HuskSync tasks

Task guides and troubleshooting for HuskSync. Check the installed release before changing configuration or data.

- Diagnose a player held in HuskSync synchronisation lock
- Choose which player features should synchronise between backends
- Coordinate a HuskSync database recovery across all writers
- Separate Redis failures from database failures in HuskSync
- Test keepInventory death handling before allowing server transfers
- Isolate a staging server from the production HuskSync group

Diagnose a player held in HuskSync synchronisation lock

HuskSync can lock player interactions while data is synchronising. A persistent lock is a symptom to investigate, not a reason to remove the protection that prevents competing item changes. Separate one player's state from a network-wide storage or messaging failure.

Record the starting point

Record the source and destination backend names, transfer time, affected account and HuskSync versions. Ask the tester to stop repeated transfers and item actions. Keep current player-data snapshots and a restricted configuration checkpoint before making a recovery change.

Work through the change

1. Run the documented `/husksync status` command with authorised access on the relevant components. Compare its result with startup and transfer-time logs.
2. Check database and Redis connection errors separately. Confirm the backend's server.yml name matches the intended proxy server ID and that cluster membership is deliberate.
3. Use a disposable account to test one controlled transfer through the normal proxy. Record whether the lock clears and whether an inventory marker reaches the destination.
4. Correct only the identified connection or identity mismatch, then restart the affected component through the normal maintenance process. Retest before asking the original player to reconnect.

Check the result

The controlled account should finish synchronising and regain ordinary interactions with the expected data. Check a return transfer too. Keep the original player's recovery state unchanged

until the path is healthy and staff have selected the correct authoritative snapshot.

If the result differs

If every tester remains locked, inspect shared services and version consistency. If only one account fails, compare its recorded snapshots and error context before applying a restore. Disabling locked-player command restrictions can expose data races without fixing the blocked operation.

Restore an ineffective trial configuration and keep affected admission paused if safe synchronisation cannot be demonstrated. Use the documented snapshot-recovery controls only with a current checkpoint and a chosen source. Do not delete player records or allow competing backend sessions to bypass the lock.

References and related help

- HuskSync maintainer reference 1
- HuskSync maintainer reference 2
- Test cross-server inventory synchronisation with HuskSync
- Find Minecraft logs and send useful troubleshooting details

Choose which player features should synchronise between backends

HuskSync can synchronise more than inventories, but not every feature belongs across every backend. Its feature switches define what it touches; disabled features use local player data. Decide the desired boundary before assuming a missing balance or changed location is a failed sync.

Record the starting point

List the data that should travel, such as inventory, ender chest or experience, and the data that should remain local. Record the generated configuration on each participating backend. Keep snapshots for a test account before changing the scope of live player state.

Work through the change

1. Read the current sync-features table and compare it with your design. HuskSync does not synchronise Vault economy balances or ordinary unlocked maps; those need their own supported arrangements.
2. Review the synchronization.features section in the installed configuration. Check location synchronisation carefully: the documented use assumes mirrored worlds where the same coordinates make sense.
3. Apply a consistent intended policy across the participating group during controlled maintenance. Do not copy a configuration containing another backend's server ID or credentials without review.
4. Transfer a test account with a counted inventory marker and distinct local state. Verify each enabled feature and one deliberately disabled feature on both destinations.

Check the result

The account should carry only the intended shared state. Document unsupported data separately so players do not mistake an economy or map limitation for missing inventory data. Confirm the behaviour again after reconnecting, since local player files can affect disabled features.

If the result differs

If a disabled feature appears to change, inspect another plugin that manages it and compare local player data. If location sync sends the tester somewhere unsafe, stop that trial and review whether the worlds are truly mirrored. Do not enable every feature simply to make configurations match.

Return the feature policy to its saved state if the trial creates an unintended boundary. Restore only the test account's required state using a chosen checkpoint. Keep live player records and world files intact while deciding which data should be shared.

