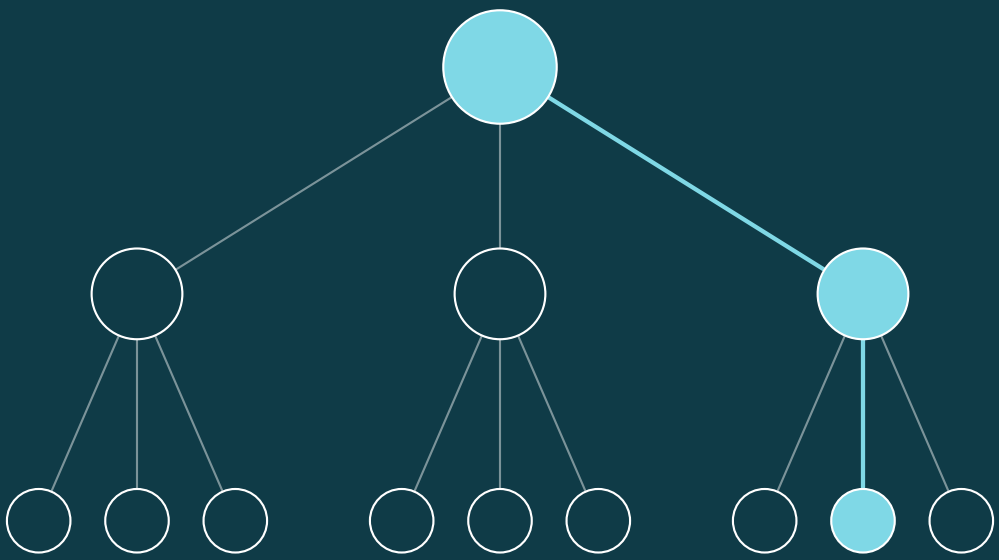
nyrvex.com

Ceph

Hands-On

Understand, operate, and repair distributed storage



Interview-ready with minimal text

Ceph

Hands-On

Understand, operate, and repair distributed storage

Interview-ready with minimal text

Thomas Zachmann

Ceph Hands-On. Understand, operate, and repair distributed storage.
As of November 2026. © 2026 Thomas Zachmann. License: CC BY-ND 4.0.

Written for Ceph Tentacle (20.2) with cephadm. The commands and sample output in this book have not been run on a cluster. Numbers in output are examples, and details can vary by version. Try everything in a lab first, never on a production cluster.

Typefaces: Charter, Fira Sans, JetBrains Mono.

Contents

How to Read This Book 5

Getting Started **7**
1 What Is Ceph? 8
2 Block, File, Object 11
3 The Daemons 13
4 The Lab 15

Thinking in Ceph **19**
5 RADOS and Objects 20
6 CRUSH 22
7 Placement Groups 25
8 PG States 27
9 Replication 30
10 Erasure Coding 32
11 BlueStore 35
12 The Network 37

The Interfaces **39**
13 Pools and Permissions 40
14 RBD: Block Storage 43
15 CephFS 46
16 RGW: Object Storage 49
17 Kubernetes: Rook and CSI 52

Operations **55**
18 cephadm and Specs 56
19 Replacing OSDs 59
20 Capacity 61
21 Maintenance and Flags 63
22 Upgrades 66
23 Monitoring 68

Troubleshooting **71**
24 The Method 72
25 An OSD Is Gone 74
26 PGs Are Stuck 76
27 Everything Is Slow 78
28 The Cluster Is Full 80
29 MON Without Quorum 82
30 CephFS Is Stuck 84
31 Scrub Errors 86

Production **89**
32 Hardware and Topology 90
33 Performance 92
34 Backup and Disaster Recovery 94
35 Security 97

The Interview **99**
36 The Questions 100
37 Design Exercises 104
38 Cheat Sheet 106
Index 109

How to Read This Book

You have an interview coming up and you need to know Ceph. Not someday. Soon. This book is built for exactly that: little text, lots of small examples, and for every topic the question you will actually be asked.

The rules

- **Type everything yourself.** Don't copy anything. Your fingers learn too.
- **One command at a time.** There are no scripts. You type, you watch what happens.
- **Lab only.** Many commands disrupt a cluster on purpose. Three VMs are enough, see chapter 4.
- **Understand the why first.** Ceph interviews rarely test commands. They test whether you can explain what happens during an outage.

> **REMEMBER**
>
> The commands and output in this book have not been run on a cluster. The output shown is shortened examples. Your numbers and text will differ.

The boxes

> **PICTURE THIS**
>
> *An everyday comparison. If you remember it, you can work out the topic from there.*

```
ceph -s
```

Memory aid: A command to type, with a memory aid below it.

> **PITFALL**
>
> The mistake almost everyone makes. And the one interviewers love to ask about.

IN THE INTERVIEW

A typical question?

An answer in two or three sentences. It can be that short.

Your study plan

You have ...	Then read ...
1 evening	Chapters 1, 3, 6, 7, 8, and 9, then chapters 36 (the questions) and 38 (cheat sheet).
2 days	Day 1: Parts I and II. Day 2: chapters 14 to 16, Part V, then chapter 36.
1 week	Day 1: Parts I and II. Then one part per day. Day 6: the design exercises from chapter 37. Day 7: review the questions.

PART I

Getting Started

What Ceph is, what it's made of, and how to build a lab.

1

What Is Ceph?

Lots of ordinary servers become one big, self-healing store.

Ceph is a software-defined, distributed storage system. Its core is a single idea:

Location of an object = function(name, cluster map)

Nobody looks up where data lives. Every participant calculates it.

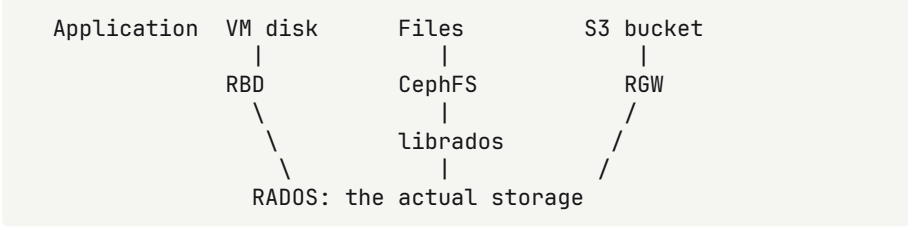
The problem

Traditional storage depends on a controller or a metadata table. As the system grows, that one spot becomes the bottleneck. If it fails, everything stops.

Ceph eliminates that spot. Clients talk directly to the disk daemons. The location of every object follows from a formula.

> **PICTURE THIS**
>
> *A library with no information desk. Every book sits at a spot you can calculate from its title. All you need is the current floor plan.*

The layer cake



- **RADOS** (Reliable Autonomic Distributed Object Store) is the foundation. It stores objects, replicates them, and heals itself.
- **RBD** provides block devices.
- **CephFS** provides a POSIX file system.
- **RGW** provides an S3- and Swift-compatible API.

All three end up storing nothing but RADOS objects.

The three promises

- **No single point of failure.** Every role runs more than once.
- **Scales horizontally.** More servers bring more capacity and more performance.
- **Self-healing.** If a disk fails, Ceph restores the copies on its own.

When there is a conflict, Ceph chooses **consistency**. It would rather block writes than allow contradictory data.

> **PITFALL**
>
> Ceph is not a backup. Replication also copies an accidental delete to all replicas immediately. Backup is a separate topic, see chapter 34.

The releases

Every release is named after a cephalopod, and the first letters run through the alphabet.

Release	Version	Status in November 2026
Squid	19.2	End of support since October 31, 2026.
Tentacle	20.2	Active. This book is based on it.
Umbrella	21.2	Active since October 28, 2026.

IN THE INTERVIEW

What is Ceph?

A software-defined, distributed storage system with no central bottleneck. The foundation is RADOS, a self-healing object store. On top sit three interfaces: RBD for block, CephFS for files, and RGW for S3.

IN THE INTERVIEW

Why does Ceph have no central metadata server for objects?

Because clients calculate the location themselves with CRUSH. They only need the current maps from the monitor. So there is no lookup in the data path, and performance grows with the cluster.

KEY TAKEAWAYS

- RADOS is the foundation. RBD, CephFS, and RGW are interfaces.
- Calculate instead of look up.
- Ceph picks consistency over availability.

2

Block, File, Object

Three ways to address data. Ceph does all three.

	Block	File	Object
You see	An empty disk	Folders and files	Buckets and keys
Access	Read and write sectors	POSIX: `open`, `read`, `rename`	HTTP: `PUT`, `GET`, `DELETE`
Sharing	Usually one client	Many clients	Any number of clients
In Ceph	RBD	CephFS	RGW
Typical use	VM disks, databases	Shared storage, home directories	Backups, media, data lakes

PICTURE THIS

Block is an empty lot: you build whatever you want on it. File is a finished apartment building with house rules. Object is a parcel counter: you drop off a package and get it back by its number.

When to use what?

- A database or VM needs its own fast disk: **RBD**.
- Several Pods need to read and write the same files: **CephFS**.
- An application stores large files over HTTP: **RGW**.

PITFALL

Two servers mounting the same RBD image with a normal file system like ext4 will destroy it. ext4 knows nothing about other writers. For shared access, use CephFS.

IN THE INTERVIEW

When do you pick RBD, when CephFS, when RGW?

RBD for a single consumer with block access, meaning VMs and databases. CephFS when many clients access the same files at the same time. RGW for HTTP applications, large amounts of data, and anything that speaks S3.

KEY TAKEAWAYS

- Block: disk. File: folders. Object: key over HTTP.
- RBD is normally meant for one writer.

3

The Daemons

Five processes, each with exactly one job.

Daemon	Job	How many?
MON	Holds the cluster maps and forms the quorum.	3 or 5, odd
MGR	Modules: dashboard, metrics, balancer, cephadm.	At least 2
OSD	Stores data, replicates, heals. One per disk.	At least 3
MDS	Metadata for CephFS. Only needed for that.	At least 2
RGW	HTTP gateway for S3. Only needed for that.	At least 2

> **PICTURE THIS**
>
> *The MON is the land registry: it knows the maps but doesn't carry any packages. The OSDs are the warehouse workers. The MGR is the back office that writes reports. The MDS is the library catalog. The RGW is the parcel counter.*

MON: the truth about the cluster

MONs hold the **cluster map**, which consists of several maps: MON map, OSD map, CRUSH map, MDS map, and more. They coordinate using the Paxos algorithm. Changes need a majority, the **quorum**.

