.10    ubuntucloudstack
```

重启后检验:

```
adminubuntu@ubuntucloudstack:~$ hostname
ubuntucloudstack
adminubuntu@ubuntucloudstack:~$ hostname --fqdn
ubuntucloudstack
```

允许**ROOT**登录

首先sudo 到root用户更改其密码.

而后更改sshd的配置文件如下:

```
root@ubuntucloudstack:~# vim /etc/ssh/sshd_config
+ PermitRootLogin yes
root@ubuntucloudstack:~# service ssh restart
```

现在开始你可以用root用户名登录.

配置桥接网络

安装桥接包:

```
# apt-get install -y bridge-utils
```

配置网络:

```
# vim /etc/nework/intefaces
    # The primary network interface
    auto eth0
    iface eth0 inet manual

    auto cloudbr0
    iface cloudbr0 inet static
    address 10.168.100.10
    netmask 255.255.255.0
    gateway 10.168.100.1
    dns-nameservers 223.5.5.5
    bridge_ports eth0
    bridge_fd 5
    bridge_stp off
    bridge_maxwait 1
```

重启网络后可以看到cloudbr0被激活。

配置仓库

```
root@ubuntucloudstack:~# vim /etc/apt/sources.list
    ### Add cloudstack local repository
    deb http://192.168.0.79/    cloudstackdeb/

root@ubuntucloudstack:~# apt-cache search cloudstack
cloudstack-agent - CloudStack agent
cloudstack-awsapi - CloudStack Amazon EC2 API
cloudstack-cli - The CloudStack CLI called CloudMonkey
cloudstack-common - A common package which contains files which are shared by several Cl
cloudstack-docs - The CloudStack documentation
cloudstack-management - CloudStack server library
cloudstack-usage - CloudStack usage monitor
```

现在一切准备就绪，接下来将开始安装Cloudstack

安装CloudStack

openntpd

用于保持本机时间同步。

```
$ sudo apt-get install -y openntpd
```

cloudstack-management

因为我们使用了本地的源，所以需要加上`--force-yes`选项

```
# apt-get --yes install cloudstack-management --force-yes
```

mysql-server

安装my-sql数据库:

```
# apt-get install -y mysql-server
```

安装时需要制定root的密码，这里我们设置为xxxxxx, 这个值在下面会被用到.

配置数据库:

```
root@ubuntucloudstack:~# cat /etc/mysql/conf.d/cloudstack.cnf
[mysqld]
innodb_rollback_on_timeout=1
innodb_lock_wait_timeout=600
max_connections=350
log-bin=mysql-bin
binlog-format = 'ROW'
root@ubuntucloudstack:~# service mysql restart
mysql stop/waiting
mysql start/running, process 5812
```

创建数据库:

```
root@ubuntucloudstack:~# cloudstack-setup-databases \
    cloud:engine@localhost \
    --deploy-as=root:xxxxxx \
    -e file -m mymskey44 -k mydbkey00
```

参数说明:

```
# mysql root 密码: xxxxxxx
# cloud user 密码: engine
# management_server_key: mymskey44
# database_key: mydbkey00
```

准备NFS存储

首先准备NFS共享存储，创建主存储和二级存储:

```
$ sudo mkdir -p /export/primary /export/secondary
```

安装nfs:

```
$ sudo apt-get install nfs-kernel-server
```

引出/export目录:

```
$ cat >>/etc/exports <<EOM
/export  *(rw,async,no_root_squash,no_subtree_check)
EOM
$ exportfs -a
```

配置NFS的statd，在指定端口:

```
$ apt-get install nfs-common
$ cp /etc/default/nfs-common /etc/default/nfs-common.orig
$ sed -i '/NEED_STATD=/ a NEED_STATD=yes' /etc/default/nfs-common
$ sed -i '/STATDOPTS=/ a STATDOPTS="--port 662 --outgoing-port 2020"'
/etc/default/nfs-common
$ diff -du /etc/default/nfs-common.orig /etc/default/nfs-common
```

配置lockd:

```
$ cat >> /etc/modprobe.d/lockd.conf <<EOM
options lockd nlm_udpport=32769 nlm_tcpport=32803
EOM
```

重新启动NFS并测试其导出的目录:

```
service nfs-kernel-server restart
# test:
```

```
showmount -e 127.0.0.1
```

将NFS卷加载到本地.

```
$ IP=10.168.100.10
$ mkdir -p /mnt/primary /mnt/secondary
$ cat >>/etc/fstab <<EOM
$IP:/export/primary   /mnt/primary    nfs
rsize=8192,wsize=8192,timeo=14,intr,vers=3,noauto  0   2
$IP:/export/secondary /mnt/secondary  nfs
rsize=8192,wsize=8192,timeo=14,intr,vers=3,noauto  0   2
EOM
$ mount /mnt/primary
$ mount /mnt/secondary
```

安装和配置**libvirt**

安装cloudstack-agent，则可以同时安装和配置libvirt:

```
$ apt-get install cloudstack-agent
```

配置libvirtd:

```
$ cp /etc/libvirt/libvirtd.conf /etc/libvirt/libvirtd.conf.orig

$ sed -i '/#listen_tls = 0/ a listen_tls = 0' /etc/libvirt/libvirtd.conf
$ sed -i '/#listen_tcp = 1/ a listen_tcp = 1' /etc/libvirt/libvirtd.conf
$ sed -i '/#tcp_port = "16509"/ a tcp_port = "16509"' /etc/libvirt/libvirtd.conf
$ sed -i '/#auth_tcp = "sasl"/ a auth_tcp = "none"' /etc/libvirt/libvirtd.conf
$ diff -du /etc/libvirt/libvirtd.conf.orig /etc/libvirt/libvirtd.conf
```

Patch libvirt-bin.conf:

```
$ cp /etc/default/libvirt-bin /etc/default/libvirt-bin.orig
$ sed -i -e 's/libvirtd_opts="-d"/libvirtd_opts="-d -l"/' /etc/default/libvirt-bin
$ diff -du /etc/default/libvirt-bin.orig /etc/default/libvirt-bin
$ service libvirt-bin restart
```

Patch qemu.conf以监听所有端口:

```
$ cp /etc/libvirt/qemu.conf /etc/libvirt/qemu.conf.orig
$ sed -i '/#vnc_listen = "0.0.0.0"/ a vnc_listen = "0.0.0.0"' /etc/libvirt/qemu.conf
$ diff -du /etc/libvirt/qemu.conf.orig /etc/libvirt/qemu.conf
$ service libvirt-bin restart
```

关闭 AppArmor:

```
$ ln -s /etc/apparmor.d/usr.sbin.libvirtd /etc/apparmor.d/disable/
$ ln -s /etc/apparmor.d/usr.lib.libvirt.virt-aa-helper /etc/apparmor.d/disable/
$ apparmor_parser -R /etc/apparmor.d/usr.sbin.libvirtd
$ apparmor_parser -R /etc/apparmor.d/usr.lib.libvirt.virt-aa-helper
$ service libvirt-bin restart
```

在配置防火墙部分，打开以下端口:

```
$ ufw allow proto tcp from any to any port 22
$ ufw allow proto tcp from any to any port 1798
$ ufw allow proto tcp from any to any port 16509
$ ufw allow proto tcp from any to any port 5900:6100
$ ufw allow proto tcp from any to any port 49152:49216
```

现在重新启动机器，看NFS是否正常:

```
$ reboot
$ rpcinfo -u 192.168.77.10 mount
$ showmount -e 192.168.77.10
$ mount /mnt/primary
$ mount /mnt/secondary
```

安装系统虚拟机模板：

