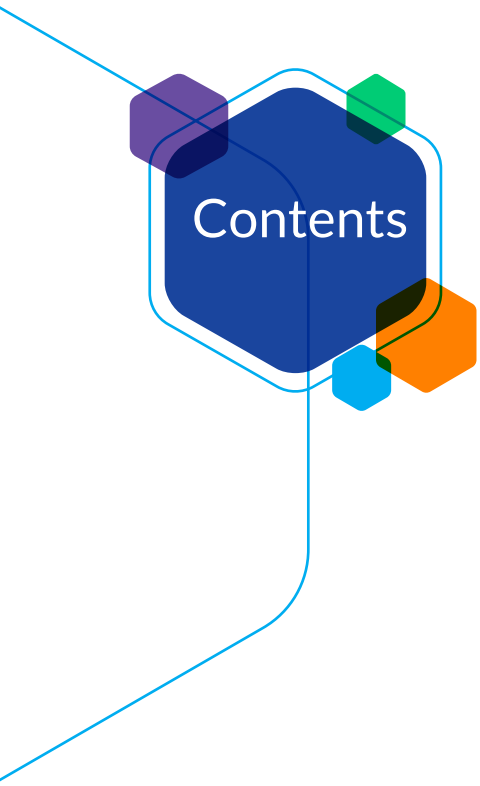
CloudBees.

# Feature Flags Across CI/CD



# Introduction

## There's no question that continuous integration (CI) and continuous delivery (CD) make life easier and more productive for software teams. Quality and customer satisfaction increase when you give them tools that deliver code faster.

While CI/CD is no longer a new process for many companies, the concept of feature flagging and progressive delivery creating the "next generation" of CI/CD is something that few companies have embraced. Feature flags are one of the tools that make your CI/CD pipeline work better. They help your team shepherd features through your pipeline safely, quickly and effectively. They make your team more productive while shortening the feedback loop with end customers.

At CloudBees, we not only believe that feature flags will directly impact how CI/CD is done in future years, but that these changes will be the foundation for all companies delivering software progressively in the future. This whitepaper is designed to give you the background needed to understand how feature flags impact CI/CD processes. After reading it, you will be be able to answer the following questions:

- How should I implement feature flags into my software delivery pipelines?
- When or where in the software delivery lifecycle should I use feature flags?
- How can feature flags help me innovate, improve developer productivity or minimize risk?
- How should my DevOps teams work together to use feature flags?

### What Is CI/CD?

CI means integrating code changes from development teams, ideally more than once a day. Developers frequently push their code changes to a source code repo. Automated tools merge the changes, trigger a build and run automated tests.

CD is preparing software for delivery to environments often and automatically. Like CI, CD should deliver code several times a day. It relies heavily on automated tools for testing and packaging software.

CI and CD have many benefits because when they work together, they shorten the software development cycle. Product teams receive a constant stream of feedback on their work. You release fixes faster since you no longer wait for periodic releases. They also reduce errors and bottlenecks since you automate all the repetitive tasks in the release process.

### What Are Feature Flags?

Feature flags are switches that let you change application behavior in runtime without redeploying. They're tools that enable you to deliver new features to customers quickly, safely and without building or deploying new code.

A feature is a collection of software functionality. It can be as simple as a new value in a display screen or as extensive as an entirely new set of capabilities in a mobile application. A flag is a variable that's either true or false. Your application code evaluates a banner and decides how to proceed based on its setting.

So, a feature flag is a variable that surrounds a feature and switches it on or off. Feature flags are typically managed by an internal "homegrown" management system or a third party enterprise system like CloudBees Feature Management with richer feature sets and scalability.

# Feature Flags and CI

Continuous integration and feature flags look like a match made in heaven. Integrating code regularly, combined with the ability to isolate new features and turn them on and off at will, means you can move fast and still manage risk.

But with great power comes great responsibility. Feature flags require careful planning and continuous management, just like your builds do. Let's review what CI without feature flags looks like and cover what you need to know when you add them to the mix.

