Guides & Reference

Deeper reference material: Git providers, CI/CD, databases, connecting your own AI agent, and other service detail.

- Connect Your AI Agent
- Git Providers
- CI/CD Pipelines
- Deploying Apps — Reference
- Environment Variables & Secrets
- Finding Your Connection Details
- Container Registry
- PostgreSQL Database
- Custom Domains
- Service Catalog Reference
- Using AI Coding Assistants

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.

Git Providers

Forgejo (built-in, default)

Forgejo is the default and recommended Git provider for every new tenant — it's self-hosted inside your namespace, needs zero setup, and is ready to push to the moment your tenant finishes provisioning. It's a lightweight, open-source Git server (a GitHub-style web UI, repos, issues, pull requests) included free at every tier.

For built-in Forgejo, connect your agent with OAuth and opt in to **Push code** to let it configure a Git-only token, or use one CompactHive PAT for MCP and HTTPS Git. See **Connect Your AI Agent** for consent, credential setup and independent revocation of MCP-minted Git tokens. Repository-scoped Forgejo tokens remain an Advanced / CI option in Settings → Git access.

GitHub (beta)

GitHub can be connected instead of, or alongside, Forgejo:

1. Open **Settings → Git Provider** in the portal.
2. Connect through the **CompactHive GitHub App** and grant access to the repositories you want to use.
3. Use **Set Up** on a connected repository to configure its CI pipeline.

Today, GitHub support is **beta** and covers manual repository connection plus CI setup only. Not yet available for GitHub:

- **Automatic shipping** — CompactHive doesn't yet auto-detect and wire up a GitHub repo end-to-end the way it does for a Forgejo repo.
- **MCP project scaffolding** (`setup_project`) — your own AI agent (see **Connect Your AI Agent**) can only scaffold a brand-new project on Forgejo today.
- **Hive Code sessions against a GitHub repo** — the in-portal coding agent works against Forgejo repositories only; for a GitHub-connected tenant, build and push manually and let the normal CI/CD pipeline take over.

These are on the roadmap, gated on demand from early design partners rather than a fixed date.

Switching providers

You can switch your tenant's Git provider in **Settings → Git Provider** at any time. Switching is destructive for the provider you're leaving — for example, moving away from Forgejo deletes your Forgejo repositories and their history, and moving away from GitHub clears the GitHub App connection — so the portal shows a clear warning and asks you to confirm before starting the switch. Your CI pipeline (Woodpecker) is automatically reconnected to the new provider once the switch completes.

CI/CD Pipelines

How CI/CD works on CompactHive

CompactHive uses Woodpecker CI for continuous integration and delivery. It runs inside your Kubernetes namespace using the Kubernetes backend — each pipeline step executes as its own pod.

- **Woodpecker Server** — receives webhooks from your Git provider (Forgejo, or GitHub in beta) and stores pipeline state.
- **Woodpecker Agent** — picks up jobs and creates pipeline pods in your namespace.
- **Kaniko** — builds your container image without a Docker daemon (runs as a regular pod).
- **Your private registry** — stores the images Kaniko builds.

Setting up your pipeline

1. **Registry secrets (automatic).** `registry_url`, `registry_user`, and `registry_password` are auto-injected into Woodpecker during provisioning — no manual setup needed. You can verify they exist in Woodpecker under **Settings → Secrets** (they appear as global secrets).
2. **Pipeline file.** Add a `.woodpecker.yaml` to your repository root. The portal's Developer Guide generates a ready-to-use template, pre-filled with your tenant's registry path.
3. **Push.** Woodpecker picks up the push via webhook, creates a pipeline pod, builds with Kaniko, and pushes to your registry.

