Magertron™

VS Code Extension
User Guide

Hand an MCP server to your platform team without leaving your editor

Developer Edition

Submitting an MCP Server from VS Code

For developers. You have built an MCP server and want your platform team to govern it. This is how you hand it over without leaving your editor.

> ⚠ **Submitting is not deploying.** This records that your server exists so your platform team can review it. Nothing is deployed, routed or metered until they decide it should be. That gate is the point — it is why they can let you submit at all.

What you need before you start

Two things from your platform administrator:

- **Your platform's URL** — the same address you would use for the admin UI.
- **A service-account token** issued to you or your team.

> ⚠ You may not need the token. If you already use `mcpctl` from a terminal and have run `mcpctl login`, the extension will use that credential and you only need to supply the URL.

---

1. Install the extension

The extension ships as a `.vsix` file — a single package you install directly. It is not on the VS Code Marketplace.

Get the file

Download it from your platform's documentation site:

```
https://magertron.com/vscode
```

> ⚠ **Match the version to your platform.** The extension bundles the `mcpctl` binary it speaks to, and the two are versioned together. A mismatched pair may send a payload your platform does not understand — ask your administrator which version to use if you are unsure.

Install it

From a terminal — the quickest way:

```
code --install-extension magertron-submit-<version>.vsix
```

Or from inside VS Code:

1. Open the **Extensions** view (`Ctrl+Shift+X` / `Cmd+Shift+X`)
2. Click the **⋯** menu at the top of the panel
3. Choose **Install from VSIX…**
4. Select the file you downloaded

> ⚠ **Reload the window** if VS Code asks. The extension will not appear in the command palette until it has.

⚠ What it bundles, and why that matters

The `.vsix` contains the `mcpctl` binary for your platform. You do not install anything else — and it will **not** interfere with an existing `mcpctl` you may already use from a terminal. The extension runs its own copy and never writes to your `mcpctl` configuration.

> ⚠ **Binaries are per-platform.** If you see "*no mcpctl binary bundled for `<your platform>`*", the package you have was built for a different operating system or architecture. Ask your administrator for the right one.

---

2. Point it at your platform

Open **Settings** (`Ctrl+,` / `Cmd+,`) and search for `magertron`.

