Portfolio

Patchy: an AI teammate for engineering

Patchy is an AI assistant I designed, built, and operate for Dispatch Product & Engineering. It started as a Slack bot and has grown into an agent platform: it handles Jira, release, code, infrastructure, and incident-response busywork, analyzes codebases, and opens pull requests on request. Dozens of features shipped and in daily use across the engineering org.

One conversation, three surfaces

Patchy answers in Slack (Socket Mode), in GitHub pull request comments, and in Jira issue comments, and a single conversation identity spans all three. Ask it in Slack about the PR it just opened, and it knows. Mention it on that PR, and it picks up the same history with no re-explaining.

Platform

- **Orchestration:** Anthropic Messages API agent loop with streaming, tool use, server tools (web search/fetch, code-execution sandbox), and adaptive thinking. Vanilla ESM JavaScript on Node 22, no build step.
- **Infrastructure:** Kubernetes deployments across dev, staging, and production. Redis for conversation state, memory summaries, code sessions, and tasks. Datadog dashboards and structured logging for observability.
- **Tool system:** an integration catalog covering Jira, GitHub, Confluence, and AWS/K8s tooling. High-frequency tools stay in context, and the long tail is discovered on demand via BM25 tool search. Programmatic tool calling lets the model chain read-only tools from inside the code-execution sandbox, with write-capable tools refused client-side.
- **Cost engineering:** prompt caching, server-side context compaction, and token accounting keep an always-on assistant affordable.

An embedded coding agent

For real code work, Patchy runs Claude Code sessions inside its pod: it loads the target repo, analyzes or edits, then branches, commits, and opens or updates the PR, all from a Slack mention. A skills system backed by S3 gives it reusable, versioned procedures with CRUD tools to manage them, and LSP plugins provide code intelligence across Go, TypeScript, and Python.

Guardrails

An org-wide AI agent with infrastructure access needs real access control: role-based tool permissions decide who can trigger write actions, identity enrichment resolves each requester across Slack, Atlassian, and GitHub, and `kubectl`/`aws` execution is gated rather than free-form.

Toward autonomy

The current frontier is moving Patchy from on-mention to autonomous. Two threads of work:

- **Sprint queue:** Patchy pulls tickets from the Jira sprint queue and works them end to end (branch, implementation, PR) instead of waiting to be asked.
- **Loop:** a LangGraph-based service that Patchy delegates work to over an internal API, with runs recorded in Redis and traced in LangSmith. The target workflow: a request arrives by chat, gets evaluated and categorized into a ticket, fans out to parallel Claude Code workers, and converges on an eval gate before feeding an agentic release workflow, with humans at two review points instead of in the middle of every step.

Keywords: Anthropic Messages API, Claude Code, LangGraph, LangSmith, Node.js, Kubernetes, Redis, Datadog, Slack, GitHub, Jira, prompt caching, agent orchestration

Making Kubernetes maintenance boring

Our EKS environments used to be unpredictable, the upgrade path was murky, and system maintenance felt like performing open-heart surgery. I got sick of it, rebuilt the cluster infrastructure in Terraform, and wrapped its lifecycle in a rigid quarterly playbook. Maintenance has been a breeze ever since, and our HA/DR posture has never been stronger.

- **One stack, every environment.** Environment-agnostic Terraform root modules with per-environment variable and backend files, so dev, staging, sandbox, and production are the same code with different pins. Kubernetes version,

EKS add-ons, node AMIs (AL2023 and Bottlerocket, x86 and Arm), and Helm chart versions (Datadog, cluster autoscaler, KEDA, load balancer controller, and more) all live in tfvars, which turns a cluster upgrade into a reviewable diff.

- **A rigid quarterly playbook.** Every maintenance window runs the same documented runbook: preparation steps, maintenance night, testing, expected outcome, and a rollback procedure, with each step assigned an owner and tracked per environment. Changes roll dev through production in stages, and helper tooling generates the exact version-bump Terraform for each quarter.
- **Predictability pays out.** Rolling node-group replacements keep upgrades boring, multi-arch node groups enabled Graviton migrations (including moving DocumentDB from x86 to Arm) for meaningful cost savings, and high availability and disaster recovery are engineered into the stack rather than bolted on.

Keywords: Terraform, AWS EKS, Kubernetes, Bottlerocket, Graviton, Helm, KEDA, DocumentDB, HA/DR

The best secret is no secret

Stored credentials have to be secured, rotated, audited, and eventually they leak. I spent the last few years removing them from Dispatch's platform, one credential class at a time, and replacing them with short-lived, identity-based auth.

- **Workload identity for services.** Migrated microservices off per-service IAM users and static access keys onto Kubernetes service accounts federated with AWS IAM. Each pod assumes a Terraform-managed IAM role through the EKS OIDC provider, trusted only for that exact service account and scoped to least-privilege policies.
- **Passwordless databases.** Built on top of workload identity to remove database passwords entirely: services authenticate to DocumentDB and RDS Postgres with IAM-based auth instead of connection-string credentials. I owned the full stack of this change, the IAM roles and policies, the backend Go integration, the Kubernetes service account administration, and the careful production rollout.
- **Keyless CI/CD.** Cut long-lived credentials out of the pipelines on both sides. GitHub Actions authenticates to AWS through OIDC federation, minting short-lived sessions that role-chain into the right account, and deploy tooling authenticates to GitHub through a GitHub App with short-lived installation tokens instead of stored PATs.

The secrets that still have to exist live in AWS Secrets Manager, synced into Kubernetes by an external secrets provider. The rest no longer exist at all.

Keywords: AWS IAM, OIDC, EKS workload identity, IAM database authentication, GitHub Actions, GitHub Apps, Terraform, Go

Homelab

Fully automated provisioning and management of a home media and infrastructure server, dual-Xeon Supermicro hardware with ZFS storage pools, driven end to end by Ansible, Docker Compose, and GitHub Actions. A 25+ container Compose stack (media services, Immich photo management, Mattermost, and Pochi, a self-hosted AI assistant bot I built for Mattermost) is templated by Ansible and deployed by workflow. Monitoring runs on Grafana, Prometheus, node-exporter, and cAdvisor, and remote access is locked down with SSH hardening and Tailscale. A Terraform-provisioned "cloud lab" twin on AWS EC2, with mocked ZFS pools, lets me test playbook changes before they touch the real hardware.

Private repo. Access available on request.

Keywords: Ansible, Docker Compose, GitHub Actions, Terraform, ZFS, Grafana, Prometheus, Tailscale

This website

Built with Hugo and deployed on AWS Amplify with a custom domain. Terraform defines the infrastructure, GitHub Actions stands it up and tears it down, and every push to main triggers a build and deploy.

Keywords: Hugo, Terraform, AWS Amplify, GitHub Actions

Citizen Science Aerial Image Aggregation

Web application supporting drone flight uploads: extracts metadata from drone images, stores it for efficient querying, and builds composite images. GitHub →

Keywords: Python, MySQL, Flask, Angular, OpenStack