A Practical Look at QEMU and libvirt Block Layer Primitives

Kashyap Chamarthy <kchamart@redhat.com>

NLUUG (Netherlands Local Unix / Linux User Group)
Spring 2017

Thanks and Acknowledgements

The QEMU and libvirt Block Layer contributors and maintainers:

Kevin Wolf, Markus Armbruster,
Stefan Hajnoczi, Eric Blake, John Snow,
Max Reitz, Alberto Garcia, Jeff Cody,
Daniel P. Berrangé, Paolo Bonzini,
Peter Krempa, Michal Privoznik,
John Ferlan and *mutiple others*...

In this presentation

* Background

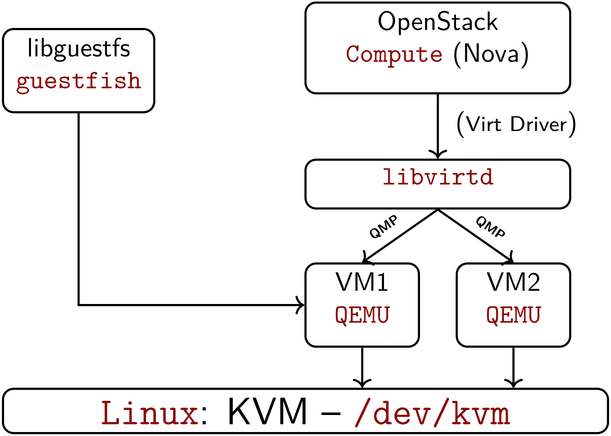
* Primer on operating a QEMU instance

* Configuring block devices

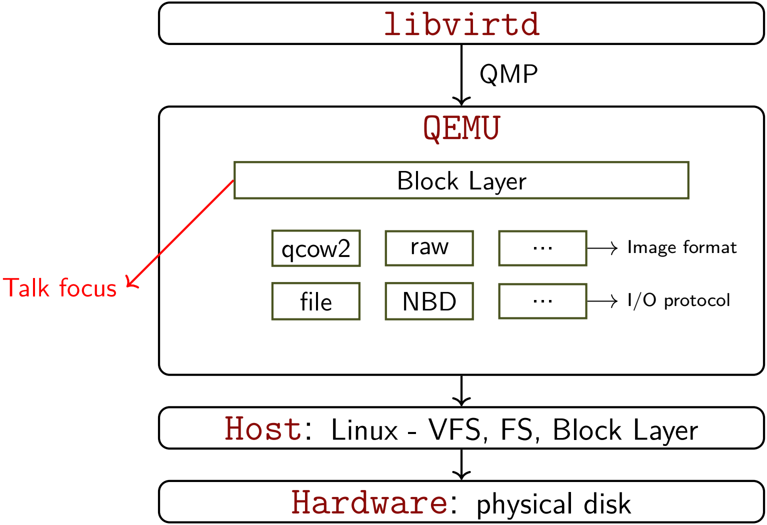
* Live block operations

Part I
Background

KVM / QEMU Virtualization components



Storage layers



QEMU's block subsystem

- Emulated storage devices: IDE, SCSI, virtio-blk, ...

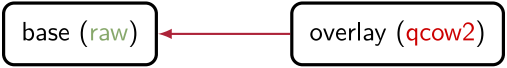
  Look for "Storage devices" in output of:

  ```
  $ qemu-system-x86_64 -device help
  ```

- Block driver types:
  - Format: qcow2, raw, vmdk
  - I/O Protocol: NBD, file, RBD/Ceph

- Block device operations:
  - `qemu-img`: For offline image manipulation
  - Live: snapshots, image streaming,
    storage migration, ...

QEMU Copy-On-Write overlays



('base' is the backing file of 'overlay')

– Read from overlay if allocated, otherwise from base
– Write to overlay only

Use cases: Thin provisioning, snapshots, backups, ...

