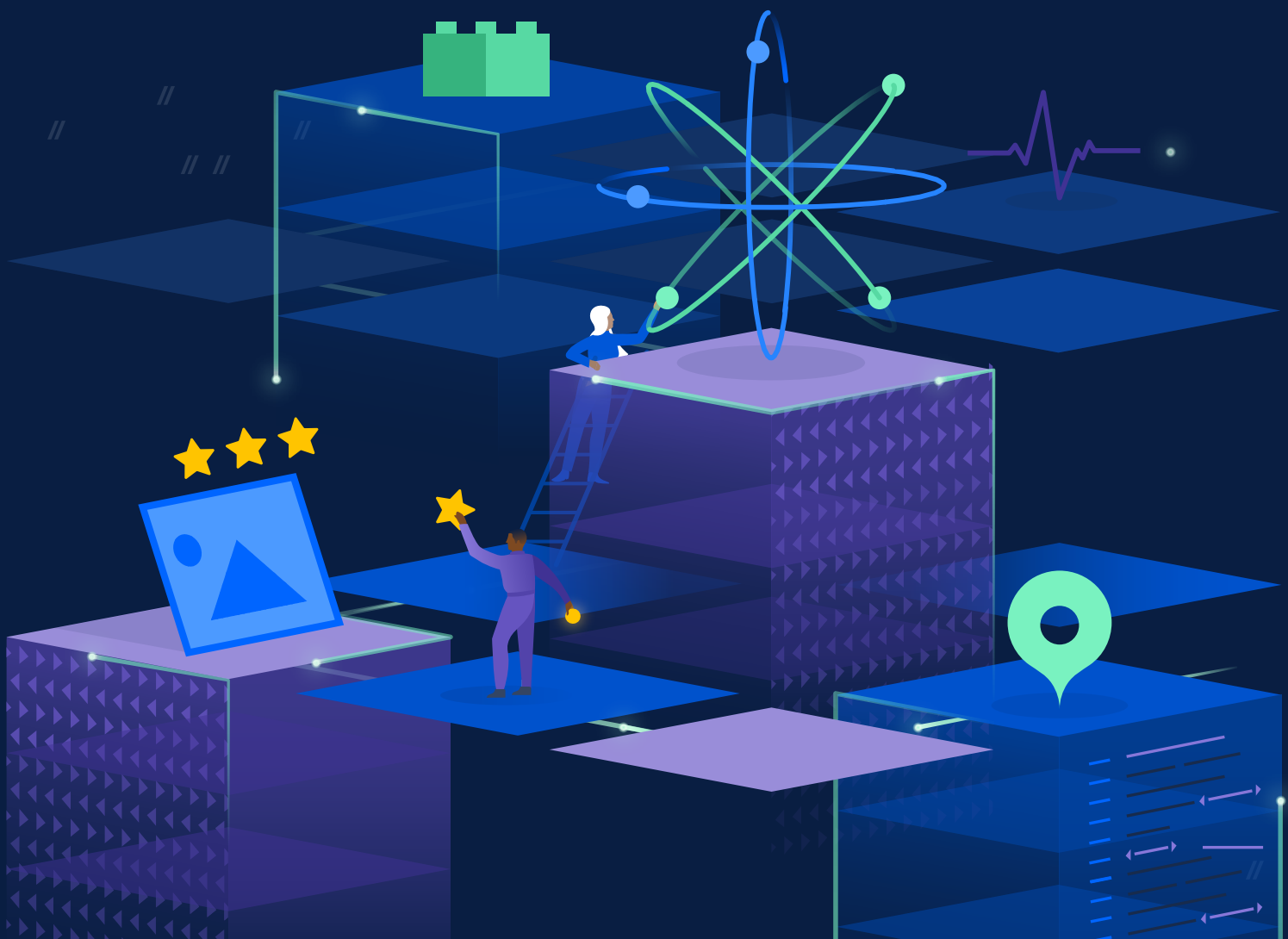
ATLASSIAN

The definitive feature flagging guide for progressive software delivery

An Atlassian buyer's guide



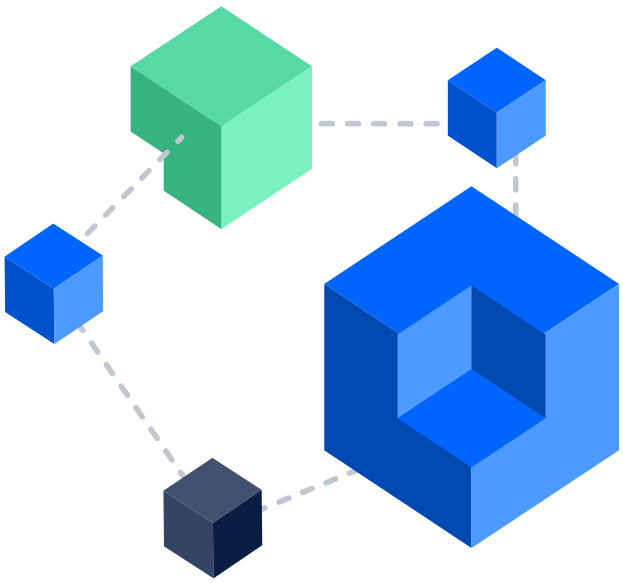
WHY READ THIS GUIDE?

