Object Storage for Data Platforms at Scale

Julien Laurenceau | Published 2026-09-30 | Updated 2026-10-21

https://pepitedata.com/object-storage-for-data-platforms-at-scale/

PepiteData | LinkedIn: Julien Laurenceau | X: @Julien_Lau



A terabyte of small objects and a terabyte of large objects ask very different things of a storage platform. The payload volume can match while request counts, metadata, listing cost and maintenance work differ by orders of magnitude. A capacity plan that stops at usable terabytes misses those limits.

I size object storage against four budgets: bytes, object metadata, requests and maintenance time. This guide explains what to measure, which limits belong to a particular implementation, and how to test an on-premises or hybrid platform before committing a data lake to it.

Updated October 21, 2026. This revision separates S3 API behavior from backend layout, corrects the capacity arithmetic, and replaces universal object-count thresholds with a workload-based sizing method. Numerical examples below are calculations, not customer measurements.

S3 compatibility does not specify the backend

The S3 API does not tell you how an object store keeps metadata, places data or repairs a failed drive. Amazon S3, Ceph RGW and filesystem-backed MinIO deployments expose similar operations through different implementations. A filesystem inode limit, a bucket-index bottleneck and a managed-service request limit require different remedies.

Implementation	Relevant mechanism	Sizing consequence
Amazon S3 general-purpose buckets	Flat key namespace; slash-delimited prefixes provide logical grouping	Measure requests, listing behavior, storage classes and cost. Do not apply local XFS inode arithmetic.
Filesystem-backed MinIO erasure deployments	Object paths, per-object metadata such as `xl.meta`, data layout and inline behavior depend on the release	Measure allocated space and filesystem entries on the actual backend and erasure geometry.
Ceph RGW	Bucket indexes and object data live in RADOS; index sharding is a separate mechanism	Inspect index shards, metadata placement, OSD behavior and recovery under load.

A shared API does not imply a shared metadata layout or capacity limit.

The distinctions come from AWS's object-key documentation, MinIO's metadata-format design and Ceph's RGW data layout. In particular, a slash in an S3 key is not proof of a physical directory.

Consistency also needs precise wording. Amazon S3 documents strong read-after-write consistency for object writes and deletes, including subsequent GET and LIST requests, and atomic updates to a single key. That does not provide a transaction across several keys. A table format or application must supply its own commit protocol where one is needed; verify the corresponding guarantees on any compatible service.

Object count is an independent sizing dimension

Object count changes the work required to store and operate a fixed payload volume. One TiB divided into 1 MiB objects contains 1,048,576 objects; divided into 1 KiB objects, it contains 1,073,741,824. That is 1,024 times as many objects before accounting for versions, delete markers or incomplete uploads. It does not imply a fixed 1,024-fold latency or space penalty.

Uniform object size	Objects in 1 TiB	Logical payload
128 MiB	8,192	1 TiB
1 MiB	1,048,576	1 TiB
1 KiB	1,073,741,824	1 TiB

Illustrative arithmetic with binary units: 1 TiB = 2^{40} bytes. No metadata, parity or filesystem allocation is included.

Real workloads need a size histogram, not just an average. A large byte-weighted average can hide a long tail of tiny objects that dominates request counts. Record counts and bytes by size band, plus creation rate, overwrite rate, retention and the number of retained versions. Format comparisons should include object count alongside compression ratio and read throughput; file count versus file size develops that distinction.

For a filesystem-backed MinIO deployment, measure metadata and allocation on the release you intend to run. Inline payloads, metadata size, allocation units and erasure-set width can change small-object overhead. The small-file erasure-coding analysis is a backend-specific investigation, not a universal S3 multiplier.

