The n8n Self-Hosting Starter

Shannon Atkinson, House of Loops

September 2026

The n8n Self-Hosting Starter

Two chapters from **The Ultimate Guide to Deploying n8n Community Edition**, *second edition, revised September 2026 for n8n 2.38 and the 3.0 transition.*

n8n 3.0, due in October 2026, drops npm installs: self-hosting means Docker. These two chapters get you from nothing to a working, production-shaped n8n on your laptop in under an hour, and explain the one component that most tutorials skip and that n8n 2.0 made mandatory: task runners.

Chapter 1 builds the reference stack this whole book uses: PostgreSQL 18, a pinned n8n image, an encryption key you own, and the task-runner sidecar, all in one Compose file. **Chapter 2** explains what task runners are, why external mode is the only isolation between user code and your credentials, and how to run JavaScript and native Python safely.

The full book has 34 chapters: VPS and cloud deployments, reverse proxies, tunnels, queue mode, Swarm, Kubernetes, Helm, high availability, backups, observability, security hardening, and the 2.0 and 3.0 upgrade chapters. It is free in the House of Loops community.

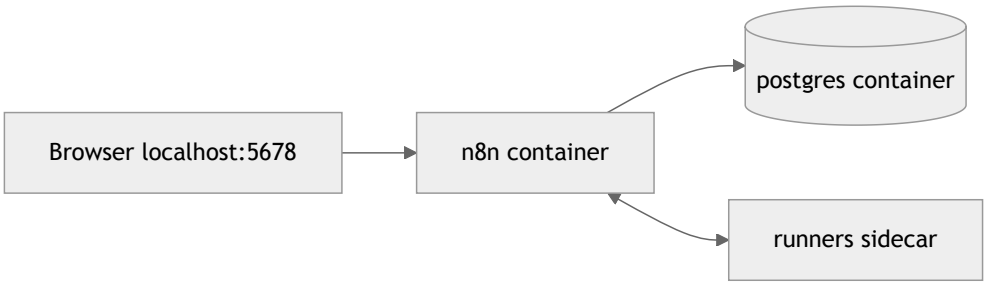
Component	Version used
n8n	2.38.1 (`n8nio/n8n`)
Task runners	2.38.1 (`n8nio/runners`)
PostgreSQL	18
Docker Compose	v2

Chapter 1: n8n on Your Laptop with Docker Compose

Overview

This chapter gets a working n8n on your own machine in about fifteen minutes. You will install Docker, write two small files, start three containers, create your owner account, and run a workflow that proves the whole stack works end to end.

The stack you build here is the same stack the rest of the book builds on: PostgreSQL for storage, the `n8nio/n8n` image for the editor and the execution engine, and the `n8nio/runners` sidecar that executes every Code node. Later chapters change the hostname, the proxy, and the number of containers. They do not change the shape of this file.



Diagram

Prerequisites

- A laptop with at least 4 GB of free RAM and 5 GB of free disk.
- macOS 13 or later, Windows 10/11 with WSL2, or a Linux distribution with a current kernel.
- A terminal you are comfortable typing in.
- Administrator rights on the machine, to install Docker.

No Node.js. No `npm`. Nothing on your machine but Docker.

1. Why Docker, and Only Docker

n8n 3.0 ships in **October 2026** and supports **Docker-based self-hosting only**. The npm and `npx n8n` install paths are already deprecated in the current 2.x line and will not carry forward.

Even before that deadline, Docker is the honest choice for a beginner:

- **The runtime is fixed.** The npm path still works today, but only on Node.js 20.19 through 24. Get that wrong and you get failures that look like n8n bugs. The image pins the runtime for you.
- **Task runners need a second process.** Since n8n 2.0, every Code node runs inside a task runner, not inside the main process. The supported production mode is `external`, which means a separate `n8nio/runners` container. That is a Compose file, not an `npm install`.
- **What you build here is what you deploy.** The file in this chapter is the file you copy to a VPS in the Ubuntu VPS with Docker Compose chapter. Nothing has to be relearned.

Two rules for the whole book. The command is `docker compose` with a space, never the old `docker-compose` binary. And every image is pinned to an exact version, never `latest`. Compose v2 is the only supported version, and the `version:` key at the top of a Compose file is obsolete — you will not see it here.

---

2. Install Docker

2.1 macOS

Download Docker Desktop from https://docs.docker.com/desktop/setup/install/mac-install/ and pick the build for your chip. Apple Silicon and Intel each have their own installer.

Drag it to Applications, launch it, and accept the terms. When the whale icon in the menu bar stops animating, Docker is running. Confirm from a terminal:

```
docker --version
docker compose version
```

You want Docker 24 or newer and Compose v2.x. If `docker compose version` errors but `docker --version` works, Docker Desktop is out of date; update it before continuing.

Apple Silicon needs no special flags. The n8n images are multi-arch and run natively on `arm64`.

2.2 Windows with WSL2

On Windows you run Docker Desktop with the WSL2 backend, and you keep your project files inside the Linux filesystem.

1. Open PowerShell as Administrator and install WSL2 with Ubuntu:

   ```
   wsl --install -d Ubuntu
   ```

   Reboot when prompted, then finish the Ubuntu first-run setup (username and password).

2. Install Docker Desktop from https://docs.docker.com/desktop/setup/install/windows-install/. In **Settings → General**, confirm **Use the WSL 2 based engine** is checked. In **Settings → Resources → WSL Integration**, enable integration for your Ubuntu distribution.

3. From now on, work inside the Ubuntu shell, not PowerShell. Open it from the Start menu or run `wsl` in a terminal, then verify:

   ```
   docker --version
   docker compose version
   ```

Put the project folder inside WSL, not on `/mnt/c`. Your Windows drives are mounted at `/mnt/c`, and that path works — but every file read crosses a translation layer between Windows and Linux. Bind mounts on `/mnt/c` are slow enough to make container startup and workflow execution visibly sluggish. Create your project under your WSL home directory (`~/`, which is `/home/<you>`) instead. Everything in this chapter uses named volumes, which live inside WSL regardless, so the only file that matters is the Compose file itself.

2.3 Linux

Do not install Docker Desktop on Linux. Install Docker Engine plus the Compose plugin, using Docker's convenience script or your distribution's instructions at https://docs.docker.com/engine/install/.

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Then add yourself to the `docker` group so you do not need `sudo` for every command:

```
sudo usermod -aG docker $USER
newgrp docker
docker compose version
```

Log out and back in if `newgrp` does not take effect in new terminals.

