Ask → live. Same moment.

ndexr — agentic development on an immutable foundation

ndexr

The phase we are in

Software at the speed
you can **think about it**.

A fleet of agents. One immutable foundation. The live site changes while you watch.

The claim, plainly

*Describe the change in plain language, and it is live — on a foundation where every version is **named, pinned and rebuildable**.*

- Fast on its own is a demo.
- Reproducible on its own is a build farm.
- The point is that this system is **both at once**.

What this replaces

The old shape: describe it to a developer, wait days, get back something close, explain the gap, wait again.

- Every handoff is a chance for the intent to arrive wrong.
- The wait is where the requirement goes stale.
- By the time it lands, the question has usually moved.

That loop is not slow because people are slow. It is slow because it has a handoff in it.

The office is wherever you are

These systems are run and changed **from a phone**.

- Not a dashboard that reports on the work — the actual controls.
- The change lands in front of the person who asked for it, in the same conversation.
- They react to what they see, and that reaction is the next change.

No ticket, no estimate, no “we’ll come back to you Thursday”

Two things that usually trade off

Speed normally costs you provenance. Provenance normally costs you speed.

- Move fast $\rightarrow$ nobody can say what version is running, or rebuild it.
- Lock it down $\rightarrow$ a one-line change takes a ticket, a sprint, a release.
- ndexr refuses the trade: the fast path *is* the reproducible path.

The loop

Plain language in, live display out — and seeing it live is what produces the next request.

Ask → agent edits the module → rolling deploy → your users see it

- The ask lands on the domain's own agent.
- The edit becomes a **git commit** — nothing is changed in place.
- The app rotates onto the new build without dropping a session.
- No staging step. This is the live site.

Every domain is one system with five faces

A domain is not a page. It is a workspace you can enter five ways — and they are the *same thing*, not five copies of it.

Face	What it is
agent	describe what you want, it builds
terminal	a real shell
code	a browser IDE
git	the repo and its provenance
site	the live URL

Move from one.ndexr.io to another.ndexr.io and you are working in seconds

Many addresses, one application

Every one of those domains — the marketing site, the customer portals, the internal tooling, the admin consoles — is the **same running Shiny application**.

- **n domains → 1 router → 1 app → n modules.**
- The router dispatches on the request, so what renders is decided by where you came in.
- Each site is a module inside that one app. Nothing more.
- One repository holds all of it. You add a module, not a codebase.

This is why the 205th domain costs what the 2nd did

What that consolidation is worth

Two hundred and five sites would normally be two hundred and five things to deploy, patch, monitor and remember.

- One deploy moves every site at once — no fleet drifting out of step.
- One dependency set to audit, not two hundred.
- A fix to shared behaviour is a fix everywhere, the same afternoon.
- The blast radius stays small anyway: each module owns its own boundary.

The whole estate is one thing to reason about, and one thing to secure

Immutable by design

Built from parts that don't drift.

- **Versioned** — every change is a commit; every state of the system is a point in history you can rebuild from exactly.
- **Continuous** — build, migrate, drain, rotate. The new version is up before the old one stands down.
- **Self-contained** — one module per host, each with its own boundary. Sites share no mutable state.

The foundation is compiled, not downloaded

The software underneath is built from source, in a fixed order, on a named toolchain — then distributed as one signed tree.

floor → build → distribute → catalogue → serve

- **The floor** — the compiler and C library are pinned first, so the toolchain is something we control rather than whatever the host happened to ship.
- **The build** — every layer above is compiled against that floor, in a fixed order, from source.
- **Distribution** — the result is published as a single signed tree.
- **The catalogue** — loading a specific, named build is one command.
- **The repository** — the validated R and Python binaries on top.

Both architectures, every layer

Each link in the chain is built for **x86_64** and **aarch64** — not one ported to the other afterwards.

- Both compiled on dedicated high-core-count machines, from the same sources.
- The same module tree mounts on a laptop and on a many-core cluster.
- No per-node installs, no rsync, no drift between machines.

Where it actually runs

- **HQ** — the control plane. Every `*.ndexr.io` request arrives here.
- **route** — dispatches each domain to its site.
- **Ways in** — an agent surface, a browser IDE, an HPC portal, a shell.
- **Foundation** — cloud provisioning underneath, and git for source and provenance beside it.

The front page renders this map live — every node is a real status read

The numbers

205domains live

28customer sites

24,924R packages built

- Counts, read from the production database as this was written.
- ndexr.io itself is one of the 205 — this deck is served by the system it describes.

Read from `domain_registry` and `r_site_library` on the production database. The front page renders the same counts live.

Why this makes large applications tractable

The hard part of a large system is not writing it. It is that nobody can hold it all, and nothing can be rebuilt exactly.

- Every piece is a module with a boundary — so a change has a blast radius you can name.
- Every dependency is pinned to the compiler floor — so "works on my machine" stops being a category of bug.
- Every change is a commit on an immutable base — so the question "what was running on Tuesday?" has an answer you can boot.

What reproducible actually buys you

- **Rebuild it** — same inputs, same system, on your hardware or ours.
- **Name it** — every layer has a version, down to the libc.
- **Move it** — the stack mounts on any Linux box that speaks the distribution tree.
- **Account for it** — designed for settings where every version has to be explained, not just recorded.

Agentic development, in a regulated setting

Most agentic tooling adds speed on top of whatever infrastructure it finds. The infrastructure is where the accountability has to live, so we started there.

What a regulated setting asks	Where it already comes from
What version is running?	Every layer pinned, down to the libc
Can you rebuild it exactly?	Same inputs, same tree, either architecture
Who changed it, and when?	Every change is a commit — nothing edited in place
Can we run it in our own walls?	The whole stack mounts on your hardware

The speed is the visible part. The foundation is the part that makes it defensible.

What is already proven

Not a plan. Every one of these is running now.

- The loop is real — this deck is served by the system it describes.
- The builds are reproducible, on both architectures, from source.
- The platform runs its own production on the same foundation it hands you.
- 205 domains, one application, one deploy.

The question stopped being whether this works. It works.

What you get

- An address of your own, and an agent behind it.
- A live site from the first session — not a mockup.
- Your code, your data, your history — exportable, rebuildable, yours.
- The same validated foundation we run production on.

Ask → live.
Same moment.

ndexr.io — this page is one of them