Linux Storage

INTRODUCTION

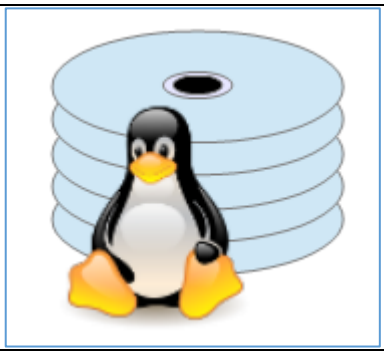


Table of contents

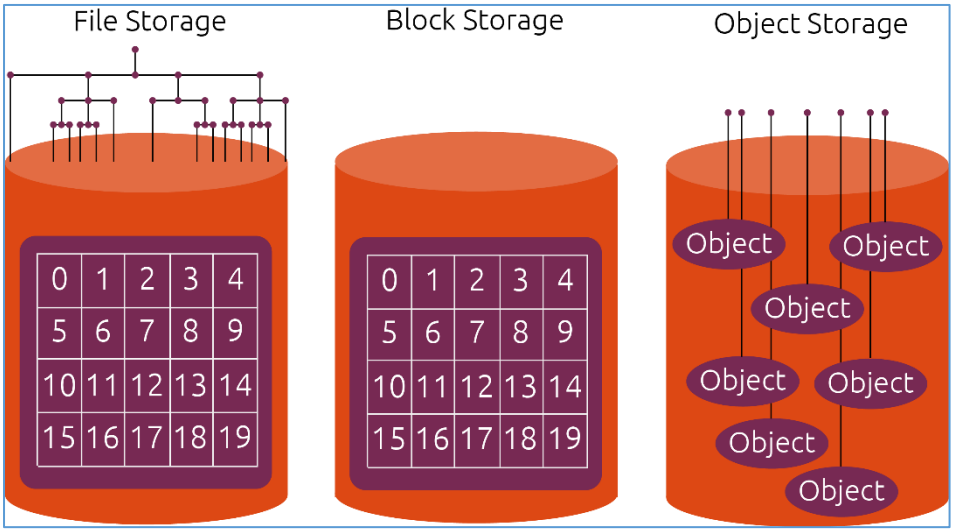
1. Introduction

Linux has robust systems and tooling to manage hardware devices, including storage drives. In this document we'll cover, at a high level, how Linux represents these devices and how raw storage is made into usable space on the server.

2. What is Block Storage?

Block storage is another name for what the Linux kernel calls a block device. A block device is a piece of hardware that can be used to store data, like a traditional spinning hard disk drive (HDD), solid state drive (SSD), flash memory stick, etc. It is called a block device because the kernel interfaces with the hardware by referencing fixed-size blocks, or chunks of space.

So basically, block storage is what you think of as regular disk storage on a computer. Once it is set up, it basically acts as an extension of the current filesystem tree, and you can write to or read information from the drive seamlessly.



3. What are Disk Partitions?

Disk partitions are a way of breaking up a storage drive into smaller usable units. A partition is a section of a storage drive that can be treated in much the same way as a drive itself.

Partitioning allows you to segment the available space and use each partition for a different purpose. This gives the user a lot of flexibility allowing them to potentially segment their installation for easy upgrading, multiple operating systems, swap space, or specialized filesystems.

While disks can be formatted and used without partitioning, some operating systems expect to find a partition table, even if there is only a single partition written to the disk. It is generally recommended to partition new drives for greater flexibility down the road.

MBR vs GPT

When partitioning a disk, it is important to know what partitioning *format* will be used. This generally comes down to a choice between MBR (Master Boot Record) and GPT (GUID Partition Table).

MBR is the traditional partitioning system, which has been in use for over 30 years. Because of its age, it has some serious limitations. For instance, it cannot be used for disks over 2TB in size, and can only have a maximum of four primary partitions. Because of this, the fourth partition is typically set up as an "extended partition", in which "logical partitions" can be created. This allows you to subdivide the last partition to effectively allow additional partitions.

GPT is a more modern partitioning scheme that attempts to resolve some of the issues inherent with MBR. Systems running GPT can have many more partitions per disk. This is usually only limited by the restrictions imposed by the operating system itself. Additionally, the disk size limitation does not exist with GPT and the partition table information is available in multiple locations to guard against corruption. GPT can also write a "protective MBR" which tells MBR-only tools that the disk is being used.
In most cases, GPT is the better choice unless your operating system or tooling prevent you from using it.