Connecting and implementing feature flags to Jira Software gives your team immediate insight into the release status of your work and helps you monitor the rollout of new features.

You can use this guide to understand more about feature flags and the critical business considerations to make when implementing or deciding what feature flag integration to go with.

We collaborated with the partners behind some of our favorite feature flag solutions to put this guide together so you can get practical tips on how to evaluate and implement feature flags for your team's software development practices.

Table of contents

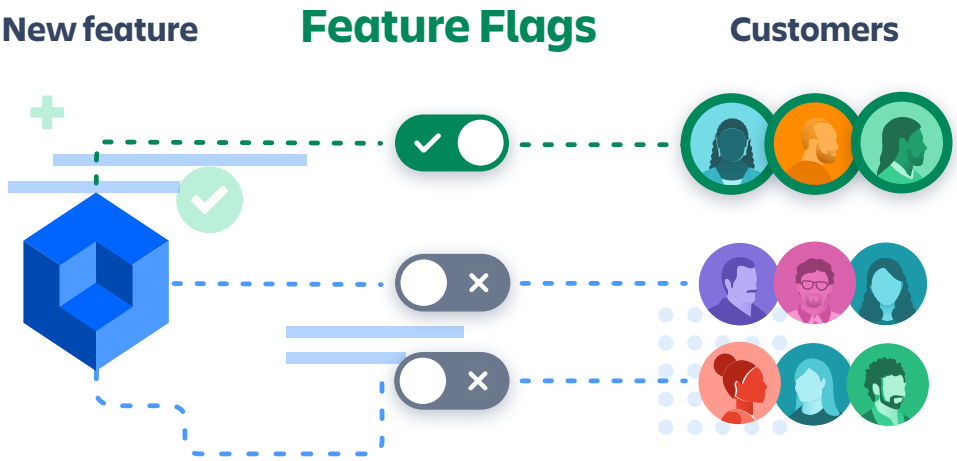


What's a feature flag?

Feature flagging is an emerging best practice used by modern software teams. It offers a new way to get immediate feedback directly from your users, signaling when a feature is ready to deploy.

In its simplest definition, a feature flag, also known as a feature toggle, is "just an if/else statement." However, this definition is deceptively simple.

Let's dive In.



Feature flagging is a versatile technique that can be used for various purposes. As a result, the definition of a feature flag can vary depending on who you ask. At its core, the feature flagging concept remains the same: a way to change an application's behavior at runtime without changing the application [code].

As mentioned above, a feature flag is a thoughtfully placed if/else statement used to change the behavior of your application in runtime without deploying.

During development, software engineers wrap desired code paths in a feature flag. The following is an example of a basic feature flag written in Javascript:

```
if(featureFlags['new-cool-feature'] == true){
    renderNewCoolFeature();
}
```

This code demonstrates a simple statement that checks if a "new-cool-feature" is enabled. Even with advanced frameworks and tools that help manage flag data or injection and removal of the new logic path, feature flags are essentially just "if statements."

Why use feature flags?

Reduce risk

By separating code deployments from feature releases, teams benefit from increased control over the experience they deliver to customers.

Build a tighter feedback loop

Feature flags initially enable software teams to release to a small set of users. Then, they can collect customer feedback and choose to scale up or adjust their strategy.

Roll back features with ease

In case of a bug or if a new feature gets a lot of negative feedback, feature flags can serve as a kill switch, enabling you to easily roll back an update with no code changes.

Things to consider

Technical complexity

Many teams opt to build feature flagging solutions in-house, which can add technical complexity, making products more difficult to support and debug. There are many options for feature flagging on the market today that can do the heavy lifting for you, including Harness, LaunchDarkly, and Split.

Messy coordination

Teams may have multiple product versions reaching different customers. Managing and communicating the status of numerous flags across the team can be difficult.

Tech debt

Feature flags can introduce technical debt into products, although this can be managed by cleaning up flags once they're rolled out to 100% or no longer active.

What is tech debt?

The implied cost of future reworking required when choosing an easy but limited solution instead of a better approach that could take more time.

