SCALITY

WHITE PAPER

Scality ADI

(AI Data Infrastructure)



A technical deep dive into the architecture and internals of the platform

Built for GPU hunger, deep cold, and everything in between.

Version 2.0

Scality · 2026

Table of contents

I. Abstract 5
II. Executive Summary 5
III. The Storage Challenge ADI Addresses 7
- AI and HPC have made throughput and latency a storage problem again 7
- Unstructured data has reached exabyte scale: and so has its metadata 7
- Ransomware specifically targets the recovery path 8
- Sovereignty and economics now dictate where data lives 8

IV. Scality ADI Architecture 9
- Design principles 9
- Deployment components 9
- A single namespace across four temperatures 11
- Software stack and request flow 12
- The distributed storage engine 13
- Inherently immutable by design 13
  - S3 object storage 14
  - SOFS scale-out file system 14

V. Data Protection and Durability 15
- Replication and class of service 15
- Reed-Solomon erasure coding 16
- Self-healing and fast rebuilds 17
- Data integrity: end-to-end checksums 18

VI. The Metadata Layer: Built for Billions 19
- Topology 19
- The RAFT session: the unit of consistency 19
- Scaling concurrency within a cluster 20
- Online bucket migration 20
- Bucket sharding by prefix 20
- A metadata store built for billions of objects 21
  - Compaction on your schedule 21
  - Faster listings in versioned buckets 22
  - The metadata storage engine 22

VII. CORE5: Cyber-Resilience by Design 22
- Level 1: API-level resilience 23
  - S3 object versioning 24
  - S3 lifecycle 24
  - S3 Object Lock 24
  - SOFS recycle bin and volume protection 25
- Level 2: Data-level resilience 25
  - IAM: zero-trust by default 25

TLS: encryption in transit 26
S3 server-side encryption at rest 26
LUKS: full-disk encryption 26
Federation, single sign-on, and operational hardening 27
27
Level 3, Storage-level resilience 28
Level 4: Geographic-level resilience 28
S3 cross-region replication 29
SOFS multi-site asynchronous replication 29
Level 5: Architecture-level resilience 30
VIII. Performance at Scale 30
Field-proven throughput and ingest 30
Predictable read latency under partial failure 31
Seamless drive operations 32
SOFS write buffering and inline compression 32
Multi-tenant performance governance 33
Scaling without migration or downtime 33
IX. Workload Tiering, Lifecycle, and Media 33
The four temperatures in depth 33
Feeding the GPU directly 34
GPU-Direct tier: ultra-low-latency RDMA 34
Hot tier: S3 over RDMA 35
Media portfolio and cost profiles 35
XDM: lifecycle transition and restore 37
Sustainability by design 39
X. AICONNECT: Enabling AI Workflows 40
CSI: ADI as a Kubernetes volume 41
COSI: S3 buckets as Kubernetes resources 41
S3 bucket notifications for pipelines 42
The hierarchy of reasoning needs 43
XI. Guardian: Autonomous Operations 43
Three layers of monitoring 43
Guardian: AI-powered storage intelligence 44
How Guardian is built 44
Agentic operability over MCP 45
XII. Deployment and Reference Architectures 45
XIII. ADI in Context: Where It Fits, and Where It Doesn’t 46
Versus dedicated parallel file systems 46
Versus hyperscaler object storage 47
Versus the two-tier status quo 48
Where ADI is not the answer 48

XIV. Conclusion 49
Further resources 49

I. Abstract

Scality ADI, AI Data Infrastructure, is a software-defined storage platform that presents object (S3) and file (POSIX) data through a single namespace spanning four distinct performance-and-economics tiers: GPU-Direct, Hot, Warm, and Cold. Built on Scality's field-proven distributed storage engine, ADI is designed for a world in which a single organization must simultaneously feed GPU clusters at sub-50-microsecond latency, serve sovereign cloud and backup workloads at scale, and retain petabytes of archive at near-zero power, without operating four separate storage products to do it.

This paper is a technical deep dive intended for architects and engineers who evaluate storage the way they evaluate any other piece of critical infrastructure: by understanding how it actually works. It describes the deployment model and software stack; the distributed, inherently immutable storage engine and its keyspace; the flash-backed metadata layer and how it scales to hundreds of billions of objects through RAFT consensus, bucket sharding, and online migration; the data-protection mechanisms (replication and Reed-Solomon erasure coding) and self-healing; the CORE5 cyber-resilience model that places immutability in the engine rather than in a policy layer; the performance behavior of the system under sustained, production-grade load at scale; and the autonomous-operations and AI-integration capabilities, GPU-Direct data paths, Kubernetes CSI/COSI, and the Guardian operations agent, that define the platform's name.

By the end, the reader should understand not only what ADI does, but why its architecture produces the durability, security, and scaling characteristics it claims, and where the honest engineering trade-offs lie.

II. Executive Summary

For three decades, enterprise storage was organized around a single question: structured or unstructured. The architectures that answered it, scale-up NAS, monolithic SAN, and first-generation object stores, were each tuned for one temperature of data and one access pattern. That model is breaking under the weight of AI.

Modern data estates are no longer one temperature. A training pipeline needs GPU-saturating throughput and KV cache at memory-adjacent latency; the resulting models and datasets need low-latency high-throughput object storage; the broader business

needs cost-efficient capacity for sovereign cloud and backup; and compliance needs decades of immutable archive. Today these requirements are typically met by stitching together separate products, each with its own namespace, its own data-protection scheme, its own security posture, and its own operational runbook. Every boundary between them becomes a migration, a copy, a policy gap, and a place where data can be lost or leaked.

Scality ADI collapses those boundaries into one platform. Its central architectural insight is that a single distributed storage engine and a single namespace can serve all four temperatures, with data moving between them by policy rather than by migration project. That is not a marketing claim about convenience; it is a structural property with concrete consequences for cost, security, and operational risk:

- **Scale capacity, performance, and metadata independently.** Add storage nodes to grow capacity, add connectors to grow throughput, and add metadata RAFT sessions (Scality metadata consistency groups) to balance the load across the system each without downtime, migration, or re-architecting.
- **Cyber-resilience built into the engine.** The CORE5 model layers protection across the API, data, storage, geographic, and architectural levels. Immutability is a property of the storage engine itself, it never overwrites in place, not a flag that a sufficiently privileged administrator can switch off.
- **Performance proven at scale, not at benchmark peak.** In a production-path test on a 100+ node HDD cluster across three availability zones, ADI sustained roughly 420 GB/s read and 250 GB/s write on 10 MB objects, and 173,456 PUT/s with 151,091 GET/s on 4 KB objects, with TLS and load balancing in every request path and zero storage-layer errors.
- **AI-native and autonomous.** GPU-Direct moves data directly between NVMe and GPU memory; Kubernetes CSI and COSI drivers provision storage as cluster-native resources; and the Guardian agent, exposed over the Model Context Protocol (MCP), turns operations into natural-language intent.

ADI is defined by two things. First, it is built on the field-proven Scality RING engine, the same distributed storage core running in production today at more than 12 EB, which ADI inherits for durability, self-healing, and linear scale rather than reinventing. Second, it extends that engine into a platform organized around four co-equal pillars: CORE5 (cyber-resilience), MULTISCALE (independent scaling of capacity, performance, and

operations), AICONNECT (native AI-workflow enablement), and GUARDIAN (autonomous operations). The product name foregrounds autonomy, but the platform rests on all four in equal measure, a proven engine made multi-temperature, secure by construction, AI-native, and self-operating.

ADI is built for organizations standardizing on a single storage platform for AI, sovereign cloud, backup, and archive at petabyte-to-exabyte scale. It is not intended as a niche, single-server appliance or as a transactional database substitute. The remainder of this paper works from the top of the stack to the bottom: the problem, the architecture, data protection, the metadata layer, the CORE5 security model, performance at scale, workload tiering and lifecycle, AI integration, autonomous operations, and reference deployment.

III. The Storage Challenge ADI Addresses

The previous generation of storage was designed for a world that no longer exists.

The dominant storage architectures of the last two decades were designed for human-generated, human-paced, single-temperature data. Scale-up NAS assumed a filer could grow vertically until it became someone else's problem. Early object stores assumed that unstructured data was, by definition, cold, written once, read rarely, and best optimized purely for cost-per-terabyte. Both assumptions have been invalidated by the same force: machine-generated data and machine-paced access.

Four shifts, in particular, define the environment ADI is built for.

AI and HPC have made throughput and latency a storage problem again

A GPU that is waiting for data is a depreciating asset that is not doing its job. Training and inference pipelines impose two demands at once that traditional object storage was never designed to meet: aggregate throughput measured in multiple terabytes per second to keep accelerators fed, and, for caches and intermediate state, latency low enough to sit just below GPU high-bandwidth memory in the hierarchy. A KV cache that takes milliseconds to fault is worthless; it must be retrieved in microseconds. No single fixed storage tier can deliver both sub-50-microsecond cache reads and economical multi-petabyte capacity, which is precisely why most environments end up with several incompatible systems.

Unstructured data has reached exabyte scale: and so has its metadata

As datasets grow into the hundreds of petabytes, the object count grows into the hundreds of billions. At that scale the hard problem is often not storing the bytes but tracking them. A metadata layer that performs well at ten million objects can fall off a performance cliff at one billion, turning routine operations into potential outages. Metadata is no longer a footnote to the architecture; it is a first-class scaling axis that must scale independently of raw capacity.

Ransomware specifically targets the recovery path

Modern extortion campaigns do not merely encrypt primary data; they hunt for and corrupt backups first, because a clean backup is the one thing that defeats them. Perimeter security is necessary but, by the attackers' own track record, not sufficient. The storage layer that holds backup and archive data has to assume the perimeter will eventually be breached and a privileged credential eventually compromised, and it has to keep the recovery path open anyway. That requires immutability that cannot be revoked by the very administrator the attacker is trying to impersonate.

Sovereignty and economics now dictate where data lives

Regulatory and sovereignty requirements increasingly mandate that data remain within specific jurisdictions and under the operator's own key control, pushing organizations toward private and hybrid infrastructure they fully govern. At the same time, the economics of an exabyte estate are unforgiving: paying hot-tier prices and hot-tier power for data that is 65% archival is not viable. The right cost and power profile has to be applied per workload, not averaged across the whole estate.

The conventional answer to these pressures is to buy four products, a fast parallel file system for AI, an object store for the cloud and backup, a separate archive target, and a separate management plane for each, and to spend operational effort moving data between them. Every migration is risk and cost; every additional namespace is another security surface and another policy that can drift out of alignment; and the parallel file systems that deliver AI throughput rarely deliver object economics or exabyte-scale durability.

ADI is designed for this reality directly. One distributed engine provides the durability and self-healing of large-scale object storage; one namespace presents that storage through both object and file protocols; a policy-driven temperature lifecycle moves data between GPU-Direct, Hot, Warm, and Cold tiers without copying it out to a foreign system;

and an autonomous operations layer keeps the whole thing manageable as it grows. The rest of this paper explains how.

IV. Scality ADI Architecture

ADI is a set of distributed software services that abstract physical servers and disks into a single, scalable, resilient storage service. At the top of the stack, a layer of stateless protocol services, connectors, presents S3 and file interfaces to applications. The middle layers provide distributed metadata services for both object and file access, the data-protection and integrity mechanisms that keep data durable, the self-healing processes that repair failures automatically, and the management and monitoring services. At the bottom, a distributed storage layer composed of virtual storage nodes and their underlying I/O daemons abstracts the physical servers and disk drives.

Design principles

ADI inherits and extends a set of design tenets that have governed Scality's storage engine since it was first built to the criteria of hyperscale cloud providers:

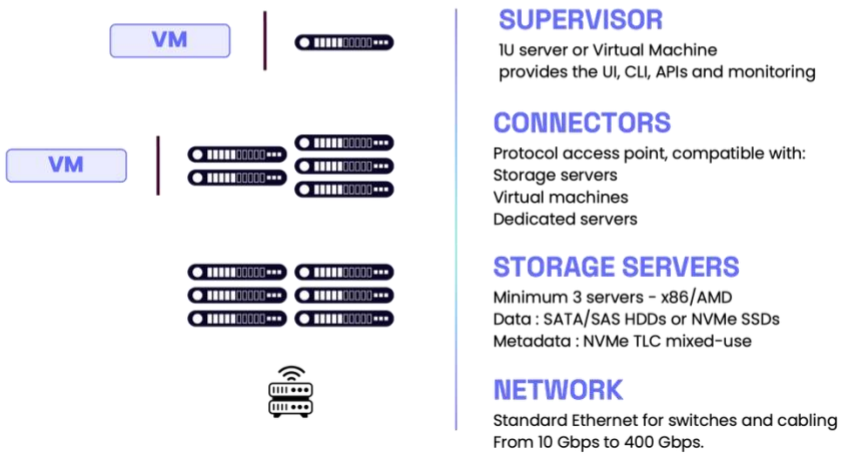
- **Parallel by design.** Data and metadata are served by fully parallel subsystems, so capacity and performance keep scaling together to effectively unlimited numbers of objects, with no single point of failure, no service disruption, and no disruptive forklift upgrades as the system grows.
- **Multi-protocol access.** Object and file applications use the same underlying storage concurrently, rather than being forced into separate systems.
- **Flexible, size-aware data protection.** Replication and erasure coding are applied per object according to size and durability requirements, trading space against performance deliberately.
- **Self-healing from component failure.** The system expects, tolerates, and automatically resolves disk, server, network, and site failures rather than treating them as exceptional events.
- **Hardware freedom.** ADI runs on commodity x86/AMD servers and standard Ethernet, eliminating lock-in and letting the operator mix hardware generations to optimize total cost of ownership.

