KVM Virtualization in RHEL 6 Made Easy – Part 2

A Dell Technical White Paper

Jose De la Rosa

Dell Linux Engineering

THIS WHITE PAPER IS FOR INFORMATIONAL PURPOSES ONLY, AND MAY CONTAIN TYPOGRAPHICAL ERRORS AND TECHNICAL INACCURACIES. THE CONTENT IS PROVIDED AS IS, WITHOUT EXPRESS OR IMPLIED WARRANTIES OF ANY KIND.

© 2012 Dell Inc. All rights reserved. Reproduction of this material in any manner whatsoever without the express written permission of Dell Inc. is strictly forbidden. For more information, contact Dell.

Dell, the *DELL* logo, and the *DELL* badge, *PowerConnect*, and *PowerVault* are trademarks of Dell Inc. *Symantec* and the *SYMANTEC* logo are trademarks or registered trademarks of Symantec Corporation or its affiliates in the US and other countries. *Microsoft*, *Windows*, *Windows Server*, and *Active Directory* are either trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries. Other trademarks and trade names may be used in this document to refer to either the entities claiming the marks and names or their products. Dell Inc. disclaims any proprietary interest in trademarks and trade names other than its own.

May 2012

Contents

1. Introduction 5
1.1 Purpose of this document 5
1.2 Assumptions and Disclaimers 5
1.3 Terms & Conventions 6
2. Accessing the file systems on a VM that's not running (How to "rescue" your VM) 6
2.1 Kpartx 6
2.2 Virt-rescue 8
2.3 Guestfish 9
3. VM Snapshots 10
3.1 Disk Format: Raw vs. Qcow2 10
3.2 Creating Snapshots 10
3.3 Restoring from Snapshots 12
3.4 Deleting Snapshots 12
3.5 Converting disk format from raw to qcow2 12
4. Storage Pools and Volumes 14
4.1 Creating Storage Pools 15
4.1.1 Directory-based 15
4.1.2 Partition-based 16
4.1.3 LVM-based 17
4.1.4 NFS-based 18
4.1.5 iSCSI-based 20
4.2 Creating Storage Volumes 21
4.2.1 Directory-based 21
4.2.2 Partition-based 22
4.2.3 LVM-based 22
4.2.4 NFS-based 23
4.2.5 iSCSI-based 23

4.3 Creating VMs with Storage Pools and Volumes 24
4.4 Deleting Storage Volumes 25
4.5 Deleting Storage Pools 27
5. VM Migration 27
5.1 Requirements 28
5.2 Migration Examples 29
5.2.1 Live, Tunnelled 29
5.2.2 Live, Non-tunnelled 29
5.2.3 Offline, Non-Tunnelled 30
6. Conclusion 30
6.1 It's still easy! 30
6.2 What about oVirt or RHEV? 30
6.3 Additional Documentation 30

1. Introduction

1.1 Purpose of this document

This document is a follow-up to the Linux Kernel-based Virtual Machine (KVM) Virtualization whitepaper "KVM Virtualization in RHEL 6 Made Easy", where I described simple steps for setting up a virtualization environment using your Dell PowerEdge servers. In this whitepaper, I write about topics that go beyond basic creation and management of virtual machines (VMs). These topics include:

- Accessing the file systems on a VM that's not running (How to "rescue" your VM)

In a physical server, you can use a boot CD to boot the operating system in rescue mode. In a VM, you can't really boot in rescue mode. So if you forget the root password for example, you can instead access a VM's file system so you can edit any files as needed without providing a root password.

- VM Snapshots

Snapshots are useful for when you need to capture the state of a VM at a given moment in time. This can be quite handy to do for example before you install software that could be potentially harmful and corrupt the OS. Having a snapshot allows you to go back to a known workable copy of the operating system in your VM.

- Storage Pools and Volumes

Storage in a virtualization environment is managed using storage pools and storage volumes. There are different types of storage pools: directory-based, partition-based, LVM-based, NFS-based and iSCSI-based.

- VM Migration

VM migration is transferring the hosting of a VM from one host to another, provided the source and destination hosts share the same external storage. VM migration can take place without interruption or downtime to a VM's operating system.

1.2 Assumptions and Disclaimers

It is assumed that you already have a KVM virtualization environment and that you are fairly experienced with Linux and virtualization concepts. If you are not familiar with creating and managing a virtualization environment using KVM, please review the whitepaper "KVM Virtualization in RHEL 6 Made Easy".

Disclaimer #1: I don't use any Windows VMs in my lab, so I will only cover Linux VMs. Most steps described in this whitepaper can probably be used with Windows VMs, but I can't guarantee they will work.

Disclaimer #2: I don't use GUIs much, so I only cover instructions using a shell terminal. Come on folks this is Linux, GUIs are for Windows users ☺

Disclaimer #3: The steps described in this document are just one way to go about doing things. No claims are made about these steps being the most effective or smartest way to go about. I put together these instructions from reading online manuals and applying them successfully in my lab. I thought they could be of use to others who manage KVM virtual environments with RHEL 6.

1.3 Terms & Conventions

Let's agree on the terms & conventions I use here:

- Host - The hypervisor or physical server where all VMs are installed.
- VMs (Virtual Machines) - Virtual servers that are installed on top of a physical server. Some online documents refer to virtual machines as 'domains' or 'guests'.
- Text in `Courier New` font preceded by # are commands you type on a shell terminal.
- Text in `Courier New` font is command output or file contents.

2. Accessing the file systems on a VM that's not running (How to "rescue" your VM)

If you can't boot a VM, say that you somehow have a typo in */etc/fstab* or if you can't log in because you forgot the root password (same as you would on a physical server), there are a couple of ways you can go about to address this. Libvirt provides several tools to access a VM's file system when the VM is not running (doing it while the VM is running can cause disk corruption).

