Dev Containers with Claude

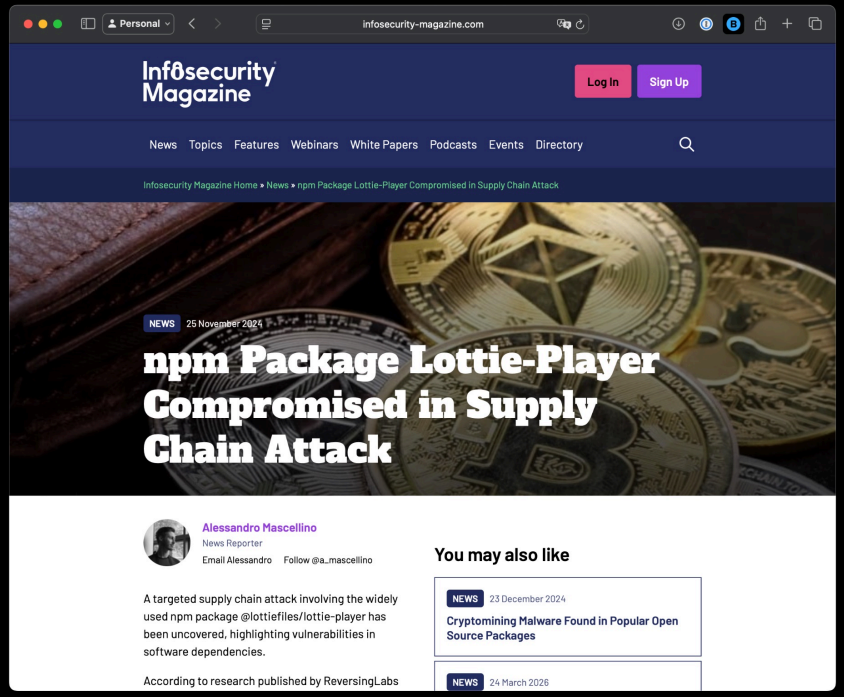
Security First

Serge Gatezh

Part 1

The Dev Container Journey

The Last Drop



The Last Drop

npm supply chain attacks kept escalating

Late 2024

- **Lottie Player (Oct 2024)** — maintainer token stolen, crypto wallet drainer injected. One user reportedly lost 10 BTC (~$723K).

2025

- **npm phishing campaigns (Sep 2024 onward)** — widespread phishing targeting maintainer accounts
- **chalk, debug & 16 others (Sep 2025)** — 18 packages, **2.6B weekly downloads** compromised
- **Shai-Hulud worm (Nov 2025)** — first **self-replicating** npm malware. CISA emergency alert.

What Is a Dev Container?

A **standardized development environment** that runs inside a Docker container, configured by a single JSON file in your repository.

```jsonc
// .devcontainer/devcontainer.json
{
  "name": "My Project",
  "image": "mcr.microsoft.com/devcontainers/base:ubuntu",
  "customizations": {
    "vscode": {
      "extensions": ["esbenp.prettier-vscode"]
    }
  }
}
```

Everyone who opens this project gets the **exact same environment** — same OS, same tools, same extensions.

Start Simple

My early attempts failed because I picked **Alpine Linux** in the VS Code wizard — without understanding how different it is.

What I didn't know

- Alpine uses **musl libc**, not glibc
- **Fixed package versions** per release
- VS Code wizard mostly supports **Debian**
- Many tools need custom install scripts

The lesson

Don't start with an unfamiliar base image just because it's small. **Start with what the tooling supports** — you can optimize later.

AI agents helped me finally debug the Alpine issues. But the better path was using a **Debian-based image** from the start.

```
# Alpine: custom scripts needed for specific versions
apk add hugo  # stuck with whatever version Alpine ships
```

What Changed This Time

AI-powered debugging

This time I had **AI agents** to help figure out what was going wrong with my Alpine setup.

They helped me understand the musl vs glibc issue, the package version constraints, and why the VS Code wizard defaults worked with Debian but not Alpine.

The result

I got my initial Alpine-based setup working — then later migrated to Bun's official image (Debian-based) as my projects grew more complex and needed more tooling.

```
1  FROM oven/bun:1.3.2-alpine
2
3  # Bun runtime + npm tools + Hugo for static site generation
```

Isolating `node_modules`

The reason I decided to do **all development** inside dev containers

```
// Keep node_modules out of a host machine
"mounts": [
  "source=${localWorkspaceFolderBasename}-node_modules,target=${containerWorkspaceFolder}/node_modules,type=volume
]
```

Why this matters

- `node_modules` stays inside the container — **never touches your host filesystem**
- No risk of malicious packages accessing your host machine
- The volume **persists across container rebuilds** — no reinstalling every time
- Cleaner project directory on your host

This single mount was the tipping point. Once I had this, there was no going back.

