



FEATURE MANAGEMENT


# Build vs. buy guide

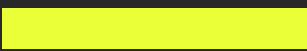
Explore the pros and cons between building or buying a feature management system.

LaunchDarkly →

# Contents

Introduction	02
To build or to buy?	05
Feature flags vs. configurations	07
Build your in-house FMP	10
What if you've already built your own FMP?	15
10 capabilities developers need in a FMP	19
The verdict is clear: buy over build	24



(  )

# Introduction

You're probably here because you've heard—or seen firsthand—how much easier life can be when you use feature flags to separate code deployments from releases. Maybe you want to try a feature management platform (FMP), but aren't sure if you should commit to a hosted version yet. Perhaps you're considering an open-source alternative or even taking time to build your own. This guide will help you determine which is the best path forward for your team.

# To build or to buy?

Some developers are apprehensive about signing up for a commercial FMP product. After all, why not just develop your own?

And they may have good reasons to consider running their own FMP instead of buying one, including:

- Tight budgets
- Unsure about relying on a service that isn't free or open source software
- Desire for full control over all the systems that involve your code's execution
- Inability to buy services without a lengthy procurement process
- Thinking feature management is just a few Booleans; why not put them in app configuration and call it a day?

While these are all valid points, you should also consider factors like FMP implementation and future organizational growth. But before we discuss these aspects, it's important to first clear up the differences between feature flagging and configuration.

### **Homegrown as a starting point**

In a recent survey, just over half of current LaunchDarkly customers said they began their feature flagging journey using a homegrown system. Read about their biggest motivations for using feature flags, rates of deployment, and more in our report, [The State of Feature Management](#).

# Feature flags vs. configurations

When deploying a server-side application, deployment usually includes a set of configuration files (config).

The config and code are bundled together for the deployment, and the config usually doesn't change until the next deployment. Even if some config is read from somewhere other than the bundled files, the app won't read it again until it restarts.

In contrast, feature flags can change at any time. Flags are used to control app behavior, and so when the flag changes, the app behavior should change immediately. No redeployment or restart is required. This does not mean that the app code is replaced. Rather, the next time the app evaluates a flag in a conditional, the returned value will be the new one.

The difference between the behavior of flags and config also shows their different uses:



Config should be used for everything the app requires to start, including connections to essential resources such as the FMP.



Feature flags should be used for behavioral decisions that the app makes after it has initialized.



# Getting started

Minimum viable functionality. Let's examine what it would take to build your own FMP.



Here is a baseline set of functionalities that will still make feature flag usage worthwhile:

- Flags are evaluated in your app code, so you'll need an API
- A persistent flag store to make flag definitions and values should stick
- A user interface to create and flip flags
- An audit trail to track changes (also vital for debugging)
- An API protocol architected to provide immediate flag updates

Because the primary goal is simply creating a minimum viable FMP, this list of functionalities intentionally leaves out higher-value features like multivariate flags, experimentation, and integrations.

## The Build It Checklist

Completing the build checklist will give you clear direction on whether building a feature management solution in-house is right for you or if it makes sense to invest in a solution like LaunchDarkly.

These are some of the key considerations your team will need to ponder when exploring a DIY feature management system, but this list is by no means comprehensive. As anyone who has built something fairly substantial from scratch understands, it's best to expect the unexpected.



## Goal

Determine what your team's feature flagging needs. Consider goals based around functionality, user experience, and cost.



## Resource

Estimate the costs incurred from building your solution: Engineering hours diverted from your core product, infrastructure, implementation, and any other resources required.



## Plan

Make a plan for maintaining the system in the long run. Who will support it? Will you dedicate an engineer to your own issues/additions? How much will maintenance cost?



## Time

Estimate timelines for scoping, building, testing, and integrating the system into your tech stack and process.



## Rollout

Think about the rollout plan for training new users on the system, and who will be in charge of that moving forward.



# Building your in-house FMP

Now let's consider the implementation aspects. When choosing to build your own FMP, you're immediately faced with two options for implementation.

1

## Code it Yourself

Depending on your organization's size and needs, creating an in-house FMP that offers basics like an API, flag store, user interface, audit trail, and immediate flag updates could take anywhere from a few hours to a few days.

However, this only applies to the first version, with an API client for just one language. Once you start to account for aspects like feature requests, things get more complicated and time-consuming. This hypothetical FMP also lacks the ability to provide testing in production.

2

## FOSS

If you don't want to write an FMP from scratch, but you still want complete control over the code and how it's run, free and open source software, (FOSS), can be a perfectly suitable solution.

