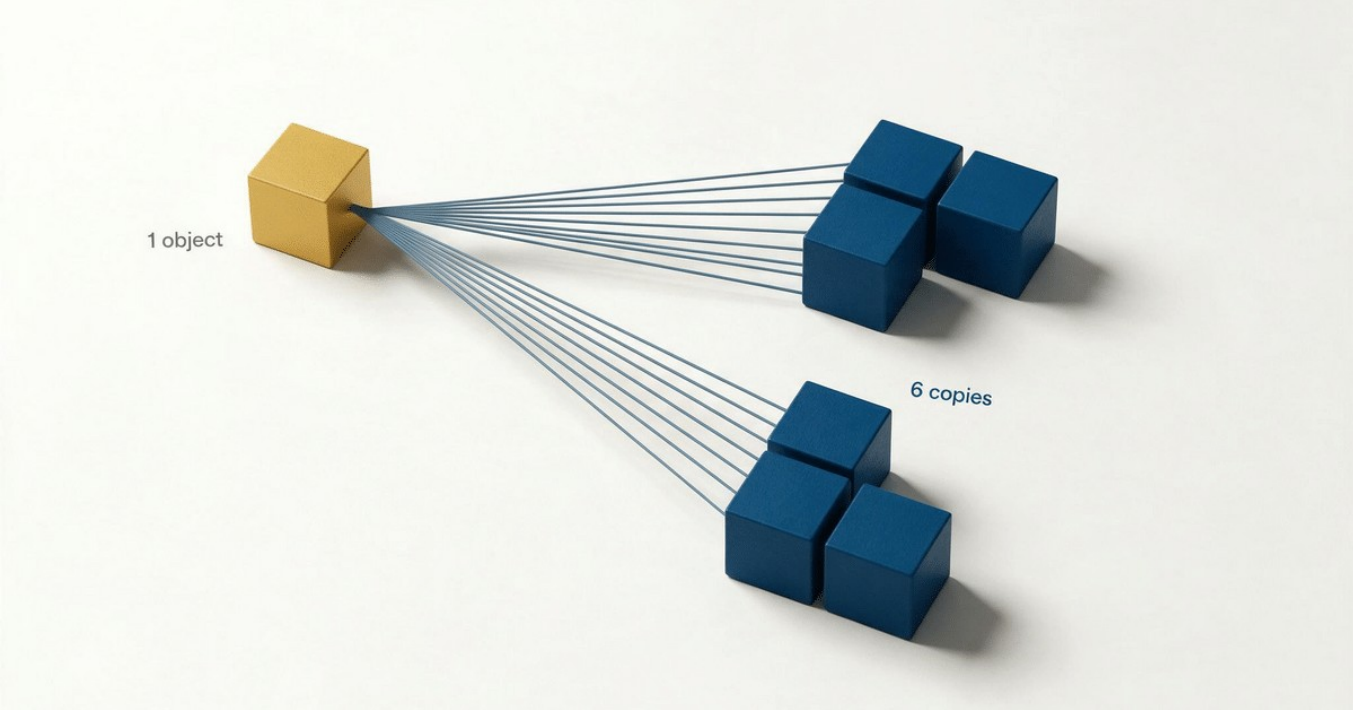
Ceph in Production: RBD, RGW and CephFS

Julien Laurenceau | Published 2026-09-30 | Updated 2026-09-30

https://pepitedata.com/ceph-in-production-rbd-rgw-cephfs/

PepiteData | LinkedIn: Julien Laurenceau | X: @Julien_Lau



Ceph is three storage products sharing one cluster: block devices (RBD), an S3-compatible gateway (RGW) and a shared POSIX filesystem (CephFS), all carved from the same object layer by CRUSH. Production questions arrive in that vocabulary and get answered in it. This guide maps the whole surface: which daemon serves which workload, what surviving a failure domain actually costs, and how the operations lifecycle (upgrades, patches, key rotation) really behaves across recent Ceph releases. It collects what the Ceph posts on this blog established on live Squid and Tentacle platforms; each section states the mechanism and links to the post that proves it.

