Skalierbarer Speicher zu Hause

Cluster Storage vom Datacenter ins Heim.

Alexander Schreiber <als@thangorodrim.ch>

http://www.thangorodrim.ch/

Chemnitzer Linux-Tage 2021, 2021-03-14

Nicht alles was zählt, kann gezählt werden, und nicht alles was gezählt werden kann, zählt!

– *Albert Einstein*

Inhalt

Über den Autor

- beschäftigt sich seit über 20 Jahren mit Linux
- tätig als Systemingenieur bei Google Switzerland
- hat den einen oder anderen Computer im Haus
- disclaimer: Dieser Vortrag hat nichts mit meinem Arbeitgeber zu tun und entspringt reinem Privatvergnügen.

Um was geht es?

- Skalierbare Speicherlösung für den Heimbedarf
- Welche Lösungen gibt es?
- Übersicht über die Kandidaten der engeren Wahl
- Praktische Erfahrungen

Was suchen wir?

- einfach erweiterbare Speicherlösung
- einfach aufzusetzen und zu betreiben (kein dedizierter Storage Admin)
- einfach zu benutzen - POSIX Dateisystem Semantik wäre nett
- moderate Komplexität und Resourcen-Anforderungen
- kleiner Preis fürs Gesamtsystem (Einsatz zu Hause)
- Redundanz (**alle** Hardware fällt irgendwann mal aus), insbesondere Robustheit auch bei Komplettausfall von Knoten.
- einfache Datenrettung auch nach katastrophalen Ausfällen
- OpenSource Lösung (keine Lizenzkosten/Herstellerbindung)

Speicherlösungen, klassisch

- einzelne Platte: zu klein, nicht redundant
- mehrere einzelne Platten: viel Spass beim Jonglieren ...
- RAID: redundant, aber nicht einfach erweiterbar
- ZFS: vielversprechend, aber nicht trivial und mit Eigenheiten
- Wie macht man das eigentlich im grossen Massstab?

Speicherlösungen für die Grossen

- grosse Datenbestände ausfallsicher vorhalten → altes Problem
- klassische Industrielösung: grosse Disk-Arrays, ggf. doppelt
- skaliert aber nur begrenzt (Stapel von Disk-Arrays ...)
- Bedarf für **noch** (viel) mehr Speicher ... z.B.:
  - Inhaltsanbieter: grosse Audio-/Bild-/Video-Archive
  - Wissenschaftliches Rechnen, z.B. CERN 280PB Kapazität
  - Cloud-Dienstleister
- Cluster-Dateisysteme, Standard- (CERN Ceph) & Hausprojekte (CERN EOS)
- grosse Datenbestände erweiterbar & ausfallsicher → gelöst für grosse Systeme
- Speicher in Rechenzentrengrösse vs. Heimeinsatz
- Skaliert das auch nach unten?

Cluster-Dateisysteme, Wunschliste

- schon eine Weile im allgemeinen Einsatz
- in aktiver Entwicklung
- einfache Installation
  - keine `wget ..; ./configure ; make install` Tänze
  - `apt install ...` wäre prima
- einfacher Betrieb
- moderate Resourcen-Anforderungen (z.B. RAM & CPU)
- Robustheit gegenüber Hardwareausfällen (kein Datenverlust)
- stellt am Client POSIX-Semantik bereit
- keine externen Infrastruktur-Abhängigkeiten

Cluster-Dateisysteme, eine Marktübersicht

- nicht abschliessende Übersicht
- begrenzt auf OpenSource Projekte
- mit Fokus auf unsere Anforderungen

Apache HDFS (Hadoop Distributed FileSystem)

- fehlertolerant und für Einsatz auf preiswerter Hardware
- optimiert für hohen Durchsatz und grosse Datenbestände (Racks)
- Redundanz durch interne Replikation (Standard: 3x)
- ursprünglich für Apache Nutch Suchmaschine
- Komponenten:
  - Namenode: Metadaten-Verwaltung (2 Kopien)
  - Datanode: Datenspeicher (alle anderen Knoten)
- grosse Datenblöcke (128MB typisch), Streaming
- Zugriff: FUSE, Java API, Rest API
- $\Rightarrow$ etwas zu gross