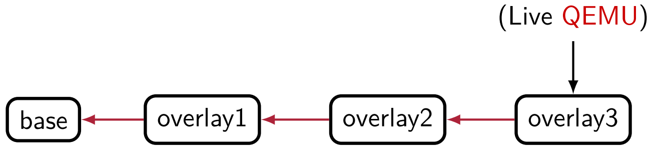
```
$ qemu-img create -f raw base.raw 2G
$ qemu-img create -f qcow2 overlay.qcow2 \
     2G -b base.raw -F raw
```

↑ (Backing file)

↑ (Backing file format)

Backing chain with multiple overlays

Disk image chain with a depth of 3:



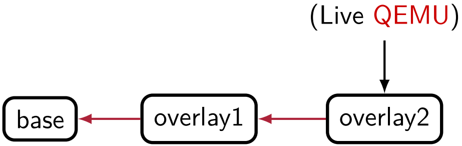
Multiple methods to configure & manipulate them:

Offline	:	`qemu-img`
Command-line	:	`qemu-system-x86 -drive [...]`
Run-time (QMP)	:	`blockdev-snapshot-sync`,
		`blockdev-add`, and more...

↖

(New in QEMU 2.9)

On accessing disk images opened by QEMU



Disk images that are opened by QEMU must not be accessed by external tools (qemu-img, qemu-nbd)

⇝ QEMU offers equivalent monitor commands

For secure, read-only access, use tools from the versatile libguestfs project:

```
$ guestfish -ro -i -a disk.img
```

Part II
Primer on operating QEMU

QEMU's QMP monitor

- Provides a JSON RPC interface
  - Send commands to query / modify VM state
  - QMP (asynchronous) events on certain state changes

If you zoom into libvirt-generated QEMU command-line:

```
$ qemu-system-x86 [...] \
   -chardev socket,id=charmonitor, \
    path=/var/lib/libvirt/qemu/vm1.monitor,server,nowait \
   -mon chardev=charmonitor,id=monitor,mode=control
```

For QMP commands

Shorthand notation for the above:

```
$ qemu-system-x86 [...] \
   -qmp unix:./qmp-sock,server,nowait
```

Interacting with QMP monitor

Connect to the QMP monitor via socat (SOcket CAT):

```
$ socat UNIX:./qmp-sock \
   READLINE,history=$HOME/.qmp_history \
{"QMP": {"version":
              {"qemu": {"micro": 50, "minor": 9, "major": 2},
               "package": " (v2.9.0-303-g81b2d5c-dirty)"},
         "capabilities": []}}

{"execute": "qmp_capabilities"}
{"return": {}}

{"execute": "query-status"}
{"return": {"status": "running", "singlestep": false,
            "running": true} }
```

Prerequisite

Send arbitrary commands: query-block, drive-backup, ...

Other ways to interact with QMP monitor

– qmp-shell: A low-level shell, located in QEMU source;
  takes key-value pairs (& JSON) dicts

```
$ qmp-shell -v -p ./qmp-sock
(QEMU) block-job-complete device=virtio1
```

– virsh: libvirt's shell interface

```
$ virsh qemu-monitor-command \
   vm1 --pretty '{"execute":"query-kvm"}'
```

(*NB: Modifying VM state behind libvirt's back voids support warranty!*)

⇝ Useful for test / development

Part III
Configuring block devices

Aspects of a QEMU block device

QEMU block devices have a notion of:

- Frontend: guest-visible devices (IDE, SCSI, virtio-blk, ...)

  ⇝ Configured via: (a) `-device` — command-line; or

  (b) `device_add` — run-time; like any other kind of guest device

- Backend: block devices / drivers (NBD, qcow2, raw, ...)

  ⇝ Configured via: (a) `-drive` — command-line;

  (b) `blockdev-add` — run-time

Configure block devices: command-line: `-drive`

Add a qcow2 disk & attach it to a 'virtio-blk' guest device:

```
$ qemu-system-x86 [...] \
   -drive file=overlay.qcow2,id=drv0,if=none \
   -device virtio-blk,drive=drv0
```

And, when relaunching QEMU, to explicitly specify (or override) the backing file:

```
[...] -drive file=overlay.qcow2, \
        backing.file.filename=newbase.qcow2, \
        if=none,id=drv0 \
      -device virtio-blk,drive=drv0
```

→ Why? Programs like libvirt need full control over backing file (for SELinux confinement)

Command-line: New interface: `-blockdev`

- Merged in the recently released QEMU 2.9.
- New command-line interface, '`-blockdev`', to configure block devices
  - Provides more fine-grained control
- Upstream intends (in the distant future) to deprecate the legacy '`-drive`' option
- Example invocation:

```
$ qemu-system-x86_64 [...] \
   -blockdev node-name=node1,driver=qcow2, \
     file.driver=file,file.filename=./base.qcow2 \
   -device virtio-blk,drive=node1
```

Configure at run-time: `blockdev-add`

QEMU aims to make this a unified interface to configure all aspects of block drivers.

