COHESITY

Version 2.5

July 2024

Cohesity Fault Tolerance—Data Integrity for Modern Web-scale Environments

ABSTRACT

In today's digital economy, data is the most valuable asset for any organization. With data resilience as one of the guiding principles, Cohesity's fault-tolerant system is designed to ensure data integrity under all conditions—near full compute, memory, or storage capacity, network congestion, or hardware (motherboard, memory, or disk) failures.

Cohesity has been architected from the ground up to handle the hardware failures that can become especially disruptive in large, web-scale environments. The goal of this white paper is to show how Cohesity addresses resiliency with its fault-tolerant, distributed architecture. This document first goes through an overview of Cohesity's underlying components and architecture, then explains various fault domains and failure scenarios, and finally demonstrates the flexibility Cohesity provides to configure the ideal resiliency.

Table of Contents

Figures

Tables

Introduction

Cohesity provides a hyperconverged solution that simplifies management of all secondary data and applications by converging secondary workloads like backups, file and object services, test/dev, and analytics into a single software-defined, web-scale platform. Cohesity's platform integrates software, compute, storage, and network into a single stack.

Certified hyperconverged platforms are organized as nodes and can be networked together in a linear web-scale architecture to form a Cohesity cluster and provide a unified view of all the secondary storage data. With a pay-as-you-grow model, Cohesity's platform can be architected to scale from as few as three nodes to tens of thousands of nodes, and store petabytes of data. In web-scale models such as with hyperscalers like Google and Facebook, hardware failures are expected, and the architecture is designed to ensure data integrity.

Similarly, Cohesity is designed to tolerate failures and provides the highest levels of data resiliency without any performance degradation—even under such failures. Cohesity provides fault tolerance at both the software and hardware layers.

Some of the primary tenets that underlie the Cohesity operations are:

- Data should always be strictly consistent across nodes at all times, in all configurations, even if a node has failed, and even during upgrades.
- There should be no single point of failure, even during upgrades.
- New writes must not be put at risk, session information must be retained so that inflight writes are not interrupted.
- Data should always be available and should be serviceable.
- There should be no data loss in the system.
- The system should be capable of healing itself quickly.

Architecture

Cohesity provides a fully distributed architecture. Every node in the cluster runs the same set of software. Cohesity software operates as a set of cooperating services that each perform a specific function and provide a single application. These services communicate with each other within the same node and also communicate with services on the other nodes on the same cluster, providing a unified set of features.

An advantage of such an architecture is that a defect in one service doesn't affect any other service, and improvements to each service can be made independently of the other as long as the communication interfaces remain the same. Each software component within Cohesity has been built with a focus on resilience.

Some of the features of the Cohesity architecture include:

- **Strongly consistent**. A read always returns the most recently written value. Consistency is achieved using the Paxos algorithm. Reading the same object on different nodes by different clients is guaranteed to return the same value. Strong consistency is maintained even during node upgrades.
- **Consistent hashing**. Data is spread across the cluster in a uniform way using a consistent hashing algorithm.
- **Fully distributed**. No single point of failure. Data is distributed on other nodes depending on the chosen EC (Erasure Coding) or RF (Replication Factor) policy.
- **Self-balancing properties**. Data is redistributed automatically when nodes are added to or removed from the cluster. Redistribution is fast, as we only move the minimum amount of data required for rebalancing.
- **No single point of failure**. When a node or a disk fails, another node or disk will serve reads and writes. When the failed component comes back, it is automatically healed via a seamless integration.
- **No disruption on failure**. When there is a failure, new writes continue to be written with the same redundancy levels. Session state information is retained so that running jobs are not interrupted, even during node software upgrades.
- **No disruption on upgrade**. Cohesity's clustered architecture allows for a sequential node-by-node upgrade. Nodes can be upgraded to the latest versions without any interruption to client IOs. Any active IO access through VIP (Virtual IP address) is transferred as the VIP is seamlessly transferred to another node.

Distributed File System Consistency

One of the biggest challenges for any distributed system is to ensure data and application consistency. Implementing strict consistency while maintaining optimal performance, and without adding more processing, can be challenging.

Even in systems where strict consistency might not be an absolute requirement, it is still an important part of delivering a flawless customer experience. For example, when a customer adds an item to buy, the shopping cart is updated, but the update is not visible across all nodes for some time, and especially under failures. The customer might notice an outdated cart, but no lasting harm would come from the inconsistency. Eventually, the cart will get updated and everything will be fine. However, imagine a situation when the customer makes a cash deposit at their local ATM and doesn't see confirmation of that deposit. Strict consistency is a must in such cases.

Enterprise applications served by distributed systems are driven by demanding availability and data integrity goals, and hence require strict consistency.

For example, if a database file is updated when data is written, it must be updated across other nodes on the cluster before the write is acknowledged. This way, even if the node on which the write happened goes down, the other nodes still carry a valid copy of that data.

Strict consistency enables safe use of instant restore. After an instant restore, until the storage is moved back to the original system, the backup system is the production system. Without strict consistency, if a node becomes unavailable, there will be an outage. Without strict consistency, committed production writes can be lost forever if a single node loses cached writes or fails outright.

Strict consistency ensures that backups continue uninterrupted if a node fails. Without strict consistency applied to the session information, in addition to the data itself, a single node interruption would fail the backup. This would delay protection and risk causing a scenario in which the backup did not complete on time. It also risks causing a backup to be restarted or production performance to be degraded. Likewise, without strict consistency of session information, an interruption during a restore could cause the restore to restart, which would delay recovery. See Figure 1 and Figure 2 below.

Figure 1: Strict Consistency

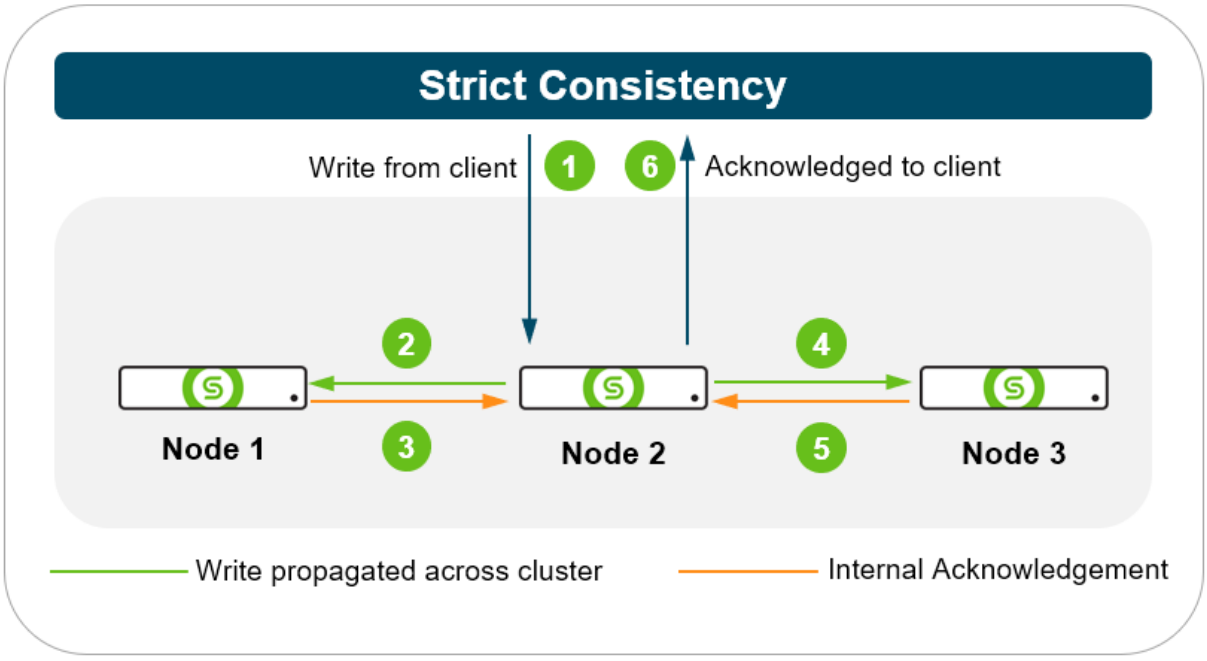
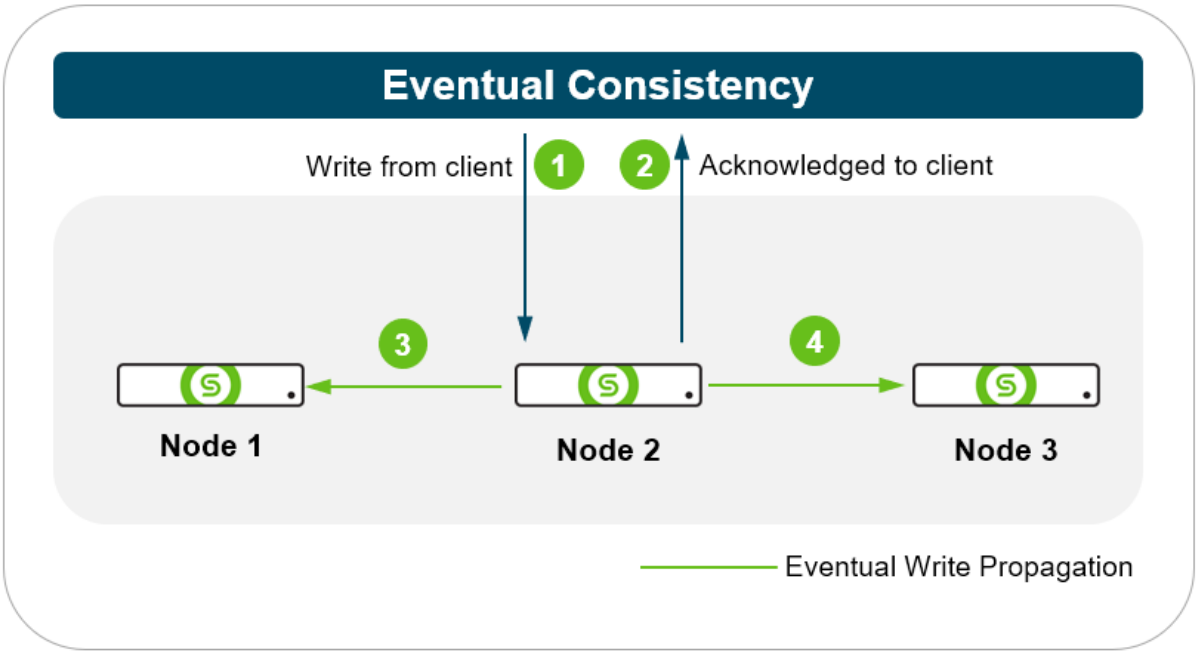


