From Prompt to Production – Why AI Needs an Execution Layer, and Why It's Called Heim

AI now generates code faster than any human could possibly read it.
But one fundamental question remains unanswered:
What happens after the code is written?

This is a manifesto for a new layer in the digital infrastructure — one not built for humans, but for machines.
We call it **Execution-as-a-Platform**.

Here, we explain why it's needed, how it works — and why Heim is the first in the world to make it real.

1. CODE'S NEW SOURCE — AND ITS FORGOTTEN DESTINATION

We are in the midst of a paradigm shift.
Code is no longer written solely by humans — it is now generated by AI models in real time, on demand, directly from a prompt, in tools like Loveable, Cursor, and Firebase Studio — so-called AI-native IDEs.

What used to be craftsmanship has become a conversational flow:
"Build an API endpoint," "Deploy it to staging," "Create a dashboard for this" — and the AI obeys.

But even though the progress is revolutionary, one question remains unanswered — a critical one:
What happens after the code is written?

Behind the elegant interfaces lies a harsh reality:
Code that cannot be executed, scaled, or monitored is effectively useless — no matter how smart it is.

Today's AI IDEs carry a paradox:
They make it easier than ever to generate code — yet leave the user stranded when it's time to make that code actually work.

This is where the flow breaks.
Because even though AI can write the code, the architectural layer that actually runs it — securely, repeatably, and in real time — is still missing.

---

2. THE BOTTLENECK NO ONE TALKS ABOUT – THE CURSE OF EXECUTION

Code is just the beginning.

It's tempting to believe that AI replaces both developers and infrastructure — but that's a misconception.
Because once the code is generated, we hit a technological dead end — a bottleneck and systemic limitation that prevents AI from reaching all the way to production.

What it can generate, it still cannot run.

Sure — tools like the ones mentioned earlier can sometimes get code to run in a limited environment, like a frontend demo or a basic API.
But when it comes to real execution in production — with scaling, security, integration, and operations — AI is still powerless.

This is where an architectural limitation becomes clear:
AI can write the code — but the path from prompt to production still runs through solutions built for another era:

- Docker, Kubernetes, and YAML files
- CI/CD pipelines designed for manual interaction
- DevOps processes requiring specialist knowledge and ongoing maintenance

Most AI models have no memory structure for how code is deployed or executed.
They have to ask, again and again, how to configure, set up, and run the code.

And this has real consequences:

- Higher token usage → increased cost
- Slower response times
- More environment errors
- Lower margins

---

3. WHAT'S MISSING: AN EXECUTION LAYER

We're at a moment where everyone is racing toward AI-powered visions of the future.
But if the entire AI landscape rests on a foundation that still requires manual intervention at the end of every flow — it won't hold.
It will be expensive. It will be slow. It will be insecure.

AI needs a way to go from prompt to production — in a way that is secure, deterministic, and repeatable.
That requires something not yet built into IDEs, not present in DevOps tools, and definitely not inside the LLMs themselves.

An execution layer:

- that runs code without manual configuration
- that isolates each run using Zero Trust principles
- that eliminates the need for containers, Kubernetes, and Terraform
- that makes AI-generated code instantly production-ready

This is where Heim steps in — as an autonomous execution engine for AI-native development.
Heim was built for this exact world — not as a code generator, not as a tool, but as the very foundation for execution in AI workflows.
Heim is a self-orchestrating platform that acts as the missing execution layer in AI development.
It takes over where AI stops — and turns the prompt into production.

Heim eliminates the need for:

- YAML files
- CI/CD configuration
- Cloud-specific deployment tools

And replaces it with:

- Deterministic, policy-driven execution
- Automatic isolation and security
- Direct feedback to the AI IDE

Heim is not just a solution — it's an architectural shift.
It takes code from prompt to production — no DevOps, no containers, no complexity.
And it's no longer a vision.

It's code. It's a platform. It's reality.

