Template implementation

7-9 minutes ⋮ 10/17/2024

Block devices of a VM

Every VM has 4 block devices connected:

- **xvda** – base root device (/) – details described below
- **xvdb** – private.img – place where VM always can write.
- **xvdc** – volatile.img, discarded at each VM restart – here is placed swap and temporal "/" modifications (see below)
- **xvdd** – modules.img – kernel modules and firmware

private.img (xvdb)

This is mounted as /rw and here is placed all VM private data. This includes:

- */home* – which is bind mounted to /rw/home
- */usr/local* – which is symlink to /rw/usrlocal
- some config files (/rw/config) called by qubes core scripts (ex /rw/config/rc.local)

Note: Whenever a TemplateBasedVM is created, the contents of the `/home` directory of its parent TemplateVM are *not* copied to the child TemplateBasedVM's `/home`. The child TemplateBasedVM's `/home` is independent from its parent TemplateVM's `/home`, which means that any changes to the parent TemplateVM's `/home` will not affect the child TemplateBasedVM's `/home`. Once a TemplateBasedVM has been created, any changes in its `/home`, `/usr/local`, or `/rw/config` directories will be persistent across reboots, which means that any files stored there will still be available after restarting the TemplateBasedVM. No changes in any other directories in TemplateBasedVMs persist in this manner. If you would like to make changes in other directories which *do* persist in this manner, you must make those changes in the parent TemplateVM.

modules.img (xvdd)

As the kernel is chosen in dom0, there must be some way to provide matching kernel modules to VM. Qubes kernel directory consists of 3 files:

- *vmlinuz* – actual kernel
- *initramfs* – initial ramdisk containing script to setup snapshot devices (see below) and mount /lib/modules
- *modules.img* – filesystem image of /lib/modules with matching kernel modules and firmware (/lib/firmware/updates is symlinked to /lib/modules/firmware)

Normally kernel "package" is common for many VMs (can be set using qvm-prefs). One of them can be set as default (qvm-set-default-kernel) to simplify kernel updates (by default all VMs use the default kernel). All installed kernels are placed in /var/lib/qubes/vm-kernels as separate subdirs. In this case,

modules.img is attached to the VM as R/O device.

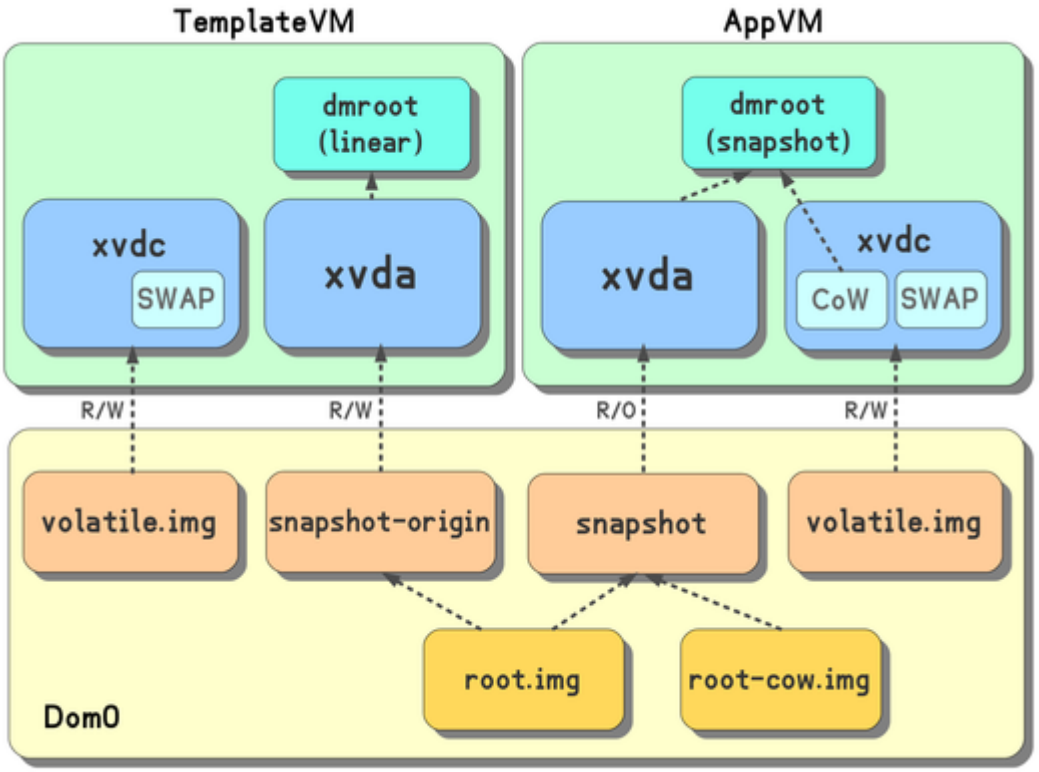
There is a special case when the VM can have a custom kernel – when it is updateable (StandaloneVM or TemplateVM) and the kernel is set to “none” (by qvm-prefs). In this case the VM uses the kernel from the “kernels” VM subdir and modules.img is attached as R/W device.