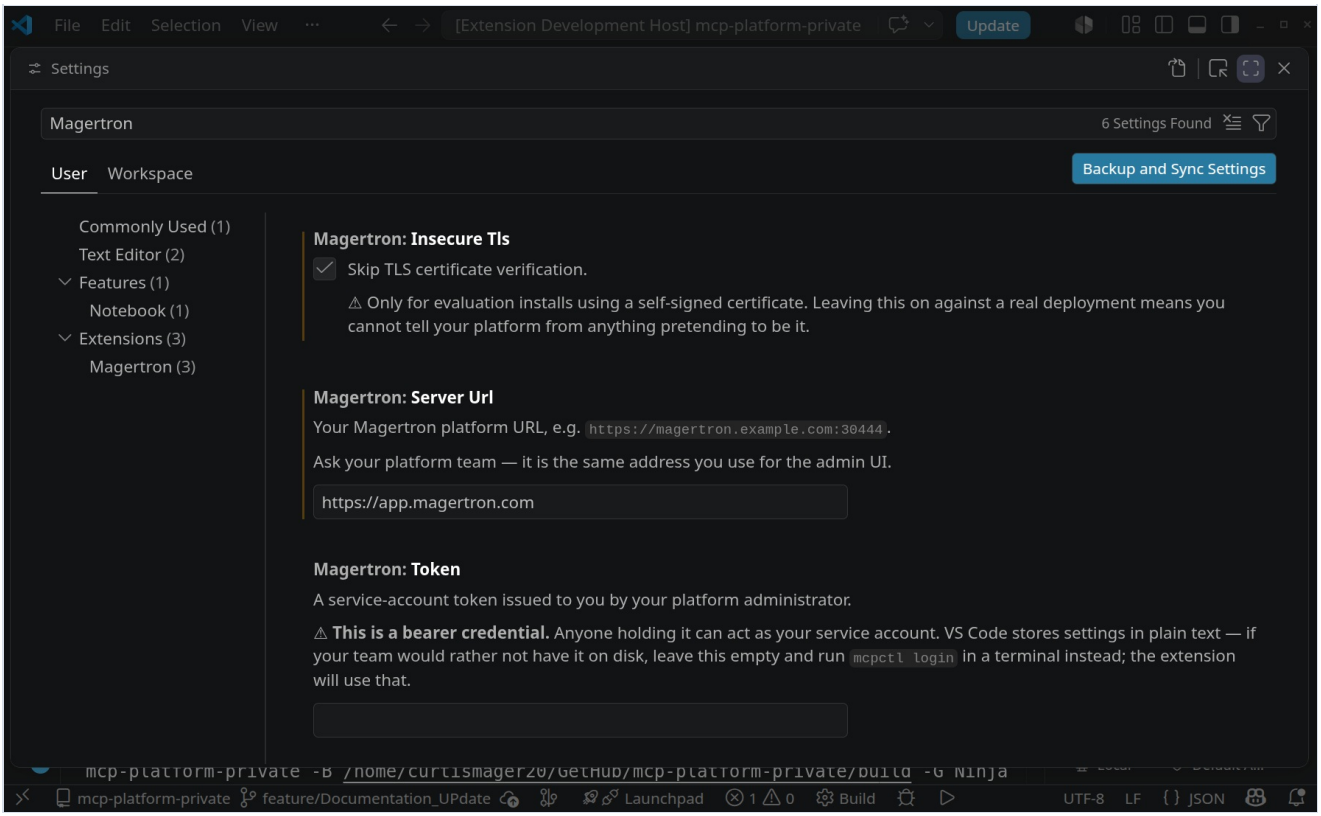

Figure 1 — The three Magertron settings. Server Url is filled in; Token is left empty because this machine already has an mcpctl login.

Setting	What to put
Server Url	Your platform's address, e.g. `https://magertron.example.com`
Token	The service-account token you were issued — **or leave empty**, see below
Insecure Tls	Only for an evaluation install using a self-signed certificate

⚠ On the token

If you already use `mcpctl` in a terminal, leave the Token setting empty. The extension will use the credential from your existing `mcpctl login` and take only the URL from settings. Nothing is duplicated and your credential stays where it already lives.

If you do not, paste the token your administrator gave you.

> ⚠ VS Code stores settings in plain text on disk. A service-account token is a bearer credential — anyone holding it can act as that account. If your team would rather it were not sitting in a settings file, use `mcpctl login` instead and leave this empty.

⚠ On the URL

Use the address exactly as your administrator gave it. In particular, do not add a port to a hostname that does not need one — a platform reached through a tunnel or load balancer answers on the standard HTTPS port, and appending the cluster's NodePort sends the request somewhere that will never answer.

If a submission hangs rather than failing quickly, a wrong port is the first thing to check.

---

3. Check the connection

Before submitting anything, run **Magertron: Show Connection Status** from the command palette (`Ctrl+Shift+P`).

You should see your server address and **Status: connected**.

> ⚠ This only reads. It changes nothing and submits nothing, which makes it the safe way to confirm your settings before you rely on them.

---

4. Submit your server

You need a `.mcpb` **bundle** or a `server.json` **manifest** — whatever your build already produces.

Either:

- Open the file and run **Magertron: Submit MCP Server for Review**, or
- Right-click the file in the Explorer and choose the same command.