Implementing feature flags

There are many paths to implement feature flags with varying logistical considerations and return on investment. The path your team takes depends on resourcing and organizational goals.

Feature flagging has some infrastructure dependencies that must be addressed to function correctly. As teams scale their use of feature flags and switching on/off flags becomes a business decision, it becomes critical to have an authoritative data store and a management mechanism for the flags. Many third-party feature flag services provide this data store dependency.

Work with a trusted feature flag expert

SaaS feature flag solutions like Harness, LaunchDarkly, and Split can help your team achieve the benefits of a feature flagging solution sooner. They handle heavy logistics and offer easy-to-integrate libraries that expedite installation. This allows teams to focus on core business duties instead of infrastructure management. What we learned from these companies about their integrations follows.

How is Harness most commonly used?

Harness Feature Flags is the force multiplier empowering Dev and DevOps teams to increase their deployment frequency by 3X, resolve production issues in minutes instead of days, and spend 30% more time coding instead of managing releases.

Dev and DevOps teams are working faster, safer, and smarter using Harness Feature Flags.

- **Release and manage features in code.** Unblock Dev teams and empower them to build, release, and govern features using YAML, API, and OPA through automated GitOps workflows. Deliver change to customers faster and automatically govern releases and changes without ever having to touch a UI.
- **Speed without compromising control.** Automate build, deploy, and release guardrails with flexible pipelines across CI/CD and feature management. Ship more features and reduce deployment risk with the control and governance enterprises require – all without breaking things, even in production.
- **Cut developer triage time to zero.** Alert Devs when reliability or cloud cost issues are detected against their live changes, and go straight to the problematic feature flag and turn it off, no rollback required. Enable Devs to identify and resolve issues right away and focus on developing features instead of wasting time digging through logs.

feature flag evaluations per day are powered by Harness Feature Flags

What are the top considerations Harness recommends when choosing a feature flag solution?

The top items to consider for a customer should ideally be in the context of the feature management maturity journey, which is outlined in this **datasheet**:

1. **Implementing:** The basics – boolean toggles & user targeting
2. **Managing:** Visual UI – management dashboard & analytics
3. **Automating:** Security – access control & automated workflows
4. **Scaling:** Governance – progressive delivery & native CI/CD

We can bundle these into three questions to answer:

1. Can we manage it ourselves and answer questions about how flags are used?
2. Can we automate the management of releases and feature governance?
3. Can we scale this alongside our CI/CD software delivery process?

Eventually, all teams adopting feature flags will embark on this **maturity journey** no matter their starting point. Teams should seek to work with vendors that can meet them at each stage of this journey, or risk having to migrate tools or build and maintain custom solutions on top of their feature management tool of choice. Harness Feature Flags is built with this maturity journey in mind, so our customers are able to rest easy knowing that all of these requirements will be met even at scale.

What are Harness's best practices for implementation?

1. **Naming new features:** Plan ahead for how a feature will be named, especially if it will become a permanent operational flag, or if the flag is intended to be leveraged by customer-facing (non-engineering) teams. Flags should be easy to find and use for required use cases, not require more engineering input.
2. **Plan for scale ahead of time**: Create process and ownership around managing flags. As flags start to pile up in a system, they cause massive tech debt and end up adding more work instead of speeding it up. Have those conversations up front so the right solutions are in place before running headlong into feature flag hell at scale.
3. **Use feature flags by default:** Put any code change you're making behind a **feature flag**. Instead of looking for the right use case, use them everywhere, the same way that the de facto way to deploy code is through a CI/CD process. Use them as a testing and failsafe mechanism in production; it's easy to remove them if they're not needed.
4. **Consider trunk-based development:** By having **fewer long-lived branches** and code paths and more feature flags keeping changes dark even in your production environment, you will be able to merge more, deploy more, and ultimately reduce risk substantially.
5. **Leverage operational toggles:** Flags are usually meant to be transient or experimental. But, you should also consider how to put **permanent flags** around certain code paths that control key configuration settings to leave you prepared to instantly handle production failures, reduce MTTR, and make it less stressful to deal with system outages.
6. **Smaller changes, not singular massive flags:** Most user-facing features will have multiple associated changes. It makes sense to put all of them behind a single flag, but what makes more sense is to have each change behind a single flag, and link multiple flags in sequence. Now, if there is an issue, it's much easier to triage the problematic code path or test incremental changes.

7. **Feature flags as the first line of defense:** If all changes are behind a flag, **resolving production issues becomes trivial** instead of stressful. Rather than triggering a rollback or roll forward, a flag can be flipped off for the offending change while a fix is implemented and queued for deployment. Instead of an urgent war room, it becomes a matter of flipping a flag and asking the engineer to fix it for the next deployment.

