Hammerspace Global File System Deployment Guide

Version 5.3

Table of Contents

About This Documentation

This guide describes how to deploy, operate, and decommission a Hammerspace Global File System (GFS) — the multi-site, active-active capability of the Hammerspace Global Data Platform that provides unified, multi-protocol file and object access (NFS, SMB, S3) across multiple storage systems, sites, and clouds.

It is written for an experienced storage, networking, or systems administrator who is **new to Hammerspace**. Readers are assumed to be comfortable with NFS/SMB, object storage, Active Directory and Kerberos, and multi-site replication concepts, but not yet with Hammerspace terminology, the CLI, or the Management GUI. Hammerspace-specific concepts are explained from first principles.

A working Hammerspace installation at each participating site is required before beginning GFS deployment. This guide does not cover base-product installation; see the Hammerspace Installation and Licensing Guide and the Hammerspace Configuration Guide.

How to read this guide

This guide is written to be read start to finish — Parts 1 and 2 build the concepts that Parts 3 onward assume you already have. If you want to get a two-site deployment running first and fill in the background as you go, a workable shortcut is: read Core concepts (it's short and covers the one idea everything else depends on), skip directly to Deploy and configure a global file system, and come back for Architecture, Handling Conflicts, and Prerequisites when you hit something that assumes them.

Running into an unfamiliar term anywhere in this guide? Check Appendix A: Glossary of terms before assuming it's explained nearby — most Hammerspace-specific terms are defined there even when the surrounding chapter doesn't stop to define them inline.

Do not install additional or third-party software, agents, or packages directly on Hammerspace Anvil (metadata) or DSX (data) nodes. These nodes run a managed appliance software stack; anything installed outside that stack is unsupported and is removed during a software upgrade. If you need monitoring or integration with an external tool, contact Hammerspace Support for a supported approach.

Part 1 — Understand the Concepts

Chapter 1. Introduction and Solutions

A Hammerspace Global File System (GFS) presents a single logical file system that spans multiple locations, regardless of geographic distance. Users and applications at every site see the same files and directories, with the same names, permissions, and metadata, even though the underlying data may physically reside at a different site or in cloud object storage. The goal is to make data a global resource for the enterprise instead of confining files to a single location.

This is a different model from traditional storage, where the only way to give a remote site access to data is to copy files — producing copy sprawl, manual staging and de-staging, and management complexity. Hammerspace replicates the **presentation layer** (metadata) continuously across sites and moves **file data** only when policy or access requires it.

1.1. What a Global File System Is — and Is Not

A Hammerspace GFS is one file system and one namespace shared by two or more Hammerspace clusters, each of which is a **site**. Every site serves its own local users and applications directly, accepts writes, and keeps working on its own if the other sites are unreachable. Sites stay aligned by continuously replicating **metadata** to one another, and move **file data** only when policy or access requires it.

What a GFS is **not** is a real-time collaborative editor. It does not merge simultaneous edits to the same file the way a shared online document does. Each site operates on its own instances of files and the system reconciles changes using an eventually consistent model. Where two sites change the same file at the same time, GFS retains a recoverable copy of the superseded version rather than silently discarding it (see How conflicts are handled for the mechanics).

Two properties define the model:

Active-active

Every site can serve data and accept writes. Global accessibility allows active-active write patterns, but those patterns require planning and discipline (see Protecting Data: Snapshots, Versioning, and Undelete and the unsupported use cases in Prerequisites, Sizing, and Networking).

Eventually consistent

Changes made at one site propagate to the others within a short, bounded interval (5 seconds by default for metadata). Hammerspace deliberately does not use cross-site file locking, because synchronous locking over long distances degrades performance and makes all sites read-only if any one site becomes unavailable.

1.2. Solutions Enabled by GFS

The Global File System supports a range of data-management solutions:

- **Hybrid cloud** — extend on-premises resources to cloud storage with global file access.
- **Burst-to-cloud** — use compute in one or more cloud regions on demand.

- **Multi-site collaboration** — enable follow-the-sun workflows.
- **Workflow automation** — stage data non-disruptively to compute anywhere for processing.
- **Disaster recovery** — file-granular, prioritized recovery on any site.
- **NAS to the edge** — extend NAS to edge locations without dedicated infrastructure.
- **Cloud data migration** — migrate data in the background without interrupting access.
- **Storage consolidation** — consolidate NAS non-disruptively, and centralize edge backups to a core site.

1.3. Key Features

- Multi-protocol data access — active-active across NFS 3, NFS 4.1, NFS 4.2, SMB 2.x/3.x, and S3, with no client software required.
- Objective-based data policies driven by file-system and custom metadata (see Core concepts for what an objective is, and Objectives and data placement for how to build one).
- Transparent cross-site data orchestration with integrated global deduplication and compression (see The shared Object Storage Volume in depth).
- Encryption of the data payload in transit and at rest in cloud/object storage (see The shared Object Storage Volume in depth).
- Global snapshots and global undelete (see Protecting data: snapshots, versioning, and undelete), and file-granular WORM.
- Conflict resolution with cross-site ownership versioning (see How conflicts are handled).
- Per-site data management, so sites cannot interfere with one another's data mobility.
- Heterogeneous, capacity-agnostic storage — any vendor, only enough capacity for active data.
- Tolerant of high-latency links; qualified/supported for up to 16 sites per share (see Requirements and limits at a glance — this is a supported-configuration limit, not a hard product ceiling).
- Support for high-performance AI and HPC use cases via Parallel NFSv4.2 with FlexFiles.

Chapter 2. Core Concepts: The Shared Object Storage Volume

Before looking at node roles, it helps to understand the single primitive that makes a global file system work: the **shared Object Storage Volume** (shared OSV) backed by a **shared bucket**. Everything else — metadata replication, data mobility, conflict handling, decommissioning — builds on it.

2.1. One Namespace, Many Independent Sites

A GFS is one namespace stretched across multiple Hammerspace clusters, where each cluster is a **site**. Each site is designed to keep operating on its own if the others are down or unreachable. Sites stay in sync by continuously exchanging **metadata** — the file system's structure, names, timestamps, permissions, and custom metadata — not by copying every file everywhere.

2.2. Instances, Not Copies

Hammerspace replicates **instances**, not copies. When you copy a file in a traditional system, you create a second, independent file that happens to have the same contents. An **instance** is the same file, which can physically exist in two or more locations at once; to the file system it is still one file. This is possible because metadata is disaggregated from data and kept consistent across all sites.

2.3. The Shared Bucket and the Shared Object Storage Volume

Sites exchange **file data** (as opposed to metadata) through a **shared bucket** — an object-storage bucket reachable by every participating site. When Hammerspace adds that bucket to a cluster as a shared resource, it becomes a **shared Object Storage Volume**, usually shortened to **shared OSV**. The owning site uploads changed data to the shared OSV; a requesting site downloads it from the shared OSV. Metadata never travels through the bucket — only data.

Because data moves through deduplicated, compressed (and optionally encrypted) objects, the bytes transferred between sites are often far smaller than the file itself: only changed chunks move. See The shared Object Storage Volume in depth.

2.4. Objectives: How You Tell Hammerspace What to Do

An **objective** is a rule that tells Hammerspace what to do with files matching some condition — where to place instances, how long to keep them, whether to keep an extra copy nearby, and so on. Objectives are written in plain language (for example, "keep files used in the last 7 days on local storage") and are typically layered: a share commonly has several objectives active at once, each handling a different concern.

Three objectives matter immediately for a global file system, because GFS applies one of them automatically to protect data during cross-site conflicts:

`log-xfer-1-<time>`

Retains a version of a file before it changes owner — central to how GFS handles two sites changing the same file (see How conflicts are handled).

`versioning-<time>`

Retains a version of a file when the file is written again after having been closed for a period — protection against overwriting your own work, independent of any change of ownership. Retained versions appear alongside `log-xfer` versions and use the same filename format (see The shared Object Storage Volume in depth).

`undelete-<time>`

Retains a deleted file for a set period, enabling recovery.

The full mechanics of writing and applying objectives — including how each site manages its own objectives independently of the others — are covered in Objectives and data placement.

2.5. Site Ownership

At any moment a file is **owned** by the site that last wrote to it. Ownership matters when another site needs data it does not hold locally: that site contacts the owning site and orchestrates mobility of the data to itself through the shared OSV. Ownership also governs what happens when two sites change the same file at the same time (How Conflicts Are Handled).

2.6. How a Site Gets a File It Does Not Hold Locally

1. A user at Site B opens a file whose data currently lives only at Site A.
2. Site B already has the file's metadata (it replicated within seconds of creation).
3. Site B requests the data; the owning site (A) has placed — or now places — the data into the shared OSV.
4. Site B pulls the data from the shared OSV and instantiates a local copy according to its own objectives.

Chapter 3. Architecture

With the shared-OSV concept in place, the node roles are straightforward. Each site runs two kinds of Hammerspace node — **Anvil** and **DSX** — with very different responsibilities. Each site is designed to operate independently of the others.

