CORE SYNC

Complete System Report, Installation Guide, and Operations Runbook

Product version	0.4.1
Prepared for	First Age Interactive
Production URL	https://sync.firstageinteractive.com
Report date	September 12, 2026
Classification	Private studio technical documentation

Document Control

This report describes the deployed Core Sync system as of version 0.4.1. It combines product documentation, technical architecture, installation instructions, daily workflows, administrative procedures, backup operations, troubleshooting, and recovery guidance.

Item	Value
System owner	First Age Interactive
Application	Core Sync
Unity support	Unity 6 (6000.0 or newer)
Primary host	Relay9 VPS
VPS address	15.204.38.154
Public hostname	sync.firstageinteractive.com
Server installation	/opt/core-sync
Unity package ID	com.firstage.core-sync
Package source	https://sync.firstageinteractive.com/packages/core-sync.git#v0.4.1

Note: Do not place passwords, SSH private keys, database credentials, or the Restic password inside a Unity project or a Core Sync repository. This report intentionally identifies credential locations without reproducing the secrets.

Contents

1. Executive Summary

Core Sync is First Age Interactive's private, self-controlled version control platform for Unity 6 projects. Each developer retains a complete working Unity project, while Core Sync stores verified file chunks, immutable project snapshots, membership data, and exclusive asset locks on the studio VPS. The Unity Editor package provides the working-copy interface; the HTTPS server provides authentication, storage, history, coordination, monitoring, and administration.

The design emphasizes recoverability and control. Uploaded content is addressed by SHA-256, files are reconstructed from verified chunks, snapshots are immutable, and server backups preserve both the PostgreSQL metadata and the stored object data required to rebuild repositories.

Current production state

Component	Current state
Core Sync server	Online behind HTTPS at sync.firstageinteractive.com
Product version	0.4.1; health API and web footer share the same package version
Database	PostgreSQL 17 Alpine in Docker with a persistent named volume
Object storage	Persistent Docker volume with hash-addressed files
Unity delivery	Git-based Unity Package Manager repository with version tags
Backups	Encrypted Restic snapshots: latest 2, 7 daily, 4 weekly, 3 monthly
Monitoring	Overview health panel, storage dashboard, activity log, backup status
Capacity at report time	150 GB VPS volume; approximately 142 GB available

Core operating principle

Local edit -> scan -> review -> lock when required -> submit -> immutable server snapshot -> teammate Get Latest. Changes that have not been submitted remain only on the developer's computer and are outside server backup coverage.

2. Product Scope and Capabilities

Accounts, teams, and repositories

- First-owner bootstrap registration; public registration closes after the owner exists.
- Email and password authentication with 30-day sessions.
- Optional Remember Me stores a session token and email on the workstation; it never stores the password.
- Repository creation, selection, invitation codes, member onboarding, role changes, and member removal.
- Owner, admin, and member access levels. Destructive repository deletion is owner-only and requires exact-name confirmation.

Unity project synchronization

- Tracks Assets, Packages, and ProjectSettings.
- Preserves Unity .meta files and GUID relationships.
- Excludes generated or machine-local folders: Library, Temp, Logs, obj, Builds, and UserSettings.
- Streaming SHA-256 scanner avoids loading an entire large project into memory.
- Metadata-assisted incremental scans avoid rehashing unchanged files.
- Files are divided into verified chunks for upload, caching, retry, and reconstruction.
- Downloads use a persistent verified chunk cache and skip unchanged local files.
- Pause, resume, cancel, retry, progress, throughput, and ETA controls support large transfers.

Coordination and protection

- Immutable snapshots form a chronological repository history.
- Stale-snapshot protection checks the repository before upload and again atomically at commit time.
- An outdated developer is blocked before large files upload and shown server-changed paths.
- Exclusive asset locks protect files that cannot be merged safely, with visible Unity Project-window badges.
- Activity records submissions, restores, Get Latest operations, locks, invitations, membership changes, repository creation, and backups.

Operations and visibility

- Overview provides live status for the server, database, object storage, backups, and Unity package feed.
- Storage reports VPS capacity, repository physical storage, logical project size, type breakdown, largest files, backup size, and cleanup eligibility.
- Capacity warnings appear at 70, 85, and 95 percent.
- Backups page shows last success, state, recovery points, integrity status, schedule, and a Back Up Now control.