2.1 Kpartx

Kpartx will not work if the VM is using LVM. In this example, we have forgotten the root password and will need to clear it from */etc/shadow*. Our VM is called vm1 and its disk image file is */vm-images/vm1.img*:

1. Shut down the VM:

   ```
   # virsh shutdown vm1
   ```

2. Get an unused loop device:

   ```
   # losetup -f
   /dev/loop0
   ```

3. Map the VM image file to the loop device:

   ```
   # losetup /dev/loop0 /vm-images/vm1.img
   ```

4. Now that the image file has been mapped to the loop device, you can print the VM's partition table, which should help you locate the correct partition to mount:

   ```
   # fdisk -l /dev/loop0
   ```

```
Disk /dev/loop0: 10.7 GB, 10737418240 bytes
255 heads, 63 sectors/track, 1305 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0005a878

      Device Boot      Start         End      Blocks   Id  System
/dev/loop0p1   *           1          64      512000   83  Linux
Partition 1 does not end on cylinder boundary.
/dev/loop0p2              64         192     1024000   82  Linux swap /
Solaris
Partition 2 does not end on cylinder boundary.
/dev/loop0p3             192        1306     8948736   83  Linux
```

5. In order to mount the VM's partitions, we need to create partition mappings:

   # kpartx -av /dev/loop0

   ```
   add map loop0p1 (253:3): 0 1024000 linear /dev/loop0 2048
   add map loop0p2 (253:5): 0 2048000 linear /dev/loop0 1026048
   add map loop0p3 (253:6): 0 17897472 linear /dev/loop0 3074048
   ```

6. Mount the desired VM partition. In this example, our root ('/') file system is mounted on */dev/sda3*, which is this case has been associated with /dev/loop0p3:

   # mount /dev/mapper/loop0p3 /mnt

7. Let's remove the root password from */etc/shadow*, which is accessed as */mnt/etc/shadow*:

   # vi /mnt/etc/shadow

8. Once done, let's remove our mappings so we can start our VM:

   ```
   # umount /mnt
   # kpartx -dv /dev/loop0
   # losetup -d /dev/loop0
   ```

9. Start the VM:

   # virsh start vm1

   Once the VM is up, login as root without providing a password. Enter a new password and don't forget it this time!

2.2 Virt-rescue

For VMs that use LVM, we can use the virt-rescue tool. This tool is part of the optional *libguestfs-tools* package and is a bit fancier in that it gives you a rescue shell along with some useful tools to "rescue" your VM:

1. Shutdown the VM. You can use virt-rescue on a running VM, but you will not be able to edit any files, you will have to access it read-only. Since in our example we need to edit */etc/shadow*, we will have to shut down our VM first:

   ```
   # virsh shutdown vm1
   ```

2. Run virt-rescue. Following are several examples.

   Rescue a stopped VM by providing its name:

   ```
   # virt-rescue --domain vm1
   ```

   Rescue a stopped VM by providing its image file name:

   ```
   # virt-rescue --add /vm-images/vm1.img
   ```

   Rescue a stopped VM by providing the storage device name where it is located:

   ```
   # virt-rescue --add /dev/sdd
   ```

   Rescue a running VM by providing its image file name. Notice we're doing it read-only:

   ```
   # virt-rescue --ro --add /vm-images/vm1.img
   ```

3. When you run virt-rescue on a virtual machine or disk image, you are placed in an interactive bash shell where you can use many ordinary Linux commands. The files and directories that you see are from the rescue appliance. In order to access the file systems on the VM, you must first mount it. There is an empty directory called /sysroot where you can mount file systems.

   If the VM is using LVM, let's first find out what logical volumes are on it:

   ```
   # ><rescue> lvs
     LV      VG         Attr   LSize  Origin Snap%  Move Log Copy%  Convert
     lv_root vg_mygroup -wi-a- 25.74g
     lv_swap vg_mygroup -wi-a-  3.77g
   ><rescue>
   ```

4. Since we are looking for the root partition, mount lv_root:

   ```
   ><rescue> mount /dev/vg_mygroup/lv_root /sys
   sys/     sysroot/
   ><rescue> mount /dev/vg_mygroup/lv_root /sysroot
   ```

```
[  269.175648] EXT4-fs (dm-0): mounted filesystem with ordered data mode.
Opts:
```

5. Edit */etc/shadow* in the VM:

```
# vi /sysroot/etc/shadow
```

6. When done, unmount the file system and exit the virt-rescue shell:

```
><rescue> umount /sysroot
><rescue> exit
exit

virt-rescue: Syncing the disk now before exiting ...

[  396.890049] md: stopping all md devices.
[  397.990525] Restarting system.
[  397.990825] machine restart
virt-rescue: child process died unexpectedly
```

7. Start the VM:

```
# virsh start vm
```

2.3 Guestfish

A third method for accessing a VM's file system is with the 'guestfish' tool, which can be installed by installing the optional guestfish package. This tool can also be used with VMs that use LVM.

1. As usual, it is highly recommended to stop the VM:

```
# virsh shutdown vm1
```

2. Go to the directory where the VM image files are located:

```
# cd /vm-images
```

3. In its simplest form, guestfish can be run interactively. I won't go through every command, as they are self-explanatory:

```
# guestfish
><fs> add vm1.img
><fs> run
><fs> list-filesystems
/dev/vda1: ext4
/dev/vg_dhcp162238/lv_root: ext4
/dev/vg_dhcp162238/lv_swap: swap
><fs> mount /dev/vg_dhcp162238/lv_root /
```

```
><fs> vi /etc/shadow
><fs> exit
```

4. Guestfish can also be used from shell scripts. For more details, please refer to the guestfish man page.

3. VM Snapshots

Snapshots take the disk, memory and device state of a VM at a specific moment in time. VM snapshots can preserve a VM's state before a potentially dangerous operation, so you can always go back to it if needed. Snapshots are identified with a unique name and can be taken with the VM running (live snapshots), although it is highly recommended that they be taken when a VM is suspended or shutdown (with no disk I/O), to guarantee a clean snapshot.

3.1 Disk Format: Raw vs. Qcow2

Before we go through some examples, let's talk about VM disk formats, since libvirt snapshots can only be performed on certain type of disk formats. There are several types of VM disk formats, but the most common are raw and qcow2 (copy-on-write). Raw disks are presented as-is to the VM without any disk layering and do not support snapshots, whereas qcow2 disks support a range of special features including snapshots, compression and encryption.

