Flagsmith

FLIP THE SWITCH

Unlock Modern Software Development with Feature Flags

Contents

SKIM-READING GUIDE

You might not have time to read this front to back. Skip ahead to the most relevant sections:

Maybe releases are high stakes and you want to lower risk

Maybe release windows are stressful and you're worried things are going to break.

A release might mean a huge amount of work is going to go live. No one's quite sure what is going to happen when it does. The pressure builds and releases feel complex and risky.

Skip to 12

Maybe your development teams don't use feature flags yet

Maybe you don't use feature flags yet. 40% of developers don't.

Or maybe you have homegrown solutions or config files to handle feature toggling.

Skip to 14

Maybe deployments are coupled to releases and moving slow

Maybe you've got the deploy phase tightly coupled to the release phase.

Teams are working on different things and they're coupled in some way, too. Developers want to deploy their code when it's done, without it going live to users.

Skip to 11

Maybe you have a modernisation project going

Maybe you have a wider modernisation project and you're wondering about feature flags.

You might not feel "modern" yet, change happens slowly, and you're not going to be able to continuously deploy or release multiple times a day. But there's a project to move to a new system, migrate from legacy architecture, or to modernise to keep up.

Skip to 15

Foreword

THE MOVE TO MODERNISATION

Where Do Feature Flags Fit In?

Romano Roth
Global Chief of DevOps at Zühlke

zühlke

In my two decades of experience leading DevOps, Digital, and Agile transformations, I've observed that the path to modernisation is rarely a straight line. Organisations, particularly those in data-sensitive sectors, like banking, government, and healthcare, often find themselves at a crossroads. They need to innovate faster while maintaining rigorous security and reliability. This tension between speed and safety has traditionally forced companies to choose one over the other. But it doesn't have to be this way.

"Without feature flags, organisations struggle to separate deployment from release, limiting how frequently they can safely deploy changes."

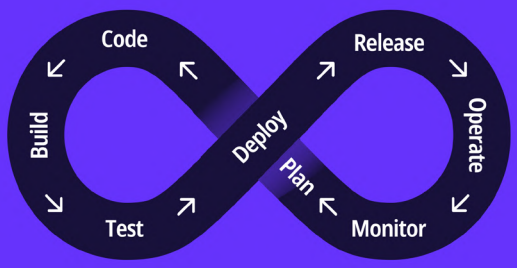
Feature flags emerge as a strategic enabler in this journey, fundamentally transforming how organisations approach software development and delivery. They are essential for achieving high performance in modern software delivery, particularly in deployment frequency—one of the four key DORA metrics that indicate Software Delivery Performance. Without feature flags, organisations struggle to separate deployment from release, limiting how frequently they can safely deploy changes. With feature flags, teams can deploy smaller changes multiple times per day while maintaining precise control. This approach significantly reduces risk: the smaller the changes in a deployment, the lower the likelihood of issues and the easier it is to identify and resolve problems when they occur.

In today's interconnected world, feature flags transcend individual application boundaries. Modern enterprises operate complex application landscapes where APIs connect various services and systems. This interconnected nature demands a holistic approach to feature management. The rise of Platform Engineering has made feature flags a fundamental platform capability. Solutions like Flagsmith provide centralised control across entire application ecosystems—from frontend applications to backend services and their APIs.

"Traditional approaches to software development, with their long release cycles and all-or-nothing deployments, are becoming increasingly unfeasible. Feature flags offer a sophisticated solution that democratises innovation while maintaining control."

MODERN SOFTWARE DEVELOPMENT IS A CONTINUOUS PROCESS

DevOps is a mindset, a culture, and a set of technical practices.



Plan
- Backlog Management
- Value Stream Management
- UX
- Lean Portfolio Management
- BDD
- Architecture
- Quality Management
- Agile SW Architecture
- DoR

Code
- Git
- IDE
- IaC
- Code Review (PR)
- Documentation
- Secret Mgmt.
- Agile SWE
- TDD
- Remote Development
- Emergent SW Architecture

Build
- Continuous Integration
- Software Composition Analysis
- License Compliance
- SAST
- Container Scanning
- Secret Detection
- Container Registry

Test
- Test Automation
- Test Data Management
- Vulnerability Management
- Compliance Management
- Audit Security Policy Management
- DoD

Deploy
- DAST
- API Security
- Scheduled Pipelines
- Environment Management
- Deployment Automation
- Staging Environment
- Continuous Delivery/ Deployment
- Production Testing

Operate
- Cross-Team Collaboration
- Proactive Detection
- Logging
- Tracing

Release
- Release Orchestration
- Feature Flags
- Canary Releases
- Dark Launches

Monitor
- Continuous Monitoring
- Full-Stack Telemetry
- Visual Displays
- Federated Monitoring
- Observability
- Metrics (DORA)
- On-Call Schedule Mgmt.
- Incident Mgmt.
- Service Desk
- Software Bill of Material

The success of feature flags at enterprise scale requires thoughtful practices. Teams must treat flags as first-class artefacts, managing their lifecycle with the same rigour as application code. This centralised approach addresses key enterprise concerns: comprehensive audit trails for compliance, role-based access control for security, and standardised implementation patterns that reduce operational complexity and total cost of ownership. By providing feature flags as a service, organisations ensure consistent implementation, governance, and best practices across their landscape while maintaining strict security and compliance standards.

