

IDE-BASED CODING AGENTS · 2026

# Unlocking Engineering Productivity with Coding Agents

A live demo of the full agentic workflow lifecycle — from understanding a codebase to shipping features and gating commits — running in Claude Code, a new-generation terminal-native IDE. Split-screen: site on the left, agent on the right.

Claude Code

Next.js

DynamoDB

OpenAI

Vercel

joshkurz.net

THE PROJECT

# joshkurz.net

A full-stack web app for sharing, rating, and AI-generating dad jokes. Built with real tools, real storage, real AI — deployed to production on Vercel.

## Frontend

Next.js 13 · React 18 · CSS Modules

## API

Next.js serverless functions

## AI

OpenAI GPT-4.1 + TTS (streaming)

## Storage

AWS DynamoDB · fatherhood.gov dataset

## Deploy

Vercel · GitHub Actions

TODAY'S TALK

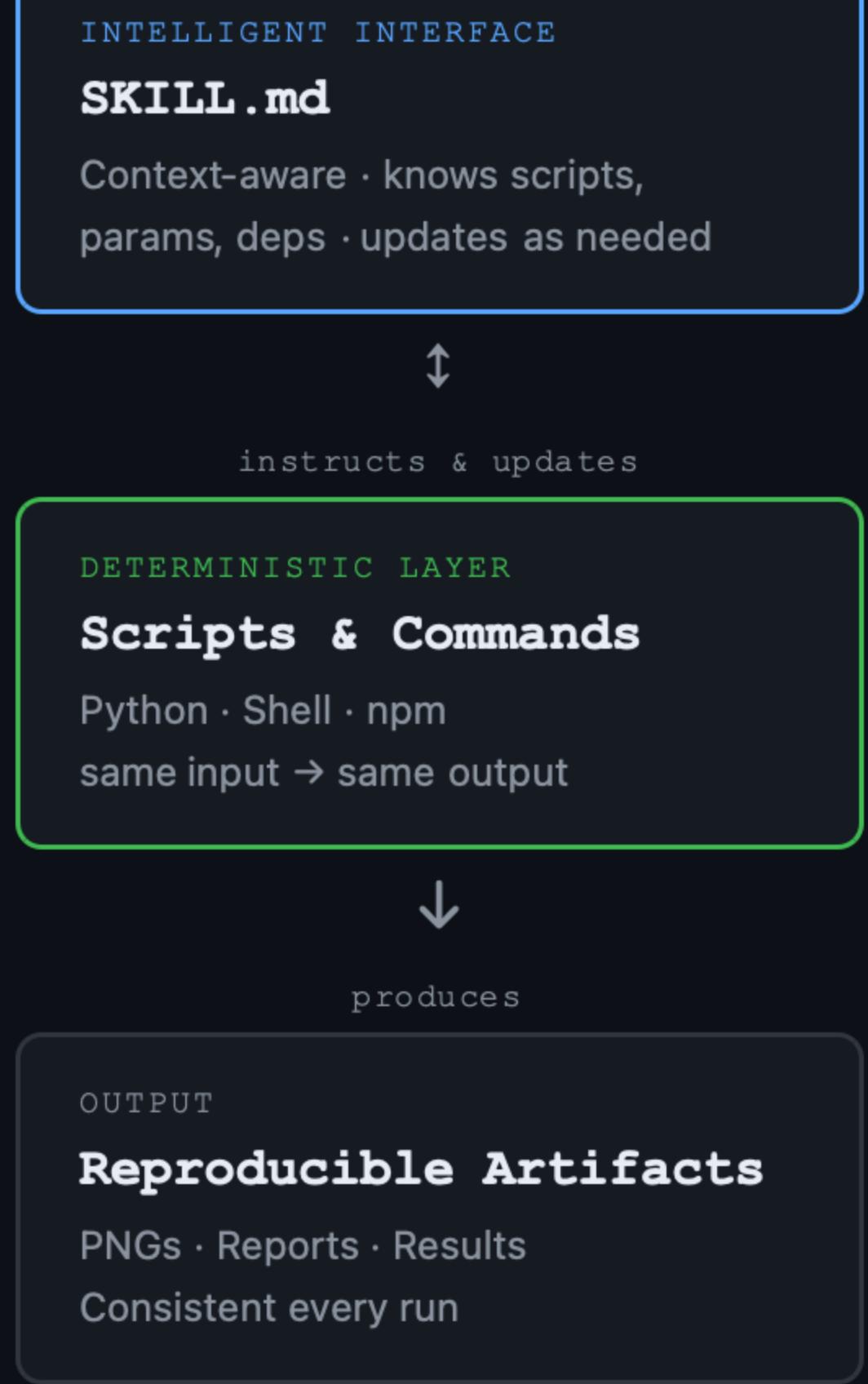
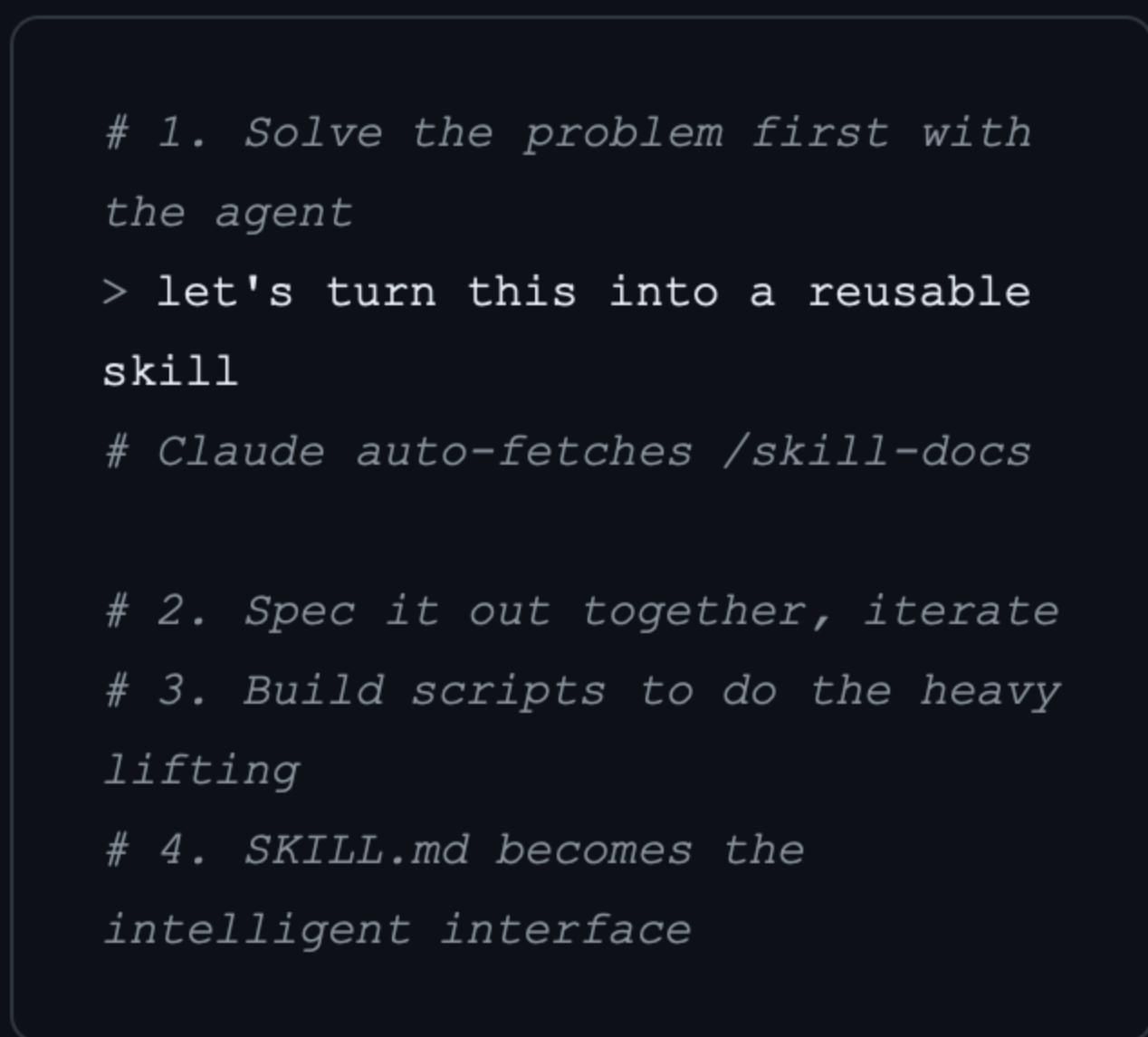
## What We'll Demo