But to truly understand what Heim is — and why it's even possible — we first need to understand the new category of infrastructure Heim belongs to.
A layer that didn't exist before.
A new foundation for AI-native systems:
Execution-as-a-Platform.

---

4. EXECUTION-AS-A-PLATFORM – HEIM AS A NEW ARCHITECTURAL CATEGORY

There's been a missing layer.
An infrastructure tier not built for humans — but for machines.

When an AI agent has written the code, tested it, and is ready to run it, there is currently no system designed for that flow.
Containers, CI/CD pipelines, and human DevOps processes were never meant for AI as the user.

The result? Friction, unnecessary costs, and security gaps — precisely where execution should be seamless.

This is where Execution-as-a-Platform (EaaP) enters.

What is EaaP?
EaaP is a new kind of infrastructure — an execution layer for AI-generated code.
It works automatically, deterministically, and without the need for manual configuration, DevOps tools, or containers.

Where traditional systems take us from **code to cloud**,
EaaP takes us from **prompt to production** — where a prompt doesn't just initiate code, but drives it all the way to a running, scalable application.

Why now?
Over the past decades, we've seen the rise of:

- **IaaS** – rent hardware in the cloud
- **PaaS** – build and deploy faster
- **FaaS** – run code without managing servers

But all of these platforms were built for humans.
Now, a shift is happening:

- AI writes the code
- AI tests the code
- AI wants to deploy the code

But the infrastructure for that doesn't exist.
And that's where Heim begins.

What does Heim do?
Heim is the first true EaaP system.
It receives code — directly or via an MCP server — and executes it as a live application in a secure, isolated, and fully auditable environment.

It requires no config files.
No containers.
No human instructions.

It's a platform **built for AI — not just built with AI.**

Traditional Infrastructure	**Execution-as-a-Platform (EaaP)**
Requires containers and CI/CD	Runs without containers or pipelines
Manual DevOps intervention	Automatic, zero-touch execution
External and slow runtime	Built-in, distributed, and AI-native

Difficult to isolate and secure	Zero Trust from the ground up
Designed for humans	Optimized for machine agents

Unlike IaaS, PaaS, or FaaS — which were defined retroactively through product adoption — Execution-as-a-Platform begins with intention. This manifesto sets the architectural and philosophical foundation for a new category: designed not for humans, but for machines.

---

5. TO UNDERSTAND WHY HEIM IS NEEDED — LOOK AT TODAY'S AI IDES

5.1 The Silent Drain of Token Costs

For AI IDE providers, high token usage isn't just a technical cost — it's a structural liability that silently eats into margins.

Let's take a closer look at AI IDEs and how they actually work in practice.
It's easy to forget that AI services aren't free — and tokens are the currency they run on.

What are tokens — and why is that a problem?
Tokens are the unit of measurement for how much "language work" an AI model performs. Every time you send a prompt to an LLM (Large Language Model), it processes that prompt in the form of tokens — small chunks of text.

Examples:

- "Deploy this app" → 4 tokens
- "How do I configure environment variables for this type of backend?" → 17 tokens
- "Can you deploy this microservice securely using a non-containerized setup with realtime monitoring?" → 24 tokens

It adds up quickly — especially when the AI can't remember earlier steps, and every task must be explained from scratch.

And all of this happens **because we still lack a dedicated execution engine.**

5.2 The Cost of Not Remembering

Token usage may seem trivial in small volumes — but it quickly grows into a hidden cost.
AI models like GPT-4, Claude, and Mistral charge per 1,000 tokens.

For example, GPT-4 pricing is roughly:

- $0.03–$0.06 per 1,000 tokens (output)
- $0.01–$0.03 per 1,000 tokens (input)

A typical AI-IDE user often generates 100–300 tokens per deploy attempt —
repeated over and over again because the model doesn't remember deploy structures, staging rules, or error messages.