Ceph

- Objekt-, Block- und Dateispeicher
- Basistechnology: RADOS (Reliable Autonomic Distributed Object Store)
- Linux Kernel Unterstützung: RDB (RADOS Block Device)
- Ceph Dateisystem entweder via FUSE oder direkt als Dateisystem
- speicherhungrig: 3-5GB für Bluestore, 1GB/1TB Speicher, 1GB/Daemon
- Komponenten: Metadata-Server (MDS), OSD (Object Storage Daemon), Monitore und Manager Dienste
- Zugriff: FUSE, Dateisystem (ceph), RADOS Blockspeicher, Rest API (S3/Openstack Swift), RBD
- Flaschenhalsvermeidung: CRUSH (Hashing) für Datenverteilung, direkte Client - OSD Kommunikation
- ⇒ Kandidat, aber der Speicherbedarf...

GlusterFS

- verteiltes, repliziertes Dateisystem, POSIX Semantik
- Basiselement: storage brick (= Server mit glusterfsd)
- Datenbestand auf lokalem Dateisystem (XFS) direkt abgelegt
- Replikation auf Server-Seite, komplexere Umsetzer auf Client-Seite
- Flaschenhalsvermeidung: Direkte Client - storage brick Kommunikation (DHT)
- Spiegelung, Replikation, Quotas, Snapshots, Trashcan
- relativ einfaches Setup, moderate Anforderungen
- Client via FUSE implementiert, NFS unterstützt
- nicht auf Linux begrenzt (z.B. NetBSD)
- $\Rightarrow$ Kandidat

MooseFS/LizardFS

- fehlertolerant, hochverfügbar, skalierbar, hochleistungsfähig
- Komponenten: Metadata Server (MDS), Metalogger Servers, Chunk Servers
- entwickelt für rechenzentrumsgrosse Dateisysteme
- Client Zugriff via FUSE
- Web GUI für Admin, Monitoring
- Verfügbar als normale und Pro Version - für vollen Leistungsumfang (erasure coding, Metadaten HA, Windowsclient) kommerzielle Lizenz nötig
- fork: LizardFS (GPLv3) liefert erasure coding und HA
- $\Rightarrow$ kein Kandidat

Lustre

- paralleles verteiltes Dateisystem
- Komponenten: Management Server (MGS), Metadata Server (MDS), Object Storage Server (OSS)
- Client-Zugriff: Linux Kernel Treiber
- optimiert für massive Umgebungen (z.B. Hochleistungsrechnen)
- nicht unerhebliche Komplexität intern
- skaliert nicht so recht nach unten
- $\Rightarrow$ kein Kandidat

Parallel Virtual File System (PVFS)

- hochskalierbares paralleles Dateisystem fürs Hochleistungsrechnen
- entwickelt für Linux-Rechencluster
- blockweise Verteilung von Dateien auf IO-Knoten (striping)
- optimiert auf hohe I/O Bandbreite & für Rechenclusterbetrieb
- Weiterentwicklung: OrangeFS (für breitere Anwendungsbereiche)
- $\Rightarrow$ kein Kandidat, wir wollen keine Rechencluster bauen

DRBD + NFS

- Distributed Replicated Block Device (DRBD) + NFS + LinuxHA
- DRBD für Redundanz & Replikation auf Blockgeräteebene
- NFS für den Clientzugriff
- Autor hat sehr gute Erfahrungen damit als “RAID1 übers Netzwerk”
- LinuxHA für die Hochverfügbarkeit bei Ausfällen
- machbar, aber nicht ganz einfach und sehr begrenzt in den Möglichkeiten
- $\Rightarrow$ kein Kandidat

Hardware-Vorstellungen

- idealerweise: viele grosse AMD64 Server mit viel RAM → teuer, laut, Platzbedarf
- realistische Anforderungen:
  - klein (keine 19-Zoll Rack-Maschinen)
  - stromsparend und leise
  - preiswert in der Beschaffung
  - von Standard Linux-Distribution unterstützt
  - keine “nur dieses Jahr verfügbar” Exoten
  - SATA-Schnittstelle, GBit-Netzwerk, ausreichend CPU und RAM
  - keine kreativen Handwerksarbeiten nötig zum Aufbau (“Laubsägeprojekt”)

