Facilitating Incremental Backup

Eric Blake <eblake@redhat.com>
KVM Forum, 26 October 2018

In this presentation

- Existing use of libvirt to capture disk backups from live guest
- Basics of proposed new API for capturing backups
- Third-party access via NBD
- Power of new API for capturing incremental backups

Conventions in these slides

- Read-only image: blue shading
- Image being modified by guest or qemu: green shading
- Image being modified outside of libvirt: red shading
- Image1 ← Image2: dependency (Image2 is based on Image1)
-
```
$ command line
command output
```
- virAPI(argument1, argument2) [qmp]: magenta/orange shading for libvirt/QMP API calls

Baseline

- qemu 3.0 (14 Aug 2018)
- libvirt 4.8 (1 Oct 2018), plus patches for new API
  - https://repo.or.cz/libvirt/ericb.git/shortlog/refs/tags/backup-v3
- New APIs shown here have been proposed on libvir-devel list, but may undergo subtle changes before going upstream

Part I

Existing libvirt live guest backup

Setup

-
  ```
  $ virsh dumpxml $dom | sed -n '/<disk/,/<\/disk/p'
      <disk type='file' device='disk'>
        <driver name='qemu' type='qcow2'/>
        <source file='/home/eblake/Active1.qcow2'/>
        <target dev='vda' bus='virtio'/>
      </disk>
      <disk type='file' device='disk'>
        <driver name='qemu' type='qcow2'/>
        <source file='/home/eblake/Active2.qcow2'/>
        <target dev='vdb' bus='virtio'/>
      </disk>
  ```

Running guest

- Start with thin-provisioned guest with two disks
- `$ virsh start $dom`
- Base1.img ← Active1.qcow2
- Base2.img ← Active2.qcow2

Backup via blockcopy – wait for synchronization

-
  ```
  $ virsh blockcopy $dom Active1.qcow2 Backup1.1.qcow2 \
    --shallow --transient-job --wait &
  ```

-
  ```
  $ virsh blockcopy $dom Active2.qcow2 Backup2.full.raw \
    --format raw --transient-job --wait
  ```

-
  ```
  $ wait $!
  ```
- Base1.img ← Active1.qcow2

  \– Backup1.1.qcow2
- Base2.img ← Active2.qcow2

  Backup2.full.raw

End the blockcopy job

- `$ virsh suspend $dom`
- `$ virsh blockjob $dom Active1.qcow2 --abort`
- `$ virsh blockjob $dom Active2.qcow2 --abort`
- `$ virsh resume $dom`
- Base1.img ← Active1.qcow2
- Base1.img ← Backup1.1.qcow2
- Base2.img ← Active2.qcow2
- Backup2.full.raw

Under the hood

- virDomainBlockCopy(...) [drive-mirror] to start job
- virConnectDomainEventRegisterAny(…, VIR_DOMAIN_EVENT_ID_BLOCK_JOB, …) [JOB_READY] and/or virDomainGetBlockJobInfo(…) [query-block-jobs] to track job progress
- virDomainBlockJobAbort(…) [block-job-cancel] to end job
- virDomainSuspend(…) [stop] and virDomainResume(…) [cont] to provide multi-disk synchronicity

Limitations

- One disk at a time – an atomic capture of multiple disks requires pausing the guest
- All-or-nothing copy – no way to identify a subset of Active1.qcow2 that was modified since last backup
- Destination has to be in format that qemu can write
  - If guest is thin-provisioned, a shallow backup must be qcow2
  - Backup to alternative format, like raw, copies full chain
- Backup is captured at point in time where job is ended

Comparison table

	blockcopy	snap/commit	Backup push	Backup pull
Point in time	End			
<domain> XML	Unmodified			
Multiple disks at point-in-time	Manual sync, guest paused			
Shallow copy	Supported			
API calls used for 2 disks	8			
3rd-party use	No			
Incremental	No			