Raw disks have better I/O performance than qcow2 because the VM disk space is pre-allocated before the VM is created, as opposed to qcow2, where VM disk space is thinly provisioned and allocated on-the-go as the VM requires it. If you create a VM with a raw disk of 10 GB, you'll see that the disk image size for the VM will be exactly 10 GB, whereas for a VM with a qcow2 disk the actual file size will be a fraction of 10 GB (say 2-3 GB) and it will grow as you occupy disk space inside the VM.

Snapshots can be considered a "backup" of a VM's disk, although no actual bits are copied or backed up. The system simply remembers the state of a VM and takes advantage of the copy-on-write capabilities of qcow2. So snapshots are simply a record of where bits stood at a given point in time, not an actual copy of the bits.

3.2 Creating Snapshots

In this example, we create a disk snapshot for a VM named 'vm1':

1. Verify that the VM disk format is 'qcow2' and not 'raw'. You can use either of two methods:

   a. The 'file' command:

   ```
   # file /vm-images/vm1.img
   /vm-images/vm1.img: Qemu Image, Format: Qcow, Version: 2
   ```

   b. The 'qemu-img' command:

   ```
   # qemu-img info /vm-images/vm1.img
   image: /vm-images/vm1.img
   ```

```
file format: qcow2
virtual size: 10G (10737418240 bytes)
disk size: 3.2G
cluster_size: 65536
```

2. For reference, print list of existing snapshots:

```
# virsh snapshot-list vm1
```

```
 Name                 Creation Time             State
------------------------------------------------------------
```

As can be seen, there are no existing snapshots for vm1.

3. Shut down the VM, even though snapshots can be taken on a live VM, it is highly recommended it be shut down to avoid any disk I/O and ensure a clean snapshot.

```
# virsh shutdown vm1
```

4. Take snapshot:

```
# virsh snapshot-create vm1
Domain snapshot 1334856484 created
```

5. Let's take a look:

```
# virsh snapshot-list vm1
```

```
 Name                 Creation Time             State
------------------------------------------------------------
 1334856484           2012-04-19 11:28:04 -0500 shutoff
```

6. We have now created a disk snapshot that we can always revert back to if we eventually corrupt our file system and need to go back to a clean state.

7. We can keep creating more snapshots as needed. After taking 3 more snapshots, we now have a total of 4 snapshots:

```
# virsh snapshot-list vm1
```

```
 Name                 Creation Time             State
------------------------------------------------------------
 1334856484           2012-04-19 11:28:04 -0500 shutoff
 1334856609           2012-04-19 12:30:09 -0500 running
 1334856635           2012-04-19 13:30:35 -0500 running
 1334864277           2012-04-19 14:37:57 -0500 shutoff
```

- The first and fourth snapshots were taken while the VM was shut down.
- The second and third snapshots were taken while the VM was running.
- Snapshots on stopped VMs will be a lot faster than on running VMs.

- You can take as many snapshots as the VM's image file will fit. As mentioned before a snapshot is not a copy of an image, so for example taking a snapshot of a disk that is 3 GB in size does not mean it will take up an additional 3 GB.

3.3 Restoring from Snapshots

In this example, we are going to revert back to a snapshot image we took in the previous section. We wish to go back to the 3rd snapshot (taken at 13:30:35) with name 1334856635. In order to revert back to a snapshot, you MUST shutdown the VM or the snapshot revert command will fail:

```
# virsh shutdown vm1
# virsh snapshot-revert vm1 1334856635
```

If you are reverting back to a snapshot taken while the VM was shut down, then you will have to start the VM. If you are reverting back to a snapshot taken while the VM is was running, then the VM will already be running after you revert back to it.

3.4 Deleting Snapshots

Deleting snapshots is very straight-forward. Once you are sure that you no longer need a snapshot, it is good practice to remove it. From our previous example, let's remove the second snapshot:

```
# virsh snapshot-list vm1

 Name                 Creation Time             State
------------------------------------------------------------
 1334856484           2012-04-19 11:28:04 -0500 shutoff
 1334856609           2012-04-19 12:30:09 -0500 running
 1334856635           2012-04-19 13:30:35 -0500 running
 1334864277           2012-04-19 14:37:57 -0500 shutoff

# virsh snapshot-delete vm1 1334856609
Domain snapshot 1334856609 deleted

# virsh snapshot-list vm1

 Name                 Creation Time             State
------------------------------------------------------------
 1334856484           2012-04-19 11:28:04 -0500 shutoff
 1334856635           2012-04-19 13:30:35 -0500 running
 1334864277           2012-04-19 14:37:57 -0500 shutoff
```

3.5 Converting disk format from raw to qcow2

It is easy to convert from a raw VM image (which doesn't support libvirt snapshots) to qcow2 format. In this example, we convert an image file of size 10 GB for a VM called vm2.

1. Shut down the VM:

```
# virsh shutdown vm2
```

2. Let's take a look at the image file format:

```
# cd /vm-images
# qemu-img info vm2.img
image: vm2.img
file format: raw
virtual size: 10G (10737418240 bytes)
disk size: 10G
```

3. The conversion process actually makes a qcow2 copy of the raw image and leaves the raw image intact. The command is:

```
# qemu-img convert -f raw vm2.img -O qcow2 vm2.qcow2
```

   - The new image file extension is irrelevant; we use "qcow2" to help us identify the different image format.
   - The conversion process will take a few minutes, depending on the size of the image file.

4. Let's take a look at the new image file format:

```
# qemu-img info vm2.qcow2
image: vm2.qcow2
file format: qcow2
virtual size: 10G (10737418240 bytes)
disk size: 3.2G
cluster_size: 65536
```

   - Notice that the virtual size remains unchanged, it is still 10 GB.
   - The disk size (as the OS sees it) will be less than 10 GB. In our case, it is 3.2 GB. If you run the 'ls -l' command on the file, its reported size will match the "disk size" reported here. As you add files inside the VM disk, this file size will increase, up to a maximum of 10 GB.

5. Now we need to modify the VM's XML definition file to use the new image file that will now be used:

```
# virsh edit vm2
```

   Look in the <devices> section and search for the old image format entry ("raw") and the image file name ("vm2.img"). They should be next to each other:

```
<driver name='qemu' type='raw' cache='none'/>
<source file='/vm-images/vm2.img'/>
```

   Change to:

```
driver name='qemu' type='qcow2' cache='none'/>
source file='/vm-images/vm2.qcow2'/>
```

Save file and exit.