Figure 2: Eventual Consistency

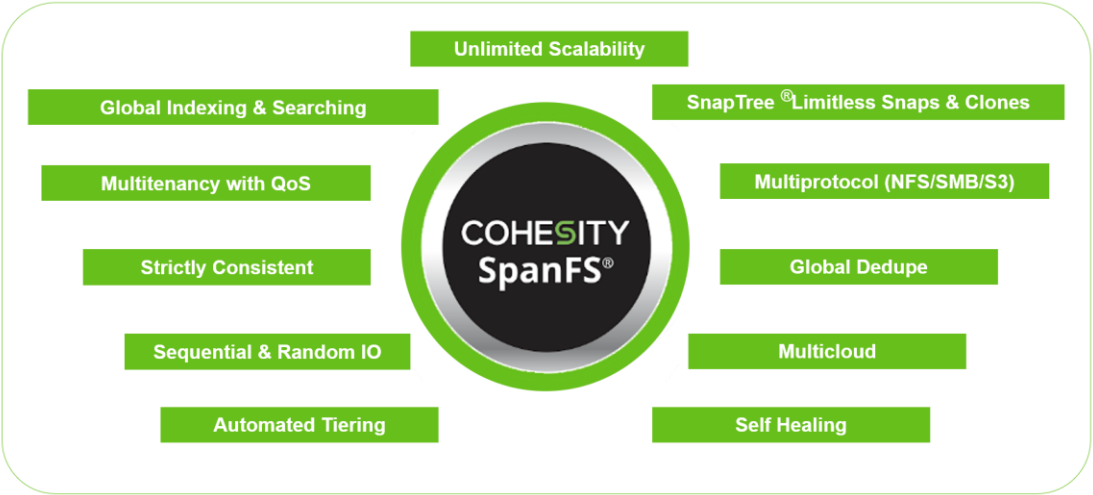


The SpanFS File System

SpanFS™ is the underlying web-scale file system. SpanFS is a fully-distributed file system that consolidates all secondary storage and eliminates legacy data and storage silos. It is the only file system that combines file services, global variable-length deduplication, and unlimited zero-cost snaps and clones, on a web-scale platform.

At the topmost layer, SpanFS exposes industry-standard, globally distributed NFS, SMB, and S3 interfaces. It also provides a proprietary interface to store backup data using the Cohesity DataProtect, a software-defined modern backup and recovery application.

Figure 3: Cohesity's SpanFS File System Features



At the lowest layer, SpanFS manages IO operations for all the data written to or read from the system. It detects random vs. sequential IO profiles, splits data, performs deduplication, and directs data to the most appropriate storage tier (SSD, HDD, or cloud storage) based on the IO profile. Random IO is placed on the SSD tier, while sequential IO is sent straight to HDD or SSD, based on QoS. Colder data might be sent to the cloud or brought back when it becomes hot. Data is distributed across the nodes in the cluster to maximize throughput and performance, and is protected with either replication factor or erasure coding.

The Data Repository stores the actual client data, such as network files, VMs, and databases in a deduplicated, compressed, and encrypted form. The Metadata Store keeps track of all the file data sitting across nodes. The Metadata Store, based on a Distributed Key-Value Store, incorporates a fully redundant, consistent, distributed NoSQL store for fast IO operations at scale. Cohesity's SnapTree® provides a distributed metadata structure based on B+ tree concepts. SnapTree is unique in its ability to support unlimited, frequent snapshots with no performance degradation. Concurrent accesses to the file Data Repository and the metadata are managed by the Distributed Lock Manager.

SpanFS has QoS controls built at all layers of the stack to support workload and tenant-based separation, and can replicate, archive, and tier data to another Cohesity cluster or to the cloud.

The file system Data Repository and the Data Journal are distributed on certified hyperconverged nodes, built with commodity x86 servers, available from Cisco, HPE, Dell, and Cohesity. Cohesity can also be deployed in the public cloud, on cloud VMs, and is also available on public cloud infrastructures.

Distributed Data Journaling

The SpanFS file system constantly looks at incoming requests and tries to estimate the IO pattern. Cohesity makes use of the distributed journal to serialize requests and provide faster performance by making use of faster SSD media. Random reads are first served by SSDs and if not found, are served by HDDs.

The journal absorbs IOs and thus acts as a write-back cache, which can be committed to hard disks at some later point. The journal helps with making data crash-consistent. The distributed journal is part of the metadata and is replicated along with the File Metadata Store.

Distributed File Data Resiliency

Cohesity presents a unified data store to the clients. Clients can access Cohesity nodes for backups (for example, VADP target data stores in the case of ESX) or as scale-out file and object services (for example, NFS, CIFS, and S3).

On each node, the underlying SpanFS file system is used to write to disks. All file data is stored on the Distributed File Data Store. All metadata is maintained on the Distributed Metadata Manager.

The unit of storage data that Cohesity uses for protection is a chunk file. A chunk file can be considered to be a collection of pieces of data from one or more client objects (files, VMs, etc.) packaged together into a single large unit. Cohesity takes a blob of storage, which can be a collection of one or more client objects, divides it into variable-sized, deduplicated chunks, compresses and encrypts them, and puts them in a chunk file. Usually, chunks from the same large client (user) file are combined to belong to the same chunk file. This will happen in most cases when the client file or VM writes are sequential and can be stored together. There may also be several smaller client files that are not large enough to form a single chunk file, in which case chunks from such client files could be packed together to form a chunk file.

A chunk file could be protected using either EC or RF schemes. Cohesity provides a configurable resiliency on HDDs or node failures.

Erasure Coding Scheme

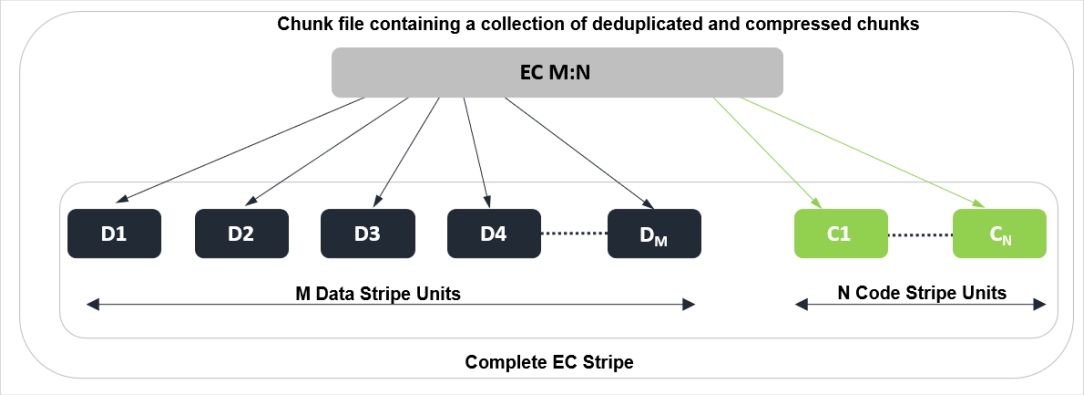
Erasure Coding refers to a scheme where a number of usable data stripe units can be protected from failures using code stripe units, which are in turn derived from the usable data stripe units. A single code stripe unit can protect against one data (or code) stripe failure, and two code stripe units can protect against two data (or code) stripe unit failures.

Each chunk file represents the data units of a single, complete EC stripe.

Cohesity supports up to three code stripe units, and hence can protect against three usable stripe unit failures. A complete data stripe unit is the set of usable data stripe units and their corresponding code stripe units.

Figure 4 illustrates an EC M:N scheme.