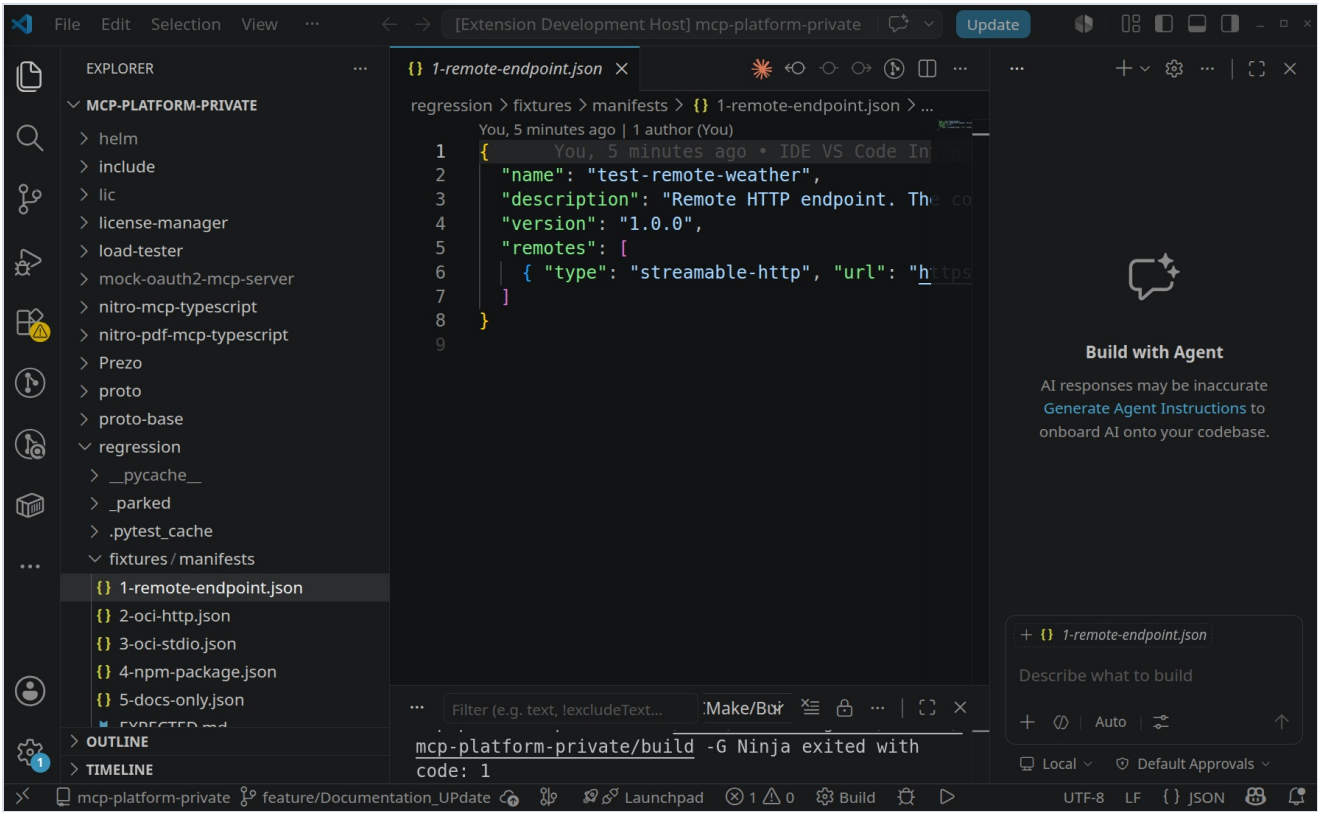

Figure 2 — A manifest open in the editor, ready to submit. This one declares a remote streamable-HTTP endpoint.

> ⚠ **Save first.** The extension submits what is on disk. If the file has unsaved changes it will offer to save before sending — accept, or you will submit a version you cannot see.

