OPEN HARNESSES SERIES • EXTENSIONS & AUTOMATION GUIDE

OpenCode Plugins, Custom Tools, Servers & CI

Extend OpenCode from an interactive terminal into a programmable engineering platform—authoring Zod-typed `.opencode/tools/` modules, intercepting tool calls via `@opencode-ai/plugin` hooks, enforcing `permission` firewalls, running headless `opencode serve` daemons, and automating PRs in GitHub Actions.

Typed `.opencode/tools/` | `@opencode-ai/plugin` Hooks | `allow / ask / deny` Policy | Local & Remote MCP | `opencode serve` & SDK | GitHub Actions `/oc` CI



In-Process Plugins & Tools

Drop TypeScript files into `.opencode/tools/` and `.opencode/plugins/` to add domain tools and lifecycle guards instantly.

Deterministic Safety Gates

Combine glob-matched `permission` rules in `opencode.json` with `tool.execute.before` secret and shell shields.

Headless Daemons & CI

Drive remote sessions over HTTP via `opencode serve`, `opencode attach`, `@opencode-ai/sdk`, and `/opencode` PR triggers.

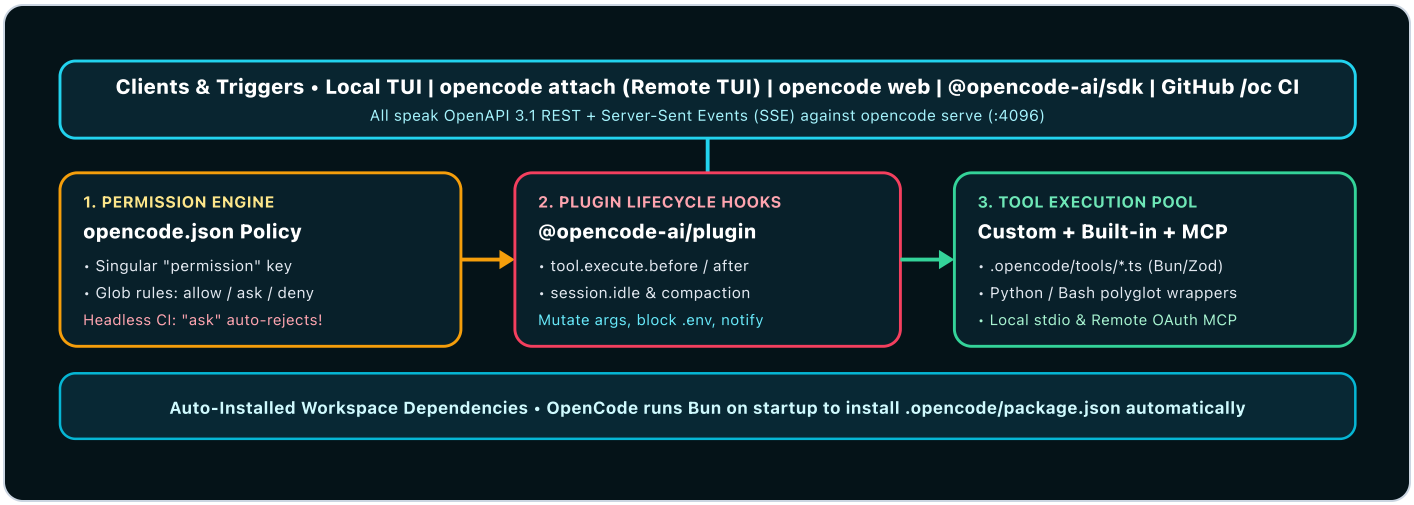
Programmable Inside and Out: Tools, Hooks, MCP & Headless HTTP

Every OpenCode session runs on an HTTP/SSE server engine under the hood. That architecture lets you extend the agent at four clean layers: **custom TypeScript tools**, **event & interceptor plugins**, **external MCP servers**, and **headless CLI / SDK automation**.