Deployment components

A deployment is built from four component roles, which may be co-located or disaggregated depending on the target tier:

- **Connectors.** The protocol access points. Each connector terminates client protocol traffic (S3 or file) and applies the configured data-protection policy. Connectors are stateless, so they can be scaled out horizontally and load-balanced freely; they may run on the storage servers themselves, in virtual machines, or on dedicated servers.
- **Storage servers.** A minimum of three x86/AMD servers hold the data. Bulk capacity lives on SATA/SAS HDDs or NVMe SSDs; metadata lives on a separate set of NVMe TLC mixed-use devices. The three-server minimum is what makes meaningful data protection possible from the smallest supported deployment upward.
- **Supervisor.** A 1U server or virtual machine that provides the management UI, CLI, REST APIs, and monitoring. It is a management-plane component and is not in the data path.
- **Network.** Standard Ethernet switching and cabling. ADI deliberately avoids exotic fabrics for general-purpose tiers; the GPU-Direct and Hot AI tiers add RDMA-capable networking, discussed later.

These roles are deployed across three logically separate network planes, a data plane carrying S3 and file traffic, a management plane for administration, and an out-of-band plane (for example IPMI on its own VLAN) for physical server management. Keeping the planes separate is both a performance and a security decision and is revisited in the CORE5 section.



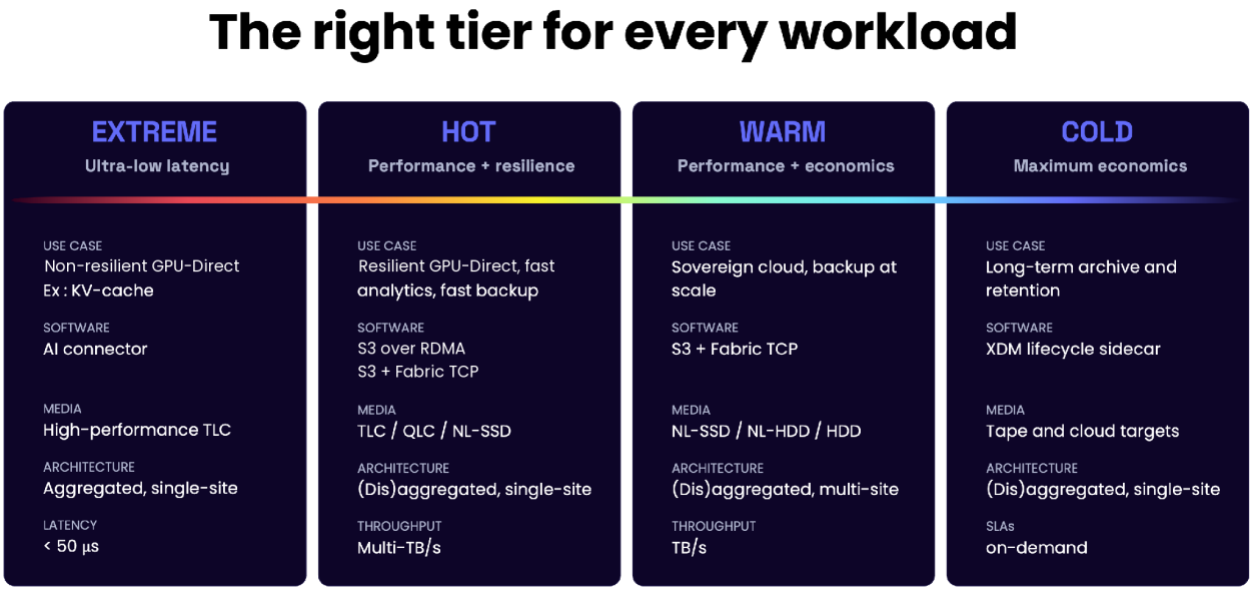
A single namespace across four temperatures

Policy-driven temperature lifecycle and autonomous operations over one ADI namespace.

The defining structural choice in ADI is that one namespace spans four operating tiers, each tuned for a different balance of latency, throughput, resilience, and cost. Data is placed in, and moved between, these tiers by policy, not by exporting it to a separate product. The tiers are summarized below and treated in depth in the workload-tiering section.

Attribute	GPU-Direct	Hot	Warm	Cold
Optimized for	Ultra-low latency	Performance + resilience	Performance + economics	Maximum economics
Use case	AI KV-cache, HPC scratch	Resilient GPU-Direct, fast analytics/backup	Sovereign cloud, backup at scale	Long-term archive & retention
Access software	GPU-Direct AI connector	S3 over RDMA, S3 over TCP	S3 + Fabric TCP	XDM lifecycle sidecar
Media	High-performance TLC	TLC / QLC / NL-SSD	NL-SSD / NL-HDD / HDD	Tape and cloud targets

Attribute	GPU-Direct	Hot	Warm	Cold
Topology	Aggregated, single-site	(Dis)aggregated, single-site	(Dis)aggregated, multi-site	Aggregated / archive
Target	< 50microseconds	Multi-TB/s throughput	TB/s throughput	On-demand SLAs



Because the tiers share one namespace and one engine, a dataset can be ingested hot, aged to warm capacity by lifecycle policy, and finally transitioned to cold archive, all under a single set of credentials, a single security model, and a single operational console.

Software stack and request flow

Application I/O enters ADI at the connector. The connector authenticates and authorizes the request, then dispatches it to the storage layer, applying the configured protection policy (replication or erasure coding). On write, objects larger than a configurable threshold are divided into chunks before being sent to the storage nodes; the storage nodes in turn hand chunks to the I/O daemons that persist them on disk or drive. On read, the connector retrieves the chunks, reconstructs the object if erasure-coded, and returns it. Because connectors are stateless, an operator scales operations-per-second or aggregate throughput simply by adding connectors behind a load balancer, independent of how capacity is scaled.

The distributed storage engine

At the heart of the storage layer is a distributed key/value store based on a peer-to-peer routing protocol. Storage is organized as a logically circular keyspace; each physical server typically contributes six virtual storage nodes (termed bizstorenodes), and each owns a contiguous range of the keyspace. A request for a given key is routed deterministically from any node to the node that owns the key, and the number of routing hops grows sub-linearly with cluster size, so lookups remain fast from three nodes to thousands. The routing tables are decentralized: each node knows only its own range, a few neighbors, and a few well-known projections across the keyspace, so there is no central map to bottleneck or lose.

This design is what allows ADI to add and remove nodes without downtime. When a node joins or departs, the system automatically rebalances the affected portion of the keyspace and, during the transition, routes around the change with proxies and alternate paths so that data remains fully accessible. Each object key encodes, among other things, the object's class of service, so the protection level travels with the data.

Beneath the virtual storage nodes sit the I/O daemons (termed biziods), with support for up to hundreds per server. Each daemon owns exactly one physical disk, manages the low-level I/O to it, and maintains the mapping from object keys to on-disk locations. Daemons store payloads and metadata in fixed-size container files, which lets the system maintain high performance even for very small objects without fragmenting the underlying file system; index files are placed on low-latency flash for fast lookups. Because a standard file system sits underneath each daemon, ordinary operating-system tools can be used to inspect or maintain the disk files if ever required. The recommended layout places the data keyspace on HDD and the associated metadata keyspace on SSD/NVMe, since metadata is typically only about 2% of capacity but benefits disproportionately from flash latency.

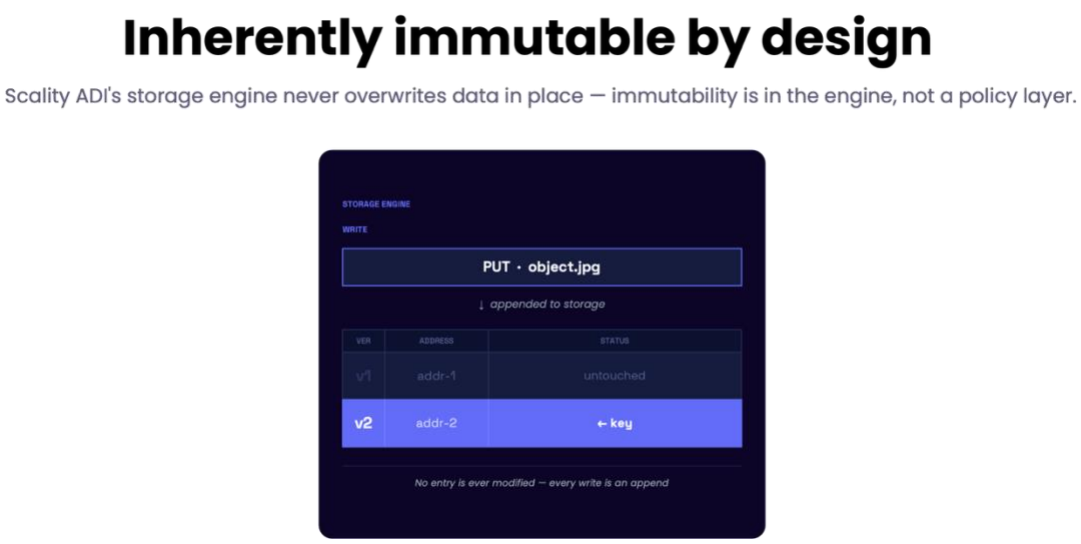
Inherently immutable by design

ADI's storage engine never overwrites data in place, immutability lives in the engine, not in a policy layer.

This property underpins much of the security model. When a client issues a PUT to a key that already exists, ADI does not modify the existing data; it allocates a fresh storage

location, writes the new object there, and updates the key index to point at the new address. The prior version is untouched. There is no operation in the engine that overwrites an object's bytes in place.

The distinction matters in contrast to architectures that place an S3 facade over a mutable file system or block device. In those systems the data is mutable underneath the API, a PUT ultimately modifies blocks in place, and immutability is a policy enforced above a substrate that does not itself guarantee it. In ADI, immutability is how the object store fundamentally works, and it cannot be reconfigured away. Everything in the CORE5 model builds on this foundation.



S3 object storage

ADI implements the AWS S3 REST API comprehensively. For example: buckets and objects, multipart uploads, presigned URLs, object tagging, versioning, lifecycle, Object Lock, cross-region replication, server-side encryption, and bucket notifications. The goal is drop-in compatibility, existing S3 SDKs, tools, and policies work without a translation layer. On top of the object model, ADI implements the AWS IAM: accounts, users, groups, roles, and policies, with up to 5,000 users and 250 roles per account for full multi-tenancy.

SOFS scale-out file system

SOFS presents the same storage as a POSIX file system over SMBv3, NFSv4, and Linux FUSE, with concurrent multi-protocol access. Multiple connectors serve the same namespace simultaneously; CTDB manages floating virtual IPs to provide high

availability for NFS and SMB clients. Crucially, file-system metadata is fully distributed in a dedicated flash keyspace, there is no central metadata server and therefore no single point of failure, and a distributed lock manager maintains consistency across concurrent connectors. Tenancy is enforced by treating separate volumes as independent namespaces, and the system supports an unlimited number of volumes and files, scaling by adding nodes rather than hitting an architectural ceiling.

V. Data Protection and Durability

Two mechanisms, chosen per object by size, deliver durability while letting the operator trade space against performance.

ADI protects data with two complementary mechanisms, replication and Reed-Solomon erasure coding, applied per object according to a configurable size threshold. By default the threshold is 60 KB: objects below it are replicated, objects at or above it are erasure-coded. Each connector can carry its own threshold, so a deployment can be tuned for the object-size distribution it actually sees.

Replication and class of service

Replication stores N identical copies of an object distributed across different servers and disks, and is optimal for small, frequently accessed objects where the latency of reconstruction would dominate. The protection level is expressed as a class of service (CoS): CoS=1 stores two copies and tolerates a single disk failure; CoS=2 stores three copies and tolerates two failures; CoS=3 stores four copies and tolerates three. The cost is storage overhead, which is why replication is reserved for small objects.

Replication & Class Of Service

Stores and distribute N identical copies of each object. Optimised for small, frequently accessed objects (< 60 KB).



Class of service	Copies stored	Storage overhead	Disk failures tolerated
CoS = 1	2 copies	100%	1
CoS = 2	3 copies	200%	2
CoS = 3	4 copies	300%	3

For a 40 KB object stored at CoS=2, the system consumes 120 KB of physical capacity. That overhead is acceptable for small objects but becomes punitive for megabyte- and gigabyte-scale data, a 200% penalty across petabytes of media files is a substantial and avoidable cost, which is exactly the case erasure coding addresses.