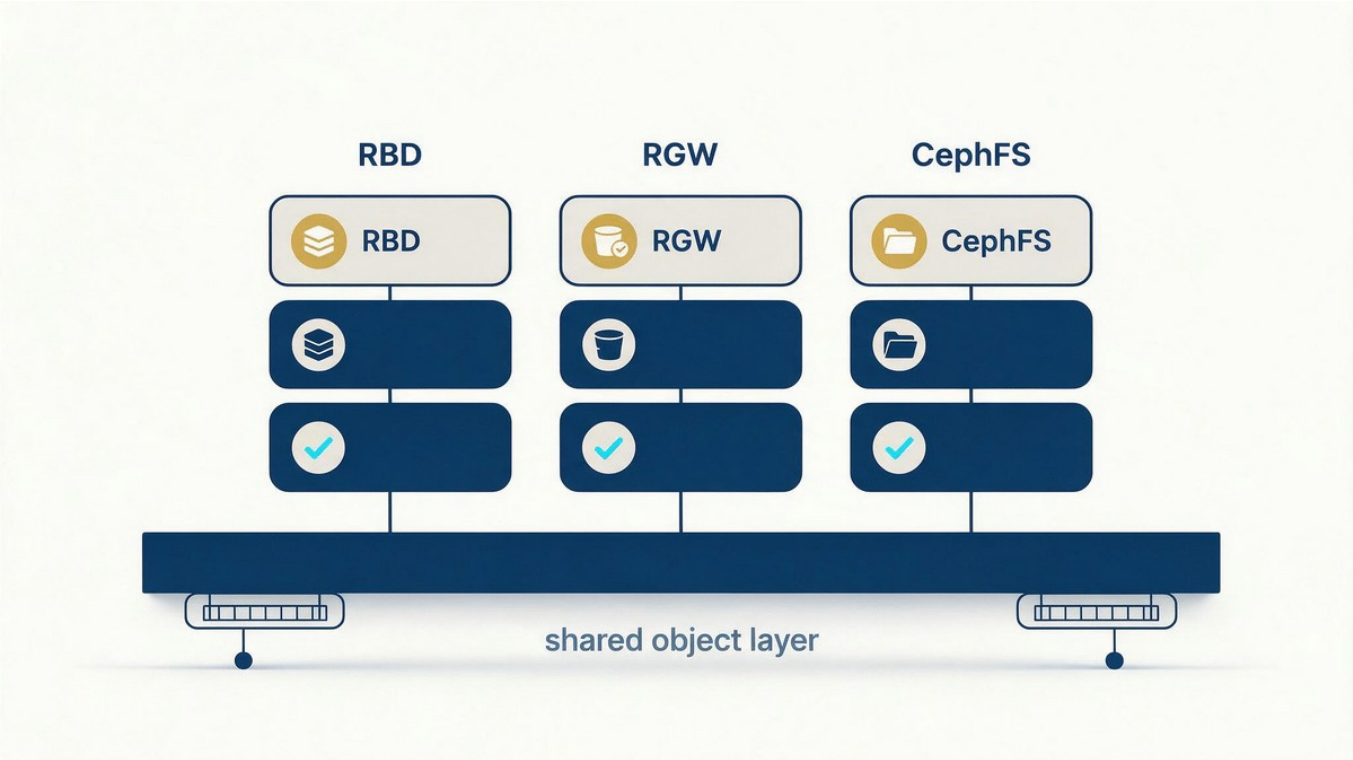
The one-line version: **let the workload's replication model choose the surface, price failure domains in replicas before you buy them, and treat every change as a migration with an honest rollback story**.

The three surfaces: RBD, RGW and CephFS

RBD gives you block devices: thin-provisioned images with snapshots and copy-on-write clones, typically consumed as Kubernetes PVCs or hypervisor disks. RGW gives you an S3 and Swift compatible object gateway with its own IAM surface, bucket policies, Object Lock and multi-site replication. CephFS gives you a shared POSIX filesystem on top of the same RADOS pools, with metadata servers (MDS) handling the directory tree so that many clients see one coherent namespace.

Choosing between them is a workload question, not a preference. Random read-write of a database volume is RBD. Key-addressed blobs with HTTP clients and lifecycle rules is RGW. Multiple machines

needing POSIX semantics on the same files is CephFS, with the operational cost that MDS becomes part of your daemon set (upgrades include it, key inventories include it). The three surfaces share the failure domains, the CRUSH topology and the upgrade machinery in the sections below, which is why they belong in one operational model instead of three products.



Three client surfaces, one object layer: the failure domains and the upgrade machinery are shared underneath.

