Nornir, and where it goes next.

State of the project, the stewardship change, and a first look at the roadmap.

Damien Garros · OpsMill · CHNUG #4

About me — Damien Garros

Co-founder and CEO of OpsMill

Creator of Infrahub, an infrastructure data management platform — a source of truth built for automation.

15+ years in infrastructure automation

Infrastructure as code, network automation and observability, at large scale. Previously led Technical Architecture at Network to Code.

OpsMill became the steward of Nornir in June 2026 — which is why I am giving this update.

OpsMill

Infrahub

in/damiengarros

gh/dgarros

What is Nornir ?

Nornir is an open source automation framework written in pure Python developed for infrastructure automation

No DSL, no playbooks, no agent on the device.

Created in 2017 by David Barroso

```
from nornir import  InitNornir

nr =  InitNornir(config_file="config.yaml")

def gather_facts(task):
    task.run(
        task=napalm_get,
        getters=["facts"]
    )


ams = nr.filter(site="ams01")

print_result(ams.run(task=check))
```

Pseudo code, syntax may change

A small core, a large ecosystem

Each plugin type is an extension point, and the community has filled all of them.

Connection	`netmiko · napalm · scrapli · netconf · pygnmi · pyez`
Inventory	`yaml · netbox · nautobot · infrahub · ansible · sql · csv`
Tasks	`jinja2 · http · pyxl · collections from nautobot, infrahub`
Runners	`threaded · serial · salt · conditional · cached`
Processors	`rich · nautobot — hooks into execution events`

Every vendor and every source of truth is reachable through a plugin.

Why people keep using it after eight years

It’s just Python

Loops, tests, type hints, pytest. No templating language wrapped around a DSL.

You can debug it

You can step through your automation with pdb. A run against one host executes inline, with no threads in between.

A small core, by design

A small core plus a plugin ecosystem — Netmiko, NAPALM, Scrapli. You choose the parts you need.

Fits your source of truth

Inventory plugins for NetBox, Nautobot and Infrahub. Concurrent execution across thousands of devices.

Widely used, and still growing

100K/mo

PyPI downloads, and growing — about 5K a day

1.6K

GitHub stars and 260 forks, plus thousands of daily plugin downloads

8yrs

in production at enterprises, ISPs and research networks

IN THE STACK `source of truth → nornir → devices`

OpsMill is the new steward of Nornir

This is continuity, not a takeover: core maintainer Patrick Ogenstad is OpsMill employee number two, and nornir-netbox author Wim Van Deun is on the team.

David Barroso, creator of Nornir, becomes Technical Advisor for OpsMill

THE COMMITMENTS

Nornir stays open source. & community-driven.

The roadmap gets built with the community, not handed to it.

What we are taking responsibility for

NOW SUPPORTED BY OPSMILL

`nornir`	The core framework
`nornir_napalm`	NAPALM connection plugin and tasks
`nornir-utils`	print_result, YAMLInventory and other utilities
`nornir_jinja2`	Jinja2 templating tasks

STAYS WHERE IT IS

The wider ecosystem stays with its maintainers — netmiko, scrapli, nornir_netbox, the Nautobot and vendor plugins.

Stewardship of the core is not ownership of the ecosystem.