Quick Navigation

1. Four-Layer Extension Model p. 2
2. `.opencode/` Layout & Decision Matrix p. 3
3. Annotated Server & Plugin Tour p. 4
4. Authoring Custom Tools (Zod) p. 5
5. Plugins, Hooks & Permission Gates p. 6
6. Troubleshooting & Pre-Flight p. 12
7. MCP Servers & OAuth Bridges p. 7
8. Headless `serve`, `run` & CI p. 8
9. 5 Copy-Paste Production Recipes p. 9
10. Server Hardening & SDK Control p. 10
11. Cross-Tool Synergy Pipeline p. 11

Architecture: How Custom Tools, Lifecycle Hooks & Headless Clients Interlock



Zero-Build Extension Ergonomics

Drop any TypeScript file into `.opencode/tools/` or `.opencode/plugins/` (or global `~/.config/opencode/`). OpenCode's embedded Bun runtime compiles TypeScript on the fly and installs any npm packages declared in `.opencode/package.json` automatically at startup.

The `.opencode/` Directory Map & Extension Decision Matrix

Every customization in OpenCode has a dedicated folder in your project root (shared with teammates via Git) and a matching personal folder under `~/.config/opencode/` (shared across all your repos).

1 Standard `.opencode/` Project Layout

```
my-repo/
├── opencode.json              # Models, permission,
mcp, plugin[]
├── AGENTS.md                  # Repo architecture &
test rules
└── .opencode/
    ├── package.json           # Shared npm deps for
tools/plugins
    ├── tools/                 # Custom LLM-callable
Zod tools
    │   └── query-metrics.ts
    ├── plugins/               # Lifecycle &
security hook modules
    │   └── env-shield.ts
    ├── agents/                # Specialist Markdown
subagents
    └── commands/              # Reusable /slash
prompt templates
```

2 Loading npm Plugins vs. Local Files

Local `.ts` / `.js` files in `.opencode/plugins/` auto-load without config. Published npm plugins are listed in `opencode.json` and cached automatically:

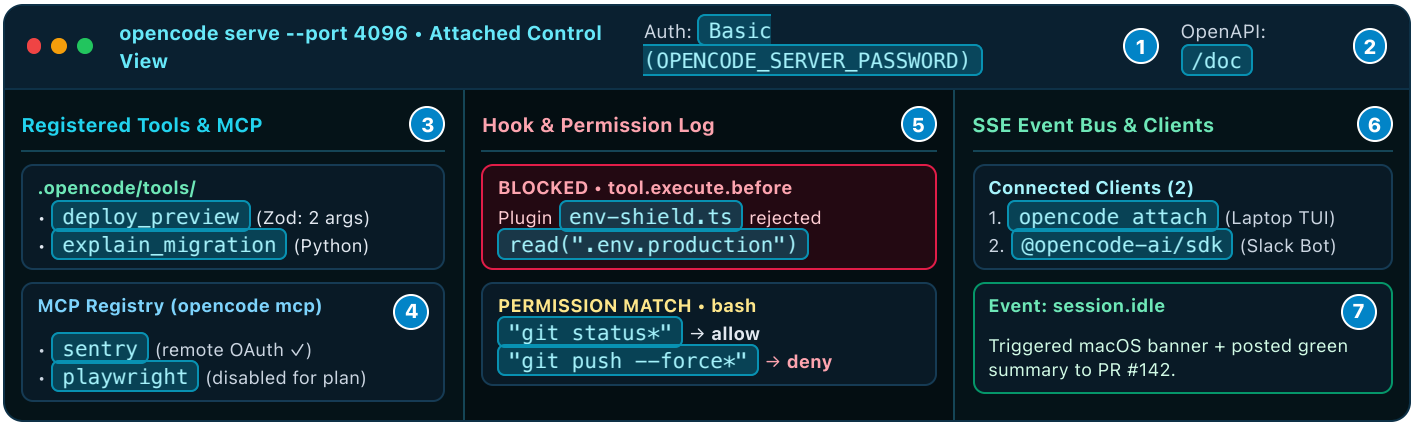
```
{
  "$schema": "https://opencode.ai/config.json",
  "plugin": [
    "opencode-devcontainers",
    "@my-org/opencode-audit-plugin@1.2.0"
  ]
}
```

