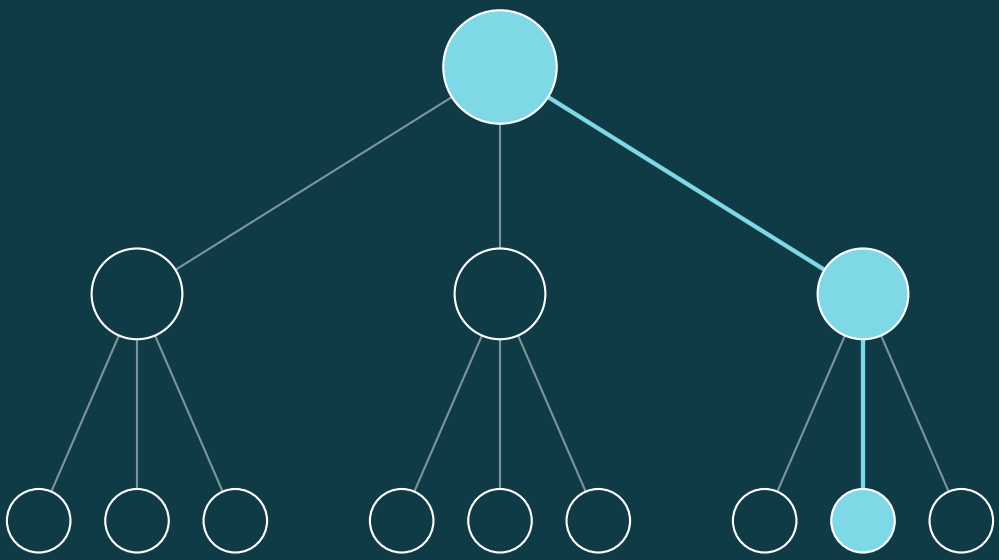
nyrvex.com

Ceph

Hands-On

Verteilten Speicher verstehen,
betreiben und reparieren



Mit wenig Text fit fürs Interview

Ceph

Hands-On

Verteilten Speicher verstehen, betreiben und reparieren

Mit wenig Text fit fürs Interview

Thomas Zachmann

Ceph Hands-On. Verteilten Speicher verstehen, betreiben und reparieren.
Stand: November 2026. © 2026 Thomas Zachmann. Lizenz: CC BY-ND 4.0.

Geschrieben für Ceph Tentacle (20.2) mit cephadm. Die Befehle und Beispielausgaben in diesem Buch wurden nicht auf einem Cluster ausgeführt. Zahlen in Ausgaben sind beispielhaft, und Details können sich je nach Version unterscheiden. Probiere alles zuerst im Labor aus, nie auf einem produktiven Cluster.

Schriften: Charter, Fira Sans, JetBrains Mono.

Inhalt

So liest du dieses Buch 5

Der Start **7**
1 Was ist Ceph? 8
2 Block, Datei, Objekt 11
3 Die Daemons 13
4 Das Labor 15

Ceph denken **19**
5 RADOS und Objekte 20
6 CRUSH 22
7 Placement Groups 25
8 PG-Zustände 28
9 Replikation 31
10 Erasure Coding 33
11 BlueStore 36
12 Das Netzwerk 38

Die Schnittstellen **41**
13 Pools und Rechte 42
14 RBD: Block Storage 45
15 CephFS 48
16 RGW: Object Storage 51
17 Kubernetes: Rook und CSI 54

Betrieb **57**
18 cephadm und Specs 58
19 OSDs tauschen 61
20 Kapazität 63
21 Wartung und Flags 65
22 Upgrades 68
23 Monitoring 70

Troubleshooting **73**
24 Die Methode 74
25 Ein OSD ist weg 76
26 PGs hängen 78
27 Alles ist langsam 80
28 Der Cluster ist voll 82
29 MON ohne Quorum 84
30 CephFS klemmt 86
31 Scrub-Fehler 88

Produktion **91**
32 Hardware und Topologie 92
33 Performance 94
34 Backup und Disaster Recovery 96
35 Sicherheit 99

Das Interview **101**
36 Die Fragen 102
37 Design-Aufgaben 106
38 Spickzettel 108
Stichwörter 111

So liest du dieses Buch

Du hast bald ein Interview und musst Ceph können. Nicht irgendwann, sondern bald. Dieses Buch ist genau dafür gebaut: wenig Text, viele kleine Beispiele, und zu jedem Thema die Frage, die dir wirklich gestellt wird.

Die Regeln

- **Tippe alles selbst.** Nichts kopieren. Deine Finger lernen mit.
- **Jeder Befehl einzeln.** Es gibt keine Skripte. Du tippst, du schaust, was passiert.
- **Nur im Labor.** Viele Befehle stören einen Cluster absichtlich. Drei VMs reichen, siehe Kapitel 4.
- **Verstehe zuerst das Warum.** Ceph-Interviews prüfen selten Befehle. Sie prüfen, ob du erklären kannst, was bei einem Ausfall passiert.

> **MERKE**
>
> Die Befehle und Ausgaben in diesem Buch wurden nicht auf einem Cluster ausgeführt. Die Ausgaben sind gekürzte Beispiele. Deine Zahlen und Texte werden abweichen.

Die Kästen

> **DAS BILD IM KOPF**
>
> *Ein Vergleich aus dem Alltag. Wenn du den behältst, kannst du das Thema herleiten.*

```
ceph -s
```

Eselsbrücke: Ein Befehl zum Tippen, mit Merkhilfe darunter.

> **STOLPERFALLE**
>
> Der Fehler, den fast alle machen. Und den Interviewer gern abfragen.

IM INTERVIEW

Eine typische Frage?

Eine Antwort in zwei, drei Sätzen. So kurz darf sie sein.

Dein Lernplan

Du hast …	Dann lies …
1 Abend	Kapitel 1, 3, 6, 7, 8 und 9, dann Kapitel 36 (die Fragen) und 38 (Spickzettel).
2 Tage	Tag 1: Teil I und II. Tag 2: Kapitel 14 bis 16, Teil V, dann Kapitel 36.
1 Woche	Tag 1: Teil I und II. Dann ein Teil pro Tag. Tag 6: die Design-Aufgaben aus Kapitel 37. Tag 7: Fragen wiederholen.

TEIL I

Der Start

Was Ceph ist, woraus es besteht und wie du ein Labor aufbaust.

1

Was ist Ceph?

Viele normale Server werden zu einem großen, selbstheilenden Speicher.

Ceph ist ein softwaredefiniertes, verteiltes Speichersystem. Sein Kern ist eine einzige Idee:

Ort eines Objekts = Funktion(Name, Cluster-Karte)

Niemand schlägt nach, wo Daten liegen. Jeder Teilnehmer rechnet es selbst aus.

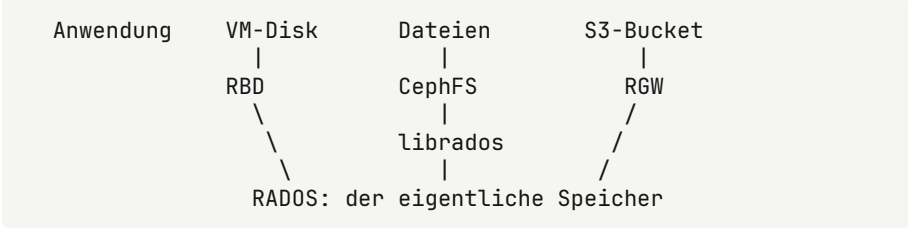
Das Problem

Klassischer Speicher hängt an einem Controller oder einer Metadaten-Tabelle. Wächst das System, wird diese eine Stelle zum Engpass. Fällt sie aus, steht alles.

Ceph streicht diese Stelle. Clients sprechen direkt mit den Festplatten-Daemons. Der Ort jedes Objekts folgt aus einer Formel.

> **DAS BILD IM KOPF**
>
> *Eine Bibliothek ohne Auskunft. Jedes Buch steht an dem Platz, den du aus seinem Titel berechnen kannst. Du brauchst nur den aktuellen Gebäudeplan.*

Der Schichtenkuchen



- **RADOS** (Reliable Autonomic Distributed Object Store) ist das Fundament. Es speichert Objekte, repliziert sie und heilt sich selbst.
- **RBD** liefert Block-Geräte.

- **CephFS** liefert ein POSIX-Dateisystem.
- **RGW** liefert eine S3- und Swift-kompatible API.

Alle drei legen am Ende nur RADOS-Objekte ab.

Die drei Versprechen

- **Kein Single Point of Failure.** Jede Rolle läuft mehrfach.
- **Skaliert horizontal.** Mehr Server bringen mehr Kapazität und mehr Leistung.
- **Selbstheilend.** Fällt eine Platte aus, stellt Ceph die Kopien von allein wieder her.

Ceph entscheidet sich bei einem Konflikt für **Konsistenz**. Lieber blockiert es Schreibzugriffe, als widersprüchliche Daten zuzulassen.

> **STOLPERFALLE**
>
> Ceph ist kein Backup. Replikation kopiert auch ein versehentliches Löschen sofort auf alle Kopien. Backup ist ein eigenes Thema, siehe Kapitel 34.

Die Releases

Jedes Release trägt den Namen eines Kopffüßers, und die Anfangsbuchstaben laufen durchs Alphabet.

Release	Version	Status im November 2026
Squid	19.2	Seit 31.10.2026 am Ende des Supports.
Tentacle	20.2	Aktiv. Darauf baut dieses Buch auf.
Umbrella	21.2	Aktiv seit 28.10.2026.

IM INTERVIEW

Was ist Ceph?

Ein softwaredefiniertes, verteiltes Speichersystem ohne zentralen Engpass. Das Fundament ist RADOS, ein selbstheilender Objektspeicher. Darauf liegen drei Schnittstellen: RBD für Block, CephFS für Dateien und RGW für S3.

IM INTERVIEW

Warum gibt es bei Ceph keinen zentralen Metadaten-Server für Objekte?

Weil Clients den Ort mit CRUSH selbst berechnen. Sie brauchen nur die aktuellen Karten vom Monitor. Dadurch gibt es keinen Lookup im Datenpfad, und die Leistung wächst mit dem Cluster.

DAS NIMMST DU MIT

- RADOS ist das Fundament, RBD, CephFS und RGW sind Schnittstellen.
- Rechnen statt Nachschlagen.
- Ceph wählt Konsistenz vor Verfügbarkeit.

2

Block, Datei, Objekt

Drei Arten, Daten anzusprechen. Ceph kann alle drei.

	Block	Datei	Objekt
Du siehst	Eine leere Festplatte	Ordner und Dateien	Buckets und Keys
Zugriff	Sektoren lesen und schreiben	POSIX: `open`, `read`, `rename`	HTTP: `PUT`, `GET`, `DELETE`
Teilen	Meist ein Client	Viele Clients	Beliebig viele Clients
In Ceph	RBD	CephFS	RGW
Typisch	VM-Disks, Datenbanken	Gemeinsame Ablagen, Home-Ordner	Backups, Medien, Data Lakes

DAS BILD IM KOPF

Block ist ein leeres Grundstück: Du baust darauf, was du willst. Datei ist ein fertiges Mietshaus mit Hausordnung. Objekt ist ein Paketschalter: Du gibst ein Paket ab und bekommst es über die Nummer zurück.

Wann was?

- Datenbank oder VM braucht eine schnelle eigene Platte: **RBD**.
- Mehrere Pods müssen dieselben Dateien lesen und schreiben: **CephFS**.
- Eine Anwendung speichert große Dateien über HTTP: **RGW**.

STOLPERFALLE

Zwei Server, die dasselbe RBD-Image mit einem normalen Dateisystem wie ext4 einhängen, zerstören es. ext4 weiß nichts von anderen Schreibern. Für gemeinsamen Zugriff nimmst du CephFS.

IM INTERVIEW

Wann nimmst du RBD, wann CephFS, wann RGW?

RBD für einen einzelnen Konsumenten mit Blockzugriff, also VMs und Datenbanken. CephFS, wenn viele Clients gleichzeitig auf dieselben Dateien zugreifen. RGW für HTTP-Anwendungen, große Datenmengen und alles, was S3 spricht.

DAS NIMMST DU MIT

- Block: Platte. Datei: Ordner. Objekt: Key über HTTP.
- RBD ist im Normalfall für einen Schreiber gedacht.

3

Die Daemons

Fünf Prozesse, jeder mit genau einer Aufgabe.

Daemon	Aufgabe	Wie viele?
MON	Hält die Cluster-Karten und bildet das Quorum.	3 oder 5, ungerade
MGR	Module: Dashboard, Metriken, Balancer, cephadm.	Mindestens 2
OSD	Speichert Daten, repliziert, heilt. Einer pro Platte.	Mindestens 3
MDS	Metadaten für CephFS. Nur dafür nötig.	Mindestens 2
RGW	HTTP-Gateway für S3. Nur dafür nötig.	Mindestens 2

> **DAS BILD IM KOPF**
>
> *Der MON ist das Grundbuchamt: Er kennt die Karten, trägt aber keine Pakete. Die OSDs sind die Lagerarbeiter. Der MGR ist die Verwaltung, die Berichte schreibt. Der MDS ist der Katalog der Bibliothek. Der RGW ist der Paketschalter.*

MON: die Wahrheit über den Cluster

MONs halten die **Cluster Map**, die aus mehreren Karten besteht: MON-Map, OSD-Map, CRUSH-Map, MDS-Map und mehr. Sie stimmen sich mit dem Paxos-Algorithmus ab. Änderungen brauchen eine Mehrheit, das **Quorum**.

MONs	Mehrheit	Verträgt Ausfälle
1	1	0
3	2	1
4	3	1
5	3	2

Vier MONs vertragen also nicht mehr als drei, haben aber eine Fehlerquelle mehr. Darum nimmst du eine ungerade Zahl.

OSD: die Arbeitstiere

Ein **Object Storage Daemon** verwaltet in der Regel eine Platte. OSDs speichern Daten, replizieren sie zueinander, beobachten sich gegenseitig per Heartbeat und melden Ausfälle an die MONs.

MGR: alles außer Daten

Fallen alle MGRs aus, lesen und schreiben Clients weiter. Es fehlen aber Dashboard, Metriken, Orchestrierung und aktuelle PG-Statistiken. Der Datenpfad braucht den MGR nicht.

> **STOLPERFALLE**
>
> MONs liegen nicht im Datenpfad, aber ohne Quorum werden keine Karten mehr geändert. Bestehende Clients arbeiten oft noch weiter, bis sich etwas ändert, zum Beispiel ein OSD ausfällt. Neue Verbindungen klappen nicht. Ein Ausfall des Quorums ist also ein Notfall, auch wenn noch I/O läuft.

IM INTERVIEW

Warum eine ungerade Anzahl MONs?

Das Quorum braucht eine Mehrheit. Drei MONs und vier MONs vertragen beide nur einen Ausfall. Der vierte erhöht nur den Aufwand. Bei einer Teilung des Netzes verhindert die Mehrheit ein Split-Brain.