`blockdev-add` lets you configure all aspects of the backend:

- Hot-plug block backends
- Specify options for backing files at run-time:
  - set cache mode;
  - change backing file, or its format, ...

⇝ In future: Avoid having two interfaces (command-line and QMP) to configure block devices

NB: `blockdev-add` interface is declared stable in QEMU 2.9

A quick example of `blockdev-add`

Goal: Add a qcow2 block device.

Raw QMP, run-time, JSON invocation:

```
{ "execute":"blockdev-add",
  "arguments":{
       "driver":"qcow2",
       "node-name":"node1",
       "file":{
         "driver":"file",
         "filename":"./disk1.qcow2"
} } }
```

Command-line is flattened mapping of JSON (from above):

```
 $ qemu-system-x86 [...]\
     -blockdev driver=qcow2,node-name=node1,\
       file.driver=file,file.filename=./disk1.qcow2
```

Part IV

Live block operations

`blockdev-snapshot-sync`: External snapshots

- While the guest is running, if a snapshot is initiated:
  1. the *existing* disk becomes the backing file; and
  2. a *new* overlay file is created to track writes

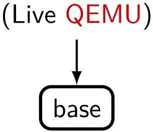
- Base image can be of any format; overlays are QCOW2

- No guest downtime; snapshot creation is instantaneous

- Allows atomic live snapshot of multiple disks

blockdev-snapshot-sync: A quick example

If you begin with:



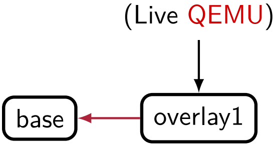
When operating via QMP:

```
blockdev-snapshot-sync device=virtio0 snapshot-file=overlay1.qcow2
```

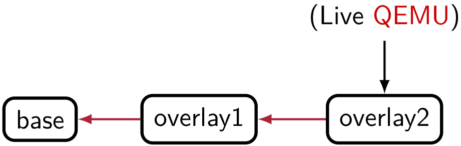
And, libvirt invocation (uses the above, under the hood):

```
$ virsh snapshot-create-as vm1 --disk-only --atomic
```

Result:



`blockdev-snapshot-sync`: Managing overlays

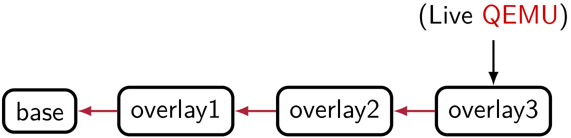


Problems:

- Revert to external snapshot is non-trivial
- Multiple files to track
- I/O penalty with a long disk image chain

There are some solutions...

`block-commit`: Live merge a disk image chain (1)


(Live QEMU)

base

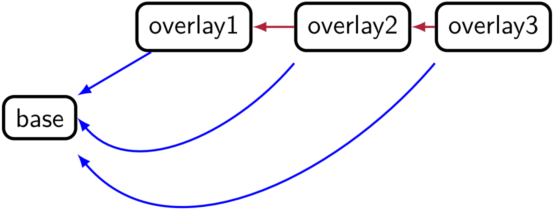
overlay1

overlay2

overlay3

Problem: Shorten the chain of overlays by merging some into a backing file, *live*

Simplest case: Merge all of them into 'base'

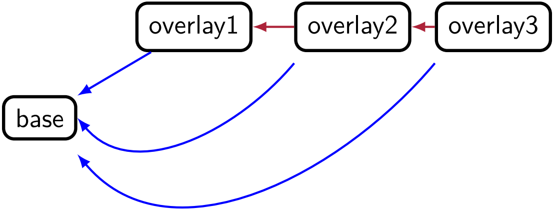

overlay1

overlay2

overlay3

base

block-commit: Live merge a disk image chain (2)