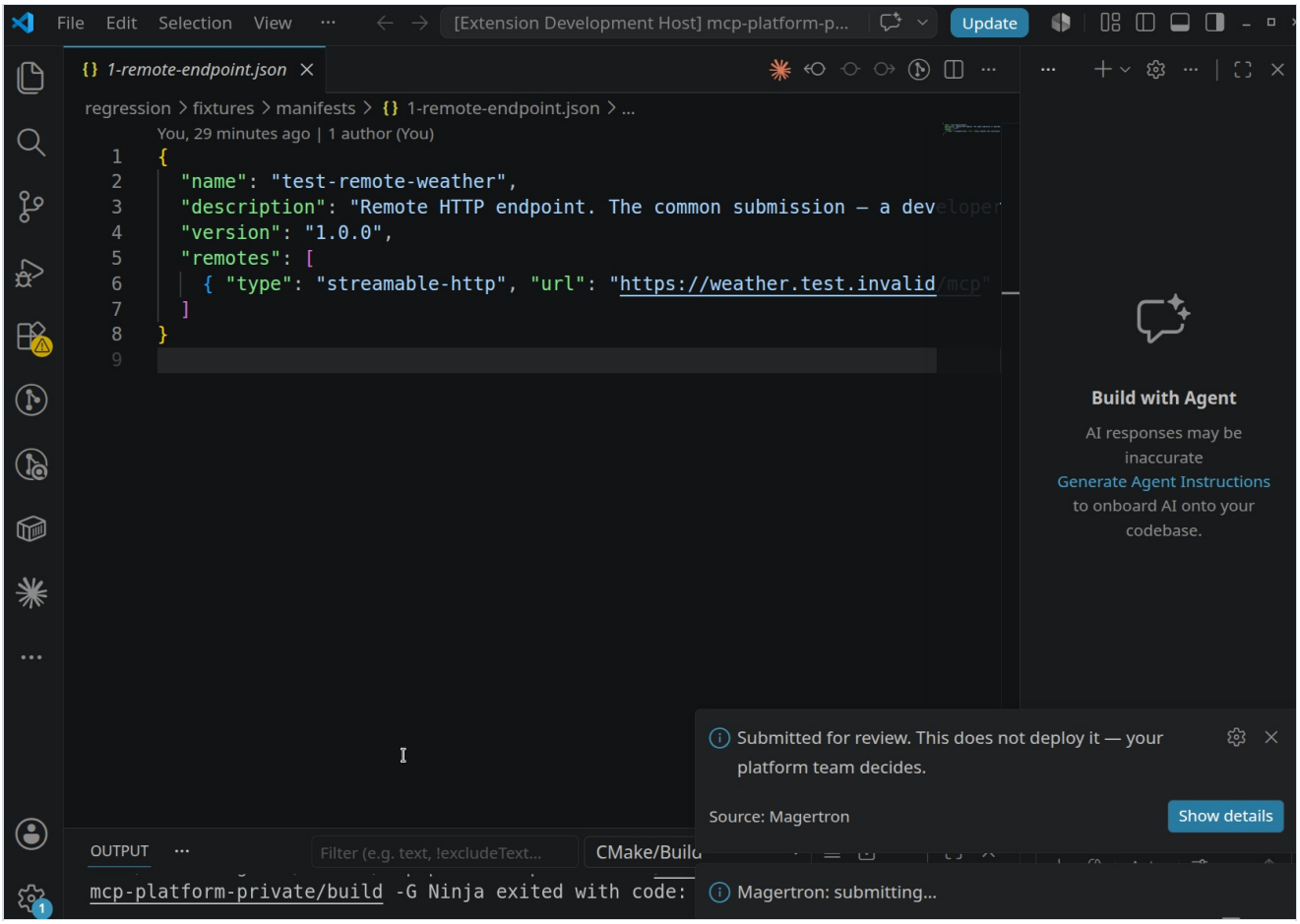

Figure 3 — The confirmation after a successful submission. Note what it says: submitted, not deployed.

5. What was actually sent

Click **Show details** on the notification, or open the **Magertron** output channel, to see exactly what left your machine.

The manifest travels with **anything credential-shaped removed before it leaves your machine**. Specifically:

- Environment variable **names** are sent. **Values are not** — including defaults written into the manifest.
- The same applies to anything named like a password, token, secret or key, wherever it appears in the document.

> ⚠ **This means your platform team can see that your server needs a secret without ever seeing the secret.** They supply the real value themselves when they deploy it. If you have been putting a working key in a manifest default, it does not reach them — and it should not be in the file either.