## Traditional CI Without Feature Flags

Continuous integration helps your team move fast. But without feature flags, that speed could become a liability. Let's look at a few reasons why.

**Broken Features, Emergency Fixes and Rollbacks**

There's no such thing as error-free code. Bugs will make it into your builds and many of them will make it past your testing, too.

This reality means you're going to integrate new features, and some of them aren't going to work. Without feature flags, you have three choices when it's time to fix them:

- Roll back to the previous build
- Code and integrate an emergency fix
- Leave the broken feature in the wild until it's fixed in a regular release

**Long-Lived Feature Branches**

How do you avoid the problems outlined above when you don't have feature flags? One way is to slow down on integrating code. Maybe the developer is waiting for the time to write more unit tests. Or, maybe the risk of releasing the new feature isn't worth it without adding in a few more. So, the feature lives on a branch that isn't integrated. In other words, you've stopped implementing CI and this becomes a problem.

**Extended User Acceptance Testing**

If you want to stick with CI, you can try another approach: Verify your features in User Acceptance Testing (UAT). How long does that take? It depends on your features, your users and your test plans. You may be able to quickly integrate code and deliver it to UAT, but your developers will miss out on getting timely feedback from production.

What happens when a test fails? You have the same three choices as above. Roll the release back, code an emergency fix or kick the can down the road. Either way, without feature flags, your code spends a lot of time in UAT instead of in front of customers.

## CI With Feature Flags

Feature flags add a belt and suspenders to your CI. The risks new features add are mitigated with toggles that manage when new code is turned on or off. But that power doesn't come for free.

**Flags Conceal Broken Features (for Better or Worse)**

Flags are a way to turn a broken feature off. That's why they exist: to make it easy to enable or disable a block of code. They diminish the risks involved with introducing a new feature into a build. You have a new choice when a feature fails in a build, in a test or in production. You can simply turn it off.

But this ability comes with some costs. How long will that feature stay turned off? Have you created a block of dead code as an alternative to an abandoned feature branch? When does a working flag transition into a permanent feature?

This is a common concern with feature flags. They can quickly turn into a way to accumulate technical debt. Without a process for managing a feature flag's lifecycle, they can be a liability. Enterprise feature flag management platforms contain features that help to easily track and manage a flag's lifecycle, allowing for the benefits of instant feature disabling with less potential for technical debt.

**Flags Off Until a Feature Is Complete — and Ready for Release**

CI works only when developers integrate early and often. Feature flags facilitate this since your default posture can be to keep a feature turned off until it's complete. The flag replaces feature branches discussed earlier and avoids the large, risky merges caused by delayed integration, known colloquially as "merge hell" by development teams.

But when exactly is the feature turned on? Is it enabled for the build and disabled for testing? How does a feature deemed ready by a developer become part of a release? Who's responsible for managing flag values?

Do you have tools that make it easy to manage them? An enterprise flag management tool will help you manage feature flags. It will provide you with a shared view of flag state across the enterprise.

**Continuous Integration Has Consequences**

Your developers are pushing code daily, often two or three times a day. They're getting valuable feedback, and you're delivering value at a rapid pace.

But it bears repeating: There's no such thing as error-free code. What happens when a build fails? What happens when that failure was because of a feature flag setting? How easy is it to toggle the feature and rebuild? How do you know which features to toggle?

Feature flags are a tool for managing complexity, but they come with a cost. They can hide complexity, which isn't always good. They can also add to it since the features they're toggling may have interdependencies. The good news here is that enterprise flag management tools make it easy to visualize and work with the dependencies of flags within a build.

**Test Features With Targeted Customers**

When you combine feature flags with CI, your velocity increases. New features can be pushed through the system quickly and sent to the clients that need them on a targeted basis. You can work with a customer to test a new capability that they need or take advantage of a good working relationship and ask a client to work with you on new features.