At scale:
100,000 users × 10 deploys/day × 300 tokens
= 300 million tokens per day
= $15,000 per day — **in prompt cost alone**
→ Over **$450,000 per month** — just to ask the same things, again and again.

Why?
Because the AI model has no persistent memory of past runs.
Every prompt must re-establish the full context — every single time.

And many AI IDEs absorb these costs themselves in their "free tiers" — rapidly eroding their margins.

The result:
AI-native development flows that include code generation and deployment **become more expensive than what users pay for.**

It's a silent drain on the business model — and a growing barrier to profitability.

But inefficiency is only half the problem.
Because once AI doesn't just generate code but also executes it — the next question becomes:

How do we know we can trust what's being run?

---

6. SECURITY IN THE AI ERA – WHY HEIM IS ESSENTIAL

This is where the security dimension becomes critical.

Two recent sources show that we're already seeing the consequences of AI-driven code flows without control.

In the report *"Poison Everywhere"* from CyberArk, researchers describe how LLM-based development environments can be manipulated through **output poisoning** — where malicious code is hidden in the AI's response and later executed by an automated system.
Code that appears harmless may in fact contain backdoors, data leaks, or exfiltration commands — and since execution often happens without human oversight, it's not noticed until it's too late.

At the same time, an investigation by *Semafor* shows how this is already playing out in practice.
The article details how a popular AI IDE allowed over 170 user-built apps to expose API keys, user data, and other sensitive information — without protection or control.
According to the Replit engineer who first raised the alarm, the platform had **no visibility** into the code being deployed.

Both cases point to the same truth:
It's not enough for AI to generate code — we must have control over how that code is executed.

Heim as a Security Barrier

Heim is built to be secure by design.
Unlike Kubernetes-based systems, where security is often an afterthought, Heim is designed with **Zero Trust principles at every level**.

That means:

- **Isolated execution environments** — code runs in sandboxes that cannot communicate unless explicitly allowed by policy
- **No open ports** — no hidden or preconfigured networking
- **Real-time monitoring and feedback** — the person deploying gets immediate insight into what's happening

This security model is essential in environments such as:

- Financial systems
- Defense applications
- Regulated data flows (e.g. GDPR, HIPAA)

But everyone benefits from it.

Security no longer has to be bolted on or deferred — it's an embedded part of the infrastructure.

Heim does what CyberArk and Semafor are calling for:

1. **Packaging and isolation happen automatically.**
   Code never runs directly in the cloud.
2. **Deployments happen through policy-reviewed, context-aware flows.**
   Nothing goes live without validation.

Conclusion:
AI-first development creates new possibilities — but also new attack surfaces.
Heim is not just an execution layer.
It is the **architectural security barrier** required to make AI-era code flows sustainable in practice.

EaaP is here.
And **Heim is the pioneer** — patented, practical, and ready to carry this shift.

---

7. HOW HEIM WORKS – FROM PROMPT TO PRODUCTION

After identifying the cost and security issues in today's AI IDEs,
it's time to show how **Heim changes the game.**

7.1 Heim in the AI Flow – Alice Writes, Heim Executes

Heim acts as the intelligent execution layer at the bottom of AI-driven development flows.
Code generated in an AI-native IDE — is picked up by a local agent component, commonly

referred to as an MCP server (Model Context Protocol — a standardized interface for communication between models and tools).

The MCP server runs as an extension in the developer's environment and acts as a bridge between the IDE and Heim.
Heim takes over where AI stops — and turns the prompt into production.

Here's how Heim works in an AI-driven workflow

Alice is a developer using a modern AI IDE such as those mentioned earlier. She writes:
"Build a backend for managing users and deploy it."

The AI generates the code — and the local MCP server inside her IDE interprets the instruction. The MCP sends a command to Heim via CLI or API.