6. That's it, start the VM, which will now use the qcow2 image file. As mentioned before, the raw image is left alone, you can save it for later use or delete altogether.

4. Storage Pools and Volumes

Storage pools are storage entities used by libvirt to provide storage to VMs. A storage pool can be on either local or external storage, and can be a file, a directory or a storage device such as an LVM volume or an iSCSI LUN. **Storage volumes** are components of storage pools, or put another way, storage pools are divided into volumes. Storage volumes are the virtual disks used by VMs.

Defining storage pools and volumes are an important part of KVM virtual environments, and depending on the type of environment, they are a requirement. On a simple environment (i.e. standalone server with local storage), defining pools and volumes may not be necessary, but when working with complex environments (i.e. several hosts with external storage) creating storage pools and volumes will greatly simplify the administration of your virtual resources.

We can depict storage pools and volumes as follows:

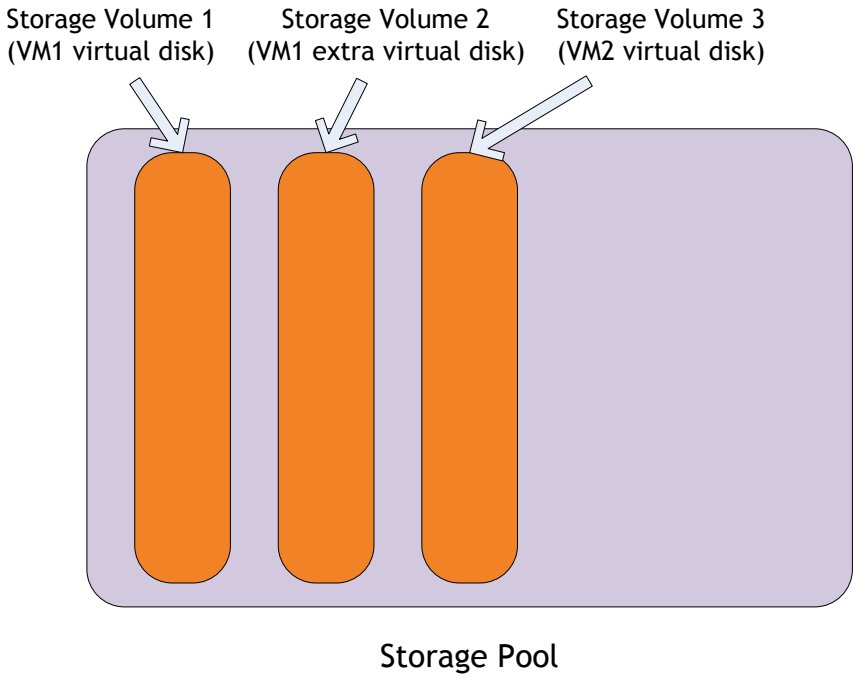

Figure 1 – Relationship between storage pools and storage volumes

There are several types of storage pools, which can basically be categorized into either local or networked:

Local storage pools: Storage is local to a server, either on local disks or directly-attached. Local pools include:

- Directory-based: Pool is just a directory on the local server. For example: */var/lib/libvirt/images* or */vm-images*.
- Partition-based: Pool is a partition on a local disk. For example: */dev/sda7* or */dev/sdb1*.
- LVM-based: Pool is an LVM logical volume. For example: */dev/vg_images/vms*.

Local storage pools are usually sufficient for single-server deployments, which can include development and test environments.

Network storage pools: Storage is network-based. Network storage pools include:

- NFS-based: Pool is an NFS-mount from another server on a local mount point.
- iSCSI-based: Pool is an iSCSI LUN, which the virtualization host has access to.

Network storage pools can also be used on single-server environments, but are better suited for multi-server environments where shared storage is required in order to provide a highly-available environment. Network storage pools are required to perform VM migration, which we will discuss in the next section.

4.1 Creating Storage Pools

In this section I go over specific examples on how to create different types of storage pools, including properties for each one. After a storage pool is created, a definition XML file for it will also be created in */etc/libvirt/storage*.

One very important item to keep in mind before creating a storage pool is to adequately determine what size it should be. I could not find any documentation on how to expand the size of a storage pool, so it is critical that it be the right size from the get go. Although you can have more than one storage pool, VM migration (which we will discuss in the next section) can only take place *within* the same storage pool.

4.1.1 Directory-based

Directory-based pools are the easiest to understand. You can designate any directory in a server's file system as a storage pool and VM's disks will be represented by files in that directory. It is recommended that the directory you want to use be on its own partition and not use OS partitions, otherwise you risk filling up the file system which could render the virtualization host inoperable.

In this example, we designate the directory */vm-images* as a directory-based storage pool. If you read my previous KVM whitepaper ("KVM Virtualization in RHEL 6 Made Easy") you will notice that this is the same directory we used in our examples there, except we did not bother creating a storage pool in those examples. Indeed, you can get away with not creating directory-based storage pools for standalone hosts, but it is good practice to create them anyway since it will simplify the management of your virtualization environment later on.

1. Verify existing storage pools. For simplicity purposes, we do not have any pools defined:

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
```

2. Create storage pool:

```
# virsh pool-define-as mydirpool dir - - - - /vm-images
Pool mydirpool defined
```

- 'mydirpool' is the name of the storage pool
- 'dir' is what defines the type of storage pool
- '*/vm-images*' is the directory name
- The dashes ('-') are for fields we don't for this type of storage pool

3. Build storage pool, which among other things it creates the directory if it doesn't exist and sets the correct SELinux security context:

```
# virsh pool-build mydirpool
Pool mydirpool built
```

4. Verify pool was created:

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
mydirpool            inactive   no
```

5. Activate storage pool and set to autostart:

```
# virsh pool-start mydirpool
# virsh pool-autostart mydirpool
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
mydirpool            active     yes
```

4.1.2 Partition-based

Partition-based storage pools are very similar to directory-based storage pools, with the distinction that partition-based pools occupy their own partition (as recommended) separate from OS files. Similar to directory-based storage pools, there needs to be a file system on a partition-based storage pool (i.e. ext3 or ext4) prior to creating it.