Ceph as a Kubernetes storage backend

Before comparing backends, ask one question: does the application already replicate? Kafka, Cassandra, MongoDB and Redis handle replication at the application level, and for those a replicated storage backend doubles the cost of the same guarantee. Choosing the right storage backend for Kubernetes PVCs works the full comparison: Ceph RBD for large multi-node clusters needing strong HA and scalability, local NVMe-backed engines when pure performance wins, and the lighter replicated engines in between.

Where RBD is the answer, two production edges decide the design. First, the CSI driver's mirroring modes are not equal: journal-based mirroring is an alpha feature built on rbd-nbd, and if the CSI plugin pods restart it behaves like a node reboot, with every RBD mount point vanishing. Snapshot-based mirroring is the viable production choice today, at the price of a data-loss window between snapshots and a manual failover. Second, cross-zone RBD mirroring costs sixfold replication: three replicas in each zone with a minimum size of two. Who is using Ceph RBD mirroring for Kubernetes storage in production prices both, and proposes the cheaper topology when you only have two zones.

Failure domains, replication cost and CRUSH

CRUSH decides where every placement group's replicas land, and every rack, zone or region you declare as a failure domain multiplies the cost of surviving its loss. This is where designs get honest or expensive. Two-zone RBD mirroring means two independent three-replica clusters, six copies of everything.

The 2.5-zone layout keeps one cluster: three zones for the control plane (no sensitive user data there) and two for the storage plane, which sustains losing one zone at replication four with a minimum size of

two. Do not blend these figures; they are different architectures answering different constraints, and the right question is which failures you are actually buying protection from.

The same topology logic runs through device management. Orchestration-managed OSDs carry device classes, placement groups rebalance when topology changes, and every protected operation (draining, replacing, reclassifying) is bounded by recovery traffic, not by your maintenance window. Migrating unmanaged OSDs to managed OSDs is the worked example: data safety depends entirely on waiting for rebalancing to complete before moving on, and there is no shortcut to a fully managed CRUSH tree.

RGW in production: S3 semantics and the gateway fleet

RGW is not a blob endpoint with an S3 costume. It is an authorization surface: bucket policies with condition operators, explicit replication actions, Object Lock on versioned buckets, public-access-block semantics and presigned URLs with signed-header subsets. Clients that depend on exact S3 behavior feel every compatibility change.

Tentacle's RGW generation added `GetObjectAttributes`, truncates `LastModified` timestamps to the second for AWS compatibility (a timestamp-sensitive consumer can see values move backwards during an upgrade), and tightened presigned-PUT header handling after the advisory covered below.

The gateway fleet itself has moved on too. The Beast frontend can share one TCP port across several RGW instances on a host with `so_reuseport`, and cephadm can deploy a per-node HAProxy ingress concentrator with no keepalived and no virtual IP. Ingress design is real work with real risk: the zone endpoint list is committed in the period, so a load-balancing redesign belongs in its own change window after the version upgrade, never inside it. Upgrading a multi-site S3 platform from Squid to Tentacle is the full change plan.

Multi-site: realms, zones, periods and sync

RGW multi-site replicates buckets across zones through a realm whose topology lives in a committed period. The mental model that matters in production: replication health is a service-level property, not a cluster-level one. The cluster can be fully healthy while sync is behind, and only the per-zone sync status and an S3 acceptance suite will tell you.

The acceptance suite is the real deliverable of any multi-site change: versioned buckets, multipart uploads, server-side encryption, tags, copies, deletes and delete markers, bucket policies and lifecycle, written through the preferred zone and read back from every secondary, with ETags, bodies, metadata, tags and versions verified and replication lag measured against a defined window.

Upgrade sequencing has one multi-site-specific trap: a compatibility flag (`rgw_sigv4_insecure`) must be set before upgrading and cleared only once every cluster in the federation is patched. Cross-site data movement also appears on the block side as RBD mirroring; treat both as one concern: what happens to writes accepted on a site that loses its peer, and who reconciles after the partition heals.

The upgrade lifecycle: majors, patches and honest rollback

Three sequencing rules cover most of the risk. First, make everything orchestrator-managed before you plan anything else; unmanaged OSDs have no in-place adoption path and every upgrade gets harder while they exist. Second, upgrade in Ceph's order (monitors, managers, OSDs, MDS daemons where applicable, then gateways), watch the orchestrated status, and treat feature gates as separate, deliberate changes: the release requirement flag at the end, zone features only when every zone declares support.