MONs	Majority	Failures tolerated
1	1	0
3	2	1
4	3	1
5	3	2

So four MONs tolerate no more failures than three, but they add one more thing that can fail. That's why you use an odd number.

OSD: the workhorses

An **Object Storage Daemon** usually manages one disk. OSDs store data, replicate it to each other, watch each other via heartbeat, and report failures to the MONs.

MGR: everything except data

If all MGRs fail, clients keep reading and writing. But you lose the dashboard, metrics, orchestration, and current PG statistics. The data path doesn't need the MGR.

> **PITFALL**
>
> MONs are not in the data path, but without quorum no maps can change. Existing clients often keep working until something changes, for example an OSD fails. New connections don't work. So losing quorum is an emergency, even if I/O is still running.

IN THE INTERVIEW

Why an odd number of MONs?

The quorum needs a majority. Three MONs and four MONs both tolerate only one failure. The fourth just adds overhead. If the network splits, the majority prevents a split-brain.

IN THE INTERVIEW

What happens if all MGRs fail?

Data keeps flowing, because clients talk directly to the OSDs. You lose metrics, the dashboard, the balancer, and cephadm. That's why a standby always runs alongside.

KEY TAKEAWAYS

- MON: maps and quorum. OSD: data. MGR: modules.
- MDS only for CephFS, RGW only for S3.
- An odd number of MONs, because of the majority.

4

The Lab

Three VMs, one bootstrap, and you have a cluster.

What you need

Three VMs running Ubuntu 24.04, each with 8 GB of RAM plus an additional empty 20 GB disk. We call them `ceph01`, `ceph02`, and `ceph03`. All three need fixed IP addresses, a synchronized clock (`chrony`), and Podman or Docker.

> **REMEMBER**
>
> Ubuntu 24.04 ships with kernel 6.8. For `rbd map` and `mount -t ceph`, though, you need the HWE kernel 7.0, because Ceph 20.2.4 and later generate AES256K keys by default, see chapter 35.

> **PITFALL**
>
> Three VMs on one hypervisor are a lab, not an HA system. Ceph believes in three hosts, but in reality they share power and hardware.

Installing cephadm

Ubuntu ships an older package. For Tentacle, cephadm comes from the Ceph repository (`download.ceph.com`).

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

Memory aid: `cephadm` builds the cluster, `ceph-common` puts the `ceph` command on the host.

The bootstrap

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

Memory aid: **Bootstrap** as in pulling yourself up by your bootstraps: the cluster lifts itself up. You get one MON and one MGR on this host.

After that, the dashboard is running, and `/etc/ceph` contains the configuration, the admin key, and the cluster's SSH key.

Bringing in the other hosts

cephadm works over SSH. So every host needs the public key:

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

Memory aid: You give the caretaker a key to building 2.

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

Memory aid: **orch** as in **orch**estrator, the conductor: take host 2 into your orchestra.

Do the same for `ceph03`.

Adding the disks

```
ceph orch device ls
```

Memory aid: The conductor shows all disks. Only `Available: Yes` is usable.

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

Memory aid: **apply** means: this is what I want, make sure it stays that way. Every free disk becomes an OSD.

> **PITFALL**
>
> This command applies permanently: empty disks you plug in later also become OSDs. In production, you describe disks specifically in a spec, see chapter 18.

The first look

```
ceph -s
```

Memory aid: **s** as in **s**tatus: How are you doing? This is the most important command in the book.

```
  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
```

Always read the output from top to bottom: health, services, data.

```
cephadm shell
```

Memory aid: A shell inside the Ceph container, with all the tools in it. Useful on hosts without `ceph-common`.

IN THE INTERVIEW

What does `cephadm bootstrap` do?

It starts one MON and one MGR as containers on the first host, creates keys and configuration, and sets up the dashboard and the orchestrator. All other hosts and daemons are then added through `ceph orch`.

KEY TAKEAWAYS

- `cephadm bootstrap`, `ceph orch host add`, `ceph orch apply osd`.
- `ceph -s` is your first move.
- A synchronized clock and fixed IPs are mandatory.

PART II

Thinking in Ceph

RADOS, CRUSH, Placement Groups, replication. The foundation of every interview.

5

RADOS and Objects

Everything in Ceph becomes objects in pools.

Four terms carry everything:

- **Object:** a chunk of data with a name, content, and metadata. Usually up to 4 MiB in size.
- **Pool:** a logical container with rules: how many copies, on which disks, who has access.
- **Placement Group (PG):** a group of objects that move together.
- **OSD:** the daemon that ultimately stores the data on disk.

```
Object  -->  PG  -->  OSDs
       Hash     CRUSH
```

This is the most important arrow in the book. Remember it exactly like this.

Try it yourself

Create a pool and write directly to RADOS, with no RBD or S3 involved.

```
ceph osd pool create playground
```

Memory aid: In the OSD area, create a pool. A name is enough, Ceph picks the rest.

```
echo hello > greeting.txt
```

Memory aid: A small file to store.

```
rados -p playground put greeting greeting.txt
```

Memory aid: `-p` as in **p**ool. `put` stores it: name first, then the file.

```
rados -p playground ls
```

Memory aid: Like `ls` in a folder, but for objects.

```
ceph osd map playground greeting
```

Memory aid: Ask the map: where does this object live?

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

Here's how to read that: the object is in a PG of pool 2, on OSDs 2, 0, and 1. OSD 2 is the **Primary**. Note that the command doesn't ask anybody. It just calculates.

> **PICTURE THIS**
>
> *The object is the letter, the PG is the ZIP code area, the OSD is the mail carrier. If a carrier moves, you don't reassign every letter one by one. You reassign the area.*

IN THE INTERVIEW

How does a file become a RADOS object?

The interface splits up the data. RBD and CephFS cut it into pieces of 4 MiB by default, and RGW splits large uploads too. Each piece becomes an object in a pool.

KEY TAKEAWAYS

- Object, PG, OSD: hash, then CRUSH.
- `ceph osd map` shows the calculation for one object.
- The Primary is the first OSD in the list.

6

CRUSH

The formula that turns a name into a location.

CRUSH (Controlled Replication Under Scalable Hashing) calculates which OSDs store a PG. The ingredients are the **CRUSH map** (your data center as a tree) and a **rule** (how to distribute).

The tree

```
ceph osd crush tree
```

Memory aid: Show me the tree: 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
```

The `WEIGHT` normally corresponds to the size in TiB. Larger disks get more data.

Failure Domain

The **Failure Domain** defines what no two copies may share. The default is `host`: three copies live on three different hosts. You can also extend the tree with `rack`, `room`, or `datacenter`.

> **PICTURE THIS**
>
> *Three copies of your key: you don't put them in the same drawer, you put them in three houses. The Failure Domain says what counts as "a house."*

Device Classes

OSDs automatically get a class: `hdd`, `ssd`, or `nvme`. This lets you separate fast and slow data without rebuilding the tree.

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

Memory aid: Read from the left: rule "fast", starting at root `default`, one copy per `host`, class `ssd` only.

```
ceph osd pool set playground crush_rule fast
```

Memory aid: Give the pool the new rule. The data then moves.

Why CRUSH is so good

- **No table.** Every client calculates for itself.
- **Little movement.** When an OSD joins, only about its fair share of data moves.
- **Pseudorandom.** The distribution is even, but everyone calculates the same result.

The balancer

Pseudorandomness is never perfectly even. So the MGR mode `upmap` adds targeted exceptions to the OSD map and moves individual PGs.

```
ceph balancer status
```

Memory aid: Picture a scale: is it on, and in which mode?

> **PITFALL**
>
> With `size 3` and Failure Domain `host`, you need at least three hosts. If you only have two, CRUSH can't find a third spot, and the PGs stay `undersized` permanently.

IN THE INTERVIEW

How does a client find an object?

It gets the current maps from the MON. Then it hashes the object name to a PG and uses CRUSH to calculate that PG's OSDs. After that, it talks directly to the Primary. There is no lookup.

IN THE INTERVIEW

What is a Failure Domain?

The level in the CRUSH tree that two copies may not share, such as host or rack. It determines which outage a pool survives.

KEY TAKEAWAYS

- CRUSH map plus rule gives you the OSDs of a PG.
- Failure Domain: what may two copies not share?
- Device Classes separate SSD and HDD.
- The balancer in `upmap` mode smooths the distribution.

7

Placement Groups

The middle layer that makes Ceph manageable.

Why not map objects directly to OSDs? Because billions of objects can't be managed individually. Instead, Ceph manages a few thousand PGs. Replication, peering, and recovery run per PG.

How many PGs?

Too few PGs distribute data unevenly and make each PG too large. Too many cost RAM and CPU. As a rule of thumb, there is a target per OSD, controlled by `mon_target_pg_per_osd`. The default is 100, and the Tentacle docs recommend 200 for most deployments.

The old formula often comes up in interviews:

PGs in the pool ≈ OSDs × target ÷ size, then the nearest power of two

Example: 100 OSDs, target 100, `size 3` gives about 3333, so 4096.

The autoscaler

These days you rarely calculate this yourself. The MGR module `pg_autoscaler` adjusts `pg_num`.

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

Memory aid: Shows per pool: how many PGs do I have, how many would be right?

Mode	Effect
`on`	Adjusts `pg_num` on its own.
`warn`	Only warns.
`off`	Does nothing.

> **REMEMBER**
>
> Help the autoscaler: with `target_size_ratio`, you tell it what share a pool will eventually occupy. Then it doesn't have to do expensive reshuffling while the pool fills up.

Up Set and Acting Set

- **Up Set:** the OSDs that should be responsible according to CRUSH.
- **Acting Set:** the OSDs actually serving the PG right now.

They're usually the same. They differ while data is still moving.

Peering

Before a PG does any work, its OSDs agree on the current state. This is called **peering**. Every OSD keeps a **PG Log** of the latest changes. During peering, the OSDs compare these logs and work out who needs to catch up on what.

> **PICTURE THIS**
>
> *Three note-takers meet after a break. They lay their minutes side by side and agree on what was decided last. Only then does the meeting continue.*

IN THE INTERVIEW

Why do Placement Groups exist?

They decouple objects from OSDs. Otherwise Ceph would have to track every object individually. With PGs, replication, peering, and recovery run on groups of objects. That scales much better.

IN THE INTERVIEW

What happens when you increase `pg_num`?

PGs split. The objects are redistributed to the child PGs, and Ceph gradually moves data to OSDs. That creates load. So it runs in stages and, in operation, should be planned where possible.

KEY TAKEAWAYS

- PGs bundle objects for replication and recovery.
- Autoscaler on, set `target_size_ratio`.
- Up Set: should be. Acting Set: is.
- Peering: agreement via the PG Log.

8

PG States

If you can read these words, you can read Ceph.

The healthy state is called `active+clean`. Everything else is a story. States combine, for example `active+undersized+degraded`.

State	Meaning	I/O?
`active`	The PG is serving requests.	Yes
`clean`	All copies are present and up to date.	Yes
`peering`	The OSDs are reaching agreement.	Briefly no
`degraded`	Some objects have too few copies.	Yes
`undersized`	Fewer OSDs in the Acting Set than `size`.	Yes
`recovering`	Missing objects are being brought up to date.	Yes
`backfilling`	An entire PG is being copied.	Yes
`remapped`	The PG is temporarily running elsewhere.	Yes
`peered`	Peering is done, but there are too few copies for `min_size`.	No
`down`	An OSD with required data is missing.	No
`stale`	The Primary has stopped reporting to the MON.	No
`incomplete`	History is missing. Serious case.	No
`inconsistent`	Scrub found copies that differ.	Yes

> **REMEMBER**
>
> `inactive` is not a PG state of its own. The word is the umbrella term for PGs that aren't `active`, as in `ceph pg dump_stuck inactive`. What you then see are states like `peered` or `down`.

Recovery or backfill?

- **Recovery** uses the PG Log. An OSD was gone briefly and only catches up on the changes.
- **Backfill** copies the entire PG. This happens when the log isn't enough or an OSD is new.

PICTURE THIS

Recovery: you were out sick for two days and read the emails you missed. Backfill: you're new at the company and get the complete archive.

What happens during an outage