4. Formatting and Filesystems

While the Linux kernel can recognize a raw disk, the drive cannot be used as-is. To use it, it must be formatted. Formatting is the process of writing a filesystem to the disk and preparing it for file operations. A filesystem is the system that structures data and controls how information is written to and retrieved from the underlying disk. Without a filesystem, you could not use the storage device for any file-related operations.
There are many different filesystem formats, each with trade-offs across a number of different dimensions, including operating system support. On a basic level, they all present the user with a similar representation of the disk, but the features that each supports and the mechanisms used to enable user and maintenance operations can be very different.
Some of the more popular filesystems for Linux are:

Ext4: The most popular default filesystem is Ext4, or the fourth version of the extended filesystem. The Ext4 filesystem is journaled, backwards compatible with legacy systems, incredibly stable, and has mature support and tooling. It is a good choice if you have no specialized needs.

XFS: XFS specializes in performance and large data files. It formats quickly and has good throughput characteristics when handling large files and when working with large disks. It also has live snapshotting features. XFS uses metadata journaling as opposed to journaling both the metadata and data. This leads to fast performance, but can potentially lead to data corruption in the event of an abrupt power loss.

Btrfs: Btrfs is modern, feature-rich copy-on-write filesystem. This architecture allows for some volume management functionality to be integrated within the filesystem layer, including snapshots, cloning, volumes, etc. Btrfs still runs into some problems when dealing with full disks. There is some debate over its readiness for production workloads and many system administrators are waiting for the filesystem to reach greater maturity.

ZFS: ZFS is a copy-on-write filesystem and volume manager with a robust and mature feature set. It has great data integrity features, can handle large filesystem sizes, has typical volume features like

snapshotting and cloning, and can organize volumes into RAID and RAID-like arrays for redundancy and performance purposes. In terms of use on Linux, ZFS has a controversial history due to licensing concerns. Ubuntu is now shipping a binary kernel module for it however, and Debian includes the source code in its repositories. Support across other distributions is yet to be determined.

5. How Linux Manages Storage Devices

5.1. Device Files in /dev

In Linux, almost everything is represented by a file. This includes hardware like storage drives, which are represented on the system as files in the `/dev` directory. Typically, files representing storage devices start with `sd` or `hd` followed by a letter. For instance, the first drive on a server is usually something like `/dev/sda`.
Partitions on these drives also have files within `/dev`, represented by appending the partition number to the end of the drive name. For example, the first partition on the drive from the previous example would be `/dev/sda1`.
While the `/dev/sd*` and `/dev/hd*` device files represent the traditional way to refer to drives and partitions, there is a significant disadvantage of in using these values by themselves. The Linux kernel decides which device gets which name on each boot, so this can lead to confusing scenarios where your devices change device nodes.

To work around this issue, the `/dev/disk` directory contains subdirectories corresponding with different, more persistent ways to identify disks and partitions on the system. These contain symbolic links that are created at boot back to the correct `/dev/[sh]da*` files. The links are named according to the directory's identifying trait (for example, by partition label in for the `/dev/disk/by-partlabel` directory). These links will always point to the correct devices, so they can be used as static identifiers for storage spaces.

Some or all of the following subdirectories may exist under `/dev/disk`:

`by-label`: Most filesystems have a labeling mechanism that allows the assignment of arbitrary user-specified names for a disk or partition. This directory consists of links that named after these user-supplied labels.
`by-uuid`: UUIDs, or universally unique identifiers, are a long, unique string of letters and numbers that can be used as an ID for a storage resource. These are generally not very human-readable, but are pretty much guaranteed to be unique, even across systems. As such, it might be a good idea to use UUIDs to reference storage that may migrate between systems, since naming collisions are less likely.
`by-partlabel` and `by-partuuid`: GPT tables offer their own set of labels and UUIDs, which can also be used for identification. This functions in much the same way as the previous two directories, but uses GPT-specific identifiers.
`by-id`: This directory contains links generated by the hardware's own serial numbers and the hardware they are attached to. This is not entirely persistent, because the way that the device is connected to the system may change its `by-id` name.
`by-path`: Like `by-id`, this directory relies on the storage devices connection to the system itself. The links here are constructed using the system's interpretation of the hardware used to access the device. This has the same drawbacks as `by-id` as connecting a device to a different port can alter this value.

