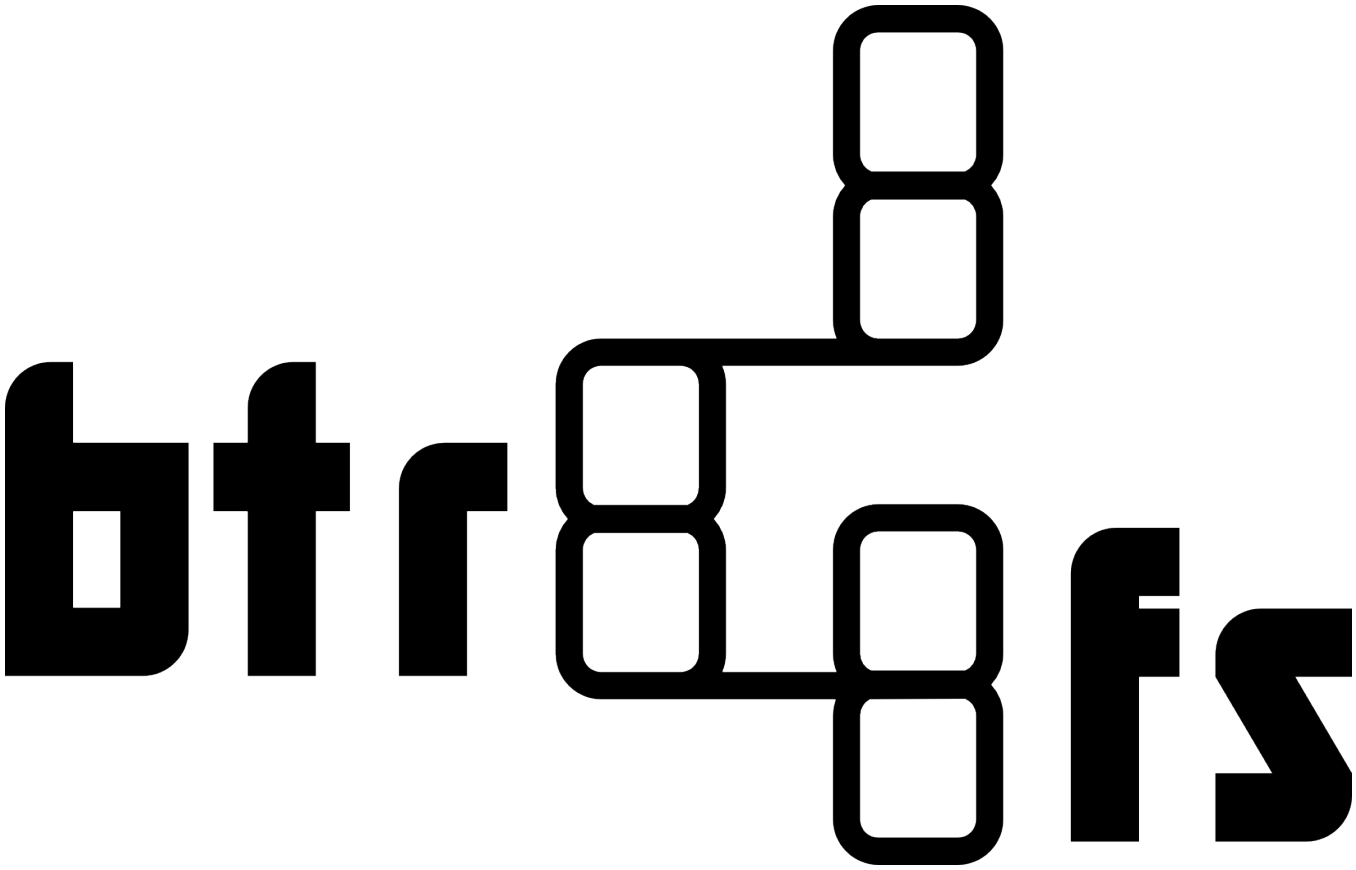
Next-Gen Backup with BTRFS Snapshots for Root-Fs and Databases

Bernd Helm | Linux | No comment



This article is about using BTRFS snapshots as backup solution, usable for databases and full root partition backups. This is not a detailed Step-By-Step guide and requires some linux skills. This Post aims to share my experiences and save you some hassle. We use it in production for years now.

