Connect Your AI Agent

CompactHive runs a remote **MCP server** (Model Context Protocol) at `https://app.compacthive.io/mcp`. It lets your own AI coding agent — Claude Code, Codex CLI, OpenCode, Cursor, VS Code, or any other MCP-capable client — see and act on your tenant directly, instead of you copy-pasting context back and forth.

This is separate from **Hive Code** (the in-portal browser agent covered earlier in this book). Hive Code is CompactHive's own agent, running server-side in your namespace. The MCP server is for *your own* agent, running wherever you already use it (your laptop, your CI, your editor). You can use either, both, or neither.

Connecting

In the portal, go to **AI Agents** in the sidebar for client snippets, or Settings → **Agent access** to create a token. There are two ways to authenticate:

- **One CompactHive access token for MCP + Git** — generate it once in Settings → Agent access, configure your MCP client, then run the Git credential-helper snippet shown there once. Use this for the full code → push → deploy workflow.
- **OAuth (recommended)** — your agent opens a login/consent screen; no token to copy. Tick **Push code** if you want the agent to create a separate Git-only credential for local clone/push. This checkbox is off by default and separate from **Operate**.

Shortest example (Claude Code):

```
claude mcp add --transport http compacthive https://app.compacthive.io/mcp
# then run /mcp inside Claude Code and choose "Authenticate"
```

Every token and connected app can be reviewed and revoked from Settings → Agent access. Revoking a PAT removes both MCP and Git access (Git caches expire within 3 seconds).

One token: configure once, then clone and push

1. Create a token with **Read + deploy + push code** access in Settings → Agent access. Copy it now: it is shown once.
2. Paste it into your MCP client's local secret/config, never a committed file or agent chat.
3. Save the credential block from the same screen (including its final blank line) to a temporary file outside your repository using a local editor or your agent's FILE-WRITE tool, never shell echo/printf/heredoc. It uses your real tenant Git host and the SAME token. Use file mode 0600 where supported (best effort; Windows ignores it). For illustration only (replace placeholders in the file, not a command):

```
protocol=https
host=<workspace>-forgejo.compacthive.dev
username=compacthive
password=<YOUR_COMPACTHIVE_TOKEN>

```

Configure a helper, then approve from the file. macOS/Linux/Windows Git Bash:

```
git config --global credential.https://<workspace>-forgejo.compacthive.dev.helper store
git credential approve < /path/file
rm /path/file
```

PowerShell has no `<` redirect. After configuring the same helper, use:

```
Get-Content /path/file | git credential approve
Remove-Item /path/file
```

Replace `/path/file` with your temporary file path. Delete it even if approval fails. Never put the token in a command line or shell history.

The `store` helper saves credentials unencrypted on disk. Linux/macOS can use it; macOS users can replace `store` with `osxkeychain` for Keychain. On Windows, Git Credential Manager prompts once; use `manager` instead of `store`. Keep secrets out of shell history and shared terminals. Any non-empty Git username works; we recommend `compacthive`.

4. Ask the agent to call `setup_project` for your repo. It applies immediately and returns the clone URL; wait for CI setup with `get_project_status` before your first push.
5. Clone the unchanged URL (`git clone https://<workspace>-forgejo.compacthive.dev/<owner>/<repo>.git`), edit, commit and push. CI builds and deploys automatically; inspect status and health with MCP.

Read access includes MCP read tools and Git clone/fetch for writable members, but no push. **Read + deploy + push code** adds MCP operate tools and `git:write`. Push-code permission covers every repository accessible to you in this account. Readonly members have no Git access. Existing

tokens do not silently gain push permission: create a new push-code token if needed. For repository-scoped CI, manual Forgejo tokens remain under Settings → Git access (Advanced / CI).

OAuth: let your agent configure Git

On the consent screen, tick **Push code** to allow Git-token creation; tick **Operate** separately if the agent should also set up or change apps.

Before cloning or pushing, your agent should test `git ls-remote <git url>`. If authentication fails, it asks you for a token name (through a client prompt if supported, otherwise chat), calls `create_git_token`, then writes the returned `credential_block` (protocol/host/username/password lines plus a final blank line) to a temporary file outside the repository using its FILE-WRITE tool, never shell echo/printf/heredoc. Use mode 0600 where supported (best effort; Windows ignores it). On macOS/Linux/Windows Git Bash it runs `git credential approve < /path/file`, then `rm /path/file`. PowerShell has no `<` redirect: use `Get-Content /path/file | git credential approve`, then `Remove-Item /path/file`. Delete the file even on failure, and keep the token out of every command line and shell history. It can now clone, work and push. If permission is missing, reconnect MCP and tick Push code, or create a token in Settings → Git access. `list_git_tokens` shows your token metadata, never the secret values.

Prefer an OS credential helper: macOS Keychain (`osxkeychain`), Windows Git Credential Manager (`manager`), or Linux `libsecret`/`cache`. The `store` fallback saves the token **in plaintext** in `~/.git-credentials`; your agent must warn you before using it. Never put a token in a remote URL, `.git/config`, repository files, commit messages or chat.

The Git token is returned once through your agent's model context to configure the helper. Grant Push code only if you accept that exposure. It is Git-only (cannot access MCP), covers repositories accessible to your member, and has no expiry unless one is requested. You can have up to 20 active Git tokens. Disconnecting OAuth does **not** revoke these Git tokens: in Settings → Git access, find the **created by … via MCP** label and choose **Revoke** separately. Tokens created manually are labelled **created in portal**. Member removal revokes member Git access regardless of where a token was created.

Bring your own model credentials

Just like Hive Code, connecting the MCP server does not give your agent model access — you keep using whatever subscription or API key you already pay for (Claude Code's own Anthropic login, Codex's own OpenAI login, your own API keys, etc.). CompactHive only exposes your tenant's apps, services, and deploy pipeline as tools the agent can call; it is not a model provider.

What your agent can do

- **Read-only by default**: list your apps and services, check app health and logs, see build status and recent deploys, check your usage against your tier's limits, and pull the same platform reference documentation (CI setup, Dockerfile conventions, deploy API) that's built into the portal's Developer Guide.
- **Operate (opt-in)**: choose Read + deploy + push code for a PAT, or Operate on the OAuth consent screen, to let the agent restart/redeploy apps, set environment variables and write-only secrets, enable services or set up a project on Forgejo. Calls apply immediately; your agent client's approval settings control when it asks you first. Enabling a paid service starts billing — check the catalog price before calling. OAuth Operate alone does not grant Push code.
- Every tool call is logged in your tenant's audit trail, and you can disconnect an agent at any time.

Limitations

- **GitHub repositories/sessions are not yet supported** over MCP for project scaffolding — that tool works against the built-in Forgejo provider only. If your tenant is connected to GitHub (beta), connect and push manually via Settings → "Set Up" instead; see the **Git Providers** page for the current state of GitHub support.
- There's no delete, no pod exec, no secret *reads* (you can set secrets, not read them back), and no deployment rollback — the same boundaries that apply in the portal UI apply here.
- Model access, as above, is entirely bring-your-own.

Revision #4
Created 2026-11-03 14:21:28 UTC by Docs Sync
Updated 2026-11-05 18:06:13 UTC by Docs Sync