The timing of this book is significant. We're in an era where digital transformation is no longer optional. Data-sensitive enterprises face mounting pressure to deliver value faster while managing risk. The implementation of feature flags requires careful consideration of organisational readiness, including developer training and process adaptation. However, when executed thoughtfully, feature flags become a cornerstone of digital transformation. This is particularly crucial for AI-enabled applications, where feature flags enable organisations to gradually introduce AI features, conduct A/B tests with different AI models, and quickly disable problematic AI behaviours—all while maintaining audit trails and regulatory compliance.

> **"The journey to modern software development is complex but it doesn't have to be overwhelming. Feature flags represent a concrete, actionable step forward that delivers immediate value while paving the way for broader transformation."**

Traditional approaches to software development, with their long release cycles and all-or-nothing deployments, are becoming increasingly unfeasible. Feature flags offer a sophisticated solution that democratises innovation while maintaining control. When implemented thoughtfully with clear governance models, they enable teams to work with greater

autonomy and confidence, transforming high-stakes releases into controlled, observable processes. This is particularly crucial for data-sensitive enterprises, where the cost of failure can be extraordinary and regulatory requirements are stringent.

I've seen feature flags consistently emerge as a critical topic in modernisation discussions. They represent a practical first step towards more sophisticated development practices by providing a safety net for confident experimentation, effective risk management control, and the flexibility to respond rapidly to market demands—all while maintaining the security and compliance standards that enterprises require.

This book builds upon the banking modernisation playbook, offering a detailed roadmap for organisations ready to embrace modern development practices. The journey to modern software development is complex but it doesn't have to be overwhelming. Feature flags represent a concrete, actionable step forward that delivers immediate value while paving the way for broader transformation. The investment in feature flag infrastructure typically pays for itself through reduced deployment risks, faster time to market, and decreased incident response times.

Let's embark on this journey together, exploring how feature flags can help your organisation bridge the gap between where you are today and where you aspire to be tomorrow.

Romano Roth
Global Chief of DevOps at Zühlke

ABOUT ROMANO

Romano Roth is a pioneering thought leader in DevOps and platform engineering, with over two decades of experience in the technology sector. As Global Chief of DevOps at Zühlke, he shapes strategies across industries like finance, healthcare, and aerospace. Starting as a software engineer, he advanced to his current role, driven by a passion for delivering continuous value, quality, and automation. Romano also organises the DevOps Meetup in Zürich, leads DevOps Days Zürich, and shares his insights on his YouTube channel, featuring over 200 videos on DevOps, architecture, and leadership.

ABOUT ZÜHLKE

Zühlke is an innovation powerhouse specialising in turning visionary ideas into reality through product and software engineering as well as management consulting. With 50+ years of experience and a global footprint, we drive digital transformation in the finance, healthcare, industrial, and consumer goods sectors. Our approach emphasises excellence, agility, and collaboration, making us a catalyst for sustainable growth and innovation. As a leader in the DevOps community, Zühlke stands at the forefront of pioneering practices that significantly boost operational excellence and efficiency. We transform organisations for a sustainable future.

THE ROI OF FEATURE FLAG MANAGEMENT

At a recent talk in London, our co-founder and lead front-end developer, Kyle asked a room of ~120 developers if they were using feature flags. Only about 40% raised their hands. This is pretty standard. Feature flags are a missing part of the equation for many companies.

But there's an opportunity cost to this. Feature flags allow you to add a layer of security to your releases—while lightening the load of traditional deployment strategies. They allow you to reduce the risk of large-scale failures (as well as improve time-to-recovery) and become more agile, opening the door to greater innovation.

In this November 2024 survey of our enterprise customers, we wanted to get to the heart of what it looks like when companies open this door.

