Coolify in Production

Why Nashua endorses Coolify, and how we build it: a hardened Ubuntu LTS host, the Coolify installer, CrowdSec with Traefik AppSec, Docker and Kubernetes, and a Claude Code and GitHub pipeline for CI/CD.

Abstract

Coolify is an open-source, self-hosted platform that turns a plain Linux server into a deployment platform with the convenience of a managed cloud and none of the lock-in. At Nashua we endorse it as the best-of-breed way to self-host, for the organisations that have reason to run their own platform, and this is the practitioner paper behind that endorsement: a complete, opinionated build. We start from a hardened Ubuntu LTS host, install Coolify, and put a real security edge in front of it with CrowdSec and the Traefik AppSec web application firewall. We show how workloads actually run, Docker first, because that is what Coolify orchestrates, and Kubernetes honestly, because that is a different decision. And we wire up the development loop that makes the platform worth having: Claude Code for building, GitHub for review and continuous integration, and Coolify for continuous deployment, so that a commit becomes a tested, running service without a human touching a server. It is a sovereign home for your own software when that is the right choice, and cloud and on-premises are both good homes; Nashua helps you pick, and then builds, secures and runs it.

Keywords: Coolify; self-hosted PaaS; Ubuntu LTS; server hardening; CrowdSec; Traefik; AppSec; web application firewall; Docker; Docker Swarm; Kubernetes; Claude Code; GitHub; CI/CD; continuous deployment; vibe coding; digital sovereignty; DevOps; infrastructure as a service

1. Introduction

A development team today faces a choice that used to be simple. You can rent a managed platform, Heroku, Vercel, Render and their kind, and get a deploy on every push in exchange for a per-seat, per-service bill that compounds and a dependency you do not control. Or you can run your own servers and accept that you are now responsible for the undifferentiated heavy lifting that the managed platforms hide: provisioning, reverse proxies, TLS certificates, build pipelines, rollbacks and the rest. For years the first option won by default, because the second was genuinely too much work.

That calculus has changed, and the thing that changed it is Coolify. Coolify is an open-source, self-hosted platform that sits on top of Docker and gives a plain Linux server the ergonomics of a managed cloud: connect a Git repository, and it builds your application, runs it in a container, puts it behind a reverse proxy, issues and renews its TLS certificate, and redeploys it on every push. It does for your own software what Nextcloud does for your files and collaboration: it lets you own the platform when you would rather not rent it. That is why Nashua endorses it, for the organisations that choose to run their own, and this paper is the build behind the endorsement.

This is a practitioner paper, not a survey. It is an opinionated, end-to-end recipe for standing up Coolify the way we would run it for a client, and it is deliberately concrete. We harden an Ubuntu LTS host (Section 3), install Coolify and secure its control plane (Section 4), and put a real security edge in front of it with CrowdSec and the Traefik AppSec web application firewall (Section 5). We are clear-eyed about workloads: Docker is what Coolify orchestrates and what most teams should use (Section 6), and Kubernetes is a separate, heavier decision we treat honestly rather than as a checkbox (Section 7). Finally we wire up the loop that makes the whole thing pay off: Claude Code, GitHub and continuous deployment (Section 8), before a word on operating it (Section 9). Figure 1 shows the shape of the result.

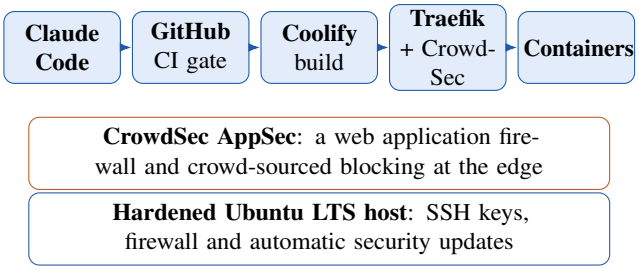

Figure 1. The Nashua Coolify stack. A commit flows from Claude Code through a GitHub continuous-integration gate to Coolify, which builds it and runs it as a container behind Traefik; CrowdSec and the Traefik AppSec firewall guard the edge, and the whole platform sits on a hardened Ubuntu LTS host.

2. Why Nashua endorses Coolify

Nashua is not dogmatic about where software runs. A managed cloud platform and a self-hosted one are both legitimate, valuable choices, and each is the right answer for different organisations: the managed cloud buys convenience, elasticity and an operational burden someone else carries, while running your own buys control, data sovereignty and a cost that does not scale with a vendor's pricing. We endorse best-of-breed tools and sound practice, not a single destination, and much of our work is helping an organisation weigh that choice honestly against its needs, its regulatory exposure and its appetite for operations. What we do hold is narrower: when an organisation has reason to

run its own platform, for sovereignty, for cost control, for regulatory demonstrability or simply to own its roadmap, it should not have to surrender the ergonomics of the managed cloud to do it, and that surrender used to be the price of self-hosting.

