

MinIO Site Replication: Sync Mode Will Not Save Your RPO

Julien Laurenceau | Published 2026-08-03 | Updated 2026-10-22

<https://pepitedata.com/minio-site-replication-rto-rpo/>

PepiteData | LinkedIn: Julien Laurenceau | X: @Julien_Lau



MinIO synchronous site replication is not a cross-site commit quorum. A successful source PUT does not, by itself, prove that a recoverable copy exists at the peer. That distinction matters when the source disappears before replication completes: changing the replication mode cannot reconstruct bytes that never reached a surviving site.

I would not accept a zero-RPO claim based on the sync flag alone. I would ask for the exact server and client builds, a peer-down write test and a recovery test that reconciles acknowledged versions against the surviving destination. Healthy-path replication lag and worst-case data protection answer different questions.

Revised October 22, 2026. This version removes unsourced latency and throughput ranges, corrects the description of persisted retry state, and separates recovery time from recovery-point guarantees. The code reference is MinIO Community RELEASE.2025-09-07T16-13-09Z; current AIStor documentation is identified separately. Forks and other releases require their own verification.

What does synchronous site replication actually guarantee?

Synchronous site replication attempts the remote operation before completing the originating write, but it does not make the peer's availability a universal condition for acknowledging that write. The current AIStor site-replication documentation explicitly permits a successful PUT even when replication fails.

Treat this as a site-replication behavior, not a guarantee shared by every feature called synchronous replication.

The [AIStor site-replication reference](#) states this failure behavior and recommends asynchronous site replication. The separate [bucket-replication reference](#) describes a different synchronous success condition. Do not substitute one page's contract for the other. Confirm the feature, release and configuration on the deployment under review.

For the pinned Community release, the [replication implementation](#) calls the replication function inline for a synchronous decision. That function does not return a replication error to its caller. The [PUT handler](#) schedules replication after the local write rather than acquiring a cross-site commit quorum. This code reading supports the distinction; it is not a substitute for a fault-injection test.

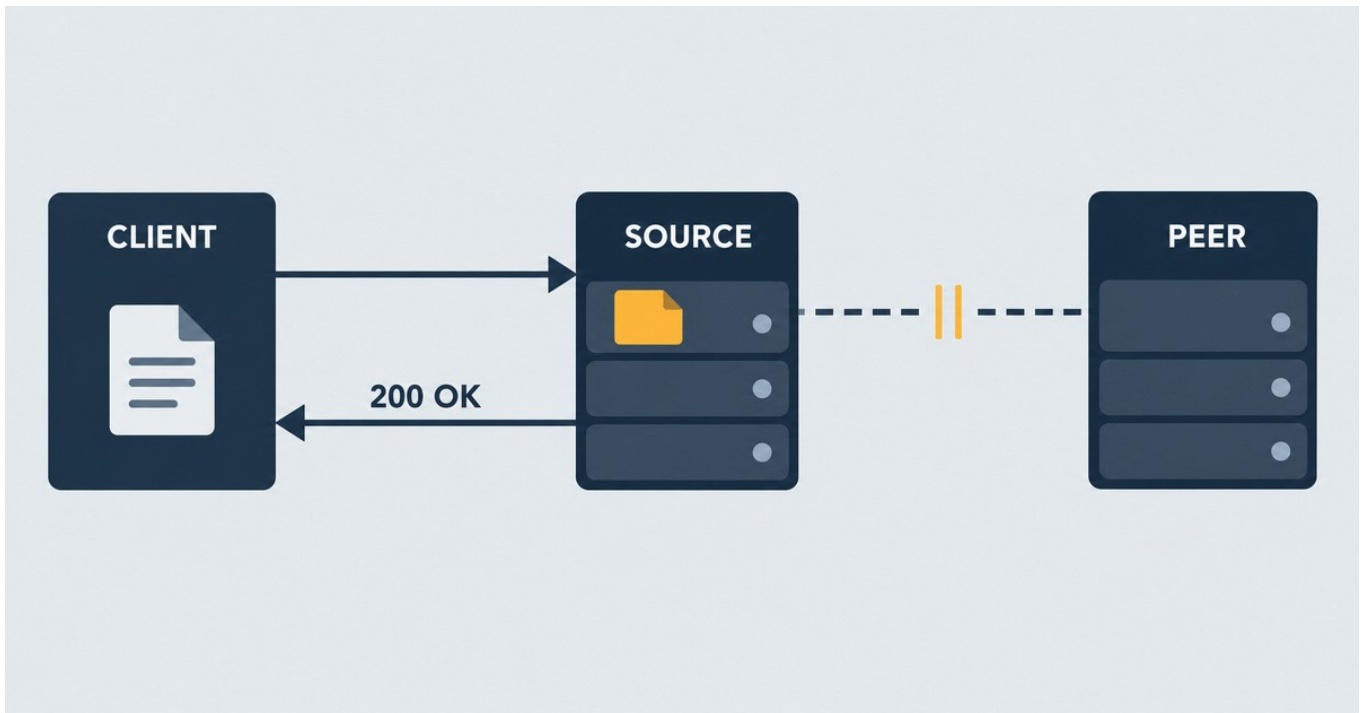
Separate local durability from remote recovery