3. Architecture and Data Model

Logical architecture

```
Unity 6 Editor
| HTTPS + bearer session
v
Caddy reverse proxy / TLS
|
v
Core Sync Node.js / Express service
|--------------------|-----------------------|
v v v
PostgreSQL 17 Object volume UPM Git repository
(metadata) (SHA-256 chunks) (versioned package)
| |
+----------+---------+
v
Restic encrypted backup repository
```

Server components

Component	Responsibility	Persistence
Caddy	TLS termination and public reverse proxy	Caddy data/config volumes
Core Sync app	API, authentication, dashboard, package Git HTTP backend	Source at /opt/core-sync; container rebuilt for releases
PostgreSQL	Users, sessions, projects, membership, snapshots, manifests, locks, invitations, activity	core-sync_coresync_database volume
Object store	Verified chunk files addressed by SHA-256	core-sync_coresync_objects volume
Restic	Encrypted, deduplicated rotating backups	/var/backups/core-sync/restic
systemd	Nightly backup, weekly verification, manual request relay	Units under /etc/systemd/system

Principal database records

Record	Purpose
users	Email, nickname, and password hash
sessions	Hashed bearer/session tokens and expiration
projects	Repository identity and creator
project_members	Per-project owner/admin/member relationship
objects	SHA-256, byte size, and storage key
project_objects	Project ownership/reference mapping for objects
snapshots	Parent link, message, JSON manifest, creator, timestamp
asset_locks	Exclusive path, owner, and timestamp
project_invites	Hashed join code, email, role, expiration, and redemption
activity_events	Chronological audit-style operational events

Snapshot and object behavior

A snapshot manifest describes the complete tracked project state. Each manifest file stores its project-relative path, logical size, complete-file hash, and ordered chunk hashes. Chunks are uploaded only when the server does not already possess them. Multiple files, snapshots, or repositories can therefore reuse identical content without duplicating the underlying object.

> **Note:** Logical project size and physical server storage are intentionally different. Logical size totals current snapshot files. Physical repository storage totals unique deduplicated objects associated with the repository.

4. Security Model

Authentication and authorization

- Passwords require at least 12 characters during account creation and are stored as password hashes.
- Session tokens are stored by the server as hashes and expire after 30 days.
- Web cookies are HTTP-only, same-site strict, and secure in production.
- API operations verify repository membership. Owner/admin restrictions are enforced server-side.
- Invitation codes are random, stored as hashes, tied to an invited email, expire after seven days, and become unusable after redemption.
- The Back Up Now web action writes only a narrow request marker. The application container is not granted Docker or systemd control.

Content integrity

- Upload paths must remain inside Assets, Packages, or ProjectSettings.
- Chunk addresses are SHA-256 hashes; the server verifies uploaded content against the claimed hash.
- Downloads are hashed again before use.
- A snapshot is created only after every referenced chunk exists.
- Restored files are staged and verified before replacing project files.
- Core Sync preserves its own UPM dependency when restoring Packages/manifest.json.

Secrets and privileged materials

Secret	Location / handling
SSH private key	Administrator workstation only; never upload to Core Sync
Server .env	/opt/core-sync/.env; contains database/session configuration
Restic password	/etc/core-sync-backup/restic-password; root-only on VPS
Recovery copies	C:\Users\Ghostdog Studio\Documents\Core Sync Recovery and E:\Core Sync Recovery
User passwords	Known only to users; never stored by the Unity Remember Me option

Note: Anyone who obtains the Restic password and the encrypted repository can decrypt backups. Keep recovery copies private and outside all synchronized project folders.

5. Production Server Installation

Prerequisites

- A Linux VPS with root or equivalent administrative access.
- A DNS A record for sync.firstageinteractive.com pointing to the VPS public address.
- Docker Engine and docker-compose 1.29 or compatible Compose tooling.
- Caddy or another TLS reverse proxy attached to the external core_sync_edge Docker network.
- Ports 80 and 443 available publicly; Core Sync port 3100 remains bound to 127.0.0.1.
- Restic and systemd for the documented backup configuration.