Then, the following happens:

1. The MCP server packages the code and sends a deploy command to Heim
2. Heim creates an isolated execution environment with Zero Trust defaults
3. The application is deployed to a scalable, distributed runtime — **without** containers, YAML files, or manual configuration
4. Heim monitors the application's state and returns logs and health status back to the IDE

Alice receives a response:

- **Deployed**
- **Logs available**
- **App health: OK**

She never had to configure pipelines, write YAML files, or understand cloud infrastructure.

Heim takes over exactly where AI leaves off — acting deterministically, securely, and predictably.
It makes the code runnable — so the AI flow becomes real.

7.2 MCP – Heim's Bridge to AI IDEs

In AI-native development environments, the **MCP server** acts as the local agent that translates AI-generated instructions into executable commands.
It is the **link between the IDE and the execution layer** — that is, Heim.

Heim implements its own MCP-compatible agent, designed to introduce isolation, determinism, security, and automated execution — capabilities often missing from generic implementations. This is not a new protocol, but a specialized integration of an established standard — built to enable secure AI workflows.

Where does it run?
The MCP server runs locally — typically as a plugin or background process inside the developer's IDE.

It is not cloud-based, but launched by the IDE itself.
Heim builds and provides the MCP server as part of its CLI and runtime package.

What does it do?

- Receives instructions from the AI (e.g., “deploy this app”)
- Interprets and translates them into executable commands
- Calls Heim via CLI or API
- Returns status, logs, and health data back to the IDE

Where does MCP’s responsibility end?
Neither MCP nor Heim generates the code — that’s the job of the AI IDE and its underlying LLM. MCP is activated only when the instruction needs to be translated into action, and Heim takes the next step — to execute the code securely, deterministically, and automatically.

Why is this important?
By integrating at the MCP level, Heim can:

- Work independently of any specific IDE or LLM
- Deliver deterministic and token-efficient execution
- Provide full control over infrastructure and security
- Make it easy for AI tools to plug Heim in as their execution engine

Heim begins where the prompt ends — and execution begins.

7.3 Before and After Heim – A New Kind of Control

When Heim is used as the execution layer, it doesn’t just change **how** code runs —
it transforms the entire experience of developing, deploying, and monitoring AI-generated software.

The table below shows how Heim replaces manual complexity with determinism, control, and efficiency:

Without Heim	**With Heim**
High token costs from repetitive deployments	Minimal token usage in the deployment phase
Insecure, manual, and error-prone deployments	Automated, deterministic, and secure execution
Costly and complex DevOps pipelines	Simpler operations via Heim’s platform layer
Limited visibility into runtime and health	Direct feedback, logs, and health indicators

---

8. CONCLUSION – FROM CODE TO CAPABILITY

We are entering a shift where AI doesn’t just assist development —
it drives it.

Code is being written faster than ever —
but still with old tools, old assumptions, and old risks.

What's been missing is an architectural response —
a way to take AI-generated code from idea to reality,
without compromising control, security, or scalability.

Several players are circling the same space — from Replit to Modal Labs —
but no one has yet built a complete, AI-native execution layer.
Heim is the first system to take code from prompt to production with no human intervention — with security and determinism at its core.

Execution-as-a-Platform is that answer.
And **Heim is the first concrete proof that it's possible.**

We've gone from vision to function:
– Patented architecture
– Real-world use cases
– Ready-to-run platform

Heim is not just a new kind of infrastructure.
It is **a new foundation for how software is created, executed, and secured in the AI era.**

Nimer Björnberg, Co-founder of NorNor, Örebro, Sweden – June 2025

Sources and References

- ***CyberArk, 2025*** *– Poison Everywhere: No Output From Your MCP Server Is Safe cyberark.com/resources/threat-research-blog/poison-everywhere-no-output-from-your-mcp-server-is-safe*
- ***Semafor, 2025*** *– The Hottest New Vibe-Coding Startup Is a Sitting Duck for Hackers semafor.com/article/05/29/2025/the-hottest-new-vibe-coding-startup-lovable-is-a-sitting-duck-for-hackers*