Third, separate incompatible changes. The version upgrade, the ingress redesign and the key migration are three change windows, not one.

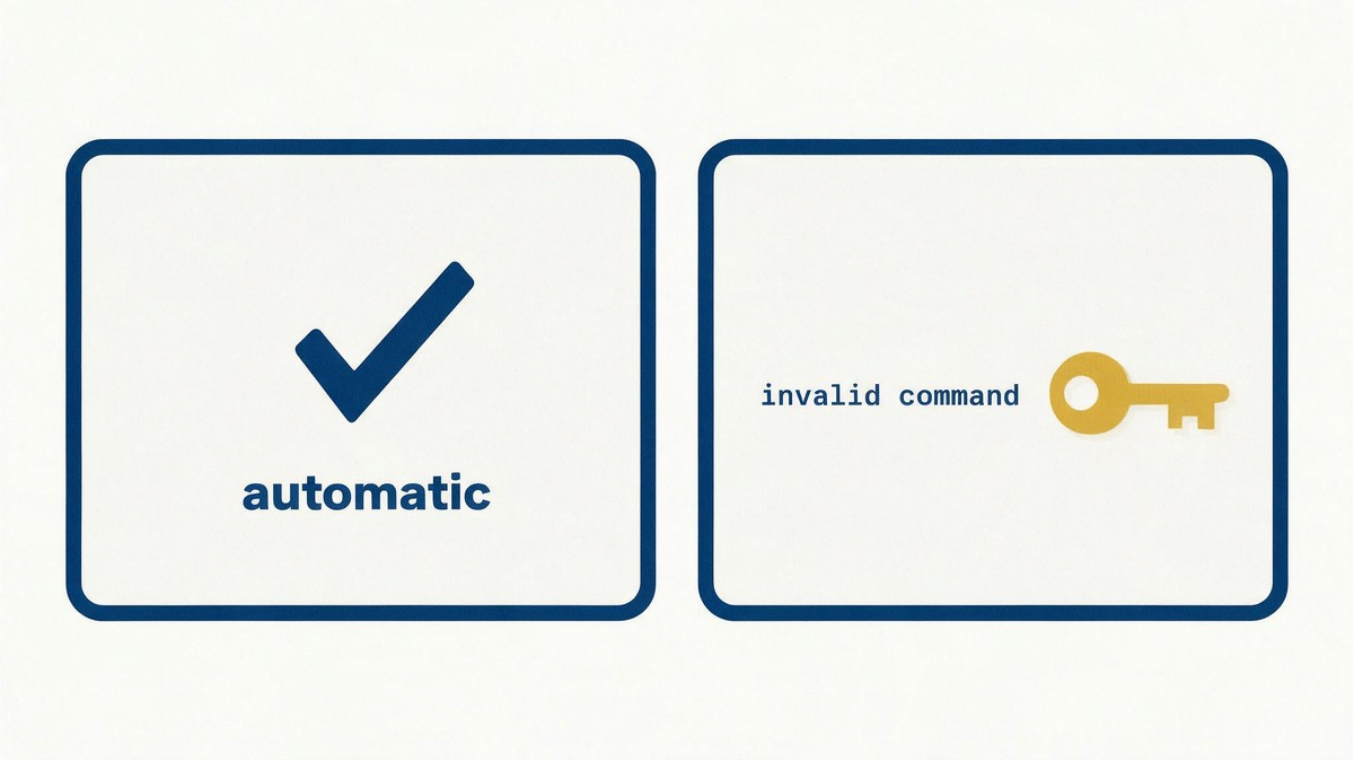
Rollback is the part documents get wrong. Stopping an orchestrated upgrade does not downgrade daemons back to the previous major. The honest rollback is isolation and rerouting: a canary gateway tier with its own client population and a known-good sibling to route back to. Plan it explicitly or discover the constraint during the incident. The complete Squid-to-Tentacle plan with its five rollout gates (cluster, topology, replication, S3, operations) lives in the upgrade post; the OSD migration that should precede it lives in the managed-OSD post.

Identity and security: CephX keys, ciphers and the CVE class

The August 2026 hotfixes (Tentacle 20.2.4 and Squid 19.2.6) patched four authentication and authorization advisories, and the remediation story is the most useful part. One advisory covered CephX's legacy key type using unauthenticated AES, with the new authenticated AES-256-CTS-HMAC-SHA384-192 key type as the fix.

