Grow Continuous Delivery Maturity Using Feature Flags

CloudBees.

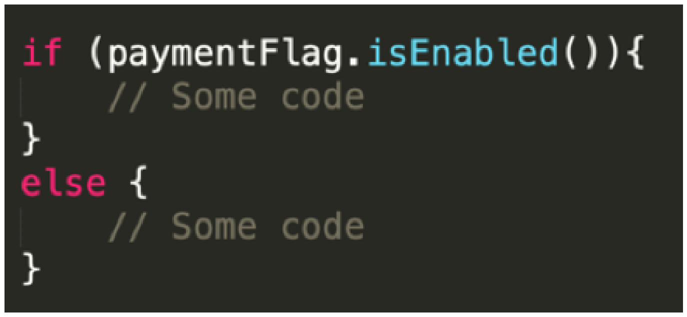
Contents

Introduction

Continuous Delivery (CD) is not a new concept, yet it is one that few companies have fully mastered. 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.

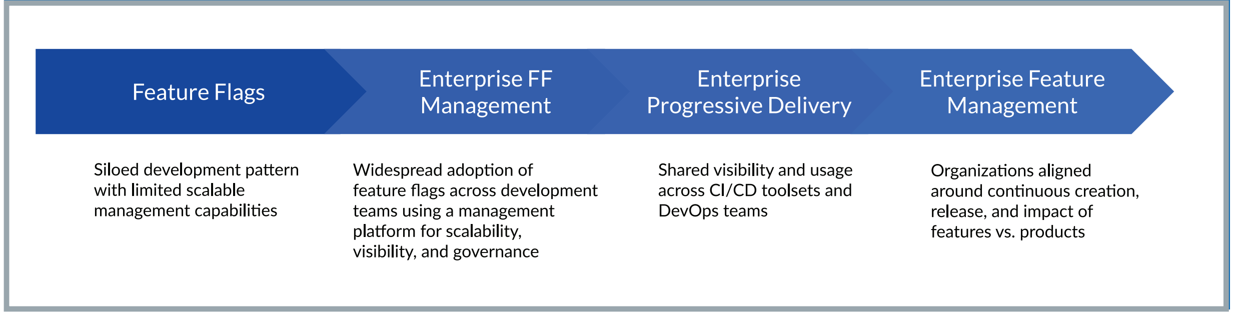
Enter feature flags. Feature flags, also known as feature "toggles" or "flippers" are, in their simplest definition, an "if/then" statement in your code that allows you to make changes in runtime without redeploying. The ability to change feature behavior without redeployment changes how software is released as it decouples technical deployments from the business decision of feature release. This has positive implications for continuous delivery, which this whitepaper will cover in detail.



```
if (paymentFlag.isEnabled()){
    // Some code
}
else {
    // Some code
}
```

Now, there is a misconception in the world of DevOps that feature flags are the “frosting on the cake” of CI/CD, in that they’re a “nice to have” item that teams should consider implementing after they feel they are mature enough with their CI/CD pipelines. When asked, many DevOps practitioners will say, “We can’t even consider feature flags right now. We’re still trying to master CI, and after that, we need to look at growing CD. After that, we’ll give you a call to discuss feature flags.” That “roadmap” approach to DevOps toolchains is understandable, but anyone in this industry will tell you that these concepts are never perfectly linear. Feature flags should actually be used as a tool to expedite CD maturity versus one that should be used after CD is in place.

Feature Flag Maturity in CI/CD


Feature Flags
Enterprise FF Management
Enterprise Progressive Delivery
Enterprise Feature Management

Feature Flags	Enterprise FF Management	Enterprise Progressive Delivery	Enterprise Feature Management
Siloed development pattern with limited scalable management capabilities	Widespread adoption of feature flags across development teams using a management platform for scalability, visibility, and governance	Shared visibility and usage across CI/CD toolsets and DevOps teams	Organizations aligned around continuous creation, release, and impact of features vs. products

Traditional CD vs. CD with Feature Flags

Before looking at how feature flags alter the CD process, let’s first think about how “traditional” CD can fall short at some companies:

- **Incomplete testing process:** Nearly all testing occurs in pre-production environments without the “real world” experience of production available to ensure the code is truly ready to go. This approach creates a slower testing process - one that can lead to unforeseen surprises to code once it hits production and the need for an emergency rollback of a release
- **“Big bang” releases:** Typically releases contain a variety of features that are deployed to production for all users at once. If one particular feature is delayed or buggy, it impacts the entire release and the whole update must be rolled back. This scenario also means bugs impact the entire user base, creating a large “blast radius” for issues seen by users
- **Limited granularity in user testing:** While some companies will perform smaller testing releases (such as a canary release or blue green deployment), the selection of users tends to be random and focused on moving subsets of users to different target environments.These tests don’t allow for experimentation based on specific user characteristics
- **Disjointed teams:** Business teams and developers must work in tandem to deploy a release to production on the same schedule. But how often are these teams fully on the same schedule? More commonly, developers are delayed in starting their next sprint as they try to get a release out the door

So how do feature flags change CD for the better? Remember, feature flags give the ability to turn sub-sections on an application on or off remotely in runtime without redeploying. For CD, this provides:

- **Release schedule consistency:** Features are individually flagged and decoupled from one another in a release, the granularity of release is now on the feature level versus the release level. When one feature is delayed, it is simply flagged off. The release can move forward as planned. Likewise, a full release never has to be rolled back due to an issue with a single feature. That feature could be rolled back with the rest of the release remaining intact)
- **An emphasis on customers:** Traditional CD is focused on the tactical delivery of bits to different target environments. But modern CD involves delivering features to very specific customers at specific times for testing, customization or experimentation. Feature flags allow you to send a new feature to customers based on any criteria you have on them in your database, be it geography, version, beta designation, language and more. Gone are the days of a generic, random canary release: features can now be created, targeted and tested with specific customer segments in mind
- **Shifting certain testing to production:** Software testing comes down to a balance of accepting risk or compromising velocity in a release. Even the best test environments cannot perfectly mirror production. Feature flags allow safe, partitioned testing in production environments that can start with QA teams, grow to beta users and eventually be progressively rolled out to live customers - with careful analytics and measurement along the way. Using flags in this way speeds up the testing process, while also giving better feedback on the performance of a feature from real users. If done correctly, it makes the release of a feature beautifully boring and uneventful and helps release managers sleep better at night

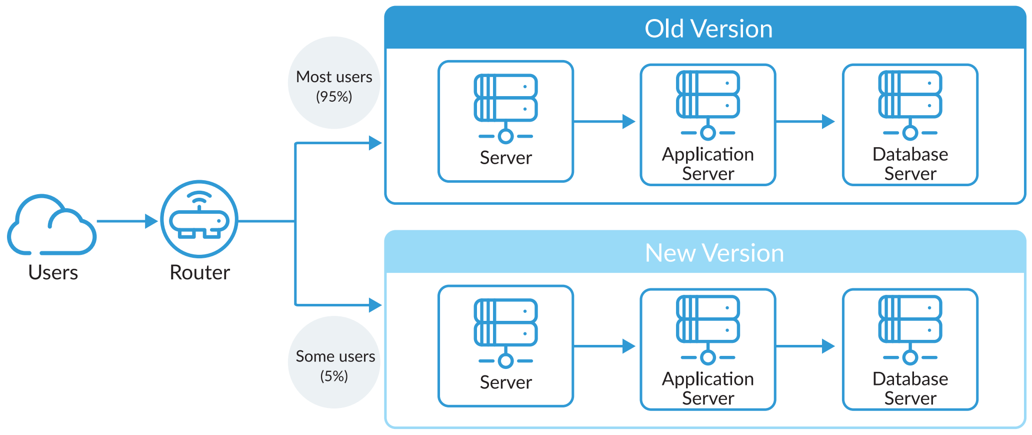
Traditional CD	Modern CD with Feature Flags
Focused on delivering releases	Focused on delivering features
Emphasis on target environments	Emphasis on customers
Emphasis on intensive pre-production testing	Greater shift to testing in production
Limited granularity of user testing based on available criteria	User testing based on any criteria available in the database
"All or nothing" rollout or rollback of a release based on feature dependencies and performance	Features are decoupled from each other within a release and can be rolled out or rolled back independently
Business teams and developers must work in tandem to deploy a release to production at the ideal time	Deploying code to production while the business determines feature launch timing decouples developers from the release process

Example: Canary Release with and without Feature Flags

To understand how feature flags impact the release process on a technical level, let's look at a canary release - with and without the use of feature flags - to see how flags improve companies testing options and their ability to roll out new features to specific customer segments.