Directory and environment setup

```
install -d -m 750 /opt/core-sync
cd /opt/core-sync
# Copy the Core Sync server release into this directory.
cp .env.example .env
chmod 600 .env
```

Set unique production values in **/opt/core-sync/.env**. Required values include PORT, POSTGRES_PASSWORD, SESSION_SECRET, MAX_CHUNK_MB, APP_ORIGIN, and bootstrap registration behavior. Use a long random database password and a session secret of at least 32 random characters.

```
PORT=3100
POSTGRES_PASSWORD=<random strong value>
SESSION_SECRET=<at least 32 random characters>
MAX_CHUNK_MB=32
APP_ORIGIN=https://sync.firstageinteractive.com
BOOTSTRAP_REGISTRATION=true
```

Docker deployment

```
cd /opt/core-sync
docker-compose -f compose.production.yml build
docker-compose -f compose.production.yml up -d
docker-compose -f compose.production.yml ps
```

The Compose stack creates persistent database and object volumes. The application waits for PostgreSQL health before starting. Its local health check calls http://127.0.0.1:3100/api/health.

Reverse proxy and TLS

Attach Caddy to the external Docker network named **core_sync_edge** and proxy the public hostname to the application alias **core-sync-app:3100**. Caddy should obtain and renew the TLS certificate automatically. Do not expose PostgreSQL or the raw application port publicly.

```
sync.firstageinteractive.com {
reverse_proxy core-sync-app:3100
}
```

Installation verification

```
curl https://sync.firstageinteractive.com/api/health
# Expected: {"status":"ok","service":"core-sync","version":"0.4.1"}

docker-compose -f /opt/core-sync/compose.production.yml ps
docker logs --tail 100 core-sync_app_1
```

First-owner bootstrap

Keep BOOTSTRAP_REGISTRATION=true only while creating the first owner. In Unity, open Window > Core Sync, test the server, enter the owner email, nickname, and a password of at least 12 characters, and select Create First Owner. The API refuses additional first-owner registrations once a user exists. For defense in depth, set BOOTSTRAP_REGISTRATION=false afterward and recreate the app container.

6. Unity Package Installation

Supported environment

- Unity 6 / 6000.0 or newer.
- A complete local Unity project working copy.
- Network access to https://sync.firstageinteractive.com.
- A Core Sync account or invitation code.

Install through Unity Package Manager

In Unity, open **Window > Package Manager**. Select the plus menu, choose **Install package from Git URL**, and enter:

```
https://sync.firstageinteractive.com/packages/core-sync.git#v0.4.1
```

Wait for compilation to finish, then open **Window > Core Sync**. A tag-pinned URL provides a predictable version. When a new tested release is available, update the tag in Package Manager or Packages/manifest.json.

Recommended Unity settings

Open Core Sync Settings and select **Apply Recommended Unity Settings**. This enables Force Text serialization and Visible Meta Files. Text serialization improves the inspectability of scenes and prefabs, while visible .meta files preserve GUID references between developers.

Files tracked and excluded

Tracked	Excluded
Assets and all .meta companions	Library
Packages, including manifest and lock data	Temp
ProjectSettings	Logs
Scenes, prefabs, code, models, textures, audio, video, and configuration	obj, Builds, UserSettings

Note: Core Sync protects only content that has been submitted. Unsaved or unsubmitted workstation changes are not included in VPS backups.

Updating an installed client

- Confirm the target version has been published and tested.
- Create or confirm a recent server backup.
- Update the package Git tag.
- Allow Unity to resolve and compile.
- Open Window > Core Sync and confirm the server connection.
- For version 0.4.x tracking initialization, use Connect & Get Latest once if the local base snapshot is not yet recorded.

7. Initial Setup and Developer Onboarding

Create the first repository

- Sign in to the web panel or Unity client as the first owner.
- Create a repository from web Settings and give it the project name.
- In Unity Settings, refresh repositories and select the correct repository.
- Choose Connect & Get Latest. An empty repository reports that no snapshots exist; this is expected.
- Scan the project, enter an informative snapshot message, and submit the initial snapshot.

Invite another developer

- In the web panel, select the repository and open Team.
- Enter the developer email and choose member or admin.
- Generate the invitation and send the displayed code through a private channel.
- The developer installs Core Sync, creates the invited account with the same email, or signs in and redeems the code.
- The developer selects the repository and runs Connect & Get Latest before editing.

Roles

Role	Typical authority
Owner	Full repository administration, role changes, repository deletion, backup controls
Admin	Team invitations, member management within server rules, normal repository operation
Member	Scan, download, submit, lock, unlock owned locks, and view permitted repository information

Remember Me

Remember Me stores the 30-day session token and remembered email in Unity Editor preferences. It does not store the password. Signing out clears the saved session. Shared workstations should not use this option.

8. Daily Operating Procedures

Start of work

- Open the correct Unity project and wait for compilation/import to finish.
- Open Window > Core Sync and confirm the selected repository.
- Choose Get Latest before beginning work.
- Review incoming file changes and allow Unity to refresh.
- Lock binary or high-risk assets before editing them.