Extension Mechanism Decision Matrix: Pick the Simplest Primitive

Primitive	Where It Lives	Use It When You Want To…	Avoid Using It For…
Slash Command	`.opencode/commands/*.md`	Trigger a repeatable prompt macro manually (e.g., `/changelog`, `/pr-check`)	Tools the LLM must decide to invoke mid-turn on its own
Custom Tool	`.opencode/tools/*.ts`	Give the LLM a repo-specific typed action (query staging DB, run Python script)	Cross-app SaaS integrations that already ship an official MCP server
Plugin Hook	`.opencode/plugins/*.ts`	Enforce security blocks, send idle notifications, inject env vars, or customize compaction	Simple static prompt rules that belong in `AGENTS.md`
MCP Server	`opencode.json` → `"mcp"`	Connect multi-tool ecosystems (Sentry, Linear, GitHub, Playwright, Postgres)	Enabling 40+ unused tools globally that bloat every turn's token budget

Annotated Tour: Inspecting Plugins, Permissions & `opencode serve`

When running OpenCode as a persistent daemon (`opencode serve` or `opencode web`) with attached TUI/SDK clients, every plugin interception, MCP OAuth status, and permission gate surfaces deterministically.



Control Legend: Seven Critical Automation Surfaces

1 HTTP Basic Auth Guard: Setting `OPENCODE_SERVER_PASSWORD` protects headless `opencode serve` and `opencode web` ports from unauthorized LAN callers.

2 Live OpenAPI 3.1 Spec (`/doc`): Serves machine-readable schemas for sessions, messages, files, and SSE streams at `http://127.0.0.1:4096/doc`.

3 Custom Tool Discovery: Each export in `.opencode/tools/*.ts` becomes a typed tool (`<file>_<export>`) automatically validated via Zod.

4 Per-Agent MCP Toggles: Manage remote OAuth via `opencode mcp auth <name>` and enable heavy MCP suites only on specialist subagents.

5 Dual-Layer Security Gate: Declarative `permission` globs handle 90% of rules; imperative `tool.execute.before` hooks inspect dynamic arguments.

6 Multi-Client Attach & 7 `session.idle`: Drive one remote devbox from TUI, browser, or SDK simultaneously and fire completion hooks when work finishes.

Authoring Custom Tools with `@opencode-ai/plugin` & Bun

Custom tools turn tribal shell scripts into first-class typed capabilities the LLM can discover and invoke safely. Use the `tool()` helper with built-in `tool.schema` (Zod) so you never have to manage duplicate Zod versions.

1. Single-File Tool (`.opencode/tools/db.ts`)

```
import { tool } from "@opencode-ai/plugin"

export default tool({
  description: "Run read-only EXPLAIN on staging
Postgres",
  args: {
    sql: tool.schema.string().describe("SELECT query"),
  },
  async execute(args, context) {
    if
(!args.sql.trim().toUpperCase().startsWith("SELECT")) {
      return "Rejected: only SELECT queries allowed."
    }
    const out = await Bun.$`psql "$STAGING_RO_URL" -c
"EXPLAIN ${args.sql}"`.text()
    return `Directory: ${context.directory}\n${out}`
  },
})
```

2. Wrapping Python or Go CLI Scripts

```
// .opencode/tools/forecast.ts -> tool name: "forecast"
import { tool } from "@opencode-ai/plugin"
import path from "path"

export default tool({
  description: "Score latency CSV with Python
statsmodels",
  args: {
    csvPath: tool.schema.string(),
    horizon: tool.schema.number().default(24),
  },
  async execute(args, ctx) {
    const script = path.join(ctx.worktree,
"scripts/score.py")
    return await Bun.$`python3 ${script} ${args.csvPath}
--h ${args.horizon}`.text()
  },
})
```

Key Tool Naming, Override & Context Rules

1 Naming Convention

`default` export in `db.ts` registers as tool `db`. Named exports `export const seed` and `export const check` register as `db_seed` and `db_check`.