Pipeline template

This is the general shape of a CompactHive `.woodpecker.yaml`. Your portal's Developer Guide shows the exact version with your registry project path already filled in.

```
# Registry secrets are auto-injected — no manual setup needed.
steps:
  - name: build
    image: ghcr.io/osscontainertools/kaniko:v1.28.1-debug
    commands:
```

```
      - mkdir -p /kaniko/.docker
      - |
        cat > /kaniko/.docker/config.json << DOCKEREOF
        {
          "auths": {
            "$${REGISTRY_URL}": {
              "username": "$${REGISTRY_USER}",
              "password": "$${REGISTRY_PASSWORD}"
            }
          }
        }
        DOCKEREOF
      - /kaniko/executor
          --context=$CI_WORKSPACE
          --dockerfile=Dockerfile
          --destination=$${REGISTRY_URL}/YOUR_PROJECT/$CI_REPO_NAME:$CI_COMMIT_SHA
          --destination=$${REGISTRY_URL}/YOUR_PROJECT/$CI_REPO_NAME:latest
          --cache=true
          --insecure
    environment:
      REGISTRY_URL: { from_secret: registry_url }
      REGISTRY_USER: { from_secret: registry_user }
      REGISTRY_PASSWORD: { from_secret: registry_password }
```

Notes:

- `YOUR_PROJECT` is your registry project name (shown on the portal Connect page).
- **Two tags per build** — the commit SHA (for rollback) and `:latest` (for convenience).
- `--insecure` is required because, inside the cluster, the registry is reached over plain HTTP (TLS is terminated at the edge, in front of the registry's public hostname).
- `--cache=true` enables Kaniko layer caching for faster rebuilds.

Customization

Only build on your main branch:

```
steps:
  - name: build
    image: ghcr.io/osscontainertools/kaniko:v1.28.1-debug
    when:
      - branch: main
```

Add a test step before the build:

```
steps:
  - name: test
    image: node:20-alpine
    commands:
      - npm ci
      - npm test

  - name: build
    image: ghcr.io/osscontainertools/kaniko:v1.28.1-debug
    depends_on: [test]
```

Useful Woodpecker built-in variables

Variable	Example	Description
`CI_REPO_NAME`	`my-app`	Repository name
`CI_COMMIT_SHA`	`a1b2c3d`	Full commit hash
`CI_COMMIT_BRANCH`	`main`	Branch name
`CI_WORKSPACE`	`/woodpecker/src/...`	Workspace path (source code)
`CI_PIPELINE_EVENT`	`push`	Event type (push, pull_request, tag)

See Woodpecker's own documentation for advanced features (matrix builds, cron, services, volumes) not covered here.

Troubleshooting

Problem	Solution
Pipeline not triggering	Make sure the repo is **activated** in the Woodpecker UI. For GitHub (beta), check webhook delivery in your GitHub repo settings.
Kaniko auth error	Verify the 3 registry secrets exist and are correct — check for typos if you ever set them manually.

Problem	Solution
Build timeout	Lower tiers have less CI CPU/memory headroom. Multi-stage Dockerfiles that shrink the build context usually help.
Image not found on deploy	Check that the image tag on the Apps page matches what the pipeline pushed — the Registry page in the portal shows what's actually there.

Deploying Apps — Reference

This page is the detailed reference for custom apps. For the push-to-deploy overview, see **Deploying Your App** in Getting Started.

Deploy from the portal

1. Go to portal → **Apps**.
2. Fill in the deploy form: **App Name** (lowercase, alphanumeric + hyphens, e.g. `my-api`), **Image** (full registry path, e.g. `registry.compacthive.dev/<project>/my-api:latest`), **Port** (the port your app listens on inside the container).
3. Click **Deploy**.

CompactHive creates a Kubernetes Deployment, Service, and Ingress for your app, reachable at `https://<your-slug>-<app-name>.compacthive.dev`.

Resource	Purpose
Deployment	Runs your container (1 replica by default)
Service	ClusterIP service on your specified port
Ingress	Public URL via the `*.compacthive.dev` wildcard

Resource defaults

Custom apps get a default **request** of ~25m CPU / 64Mi memory if you don't set one. If you don't set an explicit memory/CPU **limit**, the namespace's default limit applies (200m CPU / 256Mi memory) — raise it on the Apps page if your app needs more headroom, within your tier's overall quota.

App limits per tier

Tier	Max apps
Basic	2

Tier	Max apps
Team	5
Pro	10
Business	25
Enterprise	50

Update or delete a running app

On the Apps page, click your app to expand its details. You can update the **image** (triggers a rolling update), **replicas**, and **environment variables**. Deleting an app removes its Deployment, Service, and Ingress — registry images are not deleted (manage those separately from the Registry page).