8. **Test in production with live data:** While feature flags should not be used to bypass testing and acceptance in CI/CD, they should be used to incrementally **test the behavior of a change in production**. Testing on smaller user bases to validate the feature and then scaling up to test real user load and behavior is the most common use case we see for testing in production.

9. **Create enforceable governance processes:** One way to avoid feature flag hell is to have **good governance** that ensures procedures and best practices are followed. This can include approvals, meeting performance benchmarks, and only committing to release when certain triggers are met. But, things do go wrong, and it's important also to be able to audit what happened to minimize the time to resolve the issue. The combination of these make it less likely an issue will occur and make resolution simpler. Ideally, this process would be automated as a development guardrail.

LEARN MORE ABOUT FEATURE FLAGS BEST PRACTICES

Check out this **blog post** that goes in-depth about each of these. At Harness, we support our customers in implementing best practices that we ourselves use, so you're in good hands.

What are common pitfalls Harness sees in feature management?

- **Building and maintaining your own system.** Teams often **build their own** internal system for feature flags, which makes sense as they are not complicated. However, this approach limits the value of the solution since it's difficult to support multiple teams, use cases, and manage it at scale. It's also r**isky to use** without the right guardrails in place. Harness Feature Flags provides core functionality and robust capabilities such as RBAC and security, release management and governance automation, flag usage analytics, change audit trails, and triage alerts for reliability and cost. These may not seem immediately valuable, but become critical requirements over time.
- **Separating feature management from CI/CD.** Feature management is mission critical in software delivery. Teams need end-to-end visibility, control, security, and automation without cobbling together separate solutions to maximize the value of **CI/CD and feature management**. Harness Feature Flags is part of an integrated software delivery platform that connects all of these elements with a single schema for analytics, access control, security, audits, governance, and workflow automation. Harness can also help teams automatically clear old flags, verify features, triage cost and reliability issues, and surface business insights for optimized software delivery.
- **Managing flag tech debt.** Feature flags are usually transient, and can create tech debt if left in the code for a long time. Traditionally, identifying and removing stale flags trades off with new feature development. Harness Feature Flags automatically surfaces stale flags and can trigger a PR to **remove them from the code**, allowing teams to avoid this tradeoff.
- **Not knowing where to start.** The most common question we receive about feature flags is, "How do I get started?" The solution is simple: wrap any small change in a flag, which can be done by a single developer. It doesn't have to involve multiple teams or be a larger project. After testing, it's easy to scale it out to more users, both technical and non-technical.

- **Not empowering customer-facing teams to manage access.** Feature flags enable non-engineering teams like Product, Customer Support, and Sales to manage customer access without requiring manual backend changes from Dev or redeployments by DevOps. Once a good RBAC model is in place to keep changes secure, engineers can focus on shipping code and customer-facing teams handle customer requests.

Teams should seek to work with vendors that can help them avoid these pitfalls and partner with them as they embark on their **feature management maturity journey**. Harness Feature Flags is built with implementation and maturity in mind, so our customers are able to rest easy knowing that they can avoid these pitfalls and leverage feature flags successfully.

What are Harness's recommendations for feature flag maintenance and management?

Our customers break this into two categories:

1

Enforcing governance and best practices for change management: Category 1 puts development guardrails (aka governance) in place so developers can focus on building features within the given guardrails, rather than the minutia of deploying those features. We recommend establishing that framework across Dev and DevOps early to gain efficiencies in building and delivering features. Harness provides this out of the box, with room to customize based on specific customer need.

2

Removing old and stale flags in production: Category 2 minimizes the tech debt that flags create. Most flags are, by nature, transient and should be removed once the feature is fully rolled out. We recommend making a plan for flag use and deprecation early and using a tool like Harness that surfaces old and stale flags for removal – and automates it.

How other teams use Harness

- **ZeroFlucs** reduced their infrastructure cost by 60% in 1 week
- **Tyler Technologies** replaced LaunchDarkly with Harness because they wanted integrated CI/CD/FF to create a better software delivery process
- **Metrikus** cut down their commit-to-deploy time by 66% with FF and are now delivering features with 3x velocity

Technical architecture

Architecting Feature Flags for Performance at Scale

GET STARTED

Try Harness

Download Marketplace App

LaunchDarkly

How is LaunchDarkly most commonly used?

LaunchDarkly gives developers the confidence to ship code more often while empowering operations and product teams to minimize risk and continuously deliver customer value.

Teams building with LaunchDarkly use frequent, low-risk releases and actionable, data-driven insights to maximize the impact of their software delivery efforts.