**Add Speed and Safety to Your Continuous Builds**

As we've discussed, adding feature flags to CI can make it easier to manage change. But flags aren't free. You need to have a plan for how you're going to manage them and a way to deal with the complexity that new features always add, regardless of how they're added to your code. An enterprise management system like CloudBees provides you with the tools you need to successfully integrate feature flags with CI.

# Feature Flags and CD

Continuous delivery (CD) is not a new concept, yet it is one that few companies have fully controlled. Why? Simply put, CD can be really hard to do well. Increasingly complex testing requirements, shifting infrastructure and ever-changing customer needs are some of "usual culprits" to common CD speedbumps, but what are release teams to do? Customer expectations for better features, delivered faster, only grow by the year.

While continuous delivery can be difficult to fully implement in complex organizations, feature flags provide "guard rails" for teams wanting to start or grow their current CD program. Teams' focus can move beyond moving bits from different target environments to growing customer value, with shared visibility and governance across Dev and Ops along the way. Let's dig in and discuss the problems and risks of CD without feature flags, how flags make it easier to implement CD and what it takes to achieve continuous deployment. Then we'll wrap up by going over an example release scenario with and without feature flags.

## Benefits of Feature Flags With CD

**Feature Flags Make It Easier to Maintain Release Velocity**

CD means moving fast. If you have to slow down to finish a feature or, even worse, reverse a release because of a bug, you've lost velocity.

With feature flags, you have more options. If you haven't finished coding or testing new code yet, you can leave it turned off until it's ready. If one feature fails in testing, you can still proceed with the rest of the release. Most importantly, you never have to roll back because of a feature with a flag.

Feature flags decouple features from each other. So one feature can't delay an entire release. They can be addressed on an individual basis. At the same time, you're able to maintain velocity.

**Release Agility and Flexibility**

By severing the connections between features, flags provide you with more flexibility in handling a feature once it's been delivered. You also have a choice on where and how you test it. You can verify some features with unit testing. Others can be tested quickly and effectively by your QA teams in a test environment.

But you're better off testing some features in production. Maybe they're targeted to a specific client who'll be the final word on when they're ready. Or perhaps you just can't be sure on performance and user acceptance until they're out in the wild.

With feature flags, you have the flexibility to roll out features to select customers. You can turn them back off when they fail, or you can turn them on for everyone when you're sure they're ready. You can even tie them to different service tiers for feature provisioning.

CD focuses on delivering code quickly to different environments at scale. Feature flags add in the ability to focus on individual features instead of releases. This means you can spend more time delivering quality to your clients.

This attention can take the shape of a canary release that gradually enables new features via flags instead of staged deployments. Or with feature toggles that set up A/B testing to find out which version your clients prefer.

**Continuous Deployment vs. Continuous Delivery**

If you've had a taste of what continuous delivery can do for your team, then you've probably imagined how robust continuous deployment might be.

But the idea of automatically deploying code as soon as it's committed still sounds risky. You're still worried about what might make it through your automated testing, regardless of how robust it is.

Feature flags give you an extra layer of safety. Developers disable incomplete code until it's ready. You can target new capabilities to environments or customers that are prepared for the risks. And, as we've discussed before, you can switch failed experiments off until you have a fix for them. Even if continuous deployment is possible, you'll have to decide if it's right for your organization and your users' preferences. But feature flags provide the safety to make this a viable option.

## Example Release Case

Let's review what a release looks like with and without feature flags. Let's imagine a release with a handful of bug fixes and three new features.

**Without Feature Flags**

*Merging the Feature Branch*

The bug fixes included in this release are already on the main code branch. A few of them were already delivered as patches to a few systems, and part of the purpose of this new release is to get everything in production on the same baseline.

But without the ability to toggle a feature on or off, the new features haven't been delivered yet. They're on a feature branch or, even worse, not in the code repository yet. Someone needs to merge them into the main branch.