Btrfs snapshots are storing a „freezed“ Version of the whole filesystem. This is done by btrfs's copy-on-write technique. This means that modifications on snapshotted files are not written directly into the file like it is done on classic filesystems like ext4. instead, all modifications are written in some other place, so both, the current and the snapshotted version of the file is present. This Copy-On-Write (COW) technique brings

a lot nice features to BTRFS, **but has also some drawbacks you should know about before using BTRFS in production**. **See „Always disable COW on Database directories“ below.**

BTRFS has a powerful snapshot capability that allows to incrementally transfer snapshots to remote systems. Its just so simple: create a snapshot, transfer it to a remote system, create a 2nd snapshot and only transfer the changed blocks from the previous one. Solaris/FreeBSD users are doing this for almost a decade with ZFS. A ZFS port is also available for linux, so you can also use it instead of BTRFS if you prefer it for some reason.

Use-Cases of BTRFS-Snapshots

- Backup Root Partitions of one or more Linux-Servers to a central backup Server
    - Its possible to directly boot the backup server into one of the snapshots, allowing a manual fail-over
    - The snapshots can also be transferred into virtual machines. This becomes very handy if you want your “dev vm” to be up to date with the live system and update it with incremental snapshots regularly.
- Backup MySQL, Postgres or whatever data partitions separately at a cycle different from other mount points
- You can use BTRFS Snapshots on the Root partitions to return to a functional rootfs after a failed system upgrade
- If you are a developer, you can use BTRFS Snapshots locally with your Database to test your data-modifying application again and again against the same starting dataset.
    - you can also share your dev database snapshots with your coworkers and give incremental updates to them.

Backing up a Database, pro and cons of different methods

When thinking about MySQL/MariaDB (and some other databases)

backups, you have some options to consider:

1. **use mysqldump to create a logical backup (also on per-table basis)**
    1. It is slow and creates noticeable load on the server
    2. Hard to create consistent state (requires locks or a desync replication slave)
    3. Not incremental, even when backing up only modified tables; if those tables are large, this is a problem.
    4. Restore: import the dump somewhere, can take ages with large datasets.
2. **use mysql binary logs for incremental backup**
    1. allows point-in-time recovery (recover to any moment in the past)
    2. replaying binary logs can take a long time on write-intensive setups
    3. you have to do full backups regulary because of this
3. **use innobackupex incremental method**
    1. while its more advanced, it seems to have the same characteristics as with mysql binary logs.
    2. Its also resource intensive and requires a full database lock at the end of the data transfer.
4. **use filesystem snapshots**
    1. no point-in-time recovery
    2. endless incremental, do one full backup and then only incremental ones.
    3. fast and resource-saving backup process
    4. restore: you can launch a instance of your database from every snapshot directly on the backup system or move the full mysql directory back to the source server if needet.
    5. only downside is that you have to relie on the databases own power-outage recovery mechanism.

While doing mysqldump backups in the past, we now use the filesystem snapshot way. It has proven to be a very reliable, fast and resource-saving way of doing backups and also provides easy restore of the backed up data. It is very common that our Devs ask for a specific data version to be started on the backup server. this is done within 10 minutes and saves them a lot of time.

A filesystem snapshot is a consistent state of a filesystem that is not modified anymore. From data-integrity point of view, a snapshot features a unclean state like you have after an power failure, a kill -9 or an “Out of Memory” error. Because MySQL/MariaDB and other database engines are designed to survive power failures, you are always able to recover from a snapshot too.

Stability of BTRFS and production-readyness

The stability and reliability of BTRFS has been discussed a lot in the past. While it still may have some bugs and problems, they are said to be unlikely to cause data loss. BTRFS improves a lot with every new linux kernel release, we are using it since linux 3.17 in production and even longer on our workstations.

Using BTRFS Snapshots for Backup

- As said before, BTRFS is improving with every linux kernel release, so it is recommended to use a recent kernel version, like 4.9 LTS or newer.
- Obvious: because we want to use BTRFS snapshotting, we need to put the data we want to save on a btrfs volume.
- You need to have BTRFS filesystems running on **both** source (production) and destination (backup) server.
- if you dont have btrfs already, you can resize your current filesystem to make some space on the hard drive and create a btrfs there. If you just want to test, you can create a image file and mount it using

losetup.