Backup via snapshot/commit – temporary snapshot

-
  ```
  $ virsh snapshot-create-as $dom tmp --no-metadata \
     --live --disk-only
  ```
- Base1.img ← Active1.qcow2 ← Active1.qcow2.tmp
- Base2.img ← Active2.qcow2 ← Active2.qcow2.tmp

Copy files

- `$ cp --reflink=always Active1.qcow2 Backup1.1.qcow2`
- `$ qemu-img convert -O raw Active2.qcow2 Backup2.full.raw`
- Base1.img ← Active1.qcow2 ← Active1.qcow2.tmp
- Base1.img ← Backup1.1.qcow2
- Base2.img ← Active2.qcow2 ← Active2.qcow2.tmp
- Backup2.full.raw

End job and cleanup

- `$ virsh blockcommit $dom vda --shallow --pivot`
- `$ virsh blockcommit $dom vdb --shallow --pivot`
- `$ rm Active[12].qcow2.tmp`
- Base1.img ← Active1.qcow2
- Base1.img ← Backup1.1.qcow2
- Base2.img ← Active2.qcow2
- Backup2.full.raw

Under the hood

- virDomainSnapshotCreateXML(...) [transaction:blockdev-snapshot-sync] to create external snapshot
- virDomainBlockCommit(…) [block-commit] to commit temporary snapshot
- virConnectDomainEventRegisterAny(…, VIR_DOMAIN_EVENT_ID_BLOCK_JOB, …) [JOB_READY] and/or virDomainGetBlockJobInfo(...) [query-block-jobs] to track commit job progress
- virDomainBlockJobAbort(…) [block-job-complete] to end job

Limitations

- <domain> is temporarily modified – recovery if libvirtd restarts can get hairy, particularly if transient domains are in use
- All-or-nothing copy – after live commit, no way to identify a subset of Active1.qcow2 that was modified since last backup
  - However, managing chains of snapshots, and committing the middle of the chain rather than the active layer, can be used to capture incremental backups in qcow2 format
- Backup to alternative format, like raw, requires being able to read files that qemu understands
- Block commit phase still requires per-disk commands

Comparison table

	blockcopy	snap/commit	Backup push	Backup pull
Point in time	End	Start		
<domain> XML	Unmodified	Temporarily changed		
Multiple disks at point-in-time	Manual sync, guest paused	Atomic group		
Shallow copy	Supported	Supported		
API calls used for 2 disks	8	7		
3rd-party use	No	Limited to snapshots		
Incremental	No	Limited to snapshots		

Part II

Full backups with new API

Creating a full backup via push

```
$ cat backup_push_full.xml
<domainbackup mode="push">
  <disks>
    <disk name="vda" type="file">
      <target file="/home/eblake/Backup1.1.qcow2"/>
      <driver type="qcow2" shallow="on"/>
    </disk>
    <disk name="vdb" type="file">
      <target file="/home/eblake/Backup2.full.raw"/>
      <driver type="raw"/>
    </disk>
  </disks>
</domainbackup>
```

Full Backup – start the job

-
  ```
  $ virsh backup-begin $dom backup_push_full.xml
  Backup id 1 created
  backup used description from 'backup_push_full.xml'
  ```
- Base1.img ← Active1.qcow2

  \– Backup1.1.qcow2
- Base2.img ← Active2.qcow2

  Backup2.full.raw

Full Backup – wait for completion

-
  ```
  $ while virsh backup-end $dom 1; do
      virsh domjobinfo $dom; sleep 1; done
  Backup id 1 still active, need --abort to end now
  Job type:      Backup
  … Backup id 1 completed
  ```
- Base1.img ← Active1.qcow2
- Base1.img ← Backup1.1.qcow2
- Base2.img ← Active2.qcow2
- Backup2.full.raw

Under the hood

- virDomainBackupBegin(dom, "<domainbackup…", NULL, 0) [blockdev-add, transaction:blockdev-backup] to start the job
- virDomainGetJobStats(...) [query-block-jobs] to track job progress
- virDomainBackupEnd(…) [no QMP] to end job
- Can also use virConnectDomainEventRegisterAny(…, VIR_DOMAIN_EVENT_ID_BLOCK_JOB, …) [JOB_COMPLETE] instead of polling