Figure 4: Example of an EC M:N Scheme

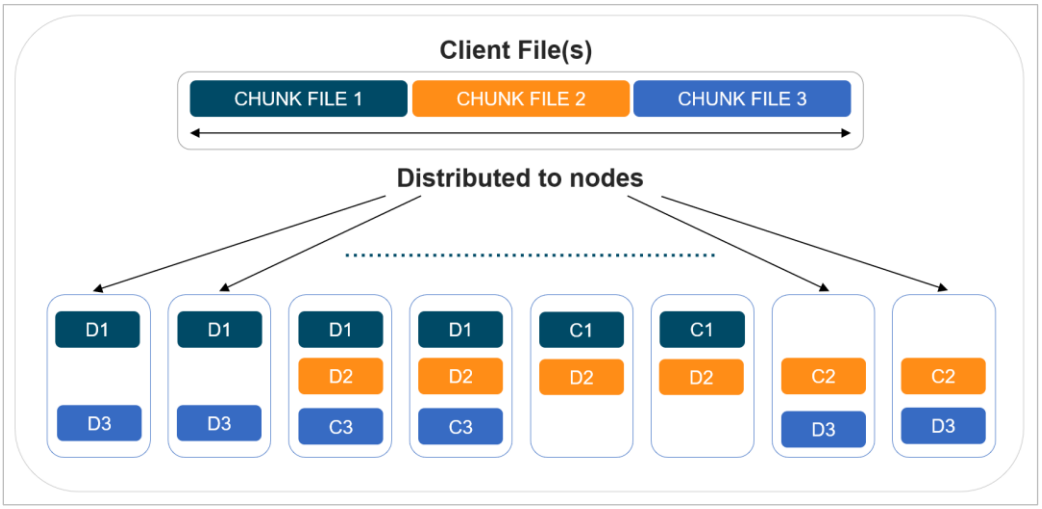


Cohesity Data Stripes

Note that in an Erasure Coding M:N scheme, a single chunk file is divided by the total number of M units to store in the M nodes. The code units are calculated from the data units and are stored in the remaining N nodes. In a Replication Factor scheme, the entire chunk file is replicated. Hence, a single, large file could be a part of several different chunk files and will end up getting distributed evenly across all the nodes of the cluster. Furthermore, each data or code unit from the same EC stripe will reside on a separate disk.

Figure 5 shows how one or more client files are represented by chunk files to form several EC 4:2 stripes to be distributed across all nodes equally.

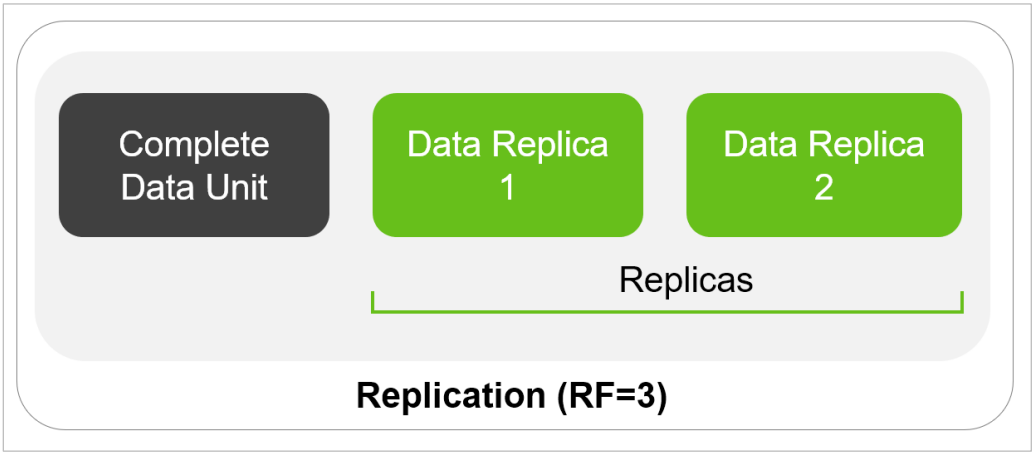
Figure 5: Single Chunk File Distributed Across Nodes



Replication Scheme

Replication Factor refers to the number of replicas of a unit of data. The unit of replication is a chunk file, and a chunk file is mirrored into either one or two other nodes depending on the Replication Factor number chosen. An RF2 mechanism provides resilience against a single data unit failure, and a RF3 provides resilience against two data unit failures.

Figure 6: Example of an RF Scheme



Distributed Metadata Resiliency

Metadata resiliency is the same as node resiliency on Cohesity. The Distributed Metadata Store is a fast key-value store containing all the metadata in the Cohesity cluster. Metadata is stored on SSD for performance reasons. It keeps track of all user objects stored on Cohesity nodes—VMs, files, S3 objects, clones, deduplication, metadata, etc. More deduplications can result in a larger Metadata Store size and Cohesity redistributes metadata equally among available nodes.

How metadata is replicated depends on the configured resiliency:

- **1 Failure**. Allows for either a single node or a single SSD failure.
- **2 Failures**. Allows for either two nodes or two SSD failures.

The minimum number of nodes required to sustain a single SSD failure is 3 and for two SSD failures is 5, to maintain Paxos quorum. We use the Paxos algorithm with 2-phase commit to ensure strong consistency for both metadata and data. This ensures that both metadata and data are always consistent across the required nodes and that there is always a replica available in case of a node failure.

Fault Tolerance in Cohesity

Cohesity provides fault tolerance at two levels: *Global Cluster Level* and *Storage Domain Level*.

Global Cluster Level fault tolerance settings specify how many node, chassis, or rack failures the cluster can survive while continuing to run and maintain strict consistency. Node, chassis, or rack failure settings at the Global Cluster Level protect against entire node, chassis, or rack failures respectively, for example, due to a catastrophic hardware (motherboard) failure or power outage on a rack.

Storage Domain Level fault tolerance can be configured to survive disk, node, chassis, or rack failures at a more granular level than the cluster as a whole. A Storage Domain configured to survive three-disk failures will not lose data even if there are simultaneous three disk failures. In addition, EC and RF levels allow for further file data resiliency at the stripe level.

For each Storage Domain, the user directly selects the number of disk failures and node failures to protect against. The system then presents the available choices for EC or RF, based on the size of the cluster. If the cluster is expanded later, more choices become available then; the system handles the conversion to any newly selected choices automatically.

Cohesity protects new writes redundantly across two nodes even while any one node is offline or inaccessible. For example, in a three-node cluster, it is typically sufficient to protect against one node failure while other scale-out storages would require protection against dual node failure.

Fault Tolerance Settings

These settings can be applied at an individual storage domain level.

Table 1: Failure Tolerance Settings

MINIMUM FAILURES TOLERATED	DESCRIPTION
0D:0N	Tolerate failure of 0 drive or 0 node. Using this setting, the available options to select are RF1. This is applicable only for Virtual Edition clusters.
1D:0N	Tolerate failure of 1 drive or 0 node. Using this setting, the available options to select are RF2. This is applicable only for physical ROBO clusters.
1D:1N	Tolerate failure of 1 drive or 1 node. Using this setting, the available options to select are RF2 or EC x:1.
2D:1N	Tolerate failure of 2 drives or at least 1 node. Using this setting, the available options to select include RF3 and EC x:2.

MINIMUM FAILURES TOLERATED	DESCRIPTION
	Cohesity will always make the best effort to place each EC stripe units to be on separate nodes when sufficient nodes are available. Only when sufficient nodes are not present will the EC stripe units be within the same node but will still be on separate disks.
2D:2N	Tolerate failure of 2 drives or 2 nodes. This setting is only available if there are at least four nodes. With this setting, the available options are RF3 or EC x:2, where the cluster size is at least x+2. Cohesity will always try to place each EC stripe unit on separate nodes when sufficient nodes are available.
3D:1N	Tolerate failure of 3 drives or 1 node. This setting is only available if there are at least three nodes. With this setting, the available options are RF4 or EC x:3.
3D:2N	Tolerate failure of 3 drives or 2 nodes. This setting is only available if there are at least seven nodes. With this setting, the available options are RF4 or EC x:3.
1D:1C	Tolerate failure of 1 drive or 1 chassis. This setting is only available if there are at least three chassis. With this setting, the available options are RF2 or EC x:1.
2D:1C	Tolerate failure of 2 drives or 1 chassis. This setting is only available if there are at least three chassis. With this setting, the available options are RF3 or EC x:2.
2D:2C	Tolerate failure of 2 drives or 2 chassis. This setting is only available if there are at least four chassis. With this setting, the available options are RF3 or EC x:2.
3D:1C	Tolerate failure of 3 drives or 1 chassis. This setting is only available if there are at least three chassis. With this setting, the available options are RF4 or EC x:3.
3D:2C	Tolerate failure of 3 drives or 2 chassis. This setting is only available if there are at least seven chassis. With this setting, the available options are RF4 or EC x:3.
1D:1R	Tolerate failure of 1 drive or 1 rack. This setting is only available if there are at least three racks. With this setting, the available options are RF2 or EC x:1.
2D:1R	Tolerate failure of 2 drives or 1 rack.

MINIMUM FAILURES TOLERATED	DESCRIPTION
	This setting is only available if there are at least three racks. With this setting, the available options are RF3 or EC x:2.
2D:2R	Tolerate failure of 2 drives or 2 racks. This setting is only available if there are at least four racks. With this setting, the available options are RF3 or EC x:2.
3D:1R	Tolerate failure of 3 drives or 1 rack. This setting is only available if there are at least three racks. With this setting, the available options are RF4 or EC x:3.
3D:2R	Tolerate failure of 3 drives or 2 racks. This setting is only available if there are at least seven racks. With this setting, the available options are RF4 or EC x:3.

The 2D in 2D:1N or 2D:2N has several effects at the same time:

- 2D allows selecting higher EC X:Y levels with fewer nodes. For example, an X+Y complete EC unit can be formed from n/2 nodes (assuming 2 disks per node taken for the single EC unit).
- Survive 2 simultaneous disk (HDD) failures or 2 simultaneous node failures with 2D:2N with EC X:2.
- Data stripe units from a single EC stripe are spread across as many nodes as possible. For example, if there are 10 nodes and EC 5:2 has been configured, the EC stripes will be spread across 7 separate nodes even if the protection level is 2D:1N. If there are 4 nodes, the EC 5:2 bits will be spread across 4 nodes with a 2D:1N protection level, with 3 of the 4 nodes having 2 stripe units each on separate disks.
- 2 node failures can be tolerated with EC X:2 if the number of nodes is at least X+2.