```
Time 0      OSD dies
~20 s       neighbors report: down
            PGs: active+undersized+degraded
10 min      Ceph marks the OSD out
            PGs: backfilling to other OSDs
after       PGs: active+clean
```

`down` describes the process, `out` the data placement. These are two separate switches. The wait time is controlled by `mon_osd_down_out_interval`, and the default is 600 seconds.

State	Meaning
`up` + `in`	Normal operation.
`down` + `in`	Brief outage. Ceph waits, the data stays assigned.
`down` + `out`	Ceph gives up on the OSD and redistributes its PGs.
`up` + `out`	Running, but gets no data. Typical when draining on purpose.

PITFALL

`degraded` doesn't mean broken. All data is readable and the PG is working, only the redundancy is missing. It gets dangerous only at `inactive` or `incomplete`.

IN THE INTERVIEW

What is the difference between `down` and `out`?

`down` means the process isn't responding. `out` means CRUSH redistributes its data. An OSD first goes `down` and, after ten minutes, automatically `out`. That way a short restart doesn't trigger a large data migration.

IN THE INTERVIEW

What is the difference between recovery and backfill?

Recovery uses the PG Log to catch up on missing changes only. Backfill scans and copies the entire PG, for example onto a new OSD. Backfill takes longer and creates more load.

KEY TAKEAWAYS

- Goal: `active+clean`.
- `degraded`: redundancy is missing, I/O runs. `inactive`: I/O stops.
- `down` is the process, `out` the data placement.
- Recovery reads the log, backfill copies everything.

9

Replication

Three copies, and a clear rule for when writes are allowed.

`size` and `min_size`

- `size`: this is how many copies I want.
- `min_size`: this many must be present for the PG to accept I/O.

Default and recommendation: `size 3`, `min_size 2`.

Available copies	State with `size 3`, `min_size 2`
3	`active+clean`
2	`active+undersized+degraded`. I/O runs.
1	`undersized+degraded+peered`, so not `active`. I/O blocks.
0	`down` or `stale`. No data reachable.

```
ceph osd pool get playground size
```

Memory aid: Read the `size` setting.

```
ceph osd pool set playground min_size 2
```

Memory aid: Set the lower limit for I/O.

The write path

```
Client --> Primary --> two replicas
Client <-- Primary <-- both have confirmed
```

1. The client writes to the PG's Primary.
2. The Primary writes locally and sends to the replicas in parallel.
3. Only when everyone in the Acting Set has confirmed does the client get its OK.

So an acknowledgment means the data is on all active copies. By default, reads come from the Primary. That makes replication synchronous and **strongly consistent**.

> **PICTURE THIS**
>
> *A notary with two witnesses. You get your receipt only after everyone has signed.*

> **PITFALL**
>
> `size 2`, `min_size 1` is the most famous misconfiguration. If one copy fails, new writes land on just a single disk. If that one dies too before the other has caught up, acknowledged data is gone.

IN THE INTERVIEW

Why is `min_size 1` dangerous?

The cluster then acknowledges writes that exist on only one copy. If that copy fails, acknowledged data is lost, and the returning copy has an older state. `min_size 2` deliberately sacrifices availability for data integrity.

IN THE INTERVIEW

When does the client get the write ACK?

When all OSDs in the Acting Set have durably stored the write. So latency depends on the slowest replica.

KEY TAKEAWAYS

- `size 3`, `min_size 2` is the rule.
- ACK only after all copies in the Acting Set.
- The slowest OSD determines write latency.

10

Erasure Coding

Less storage for the same safety. Paid for with compute time.

Erasure Coding (EC) splits an object into `k` data chunks and calculates `m` parity chunks. As long as `k` chunks remain, the object can be recovered.

Scheme	Failures tolerated	Raw data for 1 TB	Usable
Replica, `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%

> **PICTURE THIS**
>
> *A RAID 6 across servers. Instead of three whole copies, you have puzzle pieces plus a few spare pieces from which missing pieces can be calculated.*

Creating an EC pool

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

Memory aid: A named recipe: 4 data, 2 parity, every chunk on a different host.

```
ceph osd pool create archive erasure ec42
```

Memory aid: Pool `archive`, type `erasure`, following recipe `ec42`.

```
ceph osd pool set archive allow_ec_overwrites true
```

Memory aid: Allows overwrites in the EC pool. RBD and CephFS need this.

The price

- **More hosts.** `k=4, m=2` needs at least six Failure Domains. With a spare for healing, seven.

- **More CPU.** Parity has to be calculated.
- **Expensive small writes.** A change means read, recalculate, write (read-modify-write).
- **Expensive recovery.** For one missing chunk, Ceph reads `k` others.

`min_size` with EC

The docs recommend at least `k+1`. For `k=4, m=2`, that's 5. This way, writes always have at least one chunk in reserve.

RBD on EC

The metadata of an RBD image doesn't fit into an EC pool. So it lives in a replicated pool, and only the data goes in the EC pool:

```
rbd create fast/disk1 --size 100G --data-pool archive
```

Memory aid: Head in the replicated pool `fast`, belly in the EC pool `archive`.

> **REMEMBER**
>
> Tentacle brings improvements for small I/Os with the pool flag `allow_ec_optimizations`. The flag can't be turned off again and requires all MONs and OSDs to run at least Tentacle.

> **PITFALL**
>
> You can't change the EC profile of an existing pool. `k` and `m` are a decision for the life of the pool. To change them, you create a new pool and migrate the data.

IN THE INTERVIEW

Replication or Erasure Coding?

Replication for latency: RBD, databases, metadata, RGW index. EC for large, mostly sequential data, meaning S3 objects and archives. EC saves storage, but costs CPU, more hosts, and slower recovery.

IN THE INTERVIEW

Why does `k=4, m=2` actually need seven hosts?

Six hosts hold the six chunks. If one fails, there is no free host to heal onto, and the PGs stay degraded. The seventh provides the room for self-healing.

KEY TAKEAWAYS

- `k` data, `m` parity, `m` failures are survivable.
- `min_size` at least `k+1`.
- Metadata always replicated.
- `k` and `m` can't be changed later.

11

BlueStore

How an OSD writes directly to the disk.

BlueStore is the storage backend of the OSDs. It writes data directly to the raw block device, with no file system in between. Metadata lives in **RocksDB**, an embedded key-value database.

```
            OSD (BlueStore)
            /             \
      Data                Metadata
        |                     |
     block             RocksDB (block.db + WAL)
```

Device	Content	Typically on
`block`	The data itself.	HDD or SSD
`block.db`	RocksDB metadata.	Fast SSD or NVMe
`block.wal`	RocksDB's write-ahead log.	The fastest device, often together with `block.db`

What BlueStore brings

- **Checksums.** Every read verifies the data. Silent errors get noticed.
- **Compression** inline, selectable per pool.
- **Copy-on-write** for fast snapshots and clones.
- **Its own cache** in RAM. The limit is set by `osd_memory_target`, 4 GiB per OSD by default.

> **PICTURE THIS**
>
> *`block` is the warehouse. `block.db` is the card index that says what's where. If you put the card index of an HDD warehouse on an SSD, you find everything much faster.*

PITFALL

If `block.db` is too small, RocksDB spills over to the slow disk. Ceph then reports `BLUEFS_SPILLOVER`. And if a shared DB SSD fails, you lose all the OSDs attached to it. Plan for that as a Failure Domain.

IN THE INTERVIEW

Why did BlueStore replace FileStore?

FileStore stored objects as files in XFS and wrote everything twice, first to the journal, then to the file system. BlueStore writes directly to the device, stores metadata in RocksDB, and brings checksums and compression. That's faster and safer.

KEY TAKEAWAYS

- BlueStore: raw disk plus RocksDB.
- `block`, `block.db`, `block.wal`.
- HDD OSDs with the DB on SSD. Plan for the SSD as a shared point of failure.
- `osd_memory_target` controls the cache.

12

The Network

In Ceph, the network is the backplane.

Two networks

- **Public Network:** clients talk to MONs and OSDs.
- **Cluster Network:** OSDs replicate among themselves. Optional.

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

With `size 3`, every client write creates two replica writes. So the internal network carries more load than the public one. Add recovery on top of that.

> **PITFALL**
>
> If only the Cluster Network fails but not the Public Network, MONs still see the OSDs while OSDs lose their neighbors. OSDs then report each other as `down` (flapping). That's why many operators prefer one fast bonded network over two separate ones.

Ports

Service	Port	Protocol
MON	3300	msgr2
MON	6789	msgr1 (legacy)
OSD, MGR, MDS	6800–7300	msgr2 and msgr1

msgr2 is the current protocol. It can also encrypt.

Key points

- **Jumbo frames (MTU 9000)** only end to end. A single spot with MTU 1500 leads to strange hangs: small packets get through, large ones don't.
- **Bonding** with LACP for bandwidth and fault tolerance.
- **Latency** counts as much as bandwidth. Ceph waits synchronously for all replicas.
- **To test:** `ping -M do -s 8972 ceph02` checks MTU 9000 without fragmentation.

IN THE INTERVIEW

Does Ceph need a separate Cluster Network?

Not necessarily. It separates replication and recovery from client traffic, but adds another source of failure. With enough bandwidth, a shared, redundant network is sufficient. These days that's often the simpler choice.

KEY TAKEAWAYS

- Public: clients. Cluster: replication.
- MON 3300 and 6789, OSDs 6800–7300.
- MTU the same everywhere.
- Latency matters, not just bandwidth.

PART III

The Interfaces

Pools, permissions, RBD, CephFS, S3, and Kubernetes.

13

Pools and Permissions

A pool is the unit for rules. CephX decides who gets in.

Basic pool commands

```
ceph osd pool ls detail
```

Memory aid: `ls` lists, `detail` shows `size`, `crush_rule`, and PGs.