End-to-end example

This walkthrough takes an empty repo to a running app. The same process works for any language with a Dockerfile — this example uses Node.js.

`server.js`

```
const http = require('http');
const port = process.env.PORT || 3000;

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.end('<h1>Hello from CompactHive!</h1>');
});

server.listen(port, () => console.log(`Listening on :${port}`));
```

`Dockerfile`

```
FROM node:20-alpine
WORKDIR /app
COPY server.js .
EXPOSE 3000
CMD ["node", "server.js"]
```

1. Push `server.js`, `Dockerfile`, and a `.woodpecker.yaml` (see **CI/CD Pipelines**) to your Forgejo repo — or, on GitHub (beta), connect it and run Settings → "Set Up" to add CI.
2. Push. Woodpecker clones the repo, builds the image with Kaniko, and pushes it to `registry.compacthive.dev/<project>/<repo>:latest`.
3. Go to portal → **Apps**, enter the app name, the image above, and port `3000`, then click **Deploy**.
4. Your app is live at `https://<your-slug>-<app-name>.compacthive.dev`. Every subsequent push rebuilds and (with auto-deploy on) redeploys it automatically.

Environment Variables & Secrets

Setting plain environment variables

When deploying a custom app, you can pass environment variables that get injected into your container at runtime.

Via the portal: on the Apps page deploy form, add key-value pairs in the environment variables section. After deploying, update them by editing the app.

Via the API:

```
POST /api/portal/apps
{
  "name": "my-api",
  "image": "registry.compacthive.dev/<project>/my-api:latest",
  "port": 3000,
  "env": {
    "NODE_ENV": "production"
  }
}
```

Plain environment variables set this way are stored in the Kubernetes Deployment spec — fine for non-sensitive config, but don't put API keys or passwords in them.

Secrets (sensitive values)

For API keys, database passwords, and other sensitive values, use the app's **Secrets** tab instead of a plain env var. Secrets are written to your tenant's Vault, injected into the container, and rotated (the previous value stays valid briefly during a rotation so an in-flight request doesn't break) — the

value itself never appears in the Deployment spec or in logs.

Connecting to platform services

CompactHive services run in your namespace and are reachable over Kubernetes' internal DNS — no VPN or external access needed from your own app.

Service	Internal URL pattern
PostgreSQL	`pg-<tenant-id>-rw.<namespace>.svc.cluster.local:5432`
KeyDB cache	`cache-<tenant-id>.<namespace>.svc.cluster.local:6379`
MinIO (S3 API)	`minio-<tenant-id>.<namespace>.svc.cluster.local:9000`

Check the portal **Connect** page for the exact values for your tenant — it also has a pre-filled connection string for each service.

Common patterns

```
# Node.js
PORT=3000
NODE_ENV=production
DATABASE_URL=postgresql://user:pass@pg-<tenant-id>-rw:5432/app

# Python / FastAPI
APP_ENV=production
DATABASE_URL=postgresql://user:pass@pg-<tenant-id>-rw:5432/app
REDIS_URL=redis://cache-<tenant-id>:6379

# Go
GIN_MODE=release
DB_HOST=pg-<tenant-id>-rw
DB_PORT=5432
```

Finding Your Connection Details

Finding your credentials

Go to portal → **Connect**. This page shows connection details for every deployed service, including endpoints, usernames, and passwords.

- Passwords are masked by default — click to reveal.
- Copy buttons next to each field for quick clipboard access.
- Only deployed (enabled) services appear.

