Disk images for virtual machines

November 12, 2013

Virtual machines have disks

The host (on which the virtual machines are running) needs to provide some kind of resource to the virtual machine, that it can use as a disk.

There are multiple options:

- A file on disk (virtual disk image file)
- A block device (physical disk partition, or logical volume)

We typically use disk image files: they are easier to manipulate, copy, compress, etc.

What are we trying to do ?

Each virtual machine (VM) is going to need a disk image. If we are only going to create a single machine, it's pretty easy:

- create VM
- create disk image
- attach ISO image
- start VM
- install operating system

Done!

What if we want to install 2 machines ? We could probably install a second time. Or use an automated tool like `vmbuilder`. What about when we have to build 5 ? 40 ? And do this every week, to teach a class ?

About disk images. . .

There are in fact two different disk image types we can use:

- A “preallocated” disk image (1:1 blocks): a 10 GB disk image reserves 10 GB of disk space, regardless of whether the virtual machine guests uses 1 GB or 10 GB (allocated at creation time)
- An “extensible” disk image, useful for growing on demand

Disk images - pros / cons

There is a tradeoff in the above:

- A preallocated disk image means just that: the entire space allocated to the disk is zero'ed out (meaning that the file is created, then zeros are written to it until all the space allocated is filled up). This may be slower at first if the disk is big (writing zeros to, say, a 100GB file), but means the VM's disk IO will be faster later since it doesn't have to wait for the host OS to grow the file. But unused space on the VM disk is still allocated on the HOST filesystem...
- An extensible disk image uses a special disk format (in KVM: QCOW2) where the image file "grows" over time, as the guest VM starts using more and more disk space. From the VM point of view, it sees a full size disk, but the hypervisor is actually lying to the VM, and is allocating the disk blocks on the HOST side. Downside is slower performance and fragmentation, upside is less disk use.

Disk images - pros / cons

	Pros	Cons
Preallocated	- Faster IO for VM	- Longer init time (first time)
	- No fragmentation	- Uses all disk space right now
Growing	- Uses less disk (at first)	- Slower IO, fragmentation

So, what are we trying to achieve ?

Let's go back to our initial problem statement: we need to deploy multiple machines, all identical. Here are several methods we could use to achieve this:

- install the OS by hand N times... (No thanks!)
- run an automated image building system N times (Better, but...)
- copy the resulting disk image N times, create a new XML definition for each, with a slightly different MAC address, UUID, and pointing to the right disk file

Cloning images

The last solution described (copying the disk image N times) is already an improvement, but it will use a lot of disk space and time to copy.

For example: 40 machines x 10 GB = 400 GB of disk used.

Can we improve this even further ?

File formats

KVM typically supports two file format for disk images:

- So called ‘raw’ disks (byte-for-byte disk image, byte 0 = byte 0 of the disk)

If you were to do a byte-for-byte copy of a physical disk from an existing machine, you would have a “raw” image.

- QEMU-KVM’s “QCOW2” (Qemu Copy On Write, v.2) format

The QCOW2 format

QCOW2 files offer a number of features:

- grow-on-demand
- compression support
- encryption support

QCOW2 files can either be preallocated, as we discussed earlier, or set to grow on demand. Tools allow you to convert between one form or another (or even between images from other formats, like VMWare or VirtualBox)

The QCOW2 format

QCOW2 also has a very useful functionality, called “copy on write” (COW).

When programs inside the guest VM write to the virtual disk, the changes are written to the disk image on the host (for example, this could be `/var/lib/libvirt/images/master_image.qcow2`)

But maybe we don’t to write to `master_image.qcow2`, because it’s our “reference” installation ?

- We could copy the file as many times as we have machines.
- Or we can use the “copy on write” feature of QCOW2.

Using copy on write

We create a new disk image, but instead of copying the existing image, we tell the image creation tool that the new image should use the “master” image as a *backing file*. For example:

```
qemu-img create -o backing_file=master_image.qcow2
  -f guest1.qcow2 10G
```

This would create a 10GB QCOW2 image called `guest1.qcow2`, which uses `master_image.qcow2` as a backing file.

The `file` command tells us this:

```
guest1.qcow2: QEMU QCOW Image (v2), has backing file
  (path master_image.qcow2), 10737418240 bytes
```

Using copy on write (cont'd)

How does it work ?

- When the guest VM using `guest1.qcow2` reads from its disk, KVM will actually read the block from the `master_image.qcow2`.
- When the guest VM writes to its disk, KVM will write the changes to the guest1.qcow2)

The file `master_image.qcow2` is *never* written to.
You can use this method to have many guest disks using a single backing file. It's then very easy to "reset" the machines by deleting the `guest*.qcow2` files and create new ones using `master_image.qcow2` as the backing file.

Disk images - sparse files

Some filesystems support files with "holes" - also called sparse files.

Example...

```
truncate --size 20GB filename
```

What happens if we run `ls -l` and `du` on this file ?