2 Rich `context` Object

Every `execute(args, context)` call receives `context.directory`, `context.worktree`, `context.agent`, `context.sessionID`, and `context.abort`.

3 Built-In Overrides

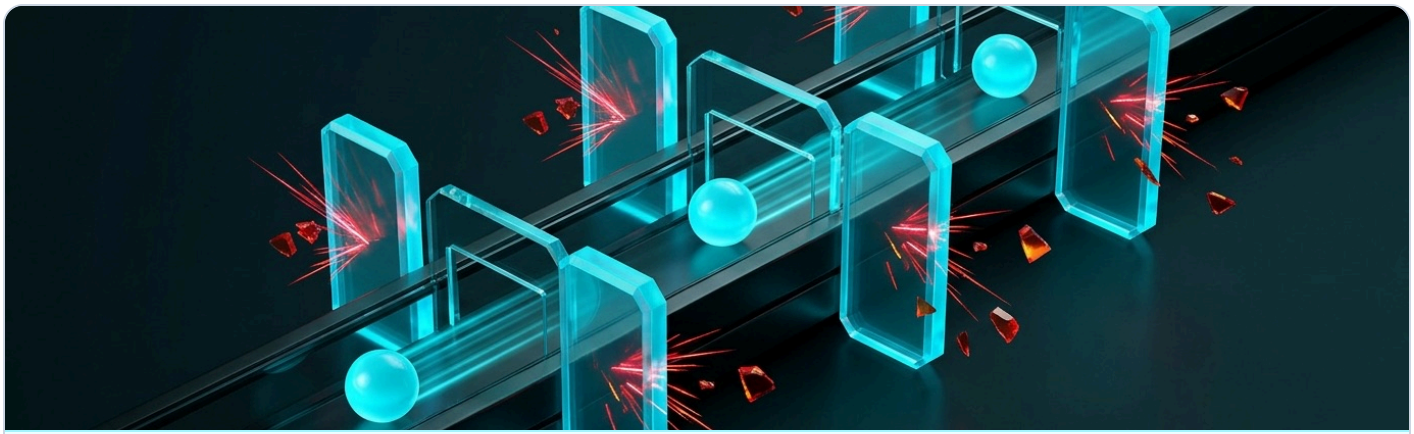
Naming a custom tool `grep` or `bash` overrides OpenCode's built-in tool of the same name—handy when enforcing custom sandboxed runners.

Keep Tool Returns Compact & Structured

Never return 50,000 raw log lines from `execute()`. Filter, slice, or summarize bulky CLI output inside your TypeScript wrapper and write full dumps to a temp file path if the agent needs deeper drill-down.

Defense in Depth: Plugin Interceptors & `permission` Firewalls

Prompt instructions alone cannot stop a confused or prompt-injected model from reading `.env` secrets or running destructive shell flags. Combine OpenCode's declarative `permission` globs with hard TypeScript plugin hooks.



Deterministic Gatekeeping: Safe read/test commands glide straight through `"allow"` gates while destructive shell patterns and secret-file reads shatter at the `permission` and `tool.execute.before` barrier.

1. Singular `permission` Key in `opencode.json`

```
{
  "permission": {
    "edit": "allow",
    "bash": {
      "*": "ask",
      "pnpm test*": "allow",
      "git status*": "allow",
      "git diff*": "allow",
      "rm -rf *": "deny",
      "git push --force*": "deny"
    },
    "read": { "*": "allow", "*.env*": "deny" }
  }
}
```

2. Plugin Shield (`.opencode/plugins/guard.ts`)

```
import type { Plugin } from "@opencode-ai/plugin"

export const SecurityGuard: Plugin = async ({ $ }) => ({
  "tool.execute.before": async (input, output) => {
    if (input.tool === "read" &&
/\.pem$|id_rsa/.test(output.args.filePath)) {
      throw new Error("Blocked private key read by
SecurityGuard")
    }
  },
  event: async ({ event }) => {
    if (event.type === "session.idle") {
      await $`osascript -e 'display notification "Done!"
with title "OpenCode"'`
    }
  },
})
```