- **Scale safer releases**: Eliminate the fear of deploying code to production. Standardize release workflows across large teams while driving governance and efficiency through automation.
- **Accelerate modernization:** Update your tech stack without disruption. Gradually introduce and validate new infrastructure and cloud services to ensure your applications are performing at their best.
- **Experiment and optimize:** Don't just ship features, ship value. Target new capabilities to select users and run actionable experiments to learn and iterate quicker.

What are the top considerations LaunchDarkly recommends when choosing a feature flag solution?

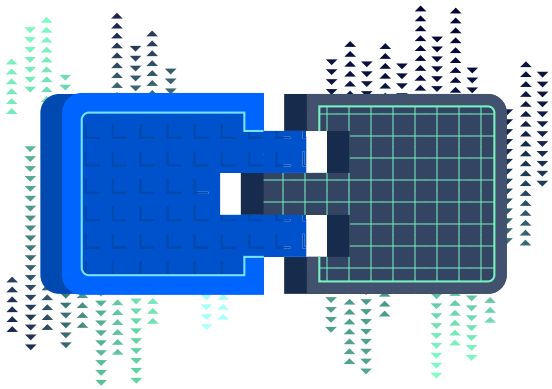
Development leaders in 4000+ organizations, including many of the Fortune 500, trust LaunchDarkly to power their digital applications.

Here's what they considered when evaluating feature flagging tools:

1. **Choose a tool developers love to use.** Check platforms like **G2** to see developer reviews of the top feature flagging tools. Ensure that your feature management platform offers **integrations** with your tech stack and maintains a **library of SDKs** for the programming languages your team uses. Find a tool with an automation engine to help teams **build and automate release workflows**, ensuring governance and reducing context switching.

2. **Ensure security, consistency, and flag hygiene at scale.** Choose a tool with **first-class architecture** that processes flag changes in real time to any device globally. Confirm it never exposes flag rules to the client side and offers role-based access control (RBAC) for granular control over platform permissions. Find a platform that helps you **identify outdated flags** in your code so you can reduce tech debt and maintain good hygiene. Feature flags + experiments = the best user experience. Deliver new features with confidence by learning from your users and systems before a full rollout with LaunchDarkly Experimentation.

3. **Deliver a better user experience with experiments.** Deliver new features with confidence by choosing a feature management platform that offers **experimentation** so you can learn from your users and systems before a full rollout.



What are LaunchDarkly's best practices for implementation?

1. **Use feature flags liberally:** Keep changes small, gate them all with feature flags, and adopt "**feature-driven development**" for fine-grained control of your application across the stack. Ensure that each feature flag has a dedicated purpose. Feature management unlocks the ability to **gracefully degrade** your system without having to redeploy. Flip a switch to strategically disable and enable components however big or small.

2. **Backend feature flagging:** Server side feature flag evaluation often focuses on use cases like API changes, database changes, or migrations. These types of rollouts become trivial when using LaunchDarkly and context-based targeting to roll out changes.

   To optimize your usage and application performance, use **self-healing SDKs** and a global **Flag Delivery Network** and cache feature flag values internally.

3. **Frontend feature flagging:** To provide the best user experience, ensure that feature flag changes are streamed to any device, anywhere in the world, in real time. LaunchDarkly **client SDKs** initialize during application start-up, in as little as 25ms, and maintain an **always-on connection to the Flag Delivery Network**. This allows us to stream feature flag updates to your application within milliseconds.

4. **Limited access networks:** To maintain limited network access and advantages to security posture, LaunchDarkly offers the **Enterprise Relay Proxy**. By allowing only the Relay Proxy external network access, you can keep your existing security posture intact without exposing your applications to the outside world.

5. **Serverless:** In **serverless applications**, optimize warm start performance by initializing the LaunchDarkly SDK client outside of your function handler. On a warm invocation, it can be reused by subsequent invocations and reduce both execution and response time. Always **close the client or flush the client** to ensure your analytics data remain accurate and up-to-date.

What are common pitfalls LaunchDarkly sees in feature management?

- **Missing the Target:** Manually writing targeting code in your application can make it exceptionally difficult to manage. Targeting changes can be isolated to your feature management platform to keep it transparent, auditable, and eliminate the risk inherent in changing code. LaunchDarkly **supports targeting** based off of the context of the request including properties like user, device type, location, or any other criteria you'd like to focus on.

- **Rolling out your own:** Feature flagging is likely not your core business. Creating your own tooling requires engineering expertise and time away from serving primary development needs. Take advantage of LaunchDarkly's experience pioneering, innovating, and scaling feature management.

  Less mature feature management systems lack enterprise capabilities like targeting, global scale, comprehensive caching strategies, ability to integrate with the most popular languages, custom RBAC, automation, and auditability. Many organizations start without appreciating that they will need many of these capabilities over time.