There are plenty of FMPs out there with FOSS licenses. Most of them are pretty simple—light on features but also limited to a specific language or application framework. Here are some examples of language/framework-specific feature management platforms:

Project Name	Language/ Platform	Features
FF4J	Java	Custom extensible targeting rules; web-based dashboard with monitoring and audit trail; CLI and REST API interfaces
Django Waffle	Django (Python web framework)	Basic-but-extensible targeting rules; some Javascript support; test utilities
Flipper	Ruby	User targeting percentage rollouts; web UI; REST API; basic instrumentation
Flagception	Symphony (PHP framework)	Powerful targeting rules; limited percentage rollouts; basic web API
FunWithFlags	Elixir	Basic user & group targeting; percentage rollouts

But if your team uses more than one programming language, you'll need an FMP that caters to them, and is available as a network service with a web user interface. These systems offer client SDKs for multiple languages as well as network APIs (using REST or GRPC) so they can be accessed by other platforms:

Project name	Language/Platform	Features	Update checks
Bullet Train	Javascript (browser), Node.js, Java, Python, Ruby, iOS, Android, .NET	Multiple projects & environments; segments; user attribute storage; percentage rollouts; audit log; experimentation	Polling
Flipt	Go, Ruby	Segments; basic rule editor; percentage rollouts	Stream (GRPC)
Unleash	Node.js, Java, Python, Ruby, Go, .NET	Rollout strategies; webhook integration	Polling
Flag	Javascript, Python, Ruby, Go	Powerful targeting rules; limited percentage rollouts; basic web API	Polling
Tweek	Javascript, .NET	Percentage rollouts; experimentation; rule editor; flag prerequisites; audit log; OAuth support; webhook integration	Polling

One primary drawback these options have in common? They are all time-consuming. And, as you know, this translates into added costs. Plus, you'll still need to account for factors like:

- Installing and configuring the FMP
- Getting agreement from the operations team, which has to adopt the FMP as another component of the production system
- Ensuring that enough other people know how to maintain the FMP

Ultimately, paying a vendor ends up being no different than building and maintaining your own FMP. The added expenses could include diverting developer resources or missing other tasks and opportunities due to focusing on the in-house FMP.

# What if you've already built your own FMP?

You may already be using an FMP developed in-house that is currently meeting your needs. However, as your organization grows, you'll inevitably outgrow certain tools and solutions.

This certainly applies to an FMP, so if you're currently using an in-house solution, here's how you can determine if you've outgrown it—and whether it's time to explore alternatives.



## How has your application changed?

Companies typically use multiple languages in their applications. Can your solution adapt to the addition of new programming languages in the application stack? How many languages do you need to support? What are your plans to add support for more languages?

## Is there a need to run client-side vs. server-side?

Opening connections from the client-side resource to the server requires authorization and implementation of security measures. If your user base is geographically diverse, a content delivery network (CDN) may be needed to help with latency.

## How will you support multiple use cases across multiple teams?

When you started, a single team was using a handful of flags to separate code deployments from feature releases. But what if other departments now want to create feature flags? New use cases require access control and permissions for safety and security purposes, so they will need additional functionality.



### Operational needs

- Audit logs to see flag changes across the organization
- Integrations with existing tools to toggle flags programmatically or kickoff other workflows
- Adjust logging levels programmatically when an alert is received



### Customer Support needs

- Troubleshooting tools to know which variation a user was served to troubleshoot issues



### Sales and Product Management needs

- Multivariate flags to define multiple options that a flag will serve for running experiments
- Advanced targeting rules to control who does or does not have access to certain features
- Flag prerequisites and relationships to control groups of features



### All teams eventually need

- Access control logs to restrict access to their team's flags and ensure only authorized people can toggle a flag on or off
- Reporting and insights on the flags served
- An intuitive user interface to toggle flags on and off
- Scalability for increased page views and greater numbers of flags being toggled
- An organization-wide dashboard to display the flags, what's enabled, a flag's purpose, and a point of contact for each flag
- Audit logs to see flag changes across the organization

- Integrations with existing tools to toggle flags programmatically or kickoff other workflows
- Adjust logging levels programmatically when an alert is received

## How will you support multiple feature flag systems?

If you have multiple languages to support, you may need to deploy multiple solutions, as they are often language-specific. In addition, without centralized feature management, different teams may build or deploy a solution that meets their business needs.

Given that many feature flagging tools are language-specific, the more languages your application uses, the more tools you'll need to support it.

Bottom line: the more solutions you have with similar functionality, the more confusing it is to the people supporting and configuring them.

## You've outgrown your current FMP, now what?

If you've outgrown your in-house solution, whether due to expanded use cases or the desire for greater control over the release process, it's time to invest in a commercial solution. Read on to learn more about the capabilities you should not compromise on.

# 10 capabilities developers need in a FMP

Opening connections from the client-side resource to the server requires authorization and implementation of security measures. If your user base is geographically diverse, a content delivery network (CDN) may be needed to help with latency.



## Design phase

It's important to start thinking about using feature flags at the design phase. The decisions about your feature made during this phase will inform how you create associated flags. For example, this could influence how you configure targeting rules for the feature, or how to test and collect feedback on that flag.