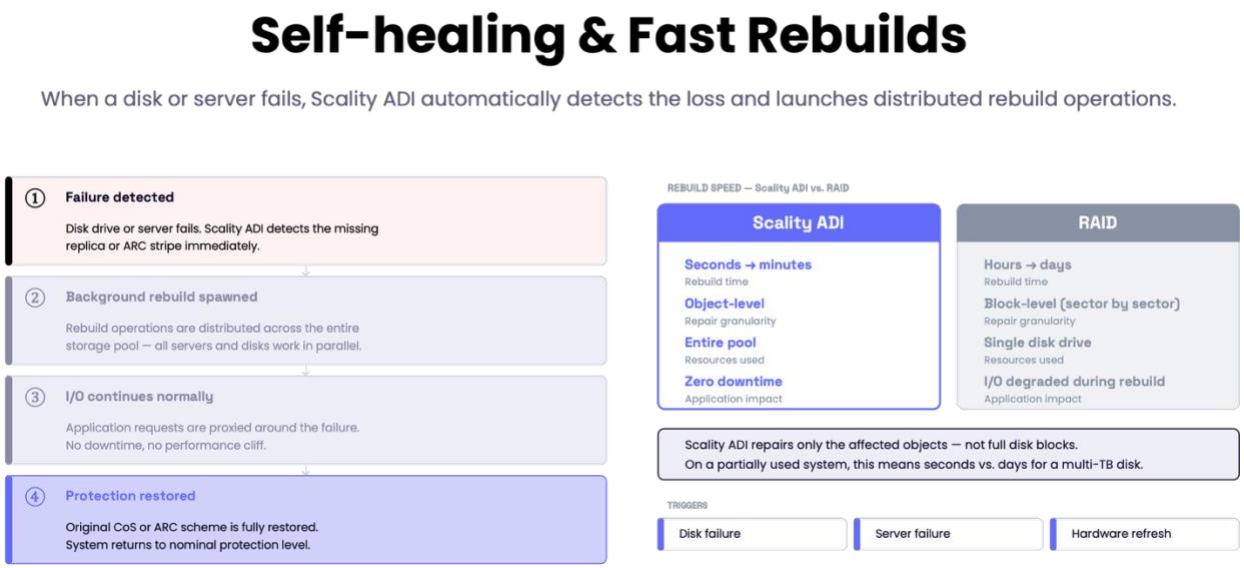
Reed-Solomon erasure coding

For large objects, ADI uses Reed-Solomon erasure coding, branded ARC (Advanced Resiliency Configuration), Scality's erasure-coding scheme: each object is split into m data chunks, an additional k parity chunks are computed, and the resulting m+k chunks are distributed across the cluster, each on a separate disk. The object can be reconstructed from any m of the m+k chunks, so the scheme tolerates k simultaneous failures with only k/m space overhead. A 9+3 schema, for example, splits a 9 MB object into nine 1 MB data chunks plus three 1 MB parity chunks: 12 MB stored, 33% overhead, and protection against three simultaneous disk failures, versus the 200% overhead that triple replication would require for the same durability.

ADI's erasure-coding implementation has a property that materially affects read performance: data chunks are stored in the clear, unencoded. As long as the data chunks are present, a read incurs no decode step and is served as fast as any other read; parity is consulted and decoded only when a data chunk is actually missing. This is a deliberate contrast with implementations that encode all stripes, including the data, and therefore impose a mandatory decode on every read regardless of whether any failure has occurred. Internally, large objects are decomposed into fixed-size stripes that are individually erasure-coded, so the protection and distribution happen at a granular level across the system.

Self-healing and fast rebuilds

When a disk or server fails, ADI detects the loss and rebuilds in parallel across the entire pool, while I/O continues.



ADI is designed to expect failure and resolve it automatically. The self-healing process follows four stages:

1. Failure detected. A disk or server fails, and ADI immediately detects the missing replica or erasure-coded chunk.
2. Background rebuild spawned. Rebuild operations are distributed across the entire storage pool, so all servers and disks contribute in parallel rather than serializing repair onto a single device.

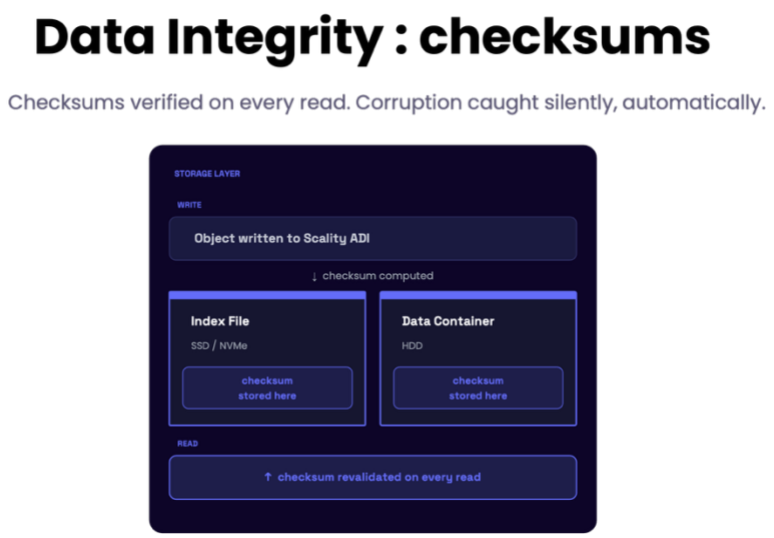
3. I/O continues normally. Application requests are proxied around the failure; there is no downtime and no performance cliff.
4. Protection restored. The original class of service or erasure-coding scheme is fully reconstituted, returning the system to its nominal protection level.

The decisive design choice is that ADI repairs only the affected objects, not entire disk blocks. A RAID array rebuilds a failed drive sector by sector, regardless of how much of it was in use, which can take hours to days for a modern multi-terabyte disk and degrades I/O throughout. ADI repairs only the data that was actually present and spreads that work across the whole pool, so on a partially used system a rebuild completes in seconds to minutes with no application impact. The same machinery handles planned events, server decommissioning and hardware refresh, as gracefully as it handles failures.

Dimension	Scality ADI	Traditional RAID
Rebuild time	Seconds to minutes	Hours to days
Repair granularity	Object-level (only affected data)	Block-level (whole disk, sector by sector)
Resources used	Entire storage pool, in parallel	Single replacement drive
Application impact	Zero downtime, no performance cliff	I/O degraded during rebuild

Data integrity: end-to-end checksums

Durability is not only about surviving drive failure; it is also about catching silent corruption. On write, ADI computes a checksum and stores it on two independent layers, the index file (on flash) and the data container file (on disk), so the two are protected separately. On every read, ADI recomputes the checksum and compares it against the stored value. This is always on; there is no configuration and no opt-in. If a mismatch is found, recovery is automatic and happens before the application sees the data: the corrupt part is reconstructed from parity or served from another replica. Bit rot is therefore detected and corrected as a routine background property of every read, not discovered months later during a restore.



VI. The Metadata Layer: Built for Billions

At exabyte scale the hard problem is tracking the objects, not storing the bytes. ADI's metadata layer is a first-class, independently scalable subsystem.

The S3 metadata service (S3 MD) is the component responsible for all S3 object metadata: the mapping of buckets and keys to object locations, plus the persistent store for accounts, users, groups, and policies. It is the service that most directly determines availability and performance at scale, and it is engineered as a distributed, strongly consistent, flash-backed cluster that scales independently of raw capacity.

Topology

S3 MD runs on a dedicated subset of ADI servers, either aggregated with storage nodes or on standalone hardware, and is backed by NVMe flash for consistently low-latency metadata operations. A metadata cluster is built from five distributed members with no single point of failure; it is quorum-based, so it survives the loss of a member without interruption. Metadata scales along its own axis: adding clusters scales the aggregate operations-per-second the namespace can sustain, and the design is proven at hundreds of billions of objects per bucket in production.

The RAFT session: the unit of consistency

The consistency unit of the metadata cluster is the RAFT session. Each session groups metadata processes across the cluster members into one leader and multiple followers, and owns a specific set of buckets, not the whole namespace. All writes for those buckets

go through the leader, which replicates state to the followers. A write commits only after a quorum acknowledges it: in a five-member session, three acknowledgments are required before the write is confirmed. This is what guarantees strong consistency, there is no divergence between members and no window in which a confirmed write can be lost.

Failover is automatic. The leader sends periodic heartbeats; if followers miss a sequence of them, they detect the leader loss and initiate a re-election. The cluster stays available as long as a majority of members are reachable, and no manual intervention is required.

Scaling concurrency within a cluster

A single RAFT session has a single leader, and that leader is a finite resource. ADI scales metadata parallelism by running multiple RAFT sessions in parallel on the same cluster, each with its own elected leader and its own subset of buckets, as many sessions as the available CPU cores allow. A dedicated bucket-map component routes every metadata operation to the correct session transparently, so the application sees one endpoint while work is spread across many leaders. Buckets can be assigned explicitly, which lets an operator keep heavy-traffic buckets from contending with quiet ones: a busy bucket on one session has no effect on buckets owned by another.

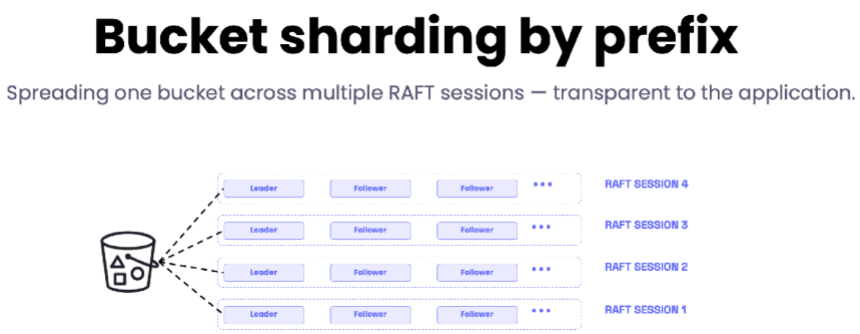
Online bucket migration

Workloads evolve unevenly. A RAFT session can become saturated when several hot buckets land on the same leader; the symptom is that low-traffic neighbor buckets on that session start returning errors despite carrying little load of their own. The fix is to move a hot bucket to a less-loaded session, on the same cluster or a different one. ADI does this without downtime: a metadata migration service copies all of the bucket's metadata to the target session while the bucket remains fully readable and writable, and at cut-over the bucket map is updated atomically, no read window, no lost operations. All attributes survive the move: name, keys, ACLs, CORS configuration, and tags. The migrated bucket lands on an uncongested leader, its former neighbors recover immediately, and the errors stop.

Bucket sharding by prefix

Migration rebalances buckets across sessions, but a single very hot bucket is still ultimately capped by its single session leader, and adding sessions does not help if all of

one bucket's traffic lands on the same one. For that case ADI provides prefix sharding: a single bucket is split across multiple RAFT sessions by key prefix, with each shard owning its own leader and processing writes independently. Clients see no change, they use the same endpoint, and routing happens transparently at the metadata layer. The result is that a single bucket can scale to match the full capacity of the metadata cluster, with no throughput ceiling and no write-serialization bottleneck, and with zero application changes. Viewed together, the multi-session model, online bucket migration, and prefix sharding are three answers to one root constraint, the single-leader bottleneck inherent to any strongly consistent cluster, applied respectively at the granularity of the whole cluster, a single bucket, and one hot bucket.



A metadata store built for billions of objects

Three properties make the metadata layer hold up at the largest scales. First, predictable latency: response time stays flat as a bucket grows past a billion objects, there is no performance cliff as datasets grow. Second, burst resilience: an intelligent request queue absorbs metadata bursts at an optimal pace, eliminating the HTTP 500 errors that heavy backup workloads otherwise provoke, so a noisy neighbor does not degrade the cluster. Scality ADI hardens this further with an improved S3 metadata write cache and an optional queueing mechanism that protects the service under extreme traffic surges. Third, efficiency: better compaction means metadata occupies two to three times less space for the same object count, which shrinks the NVMe footprint the metadata cluster needs and lowers infrastructure cost at a given capacity.

Compaction on your schedule

Metadata grows over time as objects are created, versioned, and deleted, it can grow even when the underlying data does not. ADI reclaims that space with scheduled compaction that runs node by node: a cron job runs at an operator-chosen time and

compacts all RAFT sessions on one node per day, so there is never a cluster-wide impact and never downtime. Storage usage drops as space is reclaimed, and performance stays stable no matter how long the cluster has been running.

Faster listings in versioned buckets

Buckets with heavy versioning accumulate delete markers and noncurrent versions that make LIST operations expensive. ADI introduces a smarter metadata format (V1) optimized for exactly that workload, dramatically improving listing speed under the most demanding versioning patterns. Adoption is non-disruptive: once enabled, new buckets use V1 automatically while existing V0 buckets keep working unchanged, and the two formats coexist transparently in the same cluster. The bucket-migration feature converts a V0 bucket to V1 on demand, with no service interruption, the operator decides which buckets to migrate and when.

The metadata storage engine

Through RING 9, S3 object metadata was held in a LevelDB-based engine. Scality ADI introduces RocksDB as the new default metadata storage engine, for better scalability and more stable performance under heavy S3 load. Adoption is non-disruptive and hybrid: new metadata RAFT sessions created on ADI use RocksDB, pre-existing sessions keep using LevelDB, the two coexist in the same cluster, and a bucket-level migration converts metadata from LevelDB to RocksDB on demand.

File metadata is handled with the same philosophy on the SOFS side: it is fully distributed in a dedicated flash keyspace with no central server, and a distributed lock manager preserves POSIX consistency across concurrent connectors. In both the object and file worlds, the metadata layer is a parallel, flash-resident, independently scalable subsystem, which is what lets the namespace grow to hundreds of billions of objects without the metadata becoming the bottleneck.

VII. CORE5: Cyber-Resilience by Design

Data survives the attack; recovery survives the audit.

Cyber-resilience in ADI is not a feature bolted onto the storage; it is a layered model, CORE5, that answers a specific adversarial question at each of five levels of the architecture. The framing is deliberately phrased as the questions a security architect actually asks:

Level	The question it answers	The threat addressed
1: API	Can someone neutralize my data through the S3 API?	Malicious deletes/overwrites via valid API calls
2: Data	Can someone read or steal my data?	Exfiltration, eavesdropping, stolen media
3: Storage	Does my data survive hardware failures?	Disk, server, and silent-corruption loss
4: Geographic	Does my data survive if a whole site goes down?	Site loss, regional disaster, ransomware blast radius
5: Architecture	Can a fully privileged admin destroy everything?	Insider threat, compromised credentials

No single control answers all five questions; the point of CORE5 is that the controls are layered so that a failure or compromise at one level is contained by the others. Two of the five levels, API (Level 1) and architecture (Level 5), ultimately rest on the same foundation, the inherently immutable engine of Section IV, which is why that one design choice carries so much of the security model. The remainder of this section works through each level.



Level 1: API-level resilience

The first line of defense is making the S3 and file APIs themselves non-destructive, so that a valid but malicious call cannot quietly neutralize data.