```
ceph df
```

Memory aid: Like `df` on Linux: how full is the cluster, how much can each pool still take?

```
ceph osd pool application enable playground rbd
```

Memory aid: Tell the pool what it's for: `rbd`, `cephfs`, or `rgw`.

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

Memory aid: An upper limit for the pool, here 10 GiB in bytes.

Deleting pools

Deleting is immediate and irreversible. So Ceph makes it awkward on purpose:

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

Memory aid: First pull the safety pin.

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

Memory aid: The name twice, plus the famous "really really."

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

Memory aid: Put the safety pin back in.

CephX

CephX is how you log in to the cluster. Every client has a name, a secret key, and **capabilities**, meaning permissions per daemon type. Both sides know the secret. It is never transmitted in the clear.

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

Memory aid: Get the key for user `k8s`, or create it. Permissions are granted per daemon.

```
ceph auth ls
```

Memory aid: List all identities and their permissions.

Profiles like `profile rbd` bundle exactly the permissions a client needs.

> **PICTURE THIS**
>
> *CephX is like a hotel key card: it opens exactly your room and the gym. The front desk issues it, but doesn't come along every time you open a door.*

> **PITFALL**
>
> `client.admin` can do anything. This key never belongs in an application or a Kubernetes cluster. Every application gets its own user with the least privileges it needs.

IN THE INTERVIEW

How does authentication work in Ceph?

With CephX, a ticket system with a shared secret. The client logs in to the MON and gets a ticket. With that, it talks directly to the OSDs. Permissions are attached to the user as capabilities per daemon type.

KEY TAKEAWAYS

- `ceph df` shows how much room is left.
- Deleting a pool takes the name twice, a flag, and an option.
- Every application gets its own CephX user.

14

RBD: Block Storage

A virtual disk, spread across the whole cluster.

An **RBD image** is a thin-provisioned disk. Ceph splits it into objects of 4 MiB. Storage is allocated only when you write.

Creating and using an image

```
ceph osd pool create kubernetes
```

Memory aid: The pool for our disks.

```
rbd pool init kubernetes
```

Memory aid: Gets the pool ready for RBD.

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

Memory aid: Always `pool/image`. Like a path.

```
rbd info kubernetes/disk1
```

Memory aid: Size, object size, and active features of the image.

```
rbd map kubernetes/disk1
```

Memory aid: Attach the disk to this Linux kernel. You get `/dev/rbd0`.

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

Memory aid: Now it's a perfectly normal disk.

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

Memory aid: Mount it as usual.

Two paths to the client

- **krbd:** the kernel module. You get `/dev/rbd0`. Simple and fast.
- **librbd:** the userspace library. QEMU and KVM use it, for example.

> **PITFALL**
>
> krbd often doesn't support every feature. Then `rbd map` fails with a message about image features. Check `rbd info` and adjust the features or use a newer kernel.

Snapshots and clones

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

Memory aid: The `@` marks a point in time, like a bookmark.

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

Memory aid: Shows all bookmarks.

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

Memory aid: Write protection for the snapshot. Mandatory for clone v1. Clone v2 doesn't need it, but requires `require_min_compat_client` to be at least `mimic`.

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

Memory aid: A new disk that shares the snapshot's content. This takes seconds.

```
rbd flatten kubernetes/disk2
```

Memory aid: The clone becomes independent. It copies everything and detaches from the snapshot.

Snapshots use copy-on-write: extra storage is used only when something changes.

> **REMEMBER**
>
> A snapshot of a mounted file system is only **crash-consistent**. For a clean state, freeze the file system with `fsfreeze` first and thaw it afterward.

Growing an image

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

Memory aid: The disk grows immediately.

```
resize2fs /dev/rbd0
```

Memory aid: The file system doesn't grow on its own. You have to tell it separately.

Features you should know

Feature	Purpose
`exclusive-lock`	Only one client writes. Without it, many other features can't be used.
`object-map`	Tracks which objects exist. Speeds up `du`, `clone`, `export`.
`fast-diff`	Fast diffs between snapshots.
`journaling`	For journal-based mirroring. Costs performance.

IN THE INTERVIEW

How does an RBD snapshot work?

Copy-on-write at the RADOS level. Nothing is copied when the snapshot is taken. Only on overwrite does Ceph preserve the old state of the affected object. That's why snapshots are instant and start out small.

IN THE INTERVIEW

Clone or snapshot?

A snapshot is a read-only point in time. A clone is a new writable image built on top of a snapshot. That's how you quickly start many VMs from one base image.

KEY TAKEAWAYS

- Always `pool/image`, snapshots with `@`.
- `rbd create`, `map`, `mkfs`, `mount`.
- krbd in the kernel, librbd for KVM.
- `resize`, then grow the file system.

15

CephFS

A shared POSIX file system for many clients.

CephFS keeps a strict separation:

- **Data** is written by clients directly to the data pool, meaning to the OSDs.
- **Metadata** (names, permissions, directories) goes through the **MDS** and lives in its own pool.

So the MDS is never in the data path. It's a cache with a journal.

Creating a file system

```
ceph fs volume create myfs
```

Memory aid: One command for everything: two pools, the file system, and MDS daemons.

```
ceph fs status
```

Memory aid: Who is `active`, who is `standby`? How full are the pools?

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

Memory aid: `rw` to read and write, `p` for quotas and layouts, `s` for snapshots.

Mounting

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

This reads as: user `ada`, cluster from `ceph.conf` (nothing given before the dot), file system `myfs`, path `/`.

Multiple MDS

By default there is one active MDS. Others stand by (`standby`). With `max_mds`, you split the namespace across several **ranks**.

```
ceph fs set myfs max_mds 2
```

Memory aid: Two active MDS. Only raise this if you're really short on metadata throughput.

```
ceph fs set myfs allow_standby_replay true
```

Memory aid: The standby follows the journal. That makes failover shorter.

Caps

So that many clients can work together safely, the MDS hands out **capabilities**: permissions like "you may cache this file" or "you may write." If a second client wants access, the MDS revokes the permissions.

> **PICTURE THIS**
>
> *Caps are like a relay baton: only the one holding it may run. If a client doesn't pass it on, everyone else stands and waits.*

Quotas, snapshots, subvolumes

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

Memory aid: Quotas are attributes on the directory, not on the user.

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

Memory aid: You create a snapshot with an `mkdir` in the hidden folder `.snap`.

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

Memory aid: A little world of its own with a quota and a path. The Kubernetes CSI driver uses this.

> **PITFALL**
>
> More active MDS are not automatically better. `max_mds` only distributes metadata work and costs coordination. If a larger MDS cache (`mds_cache_memory_limit`) or pinning with `ceph.dir.pin` helps, that's often the better first choice.

IN THE INTERVIEW

Why does CephFS scale well?

Data access goes directly from the client to the OSDs. The MDS only deals with metadata. When needed, several MDS ranks share the namespace, dynamically or via pinning.

IN THE INTERVIEW

What are caps?

Permissions that the MDS grants to clients, for example to cache or buffer writes. They ensure consistency across clients. If a client doesn't give them back, it blocks others.

KEY TAKEAWAYS

- Data directly to the OSDs, metadata through the MDS.
- `ceph fs volume create` creates everything you need.
- Standby-replay shortens failover.
- Caps govern shared access.

16

RGW: Object Storage

S3 on your own hardware.

The **RADOS Gateway** (RGW) is an HTTP server. It translates S3 calls into RADOS objects. RGW instances are stateless. You run several behind a load balancer.

Starting it

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

Memory aid: Two gateways, on these hosts, on port 8080.

User and bucket

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

Memory aid: RGW has its own admin command. The output contains the access key and secret key.

Now any S3 tool works, for example the AWS CLI:

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

The pools behind RGW

Pool	Content	Recommendation
`.rgw.root`	Realm and zones.	Replicated
`default.rgw.buckets.index`	The buckets' table of contents.	Replicated, on SSD
`default.rgw.buckets.data`	The objects themselves.	EC possible, HDD too
`default.rgw.log`, `.meta`	Logs and metadata.	Replicated

Bucket index and sharding

Every bucket has an index, stored in RADOS objects. A giant bucket with a single index object would be a bottleneck. So the index is split into **shards**. **Dynamic Resharding** does this automatically.

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

Memory aid: Are any buckets too full for their shards?

> **PICTURE THIS**
>
> *The index is the table of contents. With a million entries, you split it into several volumes, or else all the readers fight over the same book.*

What else RGW can do

- **Bucket Policies** and user quotas.
- **Lifecycle:** move or delete objects by rule.
- **Versioning** and **Object Lock** for immutable data.
- **Server Side Encryption.**
- **Multisite:** asynchronous replication between sites, see chapter 34.

> **PITFALL**
>
> An S3 delete doesn't free storage right away. The garbage collector cleans up objects only later. So deleting a bucket before a migration doesn't show free space immediately. `radosgw-admin gc list` shows what's still pending.

IN THE INTERVIEW

How do you scale RGW?

RGW is stateless, so you add more instances behind a load balancer. With many objects, the bucket index matters most: a replicated pool on SSD and suitable sharding.

IN THE INTERVIEW

Why do index and data live in separate pools?

They need different things. The index demands low latency, so replicated on SSD. The data needs cheap capacity, so EC on HDD.

KEY TAKEAWAYS

- RGW is a stateless HTTP gateway.
- `radosgw-admin` for users and buckets.
- Index on SSD, data on EC.
- Dynamic Resharding keeps large buckets snappy.

17

Kubernetes: Rook and CSI

How Ceph gets into your Pods.

Two building blocks

- **Ceph-CSI** is the driver. It creates volumes and mounts them into Pods.
- **Rook** is an operator. It runs Ceph itself as Pods in the Kubernetes cluster.

Two operating modes matter:

Mode	Meaning
Rook internal	Ceph runs in Kubernetes. The operator manages MONs and OSDs.
Rook External	Ceph runs outside, for example with cephadm. Rook only provides the connection.

The CRDs

Resource	Describes
`CephCluster`	The cluster itself: MONs, storage nodes, version.
`CephBlockPool`	A pool for RBD.
`CephFilesystem`	A CephFS with MDS.
`CephObjectStore`	An RGW with pools.

StorageClass and PVC

A StorageClass says how volumes are created. Here's a class for RBD with Rook:

`storageclass.yaml (core)`

