split

Achieve True Continuous Delivery with Feature Flags

Chances are you've been told how to "work smarter, not harder"—but has anyone revealed the secret to working faster? In software delivery, agility is an ongoing objective. And for product and development teams alike, figuring out how to best use the tools at your disposal to quickly get new features to end users is a key area of improvement.

While striving toward this, there's no doubt you've come across the various approaches for continuous development and release:

- **Continuous integration (CI)** is the process of merging every code change from a developer to trunk and running automated smoke tests immediately. This acts as a quality check for new code, identifying any problems and notifying the developer. Though not a prerequisite for continuous delivery, they often go hand in hand.
- **Continuous delivery (CD)**, meanwhile, refers to the ability to safely release all types of software changes — including new features and configurations, bug fixes, and experiments — to production and to end users at any time.

Achieving CD is essential for **continuous deployment**, the phase of actually getting those software changes to users automatically. CD and continuous deployment both help to decouple deployment and release from each other, so you have control of where, when, and how quickly you push code. This allows you to be as gradual or selective as you need to roll out new changes without disrupting your users or business.

But those aren't the only benefits. In this whitepaper, we'll show you how to get the most value out of your software delivery process — by implementing CD with feature flags.

Implement Continuous Delivery

There are three ways you can implement continuous delivery:

Containerizing

Break a large development project into smaller pieces, isolating and containing bits of code for teams to build and release asynchronously from one another.

Automated Deployment Pipelines

Automate the delivery process and deploy features in small but frequent increments, lowering the risk of each release.

Feature Flags

Choose between different code paths in your system at runtime, toggling features on and off, and then pushing different iterations into production and release.

These methods aren't mutually exclusive; you likely already containerize or automate as part of your delivery process. No matter where you're at right now, feature flags will help you take it up a notch. Why? Because feature flags deepen the benefits that CD provides, helping you move even faster and take greater control over your releases.

Run the Automation Engine

Continuous delivery relies on having automation in place to test and deploy code as many times a day as you need. This automation is what keeps the CD pipeline working. Developers that need to manually run all the required tests and deployments to keep code fit for release won't be able to deliver new code and features to users daily or more often, which is what allows the critical "flow" state of continuous delivery.

Feature flags help to support automation and speed up processes for deployment, release, and further testing. By applying an "if/else" control to code, feature flags streamline the process of deploying multiple software iterations without incident. As a result, teams can safely practice trunk-based development and frequently integrate code changes into the software, avoiding the issue of long-lived feature branches.

If any bugs, issues, or unintended effects arise when you release new updates, feature flags give you the fastest and most effective solution. Just turn the feature off with a click — no need for **time-consuming rollbacks or emergency hotfixes.**

Improve Software Delivery Quickly and Safely

Feature flags give developers a speed boost and a kill switch, but they also give teams the flexibility to test, deploy, and release at will by decoupling deploy from release. This is one of the most notable ways that feature flags enhance continuous delivery. For instance, your engineers can use feature flags to deploy in-progress features into production while keeping them hidden from users.

Great peace of mind and efficiency come from this — your team can tackle large development projects by breaking them down piece by piece, turning risky, sweeping, code-heavy changes into a series of smaller, more manageable tweaks. Less room for error means better results with lower stress. And once these features are finished, your engineers can use feature flags to carry out a controlled rollout.

Here's how to pull it off:

1. Deploy the feature to production, turning on access only for internal users. Your testing team can then validate the feature in the same environment as your users, with the same data and software.
2. Perform a controlled test among beta and early access users to verify the feature's performance, usability, and functional parameters.
3. Conduct a canary release, deploying the feature for just 5-10% of your user base. This allows the product team to find any additional bugs, and mitigate any errors associated with the feature.
4. Run an A/B test to better understand how your new feature affects user behavior.
5. With this insight, release the feature more broadly or press the kill switch and refine it further.

By decoupling deploy from release and powering controlled rollouts, feature flags contribute three critical things to your software delivery process:

Speed: Get small changes to users faster, test and retract them quickly, and produce better iterations

Safety: Introduce subtle changes that don't disrupt the user experience, gain control over the blast radius of each release, perform more thorough pre-release troubleshooting, and better understand the impact of each change

Sureness: Test in production to avoid surprises and stop guessing how a feature will behave once live

Boost ROI with Split's Feature Flags

When you implement feature flags with Split's feature delivery platform, you double the impacts of the investments you're already making in software delivery and add value to the process.