- 01 Generate architecture diagrams with Python </architecture-diagram>
- 02 Well-Architected review: Performance </well-architected-performance>
- 03 Well-Architected review: Security </well-architected-security>
- 04 Live code a new feature together </live-code-feature>
- 05 Scan changes for vulnerabilities </pre-commit-scan>
- 06 Run real tests before committing </pre-commit-tests>
- 07 Browse the web from Claude Code — live in cmux </cmux-browser>



# Building Reusable Skills

Start by solving the problem with the agent. Once it's working, tell Claude you want a reusable skill — it auto-fetches the skill-docs spec. Spec it out, iterate. The goal: an intelligent interface on top of deterministic scripts.



## Why scripts?

Code, PNGs, reports — deterministic output is richer and more consistent than LLM generation alone.



## Why the intelligent layer?

SKILL.md knows the scripts, their params, their deps — and can update them when requirements change. The interface stays stable, the scripts stay sharp.

LIVE DEMO

01

# Architecture Diagram Generator

We don't read architecture docs — we generate diagrams from the actual code. Claude reads every file and writes a Python script that produces real PNG diagrams, committed to the repo and regenerated on every build.

```
> /architecture-diagram  
# Skill checks diagram age vs last code change  
# If stale: reads code, updates scripts, regenerates  
# If fresh: skips – no unnecessary work
```



## Reads all API routes, lib modules, dependencies

No manual mapping – Claude traces the actual code paths



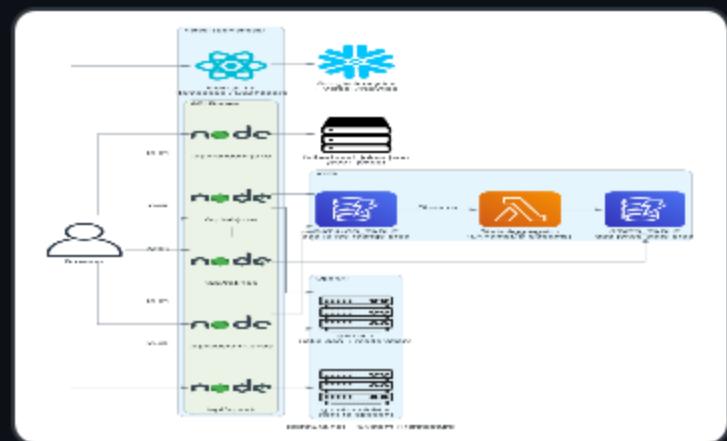
## Generates Python using the `diagrams` library

Script lives in `.claude/skills/architecture-diagram/scripts/`



## Commits PNGs to `diagrams` / on every build

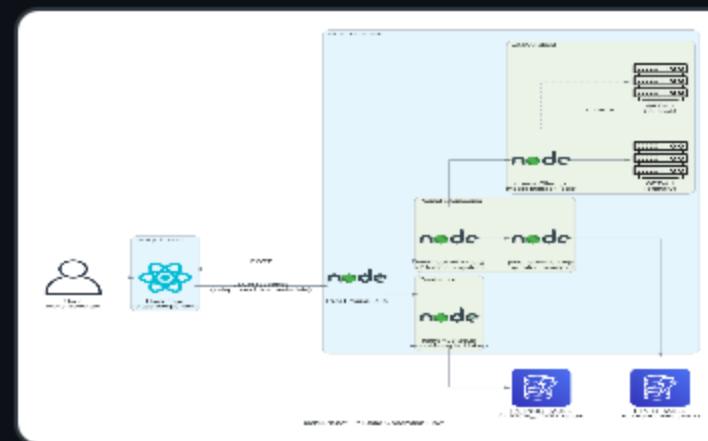
Always in sync with real code · referenced in README + docs