```
$ /usr/share/cloudstack-common/scripts/storage/secondary/cloud-install-sys-tmplt \
    -m /mnt/secondary -u  \
http://192.168.0.79/systemvm64template-4.5-kvm.qcow2.bz2 \
    -h kvm -F
```

访问 `http://10.168.100.10:8080/client`, 如果发现 404s 错误 "The requested resource is not available" :

```
service cloudstack-management status
service cloudstack-agent status
service tomcat6 status

service cloudstack-management stop
service tomcat6 stop
service cloudstack-agent stop
ps -efl | grep java

service cloudstack-management start
service cloudstack-management status
service cloudstack-agent start
service cloudstack-agent status
```

接下来我们可以开始配置CloudStack了，配置CloudStack的过程和上一章一样。

已知问题

重启机器后可能会碰到404s错误，需要调整service的启动顺序---TBD.

单物理机配置**CloudStack**

用于在单台主机上配置CloudStack.

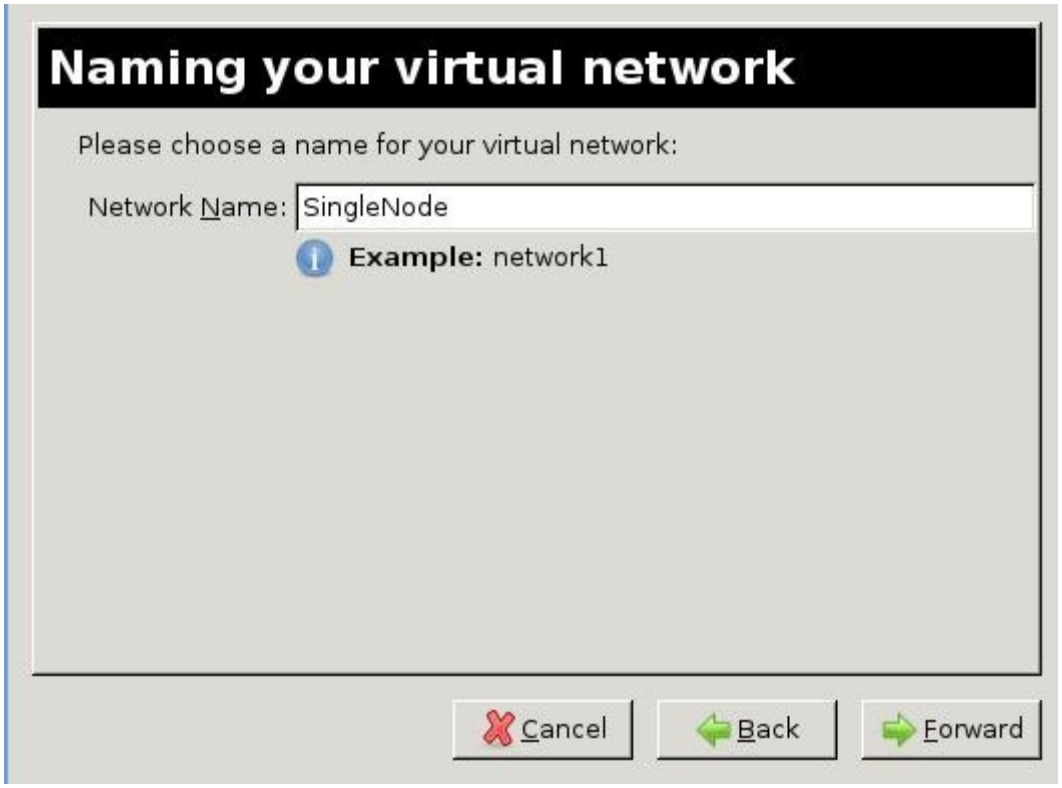
- Ubuntu Machine
- CentOS Machine

Ubuntu主机

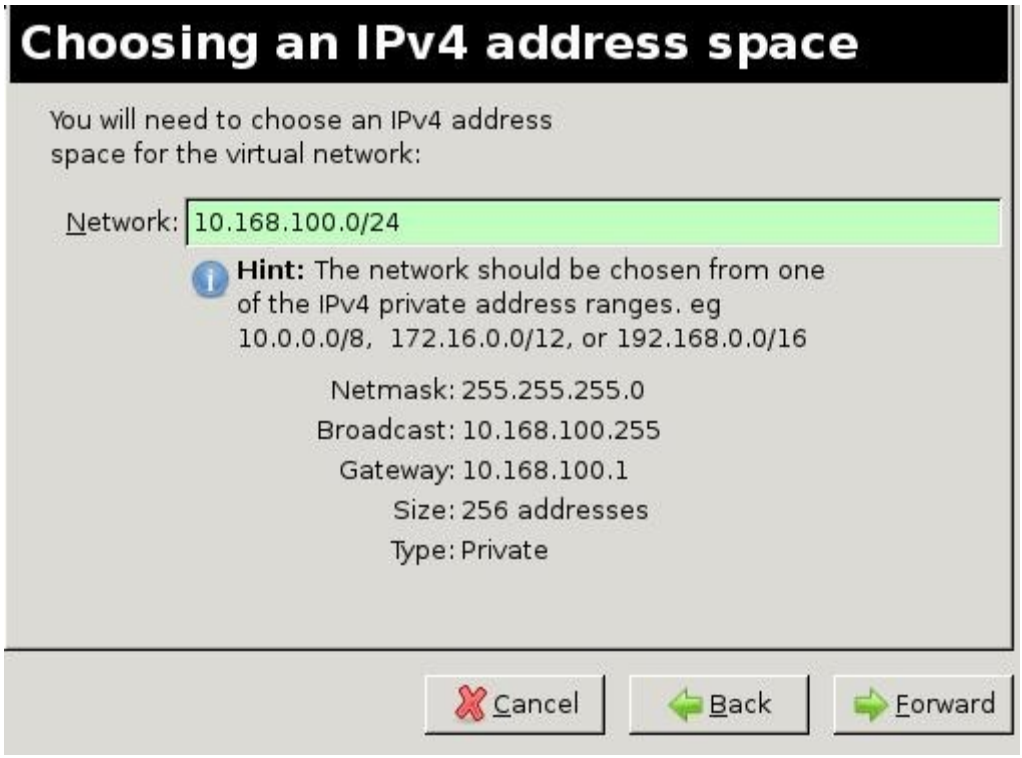
Ubuntu14.04上添加Ubuntu服务

网络添加

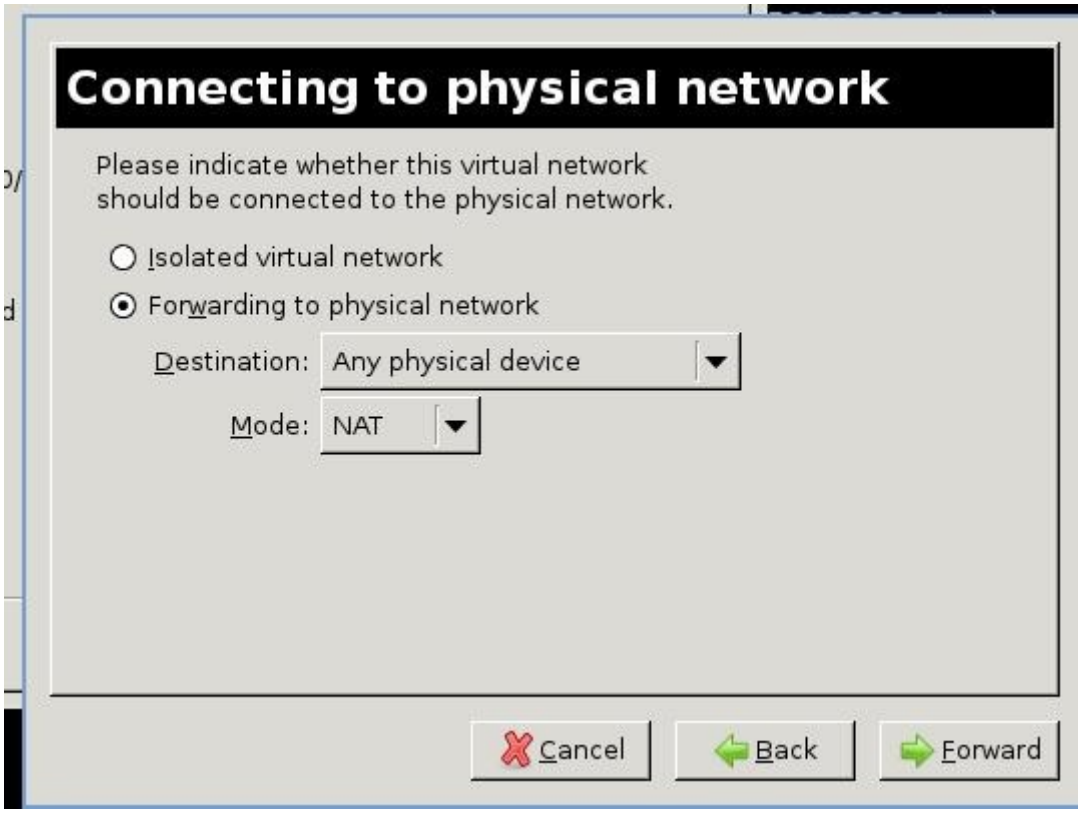
添加一个名为"SingleNode"的网络:



配置网络IP地址范围:



在接下来的配置中，去掉 `Enable DHCP` 的选项， 选择Forward选项:



虚拟机添加

虚拟机添加，安装Management服务。

虚拟机上主要安装cloudstack-management 服务:

```
# apt-get install -y openntpd
# apt-get install -y mysql-server
# service mysql restart
# cloudstack-setup-databases     cloud:engine@localhost \
--deploy-as=root:xxxx     -e file -m mymskey44 -k mydbkey00
```

主机端

主机端开启NFS服务，目录假设在/srv/nfs4上，创建目录并更改其权限:

```
$ sudo mkdir /srv/nfs4/primary
$ sudo mkdir /srv/nfs4/secondary
$ sudo chmod 777 -R /srv/nfs4
```

安装cloudstack-agent包:

```
# apt-get install cloudstack-agent
```

配置libvirt:

```
# vim /etc/libvirt/libvirtd.conf
listen_tls = 0
listen_tcp=1
tcp_port = "16509"
auth_tcp = "none"
# vim /etc/default/libvirt-bin
libvirtd_opts="-d -l"
# service libvirt-bin restart
```

配置qemu.conf:

```
$ sudo vim /etc/libvirt/qemu.conf
vnc_listen = "0.0.0.0"
$ sudo service libvirt-bin restart
```

打开规则:

```
# ln -s /etc/apparmor.d/usr.sbin.libvirtd /etc/apparmor.d/disable/
# ln -s /etc/apparmor.d/usr.lib.libvirt.virt-aa-helper /etc/apparmor.d/disable/
# apparmor_parser -R /etc/apparmor.d/usr.sbin.libvirtd
# apparmor_parser -R /etc/apparmor.d/usr.lib.libvirt.virt-aa-helper
# service libvirt-bin restart
# ufw allow proto tcp from any to any port 22
# ufw allow proto tcp from any to any port 1798
# ufw allow proto tcp from any to any port 16509
```

```
# ufw allow proto tcp from any to any port 5900:6100
# ufw allow proto tcp from any to any port 49152:49216
```

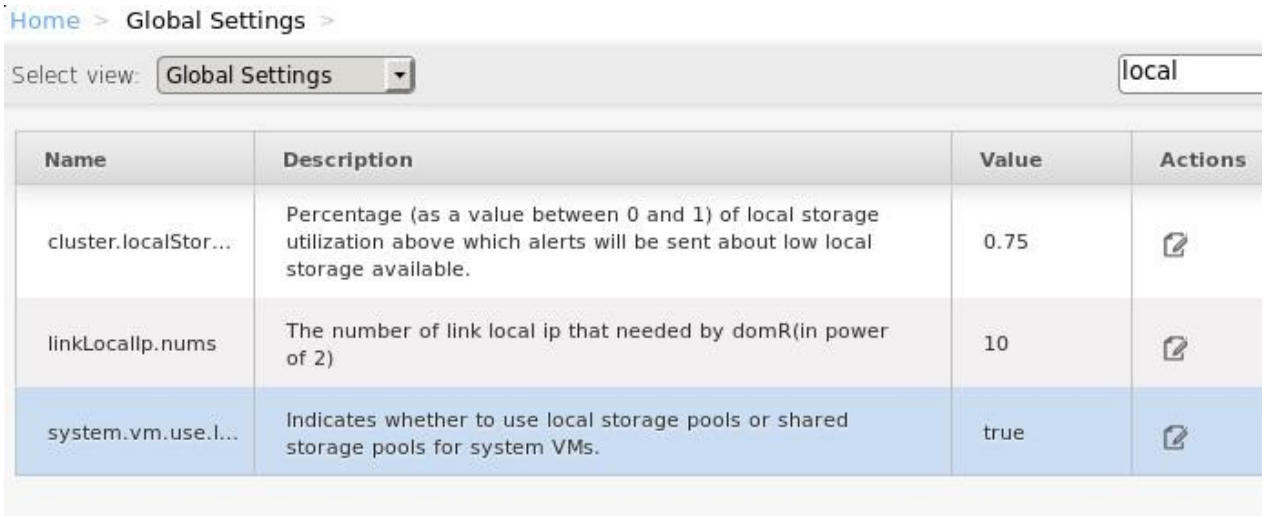
安装虚拟机模板

```
# /usr/share/cloudstack-common/scripts/storage/secondary/cloud-install-sys-tmplt \
    -m /mnt/secondary -u  \
http://192.168.0.79/systemvm64template-4.5-kvm.qcow2.bz2 \
    -h kvm -F
```

这里最好重新启动一下机器,继续下一步配置。

配置

重新启动以后，进入到配置界面 `http://10.168.100.2:8080/client`，首先更改global options 里的本地存储:



重启mangement server:

```
```

管理CloudStack

本章主要涉及到虚拟机的管理及用cloud-init对虚拟机实例进行定制化等话题。

Cloud-Init介绍

参考

主要参考了
http://huang-wei.github.io/programming/2013/12/23/run-cloud-init-in-local-kvm.html

这里主要记录的是操作步骤。

介绍

红帽介绍:
Cloud-Init 是一个用来自动配置虚拟机的初始设置（如主机名，网卡和密钥）的工具。它可以在 使用模板部署虚拟机时使用，从而达到避免网络冲突的目的。
在使用这个工具前，cloud-init 软件包必须在虚拟机上被安装。安装后，Cloud-Init 服务会在系 统启动时搜索如何配置系统的信息。您可以使用只运行一次窗口来提供只需要配置一次的设置信息 ；或在 新建虚拟机、编辑虚拟机和编辑模板窗口中输入虚拟机每次启动都需要的配置信息。

cloud-init安装

Ubuntu 14.04上可以通过以下命令来安装cloud-init:

```
$ apt-cache search cloud-utils
cloud-utils - metapackage for installation of upstream cloud-utils source
$ sudo apt-get install cloud-utils
```

镜像准备

在http://cloud-images.ubuntu.com/releases/ 可以找到Ubuntu制作的ubuntu cloud image, image分版本, 这里使用14.04的image。

```
$ wget http://cloud-images.ubuntu.com/releases/14.04.3/
release-20151008/ubuntu-14.04-server-cloudimg-amd64-disk1.img
```

取回来后的镜像可以直接使用，但解压开后读取速度会更快:

```
$ qemu-img convert -O qcow2 ubuntu-14.04-server-cloudimg-amd64-disk1.img my_vm.img
```

对比两个镜像大小可以看到:

```
$ qemu-img info ubuntu-14.04-server-cloudimg-amd64-disk1.img
```

```
image: ubuntu-14.04-server-cloudimg-amd64-disk1.img
file format: qcow2
virtual size: 2.2G (2361393152 bytes)
disk size: 246M
cluster_size: 65536
Format specific information:
    compat: 0.10
$ qemu-img info my_vm.img
image: my_vm.img
file format: qcow2
virtual size: 2.2G (2361393152 bytes)
disk size: 903M
cluster_size: 65536
Format specific information:
    compat: 1.1
    lazy refcounts: false
```

配置脚本内容

my-user-data内容:

```
$ cat my-user-data
#cloud-config
password: xxxxxx
chpasswd: { expire: False }
ssh_pwauth: True

ssh_authorized_keys:
 - ssh-rsa xxxxxx

timezone: Asia/Chongqing
```

通过my-user-data生成img文件:

```
$ cloud-localds my-seed.img my-user-data
```

由之前的my_vm.img和my-seed.img文件启动虚拟机:

```
$ qemu-system-x86_64 -net nic -net user,hostfwd=tcp::2222-:22 \
 -hda my_vm.img -hdb my-seed.img -m 512
```

通过qemu的窗口或者ssh登录系统: `ssh -p 2222 ubuntu@localhost` .

引入**meta-data**

meta-data的内容与虚拟机的实例相关，只用来做初始化，虚拟机实例运行完一次以后就不需要修改。但如果要引入更新，则重建一下instance-id即可。

更新my-meta-data文件内容:

```
$ echo "instance-id: $(uuidgen || echo i-abcdefg)" > my-meta-data
```

由my-meta-data和my-user-data生成my-seed.img文件:

```
$ cloud-localds my-seed.img my-user-data my-meta-data
```

启动虚拟机:

```
$ qemu-system-x86_64 -net nic -net user,hostfwd=tcp::2222-:22 \
 -hda my_vm.img -hdb my-seed.img -m 512
$ kvm -net nic -net user,hostfwd=tcp::2222-:22 \
 -hda my_vm.img -hdb my-seed.img -m 512
```

其他初始化行为

需要初始化的脚本:

```
$ cat hello_world.sh
#!/bin/bash
echo "hello world!" >> /home/ubuntu/test
```

将初始化脚本和cloud config data合并:

```
$ write-mime-multipart
--output=combined-userdata.txt hello_world.sh:text/x-shellscript my-user-data
```

由生成的combined-userdata.txt生成my-seed.img:

```
$ echo "instance-id: $(uuidgen || echo i-abcdefg)" > my-meta-data
$ cloud-localds my-seed.img combined-userdata.txt my-meta-data
```

重启后即可得到更新后的系统镜像.

Cloud-Init数据源

数据源介绍

数据源是供cloud-init所使用的数据，通常来自于用户(即userdata)或者来自于创建配置驱动的栈 (即metadata)。通常userdata包括了文件，yaml, 和shell命令等；而典型的metadata则包括服务器名，instance id, 显示名称以及其他云端需要的细节信息。

有很多种方法可以用来提供这类数据(每种云解决方案似乎都倾向于提供自己的解决方案)。 Cloud-Init有其内部的数据源抽象类，用一种相同的方式来处理各种不同的云系统所提供的数据源 .

在API的层面，数据源对象提供了以下的接口:

```
# returns a mime multipart message that contains
# all the various fully-expanded components that
# were found from processing the raw userdata string
# - when filtering only the mime messages targeting
#   this instance id will be returned (or messages with
#   no instance id)
def get_userdata(self, apply_filter=False)

# returns the raw userdata string (or none)
def get_userdata_raw(self)

# returns a integer (or none) which can be used to identify
# this instance in a group of instances which are typically
# created from a single command, thus allowing programatic
# filtering on this launch index (or other selective actions)
@property
def launch_index(self)

# the data sources' config_obj is a cloud-config formated
# object that came to it from ways other than cloud-config
# because cloud-config content would be handled elsewhere
def get_config_obj(self)

#returns a list of public ssh keys
def get_public_ssh_keys(self)

# translates a device 'short' name into the actual physical device
# fully qualified name (or none if said physical device is not attached
# or does not exist)
def device_name_to_device(self, name)

# gets the locale string this instance should be applying
# which typically used to adjust the instances locale settings files
def get_locale(self)

@property
def availability_zone(self)
```

```
    # gets the instance id that was assigned to this instance by the
    # cloud provider or when said instance id does not exist in the backing
    # metadata this will return 'iid-datasource'
    def get_instance_id(self)

    # gets the fully qualified domain name that this host should  be using
    # when configuring network or hostname releated settings, typically
    # assigned either by the cloud provider or the user creating the vm
    def get_hostname(self, fqdn=False)

    def get_package_mirror_info(self)
```

No Cloud数据源

上一节里所展示的就是用No Cloud来提供数据源的方式。

数据源 `NoCloud` 和 `NoCloudNet` 让用户在无网络服务的情况下提供 `user-data` 和 `meta-data` 给虚 拟机的运行实例，甚至它可以在无网络设备的情况下运行。

我们可以通过在虚拟机启动时加载一个vfat或iso9660文件系统的方式，提供出 `user-data` 和 `meta-data` 。

这些提供的 `user-data` 和 `meta-data` 文件应该具备以下的格式, 即处于该文件系统根目录下:

```
/user-data
/meta-data
```

下面给出的是ubuntu 12.04 cloud image 可以使用的 `user-data` 和 `meta-data` :

```
## create user-data and meta-data files that will be used
## to modify image on first boot
$ { echo instance-id: iid-local01; echo local-hostname: cloudimg; } > meta-data

$ printf "#cloud-config\npassword: passw0rd\nchpasswd:
{ expire: False }\nssh_pwauth: True\n" > user-data

## create a disk to attach with some user-data and meta-data
$ genisoimage  -output seed.iso -volid cidata -joliet -rock user-data meta-data

## alternatively, create a vfat filesystem with same files
## $ truncate --size 2M seed.img
## $ mkfs.vfat -n cidata seed.img
## $ mcopy -oi seed.img user-data meta-data ::

## create a new qcow image to boot, backed by your original image
$ qemu-img create -f qcow2 -b disk.img boot-disk.img

## boot the image and login as 'ubuntu' with password 'passw0rd'
## note, passw0rd was set as password through the user-data above,
## there is no password set on these images.
```

```
$ kvm -m 256 \
   -net nic -net user,hostfwd=tcp::2222-:22 \
   -drive file=boot-disk.img,if=virtio \
   -drive file=seed.iso,if=virtio
```

在上一节中使用的img方式类似。

CloudStack数据源

Apache CloudStack在Virtual-Router虚拟机上暴露出了user-data, meta-data，用户密码以及账户 的sshkey等数据。更详细的关于meta-data和user-data可以从CloudStack的管理手册上查阅到:

CloudStack管理手册

用于访问user-data和meta-data的URLs如下, 这里10.1.1.1代表Virtual Router的IP：