S3 object versioning

Versioning is a one-way switch on a bucket: its state moves never-set → Enabled → Suspended, with no path back to fully off. Once enabled, every PUT produces a new version with a unique version ID, and every DELETE writes a delete marker rather than removing data. A PUT to an existing key creates a new version while the prior one is retained; a DELETE makes the latest version invisible (a plain GET then returns 404) but any earlier version is still readable by version ID. Old versions are pruned not by destructive overwrites but by bucket lifecycle rules on a schedule the operator defines.

S3 lifecycle

Lifecycle rules are declarative and execute asynchronously, off the S3 hot path. A rule's actions include expiration of current objects by age or date, expiration of noncurrent versions, cleanup of incomplete multipart uploads, and removal of orphaned delete markers; rules are scoped by prefix, by tag, by multiple tags combined, or by prefix-and-tag. A background scanner runs the rules, by default every five minutes, configurable, and rules can be paused per bucket without being deleted. Up to 1,000 rules per bucket are supported, matching the AWS limit. Critically, lifecycle coexists safely with the protective features: Object-Locked objects are skipped so legal holds and retention are honored, and expiration is deferred while a copy is still replicating under cross-region replication.

S3 Object Lock

Object Lock provides true write-once-read-many immutability per bucket, object, or version, in three forms:

- **Compliance mode.** There is no bypass path, no API, no administrative privilege, and no storage-level access can delete or modify an object before its retention expires. Retention can only be extended, never shortened; the API rejects any value below the current one. Each version is locked independently, so a new PUT writes a new version without affecting the prior one's retention.
- **Governance mode.** Retention can be shortened, but only by a caller holding the explicit s3:BypassGovernanceRetention permission, and the bypass call is logged.

This is the right mode where operational mistakes still need an escape hatch; compliance mode is the right mode for regulated records and backups.

- **Legal hold.** An indefinite, no-expiry hold that remains active regardless of any retention period, toggled on or off only by a holder of s3:PutObjectLegalHold. It is designed for litigation and investigation, where the duration is unknown up front and the hold lifts only when the process concludes.

Because the modes are expressed through standard S3 headers, an application can switch between them without code changes.

SOFS recycle bin and volume protection

The file side has equivalents. A per-inode recycle bin keeps deleted files and truncate-to-zero copies in a /.recycle area governed by a lifecycle policy: a DELETE moves the file there with a timestamp suffix, and a truncate(0) snapshots the prior content (in-place writes are not captured). Recycled files are immutable, visibility is configurable (admin-only, admin-only purge, or owner-and-admin purge), and the bin's growth is bounded by capping the number of versions retained per rolling time window, with a configurable regex to exclude noisy files. Separately, SOFS volume protection enforces WORM at the file-system tier: a volume can be read-write (the default), entirely read-only, or modifiable for a configurable time window, by default 43,200 seconds (12 hours), after which existing files become immutable while new files can still be created. The clock is per inode, not volume-wide.

Level 2: Data-level resilience

The second level protects against reading or stealing data, through access control, encryption in transit, encryption at rest, and identity federation.

IAM: zero-trust by default

Every S3 request is evaluated against four dimensions of access control. Least privilege is the baseline, nothing happens without an explicit Allow, against a catalog of 100+ grantable actions across S3, IAM, STS, and quotas. Defense in depth means any Deny, anywhere in the union of IAM and bucket policies, short-circuits the request to refused regardless of how many Allows exist. Requests are context-aware, bound by conditions such as source IP, transport security, and required encryption headers. And the account boundary provides tenant isolation: cross-account access is possible only through an explicit AssumeRole trust, with up to 5,000 users and 250 roles per account.

TLS: encryption in transit

ADI enforces HTTPS everywhere, there is no plaintext on the wire between clients, S3 endpoints, and nodes. The stack is TLS 1.3 with NIST-validated, FIPS 140-3-ready cryptographic modules enforced in ScalityOS, and a bucket policy can be configured to refuse any non-HTTPS call outright. A full internal public-key infrastructure is provided: the CA, server certificates, and client certificates are all generated by Scality scripts, key rotation is on the operator's schedule rather than tied to software upgrades, and certificate files are locked to root on every node with no user-readable copies. This is reinforced by the three separate network planes described earlier, data, management, and out-of-band, so that administrative tooling is never reachable from where S3 clients live, and physical (IPMI) management sits on its own VLAN.

S3 server-side encryption at rest

Encryption at rest is designed around operator key ownership. Master keys live in the operator's own KMS; the cluster never holds a key in the clear, so a stolen disk is worthless. ADI brings no lock-in, it speaks open standards (KMIP and the AWS-KMS API) and is compatible with systems such as Thales CipherTrust, HashiCorp Vault, and HyTrust KeyControl. Deleting a bucket or account destroys the corresponding master key, rendering the data unreadable instantly, cryptographic erasure rather than a petabyte-scale wipe. Encryption runs inline in the S3 connector, so the payload never crosses the KMS, only the comparatively tiny data keys do, and there is therefore no KMS bottleneck on the data path. Two modes are offered (AES256 with managed keys, or aws:kms with operator-owned keys and a per-bucket key ID), and the policy can be set per object, per bucket, per account, or cluster-wide, with one flag enforcing it on every new bucket. Every key wrap and unwrap is logged on the operator's own KMS under the operator's retention policy, keeping the audit trail end-to-end on their side. The KMS integration is itself highly available: ADI round-robins across every KMS node so an outage does not stop ingest, quarantines a failing node for 30 seconds before retrying automatically, and exposes a configurable retry count.

LUKS: full-disk encryption

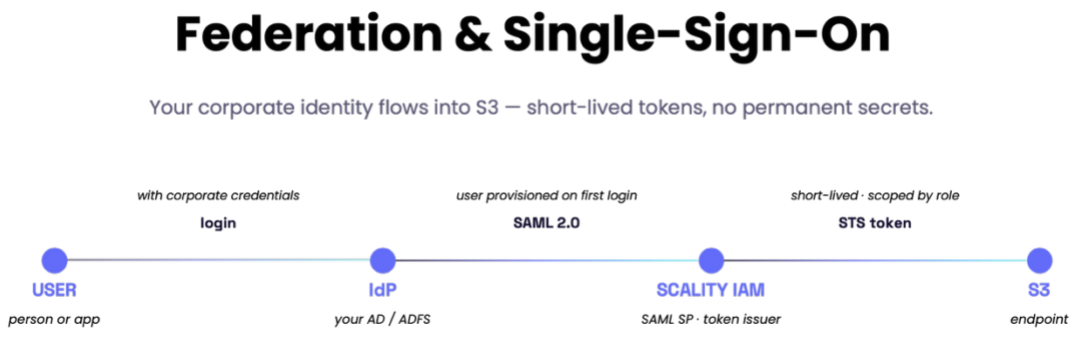
LUKS is the third encryption layer. Where server-side encryption protects object payloads at the API and TLS protects traffic on the wire, LUKS encrypts the entire drive, payloads, metadata, indexes, ADI internals, and OS files alike, in a single pass. Its purpose is defense in depth against physical attacks: a pulled disk or a stolen powered-off node is inert

without the key, and LUKS removes the metadata leakage and OS-level attack surface that payload encryption alone cannot cover. It must be enabled at first install (it cannot be added to a live cluster), and its keys live off the node, in a KMS, a TPM, a USB token, or an operator passphrase.

Layer	What it protects	Scope
TLS	Network traffic (client↔S3, node↔node)	In transit
SSE / KMS	S3 object payloads (one data key per object, wrapped by your KMS)	At the API
LUKS	Everything on the drive: payloads, metadata, indexes, OS files	At the disk

Federation, single sign-on, and operational hardening

ADI integrates with enterprise identity through SAML 2.0: a corporate AD/ADFS authenticates the user, each IdP domain is mapped to an ADI account at setup, and users are provisioned automatically on first login, there are no separate, permanent S3 secrets to manage. Credentials are ephemeral, issued through STS as short-lived tokens scoped by role, auto-renewed before expiry, and held in memory only. Operational hardening rounds this out: MFA on admin consoles, and SSH key-only access on every node with password authentication disabled by default. Management surfaces are governed by four built-in RBAC roles, ADI Admin (full control), Monitoring Manager (observability), ADI Viewer (read-only), and Identity Manager (users and groups), bound at login to LDAP/Active Directory groups or local Keycloak accounts, rather than per-object ACLs.



Level 3, Storage-level resilience

The third question, does data survive hardware failure, is answered by the durability machinery described in the data-protection section: replication and erasure coding tolerate concurrent disk and server failures, self-healing rebuilds restore the configured protection level automatically and in parallel, and end-to-end checksums detect and correct silent corruption on every read. From a cyber-resilience standpoint the relevant point is that these mechanisms operate continuously and without operator action, so the storage substrate underneath the immutability guarantees is itself continuously verified and repaired.

The data-protection layer answers this in four reinforcing ways, each treated in depth in Section V:

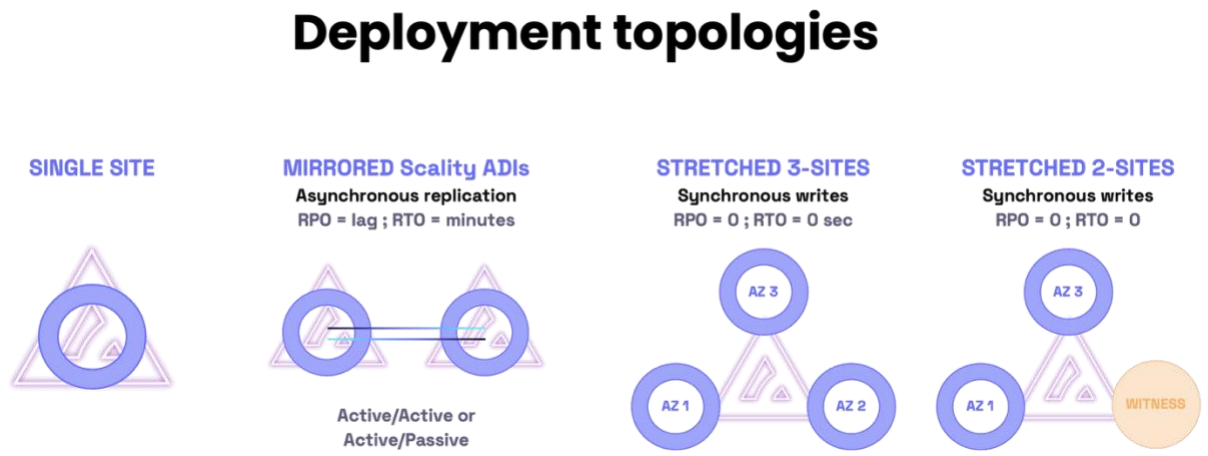
- **Replication and class of service.** Small, frequently accessed objects (under 60 KB) are stored as N identical copies on separate servers, a class of service of 2 keeps three copies and survives a double-disk failure, and the system self-heals automatically when a disk is lost.
- **Erasure coding.** Larger objects are split into data and parity chunks written across separate disks (for example nine data plus three parity, roughly 33% overhead), tolerating multiple simultaneous disk and server failures at a fraction of replication's cost.
- **Self-healing and fast rebuilds.** When a disk or server fails, ADI detects the loss immediately and rebuilds the affected objects in parallel across the whole pool, repairing only the data that was lost, not entire drives, so recovery takes seconds to minutes rather than hours to days.
- **End-to-end checksums.** Every object carries checksums on two independent layers, recomputed on every read, so silent corruption is caught and corrected from parity or a replica before the application ever sees it.

Because all four mechanisms run continuously and without operator action, the storage substrate beneath the immutability and geographic guarantees is itself constantly verified and repaired, which is what lets the upper CORE5 layers treat durable storage as a foundation rather than an assumption.

Level 4: Geographic-level resilience

The fourth level keeps data and the recovery path alive when an entire site is lost. ADI supports a spectrum of multi-site topologies trading recovery objectives against cost and distance:

Topology	Replication	RPO	RTO
Stretched 3-site	Synchronous across 3 AZs	0	0 Seconds
Stretched 2-site + witness	Synchronous, witness arbitrates	0	0 Seconds
Mirrored ADIs (single/multi-site)	Asynchronous (active/active or active/passive)	Replication lag	Minutes



S3 cross-region replication

CRR replicates asynchronously between two independent ADI systems in one of two operational models. In active/active, both sites serve reads and writes simultaneously with no standby; on a site failure, applications redirect to the survivor and all data up to the last replicated write is available, while anything written since the last replication is lost until the failed site returns and the failback procedure (re-sync, drain the queue, switch endpoints back) completes. In active/passive, the primary serves all traffic while the DR site stays in sync in the background; on primary failure the administrator redirects applications to DR and issues new credentials, credentials are deliberately not replicated, and pre-configuring reverse replication on the DR site is recommended to speed failback.

SOFS multi-site asynchronous replication

On the file side, SOFS replicates data and metadata asynchronously to one or more remote sites using a dual-path mechanism: each filesystem change emits a hint that

triggers a pull on the target connectors, while a background crawler periodically performs a full scan as a safety net so nothing is missed. Replication lateness, the age of the oldest unsynchronized change, is monitored continuously, with alerts on threshold breach and throttling configurable per connector and per site. On source failure, production switches to a target site by reconfiguring roles, and multi-target topologies extend DR coverage further.

Level 5: Architecture-level resilience

The final and most stringent question is whether a fully privileged administrator, or an attacker wielding stolen admin credentials, can destroy everything. This is where the inherently immutable engine described earlier becomes a security control rather than merely a design elegance. Because the engine never overwrites in place, and because Object Lock in compliance mode admits no bypass path at the API, the administrative, or the storage level, there is no privileged operation that can delete or alter a locked object before its retention expires. The protection is not enforced by a policy daemon that a root user could disable; it is a property of how the store writes data. Combined with the geographic air-gap rotation, this means that even a worst-case insider compromise cannot reach across to neutralize every copy, the recovery path is structurally preserved. ADI also keeps extensive audit logs of administrative, user, and internal activity in open JSON for ingestion into SIEM and logging tools, so that recovery survives not only the attack but the subsequent audit.