IM INTERVIEW

Was passiert, wenn alle MGRs ausfallen?

Die Daten fließen weiter, denn Clients sprechen direkt mit den OSDs. Es fehlen Metriken, Dashboard, Balancer und cephadm. Darum läuft immer ein Standby mit.

DAS NIMMST DU MIT

- MON: Karten und Quorum. OSD: Daten. MGR: Module.
- MDS nur für CephFS, RGW nur für S3.
- Ungerade Zahl von MONs, wegen der Mehrheit.

4

Das Labor

Drei VMs, ein Bootstrap, und du hast einen Cluster.

Was du brauchst

Drei VMs mit Ubuntu 24.04, je 8 GB RAM und zusätzlich einer leeren Platte von 20 GB. Wir nennen sie `ceph01`, `ceph02` und `ceph03`. Alle drei brauchen feste IP-Adressen, eine synchrone Uhr (`chrony`) und Podman oder Docker.

> **MERKE**
>
> Ubuntu 24.04 bringt Kernel 6.8 mit. Für `rbd map` und `mount -t ceph` brauchst du aber den HWE-Kernel 7.0, weil Ceph ab 20.2.4 standardmäßig AES256K-Schlüssel erzeugt, siehe Kapitel 35.

> **STOLPERFALLE**
>
> Drei VMs auf einem Hypervisor sind ein Labor, kein HA-System. Ceph glaubt an drei Hosts, in Wahrheit teilen sie sich Strom und Hardware.

cephadm installieren

Ubuntu liefert ein älteres Paket. Für Tentacle kommt cephadm aus dem Ceph-Repository (`download.ceph.com`).

```
apt install -y cephadm ceph-common
```

Eselsbrücke: `cephadm` baut den Cluster, `ceph-common` bringt den Befehl `ceph` auf den Host.

Der Bootstrap

```
cephadm bootstrap --mon-ip 10.0.0.11
```

Eselsbrücke: **Bootstrap** wie der Stiefelriemen: Der Cluster zieht sich selbst hoch. Es entsteht ein MON und ein MGR auf diesem Host.

Danach läuft das Dashboard, und `/etc/ceph` enthält Konfiguration, Admin-Schlüssel und den SSH-Schlüssel des Clusters.

Die anderen Hosts holen

cephadm arbeitet per SSH. Also braucht jeder Host den öffentlichen Schlüssel:

```
ssh-copy-id -f -i /etc/ceph/ceph.pub root@ceph02
```

Eselsbrücke: Du gibst dem Hausmeister einen Schlüssel für Haus 2.

```
ceph orch host add ceph02 10.0.0.12
```

Eselsbrücke: **orch** wie **orch**estrator, der Dirigent: Nimm Host 2 in dein Orchester auf.

Das Gleiche machst du für `ceph03`.

Die Platten einbinden

```
ceph orch device ls
```

Eselsbrücke: Der Dirigent zeigt alle Platten. Nur `Available: Yes` ist nutzbar.

```
ceph orch apply osd --all-available-devices
```

Eselsbrücke: **apply** heißt: Das ist mein Wunsch, sorge dauerhaft dafür. Jede freie Platte wird ein OSD.

> **STOLPERFALLE**
>
> Dieser Befehl gilt dauerhaft: Auch später eingesteckte leere Platten werden zu OSDs. In Produktion beschreibst du Platten gezielt in einer Spec, siehe Kapitel 18.

Der erste Blick

```
ceph -s
```

Eselsbrücke: **s** wie **s**tatus: Wie geht es dir? Das ist der wichtigste Befehl des Buchs.

```
  cluster:
    id:     7c1f...
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum ceph01,ceph02,ceph03
    mgr: ceph01.abc(active), standbys: ceph02.def
    osd: 3 osds: 3 up, 3 in
```

```
  data:
    pools:   1 pools, 1 pgs
    usage:   80 MiB used, 60 GiB / 60 GiB avail
    pgs:     1 active+clean
```

Lies die Ausgabe immer von oben nach unten: Gesundheit, Dienste, Daten.

```
cephadm shell
```

Eselsbrücke: Eine Shell im Ceph-Container, in der alle Werkzeuge liegen. Nützlich auf Hosts ohne `ceph-common`.

IM INTERVIEW

Was macht `cephadm bootstrap`?

Es startet auf dem ersten Host einen MON und einen MGR als Container, erzeugt Schlüssel und Konfiguration und richtet Dashboard und Orchestrator ein. Alle weiteren Hosts und Daemons kommen danach über `ceph orch` hinzu.

DAS NIMMST DU MIT

- `cephadm bootstrap`, `ceph orch host add`, `ceph orch apply osd`.
- `ceph -s` ist dein erster Griff.
- Eine synchrone Uhr und feste IPs sind Pflicht.

TEIL II

Ceph denken

RADOS, CRUSH, Placement Groups, Replikation. Das Fundament jedes Interviews.

5

RADOS und Objekte

Alles in Ceph wird zu Objekten in Pools.

Vier Begriffe tragen alles:

- **Objekt:** ein Datenblock mit Namen, Inhalt und Metadaten. Meist bis 4 MiB groß.
- **Pool:** ein logischer Behälter mit Regeln: wie viele Kopien, auf welchen Platten, wer darf.
- **Placement Group (PG):** eine Gruppe von Objekten, die gemeinsam umzieht.
- **OSD:** der Daemon, der am Ende die Daten auf der Platte speichert.

```
Objekt  -->  PG  -->  OSDs
       Hash     CRUSH
```

Das ist der wichtigste Pfeil des Buchs. Merke ihn dir genau so.

Selbst ausprobieren

Lege einen Pool an und schreibe direkt in RADOS, ganz ohne RBD oder S3.

```
ceph osd pool create spielwiese
```

Eselsbrücke: Lege im OSD-Bereich einen Pool an. Name genügt, den Rest wählt Ceph.

```
echo hallo > gruss.txt
```

Eselsbrücke: Eine kleine Datei zum Speichern.

```
rados -p spielwiese put gruss gruss.txt
```

Eselsbrücke: `-p` wie **p**ool. `put` legt ab: Name zuerst, dann die Datei.

```
rados -p spielwiese ls
```

Eselsbrücke: Wie `ls` im Ordner, nur für Objekte.

```
ceph osd map spielwiese gruss
```

Eselsbrücke: Frage die Landkarte: Wo liegt dieses Objekt?

```
osdmap e42 pool 'spielwiese' (2) object 'gruss' -> pg 2.5a8b1c3f
(2.1b)
  -> up ([2,0,1], p2) acting ([2,0,1], p2)
```

So liest du das: Das Objekt liegt in einer PG von Pool 2, auf den OSDs 2, 0 und 1. OSD 2 ist der **Primary**. Beachte: Der Befehl fragt niemanden. Er rechnet nur.

> **DAS BILD IM KOPF**
>
> *Objekt ist der Brief, PG ist der Postleitzahlbezirk, OSD ist der Zusteller. Zieht ein Zusteller um, wechselt man nicht jeden Brief einzeln, sondern den Bezirk.*

IM INTERVIEW

Wie wird aus einer Datei ein RADOS-Objekt?

Die Schnittstelle zerlegt die Daten. RBD und CephFS teilen in Stücke von standardmäßig 4 MiB, RGW teilt große Uploads ebenfalls auf. Jedes Stück wird ein Objekt in einem Pool.

DAS NIMMST DU MIT

- Objekt, PG, OSD: Hash, dann CRUSH.
- `ceph osd map` zeigt die Rechnung für ein Objekt.
- Der Primary ist der erste OSD in der Liste.

6

CRUSH

Die Formel, die aus einem Namen einen Ort macht.

CRUSH (Controlled Replication Under Scalable Hashing) berechnet, welche OSDs eine PG speichern. Zutaten sind die **CRUSH-Map** (dein Rechenzentrum als Baum) und eine **Regel** (wie verteilt werden soll).

Der Baum

```
ceph osd crush tree
```

Eselsbrücke: Zeig mir den Baum: root, Hosts, OSDs.

```
ID  CLASS  WEIGHT   TYPE NAME
-1         0.05846  root default
-3         0.01949      host ceph01
 0    hdd  0.01949          osd.0
-5         0.01949      host ceph02
 1    hdd  0.01949          osd.1
-7         0.01949      host ceph03
 2    hdd  0.01949          osd.2
```

Das `WEIGHT` entspricht normalerweise der Größe in TiB. Größere Platten bekommen mehr Daten.

Failure Domain

Die **Failure Domain** legt fest, was keine zwei Kopien teilen dürfen. Standard ist `host`: Drei Kopien liegen auf drei verschiedenen Hosts. Du kannst den Baum auch um `rack`, `room` oder `datacenter` erweitern.

> **DAS BILD IM KOPF**
>
> *Drei Kopien deines Schlüssels: Du legst sie nicht in dieselbe Schublade, sondern in drei Häuser. Die Failure Domain sagt, was „ein Haus“ ist.*

Device Classes

OSDs bekommen automatisch eine Klasse: `hdd`, `ssd` oder `nvme`. Damit trennst du schnelle und langsame Daten, ohne den Baum umzubauen.

```
ceph osd crush rule create-replicated schnell default host ssd
```

Eselsbrücke: Lies von links: Regel „schnell“, ab Wurzel `default`, pro `host` eine Kopie, nur Klasse `ssd`.

```
ceph osd pool set spielwiese crush_rule schnell
```

Eselsbrücke: Gib dem Pool die neue Regel. Die Daten ziehen dann um.

Warum CRUSH so gut ist

- **Keine Tabelle.** Jeder Client rechnet selbst.
- **Wenig Bewegung.** Kommt ein OSD dazu, wandert nur etwa der Anteil, der ihm zusteht.
- **Pseudozufällig.** Die Verteilung ist gleichmäßig, aber für alle gleich berechenbar.

Der Balancer

Pseudozufall ist nie ganz gleichmäßig. Der MGR-Modus `upmap` trägt darum gezielte Ausnahmen in die OSD-Map ein und schiebt einzelne PGs um.

```
ceph balancer status
```

Eselsbrücke: Waage: Ist sie an, und in welchem Modus?

> **STOLPERFALLE**
>
> Mit `size 3` und Failure Domain `host` brauchst du mindestens drei Hosts. Hast du nur zwei, findet CRUSH keinen dritten Platz, und die PGs bleiben dauerhaft `undersized`.

IM INTERVIEW

Wie findet ein Client ein Objekt?

Er holt sich die aktuellen Karten vom MON. Dann hasht er den Objektnamen zu einer PG und berechnet mit CRUSH die OSDs dieser PG. Anschließend spricht er direkt mit dem Primary. Es gibt keinen Lookup.

IM INTERVIEW

Was ist eine Failure Domain?

Die Ebene im CRUSH-Baum, die zwei Kopien nicht teilen dürfen, etwa Host oder Rack. Sie legt fest, welchen Ausfall ein Pool übersteht.

DAS NIMMST DU MIT

- CRUSH-Map plus Regel ergibt die OSDs einer PG.
- Failure Domain: Was dürfen zwei Kopien nicht teilen?
- Device Classes trennen SSD und HDD.
- Der Balancer im Modus `upmap` glättet die Verteilung.

7

Placement Groups

Die Zwischenschicht, die Ceph handhabbar macht.

Warum nicht direkt Objekt auf OSD? Weil Milliarden Objekte einzeln nicht zu verwalten wären. Ceph verwaltet stattdessen wenige tausend PGs. Replikation, Peering und Recovery laufen pro PG.

Wie viele PGs?

Zu wenige PGs verteilen Daten ungleich und machen eine PG zu groß. Zu viele kosten RAM und CPU. Als Faustregel gilt ein Ziel pro OSD, gesteuert durch `mon_target_pg_per_osd`. Der Standard ist 100, die Tentacle-Doku empfiehlt für die meisten Deployments 200.

Die alte Formel kommt oft im Interview vor:

PGs im Pool ≈ OSDs × Ziel ÷ size, dann auf die nächste Zweierpotenz

Beispiel: 100 OSDs, Ziel 100, `size 3`, ergibt etwa 3333, also 4096.

Der Autoscaler

Heute rechnest du das selten selbst. Das MGR-Modul `pg_autoscaler` passt `pg_num` an.

```
ceph osd pool autoscale-status
```

Eselsbrücke: Zeigt pro Pool: Wie viele PGs habe ich, wie viele wären richtig?

Modus	Wirkung
`on`	Passt `pg_num` selbst an.
`warn`	Warnt nur.
`off`	Macht nichts.

> **MERKE**
>
> Hilf dem Autoscaler: Mit `target_size_ratio` sagst du ihm, welchen Anteil ein Pool am Ende belegen wird. Dann muss er nicht während des Befüllens teuer umsortieren.

Up Set und Acting Set

- **Up Set:** die OSDs, die laut CRUSH zuständig sein sollten.
- **Acting Set:** die OSDs, die die PG gerade wirklich bedienen.

Meist sind beide gleich. Sie weichen ab, solange Daten noch umziehen.

Peering

Bevor eine PG arbeitet, einigen sich ihre OSDs auf den Stand der Dinge. Das heißt **Peering**. Jeder OSD führt ein **PG Log** mit den letzten Änderungen. Beim Peering vergleichen die OSDs diese Logs und stellen fest, wer was nachholen muss.

> **DAS BILD IM KOPF**
>
> *Drei Protokollführer treffen sich nach einer Pause. Sie legen ihre Protokolle nebeneinander und einigen sich, was zuletzt beschlossen wurde. Erst dann geht die Sitzung weiter.*

IM INTERVIEW

Warum gibt es Placement Groups?

Sie entkoppeln Objekte von OSDs. Ceph müsste sonst jedes Objekt einzeln verfolgen. Mit PGs laufen Replikation, Peering und Recovery für Gruppen von Objekten. Das skaliert viel besser.

IM INTERVIEW

Was passiert, wenn du `pg_num` erhöhst?

PGs teilen sich. Die Objekte werden neu auf die Kinder-PGs verteilt, und Ceph verschiebt Daten schrittweise auf OSDs. Das erzeugt Last. Darum läuft es gestaffelt und im Betrieb möglichst geplant.

DAS NIMMST DU MIT

- PGs bündeln Objekte für Replikation und Recovery.
- Autoscaler an, `target_size_ratio` setzen.
- Up Set: soll. Acting Set: ist.
- Peering: Einigung über das PG Log.

8

PG-Zustände

Wer diese Wörter lesen kann, kann Ceph lesen.

Der gesunde Zustand heißt `active+clean`. Alles andere ist eine Geschichte. Zustände kombinieren sich, zum Beispiel `active+undersized+degraded`.