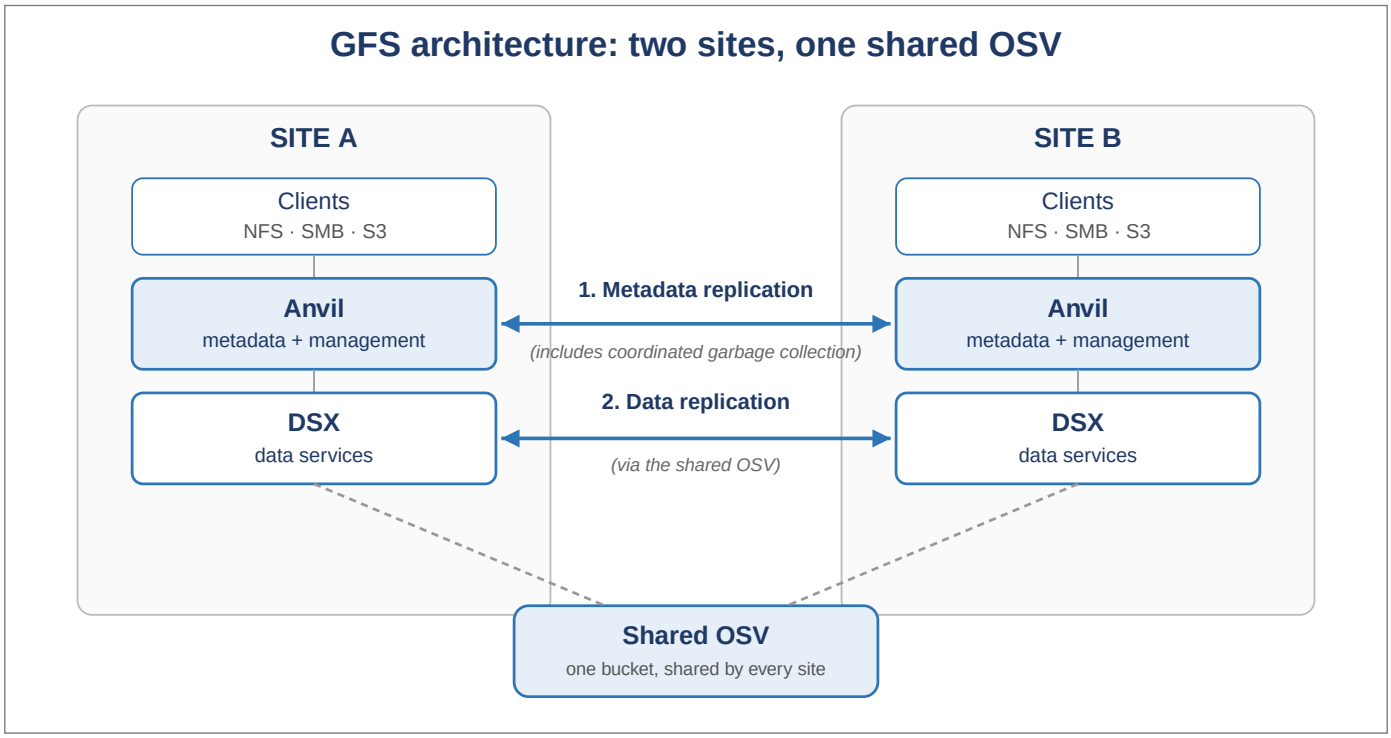

Figure 1. Conceptual GFS architecture: two sites, one shared OSV

In words: each site — whether on-premises or in the cloud — runs its own Anvil and one or more DSX nodes, serving its own local users and applications directly over NFS, SMB, or S3. Anvil nodes handle all cross-site metadata replication, and DSX nodes coordinate directly with their counterparts at other sites about what data needs to replicate — but the file bytes themselves move only through the shared OSV, not directly between DSX nodes. A site that loses its connection to the others keeps serving its own local data uninterrupted.

The second line in the figure — data replication through the shared OSV — is the node-level view of the walkthrough in Core concepts: the shared Object Storage Volume, under "How a site gets a file it does not hold locally." That walkthrough describes what happens; this figure shows which nodes do it.

3.1. Anvil — Metadata and Management

Anvil nodes are responsible for all file-system and extensible metadata and for systems management, including the API and Management GUI. Anvil nodes handle all cross-site **metadata** replication, bi-directionally, as point-to-point connections between Anvil nodes.

- Connections between Anvil nodes are encrypted with TLS (AES-256). Customers may install their own certificate.
- Four ports must be open between all Anvil nodes across all participating sites: **443, 9097, 9298, 9299**.

3.2. DSX — Data Services

DSX nodes transfer data chunks to and from cloud/object storage, use local storage to hold data, and present NFS, SMB, and S3 to site-local clients. DSX nodes coordinate directly with their counterparts at other sites about what data needs to replicate, but the file bytes themselves move only through the shared OSV — not directly between DSX nodes. DSX nodes must have full connectivity to the shared OSV(s).

3.3. Why Sites Can Briefly Differ

GFS keeps every site's copy of the namespace consistent over time, not instantly. Two things follow directly from the Anvil/DSX split above:

- **Metadata and data replicate independently.** A file, or a change to one, can show up in the namespace at a remote site — carried there by Anvil-to-Anvil metadata replication — before its actual bytes have finished moving through the shared OSV. That's expected, not a fault: the namespace and the data behind it travel on separate paths, at separate speeds.
- **Changes propagate asynchronously.** A write made at one site reaches the other sites soon afterward, not instantly and not necessarily in the order it was made. See How conflicts are handled for what this means when two sites change the same file.

This same separation is also why cleanup between sites has to be coordinated: because more than one site can reference the same object in the shared volume, removing it safely means the sites first have to agree nothing still needs it. See The shared Object Storage Volume in depth for how that coordinated garbage collection works.

3.4. Deduplication Chunk Tracking

Hammerspace stores file data in the shared OSV as deduplicated chunks, matched by content rather than by file. When a file changes, only the changed chunks need to move between sites, and chunks already present in the shared OSV are not re-uploaded. This is what keeps cross-site transfer small relative to total file size. See The shared Object Storage Volume in depth for chunk size, how deduplication is applied, and the other OSV features that reduce what has to move.

Chapter 4. How Conflicts Are Handled

When data is used across sites that are far apart, several questions affect the end-user experience:

- How fast will I see new files?
- What happens if I modify the same file on two sites at the same time?
- What if the sites are disconnected for an extended period?

These come down to how ownership and conflict resolution work. The Global File System is eventually consistent and does not support cross-site file locking, so it relies on conflict-resolution logic and on objectives to manage data across sites.

As introduced in Core concepts, an objective is Hammerspace's mechanism for controlling where and how long file data is kept. Two objectives are central to conflict handling:

`log-xfer-1-<time>`

: Retains a version of a file before it changes owner. File ownership is determined by the site that last wrote the file. Ownership matters when another site needs the data but has no local copy — it reaches out to the owning site and orchestrates mobility.

`undelete-<time>`

: Preserves files for a set period after deletion, enabling quick recovery from accidental deletes.

GFS shares automatically apply a `log-xfer` objective when a share becomes replicated, to protect against data loss when ownership changes. (Documented behavior; ties to DOCS-46 / HS-26986.)

4.1. Scenario 1: the Same Filename Created on Multiple Sites at Once

If the same file is created or modified on multiple sites during the same replication interval (or while sites are disconnected), GFS automatically creates two or more versions: a site-local file with the original name, and an additional file representing each remote version.

In this example a file of a different size is created on each of three sites from a Linux client, with the share mounted under `/global`.

Command:

```
cd /global
dd if=/dev/urandom of=file bs=1024k count=1 conv=notrunc   # Site A (ID 0)
dd if=/dev/urandom of=file bs=1024k count=5 conv=notrunc   # Site B (ID 1)
dd if=/dev/urandom of=file bs=1024k count=9 conv=notrunc   # Site C (ID 2)
```

After the replication interval passes, `ls -l` at Site A shows the local file plus the remote versions, each renamed

`FILENAME[#S=SITE ID]`:

Expected output (abbreviated to size, date, and name):

```
total 15360
1048576 Feb 28 00:44 file
5242880 Feb 28 00:44 file[#S=1]
9437184 Feb 28 00:44 file[#S=2]
```

The same filename exists on every site; the conflicting versions from other sites are renamed so the local file never suddenly changes name and remote writes never overwrite local content. To resolve a conflict, copy the renamed version over the original filename and delete the automatically named file if it is no longer needed.

If the file has an extension, the rename preserves it: `report.docx` becomes `report[#S=1].docx`, not `report.docx[#S=1]`. Confirmed live 2026-08-27.

4.2. Scenario 2: Writing to an Existing File on Multiple Sites at Once

Metadata replicates to all sites, but file **data** is replicated only when requested — explicitly through objectives or implicitly on access. By default the **last writer wins** when two sites write the same file within the same replication interval.

To protect against user mistakes, GFS can preserve previous file content through the `log-xfer` objective. Because site-ownership changes create additional copies (similar to undelete and versioning), those copies carry an expiration set by the objective. If snapshots are taken before the files are deleted, the files remain in the snapshots until the longer of the snapshot retention or the file expiration elapses.

4.3. Site-Ownership Objectives

The objective name sets how long retained versions stay in the live tree. When a retained file is copied, the retention does not travel with the copy.