---

3. Create the Project Folder and Secrets

Everything lives in one folder. Two files, both of which you will reuse in later chapters.

```
mkdir -p ~/n8n
cd ~/n8n
```

Windows users: this must be your WSL home, so run it in the Ubuntu shell.

Generate the two secrets now, because you will paste them into `.env` in a moment:

```
openssl rand -hex 32   # N8N_ENCRYPTION_KEY
openssl rand -hex 32   # N8N_RUNNERS_AUTH_TOKEN
```

`N8N_ENCRYPTION_KEY` encrypts every credential n8n stores. Generate it once and never change it. If you lose it, the credentials in your database are unreadable — the workflows survive a restore, the credentials do not.

`N8N_RUNNERS_AUTH_TOKEN` is the shared secret between the n8n container and the runners sidecar. Both containers must present the same value or the runner will not connect and every Code node will fail.

Now create `.env`, pasting your two generated values:

```
# Generate with: openssl rand -hex 32
N8N_ENCRYPTION_KEY=replace-with-64-hex-characters
N8N_RUNNERS_AUTH_TOKEN=replace-with-a-long-random-string

POSTGRES_USER=n8n
POSTGRES_PASSWORD=replace-me
POSTGRES_DB=n8n

# Laptop values. Later chapters replace these with a real hostname.
N8N_HOST=localhost
GENERIC_TIMEZONE=America/Los_Angeles
```

Set `GENERIC_TIMEZONE` to your own zone. It decides when Schedule Trigger nodes fire.

Two notes on `.env` syntax, because n8n 2.0 upgraded its dotenv parser: a `#` starts a comment, and values containing backticks must be quoted. Your generated hex strings contain neither, so plain values are fine.

If this folder is ever a git repository, add `.env` to `.gitignore` before you commit anything.

---

4. Write `compose.yml`

Create `compose.yml` in the same folder:

```yaml
services:
  postgres:
    image: postgres:18
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]
      interval: 10s
      timeout: 5s
      retries: 5

  n8n:
    image: n8nio/n8n:2.38.1
    restart: unless-stopped
    ports:
      - "5678:5678"
    environment:
      N8N_ENCRYPTION_KEY: ${N8N_ENCRYPTION_KEY}
      DB_TYPE: postgresdb
      DB_POSTGRESDB_HOST: postgres
      DB_POSTGRESDB_PORT: 5432
      DB_POSTGRESDB_DATABASE: ${POSTGRES_DB}
      DB_POSTGRESDB_USER: ${POSTGRES_USER}
      DB_POSTGRESDB_PASSWORD: ${POSTGRES_PASSWORD}
      N8N_HOST: ${N8N_HOST}
      N8N_PROTOCOL: http
      WEBHOOK_URL: http://localhost:5678/
      N8N_EDITOR_BASE_URL: http://localhost:5678/
      GENERIC_TIMEZONE: ${GENERIC_TIMEZONE}
      TZ: ${GENERIC_TIMEZONE}
      N8N_RUNNERS_MODE: external
      N8N_RUNNERS_BROKER_LISTEN_ADDRESS: 0.0.0.0
      N8N_RUNNERS_AUTH_TOKEN: ${N8N_RUNNERS_AUTH_TOKEN}
      N8N_NATIVE_PYTHON_RUNNER: "true"
      N8N_DEFAULT_BINARY_DATA_MODE: filesystem
      EXECUTIONS_DATA_PRUNE: "true"
      EXECUTIONS_DATA_MAX_AGE: 336
      N8N_DIAGNOSTICS_ENABLED: "false"
    volumes:
      - n8n_data:/home/node/.n8n
      - n8n_files:/home/node/.n8n-files
    depends_on:
      postgres:
        condition: service_healthy

  runners:
    image: n8nio/runners:2.38.1
    restart: unless-stopped
    environment:
      N8N_RUNNERS_TASK_BROKER_URI: http://n8n:5679
      N8N_RUNNERS_AUTH_TOKEN: ${N8N_RUNNERS_AUTH_TOKEN}
      N8N_RUNNERS_AUTO_SHUTDOWN_TIMEOUT: 15
    depends_on:
      - n8n
```

```
volumes:
  postgres_data:
  n8n_data:
  n8n_files:
```

What the less obvious settings do:

- `N8N_PROTOCOL: http`, `WEBHOOK_URL`, and `N8N_EDITOR_BASE_URL` are the laptop values. Every chapter that puts n8n behind a proxy switches these to `https` and a real hostname and adds `N8N_PROXY_HOPS`.
- `N8N_RUNNERS_MODE: external` with `N8N_RUNNERS_BROKER_LISTEN_ADDRESS: 0.0.0.0` tells the n8n container to open its task broker on port 5679 so the sidecar can reach it over the Compose network. The other mode, `internal`, runs the runner as a child process and is not for production; use `external` from the start so nothing changes later. (`N8N_RUNNERS_ENABLED` was required on 1.x and is deprecated from 2.0 — leave it out.)
- `N8N_NATIVE_PYTHON_RUNNER: "true"` enables the native Python runner. n8n 2.0 removed the old Pyodide-based Python in favour of this.
- `EXECUTIONS_DATA_PRUNE` with `EXECUTIONS_DATA_MAX_AGE: 336` deletes execution history older than 14 days, so your laptop's Postgres volume does not grow forever.
- `N8N_DIAGNOSTICS_ENABLED: "false"` turns off telemetry.

Both images are pinned to `2.38.1`, and they must always match. A runners image on a different version than n8n is unsupported.

---

5. Start the Stack

```
docker compose up -d
```

The first run downloads three images, so give it a minute or two. Then check what is running:

```
docker compose ps
```

You want all three services in state `running`, with `postgres` showing `healthy`. Watch n8n come up:

```
docker compose logs -f n8n
```