Zustand	Bedeutung	I/O?
`active`	Die PG bedient Anfragen.	Ja
`clean`	Alle Kopien sind da und aktuell.	Ja
`peering`	Die OSDs einigen sich gerade.	Kurz nein
`degraded`	Einige Objekte haben zu wenig Kopien.	Ja
`undersized`	Weniger OSDs im Acting Set als `size`.	Ja
`recovering`	Fehlende Objekte werden nachgezogen.	Ja
`backfilling`	Eine ganze PG wird kopiert.	Ja
`remapped`	Die PG läuft vorübergehend woanders.	Ja
`peered`	Peering ist fertig, aber es gibt zu wenig Kopien für `min_size`.	Nein
`down`	Ein OSD mit nötigen Daten fehlt.	Nein
`stale`	Der Primary meldet sich nicht mehr beim MON.	Nein
`incomplete`	Es fehlt Historie. Ernster Fall.	Nein
`inconsistent`	Scrub hat abweichende Kopien gefunden.	Ja

> **MERKE**
>
> `inactive` ist kein eigener PG-Zustand. Das Wort ist der Sammelbegriff für PGs, die nicht `active` sind, etwa in `ceph pg dump_stuck inactive`. Du siehst dann Zustände wie `peered` oder `down`.

Recovery oder Backfill?

- **Recovery** nutzt das PG Log. Ein OSD war kurz weg und holt nur die Änderungen nach.
- **Backfill** kopiert die ganze PG. Das passiert, wenn das Log nicht reicht oder ein OSD neu ist.

DAS BILD IM KOPF

Recovery: Du warst zwei Tage krank und liest die verpassten Mails. Backfill: Du bist neu in der Firma und bekommst das komplette Archiv.

Was bei einem Ausfall passiert

```
Zeit 0      OSD stirbt
ca. 20 s    Nachbarn melden: down
            PGs: active+undersized+degraded
10 min      Ceph setzt den OSD auf out
            PGs: backfilling auf andere OSDs
danach      PGs: active+clean
```

`down` beschreibt den Prozess, `out` die Datenverteilung. Das sind zwei getrennte Schalter. Die Wartezeit steuert `mon_osd_down_out_interval`, Standard sind 600 Sekunden.

Zustand	Bedeutung
`up` + `in`	Normalbetrieb.
`down` + `in`	Kurzer Ausfall. Ceph wartet, die Daten bleiben zugeordnet.
`down` + `out`	Ceph gibt den OSD auf und verteilt seine PGs neu.
`up` + `out`	Läuft, bekommt aber keine Daten. Typisch beim gezielten Leeren.

STOLPERFALLE

`degraded` heißt nicht kaputt. Alle Daten sind lesbar, die PG arbeitet, nur die Redundanz fehlt. Gefährlich wird es erst bei `inactive` oder `incomplete`.

IM INTERVIEW

Was ist der Unterschied zwischen `down` und `out`?

`down` heißt, der Prozess antwortet nicht. `out` heißt, CRUSH verteilt seine Daten neu. Ein OSD wird erst `down` und nach zehn Minuten automatisch `out`. So löst ein kurzer Neustart keinen großen Datenumzug aus.

IM INTERVIEW

Was ist der Unterschied zwischen Recovery und Backfill?

Recovery holt anhand des PG Logs nur fehlende Änderungen nach. Backfill scannt und kopiert die ganze PG, etwa auf einen neuen OSD. Backfill dauert länger und erzeugt mehr Last.

DAS NIMMST DU MIT

- Ziel: `active+clean`.
- `degraded`: Redundanz fehlt, I/O läuft. `inactive`: I/O steht.
- `down` ist der Prozess, `out` die Datenverteilung.
- Recovery liest das Log, Backfill kopiert alles.

9

Replikation

Drei Kopien, und eine klare Regel, wann geschrieben werden darf.

`size` und `min_size`

- `size`: So viele Kopien will ich haben.
- `min_size`: So viele müssen da sein, damit die PG I/O annimmt.

Standard und Empfehlung ist `size 3`, `min_size 2`.

Verfügbare Kopien	Zustand bei `size 3`, `min_size 2`
3	`active+clean`
2	`active+undersized+degraded`. I/O läuft.
1	`undersized+degraded+peered`, also nicht `active`. I/O blockiert.
0	`down` oder `stale`. Keine Daten erreichbar.

```
ceph osd pool get spielwiese size
```

Eselsbrücke: Lies die Einstellung `size`.

```
ceph osd pool set spielwiese min_size 2
```

Eselsbrücke: Setze die Untergrenze für I/O.

Der Schreibweg

```
Client --> Primary --> zwei Replikate
Client <-- Primary <-- beide haben bestätigt
```

1. Der Client schreibt zum Primary der PG.
2. Der Primary schreibt lokal und schickt parallel an die Replikate.
3. Erst wenn alle aus dem Acting Set bestätigt haben, bekommt der Client sein OK.

Eine Bestätigung bedeutet also: Die Daten liegen auf allen aktiven Kopien. Gelesen wird standardmäßig vom Primary. Damit ist die Replikation synchron und **stark konsistent**.

> **DAS BILD IM KOPF**
>
> *Ein Notar mit zwei Zeugen. Du bekommst deine Quittung erst, wenn alle unterschrieben haben.*

> **STOLPERFALLE**
>
> `size 2`, `min_size 1` ist die berühmteste Fehlkonfiguration. Fällt eine Kopie aus, landen neue Schreibzugriffe nur noch auf einer einzigen Platte. Stirbt die auch, bevor die andere aufgeholt hat, sind bestätigte Daten weg.

IM INTERVIEW

Warum ist `min_size 1` gefährlich?

Der Cluster bestätigt dann Schreibzugriffe, die nur auf einer Kopie liegen. Fällt diese aus, sind bestätigte Daten verloren, und die zurückkehrende Kopie hat einen älteren Stand. `min_size 2` opfert bewusst Verfügbarkeit für Dateninteg rität.

IM INTERVIEW

Wann bekommt der Client das Write-ACK?

Wenn alle OSDs im Acting Set den Schreibvorgang dauerhaft gespeichert haben. Die Latenz hängt also am langsamsten Replikat.

DAS NIMMST DU MIT

- `size 3`, `min_size 2` ist die Regel.
- ACK erst nach allen Kopien im Acting Set.
- Der langsamste OSD bestimmt die Schreiblatenz.

10

Erasure Coding

Weniger Speicher für die gleiche Sicherheit. Bezahlt mit Rechenzeit.

Erasure Coding (EC) zerlegt ein Objekt in `k` Datenstücke und berechnet `m` Paritätsstücke. Solange `k` Stücke übrig sind, lässt sich das Objekt wiederherstellen.

Schema	Verträgt Ausfälle	Rohdaten für 1 TB	Nutzbar
Replikat, `size 3`	2	3 TB	33 %
`k=2, m=2`	2	2 TB	50 %
`k=4, m=2`	2	1,5 TB	67 %
`k=8, m=3`	3	1,375 TB	73 %

> **DAS BILD IM KOPF**
>
> *Ein RAID 6 über Server hinweg. Statt drei ganzer Kopien hast du Puzzleteile plus ein paar Reserveteile, aus denen sich fehlende Teile berechnen lassen.*

Einen EC-Pool anlegen

```
ceph osd erasure-code-profile set ec42 k=4 m=2 crush-failure-domain=host
```

Eselsbrücke: Ein Rezept mit Namen: 4 Daten, 2 Parität, jedes Stück auf einem anderen Host.

```
ceph osd pool create archiv erasure ec42
```

Eselsbrücke: Pool `archiv`, Art `erasure`, nach Rezept `ec42`.

```
ceph osd pool set archiv allow_ec_overwrites true
```

Eselsbrücke: Erlaubt Überschreiben im EC-Pool. Das brauchen RBD und CephFS.

Der Preis

- **Mehr Hosts.** `k=4, m=2` braucht mindestens sechs Failure Domains. Mit Reserve zum Heilen sieben.
- **Mehr CPU.** Parität muss berechnet werden.
- **Teure kleine Schreibzugriffe.** Ändern heißt lesen, neu rechnen, schreiben (Read-Modify-Write).
- **Teure Recovery.** Für ein fehlendes Stück liest Ceph `k` andere.

`min_size` bei EC

Die Doku empfiehlt mindestens `k+1`. Bei `k=4, m=2` sind das 5. So bleibt beim Schreiben immer mindestens ein Stück Reserve.

RBD auf EC

Die Metadaten eines RBD-Images passen nicht in einen EC-Pool. Darum liegen sie in einem replizierten Pool, und nur die Daten im EC-Pool:

```
rbd create schnell/disk1 --size 100G --data-pool archiv
```

Eselsbrücke: Kopf im replizierten Pool `schnell`, Bauch im EC-Pool `archiv`.

> **MERKE**
>
> Tentacle bringt mit dem Pool-Flag `allow_ec_optimizations` Verbesserungen für kleine I/Os. Das Flag lässt sich nicht wieder abschalten und setzt voraus, dass alle MONs und OSDs mindestens Tentacle laufen.

> **STOLPERFALLE**
>
> Das EC-Profil eines bestehenden Pools kannst du nicht ändern. `k` und `m` sind eine Entscheidung fürs Leben des Pools. Für einen Wechsel legst du einen neuen Pool an und ziehst die Daten um.

IM INTERVIEW

Replikation oder Erasure Coding?

Replikation für Latenz: RBD, Datenbanken, Metadaten, RGW-Index. EC für große, eher sequenzielle Daten, also S3-Objekte und Archive. EC spart Speicher, kostet aber CPU, mehr Hosts und langsamere Recovery.

IM INTERVIEW

Warum braucht `k=4, m=2` eigentlich sieben Hosts?

Sechs Hosts tragen die sechs Stücke. Fällt einer aus, gibt es keinen freien Host zum Heilen, und die PGs bleiben degradiert. Der siebte schafft den Platz für die Selbstheilung.

DAS NIMMST DU MIT

- `k` Daten, `m` Parität, `m` Ausfälle sind verkraftbar.
- `min_size` mindestens `k+1`.
- Metadaten immer repliziert.
- `k` und `m` sind nachträglich nicht änderbar.

11

BlueStore

Wie ein OSD die Platte direkt beschreibt.

BlueStore ist das Speicher-Backend der OSDs. Es schreibt Daten direkt auf das rohe Blockgerät, ohne Dateisystem dazwischen. Metadaten liegen in **RocksDB**, einer eingebetteten Key-Value-Datenbank.



Gerät	Inhalt	Typisch auf
`block`	Die Daten selbst.	HDD oder SSD
`block.db`	RocksDB-Metadaten.	Schnelle SSD oder NVMe
`block.wal`	Write-Ahead Log von RocksDB.	Am schnellsten, oft mit `block.db` zusammen

Was BlueStore mitbringt

- **Prüfsummen.** Jeder Lesevorgang prüft die Daten. Stille Fehler fallen auf.
- **Kompression** inline, pro Pool wählbar.
- **Copy-on-Write** für schnelle Snapshots und Clones.
- **Eigener Cache** im RAM. Die Grenze setzt `osd_memory_target`, standardmäßig 4 GiB pro OSD.

> **DAS BILD IM KOPF**
>
> *`block` ist das Lager. `block.db` ist die Kartei, die sagt, was wo liegt. Wer bei HDD-Lagern die Kartei auf eine SSD legt, findet alles viel schneller.*

STOLPERFALLE

Ist `block.db` zu klein, läuft RocksDB auf die langsame Platte über. Ceph meldet dann `BLUEFS_SPILLOVER`. Fällt außerdem eine gemeinsame DB-SSD aus, verlierst du alle OSDs, die daran hängen. Plane das als Failure Domain ein.

IM INTERVIEW

Warum hat BlueStore FileStore abgelöst?

FileStore legte Objekte als Dateien in XFS ab und schrieb alles doppelt, erst ins Journal, dann ins Dateisystem. BlueStore schreibt direkt aufs Gerät, speichert Metadaten in RocksDB und bringt Prüfsummen und Kompression mit. Das ist schneller und sicherer.

DAS NIMMST DU MIT

- BlueStore: rohe Platte plus RocksDB.
- `block`, `block.db`, `block.wal`.
- HDD-OSDs mit DB auf SSD. Plane die SSD als gemeinsamen Ausfallpunkt ein.
- `osd_memory_target` steuert den Cache.

12

Das Netzwerk

Bei Ceph ist das Netz die Backplane.

Zwei Netze

- **Public Network:** Clients sprechen mit MONs und OSDs.
- **Cluster Network:** OSDs replizieren untereinander. Optional.

```
Clients ---- public ---- [OSD] ---- cluster ---- [OSD]
```

Bei `size 3` erzeugt jeder Schreibvorgang des Clients zwei Replikat-Schreibvorgänge. Das interne Netz trägt also mehr Last als das öffentliche. Dazu kommt Recovery.

> **STOLPERFALLE**
>
> Fällt nur das Cluster Network aus, das Public Network aber nicht, sehen MONs die OSDs noch, während OSDs ihre Nachbarn verlieren. OSDs melden sich dann gegenseitig als `down` (Flapping). Viele Betreiber nehmen deshalb lieber ein schnelles gebündeltes Netz als zwei getrennte.

Ports

Dienst	Port	Protokoll
MON	3300	msgr2
MON	6789	msgr1 (alt)
OSD, MGR, MDS	6800–7300	msgr2 und msgr1

msgr2 ist das aktuelle Protokoll. Es kann auch verschlüsseln.

Wichtige Punkte

- **Jumbo Frames (MTU 9000)** nur durchgängig. Eine Stelle mit MTU 1500 führt zu seltsamen Hängern: Kleine Pakete gehen durch, große nicht.
- **Bonding** mit LACP für Bandbreite und Ausfallsicherheit.
- **Latenz** zählt so viel wie Bandbreite. Ceph wartet synchron auf alle Replikate.

- **Für den Test:** `ping -M do -s 8972 ceph02` prüft MTU 9000 ohne Fragmentierung.

IM INTERVIEW

Braucht Ceph ein getrenntes Cluster Network?

Nicht zwingend. Es trennt Replikation und Recovery vom Client-Verkehr, bringt aber eine zusätzliche Fehlerquelle. Mit genug Bandbreite reicht ein gemeinsames, redundantes Netz. Das ist heute oft die einfachere Wahl.

DAS NIMMST DU MIT

- Public: Clients. Cluster: Replikation.
- MON 3300 und 6789, OSDs 6800–7300.
- MTU überall gleich.
- Latenz zählt, nicht nur Bandbreite.

TEIL III

Die Schnittstellen

Pools, Rechte, RBD, CephFS, S3 und Kubernetes.

13

Pools und Rechte

Ein Pool ist die Einheit für Regeln. CephX entscheidet, wer hinein darf.

Pool-Grundbefehle

```
ceph osd pool ls detail
```

Eselsbrücke: `ls` listet, `detail` zeigt `size`, `crush_rule` und PGs.

```
ceph df
```

Eselsbrücke: Wie `df` auf Linux: Wie voll ist der Cluster, wie viel darf jeder Pool noch?

```
ceph osd pool application enable spielwiese rbd
```

Eselsbrücke: Sag dem Pool, wofür er da ist: `rbd`, `cephfs` oder `rgw`.

```
ceph osd pool set-quota spielwiese max_bytes 10737418240
```

Eselsbrücke: Eine Obergrenze für den Pool, hier 10 GiB in Bytes.