Mögliche Kandidaten, Auswahl

- Raspberry Pi 4: GBit Ethernet, kein SATA, USB3 → USB HDD
- Cubox (alle Versionen): GBit Ethernet, eSATA → eSATA Gehäuse
- ODroid C1/C1+/C2: Gigabit Ethernet, USB2 (480 MBit/s Flaschenhals)
- ODroid XU4: Gigabit Ethernet, kein SATA, USB3 → USB HDD
- ODroid H2: 2x Gigabit Ethernet, 2x SATA, x86: €130 + Gehäuse für 2x HDD
- ODroid HC1/HC2 (XU4 basiert): Gigabit Ethernet, SATA, stapelbarer Kühlkörper, genug CPU & 2GB RAM, €65 ⇒ Kandidat

Hardkernel ODroid HC2: Home Cloud Two

- Samsung Exynos5422: 4x Cortex-A15 2.0Ghz & 4x Cortex-A7 1.4GHz
- 2Gbyte RAM, SATA-3 Anschluss (USB3), Gigabit Ethernet, USB 2.0 Host
- UHS-1 micro-SD card slot (Bootmedium), Kernel 4.14 LTS basierte Server-Distro
- Aluminium-Rahmen als Träger für Platine & Festplatte sowie als Kühlkörper
- z.B. bei Pollin: €64.95 (Board) + €8.50 (Netzteil) = €73.45

Hardwareauswahl

- Maschine: ODroid HC2, platz- & stromsparend, preiswert, GBit Netzwerk & SATA
- Herstellung bis Ende 2021 garantiert, aber für länger erwartet
- 3.5" Festplatten verbaubar → viel Speicherplatz möglich
- verschiedene Linux-Distributionen verfügbar: Debian, OpenMediaVault, ...
- einfach erhältlich: direkt von Hardkernel oder von Distributoren in .de (und .ch)
- System auf μSD-Karte → einfache Sicherung/Reparatur/Wiederherstellung
- für den Ernstfall serielle Konsole (@115200) vorhanden

Basisinstallation

- Debian Buster oder Armbian/Raspbian (basiert auf Debian Buster)
- ssh aktivieren, key-basiertes root-Login, pi-user löschen
- root ssh keys für reine Verwaltungs-VM verteilen
- munin & node-exporter install & config
- unattended-upgrades (apt update && apt upgrade -y wird schnell alt)
- ntp, cryptsetup, sysstat, postfix, smartmontools, rsyslog

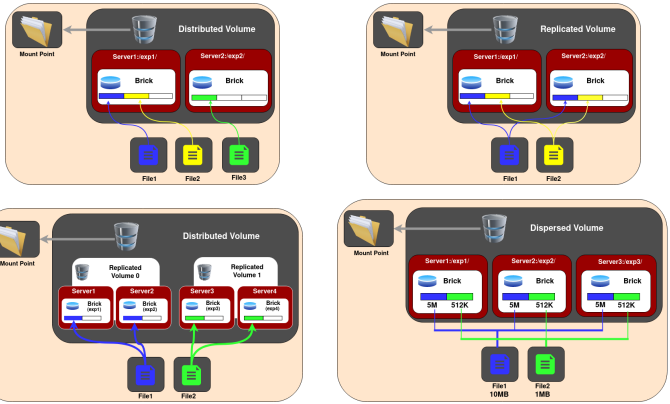
Verschlüsselung

- Szenario: Platte fällt aus, in Garantie → Austausch möglich
- kostet Durchsatz:
  - Festplatte direkt: 100+ MB/s read/write
  - dmcrypt: 20-25 MB/s read/write
  - dmcrypt + dm-integrity: 12 MB/s write, 20-25 MB/s read
- dm-crypt + dm-integrity?
  - authenticated encryption (AEAD), Blockprüfsumme (SHA256)
  - Blockprüfsumme kostet ca. 6% Speicherplatz
  - bei Prüfsummenfehler: Lesefehler ‘‘`crypt: INTEGRITY AEAD ERROR, sector 39840`’’
  - alle Schreibzugriffe doppelt (durch Journaling)

GlusterFS

- verschiedene Volume-Typen
  - distributed: Dateien über alle Disks verteilt, keine Redundanz
  - replicated: Dateien über alle Disks gespiegelt, Redundanz
  - distributed replicated: beides zusammen, Redundanz
  - dispersed: Daten verteilt, erasure codes, Redundanz
  - distributed dispersed, Redundanz
- Client Struktur:
  - FUSE, darunter verschiedene Übersetzer-Ebenen
  - I/O stats & performance
  - sharding (Option) & DHT (Verteilte Hashtabelle)
  - client-to-brick mapping