- if you are already using btrfs as rootfs, you can use `btrfs subvolume create <path>`
  to create a new subvolume, i.e. for MySQL. why should you? Because you may want to backup the mysql data dir more often than the rest of the system. Btrfs snapshots work on per-volume basis, so if you want to backup files independently (or exclude them form backup) you have to put them into a subvolume. You can backup multiple subvolumes from one machine, you just have to create a backup job for each.
    - i create a subvolumes to exclude fast-growing log directories and temporary folders with cache files that are not required to be backed up.
- you can use the BTRFS compress mount options (like compress=lzo) on the source, the destination or both.
- Further informations on incremental backups with BTRFS can be found in the official kernel wiki, https://btrfs.wiki.kernel.org/index.php/Incremental_Backup

Always disable COW on Database directories!

This is the most important lesson i have learned in years of BTRFS-Usage: Always disable Copy-On-Write on Database Data directories. BTRFS has a COW feature that is enabled by default even if there is no snapshot or hardlink present. As far as i know, this should avoid data corruption on partly overwritten files in case of power outages. But because this „feature“ is btrfs specific and not present in most other filesystems, it can be considered optional. For rootfs and most parts of the filesystem, this does not hurt, because files are rarly modified. All DBMS like Mysql or postgres do not expect a COW feature to be presend and are doing COW and transactional writes by their own.

There is no point in having FS-level COW for a database directory. Quite

the opposite: having BTRFS-COW enabled on a database directory causes massive fragmentation of the filesystem, slows it down and can lead to BTRFS crashes within some months. If you have your Database already running on BTRFS without explicitly disabled COW, DO IT ASAP before things get worse.

- If you are creating a fresh BTRFS for your database, mount it always with the *nodatacow* option BEFORE writing the first file.
- COW is enabled/disabled on file creation time. if you have created files with COW and mount the filesystem with nodatacow later, the old files are still COW enabled. You need to copy them to actually disable COW.
- there is a no-COW flag you can set on directories by using `chattr +C /path`. Setting +C on a directoriy causes all child folders/files to be no-COW. you can check this using *lsattr* command.
- If you already have your database with COW, you can disable it like this (i.e. for MySQL):

```
/etc/init.d/mysql stop # make sure your database is stopped!
mv /var/lib/mysql /var/lib/mysql_old
mkdir /var/lib/mysql
chattr +C /var/lib/mysql
cp -a /var/lib/mysql_old/* /var/lib/mysql
rm -rf /var/lib/mysql_old
chown -R mysql:mysql /var/lib/mysql
/etc/init.d/mysql start #start db again
```

- Note that if your database is large and has already a lot of fragmentation, the copy process can take very (in some cases very very) long. You may test how long the copy takes before you shutdown the database on a production system.
    - If you want to speed up the copy process, you can use `btrfs filesystem defrag -v -r -f /var/lib/mysql/` while your database is running.

BTRFS-SxBackup as Backup Manager

BTRFS-SxBackup is simple python CLI Appliation which makes it very easy to create, transfer and manage BTRFS snapshots. I highly recommend it, as its easy to use. After setting it up, you can run it with a cronjob.

- install BTRFS-SxBackup from https://github.com/masc3d/btrfs-sxbackup on the backup server. With btrfs-sxbackup on the destination, the backup will be "pulled" and centrally managed by the backup server (the client only requires to have ssh running). You also can setup btrfs-sxbackup on the source system and do a "push" backup if you want, both is supported.
    - on the destination (Backup) server, **i strongly recommend to use LVM and create one BTRFS volume for every backup task.** Do not make the volumes too large, you can easily extend them if one needs more space. In the past, we had the problem that storing and managing a lot of snapshots in one btrfs destination volume can cause bugs that render one filesystem unwriteable. Creating one btrfs volume per backup job makes BTRFS faster and safer – and in case of BTRFS problems, only one backup is affected. (and you have space left to create a fresh BTRFS and continue backups there). After doing it this way on the backup server, all the trouble we previously had with a single btrfs-backup-fs went away.
    - create a ssh key for root on the backup server and add it to the authorized_keys file on the source. also for the root user. btrfs-sxbackup requires to create, transfer and delete snapshots and only root can do that.
    - You have to use the btrfs-sxbackup init and run commands to setup and start the backup process. see the BTRFS-SxBackup git readme for more information. you can also turn on transfer compression
    - BTRFS-SxBackup has a retention feature; i.e. if you do a database backup every hour, you can keep the latest two on the source, and configure rules like "after two weeks, keep only 4 backup per day"