Keep the accounting units explicit. If a tested layout uses two inode-bearing entries per object on each of 15 participating drives, 100,000 objects imply 3,000,000 entries across that set, excluding shared directories and other overhead. The metadata-file copies alone would number 1,500,000. Neither count means every drive in a multi-set cluster holds every object. Check the actual placement and filesystem

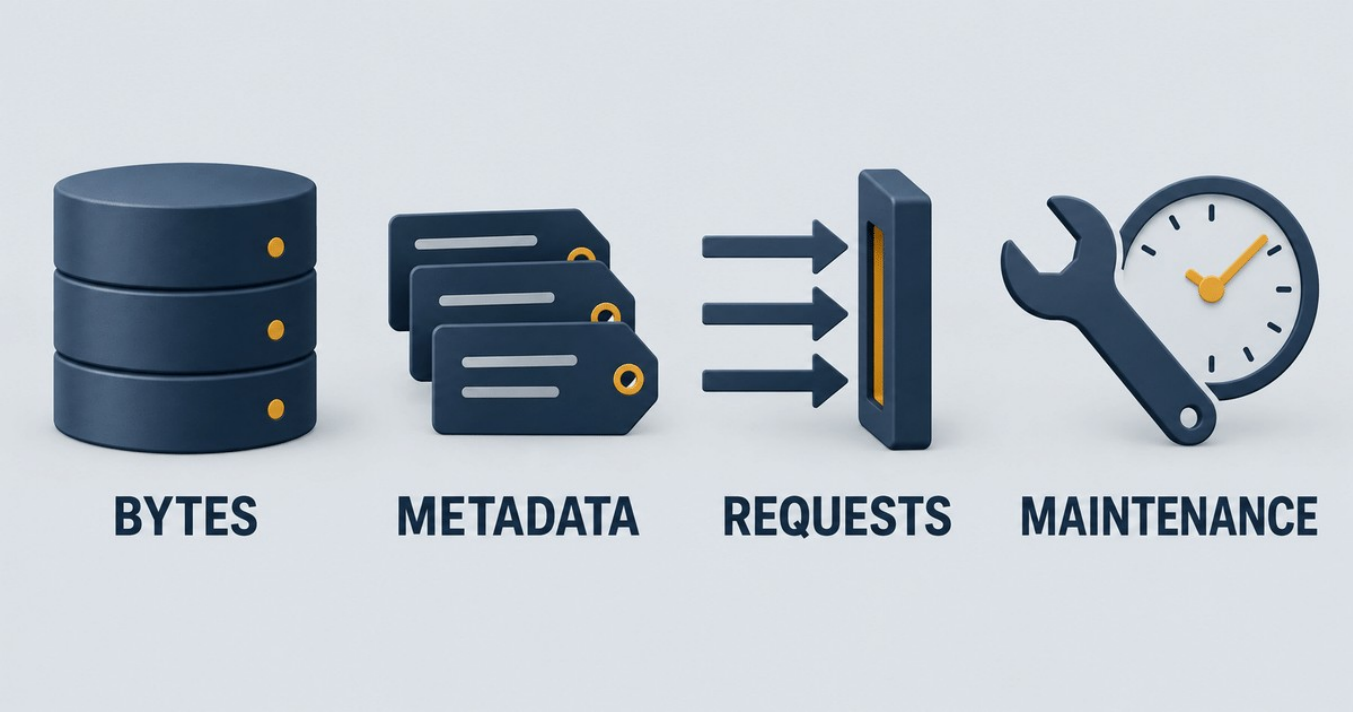
geometry; a fixed inode ceiling cannot be inferred from drive size alone.

Build a four-budget capacity worksheet

A useful capacity plan records payload bytes, metadata population, request demand and maintenance deadlines separately. Each budget needs a forecast and a measured acceptance test on representative hardware or the chosen service. Spare capacity means enough room to handle growth and complete recovery while serving the workload, rather than merely enough free space to accept today's writes.