GlusterFS Volume Typen



Grundlagen

- OS auf μSD-Karte, kein Swap
- reguläre Archivierung der μSD-Karte zwecks schneller Systemwiederherstellung wenn (nicht falls) die μSD-Karte ausfällt
- Daten auf SATA Festplatte mit dmcrypt
- Dateisystem: XFS (Empfehlung Gluster), btrfs war eher instabil
- Plan: 2 Volumes, 1x regulär, 1x dm-integrity
- Volumes nur replicated (nicht distributed oder erasure coding) $\rightarrow$ Daten ohne Gluster wiederherstellbar, n=3
- Ziel: hochverfügbares Backup-Archiv

- Basisinstallation Armbian auf ODroid HC2
- `apt install glusterfs-server`
- Dienste aktivieren (`systemctl enable ...`)
- Dienste starten (`systemctl start ...`)
- alle Knoten: Datendisk `/dev/sda`: cryptsetup, mkfs.xfs, mount `/gluster`
- `gluster peer probe $NODE`
- `gluster volume create glusterfs replica 3 node{1..3}:/gluster/glusterfs`
- `gluster volume start glusterfs`
- $\rightarrow$ 1 replicated volume mit $N = 3$

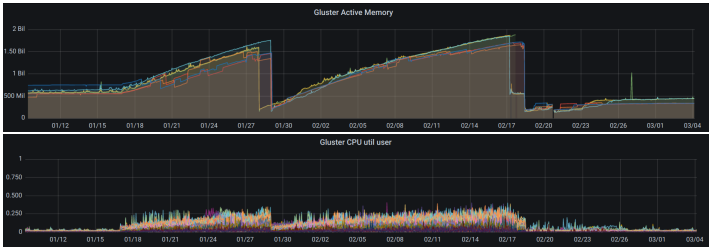
Unser erstes Gluster volume

gluster volume info glusterfs

Volume Name: glusterfs
Type: Replicate
Volume ID: 14dhi81d-j07k-49e4-a375-e7a8y41z2eb0
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: node1.some.place:/gluster/glusterfs
Brick2: node2.some.place:/gluster/glusterfs
Brick3: node3.some.place:/gluster/glusterfs

Monitoring Gluster

- munin & Prometheus node-exporter
- gluster-exporter existiert (unabhängiges Projekt)
- triggert aber Speicherlecks bis zum OOM Crash der gluster-Daemons and hält die CPU warm



client-side

- Client implementiert via FUSE (Filesystem in USErspace)
- Client-Installation:
  - Debian: `apt install glusterfs-client`
  - NetBSD: `cd /usr/pkgsrc/filesystems/pkgsrc ; make install`
- mount: `mount -t glusterfs brick01:/glusterfs /glusterfs`
- Mount kann gegen beliebigen Host im Cluster erfolgen

Betriebserfahrungen Gluster Cluster

- sehr stabil, solange min. 1 Knoten aktiv bleibt das GlusterFS verfügbar
- läuft prima mit Linux/x86, Linux/arm, NetBSD/x86, NetBSD/sparc64 Clients
- *eine* besonders langsame Platte kann den ganzen Cluster ausbremsen (Spiegel!)
- Datenwiederherstellung mit nicht-Gluster Maschine trivial: dmcrypt, mounten, kopieren (Hinweis: dmcrypt-Keys auch ausserhalb des Clusters sichern)
- dank dmcrypt nicht schnell beim Schreiben, aber schnell genug im Praxiseinsatz (primär für online-Backups), schnell im Lesen (parallel)
- aufsetzen, benutzen, tut einfach
- aber: nur replicated mode im Einsatz, distributed & erasure code haben das Potential für mehr . . . Beschäftigung

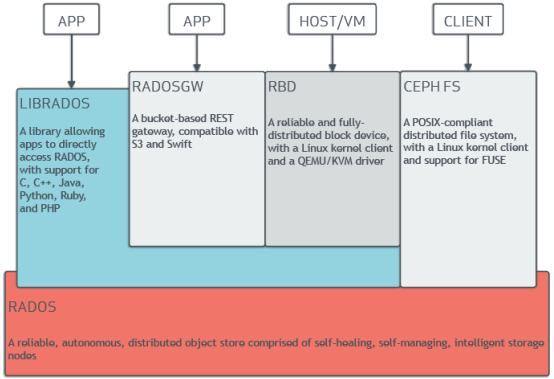
Ceph, Übersicht

- ursprünglich aussortiert wegen Speicherbedarf (1GB/1TB disk)
- ein grosser Stapel kleiner Festplatten → albern
- mittlerweile ARM Maschine mit viel Speicher verfügbar: Raspberry Pi 4 8 GB
- damit kann man doch arbeiten...
- dagegen: kein SATA, aber 2x USB3
- 5 TB USB3 HDDs relativ günstig greifbar
- Versuch macht kluch!