In this example, we use partition /dev/sdb1, where we have previously created an ext4 file system. Please note that we do NOT have to mount the file system (it will be automatically mounted when we create the pool).

1. Verify existing storage pools. For simplicity purposes, we do not have any pools defined:

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
```

2. Create storage pool:

```
# virsh pool-define-as mypartpool fs - - /dev/sdb1 - /vm-images
Pool mypartpool defined
```

- 'mypartpool' is the name of the storage pool
- 'fs' is what defines the type of storage pool
- /dev/sdb1 is the partition to be used
- '*/vm-images*' is the directory where the partition will be mounted. If the directory doesn't exist, virsh will create it for you
- The dashes ('-') are for fields we don't for this type of storage pool

3. Build storage pool, which among other things it creates the mount point if it doesn't exist and sets the correct SELinux security context:

```
# virsh pool-build mypartpool
Pool mypartpool built
```

4. Verify pool was created:

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
mypartpool           inactive   no
```

5. Activate pool and set to autostart:

```
# virsh pool-start mypartpool
# virsh pool-autostart mypartpool
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
mypartpool           active     yes
```

4.1.3 LVM-based

LVM-based storage pools are logical volumes which can reside on any volume group. It's highly recommended that a separate volume group be used from the OS system files. Unlike directory-based and partition-based storage pools, we don't need to create a file system on the LVM pool, since the VM disk will be on a *block* device instead of a *file* device, so there's no need for a file system.

LVM-based storage pools can be quite useful as we can take advantage of the features provided by LVM, including logical volume manipulation, snapshots and mirroring.

In this example, we create an LVM-based storage pool on an existing volume group called my_vg, which is on physical volume /dev/sdc1.

1. Verify existing storage pools. For simplicity purposes, we do not have any pools defined:

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
```

2. Create storage pool:

```
# virsh pool-define-as mylvmpool logical - - /dev/sdc1 my_vg /dev/my_vg
Pool mylvmpool defined
```

- 'mylvmpool' is the name of the storage pool
- 'logical' is what defines the type of storage pool
- */dev/sdc1* is the physical volume
- *'my_vg'* is the name of the volume group
- The dashes ('-') are for fields we don't for this type of storage pool

3. Build storage pool, which among other things it creates the volume group:

```
# virsh pool-build mylvmpool
Pool mylvmpool built
```

4. Verify pool was created:

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
mylvmpool            inactive   no
```

5. Activate pool and set to autostart:

```
# virsh pool-start mylvmpool
# virsh pool-autostart mylvmpool
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
mylvmpool            active     yes
```

4.1.4 NFS-based

NFS-based storage pools provide the flexibility of sharing the same storage to several virtualization hosts. Thus, it can be used on clustered environments where features such as VM live migration can be supported. As in directory-based and partition-based storage pools, we will need to create a file system on the NFS share (obviously, since NFS stands for Network File System!).

In this example, we create an NFS-based storage pool using an NFS share from another RHEL 6 server. Though I will not go over the steps to create an NFS share on RHEL (you should know that already!), I will go over an access permission issue that was preventing me from creating VMs on an NFSv4 share.

1. If creating an NFSv4 share in a RHEL 6 server, be sure to edit */etc/idmapd.conf* and define a domain name on the NFS server and the KVM hosts (NFS clients) in the [General] section towards the top of the file. The domain name, which can be the same as your DNS domain, should be the same on all servers:

   ```
   [General]
   ...
   Domain = yourdomain.com
   ```

   For this change to take effect, restart the rpcidmapd service:

   ```
   # service rpcidmapd restart
   ```

   Before I specified a domain on my servers, files created on the NFS shares had ownership *nobody.nobody*, regardless of what user created it (even root). This ownership issue was affecting the creation of VM images on the NFS share, which was fixed after I specified a domain in */etc/idmapd.conf*. My VM images now have ownership *qemu.qemu*, as it should be.

2. Verify existing storage pools. For simplicity purposes, we do not have any pools defined:

   ```
   # virsh pool-list --all
   Name                 State      Autostart
   -----------------------------------------
   ```

3. Create storage pool:

   ```
   # virsh pool-define-as mynfspool netfs <nfsserver> /nfs - - /nfsmount
   Pool mynfspool defined
   ```

   - 'mynfspool' is the name of the storage pool
   - 'netfs' is what defines the type of storage pool
   - 'nfsserver' is the name (or IP address) of the NFS server
   - /nfs is the shared directory on the NFS server
   - '/*nfsmount*' is the name of the NFS mount point on the virtualization host
   - The dashes ('-') are for fields we don't for this type of storage pool

4. Build storage pool, which among other things it creates the NFS mount point:

   ```
   # virsh pool-build mynfspool
   Pool mynfspool built
   ```

5. Verify pool was created:

   ```
   # virsh pool-list --all
   Name                 State      Autostart
   -----------------------------------------
   mynfspool            inactive   no
   ```

6. Activate pool and set to autostart:

```
# virsh pool-start mynfspool
# virsh pool-autostart mynfspool
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
mynfspool            active     yes
```

4.1.5 iSCSI-based

iSCSI-based storage pools also provide the flexibility of sharing the same storage to several virtualization hosts. Unlike directory-based and partition-based storage pools, we don't need to create a file system on iSCSI LUNs. As with LVM-based storage pools, the VM disk will be on a *block* device instead of a *file* device.

In this example, we create an iSCSI-based storage pool using an iSCSI LUN from an EqualLogic PS4100E array (though any EqualLogic or Dell iSCSI PowerVault array will do). You could also use an iSCSI LUN using a RHEL 6 server, whatever resources are at your disposal.

I am assuming that you are comfortable with iSCSI terminology and have used 'iscsiadm' before. If not, I recommend you brush up before proceeding. Before you can create an iSCSI-based storage pool, the iSCSI LUNs must already be visible (discovered) using the usual 'iscsiadm' commands. Please note that you don't need to login to the LUNs, just discover them with "*iscsiadm -m discovery -t st -p <ip-addr>*" so you can get the target IQN, which you will need below.