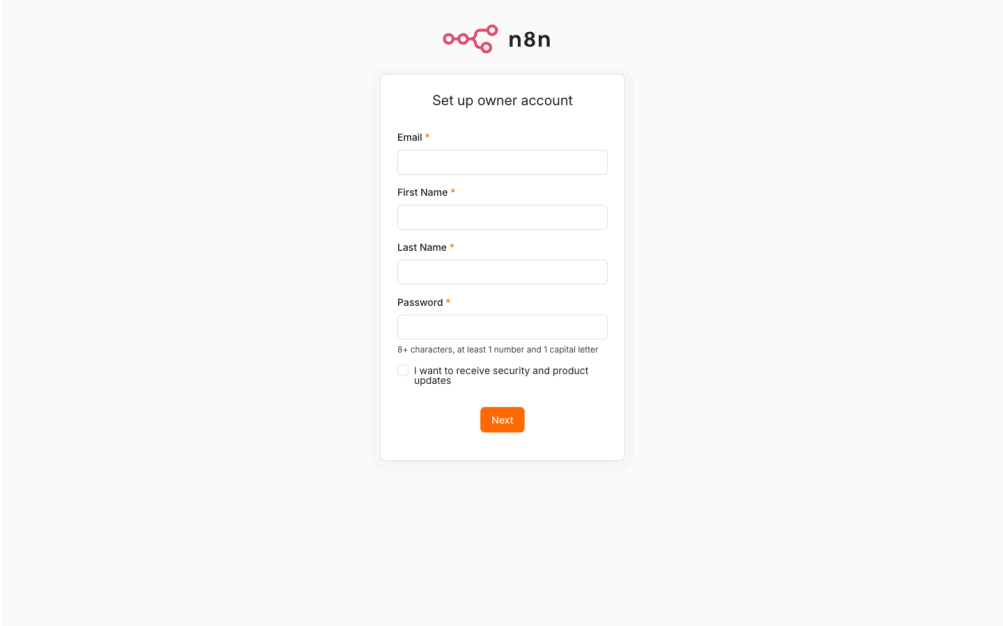
Wait for the line reading `Editor is now accessible via: http://localhost:5678/`. Press `Ctrl+C` to stop following the logs — that stops the log stream, not the containers.

---

6. Create Your Owner Account

Open http://localhost:5678 in a browser.

The first screen asks you to set up an owner account: email, first and last name, and a password. This is the account system built into n8n — there is no separate gate in front of the editor, and the first person to reach a fresh instance becomes the owner. Fill it in immediately, before anything else on your network can reach the port.

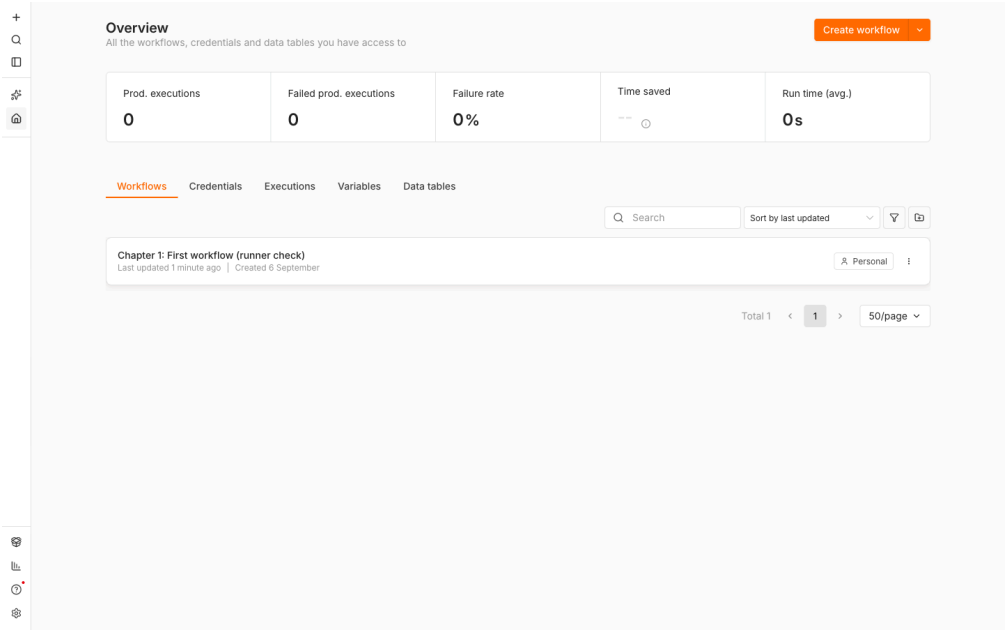


The owner-account form n8n shows on first visit. There is no basic auth in front of it; this account is the authentication.

The owner can later invite other people from **Settings** → **Users** with the roles owner, admin, and member, and can turn on two-factor authentication from the personal settings menu. Single sign-on and Projects with role-based access control are paid-plan features, not part of Community Edition.

---

7. A Two-Minute First Workflow



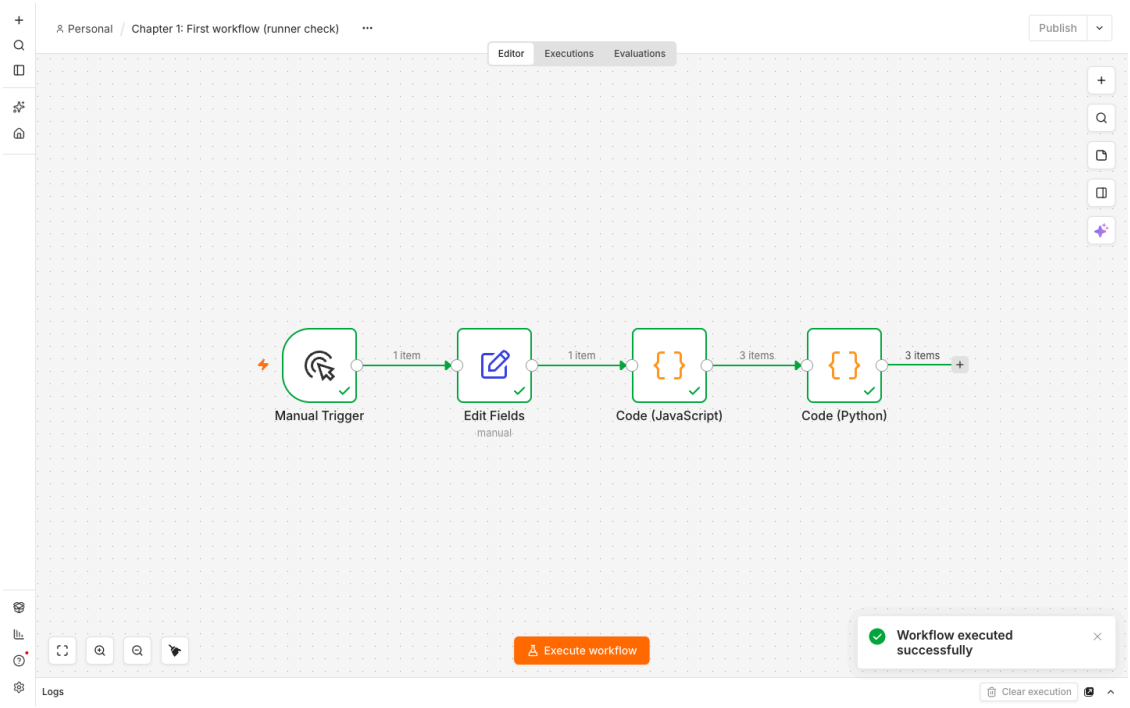
The workflows overview after the owner account exists. Everything in this book starts from here.