Local erasure-code durability and remote recovery protect against different failures. A locally durable object can survive drive failures inside its source cluster while still having no copy at another site. Site replication connects independent deployments; the legacy site-replication description does not establish a distributed transaction across them. Recovery requirements must specify which site and which failure are in scope.

The pinned [Community site-replication description](#) lists object, bucket and identity-related changes replicated between independent sites. It is broader than copying object payloads, but it is not a backup of every external dependency. Inventory keys, identity providers, application state and client routing separately.

Condition	What may remain available	What to prove
Peer unavailable, source healthy	Local service and acknowledged writes	Which versions are absent from the peer and how they will catch up
Source temporarily unavailable	Previously replicated versions at the peer	Application behavior for missing or stale versions and source recovery time
Source permanently lost before catch-up	Only copies already elsewhere, plus independent recovery sources	How missing writes are identified and reconstructed

A local acknowledgment and a remote recoverable copy are separate facts.



Illustrative peer-down path: the acknowledgment returns from the source; the broken replication path does not contain a remote copy.

Retries are source-side recovery work

Pending and failed replication state belongs to the source-side recovery process; it is not an independent inventory held by a peer that never received the object. The pinned implementation also persists retry entries to disk. Describing all replication work as only an object-metadata flag misses that persisted state, although neither form creates an off-site payload copy.

In the pinned [replication source](#), the MRF path saves entries, limits retries and leaves further work for scanner-driven recovery. I would verify the corresponding implementation and observable signals in the actual build rather than publish a universal retry count, worker count or recovery cadence.

If the source is irretrievably lost before an object reaches any surviving copy, local metadata and retry files cannot recover its content. A surviving peer alone cannot list writes it was never told about. An external producer journal, ingestion manifest or independently retained backup can change that assessment. Name that source of truth explicitly; do not assume it exists.

PENDING IS NOT A BACKUP.



Pending work records an obligation to copy. It does not establish that the copy has been made.

Scanner time is not an RPO guarantee

Scanner revisit time affects when stalled replication work may be rediscovered, but it is not an upper bound on data loss or completion time. A rediscovered object still needs a healthy destination, successful authorization and transfer capacity. Repeated failure, sustained overload or loss of the source can prevent convergence regardless of how quickly the scanner walks the namespace.

I would measure discovery delay, queue wait, transfer time and verification separately. A faster scanner setting is a tuning choice to test against foreground I/O, not a promise that every object is protected within one sweep. An explicit resynchronization can initiate recovery work; it cannot recreate a missing source version from nothing.

A simple catch-up calculation can help expose insufficient headroom. With a fixed backlog of 2 TiB, continuing replicated writes of 100 MiB/s and a sustained completion rate of 150 MiB/s, net drain is 50 MiB/s. The ideal drain time is about 11.7 hours. This synthetic byte-rate example excludes per-object work, retries and contention; equal or lower completion capacity means the backlog does not drain under that continuing load.

```
ideal_catch_up_time = backlog_bytes / (completion_bytes_per_second - incoming_bytes_per_second)
```

Condition: sustained completion rate exceeds incoming work.

Choose sync for a measured requirement

Synchronous site replication can reduce the healthy-path interval before a peer receives a write, at a workload-dependent latency cost. That is a real potential benefit, but it does not establish zero RPO during peer failure. I would start with asynchronous replication and require a measured use case before accepting the extra coupling of writes to remote attempts.