VIII. Performance at Scale

“It’s not always about who has the most horsepower; it’s about who can use it best.”, and performance is experienced at scale, not just measured at benchmark peak.

Peak benchmark numbers on a tuned, idle rig tell an architect very little about how a system behaves in production under sustained, mixed, encrypted load across many tenants. ADI's performance story is therefore framed around behavior at scale: predictable throughput and latency across large, real-world environments, validated on production-path configurations rather than synthetic harnesses.

Field-proven throughput and ingest

In a controlled test under production-grade conditions, ADI sustained the following on a warm S3-on-HDD configuration:

- **Large-object throughput:** approximately 420 GB/s read and 250 GB/s write on 10 MB objects, with peaks approaching 450 GB/s read. Throughput remained stable across the full test window with no degradation after ramp-up and zero storage-layer errors throughout, bandwidth levels historically associated with dedicated parallel file systems, delivered here over warm S3 on hard drives.
- **High-rate small-object ingest:** 173,456 PUT/s and 151,091 GET/s on 4 KB objects. Under saturation, latency rose predictably while aggregate throughput did not collapse, the system degrades gracefully rather than falling off a cliff.
- **Production-path conditions:** a 100+ node HDD cluster across three availability zones, with TLS and load balancing in every request path, sustaining 35,000 concurrent S3 operations across 696 buckets over a two-hour window with zero storage-layer errors. Throughput was measured at both the ADI layer and the load-balancer layer, that is, on externally visible encrypted traffic, not an internal best case.

Two details make these numbers credible rather than merely large. First, they were measured on HDDs with TLS active end to end, so they reflect the overheads a real deployment carries. Second, they were sustained for hours across hundreds of buckets and tens of thousands of concurrent operations, the regime in which lesser systems exhibit error storms and tail-latency blowups. That said, a rigorous reader should treat these as one well-instrumented data point and ask for the full report, drive model and count, network fabric, the object-size mix, and the latency distribution rather than only the mean, which is exactly the scrutiny the rest of this paper invites.

Predictable read latency under partial failure

Throughput at scale is undermined by tail latency, the occasional slow drive or busy node that stalls a request. ADI addresses this with a feature branded “Read Latency Predictability”. When the retrieval of a part exceeds a latency threshold, the connector fetches the part from an alternate location: for a replicated object an alternate copy is retrieved, while for an ARC (erasure-coded) object an alternate part is retrieved and the missing part is rebuilt on the fly from parity. The protection level is unchanged; only the read path adapts. The logic is available for both SOFS and S3 connectors and is transparent to the application, though it is enabled per use case, with the latency threshold tuned to the client's response-time requirements (set higher, for example, on

multi-site stretched deployments). The result is consistent low read latency regardless of the health of any individual disk.

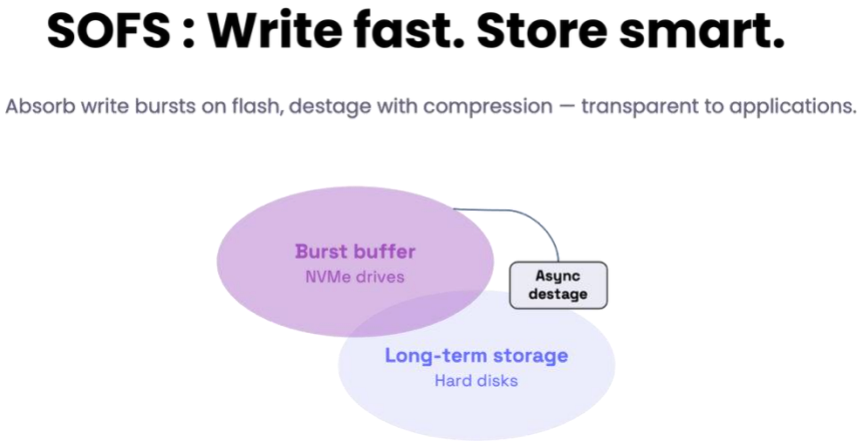
Seamless drive operations

Capacity changes and drive failures are routine at scale, and ADI is designed so they do not perturb steady-state performance. Write rebalancing steers new writes toward new or under-loaded drives by weighting disk selection according to relative fill level, while existing drives keep serving I/O without disruption. Smart disk selection is active only when it is needed, at drive insertion or when an imbalance is detected, so steady-state I/O is unaffected, and the rebalancing is write-proportional, keeping overhead low whether the system is expanding or recovering from a failure. There is no hotspot and no bottleneck during the replacement window.

SOFS write buffering and inline compression

Absorb write bursts on flash, destage with compression, transparently to applications.

For file workloads with bursty ingest, SOFS uses the Storage Accelerator to land incoming files first on a high-speed flash tier, absorbing massive write bursts without saturating long-term HDD throughput, and serving reads from the buffer while data is still hot. Data is then destaged to long-term storage asynchronously. Compression is applied intelligently through the same Storage Accelerator: each file is assessed first, and only compressible files are processed (zstd by default, with a range of algorithms available), applied inline at ingestion or asynchronously during destage. The net effect is lower latency for ingest waves and a smaller long-term footprint, with no client changes required.



Multi-tenant performance governance

At scale, the platform must protect itself and its tenants from one another. Without limits, a single tenant can monopolize metadata operations or bandwidth, starving others, and guaranteed performance becomes impossible to offer. ADI provides granular rate limiting and quality-of-service controls: per-bucket and per-account operation-rate limits, per-bucket and per-account bandwidth throttling, and independent control over GET, PUT, and DELETE rates. Operators use these to isolate workloads (capping test accounts so they cannot affect production), to enforce contracts (linking account limits to purchased capability), and for runaway protection (stopping a misbehaving job before it pages someone at 3 a.m.). The same mechanism turns performance into a sellable tier rather than a best-effort hope. Scality ADI formalizes this as S3 bucket rate limiting: a maximum request rate set globally or per bucket, with over-limit requests rejected using a configurable error code.

Scaling without migration or downtime

Underlying all of the above is the property that ADI scales in every direction while online. Horizontal scale adds nodes, from three to thousands, across one or many sites, and capacity grows linearly with no migration, no downtime, and no re-architecting. Vertical scale adds drives to existing nodes; they are detected automatically, are compatible across hardware generations, and integrate with no service interruption. Namespace scale grows with the cluster to hundreds of billions of objects per bucket and hundreds of millions of buckets, because metadata is fully distributed with no central server to bottleneck. These are not theoretical limits: ADI is deployed in production at multiple hundreds of petabytes across the globe today.

IX. Workload Tiering, Lifecycle, and Media

The right tier for every workload, and one lifecycle that moves data between them without copying it to a foreign system.

The four temperatures introduced in the architecture section are not marketing labels; each is a distinct combination of access software, media, and topology, chosen to deliver a specific latency or throughput target at the right cost. This section examines them and the lifecycle machinery that connects them.

The four temperatures in depth

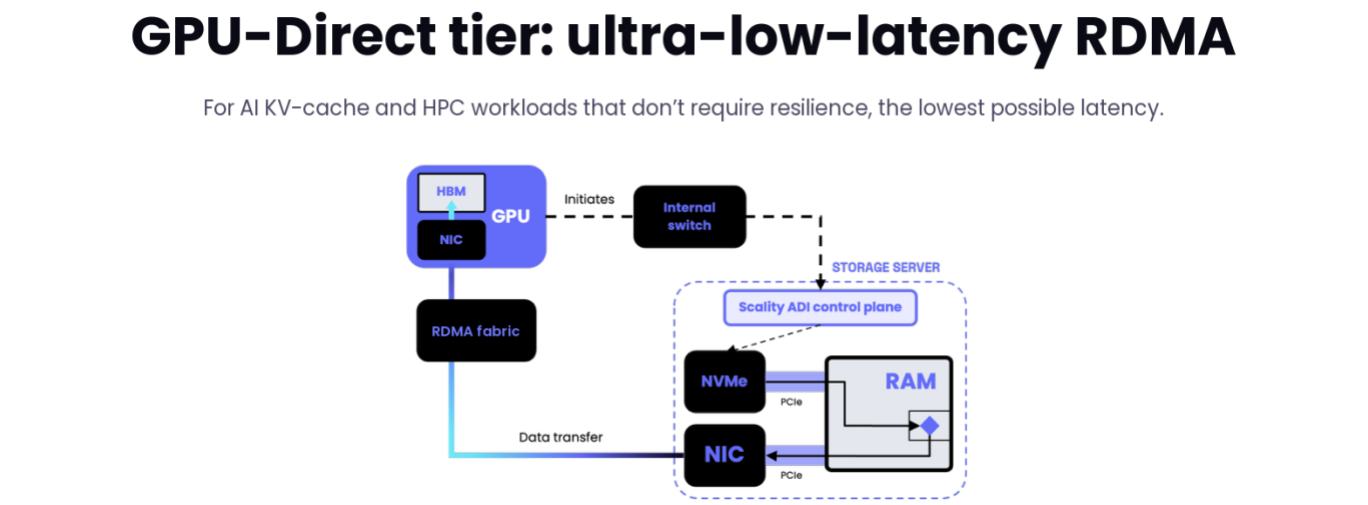
- **GPU-Direct. Ultra-low latency for AI KV-cache and HPC workloads that do not require resilience. It uses the GPU-Direct AI connector on high-performance TLC in an aggregated, single-site layout, targeting sub-50 microsecond latency by deliberately trading durability for speed where the data is ephemeral.**
- **Hot. Performance with resilience for resilient GPU-Direct, fast analytics, and fast backup. It uses S3 over RDMA or S3 over TCP on TLC/QLC/NL-SSD, single-site, targeting multi-terabyte-per-second throughput.**
- **Warm.** Performance with economics for sovereign cloud and backup at scale. It uses S3 over Fabric TCP on NL-SSD/NL-HDD/HDD in a (dis)aggregated, multi-site layout, targeting terabyte-per-second throughput at capacity economics.
- **Cold.** Maximum economics for long-term archive and retention, using the XDM lifecycle sidecar to tape and cloud targets with on-demand SLAs.

Feeding the GPU directly

The GPU-Direct tier exists because AI accelerators need data delivered to GPU memory, not to a CPU buffer that is then copied. ADI implements two GPU-Direct models.

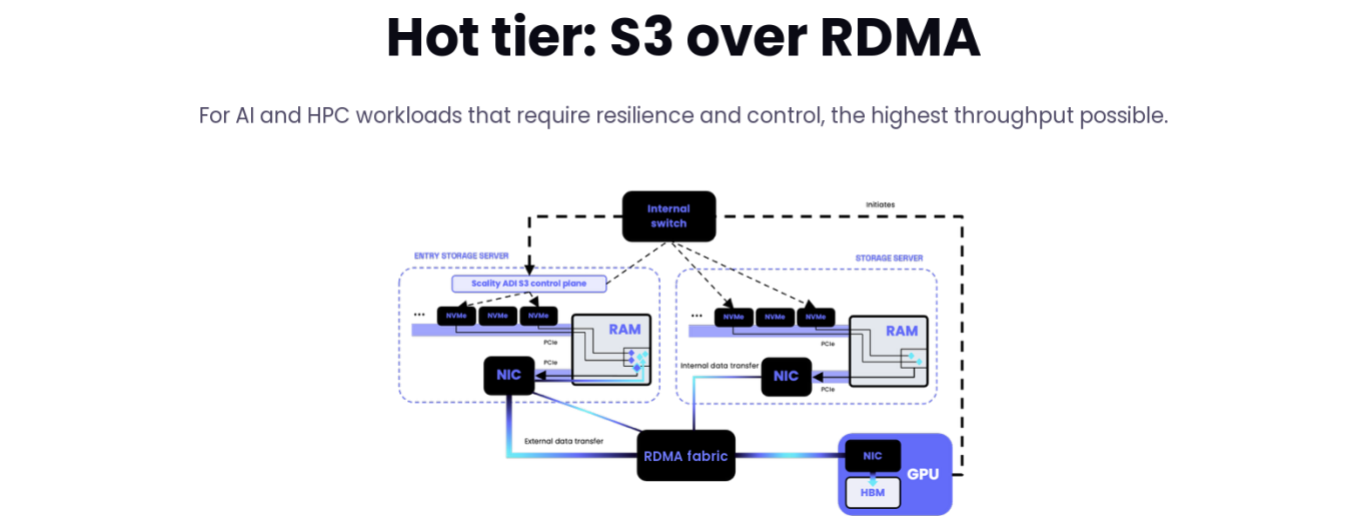
GPU-Direct tier: ultra-low-latency RDMA

On the GPU-Direct tier, data moves between GPU high-bandwidth memory and NVMe in a single zero-copy RDMA transfer that bypasses the CPU and the kernel, with a sub-50 microsecond latency target. With class of service 0, each block lives on a single drive, with no fan-out and no inter-node data movement, so the path stays short and predictable under load. The GPU-Direct tier is deliberately non-resilient, at class of service 0 there is no replica and no parity, but this is a considered trade, not a gap. The data it holds (KV-cache and HPC) is ephemeral and regenerable, so recomputing a lost block is cheaper than protecting it, and the durable copy of the underlying model or dataset lives in the Hot or Warm tier of the same namespace. GPU-Direct is a cache in front of durable storage, never the system of record.