Budget	Inputs to record	Acceptance test
Bytes	Size histogram, retention, current and noncurrent payloads, protection scheme, growth	Measure allocated raw bytes per logical byte by size band, with operating reserve.
Object metadata	Keys, versions, delete markers, incomplete multipart state; backend entries or index shards	Populate forecast counts and skew. Inspect metadata latency, allocation and per-drive or per-shard headroom.
Requests	PUT, GET, HEAD, LIST, DELETE, tags, concurrency, bursts and retries	Run the real operation mix and working set. Record p50, p99, errors and retry amplification.
Maintenance time	Expiration lag, integrity checks, repair, replication backlog and restore targets	Exercise each operation under ingest and a controlled failure. Check its deadline and backlog clearance.

A sizing worksheet to fill with workload measurements. These are test requirements, not benchmark results.



Check all four budgets. Passing a byte-capacity test does not establish metadata, request or recovery headroom.

For a simple append-only workload, one million new objects per day retained for 30 days gives 30 million current objects at steady state. Updates, retained old versions, late expiration and uploads left incomplete require additional accounting. Do not multiply current keys by an arbitrary version factor if the application can supply the actual version creation rate and retention policy.

I want the test report to identify the software build, protection geometry, filesystem or index configuration, hardware, occupancy, object distribution and client concurrency. A result without those details is hard to reproduce and easy to misapply.

Prefix design depends on the implementation

A prefix layout should support the application's access patterns and the backend's metadata behavior. There is no S3-wide rule limiting a leaf prefix to 10,000 objects. Date partitions and hash fan-out can be useful in a filesystem-backed deployment, while Ceph RGW bucket-index sharding and Amazon S3's logical prefixes follow different scaling mechanisms.

On MinIO with a filesystem backend, test realistic directory occupancy, name lengths and hot prefixes. Use the MinIO/XFS prefix-design guide as a starting point for that investigation, not as an AWS limit. On RGW, inspect the actual shard population and Ceph's resharding configuration and behavior.

Avoid making a full-bucket listing part of every request path. Give interactive callers a direct key, catalog or manifest when the application can maintain one. Keep paginated LIST operations for discovery, reconciliation and workflows that need them, and benchmark those operations explicitly. A separate catalog creates its own consistency problem: test failed uploads, missing rows, orphaned objects and replay after an outage.

Measure maintenance clocks separately

A namespace scan, a checksum verification pass, a disk rebuild and a replication catch-up are different operations. None is a substitute for an application recovery test. Record what each process examines, how it detects missing or corrupt data, and whether its backlog clears under normal ingest and the failures your service promises to tolerate.

MinIO's archived scanner source includes sampled healing decisions. A sampling probability is not a guarantee that every object has been checked after a fixed number of cycles. Scanner behavior must be tied to a particular release and configuration; it cannot establish a recovery SLA for every S3-compatible engine.

Measure the oldest pending item as well as total backlog. A large queue that drains predictably may be acceptable; a small queue containing objects that never converge is a correctness defect. Include versioned and tiered objects in the test set. The MinIO tiering and metadata-healing investigation shows why testing only ordinary hot objects leaves gaps.



A capacity purchase does not establish a maintenance deadline. Measure scan, repair and catch-up separately under load.

Versioning changes retention and cleanup

Versioning requires separate forecasts for current data, noncurrent data and delete markers. A logical deletion can hide a key while leaving older payloads stored and recoverable. Lifecycle policies therefore need explicit behavior for old versions, marker cleanup and incomplete multipart uploads, with observed completion times rather than an assumption that an expiration date means immediate physical removal.

In Amazon S3 versioning-enabled buckets, current-version expiration can create a delete marker; noncurrent-version expiration is a separate action. AWS also documents a possible delay between eligibility and removal in its expiration rules. Validate the semantics on the chosen compatible engine, including interactions with retention locks and replication. Configure and test abandoned multipart cleanup separately from completed-object expiration.

Test the features you will actually enable together. Tags, replication, versioning and tiering all add operations or state, but they do not share one universal cost multiplier. Disable features you do not need, and keep the ones required by the application's recovery or compliance requirements in the benchmark.