This exists to prove one specific thing: that the runners sidecar is connected. If a Code node runs, everything else is wired correctly.

1. Click **Create Workflow**.
2. Add a **Manual Trigger** node. (The old Start node was removed in 2.0; Manual Trigger replaces it.)
3. Add an **Edit Fields (Set)** node after it. Add a string field named `message` with the value `hello from n8n`.
4. Add a **Code** node after that. Leave the language as JavaScript and replace the body with:

```
const items = $input.all();
return [{ json: { echoed: items[0].json.message, itemCount: items.length } }];
```

5. Click **Execute workflow**. The Code node should return your message and a count of `1`.



6. Now change the Code node's language to **Python (Native)** and replace the body with one line:

```
return [{"json": {"engine": "python", "message": _input.first()["json"]["message"]}}]
```

7. Execute again. Same result, different runtime.

Both runs went to the `runners` container. If step 5 fails with a message about no runner being available, jump to the troubleshooting section — the auth token is almost always the cause.

Save the workflow. It persists in Postgres, so it survives restarts.

---

8. Where the Data Lives, and What to Back Up

The stack declares three named volumes. Docker manages them; they are not folders inside `~/n8n`.

```
docker volume ls
docker volume inspect n8n_postgres_data
```

Compose prefixes volume names with the project name, which defaults to the folder name — so `n8n_postgres_data` if your folder is `n8n`.

- `postgres_data` — workflows, credentials, executions, users. Everything that matters.
- `n8n_data` — `/home/node/.n8n`, which holds instance settings and filesystem-mode binary data.
- `n8n_files` — `/home/node/.n8n-files`, the only host path nodes may read or write by default, because n8n 2.0 sets `N8N_RESTRICT_FILE_ACCESS_TO` to that directory.

Two things need backing up: the Postgres volume and `N8N_ENCRYPTION_KEY`. One without the other is useless. Restore the database without the key and every stored credential decrypts to garbage.

Dump the database to a file in your project folder:

```
docker compose exec -T postgres pg_dump -U n8n n8n > n8n-backup-$(date +%F).sql
```

Restore into a fresh stack, with the same `.env` in place:

```
docker compose exec -T postgres psql -U n8n -d n8n < n8n-backup-2026-09-05.sql
```

Keep a copy of `N8N_ENCRYPTION_KEY` somewhere that is not the laptop — a password manager entry is fine. The Backup, Restore, and Disaster Recovery chapter covers backups properly.

---

9. Updating n8n

Updating means editing one thing: the pinned tag, on **both** images.

```
# in compose.yml, change n8nio/n8n:2.38.1 and n8nio/runners:2.38.1
# to the new version, keeping the two identical
docker compose pull
docker compose up -d
```

Compose recreates only the containers whose image changed. Your volumes, and therefore your workflows and credentials, are untouched. Database schema migrations run automatically when the new n8n container starts; watch `docker compose logs -f n8n` and let them finish.

Back up before a major version bump. If you would rather track releases than pick versions, the tag `stable` exists and always points at the current stable release — but then you no longer know what you are running, which is why the rest of this book pins.

---

10. Stopping and Removing

Stop the stack, keeping all data:

```
docker compose down
```

Start it again with `docker compose up -d`. Nothing is lost.

Delete everything, including the volumes:

```
docker compose down -v
```

`-v` removes the named volumes. Your workflows, credentials, and users are gone and cannot be recovered without a backup. There is no confirmation prompt.

---

11. Why Not SQLite Here

n8n runs perfectly well on SQLite, and for a single-user laptop it would be one less container. This book uses Postgres anyway.

The reason is continuity. Postgres is the production database for n8n — it is what queue mode requires, what every VPS and Kubernetes chapter uses, and what you would eventually migrate to. If you start on SQLite you learn one file, then throw it away and learn another. Starting on Postgres costs about 300 MB of RAM on your laptop and saves you a migration.

If you do choose SQLite for a throwaway experiment, remove the `postgres` service and all six `DB_*` variables; n8n falls back to SQLite in `/home/node/.n8n`. Note that SQLite now runs on the pooled driver only, tuned with `DB_SQLITE_POOL_SIZE`: the docs list the default as `0` (rollback journal), so set `DB_SQLITE_POOL_SIZE=2` explicitly to get the pooled WAL driver. MySQL and MariaDB are not options at all — support was dropped in 2.0.

---

12. Troubleshooting Common Issues

Port 5678 is already in use. `docker compose up -d` fails with `address already in use`. Find the offender, or just move n8n to another port by changing the mapping to `"5679:5678"` and setting `WEBHOOK_URL` and `N8N_EDITOR_BASE_URL` to `http://localhost:5679/` to match. The right side of the mapping is the port inside the container and does not change.

```
docker compose down
lsof -i :5678       # macOS and Linux
```

Everything is slow on Windows. Almost always a project folder on `/mnt/c`. Move it into your WSL home directory and run `docker compose up -d` again. Check with `pwd` inside the Ubuntu shell: you want `/home/<you>/n8n`, not `/mnt/c/Users/...`. The same applies to any bind mount you add later.

"Your n8n server is configured to use a secure cookie" when opening it from another device. You reached the editor over plain HTTP on a non-localhost address, such as `http://192.168.1.20:5678`. n8n refuses to send its session cookie over an insecure origin. You can turn that check off by adding to the n8n service:

```
      N8N_SECURE_COOKIE: "false"
```

Understand the tradeoff: the session cookie now travels unencrypted across your network, and anyone on that network can capture it and become you. It is acceptable for a few minutes of testing on a home LAN. It is not acceptable on shared Wi-Fi and never in production — for real access from other machines, use HTTPS behind a proxy (Reverse Proxy and HTTPS: Caddy, Nginx, and Traefik) or a tunnel (Exposing a Private Instance: Cloudflare Tunnel, ngrok, and SSH).

Code nodes fail, or the runner never connects. Check the sidecar's logs:

```
docker compose logs runners
```

An authentication or handshake error means `N8N_RUNNERS_AUTH_TOKEN` does not match between the two services. Both read it from the same `.env` variable, so the usual causes are a stray quote, a trailing space, or editing `.env` without recreating the containers. Compose only re-reads `.env` on `up`:

```
docker compose up -d --force-recreate
```

If the logs show a connection refused to `http://n8n:5679`, confirm `N8N_RUNNERS_BROKER_LISTEN_ADDRESS: 0.0.0.0` is set on the n8n service — the broker binds to localhost otherwise and the sidecar cannot reach it.

