Tune RoseStacker rules without losing named or custom mobs

Entity stacking changes how several mobs share a representation. Review gameplay and persistence alongside performance. A lower visible entity count does not establish that custom drops, named pets or another plugin's mob logic still behave as your players expect.

Preserve the data that represents existing stacks

Choose a RoseStacker release supported by your Minecraft version. Its generated configuration and per-entity settings take precedence over examples from older releases. Stop the server and back up the worlds, RoseStacker configuration and its data store as a matched set. Download and check the backup before introducing stacking to an established world.

Avoid removing the plugin from a live installation as a quick rollback. Existing stacks can rely on its stored data. First document the current stacking rules, affected worlds and any plugins that customise mobs or loot. Use a copy of that setup for the first trial.

Narrow the trial

1. Inspect the generated world inclusion or exclusion settings and per-entity files. Choose one test world and one ordinary mob type instead of applying new rules to every world.
2. Check the comments for the exact release before changing a setting. Keep custom-named entities excluded where your community treats names as individual ownership or identity.
3. Review MythicMobs integration if present. Current maintainer configuration documents `misc-settings.mythicmobs-allow-stacking` and recommends leaving it false except in narrowly compatible uses.
4. Preserve custom-drop and death-event settings until their owning plugins' integration requirements are understood.

The current source also documents `global-entity-settings.dont-stack-custom-named`. Confirm that key exists in your generated configuration and check its actual value; do not infer the setting from the display name shown above a stack. Per-entity or release-specific rules may affect the result.

Compare ordinary and special mobs

In the private trial area, start with two ordinary mobs of the chosen type. Record whether they combine and the resulting displayed count. Repeat with a named test mob and, where relevant, a disposable custom mob created through its owning plugin. Keep pets and players' production farms outside the test.

Use small quantities when checking drops and experience. Compare a single death, partial stack loss and any configured whole-stack behaviour. Keep a written expected result for each test instead of assuming a larger drop pile means the integration is correct. Do not enable death events for an entire large stack on a public farm without reviewing the workload.

Check persistence before release

After recording test counts, perform a normal stop and start on the copy. Return to the area and compare the saved counts and special-mob behaviour. Inspect console errors. Commands such as `/rs stats` can provide plugin information, but they do not replace checking the specific entities.

Restore the recorded rule values if a trial fails and follow the release's supported handling of existing stacks. Avoid `/rs clearall` as a fix; its purpose is removal. Escalate a persistence mismatch with redacted logs and the exact settings rather than deleting the data store or reinstalling the server.

Related CloudExa guides

back up the server files · collect the relevant Minecraft logs · plan a compatible Paper and plugin update

Official references

- RoseStacker introduction
- RoseStacker commands
- Maintainer configuration reference

Revision #1
Created 2026-11-09 10:42:41 UTC by CloudExa Hosting Admin
Updated 2026-11-09 10:42:41 UTC by CloudExa Hosting Admin