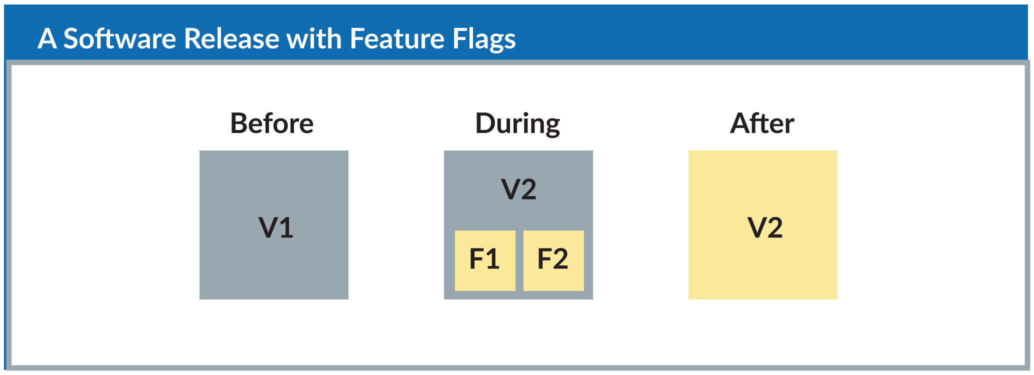
Canary releases are used by development teams to test new versions of software to small groups of users and validate the changes they've made before releasing it to their entire user base. Traditionally, this testing is executed by moving a small, random subset of users to different servers running the new software in production and observing how they interact with it. If the new version performs as intended with the subset of users, all users are gradually moved to the environment with the new version in production (as seen in the diagram below).

Traditional Canary Deployment



While canary releases are effective in giving release managers greater confidence before the entire user base sees a release, it doesn't come without downsides. Mainly, this approach requires heavy involvement from the operations team as moving users to different versions requires constant infrastructure moves and the maintenance of multiple software versions at once. This scenario becomes even worse if a version has to be rolled back! Another constraint is the targeting capabilities for which users receive the new version of software is generally limited and consists of a random sampling of the entire user base.

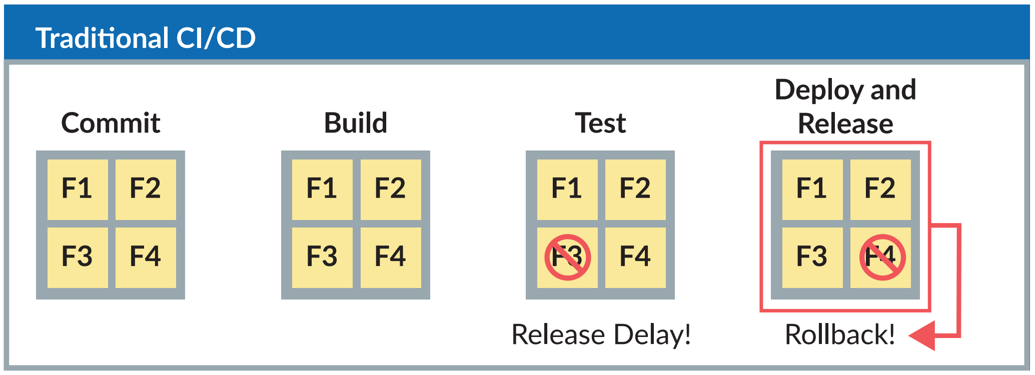
Instead of living on the infrastructure layer, a canary release using feature flags lives on the application layer. This differentiation means rather than setting up parallel infrastructure to move users between environments, the operations team deploys the new software version to production all at once. Release managers or product managers then decide when to release individual features to different users based on the criteria of their choosing, such as email address, region, language, version etc. Feature flags provide much greater granularity to which users receive the new version first. It is common for companies to release first to internal users in production, then move to beta users, then a rolling release cadence of their choosing. All the while, these organizations can analyze the performance data until all users have the new version, as shown in the graphic below. The operations team can deploy once, then remove themselves from the process.



A Tale of Two Releases

To demonstrate how feature flags change the overall dynamic of a standard release pipeline, let's consider a theoretical release using "traditional" CD processes:

- In this release, there are four independent features designed to be deployed and released all at once to customers
- In the testing phase, QA discovers a critical bug with feature #3, causing the entire release to be delayed by multiple days. The issue creates tension on the team and increased pressure from management to get the release on schedule
- Finally, feature #3 is fixed, and the entire release is deployed to production, allowing all users to see it at once. But, oh no! There was a critical bug in feature #4 that appeared in production and slipped through all pre-production testing. The entire release has to be rolled back, patched and redeployed. Customers are confused, management is frustrated and developers and release managers are swearing. Not a happy ending!