Apple Silicon. Nothing to do. `n8nio/n8n` and `n8nio/runners` are multi-arch images with native `arm64` builds. If you see a "platform mismatch" warning, you have added a `platform:` line somewhere — remove it. Do not force `linux/amd64`; emulation is slower and buys nothing.

n8n restarts in a loop. Read `docker compose logs n8n` from the top. The two common causes are Postgres credentials in `.env` that disagree with an existing `postgres_data` volume (the volume keeps

the password from first creation), and a changed `N8N_ENCRYPTION_KEY`. For the first, either restore the original password or `docker compose down -v` and start clean.

Further reading

- Install with Docker Compose: https://docs.n8n.io/deploy/host-n8n/install-options/install-using-docker-compose
- Set up task runners: https://docs.n8n.io/deploy/host-n8n/configure-n8n/set-up-task-runners
- Deployment environment variables: https://docs.n8n.io/deploy/host-n8n/configure-n8n/basic-configuration/use-environment-variables/deployment
- n8n 3.0 breaking changes: https://docs.n8n.io/changelog/v30-breaking-changes
- n8n 2.0 breaking changes: https://docs.n8n.io/changelog/v20-breaking-changes

---

Chapter 2: Task Runners and Safe Code Execution

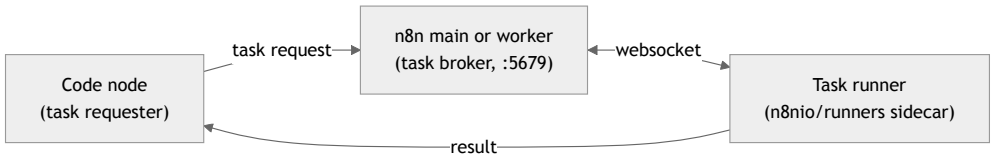
Overview

The Code node runs code that a workflow author typed into a browser. Since n8n 2.0, every Code node execution runs on a task runner. Task runners exist so that this code runs somewhere other than inside the n8n process.

The n8n docs are blunt about the stakes: task runners are the only isolation layer between user-provided code and n8n. Without them, or with a runner in internal mode, anyone who can edit a workflow could potentially read your database, your encryption key, your stored credentials, and your environment variables. That is not a theoretical concern on a shared instance. It is the reason this chapter exists.

In this chapter you will:

1. Learn what a task runner is and how the runner, broker, and requester fit together
2. Compare internal mode and external mode, and see exactly what each one exposes
3. Run the `n8nio/runners` sidecar next to n8n with a complete compose stack
4. Enable native Python in the Code node
5. Wire runners correctly in queue mode, where each worker needs its own sidecar
6. Add npm and pip packages by extending the runners image and allowlisting modules
7. Understand what 2.0 broke inside the Code node and why
8. Set resource limits, health checks, and the Kubernetes equivalent



Diagram

Prerequisites

- A working n8n stack from the reference `compose.yml` in this book, running `n8nio/n8n:2.38.1`
- Docker Engine with Compose v2 (`docker compose`)
- A long random string for `N8N_RUNNERS_AUTH_TOKEN` , generated once and stored in `.env`
- Shell access to the host so you can read container logs

1. What a Task Runner Is and Why It Exists

A task runner is a separate process whose only job is to execute one piece of user code and hand back the result. The Code node does not run JavaScript or Python itself any more. It packages the code and the input items into a task, sends the task away, and waits for an answer.

The point is blast radius. n8n holds your credential store and the `N8N_ENCRYPTION_KEY` that decrypts it. If arbitrary code runs inside that process, a sandbox escape reaches everything the process can reach. If the code runs in a different container, with a different filesystem, no database connection, and no credentials in its environment, an escape reaches a container that has nothing worth stealing.

Since 2.0 you no longer opt in. `N8N_RUNNERS_ENABLED` is deprecated and you can drop it. What you still choose is the *mode*, and that choice is the whole security story.

---

2. The Three Components

The feature has three parts, and it helps to name them because the log lines do.

Task requester. The Code node, running inside n8n. It submits a task and waits for the result.

Task broker. The n8n instance itself — main or worker — acts as the broker. It listens on port `5679` (`N8N_RUNNERS_BROKER_PORT`) and coordinates between requesters and runners. It does not execute anything.

Task runner. The process that actually executes the code. In external mode this lives in the sidecar container, started and supervised by a launcher application.

Runners connect *outward* to the broker over a websocket and authenticate with a shared secret. The broker then hands them tasks over that same connection. The runner sends a heartbeat every `N8N_RUNNERS_HEARTBEAT_INTERVAL` seconds (default 30); if it stops, the broker kills the task and the runner restarts.

The direction of that connection matters for your network design. The runner dials the broker, so the broker's port must be reachable from the runner, and the runner needs no inbound path from n8n at all.

---

3. Internal Mode Versus External Mode

`N8N_RUNNERS_MODE` takes two values, and the default is the wrong one for production.

`internal` (the default). n8n launches the runner as a child process of itself. Same host, same `uid` and `gid`, same filesystem, same environment. The n8n process manages its lifecycle. This is insecure by design and the docs say so. Code that escapes the runner's sandbox lands with exactly the access n8n has: the credentials table, the encryption key, the Postgres connection, the mounted volumes. Internal mode is acceptable on a throwaway instance holding mock data and nothing else.

`external`. A launcher application in a separate container starts runners on demand and supervises them. The runner container has its own filesystem and its own environment. It never receives `N8N_ENCRYPTION_KEY`, never receives `DB_POSTGRESDB_PASSWORD`, and cannot open a socket to Postgres unless you put it on that network yourself. An escape gets an attacker a shell in a container whose contents are a Node.js runtime, a Python runtime, and the code they already wrote.

External mode is what the rest of this chapter configures. One practical note: the runners image does **not** support file-based configuration. Variables with a `_FILE` suffix are ignored there, so pass the auth token as a plain environment variable from your `.env`.

---

4. The `n8nio/runners` Sidecar

The sidecar image contains the launcher, the JavaScript runner, and the Python runner. **Its version must match the n8n image version exactly.** These two images speak a private protocol; a mismatch produces connection errors or tasks that never complete.

