LVM Snapshots

Martin Kennedy

May 24, 2021

Background

DeviceMapper

- LVM2 relies on the DeviceMapper driver
  - At its core: just a bunch of kernel modules

```
lsmod | awk '{print $1}' | grep ^dm_ | xargs -I{} modinfo '{}'
```

`dm` mapping targets

- DeviceMapper gives you flexibility when working with block devices through its 'mapping targets'
  - `dm-linear` for e.g. treating sections of one-or-more Physical Volumes (PVs) in a Volume Group (VG) as one Logical Volume (LV)
  - `dm-crypt` for On-the-fly-encryption (OTFE)
  - `dm-snapshot` for Copy-on-Write (CoW) snapshots
    - allows a writable copy of a block device to be created atomically
    - coexists with original
    - works by tracking changes (metadata and data) in 'snapshot space'

PCC already uses LVM heavily

- Mostly for 'protecting' filesystems from one another, but ...
  - *most* of the data on *most* "PCC Server" environments we administer is in LVs ...
    - ... *except* for the boot partition, and ...
    - ... *except* for Proxmox lxc containers ("vz-*")
  - We already use `dm-crypt` for LUKS
  - I use LVM snapshots for data conversions when rolling back unwanted data changes

Snapshots

- Turn one LVM logical volume into two without unmounting
  - A snapshot needs space to record changes at block level
  - A snapshot shows up as a normal block device
  - Upon creation, the snapshot blockdev is immediately writable
  - Writing to the snapshot or the original LV consumes snapshot space

Quick demonstration

Plan

- Snapshot `/pccehr` and `/dat`
- Do some harmful things
- Merge snapshots into their parents

Snapshot creation

```
# As root
lvcreate --size 10G --snapshot --name pccehr_snap vg00/pccehr
lvcreate -L     30G -s         -n     dat_snap    vg00/dat
```

Destroy data

```
touch /cvt/data/DO_RESTORE.trigger;
make /cvt/data/FULL_RESTORE.sentinel;
```

Merge the snapshots into their parents

```
date; psql -d dat -c "select count(*) from patient";

RE systemctl stop partner pccehr mcservice postgresql-9.5;
RE lvconvert --merge vg00/pccehr_snap
RE lvconvert --merge vg00/dat_snap
RE umount /pccehr /dat;
RE lvchange -an vg00/{dat,pccehr};
RE lvchange -ay vg00/{dat,pccehr};
RE mount /dat; RE mount /pccehr
RE systemctl start partner pccehr mcservice postgresql-9.5;

date; psql -d dat -c "select count(*) from patient";
```

Further details

Under the hood

- LVM snapshots rely on blocks from the original disk (Copy on Write)

  - writes to the snapshot: move those blocks to the snapshot area
  - writes to the original LV: copy those blocks to the snapshot area
- "True" CoW, as opposed to ZFS / btrfs snapshots, which are RoW
  - Reduced performance on snapshot creation (particularly with HDDs and `fsync`)
  - ... **but**: no fragmentation, and dropping snapshot is instantaneous

COW area

```
lvcreate -L 10G -s -n ${lv}_snap ${vg}/${lv}
  Reducing COW size 10.00 GiB down to maximum usable size 4.03 GiB.
  Logical volume "pcc_snap" created.
```

- If you go over the amount of snapshot space allocated, you just make the snapshot un-usable.

Mounting

```
mkdir -p /pccehr_snap /dat_snap

# nouuid: XFS FSs have UUIDs that cannot collide
# Mounting with nouuid ignores those
mount -o nouuid /dev/vg00/pccehr_snap /pccehr_snap
mount -o nouuid /dev/vg00/dat_snap /dat_snap
# ... now check `lvs` ...

# As you might expect: The filesystem is *really* mounted and has all
# of the usual properties of a mounted filesystem
find /dat_snap
# ... now check `lvs` again ...

# Mounting with noatime prevents unneeded writes back to snapshot space
mount -o remount,noatime,nouuid /dev/vg00/pccehr_snap /pccehr_snap
mount -o remount,noatime,nouuid /dev/vg00/dat_snap /dat_snap
```

Discarding or merging

- As mentioned, we can drop snapshots instantaneously
- We can also merge them into the parent LV to *roll it back*
  - This requires momentarily de-activating the volume ...
  - ... but we can re-activate immediately after and begin using it again even if it isn't done merging:

    > While the merge is in progress, reads or writes to the origin appear as they were directed to the snapshot being merged. When the merge finishes, the merged snapshot is removed.