```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
```

REMEMBER

The real class also includes the `csi.storage.k8s.io/...-secret-name` parameters. They point to the credentials. They're left out here so the principle stays visible.

A Pod then requests storage with a PVC:

pvc.yaml

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

Backend	Access Mode	Good for
RBD	`ReadWriteOnce`	Databases, single Pods.
CephFS	`ReadWriteMany`	Shared files across Pods.

PITFALL

A node dies hard, and the Pod starts elsewhere. The RBD volume is still attached to the old node. Kubernetes doesn't notice right away, so the Pod stays in `ContainerCreating`. Use `rbd status` to check who is still a watcher of the image.

IN THE INTERVIEW

cephadm or Rook?

cephadm when Ceph is a platform of its own that also serves non-Kubernetes clients. Rook when Kubernetes is your operating model and Ceph should follow its lifecycle. Rook External combines both: a separate cluster, connected through Rook.

IN THE INTERVIEW

How does a Pod get a Ceph volume?

The PVC names a StorageClass. The CSI provisioner creates an RBD image or CephFS subvolume. On the node, the CSI driver mounts the volume, and the kubelet binds it into the Pod.

KEY TAKEAWAYS

- Ceph-CSI is the driver, Rook the operator.
- RBD for RWO, CephFS for RWX.
- Rook External connects an external cluster.

PART IV

Operations

cephadm, disk replacement, capacity, maintenance, upgrades, and monitoring.

18

cephadm and Specs

You describe the desired state. cephadm maintains it.

cephadm runs as a MGR module, connects to the hosts over SSH, and starts every daemon as a container under systemd.

> **PICTURE THIS**
>
> *Like Kubernetes for Ceph, only smaller: you write down what should run. The conductor constantly compares desired and actual state and corrects any difference.*

What's running?

```
ceph orch ls
```

Memory aid: `ls` shows **services**, meaning the wish: "3 MONs should run."

```
ceph orch ps
```

Memory aid: **ps** as on Linux: the actual daemons with host and status.

```
ceph orch host ls
```

Memory aid: All hosts and their labels.

Placement

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

Memory aid: Three MONs, and exactly there. The conductor makes sure of it.

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

Memory aid: The `_admin` label distributes the configuration and admin key to this host.

An OSD spec

For real hardware, you write a spec. This one turns every HDD into an OSD and puts RocksDB on the SSDs:

osd-spec.yaml

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

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

Memory aid: `-i` as in **i**nput. `--dry-run` shows what would happen without doing it.

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

Memory aid: Outputs all specs as YAML. This is how you back up your desired state.

> **PITFALL**
>
> An `apply` without a dry run on a wrong spec may grab disks you never meant. The spec also stays active: disks that match later become OSDs too.

Configuration

Settings live centrally in the MONs, not in files on the hosts.

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

Memory aid: Turn off autotuning. Otherwise cephadm calculates the value per host itself.

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

Memory aid: For all OSDs: a 6 GiB RAM target, in bytes.

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

Memory aid: What applies to exactly this OSD?

```
ceph config dump
```

Memory aid: All the values you've set, at a glance.

IN THE INTERVIEW

How does cephadm work?

A MGR module connects to the hosts over SSH and starts the daemons as containers with Podman or Docker, each as a systemd unit. Service specs describe the desired state, and cephadm continuously reconciles it with reality.

KEY TAKEAWAYS

- `orch ls`: the wish. `orch ps`: reality.
- Specs as YAML, always with `--dry-run`.
- Configuration lives centrally: `ceph config`.

19

Replacing OSDs

Disks die. That's everyday life, not an emergency.

The three questions up front

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

Memory aid: May I stop it briefly without I/O hanging?

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

Memory aid: May I destroy it for good without losing data?

```
ceph device ls
```

Memory aid: Which physical disk is behind which daemon?

Replacing a disk

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

Memory aid: Drain OSD 1 and keep its ID for the new disk. That keeps the tree stable.

```
ceph orch osd rm status
```

Memory aid: How far along is the draining?

After the swap, cephadm detects the empty disk and creates a new OSD according to your spec.

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

Memory aid: **zap** means: wipe the old stuff. Destroys all data on this disk. Only after checking.

PITFALL

With `zap`, double-check the host and device name. The command doesn't ask whether this makes sense. It wipes what you tell it to.

Removing a host

```
ceph orch host drain ceph03
```

Memory aid: **drain** as in a sink: all daemons flow off the host. For OSDs, this takes until the data is out.

```
ceph orch host rm ceph03
```

Memory aid: Only when nothing is running anymore, take it off the list.

IN THE INTERVIEW

How do you replace a failed disk?

I use `ok-to-stop` and `safe-to-destroy` to check that the swap is safe. Then `ceph orch osd rm --replace`, so the ID is preserved. Swap the disk, cephadm creates the new OSD from the spec, and Ceph fills it via backfill.

KEY TAKEAWAYS

- `ok-to-stop`: gone briefly. `safe-to-destroy`: gone forever.
- `--replace` keeps the OSD ID.
- Check host and device before `zap`.

20

Capacity

A full Ceph cluster is the worst Ceph cluster.

The three thresholds

Threshold	Default	Effect
`nearfull`	85%	Warning `OSD_NEARFULL`.
`backfillfull`	90%	No more backfills to this OSD.
`full`	95%	Writes in the cluster stop.

What counts is the **fullest** OSD, not the average. A single OSD above 95% can stop writes, even if the cluster is only half full.

```
ceph osd df tree
```

Memory aid: Like `df`, but per OSD and as a tree. Look at `%USE` and `VAR`.

Plan for a reserve

Example: five hosts, 10 TB each.

Calculation step	Result
Raw capacity	50 TB
Usable with `size 3`	about 16.7 TB
Reserve: one host must be able to fail, which leaves 4/5	about 13.3 TB
Stay below `nearfull` (85%)	about 11.3 TB

> **REMEMBER**
>
> Plan for the failure of the largest Failure Domain. The copies from a dead host must still fit on the other hosts without those reaching `backfillfull`.

Keeping data even

```
ceph balancer mode upmap
```

Memory aid: `upmap` mode: targeted exceptions for individual PGs.

```
ceph balancer on
```

Memory aid: Turn it on and let it smooth things out in the background.

IN THE INTERVIEW

`ceph df` shows 40%, but you still get `OSD_NEARFULL`. Why?

The thresholds apply per OSD. With an uneven distribution, a single OSD can be much fuller. I check `ceph osd df tree`, the balancer, and the PG count.

IN THE INTERVIEW

How much of 100 TB of raw storage can you use?

With `size 3`, it's 33 TB on paper. In practice less: I subtract the reserve for the failure of the largest Failure Domain and the buffer to `nearfull`. Plan on about 60 to 70% of that.

KEY TAKEAWAYS

- 85, 90, 95: `nearfull`, `backfillfull`, `full`.
- The fullest OSD decides.
- Plan a reserve for losing a host.

21

Maintenance and Flags

How to restart a host without making Ceph nervous.

The flags

Flag	Effect
`noout`	`down` doesn't automatically become `out`.
`norebalance`	No PGs are moved just for balancing.
`nobackfill`	No backfill.
`norecover`	No recovery.
`noscrub`, `nodeep-scrub`	No scrubbing.
`pause`	No reads or writes. Emergencies only.

Planned maintenance

The easy way:

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

Memory aid: The host takes a break. Daemons stop cleanly, and `noout` applies only to this host.

```
reboot
```

Memory aid: Run this on the host itself.

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

Memory aid: The host is back.

Then you wait until all PGs are `active+clean`. Only then do you move on to the next host.

The manual way:

```
ceph osd set noout
```

Memory aid: Cluster-wide: no OSD goes `out`.

```
ceph osd unset noout
```

Memory aid: Don't forget! Otherwise the cluster stops healing itself.

> **PITFALL**
>
> The forgotten `noout` is a classic. Months later a disk dies, and Ceph waits forever instead of healing. `ceph -s` shows the flags under `health`.

Controlling recovery

The default scheduler is **mClock**. You pick a profile:

Profile	Favors
`balanced`	The default: a balance of client and recovery.
`high_client_ops`	Client access. Recovery takes longer.
`high_recovery_ops`	Recovery. Clients get slower.

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

Memory aid: The application has the right of way.

> **IN THE INTERVIEW**
>
> **You need to restart a host. How do you go about it?**
>
> I check `ceph -s` and `ok-to-stop`. Then `ceph orch host maintenance enter`, reboot, `exit`. I wait for `active+clean` and only then take the next host.

> **IN THE INTERVIEW**
>
> **What is `noout` for?**
>
> It prevents OSDs from automatically going `out`. For short maintenance, that saves me the redistribution. It has to come off again afterward, or the cluster won't heal.

KEY TAKEAWAYS

- Always one host at a time, always wait for `active+clean`.
- Remove `noout` again.
- mClock profiles set client against recovery.

22

Upgrades

One daemon after another, with no outage.

Preparation

- Read the release notes. Really.
- `ceph -s` must show `HEALTH_OK`, or you must be able to explain every warning.
- Practice in the lab first.
- With a new major release, wait for the first point releases.

Running it

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

Memory aid: Dry run: does this work with this image?

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

Memory aid: Here we go. cephadm swaps the containers one after another.

```
ceph orch upgrade status
```

Memory aid: Where are we?

```
ceph versions
```

Memory aid: Who is running which version? At the end, there should be only one.

```
ceph orch upgrade pause
```

Memory aid: Stop if something looks odd. Continue with `resume`.

The order

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

Before every restart, cephadm checks that it's safe: `ok-to-stop` for OSDs, the quorum for MONs.

> **REMEMBER**
>
> Ceph supports upgrades across up to two major releases. Clients may be older than the cluster. You still update them promptly.

IN THE INTERVIEW

How does a Ceph upgrade work?

Rolling, driven by cephadm. First the MGRs, then the MONs, then OSDs and the remaining services. Before each step, cephadm checks with `ok-to-stop` that it's safe. At most, clients notice brief pauses.

KEY TAKEAWAYS

- Read and practice first.
- `ceph orch upgrade start`, `status`, `pause`.
- The order starts with `mgr` and `mon`.

23

Monitoring

Prometheus, Grafana, and four numbers that count.

cephadm brings everything along. The bootstrap sets up the stack by default, unless you pass `--skip-monitoring-stack`. If a piece is missing, you add it afterward:

```
ceph orch apply prometheus
```

Memory aid: Likewise for `grafana`, `alertmanager`, and `node-exporter`.

Three sources provide the data:

- The **MGR module** `prometheus` (port 9283) provides the cluster view.
- `ceph-exporter` (port 9926) runs on every host and provides the performance counters of the local daemons.
- `node-exporter` provides the host view.