- Restore: if you are using btrfs for (root) filesystem backups, you can directly read the plain files from the backup servers filesystem. if you require a full restore, you can use `btrfs send` and *`btrfs receive`* commands to restore your backup.
    - you can also boot your backup server into a snapshot by using the subvol= kernel option. See also here

Tipps on Restoring a MySql/MariaDB Database Snapshot

if you are using BTRFS for MySQL backups, you most likely want to investigate the data from your MySQL tables or restore a single table or database.

- In this case, install the same major MySQL/MariaDB version as you have on the source system.
- The snapshots on the backup server are read-only, mysql will be unable to start on a read-only filesystem. to resolve that, create a new snapshot of the read-only snapshot, which will be writeable by default: `btrfs sub snap sx-<backupname> <restorename>`
- In case the source system has a different user id for mysql than the backup system, so you have to do a chown -R mysql:mysql on the writeable volume.
- Configure the my.cnf data dir to point to the restore volume and start up mysql. watch the <hostname>.err logfile within the restore volume for startup process. You do not need to copy the live MySQL configuration, just make sure the most important options are set correctly.
- After the startup/recovery completed successfully, you can login with your production user/passwords on the backup instance, inspect data and use mysqldump to transfer tables back to the production system. you can also use btrfs send/receive to transfer the complete snapshot back to the source in case if a disaster recovery is needet.
- If you have a backup instance running, its a fully functional and

writeable snapshot of the production data, which can be also used for testing before deployment.

Known Problems and Solutions

Help! I get „*no space left on device*„, but i have plenty of free space left!

This is a common problem with BTRFS. It took me some time to get a Idea whats going wrong there. My explanation on this is that BTRFS has data and metadata blocks allocated on the physical device. Sometimes it happens that all free space is allocated by almost-empty data blocks. When writing new data, BTRFS comes to the point where the new Metadata does not fit within the existing blocks and btrfs needs to allocate a new metadata block. Because all free space is already claimed by partly or fully empty data blocks, there is no space left to allocate a new metadata block. This gives you „no space left on device“, not because your data does not fit, but BTRFS is unable to acquire space for the metadata.
The solution is to issue a `btrfs balance start /path -dusage=x`. The balance command searches for blocks that have only x% or less of there space occupied, moves their data to other blocks and deletes them afterwards, freeing up space for new block allocations. You can start with 5 or 10% and can go up to (or just below) the occupied percentage that is displayed by df.

I have read in the BTRFS docs that BTRFS balances it self when needed. I cannot confirm this, so on important production systems, i setup a daily cronjob with `btrfs balance start -dusage=20`to make sure i do not run into problems. Maybe this will is fixed or will be fixed in the next btrfs versions, but im not rolling a dice on that.

i also get „no space left on device“ when i run btrfs balance

BTRFS needs some disk space to free up disk space, yes. first, try -musage

instead of -dusage and see if this resolves the problem. You can also try to delete some files, but this can also lead to „no space left on device“ errors.

The final solution to this is to temporarly add a new block device to the full filesystem so it can be balanced. You can do so by the `btrfs drive add` command. if you do not have a spare partition to add, you can also write a file to other filesystems including a ram-only tmpfs using dd, then make a blockdevice using losetup and add it to btrfs. A size of 2-5GB works most of the time. Start balancing with -dusage=0 and increment slowly until some blocks are relocated. After some relocations you can remove the added device with `btrfs drive remove` command.
Attention: if you add a tmpfs file to your BTRFS filesystem, knock on the wood that there will be no power outage or kernel crash or you will damage your filesystem. Using a tmpfs is a last-resort solution. Better add a USB storage or a network storage using nfs or sshfs if possible.