Disk and Node Resiliency

Cohesity provides the following protection levels for disk or node resiliency that can be set separately for each Storage Domain. The allowed values depend on the fault tolerance setting chosen. The following table shows possible EC settings that can be set for the required resiliency of hard drives and nodes. Cohesity supports node and disk failures, and the number of failures tolerated depends on the configured resiliency.

The user selects the minimum failures to tolerate, and the system adjusts the available options for RF and EC accordingly.

Table 2: HDD and Node Resiliency Settings

Failures Tolerated (Node Failures)	Fault Tolerance (Disk: Node Failures)	Max RF Levels Available (If EC not used)	Max EC Levels Available (If RF not used)	Min # of Nodes	Recommended # of Nodes (Min + Additional Nodes for healing)
1*	1D:1N	2	2:1	3	3+1
			3:1	4	4+1
			4:1	5	5+1
			5:1	6	6+1
	2D:1N	3	2:2	3	3+1
			3:2	3	3+1
			4:2	3	3+1
			5:2	4	4+1
			6:2	4	4+1
			8:2	5	5+1
	3D:1N	4	3:3	3	3+1
			4:3	3	3+1
			5:3	3	3+1
2	2D:2N	3	2:2	4	4+2
			3:2	5	5+2
			4:2	6	6+2
			5:2	7	7+2
			6:2	8	8+2
			8:2	10	10+2
	3D:2N	4	3:3	5	5+2
			4:3	6	6+2

Failures Tolerated (Node Failures)	Fault Tolerance (Disk: Node Failures)	Max RF Levels Available (If EC not used)	Max EC Levels Available (If RF not used)	Min # of Nodes	Recommended # of Nodes (Min + Additional Nodes for healing)
			5:3	7	7+2

*** If failures tolerated is set to 1 at cluster level and an XD:2N fault tolerance setting is selected for the storage domain, then metadata will not be protected against 2 node failure.**

Refer to the Best Practices section for the recommended Fault Tolerance as per the cluster size.

NOTE:

On clusters configured with minimum nodes, the healing will start only when the failed node/nodes are replaced. This leaves the data in a vulnerable state if the component holding the stripe fails before the failed nodes are replaced. With recommended nodes the healing starts immediately.

NOTE: It is recommended to configure RF2 on a 3-node cluster.

Chassis Resiliency

Cohesity provides the following protection levels for chassis resiliency that can be set separately for each storage domain. The allowed values depend on the fault tolerance setting chosen. The following table shows possible EC settings that can be set for the required resiliency of a chassis. The user selects the minimum failures to tolerate, and the system adjusts the available options for RF and EC accordingly.

Table 3: Chassis Resiliency Settings

Failures Tolerated (Chassis Failures)	Fault Tolerance (Disk: Chassis Failures)	Max RF Levels Available (If EC not used)	Max EC Levels Available (If RF not used)	Min # of Chassis	Recommended # of Chassis (Min + Additional Chassis for healing)
1*	1D:1C	2	2:1	3	3+1
			3:1	4	4+1
			4:1	5	5+1
			5:1	6	6+1
	2D:1C	3	2:2	3	3+1
			3:2	3	3+1

Failures Tolerated (Chassis Failures)	Fault Tolerance (Disk: Chassis Failures)	Max RF Levels Available (If EC not used)	Max EC Levels Available (If RF not used)	Min # of Chassis	Recommended # of Chassis (Min + Additional Chassis for healing)
2	2D:2C	3	2:2	4	4+2
			3:2	5	5+2
			4:2	6	6+2
			5:2	7	7+2
			6:2	8	8+2
			8:2	10	10+2
	3D:2C	4	3:3	5	5+2
			4:3	6	6+2
			5:3	7	7+2
	3D:1C	4	3:3	3	3+1
			4:3	3	3+1
			5:3	3	3+1
			4:2	3	3+1
			5:2	4	4+1
			6:2	4	4+1
			8:2	5	5+1

*** If failures tolerated is set to 1 at cluster level and an XD:2C fault tolerance setting is selected for the storage domain, then metadata will not be protected against 2 chassis failure.**

*** Refer to the Best Practices section for the recommended Fault Tolerance as per the cluster size.**

NOTE:

On clusters configured with minimum chassis, the healing will start only when the failed chassis are replaced. This leaves the data in a vulnerable state if the component holding the stripe fails before the failed chassis are replaced. With recommended chassis the healing starts immediately.

Rack Resiliency

Cohesity provides the following protection levels for rack resiliency that can be set separately for each storage domain. The allowed values depend on the fault tolerance setting chosen. Table 4 shows possible EC settings that can be set for the required resiliency of a rack. The user selects the minimum failures to tolerate, and the system adjusts the available options for RF and EC accordingly.

Table 4: Rack Resiliency Settings

Failures Tolerated (Rack Failures)	Fault Tolerance (Disk: Rack Failures)	Max RF Levels Available (If EC not used)	Max EC Levels Available (If RF not used)	Min # of Rack	Recommended # of Rack (Min + Additional Rack for healing)
1*	1D:1R	2	2:1	3	3+1
			3:1	4	4+1
			4:1	5	5+1
			5:1	6	6+1
	2D:1R	3	2:2	3	3+1
			3:2	3	3+1
			4:2	3	3+1
			5:2	4	4+1
			6:2	4	4+1
			8:2	5	5+1
	3D:1R	4	3:3	3	3+1
			4:3	3	3+1
			5:3	3	3+1
2	2D:2R	3	2:2	4	4+2
			3:2	5	5+2

Failures Tolerated (Rack Failures)	Fault Tolerance (Disk: Rack Failures)	Max RF Levels Available (If EC not used)	Max EC Levels Available (If RF not used)	Min # of Rack	Recommended # of Rack (Min + Additional Rack for healing)
			4:2	6	6+2
			5:2	7	7+2
			6:2	8	8+2
			8:2	10	10+2
	3D:2R	4	3:3	5	5+2
			4:3	6	6+2
			5:3	7	7+2

* **If failures tolerated is set to 1 at cluster level and an XD:2R fault tolerance setting is selected for the storage domain, then metadata will not be protected against 2 rack failure.**

*** Refer to the Best Practices section for recommended Fault Tolerance as per the cluster size.**

NOTE:

On clusters configured with minimum racks, the healing will start only when the failed rack is replaced. This leaves the data in a vulnerable state if the component holding the stripe fails before the failed racks are replaced. With recommended rack the healing starts immediately.

Resiliency Terminology

The following table explains some of the resiliency terms in greater detail.

Table 5: Resiliency Terminology

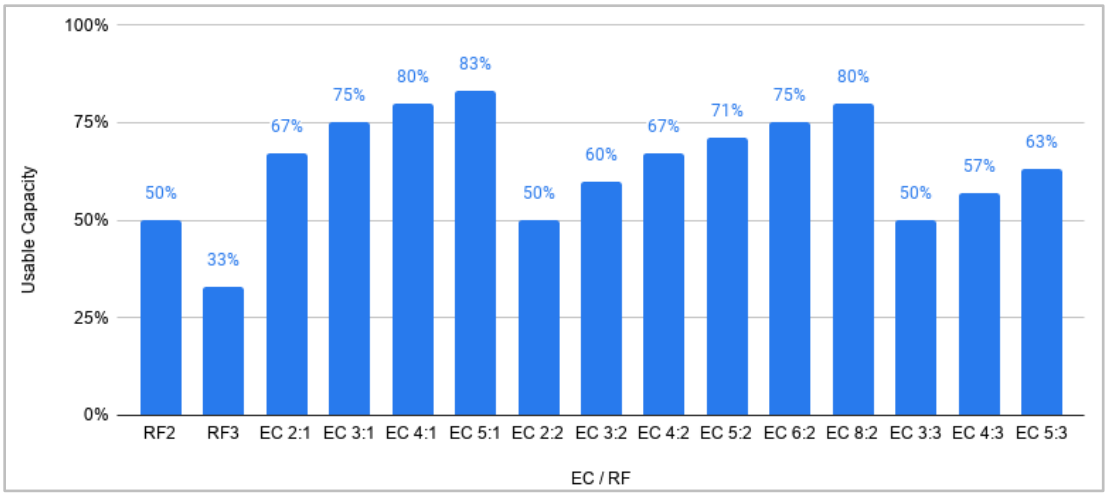
TERM	NOTES
Failures Tolerated	The number of disk, node, chassis, or rack failures that the Storage Domain must resist, in the form xD:y[N][C][R]. Example: 2D:1N: The Storage Domain will be able to survive at least two disk failures or at least 1 node failure. (And at most 2 nodes, if the EC level and the cluster size permit.)

TERM	NOTES
EC Settings	Allowable EC setting for each disk or node combination. Example: 5:2 means that for every five data bits, there are two erasure code bits, so that if any two bits are lost, they can be re-constructed from the remaining five.
Example, 2D:2N	2D suggests that there can be two stripe units of the same EC unit on the same node. 2D:2N also implies that the cluster can sustain 2 hard disk or 2 node failures.
RF Setting if EC disabled	Corresponding RF setting to achieve the same levels of resiliency. RF4 is used for a 3 hard disk or a 2-node resiliency. RF3 is used for a 2 hard disk or a 2-node resiliency. RF2 is used for a 1 disk or a 1-node resiliency.