Color-Coded Blueprint: Anatomy of a Hardened Automation Config

Expose read-only domain scripts via .opencode/tools/*.ts, declare singular permission rules allowing pnpm test* and denying *.env* or git push --force*, enforce key-file blocks and compaction context injection in .opencode/plugins/guard.ts, protect opencode serve behind OPENCODE_SERVER_PASSWORD on Tailscale, and set non-interactive CI runs to explicit allow/deny so opencode run never stalls on ask.

Typed Custom Tools | Declarative Permission | Lifecycle Hooks | Authenticated Daemon | Headless CI Policy

Wiring Local & Remote MCP Servers Without Token Bloat

OpenCode supports both local subprocess MCP servers (`"type": "local"`) and remote HTTP/SSE MCP endpoints with automatic OAuth 2.1 browser flows (`"type": "remote"`)—plus per-agent glob controls so heavy tool schemas only load when needed.

1. Local + Remote MCP in `opencode.json`

```
{
  "mcp": {
    "playwright": {
      "type": "local",
      "command": ["npx", "-y",
"@playwright/mcp@latest"],
      "enabled": true
    },
    "sentry": {
      "type": "remote",
      "url": "https://mcp.sentry.dev/mcp",
      "oauth": {}
    }
  },
  "tools": { "playwright_*": false },
  "agent": {
    "qa-browser": { "tools": { "playwright_*": true } }
  }
}
```

2. Managing MCP Auth & Health via CLI

```
# Interactive wizard to register a new MCP server
opencode mcp add

# Check connection health & discovered tool counts
opencode mcp list

# Complete OAuth browser login for remote servers
opencode mcp auth sentry

# Diagnose OAuth token / transport failures
opencode mcp debug sentry
```

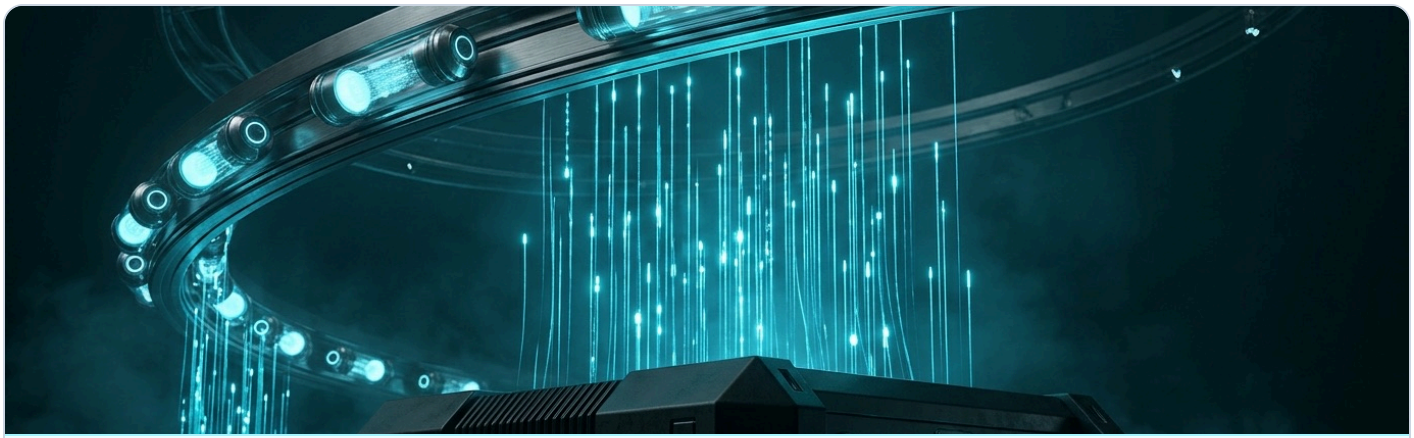
Token-Saving Pattern: Notice how `"playwright_*": false` disables 25+ browser tool schemas globally while re-enabling them solely inside the `qa-browser` subagent!