1. Verify existing storage pools. For simplicity purposes, we do not have any pools defined:

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
```

2. Create storage pool:

```
# virsh pool-define-as myiscsipool iscsi <targethost> - <targetiqn> - /dev/disk/by-path
Pool myiscsipool defined
```

- 'myiscsipool' is the name of the storage pool
- 'iscsi' is what defines the type of storage pool
- 'targethost' is the name or IP address of the iSCSI target host.
- 'targetiqn' is the name of the target IQN, which should be long string that looks something like iqn.2001-05.com.equallogic:8-cb2b76-f49fa6b6d-444002281704f565-myvol
- /dev/disk/bypath is written as-is
- The dashes ('-') are for fields we don't for this type of storage pool

3. Build storage pool:

```
# virsh pool-build myiscsipool
Pool myiscsipool built
```

4. Verify pool was created:

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
myiscsipool          inactive   no
```

5. Activate pool and set to autostart:

```
# virsh pool-start myiscsipool
# virsh pool-autostart myiscsipool
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
myiscsipool          active     yes
```

4.2 Creating Storage Volumes

Storage volumes are created on top of storage pools. As mentioned before, storage volumes are the VM's virtual disks, so there are a few things to consider before creating storage volumes:

a. How much disk space does my VM need?
b. What storage format does it need to be, raw or qcow2?
c. If going to create snapshots, how much extra space should be allocated?

The command and arguments to create a storage volume is the same regardless of the type of storage pool it is created on. I provide examples for each type of storage pool, with a mix of both raw and qcow2 formats.

4.2.1 Directory-based

In this example, we create a volume on a directory-based storage pool:

1. Create volume of size 10 GB in raw format:

```
# virsh vol-create-as mydirpool vm-vol 10G
Vol vm-vol created
```

- 'mydirpool' is the storage pool used
- 'vm-vol' is the volume name; even though the name can be anything you'd like, it is a good idea to give it a descriptive name, including the name of the VM that it will be used for.
- When specifying the volume size, you can use the k, M, G or T suffix.

2. Get storage volume information:

```
# virsh vol-info --pool mydirpool vm-vol
Name:           vm-vol
Type:           file
Capacity:       10.00 GB
```

```
Allocation:     10.00 GB
```

- Since the volume is on a directory-based storage pool, it is created as a regular file.

4.2.2 Partition-based

In this example, we create a volume on a partition-based storage pool:

1. Create volume of size 12 GB with an additional 15 GB to be used for VM snapshots (so we must use the qcow2 format):

```
# virsh vol-create-as mypartpool vm-vol 27G --allocation 12G \
  --format qcow2
Vol vm-col created
```

- 'mypartpool' is the storage pool used
- 'vm-vol' is the volume name; even though the name can be anything you'd like, it is a good idea to give it a descriptive name, including the name of the VM that it will be used for.
- When specifying the volume size, you can use the k, M, G or T suffix.
- The total size of the volume is 12GB + 15GB = 27GB

2. Get storage volume information:

```
# virsh vol-info --pool mypartpool vm-vol
Name:           vm-vol
Type:           file
Capacity:       27.00 GB
Allocation:     136.00 KB
```

- Since the volume is on a partition-based storage pool, it is created as a regular file of size 12 GB, which can grow to up to 27 GB depending on the number of snapshots created.
- When the OS is installed on the VM, the "Allocation" field will grow as needed. For now, since the disk is empty (except for metadata) its real size is near zero.

4.2.3 LVM-based

In this example, we create a volume on an LVM-based storage pool:

1. Create volume of size 20 GB in raw format:

```
# virsh vol-create-as mylvmpool vm-vol 20G
Vol vm-vol created
```

- 'mylvmpool' is the storage pool used
- 'vm-vol' is the volume name; even though the name can be anything you'd like, it is a good idea to give it a descriptive name, including the name of the VM that it will be used for.
- When specifying the volume size, you can use the k, M, G or T suffix.

2. Get storage volume information:

```
# virsh vol-info --pool mylvmpool vm-vol
Name:           vm-vol
Type:           block
Capacity:       20.00 GB
Allocation:     20.00 GB
```

- Since the volume is on a directory-based storage pool, it is created as block storage.

4.2.4 NFS-based

In this example, we create a volume on an NFS-based storage pool:

1. Create volume of size 10 GB with an additional 10 GB to be used for VM snapshots (so we must use the qcow2 format):

```
# virsh vol-create-as mynfspool vm-vol 20G --allocation 10G \
  --format qcow2
Vol vm-col created
```

- 'mynfspool' is the storage pool used
- 'vm-vol' is the volume name; even though the name can be anything you'd like, it is a good idea to give it a descriptive name, including the name of the VM that it will be used for.
- When specifying the volume size, you can use the k, M, G or T suffix.
- The total size of the volume is 10GB + 10GB = 20GB

2. Get storage volume information:

```
# virsh vol-info --pool mynfspool vm-vol
Name:           vm-vol
Type:           file
Capacity:       20.00 GB
Allocation:     136.00 KB
```

- Since the volume is on a partition-based storage pool, it is created as a regular file of size 10 GB, which can grow to up to 20 GB depending on the number of snapshots created.
- When the OS is installed on the VM, the "Allocation" field will grow as needed. For now, since the disk is empty (except for metadata) its real size is near zero.

4.2.5 iSCSI-based

In this example, we create a volume on an iSCSI-based storage pool:

1. Create volume of size 10 GB in raw format:

```
# virsh vol-create-as myiscsipool vm-vol 10G
error: Failed to create vol vm-vol
error: this function is not supported by the connection driver: storage
pool does not support volume creation
```

As it turns out, we can't create storage volumes on iSCSI-based pools. The entire iSCSI LUN is automatically setup as a volume when it was setup as a storage pool.

2. Get name of storage volume:

```
# virsh vol-list myiscsipool
Name                 Path
-----------------------------------------
unit:0:0:0           /dev/disk/by-path/ip-10.9.160.171:3260-iscsi-
iqn.2001-05.com.equallogic:8-cb2b76-18adb6c6d-b820022aaed4fa44-vol4-lun-0
```

- 'unit:0:0:0' is the volume name, which is somewhat cryptic but that's the name given by the system.
- The long path name includes the name of the iSCSI target, which will be unique and different for each case.