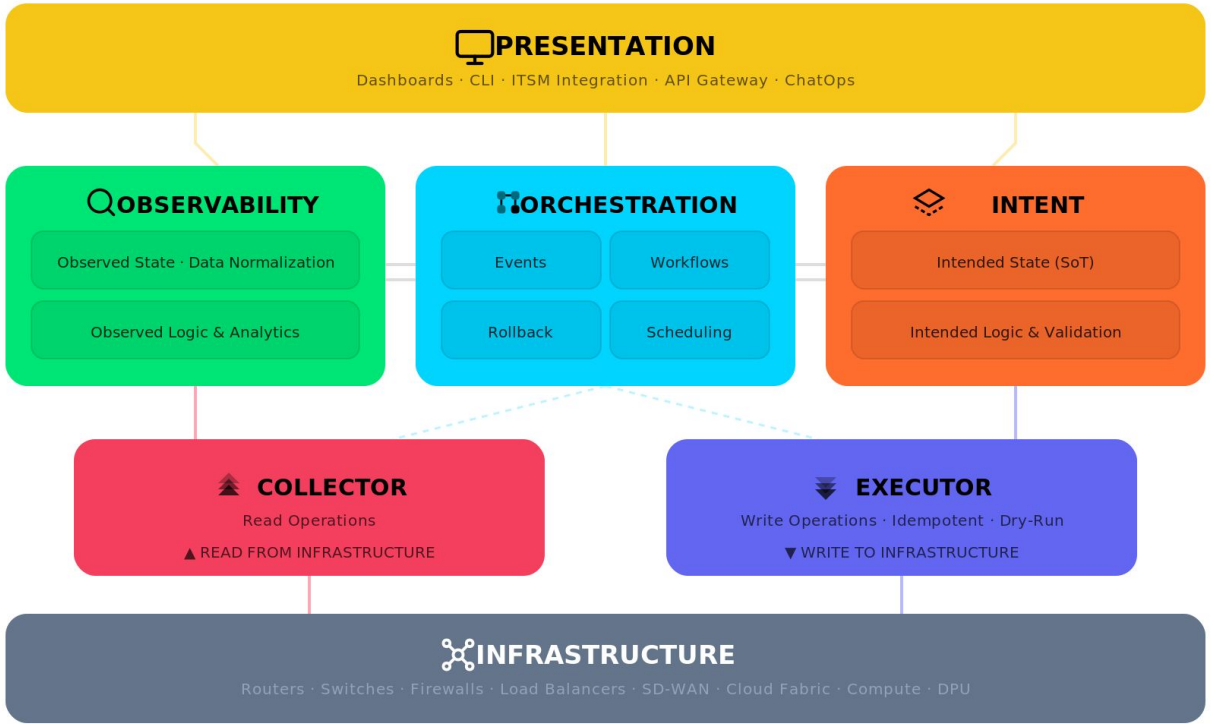
We support plugin authors.
We do not take over their work.

Why we did it

A project used this widely should receive active investment again.

Nornir fills the execution part of the stack we care about. Infrahub holds the intent and the source of truth, Nornir performs the execution. The two complement each other.

And we believe the execution layer is about to become much more important.



NAF Framework

Agents & AI are changing the automation landscape

Orchestration and execution used to ship together as one tool. That's being redefined: agents are taking over the orchestrator role, and the execution layer needs to stand on its own.

> A strong API / MCP layer to interact with the network is more critical than ever and Nornir is the foundation for it.

The rules the roadmap follows

01 **Built with the community**

The direction is public, one discussion per topic each. Nothing is final yet.

02 **Same architecture**

The core ships the protocol, the entry point, and a default.
Every implementation ships as a plugin.

03 **Additive, not breaking**

No existing signature, protocol member, default value or entry-point name changes. Your code keeps working.

The roadmap proposal

Published for discussion before the code is settled. One thread per proposal — comment where you have opinions.

01	Native Async support	Run inside an event loop, connect async transports
02	Enhanced Runner	Streaming progress, timeouts, cancellation, detach
03	Operations	Typed, multi-vendor actions with per-vendor drivers
04	Improve Documentation	A refresh, and a question about the ecosystem
05	Agentic Development	Set the repository up for Agentic development, Context, SSD

Async, as a runner you opt into

nr.run() inside an async service blocks its event loop. Async transports have nowhere to connect. Thousands of hosts mean thousands of threads.

Nornir.run() holds no concurrency logic — it delegates to the runner. So this is mostly a new plugin, not a change to the core.

Keep the threaded runner, and nothing changes.

```python
async def gather_facts (task):
    conn = await task.host.aget_connection(
                "my_transport",
                nr.config
            )

    return await conn.send("show version" )

nr =  InitNornir(config_file="config.yaml" )
results = await nr.arun(task=gather_facts)


inside a FastAPI service:
@app.post("/collect")
async def collect (nr: Depends(nr_init)):
    r = await nr.arun(gather_facts)
    return {"failed": list(r.failed_hosts)}
```

Pseudo code, syntax may change

Async inventory and processors

Inventory increasingly comes from HTTP and GraphQL APIs — Infrahub, NetBox, Nautobot — and those clients are async now.

And under an async runner, one processor that blocks — a webhook, a database write — stalls every other host.

Sibling protocols you opt into. Existing plugins stay untouched.

```
nr = await AInitNornir(config_file="config.yaml")

class InfrahubInventory :

    async def aload (self) -> Inventory:
        data = await self.client.query(Q)
        return self.build(data)

class LogProcessor :

    async def atask_instance_completed (self, ...):
        await self.http.post(URL, ...)
```

Pseudo code, syntax may change

Runners that don’t block

Today a long run blocks until every host finishes.