```
http://10.1.1.1/latest/user-data
http://10.1.1.1/latest/meta-data
http://10.1.1.1/latest/meta-data/{metadata type}
```

CloudStack与Cloud-Init

CloudStack高阶

这里主要讨论以下几个命题:

- 编译CloudStack DEB
- 编译CloudStack RPM

编译CloudStack DEB

这里我们主要记载步骤，我是用一个Ubuntu的容器开始编译的，所以附加了有关创建容器的内容在 其中。

容器的创建:

```
$ sudo docker pull ubuntu
$ sudo docker run -it ubuntu /bin/bash
root@a1e2fc3f3abe:#
```

在容器里依次执行以下操作:

```
# apt-get update
# apt-get install -y vim
# apt-get install -y  python-software-properties
# apt-get install -y  ant debhelper openjdk-7-jdk tomcat6 libws-commons-util-java \
 genisoimage python-mysqldb libcommons-codec-java libcommons-httpclient-java \
 liblog4j1.2-java maven -y
# mkdir -p ~/Code
# cd ~/Code/
# wget http://xxxxxxxxx/apache-cloudstack-4.5.2-src.tar.bz2
# tar xjvf apache-cloudstack-4.5.2-src.tar.bz2
# cd apache-cloudstack-4.5.2-src
# mvn -P deps -Dnonoss -DskipTests=true
# dpkg-buildpackage -uc -us
```

编译完毕后可以在上层找到生成的DEB包：

```
# ls ../
apache-cloudstack-4.5.2-src          cloudstack-awsapi_4.5.2_all.deb
cloudstack-docs_4.5.2_all.deb        cloudstack_4.5.2.dsc
apache-cloudstack-4.5.2-src.tar.bz2  cloudstack-cli_4.5.2_all.deb
cloudstack-management_4.5.2_all.deb  cloudstack_4.5.2.tar.gz
cloudstack-agent_4.5.2_all.deb       cloudstack-common_4.5.2_all.deb
cloudstack-usage_4.5.2_all.deb       cloudstack_4.5.2_amd64.changes
# du -hs ~/Code/
3.2G    /root/Code/
```

其他版本的CloudStack的生成过程类似。

编译CloudStack RPM

首先下载第三方依赖包:

```
# cd deps
# wget http://zooi.widodh.nl/cloudstack/build-dep/cloud-iControl.jar
# wget http://zooi.widodh.nl/cloudstack/build-dep/cloud-manageontap.jar
# wget http://zooi.widodh.nl/cloudstack/build-dep/vmware-vim.jar
# wget http://zooi.widodh.nl/cloudstack/build-dep/vmware-vim25.jar
# wget http://zooi.widodh.nl/cloudstack/build-dep/vmware-apputils.jar
# wget http://zooi.widodh.nl/cloudstack/build-dep/cloud-netscaler-jars.zip
# mv cloud-manageontap.jar manageontap.jar
# mv vmware-apputils.jar apputils.jar
# mv vmware-vim.jar vim.jar
# mv vmware-vim25.jar vim25.jar
# unzip cloud-netscaler-jars.zip
```

从Vmware网站下载到SDK，并提取内容:

```
VMware-vSphere-SDK-5.1.0-774886.zip
https://my.vmware.com/group/vmware/get-download?downloadGroup=VSP510-WEBSDK-510

# unzip VMware-vSphere-SDK-5.1.0-774886.zip
# cp -p SDK/vsphere-ws/java/JAXWS/lib/vim25.jar vim25_51.jar
# ./install-non-oss.sh
```

编译依赖关系:

```
# mvn -P deps -D nonoss -DskipTests=true
```

生成包:

```
# cd packaging/centos63
#  vi cloud.spec
if [ "%{_ossnoss}" == "NOREDIST" -o "%{_ossnoss}" == "noredist" ] ; then
   echo "Executing mvn packaging with non-redistributable libraries"
   if [ "%{_sim}" == "SIMULATOR" -o "%{_sim}" == "simulator" ] ; then
      echo "Executing mvn noredist packaging with simulator ..."
      mvn -Pawsapi,systemvm -Dnoredist -Dsimulator clean package
   else
      echo "Executing mvn noredist packaging without simulator..."
-      mvn -Pawsapi,systemvm -Dnoredist clean package
+      mvn -Pawsapi,systemvm -Dnoredist -DskipTests=true clean package
   fi
# ./package.sh -p noredist
```

编译完毕后，可以看到内容:

```
$ ls
cloudstack-agent-4.5.2-1.el6.x86_64.rpm
cloudstack-cli-4.5.2-1.el6.x86_64.rpm
cloudstack-mysql-ha-4.5.2-1.el6.x86_64.rpm
cloudstack-awsapi-4.5.2-1.el6.x86_64.rpm
cloudstack-common-4.5.2-1.el6.x86_64.rpm      cloudstack-usage-4.5.2-1.el6.x86_64.rpm
cloudstack-baremetal-agent-4.5.2-1.el6.x86_64.rpm
cloudstack-management-4.5.2-1.el6.x86_64.rpm
$ pwd
/home/xxxx/Code/apache-cloudstack-4.5.2-src/dist/rpmbuild/RPMS/x86_64
```

性能监控

性能监控框架，包括:

-

安装步骤

主要参考了 http://www.unixmen.com/install-graphite-centos-7/

```
# yum -y update
# yum install -y httpd net-snmp perl pycairo mod_wsgi python-devel git gcc-c++
# wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo
# yum install -y python-pip node npm
# pip install 'django
```

这时候打开Graphite，是什么都看不到的，因为没有往里面喂数据。

注入数据

使用statsd注入数据，注入的数据会被在graphite被显示出来。

前面我们已经clone了statsd的代码到本地，直接使用它:

```
$ cd /usr/local/src/statsd
$ npm install -g cnpm --registry=https://registry.npm.taobao.org
$ cnpm install nodeunit
$ cnpm install temp
$ cnpm install underscore
$ vim exampleConfig.js
    {
      graphitePort: 2003
    , graphiteHost: "192.168.10.191"
    , port: 8125
    , backends: [ "./backends/graphite" ]
    }
$ node stats.js ./exampleConfig.js
```

statsd将监听8125端口.

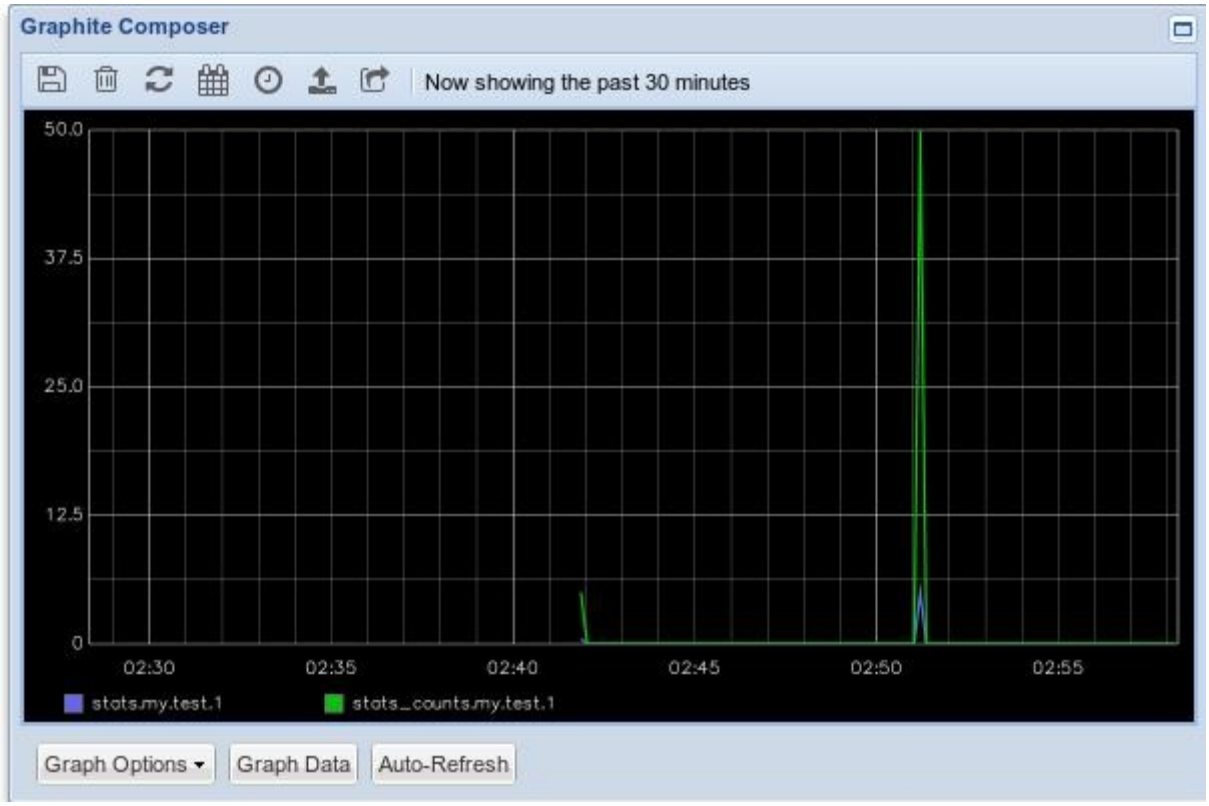
```
# cnpm install statsd-client --save
# mkdir ~/Code/statsD
# vim app.js
    var sdc_init = require('statsd-client')
    var sdc = new sdc_init({ host: 'localhost' });
    sdc.increment('my.test.1', 50);
    sdc.close();
# node ./app.js
# vim /opt/graphite/conf/storage-schemas.conf
    [default_1min_for_1day]
    pattern = .*
    retentions = 60s:1d

    [statsD]
    pattern = ^stats\.
```

```
    retentions = 10s:1d,30s:7d,1m:21d,15m:5y

    [statsD-2]
    pattern = ^stats_counts\.
    retentions = 10s:1d,30s:7d,1m:21d,15m:5y
# cd /opt/graphite/bin
# python carbon-cache.py stop
# python carbon-cache.py start
```

现在重启可以看到结果:



监测Ubuntu

Graphite

更改主机名:

```
root@packer-ubuntu-1404-server:~# cat /etc/hostname
monitorserver
root@packer-ubuntu-1404-server:~# cat /etc/hosts
127.0.0.1       localhost
192.168.11.192  monitorserver

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

安装Graphite:

```
# apt-get -y update
# apt-get install graphite-web graphite-carbon
```

配置postgresql及辅助软件:

```
#  apt-get install postgresql libpq-dev python-psycopg2
```

创建数据库:

```
root@monitorserver:~# sudo -u postgres psql
could not change directory to "/root": Permission denied
psql (9.3.9)
Type "help" for help.

postgres=# CREATE USER graphite WITH PASSWORD 'password';
CREATE ROLE
postgres=# CREATE DATABASE graphite WITH OWNER graphite;
CREATE DATABASE
postgres=# \q
root@monitorserv
```

配置Graphite Web应用程序:

```
$ sudo vim /etc/graphite/local_settings.py
SECRET_KEY = 'a_salty_string'
```

```
TIME_ZONE='Asia/Shanghai'
USE_REMOTE_USER_AUTHENTICATION = True
DATABASES = {
    'default': {
        'NAME': 'graphite',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'USER': 'graphite',
        'PASSWORD': 'password',
        'HOST': '127.0.0.1',
        'PORT': ''
        }
}
```

修改完毕后，我们可以同步数据库以创建正确的网站结构:

```
# graphite-manage syncdb
```

这里需要输入用户名和设置密码，以便下次我们登录Graphite界面用.

更改carbon后端的启动方式:

```
# vim /etc/default/graphite-carbon
    # Change to true, to enable carbon-cache on boot
    CARBON_CACHE_ENABLED=true
```

打开log rotation的选项:

```
$ sudo vim /etc/carbon/carbon.conf
ENABLE_LOGROTATION = True
```

更改storage schemas选项, 在default前添加test字段, 如下：

```
# cat /etc/carbon/storage-schemas.conf
+ [test]
+ pattern = ^test\.
+ retentions = 10s:10m,1m:1h,10m:1d

[default_1min_for_1day]
pattern = .*
retentions = 60s:1d
```

配置storage aggregation选项:

```
# cp /usr/share/doc/graphite-carbon/examples/storage-aggregation.conf.example \
/etc/carbon/storage-aggregation.conf
# service carbon-cache start
```

安装apache2服务器:

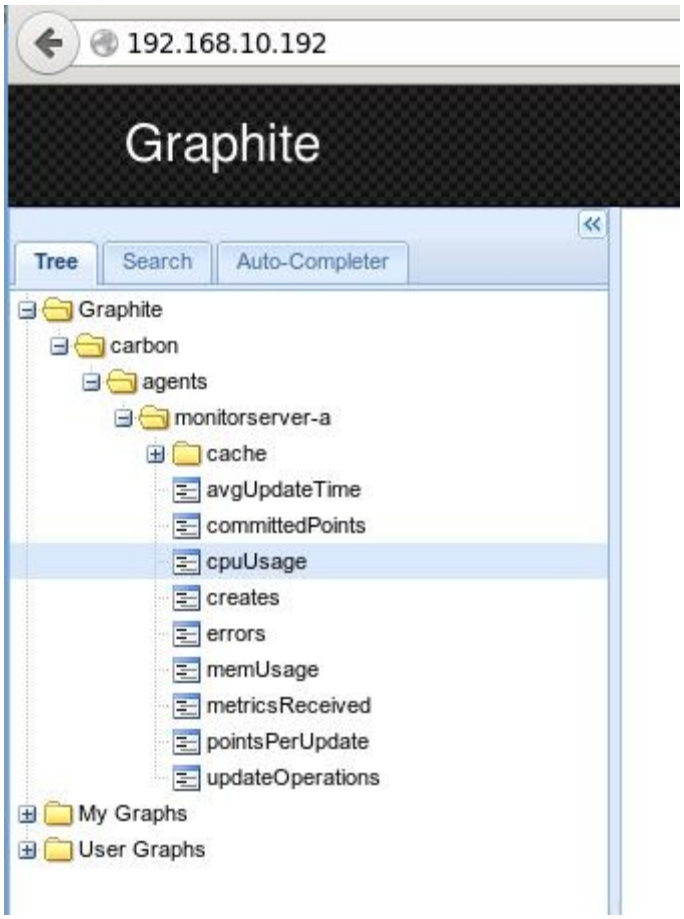
```
# apt-get install apache2 libapache2-mod-wsgi
```

禁止000-default网站, 定义apache2-graphite条目并使能:

```
# a2dissite 000-default
# cp /usr/share/graphite-web/apache2-graphite.conf \
/etc/apache2/sites-available/
# a2ensite apache2-graphite
# service apache2 reload
```