Pools löschen

Löschen ist sofort und unwiderruflich. Darum macht Ceph es absichtlich umständlich:

```
ceph config set mon mon_allow_pool_delete true
```

Eselsbrücke: Erst die Sicherung herausnehmen.

```
ceph osd pool rm spielwiese spielwiese --yes-i-really-really-mean-it
```

Eselsbrücke: Name zweimal, dazu das berühmte „really really“.

```
ceph config set mon mon_allow_pool_delete false
```

Eselsbrücke: Sicherung wieder hinein.

CephX

CephX ist die Anmeldung im Cluster. Jeder Client hat einen Namen, einen geheimen Schlüssel und **Capabilities**, also Rechte pro Daemon-Art. Beide Seiten kennen das Geheimnis. Es wird nie im Klartext übertragen.

```
ceph auth get-or-create client.k8s mon 'profile rbd' osd
'profile rbd pool=kubernetes' mgr 'profile rbd pool=kuber-
netes'
```

Eselsbrücke: Hole Schlüssel für Nutzer `k8s`, oder lege ihn an. Rechte werden pro Daemon vergeben.

```
ceph auth ls
```

Eselsbrücke: Liste alle Identitäten und ihre Rechte.

Profile wie `profile rbd` bündeln genau die Rechte, die ein Client braucht.

> **DAS BILD IM KOPF**
>
> *CephX ist wie ein Hotelzimmer-Schlüssel: Er öffnet genau dein Zimmer und den Fitnessraum. Die Rezeption gibt ihn aus, aber geht nicht bei jedem Öffnen mit.*

> **STOLPERFALLE**
>
> `client.admin` darf alles. Dieser Schlüssel gehört nie in eine Anwendung oder einen Kubernetes-Cluster. Jede Anwendung bekommt einen eigenen Nutzer mit den geringsten nötigen Rechten.

IM INTERVIEW

Wie funktioniert die Authentifizierung in Ceph?

Mit CephX, einem Ticket-Verfahren mit gemeinsamem Geheimnis. Der Client meldet sich beim MON und bekommt ein Ticket. Damit spricht er direkt mit den OSDs. Die Rechte hängen als Capabilities pro Daemon-Art am Nutzer.

DAS NIMMST DU MIT

- `ceph df` zeigt, wie viel noch geht.
- Pool löschen braucht Name zweimal, Flag und Option.
- Jede Anwendung bekommt einen eigenen CephX-Nutzer.

14

RBD: Block Storage

Eine virtuelle Festplatte, verteilt auf den ganzen Cluster.

Ein **RBD-Image** ist eine dünn provisionierte Festplatte. Ceph zerlegt sie in Objekte von 4 MiB. Speicher wird erst belegt, wenn du schreibst.

Ein Image anlegen und nutzen

```
ceph osd pool create kubernetes
```

Eselsbrücke: Der Pool für unsere Disks.

```
rbd pool init kubernetes
```

Eselsbrücke: Macht den Pool bereit für RBD.

```
rbd create kubernetes/disk1 --size 10G
```

Eselsbrücke: Immer `pool/image`. Wie ein Pfad.

```
rbd info kubernetes/disk1
```

Eselsbrücke: Größe, Objektgröße und aktive Features des Images.

```
rbd map kubernetes/disk1
```

Eselsbrücke: Häng die Disk an diesen Linux-Kernel. Es entsteht `/dev/rbd0`.

```
mkfs.ext4 /dev/rbd0
```

Eselsbrücke: Jetzt ist es eine ganz normale Platte.

```
mount /dev/rbd0 /mnt
```

Eselsbrücke: Einhängen wie immer.

Zwei Wege zum Client

- **krbd:** das Kernel-Modul. Du bekommst `/dev/rbd0`. Einfach und schnell.
- **librbd:** die Bibliothek im Userspace. Das nutzen zum Beispiel QEMU und KVM.

> **STOLPERFALLE**
>
> krbd unterstützt oft nicht jedes Feature. Dann schlägt `rbd map` mit einer Meldung über Image-Features fehl. Prüfe `rbd info` und passe die Features an oder nutze einen neueren Kernel.

Snapshots und Clones

```
rbd snap create kubernetes/disk1@vor-update
```

Eselsbrücke: Das `@` markiert einen Zeitpunkt, wie ein Lesezeichen.

```
rbd snap ls kubernetes/disk1
```

Eselsbrücke: Zeigt alle Lesezeichen.

```
rbd snap protect kubernetes/disk1@vor-update
```

Eselsbrücke: Schreibschutz für den Snapshot. Pflicht bei Clone v1. Clone v2 braucht ihn nicht, setzt aber `require_min_compat_client` auf mindestens `mimic` voraus.

```
rbd clone kubernetes/disk1@vor-update kubernetes/disk2
```

Eselsbrücke: Eine neue Disk, die sich den Inhalt des Snapshots teilt. Das geht in Sekunden.

```
rbd flatten kubernetes/disk2
```

Eselsbrücke: Der Clone wird selbstständig. Er kopiert alles und löst sich vom Snapshot.

Snapshots nutzen Copy-on-Write: Erst bei einer Änderung entsteht zusätzlicher Speicher.

> **MERKE**
>
> Ein Snapshot eines eingehängten Dateisystems ist nur **crash-konsistent**. Für einen sauberen Stand friere vorher das Dateisystem mit `fsfreeze` ein und taue es danach wieder auf.

Vergrößern

```
rbd resize kubernetes/disk1 --size 20G
```

Eselsbrücke: Die Disk wächst sofort.

```
resize2fs /dev/rbd0
```

Eselsbrücke: Das Dateisystem wächst nicht von selbst. Das musst du extra sagen.

Features, die du kennen solltest

Feature	Wozu
`exclusive-lock`	Nur ein Client schreibt. Sonst viele andere Features nicht nutzbar.
`object-map`	Merkt sich, welche Objekte existieren. Beschleunigt `du`, `clone`, `export`.
`fast-diff`	Schnelle Differenzen zwischen Snapshots.
`journaling`	Für Journal-basiertes Mirroring. Kostet Leistung.

IM INTERVIEW

Wie funktioniert ein RBD-Snapshot?

Copy-on-Write auf RADOS-Ebene. Beim Snapshot wird nichts kopiert. Erst beim Überschreiben bewahrt Ceph den alten Stand des betroffenen Objekts. Darum sind Snapshots sofort da und anfangs klein.

IM INTERVIEW

Clone oder Snapshot?

Ein Snapshot ist ein schreibgeschützter Zeitpunkt. Ein Clone ist ein beschreibbares neues Image, das auf einem Snapshot aufsetzt. So startest du schnell viele VMs aus einem Basis-Image.

DAS NIMMST DU MIT

- Immer `pool/image`, Snapshots mit `@`.
- `rbd create`, `map`, `mkfs`, `mount`.
- krbd im Kernel, librbd für KVM.
- `resize` und danach das Dateisystem vergrößern.

15

CephFS

Ein gemeinsames POSIX-Dateisystem für viele Clients.

CephFS trennt strikt:

- **Daten** schreiben Clients direkt in den Daten-Pool, also zu den OSDs.
- **Metadaten** (Namen, Rechte, Verzeichnisse) laufen über den **MDS** und liegen in einem eigenen Pool.

Der MDS ist also nie im Datenpfad. Er ist ein Cache mit Journal.

Dateisystem anlegen

```
ceph fs volume create meinfs
```

Eselsbrücke: Ein Befehl für alles: zwei Pools, Dateisystem und MDS-Daemons.

```
ceph fs status
```

Eselsbrücke: Wer ist `active`, wer `standby`? Wie voll sind die Pools?

```
ceph fs authorize meinfs client.ada / rwps
```

Eselsbrücke: `rw` zum Lesen und Schreiben, `p` für Quotas und Layouts, `s` für Snapshots.

Einhängen

```
mount -t ceph ada@.meinfs=/ /mnt/cephfs
```

Das liest sich so: Nutzer `ada`, Cluster aus `ceph.conf` (nichts vor dem Punkt angegeben), Dateisystem `meinfs`, Pfad `/`.

Mehrere MDS

Standardmäßig gibt es einen aktiven MDS. Weitere stehen bereit (`standby`). Mit `max_mds` teilst du den Namensraum auf mehrere **Ranks** auf.

```
ceph fs set meinfs max_mds 2
```

Eselsbrücke: Zwei aktive MDS. Erhöhe das nur, wenn der Metadaten-Durchsatz wirklich fehlt.

```
ceph fs set meinfs allow_standby_replay true
```

Eselsbrücke: Der Standby liest das Journal mit. Failover wird dadurch kürzer.

Caps

Damit viele Clients sicher zusammenarbeiten, verteilt der MDS **Capabilities**: Rechte wie „du darfst diese Datei cachen“ oder „du darfst schreiben“. Will ein zweiter Client zugreifen, zieht der MDS die Rechte zurück.

> **DAS BILD IM KOPF**
>
> *Caps sind wie der Staffelstab: Nur wer ihn hält, darf laufen. Reicht ein Client ihn nicht weiter, stehen alle anderen und warten.*

Quotas, Snapshots, Subvolumes

```
setfattr -n ceph.quota.max_bytes -v 10000000000 /mnt/cephfs/
team
```

Eselsbrücke: Quotas sind Attribute am Verzeichnis, nicht am Nutzer.

```
mkdir /mnt/cephfs/team/.snap/montag
```

Eselsbrücke: Ein Snapshot entsteht durch ein `mkdir` im unsichtbaren Ordner `.snap`.

```
ceph fs subvolume create meinfs pvc-1
```

Eselsbrücke: Eine eigene kleine Welt mit Quota und Pfad. Das nutzt der Kubernetes-CSI-Treiber.

> **STOLPERFALLE**
>
> Mehr aktive MDS sind nicht automatisch besser. `max_mds` verteilt nur Metadaten-Arbeit und kostet Koordination. Hilft dir ein größerer MDS-Cache (`mds_cache_memory_limit`) oder Pinning mit `ceph.dir.pin`, ist das oft die bessere erste Wahl.

IM INTERVIEW

Warum ist CephFS gut skalierbar?

Datenzugriffe gehen direkt vom Client zu den OSDs. Der MDS kümmert sich nur um Metadaten. Bei Bedarf verteilen mehrere MDS-Ranks den Namensraum, dynamisch oder per Pinning.

IM INTERVIEW

Was sind Caps?

Rechte, die der MDS an Clients vergibt, etwa zum Cachen oder Puffern von Schreibzugriffen. Sie sorgen für Konsistenz über Clients hinweg. Gibt ein Client sie nicht zurück, blockiert er andere.

DAS NIMMST DU MIT

- Daten direkt zu den OSDs, Metadaten über den MDS.
- `ceph fs volume create` erzeugt alles Nötige.
- Standby-Replay verkürzt den Failover.
- Caps regeln den gemeinsamen Zugriff.

16

RGW: Object Storage

S3 auf deiner eigenen Hardware.

Der **RADOS Gateway** (RGW) ist ein HTTP-Server. Er übersetzt S3-Aufrufe in RADOS-Objekte. RGW-Instanzen sind zustandslos. Du startest mehrere hinter einem Load Balancer.

Starten

```
ceph orch apply rgw meins --placement="2 ceph01 ceph02" --
port=8080
```

Eselsbrücke: Zwei Gateways, auf diesen Hosts, auf Port 8080.

Nutzer und Bucket

```
radosgw-admin user create --uid=ada --display-name=Ada
```

Eselsbrücke: Eigener Admin-Befehl für RGW. Die Ausgabe enthält Access Key und Secret Key.

Jetzt funktioniert jedes S3-Werkzeug, zum Beispiel die AWS CLI:

```
aws --endpoint-url http://ceph01:8080 s3 mb s3://fotos
aws --endpoint-url http://ceph01:8080 s3 cp bild.jpg s3://fotos/
aws --endpoint-url http://ceph01:8080 s3 ls s3://fotos
```

Die Pools hinter RGW

Pool	Inhalt	Empfehlung
`.rgw.root`	Realm und Zonen.	Repliziert
`default.rgw.buckets.index`	Das Inhaltsverzeichnis der Buckets.	Repliziert, auf SSD
`default.rgw.buckets.data`	Die Objekte selbst.	EC möglich, auch HDD
`default.rgw.log`, `.meta`	Logs und Metadaten.	Repliziert

Bucket Index und Sharding

Jeder Bucket hat einen Index, der in RADOS-Objekten liegt. Ein Riesen-Bucket mit einem einzigen Index-Objekt wäre ein Engpass. Darum wird der Index in **Shards** geteilt. Das **Dynamic Resharding** erledigt das automatisch.

```
radosgw-admin bucket limit check
```

Eselsbrücke: Sind Buckets zu voll für ihre Shards?

> **DAS BILD IM KOPF**
>
> *Der Index ist das Inhaltsverzeichnis. Bei einer Million Einträgen teilst du es in mehrere Bände auf, sonst streiten sich alle Leser um dasselbe Buch.*

Was RGW noch kann

- **Bucket Policies** und Nutzer-Quotas.
- **Lifecycle:** Objekte nach Regel verschieben oder löschen.
- **Versionierung** und **Object Lock** für unveränderbare Daten.
- **Server Side Encryption.**
- **Multisite:** asynchrone Replikation zwischen Standorten, siehe Kapitel 34.

> **STOLPERFALLE**
>
> S3-Löschen gibt Speicher nicht sofort frei. Der Garbage Collector räumt Objekte erst später weg. Ein Bucket-Löschen vor einer Migration zeigt also nicht sofort freien Platz. `radosgw-admin gc list` zeigt, was noch ansteht.

IM INTERVIEW

Wie skalierst du RGW?

RGW ist zustandslos, also weitere Instanzen hinter einem Load Balancer. Für viele Objekte zählt vor allem der Bucket-Index: replizierter Pool auf SSD und passendes Sharding.

IM INTERVIEW

Warum liegen Index und Daten in getrennten Pools?

Sie brauchen Unterschiedliches. Der Index verlangt niedrige Latenz, also repliziert auf SSD. Die Daten brauchen günstige Kapazität, also EC auf HDD.

DAS NIMMST DU MIT

- RGW ist zustandsloses HTTP-Gateway.
- `radosgw-admin` für Nutzer und Buckets.
- Index auf SSD, Daten auf EC.
- Dynamic Resharding hält große Buckets flott.

17

Kubernetes: Rook und CSI

So kommt Ceph in deine Pods.

Zwei Bausteine

- **Ceph-CSI** ist der Treiber. Er legt Volumes an und hängt sie in Pods.
- **Rook** ist ein Operator. Er betreibt Ceph selbst als Pods im Kubernetes-Cluster.