Service endpoints

Public URLs

Services with a web interface are reachable at `https://<your-slug>-<service>.compacthive.dev`:

Service	URL pattern
Woodpecker CI	`<your-slug>-ci.compacthive.dev`
Grafana (part of Monitoring)	`<your-slug>-grafana.compacthive.dev`
Forgejo	`<your-slug>-forgejo.compacthive.dev`
MinIO Console	`<your-slug>-minio.compacthive.dev`
Uptime Kuma	`<your-slug>-status.compacthive.dev`
FileBrowser	`<your-slug>-files.compacthive.dev`
Custom apps	`<your-slug>-<app-name>.compacthive.dev`

Internal-only services

These don't have a public URL — they're only reachable from within your namespace, or via the Headscale VPN add-on (Business tier and up):

Service	Internal address	Port
PostgreSQL	`pg-<tenant-id>-rw.<namespace>.svc`	5432
KeyDB cache	`cache-<tenant-id>.<namespace>.svc`	6379
MinIO (S3 API)	`minio-<tenant-id>.<namespace>.svc`	9000

Credential lifecycle

- Credentials are generated once during initial deployment and stored as Kubernetes Secrets.
- They survive service restarts and redeployments.
- They are deleted only when the tenant namespace (or the specific service) is removed.
- There's no self-service rotation UI today — if you need to rotate a service credential, disable and re-enable the service.

Container Registry

Overview

Every CompactHive tenant gets a private container registry backed by Harbor. It stores the images your CI pipeline builds. Your Kubernetes namespace is pre-configured with pull credentials, so deployed apps can pull images automatically.

Property	Value
Public URL	`registry.compacthive.dev`
Protocol	Docker Registry v2 (OCI compatible)
Auth	Robot account per tenant (auto-generated)
TLS	HTTPS from your machine; plain HTTP internally inside the cluster

Storage quotas

Tier	Storage
Basic	1 GB
Team	5 GB
Pro	10 GB
Business	25 GB
Enterprise	100 GB

Find your credentials

Go to portal → **Connect**. Under "Container Registry" you'll see the **Registry URL**, your **Project Name** (your isolated Harbor project), **Username** (robot account), and **Password** (click to reveal).

Push an image manually

```
docker login registry.compacthive.dev -u YOUR_USERNAME -p YOUR_PASSWORD
docker tag my-app:latest registry.compacthive.dev/YOUR_PROJECT/my-app:latest
docker push registry.compacthive.dev/YOUR_PROJECT/my-app:latest
```

Pull in Kubernetes

Your namespace has an `imagePullSecret` already configured on the `default` ServiceAccount. Any Deployment using the default ServiceAccount pulls from your registry automatically — no extra configuration needed.

View registry contents

Go to portal → **Registry** to see your repositories, tags, and storage usage.

PostgreSQL Database

Overview

CompactHive provides a managed PostgreSQL database via the CloudNativePG (CNPG) operator. Each tenant that enables the Database service gets an isolated PostgreSQL instance running in its own namespace.

Connection details

Go to portal → **Connect**. Under "PostgreSQL" you'll find:

Field	Value
Host	`pg-<tenant-id>-rw.<namespace>.svc.cluster.local`
Port	`5432`
Database	Your tenant slug
Username	Same as the database name
Password	Click to reveal on the Connect page

Note: the database and username are set to your tenant slug during provisioning, not a generic name like "app". Check the Connect page or the portal's Developer Guide for your exact values.

Connection string

```
postgresql://SLUG:PASSWORD@pg-<tenant-id>-rw.<namespace>.svc.cluster.local:5432/SLUG
```

Use this as your `DATABASE_URL` environment variable when deploying apps. The exact pre-filled connection string is on the portal → **Developer Guide** page.

From your app

Since your app runs in the same Kubernetes namespace, it reaches PostgreSQL via the internal service DNS name — no VPN or external access is needed.

```
// Node.js (pg)
const { Pool } = require('pg');
const pool = new Pool({ connectionString: process.env.DATABASE_URL });
```

```
# Python (psycopg2)
import psycopg2
conn = psycopg2.connect(os.environ['DATABASE_URL'])
```

```
// Go (pgx)
conn, err := pgx.Connect(ctx, os.Getenv("DATABASE_URL"))
```

Version

PostgreSQL 17 (the latest CNPG-supported version at time of writing).

Backups

CNPG runs a daily base backup plus continuous WAL archiving to the platform's backup object store automatically — no setup needed on your side. If you need a one-off export, run `pg_dump` from within your namespace.