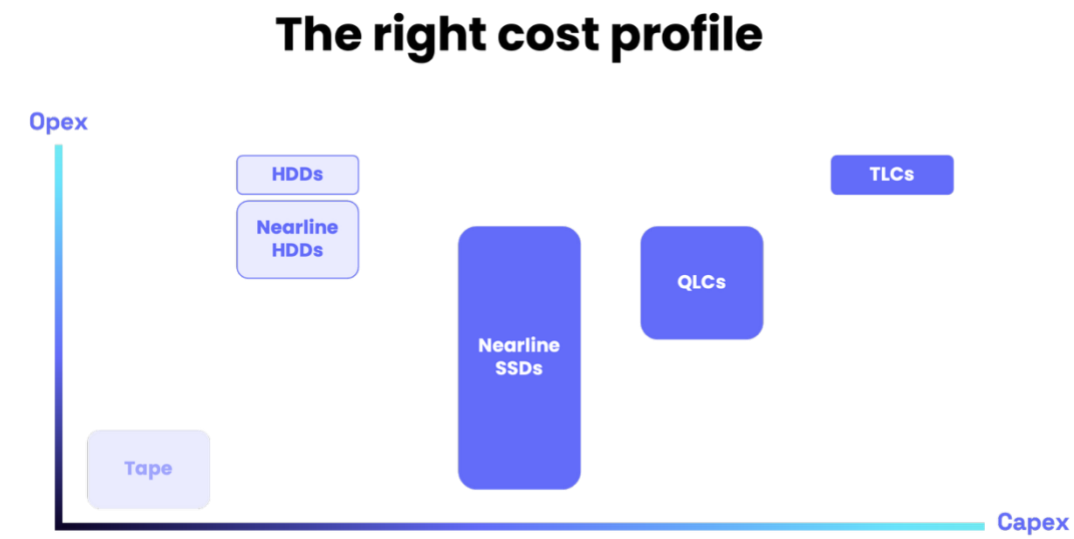
Hot tier: S3 over RDMA

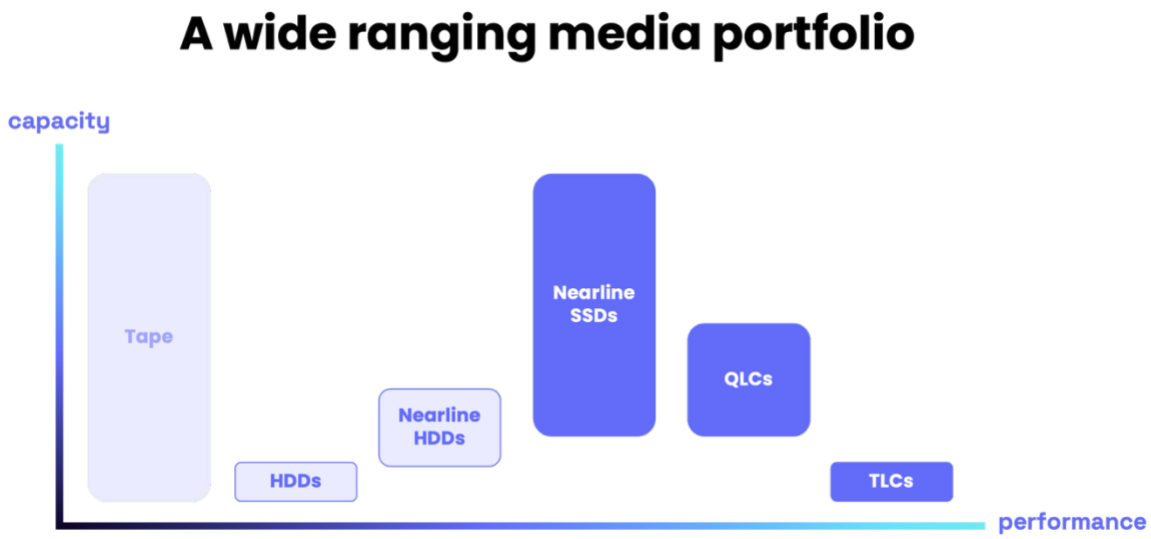
For workloads that need resilience and control alongside high throughput, Scality ADI serves standard S3 requests with the payload carried over RDMA directly to or from GPU memory. The application keeps the S3 API and toolchain it already uses. Because the data is erasure-coded, an object written from GPU memory is protected across the cluster as it lands, and a read is reconstructed and delivered to the GPU with the same durability as any other S3 object. RDMA is supported across the full media ladder, on NVMe and on HDD-based servers; the HDD validation figures are in the Performance at Scale section.

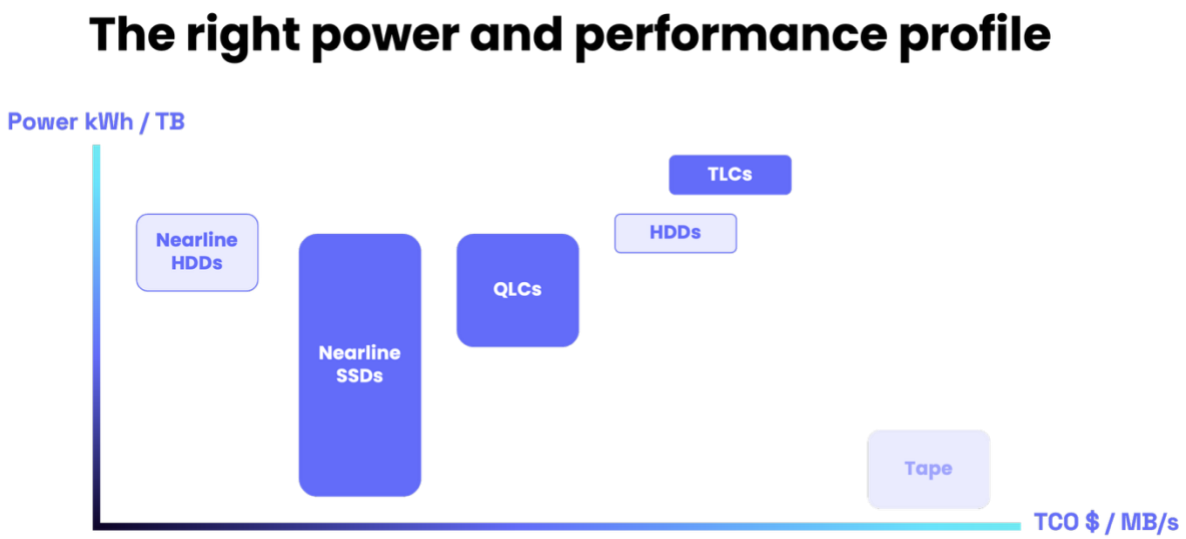


Media portfolio and cost profiles

Because ADI runs on commodity hardware, an operator can place each tier on the medium with the right cost, power, and performance profile rather than averaging one medium across the whole estate.





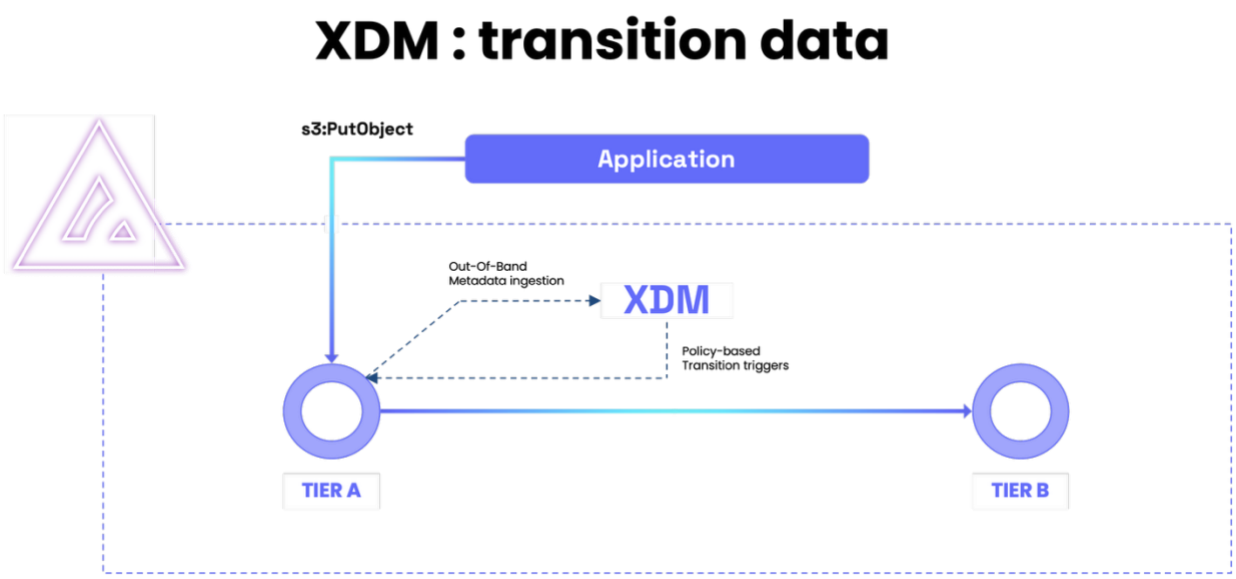


XDM: lifecycle transition and restore

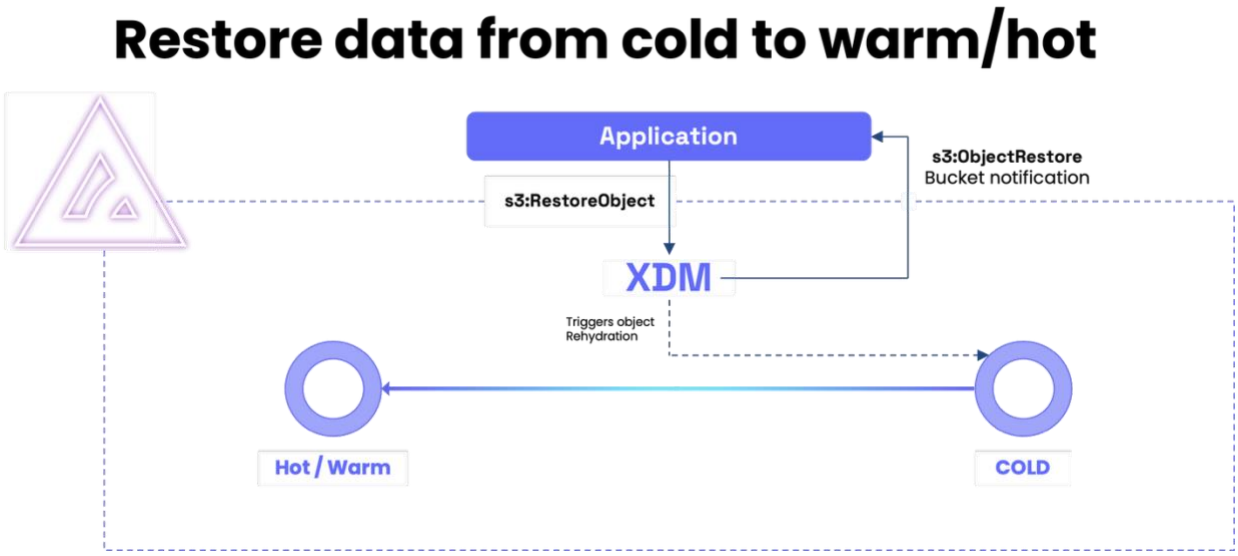
The eXtended Data Management (XDM) sidecar is what moves data between tiers without duplicating it or exporting it to a foreign system. It operates out-of-band via a mirrored S3 bucket, synchronizing metadata updates rather than copying data, so it adds lifecycle intelligence without sitting in the hot path. An operator defines transition rules once, by prefix, tag, or age, and XDM moves data from a source tier to a target tier cleanly, freeing capacity on the source.

XDM expresses lifecycle as policy and executes it out of band. Four flows cover the lifecycle:

- **Transition to a colder tier.** Out-of-band metadata ingestion ensures XDM is always in sync with tier A. Set a policy, by prefix, tag, or age, to move objects from a warmer Tier A to a colder Tier B and free capacity on tier A. .

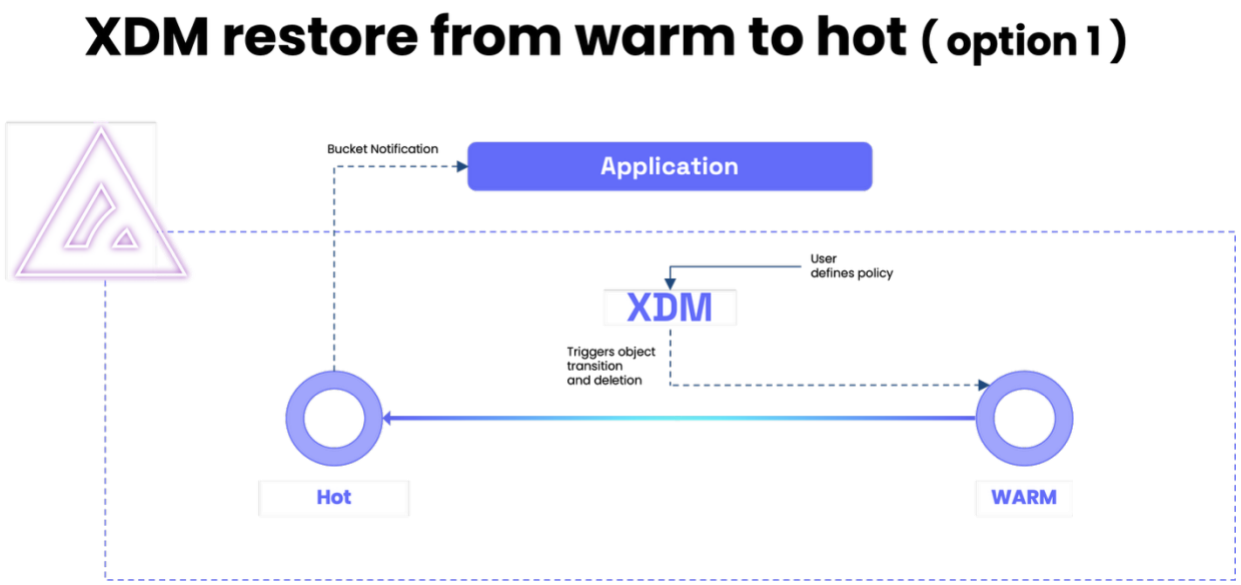


- **Restore from cold.** An application issues s3:RestoreObject and XDM rehydrates the object from the cold tier, the path that applies to tape and other cold targets, with an s3:ObjectRestore bucket notification signaling when it is available again.

