Install and troubleshoot Rust Oxide plugins

Rust plugins need a compatible server framework. This guide covers **Oxide for Rust**, whose plugin directory is `oxide/plugins`. The uMod website hosts Oxide plugin documentation, but its separate Agent workflow uses different directories. Carbon and other frameworks also need their own compatibility checks.

Check the framework first

1. Open your server's Console and check startup messages for Oxide. On a working Oxide installation, `oxide.version` reports its version. An unknown command means you should establish whether the framework loaded before uploading plugins.
2. Confirm the plugin supports Rust and your chosen framework. Read its dependencies, current release notes and permissions. A Minecraft `.jar` is not a Rust plugin.
3. If Oxide is absent, ask support about the supported framework option for your existing service. Do not use Reinstall or unpack an unknown server bundle over a working world.

Prepare a recoverable change

Plan a short maintenance window. Stop the server cleanly and create a completed backup, including the world and the entire existing Oxide directory. Download a copy, record the current plugin versions and keep a copy of the file you will replace. Framework installation must match the server's operating environment.

Download the plugin from its author's official listing or release repository. Use the actual source download, not a saved web page with a renamed extension. Keep the author's filename and `.cs` extension. Treat plugins as executable code: reject unexplained binaries, private download mirrors and requests for your panel password.

Add one plugin and verify it

1. Use Files or SFTP to upload the `.cs` file to `oxide/plugins`. Do not put it in the server root or inside an extra archive directory.
2. Start the server and read the compilation and loading messages. Oxide can automatically load changed source files while running, which is another reason to stage changes during maintenance.
3. Check the generated configuration, commonly under `oxide/config`. Follow the developer's instructions for each setting and validate JSON before saving. Keep API credentials private.
4. Grant only the permission nodes that the plugin documents, to the group that needs them. Test as a normal player as well as an administrator: owner privileges can hide a permissions problem.
5. Check the expected behaviour and logs before adding the next plugin. Record the successful version and configuration change.

Handle failures without wiping data

If compilation fails, capture the first error, plugin version and framework version. Check dependencies and compatibility. Stop before changing another component. Preserve the plugin's configuration and `oxide/data` records; removing data to fix a loading error can erase balances or progress.

To roll back, stop the service and restore the previous plugin and its matching configuration or data from your backup as needed. A `.dll` extension is a different installation category and may require a restart; follow its developer's instructions rather than putting it beside source plugins. After a Rust update, use the update checks before blaming every plugin.

Further reading

- uMod: plugin installation
- uMod: getting started
- uMod: API and plugin loading overview

Revision #2
Created 2026-11-08 21:28:53 UTC by CloudExa Hosting Admin
Updated 2026-11-08 21:28:53 UTC by CloudExa Hosting Admin