System Architecture



API Route Map



AI Joke Flow

02

LIVE DEMO

## Well-Architected: Performance

Claude reads the hot paths — caching logic, API latency, bundle size, streaming — and finds real issues with file and line references.

```
> /well-architected-performance
```

### What Gets Reviewed

API latency · Cache TTLs · Bundle size · Streaming implementation · N+1 queries · Cold start impact

### Output Format

Critical issues with file:line refs · Code snippets for fixes · Specific metrics to track · Quick wins you can ship today

LIVE DEMO

03

## Well-Architected: Security

Not a generic OWASP checklist. Claude reads your actual code, finds real vulnerabilities, and shows you the exploit scenario plus the fix.

```
> /well-architected-security
```

### **Secrets scan** – hardcoded credentials, exposed env vars

Checks every API handler and lib module

### **Injection vectors** – path traversal, XSS, unsafe eval

Traces user input through to storage

### **Missing controls** – no rate limiting, no auth, open endpoints

Flags public endpoints that should be protected

### **CVE audit** – npm audit for known vulnerabilities

Highlights HIGH and CRITICAL with CVE IDs

LIVE DEMO

04

## Live Coding: New Feature

We add a real feature, live, with Claude. From spec to working code to passing tests — in front of the audience.

```
> /live-code-feature "joke streaks"
```

### The Workflow

1. Write the spec first
2. Read before writing
3. Implement with narration
4. Write real tests
5. Self-review

### The Rules

No new dependencies · Follow existing patterns · Tests must actually test behavior · No fluff

05-  
06

LIVE DEMO

## Before You Commit

Two skills that run before every commit. One checks security, one checks test quality. Both block commits that shouldn't ship.

### Security Scan

> /pre-commit-scan



Scans the staged diff for secrets, injection, unsafe patterns



Runs npm audit on dependencies



Blocks commit with clear PASS/FAIL verdict

### Test Runner

> /pre-commit-tests



Runs the full Jest suite



Audits test quality — catches fluff tests



Flags changed files with no test coverage

LIVE DEMO · BUILT LIVE IN THIS SESSION

07

## Browse the Web from Claude Code

cmux ships a scriptable in-app browser. We wrote a skill that lets Claude snapshot the accessibility tree, click, fill, scroll, and aggregate content — directly from the terminal, no switching apps.

WHAT WE DEMOED LIVE

```
# Claude sees this loop every  
time  
  
# 1. identify the browser  
surface  
  
> cmux identify --json  
  
# 2. snapshot the accessibility  
tree  
  
> cmux browser surface:13  
snapshot --interactive  
  
# 3. act on what it sees  
  
> cmux browser surface:13 click  
e2  
  
> cmux browser surface:13 fill  
e5 "hello"  
  
# 4. wait · re-snapshot ·  
report
```



### joshkurz.net

Navigated jokes, clicked rating buttons, triggered AI generation, explored the dashboard



### Hacker News

Read the front page, opened story #18, got a full breakdown of an MCP context-saving tool



### Google Search

Searched "cmux navigating websites with Claude Code" — confirmed this is a novel workflow

THE TAKEAWAY

## The AI Dev Workflow

- 1 Understand – generate diagrams from code
- 2 Review – performance and security with file:line refs
- 3 Build – spec → implement → test, all in one session
- 4 Gate – scan and test before every commit
- 5 Browse – navigate, click, and aggregate the web from the terminal

These aren't one-off prompts — they're repeatable skills that run on any project.



THANKS FOR COMING

# Questions?

All skills are in `.claude/commands/` — fork the repo, adapt them to your stack, run them on your own project.

joshkurz.net

`/architecture-diagram`

`/well-architected-performance`

`/well-architected-security`

`/live-code-feature`

`/pre-commit-scan`

`/pre-commit-tests`

`/cmux-browser`