Also sent, so an administrator knows who to talk to:

Your **service account**	Established by the platform from your token — you cannot change it
Which tool you used	`vscode`
Your **OS username**	Sent as an unverified claim, so somebody knows whom to ask

6. What happens next

Your submission lands in your platform team's review queue.

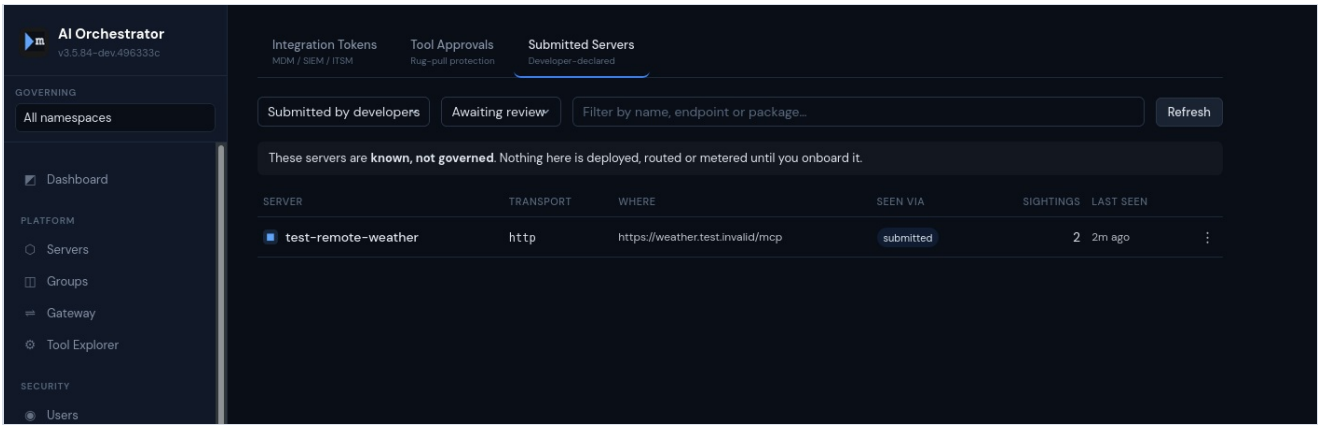

Figure 4 — The same submission as your platform team sees it, under Submitted Servers in the admin console: known, not governed.

> ⚠ **Nothing is running yet.** An administrator reads what you submitted, decides whether it should be governed, and — if so — deploys it themselves. They choose the namespace, and they supply the credentials you did not send.

If they need something changed, they will come back to you. Common reasons a server cannot be deployed as submitted:

- **It speaks stdio.** Magertron proxies servers over the network. Expose it over streamable-HTTP or SSE.
- **It is a language package** (npm, PyPI, NuGet) with no endpoint. There is no container image to run. Publish it as an image, or host it and submit the endpoint instead.

Submitting again

Submit as often as you like. A revision of a server **nobody has acted on yet** updates the existing entry rather than creating a duplicate.

> ⚠ **Once an administrator has taken your server**, a new submission queues behind it as a separate entry rather than changing what they are already working on. They deploy the version they reviewed; your revision waits its turn. This is deliberate — it means nobody deploys something different from what they read.

Troubleshooting

"Set your platform URL before submitting" The Server Url setting is empty. The error offers a button that opens the right settings page.

"Not authorized to submit" Your token is missing, expired, or lacks the scope. Ask your administrator for a new one — or if you use `mcpctl login`, run it again.

A submission hangs rather than failing Almost always a wrong URL — usually a port appended to a hostname that does not use one. Run **Show Connection Status** and check the address.

"Manifest describes neither a remote endpoint nor a package" The file says what your server *is* but not where it is or how to run it. A manifest needs either a remote URL or a package to be actionable.

Nothing happens at all Check the **Magertron** output channel. Every submission logs what it sent and what came back.