Randnotiz Raspberry Pi 4 8 GB

- 8 GB RAM → wird sicher 64bit sein, oder?
- Guter Witz, das.
- Raspbian ist 32 Bit (mit 8 GB RAM total, aber 2/2 Split: 2 GB/Process)
- Raspbian 64 Bit "in Arbeit", letzte Testversionen von Mai & August 2020
- 64Bit Kernel in `config.txt` aktivieren, reboot.
- `uname` sagt aarch64, `ceph status` sagt Bus Error.
- Ok, bleiben wir bei 32 Bit. 2 GB/Prozess wird noch interessant
- Debian Bullseye (testing) ist aarch64 auf Raspberry4, Release abwarten

Ceph Stack


APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
CEPH FS
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE
RADOS
A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes

Ceph Komponenten

- 4x Daemons:
  - `ceph-mon`: internes Monitoring
  - `ceph-mgr`: Management
  - `ceph-mds`: MetaData-Server
  - `ceph-osd`: OnlineStorageDaemon
- Daemons (ausser `ceph-osd` laufen in mehreren Instanzen (failover)
- Daten auf lokalen Dateisystem (XFS) oder direkt in Bluestore
- Bluestore: schneller, Prüfsummen für (Meta-)Daten, mit Journal, Kompression, copy-on-write (Snapshots), RocksDB integriert
- Standardsetup: Storage Pool mit 3/2 Redundanzkonfiguration

Ceph und Verschlüsselung

- Ceph Bluestore hat feste Annahmen, direkt mit Speicherhardware zu interagieren
- Ceph Bluestore mag externen blockdevice-dmcrypt-ceph Layer gar nicht
- nur blockdevice-dmcrypt-filesystem-ceph Layer machbar (Filestore)
- direkt Blockdevice (z.B. `/dev/sda1`) an Ceph Bluestore verfüttern
- Ceph baut eigene, LVM basierte Struktur darauf
- Ceph macht dmcrypt auf LVM Logical Volume Ebene
- LUKSv1 aus historischen Gründen, eigene, interne Schlüsselverwaltung

Ceph Setup, Planung

- Standardsetup
  - dedizierte Speicherknoten, viel Disk, nur `ceph-osd` plus
  - dedizierte Verwaltungsknoten, wenig Disk, viel RAM, nur `ceph-(mgr,mon,msd)`
  - Beispiel: 5x Speicher, 3x Verwaltung = 8 Knoten
- Mein Setup:
  - kleiner Cluster (3-8 Knoten)
  - Verlust *beliebiger* Knoten soll verkraftbar sein
  - → alle Daemons auf allen Knoten aktiv, kombiniert

Ceph Setup

- `apt install ceph ceph-mds ceph-mgr ceph-mon ceph-osd`
- Das war einfach, und jetzt?
- offizielle Standardmethode(n):
  - nimm Ansible/Docker/Podman/Cockpit/MaaS/sonstige Magie
  - `curl https://github.com/ceph/ceph/raw/octopus/src/cephadm/cephadm`
  - `chmod +x cephadm; ./cephadm ...` Magie passiert
  - Ernsthaft?
  - Und manchmal ist die Magie kaputt, zahlreiche Postings in Webforen.
- Das ist Unix, ich will wissen, wie das System zusammengebaut wird.