`.env`:

```
# Generate with: openssl rand -hex 32
N8N_ENCRYPTION_KEY=replace-with-64-hex-characters
N8N_RUNNERS_AUTH_TOKEN=replace-with-a-long-random-string

POSTGRES_USER=n8n
POSTGRES_PASSWORD=replace-me
POSTGRES_DB=n8n

N8N_HOST=n8n.example.com
GENERIC_TIMEZONE=America/Los_Angeles
```

`compose.yml`:

```yaml
services:
  postgres:
    image: postgres:18
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]
      interval: 10s
      timeout: 5s
      retries: 5

  n8n:
    image: n8nio/n8n:2.38.1
    restart: unless-stopped
    ports:
      - "127.0.0.1:5678:5678"
    environment:
      N8N_ENCRYPTION_KEY: ${N8N_ENCRYPTION_KEY}
      DB_TYPE: postgresdb
      DB_POSTGRESDB_HOST: postgres
      DB_POSTGRESDB_PORT: 5432
      DB_POSTGRESDB_DATABASE: ${POSTGRES_DB}
      DB_POSTGRESDB_USER: ${POSTGRES_USER}
      DB_POSTGRESDB_PASSWORD: ${POSTGRES_PASSWORD}
      N8N_HOST: ${N8N_HOST}
      N8N_PROTOCOL: https
      WEBHOOK_URL: https://${N8N_HOST}/
      N8N_EDITOR_BASE_URL: https://${N8N_HOST}/
      N8N_PROXY_HOPS: 1
      GENERIC_TIMEZONE: ${GENERIC_TIMEZONE}
      TZ: ${GENERIC_TIMEZONE}
      N8N_RUNNERS_MODE: external
      N8N_RUNNERS_BROKER_LISTEN_ADDRESS: 0.0.0.0
      N8N_RUNNERS_AUTH_TOKEN: ${N8N_RUNNERS_AUTH_TOKEN}
      N8N_NATIVE_PYTHON_RUNNER: "true"
```

```
      N8N_DEFAULT_BINARY_DATA_MODE: filesystem
      EXECUTIONS_DATA_PRUNE: "true"
      EXECUTIONS_DATA_MAX_AGE: 336
      N8N_DIAGNOSTICS_ENABLED: "false"
    volumes:
      - n8n_data:/home/node/.n8n
      - n8n_files:/home/node/.n8n-files
    depends_on:
      postgres:
        condition: service_healthy

  runners:
    image: n8nio/runners:2.38.1
    restart: unless-stopped
    environment:
      N8N_RUNNERS_TASK_BROKER_URI: http://n8n:5679
      N8N_RUNNERS_AUTH_TOKEN: ${N8N_RUNNERS_AUTH_TOKEN}
      N8N_RUNNERS_AUTO_SHUTDOWN_TIMEOUT: 15
    depends_on:
      - n8n

volumes:
  postgres_data:
  n8n_data:
  n8n_files:
```

What each runner variable does:

- `N8N_RUNNERS_MODE=external` tells n8n not to fork a child process and to wait for a runner to connect instead.
- `N8N_RUNNERS_BROKER_LISTEN_ADDRESS=0.0.0.0` is required. The broker binds `127.0.0.1` by default, which is unreachable from a second container. Without this, the runner will never connect.
- `N8N_RUNNERS_AUTH_TOKEN` must be byte-identical on both services. It is the only thing standing between your broker port and anything else that can reach it.
- `N8N_RUNNERS_TASK_BROKER_URI=http://n8n:5679` uses the compose service name. If you rename the `n8n` service, change this too.
- `N8N_RUNNERS_AUTO_SHUTDOWN_TIMEOUT=15` stops an idle runner process after 15 seconds. The launcher restarts it when the next task arrives. Set `0` to keep runners resident, which trades memory for a small latency saving on the first Code node after a quiet period.

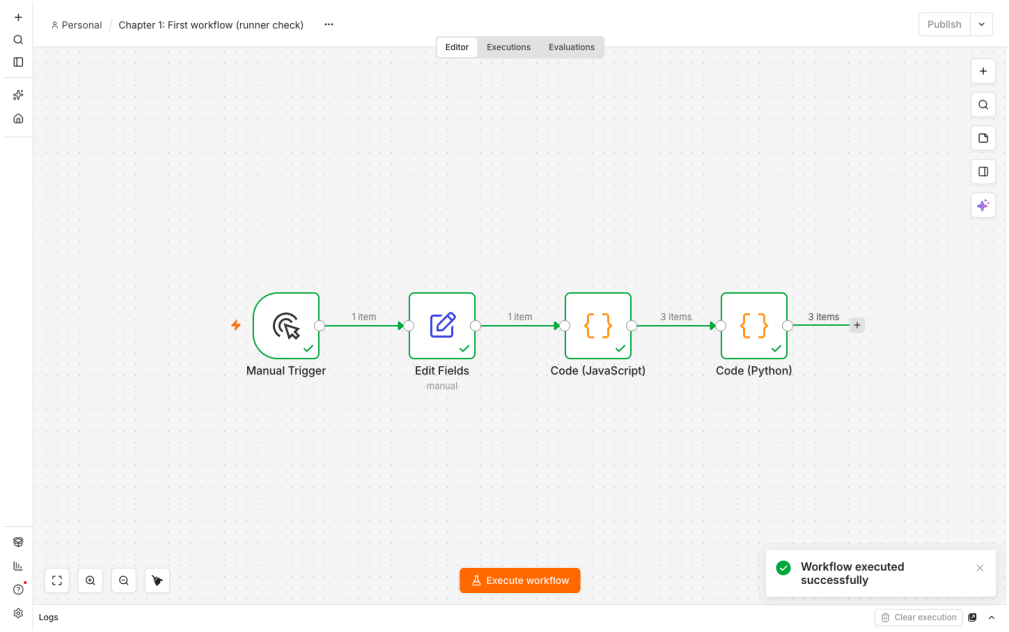
Bring it up and confirm the connection:

```
docker compose up -d
docker compose logs -f runners
```

You are looking for the launcher reporting a successful connection to the broker. Then open any workflow, add a Code node with `return [{json: {ok: true}}]`, and run it.

---

5. Native Python



A workflow with a JavaScript Code node and a native Python Code node, both executed on the external runner. The runner container logs show the `launcher:js` and `launcher:py` processes accepting the tasks.

n8n 2.0 removed the Pyodide-based Python Code node and replaced it with a native Python runner. Set `N8N_NATIVE_PYTHON_RUNNER=true` on the n8n container, as in the stack above. Python in the Code node works **only** with external mode.