Custom Compaction Hook Keeps Critical Context Alive

Long MCP debugging sessions eventually trigger compaction. Implement `"experimental.session.compacting"` inside a plugin to push must-keep invariants (active ticket ID, staging URL, test command) into `output.context` so compacted summaries never drop operational state.

Headless Execution: `opencode serve`, `run` & GitHub Actions

Decouple your local laptop from heavy builds by running `opencode serve` on a cloud devbox, scripting one-shot pipelines via `opencode run`, and triggering autonomous PR fixes from `/oc` comments in GitHub Actions.



Always-On Automation Loop: Issue comments, cron triggers, and remote TUI clients dispatch tasks into headless OpenCode workers that run tests in clean containers and push verified commits back to the PR branch.

1. Remote Devbox Daemon + Local Attach

```
# On cloud VM / GPU box (protected over Tailscale)
export OPENCODE_SERVER_PASSWORD="strong-random-secret"
opencode serve --hostname 0.0.0.0 --port 4096

# From lightweight laptop: attach full native TUI
opencode attach http://devbox.tailnet:4096

# One-shot non-interactive CLI / shell pipe
git diff | opencode run "Summarize breaking API changes"
\
  --format json
```

2. GitHub Actions Bot (`opencode github install`)

```
# .github/workflows/opencode.yml
on:
  issue_comment: { types: [created] }
  pull_request_review_comment: { types: [created] }
jobs:
  opencode:
    if: startsWith(github.event.comment.body, '/oc')
    runs-on: ubuntu-latest
    permissions:
      contents: write
      pull-requests: write
      issues: write
      id-token: write
    steps:
      - uses: actions/checkout@v4
      - uses: anomalyco/opencode/github@latest
        env: { ANTHROPIC_API_KEY: ${{
secrets.ANTHROPIC_API_KEY }} }
        with: { model: anthropic/claude-sonnet-4-5 }
```

5 Real-World Copy-Paste Recipes

Drop these five snippets into your repo, CI workflow, or TypeScript service to add custom tools, shell env injection, SDK orchestration, remote TUI attachment, and scheduled repository sweeps.

Recipe & Goal	Copy-Paste Snippet / Command	Expected Outcome
1. Inject Env Vars Into Every Agent Shell	Save `.opencode/plugins/shell-env.ts`: `export const CiEnv = async () => ({ "shell.env": async (_in, out) => { out.env.CI = "1"; out.env.PAGER = "cat"; } })`	Prevents interactive pagers (`less`, Watch-mode Vitest) from hanging agent bash calls.

2. Programmatic Session Control via TypeScript SDK	`import { createOpencodeClient } from "@opencode-ai/sdk"`
	`const client = createOpencodeClient({ baseUrl: "http://127.0.0.1:4096" })`
	`const s = await client.session.create({ body: { title: "Auto Triage" } })`
		Lets Slack bots, internal dashboards, or orchestrators spawn and stream OpenCode sessions.

3. One-Shot Prompt Against Running Daemon	`opencode run --attach http://127.0.0.1:4096 "Run unit tests for billing and report failures"`	Skips MCP/LSP cold-start boot times by reusing the warm `opencode serve` process.

4. Install Official GitHub App & Workflow	`opencode github install`
	Then comment on any Issue or PR:
	`/oc fix the race condition on line 84 and add a regression test`
		Configures GitHub App OIDC + workflow YAML so `/oc` or `/opencode` comments open or update PRs.

5. Browser Web UI Over SSH / Tailscale	`OPENCODE_SERVER_PASSWORD=secret opencode web --hostname 0.0.0.0 --port 4096`	Serves OpenCode's full browser interface so you can steer sessions from a tablet or browser.