Risk reduction

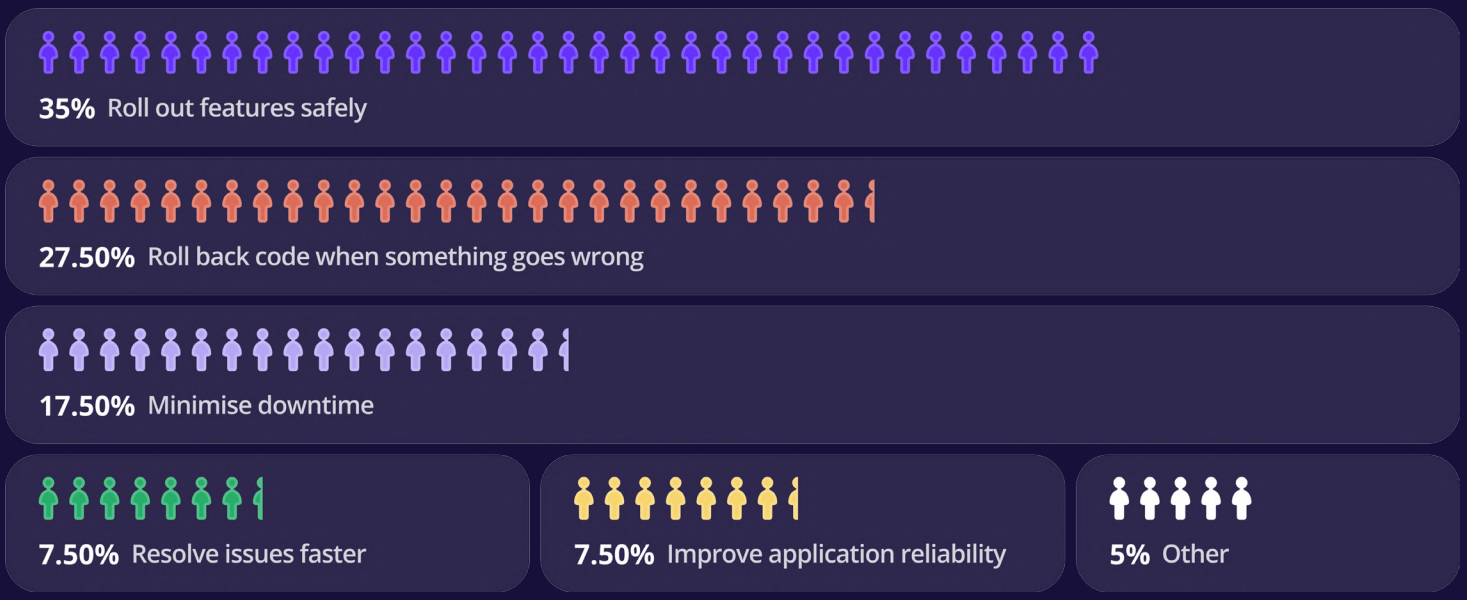
84%

Releases feel **less risky**

What does this actually mean?

Feature flags reduce risk in a really simple way. Wrap code in a flag, roll it out to users in a controlled way (e.g. with canary deployments) and see how it performs. If anything goes wrong, catch it in real time and roll back without needing to redeploy.

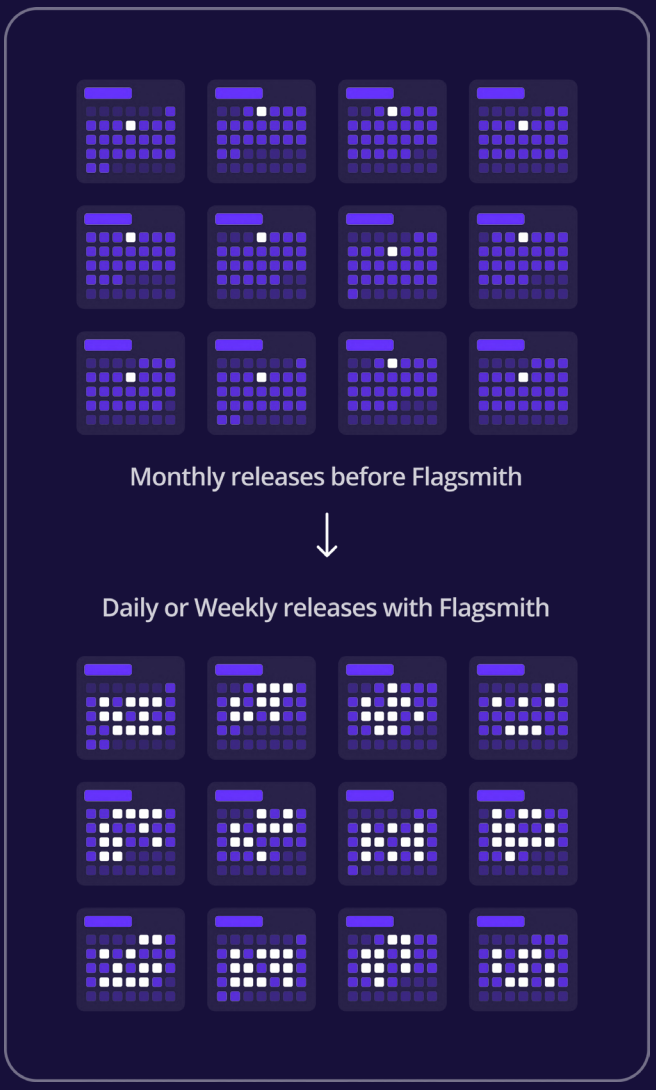
What is the *number one* way feature flags have helped you reduce risk?