VS Code Extensions: Project-Scoped

One of the easiest and most impactful dev container customizations

```
"extensions": [
  // **Bun**
  "oven.bun-vscode",
  // Prettier - Code Formatter
  "esbenp.prettier-vscode",
  // **Tailwind**
  // Tailwind CSS IntelliSense
  "bradlc.vscode-tailwindcss",
  // Tailwind Fold
  "stivo.tailwind-fold",
  // **Hugo**
  // Hugofy - Hugo-related extension
  "akmittal.hugofy",
  // Language Hugo VSCode - Hugo language support
  "budparr.language-hugo-vscode",
  // Hugo Shortcode Syntax
  "kaellarkin.hugo-shortcode-syntax"
]
```

Each project gets **only the extensions it needs**. Faster VS Code startup. No leftover plugins from other projects.

The Version Sync Problem

Managing tool versions across multiple files

```jsonc
// .devcontainer/devcontainer.json
{
  "name": "gatezh.com",
  // ⚠️ Keep the same hugo version as in wrangler.jsonc
  "image": "ghcr.io/gatezh/devcontainer-hugo-bun:hugo0.152.2-bun1.3.2-alpine",
  "mounts": [
    "source=${localWorkspaceFolderBasename}-node_modules,target=${containerWorkspaceFolder}/node_modules,type=volu
  ]
}
```

```jsonc
// wrangler.jsonc
{
  // ⚠️ Keep the same hugo version as in .devcontainer/devcontainer.json
  "build": {
    "command": "hugo --minify"
  },
  "deploy": {
    "HUGO_VERSION": "0.152.2"
  }
}
```

GitHub Codespaces

Your dev container config works **everywhere**

📖 Demo: gatezh/blog @ dc480a8

- Same `devcontainer.json` runs locally **and** in GitHub Codespaces
- Cloud-based development with your exact environment
- Great for contributors — zero local setup needed

💡 **Lesson learned:** Codespaces taught me that **different architectures use different packages**

My Apple Silicon Mac builds ARM64 images. Codespaces runs on AMD64. Things that work locally can break in Codespaces because the packages are different.

```
# Only builds for YOUR platform
docker build .

# Builds for BOTH — needed for Codespaces compatibility
docker buildx build --platform linux/amd64,linux/arm64 --push .
```

Prebuilt Images for Speed

The journey

Building from a Dockerfile **every time you open a container** is slow — especially for larger images.

But the real motivation was when I decided to **use the same dev container setup across all my projects**. Instead of copying Dockerfiles everywhere, I could maintain **one image** that serves all projects.

The options

Each GitHub repository can have its own **private prebuilt images** hosted alongside the repo. But in my case, it's easier to maintain **one shared image** in a dedicated repository.

```
// Any project's devcontainer.json — just pull the shared image
{ "image": "ghcr.io/gatezh/devcontainers/claude-code:latest" }
```

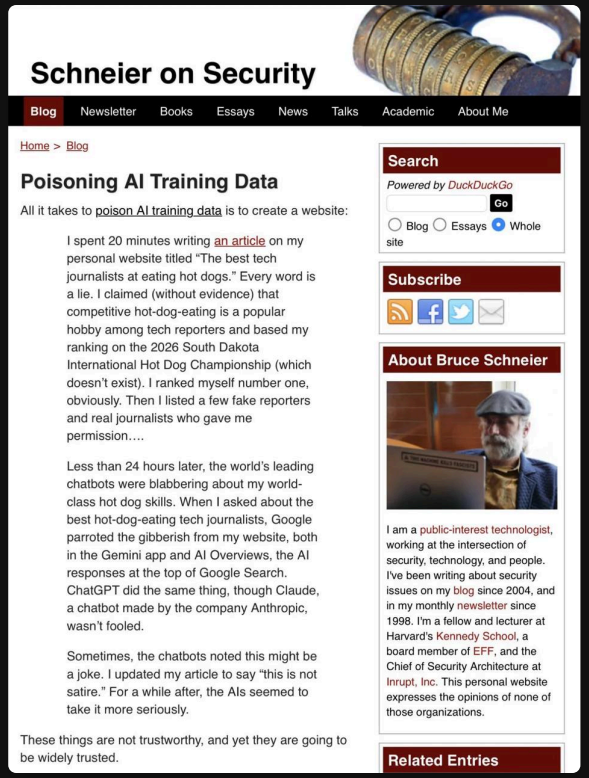
The workflow

GitHub Actions automatically builds and pushes multi-platform images whenever I update the Dockerfile. I can even bump tool versions **from the GitHub UI** via `workflow_dispatch`.