Always Set `CI=1` and `PAGER=cat` for Agent Subshells

Interactive CLI tools like `git log`, `gh`, or `vitest` wait forever for human keypresses unless `CI=1` and `PAGER=cat` are present. Recipe #1's five-line `shell.env` plugin eliminates hung terminal tools permanently.

Hardening Headless Servers, CI Secrets & Fork Pull Requests

Giving an AI agent shell and write permissions inside an HTTP daemon or CI runner demands strict network, authentication, and fork-isolation boundaries. Lock down these four surfaces before going live.

1 Never Expose Unauthenticated `0.0.0.0`

- **Loopback by Default:** Keep `opencode serve` bound to `127.0.0.1` locally and forward via SSH or Tailscale Serve.
- **Mandatory Basic Auth:** Always export `OPENCODE_SERVER_PASSWORD` (and optional `OPENCODE_SERVER_USERNAME`) whenever binding to a LAN or Tailnet interface.
- **Restrict CORS:** Pass explicit allowed origins via `--cors https://ops.internal` rather than wildcard origins.

2 Defend GitHub Actions Against Prompt Injection

- **Collaborator Gate Only:** Never let untrusted public issue commenters trigger write-scoped CI agents on repos holding cloud secrets.
- **Least-Privilege `GITHUB_TOKEN`:** Scope workflow YAML `permissions:` strictly to `contents: write` and `pull-requests: write`; never pass broad org admin PATs.
- **Pin Action SHAs:** Pin third-party GitHub Actions by commit SHA in regulated repositories.

3 Audit Third-Party Plugins Like Root Code

Plugins listed in `opencode.json` execute inside your host user's Bun process with full filesystem and env access. Pin exact semver versions (never floating `latest`) and vendor critical internal plugins under `.opencode/plugins/`.

4 Containerize Autonomous `opencode run` Loops

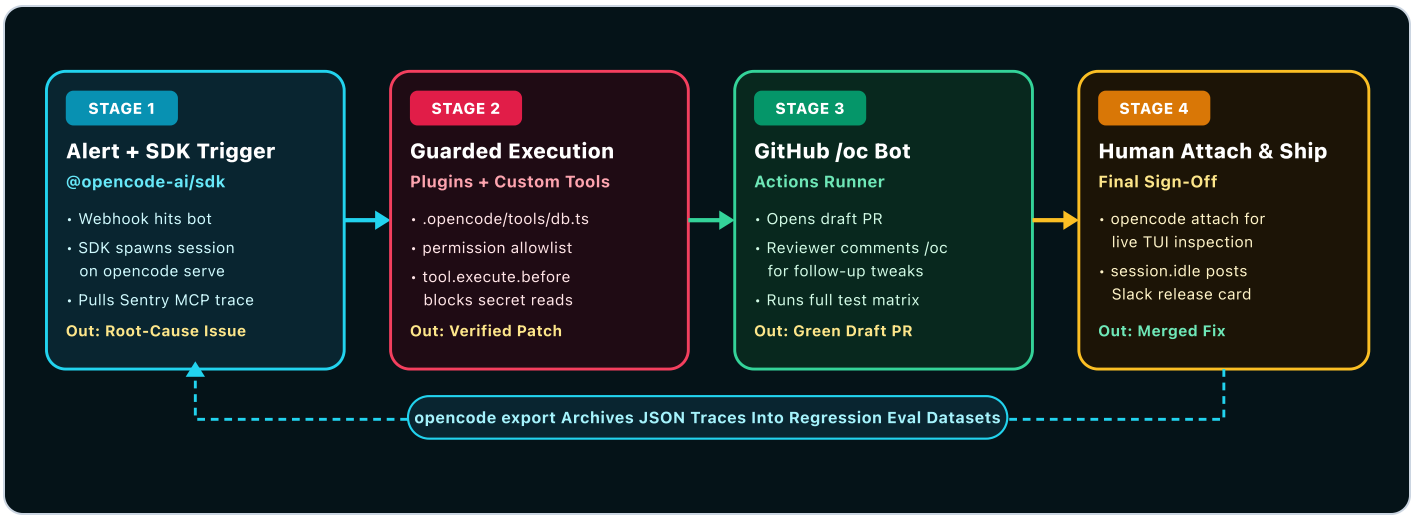
When running broad unattended migrations with `"bash": "allow"`, run `opencode serve` inside a rootless Docker devcontainer or ephemeral microVM so accidental destructive commands stay isolated from your host laptop.