Improvements over existing approaches

- Backup is tied to the point that virDomainBackupBegin() is called (beginning rather than end of job)
- Multi-disk job is atomic – guest is never paused
- Fewer API calls, and <domain> is not modified
- API will support multiple backup jobs running in parallel (once libvirt improves job management APIs to take job id)
- However, push job still limited to formats known by qemu, and no way to back up just a portion of the disk

Comparison table

	blockcopy	snap/commit	Backup push	Backup pull
Point in time	End	Start	Start	
<domain> XML	Unmodified	Temporarily changed	Unmodified	
Multiple disks at point-in-time	Manual sync, guest paused	Atomic group	Atomic group	
Shallow copy	Supported	Supported	Supported	
API calls used for 2 disks	8	7	3	
3rd-party use	No	Limited to snapshots	No	
Incremental	No	Limited to snapshots	Yes	

Part III

Accessing backups through NBD

Creating a full backup via pull

-
  ```
  $ cat backup_pull_full.xml
  <domainbackup mode="pull">
    <server name="localhost" port="10809"/>
    <disks>
      <disk name="vda"/>
      <disk name="vdb" type="file">
        <scratch file="/home/eblake/scratch.qcow2"/>
      </disk>
    </disks>
  </domainbackup>
  ```

Why a scratch file?

- A backup job reflects the state of the disk at the start of the job, so qemu needs somewhere to store the original state as the guest continues to modify the live image
- In push mode, qemu just writes the original state to the backup image
- But in pull mode, qemu is not actively writing to a backup image, but instead exposing read-only data over NBD
- Thus, qemu needs a scratch file for the duration of the backup, even though the contents of that file are useless after the job ends
- Libvirt will attempt to create a local-storage scratch file, but if libvirt fails or makes an awkward choice, an explicit qcow2 file name can be provided in the XML

Full Backup – start the job

-
  ```
  $ virsh backup-begin $dom backup_pull_full.xml
  Backup id 1 created
  backup used description from 'backup_pull_full.xml'
  ```
- Base1.img ← Active1.qcow2 ← Active1.qcow2.1540065765
- Base2.img ← Active2.qcow2 ← scratch.qcow2
- NBD server on port 10809 (specified in the xml) is now serving two exports: “vda” and “vdb”

Full Backup – third-party access via qemu-img

-
  ```
  $ qemu-img convert -f raw nbd://localhost:10809/vda \
     -O raw Backup1.full.raw
  ```
- Base1.img ← Active1.qcow2 ← Active1.qcow2.1540065765
- Backup1.full.raw

Full Backup – third-party access via kernel nbd module

- `$ sudo modprobe nbd`
- `$ qemu-nbd -c /dev/nbd0 -f raw nbd://localhost:10809/vdb`

-
  ```
  $ dd if=/dev/nbd0 of=Subset.raw bs=64k skip=$((1024/64)) \
      count=$((1024/64)) conv=fdatasync
  ```
- Base2.img ← Active2.qcow2 ← scratch.qcow2
- Subset.raw is now a 1 megabyte file containing the raw contents at an offset of 1 megabyte into the guest’s view of storage

Full Backup – third-party access via qemu-io

-
  ```
  $ v='([0-9]*)' src=nbd://localhost:10809/vdb
  ```

-
  ```
  $ qemu-img create -f qcow2 -b $src -F raw Backup2.qcow2
  ```

-
  ```
  $ while read line; do
    [[ $line =~ .*start.:.$v.*length.:.$v.*data.:.true.* ]] || continue
    start=${BASH_REMATCH[1]} len=${BASH_REMATCH[2]}
    qemu-io -C -c "r $start $len" -f qcow2 Backup2.qcow2
  done < <($qemu-img map --output=json -f raw $src)
  ```

-
  ```
  $ qemu-img rebase -u -f qcow2 -b '' Backup2.qcow2
  ```
- Used qemu-io copy-on-read (-C) to read only data portions of the NBD export (parsed from qemu-img map output), copying those clusters into Backup2.qcow2, then a final rebase (-b '') for a standalone file