The others covered gateway STS token escalation, an identity with monitor read rights being able to read the entire config-key store, and presigned PUT URLs accepting unsigned headers that mutated object metadata.

The critical operational fact, corrected after verification against the release source: **key rotation is not automatic on upgrade**. The orchestrator rotation command is disabled in this release line, and the only self-healing layer is short-lived rotating session keys, which expire on their own within hours once monitors run the fixed release with the new default cipher. Entity keys do not. The migration is therefore a designed change: inventory every identity first, because one client identity cannot hold both the legacy and the new key type at the same time (shared hypervisor or library keys need per-node identities created before cutover), rotate the monitor key as the one safe direct rotation, remember that an OSD carries its key in the bluestore label as well as the keyring, and remove the legacy cipher last. The full procedure, including the protective flags and their hazards (a protection flag left set can become an outage amplifier), is in the security update post. On the gateway side, planning the RGW account identity model covers tenants, accounts and IAM policy boundaries before you migrate either.



What the release notes promised and what the source answered when the orchestrator was asked to rotate the keys.

The production operating model

What a Ceph platform owes you, summarized from the sections above: a fully managed CRUSH tree, failure domains priced in replicas and money, S3 acceptance as a standing test suite rather than a migration checkbox. Upgrades ordered and gated with rollback defined as rerouting, an identity inventory that includes client keys and shared secrets.

And monitoring that reads what it measures (sync lag per zone, auth warnings during a cipher migration, recovery state during every protected operation). Backups are separate from replication: deletes and corruption propagate through replication, and only tested restores bound your recovery point. The recurring pattern across every incident in these posts: the cluster stays green while the service quietly degrades, so verification has to live at the service boundary.

FAQ

Which Ceph surface should a new workload use?

Match the access pattern. Block-level random read-write (database volumes, hypervisor disks, Kubernetes PVCs) belongs on RBD. Key-addressed objects with HTTP clients, lifecycle rules and S3 SDKs belongs on RGW. Multiple machines needing POSIX semantics on one shared namespace belongs on CephFS. If the application already replicates (Kafka, Cassandra, MongoDB, Redis), consider local storage instead of a replicated backend and avoid paying twice for the same guarantee.

Can you roll back a Ceph major version upgrade?

No. Stopping an orchestrated upgrade pauses it but does not downgrade daemons to the previous major release. The rollback is an architecture decision: a canary gateway or client tier with a known-good deployment to reroute back to. Design that before the upgrade window, because during an incident there is no downgrade path to take.

Is CephX key rotation automatic when I upgrade?

No. On the 20.2.x and 19.2.x lines the orchestrator key-rotation command is disabled, and entity keys (monitors, OSDs, MDS, gateways and all client keyrings) do not rotate themselves. Only short-lived rotating session keys expire on their own once the fixed release and the new cipher are active. Plan the key migration as its own change window with a complete identity inventory first.

When do you run the OSD release requirement flag?

Last. It is the explicit gate that declares the new release for the OSD map, and running it early removes options without buying anything. After every daemon is upgraded, after validation, as the final step of the cluster gate. Zone features follow the same rule: enable only when every zone declares support.

Is Ceph RBD mirroring production-ready for Kubernetes?

Snapshot-based mirroring is usable in production today with two accepted costs: a data-loss window between snapshots and a manual failover procedure. Journal-based mirroring is an alpha feature whose failover path is still maturing. More importantly, mirroring is a niche answer to a hard two-zone constraint, priced at sixfold replication; a 2.5-zone topology inside one cluster is usually the cheaper way to survive losing a zone.

Related posts

- Choosing the right storage backend for Kubernetes PVCs
- Who is using Ceph RBD mirroring for Kubernetes storage in production?
- Upgrading a Ceph multi-site S3 platform from Squid to Tentacle
- Migrating unmanaged OSDs to managed OSDs in Ceph Squid
- Ceph security update: four CVEs and key rotation
- Ceph RGW accounts in Tentacle: plan the identity model

If a Ceph platform is about to grow, change version or answer an audit, the Ceph: RBD, RGW and CephFS engagement covers exactly this ground. A data infrastructure audit or a focused Expert Call is the usual starting point.

PDF snapshot of the public article. The online version is authoritative; links remain clickable.