While the development team was working on the new features, a few emergencies came up. That's where the bug fixes and emergency patches came from. So what was supposed to be a simple merge has evolved into something more complicated than expected. How long will the merge take? How many new problems will it introduce?

*Extended QA Time*

Once the development team has pushed and built the new code, the CD pipeline delivers it to the test environments. Even though the bug fixes have already been tested and are in production, the new features are not.

QA immediately finds an issue with one of the features, and development rapidly delivers a fix. But, just as it looks like the release is ready to go, a more serious problem is identified.

Is it time to back out that feature and push ahead with the release? How long will that take? Or is it better to hold off entirely until development can address the issue? What happens if another emergency crops up?

*Problems in Production*

The release has made it to production, and there's an issue with one of the new features. Is it time to roll back? Or push an emergency fix for the new bug? Depending on the scope of the problem, when the problem occurs, and the difficulty of the fix, an emergency might be the beginning of another cycle. If development is already working on a feature branch, this fix might turn into a merge headache down the road.

**With Feature Flags**

*No Merge*

With feature flags, development adds toggles to new features when they start their work. There's no need for feature branches. So the emergency fixes that cropped up while development was working on these features don't cause any conflicts. There's no need for merges. "Releasing" a new feature means turning it on. Features are in turn decoupled from one another at release, meaning a new feature can be turned on in the trunk without affecting the others. This "decoupling" of features creates additional velocity of releases without dependencies.

*Testing New Features? No Problem.*
The new code is built and delivered to the test environment. One feature has an issue, and just like before, development pushes a quick fix.

Then, a problem with a second issue crops up. It's going to take longer to fix this one. So that feature stays toggled off, and the release proceeds to production.

*Testing in Production With Flags*
There's another path with feature flags. You have the option to test your features in production.

With tools like CloudBees, you can target specific customers and work with them to verify a new feature, without any impact on the rest of your clients. You can also use similar targeting capabilities to run A/B tests to find out which version of a product or feature your clients prefer.

*New Release Instead of Rollback*
Any issues encountered in production look just like testing in production. If one of the new features fails, turn it off and decide how and when to address it. Compared to the "all hands on deck" rollback of a feature, this is a very simple way to release and remove features for development teams.

# Integrated Feature Flags Across CI/CD At Scale

As we've now shown, there's no question that feature flags are one of the tools that make your CI/CD pipeline work better. They help your team shepherd features through your pipeline safely, quickly and effectively. They make your team more productive while giving them feedback on how they're doing at the same time.

However, there are  some of the pitfalls and best practices you should consider with feature flags and CI/CD.

## Use Them or Lose Them

If you want to turn a feature on or off, it needs a feature flag. So, developers must use the flags. It doesn't matter how trivial or complicated a feature is, flags must become a standard part of software creation for all developers to get maximum impact of their benefits.

A fix or enhancement to an existing feature might be an opportunity for a flag, too. Adding the toggle may duplicate some code, but the ability to turn the new code on or off is often worth it. The extra code can be factored out when the flag is retired.

## Feature Discrimination

How big is a new feature? The answer may change when you add feature flags to the mix.

For feature flags to work safely, the features need to operate independently. Development and product management will need to work together to make sure that toggling one element doesn't leave another in a bad state.

The initial response here is to put a single toggle around a large set of features.  But that defeats the point of both feature flags and CI/CD. It's reverting to the waterfall process, with added complexity.

The right approach is to break the behavior down into smaller chunks and introduce them into the CI/CD pipeline gradually. You need to change both the way you specify features and how you released them. But again, enterprise flag management platforms have taken this into consideration and allow you to track dependencies between flags and features.

## Flags Are Part of the Release and Rollout Process

When you add a flag to the codebase, it needs to start as toggled either on or off. Then you have to communicate the state to the operations team. The flag's state is part of the software release process. It's an important release artifact, not a footnote or undocumented feature. Its state will change. It may be toggled enterprise wide or for different environments or clients, depending on how it's used. Here again, the operations team needs to know.