Limitations

- Single instance (no HA read replicas) on every tier today.
- Storage is limited by your database's own storage allocation, billed separately per Gi — see the pricing page.
- No external access by default — connect from within your namespace only, or over the Headscale VPN add-on (Business tier and up).
- Superuser access is not provided — your role has full DDL/DML on your own database.

Custom Domains

Default URLs

Every service and app gets a URL under `*.compacthive.dev`:

```
https://<your-slug>-<service>.compacthive.dev
```

Examples: `https://acme-ci.compacthive.dev` (Woodpecker CI), `https://acme-grafana.compacthive.dev` (Grafana), `https://acme-my-api.compacthive.dev` (a custom app). These work immediately after deployment; TLS is handled for you.

Adding a custom domain

Custom domains are available on **every tier**, with a per-tier limit on how many you can register:

Tier	Custom domains
Basic	3
Team	10
Pro	25
Business	50
Enterprise	Effectively unlimited

1. Go to portal → **Domains**.
2. Enter your hostname, pick the service or app it should point to, and click **Add Domain**.
3. At your DNS provider, add a **TXT record** named `_compacthive-challenge.<host>` with the exact value shown in the domain's verification card. For example, for `app.example.com`, the full record name is `_compacthive-challenge.app.example.com`. Some providers append your zone name automatically; check that it is not duplicated. Use the copy buttons for the name and value.
4. Add an **A record** for your hostname pointing at the IP address shown in the DNS instructions. **AAAA/IPv6 is not supported yet — use an A record only.** Use an A record, **not a CNAME** — a CNAME to `compacthive.dev` does not work for a domain you own (it is a Cloudflare-proxied name in CompactHive's account, so foreign hostnames are rejected at Cloudflare's edge).

5. Click **Verify now** after DNS propagates, or wait for an automatic check. CompactHive checks the TXT record to verify ownership, then checks that the A record points at its ingress.
6. Once verified and pointing at CompactHive, a **Let's Encrypt certificate is issued automatically**, typically within minutes. When the status is **Active**, HTTP redirects to HTTPS. Certificates renew automatically; keep your DNS pointing at CompactHive.

A hostname can only be claimed by one account. CompactHive-owned names cannot be used as your own custom domains. Unverified domains are removed after **7 days**; add the domain again if you need to restart verification. Until ownership is verified, the domain does not route traffic to your app.

Domain status

Status	Meaning	What to do
Pending verification	Ownership has not been confirmed; traffic is not routed yet.	Add the TXT record exactly as shown, wait for DNS propagation, then click **Verify now** or wait for an automatic check. Check the removal countdown.
Verified	Ownership is confirmed; CompactHive is waiting for your A record.	Point the A record at the IP shown in the portal and wait for the DNS check.
Issuing certificate	The hostname is ready and a certificate is being requested.	Wait a few minutes. If an error is shown, check DNS and your proxy settings. Automatic retries may take longer.
Active	The certificate is installed and HTTPS is ready.	Open your domain. Keep DNS pointing at CompactHive so automatic renewal can succeed.
Error	Certificate issuance requires attention.	Read the error, correct DNS or proxy settings, then click **Retry**. Contact support if it persists.

If an existing domain shows a verification deadline banner, add the TXT record and verify ownership by that date or the domain stops working.

Cloudflare and other proxies

For Cloudflare-proxied customer records, use **DNS-only (grey cloud)** during issuance, or configure the proxy to pass `/.well-known/acme-challenge` requests through to CompactHive. Let's Encrypt needs to reach that path over HTTP to issue and renew certificates. Keep the challenge path reachable for automatic renewal too.

Service Catalog Reference

This page covers how the service catalog works. For the plain tier-by-tier table of what's included, see **Included Services** in Getting Started.

Pricing model

Each tier (Basic / Team / Pro / Business / Enterprise) is a flat monthly fee. Every service available at your tier is included in that fee — toggling a service off frees its compute for other workloads but does not reduce your monthly bill. See the pricing page for current numbers.

Toggling services

Go to portal → **Services**. Each service has an on/off toggle:

- **Enable** — shows a confirmation dialog, then deploys the service to your namespace.
- **Disable** — shows a confirmation dialog, then removes the service from your namespace.