Zwei Betriebsarten sind wichtig:

Modus	Bedeutung
Rook intern	Ceph läuft in Kubernetes. Der Operator verwaltet MONs und OSDs.
Rook External	Ceph läuft außerhalb, etwa mit cephadm. Rook liefert nur die Anbindung.

Die CRDs

Ressource	Beschreibt
`CephCluster`	Den Cluster selbst: MONs, Speicherknoten, Version.
`CephBlockPool`	Einen Pool für RBD.
`CephFilesystem`	Ein CephFS mit MDS.
`CephObjectStore`	Einen RGW mit Pools.

StorageClass und PVC

Eine StorageClass sagt, wie Volumes entstehen. Hier eine Klasse für RBD bei Rook:

`storageclass.yaml (Kern)`

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ceph-rbd
provisioner: rook-ceph.rbd.csi.ceph.com
parameters:
  clusterID: rook-ceph
  pool: kubernetes
```

```
  csi.storage.k8s.io/fstype: ext4
reclaimPolicy: Delete
allowVolumeExpansion: true
```

MERKE

In der echten Klasse gehören noch die `csi.storage.k8s.io/...-secret-name`-Parameter dazu. Sie zeigen auf die Zugangsdaten. Hier sind sie weggelassen, damit das Prinzip sichtbar bleibt.

Ein Pod fordert dann mit einem PVC Speicher an:

`pvc.yaml`

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: daten
spec:
  accessModes: [ReadWriteOnce]
  storageClassName: ceph-rbd
  resources:
    requests:
      storage: 10Gi
```

Backend	Access Mode	Passt für
RBD	`ReadWriteOnce`	Datenbanken, einzelne Pods.
CephFS	`ReadWriteMany`	Gemeinsame Dateien über Pods hinweg.

STOLPERFALLE

Ein Node stirbt hart, und der Pod startet woanders. Das RBD-Volume hängt noch am alten Node. Kubernetes erkennt das nicht sofort, also bleibt der Pod in `ContainerCreating`. Prüfe mit `rbd status`, wer noch ein Watcher des Images ist.

IM INTERVIEW

cephadm oder Rook?

cephadm, wenn Ceph eine eigene Plattform ist, die auch Nicht-Kubernetes-Clients bedient. Rook, wenn Kubernetes dein Betriebsmodell ist und Ceph dessen Lebenszyklus folgen soll. Rook External verbindet beides: getrennter Cluster, Anbindung über Rook.

IM INTERVIEW

Wie bekommt ein Pod ein Ceph-Volume?

Der PVC nennt eine StorageClass. Der CSI-Provisioner legt ein RBD-Image oder CephFS-Subvolume an. Auf dem Node hängt der CSI-Treiber das Volume ein, und der Kubelet bindet es in den Pod ein.

DAS NIMMST DU MIT

- Ceph-CSI ist der Treiber, Rook der Operator.
- RBD für RWO, CephFS für RWX.
- Rook External bindet einen externen Cluster an.

TEIL IV

Betrieb

cephadm, Plattentausch, Kapazität, Wartung, Upgrades und Monitoring.

18

cephadm und Specs

Du beschreibst den Sollzustand. cephadm hält ihn.

cephadm läuft als MGR-Modul, verbindet sich per SSH mit den Hosts und startet jeden Daemon als Container unter systemd.

> **DAS BILD IM KOPF**
>
> *Wie Kubernetes für Ceph, nur kleiner: Du schreibst, was laufen soll. Der Dirigent vergleicht ständig Soll und Ist und korrigiert.*

Was läuft?

```
ceph orch ls
```

Eselsbrücke: `ls` zeigt **Services**, also den Wunsch: „3 MONs sollen laufen."

```
ceph orch ps
```

Eselsbrücke: **ps** wie auf Linux: die wirklichen Daemons mit Host und Status.

```
ceph orch host ls
```

Eselsbrücke: Alle Hosts und ihre Labels.

Platzieren

```
ceph orch apply mon --placement="3 ceph01 ceph02 ceph03"
```

Eselsbrücke: Drei MONs, und zwar genau dort. Der Dirigent sorgt dafür.

```
ceph orch host label add ceph01 _admin
```

Eselsbrücke: Das Label `_admin` verteilt Konfiguration und Admin-Schlüssel auf diesen Host.

Eine OSD-Spec

Für echte Hardware schreibst du eine Spec. Diese hier macht jede HDD zum OSD und legt RocksDB auf die SSDs:

osd-spec.yaml

```yaml
service_type: osd
service_id: hdd-mit-ssd-db
placement:
  host_pattern: 'ceph*'
spec:
  data_devices:
    rotational: 1
  db_devices:
    rotational: 0
```

```
ceph orch apply -i osd-spec.yaml --dry-run
```

Eselsbrücke: `-i` wie **i**nput. `--dry-run` zeigt, was passieren würde, ohne es zu tun.

```
ceph orch ls --export
```

Eselsbrücke: Gibt alle Specs als YAML aus. Damit sicherst du deinen Sollzustand.

> **STOLPERFALLE**
>
> Ein `apply` ohne Dry Run bei einer falschen Spec zieht unter Umständen Platten ein, die du gar nicht meintest. Die Spec bleibt außerdem aktiv: Auch später passende Platten werden zu OSDs.

Konfiguration

Einstellungen liegen zentral in den MONs, nicht in Dateien auf den Hosts.

```
ceph config set osd osd_memory_target_autotune false
```

Eselsbrücke: Schalte das Autotuning ab. Sonst rechnet cephadm den Wert pro Host selbst aus.

```
ceph config set osd osd_memory_target 6442450944
```

Eselsbrücke: Für alle OSDs: 6 GiB RAM-Ziel, in Bytes.

```
ceph config get osd.0 osd_memory_target
```

Eselsbrücke: Was gilt für genau diesen OSD?

```
ceph config dump
```

Eselsbrücke: Alle von dir gesetzten Werte auf einen Blick.

IM INTERVIEW

Wie funktioniert cephadm?

Ein MGR-Modul verbindet sich per SSH mit den Hosts und startet die Daemons als Container mit Podman oder Docker, jeweils als systemd-Unit. Den Sollzustand beschreiben Service-Specs, und cephadm gleicht ihn laufend mit der Wirklichkeit ab.

DAS NIMMST DU MIT

- `orch ls`: Wunsch. `orch ps`: Wirklichkeit.
- Specs als YAML, immer mit `--dry-run`.
- Konfiguration liegt zentral: `ceph config`.

19

OSDs tauschen

Platten sterben. Das ist Alltag, kein Notfall.

Die drei Fragen vorher

```
ceph osd ok-to-stop osd.1
```

Eselsbrücke: Darf ich ihn kurz stoppen, ohne dass I/O hängen bleibt?

```
ceph osd safe-to-destroy osd.1
```

Eselsbrücke: Darf ich ihn endgültig zerstören, ohne Daten zu verlieren?

```
ceph device ls
```

Eselsbrücke: Welche physische Platte steckt hinter welchem Daemon?

Eine Platte ersetzen

```
ceph orch osd rm 1 --replace
```

Eselsbrücke: Leere OSD 1 und behalte seine ID für die neue Platte. Der Baum bleibt dadurch stabil.

```
ceph orch osd rm status
```

Eselsbrücke: Wie weit ist das Leeren?

Nach dem Tausch erkennt cephadm die leere Platte und legt nach deiner Spec einen neuen OSD an.

```
ceph orch device zap ceph02 /dev/sdb --force
```

Eselsbrücke: **zap** heißt: Altes löschen. Zerstört alle Daten auf dieser Platte. Nur nach Prüfung.

STOLPERFALLE

Prüfe beim `zap` Host und Gerätename doppelt. Der Befehl fragt nicht nach dem Sinn. Er löscht, was du ihm nennst.

Einen Host entfernen

```
ceph orch host drain ceph03
```

Eselsbrücke: **drain** wie Abfluss: Alle Daemons verlassen den Host. Bei OSDs dauert das, bis die Daten raus sind.

```
ceph orch host rm ceph03
```

Eselsbrücke: Erst wenn nichts mehr läuft, nimm ihn aus der Liste.

IM INTERVIEW

Wie tauschst du eine defekte Platte?

Ich prüfe mit `ok-to-stop` und `safe-to-destroy`, ob der Tausch gefahrlos ist. Dann `ceph orch osd rm --replace`, damit die ID erhalten bleibt. Platte wechseln, cephadm legt per Spec den neuen OSD an, und Ceph füllt ihn per Backfill.

DAS NIMMST DU MIT

- `ok-to-stop`: kurz weg. `safe-to-destroy`: für immer weg.
- `--replace` behält die OSD-ID.
- Prüfe Host und Gerät vor `zap`.

20

Kapazität

Ein voller Ceph-Cluster ist der schlimmste Ceph-Cluster.

Die drei Schwellen

Schwelle	Standard	Wirkung
`nearfull`	85 %	Warnung `OSD_NEARFULL`.
`backfillfull`	90 %	Keine Backfills mehr auf diesen OSD.
`full`	95 %	Schreibzugriffe im Cluster stoppen.

Entscheidend ist der **vollste** OSD, nicht der Durchschnitt. Ein einziger OSD über 95 % kann Schreibzugriffe stoppen, obwohl der Cluster nur halb voll ist.

```
ceph osd df tree
```

Eselsbrücke: Wie `df`, aber pro OSD und als Baum. Schau auf `%USE` und `VAR`.

Rechne mit Reserve

Beispiel: fünf Hosts, je 10 TB.

Rechenschritt	Ergebnis
Rohkapazität	50 TB
Nutzbar bei `size 3`	ca. 16,7 TB
Reserve: Ein Host muss ausfallen dürfen, das ergibt 4/5	ca. 13,3 TB
Unter `nearfull` bleiben (85 %)	ca. 11,3 TB

> **MERKE**
>
> Plane den Ausfall der größten Failure Domain ein. Die Kopien eines toten Hosts müssen auf den anderen Hosts noch Platz finden, ohne dass diese `backfillfull` erreichen.

Daten gleichmäßig halten

```
ceph balancer mode upmap
```

Eselsbrücke: Modus `upmap`: gezielte Ausnahmen für einzelne PGs.

```
ceph balancer on
```

Eselsbrücke: Einschalten und im Hintergrund glätten lassen.

IM INTERVIEW

`ceph df` zeigt 40 %, trotzdem `OSD_NEARFULL`. Warum?

Die Schwellen gelten pro OSD. Bei ungleicher Verteilung kann ein einzelner OSD viel voller sein. Ich prüfe `ceph osd df tree`, den Balancer und die PG-Anzahl.

IM INTERVIEW

Wie viel von 100 TB Rohspeicher kannst du nutzen?

Bei `size 3` sind es rechnerisch 33 TB. Praktisch weniger: Ich ziehe die Reserve für den Ausfall der größten Failure Domain und den Puffer zu `nearfull` ab. Plane mit etwa 60 bis 70 % davon.

DAS NIMMST DU MIT

- 85, 90, 95: `nearfull`, `backfillfull`, `full`.
- Der vollste OSD entscheidet.
- Reserve für den Ausfall eines Hosts einplanen.

21

Wartung und Flags

Wie du einen Host neu startest, ohne dass Ceph nervös wird.

Die Flags

Flag	Wirkung
`noout`	`down` wird nicht automatisch `out`.
`norebalance`	Keine PGs werden nur zum Ausgleich verschoben.
`nobackfill`	Kein Backfill.
`norecover`	Keine Recovery.
`noscrub`, `nodeep-scrub`	Kein Scrubbing.
`pause`	Kein Lesen und Schreiben. Nur für den Notfall.

Geplante Wartung

Der einfache Weg:

```
ceph orch host maintenance enter ceph02
```

Eselsbrücke: Host geht in die Pause. Daemons stoppen sauber, und `noout` gilt nur für diesen Host.

```
reboot
```

Eselsbrücke: Auf dem Host selbst ausführen.

```
ceph orch host maintenance exit ceph02
```

Eselsbrücke: Host wieder da.

Danach wartest du, bis alle PGs `active+clean` sind. Erst dann kommt der nächste Host.

Der Weg von Hand:

```
ceph osd set noout
```

Eselsbrücke: Clusterweit: Kein OSD wird `out`.

```
ceph osd unset noout
```

Eselsbrücke: Nicht vergessen! Sonst heilt sich der Cluster nicht mehr von selbst.

> **STOLPERFALLE**
>
> Das vergessene `noout` ist ein Klassiker. Monate später stirbt eine Platte, und Ceph wartet für immer, statt zu heilen. `ceph -s` zeigt die Flags unter `health`.

Recovery steuern

Der Standard-Scheduler ist **mClock**. Du wählst ein Profil:

Profil	Bevorzugt
`balanced`	Der Standard: ein Gleichgewicht von Client und Recovery.
`high_client_ops`	Client-Zugriffe. Recovery dauert länger.
`high_recovery_ops`	Recovery. Clients werden langsamer.

```
ceph config set osd osd_mclock_profile high_client_ops
```

Eselsbrücke: Die Anwendung hat Vorfahrt.

> **IM INTERVIEW**
>
> **Du musst einen Host neu starten. Wie gehst du vor?**
>
> Ich prüfe `ceph -s` und `ok-to-stop`. Dann `ceph orch host maintenance enter`, Neustart, `exit`. Ich warte auf `active+clean` und nehme erst dann den nächsten Host.

> **IM INTERVIEW**
>
> **Wozu dient `noout`?**
>
> Es verhindert, dass OSDs automatisch `out` gehen. Bei einer kurzen Wartung spare ich mir so das Umverteilen. Es muss danach wieder weg, sonst heilt der Cluster nicht.

DAS NIMMST DU MIT

- Immer ein Host zur Zeit, immer warten auf `active+clean`.
- `noout` wieder entfernen.
- mClock-Profile steuern Client gegen Recovery.

22

Upgrades

Ein Daemon nach dem anderen, ohne Ausfall.

Vorbereitung

- Release Notes lesen. Wirklich.
- `ceph -s` muss `HEALTH_OK` zeigen, oder du musst jede Warnung erklären können.
- Zuerst im Labor üben.
- Bei einem neuen Major-Release die ersten Point Releases abwarten.

Durchführen

```
ceph orch upgrade check --image quay.io/ceph/ceph:v20.2.5
```

Eselsbrücke: Trockenübung: Geht das mit diesem Image?

```
ceph orch upgrade start --image quay.io/ceph/ceph:v20.2.5
```

Eselsbrücke: Los geht's. cephadm tauscht die Container nacheinander aus.

```
ceph orch upgrade status
```

Eselsbrücke: Wo stehen wir?

```
ceph versions
```

Eselsbrücke: Wer läuft mit welcher Version? Am Ende sollte es nur noch eine geben.

```
ceph orch upgrade pause
```

Eselsbrücke: Anhalten, wenn etwas komisch aussieht. Weiter mit `resume`.