The earlier blanket claim that sync cannot improve any relevant risk was too broad. A smaller healthy-state single-copy window can matter if the source fails while the peer is healthy. The decision is whether that benefit meets an explicit requirement at the measured cost, while the peer-down contract remains acceptable. There is no universal threefold or fivefold throughput penalty.

Measure latency percentiles and completed operations at the intended object sizes and concurrency. Test timeout behavior too. Read proxying can hide some missing-object reads while another peer remains reachable; it does not produce a durable replica or repair a permanently lost source. A load balancer also does not fence clients that can bypass it and write directly.

Test the acknowledgment-to-recovery path

A useful site-replication DR test records acknowledged versions independently, injects the intended failure and checks which versions the recovery path actually serves. It must also test client routing, credentials and any write-fencing mechanism. A healthy cluster dashboard or an empty queue is supporting evidence, not proof that the application can recover every acknowledged operation within its objectives.

1. Use isolated test buckets and approved failure injection. Record server builds, client builds, topology, modes and dependencies.
2. Keep an independent manifest of successful test writes, including bucket, key, version identity and a content check.
3. Interrupt peer connectivity while writing. Record client responses, latency and remote availability of each test version.
4. Restore connectivity under continuing load. Measure backlog age, successful completion, errors and residual mismatches until reconciliation is complete.
5. Simulate source unavailability before catch-up without destroying production data. Verify what the surviving site serves and what requires an external recovery source.
6. Test controlled failover and failback. Prove fencing of every write path, then reconcile conflicting versions before reopening traffic.

DR claim	Evidence needed	Insufficient evidence
RPO met for a specified failure	Acknowledged-write manifest compared with recoverable versions	Sync enabled or low average replication lag
RTO met	Time until representative application requests succeed after failover	A healthy storage endpoint alone
Catch-up completed	Reconciled versions, errors resolved and sustained drain verified	A falling byte counter or one empty queue snapshot
Single writer enforced	Old-site writes denied through all reachable access paths	A DNS change or load-balancer routing preference

Write objectives against the tested failure and the application-visible result.

FAQ

MinIO site replication requires separate answers for write acknowledgment, remote copy completion and recovery after failure. The replication mode alone does not answer all three. These distinctions apply when reviewing a DR claim, while release-sensitive implementation details must be checked against the selected product and build rather than inherited from a similarly named feature.

Does synchronous site replication guarantee zero RPO?

No. A sync setting alone does not establish that every acknowledged version is recoverable after the specified failure. Verify the feature's acknowledgment behavior and compare acknowledged writes

against surviving copies during a peer-down and source-loss test.

Is retry state stored only in object metadata?

No. The pinned Community implementation also persists MRF retry entries. Those source-side records do not create an independent remote copy of the object payload. Verify persistence and recovery behavior in the deployed release.

Does one scanner sweep bound replication recovery?

No. Discovery is only one step. Destination health, permissions, transfer capacity and repeated failures can extend recovery or prevent convergence. Measure completed replication and reconcile versions rather than equate scan duration with RPO.

Does changing DNS prevent conflicting writes?

Not by itself. Clients may cache an endpoint or reach the old site directly. A single-writer failover needs effective fencing across all write paths, plus a tested reconciliation and fallback procedure.

Related resilience investigations

This article focuses on the acknowledgment and recovery contract of site replication. The related investigations cover test design and specific replication failures. Keep observed incidents scoped to their release and workload; neither a fix in one fork nor an old implementation constant establishes the behavior of every MinIO-derived product.

- [Data platform resilience: RTO, RPO and replication](#)
- [Test the complete recovery path](#)
- [MinIO replication drop accounting and monitoring](#)
- [The delete-marker replication loop investigation](#)
- [The Silo delete-marker convergence fix](#)

Primary documentation and pinned source checked October 22, 2026. This article does not report a new deployment benchmark. Numerical catch-up examples and images are illustrative. No customer data is included; no vendor affiliation or endorsement is implied.

I review these assumptions through a [Resilience and Disaster Recovery Assessment](#) or a [Flash Architecture Review](#). [Book a free 15-minute intro call](#) to discuss the failure scenarios and evidence you need.

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