- **Bloating your app:** Unlike long-lived flags, ephemeral flags are for short-term use. As a form of technical debt, they are designed to be removed once your experiment or migration is complete. Without tools like LaunchDarkly's **Code References**, maintenance becomes tedious and difficult.

- **Lacking security:** Protecting your properties and evaluation logic is critical. Transmit the minimum required properties to ensure evaluation is successful. LaunchDarkly feature flag evaluation is secure by default – we only transmit feature flag data for the current user. LaunchDarkly *never* sends your feature flag evaluation rules to the client, so you don't have to worry about users figuring out how you're targeting – and how they can circumvent it.

What are LaunchDarkly's recommendations for feature flag maintenance and management?

1

Start scheming: Develop and publish a scheme for how to organize your flags that is durable to changes in teams, organizations, and applications over time. Choose a feature management platform that offers a variety of facets by which you can organize, group, and catalog feature flags.

2

Automate everything: Reclaim your nights and weekends by scheduling automatic releases. Notify your team automatically when feature flag changes need approval. **Build self-healing systems** by pairing your observability tools with your feature management platform to **automatically revert problematic releases**.

How other teams use LaunchDarkly

- **IBM goes from deploying twice a week to 100+ times a day**
- **TrueCar deploys 20 times a day, migrates 500 websites to AWS**
- **Intuit takes control of releases and makes "speed a habit"**
- **HP standardizes and scales releases with LaunchDarkly**

Technical Architecture

Evolving Global Flag Delivery

GET STARTED

Try LaunchDarkly

Download Marketplace App

How is Split most commonly used?

The **Split Feature Data Platform™** elevates the approach to building, releasing, and experimenting with software. By pairing feature flags with data measurement capabilities, Split is helping businesses release features quickly, more intelligently, and with less risk.

Split helps engineering and product development teams support **three primary use-cases**:

- **Reducing engineering cycle time:** Empower teams to deploy when they want, roll out changes whenever they are ready, and dynamically adjust features in production. Split's feature flags provide a foundation for trunk-based development, migration to microservices, and progressive delivery.
- **Mitigating release risk:** Split's feature flags allow you to separate code deployments from releases. Roll out features with guardrail metrics and alerting in place to catch faulty releases early on without disrupting the user experience.
- **Deliver features that matter:** When paired with measurement data, feature flags also power experimentation, A/B testing, multivariate testing, and beta testing so you can prove out your best ideas without slowing down. Split can ingest data from any source, speeding up data analysis and providing your business customer insights in real time.

Feature Flags are a powerful development tool. In addition to those listed above, **here** are some less common, yet still important use cases to be considered.

3 billion+

users worldwide are served feature flags from Split

What are the top considerations Split recommends when choosing a feature flag solution?

When **evaluating a feature flag solution**, one needs to take into account the following things:

1. **Fast, safe, predictable releases:** Engineering teams often struggle to move faster without breaking anything in their infrastructure, which ultimately leads to delivery delays of weeks or months. When evaluating new solutions, teams are looking into platforms that can help:
    - Speed up deployment cadence
    - Reduce release uncertainty and cycle time for simple changes
    - Monitor and alert on key metrics
    - Focus on the highest impact projects
2. **Scalable, enterprise-level architecture:** Teams are looking for solutions that can support growing demands of enterprise applications that are highly resilient and can support millions of users. They are looking for platforms that can provide:
    - Fast and reliable feature delivery via SDKs
    - Unmatched data privacy
    - Advanced approval flows, audit logs, and SSO capabilities
    - Integration with best of breed tools and languages
3. **Drive Team Efficiency:** Engineering teams are looking for solutions that can allow them to get more done without working more hours or needing additional resources. They are looking for tools that can provide psychological safety from an operational and release standpoint, which includes:
    - Tech-debt management of feature flags
    - Alerts on first sign of feature degradation
    - Ability to identify faulty features and roll them back

Feature flagging is a technique that can be used for a range of purposes. Learn more in this **Essential Guide to Feature Flags**.