Die Reihenfolge

```
mgr -> mon -> crash -> osd -> mds -> rgw
    -> rbd-mirror -> cephfs-mirror -> iscsi -> nfs
```

Vor jedem Neustart prüft cephadm, ob das sicher ist: `ok-to-stop` bei OSDs, das Quorum bei MONs.

> **MERKE**
>
> Ceph unterstützt Upgrades über bis zu zwei Major-Releases hinweg. Clients dürfen älter sein als der Cluster. Trotzdem aktualisierst du sie zeitnah.

IM INTERVIEW

Wie läuft ein Ceph-Upgrade ab?

Rollend, gesteuert von cephadm. Erst die MGRs, dann die MONs, dann OSDs und die übrigen Dienste. Vor jedem Schritt prüft cephadm mit `ok-to-stop`, ob es sicher ist. Clients merken davon höchstens kurze Pausen.

DAS NIMMST DU MIT

- Vorher lesen und üben.
- `ceph orch upgrade start`, `status`, `pause`.
- Reihenfolge beginnt mit `mgr` und `mon`.

23

Monitoring

Prometheus, Grafana und vier Zahlen, die zählen.

cephadm bringt alles mit. Der Bootstrap richtet den Stack standardmäßig ein, außer du setzt `--skip-monitoring-stack`. Fehlt ein Teil, holst du ihn nach:

```
ceph orch apply prometheus
```

Eselsbrücke: Und genauso `grafana`, `alertmanager` und `node-exporter`.

Drei Quellen liefern die Daten:

- Das **MGR-Modul** `prometheus` (Port 9283) liefert die Cluster-Sicht.
- `ceph-exporter` (Port 9926) läuft auf jedem Host und liefert die Performance-Counter der lokalen Daemons.
- `node-exporter` liefert die Host-Sicht.

```
ceph mgr module enable prometheus
```

Eselsbrücke: Das brauchst du nur ohne cephadm: Schalte das MGR-Modul von Hand ein.

Was du beobachtest

Metrik	Wozu
`ceph_health_status`	0 ist OK, 1 Warnung, 2 Fehler.
`ceph_osd_up`	Welche OSDs laufen?
`ceph_osd_commit_latency_ms`	Latenz pro OSD. Ausreißer finden.
Nutzung pro OSD	Vorhersage, wann `nearfull` kommt.
PG-Zustände	Wie viele PGs sind nicht `active+clean`?

Gute Alerts

- `HEALTH_ERR` jederzeit.
- Ein OSD ist länger als fünf Minuten `down`.
- Ein OSD erreicht in 7 Tagen voraussichtlich `nearfull`.

- `MON_DOWN` oder Quorum verloren.
- `SLOW_OPS` über mehrere Minuten.

STOLPERFALLE

`ceph health mute` schaltet eine Warnung stumm. Sie ist deshalb nicht weg. Setze immer eine Dauer, sonst vergisst du sie: `ceph health mute OSD_NEARFULL 2h`.

IM INTERVIEW

Was überwachst du bei einem Ceph-Cluster?

Health-Status, OSDs `up` und `in`, PG-Zustände, Kapazität mit Trend, Latenzen pro OSD, Slow Ops und die Host-Metriken. Alarme gehen auf Symptome und Trends, nicht auf jede kurze Warnung.

DAS NIMMST DU MIT

- MGR-Modul, `ceph-exporter` und `node-exporter`.
- Trends auf Kapazität, Ausreißer bei der Latenz.
- `mute` nur mit Zeitlimit.

TEIL V

Troubleshooting

Hier trennt sich im Interview die Spreu vom Weizen.

24

Die Methode

Erst schauen, dann denken, dann erst handeln.

Fünf Schritte

1. **Symptom:** Was genau ist falsch? Was sagt der Anwender?
2. **Umfang:** Ein OSD, ein Host, alle?
3. **Daten:** Status, Logs, Metriken.
4. **Hypothese:** Was könnte die Ursache sein?
5. **Test:** Die kleinste sichere Prüfung.

> **MERKE**
>
> Der erste Satz im Interview lautet: „Ich schaue, was der Cluster selbst sagt, und ändere noch nichts.“ Ceph heilt sich oft selbst. Hektische Eingriffe sind die häufigste Ursache für echten Datenverlust.

Die erste Minute

```
ceph -s
```

Eselsbrücke: Der Überblick.

```
ceph health detail
```

Eselsbrücke: Klartext: Welche Warnung, welche PGs, welche OSDs?

```
ceph osd tree
```

Eselsbrücke: Wer ist `down`? Liegt ein Muster vor, etwa alle auf einem Host?

```
ceph osd df tree
```

Eselsbrücke: Ist etwas voll?

```
ceph crash ls
```

Eselsbrücke: Ist ein Daemon abgestürzt?

```
ceph log last 50
```

Eselsbrücke: Die letzten 50 Zeilen des Cluster-Logs.

```
cephadm logs --name osd.1
```

Eselsbrücke: Die Logs eines Daemons, auf seinem Host ausführen.

Die wichtigsten Health-Codes

Code	Bedeutung
`OSD_DOWN`	Ein oder mehrere OSDs antworten nicht.
`OSD_NEARFULL`, `OSD_FULL`	OSDs laufen voll.
`PG_DEGRADED`	Redundanz fehlt. I/O läuft.
`PG_AVAILABILITY`	PGs sind nicht aktiv. I/O steht.
`PG_DAMAGED`	Inkonsistenz gefunden.
`OSD_SCRUB_ERRORS`	Scrub hat Fehler gefunden.
`SLOW_OPS`	Operationen dauern zu lange.
`MON_DOWN`	Ein MON ist weg.
`MON_CLOCK_SKEW`	Die Uhren laufen auseinander.
`RECENT_CRASH`	Ein Daemon ist kürzlich abgestürzt.
`FS_DEGRADED`, `MDS_ALL_DOWN`	CephFS hat Probleme.

IM INTERVIEW

Der Cluster zeigt `HEALTH_WARN`. Was machst du zuerst?

`ceph health detail` lesen. Dann eingrenzen, ob es einen OSD, einen Host oder alle betrifft. Ich ändere nichts, bevor ich die Ursache verstanden habe. Viele Warnungen lösen sich durch Recovery.

DAS NIMMST DU MIT

- Symptom, Umfang, Daten, Hypothese, Test.
- `ceph -s`, `health detail`, `osd tree`, `crash ls`.
- Erst lesen, dann handeln.

25

Ein OSD ist weg

Der häufigste Fall. Meist ist es die Platte.

```
ceph osd tree down
```

Eselsbrücke: Zeig mir nur die, die `down` sind.

```
ceph orch ps --daemon-type osd
```

Eselsbrücke: Läuft der Container überhaupt?

```
cephadm logs --name osd.1
```

Eselsbrücke: Warum ist er gestorben?

```
dmesg -T | grep -i error
```

Eselsbrücke: Sieht der Kernel I/O-Fehler?

```
smartctl -a /dev/sdb
```

Eselsbrücke: Das Gesundheitszeugnis der Platte.

Typische Ursachen

Spur	Ursache
I/O-Fehler in `dmesg`	Platte defekt: tauschen, nicht reparieren.
`oom-killer` im Log	Zu wenig RAM. `osd_memory_target` prüfen.
Alle OSDs eines Hosts weg	Host oder Netz, nicht die Platten.
`FAILED ceph_assert`	Softwarefehler. `ceph crash info` lesen.

```
ceph orch daemon restart osd.1
```

Eselsbrücke: Ein Neustart ist sinnvoll, wenn du die Ursache kennst.

STOLPERFALLE

Flapping: Ein OSD ist abwechselnd `up` und `down`. Das bringt ständig neues Peering. Oft ist das Netz schuld, etwa eine falsche MTU oder ein überlastetes Cluster Network. Schau zuerst aufs Netz, nicht auf die OSDs.

IM INTERVIEW

Ein OSD startet nicht. Was prüfst du?

Logs mit `cephadm logs`, dann `dmesg` und SMART für die Platte, dann `ceph crash ls`. Typisch sind Platte defekt, OOM oder ein Softwarefehler. Ist die Platte defekt, ersetze ich den OSD.

DAS NIMMST DU MIT

- Logs, `dmesg`, SMART, `crash ls`.
- Alle OSDs eines Hosts: Das Problem liegt am Host.
- Flapping: erst aufs Netz schauen.

26

PGs hängen

Was tun, wenn `active+clean` nicht zurückkommt.

```
ceph pg dump_stuck inactive
```

Eselsbrücke: Zeig mir PGs, die hängen. Auch möglich: `unclean`, `stale`.

```
ceph pg 2.1a query
```

Eselsbrücke: Die Selbstauskunft der PG. Lies `recovery_state`: Dort steht, worauf sie wartet.

```
ceph pg map 2.1a
```

Eselsbrücke: Welche OSDs sind zuständig?

Was der Zustand verrät

Zustand	Erste Vermutung
`inactive`, `peered`	Zu wenig Kopien unter `min_size`. Bring die OSDs zurück.
`stale`	Alle zuständigen OSDs melden sich nicht mehr.
`undersized` für immer	CRUSH findet keinen Platz: zu wenige Hosts oder falsche Regel.
`incomplete`	Es fehlt ein OSD mit nötiger Historie. Bring ihn zurück.
`backfill_toofull`	Das Ziel ist `backfillfull`.
`activating` lange	Oft ein Limit, etwa `mon_max_pg_per_osd`.

Unfound Objects

Unfound heißt: Ceph weiß, dass ein Objekt existieren sollte, findet aber keine aktuelle Kopie.

```
ceph pg 2.1a list_unfound
```

Eselsbrücke: Welche Objekte fehlen?

```
ceph pg 2.1a mark_unfound_lost revert
```

Eselsbrücke: Letzter Ausweg: Geh zur letzten bekannten Version zurück. Das ist Datenverlust.

STOLPERFALLE

`mark_unfound_lost`, `ceph osd lost` und `min_size 1` sind keine Reparaturen. Sie sind Kapitulationen. Der beste Weg ist fast immer, den fehlenden OSD zurückzubringen. Auch eine „tote“ Platte lässt sich manchmal noch auslesen.

IM INTERVIEW

PGs bleiben `inactive`. Was tust du?

Mit `ceph pg query` sehe ich, worauf sie warten. Meist fehlen OSDs, und das Ziel ist, sie zurückzubringen. `min_size` dauerhaft zu senken, ist keine Lösung, sondern ein Datenverlustrisiko.

DAS NIMMST DU MIT

- `dump_stuck` findet PGs, `query` erklärt sie.
- Ziel: fehlende OSDs zurückbringen.
- `mark_unfound_lost` ist der letzte Ausweg.

27

Alles ist langsam

Slow Ops: Wo steckt die Operation fest?

Eine Operation gilt als **slow**, wenn sie länger als 30 Sekunden dauert.

Eingrenzen

```
ceph health detail
```

Eselsbrücke: Welche OSDs melden Slow Ops?

```
ceph osd perf
```

Eselsbrücke: Latenz pro OSD. Ein Ausreißer fällt sofort auf.

```
ceph tell osd.3 dump_ops_in_flight
```

Eselsbrücke: Was hängt gerade in diesem OSD, und in welcher Phase?

```
ceph tell osd.3 dump_historic_ops
```

Eselsbrücke: Die langsamsten der letzten Zeit.

Muster	Verdacht
Ein OSD	Sterbende Platte. `smartctl` und `iostat`.
Ein Host	Netz, CPU oder RAM auf diesem Host.
Alle OSDs	Netz, Überlast, Recovery oder voller Cluster.
`waiting for subops`	Der Primary wartet auf ein Replikat. Schau dort nach.

> **DAS BILD IM KOPF**
>
> *Ein Stau auf der Autobahn. Du fragst zuerst, ob es nur eine Spur, eine Ausfahrt oder die ganze Strecke ist. Die Antwort verrät die Ursache.*

Schnelle Prüfungen

- **Recovery läuft?** Dann zeigt `ceph -s` es. Ein mClock-Profil hilft.

- **Netz?** `ping` mit großer Paketgröße, Fehlerzähler mit `ip -s link`.
- **Platte?** `iostat -x 1` zeigt `%util` und Wartezeiten.

IM INTERVIEW

Wie grenzt du `SLOW_OPS` ein?

Ich prüfe, ob ein OSD, ein Host oder alle betroffen sind. `ceph osd perf` zeigt Ausreißer. `dump_ops_in_flight` zeigt die Phase. Ein OSD deutet auf die Platte, ein Host auf Netz oder Ressourcen, alle auf Netz, Überlast oder Recovery.

DAS NIMMST DU MIT

- Ein OSD, ein Host oder alle?
- `osd perf` und `dump_ops_in_flight`.
- `waiting for subops`: Schau beim Replikat nach.

28

Der Cluster ist voll

Wenn nichts mehr geschrieben werden kann, nicht einmal ein Löschen.

Bei `OSD_FULL` stoppt Ceph Schreibzugriffe, um keine Daten zu beschädigen. Tückisch: Auch Löschen ist bei Ceph eine Schreiboperation.

Der Weg zurück

1. **Mehr Platz schaffen:** OSDs oder Hosts dazu. Das ist der beste Weg.
2. **Daten umverteilen:** einen fast vollen OSD entlasten.
3. **Schwelle kurz anheben**, gerade so weit, dass du löschen kannst.
4. **Löschen:** Snapshots und überflüssige Daten.

```
ceph osd df tree
```

Eselsbrücke: Wer ist der Übeltäter?

```
ceph osd reweight-by-utilization
```

Eselsbrücke: Volle OSDs bekommen ein geringeres Gewicht. Vorher mit `test-reweight-by-utilization` schauen.

```
ceph osd set-full-ratio 0.96
```

Eselsbrücke: Notbremse, nur für kurze Zeit. Danach zurück auf `0.95`.

> **STOLPERFALLE**
>
> Hebe die Schwelle nie weit an. Bei wirklich vollen OSDs kann BlueStore nicht mehr arbeiten und nicht mehr starten. 95 % ist nicht die Grenze der Platte. Es ist deine letzte Reserve.

> **MERKE**
>
> Snapshots halten Daten fest. Wer „löscht“, ohne die Snapshots zu entfernen, gewinnt keinen Platz.

IM INTERVIEW

Der Cluster ist voll. Was tust du?

Ich füge möglichst Kapazität hinzu. Sonst verteile ich gezielt um und hebe die `full`-Schwelle kurz minimal an, um Daten und Snapshots zu löschen. Danach setze ich sie zurück und kläre die Kapazitätsplanung.

DAS NIMMST DU MIT

- Löschen braucht bei Ceph Platz.
- Erst erweitern, dann verteilen, dann kurz Schwelle anheben.
- Snapshots halten Speicher fest.

29

MON ohne Quorum

Der Moment, in dem `ceph -s` einfach hängt.

Einzelner MON ist weg

Das ist unkritisch. `MON_DOWN` erscheint, die anderen arbeiten weiter.

```
ceph quorum_status
```

Eselsbrücke: Wer gehört zum Quorum, und wer ist Leader?