Qubes TemplateVM implementation

TemplateVM has a shared root.img across all AppVMs that are based on it. This mechanism has some advantages over a simple common device connected to multiple VMs:

- root.img can be modified while there are AppVMs running – without corrupting the filesystem
- multiple AppVMs that are using different versions of root.img (from various points in time) can be running concurrently

There are two layers of the device-mapper snapshot device; the first one enables modifying root.img without stopping the AppVMs and the second one, which is contained in the AppVM, enables temporal modifications to its filesystem. These modifications will be discarded after a restart of the AppVM.



Snapshot device in Dom0

This device consists of:

- root.img – real template filesystem
- root-cow.img – differences between the device as seen by AppVM and the current root.img

The above is achieved through creating device-mapper snapshots for each version of root.img. When an AppVM is started, a xen hotplug script (/etc/xen/scripts/block-snapshot) reads the inode numbers of

root.img and root-cow.img; these numbers are used as the snapshot device's name. When a device with the same name exists the new AppVM will use it – therefore, AppVMs based on the same version of root.img will use the same device. Of course, the device-mapper cannot use the files directly – it must be connected through /dev/loop*. The same mechanism detects if there is a loop device associated with a file determined by the device and inode numbers – or if creating a new loop device is necessary.

When an AppVM is stopped the xen hotplug script checks whether the device is still in use – if it is not, the script removes the snapshot and frees the loop device.

Changes to template filesystem

In order for the full potential of the snapshot device to be realized, every change in root.img must save the original version of the modified block in root-cow.img. This is achieved by a snapshot-origin device.

When TemplateVM is started, it receives the snapshot-origin device connected as a root device (in read-write mode). Therefore, every change to this device is immediately saved in root.img – but remains invisible to the AppVM, which uses the snapshot.

When TemplateVM is stopped, the xen script moves root-cow.img to root-cow.img.old and creates a new one (using the `qvm-template-commit` tool). The snapshot device will remain untouched due to the loop device, which uses an actual file on the disk (by inode, not by name). Linux kernel frees the old root-cow.img files as soon as they are unused by all snapshot devices (to be exact, loop devices). The new root-cow.img file will get a new inode number, and so new AppVMs will get new snapshot devices (with different names).

Rollback template changes

There is possibility to rollback last template changes. Saved root-cow.img.old contains all changes made during last TemplateVM run. Rolling back changes is done by reverting this "binary patch".

This is done using snapshot-merge device-mapper target (available from 2.6.34 kernel). It requires that no other snapshot device uses underlying block devices (root.img, root-cow.img via loop device). Because of this all AppVMs based on this template must be halted during this operation.

Steps performed by **qvm-revert-template-changes**:

1. Ensure that no other VMs uses this template.
2. Prepare snapshot device with ***root-cow.img.old*** instead of *root-cow.img* (*/etc/xen/scripts/block-snapshot prepare*).
3. Replace *snapshot* device-mapper target with *snapshot-merge*, other parameters (chunk size etc) remains untouched. Now kernel starts merging changes stored in *root-cow.img.old* into *root.img*. d-m device can be used normally (if needed).
4. Waits for merge completed: *dmsetup status* shows used snapshot blocks – it should be equal to metadata size when completed.
5. Replace *snapshot-merge* d-m target back to *snapshot*.
6. Cleanup snapshot device (if nobody uses it at the moment).
7. Move *root-cow.img.old* to *root-cow.img* (overriding existing file).

Snapshot device in AppVM

Root device is exposed to AppVM in read-only mode. AppVM can write only in:

- private.img – persistent storage (mounted in /rw) used for /home, /usr/local – in future versions, its use may be extended
- volatile.img – temporary storage, which is discarded after an AppVM restart

volatile.img is divided into two partitions:

1. changes to root device
2. swap partition

Inside of an AppVM, the root device is wrapped by the snapshot in the first partition of volatile.img. Therefore, the AppVM can write anything to its filesystem – however, such changes will be discarded after a restart.

StandaloneVM

Standalone VM enables user to modify root filesystem persistently. It can be created using *–standalone* switch to *qvm-create*.

It is implemented just like TemplateVM (has own root.img connected as R/W device), but no other VMs can be based on it.