References and related help

- HuskSync maintainer reference 1
- HuskSync maintainer reference 2
- Test cross-server inventory synchronisation with HuskSync
- Back up game files before making changes

Coordinate a HuskSync database recovery across all writers

A shared HuskSync database can be written by several backends. Restoring it while those writers remain active can mix old and new state or replace recovered data. Treat recovery as a coordinated network operation rather than a database file copied beneath running servers.

Record the starting point

Identify every participating backend, its cluster ID and the database being recovered. Select the intended recovery point and keep a fresh checkpoint of the current database and relevant configuration. Decide how player sessions and new progress will be handled before applying old state.

Work through the change

1. Read the installed HuskSync recovery and snapshot commands along with your database's supported restore procedure. Choose whether a targeted user snapshot or a wider database recovery is actually required.
2. Close the affected route to players and stop the relevant writers through normal controls. Confirm no backend or test copy still points at the production database.
3. Recover the chosen data using the supported database or snapshot procedure. Keep the pre-recovery checkpoint available privately and preserve the configuration mapping to each backend.
4. Start in a controlled order and test a disposable account before reopening admission. Check an intended recovered player record and a return transfer without creating duplicate sessions.

Check the result

Every active writer should use the same intended recovered state, and new test data should persist normally. Record which recovery point was used and what later progress may need reconciliation. A database service starting successfully does not establish application-level consistency.

If the result differs

If an old state reappears after a transfer, inspect another writer or cached active session before repeating the restore. If only one player is affected, a targeted snapshot recovery may be more appropriate than another network-wide rollback. Keep logs from the controlled startup.

Keep admission closed if consistency cannot be demonstrated and return to the fresh pre-recovery checkpoint only through the same coordinated process. Do not alternate database restores while players continue trading. Never delete the original checkpoint until the recovery has been reviewed.

> Create a backup from the CloudExa game panel
> A panel backup is separate from the plugin-specific record or remote archive discussed here.

References and related help

- HuskSync maintainer reference 1
- HuskSync maintainer reference 2
- Test cross-server inventory synchronisation with HuskSync
- Back up game files before making changes

Separate Redis failures from database failures in HuskSync

HuskSync has separate database and Redis configuration. A connection failure in one service does not prove the other is unavailable, and repairing one password does not automatically repair both. Compare status and the first relevant log message to identify the failed dependency.

Record the starting point

Record the failing backend, transfer timestamp and installed HuskSync release. Keep private copies of the database and Redis endpoint configuration. Redact passwords and usernames before sharing logs, and avoid changing several credentials at the same time.

Work through the change

1. Run authorised `/husksync status` and inspect startup or transfer-time errors. Identify whether the failing connection names the database, Redis, authentication or a timeout.
2. Compare the relevant configured host and port with the intended service. Localhost inside a game container refers to that container, not automatically to a separate database or Redis machine.
3. Ask support to check the specific source-to-destination path when a connection times out. Keep management services private; do not expose an unauthenticated Redis listener to every public address.
4. Correct the identified endpoint, access rule or credential through the documented configuration. Restart the affected component normally and repeat a controlled cross-backend transfer.

Check the result

Status and logs should show healthy dependencies, and the test account should transfer with the intended inventory state. Confirm a return transfer and a new saved snapshot where appropriate. A TCP connection alone proves reachability, not successful authentication or application operation.

If the result differs

An authentication error points to a different check from a timeout. A healthy database with failed messaging still warrants investigation of Redis; do not rewrite the database schema because the transfer stalls. Compare a working backend's redacted endpoint arrangement for useful differences.

Restore any ineffective trial endpoint change and keep the protected network boundary. Pause affected transfers if data consistency remains uncertain. Do not disable player locks, delete snapshots or reset shared services simply to suppress a connection error.