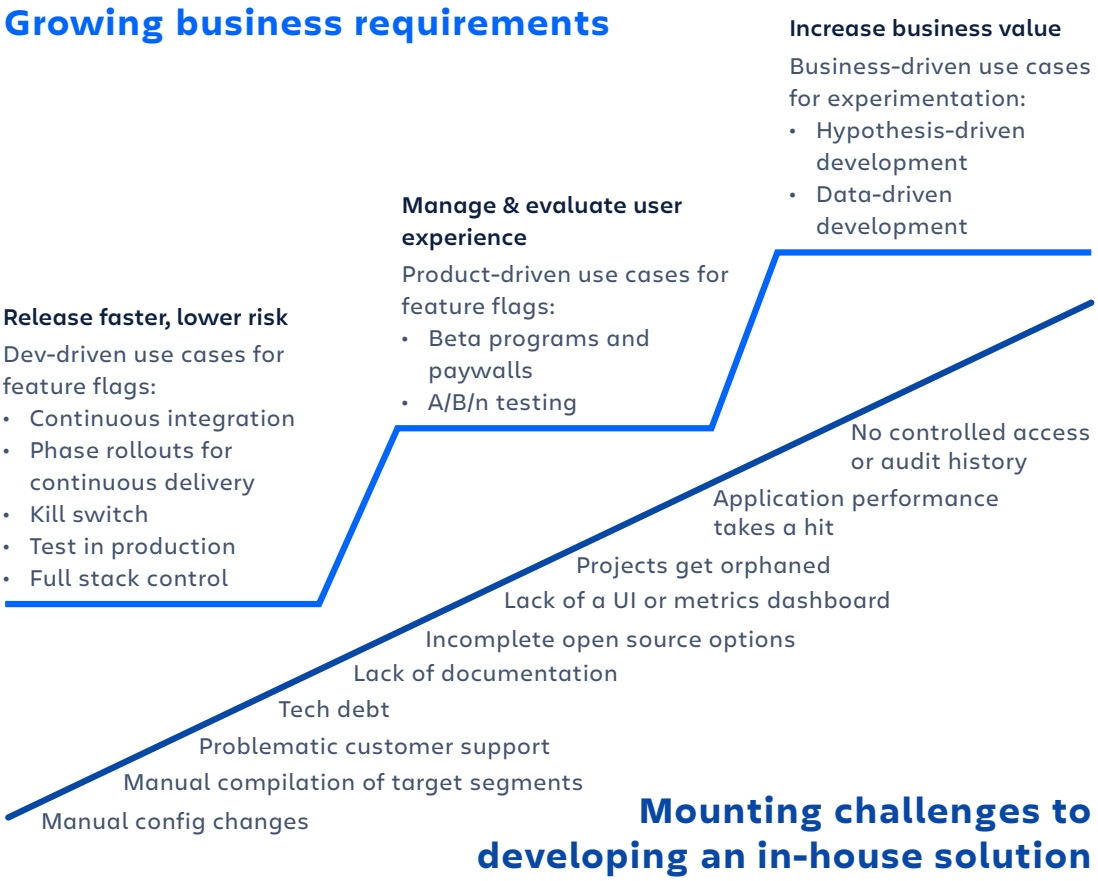
What are Split's best practices for implementation?

1. **Feature flags provide the speed and safety needed to move fast without breaking things:** Gate new features behind flags to develop and deploy to production without disrupting the user experience. Minimize the blast radius of potential bugs by gradually releasing new features to target audiences and disabling faulty features with a kill switch.
2. **Manage and govern releases to ensure compliance and reduce release errors:** Place mandatory peer review and approvals before code changes in production take effect. Control which users can edit or publish changes by setting permissions for specific objects and environments.
3. **Protect your private data:** Keep sensitive user data within your app or server so no private data (PII) is sent back to Split. When targeting on private attributes, email or demographics, let the SDKs do the heavy lifting of grabbing information from your feature management solution and sending it to your application, so that critical information is never shared outside your application.
4. **Caching keeps your app resilient:** All flags and rules should be cached locally in your SDK, so your application continues to perform as expected even in the case of a CDN outage. Our SDKs update their state by asynchronously fetching data from our CDN provider, which means no need for open connections, and no streaming connections dying unexpectedly.
5. **Put guardrail metrics in place:** Monitor features for movement in any key metric that matters for your application. Once a bad feature is identified, teams responsible for the feature flag will be notified. This allows you to simply disable the feature in production without any rollbacks or hotfixes required.

What are common pitfalls Split sees in feature management?

Build versus Buy: Product teams often consider building a homegrown or in-house feature management solution. This may seem like a good idea at first, and at the very least, can help demonstrate the benefits of feature management technology. Dev-driven use cases such as continuous integration, phased rollouts, kill switches, testing in production, and full stack control help engineering teams meet business needs of releasing faster with less risk to application stability. However, with the increased use cases comes an increase in both project scope and challenges to continued in-house development. A closer look at the full scope of challenges and incremental use-cases against internal resources cannot be understated.

As use cases for feature flags grow, so do the challenges of building an in-house system:



What are Split's recommendations for feature flag maintenance and management?

Split offers tools within its platform to help streamline the technical debt caused by feature flags.

1. Development teams can easily automate the removal of feature flags based on where they are in the feature delivery lifecycle using our Admin APIs.

2. Teams can also leverage Split's Rollout Boards to understand which flags are ready to be cleaned up based upon status.