Two differences will bite you when migrating old workflows:

- The native runner does not provide the Pyodide built-ins. `_input` is gone, and so is dot-access notation on items. Read items through the documented native API instead.
- In a Python Code *tool* — the Code node attached to an AI agent — the agent's query arrives as `_query`.

Everything else is a normal CPython process, which is why `numpy` and `pandas` are now realistic (see section 7) and why the performance is nothing like Pyodide's.

---

6. Runners in Queue Mode

Queue mode changes the rule, and this is the most common misconfiguration in production stacks.

Every n8n process that acts as a broker needs its own runners sidecar. Each worker is a broker for the Code nodes it executes. One shared runners container pointed at main does not serve the workers.

So:

- Each `n8n-worker` gets its own `runners` service pointing at that worker's port 5679.
- Main also needs a sidecar **unless** `OFFLOAD_MANUAL_EXECUTIONS_TO_WORKERS=true`, which hands manual editor executions to workers so main never runs a Code node. Running with offloading disabled is not recommended for production anyway.

A worker plus its runner, as a fragment of the queue-mode stack:

```
n8n-worker:
  image: n8nio/n8n:2.38.1
  restart: unless-stopped
  command: worker --concurrency=10
  environment:
    N8N_ENCRYPTION_KEY: ${N8N_ENCRYPTION_KEY}
    DB_TYPE: postgresdb
    DB_POSTGRESDB_HOST: postgres
    DB_POSTGRESDB_DATABASE: ${POSTGRES_DB}
    DB_POSTGRESDB_USER: ${POSTGRES_USER}
    DB_POSTGRESDB_PASSWORD: ${POSTGRES_PASSWORD}
    EXECUTIONS_MODE: queue
    QUEUE_BULL_REDIS_HOST: redis
    N8N_DEFAULT_BINARY_DATA_MODE: database
    N8N_RUNNERS_MODE: external
    N8N_RUNNERS_BROKER_LISTEN_ADDRESS: 0.0.0.0
    N8N_RUNNERS_AUTH_TOKEN: ${N8N_RUNNERS_AUTH_TOKEN}
    N8N_NATIVE_PYTHON_RUNNER: "true"
    GENERIC_TIMEZONE: ${GENERIC_TIMEZONE}
    TZ: ${GENERIC_TIMEZONE}
  depends_on:
    postgres:
      condition: service_healthy

worker-runners:
  image: n8nio/runners:2.38.1
  restart: unless-stopped
  environment:
    N8N_RUNNERS_TASK_BROKER_URI: http://n8n-worker:5679
    N8N_RUNNERS_AUTH_TOKEN: ${N8N_RUNNERS_AUTH_TOKEN}
    N8N_RUNNERS_AUTO_SHUTDOWN_TIMEOUT: 15
  depends_on:
    - n8n-worker
```

If you scale workers with `docker compose up -d --scale`, this pairing breaks — the service name resolves to several containers and runners will attach arbitrarily. Define workers as distinct services (`n8n-worker-1`, `n8n-worker-2`) each with its own sidecar, or move to an orchestrator that models sidecars properly.

---

7. Adding npm and pip Dependencies

The runners image ships a deliberately minimal set of modules, and the launcher configuration file is locked down. Adding a package takes two steps: install it in the image, then allowlist it for the Code node. Doing only the first gives you a package the Code node still refuses to import.

Extend the image:

```
FROM n8nio/runners:2.38.1
USER root
RUN cd /opt/runners/task-runner-javascript && pnpm add moment uuid
RUN cd /opt/runners/task-runner-python && uv pip install numpy pandas
COPY n8n-task-runners.json /etc/n8n-task-runners.json
USER runner
```

The launcher reads its configuration from `/etc/n8n-task-runners.json`. You can bake it in with `COPY`, as above, or mount it at that path from the host:

```
    volumes:
      - ./n8n-task-runners.json:/etc/n8n-task-runners.json:ro
```

`n8n-task-runners.json`:

```
{
  "task-runners": [
    {
      "runner-type": "javascript",
      "env-overrides": {
        "NODE_FUNCTION_ALLOW_BUILTIN": "crypto",
        "NODE_FUNCTION_ALLOW_EXTERNAL": "moment,uuid"
      }
    },
    {
      "runner-type": "python",
      "env-overrides": {
        "PYTHONPATH": "/opt/runners/task-runner-python",
        "N8N_RUNNERS_STDLIB_ALLOW": "json",
        "N8N_RUNNERS_EXTERNAL_ALLOW": "numpy,pandas"
      }
    }
  ]
}
```

The four allowlists are:

- `NODE_FUNCTION_ALLOW_BUILTIN` — Node.js built-in modules
- `NODE_FUNCTION_ALLOW_EXTERNAL` — third-party JavaScript packages
- `N8N_RUNNERS_STDLIB_ALLOW` — Python standard library modules
- `N8N_RUNNERS_EXTERNAL_ALLOW` — third-party Python packages

Keep these lists short and specific. Every entry is a capability you are granting to anyone who can edit a workflow. Allowlisting `child_process` or `fs` hands back most of what external mode took away.

Build and pin your own tag, and rebuild it every time you bump n8n:

```
docker build -t registry.example.com/n8n-runners:2.38.1 .
```

---

8. What 2.0 Broke, and the Escape Hatch You Should Not Use

Two changes catch people upgrading.

`$evaluateExpression()` no longer works in the Code node. Code node executions now run on runners in secure mode, and secure mode disables evaluating strings as code — which is precisely the mechanism expressions rely on. Calls return `null` or throw. Expressions in ordinary node fields, such as Edit Fields (Set), are unaffected. Rewrite the logic in plain JavaScript.

`N8N_RUNNERS_INSECURE_MODE=true` disables all security measures in the runner for compatibility with modules that need those insecure features. The docs discourage it for production and so does this book: turning it on undoes the isolation you set up in section 4 while leaving the sidecar in place, which is worse than obvious, because the stack still looks correct.

`N8N_BLOCK_ENV_ACCESS_IN_NODE` now defaults to `true`. Code nodes cannot read `process.env`. If a workflow depended on that, the migration path is to move the value into a credential. Setting `N8N_BLOCK_ENV_ACCESS_IN_NODE=false` restores the old behaviour and re-exposes every environment variable on the process to anyone who can edit a workflow.

---

9. Limits, Health Checks, and Kubernetes

The sidecar runs untrusted code, so cap what it can consume. A runaway loop should hit a limit, not the host's OOM killer.

