Test a Rust staging branch on a separate server identity

Rust's staging branch lets administrators examine changes before using them on a normal server. Treat it as a separate environment. A staging test should never silently change the software or save files of your active community server. Ask support to supply a separate test service if your game panel does not expose branch selection.

Define the test boundary

Write down the production branch, Rust build, plugin loader and identity. Decide what you are trying to learn: whether a particular plugin loads, whether a custom map starts or whether a reported bug reproduces. A narrow question makes it easier to distinguish a useful test from an unrelated software upgrade.

Give the test service its own installation, identity, connection allocations and credentials. Avoid sharing a writable world directory with production. If you need representative world data, begin with an authorised stopped checkpoint and copy it into the test environment. Keep an untouched copy outside the test installation.

Select the branch through the supported control

1. Confirm the test service can start on its current branch before changing anything. Record that baseline and its configuration.
2. Use the panel's branch setting or have support apply the change. Facepunch's server guide gives `app_update 258550 -beta staging` as a SteamCMD branch example. This is an installer instruction, not a command for the running Rust console.
3. Keep production's automatic updater pointed at its original branch. Do not edit a shared startup template to change one test service.
4. Check that your test client uses the corresponding Rust branch. A stable client failing to join staging is not proof of a broken network.

5. Verify that any loader or plugin build supports the tested branch. An ordinary public-release plugin package is not automatically compatible with unreleased server changes.

Run a controlled check

Start with only the components needed for your question. Capture the server build and startup log, then connect with the matching client. Follow a short written scenario and record expected versus observed behaviour. For a plugin, check its actual function and persistence after a normal restart, not merely whether the source compiled.

If you use copied player data, restrict access and remove or disable production webhooks, reward integrations and external database connections. Do not let staging send real purchase rewards or overwrite shared records. Avoid publishing performance comparisons unless the environments and measurement method genuinely support them.

Return to the public branch safely

Facepunch notes that returning from a beta can require explicitly selecting the `public` branch. Ask support to confirm the effective branch and build after the change. Keep the original pre-test world checkpoint; a branch rollback does not establish that a save modified by a newer build remains compatible.

If the staging save will not load on public, retain the evidence and restore only the test service's original compatible checkpoint. Do not rename save protocol suffixes to bypass Rust's compatibility checks. Record the test conclusion, retire the temporary service through your usual approved process, and leave production unchanged until a separate rollout is planned.

Related help

- Find Rust server.cfg and the correct identity save folder
- Schedule game server backups and check retention
- Fix Rust update and startup failures without wiping

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