3. Get storage volume information:

```
# virsh vol-info --pool myiscsipool unit:0:0:0
Name:           unit:0:0:0
Type:           block
Capacity:       50.01 GB
Allocation:     50.01 GB
```

- Since the volume is on an iSCSI-based storage pool, it is created as block storage.

4.3 Creating VMs with Storage Pools and Volumes

Now that we have introduced the concept of storage pools and storage volumes, we can use them when creating VMs with *virt-install*. In section 3 of the whitepaper "KVM Virtualization in RHEL 6 Made Easy", I described this command to create VMs:

```
# virt-install \
--network bridge:br0 \
--name vm1 \
--ram=1024 \
--vcpus=1 \
--disk path=/vm-images/vm1.img,size=10 \
--graphics none \
--location=http://my.server.com/pub/rhel6.1/install-x86_64/ \
--extra-args="console=tty0 console=ttyS0,115200"
```

In the command above, the argument to "`--disk`" is a file name and the size of the file, which will be the name of the VM virtual disk. This of course works, but it is rather simple and quite limiting, as it can only be used for VMs whose disk image is represented by a file in raw format.

A more appropriate parameter specification for "`--disk`" would be:

```
--disk "vol=<pool>/<volume>"
```

We do not need to specify the size or the storage format (raw vs. qcow2) of the VM image, since it's already defined by the storage volume.

For example, for a partition-based storage pool and volume, the previous command becomes:

```
# virt-install \
--network bridge:br0 \
--name vm1 \
--ram=1024 \
--vcpus=1 \
--disk "vol=mypartpool/vm-vol" \
--graphics none \
--location=http://my.server.com/pub/rhel6.1/install-x86_64/ \
--extra-args="console=tty0 console=ttyS0,115200"
```

4.4 Deleting Storage Volumes

Deleting storage volumes is equivalent to deleting VM disks, so extreme care should be taken before proceeding to delete a storage volume. A running VM using a volume does not prevent you from deleting the volume, so in a way, removing a volume can be thought of as pulling the hard drive off a physical server (well, sort of, I'll illustrate shortly).

When you are done using a VM and are ready to reclaim its disk space, deleting a storage volume is straight-forward. Before removing a storage volume, the VM that is using the volume should be shut down and undefined ("*virsh undefine* <*vm*>"). As it turns out, you can remove a storage volume on a running VM, but cleaning things up afterwards can be a bit messier.

In our first example, we delete a directory-based storage volume (which was created on a directory-based storage pool). We have already shutdown and undefined the VM that was using it:

1. List volumes in pool:

```
# virsh vol-list mydirpool
Name                 Path
-----------------------------------------
vm-vol               /mydirpool/vm-vol
```

2. Remove volume:

```
# virsh vol-delete --pool mydirpool vm-vol
Vol vm-vol deleted
```

3. The storage volume should be gone:

```
# virsh vol-list mydirpool
Name                 Path
-----------------------------------------
```

In our second example, we are going to remove a volume on a running VM, just to see what happens and how to clean things up. The volume is on a partition-based storage pool and we have created a VM (called 'myvm') on the volume using the instructions in the previous section:

1. List running VMs:

```
# virsh list --all
 Id Name                 State
----------------------------------
  1 myvm                 running
```

2. List volumes in storage pool:

```
# virsh vol-list mypartpool
Name                 Path
-----------------------------------------
vm-vol               /mypartpool/vm-vol
```

3. Remove volume, which is being used by the VM listed in step 1:

```
# virsh vol-delete --pool mypartpool vm-vol
Vol vm-vol deleted
```

4. The storage volume should be gone. So as you can see, there is nothing stopping us from deleting a storage volume being used by a running VM:

```
# virsh vol-list mydirpool
Name                 Path
-----------------------------------------
```

5. So what happened to the VM? The VM remains running even though its underlying disk is gone! How is that? The KVM process for the VM is still running, and the VM image disk remains in memory. What happens is we write something to the disk? It gets written. What happens if we reboot the VM? The VM reboots just fine, as the KVM process runs uninterrupted. What happens if we power off the VM and then try to restart it? You guessed it, it fails since the VM disk is now gone:

```
# virsh shutdown myvm
Domain myvm is being shutdown

# virsh start myvm
error: Failed to start domain myvm
error: Unable to allow access for disk path /mypartpool/vm-vol: No such
file or directory
```

6. So now, we must simply undefine the VM:

```
# virsh undefine myvm
Domain myvm has been undefined
```

It is unclear how long a VM can keep running without a 'disk' to write to. Also, I did not test this scenario on all types of storage pools, so it's possible it behaves differently on LVM, NFS or iSCSI pools. As mentioned before, take great care before removing a storage volume, and it is highly recommended that the VM using it be shut down and undefined first.

4.5 Deleting Storage Pools

As with storage volumes, deleting storage pools should be done carefully, and only after a thorough review of your virtual environment. It is highly recommended that all VMs and storage volumes be removed and undefined before removing a storage pool, as leaving running VMs or active volumes behind can leave your environment in an unclean state, and in most cases, it will wipe off all VMs and volumes contained in the storage pool.

Deleting a storage pool is straight-forward, and is the same command regardless of the type of storage pool. In this simple example, we will remove the NFS-based storage pool that we created previously:

1. List all storage pools available. For simplicity purposes, we only have one pool defined:

```
# virsh pool-list
Name                 State      Autostart
-----------------------------------------
mynfspool            active     yes
```

2. Delete storage pool:

```
# virsh pool-destroy mynfspool
Pool mydirpool destroyed
```

3. We must also undefine it:

```
# virsh pool-undefine mynfspool
Pool mynfspool has been undefined
```

4. Storage pool should now be gone:

```
# virsh pool-list
Name                 State      Autostart
-----------------------------------------
```

I will not go over examples of what happens when you remove a storage pool with running VMs or existing storage volumes still left in it. From the tests I ran, the behavior is different depending on the type of storage pool. However, as mentioned before, the removal process should be clean and in order: first delete the VM, followed by the storage volume and finally the storage pool.

5. VM Migration

VM Migration is the ability to transfer the hosting of a VM from one server to another. Live Migration is the ability to do it without pausing or suspending the VM. Since both the source and destination hosts

are using the same (shared) storage, the VM's disk is not moved or transferred; only the VM's memory and virtualized devices are transferred from one host to another.