Coolify is what removes that price. It is open source, licensed so that you can read it, run it and extend it. It runs on hardware you choose, in a country you choose, under a law you choose, which makes it a sovereign platform in exactly the sense that matters for the GDPR and NIS2. Its cost is the cost of a server and the work to run it well, not a per-seat annuity that grows with your success. And it is not a toy: it manages builds, deployments, databases, TLS, backups and previews across one server or many, with a web interface a team can actually use. For an organisation that has chosen to keep its data under its own control, keeping the platform there too is a natural next step rather than a leap. That is why, for the self-hosted case, Coolify is the platform we endorse, and why we are willing to put our name on a build guide for it.

3. The host: a hardened Ubuntu LTS server

Everything starts with the server, and the server starts with a long-term-support operating system. We build on Ubuntu LTS because it is the platform Coolify, Docker and the wider ecosystem target first and support longest. At the time of writing we deploy on Ubuntu 24.04 LTS for production: it is proven, and it is supported with security updates into 2029. Ubuntu 26.04 LTS is available for new builds that want the longer horizon; we let a point release settle before we put an LTS under a client's production workload. Two cores, 2 GB of RAM and 10 GB of disk are the documented floor for Coolify; we size up from there, because builds are hungry and headroom is cheap.

A fresh server is not a secure server. Before Coolify touches it, we bring it fully up to date and install the tools that keep it current and guarded:

```
sudo apt update && sudo apt full-upgrade -y
sudo apt install -y ufw fail2ban \
  unattended-upgrades
```

We then create an unprivileged administrative user and stop using `root` directly:

```
sudo adduser deploy
sudo usermod -aG sudo deploy
```

3.1. Lock down SSH

The single most valuable hardening step is to make remote login depend on a key, not a password, and to deny the `root` account a remote shell entirely. Copy your public key to the new user with `ssh-copy-id deploy@your-server`, confirm you can log in with it, and then write a drop-in that forecloses the weaker options:

```
# /etc/ssh/sshd_config.d/99-hardening.conf
PermitRootLogin no
PasswordAuthentication no
KbdInteractiveAuthentication no
PubkeyAuthentication yes
```

Reload the daemon with `sudo systemctl restart ssh`, keeping your current session open until a second session proves the key works. With passwords off, the bulk of automated SSH attacks simply stop mattering, and `fail2ban` handles the noise that remains.

3.2. A default-deny firewall

The host should accept only the traffic it must. We use the uncomplicated firewall to deny everything inbound by default and then open the three doors the platform needs: SSH for us, and HTTP and HTTPS for the applications Coolify will serve.

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow OpenSSH
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable
```

Notice what we do *not* open: port 8000, where the Coolify dashboard listens. The control plane has no business being on the public internet, and Section 4 explains how we reach it without exposing it.

3.3. Patch while you sleep

A server that is current is a server that is not vulnerable to last month's exploit. We let Ubuntu apply security updates automatically and unattended:

```
sudo dpkg-reconfigure \
  -plow unattended-upgrades
```

This, a key-only SSH door, a default-deny firewall and `fail2ban` are the baseline. They are not exotic, and that is the point: most breaches exploit the absence of the ordinary, not the presence of the clever. For regulated clients we go further, to a recognised benchmark such as the CIS Ubuntu baseline, but the steps above are the ones no server should launch without.

4. Installing Coolify

With a hardened host in place, Coolify itself is almost anticlimactic. One command, run as a privileged user, installs Docker if it is missing, pulls the Coolify services and starts them:

```
curl -fsSL \
  https://cdn.coollabs.io/coolify/install.sh \
  | sudo bash
```

When it finishes, the dashboard is listening on port 8000, and Coolify has also started its own reverse proxy. By default that proxy is Traefik, which matters a great deal for the next section, because it is the Traefik instance we will arm with a firewall.

4.1. Secure the control plane first

Before anything else, reach the dashboard and create the administrator account, immediately, so that nobody else does. Because we did not open port 8000 on the firewall, we reach it over an SSH tunnel from our own machine rather than across the internet:

```
ssh -L 8000:localhost:8000 deploy@your-server
```

Now `http://localhost:8000` in your browser is the dashboard, travelling entirely inside the encrypted SSH session. Create the admin user, then give the dashboard

its own domain and TLS certificate through Coolify's own settings, so that ongoing administration uses a real HTTPS URL protected by the proxy rather than a tunnel. The principle is simple and it is the one most self-hosters get wrong: the thing that can deploy all your other things is the most sensitive surface you run, so it is the last thing you expose and the first thing you protect.

5. The security edge: CrowdSec and Traefik AppSec

A platform that deploys public web applications needs a security edge, something between the open internet and your containers that inspects requests, blocks the obviously hostile, and learns. Because Coolify runs Traefik as its proxy, we can add exactly that without introducing another moving part, using CrowdSec.

CrowdSec is an open-source security engine with two halves that matter here. It reads logs and decides which sources are behaving maliciously, and it shares those verdicts across a global community, so your server benefits from attacks seen on everyone else's. And it ships an AppSec component: a web application firewall, built on the Coraza engine and able to run the OWASP Core Rule Set, that inspects the content of requests and blocks exploit attempts inline, before they reach your application. Install the agent, then add the firewall rule collections:

```
curl -s https://install.crowdsec.net | sudo sh
sudo apt install -y crowdsec
sudo cscli collections install \
  crowdsecurity/appsec-virtual-patching
sudo cscli collections install \
  crowdsecurity/appsec-generic-rules
```

Turn on the AppSec listener by adding an acquisition file that tells CrowdSec to serve the firewall on port 7422:

```
# /etc/crowdsec/acquis.d/appsec.yaml
listen_addr: 0.0.0.0:7422
appsec_config: crowdsecurity/appsec-default
name: appsec
source: appsec
labels:
  type: appsec
```

Then mint an API key that Traefik will use to consult CrowdSec, and restart the agent:

```
sudo cscli bouncers add traefik-bouncer
sudo systemctl restart crowdsec
```

5.1. Arming Traefik

The connection between Traefik and CrowdSec is a community plugin, the CrowdSec Bouncer Traefik Plugin, which is the maintained path to the AppSec firewall. It loads as an experimental plugin in Traefik's static configuration:

```
# traefik static config
experimental:
  plugins:
    bouncer:
      moduleName: github.com/maxlerebourg/crowdsec-
    ↪ bouncer-traefik-plugin
      version: v1.8.2
```

and it becomes a middleware in the dynamic configuration, pointed at both halves of CrowdSec: the decision API on port 8080 and the AppSec firewall on port 7422, with the key you just minted. Setting `crowdsecAppsecEnabled` to `true` is what turns the web application firewall on:

```
# traefik dynamic config
http:
  middlewares:
    crowdsec:
      plugin:
        bouncer:
          enabled: true
          crowdsecMode: live
          crowdsecAppsecEnabled: true
          crowdsecAppsecHost: 172.17.0.1:7422
          crowdsecLapiHost: 172.17.0.1:8080
          crowdsecLapiKey: YOUR_BOUNCER_KEY
```

The host address `172.17.0.1` is the Docker bridge gateway, how a container reaches the CrowdSec agent running on the host; adjust it if your network differs. Because Coolify manages Traefik, these snippets go into the proxy configuration that Coolify exposes rather than a hand-run Traefik, and you then attach the `crowdsec` middleware to a service through its Traefik labels in the Coolify dashboard. This is an advanced step, and it is one we most often set up and then operate for clients, but the result is worth it: every public request now passes through a crowd-informed reputation check and a real web application firewall before it reaches a line of your code.

6. Running workloads with Docker

Coolify is, at its core, a friendly face over Docker, and Docker is how the overwhelming majority of workloads should run on it. You rarely write container plumbing by hand. You point Coolify at a Git repository, and it detects how to build the project (through Nixpacks, or your own Dockerfile if you provide one), produces an image, and runs it as a container that Traefik routes to with automatic TLS. Databases, caches and other backing services, PostgreSQL, Redis, MySQL and more, are one-click resources that Coolify provisions and connects. Environment variables, persistent volumes, health checks and scheduled tasks are all dashboard concerns rather than YAML archaeology.

When one server is not enough, Coolify scales out to several using Docker Swarm, Docker's own clustering, so the same model, containers behind a proxy, stretches across a handful of nodes without a change in how you think about it. For the great majority of teams, a single well-specified host or a small Swarm is not a compromise on the way to something serious; it is the right answer, and it is a great deal less to operate than the alternative.

7. When Kubernetes, and when not

Honesty matters more than hype here, so we are plain about it. Coolify orchestrates Docker and Docker Swarm; it does not, in its stable product, natively orchestrate Kubernetes. Native Kubernetes support has been discussed and prototyped, but at the time of writing it is not something we would stake a client's production platform on inside Coolify.

That is not the limitation it first appears, because most teams do not need Kubernetes. Kubernetes earns its considerable operational cost when you genuinely require multi-node scheduling, fine-grained autoscaling, and the vast eco-

system of operators and controllers that assume it, typically at a scale and with a platform team to match. Below that threshold it is a tax paid in complexity for capacity you do not use.

So our guidance is a decision, not a default. Run your workloads on Coolify with Docker, and Swarm when you outgrow one node; that covers most organisations completely. If and when you truly need Kubernetes, run it as what it is, a separate, dedicated cluster, k3s where the footprint must stay small or a full distribution where it must not, provisioned and operated on its own terms rather than bolted onto the convenience platform. The good news is that the security pattern carries over unchanged: Kubernetes ingress commonly runs on Traefik too, and the same CrowdSec AppSec firewall guards it. You do not relearn security to change orchestrators; you relearn operations, which is exactly the cost to weigh.

8. The development loop: Claude Code, GitHub and CI/CD

A deployment platform is only worth having if getting code onto it is effortless and safe. This is where Coolify stops being a hosting tool and becomes the last link in a chain that starts at the keyboard, and it is the part of the build that changes how a team works day to day.

The loop runs like this. You build with Claude Code, describing what you want and reviewing, running and refining what it writes, the practice that has come to be called vibe coding. The hard part of that practice, as we have argued elsewhere, is not generating code but verifying it, so the loop is built around verification. Claude Code commits and pushes to GitHub. The push opens a pull request, and GitHub runs continuous integration against it: your tests, your linter, your type checker and your build, as a gate that a change must pass before it is allowed to merge. A minimal gate is a few lines:

```
# .github/workflows/ci.yml
name: ci
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 22
      - run: npm ci
      - run: npm test
```

Require that check to pass on your protected branch, and no unverified change can reach production. Now Coolify closes the loop. You install the Coolify GitHub App once, under the dashboard's sources, and enable automatic deployment for the project. From then on, a merge to your production branch fires a webhook, and Coolify builds the new commit and deploys it, with no human touching a server. Pull requests can get their own preview deployments, a live URL per branch, so a change is reviewed running rather than merely read. Secrets live in Coolify's environment variables, not in the repository, and a bad release is one click to roll back to the previous image.

The result is the best-of-breed chain we build our own products on, assembled from parts that each do one thing well: Claude Code generates, GitHub verifies and gates, Coolify deploys, and CrowdSec guards. No single vendor owns the whole of it, every link is open or replaceable, and the thing it produces is a path from an idea to a tested, running, defended service that a small team can operate with confidence.

9. Operating it

A platform is not finished when it is built; it is finished when it is being run well. Coolify keeps itself and its deployments updatable from the dashboard, takes scheduled backups of its databases to S3-compatible storage, sends notifications to email, Slack or the chat tool you already watch, and shows the resource use of the server and its containers. The hardened host underneath still needs its patches applied, its firewall reviewed and its CrowdSec decisions watched. None of this is heavy, but all of it is necessary, and it is precisely the ongoing responsibility that a managed service exists to carry. It is the part of the Coolify story that Nashua most often owns for a client: we build the platform, secure its edge, wire up the pipeline, and then stay to run it.

10. Conclusion

Running your own platform used to mean choosing between convenience and control. Coolify ends that trade-off: it gives a server you own the ergonomics of a cloud you do not. Put it on a hardened Ubuntu LTS host, guard it with CrowdSec and the Traefik AppSec firewall, run your workloads on Docker with a clear head about when Kubernetes is worth its cost, and wire it to Claude Code and GitHub, and you have a sovereign, secure, automated path from a commit to a running service. Cloud and on-premises are both good homes for software, and the right one depends on the organisation, its regulation and its appetite for operations; when the answer is to run your own, Coolify is the platform Nashua endorses, builds, secures and operates. Talk to us about which fits you.

References

[1] Coolify (2026). *Coolify documentation: installation and core concepts*. https://coolify.io/docs

[2] coolLabs (2026). *Coolify installation script*. https://cdn.coollabs.io/coolify/install.sh

[3] CrowdSec (2026). *Application Security (AppSec) WAF QuickStart for Traefik*. https://docs.crowdsec.net/docs/appsec/quickstart/traefik

[4] Lerebourg, M. (2026). *CrowdSec Bouncer Traefik Plugin*. https://github.com/maxlerebourg/crowdsec-bouncer-traefik-plugin

[5] Traefik Labs (2026). *Traefik Proxy documentation*. https://doc.traefik.io/traefik/

[6] OWASP Foundation (2026). *OWASP Coraza Web Application Firewall*. https://coraza.io/

[7] OWASP Foundation (2026). *OWASP CRS (Core Rule Set)*. https://coreruleset.org/

[8] Canonical (2026). *Ubuntu 24.04 LTS (Noble Numbat) release notes*. https://releases.ubuntu.com/24.04/

[9] Center for Internet Security (2026). *CIS Ubuntu Linux Benchmark*. https://www.cisecurity.org/benchmark/ubuntu_linux

[10] Docker, Inc. (2026). *Docker documentation*. https://docs.docker.com/

[11] The Kubernetes Authors (2026). *Kubernetes documentation*. https://kubernetes.io/docs/

[12] GitHub (2026). *GitHub Actions documentation*. https://docs.github.com/actions

[13] Anthropic (2026). *Claude Code documentation*. https://code.claude.com/docs