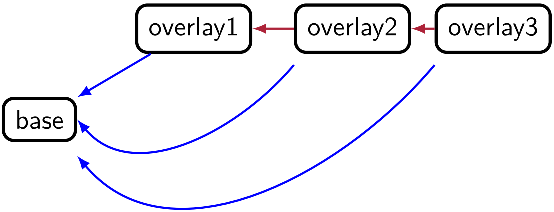
QEMU run-time invocation (simplified, using qmp-shell):

```
blockdev-snapshot-sync [...]
block-commit device=virtio-disk0
block-job-complete device=virtio-disk0
```

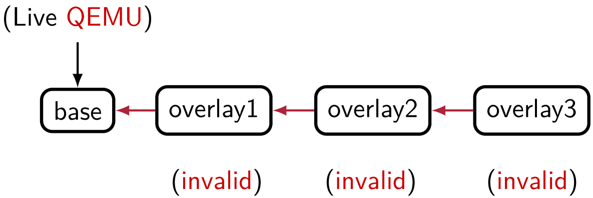
libvirt invocation (to merge overlays into base):

```
$ virsh blockcommit vm1 vda --verbose --pivot
```

block-commit: Live merge a disk image chain (3)



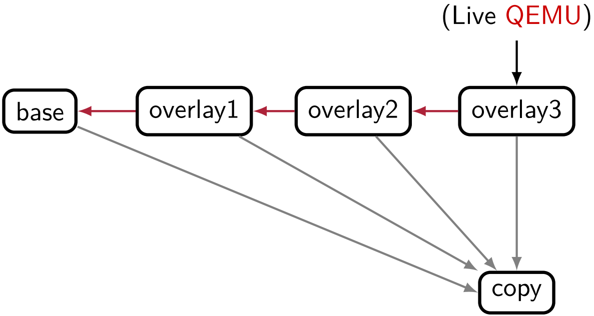
Two phase (sync & pivot) operation == a coalesced image



`block-stream`: The inverse of `block-commit`

- – Live copy data from backing files into overlays
- – The operation is safe – as data is being pulled forward
- – Intermediate overlays remain valid (*unlike* `block-commit`)
- – New in QEMU 2.8+ : Intermediate image streaming

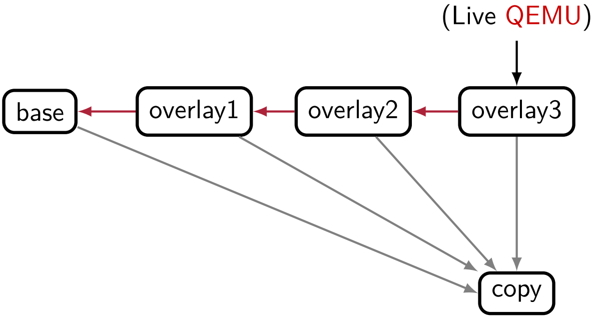
`drive-mirror`: Sync running disk to another image



Destination targets:

- an image file
- file served via NBD over UNIX socket
- file served via NBD over TCP socket
- more

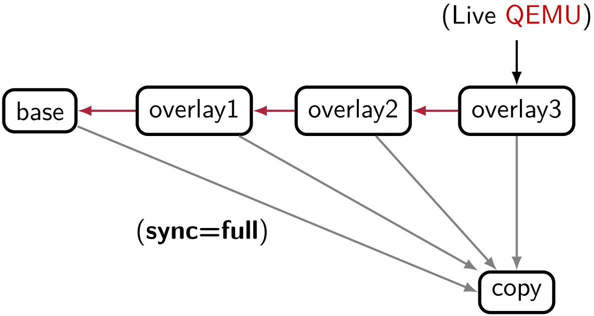
`drive-mirror`: Synchronization modes



Synchronization modes:

`'full'` – copy the entire chain
`'top'` – only from the topmost (active) image
`'none'` – copy only new writes from now on

`drive-mirror`: Operation



```
drive-mirror device=virtio0 target=copy1.qcow2 sync=full
```

```
query-block-jobs
```

```
block-job-complete device=virtio0
```

⇝ Issuing explicit `block-job-complete` will end sync and pivots the live QEMU to the mirror

QEMU NBD server

- **Network Block Device** server built into QEMU
  - Lets you export images *while in-use*

