A Practical Look at QEMU's Block Layer Primitives

Kashyap Chamarthy <kchamart@redhat.com>

LinuxCon 2016
Toronto

In this presentation

* Background

* Primer on operating QEMU

* Configuring block devices

* Live block operations

Part I
Background

KVM / QEMU Virtualization components

```
                        .------------------.
                        |    OpenStack     |
.----------.            |     Compute      |
|libguestfs|            '------------------'
'----------'                    |
     |                          | (Virt driver)
     |                     .----------.
     |                     | libvirtd |
     |                     '----------'
     |                          |
     |                          | (QMP)
     '----------------.--------.--.--------.
  [Device             |  VM1   |  |  VM2   |
       emulation] --> |  QEMU  |  |  QEMU  |
.---------------------'--------'--'--------'
|             Linux -- KVM (/dev/kvm)      |
'------------------------------------------'
```

QEMU's block subsystem

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

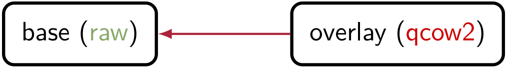
  Look for "Storage devices" in output of:

  ```
  $ 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 (raw)

overlay (qcow2)

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

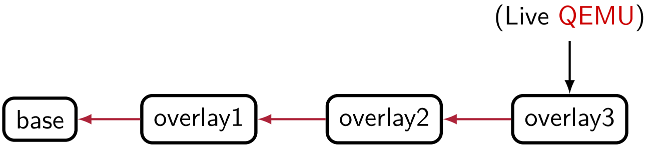
– Read from overlay if allocated, otherwise from base
– Write to overlay only

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:

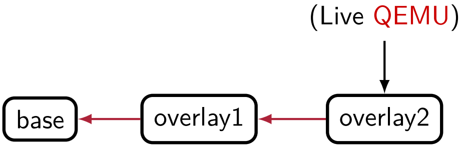


Multiple methods to configure & manipulate them:

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

↖ (Experimental as of QEMU 2.7)

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 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:

```
$ 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": 92, "minor": 6, "major": 2},
               "package": " (v2.7.0-rc2-65-g1182b8f-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=drive-virtio1
```

– virsh: libvirt's shell interface

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

Caveat: 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, USB, SCSI, ...)
  ⇝ Configured via: `-device` [command-line];
  `device_add` [run-time]; like any other kind of guest device

- Backend: block devices / drivers (NBD, qcow2, raw, ...)
  ⇝ Configured via: `-drive` [command-line];
  `blockdev-add` [run-time]

Configure block devices: command-line

Add a qcow2 disk & attach it to an IDE guest device:

```
$ qemu-system-x86 [...] \
    -drive file=overlay.qcow2,id=drive-ide0,if=none \
    -device ide-hd,drive=drive-ide0,id=ide0
```

To explicitly specify (or override) the backing file:

```
-drive file=overlay.qcow2,\
  backing.file.filename=base2.qcow2, \
  id=drive-ide0,if=none
```

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

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: cache mode, change backing file (or its format), ...

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

NB: `blockdev-add` is still being developed (as of QEMU 2.7)

blockdev-add: Add a simple block device

Raw QMP invocation:

```
{ "execute":"blockdev-add",
  "arguments":{
      "options":{
          "driver":"qcow2",
          "id":"virtio1",
          "file":{
              "driver":"file",
              "filename":"./disk1.qcow2"
} } } }
```

Command-line is a flattened mapping of JSON:

```
-drive driver=qcow2,id=virtio1,\
   file.driver=file,file.filename=./disk1.qcow2
```

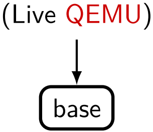
Part IV
Live block operations

`blockdev-snapshot-sync`: External snapshots

- While the guest is running, if a snapshot is initiated:
  - the existing disk becomes the backing file
  - a new overlay file is created to track new writes
- Base image can be of any format; overlays are qcow2
- No guest downtime; snapshot creation is instantaneous
- Atomic live snapshot of multiple disks

blockdev-snapshot-sync: A quick example

If you begin with:



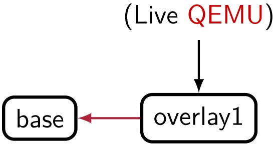
When operating via QMP:

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

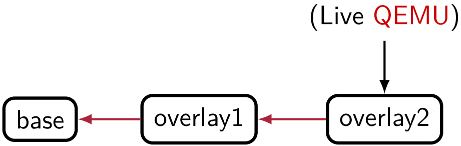
libvirt (invokes the above, under the hood):

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

Result:



blockdev-snapshot-sync: Managing overlays

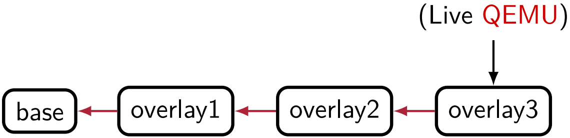


Problems:

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

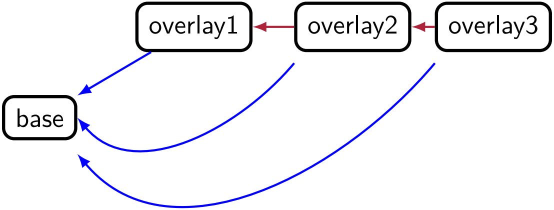
There are some solutions...

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

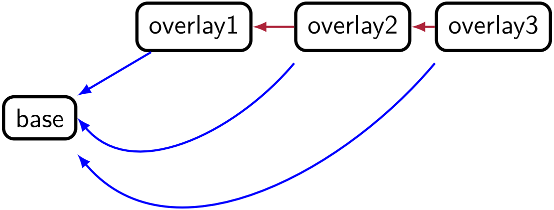


Problem: Shorten the chain of overlays by merging some into a backing file, live

Simplest case: Merge all of them into base



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



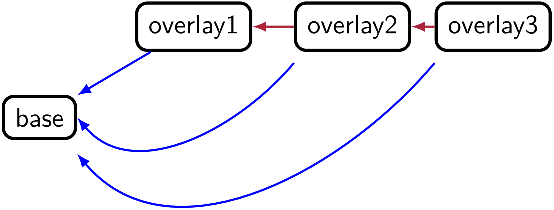
QEMU invocation (simplified, using qmp-shell):

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

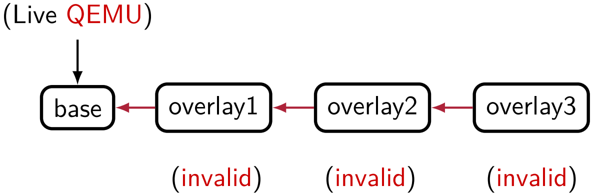
libvirt invocation:

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

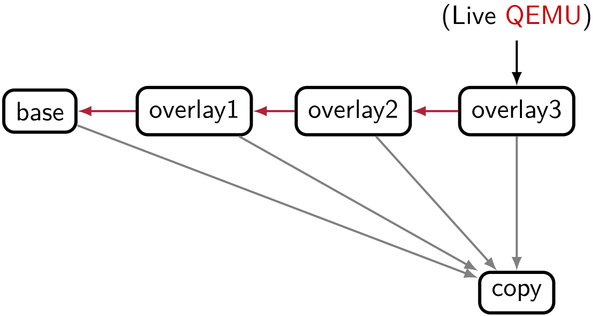
`block-commit`: Live merge a disk image chain (3)



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



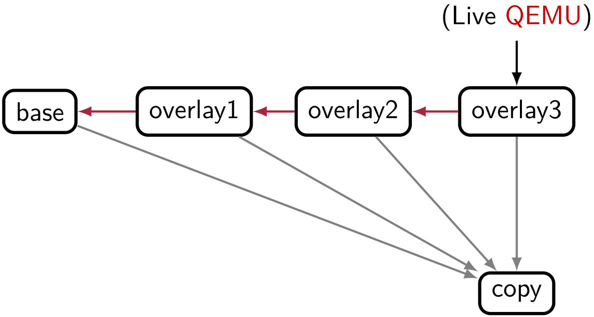
`drive-mirror`: Sync running disk to another image



Destination targets:

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

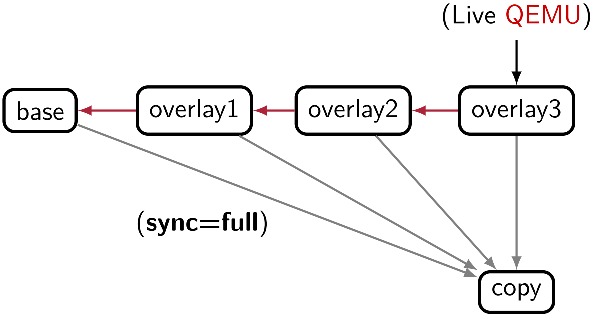
`drive-mirror`: Synchronization modes



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=mirror1.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

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

`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

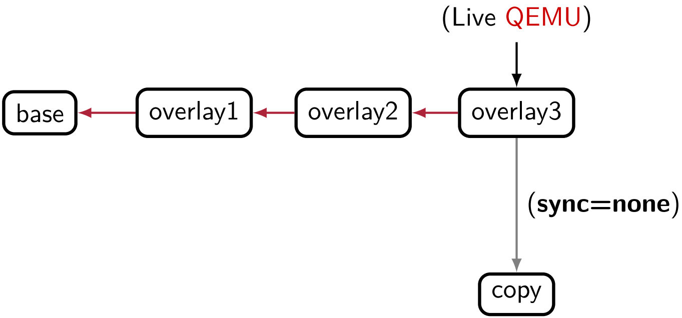
- Sync modes:
  - `'top'`
  - `'full'`
  - `'none'`
  - `'incremental'`

↖ (WIP, as of 2.7;
for incremental backups)

⇝ Not wired into libvirt yet

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

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

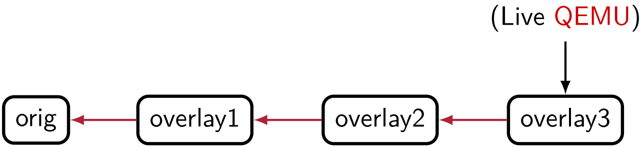


```
drive-backup device=virtio0 sync=none target=copy.qcow2
```

Don't miss: "Backups with QEMU" by Max Reitz at KVMForum;
Thu at 15:30

libvirt block APIs used by OpenStack Nova

QEMU block primitive	libvirt mapping	Purpose
blockdev-snapshot-sync	snapshot-create-as snapshotCreateXML()	Live disk snapshots
block-commit	blockcommit blockCommit()	Move data from overlays into backing files
block-stream	blockpull blockRebase()	Move data from backing files into overlays
drive-mirror	blockcopy blockCopy()	Live storage migration



References

- "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.com

redhat.

Thanks for listening.