How other teams use Split

Customers who use Split have achieved a 50x increase in deploy frequency, an ~1 minute MTTR, and 10.7% increase in engineering efficiency. Hear what a few of our customers have to say.

- **Creating customer impact via experimentation:** Learn how Imperfect Foods leverages Split to experiment and measure customer impact
- **Reduce engineering cycle time and mitigate release risk:** Learn how Speedway Motors has sped up engineering cycle time with 14x more releases per month
- **Support continuous deployment:** Learn how WePay ramped up its release cadence, separating code deployment from release

Technical architecture

Stateless Architecture

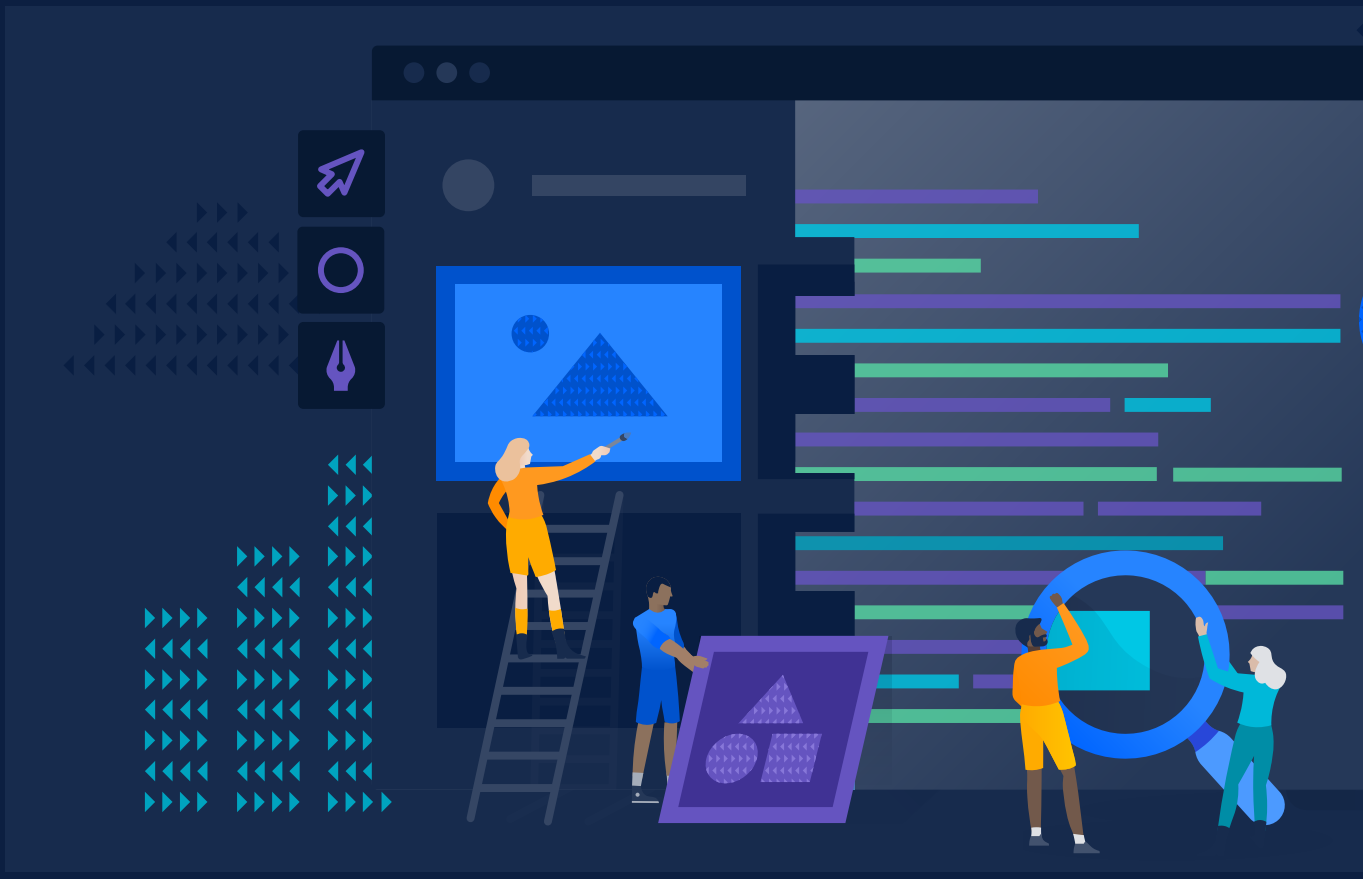
GET STARTED

Try Split

Download Marketplace App

Ready to get started with Feature Flags in Jira?

Learn more



ATLASSIAN