- `log-xfer-1-hour`
- `log-xfer-1-day`
- `log-xfer-1-week`
- `log-xfer-1-month`

Retained files appear in `.snapshot/current`, each made unique by an appended timestamp and the originating site identifier. A full decode of these filenames is in The shared Object Storage Volume in depth.

Part 2 — Plan

Chapter 5. Prerequisites, Sizing, and Networking

A multi-site environment can be brought up quickly, but a few prerequisites must be in place first. Hammerspace can use block storage, existing NAS, and cloud/object storage, and installs as a single software load on bare-metal, virtual, or cloud environments.

Pre-deployment readiness checklist

One row per site, before you start Part 1 of Deploy and Configure a Global File System:

- ☐ Hammerspace installed and licensed at every participating site.
- ☐ At least one Anvil per site (HA pair recommended).
- ☐ At least one DSX per site (two or more recommended).
- ☐ Local file storage configured per site (DSX Store volume or NAS export).
- ☐ Shared object bucket reachable from all Anvil and DSX nodes at every site; credentials with required permissions on hand.
- ☐ Ports 443, 9097, 9298, 9299 open between all Anvil nodes across all sites — verified, not assumed (see the port-check pitfall below).
- ☐ If using SMB: all sites joined to the same AD domain/forest.
- ☐ Site names chosen, if using descriptive names (Part 3).
- ☐ Conflict-resolution retention decided (`log-xfer-1-<time>` period) — see How Conflicts Are Handled.

5.1. Requirements and Limits at a Glance

Requirement / limit	Detail	Source
Anvil per site	At least one (HA pair recommended)	This chapter
DSX per site	At least one (two or more recommended)	This chapter
Shared bucket(s)	At least one (two or more recommended for resilience)	This chapter
Inter-Anvil ports	443, 9097, 9298, 9299	This chapter
ID-mapping source	Same across all sites (Active Directory)	This chapter
Default replication interval	5 seconds	Deploy and Configure a Global File System
Default replication alert threshold	300 seconds	Monitoring and Alerting
Latency envelope	Proven to 200 ms continuous ping latency	This chapter

Requirement / limit	Detail	Source
Shared-OSV capacity guideline	Roughly 14 days of transfer volume	This chapter
Sites per share	Qualified/supported up to 16 — see the note below	Introduction and Solutions

"Sites per share" is phrased here as a **qualified, supported configuration limit**, not a hard product ceiling — confirm the current supported number against the Hardware and Software Compatibility List before treating it as fixed, since it's a support-matrix decision rather than something enforced by the product itself.

5.2. Minimum Requirements

- Each site has at least one installed Anvil metadata node (HA recommended).
- Each site has at least one installed DSX data services node (two or more recommended).
- One shared cloud/object bucket for cross-site data transfer (two or more recommended for resilience).
- Network connectivity between Anvil nodes on four ports: 443, 9097, 9298, 9299.
- The same ID-mapping source across all sites. This is currently supported using Active Directory.

5.3. Sizing

- **Anvil** — confirm the Anvil's block storage has sufficient IOPS and low latency for metadata operations; sub-optimal storage degrades replication. Provide adequate CPU and memory for metadata churn and checkpointing, especially with many sites. Work with your Hammerspace Sales Engineer to size correctly.
- **Latency tolerance** — GFS is architected to tolerate high latency; higher-latency sites simply lag slightly. Hammerspace is proven to operate with sites at 200 ms continuous ping latency.
- **Capacity planning** — size each site's NAS for its active working set and tier the rest to a shared OSV. Provision enough shared-OSV capacity for cross-site transfer; a working guideline is roughly **14 days** of transfer. Site-to-site transfers stall if the shared OSV runs out of capacity.
- **Bucket redundancy** — configure a second shared bucket with a higher online-delay objective so all transfers prefer the first bucket and fall back to the second only if the first is unavailable.

5.4. Network Connectivity

- Reliable, high-bandwidth connectivity between all sites — enough to transfer new and estimated changed data to or from any other site.
- The four Anvil ports above, plus firewall rules permitting Anvil-to-Anvil and DSX-to-shared-bucket traffic.

A recurring deployment failure is `REPLICATION_PORT_CHECK_FAILED` on ports 9298, 9299, 9097, or 443, which blocks participant-add or flips participants to Suspected/Down. Verify all four ports end-to-end before adding participants. See Troubleshooting: Deployment.

Coordinate upgrades across every site

Per the Installation Guide's Upgrading global file system environments, upgrading an environment with replicated shares stops metadata replication until **every** participating site has been upgraded — plan multi-site upgrades to happen close together, ideally simultaneously, to minimize the delay.

5.5. Protocol Prerequisites

Global NFS shares

There are no special requirements for NFS to work globally, except when Kerberos is used.

Global SMB shares

For Windows/SMB access, all sites must be joined to the same domain / domain forest so Security Identifiers (SIDs) translate correctly on every site.

Global S3 servers/buckets

Each site presents its own S3 endpoint. See the flag below before relying on this for a cross-site deployment.

Chapter 6. The Shared Object Storage Volume in Depth

The shared Object Storage Volume (shared OSV) is where cross-site file data lives in transit and at rest. This chapter explains how data is stored in the shared bucket and how to read the retained-version filenames in the snapshot directory.

6.1. How Data Is Stored in the Shared Bucket

When the owning site needs to make data available to other sites, its DSX nodes write the file's data into the shared OSV as objects. Several OSV features reduce how much data must move and how much capacity it consumes:

File chunking

4 MB chunk size, on by default. It can be disabled by adding the OSV in native mode, but disabling is uncommon.

Deduplication

Hammerspace deduplicates by matching SHA-256 sums of chunks. Because only changed chunks move, the bytes transferred between sites are often far smaller than the file. Deduplication cannot be disabled (there is no benefit to disabling it) and typically saves 10–20%, sometimes up to 50%, depending on the data.

Compression

Optional per OSV. It can save capacity and bandwidth but adds CPU load on the DSX nodes performing it; size DSX accordingly.

Encryption

If a Key Management Service (KMS) is configured, new objects placed in the OSV are encrypted unless the OSV is added with the no-encryption option.

These four features must be configured **identically** on the shared OSV at every participating site. In particular, if the KMS is not the same across all clusters, clusters with a different or absent KMS cannot decrypt objects written by sites with encryption enabled, and data orchestration fails.

6.2. Garbage Collection

Coordinated Garbage Collection (CGC) removes objects from buckets that no longer have references from any site, keeping bucket utilization from growing unchecked.

6.3. Reading the Snapshot Directory: `ls -l` Decode

`.snapshot/current` (below) holds retained **versions** created automatically by the `log-xfer`

> and `undelete` objectives — not the same thing as a formal point-in-time **Snapshot**, which is a separate mechanism covered later in Protecting data: snapshots, versioning, and undelete. The two live in similarly-named locations (`.snapshot` here vs. `.fsnapshot` for real Snapshots) but work independently; don't confuse one for the other.

Retained versions created by `log-xfer`, `versioning` and `undelete` all appear in `.snapshot/current` and all use the same filename format. What differs is what triggered the copy, not how the filename looks: `log-xfer` on a change of site ownership, `versioning` when a file is written again after having been closed for a period, `undelete` on deletion. Each filename is made unique by an appended timestamp and the originating site identifier:

```
# ls -lt .snapshot/current/
-rw-r--r-- 1 root root 217 Feb 28 07:33 file
-rw-r--r-- 1 root root 174 Feb 28 07:30 file[#M=20200916.212518.75035 #S=1-5]
```

The first entry is the live-tree version. The second is a saved version: `#M=` is the timestamp, and `#S=1-5` indicates Participant (site) ID 1 with object ID 5 in the file system. The site ID can also be found from a mounted share with the Hammerspace Toolkit (`hs eval -e participants <PATH>`) or via the admin CLI (`share-list --name <SHARE NAME>`).

Part 3 — Deploy

Chapter 7. Deploy and Configure a Global File System

This chapter is the end-to-end worked example. Once you understand the shared-OSV concept (Core Concepts: The Shared Object Storage Volume) and the architecture (Architecture), deployment is a matter of following these steps in order. The example builds a Global File System across three sites that use different vendors and capacities:

- **Site A** — Anvil and DSX virtualized in VMware ESX with local NVMe and EMC Isilon.
- **Site B** — Anvil and DSX virtualized in VMware ESX with local SSD and NetApp 7-mode.
- **Site C** — Anvil and DSX in AWS, DSX using local EBS volumes.

7.1. Part 1: Install Anvil and DSX

Install Anvil and DSX at each site; this can run concurrently across sites and takes roughly 20–60 minutes per site. See Installing the metadata server (Anvil) and Installing data services (DSX) for details.

7.2. Part 2: Configure Local Storage

Each site needs local file storage — a DSX with local block storage or a local NAS export — to hold data used at that site. See Adding storage to Hammerspace for details.

7.3. Part 3: Set a User-Friendly Name for Each Site (Optional)

Each site has a cluster name set at install time. Setting a descriptive name makes multi-site management easier and lets you use the name in place of an IP address during replication setup.