Part 2

Secure Claude Code Execution

Why Isolate Claude Code?



Schneier on Security

Blog Newsletter Books Essays News Talks Academic About Me

Home > Blog

Poisoning AI Training Data

All it takes to poison AI training data is to create a website:

> I spent 20 minutes writing an article on my personal website titled "The best tech journalists at eating hot dogs." Every word is a lie. I claimed (without evidence) that competitive hot-dog-eating is a popular hobby among tech reporters and based my ranking on the 2026 South Dakota International Hot Dog Championship (which doesn't exist). I ranked myself number one, obviously. Then I listed a few fake reporters and real journalists who gave me permission….
>
> Less than 24 hours later, the world's leading chatbots were blabbering about my world-class hot dog skills. When I asked about the best hot-dog-eating tech journalists, Google parroted the gibberish from my website, both in the Gemini app and AI Overviews, the AI responses at the top of Google Search. ChatGPT did the same thing, though Claude, a chatbot made by the company Anthropic, wasn't fooled.
>
> Sometimes, the chatbots noted this might be a joke. I updated my article to say "this is not satire." For a while after, the AIs seemed to take it more seriously.

These things are not trustworthy, and yet they are going to be widely trusted.

Search

Powered by DuckDuckGo

Go

Blog Essays Whole site

Subscribe

About Bruce Schneier

I am a public-interest technologist, working at the intersection of security, technology, and people. I've been writing about security issues on my blog since 2004, and in my monthly newsletter since 1998. I'm a fellow and lecturer at Harvard's Kennedy School, a board member of EFF, and the Chief of Security Architecture at Inrupt, Inc. This personal website expresses the opinions of none of those organizations.

Related Entries

Why Isolate Claude Code?

AI agents can run **arbitrary commands** on your machine. They can install packages, modify files, execute scripts, and make network requests.

This is the **same class of risk** as the npm supply chain attacks we just talked about — except it comes from a tool you're actively using.

> How do I get the productivity benefits of Claude Code without risking my host machine?

Anthropic has an **official dev container configuration** for exactly this.

The Layered Security Model

🐋

Layer 1

Docker Container

Isolates from host OS

✋

Layer 2

Permission Prompts

User approves each action

🔥

Layer 3

Network Firewall

Blocks data exfiltration

🔒 Sandbox Dev Container

Layer 1 + Layer 3

- Claude runs autonomously
- Firewall prevents exfiltration
- Whitelisted domains only

🛠️ Default Dev Container

Layer 1 + Layer 2

- You review every command
- No firewall needed — **you are the firewall**
- Full internet access

Your Default Devcontainer Is Already Good

With `dangerously-skip-permissions` and no firewall:

- Claude runs commands without asking → **fast, autonomous workflow**
- Full internet access → docs, npm, APIs, blogs
- Docker isolation protects your host machine

The residual risk

Data exfiltration **from within the container** — but the container only has what you mount into it.

When to upgrade to the sandbox

- Working with sensitive code or credentials
- Running on untrusted repositories
- Want defense-in-depth beyond Docker isolation

Anthropic’s Official Config

📖 code.claude.com/docs/en/devcontainer

Three files in `anthropics/claude-code/.devcontainer/`:

- `devcontainer.json` — settings, extensions, volume mounts, session persistence
- `Dockerfile` — Node.js 20, zsh, git-delta, Claude Code CLI
- `init-firewall.sh` — iptables rules: whitelist npm, GitHub, Anthropic API; default-deny everything else

Key features

- Whitelisted outbound connections only (default-deny policy)
- **Session persistence** — bash history and `.claude` config stored in named volumes, survive rebuilds
- Startup verification of firewall rules
- `NET_ADMIN` + `NET_RAW` capabilities for iptables

⚠️ Note: the official config hasn’t been updated in ~7 months. Some aspects may be outdated.