```
ceph mgr module enable prometheus
```

Memory aid: You only need this without cephadm: turn on the MGR module by hand.

What to watch

Metric	Purpose
`ceph_health_status`	0 is OK, 1 warning, 2 error.
`ceph_osd_up`	Which OSDs are running?
`ceph_osd_commit_latency_ms`	Latency per OSD. Find outliers.
Usage per OSD	Predict when `nearfull` will arrive.
PG states	How many PGs are not `active+clean`?

Good alerts

- `HEALTH_ERR` at any time.
- An OSD is `down` for more than five minutes.
- An OSD is projected to reach `nearfull` within 7 days.

- `MON_DOWN` or quorum lost.
- `SLOW_OPS` for several minutes.

> **PITFALL**
>
> `ceph health mute` silences a warning. That doesn't make it go away. Always set a duration, or you'll forget about it: `ceph health mute OSD_NEARFULL 2h`.

IN THE INTERVIEW

What do you monitor on a Ceph cluster?

Health status, OSDs `up` and `in`, PG states, capacity with trends, latency per OSD, slow ops, and the host metrics. Alerts go on symptoms and trends, not on every short warning.

KEY TAKEAWAYS

- MGR module, `ceph-exporter`, and `node-exporter`.
- Trends for capacity, outliers for latency.
- `mute` only with a time limit.

PART V

Troubleshooting

This is where the interview separates the wheat from the chaff.

24

The Method

First look, then think, and only then act.

Five steps

1. **Symptom:** What exactly is wrong? What does the user say?
2. **Scope:** One OSD, one host, all of them?
3. **Data:** Status, logs, metrics.
4. **Hypothesis:** What could the cause be?
5. **Test:** The smallest safe check.

> **REMEMBER**
>
> Your first sentence in the interview is: “I’ll look at what the cluster itself says, and I won’t change anything yet.” Ceph often heals itself. Frantic interventions are the most common cause of real data loss.

The first minute

```
ceph -s
```

Memory aid: The overview.

```
ceph health detail
```

Memory aid: In plain language: which warning, which PGs, which OSDs?

```
ceph osd tree
```

Memory aid: Who is `down`? Is there a pattern, such as all on one host?

```
ceph osd df tree
```

Memory aid: Is anything full?

```
ceph crash ls
```

Memory aid: Did a daemon crash?

```
ceph log last 50
```

Memory aid: The last 50 lines of the cluster log.

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

Memory aid: A daemon's logs. Run this on its host.

The most important health codes

Code	Meaning
`OSD_DOWN`	One or more OSDs aren't responding.
`OSD_NEARFULL`, `OSD_FULL`	OSDs are filling up.
`PG_DEGRADED`	Redundancy is missing. I/O runs.
`PG_AVAILABILITY`	PGs aren't active. I/O stops.
`PG_DAMAGED`	Inconsistency found.
`OSD_SCRUB_ERRORS`	Scrub found errors.
`SLOW_OPS`	Operations are taking too long.
`MON_DOWN`	A MON is gone.
`MON_CLOCK_SKEW`	The clocks are drifting apart.
`RECENT_CRASH`	A daemon crashed recently.
`FS_DEGRADED`, `MDS_ALL_DOWN`	CephFS has problems.

IN THE INTERVIEW

The cluster shows `HEALTH_WARN`. What do you do first?

Read `ceph health detail`. Then narrow down whether it affects one OSD, one host, or all of them. I don't change anything until I understand the cause. Many warnings clear up through recovery.

KEY TAKEAWAYS

- Symptom, scope, data, hypothesis, test.
- `ceph -s`, `health detail`, `osd tree`, `crash ls`.
- Read first, then act.

25

An OSD Is Gone

The most common case. Usually it's the disk.

```
ceph osd tree down
```

Memory aid: Show me only the ones that are `down`.

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

Memory aid: Is the container running at all?

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

Memory aid: Why did it die?

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

Memory aid: Does the kernel see I/O errors?

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

Memory aid: The disk's bill of health.

Typical causes

Clue	Cause
I/O errors in `dmesg`	Disk failed: replace it, don't repair it.
`oom-killer` in the log	Not enough RAM. Check `osd_memory_target`.
All OSDs on a host gone	Host or network, not the disks.
`FAILED ceph_assert`	Software bug. Read `ceph crash info`.

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

Memory aid: A restart makes sense once you know the cause.

PITFALL

Flapping: An OSD alternates between `up` and `down`. That causes constant re-peering. The network is often to blame, such as a wrong MTU or an overloaded Cluster Network. Look at the network first, not the OSDs.

IN THE INTERVIEW

An OSD won't start. What do you check?

Logs with `cephadm logs`, then `dmesg` and SMART for the disk, then `ceph crash ls`. The usual suspects are a failed disk, OOM, or a software bug. If the disk has failed, I replace the OSD.

KEY TAKEAWAYS

- Logs, `dmesg`, SMART, `crash ls`.
- All OSDs on a host: the problem is the host.
- Flapping: look at the network first.

26

PGs Are Stuck

What to do when `active+clean` doesn't come back.

```
ceph pg dump_stuck inactive
```

Memory aid: Show me PGs that are stuck. Also possible: `unclean`, `stale`.

```
ceph pg 2.1a query
```

Memory aid: The PG's self-report. Read `recovery_state`: it says what the PG is waiting for.

```
ceph pg map 2.1a
```

Memory aid: Which OSDs are responsible?

What the state reveals

State	First guess
`inactive`, `peered`	Too few copies, below `min_size`. Bring the OSDs back.
`stale`	All responsible OSDs have stopped reporting.
`undersized` forever	CRUSH can't find a spot: too few hosts or a wrong rule.
`incomplete`	An OSD with required history is missing. Bring it back.
`backfill_toofull`	The target is `backfillfull`.
`activating` for a long time	Often a limit, such as `mon_max_pg_per_osd`.

Unfound objects

Unfound means: Ceph knows an object should exist, but can't find a current copy.

```
ceph pg 2.1a list_unfound
```

Memory aid: Which objects are missing?

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

Memory aid: Last resort: go back to the last known version. This is data loss.

PITFALL

`mark_unfound_lost`, `ceph osd lost`, and `min_size 1` are not repairs. They are capitulations. The best path is almost always to bring the missing OSD back. Even a “dead” disk can sometimes still be read.

IN THE INTERVIEW

PGs stay `inactive`. What do you do?

With `ceph pg query`, I see what they’re waiting for. Usually OSDs are missing, and the goal is to bring them back. Permanently lowering `min_size` is not a solution, it’s a data loss risk.

KEY TAKEAWAYS

- `dump_stuck` finds PGs, `query` explains them.
- Goal: bring missing OSDs back.
- `mark_unfound_lost` is the last resort.

27

Everything Is Slow

Slow ops: where is the operation stuck?

An operation counts as **slow** if it takes longer than 30 seconds.

Narrowing it down

```
ceph health detail
```

Memory aid: Which OSDs report slow ops?

```
ceph osd perf
```

Memory aid: Latency per OSD. An outlier stands out immediately.

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

Memory aid: What is stuck in this OSD right now, and in which phase?

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

Memory aid: The slowest ones from recent history.

Pattern	Suspect
One OSD	Dying disk. `smartctl` and `iostat`.
One host	Network, CPU, or RAM on that host.
All OSDs	Network, overload, recovery, or a full cluster.
`waiting for subops`	The Primary is waiting for a replica. Look there.

> **PICTURE THIS**
>
> *A traffic jam on the highway. You first ask whether it's one lane, one exit, or the whole stretch. The answer gives away the cause.*

Quick checks

- **Recovery running?** Then `ceph -s` shows it. An mClock profile helps.

- **Network?** `ping` with a large packet size, error counters with `ip -s link`.
- **Disk?** `iostat -x 1` shows `%util` and wait times.

IN THE INTERVIEW

How do you narrow down `SLOW_OPS`?

I check whether one OSD, one host, or all are affected. `ceph osd perf` shows outliers. `dump_ops_in_flight` shows the phase. One OSD points to the disk, one host to network or resources, all of them to network, overload, or recovery.

KEY TAKEAWAYS

- One OSD, one host, or all?
- `osd perf` and `dump_ops_in_flight`.
- `waiting for subops`: look at the replica.

28

The Cluster Is Full

When nothing can be written anymore, not even a delete.

With `OSD_FULL`, Ceph stops writes to avoid damaging data. The catch: in Ceph, deleting is also a write operation.

The way back

1. **Make more room:** add OSDs or hosts. This is the best way.
2. **Redistribute data:** take the load off a nearly full OSD.
3. **Briefly raise the threshold**, just far enough that you can delete.
4. **Delete:** snapshots and unneeded data.

```
ceph osd df tree
```

Memory aid: Who's the culprit?

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

Memory aid: Full OSDs get a lower weight. Check with `test-reweight-by-utilization` first.

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

Memory aid: Emergency brake, for a short time only. Then back to `0.95`.

> **PITFALL**
>
> Never raise the threshold far. On truly full OSDs, BlueStore can no longer work or even start. 95% is not the limit of the disk. It's your last reserve.

> **REMEMBER**
>
> Snapshots hold on to data. If you "delete" without removing the snapshots, you gain no space.

IN THE INTERVIEW

The cluster is full. What do you do?

I add capacity if at all possible. Otherwise I redistribute selectively and briefly raise the `full` threshold a tiny bit to delete data and snapshots. Then I reset it and sort out the capacity planning.

KEY TAKEAWAYS

- In Ceph, deleting needs space.
- First expand, then redistribute, then briefly raise the threshold.
- Snapshots hold on to storage.

29

MON Without Quorum

The moment when `ceph -s` simply hangs.

A single MON is gone

This isn't critical. `MON_DOWN` appears, and the others keep working.