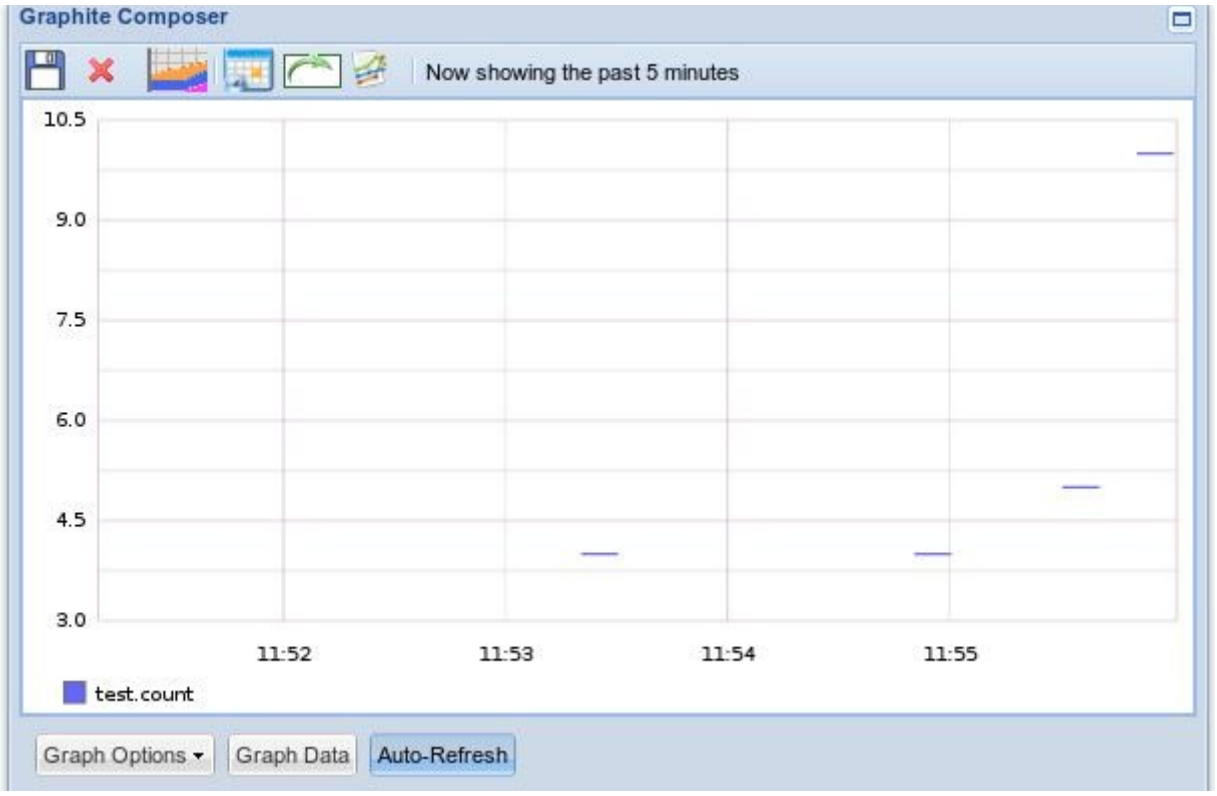
现在访问http://Your_IP， 可以看到界面如下:

用预设定的用户名/密码登录后，可以看到下面的页面：



简单测试：

```
# echo "test.count 4 `date +%s`" | nc -q0 127.0.0.1 2003
# echo "test.count 5 `date +%s`" | nc -q0 127.0.0.1 2003
# echo "test.count 10 `date +%s`" | nc -q0 127.0.0.1 2003
```



Collected

安装:

```
# apt-get install collectd collectd-utils
```

配置:

```
# vim /etc/collectd/collectd.conf
Hostname "localhost"

LoadPlugin apache
LoadPlugin cpu
LoadPlugin df
LoadPlugin entropy
LoadPlugin interface
LoadPlugin load
LoadPlugin memory
LoadPlugin processes
LoadPlugin rrdtool
LoadPlugin users
LoadPlugin write_graphite

<Plugin apache>
    <Instance "Graphite">
        URL "http://domain_name_or_IP/server-status?auto"
        Server "apache"
    </Instance>
</Plugin>
```

```
<Plugin interface>
    Interface "eth0"
    IgnoreSelected false
</Plugin>

<Plugin write_graphite>
    <Node "graphing">
        Host "192.168.10.192"
        Port "2003"
        Protocol "tcp"
        LogSendErrors true
        Prefix "collectd."
        StoreRates true
        AlwaysAppendDS false
        EscapeCharacter "_"
    </Node>
</Plugin>
```

添加apache2 状态页:

```
# vim /etc/apache2/sites-available/apache2-graphite.conf
 +   <Location "/server-status">
 +       SetHandler server-status
 +       Require all granted
 +   </Location>

    ErrorLog ${APACHE_LOG_DIR}/graphite-web_error.log
# service apache2 reload
```

现在访问 `http://192.168.10.192/server-status` 可以看到有关状态.

更改：

```
# vim /etc/carbon/storage-schemas.conf
[collectd]
pattern = ^collectd.*
retentions = 10s:1d,1m:7d,10m:1y
```

重启服务:

```
# sudo service carbon-cache stop
# sudo service carbon-cache start
# sudo service collectd stop
# sudo service collectd start
```

现在查看collectd收集到的信息:

Added CentOS7 Agent

Install, configure, and enable the collectd service:

```
# yum install -y collectd*
# vim /etc/collectd.conf
# service collectd start
# systemctl enable collectd.service
```

Monitor CloudStack

https://github.com/exoscale/collectd-cloudstack

Monitor Windows

http://ssc-serv.com/licensing.shtml

Customization

Add new templates for graphite.

监控CloudStack

CentOS6 的CloudStack Management节点上:

编译collectd RPM

准备编译环境

安装编译RPM所需要的包:

```
# yum groupinstall -y 'Development Tools'
# yum install -y rpmdevtools yum-utils
```

创建一个专门用来编译rpm的用户，并创建编译树:

```
# su - makerpm
$ rpmdev-setuptree
$ ls rpmbuild/
BUILD  RPMS  SOURCES  SPECS  SRPMS
```

首先准备源代码：

```
$  wget https://collectd.org/files/collectd-5.5.0.tar.gz
$  tar xzvf collectd-5.5.0.tar.gz
$  cd collectd-5.5.0
```

编译过程:

```
$ cp ~/Code/collectd-5.5.0/contrib/redhat/collectd.spec rpmbuild/SPECS/
$ cd ~/rpmbuild/SPECS
$ spectool -g -R collectd.spec
$ rpm --eval "%{_sourcedir}"
$ rpmbuild --bb collectd.spec
$ sudo yum-builddep collectd.spec
$ sudo yum install perl-Heap
$ rpmbuild --bb collectd.spec
```

编译完毕后，可以看到rpm包：

```
$ pwd
/home/makerpm/rpmbuild/RPMS
$ ls x86_64/
collectd-5.5.0-1.el6.x86_64.rpm             collectd-hddtemp-5.5.0-1.el6.x86_64.rpm
collectd-amqp-5.5.0-1.el6.x86_64.rpm        collectd-ipmi-5.5.0-1.el6.x86_64.rpm
```

```
collectd-apache-5.5.0-1.el6.x86_64.rpm       collectd-iptables-5.5.0-1.el6.x86_64.rpm
collectd-ascent-5.5.0-1.el6.x86_64.rpm       collectd-java-5.5.0-1.el6.x86_64.rpm
```