Development velocity

50%

of customers that released **every month** before Flagsmith, now release **every day** or **every week** using Flagsmith



What does this actually mean?

By decoupling deploy from release, developers can push their code when it's ready. Code quality goes up as developers push code they're confident in, and risk goes down. For a chunk of teams, this leads to more frequent releases. For the other chunk, regulations mean monthly releases are still standard (though many will still deploy more regularly, especially to pre-production environments).

What is the *number one* way feature flags have helped reduce lag time between finishing and releasing code?

45.5%

Feature flags have made internal release processes better

Development teams are spending less time solving merge conflicts, dealing with code freezes, waiting on pull requests, and on testing.

45.5%

Feature flags have made cross-team collaboration easier

Teams slow each other down less. Product isn't waiting on engineers to roll out features or adjust config files. Development teams can release their code when it's ready rather than waiting on other teams whose work is somehow coupled to theirs in a release.

9.09% **No Impact**

Product innovation & pushing new features

What is the *number one* way feature flagging has contributed to feature innovation?

57.14% Encouraged experimentation

21.43% Improved end-user satisfaction

14.29% Helps stay ahead of the competition

7.14% Helps keep up with the competition

What is the *biggest impact* development teams have felt?

25% Developers are more confident just shipping features

25% Developers are less worried something could break

18.75% Teams are working better together (developer teams but also developers working better with Product or Marketing)