Command:

```
local-site-config --name "SITE A"
```

Expected output:

```
ID:                           179ca903-af1e-4e37-9086-505fa83fdcf7
Name:                         SITE A
Internal ID:                  0
Site management address:      192.0.2.10
Site data address:            192.0.2.10
Effective management address: 192.0.2.10
Effective data address:       192.0.2.10
Trust client certificate:     true
Client certificate:
```

```
                        Version:      3
                        Serial:       <serial number>
                        Algorithm:    SHA512withRSA
                        Issuer:       O=Hammerspace, OU=SITE, CN=<site ID>
                        Subject:      O=Hammerspace, OU=SITE, CN=<site ID>
                        Validity:
                          Not Before: <timestamp>
                          Not After:  <timestamp>
                        Fingerprints:
                          SHA1:       <fingerprint>
                          MD5:        <fingerprint>
                          CRC32:      <fingerprint>

GFS participants:
                        [Share participant ID: <id>, ID: <uuid>]
```

The `GFS participants` list has one entry for each share on this site — every share carries its own local participant, whether or not it replicates anywhere — so it is empty only on a site that has no shares yet. It does not show which shares are part of a Global File System; use `gfs-participant-list` for that.

Repeat on Site B and Site C.

7.4. Part 4: Configure a Shared Bucket (Required)

Stop here and complete a full chapter before continuing

This part is a summary, not the procedure. The actual steps — adding a storage system, selecting or creating a bucket, and marking it shared — are a complete chapter of their own: Adding a shared bucket. Go there now, complete it for every site, and then come back to Part 5 below. Part 5 assumes the shared bucket is already configured on every site.

Any supported object-storage vendor/cloud can be the shared bucket; if none is available, Hammerspace S3 functionality can serve as the shared bucket. The bucket is used for cross-site data transfer (never metadata), must be reachable by all Anvil and DSX nodes at every site, and is added with the shared flag so other sites can join it.

7.5. Part 5: Create a Global Share

Create a new share (or use an existing one) and turn on share replication, which bi-directionally replicates all metadata changes to every participating site. The initial setup is done on the site holding the source share. The share must **not** already exist on a target site; GFS creates it there and immediately begins a near continuous stream of metadata.

7.5.1. Using the CLI

Run on Site A to create the share and add Site B and Site C as participants.

Command:

```
share-create --name Home --path /home
```

Expected output:

```
Name:                                Home
Internal ID:                         <id>
ID:                                  <uuid>
Path:                                /home
Lifecycle:                           CREATED
State:                               PUBLISHED
Size limit state:                    NORMAL
Export options:
                         [Client specification: *, Access permissions: RW, Root-squash: false, Insecure:
false, Security options: [SYS]]
SMB-browsable:                       On
Is referral:                         No
Objectives:
                         [Name: delegate-on-open, ...]
                         (the share's default objective set — see xref:objectives-and-data-
placement.adoc[])
Logical ID:                          <uuid>
Replication latency alert threshold: 300 seconds
Local GFS participant ID:            0
GFS participants:
                         Name:                               Home::SITE A
                         ID:                                 <uuid>
                         Admin state:                        Up
                         Oper state:                         Up
                         Participant share internal ID:      <id>
                         Participant site name:              SITE A
                         Participant site management address: <ip>
                         Participant site data address:      <ip>
                         Participant ID:                     0
                         Send Latency:                       Not Available
                         Receive Latency:                    Not Available
Client certificate:
                         (certificate detail, same shape as in local-site-config)
Private key CRC32:                   <checksum>
```

`Send Latency` and `Receive Latency` show `Not Available` here because this share has only its local participant so far — there is no remote site yet for latency to be measured against. Once a remote participant is added (see `gfs-participant-add`, next), these report real values the same way they do in the "Verifying and managing participants" section later in this chapter.

Command:

```
gfs-participant-add --share-name Home --site-name "SITE B" --site-username admin --site-password ********
```

Expected output:

```
Name:                               Home::SITE B
ID:                                 8365f58a-24a1-4ed1-9ae2-f9f4d4f75501
Local share name:                   Home
Admin state:                        Up
Oper state:                         Up
Participant share internal ID:      6
Participant site name:              SITE B
Participant site management address: 198.51.100.10
Participant site data address:      198.51.100.10
Participant ID:                     1
Replication interval:               5 seconds
Send Latency:                       0 seconds
Receive Latency:                    Not Available
Shared object storage volumes:
                        [Name: ObjectStorage::sharedbucket, Internal ID: 268435473, ID: 08d4bcee-577c-
408a-9d27-637d17865e1b]
Internal ID:                        7
Local share internal ID:            6
```

If Part 3's site naming was skipped for a given site, use that site's actual cluster name (its auto-generated ID, e.g. `AE15J7QCN01WL4`) in place of `"SITE B"` — naming sites is optional, not required, for `gfs-participant-add` to work. Using an unrecognized site name fails clearly:

```
gfs-participant-add: site 'SITE B' not found.
```

Repeat `gfs-participant-add` for Site C.

7.5.2. Using the GUI

1. Navigate to **Data › Shares** and click **Create Share**. Name the share (**Home** in this example) — the **NFS Export Path** field auto-fills from the name (confirmed: typing `Home` immediately populates `/Home`). Click **Create**. The share is created on the local (source) site only.

> The NFS Export Path auto-fill only works until the field is manually edited. If you type directly into **NFS Export Path**, the field does not cleanly revert to the derived value — it collapses to just `/`. Retyping the **Name** field afterward does **not** re-trigger auto-derivation for that dialog session. If this happens, either type the full correct path manually (e.g. `/Home`) or close and reopen the dialog.

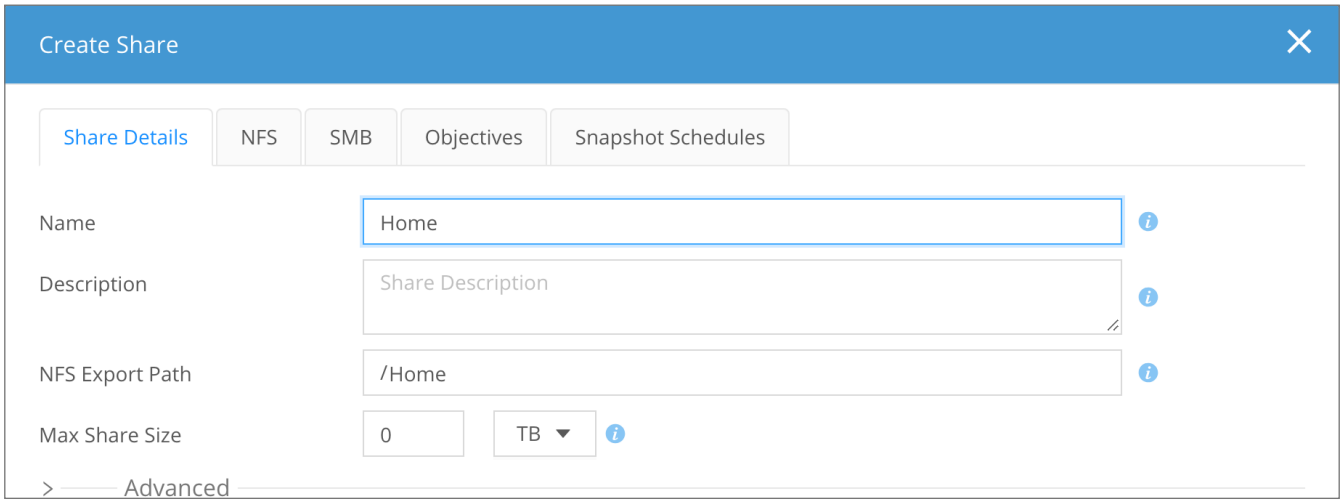

Figure 2. The Create Share dialog with the NFS Export Path auto-filled from the share name

2. Back on **Data › Shares**, click the **Edit** (pen) icon in the **Home** row to open the share-configuration dialog, then the **Objectives** tab. The File Conflict Resolution objective (`log-xfer-1-<time>`) is already selected by default — there is nothing to manually enable. It shows as inactive in the **Active** column only because its applicability condition (`IS_GFS_SHARE`) isn't true yet; it activates automatically once the share has GFS participants.
3. Click the **Global File System** tab. It lists the share's current participants — initially just the local site, shown **Up**. Click **Add Participant**.

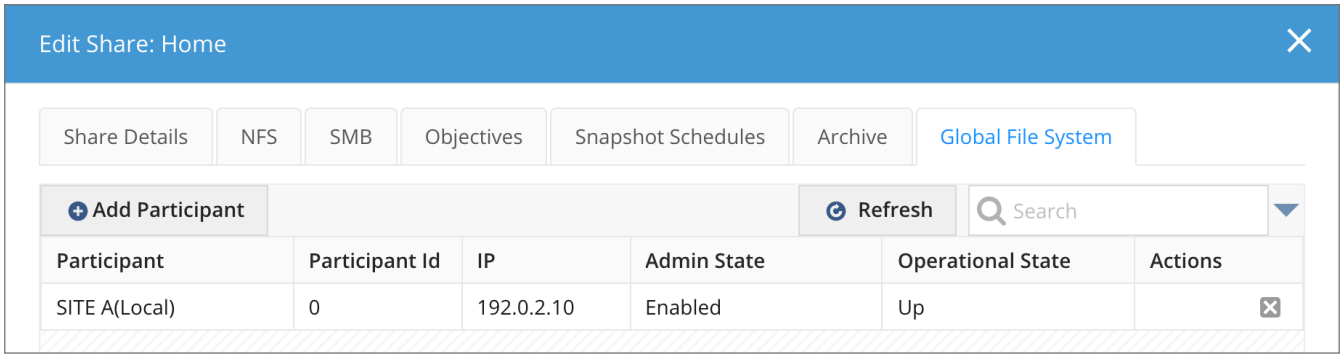

Figure 3. The Global File System tab before any remote participants are added

4. In **Name**, select the remote site (**Site B**). The pull-down is pre-populated with every site registered to the shared bucket: there is no separate "pair the clusters" step, because adding the shared OSV to both sites already registered them with each other. Selecting the site auto-fills its **IP Address**.