Kein Quorum

Ohne Mehrheit hängen die `ceph`-Befehle. Dann fragst du einen MON direkt über seinen Admin Socket:

```
cephadm enter --name mon.ceph01
```

Eselsbrücke: Geh direkt in den Container des MON.

```
ceph daemon mon.ceph01 mon_status
```

Eselsbrücke: Der Socket funktioniert auch ohne Quorum. Schau auf `state`: `probing` oder `electing` heißen, er findet keine Mehrheit.

Prüfe in dieser Reihenfolge:

1. **Netz:** Erreichen sich die MONs auf den Ports 3300 und 6789? Eine Firewall?
2. **Uhr:** Läuft `chrony`? `MON_CLOCK_SKEW` warnt ab 0,05 Sekunden Abweichung.
3. **Platte:** Ist `/var` voll? Ein MON stoppt bei zu wenig Platz.
4. **IP-Adresse:** Hat ein MON durch DHCP seine Adresse geändert?

> **STOLPERFALLE**
>
> Die letzte Rettung ist, eine neue MON-Map mit `monmaptool` in den letzten gesunden MON zu laden. Das ist Handarbeit mit gestoppten Daemons. Mach das nie aus dem Gedächtnis, sondern nach der Doku und nach einem Backup von `store.db`.

IM INTERVIEW

Zwei von drei MONs sind weg. Was passiert?

Das Quorum ist verloren. Die `ceph`-Befehle hängen, und es gibt keine Kartenänderungen mehr. Bestehende Clients laufen eventuell kurz weiter, aber bei jeder Änderung bleibt alles stehen. Ich bringe einen MON zurück. Reicht das nicht, baue ich die Monmap neu.

DAS NIMMST DU MIT

- Ohne Quorum hilft der Admin Socket.
- Netz, Uhr, Platte, IP.
- Monmap-Rettung nur nach Doku.

30

CephFS klemmt

Wenn `ls` hängt, ist meist der MDS beteiligt.

```
ceph fs status
```

Eselsbrücke: Wer ist aktiv, und in welchem Zustand?

```
ceph health detail
```

Eselsbrücke: Sucht nach `MDS_`-Meldungen wie `MDS_SLOW_METADATA_IO`.

```
ceph tell mds.meinfs:0 session ls
```

Eselsbrücke: Welche Clients hängen an Rank 0?

Der MDS-Failover

```
replay -> resolve -> reconnect -> rejoin -> clientreplay ->
active
```

- `replay`: Der neue MDS liest das Journal.
- `reconnect`: Clients melden sich und beanspruchen ihre Caps neu.
- `rejoin`: Der Cache wird wieder aufgebaut.

Bleibt ein MDS lange in `replay`, ist das Journal groß. Das ist oft noch normal.

Typische Fälle

Meldung	Bedeutung
`MDS_CLIENT_LATE_RELEASE`	Ein Client gibt Caps nicht zurück.
`MDS_CACHE_OVERSIZED`	Der Cache ist zu groß, oft wegen vieler offener Dateien.
`MDS_SLOW_METADATA_IO`	RADOS ist langsam. Schau auf die OSDs.

STOLPERFALLE

Der MDS speichert nichts selbst, er hängt an RADOS. Ein langsamer Metadaten-Pool sieht wie ein MDS-Problem aus. Prüfe darum erst, ob die OSDs gesund sind. Und: Fasse `cephfs-journal-tool` nicht ohne genaue Anleitung an.

MERKE

Ein einzelner hängender Client kann alle anderen blockieren. `ceph tell mds.meinfs:0 client evict id=...` wirft ihn raus. Das ist für diesen Client ein harter Schnitt.

IM INTERVIEW

`ls` hängt auf CephFS. Wie gehst du vor?

`ceph fs status` und `ceph health detail`. Ich prüfe, ob der MDS aktiv ist und ob ein Client Caps festhält. Danach die OSDs des Metadaten-Pools. Notfalls werfe ich den blockierenden Client raus.

DAS NIMMST DU MIT

- Erst RADOS prüfen, dann den MDS.
- `replay` bis `active` ist der Failover.
- Ein Client kann alle blockieren.

31

Scrub-Fehler

Ceph findet Bitfäule, bevor du sie liest.

- **Scrub** (etwa täglich) vergleicht Metadaten der Kopien: Größe und Attribute.
- **Deep Scrub** (etwa wöchentlich) liest alle Daten und prüft die Prüfsummen.

Bei einem Fehler

```
ceph health detail
```

Eselsbrücke: Zeigt `OSD_SCRUB_ERRORS` und die PG, etwa `2.1a`.

```
rados list-inconsistent-obj 2.1a --format=json-pretty
```

Eselsbrücke: Welches Objekt, welche Kopie, welcher Fehler?

```
ceph pg repair 2.1a
```

Eselsbrücke: Repariere die PG aus den guten Kopien.

```
ceph pg deep-scrub 2.1a
```

Eselsbrücke: Prüfe sie danach noch einmal.

> **MERKE**
>
> Dank der Prüfsummen weiß BlueStore, welche Kopie falsch ist. Fragen wie „Welche Mehrheit gewinnt?“ entstehen dadurch kaum. Scrub-Fehler deuten oft auf eine sterbende Platte hin. Prüfe SMART, und tausche sie bei Wiederholung.

IM INTERVIEW

`PG_DAMAGED` erscheint. Wie reagierst du?

Ich ermittle die PG und mit `rados list-inconsistent-obj` die betroffene Kopie. Dann prüfe ich die Platte und starte `ceph pg repair`. Treten Fehler auf demselben OSD immer wieder auf, ersetze ich ihn.

DAS NIMMST DU MIT

- Scrub: Metadaten. Deep Scrub: Daten.
- `list-inconsistent-obj`, dann `pg repair`.
- Wiederholte Fehler: Platte tauschen.

TEIL VI

Produktion

Hardware, Leistung, Disaster Recovery und Sicherheit.

32

Hardware und Topologie

Ceph ist nur so gut wie sein Unterbau.

Faustregeln

Bereich	Empfehlung
Hosts	Mindestens 5 für echte Selbstheilung. 3 sind das Minimum.
Platten	Enterprise-SSDs mit Power-Loss-Protection (PLP).
RAM	Mindestens das `osd_memory_target` pro OSD, plus Reserve für Recovery.
CPU	Bei NVMe mehrere Kerne pro OSD.
Netz	25 GbE und mehr, redundant. HDD-Cluster kommen mit weniger aus.
MONs	3 oder 5, auf getrennten Failure Domains, SSD für den Store.
RAID	Nein. Ceph will die rohe Platte. Controller im HBA-Modus.

> **DAS BILD IM KOPF**
>
> *Mehr kleine Hosts sind besser als wenige große. Stirbt einer von zehn Hosts, zieht ein Zehntel um. Stirbt einer von drei, fehlt der Platz zum Heilen.*

> **STOLPERFALLE**
>
> Consumer-SSDs ohne PLP sind die Standard-Falle. Ceph wartet auf `fsync`. Ohne PLP muss die SSD wirklich in den Flash schreiben und wird um Größenordnungen langsamer als im Datenblatt. Bei einem Stromausfall kann sie bestätigte Daten verlieren.

Warum mindestens fünf Hosts?

Mit drei Hosts und `size 3` gibt es nach einem Host-Ausfall keinen Platz für die dritte Kopie. Die PGs bleiben `undersized`, bis der Host wiederkommt. Fünf Hosts heilen sich selbst und erlauben Wartung, während etwas anderes kaputtgeht.

IM INTERVIEW

Drei Nodes reichen doch, oder?

Für den Start ja, aber nicht für Selbstheilung. Fällt ein Host aus, findet CRUSH keinen Platz für die dritte Kopie. Ich empfehle fünf Hosts. Dann heilt der Cluster sich und übersteht eine Wartung mit gleichzeitigem Ausfall.

DAS NIMMST DU MIT

- Enterprise-SSDs mit PLP.
- Kein RAID unter OSDs.
- Viele kleine Hosts schlagen wenige große.

33

Performance

Erst messen, dann drehen.

Von unten nach oben messen

```
ceph tell osd.0 bench
```

Eselsbrücke: Wie schnell schreibt dieser eine OSD?

```
rados bench -p test 30 write --no-cleanup
```

Eselsbrücke: 30 Sekunden auf RADOS-Ebene schreiben. Die Daten bleiben für den Lesetest liegen.

```
rados bench -p test 30 seq
```

Eselsbrücke: Sequenziell lesen, was du gerade geschrieben hast.

```
rados -p test cleanup
```

Eselsbrücke: Aufräumen.

```
rbd bench --io-type write kubernetes/disk1
```

Eselsbrücke: Das Gleiche auf RBD-Ebene.

Für realistische Tests gibt es `fio`:

```
fio --name=test --filename=/mnt/test.bin --size=4G --direct=1 \
    --rw=randwrite --bs=4k --iodepth=32 --runtime=60 --
time_based
```

> **MERKE**
>
> Unterscheide **IOPS** (kleine Blöcke, wichtig für Datenbanken) und **Durchsatz** (große Blöcke, wichtig für Backups). Ein Cluster ist selten in beidem gut.

Stellschrauben

Stellschraube	Wirkung
Netz	Latenz und MTU beheben. Meist der größte Hebel.
Hardware	NVMe statt HDD, `block.db` auf SSD.
`osd_memory_target`	Mehr BlueStore-Cache.
PG-Anzahl	Bessere Verteilung.
Mehr OSDs pro NVMe	Nur nach Messung sinnvoll.
Kompression	Spart Platz, kostet CPU.

```
ceph osd pool set archiv compression_mode aggressive
```

Eselsbrücke: Komprimiere alles, was sich lohnt.

STOLPERFALLE

Benchmarks in einem Hypervisor oder mit Cache messen vor allem den Cache. Teste außerdem mit `--direct=1` und lange genug, damit Puffer leer laufen.

IM INTERVIEW

Ceph ist langsam. Wie gehst du vor?

Ich messe Schicht für Schicht: Platte, Netz, OSD, RADOS, Client. So finde ich den Engpass. Erst danach ändere ich etwas, und zwar eine Sache auf einmal. Oft sind Netz oder ungeeignete SSDs die Ursache, nicht die Konfiguration.

DAS NIMMST DU MIT

- `ceph tell osd bench`, `rados bench`, `rbd bench`, `fio`.
- IOPS und Durchsatz sind zwei Ziele.
- Eine Änderung pro Messung.

34

Backup und Disaster Recovery

Replikation schützt vor Hardware, nicht vor Fehlern.

Begriffe

- **RPO:** Wie viele Daten darf ich verlieren?
- **RTO:** Wie lange darf der Ausfall dauern?

Die Wahl der Technik folgt aus diesen beiden Zahlen.

Die Optionen

Technik	Wie	RPO
RBD Mirroring	Asynchron zu einem zweiten Cluster.	Minuten
CephFS Mirroring	Snapshots asynchron übertragen.	Minuten
RGW Multisite	Asynchron zwischen Zonen.	Sekunden bis Minuten
Stretch Cluster	Ein Cluster, zwei Standorte, synchron.	0
Backup	Kopie nach außen, etwa auf S3.	Stunden

RBD Mirroring

Es gibt zwei Modi:

- **Journal:** Jeder Schreibzugriff wird mitprotokolliert. Das kostet Leistung.
- **Snapshot:** Regelmäßige Snapshots werden übertragen. Einfacher und heute üblich.

```
rbd mirror pool enable kubernetes image
```

Eselsbrücke: Mirroring im Pool an, pro Image wählbar.

```
rbd mirror image enable kubernetes/disk1 snapshot
```

Eselsbrücke: Dieses Image im Snapshot-Modus spiegeln.

Zusätzlich braucht der Zielcluster den Daemon `rbd-mirror`.

Stretch Cluster

Zwei Rechenzentren, je zwei MONs, ein **Tiebreaker-MON** an einem dritten Ort. Pools bekommen `size 4`, also zwei Kopien pro Standort. Das setzt niedrige Latenz zwischen den Standorten voraus. EC-Pools und CRUSH-Regeln mit Device Class gehen im Stretch Mode nicht.

> **DAS BILD IM KOPF**
>
> *Zwei Teams streiten. Der Schiedsrichter steht auf neutralem Boden und entscheidet, wer weiterspielen darf. Ohne ihn wäre die Entscheidung ein Münzwurf.*

Backup

Exportiere RBD-Images, spiegele Daten auf anderen Speicher, und halte eine Kopie außerhalb des Clusters. Differenzen zwischen zwei Snapshots sparen viel Zeit:

```
rbd export-diff --from-snap montag kubernetes/disk1@dienstag
diff.bin
```

Eselsbrücke: Nur der Unterschied zwischen zwei Lesezeichen.

> **STOLPERFALLE**
>
> Ein Backup, dessen Wiederherstellung nie getestet wurde, ist kein Backup. Plane regelmäßige Restore-Tests ein und miss dabei die echte RTO.

> **IM INTERVIEW**
>
> **Wie schützt du Ceph vor dem Ausfall eines Rechenzentrums?**
>
> Das hängt von RPO und RTO ab. Für RPO null ein Stretch Cluster mit Tiebreaker, wenn die Latenz stimmt. Sonst zwei Cluster mit RBD Mirroring und RGW Multisite. In beiden Fällen braucht es zusätzlich ein Backup, denn Replikation kopiert auch Fehler.

> **IM INTERVIEW**
>
> **Sind Snapshots ein Backup?**
>
> Nein. Sie liegen im selben Cluster und gehen mit ihm unter. Sie helfen gegen versehentliches Löschen, aber nicht gegen Clusterausfall oder einen Angreifer mit Admin-Rechten.

DAS NIMMST DU MIT

- RPO und RTO entscheiden.
- Stretch: synchron mit Tiebreaker. Mirroring: asynchron.
- Backup liegt außerhalb. Restore testen.

35

Sicherheit

Rechte, Verschlüsselung, Netz.

Daten im Transit

msgr2 kann den Verkehr verschlüsseln. Das steuern drei Optionen:

```
ms_cluster_mode   zwischen den Daemons
ms_service_mode   Daemons nehmen Clients an
ms_client_mode    Clients verbinden sich
```

Der Wert `secure` verschlüsselt. `crc` prüft nur die Integrität.

Daten auf der Platte

- **OSD-Verschlüsselung:** `encrypted: true` in der OSD-Spec. dm-crypt schützt ausgebaute Platten.
- **RBD:** Verschlüsselung pro Image mit LUKS auf Client-Seite.
- **RGW:** Server Side Encryption mit einem externen Schlüsselspeicher.

Zugang

- Eigener CephX-Nutzer pro Anwendung, geringste Rechte.
- `client.admin` nur auf Admin-Hosts.
- Dashboard und Monitoring nicht öffentlich erreichbar machen.
- Management-Netz trennen.