15.63% Team morale and developer experience feel better

9.37% Teams seem to have more creativity and a willingness to experiment

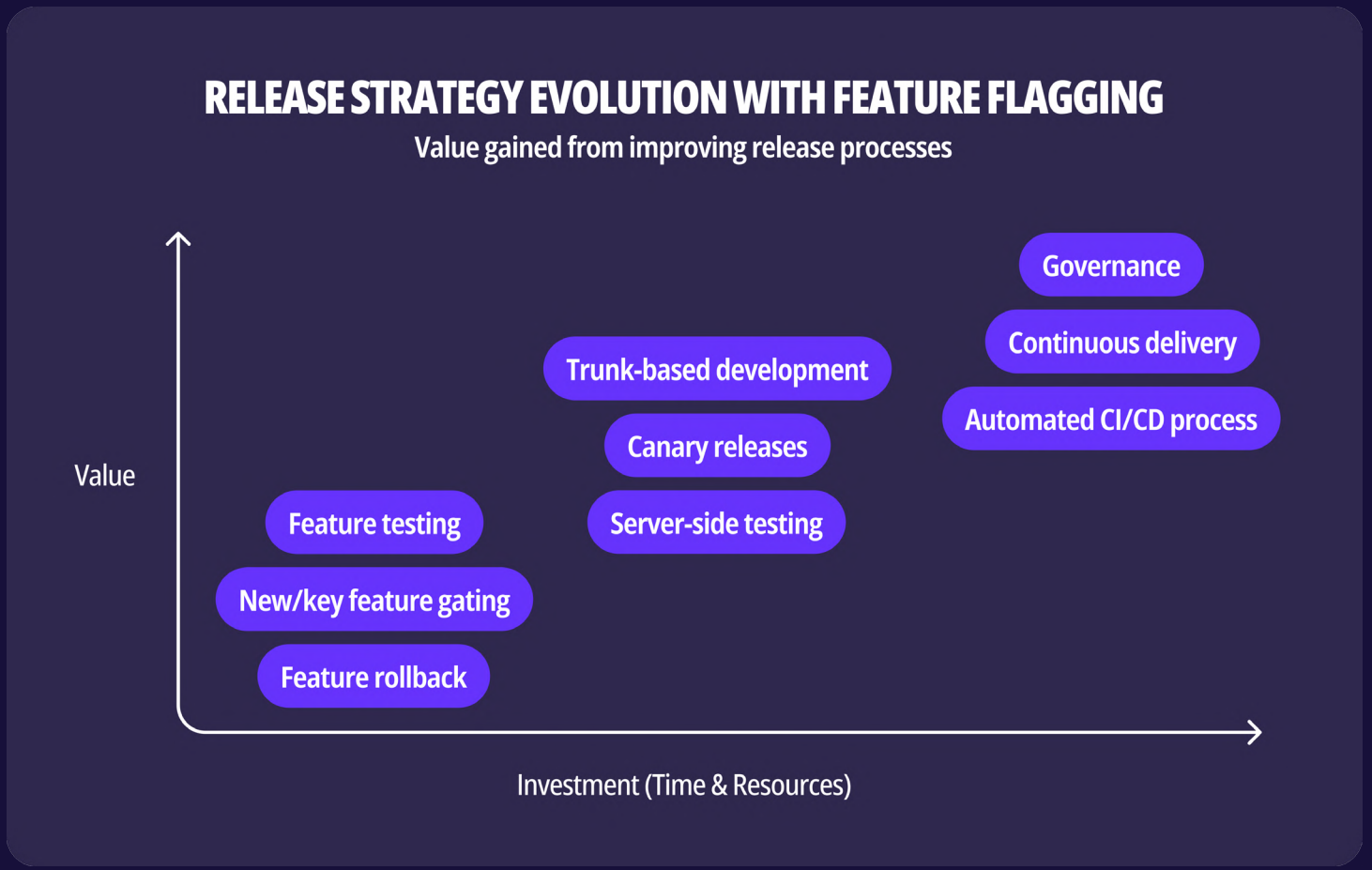
6.25% No Impact

FEATURE FLAGS FOR DATA-SENSITIVE ENTERPRISES

Stability and predictability are two of the most important elements of building software. And when it comes to highly-regulated industries like banking, government, and healthcare, this is even more pronounced. This can mean these organisations are slower to modernise, fearing change—and fearing risk.

But predictability can have a downside: if your company's deployment practices are predictably slow and painful, you can block innovation and open yourself up to more risk, rather than less.

Feature flags, which work equally well in monolithic architecture or microservices and on the front end or back end of your software, can serve as a first step towards the holy grail of CI/CD, or simply as a way to de-risk your current large-scale releases. In the following chapters, we'll explore key ways feature flags can help you on your modernisation journey. We'll also share some first-hand accounts of how large companies in data-sensitive industries have used feature flags to modernise.



Removing development bottlenecks with feature flags

At first glance, a feature flag is just a Boolean that allows features to be turned on or off for individual users or groups without redeploying code. However, this simplicity belies the powerful ways feature flags can also help remove development bottlenecks, many of which stem from legacy systems that necessitate large-scale, infrequent releases and too many dependencies.

Legacy systems can mean that engineering teams can't release as often as they'd like, and they're stuck with monthly—or even quarterly releases. Enterprise releases are already high-stakes, but if they're also infrequent, they can end up as catch-alls, with the potential for error only increasing.

If a release gets pushed back because of a hard dependency, developers feel the strain. The next release gets bigger, there's increased potential for issues, and it can end in a Hail Mary, which no one wants.

By wrapping a feature in the safety net of a feature flag, developers simply deploy whenever their work is done and release when a feature is ready. By decoupling deployment from release, releases get smaller, easier, less risky, and more frequent.

Here are some typical reasons releases get pushed back and how feature flags prevent them:

Before feature flags	After feature flags
The back end or front end isn't finished in time for a scheduled release	Individual developers can push code into production when their work is done, and can release remotely when other teams are ready
Lots of waiting on pull requests	Pull requests go through more quickly, reducing merge conflicts
QA or teammates aren't ready	Engineers don't have to wait for QA or other teams whose work might be coupled to theirs
Promoting features through many development environments	The number of development environments can be reduced, letting developers test in production, or as close to it as possible

FEATURE FLAGS FOR MODERN RELEASES

Using feature flags to *de-risk* releases

In software development, every new feature is a leap into the unknown and brings up questions like: will this work as intended, is it going to be well-received by the users, will it introduce new bugs?
Feature flags form a shield against these uncertainties.

Improve the safety of releases

Risk mitigation is a critical part of software development. This was reinforced when, in a now-infamous incident, the cybersecurity company CrowdStrike brought the global business community to its knees with a flawed release that impacted PCs worldwide. In addition to wiping billions of dollars off the company's market cap, it exposed the danger of foregoing modern release tactics.

The crisis could have been avoided using feature flags really simply and easily via canary deployments and kill switches:

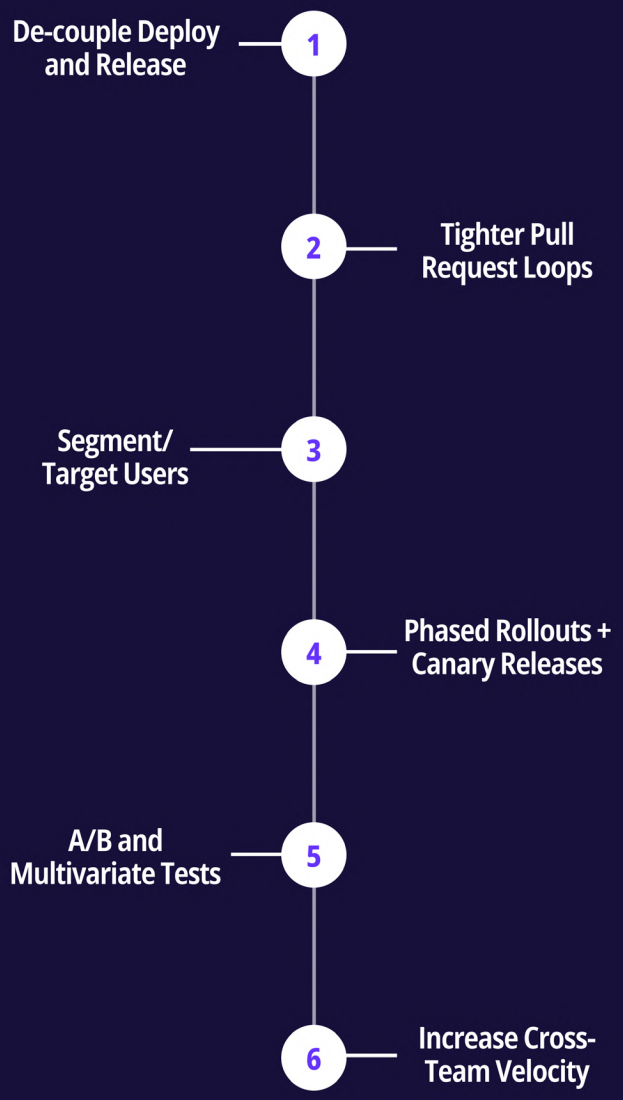
Canary deployment

Using a canary deployment, which is a progressive rollout that allows you to release code to a small subset of users before gradually scaling up after checking its viability, the team would've immediately seen the blue screen of death and halted the rollout.

Kill switch

Another critical tool is the kill switch.

If something is broken in your code and it's wrapped in a feature flag, you can immediately disable it by turning the flag off with the click of a button. This has an added benefit of allowing you to continuously progress in your code commits, rather than having to remove the code at issue or redeploy. This way one issue doesn't stall the whole development path in its tracks. Plus, you can roll it back instantly rather than expose more and more users to buggy code.



Faster recovery

When things don't go as planned, instead of rolling back an entire deployment or trying to cherry-pick what to roll back, you can just toggle the offending feature off. This keeps your services stable and your users happy. And if you need to isolate an unstable or underperforming feature during incidents (e.g. downtime, cyberattacks, etc.), you can do so without bringing down the entire application.

Using feature flags to release more frequently

Release faster and with more confidence

By separating deploy and release, feature flags enable individual developers and teams to simply push code when they're ready instead of waiting around for the next release window.

This prevents:

- A buildup of extra stress during each release window, likely causing rushed code and potential bugs
- Extra stress on developers as releases get bigger and bigger
- Having to redeploy code to make changes

In addition to this, feature flags allow for numerous other efficiency gains:

Reduce the number of development environments

Feature flags naturally lead towards a mono-environment methodology and cut down on the reliance on environments, letting developers release straight to production—or in the case of highly-regulated companies like banks, pre-production. The money saved maintaining the environments can allow you to invest in the resilience of your production environment.

Combine features from previously disparate teams

Say you've got web and mobile teams that are running in isolation—working on the same features, but for different platforms. This lack of communication can lead to bugs and also a lack of parity. Having a single feature flag project can act as a talking point between these isolated teams.

Improve collaboration between technical teams and non-technical teams

In large organisations, a lot of engineering time is spent coordinating with non-technical team members like product owners (and vice-versa). This could look like Product relying on engineers to manually adjust code to run a beta test or try targeting a specific user segment for a feature rollout. By giving non-technical team members the ability to manage feature releases, this massive bottleneck disappears, freeing up engineering time for building your product.

Using feature flags to stay ahead

Build a stronger product

In large enterprises, release cycles tend to be long and slow, whereas the market is always rapidly evolving. Apart from freeing up engineering time for writing code, working with a feature management platform allows you to balance rapid innovation with system stability. By decoupling deploy and release, your team can introduce new features behind flags and enable testing in production.

Use real-world data to test and improve features

Feature flags give individual developers and teams the freedom to test and iterate new features with internal stakeholders, beta groups, and different segments of customers. This freedom is enabled without risk, because features are hidden behind flags. This allows teams to use data from real users to guide development strategy.

CO-FOUNDER PERSPECTIVE

Feature flags in action

Working with complex applications and large development teams

Ben Rometsch, ***Co-Founder of Flagsmith***

Lots of technical leaders see the potential benefit of flags, but want to understand where it would make sense to bring them in. I have this conversation a lot so wanted to share what I say to them:

Feature flags are generally easier to do when you deploy them on the front end of your application because you have "if this feature is enabled, show this button, menu item, tab, etc." But it's also very common for Flagsmith users to put feature flags into server-side applications. That's where a lot of the power of flags comes in, especially if you're working as part of a larger team.

Decoupling deploy and release in microservices

Imagine you're running a microservice architecture, and you've got say 10 microservices. It can be difficult if there's coupling between those microservices, for whatever reason, like API contract coupling, version coupling, coordinating the release of a large number of microservices, etc. this can be really challenging.

So feature flags allow you to sidestep that problem by decoupling deployment from release. Let's say you want to deploy a new API contract for your microservice, so you're changing the API interface. You don't want to go through the hassle of versioning your API, because that's just working to keep the legacy code running, etc., so feature flags allow you to update the coupled services to work with the new API contract, and then they can just all flip over to the new API version at the same time because you can very easily coordinate the release of that new API contract. That's a pattern we see quite a lot with larger teams working with Flagsmith.