Usually, `by-label` or `by-uuid` are the best options for persistent identification of specific devices.

5.2. Mounting Block Devices

The device file within `/dev` are used to communicate with the Kernel driver for the device in question. However, a more helpful abstraction is needed in order to treat the device as a segment of available space.

In Linux and other Unix-like operating systems, the entire system, regardless of how many physical devices are involved, is represented by a single unified file tree. As such, when a filesystem on a drive or partition is to be used, it must be hooked into the existing tree. Mounting is the process of attaching a formatted partition or drive to a directory within the Linux filesystem. The drive's contents can then be accessed from that directory.

Drives are almost always mounted on dedicated empty directories (mounting on a non-empty directory means that the directory's usual contents will be inaccessible until the drive is unmounted). There are many different mounting options that can be set to alter the behavior of the mounted device. For example, the drive can be mounted in read-only mode to ensure that its contents won't be altered.

The Filesystem Hierarchy Standard recommends using `/mnt` or a subdirectory under it for temporarily mounted filesystems. If this matches your use case, this is probably the best place to mount it. It makes no recommendations on where to mount more permanent storage, so you can choose whichever scheme you'd like. In many cases, `/mnt` or `/mnt` subdirectories are used for more permanent storage as well.

5.3. Making Mounts Permanent with /etc/fstab

Linux systems look at a file called `/etc/fstab` (filesystem table) to determine which filesystems to mount during the boot process. Filesystems that do not have an entry in this file will not be automatically mounted (the exception being those defined by systemd `.mount` unit files, although these are not common at the moment).
The `/etc/fstab` file is fairly simple. Each line represents a different filesystem that should be mounted. This line specifies the block device, the mount point to attach it to, the format of the drive, and the mount options, as well as a few other pieces of information.

6. More Complex Storage Management

While most simple use cases do not need additional management structures, more performance, redundancy, or flexibility can be obtained by more complex management paradigms.

6.1. What is RAID ?

RAID stands for redundant array of independent disks. RAID is a storage management and virtualization technology that allows you to group drives together and manage them as a single unit with additional capabilities.
The characteristics of a RAID array depend on its RAID level, which basically defines how the disks in the array relate to each other. The level chosen has an impact on the performance and redundancy of the set. Some of the more common levels are:
RAID 0: This level indicates drive striping. This means that as data is written to the array, it is split up and distributed among the disks in the set. This offers a performance boost as multiple disks can be written to or read from simultaneously. The downside is that a single drive failure can lose all of the data in the entire array, since no one disk contains enough information about the contents to rebuild.
RAID 1: RAID 1 is basically drive mirroring. Anything written to a RAID 1 array is written to multiple disks. The main advantage is data redundancy, which allows data to survive hard drive lose in either side of the mirror. Because multiple drives contain the same data, usable capacity is reduced half.
RAID 5: RAID 5 stripes data across multiple drives, similar to RAID 0. However, this level also implements a distributed parity across the drives. This basically means that if drive fails, the remaining drives can rebuild the array using the parity information shared between them. The parity information is enough to rebuild any one disk, meaning the array can survive any one disk loss. The parity information reduces the available space in the array by the capacity of one disk.
RAID 6: RAID 6 has the same properties as RAID 5, but provides double parity. This means that RAID 6 arrays can withstand the loss of any 2 drives. The capacity of the array is again affected by the parity amount, meaning that the usable capacity is reduced by two disks worth of space.
RAID 10: RAID 10 is a combination of levels 1 and 0. First, two sets of mirrored arrays are made. Then, data is striped across them. This creates an array that has some redundancy characteristics while providing good performance. This requires quite a few drives however, and the total capacity is half of the combined disk space.