- Built-in QMP commands

```
nbd-server-start addr={"type":"unix",
                   "data":{"path":"./nbd-sock"}}
```

```
nbd-server-add device=virtio0
```

```
nbd-server-stop
```

- Also external program for offline use: `qemu-nbd`

Combining drive-mirror and NBD

Use case: Efficient live storage migration without shared storage (as done by libvirt)

- Destination QEMU starts the NBD server, & exports a pre-created empty disk
- Source QEMU issues drive-mirror to sync disk(s) via NBD over TCP

Raw QMP JSON invocation of drive-mirror:

```
{ "execute": "drive-mirror",
  "arguments": {
     "device": "disk0",
     "target": "nbd:desthost:49153:exportname=disk0",
     "sync": "top",
     "mode":"existing"
  }
}
```

Combining `drive-mirror` & NBD: libvirt automation

libvirt automates all the workflow for NBD-based live storage migration:

```
$ virsh migrate \
   --live \
   --verbose \
   --p2p \
   --copy-storage-all \
   vm1 \
   qemu+ssh://root@desthost/system
```

`drive-backup`: Point-in-time copy of a block device

- Point-in-time is when you *start* `drive-backup`
  - For `drive-mirror`, it is when you *end* the sync

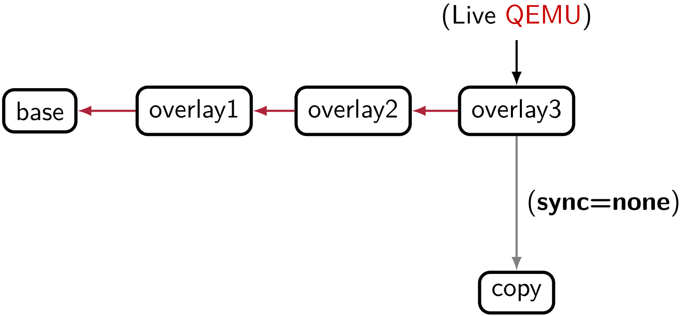
- Synchronization modes:
  - `'top'`
  - `'full'`
  - `'none'`
  - `'incremental'`

↖ (WIP as of 2.9; for incremental backups)

⇝ Not yet wired into libvirt; WIP

`drive-backup`: Point-in-time copy of a block device

Scenario: Copy only the new writes from now on to the target

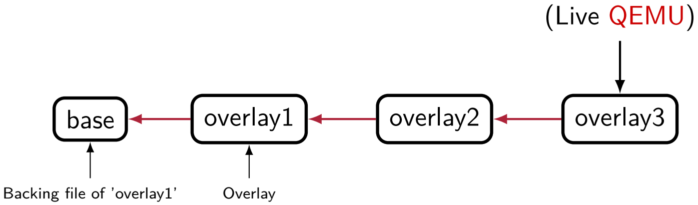


Run-time QEMU invocation (using qmp-shell):

```
drive-backup device=virtio0 sync=none target=copy.qcow2
```

Mapping of QEMU block primitives to libvirt APIs

QEMU block primitive	libvirt mapping	Purpose
`blockdev-snapshot-sync`	snapshotCreateXML()	Live disk snapshots
`block-commit`	blockCommit()	Move data from overlays into backing files
`block-stream`	blockRebase()	Move data from backing files into overlays
`drive-mirror`	blockCopy()	Live storage migration
`drive-backup`	(Not yet wired)	Point-in-time backup



References

- "Incremental Backups - Good things come in small packages!"
  https://fosdem.org/2017/schedule/event/backup_dr_incr_backups/
- "Backing Chain Management in libvirt and qemu" by Eric Blake
  http://events.linuxfoundation.org/sites/events/files/slides/2015-qcow2-expanded.pdf
- "More Block Device Configuration" by Kevin Wolf & Max Reitz
  https://archive.fosdem.org/2015/schedule/event/observability/
- "QEMU interface introspection: From hacks to solutions" by Markus Armburster
  https://events.linuxfoundation.org/sites/events/files/slides/armbru-qemu-introspection.pdf
- "qcow2 – why (not)?", by Max Reitz & Kevin Wolf
  http://www.linux-kvm.org/images/9/92/Qcow2-why-not.pdf
- Blog:
  http://kashyapc.wordpress.com

Thanks for listening.