Choose the storage unit for the workload

Object storage fits immutable payloads and large sequential reads well, while record-level mutation and indexed queries often need a database. The decision depends on access patterns, consistency, object size and operational cost, not a universal row-count cutoff. A hybrid design can use a database for discovery and object storage for payloads, provided failures between the two systems are handled explicitly.

Workload	Candidate design	Tradeoff to test
Small records, frequent updates, indexed filters	A relational or distributed key-value database matched to the query model	Indexes, replication, compaction, hot partitions and transaction requirements
Immutable blobs, direct-key reads	Object storage, optionally with a catalog	Request cost, metadata growth, tail latency and catalog reconciliation

Workload	Candidate design	Tradeoff to test
Analytics or training over many small records	Batch records into suitable columnar files or indexed shards	Read amplification, random access, rewrite cost and visibility delay

Choose a unit of storage that matches the unit of access. Each design still needs workload testing.

The MinIO versus NoSQL discussion and small-file workload analysis explain the mismatch in more detail. PostgreSQL does not become unsuitable at a fixed number of rows, and a distributed database does not become suitable merely because a workload is large.

Batching reduces object count only by changing the storage unit. Parquet can suit analytical scans; indexed archives or tensor-oriented formats can suit training readers. The .smol format discussion examines the latter case. Measure the whole read path before choosing a format. Larger shards can increase point-read amplification, delay visibility, complicate per-record deletion and require expensive rewrites.

Protection geometry changes usable capacity

For an erasure code with k data shards and m parity shards, ideal payload efficiency is $k / (k + m)$. That calculation excludes metadata, padding, allocation and reserve space. The placement of shards across failure domains determines which failures the platform can survive; the ratio alone does not establish durability or recovery speed.

Data + parity shards	Ideal payload efficiency	Ideal raw / payload
4 + 2	66.7%	1.5x
10 + 2	83.3%	1.2x
8 + 8	50.0%	2.0x

Mathematical examples for sufficiently large payloads. These are not recommended configurations or measured usable-capacity figures.

Ceph's erasure-coding documentation uses the explicit `k` and `m` model. Product configuration syntax is not interchangeable: a setting expressing parity count must not be read as a data-plus-parity pair. Check the actual deployment geometry before using any capacity formula.

HDD versus SSD is a workload decision. HDDs can suit large sequential capacity workloads; metadata-heavy small operations and rebuild contention can favor SSDs. Test foreground p99 latency while a failed device is being rebuilt at realistic occupancy. Include power, network, spares, raw capacity and operator time in the comparison. The HDD cost analysis for MinIO is a useful investigation path, not a universal limit on erasure-set width.

Choose a release channel you can operate

A production storage choice needs a maintained release channel, a documented upgrade path and tested recovery procedures. API compatibility does not establish security coverage or support for every on-disk state. Record the exact binary and source revision, map relevant advisories to that build, and rehearse upgrades with representative versioned, replicated and tiered data before changing production.

On October 21, 2026, the MinIO community repository displays an archive date of April 25, 2026. The independent PGSTY Silo fork has an October 18, 2026 release; its notes include fixes and remaining

limitations. A recent release is evidence of maintenance activity, not proof that every inherited issue or deployment mode is covered.

Use the MinIO community risk assessment, community-edition and licensing discussion, and migration guide as issue inventories. Recheck their time-sensitive conclusions against current release notes and advisories. Select among a supported product, a maintained community distribution or a migration based on tested requirements and the team's ability to operate it.

The same evaluation applies beyond MinIO. Assign ownership for patching, capacity forecasts, lifecycle verification, recovery drills and on-call diagnosis. The on-premises operating-model discussion covers the staffing side of that choice.