Scan and submit

- Save scenes and assets in Unity.
- Open Changes and select Scan Project.
- Review added, modified, and deleted files.
- Enter a snapshot message that explains the completed change, not merely "update".
- Submit. Core Sync checks server freshness before uploading and again during the atomic commit.
- Wait for Snapshot complete. Do not close Unity during an active transfer unless cancelling intentionally.
- Release locks that are no longer needed.

Good snapshot messages

Weak	Useful
Changes	Add forest combat arena and update lighting
Fix	Fix wolf animator transition from stagger to idle
Stuff	Import blacksmith audio set and connect forge ambience
Update	Tune inventory UI anchors for ultrawide resolutions

Transfer controls

- Pause temporarily stops active chunk transfer while preserving progress.
- Resume continues the transfer.
- Cancel stops the current operation safely.
- Transient failures retry up to four times with exponential delay.
- The verified local cache allows later restores to reuse chunks already downloaded.
- Clear Cache removes only verified local Core Sync cache data, not Unity assets or server snapshots.

End of work

- Save and rescan.
- Submit all intended work.
- Verify the new snapshot appears in History or web Activity.

- Release completed locks.
- Do not assume locally modified but unsubmitted files are protected by the server.

COLLABORATION SAFETY

9. Snapshots, Conflicts, and Locks

Immutable snapshots

Each successful submit creates a new immutable snapshot with a parent pointer, author, message, timestamp, and complete manifest. Restoring an older snapshot updates the working copy but does not rewrite server history. A later intentional submission creates another snapshot.

Stale-snapshot protection

Every Unity working copy stores the snapshot ID it is based on. Before upload, Core Sync asks whether that ID is still the repository head. If another developer has submitted, the client is blocked and displays paths changed on the server. The server repeats the comparison under a project-level database lock during snapshot creation, preventing two simultaneous submissions from both succeeding against the same parent.

```
Developer A and Developer B both start at snapshot 100.
Developer A submits snapshot 101.
Developer B attempts to submit from snapshot 100.
Core Sync blocks B before large uploads and lists server-changed paths.
Developer B runs Get Latest, resolves local overlap, rescans, and submits from 101.
```

Responding to a stale submission

- Copy or commit any especially risky local work to a separate safe location if desired.
- Read the changed-path list.
- Choose Get Latest.
- When Core Sync warns about local changes, review the risk before allowing replacement.
- Open affected Unity scenes, prefabs, or files and verify the combined state.
- Rescan and submit only after the project is correct.

Exclusive locks

Locks reserve a project-relative file path for one developer. The Unity overlay displays lock badges. Core Sync also handles .meta companions when locking selections so asset identity cannot be edited independently of the asset.

Assets that should normally be locked

- Unity scenes when multiple editors cannot safely coordinate text merges.
- Prefabs undergoing structural edits.
- PSD, Blender, FBX, audio, video, and other binary source files.
- Large textures or art files.
- Animator controllers or complex serialized assets when only one person should modify them.

Note: A lock coordinates people; it is not a backup and does not upload the file. Submit the finished work before unlocking.

10. Web Administration Panel

Page	Purpose
Overview	Repository count, active repository, and live service status
Changes	Directs working-copy changes to the Unity client
Snapshots	Recent immutable snapshot history
File locks	Current locked paths and owners
Team	Invitations, join flow, members, and roles
Storage	Capacity, project breakdown, largest files, backups, cleanup eligibility
Activity	Chronological operational events
Backups	Backup health, retention, integrity, and manual backup
Settings	Authentication, repository creation/selection, logout, and owner deletion controls

Overview health states

- Core Sync server: request handling is available.
- Database: the application can query PostgreSQL.
- Object storage: the persistent object filesystem is accessible.
- Backups: latest status from the host backup service.
- Unity package feed: the Git package repository is present and readable.

Storage interpretation

- VPS used is filesystem-wide usage, not Core Sync alone.
- Repository stored is physical deduplicated object storage associated with the selected repository.
- Current project is the logical total represented by the latest snapshot.
- Backups is the encrypted Restic repository size recorded after the latest successful backup.
- Unreferenced objects are objects no repository currently references; the dashboard reports but does not automatically delete them.
- Warnings activate at 70 percent notice, 85 percent warning, and 95 percent critical.