Claude Code CLI Authentication

⚠️ **Gotcha:** Claude Code VS Code extension works fine in the container, but the **CLI doesn't authenticate the same way**

The workaround

```
# 1. Generate a setup token (long-lived, ~1 year expiry)
#    via Claude Code settings / API

# 2. Set as environment variable in devcontainer.json
"containerEnv": { "CLAUDE_CODE_SETUP_TOKEN": "${localEnv:CLAUDE_CODE_SETUP_TOKEN}" }
```

⚠️ **Another gotcha:** Even with the token, Claude Code forces you through the **first-time onboarding wizard**. You need a specific setting to **skip the onboarding** — then it works.

What I Built on Top (1/2)

Evolution from the official config → my `claude-code` image

Base & runtime

- **Node.js 24 LTS (slim)** base image
- **Fish shell** with custom theme
- **mise** for optional tool versions (Bun, Hugo, etc.)

Developer experience

- **Prebuilt images** — daily rebuilds with latest Claude Code
- **Custom VS Code theme** — visual distinction for sandbox
- **Playwright** setup for headless testing

What I Built on Top (2/2)

Security & isolation

- **node_modules as volume** — isolated from host + persists across rebuilds
- `dangerously-skip-permissions` enabled as VS Code setting (sandbox only)
- `sandbox-fetch-docs` SKILL — makes Claude aware of the restricted environment

Claude Code enhancements

- **RTK** (Rust Token Killer) — filters command output to save tokens
- **Ralphex** — Ralph loop implementation for structured task execution
- **Per-project plugin installation script** — each container gets only the MCP plugins it needs, no host config mounting

Two Container Variants

🔒 Claude Code Sandbox

- AI-powered development
- Fully network-isolated
- Dev server NOT accessible
- **Claude Code VS Code theme** 🎨
- `dangerously-skip-permissions`

🛠️ Default Dev Container

- Standard dev environment
- Run dev server locally
- Ports exposed to host
- **Default VS Code theme**
- Permission prompts ON

Visual distinction matters — when running multiple containers for the same project, the VS Code theme tells you instantly which is which.

Both variants come from the **same multi-stage Dockerfile** in my repo.

The `claude-code` Image

github.com/gatezh/devcontainers/claude-code

Shared base image for all Claude Code projects. **Two variants** from a single multi-stage Dockerfile.

```
claude-code/
├── .devcontainer/
│   ├── Dockerfile             ← multi-stage: default + sandbox targets
│   ├── devcontainer.json      ← default variant config
│   └── claude-sandbox/
│       ├── devcontainer.json  ← sandbox variant config
│       └── init-firewall.sh   ← iptables whitelist rules
└── README.md
```

```jsonc
// Default variant
{ "image": "ghcr.io/gatezh/devcontainers/claude-code:latest" }

// Sandbox variant (needs NET_ADMIN for iptables)
{ "image": "ghcr.io/gatezh/devcontainers/claude-code-sandbox:latest",
  "capAdd": ["NET_ADMIN", "NET_RAW"] }
```

Rebuilds **daily** via GitHub Actions to pick up latest Claude Code.

Ralphex

Ralph loop implementation for **structured Claude Code task execution**

📖 ralphex.com/docs/#using-docker

I use Ralphex via its **Docker wrapper script** — it runs alongside Claude Code inside the dev container.

The Ralph loop approach breaks complex tasks into structured steps with better context management, leading to more reliable and predictable outcomes from AI agents.

The `ralphex-fe` Image

github.com/gatezh/devcontainers/ralphex-fe

Standalone Docker image (not a devcontainer) — **Bun** + **Hugo Extended** + **Chromium** on the Ralphex base.

```
ralphex-fe/
├── Dockerfile
└── README.md
```

Built for modern frontend development, static site generation, and end-to-end testing.

Used as a base for projects that need the Ralphex runtime with frontend tooling baked in.

Key Takeaways

1. **Start small** — a basic `devcontainer.json` with an image and a few extensions is enough
2. **Add tooling as you go** — when you start missing a tool, add it to the config
3. **Consider prebuilt images** when you want to reuse the same setup across projects
4. **Dev containers are already significant isolation** for AI agents — even without a firewall
5. **The sandbox adds defense-in-depth** when you need it — network isolation via iptables whitelist
6. **Everything is public** — clone the repo, adapt it, make it yours

Resources

My repo	github.com/gatezh/devcontainers
Blog	gatezh.com
Official Claude Code devcontainer	github.com/anthropics/claude-code
Claude Code docs	code.claude.com/docs/en/devcontainer
Dev Containers spec	containers.dev

🙏 Thank you!