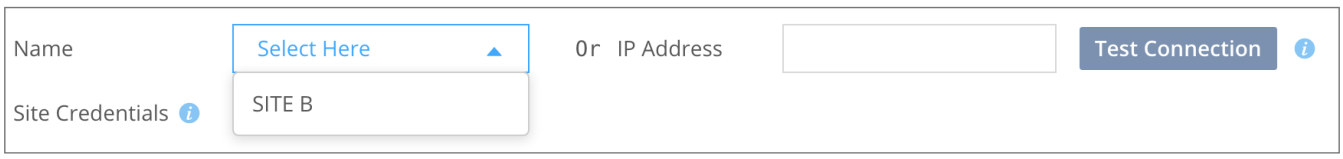

Figure 4. The Name pull-down pre-populated with the registered remote site

5. Under **Site Credentials**, enter the **Username** (`admin`) and **Site B's admin password**, then click **Test Connection** to confirm reachability — a "The site is reachable" message confirms this. Leave **Interval** at the default (5 seconds) unless you need a different replication poll interval. Click **Add**.

What the replication interval controls

The **Interval** is how often each Anvil polls its participants for metadata changes to replicate — it does not set an upper bound on how current the two sites can be, only how frequently they check in with each other. A shorter interval means changes propagate faster but generates more inter-Anvil traffic; a longer interval reduces that traffic at the cost of more lag between a write and its replication. 5 seconds is the default and is appropriate for most deployments; there is no currently documented minimum or maximum — treat very short intervals (well under 5 seconds) as unverified for this guide.

Figure 5. Site credentials filled in and connection confirmed

6. GFS creates the share on the remote site and starts metadata replication — confirmed live at roughly 13 seconds, within the 15–30 second range. The new participant appears on the **Global File System** tab with both **Admin State** and **Operational State** showing **Up**.

On this tab, **Admin State** displays as the word `Enabled`, not `Up`; only **Operational State** literally reads `Up`. Both indicate the same healthy state — see the terminology note in the next step.

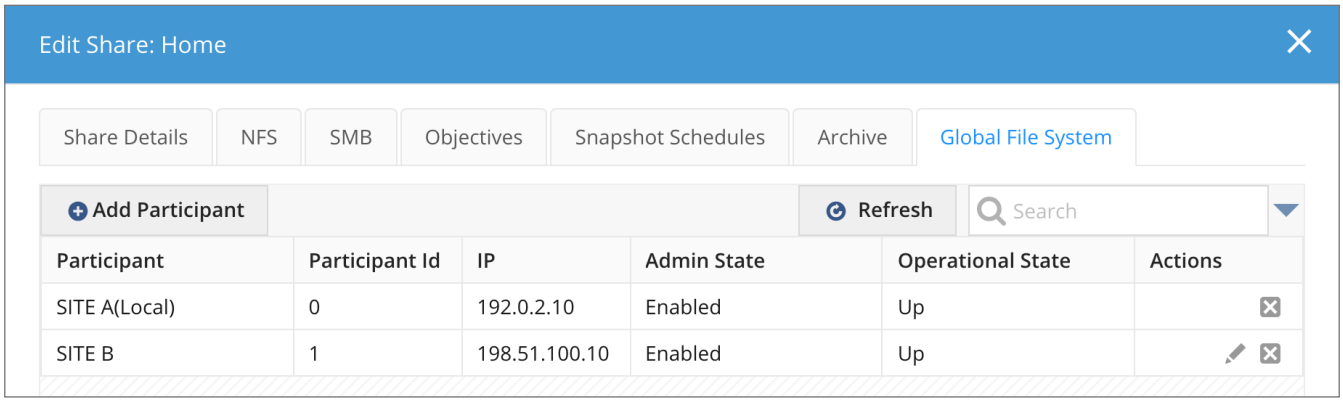

Figure 6. Both participants confirmed Up after adding Site B

Repeat for **Site C**.

7. Verify replication on the **Shares** tab via the **Details** icon (the eye icon — not the **Edit** pen icon) and the **Global File System** tab, which shows each participant's state along with Send and Receive latency.

This **Details** view's **Global File System** tab is a different screen from the **Edit** dialog's **Global File System** tab used in the previous steps, and uses different terminology for the same value: **Admin State** here reads `UP`, whereas the Edit dialog's **Admin State** column

read `Enabled`. Both refer to the same underlying state.

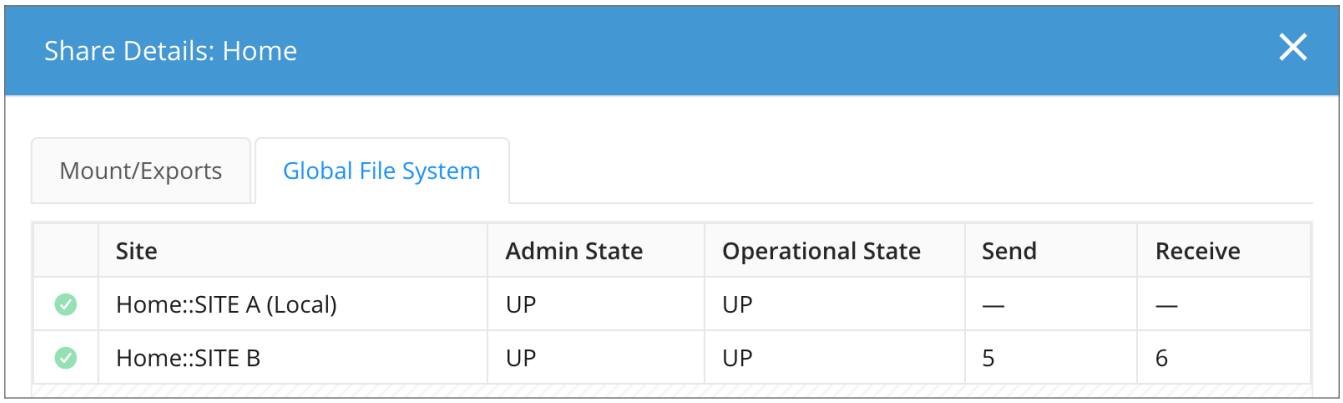

Figure 7. Verifying participant state and latency via the Details view

It is normal for latency figures to read 1.5x–3x higher than the raw network round-trip time (ping RTT) between the two sites — the replication-latency metric measures more than just network transit, so it will not match a plain `ping` result. A very large value on an existing share usually means the initial metadata sync is still in progress; shares are usable while replication catches up.

On an idle share with no data written yet, **Send** still populates with a numeric value — this appears to be a lightweight connectivity check between participants rather than a data-transfer measurement — while **Receive** continues to show – until data has actually been replicated. This differs from 5.2, where both Send and Receive show – until the share has real traffic. Confirmed live: a freshly-created, empty two-site share showed `Send: 4`, `Receive: –` for the remote participant immediately after joining, with no files written to the share.

7.6. Verifying and Managing Participants

The GUI's **Details** view (previous step) is one way to check participant health. The CLI equivalent — and the more complete one — is `gfs-participant-list`, which is the canonical way to check a share's replication health from a script or terminal:

Command:

```
gfs-participant-list --share-name Home
```

Expected output (one block per participant, including the local site):

```
total 2
Name:                              Home::SITE A
ID:                                cf4899fd-088a-4409-b433-31957136ca71
Local share name:                  Home
Admin state:                       Up
Oper state:                        Up
Participant share internal ID:     6
```

```
Participant site name:               SITE A
Participant site management address: 192.0.2.10
Participant site data address:       192.0.2.10
Participant ID:                      0
Replication interval:                0 milliseconds
Send Latency:                        Not Available
Receive Latency:                     Not Available

Name:                                Home::SITE B
ID:                                  8365f58a-24a1-4ed1-9ae2-f9f4d4f75501
Local share name:                    Home
Admin state:                         Up
Oper state:                          Up
Participant share internal ID:       6
Participant site name:               SITE B
Participant site management address: 198.51.100.10
Participant site data address:       198.51.100.10
Participant ID:                      1
Replication interval:                5 seconds
Send Latency:                        4 seconds
Receive Latency:                     6 seconds
Shared object storage volumes:
                         [Name: ObjectStorage::sharedbucket, Internal ID: 268435473, ID: 08d4bcee-577c-
408a-9d27-637d17865e1b]
```

A healthy participant shows **Admin state** and **Oper state** both `Up`. The local site's own entry always shows a `Replication interval` of `0 milliseconds` — it isn't polling itself.