Usable Storage

Figure 7 below shows the formatted usable capacity based on various EC and RF settings. For example, an EC X:2 provides a lower usable capacity than an EC X:1 scheme, but a higher resiliency by allowing for 2 stripe unit failures.

Figure 7: Usable Storage: Effective Capacity



Best Practices

As the number of nodes in a cluster increase, the cluster becomes more resilient to failures. Customers can leverage these fault tolerance best practices to deploy their clusters with increased resiliency. Once adopted, these recommendations help prevent data loss escalations and enable better utilization of your resources.

These recommendations focus on building resilience, not performance.

These guidelines do not apply to the VE or PXG platforms.

Hardware platforms from different vendors come with a variety of storage options. The below table describes different hardware platforms and the number of drives per node.

Table 6: Hardware Platforms and Drives Per Node

PLATFORM		DATA DRIVES PER NODE
Cohesity	C4000	3
	C5000	3
	C6000	12
	CX8000[AF]	6
HPE	Proliant DL360 G10[AF]	10
	Proliant DL380 G10	12
	Apollo 4200 G10	24
	Apollo 4510 G10	25
		50
Cisco	UCS C220 M5	3
	UCS C240 M5	12
	UCS S3260 M5	44 – Full shelf
		22 – Half shelf
		48 – Dual node

PLATFORM		DATA DRIVES PER NODE
	UCS C220 M5[AF]	10
	X201c M6[AF]	6
Dell	PowerEdge C6420	3
	PowerEdge R740XD	12
	PowerEdge R640[AF]	10
Fujitsu	PRIMERGY RX2540 M5	12
Lenovo	ThinkSystem SR650	12
Intel	R1208WF[AF]	8
	R2208WF[AF]	24
SuperMicro	6029P-E1CR12L	12
	1029U-TN12RV[AF]	12

NOTE:

[AF] indicates an All-Flash platform.

The following are the fault tolerance best practices for metadata and data:

Table 7: Scenario Types

SCENARIOS	TYPE	NO. OF NODES	NO. OF DRIVES	RECOMMENDATION
Scenario 1.1	Data	>= 16		2D:2N
Scenario 1.2		< 16	< 48	1D:1N
Scenario 1.3		< 16	>= 48	2D:1N
Scenario 2.1	Metadata	> 40		2 Failures*
Scenario 2.2		< =40		1 Failure*

NOTES:

- N is the number of nodes and D is the number of drives.
- *2 Failures indicate the cluster can tolerate two nodes or SSDs (metadata drives) failures and *1 Failure indicates that the cluster can tolerate one node or one SSD (metadata drive) failure.

For example, In Scenario 1.3, consider a C6000 series cluster with 10 nodes. Since the number of drives per node in a C6000 node is 12, the total number of drives per cluster would be 120 (12x10).

This would satisfy the condition of N<16 and D > 48. Hence the recommended fault tolerance for such a cluster would be 2D:1N.

Fault Tolerance Features

Cohesity's fault-tolerant architecture produces many significant benefits that result in enterprise-class resiliency and data integrity.

Self-Healing

The Cohesity self-healer process monitors the state of the cluster and performs maintenance tasks such as clean-up and healing. The self-healer is always running in the background and performs fast and predictable healing.

The self-healer is responsible for:

- Collecting garbage and fixing metadata (for example, dangling references from a failed backup or an unfinished rebalance).
- Rebalancing data between nodes (for example, when nodes are added or removed from the system) and auto-tiering data. It also performs individual disk-rebalancing.

The self-healer constantly validates both data and metadata and automatically heals in the event of hardware and software failures without external intervention by the user.

Drive Replacement and Recovery

When an HDD fails, the absence is immediately detected, and the disk is marked as unusable, and all data on the hard disk is automatically reconstructed on other available HDDs.

SSDs store all metadata on Cohesity. Metadata consistency is guaranteed by storing replicas on SSDs on other nodes. Other nodes continue to service metadata when an SSD fails. If an SSD fails or becomes corrupt, it can be replaced in isolation and the node will automatically heal when it is brought back up. On platforms with multiple SSDs, Cohesity ensures that the load on the SSDs is evenly distributed, to increase their lifetime.

Load Balancing and Rebalancing

Cohesity provides a uniform view of data from any node, and thus an external DNS load-balancer can balance incoming traffic loads across the nodes of the cluster, ensuring all nodes participate equally in serving client IOs and storing data. Cohesity also ensures that data on the nodes are continuously rebalanced so that throughput remains the same even during disruptions.

Data Validation and Active Repair

Cohesity ensures the integrity of data at all times. It constantly checks for silent corruption from memory, disk, or network errors, for example, and helps heal these errors. An example of silent disk corruption is a bit rot.

The self-healer performs constant background scans on data. A checksum is calculated for each chunk file, which is the unit of replication and forms an EC stripe. If there isn't a match, the chunk file is actively repaired.

The self-healer also actively validates metadata in the background, and if there is an inconsistency, such as a replica mismatch, the metadata is repaired.

Deduplication and Fingerprinting

Cohesity does not commit the whole file as-is into storage. It maintains a SHA-1 based hash list of deduplicated chunks. This list is global across the cluster, and applies for every write, no matter which service—such as the network file system or a VM backup—does the write.

Cohesity commits a new chunk only if the chunk from the file is not already found. Otherwise, it just points to the existing deduplicated chunks. Depending on the protection policy, these chunks might also be compressed before being committed as part of a larger chunk file.

The deduplicated chunk boundaries are calculated based on a Rabin fingerprinting mechanism. The Rabin fingerprinting makes it computationally efficient for the system to use a sliding window to recognize duplicate information: If an insertion or deletion causes the duplicate information to no longer align to the same fixed-block boundaries, the system will still recognize the information as duplicate. The system also uses variable block sizes. The result is greater space reduction when compared with systems that don't use sliding windows. The deduplication table containing the hash list is accessible across the cluster and prevents duplicate entries. The deduplication hash list is also used during archival to any object storage in the cloud, and during replication to prevent duplicates from being sent.

The self-healer checks for the consistency of the deduplication hash list and repairs it when, for example, a rebalance is stopped midway.

IP Failover and Load Balancing

Each node has a Virtual IP address, and clients can access any node on the cluster with its VIP. When the node that was directly serving a client goes down, the VIP seamlessly switches over to another available node. The client continues to access the same IP address and the fully distributed architecture allows Cohesity to access client data that is physically present on any node on the cluster. Cohesity depends on an external DNS server to do a round-robin on the VIP addresses so that client requests can be load-balanced.

Only Full Data Stripes and Rollback

When a stripe is being written, it is ensured that it is written completely on the chosen nodes. If, for example, there is a node or a disk failure on the node or disk on which one of the units was being written, it is rolled back, and a new set of node and disk combinations is chosen. A two-phase commit scheme is utilized to ensure all units for the stripes are consistently written across nodes.

Redistribution Only on Available Space

Redistribution on a node or disk failure only happens when available space on those remaining nodes/disks is above a threshold capacity. There is always a “virtual spare” that can be used to redistribute data. This provides the best balance between performance and capacity utilization, ensuring the units that have been redistributed do not get inefficiently redistributed back.

Immediate Reconstruction and Fast Healing

The Cohesity self-healer keeps monitoring for failures in the background at all times. When a failure occurs, the self-healing process immediately starts reconstructing the lost data on other nodes or other disks. On traditional RAID systems, where a RAID group is constructed out of a set of disks, rebuilding the stripes can take a long time as it is done entirely within a single node. RAID systems also need dedicated spare drives just to start the rebuild, whereas on Cohesity, data is chunked and spread evenly across the entire cluster and all nodes participate in the reconstruction process, leading to very fast reconstructions and healing. Spare capacity is taken from capacity available on any nodes. Erasure Coding in Cohesity is performed at much smaller granularities compared to whole-disk levels. This produces faster rebuild times compared to EC done at the drive level.

Resiliency Level Retention after Healing

Cohesity can sustain a simultaneous failure of up to 2 nodes or 3 drives. It is important to note that after Cohesity reconstructs data that has been lost, the same resiliency levels are returned after the healing. This means that, given immediate reconstructs and the faster repair process, the cluster is returned to the same state of resiliency that it started with. For example, if the cluster had been configured to sustain 2 node failures, it will again be able to sustain failure of 2 more nodes without any performance impact once healing is complete.

Same Resiliency with Fewer Nodes

Cohesity offers an additional 2D (two HDDs on the same node) protection level, which allows for striping data on two disks within the same node. This additional resiliency setting provides more flexibility of offering a higher resiliency for clusters with 3 or 4 nodes.

Continuation of Writes

Cohesity automatically switches to the appropriate level of redundancy when faced with node constraints so that write requests continue to be satisfied. For example, Cohesity will switch to an RF2 scheme in some situations when a node fails. In a 3-node cluster with EC 2:1 (1D:1N), Cohesity will automatically switch to an RF2 scheme when a node fails and switch back to EC 2:1 when a replacement node is provisioned.

Failure Scenarios

In this section, we look into a few failure scenarios and the responses that Cohesity uses to handle those failures.