What to validate before production

A production decision should be backed by a representative population test and a recovery exercise. Test the forecast object distribution at realistic occupancy, with the intended features enabled and sufficient time for maintenance behavior to appear. Acceptance criteria should cover latency, correctness, cleanup and recovery, including the period when a failed component is being rebuilt.

1. Capture the payload histogram, key distribution, version creation rate and operation mix.
2. Populate representative counts, occupancy and hot prefixes. Verify the allocated-space and metadata forecast.
3. Run sustained ingest and reads with listing, lifecycle, tagging and replication where required.
4. Inject a controlled failure in an isolated test environment. Measure detection, degraded service, repair and backlog clearance.
5. Restore application-visible data and reconcile the catalog, versions and permissions. Record gaps and repeat the failing cases.

FAQ

Object storage sizing depends on both the workload and the implementation. Object size, retained versions, request mix, metadata behavior and recovery targets determine the useful operating range. The answers below distinguish API guarantees from backend-specific limits; none supplies a universal object-count ceiling or replaces a representative load and failure test.

Can object storage hold billions of small objects?

Some services and deployments can, but count alone does not establish suitability. Measure per-object metadata, request demand, cost and maintenance behavior on the chosen backend. Compare a database for mutable indexed records and batched files for analytical workloads. Do not transfer a MinIO filesystem limit to Amazon S3 or Ceph RGW.

Are S3 prefixes physical directories?

Not as an API rule. Amazon S3 general-purpose buckets use a flat key namespace with logical prefixes. A filesystem-backed engine may map keys into directories, while Ceph RGW uses RADOS objects and bucket indexes. Prefix limits and tuning advice must identify the implementation they apply to.

Does erasure coding guarantee a fixed space multiplier?

No. The ideal raw-to-payload ratio is (k + m) / k for k data shards and m parity shards. Actual consumption also includes metadata, padding, allocation and reserve space. Small-object layouts can depart substantially from the ideal ratio, so measure allocated bytes by object-size band on the

deployed build.

Does a completed scan prove recovery readiness?

No. A scan can traverse metadata without reading and validating every payload, and sampled checks do not guarantee full coverage after a fixed number of cycles. Measure integrity verification, repair, replication catch-up and application restore separately, including their behavior under continued ingest and component failure.

Does lifecycle expiration remove all object versions?

Not automatically. In an Amazon S3 versioning-enabled bucket, current-version expiration can add a delete marker while noncurrent versions remain. Noncurrent expiration and marker cleanup need appropriate policies, and physical removal can lag eligibility. Verify compatible-engine behavior and handle incomplete multipart uploads separately.

Should on-premises object storage always use SSDs?

No. HDDs can suit large sequential workloads, while SSDs can improve metadata-heavy operations and reduce latency under contention. Compare the complete platform cost and measure foreground latency during recovery at realistic occupancy. Do not assume one medium requires a universal erasure-set width.

Related object-storage guides

This pillar links to focused investigations of object count, metadata and operations. Read the implementation-specific articles as diagnostic guides, then verify their release-sensitive details against the system you run. Their numerical cases describe particular assumptions or observations, rather than limits that apply to every S3-compatible platform.

- MinIO versus NoSQL for small-record workloads
- MinIO and lots of small files
- Small-file overhead in erasure-coded MinIO
- MinIO/XFS inode accounting and prefix design
- File count versus file size in data formats
- HDD cost and operational tradeoffs in MinIO
- Training formats and the small-file read path
- Tiering and metadata-healing failure modes
- MinIO community release-channel risks
- MinIO community edition and licensing
- Migration planning after maintenance mode
- The operating model behind an on-premises lakehouse

Sources are linked next to the claims they support. Product names identify the implementations discussed; no vendor endorsement is implied.

If you need to validate a sizing model, metadata bottleneck or release choice, I cover those questions in MinIO, S3 and object-storage engagements. For a broader platform review, see the data infrastructure

audit or book a free 15-minute intro call. A focused technical question can also fit a standalone Expert Call.

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