> **MERKE**
>
> Ceph erzeugt ab 20.2.4 standardmäßig AES256K-CephX-Schlüssel. Kernel-Clients, also krbd, der CephFS-Kernel-Mount und damit auch Ceph-CSI, verlangen dafür Linux 7.0 oder neuer. Wenn ein `rbd map` an der Anmeldung scheitert, prüfe die Kernel-Version.

Ransomware

Replikation und Snapshots schützen nicht gegen einen Angreifer mit Admin-Rechten. Du brauchst Backups außerhalb des Clusters mit eigener Zugangs-

kontrolle, möglichst unveränderlich, etwa mit S3 Object Lock auf einem getrennten System.

IM INTERVIEW

Wie sicherst du einen Ceph-Cluster ab?

CephX mit geringsten Rechten, `client.admin` nur auf Admin-Hosts, getrennte Netze, msgr2 im Modus `secure`, verschlüsselte OSDs, Monitoring auf Ereignisse und unveränderliche Backups außerhalb des Clusters.

DAS NIMMST DU MIT

- `secure` für Transit, dm-crypt für die Platte.
- Geringste Rechte für jeden Client.
- Backups außerhalb des Clusters.

TEIL VII

Das Interview

Fragen, Design-Aufgaben und ein Spickzettel für den letzten Abend.

36

Die Fragen

Kurze Fragen, kurze Antworten. Zum Wiederholen vor dem Gespräch.

Grundlagen

Was ist Ceph?

Ein softwaredefiniertes, verteiltes Speichersystem ohne zentralen Engpass, mit Block, Datei und Objekt auf einem Fundament namens RADOS.

Was ist RADOS?

Der selbstheilende Objektspeicher unter allen Schnittstellen.

Welche Daemons gibt es?

MON, MGR, OSD, dazu MDS für CephFS und RGW für S3.

Was macht ein MON?

Er hält die Cluster-Karten und bildet per Paxos ein Quorum.

Warum ungerade MONs?

Die Mehrheit zählt. Vier vertragen nicht mehr Ausfälle als drei.

Liegt der MON im Datenpfad?

Nein. Clients holen nur Karten bei ihm und sprechen dann direkt mit OSDs.

Was passiert ohne MGR?

Die Daten fließen weiter. Es fehlen Metriken, Dashboard, Balancer und cephadm.

Wie viele OSDs pro Platte?

In der Regel einer.

Datenverteilung

Wie findet ein Client ein Objekt?

Objektname hashen zu einer PG, dann mit CRUSH die OSDs berechnen, dann direkt zum Primary.

Was ist CRUSH?

Ein Algorithmus, der aus PG, CRUSH-Map und Regel die OSDs berechnet, ohne Tabelle.

Warum PGs?

Objekte einzeln zu verfolgen, skaliert nicht. PGs bündeln Replikation und Recovery.

Wie viele PGs?

Der Autoscaler entscheidet. Früher: OSDs mal Ziel durch `size`, auf eine Zweierpotenz.

Was ist eine Failure Domain?

Die Ebene im Baum, die zwei Kopien nicht teilen dürfen.

Was sind Device Classes?

Etiketten wie `hdd` und `ssd`, mit denen Regeln Pools auf Plattentypen lenken.

Up Set oder Acting Set?

Up: Was CRUSH will. Acting: Wer die PG gerade bedient.

Was ist Peering?

Die Einigung der OSDs einer PG auf den aktuellen Stand, anhand des PG Logs.

Was macht der Balancer?

Er glättet die Verteilung, im Modus `upmap` mit gezielten Ausnahmen.

Daten und Zustände

`size` und `min_size`?

`size`: gewünschte Kopien. `min_size`: Minimum, damit I/O läuft.

Warum nicht `min_size 1`?

Bestätigte Daten lägen auf nur einer Kopie. Stirbt sie, sind sie weg.

Wann kommt das Write-ACK?

Wenn alle OSDs im Acting Set dauerhaft geschrieben haben.

`down` oder `out`?

`down`: Prozess antwortet nicht. `out`: Daten werden neu verteilt.

Wie lange bis `out`?

600 Sekunden, einstellbar über `mon_osd_down_out_interval`.

`degraded` oder `undersized`?

`degraded`: Objekte haben zu wenig Kopien. `undersized`: Es sind zu wenige OSDs im Acting Set.

`inactive`?

Sammelbegriff für PGs, die nicht `active` sind und keine I/O bedienen, meist unter `min_size`.

`incomplete`?

Es fehlt ein OSD mit nötiger Historie.

Recovery oder Backfill?

Recovery: nur Änderungen laut Log. Backfill: die ganze PG.

Replikation oder EC?

Replikation für Latenz, EC für günstige Kapazität bei großen Daten.

`min_size` bei EC?

Mindestens `k+1`.

Was ist BlueStore?

Backend, das direkt aufs Gerät schreibt, Metadaten in RocksDB, mit Prüfsummen.

Scrub oder Deep Scrub?

Scrub: Metadaten vergleichen. Deep Scrub: Daten und Prüfsummen lesen.

Schnittstellen

Wie funktioniert ein RBD-Snapshot?

Copy-on-Write. Nichts wird kopiert, bis jemand überschreibt.

Darf ein RBD-Image zwei Schreiber haben?

Mit ext4 nie. Für gemeinsamen Zugriff nimm CephFS oder ein Cluster-Dateisystem.

Was macht der MDS?

Metadaten für CephFS. Daten gehen direkt zu den OSDs.

Standby-Replay?

Ein Standby liest das Journal mit und übernimmt schneller.

Was sind Caps?

Rechte, die der MDS an Clients vergibt, damit alle konsistent cachen.

Ist RGW zustandslos?

Ja. Alles liegt in RADOS. Mehrere Instanzen stehen hinter einem Load Balancer.

Warum Bucket-Index-Sharding?

Ein einzelnes Index-Objekt wird sonst zum Engpass.

RWO oder RWX in Kubernetes?

RBD für RWO, CephFS für RWX.

cephadm oder Rook?

cephadm für einen eigenständigen Cluster, Rook für Ceph als Teil von Kubernetes.

Betrieb

Wie startest du einen Host sauber neu?

`maintenance enter`, Neustart, `exit`, dann auf `active+clean` warten.

Wozu `noout`?

Kein automatisches `out` bei Wartung. Danach wieder entfernen.

`ok-to-stop` oder `safe-to-destroy`?

Das erste: kurz stoppen. Das zweite: endgültig entfernen.

Die Full-Schwellen?

85 `nearfull`, 90 `backfillfull`, 95 `full`.

Warum ist Löschen im vollen Cluster schwer?

Auch Löschen schreibt.

Reihenfolge beim Upgrade?

MGR, MON, dann OSD, MDS, RGW und der Rest.

Warum braucht Ceph synchrone Uhren?

MONs brauchen sie für Quorum und Leases. Abweichungen ergeben `MON_CLOCK_SKEW`.

Wie löschst du einen Pool?

`mon_allow_pool_delete` setzen, Name zweimal, `--yes-i-really-really-mean-it`, zurücksetzen.

Warum Consumer-SSDs meiden?

Ohne PLP brechen sie bei synchronen Schreibzugriffen ein.

Warum kein RAID unter OSDs?

Ceph übernimmt die Redundanz selbst und will die rohe Platte sehen.

Wenn du etwas nicht weißt

Sag es. Dann denk laut: „Das habe ich so nicht betrieben. Ich vermute X, weil Y. Ich würde Z prüfen, bevor ich etwas ändere.“ Bei Speichersystemen ist genau diese Vorsicht ein gutes Zeichen.

DAS NIMMST DU MIT

- Kurze Antwort zuerst. Details nur, wenn jemand nachfragt.
- Erkläre immer, was bei einem Ausfall passiert.
- Sage auch, was du **nicht** tun würdest.

37

Design-Aufgaben

Vier Szenarien, die im Senior-Interview oft kommen.

> **MERKE**
>
> Ablauf im Interview: Anforderungen nachfragen, Zahlen klären, Architektur skizzieren, Ausfälle durchspielen, Grenzen nennen. Fragen zuerst: Wie viele Daten? Welcher Workload? Welche Ausfälle? Welches RPO und RTO? Welches Budget?

Aufgabe 1: Kubernetes-Plattform, 50 TB

Situation: Datenbanken und gemeinsame Dateien für 200 Pods.

- **Pools:** RBD repliziert auf NVMe, CephFS für RWX.
- **Hosts:** fünf, je sechs NVMe, 25 GbE doppelt.
- **Rechnung:** 50 TB mal 3 ergibt 150 TB. Mit 20 % Hostreserve und `nearfull`-Puffer rund 220 TB Rohkapazität.
- **Betrieb:** Rook intern oder Rook External, je nachdem, wer den Cluster betreibt.

Aufgabe 2: S3 für 1 PB Archiv

Situation: Große Objekte, wenige Zugriffe, niedrige Kosten.

- **Daten:** EC `k=8, m=3` auf HDD, Daten-Pool mit 11 Failure Domains plus Reserve, also 12 Hosts.
- **Index:** repliziert auf NVMe.
- **Rechnung:** 1 PB mal 1,375 ergibt 1,375 PB. Mit Reserve und Puffer etwa 2 PB roh.
- **RGW:** mehrere Instanzen hinter einem Load Balancer.
- **DB:** `block.db` auf SSD, und jede SSD als gemeinsamen Ausfallpunkt planen.

Aufgabe 3: Zwei Brandabschnitte

Situation: Der Betrieb soll weiterlaufen, wenn ein Abschnitt ausfällt.

- **Mit dritter Stelle für den Tiebreaker und RPO 0:** Stretch Cluster, je zwei MONs, `size 4`.
- **Ohne dritte Stelle:** zwei Cluster mit asynchroner Replikation, und du nennst das RPO offen.
- **Immer:** zusätzlich ein Backup außerhalb beider Abschnitte.

Aufgabe 4: Der Kunde will drei Nodes und `min_size 1`

- Drei Nodes: möglich, aber ohne Selbstheilung nach einem Host-Ausfall.
- `min_size 1`: ablehnen, denn es riskiert bestätigte Daten.
- Empfehlung: `size 3`, `min_size 2`, möglichst fünf Nodes.
- Offen sagen: Bei zwei Ausfällen blockiert der Cluster bewusst, um Daten zu schützen.

IM INTERVIEW

Warum fragst du zuerst nach RPO und RTO?

Weil sie die Architektur festlegen. RPO null erzwingt synchrone Replikation, RPO größer null erlaubt asynchrone. Ohne diese Zahlen entwerfe ich am Bedarf vorbei.

DAS NIMMST DU MIT

- Erst fragen, dann rechnen, dann zeichnen.
- Immer Reserve für den größten Ausfall einplanen.
- Grenzen und Kompromisse offen nennen.

38

Spickzettel

Alles Wichtige auf ein paar Seiten, für den letzten Abend.

Der eine Satz zu allem

Begriff	Merksatz
RADOS	Das Fundament.
CRUSH	Rechnen statt Nachschlagen.
PG	Der Postleitzahlbezirk.
Quorum	Die Mehrheit der MONs.
`min_size`	Unter diesem Wert steht I/O.
`noout`	Nach der Wartung entfernen.
EC	Weniger Platz, mehr Rechnen.
BlueStore	Rohe Platte plus RocksDB.
Snapshot	Copy-on-Write, kein Backup.

Status und Diagnose

```
ceph -s
ceph health detail
ceph osd tree
ceph osd df tree
ceph osd perf
ceph pg dump_stuck inactive
ceph pg 2.1a query
ceph crash ls
ceph log last 50
cephadm logs --name osd.1
```

Orchestrierung

```
ceph orch ls
ceph orch ps
ceph orch host ls
ceph orch device ls
ceph orch apply -i spec.yaml --dry-run
ceph orch host maintenance enter ceph02
```

```
ceph orch host drain ceph03
ceph orch osd rm 1 --replace
ceph orch upgrade start --image quay.io/ceph/ceph:v20.2.5
```

Pools und Daten

```
ceph osd pool create NAME
ceph osd pool ls detail
ceph osd pool set NAME size 3
ceph osd pool autoscale-status
ceph osd map POOL OBJEKT
ceph df
```

Clients

```
rbd create POOL/IMAGE --size 10G
rbd snap create POOL/IMAGE@SNAP
ceph fs volume create NAME
ceph fs status
radosgw-admin user create --uid=ada --display-name=Ada
ceph auth get-or-create client.NAME mon '...' osd '...'
```

Zahlen, die du kennen musst

Wert	Bedeutung
3300, 6789	MON-Ports: msgr2, msgr1.
6800–7300	Ports der OSDs, MGRs und MDS.
600 s	Bis ein `down`-OSD automatisch `out` wird.
30 s	Ab hier gilt eine Operation als slow.
85, 90, 95 %	`nearfull`, `backfillfull`, `full`.
4 MiB	Standardgröße der RBD-Objekte.
4 GiB	Standardwert von `osd_memory_target`.
`size 3`, `min_size 2`	Der Standard bei Replikation.

Niemals ohne Nachdenken

- `min_size 1` setzen.
- `ceph osd lost` oder `mark_unfound_lost` ausführen.
- `ceph orch device zap` auf einem ungeprüften Gerät.
- `ceph osd pool rm` ohne Zweitprüfung.
- `noout` setzen und vergessen.
- Full-Schwellen dauerhaft anheben.

DAS NIMMST DU MIT

- Daten vor Verfügbarkeit.
- Erst lesen, dann ändern.
- Ein Host zur Zeit, immer warten auf `active+clean`.

Stichwörter

active+clean 28
allow_ec_optimizations 34
backfillfull 63
Balancer 23
block.db 36
block.wal 36
BLUEFS_SPILLOVER 37
BlueStore 36
Bucket Index 52
Capabilities 48
Ceph 8
ceph health detail 74
ceph orch 58
ceph pg query 78
ceph-exporter 70
cephadm 15
CephFS 48
CephX 42
cluster_network 38
CRUSH 22
CSI 54
Deep Scrub 88
degraded 28
Device Class 23
dm-crypt 99
dump_ops_in_flight 80
Erasure Coding 33
exclusive-lock 47
Failure Domain 22
fio 94
Flapping 77
full 63
HEALTH_WARN 75
inactive 28
incomplete 28
max_mds 49
mClock 66
MDS 13
MGR 13
min_size 31
MON 13
mon_osd_down_out_interval 29
msgr2 38
MTU 39
Multisite 96
nearfull 63
noout 65
norebalance 65
ok-to-stop 61
OSD 13
osd_memory_target 36
Paxos 13
peered 28
Peering 26
pg_autoscaler 25
pg_num 25
Placement Group 25
PLP 92
Pool 20
Primary 21
Prometheus 70
public_network 38
Quorum 13
quorum_status 84
RADOS 8
rados bench 94
RBD 45
rbd-mirror 96
Resharding 52
RGW 13
RocksDB 36
Rook 54
S3 51
safe-to-destroy 61
Scrubbing 88
Service Spec 58
size 31
SLOW_OPS 75
Stretch Cluster 96
undersized 28
Upgrade 68
upmap 23