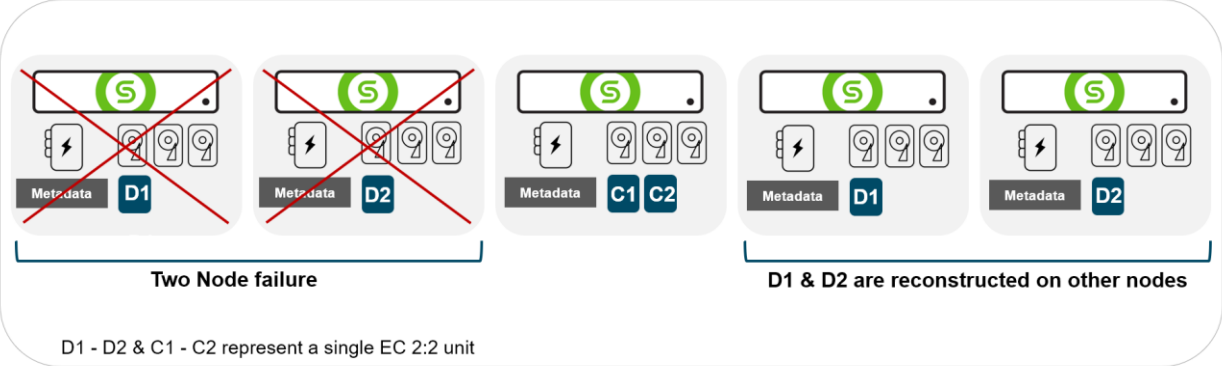
Two-Node Failures

Two-node failures are allowed when a node resiliency of 2N is configured.

File data is reconstructed from the surviving nodes that have units from the same stripes. This reconstructed data is placed in nodes that have available (spare) capacity.

File metadata is constructed from surviving nodes by copying the replicas from other nodes and placing them in nodes that have the most available SSD capacity. Note that the replacements are made to ensure the same resiliency as at the beginning of the cluster lifetime. See Figure 8.

Figure 8: EC 2:2 with 2D:2N Protection Level

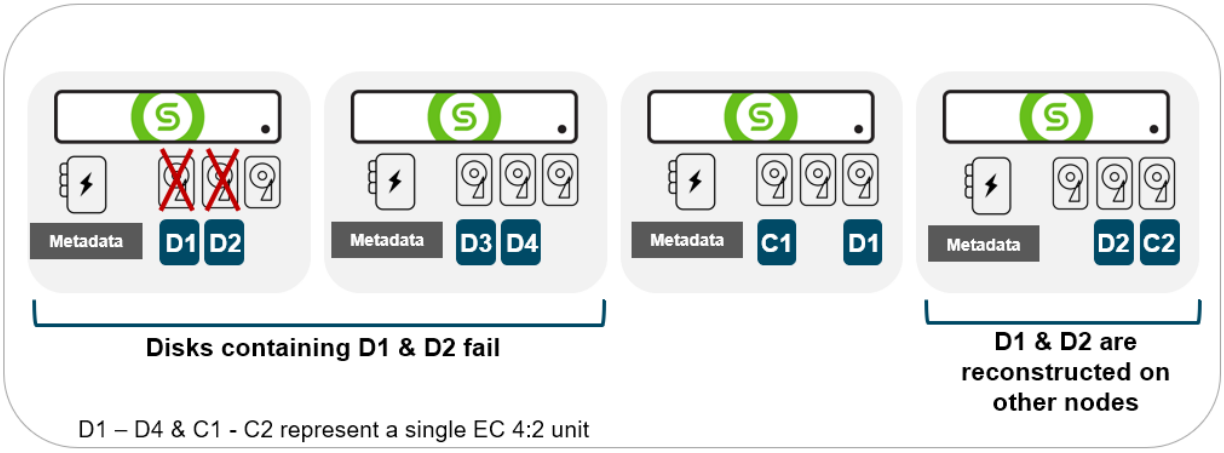


Two-HDD Failures

HDDs contain only file data. Two HDD failures on random nodes are allowed when a resiliency of 2D is configured.

File data is reconstructed from the surviving nodes that have units from the same stripes as the failed HDDs. This reconstructed data is placed in HDDs that have available (spare) capacity on the same node when possible. See Figure 9 below.

Figure 9: EC 4:2 with 2D:1N Protection Level

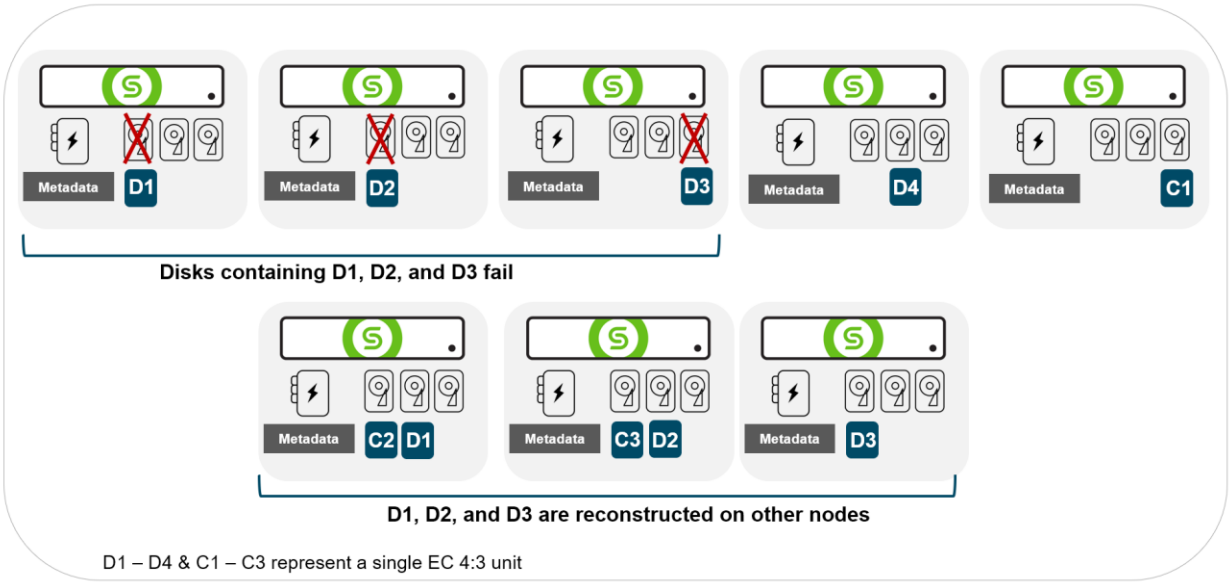


Three-HDD Failures

Three HDD failures on random nodes are allowed when a resiliency of 3D is configured. A minimum of 8 nodes in your cluster is required to handle three-HDD failures.

File data is reconstructed from the surviving nodes that have units from the same stripes as the failed HDDs. This reconstructed data is placed in HDDs that have available (spare) capacity on the same node when possible. See Figure 10.

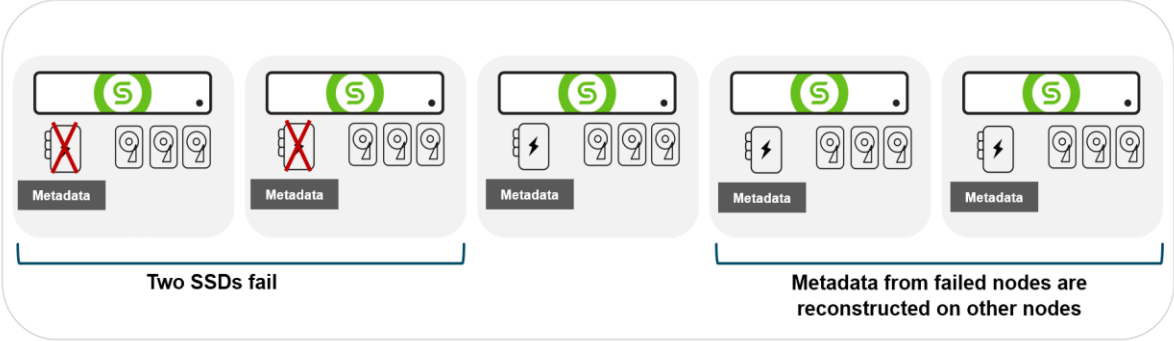
Figure 10: EC 4:3 with 3D:2N Protection Level



Two-SSD Failures

File metadata is always stored on SSDs. Up to two SSD failures are allowed when two-node failures are configured. In this case, 3 replica copies of the metadata are stored on the SSDs to ensure that a loss of any 2 replicas will still allow recovery from that loss. See Figure 11.

Figure 11: 2 SSDs Failure Protection with 3 Metadata Replicas

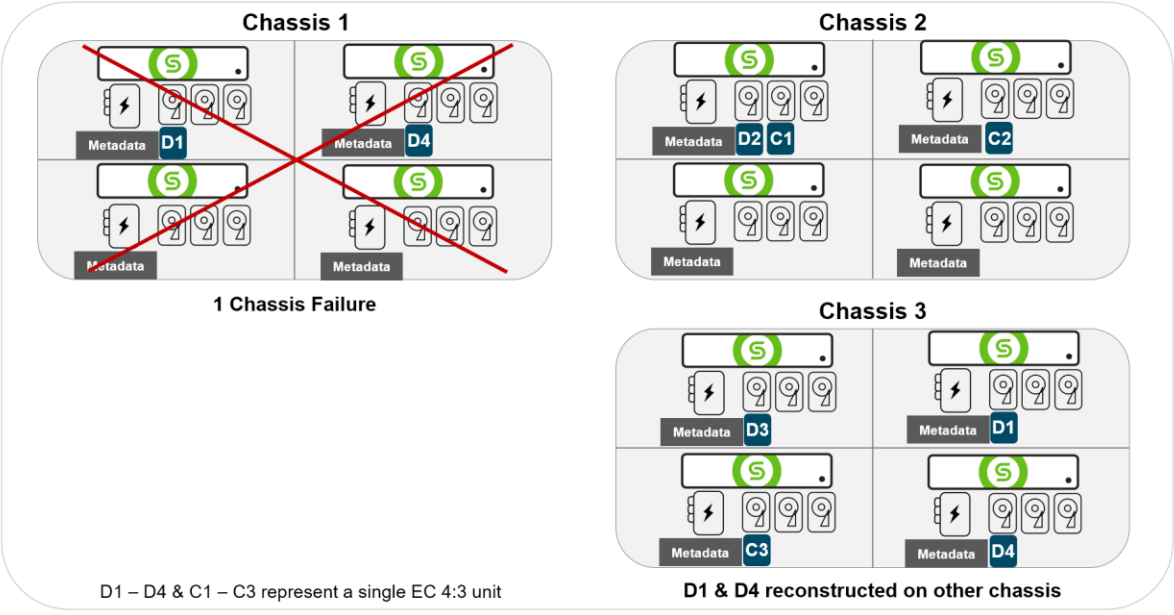


One-Chassis Failure

A maximum of two chassis failures are allowed in a Cohesity cluster. Figure 12 below illustrates one chassis failure with a storage domain resiliency of 3D:1C and EC 4:3. A minimum of 3 chassis in your cluster is required to handle one-chassis failure.

