KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

How We Are Enabling Feature Flagging At Scale

Juan Bernal, CVS Health



Our Feature Flag Platform Journey

At **CVS Health**, engineers are constantly delivering new innovations in health services. Within our Developer Experience org, we recognized a clear need for a better **feature flagging** solution.

Existing options at CVS pose some problems:

- **Seat limitations**
- **Feature set complexity**
- **Limited home-grown tools**

To find a better path, we turned to the **open-source community**.

Today, we're excited to share both **what we've built** and **the journey that brought us here**.

What are feature flags?

Runtime awesomeness

Feature flags are controls in software that let teams turn functionality **on or off at runtime** without requiring a new code deployment.

- Separate **deployment** from **release**
- Let teams expose features to specific groups such as **internal users** or **beta testers**
- Support **progressive delivery**, including gradual rollouts and canary releases
- Provide a **kill switch** to quickly disable a feature if issues appear

Open Feature

Utah, we have a *problem.*

Tool-choice fragmentation

CVS Health teams had a couple options to turn to for their feature flagging solution:

- **Vendor solutions**
  - Seat limits restrict adoption
  - Cost vs. value is unclear
  - No path when seats run out
  - Contract end creates risk
- **Limited home-grown tooling**
  - Basic tooling drives centralization
  - Tooling gaps create dependency
  - Limited features reduce efficiency
  - Limits become blockers

An open path forward

The adopted standard would have to satisfy:

- **Consistent evaluation behavior** across all flag sources
- **Low-friction migration** from third-party tools to a new platform
- **Speed and scale** for high-volume feature flag evaluations
- **Broad compatibility** across languages and frameworks
- **An open-source standard** to support long-term platform flexibility

Open Feature

So, how do we solve this?

Enter OpenFeature

What is OpenFeature?

OpenFeature is an open-source CNCF incubating project that provides not just a way to evaluate feature flags, but a standard around how.

The OpenFeature specification targets simple to complex uses for feature flags.

OpenFeature uses application Providers to talk to the evaluation engine which evaluates flags based off a configuration.

Why we chose OpenFeature

- **One contract across platforms and flag sources**
  Developers use the same API regardless of the underlying provider.
- **Faster path to feature parity**
  The standard supports a wide range of feature flagging use cases and already has broad ecosystem support.
- **Built for scale, flexibility, and broad compatibility**
  Supports major programming languages, works across different hosting models and vendor solutions.

tl;dr it checked our boxes

A platform was born.

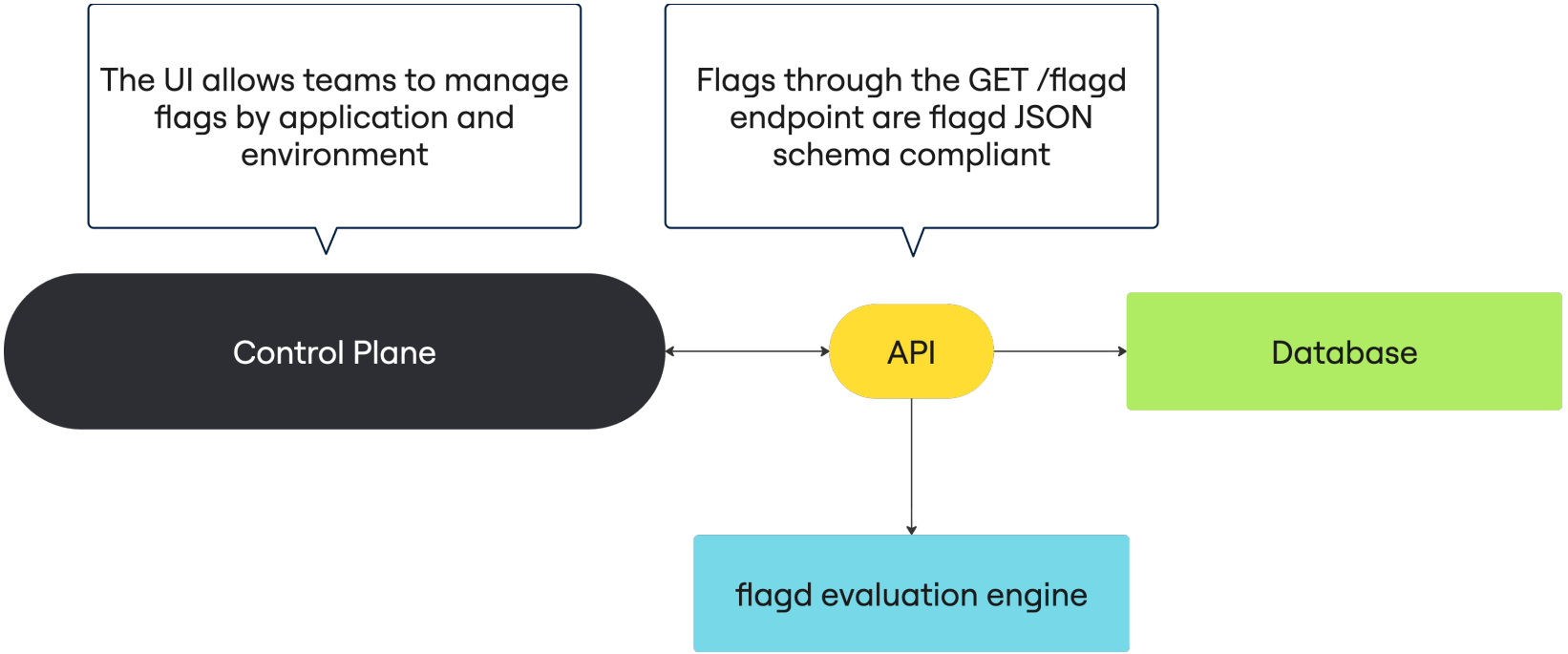
So, *how* did we build this thing?