An enterprise-wide view of feature flag states is a critical resource. Feature flags can become a liability without shared governance, and for that reason, enterprise flag management platforms are focused on creating pipeline-wide shared visibility of flag status across all teams.

## Flags, Like Fish, Begin to Smell…

Feature flags have a lifecycle, whether you plan one for them or not. If you don't plan, that life cycle looks something like "forever." Depending on what kind of flags they are, that means they turn into a code smell.

You can put flags into at least four categories: release, experiment, ops and permissioning. Release flags should have short lifecycles tied to the feature they toggle. When the feature is complete, you remove the flag. Experiment toggles can live for a long time since you can repeatedly refine the experiment. Permissioning flags might live forever, so the code they control needs to be designed with the flag in mind. Having a flag management tool that can determine which type of flag is which, and provide recommendations for flag removal or monitoring helps eliminate the bloated code headaches created from feature flags.

## Added Insights

CI/CD means constant activity. Code moves through a pipeline that starts at each developer's desk and ends in production. How do you know what's working and what's failing? Feature flags can help with that.

**Mapping Flags to Tests**

The first step is to use your feature flags in testing. Each test is an opportunity to toggle a flag and find unexpected side effects and undesired dependencies. This is true at CI time, with automated unit tests, and between delivery and deployment with your integration tests.

Testing doesn't end when code transitions to production, though. The results of A/B experiments, alpha and beta testing and staged rollouts are all made possible by adding feature flags to your CI/CD toolbox.

**Performance Testing and Reporting**

If you need to know how new code will impact performance, add a flag, even if it's not a new feature that clients will know about. Now you have a new tool for evaluating performance. You can create a new A/B test or conduct a staged rollout. You can run your tests based on the state of the new toggle, and you can use them to evaluate performance reports from clients.

**Shared View of Flag States**

We mentioned how critical it is to share and agree on flag lifecycles and states across all product teams. But a common view of feature toggle status is also an asset. When all stakeholders have visibility into a flag's state, they all have an opportunity to apply it to their reporting and gain new insights. This is true across product management, development, operations and the executive teams viewing the data this is funneled into.

## Feature Flags and CI/CD: Better Together

There was a time (not so long ago) that many companies never thought they could be delivering software as quickly and efficiently as they are today utilizing CI/CD methodologies and technologies. Now, it would be difficult to find a company that isn't utilizing some level of CI/CD, or that doesn't at least want to. The same evolution will occur with feature flags evolving CI/CD to the next state of "progressive software delivery" where every feature at every company is flagged to achieve the benefits across CI/CD we mention here.

However, this evolution will not occur at the flip of a switch. In most companies, it will be a process over time. This process starts with understanding feature flags and their benefits. Hopefully, after reading this paper, you have a better idea of their value, along with the organizational shifts that must occur for their scaling within a company. Next is the fun part: adding them into the tools and processes you currently have!

As we have discussed, feature flags have clear benefits within software development and delivery, but they do not come without drawbacks, such as the potential for technical debt and added complexity across development and operations if not managed properly. Common visibility, governance and auditability of feature flags across CI/CD toolchains is key for long-term, scaled adoption across teams.

CloudBees Software Delivery Automation allows DevOps teams to gain the benefits of integrated feature flags across CI/CD with as little upfront effort as possible. Shared insights, visibility and governance ensure that feature flags can be scaled as seamlessly as possible.

## Learn More


- **Read the Whitepaper**
  *The Definitive Guide to Modern Software Delivery*
- **Read the Whitepaper**
  *Grow Continuous Delivery Maturity Using Feature Flags*
- **Watch the Webinar**
  *Release Management in the Progressive Delivery Era*
- **Read the Whitepaper**
  *Feature Flag Management: Should I Build or Buy?*

## Learn more

**www.cloudbees.com/products/feature-management**



0122v00

CloudBees, Inc.
4 North Second Street | Suite 1270
San Jose, CA 95113
United States
www.cloudbees.com
info@cloudbees.com