Make decisions with real-world data

Feature flags are powerful for A/B and multivariate tests, too. For example, you're writing some server-side code, and you're working on a new search algorithm. You have your existing search algorithm and your version two. Because you've got the Flagsmith SDK running within your server infrastructure, you can add a simple condition to say: if the new search flag is on, then use the version 2 search algorithm, otherwise use version 1. You can then toggle the flag and turn the v2 on or off. You can also say, I want 50% of my customers to experience v1 and 50% to experience v2. Flagsmith allows you to connect and integrate the platform into downstream analytics providers, so when someone uses the search algorithm, you can measure the performance of this.

A lot of A/B testing tools generally focus on doing cosmetic changes, so front-end applications, but because your feature flag SDK is built into your entire application, it's very easy for you to test any aspects of a new feature, just by doing a percentage split.

Phased rollouts

Let's say you're worried the new search algorithm is going to be computationally expensive, and you're worried it'll put too much load on your search infrastructure. Instead of saying, I want half my customers to get the flag as true and half as false, you can say, I want 1% of my customers to get the flag as true, so you can enable 1% of your customers and then check that you're happy with the performance of the search engine. You can gradually roll out and increase that percentage. Because you're doing all this stuff remotely, you don't have to redeploy your application, stop anything, or roll anything back, you just do all the work from the Flagsmith interface. If you're happy with the performance, you can just enable it for everyone from Flagsmith.

FEATURE FLAGS FOR MIGRATIONS

Whether you're planning a move from a monolithic architecture to microservices, merging two apps, or migrating users and components from a legacy system to a new one, feature flags can help with a smooth and gradual transition that reduces risk.

Migrating from a monolithic architecture to microservices

Moving from a monolith to a microservices-based architecture can be daunting. The stakes are high, and you don't want to disrupt your platform stability or user experience during this massive undertaking.

Feature flags can give you much-needed flexibility as you make this move, helping you slowly migrate to microservices via phased rollouts. This reduces risk by letting you evaluate if the code works before you roll it out to all your users. And if something does go wrong, you can remotely roll it back without having to redeploy any code.

Every architecture is different, but you can follow this general approach as you begin your migration:

1. Start by mapping out and understanding your monolith before you begin the migration.
2. Identify the first part of your monolith that you want to break into a microservice and create a feature flag wrapper for it.
3. Deploy the new microservice implementation of your component (wrapped in that feature flag), while continuing to direct traffic to the monolith.
4. You can then use the feature flag to test the microservice and gradually roll it out to users until you reach 100% of your user base.
5. If something does go wrong as you gradually roll it out, you can just flip the switch and roll it back until you've fixed it.

Merging two applications

When merging two apps, both the act of deleting old code and the act of adding new code is high risk. But if you keep your pull requests small and methodically merge your two code bases, you're less likely to make mistakes. If, on the other hand, you take an all-or-nothing approach, with one massive pull request attempting to merge two applications together, the likelihood of human error will be high.

Feature flags allow you to work with small pull requests, iteratively rolling out changes while keeping feature flags turned off for users. This means all your work can go on behind the scenes, with nothing changing for your users until you want it to. As you introduce the new code, you can test it in production without your customers being impacted. This means when it comes time to make the switch, there's no big scary release. You're just progressively turning on feature flags that have already been tested. And if something does go wrong when you turn on a flag, you simply toggle it off until you've fixed the issue.

GETTING STARTED WITH FEATURE FLAG MANAGEMENT

5 Best Practices

At a base level, feature flags are simple Booleans. But, even if your impulse might be to immediately jump into the most advanced feature flag use cases, we advise you to do the exact opposite. Start with simple use cases like kill switches, and as you see success you can gradually build your muscle memory until you're ready to level up.

Start by using feature flags on very simple features that are isolated. Not only with the aim of technically getting used to feature flags but also getting the whole team, including non-technical stakeholders, to build confidence in this new way of working.

You can progressively get more and more advanced as your confidence builds. You'll then naturally get to a point where you feel ready to start implementing them for more complex use cases.

With this in mind, we've brought together the most common best practices that help set teams up for success when starting with feature flags.

1. Design features around flags

Feature flag deployment should be planned early on in the software development process. If flags are simply layered in as an afterthought, they won't be as effective. Whenever you start work on a new feature, think about how you can put it behind a feature flag. The possibilities of feature flags are endless, which makes it even more important to define a flag's scope.

- Start by defining exactly what each flag will do when:
- Disabled/enabled
- If there's a remote configuration value
- Think about the rollout plan. Will you roll the feature out all at once, to segments, or to individuals?
- Avoid nesting flags >1 level deep

2. Understand the flag lifecycle and clean up old flags

Having old, unused features makes managing your flags more difficult and can lead to confusion or conflicts. Trimming unused/expired feature flags will help you and your team lessen the feature flag debt that can otherwise build up. Platforms like Flagsmith will help automate this for you by labelling old "stale" flags as well as let you know if a flag is safe to delete to help you with your pruning!