→ Partial updates — results stream as hosts finish

→ Timeouts — per host and per run

→ Cancellation — stop a run in flight

→ Detached mode — submit, reattach later

Same task on both runners. Early shape — nothing final.

```
def upgrade (task, image):
    yield Result(
        host=task.host,
        result= "uploading",
        progress=0.2)

    return Result(
            host=task.host,
            result="done",
            changed=True)

run = nr.submit(upgrade,
            image="ios-17.9.bin",
            per_host_timeout=600)

for event in run.events ():
     if event.host_done and  event.failed:
        run.cancel()  # stop the rest
```

Pseudo code, syntax may change

Nornir Operations

One logical action across IOS, EOS and Junos still means hand-written platform dispatch, and inputs you learn by reading source.

A new task type, next to plain tasks

Operations sit alongside the tasks you run today — nothing existing changes or breaks.

Typed, end to end

Pydantic input and output. Validated before any device is contacted; a host with no driver fails cleanly.

Non-blocking by default

Operations run detached: submit, stream progress, reattach — built on the runner work.

Definitions can live anywhere

In core, in a plugin, or in your own repo. Anyone can publish drivers.

Defining an Operation

```
class GetConfigInput (BaseModel):
    retrieve: str = "running"

class GetConfigOutput (BaseModel):
    config: str
    source: str

class GetConfig(Operation):
    name = "get_config"
    input = GetConfigInput
    output = GetConfigOutput
```

Pseudo code, syntax may change

Register an Operation 1/2

```python
from nornir.core.operations import OperationRegistry
from myproject.operations import GetConfigInput, GetConfigOutput

# via decorator
@driver("get_config", "eos", "scrapli")
async def eos_get_config (task, params):
    resp = await task.conn.send("show run" )
    return GetConfigOutput(config=resp.result)

# Direct registration
OperationRegistry.register(
    operation="get_config", platform="ios", connection="napalm",
    driver=eos_get_config,
)
```

Pseudo code, syntax may change

Register an Operation 2/2

```
[project.entry-points."nornir.operations.get_config"]
"ios.napalm"  = "nornir_napalm.operations:IosNapalmGetConfig"
"eos.napalm"  = "nornir_napalm.operations:EosNapalmGetConfig"
"eos.scrapli" = "nornir_scrapli.operations:EosScrapliGetConfig"
```

Pseudo code, syntax may change

Improve the documentation

Better search
It should get you to the page you need

A modern site
The navigation and layout people expect

A real plugin catalogue
Filter by type, see maintenance status

A clearer path
Tutorial, how-to, reference kept distinct

A QUESTION FOR THE ECOSYSTEM

The docs are not one site: the core, plus about thirty plugin sites on separate domains. 26 of 29 community plugins live outside the organisation, across 22 owners.

One shared entry point, or a good catalogue that links out?

Whatever the answer, taking part stays opt-in for plugin maintainers.

Support Agentic Development

People already use AI agents on Nornir, and the quality of the result depends on luck.

Most of what an agent needs to know is real but unwritten — it lives in maintainers' heads.

We write it down once, in a form both people and agents can read.

```
AGENTS.md   what agents read first
CLAUDE.md   imports AGENTS.md
.agents/    shared skills + rules
.specify/   plan large changes first
```

Nothing changes in the released package.
Nobody has to use AI tools to contribute.

SpecKit: spec-driven development

A toolkit for coding agents. You write down what you want; the agent works out how; a separate step checks the code against the spec.

VIBE CODING

`prompt → code`

Verified by a hunch: "looks fine?" Nothing a machine can check. Fine for a one-off script.

SPEC-DRIVEN DEVELOPMENT

`spec → plan → tasks → code`

Three markdown files a machine can read back and disagree with — reviewable in a pull request like anything else.

Vibe coding is not SDD. For changes to a framework people depend on, the spec comes first.

The path to 3.7

3.7, not 4.0 — no breaking changes, so no migration guide needed. The best time to weigh in is now, while the design is still open.

`weeks 0–6`	Design discussion — the threads are open now
`in parallel`	Alpha releases ship with whatever is already implemented
`after that`	Beta, once the API discussion converges
`+4–6 weeks`	Stable release, if the beta holds

Get involved

`Comment`	Proposal threads are open — direct feedback is more useful than polite feedback Join the discussion on discord / Slack
`Plugins`	If you maintain one, we want to talk to you specifically
`Alphas`	Alpha releases ship as pieces land — test them and tell us what breaks

THANKS

Questions and comments

Nornir stays open source, stays community-driven, and receives the investment it needs.

nornir.tech github.com/nornir-automation opsmill.com