QuickCloud

CloudStack uses system vms extensively to offload the control plane ("management server") from being in the data path. There are 3 kinds of system vms:

1. Those used to provide network services such as DHCP, routing, firewall, load balancing (e.g., VirtualRouter)
2. Those used to provide image services ("Secondary Storage Virtual Machine" SSVM)
3. Those used to provide console services ("Console Proxy VM")

Some of the virtues of these system vms are:

1. automated scaling of services provided by these system vms
2. standardized deployments
3. minimal requirements on the control plane (i.e., just the hardware required to run the management cluster)

However the system vms are intertwined into the 'bootstrap' phase of the cloud and often times result in failure to start the cloud.
Another issue is that certain choices are made inside these system vms (for e.g., mount options for the secondary storage) that may not be ideal or optimal for the cloud environment.
A third issue is that the dependence on system vms hampers the introduction of support for new hypervisors.
Finally the requirement to bring up system vms prior to first user vm booting leads to longer development-debug cycles.
To avoid these issues while preserving the benefits of the system vms, it is proposed:

1. To enable a cloud to "boot" with services managed and running on non-cloudstack-managed servers
2. To enable a development environment based on DevCloud ("QuickCloud") that eschews the use of system vms
3. To allow the cloud administrator to choose to provide these services the "old" way after booting in the QuickCloud fashion

Secondary Storage

The secondary storage vm runs a java daemon that starts up on boot of the SSVM. This java daemon connects back to the management server. The daemon was originally designed to run alternately as a standalone process (just like the KVM agent), but over time the execution environment has been assumed to be the system vm. QuickCloud will bring back that ability

Console Proxy

The concerns and the approach are the same as Secondary Storage

DHCP and DNS

DHCP and DNS services are traditionally provided by the dnsmasq service running inside the system vm.

1. Basic Zone: It is proposed to run a single dnsmasq process in the entire zone that is controlled via a REST API
2. Advanced Zone: System VMs will continue to be used.

HOWTO: maven workflow for QuickCloud

[At present QuickCloud on DevCloud2 does not use any DHCP services]
1. Get the DevCloud2 VM up and running.
2. Mount the secondary storage export from the DevCloud2 on to your laptop:

```
export SS_MOUNT= mount-point-for-sec-storage e.g., /users/myhome/secondary-storage
mkdir $SS_MOUNT
mount -t nfs 192.168.56.10:/opt/storage/secondary $SS_MOUNT
mount -t nfs -o resvport 192.168.56.10:/opt/storage/secondary $SS_MOUNT #Mac OS X
```

3. Checkout the 'master' branch and build it and run it

```
git checkout master
mvn -P developer,systemvm  clean install -Dquickcloud
mvn -P developer -pl developer,tools/devcloud -Ddeploydb
mvn -pl :cloud-client-ui jetty:run
```

In a different terminal, execute the quickcloud zone creation script via marvin

```
mvn -P developer -pl tools/devcloud -Ddeployquick
```

4. Open two more terminals. In the first terminal

```
cd services/secondary-storage
```

In the second terminal

```
cd services/console-proxy/server
```

5. In the secondary-storage terminal

```
echo "mount.path=/users/myhome/secondary-storage" >> conf/agent.properties
mvn exec:java
```

In the console-proxy terminal,

```
echo "paths.pid=." > conf/environment.properties
mvn exec:java -Pquickcloud
```

6. Open the GUI and deploy a VM. Although the VM won't get an IP, you should be able to view the console and log in and set an ip.