Services above your tier level are shown greyed out with a tier badge (e.g. "Requires Team"). Upgrade your plan to unlock them.

Platform-provided defaults (always available, not in the catalog)

These aren't toggleable services — they're part of every tenant automatically:

- **Tenant SSO** — a shared Authentik instance handles login for your team members.
- **Tenant Secrets** — a shared HashiCorp Vault holds service credentials and app secrets, isolated per tenant.
- **Backups** — nightly Velero backups, 7-day retention on every tier, extended to 30 days on Business and up.
- **Email** — bring your own outbound email provider (Resend or Postmark are good fits). CompactHive doesn't run a shared outbound SMTP relay.

Using AI Coding Assistants

If you use an AI coding assistant — Claude Code, GitHub Copilot, Cursor, or similar — this page gives it static context about your CompactHive environment via a `CLAUDE.md` / `AGENTS.md`-style file in your repo.

Prefer a live connection instead of a static file? See **Connect Your AI Agent** for the MCP server — it lets a supported agent query your actual tenant (apps, services, build status, logs) instead of you keeping a file like this up to date by hand. Both approaches work; use whichever fits your workflow.

Template

For the live MCP + Git workflow, use one CompactHive token from Settings → Agent access, with Read + deploy + push code. Follow **Connect Your AI Agent** for local credential setup; never paste the token into chat or an instructions file.

Create a `CLAUDE.md` (or `AGENTS.md`) in your project root:

```
# Project Infrastructure (CompactHive)

## Environment
- Platform: CompactHive (self-hosted managed infra)
- Tier: Basic / Team / Pro / Business / Enterprise
- Slug: YOUR_SLUG

## URLs
- Portal: https://app.compacthive.io
- CI/CD: https://YOUR_SLUG-ci.compacthive.dev
- App URL pattern: https://YOUR_SLUG-APP_NAME.compacthive.dev
- Registry: registry.compacthive.dev

## Git provider
- Forgejo (built-in, default) or GitHub (beta — manual connect + Settings "Set Up" for CI).

## CI/CD
- Engine: Woodpecker CI (Kubernetes backend)
```

```
- Pipeline file: .woodpecker.yaml
- Build: Kaniko (daemonless, no Docker daemon)
- Registry auth: auto-injected Woodpecker secrets (registry_url, registry_user,
registry_password)

## Database
- Engine: PostgreSQL 17 (CNPG operator)
- Host: pg-<tenant-id>-rw.NAMESPACE.svc.cluster.local, port 5432
- Connection string: see portal Connect or Developer Guide page

## Deploying apps
1. Push to Forgejo (or a connected GitHub repo, beta) — Woodpecker + Kaniko builds the image.
2. Image goes to registry.compacthive.dev/PROJECT/REPO:TAG.
3. Deploy via the portal Apps page or POST /api/portal/apps.
4. URL: https://YOUR_SLUG-APP_NAME.compacthive.dev

## Key constraints
- Kubernetes namespace isolation — no cluster-admin access.
- ResourceQuota limits CPU/memory/storage per tier; app count is capped per tier too.
- No Docker daemon in CI — use Kaniko, not `docker build`.
- TLS is terminated at the edge; apps receive plain HTTP on their configured port.

## Documentation
- https://docs.compacthive.io
```

Giving an assistant task-specific context

Task	What to tell the assistant
Write a Dockerfile	Target amd64/arm64, keep the image small, `EXPOSE` the right port
Write a CI pipeline	Engine Woodpecker, builder Kaniko (`--insecure`, registry secrets via `from_secret`) — see **CI/CD Pipelines**
Connect to the database	Host `pg-<tenant-id>-rw`, standard PostgreSQL driver, connection string from the Connect page
Debug a deployment	Check the portal Resources page for pod status, and the Logs viewer for app logs

Task	What to tell the assistant
Set up monitoring	Grafana at `YOUR_SLUG-grafana.compacthive.dev` (Monitoring service); Prometheus scrapes your pods automatically once Monitoring is enabled