`N8N_RUNNERS_MAX_CONCURRENCY` and `N8N_RUNNERS_MAX_OLD_SPACE_SIZE` are both **n8n instance** environment variables per the docs, not runner variables — add them to the `n8n` service's environment block:

```
      N8N_RUNNERS_MAX_CONCURRENCY: 5
      N8N_RUNNERS_MAX_OLD_SPACE_SIZE: 1024
```

The runners sidecar itself only needs its connection settings, a resource ceiling, and a health check:

```
  runners:
    image: n8nio/runners:2.38.1
    restart: unless-stopped
    environment:
      N8N_RUNNERS_TASK_BROKER_URI: http://n8n:5679
      N8N_RUNNERS_AUTH_TOKEN: ${N8N_RUNNERS_AUTH_TOKEN}
      N8N_RUNNERS_AUTO_SHUTDOWN_TIMEOUT: 15
    deploy:
      resources:
        limits:
          cpus: "2.0"
          memory: 2G
    healthcheck:
      test: ["CMD-SHELL", "wget -q -O- http://127.0.0.1:5680/healthz || exit 1"]
      interval: 30s
      timeout: 5s
      retries: 3
    depends_on:
      - n8n
```

The launcher serves `/healthz` on port `5680` (`N8N_RUNNERS_LAUNCHER_HEALTH_CHECK_PORT`); the broker serves `/healthz` on `5679`. If your image has no `wget`, drop the `healthcheck` block and probe port 5680 from outside instead. To be clear about where each setting lives: `N8N_RUNNERS_MAX_CONCURRENCY` (default 5) and `N8N_RUNNERS_TASK_TIMEOUT` (default 300 seconds) are set on the **n8n** container, not the runner; so is `N8N_RUNNERS_MAX_OLD_SPACE_SIZE`, which caps the Node.js heap (in MB) that the runner is launched with.

On Kubernetes, put the launcher in the **same pod** as the n8n main or worker container. They share the pod network, so `N8N_RUNNERS_TASK_BROKER_URI=http://127.0.0.1:5679` and you can leave the broker on its default listen address. Give the sidecar its own `resources.limits` and a `livenessProbe` on `5680`. If you prefer a separate Deployment, keep `N8N_RUNNERS_BROKER_LISTEN_ADDRESS=0.0.0.0`, expose 5679 through a Service, and point the runners at it — but you lose the one-runner-per-broker guarantee, so the sidecar pattern is the safer default.

---

10. Troubleshooting Common Issues

Issue	Cause	Solution
Runner never connects; broker logs show nothing	`N8N_RUNNERS_AUTH_TOKEN` differs between the two services	Compare with `docker compose exec n8n printenv N8N_RUNNERS_AUTH_TOKEN` and the same on `runners`. A trailing space or unexpc `.env` value is the usual cause
Runner logs “connection refused” to port 5679	Broker still bound to `127.0.0.1`	Set `N8N_RUNNERS_BROKER_LISTEN_ADDRESS=0.0.` on the n8n container and recreate it
Runner connects, then drops repeatedly	Version mismatch between `n8nio/n8n` and `n8nio/runners`	Pin both to the same tag (`2.38.1`) and recrea both containers
“Code node timed out”	Task exceeded `N8N_RUNNERS_TASK_TIMEOUT` (300s), or no runner was free within `N8N_RUNNERS_TASK_REQUEST_TIMEOUT` (60s)	Raise the timeout, raise `N8N_RUNNERS_MAX_CONCURRENCY`, or add a second sidecar. Also check `N8N_RUNNERS_AUTO_SHUTDOWN_TIMEOUT` isn’t fighting a very slow first task
Code node hangs forever in queue mode	A worker has no runners sidecar	Give every worker its own sidecar pointing at t worker’s `:5679`
Manual runs hang, scheduled runs work	Main has no sidecar and manual executions aren’t offloaded	Set `OFFLOAD_MANUAL_EXECUTIONS_TO_WORKERS=t` or add a sidecar for main
Python: `ModuleNotFoundError` for an installed package	Package installed but not allowlisted	Add it to `N8N_RUNNERS_EXTERNAL_ALLOW` (or `N8N_RUNNERS_STDLIB_ALLOW`) in `/etc/n8n-task-runners.json` and restart the sidecar
JS: “Cannot find module”	Same, on the JavaScript side	Add it to `NODE_FUNCTION_ALLOW_EXTERNAL`, a confirm `pnpm add` ran in `/opt/runners/tas runner-javascript`
Python Code node unavailable in the editor	`N8N_NATIVE_PYTHON_RUNNER` unset, or mode is `internal`	Set `N8N_NATIVE_PYTHON_RUNNER=true` and `N8N_RUNNERS_MODE=external` on n8n
`_input` or dot access throws in Python	Pyodide built-ins removed in 2.0	Rewrite against the native Python API; in Cod tools use `_query`
`$evaluateExpression()` returns `null`	Secure mode disables string evaluation	Rewrite as plain JavaScript. Do not enable `N8N_RUNNERS_INSECURE_MODE`
`process.env` is empty in a Code node	`N8N_BLOCK_ENV_ACCESS_IN_NODE` defaults to `true` since 2.0	Move the value into a credential rather than flipping the flag
Token set via `N8N_RUNNERS_AUTH_TOKEN_FILE` is ignored	The runners image has no file-based configuration	Pass the plain variable from `.env`

Further reading

- https://docs.n8n.io/deploy/host-n8n/configure-n8n/set-up-task-runners
- https://docs.n8n.io/deploy/host-n8n/configure-n8n/basic-configuration/use-environment-variables/task-runners
- https://docs.n8n.io/deploy/host-n8n/configure-n8n/security/harden-task-runners
- https://docs.n8n.io/changelog/v20-breaking-changes
- https://docs.n8n.io/deploy/host-n8n/configure-n8n/scaling/enable-queue-mode
- https://docs.n8n.io/deploy/host-n8n/configure-n8n/basic-configuration/use-environment-variables/queue-mode
- https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.code
- https://github.com/n8n-io/task-runner-launcher/blob/main/docs/setup.md

---

Get the rest of the book, and the person who wrote it

The remaining 32 chapters, the compose files, and the Kubernetes manifests are shared in the **House of Loops** community on Skool, free:

https://www.skool.com/houseofloops

House of Loops is a small community for people building with n8n, Make, and AI agents who want workflows that keep working after launch. Post the workflow that broke and the reply comes from the person who wrote this guide.

Weekly newsletter, one working workflow per issue, at **https://www.houseofloops.com**.

The Ultimate Guide to Deploying n8n Community Edition, second edition. First edition April 2025, revised September 2026.