```
ceph quorum_status
```

Memory aid: Who is in the quorum, and who is the leader?

No quorum

Without a majority, the `ceph` commands hang. Then you ask a MON directly through its admin socket:

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

Memory aid: Go directly into the MON's container.

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

Memory aid: The socket works even without quorum. Look at `state`: `probing` or `electing` means it can't find a majority.

Check in this order:

1. **Network:** Can the MONs reach each other on ports 3300 and 6789? A firewall?
2. **Clock:** Is `chrony` running? `MON_CLOCK_SKEW` warns from a deviation of 0.05 seconds.
3. **Disk:** Is `/var` full? A MON stops when it has too little space.
4. **IP address:** Did a MON change its address through DHCP?

PITFALL

The last rescue is to load a new MON map with `monmaptool` into the last healthy MON. That's manual work with stopped daemons. Never do this from memory. Follow the docs, and only after a backup of `store.db`.

IN THE INTERVIEW

Two out of three MONs are gone. What happens?

Quorum is lost. The `ceph` commands hang, and there are no more map changes. Existing clients may keep running for a short while, but with any change everything stops. I bring a MON back. If that's not enough, I rebuild the monmap.

KEY TAKEAWAYS

- Without quorum, the admin socket helps.
- Network, clock, disk, IP.
- Monmap rescue only by the docs.

30

CephFS Is Stuck

When `ls` hangs, the MDS is usually involved.

```
ceph fs status
```

Memory aid: Who is active, and in which state?

```
ceph health detail
```

Memory aid: Look for `MDS_` messages like `MDS_SLOW_METADATA_IO`.

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

Memory aid: Which clients are attached to rank 0?

The MDS failover

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

- `replay`: the new MDS reads the journal.
- `reconnect`: clients check in and reclaim their caps.
- `rejoin`: the cache is rebuilt.

If an MDS stays in `replay` for a long time, the journal is large. That's often still normal.

Typical cases

Message	Meaning
`MDS_CLIENT_LATE_RELEASE`	A client isn't returning caps.
`MDS_CACHE_OVERSIZED`	The cache is too large, often because of many open files.
`MDS_SLOW_METADATA_IO`	RADOS is slow. Look at the OSDs.

PITFALL

The MDS doesn't store anything itself, it depends on RADOS. A slow metadata pool looks like an MDS problem. So first check whether the OSDs are healthy. And don't touch `cephfs-journal-tool` without exact instructions.

REMEMBER

A single hanging client can block all the others. `ceph tell mds.myfs:0 client evict id=...` kicks it out. For that client, it's a hard cut.

IN THE INTERVIEW

`ls` hangs on CephFS. How do you proceed?

`ceph fs status` and `ceph health detail`. I check whether the MDS is active and whether a client is holding on to caps. Then the OSDs of the metadata pool. As a last resort, I kick out the blocking client.

KEY TAKEAWAYS

- Check RADOS first, then the MDS.
- `replay` through `active` is the failover.
- One client can block everyone.

31

Scrub Errors

Ceph finds bit rot before you read it.

- **Scrub** (roughly daily) compares the metadata of the copies: size and attributes.
- **Deep Scrub** (roughly weekly) reads all the data and verifies the checksums.

When there's an error

```
ceph health detail
```

Memory aid: Shows `OSD_SCRUB_ERRORS` and the PG, for example `2.1a`.

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

Memory aid: Which object, which copy, which error?

```
ceph pg repair 2.1a
```

Memory aid: Repair the PG from the good copies.

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

Memory aid: Check it once more afterward.

> **REMEMBER**
>
> Thanks to checksums, BlueStore knows which copy is wrong. Questions like "Which majority wins?" hardly ever arise. Scrub errors often point to a dying disk. Check SMART, and replace the disk if it happens again.

IN THE INTERVIEW

`PG_DAMAGED` appears. How do you respond?

I identify the PG and, with `rados list-inconsistent-obj`, the affected copy. Then I check the disk and run `ceph pg repair`. If errors keep showing up on the same OSD, I replace it.

KEY TAKEAWAYS

- Scrub: metadata. Deep Scrub: data.
- `list-inconsistent-obj`, then `pg repair`.
- Repeated errors: replace the disk.

PART VI

Production

Hardware, performance, disaster recovery, and security.

32

Hardware and Topology

Ceph is only as good as what's underneath it.

Rules of thumb

Area	Recommendation
Hosts	At least 5 for real self-healing. 3 is the minimum.
Disks	Enterprise SSDs with power-loss protection (PLP).
RAM	At least the `osd_memory_target` per OSD, plus a reserve for recovery.
CPU	With NVMe, several cores per OSD.
Network	25 GbE or more, redundant. HDD clusters get by with less.
MONs	3 or 5, on separate Failure Domains, SSD for the store.
RAID	No. Ceph wants the raw disk. Controllers in HBA mode.

PICTURE THIS

More small hosts beat a few big ones. If one of ten hosts dies, a tenth of the data moves. If one of three dies, there's no room left to heal.

PITFALL

Consumer SSDs without PLP are the standard trap. Ceph waits on `fsync`. Without PLP, the SSD really has to write to flash and gets orders of magnitude slower than the data sheet says. In a power failure, it can lose acknowledged data.

Why at least five hosts?

With three hosts and `size 3`, there is no room for the third copy after a host failure. The PGs stay `undersized` until the host comes back. Five hosts heal themselves and allow maintenance while something else breaks.

IN THE INTERVIEW

Three nodes are enough, right?

For getting started, yes, but not for self-healing. If a host fails, CRUSH finds no room for the third copy. I recommend five hosts. Then the cluster heals itself and survives maintenance with a simultaneous failure.

KEY TAKEAWAYS

- Enterprise SSDs with PLP.
- No RAID under OSDs.
- Many small hosts beat a few big ones.

33

Performance

Measure first, then tune.

Measure from the bottom up

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

Memory aid: How fast does this one OSD write?

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

Memory aid: Write for 30 seconds at the RADOS level. The data stays around for the read test.

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

Memory aid: Sequentially read what you just wrote.

```
rados -p test cleanup
```

Memory aid: Clean up.

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

Memory aid: The same at the RBD level.

For realistic tests, there's `fio`:

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

> **REMEMBER**
>
> Distinguish **IOPS** (small blocks, important for databases) from **throughput** (large blocks, important for backups). A cluster is rarely good at both.

Tuning knobs

Knob	Effect
Network	Fix latency and MTU. Usually the biggest lever.
Hardware	NVMe instead of HDD, `block.db` on SSD.
`osd_memory_target`	More BlueStore cache.
PG count	Better distribution.
More OSDs per NVMe	Only makes sense after measuring.
Compression	Saves space, costs CPU.

```
ceph osd pool set archive compression_mode aggressive
```

Memory aid: Compress everything worth compressing.

> **PITFALL**
>
> Benchmarks in a hypervisor or with a cache mostly measure the cache. Also test with `--direct=1` and long enough for buffers to drain.

> **IN THE INTERVIEW**
>
> **Ceph is slow. How do you proceed?**
>
> I measure layer by layer: disk, network, OSD, RADOS, client. That's how I find the bottleneck. Only then do I change something, and one thing at a time. The cause is often the network or unsuitable SSDs, not the configuration.

KEY TAKEAWAYS

- `ceph tell osd bench`, `rados bench`, `rbd bench`, `fio`.
- IOPS and throughput are two different goals.
- One change per measurement.

34

Backup and Disaster Recovery

Replication protects against hardware, not against mistakes.

Terms

- **RPO:** How much data can I afford to lose?
- **RTO:** How long can the outage last?

The choice of technology follows from these two numbers.

The options

Technology	How	RPO
RBD Mirroring	Asynchronously to a second cluster.	Minutes
CephFS Mirroring	Snapshots transferred asynchronously.	Minutes
RGW Multisite	Asynchronously between zones.	Seconds to minutes
Stretch Cluster	One cluster, two sites, synchronous.	0
Backup	A copy outside, for example on S3.	Hours

RBD Mirroring

There are two modes:

- **Journal:** every write is logged along the way. This costs performance.
- **Snapshot:** regular snapshots are transferred. Simpler and the norm today.

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

Memory aid: Mirroring on in the pool, selectable per image.

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

Memory aid: Mirror this image in snapshot mode.

The target cluster also needs the `rbd-mirror` daemon.

Stretch Cluster

Two data centers, two MONs each, a **tiebreaker MON** at a third location. Pools get `size 4`, meaning two copies per site. This requires low latency between the sites. EC pools and CRUSH rules with a Device Class don't work in stretch mode.

> **PICTURE THIS**
>
> *Two teams are arguing. The referee stands on neutral ground and decides who may keep playing. Without the referee, the decision would be a coin toss.*

Backup

Export RBD images, mirror data to other storage, and keep a copy outside the cluster. Diffs between two snapshots save a lot of time:

```
rbd export-diff --from-snap monday kubernetes/disk1@tuesday
diff.bin
```

Memory aid: Only the difference between two bookmarks.

> **PITFALL**
>
> A backup whose restore has never been tested is not a backup. Schedule regular restore tests and measure the real RTO while you're at it.

> **IN THE INTERVIEW**
>
> **How do you protect Ceph against the loss of a data center?**
>
> That depends on RPO and RTO. For an RPO of zero, a Stretch Cluster with a tiebreaker, if the latency is right. Otherwise, two clusters with RBD Mirroring and RGW Multisite. Either way, you also need a backup, because replication copies mistakes too.

> **IN THE INTERVIEW**
>
> **Are snapshots a backup?**
>
> No. They live in the same cluster and go down with it. They help against accidental deletes, but not against a cluster outage or an attacker with admin rights.

KEY TAKEAWAYS

- RPO and RTO decide.
- Stretch: synchronous with a tiebreaker. Mirroring: asynchronous.
- A backup lives outside. Test the restore.

35

Security

Permissions, encryption, network.

Data in transit

msgr2 can encrypt traffic. Three options control this:

```
ms_cluster_mode   between the daemons
ms_service_mode   daemons accept clients
ms_client_mode    clients connect
```

The value `secure` encrypts. `crc` only checks integrity.

Data on disk

- **OSD encryption:** `encrypted: true` in the OSD spec. dm-crypt protects removed disks.
- **RBD:** per-image encryption with LUKS on the client side.
- **RGW:** Server Side Encryption with an external key store.

Access

- A separate CephX user per application, least privilege.
- `client.admin` only on admin hosts.
- Don't make the dashboard and monitoring publicly reachable.
- Separate the management network.

> **REMEMBER**
>
> Ceph 20.2.4 and later generate AES256K CephX keys by default. Kernel clients, meaning krbd, the CephFS kernel mount, and therefore also Ceph-CSI, need Linux 7.0 or newer for them. If an `rbd map` fails at login, check the kernel version.

Ransomware

Replication and snapshots don't protect against an attacker with admin rights. You need backups outside the cluster with their own access control, immutable if possible, for example with S3 Object Lock on a separate system.

IN THE INTERVIEW

How do you secure a Ceph cluster?

CephX with least privilege, `client.admin` only on admin hosts, separate networks, msgr2 in `secure` mode, encrypted OSDs, monitoring for events, and immutable backups outside the cluster.

KEY TAKEAWAYS

- `secure` for transit, dm-crypt for the disk.
- Least privilege for every client.
- Backups outside the cluster.

PART VII

The Interview

Questions, design exercises, and a cheat sheet for the last evening.

36

The Questions

Short questions, short answers. For review before the interview.

Fundamentals

What is Ceph?

A software-defined, distributed storage system with no central bottleneck, offering block, file, and object on one foundation called RADOS.

What is RADOS?

The self-healing object store beneath all the interfaces.

Which daemons are there?

MON, MGR, OSD, plus MDS for CephFS and RGW for S3.

What does a MON do?

It holds the cluster maps and forms a quorum using Paxos.

Why an odd number of MONs?

The majority counts. Four tolerate no more failures than three.

Is the MON in the data path?

No. Clients only fetch maps from it and then talk directly to OSDs.

What happens without a MGR?

Data keeps flowing. You lose metrics, the dashboard, the balancer, and cephadm.

How many OSDs per disk?

Usually one.

Data distribution

How does a client find an object?

Hash the object name to a PG, then use CRUSH to calculate the OSDs, then go directly to the Primary.

What is CRUSH?

An algorithm that calculates the OSDs from the PG, the CRUSH map, and a rule, with no table.

Why PGs?

Tracking objects individually doesn't scale. PGs bundle replication and recovery.

How many PGs?

The autoscaler decides. In the past: OSDs times target divided by `size`, to a power of two.

What is a Failure Domain?

The level in the tree that two copies may not share.

What are Device Classes?

Labels like `hdd` and `ssd` that rules use to steer pools onto disk types.

Up Set or Acting Set?

Up: what CRUSH wants. Acting: who is serving the PG right now.

What is peering?

The OSDs of a PG agreeing on the current state, based on the PG Log.

What does the balancer do?

It smooths the distribution, in `upmap` mode with targeted exceptions.

Data and states

`size` and `min_size`?

`size`: desired copies. `min_size`: the minimum for I/O to run.

Why not `min_size 1`?

Acknowledged data would be on only one copy. If it dies, the data is gone.

When does the write ACK arrive?

When all OSDs in the Acting Set have durably written.

`down` or `out`?

`down`: the process isn't responding. `out`: data gets redistributed.

How long until `out`?

600 seconds, configurable via `mon_osd_down_out_interval`.

`degraded` or `undersized`?

`degraded`: objects have too few copies. `undersized`: there are too few OSDs in the Acting Set.

`inactive`?

Umbrella term for PGs that aren't `active` and don't serve I/O, usually below `min_size`.

`incomplete`?

An OSD with required history is missing.

Recovery or backfill?

Recovery: only the changes per the log. Backfill: the entire PG.

Replication or EC?

Replication for latency, EC for cheap capacity with large data.

`min_size` with EC?

At least `k+1`.

What is BlueStore?

A backend that writes directly to the device, with metadata in RocksDB and checksums.

Scrub or Deep Scrub?

Scrub: compare metadata. Deep Scrub: read data and checksums.

Interfaces

How does an RBD snapshot work?

Copy-on-write. Nothing is copied until someone overwrites.

Can an RBD image have two writers?

With ext4, never. For shared access, use CephFS or a cluster file system.

What does the MDS do?

Metadata for CephFS. Data goes directly to the OSDs.

Standby-replay?

A standby follows the journal and takes over faster.

What are caps?

Permissions the MDS grants to clients so that everyone caches consistently.

Is RGW stateless?

Yes. Everything lives in RADOS. Several instances sit behind a load balancer.

Why bucket index sharding?

Otherwise a single index object becomes the bottleneck.

RWO or RWX in Kubernetes?

RBD for RWO, CephFS for RWX.

cephadm or Rook?

cephadm for a standalone cluster, Rook for Ceph as part of Kubernetes.

Operations

How do you restart a host cleanly?

`maintenance enter`, reboot, `exit`, then wait for `active+clean`.

What is `noout` for?

No automatic `out` during maintenance. Remove it afterward.

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

The first: stop briefly. The second: remove for good.

The full thresholds?

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

Why is deleting hard on a full cluster?

Deleting writes too.

Upgrade order?

MGR, MON, then OSD, MDS, RGW, and the rest.

Why does Ceph need synchronized clocks?

MONs need them for quorum and leases. Deviations result in `MON_CLOCK_SKEW`.

How do you delete a pool?

Set `mon_allow_pool_delete`, name twice, `--yes-i-really-really-mean-it`, reset.

Why avoid consumer SSDs?

Without PLP, they collapse under synchronous writes.

Why no RAID under OSDs?

Ceph handles redundancy itself and wants to see the raw disk.

If you don't know something

Say so. Then think out loud: "I haven't run this myself. I suspect X, because Y. I would check Z before changing anything." With storage systems, exactly this caution is a good sign.

KEY TAKEAWAYS

- Short answer first. Details only if someone asks.
- Always explain what happens during an outage.
- Also say what you would **not** do.

37

Design Exercises

Four scenarios that often come up in senior interviews.

> **REMEMBER**
>
> The flow in the interview: ask about requirements, pin down numbers, sketch the architecture, walk through failures, name the limits. Ask first: How much data? What workload? Which failures? What RPO and RTO? What budget?

Exercise 1: Kubernetes platform, 50 TB

Situation: Databases and shared files for 200 Pods.

- **Pools:** RBD replicated on NVMe, CephFS for RWX.
- **Hosts:** five, six NVMe each, dual 25 GbE.
- **Calculation:** 50 TB times 3 gives 150 TB. With a 20% host reserve and a `nearfull` buffer, about 220 TB of raw capacity.
- **Operations:** Rook internal or Rook External, depending on who runs the cluster.

Exercise 2: S3 for a 1 PB archive

Situation: Large objects, few accesses, low cost.

- **Data:** EC `k=8, m=3` on HDD, data pool with 11 Failure Domains plus a spare, so 12 hosts.
- **Index:** replicated on NVMe.
- **Calculation:** 1 PB times 1.375 gives 1.375 PB. With reserve and buffer, about 2 PB raw.
- **RGW:** several instances behind a load balancer.
- **DB:** `block.db` on SSD, and plan for each SSD as a shared point of failure.

Exercise 3: Two fire compartments

Situation: Operations must continue if one compartment fails.

- **With a third location for the tiebreaker and RPO 0:** Stretch Cluster, two MONs each, `size 4`.

- **Without a third location:** two clusters with asynchronous replication, and you state the RPO openly.
- **Always:** an additional backup outside both compartments.

Exercise 4: The customer wants three nodes and `min_size 1`

- Three nodes: possible, but with no self-healing after a host failure.
- `min_size 1`: decline, because it puts acknowledged data at risk.
- Recommendation: `size 3`, `min_size 2`, five nodes if possible.
- Say openly: with two failures, the cluster deliberately blocks in order to protect data.

IN THE INTERVIEW

Why do you ask about RPO and RTO first?

Because they determine the architecture. An RPO of zero forces synchronous replication, an RPO above zero allows asynchronous. Without these numbers, I'd be designing past the actual need.

KEY TAKEAWAYS

- First ask, then calculate, then draw.
- Always plan a reserve for the largest failure.
- Name limits and trade-offs openly.

38

Cheat Sheet

Everything important on a few pages, for the last evening.

One sentence for everything

Term	Takeaway
RADOS	The foundation.
CRUSH	Calculate instead of look up.
PG	The ZIP code area.
Quorum	The majority of the MONs.
`min_size`	Below this value, I/O stops.
`noout`	Remove after maintenance.
EC	Less space, more calculating.
BlueStore	Raw disk plus RocksDB.
Snapshot	Copy-on-write, not a backup.

Status and diagnostics

```
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
```

Orchestration

```
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 and data

```
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 OBJECT
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 '...'
```

Numbers you need to know

Value	Meaning
3300, 6789	MON ports: msgr2, msgr1.
6800–7300	Ports of the OSDs, MGRs, and MDS.
600 s	Until a `down` OSD automatically goes `out`.
30 s	From here on, an operation counts as slow.
85, 90, 95%	`nearfull`, `backfillfull`, `full`.
4 MiB	Default size of RBD objects.
4 GiB	Default value of `osd_memory_target`.
`size 3`, `min_size 2`	The default for replication.

Never without thinking

- Setting `min_size 1`.
- Running `ceph osd lost` or `mark_unfound_lost`.
- `ceph orch device zap` on an unchecked device.
- `ceph osd pool rm` without a second check.
- Setting `noout` and forgetting it.
- Permanently raising the full thresholds.

KEY TAKEAWAYS

- Data before availability.
- Read first, then change.
- One host at a time, always wait for `active+clean`.

Index

active+clean 27
allow_ec_optimizations 33
backfillfull 61
Balancer 23
block.db 35
block.wal 35
BLUEFS_SPILLOVER 36
BlueStore 35
Bucket Index 50
Capabilities 46
Ceph 8
ceph health detail 72
ceph orch 56
ceph pg query 76
ceph-exporter 68
cephadm 15
CephFS 46
CephX 40
cluster_network 37
CRUSH 22
CSI 52
Deep Scrub 86
degraded 27
Device Class 23
dm-crypt 97
dump_ops_in_flight 78
Erasure Coding 32
exclusive-lock 45
Failure Domain 22
fio 92
Flapping 75
full 61
HEALTH_WARN 73
inactive 27
incomplete 27
max_mds 47
mClock 64
MDS 13
MGR 13
min_size 30
MON 13
mon_osd_down_out_interval 28
msgr2 37
MTU 38
Multisite 94
nearfull 61
noout 63
norebalance 63
ok-to-stop 59
OSD 13
osd_memory_target 35
Paxos 13
peered 27
Peering 26
pg_autoscaler 25
pg_num 25
Placement Group 25
PLP 90
Pool 20
Primary 21
Prometheus 68
public_network 37
Quorum 13
quorum_status 82
RADOS 8
rados bench 92
RBD 43
rbd-mirror 94
Resharding 50
RGW 13
RocksDB 35
Rook 52
S3 49
safe-to-destroy 59
Scrubbing 86
Service Spec 56
size 30
SLOW_OPS 73
Stretch Cluster 94
undersized 27
Upgrade 66
upmap 23