HOWTO: Using standalone Secondary Storage and Console Proxy Servers in production

0. The following assumes you are working on the same server as the management server, but equivalently you can set up another VM/server for the purpose of running these system services. This will also work with DevCloud2.
1. Mount the secondary storage export from your NFS server on to a mount point on your (management) server. Ensure that the user that runs the secondary storage service is able to write and read this filesystem.

```
export SS_MOUNT= mount-point-for-sec-storage e.g., /var/mount/secondary-storage
mount -t nfs $NFS_IP:$NFS_EXPORT $SS_MOUNT #use your favorite NFS mount options
```

2. Find the 'systemvm.zip' archive on your management server

```
find /usr/share -name systemvm.zip #centos
find client -name systemvm.zip #in your git workspace
```

3. Create 2 directories for running the SS and CP services respectively. For example:

```
mkdir -p /var/cloudstack/secondary #or wherever
mkdir -p /var/cloudstack/console
cp systemvm.zip /var/cloudstack/secondary
cp systemvm.zip /var/cloudstack/console
```

4. Secondary Storage server:

```
 cd /var/cloudstack/secondary
unzip systemvm.zip
```

Edit conf/agent.properties and add

```
zone=1
# eth1ip needed only if you have one network adapter, or you need to the service to use a specific private ip
address.
eth1ip=<ip address available to service> # should offer similar functionality to systemvm private ip address
#or actual zone id from management server database table datacenter
host=<ip of management server>
guid=Secondary.1
instance=SecondaryStorage
resource=org.apache.cloudstack.storage.resource.NfsSecondaryStorageResource
#resource=com.cloud.storage.resource.PremiumSecondaryStorageResource for VMWare
mount.path=<value of $SS_MOUNT>
secondary.storage.vm=false
```

If the user running these services is not able to write to /var/run/, create an environment.properties file in conf

```
echo "paths.pid=." > conf/environment.properties
```

5. Console Proxy Server:

```
 cd /var/cloudstack/console
unzip systemvm.zip
```

Edit conf/agent.properties and add

```
zone=1
# or actual zone id from management server database table datacenter
host=<ip of management server>
guid=ConsoleProxy.1
```

If the user running these services is not able to write to /var/run/, create an environment.properties file in conf

```
echo "paths.pid=." > conf/environment.properties
```

6. Set Console Proxy "public" IP
The console proxy service has to be made available at a "public" ip. Tell the management server the value of this ip by adding

```
INSERT INTO `configuration` (`category`, `instance`, `component`, `name`, `value`, `description`) VALUES
        ('Console Proxy', 'DEFAULT', 'AgentManager', 'consoleproxy.static.publicIp', '<public ip>', 'Console
proxy public ip');
```

Edit the applicationContext.xml in the web app (WEB-INF/classes). You need to change the consoleProxyManagerImpl line to

```
<bean id="consoleProxyManagerImpl" class="com.cloud.consoleproxy.StaticConsoleProxyManager" />
```

A quick way to do this is using sed. E.g. if you start your dev server using 'mvn -pl :cloud-clien-ui jetty:run', the project will create an applicationContext.xml ./client/tomcatconf/applicationContext.xml.in, so use

```
sed -e 's/com.cloud.consoleproxy.ConsoleProxyManagerImpl/com.cloud.consoleproxy.StaticConsoleProxyManager/g' .
/client/tomcatconf/applicationContext.xml.in -i
```

7. Restart the management server
8. Open the GUI, create a zone and add your secondary storage nfs://$NFS_IP/$NFS_EXPORT
9. Disable secondary storage and console proxy vms in the zone:

```
INSERT INTO `data_center_details` (`dc_id`, `name`, `value`) VALUES
        (<zone id>, 'enable.consoleproxy.vm', 'False');
INSERT INTO `data_center_details` (`dc_id`, `name`, `value`) VALUES
        (<zone id), 'enable.secstorage.vm', 'False');
```

Alternatively, use CloudStack cloudmonkey CLI

```
updateZone id=1 details[0].key=enable.consoleproxy.vm details[0].value='False' details[1].key=enable.secstorage.
vm details[1].value='False'
```

10. Run the console proxy service:

```
cd /var/cloudstack/console
./consoleproxy.sh
```

Run the secondary storage service:

```
cd /var/cloudstack/secondary
./secstorage.sh
```