A full demonstration

Plan

- Snapshot `/pccehr` and `/dat`
- Mount, backup, unmount and `lvremove` both snapshots
- Do some harmful things
- Restore the backups to new snapshots, then merge those snapshots into their parents

Snapshot

```
RE lvcreate -L 10G -s -n pccehr_snap vg00/pccehr;
RE lvcreate -L 10G -s -n dat_snap vg00/dat;
```

Backup

```
RE mount -o noatime,nouuid /dev/vg00/dat_snap /dat_snap
RE mount -o noatime,nouuid /dev/vg00/pccehr_snap /pccehr_snap
. /cvt/tk/bak/bak_vars.sh
RE restic backup /dat_snap /pccehr_snap
RE umount /{pccehr,dat}_snap
RE lvremove vg00/{pccehr,dat}_snap
```

Destroy

```
touch /cvt/data/DO_RESTORE.trigger;
make /cvt/data/FULL_RESTORE.sentinel;
```

The restore process

Create snapshots

```
RE lvcreate -L 10G -s -n pccehr_snap vg00/pccehr;
RE lvcreate -L 10G -s -n dat_snap vg00/dat;
RE mount -o noatime,nouuid /dev/vg00/dat_snap /dat_snap
RE mount -o noatime,nouuid /dev/vg00/pccehr_snap /pccehr_snap
```

Restore each snapshot from backup

```
# As root
. /cvt/mk/mkennedy.profile
. /cvt/tk/bak/bak_vars.sh
mkdir -p /restic; restic mount /restic;

# In a second terminal
rsync -avP --delete /restic/snapshots/latest/pccehr_snap/ /pccehr_snap/
rsync -avP --delete /restic/snapshots/latest/dat_snap/ /dat_snap/
umount /{pccehr,dat}_snap;
```

Final step: merge the snapshots into their parents

```
date; psql -d dat -c "select count(*) from patient";

RE systemctl stop partner pccehr mcservice postgresql-9.5;
RE lvconvert --merge vg00/pccehr_snap
RE lvconvert --merge vg00/dat_snap
RE umount /pccehr /dat;
RE lvchange -an vg00/{dat,pccehr};
RE lvchange -ay vg00/{dat,pccehr};
RE mount /dat; RE mount /pccehr
RE systemctl start partner pccehr mcservice postgresql-9.5;

date; psql -d dat -c "select count(*) from patient";
```

Integrating into PCC's backup solution

`PCCbackup`

- `rear` runs as per usual before any backups
- We can do away with locking `/dat` during the backup ...
  - ... what we are doing is essentially the same as an `xfs_freeze`
- More difficult: dealing with Postgres.
  - Logical backups are **possible** but not **smart** here.

Dealing with Postgres

In summmary:

- As-is: use $\text{pg}_{\text{dumpall}}$, but we really want to speed it up.
- Idea: Start `pg_dumpall` at the same time a snapshot of `/dat` is made
  - Caveat: Only solves `/dat` <-> `/pccehr` consistency issue
- Idea: Start `pg_dump -j` at the same time a snapshot of `/dat` made
  - Caveat: Must begin one on **each database** to maintain consistency
- Idea: Start a second server on the snapshot, and do logical backup
  - Horrible idea: poor performance (especially so on spinning media)
- Idea: Physical (base) backup
  - Much faster backup process

As-is: `PCCbackup` for Postgres

```
Log "pg_dumpall: Started"
set -o pipefail
if ! su postgres -c "pg_dumpall --clean" 2>>$logtmp |
    gzip --rsyncable >/backup/pgbackup.tmp.sql.gz 2>>$logtmp ; then
SoftFail 2 "pg_dumpall"
```

Physical backups

- For our database, use block-based backups instead
  - pgBackRest delta backup / restore
  - restic
  - rsync