11. Backup and Recovery

Backup coverage

- Consistent PostgreSQL custom-format dump containing accounts, repositories, membership, manifests, snapshots, locks, invitations, and activity.
- All uploaded Core Sync object chunks, including submitted art, models, textures, audio, video, scenes, prefabs, scripts, and configuration.
- /opt/core-sync deployment source and configuration.
- Encrypted, deduplicated storage through Restic.

Schedule and retention

Policy	Value
Automatic run	Nightly around 3:15 AM VPS server time, randomized by up to 15 minutes
Always retain	Latest 2 snapshots
Daily	7
Weekly	4
Monthly	3
Integrity check	Weekly Sunday morning; rotating 5 percent data sample
Manual	Web Backups > Back Up Now or systemctl start core-sync-backup.service

Backup sequence

- Acquire a host lock so two backup jobs cannot overlap.
- Write a running status record.
- Create a consistent pg_dump inside the database container.
- Back up the dump, object volume, and deployment directory.
- Apply retention only after the new snapshot succeeds.
- Prune data unused by any retained recovery point.
- Write success/failure status and record successful backup activity.

Manual backup before maintenance

```
systemctl start core-sync-backup.service
systemctl status core-sync-backup.service --no-pager
journalctl -u core-sync-backup.service -n 100 --no-pager
```

The latest-two rule preserves a manual pre-upgrade recovery point even if another backup runs on the same calendar day.

Recovery materials

The Restic password is stored root-only at /etc/core-sync-backup/restic-password. Recovery copies exist on the C and E local drives. Losing both the repository and its password makes encrypted backups unrecoverable.

Safe restore outline

- Do not restore over a running production system.
- Identify the desired Restic snapshot.
- Restore into a new temporary directory.
- Run Restic integrity checks.
- Validate the PostgreSQL dump with pg_restore --list.
- Stop Core Sync before replacing live object data.
- Restore the database into a controlled PostgreSQL instance.
- Restore objects and configuration only after validating exact target paths.
- Start services, test health, authenticate, inspect repositories, and download a known snapshot.

```
RESTIC_REPOSITORY=/var/backups/core-sync/restic \
RESTIC_PASSWORD_FILE=/etc/core-sync-backup/restic-password \
restic snapshots

mkdir -p /var/lib/core-sync-restore-test
RESTIC_REPOSITORY=/var/backups/core-sync/restic \
RESTIC_PASSWORD_FILE=/etc/core-sync-backup/restic-password \
restic restore latest --target /var/lib/core-sync-restore-test
```

Note: The current automated repository lives on the VPS. The studio additionally maintains local project copies on two drives. Local working copies are useful additional protection but do not replace a tested server restore procedure.

12. Monitoring and Maintenance

Daily checks

- Overview shows all components online.
- Backups shows a recent successful run.
- Storage remains below warning thresholds.
- No unexplained failed activity or lock contention is present.

Weekly checks

- Confirm the weekly Restic integrity result is healthy.
- Review Activity for unexpected membership or invitation changes.
- Review largest files and growth trends.
- Confirm developers are using the current tested Unity package.

Before every server upgrade

- Use Back Up Now and wait for healthy status.
- Record the current application and Unity package versions.
- Copy replaced server files to timestamped rollback files.
- Build the new Docker image.
- Recreate only the application service unless database changes require more.
- Verify local and public health, web login, package Git refs, and a normal repository operation.

Useful commands

```
docker ps --format "{{.Names}} {{.Status}}"
docker logs --tail 100 core-sync_app_1
docker logs --tail 100 core-sync_database_1
df -h /
du -sh /var/lib/docker/volumes/core-sync_coresync_objects/_data
du -sh /var/backups/core-sync/restic
systemctl list-timers core-sync-backup.timer core-sync-backup-check.timer --all
journalctl -u core-sync-backup.service -n 100 --no-pager
```

Version release discipline

- Increment the Unity package semantic version.
- Update the changelog.
- Commit and create an annotated Git tag.
- Publish main and the version tag to the server package repository.
- Rebuild the app container so the embedded Git repository is current.
- Verify git ls-remote through the public HTTPS URL.
- Keep older tags for rollback and compatibility with projects pinned to them.