Ceph Setup, von Hand

- Deutlich komplexer als Gluster.
- Komponenten Kommunikation gesichert mit pre-shared keys (cephx).
- Namenskonvention Daemons: Hostname
- https://wiki.gentoo.org/wiki/Ceph/Installation von "Cluster creation" bis zum Abschluss der Monitor-Installation
- dann https://docs.ceph.com/en/latest/install/index_manual/ ab "Manager Daemon Bootstraping"
- `ceph mgr module enable dashboard`
- `ceph mgr module enable prometheus`
- `systemctl enable` nicht vergessen für die Horde Daemons

Cluster status, Konsole

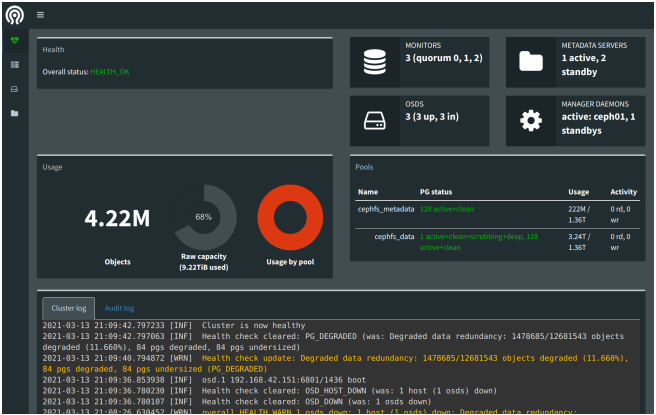
```
root@ceph01:~# ceph status
  cluster:
    id:     b6a384d6-9741-4352-9810-48c29375b426
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum ceph01,ceph02,ceph03
    mgr: ceph01(active), standbys: ceph02
    mds: cephfs-1/1/1 up  {0=ceph03=up:active}, 2 up:standby
    osd: 3 osds: 3 up, 3 in

  data:
    pools:   2 pools, 248 pgs
    objects: 4.23M objects, 2.95TiB
    usage:   9.23TiB used, 4.42TiB / 13.6TiB avail
    pgs:     247 active+clean
             1   active+clean+scrubbing+deep

  io:
    client:   0B/s rd, 0op/s rd, 0op/s wr
```

Cluster status, Ceph web



Ceph monitoring

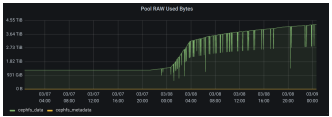
- integriertes Web-Dashboard: ceph-mgr Knoten, Port 7000
- integrierter Ceph-Prometheus Exporter (ceph-mgr)
- Prometheus Node-Exporter & Munin für Monitoring der Knoten
- Prometheus Process-Exporter für Prozess-Überwachung der ceph-* Daemons
- Grafana Community hat fertige Ceph Dashboards.

Ceph client

- `ceph.conf` und preshared Keys (cephx) auf Client hinterlegen
- CephFS Client im Linux Kernel seit 2.6.34 (2010-03-19)
- `mount -t ceph ceph01:/ /ceph`
- RBD (Rados Block Device) Support im Linux Kernel, mit KVM integriert
- qemu unterstützt RDB direkt für virtuelle Disks
- Zudem iSCSI Unterstützung.
- Ceph Object Storage auf Basis von radosgw (Ceph Object Gateway Daemon) bietet Object Storage mit S3 & Swift kompatibler API.

Betriebserfahrungen Ceph

- Einfach erweiterbares Speichersystem (einfach neue Knoten/HDDs) via CephFS.
- Schreiben mit ca. 35 MB/s (1 GBit LAN, 3 Kopien), lesen mit bis zu 100 MB/s.
- Unter Last laufen die OSDs in Speicherlimits und crashen mit Allozierungsfehlern, aber CephFS bleibt verfügbar, `ceph-osd` wird von `systemd` neu gestarted, Selbstheilung von Ceph, stabil. Keine Auffälligkeiten am Client.
- Komplettausfall eines Knotens: stabil nach Wiederverfügbarkeit Selbstheilung.
- Graph: Zacken nach unten = OSD crash:



Zusammenfassung

- Skalierbarer Speicher zu Hause: Cluster Dateisysteme.
- machbar mit sehr moderaten Hardware- und Betriebskosten (Strom).
- Empfehlung je nach Zweck:
  - Hochverfügbares, einfach rettbares Archiv für Backups, leicht aufzusetzen: GlusterFS
  - Hochverfügbarer Speicherplatz der einfach nachwachsen kann: Ceph
- Beide Projekte mit langer Erfahrung und aktuellem, stabilem Support.
- Bei beiden Projekten ist der langjährige Einsatz in grossen Umgebungen erkennbar (u.a. Stabilität trotz Knotenausfall).

Links

- Gluster https://gluster.org/
- Ceph: https://ceph.io/

Fragen?

Vielen Dank für Euer Interesse!