Full Backup – declare completion

-
  ```
  $ virsh backup-end $dom 1
  Backup id 1 completed
  ```

-
  ```
  $ rm scratch.qcow2
  ```
- Base1.img ← Active1.qcow2
- Backup1.full.raw
- Base2.img ← Active2.qcow2
- Backup2.qcow2
- Subset.raw

Under the hood

- virDomainBackupBegin(dom, "<domainbackup…", NULL, 0) [blockdev-add, transaction:blockdev-backup, nbd-server-start, nbd-server-add] to start the job
- virDomainBackupEnd(…) [nbd-server-stop, blockdev-del] to end job

Comparison to push backup

- qemu implementation uses a single NBD server for third-party integration, even when multiple disks are involved in the backup; but the XML is flexible enough that the same API supports other modes if other hypervisors have another third-party integration
- Once the third party has read data from NBD, it can store the backup in whatever format it desires; in fact, more than one client can connect as long as the NBD server is up
- Push model ends with an event, and can be aborted early; but pull model backup has no event (qemu does not know when the third party is done reading NBD, nor does qemu care whether it reads everything or just a subset) and thus abort has no meaning

Comparison table

	blockcopy	snap/commit	Backup push	Backup pull
Point in time	End	Start	Start	Start
<domain> XML	Unmodified	Temporarily changed	Unmodified	Unmodified
Multiple disks at point-in-time	Manual sync, guest paused	Atomic group	Atomic group	Atomic group
Shallow copy	Supported	Supported	Supported	Future extension
API calls used for 2 disks	8	7	3	2
3rd-party use	No	Limited to snapshots	No	Yes
Incremental	No	Limited to snapshots	Yes	Yes

Obvious future enhancements

- The previous demo showed the client specifying the NBD port in XML; but libvirt should be able to auto-assign an available port, which the client then queries with virBackupGetXMLDesc(...)
- Libvirt should allow the client to request that qemu’s NBD server use TLS encryption and/or user authentication to ensure qemu only exposes data to the correct third-party clients
- Libvirt should allow a Unix socket NBD server, not just TCP
- Qemu should allow more than one NBD server in parallel, in order to permit parallel backup jobs
- Qemu extension to NBD to let client learn which clusters come from active overlay vs. backing file, for shallow backups

Part IV

Incremental/Differential backups

Definitions

- Incremental backup – only the portions of the disk changed since the previous backup to the current moment
- Differential backup – all changes to the disk from a point in time to the present, even if other backups occurred in between
- Persistent dirty bitmap – a means of tracking which portions of a disk have changed (become dirty) since the bitmap was created
- Checkpoint – a point in time that can be used for incremental or differential backups

How much disk is the guest dirtying?

- The simplest use of a checkpoint is to determine how much data the guest is actively writing over a period of time

```
$ virsh checkpoint-create-as $dom c1
```

```
$ virsh checkpoint-dumpxml $dom c1 --size --no-domain
<domaincheckpoint>
  <name>c1</name>
  <creationTime>1540073217</creationTime>
  <disks>
    <disk name='vda' checkpoint='bitmap' bitmap='c1' size='131072'/>
    <disk name='vdb' checkpoint='bitmap' bitmap='c1' size='0'/>
  </disks>
</domaincheckpoint>
```

Interesting, but let’s make this useful

- The size reported gives an estimate of the size of a differential or incremental snapshot taken from that checkpoint
- Remember to add in padding for metadata, and to account for sectors changed between the size query and the actual backup
-
```
$ qemu-img measure --size 131072 -O qcow2
required size: 327680
fully allocated size: 458752
```
- But most checkpoints are NOT created in isolation, so time to clean up this one
-
```
$ virsh checkpoint-delete $dom c1
```