### Broad SDK support

Having a wide variety of SDKs at your fingertips in a single platform helps you choose the best programming language for your features. Don't limit yourself to a single SDK or deploy multiple tools to cover all your SDKs.

### Experimentation

As you are designing a feature, there may be questions about the best way to proceed. What if you could run an experiment and make a data-driven decision instead of trusting your gut?

Having data to support a design decision can save hours of work having to refactor code or start over from scratch. Also, sharing information on experiments with product, customer success, or support is an essential part of running a successful experiment. Look at what types of collaboration and access is available to people outside of the development organization.



## Build and testing phases

Quality, safety, and reliability are essential to any FMP. Here are some ways a good FMP can ensure what you're releasing is safe and hits its targets.

### Granular targeting rules for developing and testing

Targeting a single user or group of users based on various attributes gives you the flexibility you need when determining who should see what, when. You can deploy code to production early in the development phase in order to test out the design. After all, feature flagging is about testing your features in production without releasing them to everyone.

### Safety & security

Feature flags are not only utilized during the build process, they're also used for operational purposes and managing entitlements. You need to ensure you can't accidentally disable or change the wrong flag. And if something goes wrong, you need detailed audit logs to see what changes were made and by whom for a given environment. These logs can quickly identify whether a recent flag change resulted in unexpected behavior.



## Release phase

It all leads up to this, but even if things go haywire at the release phase, a quality FMP will allow you to pull the plug on new feature ASAP.

### Targeted and percentage-based releases

Not all features release to production in the same way. Some features target specific groups of users, while others can be released more broadly. What's key is the flexibility to release features in a variety of ways and the ability to target users based on multiple attributes, whether predefined or custom.

### Schedule release progression

Releases may not always be convenient to your schedule, especially when considering time zones or scheduled time off. Scheduling changes to targeting rules for future dates and times means you don't have to plan your day/week/life around a scheduled release.

### Automatically disable a flag

Releases don't always go smoothly. When a failure occurs, you want to limit the blast radius and get that feature turned off as quickly as possible. Integrations with observability and monitoring tools to automatically trigger a flag upon an alert can stop a situation from impacting more users.



## Post-Launch

A feature isn't done until the code for the feature flag has been removed. Nobody likes it when technical debt accumulates. Functionalities should exist to help you identify and remove flag code from launched features.

### Filter relevant flags to reduce the accumulation of technical debt

If your company has many flags, you need a way to quickly filter and find relevant flags to address technical debt. Filtering all flags by status, you can see which flags have not rolled out to all users, which are permanent, and those that can be safely removed.


### Configure Slack reminders

One way to make sure flags don't build up in your codebase is to have reminders and notifications sent to you regularly. Through the tools you use regularly, you can have your platform send you a message when a flag's status becomes launched or inactive. It could even provide a helpful reminder to remove the flag.

### Find all flag references in your codebase

Knowing a flag needs to be removed isn't the same as remembering all the files that reference that flag. Ensure you can integrate your flag platform with your code repository to see which files refer to a flag quickly.





# The verdict is clear: buy over build

For most developers, a feature flag system sounds simple to implement—and it often is in its most basic form. The majority of engineers could build a bare-bones feature management platform in just a few days

However, things aren't that simple. If you do the math and factor in time and support costs, you'll discover that running your own flag system is usually far less economical than paying a trusted vendor.

But this isn't just about costs; you also need to account for the features of the FMP. By investing in a feature management platform that offers the ten capabilities listed above, you ensure a more seamless deployment environment that streamlines every process, from design to release and beyond.

Besides, once you've experienced the profound relief of turning off buggy code by flipping a switch, you'll never want to go without flags again.

Ultimately, the choice to build or buy is your call. If you're interested in taking the plunge with LaunchDarkly, see why consulting firm Forrester said you'll get a 245% ROI from our platform in their report, [The Total Economic Impact of LaunchDarkly](#).

# Life with Launchdarkly

Customers using LaunchDarkly for feature management achieved:



9x

increase in  
deployment  
frequency



76%

decrease in time  
from commit to  
deploy




-1 day

mean time to  
repair (MTTR)

See more of the [impact of LaunchDarkly](#).

# How we stack up to the competition

Choose the best feature management platform for you.

Compare Features		Company A	Company B	Company C
Basic feature flag use cases	✓	✓	✓	✓
Advanced targeting	✓	✓		✓
Streaming architecture	✓			
Flag scheduling	✓			✓
Flag approvals	✓			
Flag triggers	✓	✓		
Workflow templates	✓			
Experimentation	✓	✓	✓	✓
20+ integrations	✓		✓	
SDKs for every major language	✓			

Empowering all  
teams to deliver and  
control their software.

[launchdarkly.com](https://launchdarkly.com)

[sales@launchdarkly.com](mailto:sales@launchdarkly.com)

LaunchDarkly →