6.2. What is LVM ?

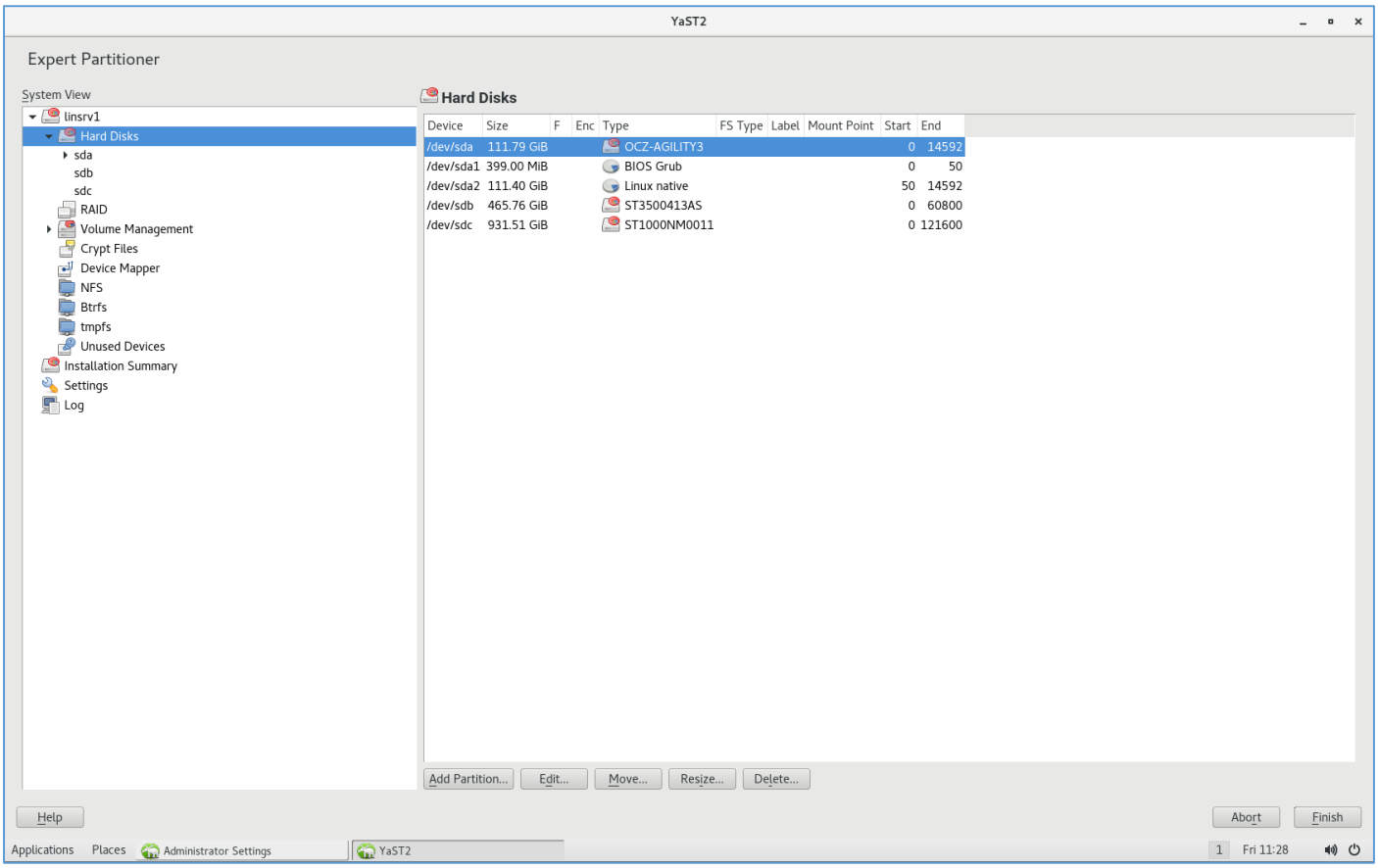
LVM, or Logical Volume Management, is a system that abstracts the physical characteristics of the underlying storage devices in order to provide increased flexibility and power. LVM allows you to create groups of physical devices and manage it as if it were one single block of space. You can then segment the space as needed into logical volumes, which function as partitions.

LVM is implemented on top of regular partitions, and works around many of the limitations inherent with classical partitions. For instance, using LVM volumes, you can easily expand partitions, create partitions that span multiple drives, take live snapshots of partitions, and moving volumes to different physical disks.

LVM can be used in conjunction with RAID to provide flexible management with traditional RAID performance characteristics.

7. Practices

With Linux SLES 12 Expert Partitioner



8. References

https://www.digitalocean.com/community/tutorials/an-introduction-to-storage-terminology-and-concepts-in-linux