File data is reconstructed from the surviving chassis that have units from the same stripes as the failed chassis. This reconstructed data is placed in chassis that have available (spare) capacity.

Figure 12: EC 4:3 with 3D:1C Protection Level

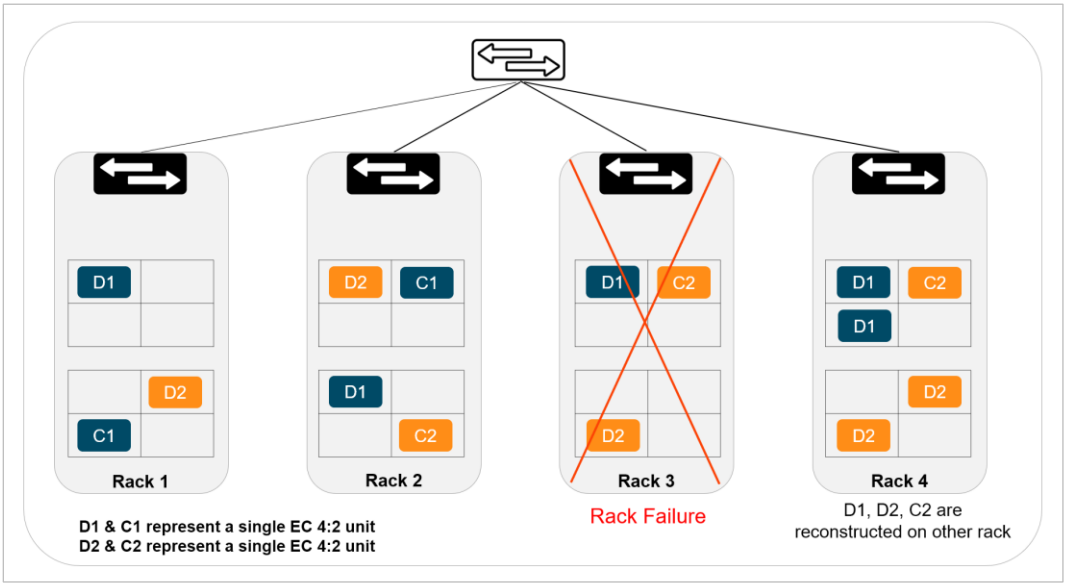


One-Rack Failure

A maximum of two rack failures are allowed in a Cohesity cluster. Figure 13 illustrates one rack failure with a storage domain resiliency of 2D:1R and EC 4:2. A minimum of 2 rack in your cluster is required to handle one-rack failure.

File data is reconstructed from the surviving rack that have units from the same stripes as the failed rack. This reconstructed data is placed in rack that have available (spare) capacity.

Figure 13: EC 4:2 with 2D:1R Protection Level



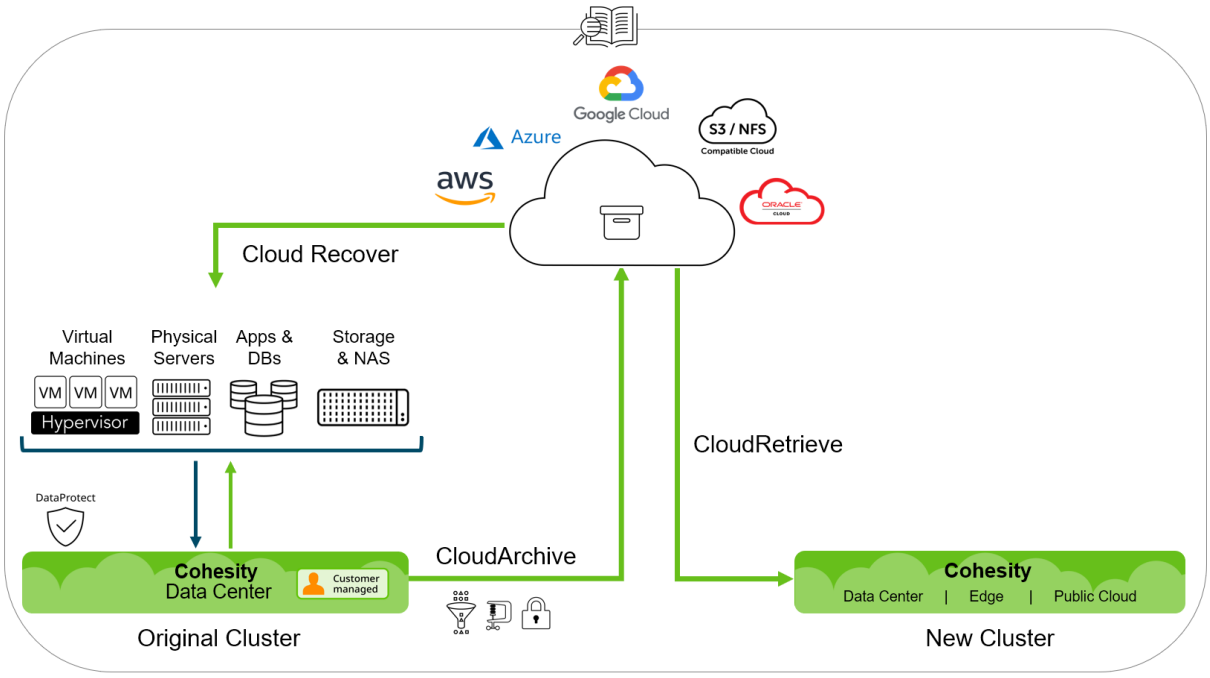
Disaster Recovery and Business Continuity

Ensuring business continuity during a disaster can be important for many businesses. Cohesity provides several options for disaster recovery (DR) that are fully integrated into the platform without the need for third-party solutions. Cohesity provides in-built archival and replication functionality to ensure business continuity.

CloudArchive and CloudRetrieve

Data can be archived to the cloud or an S3-compliant storage using CloudArchive. This data is entirely self-contained, which means that it contains both backup data and metadata and can be recovered to an entirely new Cohesity cluster using CloudRetrieve. See Figure 14.

Figure 14: CloudArchive Enables CloudRetrieve for Disaster Recovery



Native Replication

Cohesity also provides options to replicate data from a primary, on-premises cluster to a Cohesity cluster in the cloud or to another on-premises Cohesity cluster in your disaster-recovery site. With both options, Cohesity does a source-side deduplication and sends only changed data in increments to the other site. Cohesity also continues to scan and auto-heal on the other site.