References and related help

- HuskSync maintainer reference 1
- HuskSync maintainer reference 2
- Test cross-server inventory synchronisation with HuskSync
- Find Minecraft logs and send useful troubleshooting details

Test keepInventory death handling before allowing server transfers

HuskSync gives special treatment to death snapshots and players who change servers while dead. With keepInventory, dropped-item assumptions can conflict with retained items. The maintainer documents a specific configuration and additional care for custom partial-retention plugins.

Record the starting point

Use a disposable account with counted low-value items and record keepInventory behaviour on each participating backend. Save the current HuskSync configuration and test-account snapshot. Check the server requirement for the documented ITEMS_TO_KEEP mode before changing it.

Work through the change

1. Read the current keep-inventory guidance and inspect synchronization.save_on_death. The documented arrangement uses ITEMS_TO_KEEP with save_empty_items and sync_dead_players_changing_server enabled; confirm it matches the installed release and supported Paper version.
2. Test a normal death and respawn on one backend. Compare inventory counts and the resulting snapshot before involving a server transfer.
3. Repeat a controlled death-screen disconnect and join the other backend through the normal route. Check the final alive state and inventory, then return to the first backend.
4. If a custom plugin keeps only selected items, review its supported HuskSync integration with its maintainer. A general keepInventory setting may not represent that plugin's item transformation.

Check the result

The test account should retain exactly the intended items with no extra death drops or duplicate recovery state. Record both the same-server and cross-server results. A successful ordinary respawn does not prove that changing servers while dead is safe.

If the result differs

If counts differ, stop the trial and compare snapshot contents, event ordering and the custom death plugin. The maintainer notes that advanced partial-retention setups may require a dedicated API hook. Avoid compensating items repeatedly while the source of duplication remains active.

Restore the saved test configuration if the change regresses death handling and recover only the disposable account's required state. Keep affected transfer behaviour restricted until verified. Do not delete all player snapshots or reset inventories to conceal an inconsistent test.

References and related help

- HuskSync maintainer reference 1
- HuskSync maintainer reference 2
- Test cross-server inventory synchronisation with HuskSync
- Recover a Minecraft player inventory from a backup

Isolate a staging server from the production HuskSync group

A copied backend can retain production HuskSync credentials and cluster membership. Starting it for testing may allow it to read or write real player state. Isolate shared services before the first staging login rather than relying on a different public Minecraft port.

Record the starting point

Keep the staging copy stopped while listing inherited database, Redis and cluster configuration. Record which production servers belong to the live group. Prepare separate authorised test storage and accounts, and keep the copied credentials out of public setup notes.

Work through the change

1. Read the installed HuskSync configuration structure. Review database and Redis destinations, cluster_id and the server ID in server.yml as separate settings.
2. Configure the test copy for deliberately isolated services and identities using the documented setup. A changed server name alone is not proof that it cannot touch production storage.
3. Ask support to verify that the staging service cannot reach production credentials or unintended endpoints where practical. Keep the test instance out of the public proxy route until isolation is checked.
4. Start with a disposable test account and confirm where its snapshot is stored. Verify that production snapshots and active player sessions remain unchanged before conducting transfer or recovery experiments.

Check the result

The staging account's changes should remain in the intended isolated environment. Record the test endpoints and cluster design without secrets. A useful test includes checking the production side for unintended writes, not simply observing that the copied server starts.

If the result differs

If test data appears in production, stop staging immediately and preserve the write timestamps. Compare all inherited connection settings rather than changing only the cluster label again. Review affected snapshots before deciding whether a targeted recovery is needed.

Keep the copy stopped until its isolation can be demonstrated. Restore any accidentally changed production state through a coordinated, checkpointed recovery. Do not delete production users or snapshots to make a test account disappear; preserve evidence and repair the boundary first.

References and related help

- HuskSync maintainer reference 1
- HuskSync maintainer reference 2
- Test cross-server inventory synchronisation with HuskSync
- Back up game files before making changes