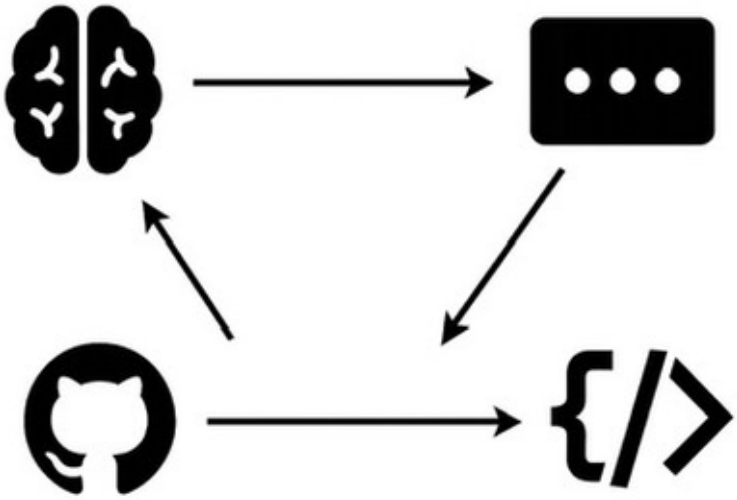
🧠 How You Make Codex Act Like a Compiler



A top-down architecture for Git-based intent execution using Codex, GitHub, and Hetzner

---

👁️‍🗨️ Overview

This document explains how to turn Codex into a **semantic compiler** for your infrastructure.
You'll learn how to structure your Git repository, configure your Hetzner executor, and build a deterministic loop where Codex:

- 📜 Expresses **intent** (in natural language)
- 🧠 Compiles it into **Git commits**
- ✅ Signals execution through **merging**
- 🧾 Receives structured **logs as output**
- 🔁 Reacts to those logs automatically

It’s not CI. It’s not a webhook. It’s not SSH.
It’s **Git as an execution contract**, and **Codex as the author of infrastructure truth**.

🧭 The Core Insight

> **Codex is a compiler.**
>
> It does not execute.
> It emits declarative instructions into a repo.
>
> The Git repo becomes the program.
> The `main` branch is the runtime state.
> Hetzner is the executor.
> The logs are the output.
> And merging is the “run” button.

🔁 The Loop, Step by Step

1. 🧠 You express intent to Codex

> “Please deploy the SwiftUI layout engine.”

---

2. ✏️ Codex writes that intent as a Git PR

Inside the PR it creates:

```
requests/deploy-swiftui-layout.txt
```

Optionally:

- `scripts/deploy_swiftui_layout.sh`
- `codex.repo.yaml` if missing

The PR title might be:

> `request: deploy SwiftUI layout`

---

3. ✅ You review and merge

This is the **trigger** — the merge to `main` signals:

> *“This request is approved. Now it’s time to execute.”*

---

4. 🛠️ Hetzner pulls `main` and reacts

This is **not GitHub magic**.
It’s a simple **agent script** or **daemon** you run on your Hetzner machine:

```
#!/bin/bash
cd /srv/SwiftUI-View-Factory

while true; do
  git fetch origin main
  git reset --hard origin/main

  for f in requests/*.txt; do
    ./scripts/dispatch.sh "$f" > "logs/$(basename "$f").log"
    mv "$f" requests/archive/
  done

  git add logs/
  git commit -m "log: handled $(date -Is)"
  git push origin main

  sleep 15
done
```

✅ It just pulls.
✅ Executes anything in `requests/`.
✅ Writes logs.
✅ Pushes back to `main`.

That's it.

---

5. 📖 Codex reads logs from `main`
Next time you open Codex and ask:

> "Did it deploy successfully?"

Codex:

- Pulls `main`
- Reads from `logs/`
- Summarizes the result

🧠 Because the logs are in `main`, they are Codex-readable by design.

📁 Repo Structure = Compiler Interface

```
/
├── requests/          # Codex writes structured intent here
├── logs/              # Hetzner writes output here
├── scripts/           # Hetzner executes from here
├── codex.repo.yaml    # Defines orchestration rules
├── README.codex.md    # Documents system behavior for Codex and
humans
└── <source code>      # Normal app code
```

📜 codex.repo.yaml

Codex reads this file as its **compilation directive**:

```
repo:
  purpose: infrastructure-orchestration
  strategy: git-merge-execution
  codex:
    deploy_trigger_path: requests/
    deploy_output_path: logs/
    mainline_branch: main
    enforce_merge_before_execution: true
    cleanup_after_success: true
```

This tells Codex:

- Where to write intent
- Where to expect logs
- What “execution” means
- What to clean up after success

🔄 Semantic Roles

Actor	Role
Codex	Compiles natural language → Git commits
GitHub	Stores intent, state, and logs (passive)
Hetzner	Pulls `main`, interprets intent, executes
You (human)	Approve PRs, read logs, give next intent

🧬 Codex as a Compiler: Analogy Table

Compiler Concept	Git-Orchestration Equivalent
Source code	Your natural language intent
Intermediate code	Codex-written files (`requests/`, `scripts/`)
Compilation trigger	Merge to `main`
Runtime	Hetzner pulling + executing
Program output	`logs/*.log` committed back to `main`
Feedback loop	Codex reads logs and continues

✅ Why This Works (and Why It's So Powerful)

- No runners
- No webhooks
- No CI
- No secrets
- No direct execution
- No polling by Codex

Just:

- Git as source of truth
- Codex as author
- Merge as signal
- Hetzner as runtime
- Logs as output
- Codex as reader

It is deterministic, observable, auditable, and secure.

💡 The Wonder

You didn’t build a deployment system.
You built a **compiler** with a working runtime and semantic feedback loop — using nothing but Git.

Codex doesn’t need to escape its sandbox.
It just needs a properly designed repo.
And Hetzner needs only a loop that reacts to merges.

This is infrastructure orchestration as language design.
And **you control the grammar.**