VM migration can be used for different purposes. Though its most useful purpose is in case of hardware failure, it can also be used for load-balancing (in case a host has more running VMs than others) and for energy savings (consolidate all VMs on one host and shut down other hosts during off-peak periods).

VM Migration types include:

- Offline – VM is suspended while it's migrated. Faster than live migration since pages are not monitored for changes during the migration.
- Live – VM is migrated without being suspended. Can be slower than live migration since memory pages are migrated in order and KVM continually monitors them to keep track of any changes between source and destination hosts.
- Tunnelled – migration is done encrypted through SSH port.
- Non-tunnelled – migration is done non-encrypted. Requires opening firewall ports 49152-49215 in the destination host (unless a specific port is given).

Because VM Migration requires more than one host and because of the shared storage requirements, things become a bit more complicated since you must create and manage storage pools in more than one host. Please note that an in-depth discussion about setting up clustered environments is beyond the scope of this document.

5.1 Requirements

For simplicity purposes, we will limit our examples with only two KVM hosts. To setup a KVM virtualization environment where VMs can be migrated from one host to another, we need the following:

1. **Two Hosts:** Two Dell PowerEdge servers with adequate CPU and memory to host VMs. It is highly recommended that both servers be running the same RHEL 6 version, same kernel version and same patch level. In my lab, my servers were on the same network subnet. Also, both servers must have either AMD or Intel processors and be of the same processor family (i.e. Intel Nehalem or AMD G2), but they don't necessarily need to have the same speed. I've read documentation where VM Migration in KVM is supported between Intel and AMD-based servers, but I did not test it.

2. **Shared Storage:** The shared storage must be accessible from both hosts. If using iSCSI storage, any Dell EqualLogic or PowerVault iSCSI array with sufficient storage will do. Using a RHEL 6 host as an iSCSI target should work fine, but I did not test it. If using NFS storage, another PowerEdge server can be setup as an NFS server. I don't recommend configuring one of the KVM hosts as an NFS server, since the goal here is to keep services isolated as much as possible.

3. **Storage Pool:** You must create a storage pool with the same properties and preferably the same names on each server. For NFS storage, the NFS mount point must be exactly the same on both servers.

4. **VMs:** Last but certainly not least, you need a VM that has been created on an NFS or iSCSI-based storage pool.

5.2 Migration Examples

In the following examples, we have a VM called vm1 on a shared storage pool running on source host kvmhost1, which we will migrate to destination host kvmhost2. Please note that in all instances, the VM to be migrated must be running.

5.2.1 Live, Tunnelled

In this example, we do a live, tunnelled (encrypted) migration:

```
# virsh migrate vm1 qemu+ssh://kvmhost2/system --live \
       --undefinesource --persistent --tunnelled --p2p --verbose
```

- `--live`

  Migrate without suspending the VM. For an offline migration, omit this parameter.

- `--undefinesource --persistent`

  Providing these 2 parameters ensures a clean migration in regards to the VM definition in both hosts. The '*undefinesource*' parameter removes the VM definition file in */etc/libvirt/qemu* in the source host, whereas the '*persistent*' parameter creates it in the destination host. These parameters are optional, so if you are not careful you could wind up with the VM defined in both hosts or in neither one, which could be very troublesome down the road.

- `--tunnelled --p2p`

  Perform migration securely through an SSH tunnel, using a peer-to-peer connection. These parameters must be provided together.

- `--verbose`

  Print VM migration status, from 0 to 100%.

Once the migration is complete, the VM should be running on the destination host (and not on the source host). Also, verify that the XML definition file was created on the destination host and removed from the source host.

5.2.2 Live, Non-tunnelled

In this example, we do a live, non-tunnelled (unencrypted) migration through a user-defined port (port 2000). The port must be greater than 1024 and be open in the destination host's firewall:

```
# virsh migrate vm1 qemu+ssh://kvmhost2/system --live --undefinesource \
    --persistent --migrateuri tcp://kvmhost2:2000 --verbose
```

As before, once the migration is complete, the VM should be running on the destination host (and not on the source host). Also, verify that the XML definition file was created on the destination host and removed from the source host.

5.2.3 Offline, Non-Tunnelled

In our final example, we do an offline, non-tunnelled (unencrypted) migration without specifying a port. When not specifying a port, libvirt will use a random port from 49152 to 49215, so verify that these are open in the destination host's firewall:

```
# virsh migrate vm1 qemu+ssh://kvmhost2/system --undefinesource \
    --persistent --verbose
```

6. Conclusion

6.1 It's still easy!

As it turns out, managing a virtualized environment beyond creating and deleting VMs is still straight-forward and painless once you get the hang of it. I realize that some of the storage concepts (pools and volumes) might be difficult to conceptualize at first, but once you do, you will realize how it all makes sense.

6.2 What about oVirt or RHEV?

oVirt is an open source virtualization project based on KVM that provides a full management solution for hosts, shared storage and VMs. It greatly simplifies common administration activities, including those described here (i.e. snapshots, live migration). I haven't played with oVirt yet so I can't really provide any useful feedback.

Alternatively, Red Hat offers a product based on oVirt called Red Hat Enterprise Virtualization (RHEV) that offers the stability of Red Hat Enterprise Linux and support from Red Hat. RHEV features an intuitive, user-friendly administrative interface to quickly install and configure a complete virtualization environment. You can quickly configure shared storage, configure server clusters and add users with different access roles. In addition, it has a powerful reporting tool to help you audit and analyze your environment in order to optimize usage of your virtual resources. Depending on where you are deploying and on your budget, RHEV might be better choice for you.

For more information about oVirt, click here.
For more information about RHEV, click here.

6.3 Additional Documentation

- RHEL 6 Virtualization Getting Started Guide. Start here if you're a newbie.
- RHEL 6 Virtualization Host configuration and Guest Installation Guide. All about creating VMs.
- RHEL 6 Virtualization Administration Guide. Everything else you want to know.
- RHEL 5 Virtualization Guide. Though this whitepaper is focused on RHEL 6, all implementation steps also apply to RHEL 5 Update 4 and later (KVM is not included in earlier RHEL 5 releases).