Resolve “Circular plugin loading detected!” on Paper

`Circular plugin loading detected!` reports an impossible plugin load order. One plugin must load before another, but the declared relationships eventually point back to the first plugin. The server cannot satisfy every relationship at once. This is different from a single missing dependency or an invalid YAML setting in your normal configuration.

Capture the entire loop

Download the startup log and find the circular-load message. Copy the plugin sequence reported immediately after it, including every return to the starting plugin. Record each plugin's installed version, download source and the exact Paper build. A later failure may be a consequence of this loop, so preserve the first detection rather than only the final shutdown line.

Message formatting and logger names vary across Paper releases and plugin-loader implementations. Current developer documentation also shows a colon-ended form of the circular-loading message. Match the reported loop and versions rather than assuming every build uses the same punctuation or logger name.

Map the relationships

Check the official dependency and compatibility notes for every plugin in the loop. A hard dependency, a soft ordering preference and a declared load-before relationship can all influence startup, but they do not mean the same thing. Paper-plugin metadata and traditional Bukkit plugin metadata also have different rules.

Start with the last changed component. Was a companion plugin added, a dependency updated or an older add-on retained alongside a new major release? Compare the supported combination as a whole. Merely changing filenames does not change the internal dependency relationships, and adding more copies of the same plugin can introduce another failure.

Prepare an isolated test

Preserve a backup, plugin configurations and any external database state needed for recovery. Use a separate test copy where possible, with mail, scheduled jobs and production database writes disabled. Do not remove moderation or protection plugins from a public server and allow players to join just to see whether startup succeeds.

During a clean stop, move only a deliberately chosen suspect JAR outside the active plugin directory on the test copy. Keep its data. Review which dependants will also fail or need to be excluded from that test. Start and compare the new load sequence. Removing a participant can demonstrate the incompatible combination, but it does not establish which author made the original mistake.

Choose a supported resolution

Use compatible releases recommended by the involved authors, or keep the incompatible optional component out of the deployment. Send the complete loop and version list to the relevant maintainer. Do not unpack JARs and rewrite dependency declarations as an ordinary hosting fix; the declared order may reflect real code requirements.

Paper documents a legacy-loading escape hatch on some versions. Do not treat bypassing the check as proof that the plugin combination is safe. This guide deliberately uses supported combinations and maintainer investigation rather than a blanket JVM flag that can hide ordering problems.

Verify and recover

After the planned correction, check that startup completes without a new dependency failure and test the affected features, permissions and saved data. Keep the old compatible state available. If the trial fails, restore the matched plugin/configuration set, accounting for migrations before restoring an older JAR. No world reinstall is needed to investigate a plugin load-order graph.

Related help

- Install plugins on a Paper or Purpur Minecraft server
- Find Minecraft logs and send useful troubleshooting details
- Back up game files before making changes

Official references

Paper circular plugin loading diagnosis. Paper-plugin dependency and circular-load behaviour.

Traditional plugin metadata and dependency declarations.

Revision #1
Created 2026-11-09 10:55:44 UTC by CloudExa Hosting Admin
Updated 2026-11-09 10:55:44 UTC by CloudExa Hosting Admin