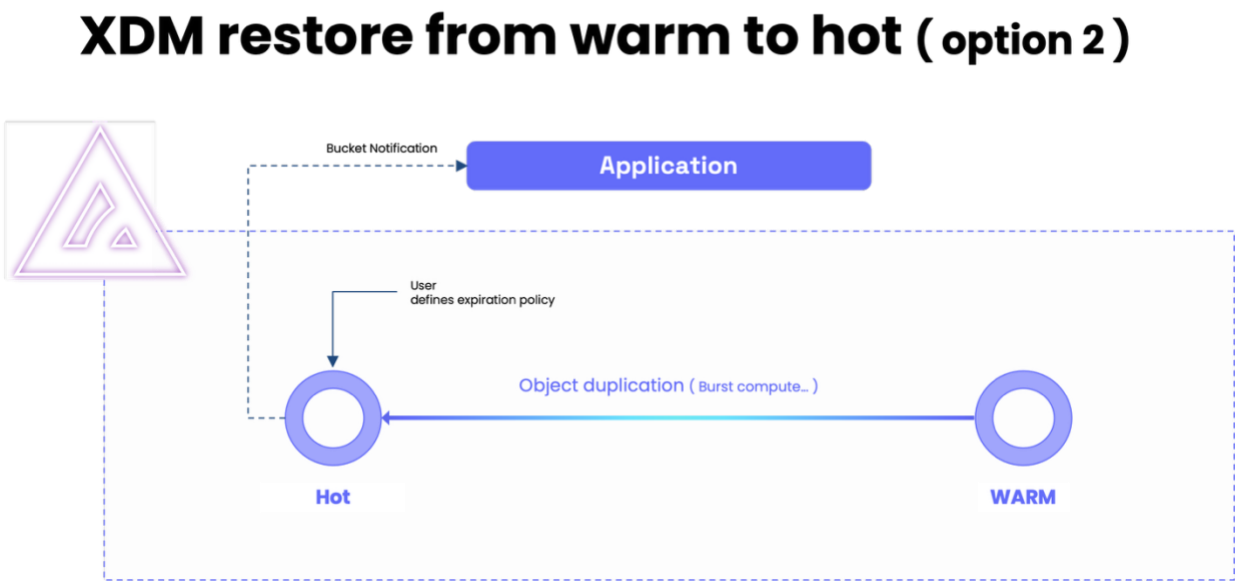


- **Warm to hot, option 1, transition and delete.** A user-defined policy promotes an object from warm to hot and removes the warm copy, signaled by a bucket

notification.



- **Warm to hot, option 2, duplication for burst compute.** For burst-compute scenarios the object is duplicated to the hot tier under a user-defined expiration policy, leaving the warm copy intact.



Sustainability by design

Workload-aligned tiering is also the platform's sustainability story, because power is dominated by where data sits. In a representative estate, hot data is roughly 5% of capacity and draws full power; warm is about 30% at up to one twenty-fifth of the per-

terabyte draw; and cold is about 65% at near-zero power at rest. Aligning each slice of data to the right tier yields up to a claimed 20x overall power-efficiency improvement versus treating the whole estate as hot. The operational and cost dimensions reinforce this: one cross-temperature lifecycle with no silos and no separate policies, exabyte-scale failure domains that keep resilience built in, and an appliance-like experience with non-event upgrades so that scaling does not add planning, time, or headcount. One platform also means one procurement motion, avoiding the over-buying that siloed systems force.

X. AICONNECT: Enabling AI Workflows

Native integration for modern AI and HPC toolchains, Kubernetes-native provisioning, event-driven pipelines, and a global namespace that anchors the GPU memory hierarchy.

Beyond raw data movement, ADI integrates into the way AI and HPC platforms are actually built and operated: declaratively, on Kubernetes, with event-driven pipelines.

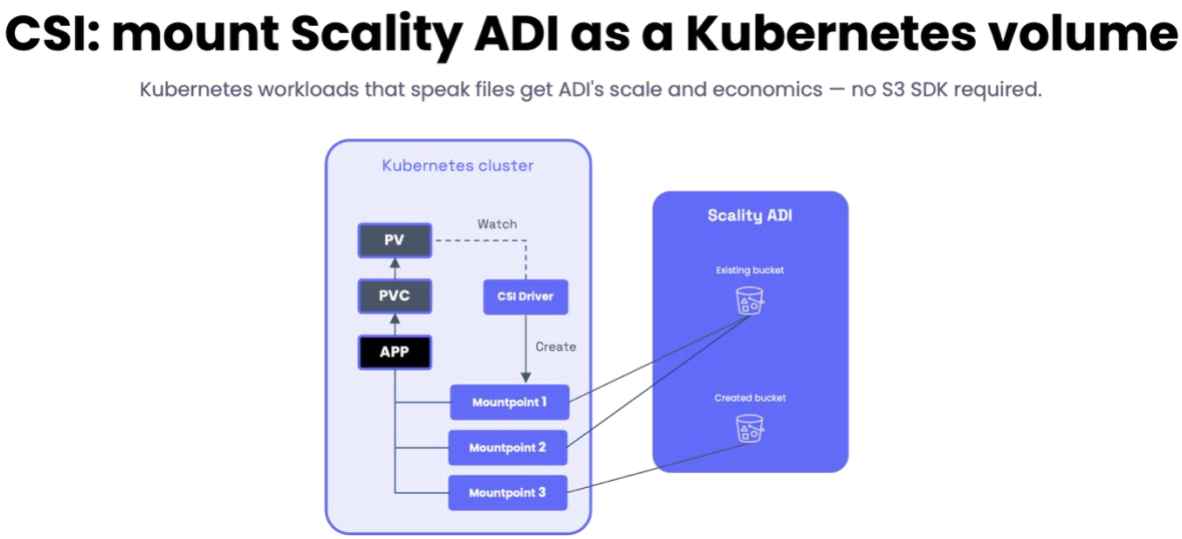
AICONNECT is the connective tissue that lets an AI or HPC platform consume ADI as native infrastructure rather than as a remote bucket. It spans four areas:

- **Kubernetes-native provisioning (CSI).** ADI is presented as a File volume, so clusters request and mount storage declaratively through standard PersistentVolumeClaims.
- **S3 buckets as Kubernetes resources (COSI).** Buckets are created, granted, and revoked as first-class Kubernetes objects through the Container Object Storage Interface, placing bucket lifecycle under the same GitOps control as the rest of the stack.
- **Event-driven pipelines.** S3 bucket notifications fire on object events, triggering downstream ingestion, transformation, and training jobs without polling.
- **A GPU-anchored namespace.** One protocol spans the GPU memory hierarchy, from RDMA-fed GPU-Direct and Hot tiers through warm capacity to cold archive, so data moves toward and away from compute by policy, not by manual copy.

Together these turn ADI from a storage endpoint into a programmable substrate for the full AI data lifecycle, ingest, prepare, train, serve, and retain, under one API and one security model.

CSI: ADI as a Kubernetes volume

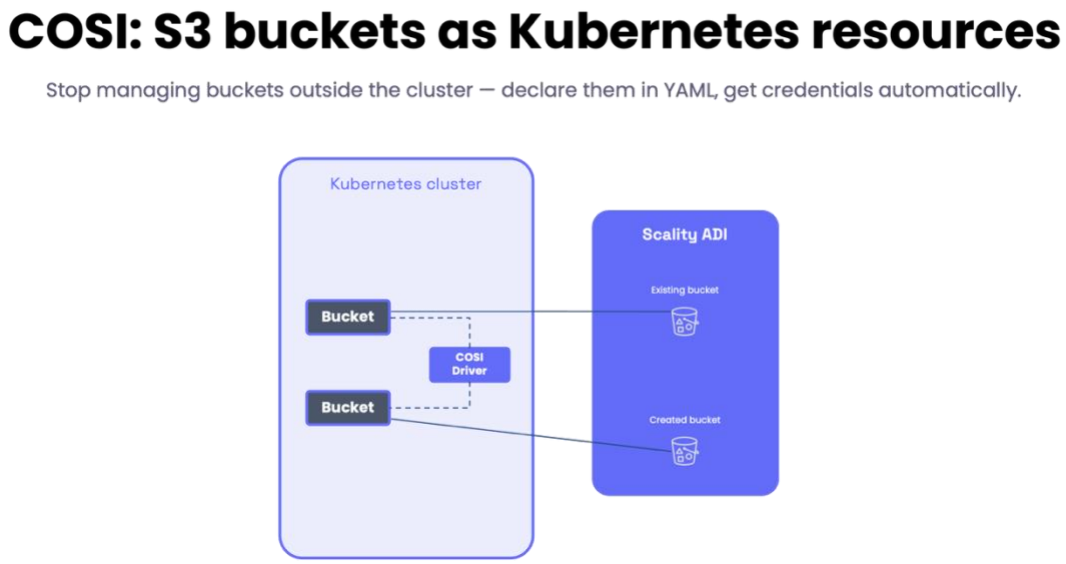
Many AI and HPC applications expect data at file paths, not behind an S3 SDK, and retrofitting them to call S3 directly is expensive and risky, while traditional file systems hit scale limits quickly and cost more. The ADI CSI driver resolves this by turning an ADI bucket into a mounted, POSIX-like volume: an application declares a PersistentVolumeClaim and the driver mounts the bucket on the node, with either static or dynamic provisioning and full multi-tenancy. It runs as a DaemonSet across the cluster, requires no changes to the application, and is deployed via Helm. File-based workloads thus get ADI's scale and economics through standard file paths with no code changes.



COSI: S3 buckets as Kubernetes resources

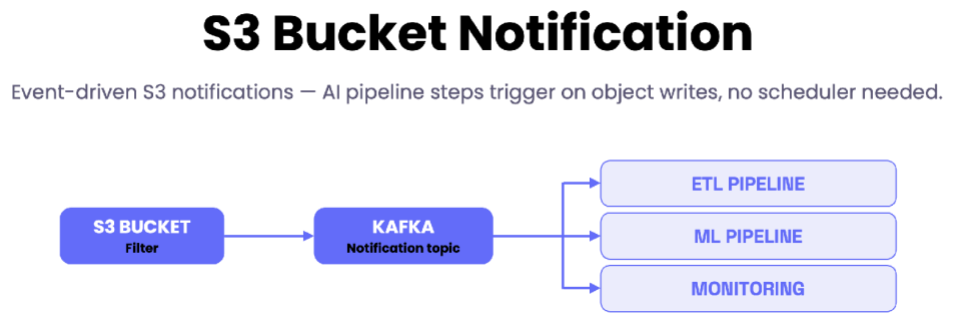
For applications that do speak S3, COSI brings bucket provisioning into the same declarative workflow as everything else in Kubernetes. Traditionally, storage lives outside the cluster: every bucket needs a ticket, a script, or keys pasted into a ConfigMap, which is manual, error-prone, and breaks GitOps. COSI is to object storage what CSI is to block and file, a standard driver API. Applying a BucketClaim makes the COSI controller create the bucket on ADI, and a BucketAccess request lands credentials in a Kubernetes Secret

automatically, with no manual steps. The bucket follows the application: it is deployed, versioned, and deleted with it, auditable and GitOps-compatible.



S3 bucket notifications for pipelines

AI pipelines are event-driven, and ADI's bucket notifications let each step trigger on object writes with no scheduler and no polling loop. Notifications fire immediately on events such as ObjectCreated, ObjectRemoved, and ObjectRestore, can be filtered by prefix and/or suffix, and are delivered to open destinations like Kafka topics or RabbitMQ queues via a queue ARN, no proprietary lock-in. The pattern is write-and-forget: each pipeline subscribes and reacts on its own. Typical uses include kicking off preprocessing when new files land, triggering a validation job when a model or index is written, and resuming downstream jobs automatically when an archived dataset is restored to a warmer tier.



The hierarchy of reasoning needs

Taken together, these capabilities let ADI span the full AI memory hierarchy, from GPU high-bandwidth memory at the top, through S3-over-RDMA, down to the global namespace that anchors the ephemeral tiers beneath it. The same data is reachable as an ultra-low-latency KV-cache, as a resilient high-throughput object store, and as cost-efficient capacity, without leaving the namespace. ADI is certified against the stages of the AI data pipeline, collection, filtering, cleaning, training, fine-tuning, and inference, so that one platform underpins the workflow end to end.

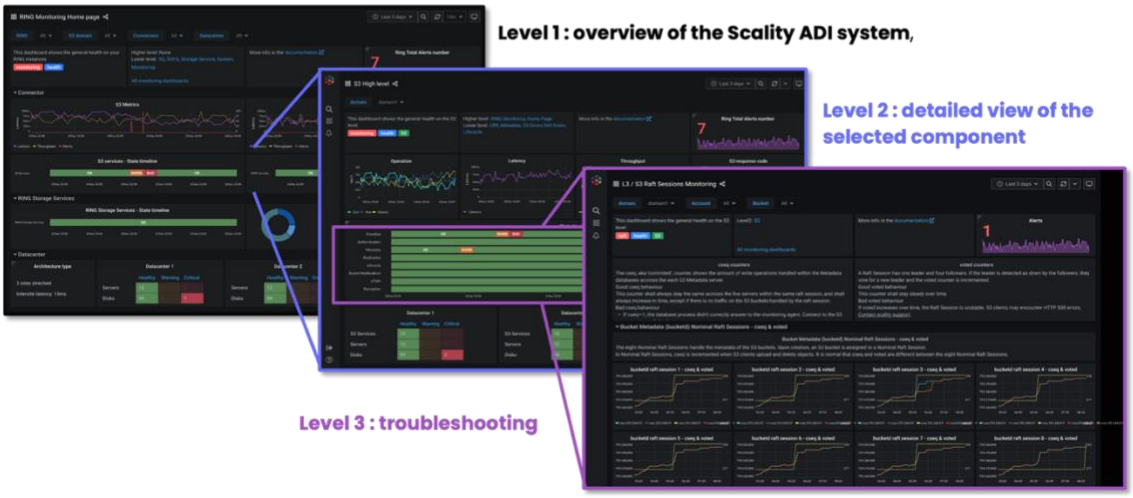
XI. Guardian: Autonomous Operations

The "A" in ADI. Conversational analytics, anomaly detection, and telemetry exploration, through a built-in agent or your own AI stack over MCP.