本地安装库的创建

注意要移除掉epel里的定义, CentOS 6 epel仓库中含有的collectd的版本太旧:

```
# cat /etc/yum.repos.d/epel.repo
[epel]
name=Extra Packages for Enterprise Linux 6 - $basearch
#baseurl=http://download.fedoraproject.org/pub/epel/6/$basearch
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-6&arch=$basearch
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6

# Blacklist for collectd
exclude=collectd*
```

CloudStack添加

首先确保 `collectd-python` 包被安装:

```
yum install -y collectd-python
```

取得cloudstack-python的文件:

```
# git clone https://github.com/exoscale/collectd-cloudstack.git
# cd /usr/lib/collectd/
# cp ~/collectd-cloudstack/*.py ./
# cd /usr/lib64/collectd/
# cp ~/collectd-cloudstack/*.py ./
```

配置文件:

```
Hostname "192.168.139.2"
FQDNLookup true
Interval 10
Timeout 2
ReadThreads 5
LoadPlugin syslog
<Plugin syslog>
        LogLevel info
</Plugin>
<LoadPlugin python>
```

```
    Globals true
</LoadPlugin>
<Plugin python>
    # cloudstack.py is at /usr/lib/collectd/cloudstack.py
    ModulePath "/usr/lib64/collectd/"
    #ModulePath "/usr/lib/collectd/"

    Import "cloudstack"

<Module cloudstack>
  Api "http://192.168.139.2:8080/client/api"
  Auth "True"
  ApiKey
"MZdZARfnQ5pvRLbWC4ORQvo5WXWioTVfz7f2q2Lf5R1Q8QGi0HxrxzkkMi4Gg0nMrqSBBYLy-Ooy1T1CKDwLzA'
  Secret
"82uaYtqj-G4dR4GbTWDagMXyPxOkcJWlZc6UV5SObA4V7-V5a-2cRvyC_aaKxzNC6UdqXuwFC3WbSU48y11BYw'
</Module>
</Plugin>


LoadPlugin battery
#LoadPlugin cloudstack
LoadPlugin cpu
LoadPlugin df
LoadPlugin disk
LoadPlugin entropy
LoadPlugin interface
LoadPlugin irq
LoadPlugin load
LoadPlugin memory
LoadPlugin network
LoadPlugin processes
LoadPlugin rrdtool
LoadPlugin swap
LoadPlugin users
LoadPlugin write_graphite
# <Plugin network>
#    Listen "10.211.55.46" "25826"
# </Plugin>
<Plugin write_graphite>
    <Node "graphing">
        Host "192.168.10.192"
        Port "2003"
        Protocol "tcp"
        LogSendErrors true
        Prefix "collectd."
        StoreRates true
        AlwaysAppendDS false
        EscapeCharacter "_"
    </Node>
</Plugin>


<Plugin rrdtool>
    DataDir "/var/lib/collectd/rrd"
</Plugin>
```

启动并设置为开机自启动:

```
# service collectd start
# chkconfig collectd on
```

并发测试

这里介绍一个并发10W~100W的解决方案.
主要参考了:
http://www.blogjava.net/yongboy/archive/2015/10/13/397559.html#427723

服务端/客户端准备

服务端: i5 4核4线程, 32G 内存, 1000M网口. Ubuntu 15.04 x86_64.
客户端: i7 4核8线程, 12G 内存, 1000M网口. ArchLinux x86_64.

均打开security的limits:

```
# vim /etc/security/limits.conf

* soft nofile 104857
* hard nofile 104857
```

而后重启动机器.

服务端准备

服务器端依赖于libev, 首先编译:

```
$ wget http://dist.schmorp.de/libev/libev-4.20.tar.gz
$ tar xzvf libev-4.20.tar.gz
$ cd libev-4.20/
$ ./configure --enable-static
$ make -j4
$ cp .libs/libev.a ./
$ mkdir include
$ cp ./libev-4.20/*.h include/
```

获取源文件:

```
$ wget
https://gist.github.com/yongboy/5318994/raw/f0cc8650c3350df1578dd986a38f4700152cd976/cli
```

```
$ wget
https://gist.githubusercontent.com/yongboy/5318930/raw/ccf8dc236da30fcf4f89567d567eaf295
```

编译:

```
$ vim server.c
- #include "../include/ev.h"
+ #include "./include/ev.h"
$ gcc -o server server.c  ./libev-4.20/libev.a -lm
$ ls server*
server  server.c
```

运行 `./server` 将打印出当前的内存数,并监听8000端口.

服务端调优

需要注意的是conntrack的最大数需要修改:

```
# sysctl -a | grep -i conntrack_max
net.netfilter.nf_conntrack_max = 65536
net.nf_conntrack_max = 65536
```

这意味着如果连接数超过65536时,将无法进行其他连接操作,需要手动调大其值:

```
# vim /etc/sysctl.conf
net.netfilter.nf_conntrack_max = 6553500
# sysctl -p /etc/sysctl.conf
# sysctl -a | grep -i conntrack_max
net.netfilter.nf_conntrack_max = 6553500
net.nf_conntrack_max = 6553500
```

ulimit的限制数目:

```
$ ulimit -HSn 1048576
```

客户端准备

客户端依赖于libevent, 所以先下载libevent并编译:

```
$ wget https://github.com/downloads/libevent/libevent/libevent-2.0.21-stable.tar.gz
$ tar xzvf libevent-2.0.21-stable.tar.gz
$ cd libevent-2.0.21-stable
$ ./configure --prefix=/usr
$ make -j5
$ sudo make install
```

修改服务器地址:

```
# vim client1.c
+ #define SERVERADDR "192.168.1.13"
```

编译:

```
$ gcc -o client client1.c -levent
$ sudo ln -s /usr/lib/libevent-2.0.so.5 /lib64/libevent-2.0.so.5
```

客户端突破28200限制:

```
# cat /proc/sys/net/ipv4/ip_local_port_range
32768 61000
# echo "1024 65535"> /proc/sys/net/ipv4/ip_local_port_range
```

通过sysctl写入:

```
$ sudo vim /etc/sysctl.conf
net.ipv4.ip_local_port_range= 1024 65535
```

测试调优

接下来我们将在CloudStack环境里开始轰炸VR, 看到多少连接时它将挂掉.

OpenStack Liberty

这里记录手工搭建OpenStack Liberty的过程:

- Ubuntu

OpenStack Ubuntu安装

主要参考 http://docs.openstack.org/liberty/install-guide-ubuntu/

OpenStack服务介绍

工具

本章主要涉及到以下工具的配置和使用:

- Packer.io

Packer.io

Packer是一个用于从预定义好的配置文件中创建出虚拟机和容器镜像的工具，用Packer创 建出来的镜像可以运行于多种平台，也可以直接部署到云端。Packer可以运行在 Linux/Unix/MacOS以及Windows上。

官方网站:
https://www.packer.io/

安装

首先从 https://www.packer.io/downloads.html下 载相应版本的Packer.io,我们这里下载Linux 64-bit的安装包，下载完毕后的大小是 127M:

```
$ wget https://dl.bintray.com/mitchellh/packer/packer_0.8.6_linux_amd64.zip
```

直接解压压缩包到某个目录而后把该目录加入到系统路径:

```
$ mkdir ~/bin
$ mv packer_0.8.6_linux_amd64.zip ~/bin
$ cd ~/bin
$ unzip *.zip
$ vim ~/.bashrc
export PATH=/home/XXXXX/bin:$PATH
$ source ~/.bashc
$ which packer
/home/XXXXX/bin/packer
```

Vagrant

Vagrant的好处是可以真正跨平台，Windows/Mac/Linux/Unix均可运行。适于快速验证某些工具和环境。

Packer.io编译vbox

可以参考:

https://www.packer.io/intro/getting-started/vagrant.html