13. Troubleshooting

Symptom	Likely cause	Action
Server test fails	DNS, TLS, Caddy, app container, or firewall	Check public /api/health, Docker state, Caddy logs, and DNS A record
Sign-in fails	Incorrect credentials or expired/cleared session	Retry credentials; sign out/in; verify account exists
No repositories	Account is not a member	Create a repository or redeem the correct invitation
Submit disabled	No scan, message, login, or repository	Complete all prerequisites shown in Changes
Stale snapshot error	Another developer submitted first	Get Latest, verify overlap, rescan, submit
Asset already locked	Another member owns the path lock	Coordinate with owner; do not bypass the lock
Upload/download interrupted	Network or server interruption	Resume or retry; verified chunks avoid repeating completed work
Unity package disappears	Package manifest resolution or Git accessibility	Check Packages/manifest.json and public Git URL/tag
Package install errors	Bad cached revision, missing tag, or compile mismatch	Use exact tested tag; inspect Package Manager and Console errors
Backup warning	Backup script, repository lock, disk, DB dump, or Restic issue	Read systemd status and journal before clearing any lock
Storage warning	VPS filesystem approaching capacity	Review growth/largest files; expand storage before critical level
502 after deployment	App container is recreating or unhealthy	Wait for health check, inspect app logs and Caddy network alias

Restic lock handling

Never delete a Restic lock merely because a backup failed. First identify the recorded PID and verify that no Restic process is alive. Only then use restic unlock with the correct repository and password file. Rerun the backup and confirm healthy status.

Clear Local Core Sync Data

This Unity option clears the project's Core Sync baseline/cache and disconnects the repository ID. It does not delete Unity assets or server data. After using it, select the repository again and Connect & Get Latest before submitting.

14. Limitations and Roadmap

Current boundaries

- Core Sync does not protect workstation changes until they are submitted.
- It does not perform semantic three-way merges for Unity YAML or source code; stale work is blocked for manual resolution.
- Locks are cooperative coordination controls, not operating-system file locks.
- The activity feed starts from its deployment date; older actions were not fabricated.
- Backup failure notifications are persistent in the web panel, not email or SMS.
- The encrypted Restic repository currently resides on the same VPS; local project copies provide separate workstation storage.
- Storage categories use file extensions and the latest manifest; uncommon formats may appear as Other.
- A monthly automated full restore drill was intentionally not enabled.

Recommended future work

- Project-specific ignore rules managed from Unity.
- Read-only and more granular repository permissions.
- Downloadable CSV/JSON activity exports.
- Guided disaster-recovery tooling with explicit non-production restore targets.
- Optional external notification channels for backup and capacity failures.
- Storage trend history and growth forecasting.
- Administrative session revocation and account recovery workflow.

15. Quick Reference and Checklists

Developer quick start

- Install https://sync.firstageinteractive.com/packages/core-sync.git#v0.4.1.
- Open Window > Core Sync.
- Sign in or redeem an invitation.
- Select the repository.
- Apply recommended Unity settings.
- Connect & Get Latest.
- Lock risky assets.
- Edit and save.
- Scan Project.
- Enter a useful message and Submit.
- Release completed locks.

Owner onboarding checklist

- Create or select repository.
- Generate invitation for the exact developer email.
- Assign member or admin appropriately.
- Send code privately.
- Confirm the developer joins and completes Get Latest.
- Confirm package version compatibility.

Pre-upgrade checklist

- Back Up Now reports healthy.
- At least two recovery points are retained.
- Recovery password copies remain accessible.
- Current version and rollback files are known.
- Build succeeds.
- Deployment recreates only intended services.
- Public health, login, Overview, package Git, and one repository action pass.

Incident checklist

- Stop making destructive changes.
- Record the symptom and exact time.
- Check Overview, Backups, Storage, and Activity.
- Inspect app/database/systemd logs.
- Protect current data before repair.
- Restore into a temporary target first.

- Validate database dump and object paths.
- Document the fix and verify with a normal Unity workflow.

Key locations and addresses

Purpose	Location
Web panel and API	https://sync.firstageinteractive.com
Unity package	https://sync.firstageinteractive.com/packages/core-sync.git#v0.4.1
Server deployment	/opt/core-sync
Object data	/var/lib/docker/volumes/core-sync_coresync_objects/_data
Restic repository	/var/backups/core-sync/restic
Restic password	/etc/core-sync-backup/restic-password
Backup status/control	/var/lib/core-sync-backup/control
Local recovery kit C	C:\Users\Ghostdog Studio\Documents\Core Sync Recovery
Local recovery kit E	E:\Core Sync Recovery

Final operational rule

Note: Get Latest before work, lock non-mergeable assets, submit before relying on server protection, verify backups before maintenance, and restore only into a separately validated target.