Under the hood

- virDomainCheckpointCreateXML(dom, "<domaincheckpoint...", 0) [transaction:block-dirty-bitmap-add, x-block-dirty-bitmap-disable] to create checkpoint (`virsh checkpoint-create-as --print-xml` can be used to generate the right XML), using persistent bitmap in the qcow2 file (survives both guest and libvirtd restarts)
- virDomainCheckpointGetXMLDesc(...) [query-block] with flags to query live size estimates of qemu bitmaps
- virDomainCheckpointDelete(…) [x-block-dirty-bitmap-enable, x-block-dirty-bitmap-merge, block-dirty-bitmap-remove] to remove
- Several other API for checking relations between checkpoints

Modifying the initial full backup

-
  ```
  $ cat backup_pull_1.xml
  <domainbackup mode="pull">
    <server name="localhost" port="10809"/>
  </domainbackup>
  ```

-
  ```
  $ cat check1.xml
  <domaincheckpoint>
    <name>check1</name>
  </domaincheckpoint>
  ```

-
  ```
  $ virsh backup-begin $dom backup_pull_1.xml check1.xml
  Backup id 1 created
  backup used description from 'backup_pull_1.xml'
  checkpoint created from 'check1.xml'
  ```

Save off full backups using NBD as before

-
  ```
  $ qemu-img convert -f raw nbd://localhost:10809/vda \
     -O qcow2 Backup1.1.qcow2
  ```

-
  ```
  $ qemu-img convert -f raw nbd://localhost:10809/vdb \
     -O qcow2 Backup2.1.qcow2
  ```
- Base1.img ← Active1.qcow2 ← Active1.qcow2.check1
- Backup1.1.qcow2
- Base2.img ← Active2.qcow2 ← Active2.qcow2.check1
- Backup2.1.qcow2

-
  ```
  $ qemu-img backup-end $dom 1
  Backup id 1 completed
  ```

Now for an incremental backup

-
  ```
  $ cat backup_pull_2.xml
  <domainbackup mode="pull">
    <incremental>check1</incremental>
    <server name="localhost" port="10809"/>
  </domainbackup>
  ```

-
  ```
  $ cat check2.xml
  <domaincheckpoint>
    <name>check2</name>
  </domaincheckpoint>
  ```

-
  ```
  $ virsh backup-begin $dom backup_pull_2.xml check2.xml
  Backup id 1 created
  backup used description from 'backup_pull_2.xml'
  checkpoint created from 'check2.xml'
  ```

NBD tells us dirty clusters

- Use your NBD client’s NBD_CMD_BLOCK_STATUS support on the “qemu:dirty-bitmap:backup-vda” namespace to learn which portions of the image are dirtied
  - Haven’t written your own NBD client yet? qemu-img can do it:
- `$ v='([0-9]*)' src=nbd://localhost:10809/vda`
- `$ qemu-img create -f qcow2 -b $src -F raw Backup1.2.qcow2`
- `$ img=driver=nbd,export=vda,server.type=inet,`
- `$ img+=server.host=localhost,server.port=10809,`
- `$ img+=x-dirty-bitmap=qemu:dirty-bitmap:backup-vda`
- …

NBD tells us dirty clusters

- Use your NBD client’s NBD_CMD_BLOCK_STATUS support on the “qemu:dirty-bitmap:backup-vda” namespace to learn which portions of the image are dirtied
  - Haven’t written your own NBD client yet? qemu-img can do it:
- …

-
  ```
  $ while read line; do
    [[ $line =~ .*start.:.$v.*length.:.$v.*data.:.false.* ]] || continue
    start=${BASH_REMATCH[1]} len=${BASH_REMATCH[2]}
    qemu-io -C -c "r $start $len" -f qcow2 Backup1.2.qcow2
  done < <($qemu-img map --output=json --image-opts $img)
  ```

-
  ```
  $ qemu-img rebase -u -f qcow2 -b Backup1.1.qcow2 -F qcow2 \
    Backup1.2.qcow2
  ```