Generally, a healthy feature flag lifecycle will look something like this:

1. Create your flag
2. Build your feature and put it behind the flag
3. Deploy your code
4. Release your feature
5. Clean up the flag—or make it permanent

3. Make flags as small as possible

Multitasking is great, but not when it comes to feature flags. Keep your feature flag scope specific.

- Start small—on/off flags are the simplest way to get started. When you're comfortable with this, you can introduce more advanced capabilities
- Nest multiple flags under a parent flag if needed, but no further

4. Decide and enforce naming conventions & descriptions

Name your feature flags in a way that's so clear, that if you come back after 6 months, it's immediately apparent why the feature flag was created. This is where the power of having a well-defined naming convention comes in (it also removes the burden of decision every time you introduce a new flag!). Ideally, your names and descriptions should communicate what the feature is and what it does, as this will allow other developers to get the information they need right away.

Think about:

- Which feature a flag relates to
- What happens when a flag is changed
- Which users are affected by a flag
- Which components are affected by a flag

5. Set up access controls & audit logs

Last, but definitely not least. For highly-regulated data-sensitive businesses, this best practice is particularly important. The last thing you want is for someone to accidentally switch on a faulty feature for your entire user base. This is where access controls come in handy. These should be used to give individuals different levels of control and restrict usage based on need.

In addition to this, find a feature flag platform that allows you to maintain compliance by automatically tracking any features that are created, changed, or deleted as well as who took those actions. This is especially important if you have a large number of developers working in different teams. Having this information can save a lot of time and confusion in knowing what has been done, when, and by whom.

FEATURE FLAGS AT WORK

How a Czech bank uses feature flags in new and legacy systems

Komerční Banka, one of the Czech Republic's largest banks, moved from a homegrown feature flag tool to Flagsmith as part of a company-wide modernisation project. Prior to this, they were only able to deploy three times a year, which meant that every feature release, update, or product launch had to wait months for a large-scale, coordinated release.

This stymied innovation and slowed development velocity, with engineering time going towards managing legacy systems rather than innovating and making improvements that could impact their bottom line.

Now they use feature flags in their production and non-production environments and deploy daily to two non-production (testing and staging) environments. This means they ship new features at a much faster pace. Engineers can easily accommodate internal requests and move quickly on important updates—this has encouraged an independence that's key for their move to microservices.

Hear more from Jindrich Kubat, Head of Development & COE at KB

How a Privileged Access Management platform uses on-prem feature flags to stay secure while innovating

Delinea moves fast. Not only is it the result of a merger between two leaders in the privileged access management space, but it has continued to make acquisitions to strengthen its product offering. Its newest product, Delinea Platform, was launched in March 2023 and forms a cloud-native foundation for the company's PAM solutions.

Before Flagsmith, the team was using a homegrown feature flag service that didn't have the capabilities that engineers or PMs needed. The team needed something more robust that could help them strategically and safely roll out and test new features and capabilities.

Today, Engineering works in tandem with Product, safely testing and rolling out new features to customers using Flagsmith.

Hear more from Dariel Marlow, VP of Cloud Engineering at Delinea

"Having as much control as possible and ensuring things like business continuity, failover, disaster recovery should a system go down... these are all vital. Flagsmith checked a lot of boxes for us. It was self-hostable, it was feature-rich, and it was affordable."

Dariel Marlow VP of Cloud Engineering at Delinea

WHY FEATURE FLAG MANAGEMENT? WHY NOW?

Data-sensitive enterprises in banking, healthcare, and government are balancing two competing demands: maintaining stability and increasing agility. Feature flags let you do both. By decoupling deployments from releases, you can introduce agility into your organisation without compromising stability. Development teams can move faster, collaborate better, and build stronger products—all while reducing risk.

For companies beginning or building on a modernisation motion, feature flags can help create a structure and way of working that paves the way for CI/CD and other modern development practices.

Flagsmith regularly helps data-sensitive enterprises transition to new ways of working using feature flags, and we would be glad to chat about your team and needs and answer any questions you might have.

Contact Us

Flagsmith

Release with confidence

Flagsmith is an open source feature flag software that gives developers peace of mind. We work with data-sensitive enterprises across the world to help them transition to modern feature management and software development, offering on-prem deployments, security features, and technical support to cover your needs. We also partner with OpenFeature to support open standards and prevent vendor lock-in.

Get in Touch

"**Our development speed and velocity have increased.** Mainly, though, I just feel good about releases. I know when I ship something to production it's going to be safe and I won't have to do a thousand tests to make sure I don't miss something. When things are behind a feature flag, I know what is and isn't enabled in production and I have the visibility I need."

Vontobel
Globally active investment firm with Swiss roots

WE WORK WITH BANKS, HEALTHCARE, AND GOVERNMENT AGENCIES ACROSS THE WORLD

KB creditas Vontobel okta citi :ubank

Flagsmith.com