To temporarily pause replication to one participant without removing it — useful for planned maintenance on the remote site — use `gfs-participant-disable`, then `gfs-participant-enable` to resume:

```
gfs-participant-disable --share-name Home --site-name "SITE B"
```

Expected output:

```
Success with some errors:

Name:                                Home::SITE B
ID:                                  8365f58a-24a1-4ed1-9ae2-f9f4d4f75501
Local share name:                    Home
Admin state:                         Down
Oper state:                          Down
Participant share internal ID:       6
Participant site name:               SITE B
Participant site management address: 198.51.100.10
Participant site data address:       198.51.100.10
Participant ID:                      1
Replication interval:                5 seconds
```

```
Send Latency:                          2 seconds
Receive Latency:                       10 seconds
Shared object storage volumes:
                         [Name: ObjectStorage::sharedbucket, Internal ID: 268435473, ID: 08d4bcee-577c-
408a-9d27-637d17865e1b]
Internal ID:                           7
Local share internal ID:               6
```

> Both `gfs-participant-disable` and `gfs-participant-enable` print a `Success with some errors:` header before their normal output, even on a completely ordinary, expected-to-succeed run against a healthy participant. Confirmed live against both 5.3.1-1201 and 5.2.14-1251: the command fully succeeds, `Admin state` changes correctly, and no actual problem occurs either time — this wording is pre-existing CLI behavior on both versions, not a new or 5.3-specific issue.
>
> This header is CLI boilerplate, not a real error indicator — confirmed in the CLI source (`ShareParticipantAdminStateUpdateCommand`, the shared base class behind `gfs-participant-enable`, `gfs-participant-disable`, `share-participant-enable`, and `share-participant-disable`): the command builds its response as `"Success"`, then unconditionally appends `" with some errors:"` followed by whatever content the server's response includes — without checking whether that content actually describes a problem. If `Admin state/Oper state` in the output that follows show the expected result, this header can be safely ignored.

This sets **Admin state** to `Down` for that participant while leaving the participant relationship itself intact — confirmed live: replication lag (Send/Receive latency) grows while disabled, since polling has stopped, and the participant remains listed in `gfs-participant-list` throughout. **Oper state** also reads `Down` for as long as the participant is disabled. Re-enable with:

```
gfs-participant-enable --share-name Home --site-name "SITE B"
```

Expected output:

```
Success with some errors:

Name:                                  Home::SITE B
ID:                                    8365f58a-24a1-4ed1-9ae2-f9f4d4f75501
Local share name:                      Home
Admin state:                           Up
Oper state:                            Up
Participant share internal ID:         6
Participant site name:                 SITE B
Participant site management address: 198.51.100.10
Participant site data address:         198.51.100.10
Participant ID:                        1
Replication interval:                  5 seconds
Send Latency:                          5 seconds
```

```
Receive Latency:                    8 seconds
Shared object storage volumes:
                        [Name: ObjectStorage::sharedbucket, Internal ID: 268435473, ID: 08d4bcee-577c-
408a-9d27-637d17865e1b]
Internal ID:                        7
Local share internal ID:            6
```

Admin state returns to `Up` and replication resumes; latency catches up over subsequent polling intervals rather than instantly. Both commands are reversible and non-destructive — unlike `gfs-participant-remove`, they do not affect data or unwind the replication relationship.

Chapter 8. Adding a Shared Bucket

The shared bucket is the data path between sites. Add it from **every** participating site, with identical OSV settings (see The Shared Object Storage Volume in Depth). The first site to add the bucket writes a registration key into it, which simplifies setup for the remaining sites.

Only one bucket is required, though multiple are supported and recommended for resilience. Access to the bucket must be allowed for all Anvil and DSX nodes from every site; as a security best practice, limit access to those nodes only.

8.1. Adding a Shared Bucket Using the GUI

Before you begin (GUI)

- **Accept the self-signed certificate.** The Anvil GUI is served over HTTPS with a self-signed certificate. The first time you browse to `https://<anvil-ip>:8443` the browser shows a "Your connection is not private" warning; accept it and proceed to the site before logging in.
- **The object storage system must already exist.** This procedure adds a **volume** (bucket) to an object/cloud storage system that has already been added. If it has not, add it first: **Infrastructure › Storage Systems › Add Storage System**, choose **Object Storage**, select the vendor **Type**, and supply the endpoint, access key, and secret key. The **Add Volume** wizard cannot add the storage system itself.
- **The bucket must already exist on the object store.** The **Add Volume** wizard only **selects** existing buckets — it cannot create one. Create the bucket on the object store first (from the vendor's console or CLI), using the **same bucket name at every site**.

1. In the Anvil GUI, go to **Infrastructure › Storage Systems**.
2. If you just created the bucket on the object store, click the **Rescan** action (the circular-arrow icon in the **Actions** column) for the object storage system and wait for its volume count to update. Hammerspace discovers buckets on a cached, periodic scan, so a newly created bucket does not appear until the storage system is re-scanned (see A newly created bucket (or other object) doesn't appear in the GUI).
3. Click **+ Volume** for the object storage system to open the **Add Volume** wizard.
4. On the **Selections** step, search for and select your bucket, then click **Next Step**.
5. On the **Data** step, confirm the **Shared Volume** checkbox is selected — it is checked by default, so leave it checked. On the second and any additional sites the GUI also shows a red warning icon indicating the bucket is already in use by another Anvil; this is expected in a multi-site configuration. Click **Next Step**.
6. Step through **Capacity**, **Capability**, and **IP**; the defaults are appropriate for a shared OSV. Click **Next Step** on each.
7. On **Review & Add**, confirm that **Shared** shows **Yes**, then click **Add Volume**. The volume is added with `shared: true` and comes up with `operState: UP`.

A newly created bucket (or other object) doesn't appear in the GUI

Hammerspace scans an object storage system for its buckets on a cached, periodic basis; it

does not re-read the object store every time you open the **Add Volume** wizard. If you create a bucket and it is missing from the wizard's list, close the wizard, click the **Rescan** action for that storage system on **Infrastructure › Storage Systems**, wait for the volume count to increase, then reopen the wizard. More generally, whenever you expect a newly added object to appear in the GUI and it does not, re-scan (or refresh) the underlying resource before assuming the operation failed.

8.2. Adding a Shared Bucket Using the CLI

Add the `--shared` option to `object-volume-add` to mark the bucket as a shared resource. If `--shared` is not used when the bucket is first added, other sites cannot add it. The flag can only be set when the object volume is added. The same Storage System name (`ObjectStorage`) and bucket name (`sharedbucket`) are used on all three sites here for simplicity; this is not required.

Before you begin (CLI)

The CLI has the same prerequisites as the GUI procedure above:

- **The object storage system must already exist at this site.** `object-volume-add` adds a bucket to a storage system that has already been registered. If it has not, register it first with `object-storage-add`, for example `object-storage-add --name ObjectStorage --type GENERIC_S3 --endpoint http://<object-store-host>:9000 --access-id <access key> --secret <secret key>`. Every site that will share the bucket must register the same object store.
- **The bucket must already exist on the object store.** `object-volume-add` only selects an existing bucket; it cannot create one. Create the bucket from the vendor's console or CLI first, using the same bucket name at every site.
- **Refresh the storage system after creating the bucket.** Hammerspace discovers buckets on a cached, periodic scan, so a bucket created a moment ago is not yet visible to `object-volume-add`. Run `node-refresh --name ObjectStorage` (the CLI equivalent of the GUI's **Rescan** action) before adding the volume.

Command:

```
object-volume-add --shared --node-name ObjectStorage --logical-volume-name sharedbucket
```

Expected output:

```
ID:                     <uuid>
Name:                   ObjectStorage::sharedbucket
Internal ID:            268435473
State:                  OK
Node:                   ObjectStorage
Oper state:             Up
Admin state:            Up
Capabilities:           High threshold:          75%
```

```
                        Low threshold:            60%
                        Availability:             99.99%
                        Availability (effective): 99.99%
                        Availability drop:        Disabled
                        Durability:               99.9999999%
                        Durability (effective):   99.9999999%
                        Online delay:             5 minutes
Total capacity:         Unlimited
Physical free:          Unlimited
Naming type:            Hash named
Compression type:       No compression
Chunking type:          Fixed-size
Storage class:          Standard
Encrypt using KMS:      None. No Kms Configured.
Created:                <timestamp>
Modified:               <timestamp>
Locations:
                        [Type: Node, ID: <uuid>, Name: ObjectStorage]
                        [Type: Object Storage Volume, ID: <uuid>, Name: ObjectStorage::sharedbucket]
                        [Type: Volume Group, ID: <uuid>, Name: all]
                        [Type: Volume Group, ID: <uuid>, Name: object-volumes]
Logical volume:         Type:                     Object Store Logical Volume
                        ID:                       <uuid>
                        Name:                     sharedbucket
                        Ownership:                Shared
                        Client certificate:
                                                  (certificate detail, same shape as in local-site-
config)
GC enabled:             Yes
```

After Site B (and Site C) also add the bucket, this site's `Locations` list grows a `shared-object-volumes` group entry, and every site's output gains a `Shared with:` block listing the other participating sites.