Operating storage at exabyte scale is itself a scaling problem: the number of metrics, alerts, and possible failure modes grows with the cluster. Guardian is ADI's answer, an operations layer that combines deep conventional monitoring with an AI agent, so that what an operator does for one operation, an agent can do for a thousand.

Three layers of monitoring

Conventional observability is structured in three levels: a system-wide overview, a detailed view of any selected component, and a troubleshooting level for root-cause work. ADI exposes more than 2,000 metrics through Grafana and Prometheus dashboards with full alerting. Troubleshooting can proceed top-down, identify the faulty connector, then the faulty service (metadata, IAM, S3, or RING connector), then the faulty component, or bottom-up, where an alert raised in a software component propagates up through domain and system status. The two directions meet in the middle, which is what makes diagnosis fast even in a large cluster.

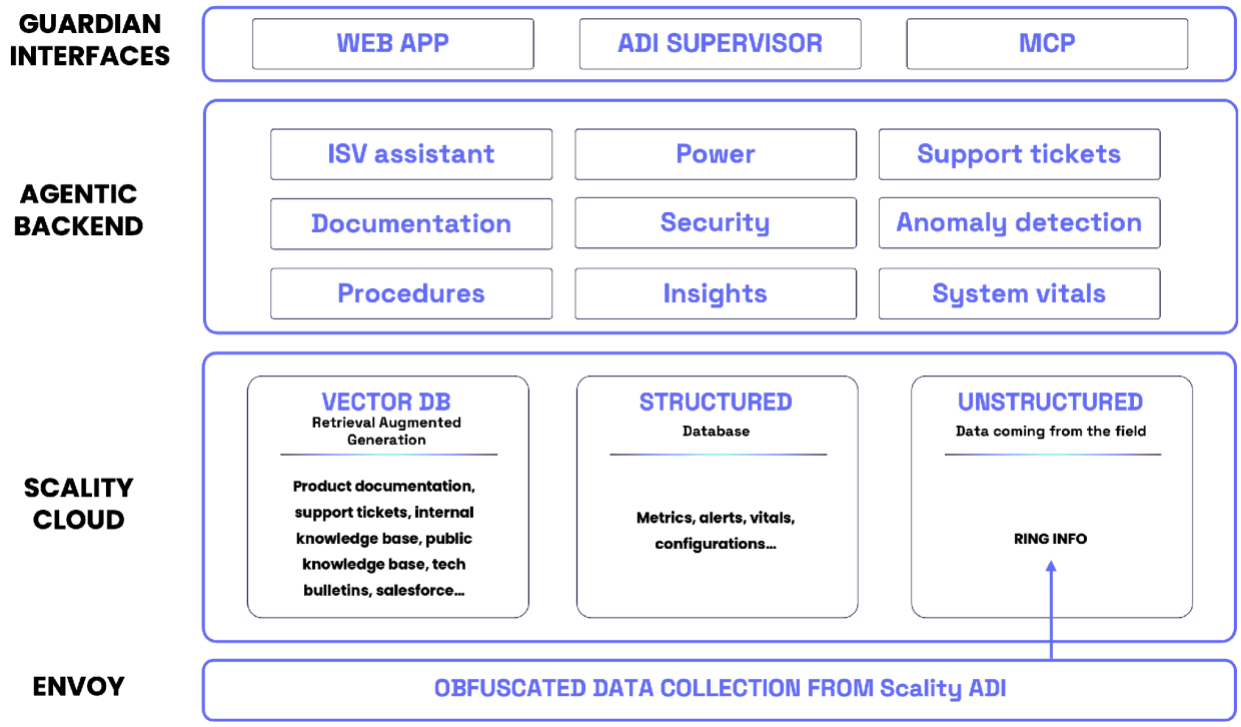


Guardian: AI-powered storage intelligence

Guardian adds five AI-driven capabilities on top of the telemetry: continuous monitoring of system vitals (health, capacity, power, performance); anomaly detection that gives early warning of disk failures, capacity trends, and misconfigurations; actionable insights derived from patterns observed across the entire installed base, not just the local cluster; augmented troubleshooting that walks an operator through guided root-cause analysis with step-by-step resolution; and smart documentation that allows natural-language search across all technical documentation.

How Guardian is built

Architecturally, Guardian is a multimodal AI stack that leverages field data. Telemetry is collected from ADI in obfuscated form and routed (via an Envoy proxy) to the Scality Cloud, where an agentic backend reasons over the combined corpus. Then data engineering works happen to structure the data and index it in a vectorial database that LLM can leverage. Operators reach Guardian through a web application or through MCP.



Agentic operability over MCP

The most forward-looking element is that ADI exposes its operations through the Model Context Protocol, the emerging open standard for connecting AI agents to infrastructure, backed by a broad industry group. Because the interface is MCP, ADI plugs into whatever AI stack an organization is already building: Claude or any other agent can drive it. The model is intent over mechanics, an operator describes the desired outcome and the agent sequences the underlying API calls for review. Agents do real, programmatic tool use, not just chat: they can create, configure, and query ADI inside multi-step reasoning chains; provision storage automatically as upstream systems demand it; and treat ADI as one tool alongside compute and networking in a larger orchestrator. The payoff is operational leverage, one instruction replacing dozens of API calls, UI steps, and documentation lookups, at any scale.

XII. Deployment and Reference Architectures

An ADI deployment is assembled from the four component roles introduced earlier, connectors, storage servers, supervisor, and network, sized and arranged according to the target tier or tiers. A minimum of three storage servers establishes a deployment with meaningful data protection; bulk data sits on SATA/SAS HDD or NVMe SSD while metadata sits on dedicated NVMe TLC mixed-use flash, reflecting the rule that metadata

is a small fraction of capacity but benefits most from low latency. The supervisor (a 1U server or VM) provides management and monitoring out of the data path, and standard Ethernet carries data, with RDMA-capable networking added for the GPU-Direct and Hot AI tiers.

Because connectors are stateless, production deployments place a load balancer in front of them, typically a dedicated hardware balancer, and scale operations or throughput by adding connectors. The three network planes (data, management, out-of-band) are kept separate for both performance and security. Geographic resilience is configured by choosing among the topologies in the CORE5 section: stretched three-site for an RPO of zero seconds and an RTO of zero seconds, stretched two-site with a witness for an RPO of zero seconds and an RTO of zero seconds, or mirrored asynchronous deployments for looser objectives and longer distances.

Operationally, ADI is designed to behave like an appliance even at exascale: drives and nodes are added online and detected automatically, upgrades are non-events, and the same management plane and lifecycle policies span every tier. This is what allows a single team to operate a multi-hundred-petabyte estate without a proportional increase in headcount, and what lets the platform present an appliance-like experience over commodity hardware the operator chooses.

XIII. ADI in Context: Where It Fits, and Where It Doesn't

An honest read on where ADI is the right tool, and where another one is.

Architects evaluating ADI are rarely choosing between ADI and nothing. They are choosing between one unified platform and a best-of-breed stack of specialized systems. The useful question is therefore not whether some specialized system can beat ADI on a single axis, one usually can, but whether the operational, economic, and security cost of running several specialized systems is justified by the marginal gain. The honest comparisons follow.

Versus dedicated parallel file systems

Systems such as VAST Data, WEKA, and the established HPC file systems (Lustre, IBM Storage Scale / GPFS) are built to do one thing extremely well: saturate GPUs from an all-flash, single-tier scratch cluster. That is one tier of the problem, not the whole of it. ADI meets the latency-critical path on its own terms, the GPU-Direct and Hot tiers place data

directly in GPU memory over RDMA, and then does what a single-tier flash system structurally cannot: carry that same data, in one namespace, down through warm capacity to cold archive with no second product and no migration. The right question is therefore not who wins an all-flash micro-benchmark, but whether an organization would rather run one platform across the entire data lifecycle or bolt a fast island onto three other systems and migrate between them. A scratch file system is not where a 200 PB training corpus or a seven-year immutable backup lives, and that is the ground on which ADI is chosen.

Two structural differences compound this. On scalability, a scratch file system is tuned for a bounded, low latency working set, whereas ADI is built to scale to exabytes and tens of billions of objects in a single namespace, adding and retiring hardware generations online with no rebalancing event and no forklift migration. On governance, ADI brings multi-tenant quality-of-service, S3 Object Lock and versioning, IAM with short-lived credentials, end-to-end encryption, and audit-ready evidence into the same platform, controls a pure performance tier is not designed to provide.

Versus hyperscaler object storage

Amazon S3, Azure Blob, and Google Cloud Storage win on elasticity and the near-absence of operational burden. For a workload that is bursty, cloud-native, and unconstrained by data-residency rules, that combination is hard to beat, and ADI does not pretend otherwise. ADI competes precisely where those assumptions break: where data sovereignty and on-premises control matter (your jurisdiction, your KMS, your audit trail); where economics must stay predictable across sustained petabyte-to-exabyte capacity, with no per-gigabyte and egress meters running against a large, long-lived dataset; and where storage must sit adjacent to on-premises compute for performance. Because ADI is S3-compatible, the application code is identical either way, so the decision is about where the data must live and what it costs to keep it there, not about rewriting software.

Two points deserve to be stated plainly. On cost, public-cloud object storage is expensive at sustained scale: per-gigabyte charges accrue every month a large dataset simply exists, and egress and request fees turn routine reads, analytics, and recovery into a recurring tax, so a multi-petabyte, long-lived corpus is precisely the worst-fit workload for hyperscaler pricing. On performance, storage sitting in a distant region is the wrong

place for data that must feed on-premises GPUs: round-trip latency and shared, unpredictable throughput starve accelerators that ADI keeps fed by placing data adjacent to compute over RDMA. For sustained, sovereignty- or performance-sensitive estates, the public cloud is usually both the costliest and the slowest option, not the cheapest and easiest.

Versus the two-tier status quo

The configuration ADI most directly replaces is the familiar pairing of a fast NAS or scale-up filer with a separate tape or cloud archive. Here the comparison is cleanest: two systems, two namespaces, two security models, and a migration step between them, versus one of each. Every boundary removed is a class of operational risk and policy drift removed, the same unification argument that runs through this paper, restated in procurement terms.

Where ADI is not the answer

Using a platform well includes naming where it does not fit. ADI is a distributed, minimum-three-server system built for scale; it is not aimed at a single-server edge footprint, at transactional or low-latency block and database workloads that belong on a different storage class, or at an organization that has deliberately standardized on fully managed public cloud and has no sovereignty, cost, or performance reason to run its own infrastructure. The unification thesis is strongest, and ADI is the right call, where the estate is large, long-lived, multi-temperature, sovereignty- or ransomware-sensitive, and spans both AI and conventional workloads at once. That is the environment the architecture was designed for, and the one where the cost of the alternative, several silos and the migrations between them, is highest.

Where the need is genuinely small-scale or backup-first, a single-site, mid-market footprint that wants ransomware-proof object storage without operating a distributed platform, the right Scality answer is ARTESCA, not ADI. ARTESCA is the simple, backup-first member of the family, ready from roughly 20 TB and growing into the petabytes, and it shares ADI's DNA: the S3 API, CORE5 cyber-resilience, and an inherently immutable engine. The natural path is to start with ARTESCA for backup and grow into ADI (and RING) when the estate becomes large, multi-temperature, and AI-scale. Naming that boundary is part of using ADI well, it is the platform for the large, long-lived, multi-workload estate, and it points cleanly to its sibling for the cases it is not built to serve.

XIV. Conclusion

Scality ADI is an argument that the proliferation of storage silos, a parallel file system for AI, an object store for the cloud, an archive target, and a separate console for each, is an accident of history rather than a necessity. By building one distributed, inherently immutable storage engine, presenting it through both object and file protocols over a single namespace, and letting data move between four temperature tiers by policy rather than by migration, ADI delivers the throughput AI demands, the economics archive demands, and the security that ransomware and regulation demand, from one platform.

The architecture's three load-bearing ideas are worth restating. First, immutability lives in the engine: because the store never overwrites in place, the cyber-resilience guarantees of CORE5 rest on a structural property rather than a revocable policy, which is why not even a fully privileged administrator can neutralize protected data. Second, every scaling axis is independent: capacity grows by adding storage nodes, throughput by adding stateless connectors, and metadata operations by adding RAFT sessions, sharding hot buckets, and migrating them online, so the system has no architectural ceiling and no forced migrations. Third, performance is proven at scale, on production paths with encryption and load balancing in every request, rather than at benchmark peak.

For an architect evaluating where AI, sovereign cloud, backup, and archive data should live for the next decade, the question ADI poses is whether the boundaries between those workloads need to exist in the storage layer at all. The evidence in this paper is that they do not.

Further resources

- Scality product documentation, installation, configuration, API reference, and release notes for ADI / ScalityOS.
- Companion whitepapers: the Scality RING and ARTESCA technical whitepapers, which detail the underlying storage engine and the cyber-resilient backup use case in depth.
- Reference architectures and sizing guidance, available from Scality solutions engineering for specific tiers and workloads.