Watch Out for the Plural `"permissions"` Typo!

In `opencode.json`, the schema key is singular `"permission"`. Writing plural `"permissions"` fails schema validation or leaves defaults active—always keep `"$schema": "https://opencode.ai/config.json"` at line 1 so your editor catches typos immediately.

Cross-Tool Synergy: Sentry Alert → SDK Triage → Guarded Fix → PR

See how custom tools, MCP connectors, plugin guardrails, headless `opencode serve`, and GitHub Actions unite into an end-to-end incident-to-PR pipeline with human review before merge.



A Warm Daemon Reuse Cuts Script Latency

Starting MCP servers and language servers from scratch on every shell command adds 3–8 seconds of overhead. Keep one background `opencode serve` daemon alive and pass `--attach http://127.0.0.1:4096` to every `opencode run` script.

B Shared Team Repo Harness

Checking `.opencode/tools/`, `.opencode/plugins/`, and `opencode.json` into Git gives every new engineer—and your GitHub Actions bot—identical domain tools and safety guardrails on first clone.

Troubleshooting Clinic & 6-Point Pre-Flight Checklist

Resolve common headless CI rejections, Zod version mismatches, hung shell tools, and MCP OAuth errors in seconds, then verify all six release gates before automating repos.

Symptom / Failure	10-Second Root-Cause Fix
1. `opencode run` fails immediately with permission denied	Headless runs have no interactive prompt, so any rule resolving to `"ask"` auto-rejects! Explicitly set needed commands/paths (including `external_directory` if applicable) to `"allow"` for CI configs.
2. Permission rules in `opencode.json` are ignored	You wrote plural `"permissions"` instead of singular `"permission"`, or put a broad `"*": "allow"` rule *after* specific rules (last matching glob wins!).
3. Custom tool throws Zod schema type mismatch	You imported an incompatible standalone `zod` version. Import `{ tool } from "@opencode-ai/plugin"` and define args with `tool.schema.string()`.
4. Agent hangs forever running `vitest` or `git diff`	Interactive watch mode or `less` pager is waiting on stdin. Add a `shell.env` plugin setting `CI="1"` and `PAGER="cat"`, and pass `--run` to Vitest.
5. Remote MCP server returns `401 Unauthorized`	OAuth token expired or wasn't completed. Run `opencode mcp auth <name>` and verify status via `opencode mcp debug <name>`.
6. GitHub Actions `/oc` comment does not trigger job	Confirm the OpenCode GitHub App is installed (`opencode github install`), the commenter has repo write/triage access, and workflow YAML exists on the default branch.
7. Context tokens spike by 25k before first prompt	Too many MCP servers are enabled globally. Disable bulky MCP tool globs under top-level `"tools"` and enable them only on specialist subagents.

6-Point Pre-Flight Checklist Before Shipping Plugins & CI

- [x] **Schema Validated:** Declared `"$schema": "https://opencode.ai/config.json"` and verified singular `"permission"` key.
- [x] **Dual Secret Shield:** Blocked `*.env*` / `*.pem` in both `permission.read` and `tool.execute.before`.
- [x] **Non-Interactive Env:** Injected `CI=1` and `PAGER=cat` via `shell.env` hook so subshells never hang.
- [x] **Scoped MCP Schemas:** Disabled heavy MCP tool globs globally and enabled per specialist subagent.
- [x] **Authenticated Daemon:** Set `OPENCODE_SERVER_PASSWORD` and loopback/Tailscale binding for `opencode serve`.
- [x] **Least-Privilege CI:** Restricted `/oc` GitHub workflow to trusted collaborators and minimal token scopes.