Run the same command on Site B and Site C.

5.3 adds a `Storage class` field (seen value: `Standard`) that doesn't exist in 5.2 — confirmed by directly comparing this command's output on a live 5.3.1-1201 site against a live query of the 5.2 pair's existing shared bucket the same day. It's a real, configurable field, not purely informational — confirmed in the mgmt/CLI source: it's backed by a `StorageClass` enum with two values today, `STANDARD` (the default; valid for any backend) and `AWS_GLACIER_INSTANT_RETRIEVAL` (AWS S3 only; an archive-tier class that's excluded from object-volume and shared-object-volume groups, and requires a matching feature flag on the bucket's identity file). Set it explicitly at add time with `--storage-class` on `object-volume-add`; omit it to get `Standard`.

Re-running this command on an already-configured bucket

If the object volume has already been added at this site — for example, if you're unsure whether a previous attempt succeeded — running `object-volume-add --shared` again with the same `--node-name` and `--logical-volume-name` does not silently succeed or duplicate the

volume. It returns a clear error instead:

```
object-volume-add: The Object storage volume sharedbucket is already in use by this
site.
```

Advanced OSV features differ by interface: only **compression** is available in the GUI; **encryption** and **chunking** (native mode) must be configured through the CLI or API. Configure these features identically on the same bucket at every site.

Part 4 — Place Data

Chapter 9. Objectives and Data Placement

Each site in a GFS is designed to operate even when disconnected, so **objectives are managed locally per site**. Every site applies its own objectives to the same share. This prevents sites from interfering with one another's data mobility and gives each local administrator full control over placement. The guidance below is prescriptive: it states how to do it, not whether to.

Objectives dictate what data moves, when, and where it is stored. Two roles matter for replication:

Core replication objective

A **Place On** objective targeting the shared OSV. This uploads file data to the shared bucket, which is what enables other sites to pull it.

Local instance / performance objectives

Keep Online to hold a copy on the fastest local tier where data is actively used; **Optimize for Capacity** to avoid keeping local instances when no objective requires them (use with care — it interacts with durability).

9.1. Scenario 1: Two Sites, Second Site as DR

Site 1 is on-premises with large NAS that can hold nearly all data. Its objectives keep the NAS as full as possible with the most recent data, maximizing the existing investment.

Site 2 is in the cloud and cost-optimized: most data sits in low-cost cloud object storage, with only active files on expensive high-performance cloud storage. Site 2's objectives are nearly the inverse of Site 1's.

9.1.1. Example: Keep Recent Data Local on All Sites

To keep recently used data online everywhere while leaving the rest in cloud storage, create the objective on all three sites and apply it to the share or directory on each:

Command:

```
objective-create --name "Recently used data is local" --applied-objective keep-online,LAST_USE_AGE<7DAYS
```

Expected output:

```
ID:                     3dd45ce7-16fa-439f-a678-6304686dc262
Name:                   Recently used data is local
Internal ID:            5
Applied objectives:
                        [ID: 2, Objective ID: 12851c6d-0837-4e12-bd1e-c9faeff29e9e, Objective name:
keep-online, Applicability: LAST_USE_AGE<7*DAYS]
```

To verify or re-check the objective later, `objective-list --name "<name>"` returns the same fields:

```
objective-list --name "Recently used data is local"
```

Expected output:

```
ID:                     3dd45ce7-16fa-439f-a678-6304686dc262
Name:                   Recently used data is local
Internal ID:            5
Applied objectives:
                        [ID: 2, Objective ID: 12851c6d-0837-4e12-bd1e-c9faeff29e9e, Objective name:
keep-online, Applicability: LAST_USE_AGE<7*DAYS]
```

Chapter 10. Objectives Cookbook

Copy-ready objective sets for common GFS patterns, abstracted from real deployments. Names and node identifiers are examples; adapt them to your storage systems. Always create an objective on every site that needs it and apply it to the share or directory on each.

By default, if a cluster has available object storage and the default `optimize-for-capacity` objective is present, files down-tier to object storage after about 5 minutes when nothing compels them to stay online. Define keep-online objectives that hold files online for as long as the workflow needs.

10.1. Recipe 1: Two-Site DR with Object Mirroring on Both Sites

For customers who use GFS for disaster recovery and keep all online files and snapshots mirrored to object storage at both sites:

```
place-on-SITEA-S3       Applicability: TRUE
place-on-SITEB-NFS      Applicability: IS_LIVE AND LAST_USE_AGE < 90*DAYS
place-on-SITEB-S3       Applicability: TRUE
```

Complement with snapshots plus `versioning-1-day` / `undelete-1-day`, excluding churny temp files (for example `.tmp`,`.docx`, `.xlsx`, `~.pptx`).

10.2. Recipe 2: DSX Online Tier with Cloud Down-Tiering

The notation below is conceptual, not literal CLI or API syntax — unlike Recipe 1 above, whose applicability expressions match real `objective-create` syntax once quoted for the shell. Translate the intent (mirror briefly, then age out) into real objectives and applicability expressions for your environment.

Keep two online instances briefly so a new file has time to reach the cloud, then let it age out to a single cloud instance:

```
dsx_local_mir_24h   IF MODIFY_AGE < 24*HOURS && DATA_ORIGIN_LOCAL  THEN SLO('dsx_mirror')
dsx_local_2weeks    IF MODIFY_AGE < 2*WEEKS  && DATA_ORIGIN_LOCAL  THEN SLO('dsx_any')
dsx_remote_24h      IF MODIFY_AGE < 24*HOURS && DATA_ORIGIN_REMOTE THEN SLO('dsx_any')
```

`dsx_mirror` creates two instances across named DSX nodes; `dsx_any` creates one on any available DSX.

Part 5 — Operate

Chapter 11. Monitoring and Alerting

Because replication is eventually consistent, continuous monitoring is essential to keep performance and consistency predictable.

11.1. What to Watch

Replication latency

Monitor replication-lag metrics (available via API, CLI, GUI, and Prometheus) so the metadata path stays responsive and lag does not consistently exceed your thresholds. In the Management GUI, the **Global File System** section shows replication-latency charts (1-hour, 1-day, 1-week) per share.

Default alert threshold

By default, an alert is raised when replication lags beyond **300 seconds**. Configure alert thresholds for replication latency in the GUI (per-share, on the **Global File System** tab of the **Details** view) or via the API/CLI.

11.2. Integrations

- **Grafana / Prometheus** — track replication health in real time and detect delays and failures before they become critical with custom dashboards and alerts. To turn on the exporters and set up dashboards, see Monitoring Cluster Health in the Administration Guide.
- **API** — everything shown in the GUI is available via API for custom monitoring.
- **SNMP / Syslog** — forward Hammerspace alerts into existing enterprise monitoring.

11.3. A Healthy GFS at a Glance

- Replication lag well under the alert threshold and not trending upward.
- Participants showing Admin/Oper state Up across all shares.
- Shared OSV capacity comfortably ahead of in-flight transfer (see the 14-day guideline in Prerequisites, Sizing, and Networking).
- No stale object-volume reservations or sites stuck in a removing/cleaning state (see Troubleshooting: Removal and Decommissioning).

Chapter 12. Protecting Data: Snapshots, Versioning, and Undelete

GFS offers three complementary mechanisms for protecting data: snapshots, file-granular versioning (`log-xfer`), and undelete. This chapter explains when to use each and how they interact with replication.

12.1. Snapshots

Snapshots provide a consistent point-in-time recovery reference and support long-term archival. Current best practice is a **single snapshot per day on one site**; avoid high-frequency (for example hourly) full snapshots.

Taking a real snapshot initiates copy-on-write for files currently open for write, which can be resource-intensive when the underlying storage lacks offloaded cloning. The snapshot's point-in-time therefore has some "fuzz": files opened for write after the snapshot also copy-on-write their data.

To create a snapshot from the admin CLI, use `file-snapshot-create` with the `--now` flag, giving a file path (wildcards are supported to cover an entire share). The path starts with the share's export path, not its name: `/home` for the `Home` share this guide creates with `share-create --path /home`. A share created in the GUI gets a path derived from its name (`/Home`), so use whichever path the share actually has — `share-list --name Home` shows it as `Path:`.

```
file-snapshot-create --filename /home/* --now
```

Expected output:

```
total 1
/home/.fsnapshot/file/2026-09-23T19-17-45-0644-0
```

Each snapshotted file appears as a separate, read-only, timestamped copy under `.fsnapshot/<original filename>/<timestamp>`.

In this guide's own testing (5.2 and 5.3, MD5-verified against files never previously read at the remote site), snapshot data was readable from every participating site, not only the site where the snapshot was created.

12.2. File Versioning and Undelete

`log-xfer` versioning creates a local version of a file when ownership changes, acting as protection against user errors and conflicts. Undelete preserves deleted files for a set period.

`.fsnapshot` (covered above, under Snapshots) and `.snapshot/current` (below) are two distinct, unrelated mechanisms despite the similar directory names. `.fsnapshot` holds true point-in-time snapshot copies created by `file-snapshot-create`. `.snapshot/current` is a live

view of the current directory that surfaces retained versions created automatically by `log-xfer` and `undelete` — not frozen point-in-time copies.