Incremental chain is ready

- Use of `--image-opts` and `x-dirty-bitmap=qemu:dirty-bitmap:backup-vda` exposes dirty region boundaries in NBD_CMD_BLOCK_STATUS
- Base1.img ← Active1.qcow2
- Backup1.1.qcow2 ← Backup1.2.qcow2
- Base2.img ← Active2.qcow2
- Backup2.1.qcow2 ← Backup2.2.qcow2
-
```
$ qemu-img backup-end $dom 1
Backup id 1 completed
```

Rinse and repeat

-
  ```
  $ cat backup_pull_3.xml
  <domainbackup mode="pull">
    <incremental>check2</incremental>
    <server name="localhost" port="10809"/>
  </domainbackup>
  ```

-
  ```
  $ cat check3.xml
  <domaincheckpoint>
    <name>check3</name>
  </domaincheckpoint>
  ```

-
  ```
  $ virsh backup-begin $dom backup_pull_3.xml check3.xml
  Backup id 1 created
  backup used description from 'backup_pull_3.xml'
  checkpoint created from 'check3.xml'
  ```

Longer incremental chain is ready

- Another round of 3rd-party NBD access...
- Base1.img ← Active1.qcow2
- Backup1.1.qcow2 ← Backup1.2.qcow2 ← Backup1.3.qcow2
- Base2.img ← Active2.qcow2
- Backup2.1.qcow2 ← Backup2.2.qcow2 ← Backup2.3.qcow2
-
```
$ qemu-img backup-end $dom 1
Backup id 1 completed
```

Differential, snooping

-
  ```
  $ cat backup_pull_4.xml
  <domainbackup mode="pull">
    <incremental>check1</incremental>
    <server name="localhost" port="10809"/>
  </domainbackup>
  ```

-
  ```
  $ virsh backup-begin $dom backup_pull_4.xml
  Backup id 1 created
  backup used description from 'backup_pull_4.xml'
  ```
- Note that with no checkpoint created, this backup cannot be used as the base for a future backup, but rather just snoops the disk

Multiple checkpoints allow differential backups

- Another round of 3rd-party NBD access...
- Base1.img ← Active1.qcow2
- Backup1.1.qcow2 ← Backup1.2.qcow2 ← Backup1.3.qcow2
- Backup1.1.qcow2 ← Backup1.4.qcow2
- Base2.img ← Active2.qcow2
- Backup2.1.qcow2 ← Backup2.2.qcow2 ← Backup2.3.qcow2
- Backup2.1.qcow2 ← Backup2.4.qcow2

-
  ```
  $ qemu-img backup-end $dom 1
  Backup id 1 completed
  ```

Under the hood

- virDomainBackupBegin(dom, "<domainbackup…", "<domaincheckpoint...", 0) [blockdev-add, x-block-dirty-bitmap-merge, transaction:blockdev-backup+block-dirty-bitmap-add, nbd-server-start, nbd-server-add, x-nbd-server-add-bitmap] to create a checkpoint at the same time as starting a backup job
- Add "<incremental>" element to backup definition to choose all changes since that point in time – libvirt creates temporary bitmap as merge of all persistent bitmaps since checkpoint
  - In push mode, qemu pushes just those changes
  - In pull mode, NBD exposes that bitmap as block status context

Beyond this talk...

- For more details on qemu’s bitmaps and NBD:
  - Stick around: 16:15 Vladimir Sementsov-Ogievsky: “Qemu Backup Status”
  - Or refer to the past: 2015 John Snow: "Incremental Backups"
- qemu interfaces will be finalized (remove “x-” prefix, perhaps add some convenience commands so libvirt has fewer QMP calls…)
- libvirt API needs upstream acceptance (may change slightly from what was presented here)
- Design for supporting checkpoints with external snapshots still a work in progress (issues: hotplug, disk resize, ...)

Questions?