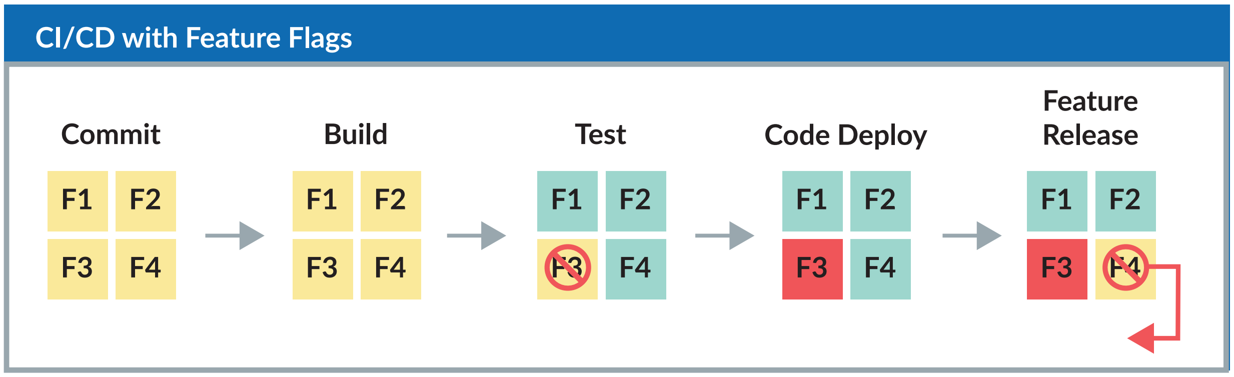


To demonstrate how feature flags change the overall dynamic of a standard release pipeline, let's consider a theoretical release using "traditional" CD processes:

» With all features now flagged, they are decoupled from one another and the "box" containing the conjoined release is now removed

» The same bug in feature #3 is discovered, but rather than slowing down the release as a whole, the feature is deactivated. The remaining features remain on schedule (feature #3 would then be troubleshooted in parallel and released days later when ready)

» Features #1,2 and 4 are deployed to production and progressively delivered to live customers on a rolling cadence. Early on, the production bug in feature #4 is discovered before the majority of new users see it. It is immediately deactivated using the "feature kill switch" capability of flags and it is fixed without having to redeploy the entire release as a whole. There was a very small blast radius for the error. Management is happy with the velocity of features delivered overall and satisfaction in and out of the company remains high. In the complex world of software delivery, this is a "fairytale ending," but one that is quite achievable in real life



Feature Flags Make Continuous Delivery Safer, and Easier to Adopt and Master

While continuous delivery can be difficult to truly master in complex organizations, feature flags provide "guard rails" for teams wanting to start or grow their current CD program. Shifting from release to feature granularity will keep release teams on schedule and avoid large rollbacks due to feature dependencies. Feature "kill switches" provide the safety of turning off any buggy features without having to redeploy code. Testing in production gives QA teams increased confidence that the code will perform as intended in real environments. Finally, providing the shipping granularity for end users based on their certain characteristics allows for increased testing or rollout granularity, experimentation and customization. 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.

CloudBees Software Delivery Automation Solutions Supercharge Your Software Delivery

Gaining the advantages of combining feature flags with continuous delivery is easiest when tools are natively connected to one another.

CloudBees Software Delivery Automation solutions are a set of tightly-integrated, proven products that enable any organization to implement established best practices in continuous integration, continuous delivery, release orchestration, feature flagging and progressive delivery, across a wide range of applications, tools and technologies

CloudBees Feature Management and our industry leading CloudBees CD solution now have native integration with one another for immediately increased CD maturity and capability described in this eBook. This integration is part of our commitment to deliver end to end SDLC value with Software Delivery Automation (SDA).

CloudBees Software Delivery Automation solutions enable enterprises to optimize their software delivery process for increased innovation and security by connecting, automating and orchestrating the tools and functions across development, operations and shared service teams.

Learn More

Watch the Webinar
Evolving Continuous Delivery Using Feature Flags

Replay the DevOps World Session
The Future of Feature Flags and CI/CD: Shifting to Feature-First Organizations

Review More Resources
CloudBees Software Delivery Automation

Learn more
www.cloudbees.com/software-delivery-automation

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