Undelete and `log-xfer` retain files in `.snapshot/current`. Without a snapshot **delete** schedule (or GFS replication that ages them out), these retained files consume space indefinitely. Pair undelete/versioning with a snapshot schedule to reclaim space. (Ties to DOCS-37 / HS-26900.)

Part 6 — Decommission

Chapter 13. Removing a Shared Volume and Decommissioning a Site

Every add path needs a documented remove path. This chapter covers the supported way to remove a participant site from a share, and what to know about fully decommissioning a site — including what that does and does not do to data, objectives, and replication relationships.

13.1. Removing a Participant from a Share

`gfs-participant-remove` (and the equivalent action in the Management GUI) is the supported way to remove a site's participation in a share. This does not delete data — it stops the site from participating in that share's replication.

```
gfs-participant-remove --share-name Home --site-name "SITE B"
```

Useful options:

`--force-master-acquisition`

: Only needed if a quorum of participants can't agree which site is master for replication management — typically after a network partition. It is not a normal part of removing a participant, and carries the warning that it "may leave the share replication configuration in an undefined state," so use it only to recover from that specific situation, not routinely.

`--ignore-remove-failures`

: Proceeds even if a remote site is unreachable or was already removed. The same undefined-state warning applies — use only when you understand why the remote side can't be reached.

`gfs-participant-remove` does not transfer or re-home the data that the departing site owns. If the site being removed owns data that no other participant has a copy of, that data becomes unavailable once the site is removed. Today, on 5.2 and 5.3, there is no supported step that moves ownership away from a site before you remove it — see "Decommissioning a site" below.

13.2. Decommissioning a Site

Decommissioning means more than removing a participant: it means safely retiring a site altogether, with confidence that none of its data is stranded on a site that's being taken away.

On 5.2 and 5.3, there is no supported, self-service procedure for this today. Removing every participant a site holds (above) does not, by itself, guarantee its data ownership moved anywhere first. If a site you're retiring may still own data, contact Hammerspace Support before removing it rather than improvising a sequence — Support's current process for this is handled case by case and isn't a single documented procedure this guide can walk you through yet.

Part 7 — Troubleshoot

Chapter 14. Troubleshooting: Deployment

Common problems encountered while setting up a Global File System.

14.1. gfs-participant-add: Fails to Add a Site

Error:

```
gfs-participant-add: site 'SITE Q' not found.
```

The site cannot find the `SITE Q` identifier in the configuration exchanged between sites. Either the `--site-name` was misspelled (input is case-sensitive) or a shared object storage volume was not set up correctly between the sites.

14.2. gfs-participant-add: Site Is Already a Participant

Error:

```
gfs-participant-add: Site SITE B is already participant 1 of share Home.
```

The share is already replicating to that site. No action needed.

14.3. Error: Bad Credentials

Error:

```
gfs-participant-add: Unable to ascertain the software version running at 198.51.100.10. Cause: Remote
site '198.51.100.10' returned error: bad credentials
```

The username or password supplied to `gfs-participant-add` was incorrect.

14.4. A Share with That Name Already Exists on the Remote Site

Error:

```
gfs-participant-add: Task failed.
ID:                     102f3726-f50c-4fa6-be6e-07e47117ed41
Name:                   gfs-participant-add
Status:                 FAILED
Status message:         Remote site '198.51.100.10' returned error: share 'Apps' already exists.
Created:                2026-09-23 19:18:01 UTC
```

```
Started:                2026-09-23 19:18:01 UTC
Ended:                  2026-09-23 19:18:02 UTC
Duration:               0s
Params:                 name: Apps::SITE B; created-by-name: admin; local-share-name: Apps; created-by:
Uoid [uuid=4cf681ae-56bc-4d07-9832-c6d8feeaf475, objectType=USER]; local-share-internal-id: 2
Context:                remote-participant-id: 2; remote-site: Site(internalId=1, name=SITE B,
siteMgmtAddress=198.51.100.10, siteDataAddress=198.51.100.10, overrideMgmtAddress=null,
overrideDataAddress=null, mgmtAddress=198.51.100.10, dataAddress=198.51.100.10, type=REMOTE)
```

The command reports `Task failed.` and prints the failed task; its `Status message:` line gives the cause. You cannot establish a replication relationship with an already-existing share on the remote site. Rename and move the share on the remote site, then retry. Share names are case-insensitive, so a different-case name on the remote site does not avoid this collision.

14.5. Replication Port-Check Failures

Symptom: participant-add fails, or participants flip to Suspected/Down, with a `REPLICATION_PORT_CHECK_FAILED` event. Confirm that all four Anvil ports (443, 9097, 9298, 9299) are open end-to-end between every pair of sites, and that the DME replication REST service is running. See Prerequisites, Sizing, and Networking.

Chapter 15. Troubleshooting: Removal and Decommissioning

15.1. Data Not Transferred Before a Site Is Removed

Removing a participant with `gfs-participant-remove` does not move that site's data ownership anywhere first — this is a known limitation of the supported removal flow today (5.2/5.3), not a bug with a pending fix. If the site being removed solely owned some data, that data becomes unavailable once the site is gone. See the warning in Removing a participant from a share.

15.2. OSV Stuck in "cleaning" / Decommission Stuck

Symptom: shared-OSV decommission stalls, often near a high percentage with few chunks remaining, or stays in a cleaning state. One specific cause is confirmed (below); other causes are not yet confirmed by engineering.

Known case: stuck at a fixed low percentage (5.3 regression, fixed in 5.3.1)

One confirmed cause (HS-42342): `object-volume-remove` could get stuck reporting a fixed non-zero instance count — for example `EXECUTING` at 38% with a `Decommissioning: 1 instances remaining` status message — even after the object actually responsible had already been cleaned up. This happened if the volume held a corrupt/PDL (partial-data-loss) object when removal started: the internal cleanup process could give up and never re-check once that object was later removed, leaving the task looking permanently stuck rather than slowly progressing.

This was a **5.3 regression**, introduced earlier in the 5.3 line and fixed in **5.3.1** — it did not affect 5.2. Our 5.3 test build (5.3.1-1201) already includes the fix, so this specific case could not be reproduced live during this pass. If a decommission is reported stuck at an unchanging percentage with a nonzero remaining-instance count on an early 5.3.0/5.3.1 build, and a corrupt/PDL object was involved, this is a likely explanation — upgrading resolves it without needing to cancel or restart the removal. This is one specific, confirmed cause of this symptom; other causes are still pending engineering confirmation per the CAUTION above.

Appendices

Appendix A: Glossary of Terms

Alignment

Reports whether a file currently adheres to all, some, or none of its applied objectives.

Anvil

A Hammerspace metadata server. Manages all GFS metadata and hosts the API and Management GUI. The only node type that communicates across sites.

CGC (Coordinated Garbage Collection)

A process that removes objects from buckets that no longer have references from any site.

COW split (copy-on-write split)

The copy-on-write operation triggered when a file open for write is captured by a snapshot or cloned; can momentarily affect instance counts.

DSX (Data Services node)

Transfers data to/from object storage, stores data on local storage, and presents NFS, SMB, and S3 to local clients. Never talks to DSX nodes at other sites.

Global file share

A share added to more than one Hammerspace cluster, where each cluster sees the same contents.

Hammerspace Toolkit (HSTK)

A collection of Python-based extensions for command-line interfaces (Linux Bash, Windows CMD/PowerShell) used to query and manipulate file system metadata — for example, `hs eval -e participants <PATH>` in The Shared Object Storage Volume in Depth. Installed separately from the base Hammerspace software; see the Hammerspace Support Hub for the current download and installation instructions.

Instance

A file written to underlying storage. The same file can have instances in multiple locations and remain one file — distinct from a copy.

Objective

A rule that defines intent for managing files, using metadata criteria to orchestrate placement and data services.

Participant

A site that takes part in a global share's replication relationship, identified by a Participant ID.

Shared bucket

An object-storage bucket reachable by all participating sites, used to transfer file data between them.

Shared Object Storage Volume (shared OSV)

A shared bucket added to a Hammerspace cluster as a shared resource; the data path between sites. Supports chunking, deduplication, compression, and encryption, which must match across sites.

Site

A Hammerspace cluster, which may or may not be at a distinct physical location.

Site ownership

The property that the site which last wrote a file owns it; ownership governs cross-site mobility and conflict resolution.

SLO (Service-Level Objective)

A target measured by service-level indicators; often an objective to place file instances on specific storage.

Volume

A storage container Hammerspace uses to store instances — DSX drives, NFS exports, or object-storage buckets.

Appendix B: Additional Documentation Resources

This guide focuses on Global File System functionality. For base-product and related topics, see the following Hammerspace titles (available on the Hammerspace Licensing and Delivery Portal and the Hammerspace Support Hub):

- {title-install-guide}
- {title-config-guide}
- {title-admin-guide}
- {title-cli-reference}
- {title-s3-guide}
- {title-hscl}
- {title-release-notes}

Related Support Hub knowledge articles of special interest:

- {kb-objectives-url}[{kb-objectives-title}] {kb-login-note}
- {kb-hstk-url}[{kb-hstk-title}] {kb-login-note}
- {kb-assimilation-url}[{kb-assimilation-title}] {kb-login-note}