Platform needs

To create a feature flag platform for CVS Health, we would need:

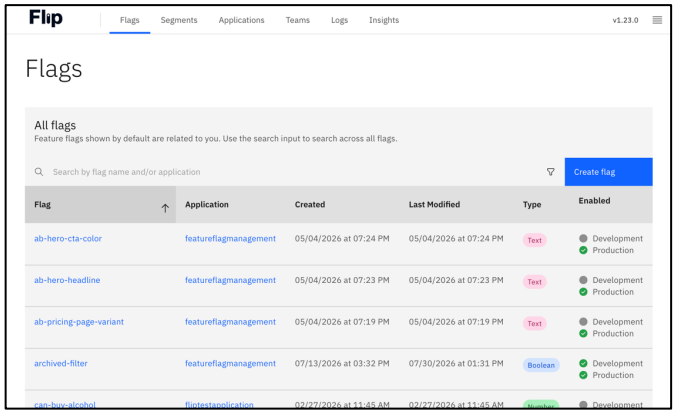
- **Intuitive, user-focused interface** for managing feature flags
- **Shared backend service** powering both the UI and the flagd evaluation service
- **Custom OpenFeature Providers** by language and framework to support environment by application needs
- **Robust end-user documentation** to support onboarding, support, and day-to-day usage

Orchestration



User interface

While most user interfaces were built with an internal CVS Health design system internally, this interface was built exclusively with open-source libraries and frameworks.



- **Framework:** Next.js 16 with React 19
- **UI system:** IBM Carbon Design System
- **Data and state:** TanStack React Query
- **Interactions:** Atlassian Pragmatic Drag and Drop
- **Authentication:** NextAuth
- **Observability:** OpenTelemetry
- **Testing:** Vitest, Testing Library, Playwright, axe-core
- **Styling/utilities:** Sass, date-fns

API

We turned to widely adopted libraries like **Fastify** and **Drizzle** for our API. This API serves both the UI and the flagd service.







- **Framework:** Fastify
- **Database:** PostgreSQL via pg
- **ORM and migrations:** Drizzle ORM and Drizzle Kit
- **Validation and schemas:** AJV, JSON Schema, JSON Schema-to-TypeScript
- **Authentication/security:** JSON Web Tokens and JWKS RSA
- **Observability:** OpenTelemetry Fastify integration
- **Testing:** Vitest

API to flagd schema compliance

flagd requires a defined schema, and our control plane data is transformed into schema-compliant JSON. That API response becomes flagd’s source of truth, giving the evaluation engine exactly what it needs to evaluate flags correctly.

Why does this matter? The API is used by the flagd instance (the evaluation engine) as its source of truth for the feature flag configuration.

That API response becomes flagd’s source of truth, giving the evaluation engine exactly what it needs to evaluate flags correctly.

Where are we today?

The numbers

60+

Teams

~390M

Evaluations per day

What if we went open source?

We did.



Flag Controller

self-hostable, open-source feature flag control plane

What’s included?

For end users (PMs, QA, non-engineers)

Outside of the commonly known benefits of feature flagging platforms, Flag Controller lets you:

- **Toggle features on/off instantly** without code changes or redeploys
- **Create and manage targeting rules** to control who sees a feature (by user attributes, segments, etc.)
- **Build reusable audience segments** shared across multiple flags
- **Set up gradual/weighted rollouts** to release features to a percentage of users
- **Test-run a flag against** a sample evaluation context to preview results before going live
- **View a full audit log/history** of every change made on the control plane
- **Permissions** based on team membership

For engineers

- **Integrate flags into applications** using standard OpenFeature SDKs (language-agnostic)
- **Evaluate flags via a self-hosted flagd engine** over gRPC/protobuf or the OFREP REST API
- **Scope flag resolution per app/environment** using flag set IDs
- **Extend or self-host the platform** since API, UI, and evaluation engine are all open source
- **Flexible authentication via better-auth** lets you swap or add sign-in providers/methods to fit your org's auth requirements without re-architecting the API.

Let’s recap what the heck this all means.

For CVS Health

- **Guided innovation** through community feedback
- Improve **transparency and trust**
- Attract **contributors, talent, and external partnerships**
- **Reusable work** beyond our immediate team
- **Useful solution** to a real problem, for all

For the community

- No open-core paywall
- Full stack is self-hostable
- UI, API, and docs come together
- Built on the OpenFeature standard
- Uses OpenFeature's flagd evaluation engine instead of a proprietary model
- Vendor-neutral by design
- Open across the full flag lifecycle
- Community can audit, improve, and evolve it
- Management and evaluation planes are both open

OpenFeature standardizes. flagd evaluates. Flag Controller manages.

Code



github.com/cvs-health/flag-controller

Docs



cvs-health.github.io/flag-controller

Thank you