Figure 15: Cohesity Native Replication to Cohesity Cluster on Premises

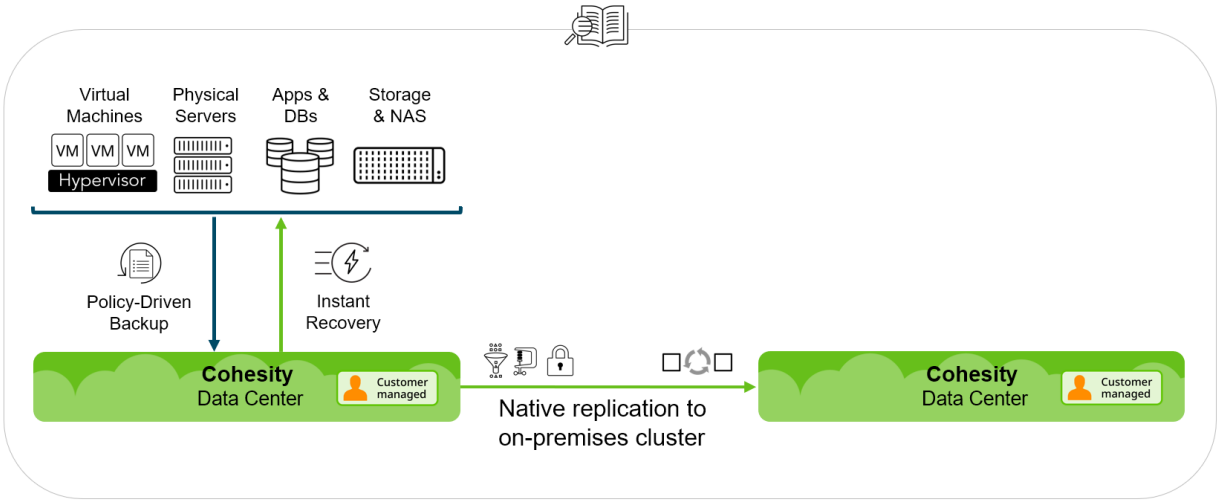
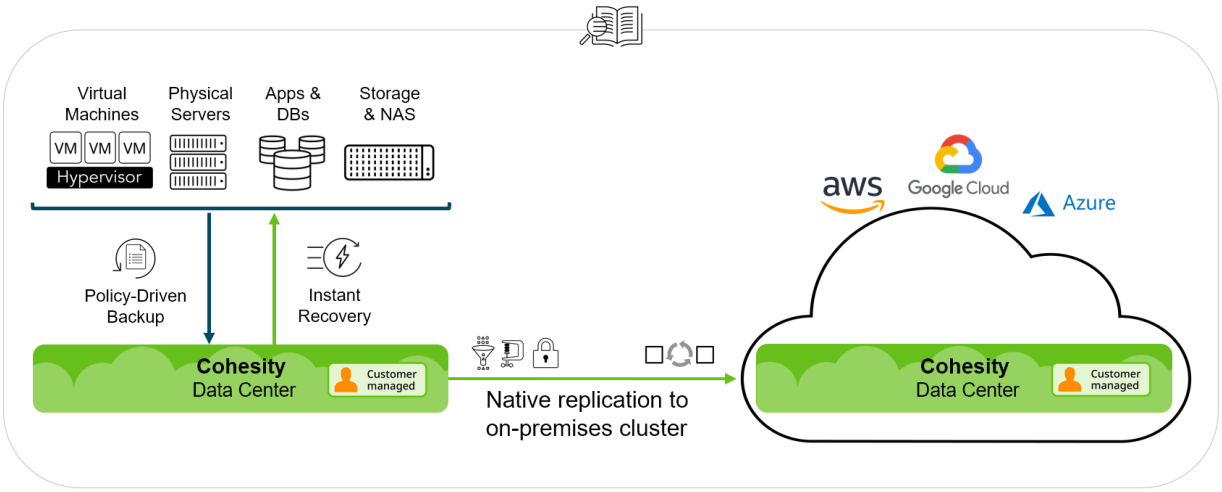


Figure 16: Cohesity Native Replication to Cohesity Cluster in the Cloud



Failures

Table 8 below describes common software or hardware component failures and their Cohesity remedies. During any of these failures, all operations of the cluster, such as backups and recovery, file IO, replication, and archival are unaffected. For most, the self-healer automatically heals the failure and brings the cluster back to a normal state. The scenarios are listed in increasing order of ease of resolution.

Table 8: Types of Failures

FAILURE	REMEDY
Node Failure	Cohesity can sustain up to two simultaneous node failures. Some aspects for a node or a disk failure are: • Node failure results in both file data and metadata failures. • File data is recovered from the erasure coding or replication scheme. Metadata is recovered from one of the replicas.
HDD Failure	Cohesity can sustain *up to three simultaneous HDD failures* at any point in time. HDDs are used to store the Distributed File Data. The number of HDD failures that can be tolerated depends on the RF or EC settings. Chunk files are units of replication. Cohesity ensures that there is always a chunk file replica on other nodes, and internal chunk files representing the same user object (or user file) are spread across multiple disks.
SSD Failure	Cohesity can sustain *up to two simultaneous SSD failures*. SSDs are used to store the distributed metadata and the distributed journal, and are replicated on either 2 or 3 nodes. Parts of the Distributed Metadata and Journal are stored on every node of the Cohesity cluster. An SSD failure can be considered the same as a node failure.
Network Failure	Each node gets its own node IP address, a VIP address, and a hardware management IPMI address. The DNS is set-up to do round-robin, which gives each VIP a turn to satisfy client read or write requests and equally distributes them across all the nodes. **VIP Failover**: Cohesity has a heartbeat mechanism that detects a node failure. If the directly connected network link goes down, the VIP of the failed node gets reassigned to another node, and node rebalancing similar to a node failure is initiated. Cohesity nodes are typically configured with 2x 10GbE connected to two separate switches. This allows for failure of a single switch or a single network port, while still allowing continuous operation.

FAILURE	REMEDY
	For more details, see Optimal Network Designs with Cohesity and Cohesity Cluster Networking Quick Start Guide.
Boot Device Failure	Cohesity boot device is isolated to a separate partition on flash. Since Cohesity file system is already saved within persistent hard drives, there is no impact on a boot device crash. There is no inconsistency or file system repair required.
Power Supply	Cohesity supports having a redundant power supply per chassis so that if one goes down, the other can still power the nodes.
Fans	Cohesity platforms support a fan redundancy of N+1 (there is one extra fan per chassis). Hence, they will support one per node and an extra one to cover for failures.

Other Scenarios

Table 9 below covers several other likely scenarios and how Cohesity responds to each.

Table 9: Other Scenarios

EVENT	RESPONSE
Node Addition	Cluster rebalancing is initiated by the self-healer. Every time another node is added, part of the existing file data and metadata from the remaining nodes is rebalanced to the new node. This is done at a chunk-file level. The larger the cluster size, the faster the rebalancing. Common scenarios for node addition: • Cluster expansion, for example from 4 to 5 nodes Failed node replacement when the cluster size remains the same. For example, one node in a 4-node cluster fails and is replaced.
Node Removal	Node removal involves a removal of a node from the cluster. Cluster rebalancing is initiated. Similar to a node addition, after a node failure or a node removal, the cluster is healed. The file data and metadata are rebalanced on the surviving nodes. If the number of nodes falls to 2, there is an alert and the write policy changes from an EC- to an RF-based one, and write continues. If a replacement node is added, the file data and metadata are rebalanced back. If a node goes down, other replicas ensure that newer disks are selected to store the data chunks that were lost from that node. Common scenarios for node removal from a cluster: • Node removal (graceful) during a hardware refresh • Node failure. For example, one node in a 4-node cluster fails and needs to be replaced.
Node Temporarily Goes Down and Comes Back Up	This occurs when the node is assumed to be dead after a timeout and rebalance has already kicked in. The self-healer will stop healing (redistributing) when the node comes back. If the healing process is interrupted, some chunks will be over-replicated compared to others, but those chunks will be cleaned up later by the garbage-collection process.
Attached Network is Temporarily Down and Comes Back Up	The node that is not reachable will be considered to be down. The self-healing process kicks in and rebalances the surviving nodes. When the network to the node is restored, the extra data and metadata replicas are cleaned up by the garbage collector.

EVENT	RESPONSE
Node Failure and Insufficient Space	If there is a node failure and the remaining nodes do not have sufficient available space to support the file data and metadata from the failed node, the self-healer will not perform rebalancing. In such cases, an alert is raised until a new node is added.
Non-Disruptive and Rolling Upgrades	Cohesity's clustered architecture allows for a sequential node-by-node upgrade. Nodes can be upgraded to the latest versions without any interruption to client IOs. Any active IO access through the VIP is transferred as the VIP is seamlessly transferred to another node.
Fault Tolerance with Cohesity for the Edge	Cohesity for the Edge is the virtual platform running as a VM on either VMware vSphere or Microsoft Hyper-V hypervisors using commodity hardware. An environment such as ESXi or VMware vSAN with shared infrastructure can provide VM/storage migration capabilities on the underlying host failures.

Conclusion

Cohesity ensures data integrity and provides data protection at all levels. The Cohesity's platform has been architected with the customer's business-continuity needs as a central requirement. Protecting data from loss and corruption, and efficiently providing additional levels of redundancy, are essential to Cohesity's design philosophy. Similar to hyperscalers, Cohesity anticipates and accounts for inherent hardware and software failures and ensures a quick recovery to provide the best fault-tolerant solution.

Your Feedback

Was this document helpful? Send us your feedback!

About the Authors

Karthick Radhakrishnan is Director, Technical Solution Engineering. In his role, Karthick focuses on DataProtection, Platform Security and leads the technical solutions team.

Other major contributors include:

- Arvind Jagannath, Product Manager
- Hashim Zargar, Technical Solutions Engineer

Document Version History

VERSION	DATE	DOCUMENT HISTORY
2.5	July 2024	Republishing
2.4	May 2023	Minor updates
2.3	Aug 2022	Minor updates
2.2	Mar 2022	Updated Table 6
2.1	Feb 2022	Minor updates
2.0	Jan 2022	Added Best Practices
1.9	July 2021	Fixed content in Table 1
1.8	May 2021	Cohesity terminology updates
1.7	Jan 2021	Minor updates
1.6	Sep 2020	Minor updates
1.5	Apr 2020	Minor updates
1.4	Mar 2020	Added support for Chassis and Rack Awareness
1.3	Feb 2020	Added support for Three-HDD Failures

VERSION	DATE	DOCUMENT HISTORY
1.2	Jan 2020	Added support for 3D:2N
1.1	Mar 2019	Minor updates
1.0	Dec 2018	First full release

ABOUT COHESITY

Cohesity is a leader in AI-powered data security and management. Aided by an extensive ecosystem of partners, Cohesity makes it easier to protect, manage, and get value from data – across the data center, edge, and cloud. Cohesity helps organizations defend against cybersecurity threats with comprehensive data security and management capabilities, including immutable backup snapshots, AI-based threat detection, monitoring for malicious behavior, and rapid recovery at scale. Cohesity solutions are delivered as a service, self-managed, or provided by a Cohesity-powered partner. Cohesity is headquartered in San Jose, CA, and is trusted by the world's largest enterprises, including six of the Fortune 10 and 44 of the Fortune 100.

Visit our website and blog, follow us on Twitter and LinkedIn and like us on Facebook.