Reduce environment costs

Multiple "lower" environments — such as integration, quality assurance (QA), performance, and user acceptance testing — are expensive to maintain, and they don't always produce the same results as testing in production. Split's feature flags create the operating conditions to test features safely in a production environment, whether you're targeting internal QA engineers or end users. Not only does this make testing more useful, it simplifies your QA and lowers the storage, compute, and labor costs of running testing environments.

Lower costs per release

Split's platform helps you produce 4–10 times more releases per dollar spent on engineer labor. Fast and frequent delivery also helps you avoid bugs, delays, and painful merges later down the line.

Release more features, more frequently

Split's customers have used feature flags to release four times more features and reduce 95% of engineering time per release.

FEATURE FLAGS: ⬇ Operational Costs

Minimize environment costs by consolidating test environments. Lower costs per release by releasing early and often.

ALERTING: ⬇ Business Risk

Reduce MTTR by instantly identifying and killing bad features. Protect revenue by avoiding unplanned outages and downtime.

EXPERIMENTS: ⬆ Engineering Efficiency

Decrease maintenance cost of supporting low-impact features. Boost engineering productivity by focusing on high-impact features.

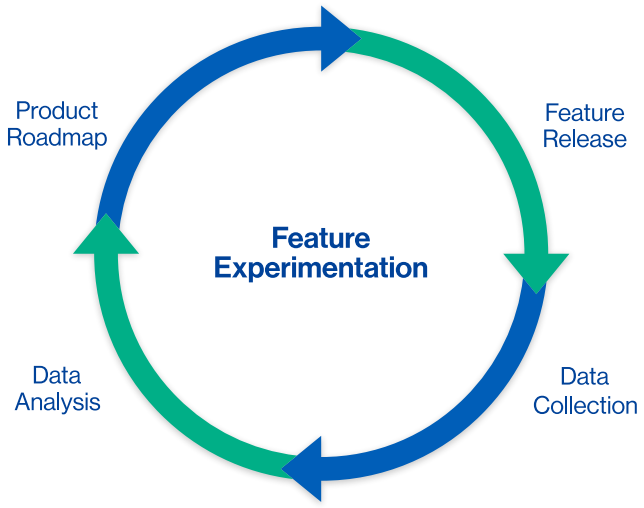
Take the Next Step: Combine Feature Flags and Data

Feature flags are certainly powerful on their own, but there is a way to make them more effective: layering in behavior and performance data. Ultimately, this contributes to an even more mature version of continuous delivery.

With business and engineering metrics on your side, you can grasp a feature's effect in real time. Whether a feature performs better or worse than you expect, impacts user segments differently than you imagined, or varies depending on the device or time of day, you'll know instantly how each change affects the customer experience — and the data will help you decide what to do next.

Heightened visibility goes a long way. These metrics let you make faster and better product decisions, clearing the path for successful experiments as you test new features and tweaks. And if a new feature ever causes performance degradation, metrics will improve your detection and recovery efforts by alerting you to the problem.

This means engineers no longer have to spend time investigating and isolating the root cause of performance issues; just press the relevant kill switch and take care of it. That's MTTD and MTTR under control.



Together, feature flags and data produce the ideal version of continuous delivery. While feature flags help you roll out features with greater agility and control, data creates a continuous feedback loop. When you run and measure small experiments on a consistent basis, you'll reach a state of constant improvement — and your features will be as attuned to your users as possible.

Go Forward with Split

This is what the future of continuous delivery looks like: using feature flags to run controlled experiments, measure the impact each change has on your users, refine your features further, and release the best versions faster.

And Split's feature delivery platform has everything you need to get there:

Testing

Run A/B tests, multivariate testing, and beta testing to identify the ideas that work — and develop a culture of hypothesis-driven experimentation, where evidence guides decisions.

Alerting

Split compares your feature flags and performance data and — within 30 minutes — identifies if new features improve or worsen the user experience. From new features to backend configurations, Split measures and tells you everything.

Analytics and integrations

Split captures data from a variety of sources (Google Analytics, Sentry, Segment, New Relic, etc.) so you can run and analyze experiments, segmenting users for advanced insights. Other integrations like automation servers (e.g., Jenkins) and SDKs (e.g., Android, Java) power continuous delivery across frontend, backend, and mobile.

Webhooks

Use Split webhooks to add feature flag data to your analytics, improve performance monitoring, and update issue trackers.

APIs

Use Split's API to integrate your favorite tools. Push data into Split, extract data from it, build custom integrations, or build on Split's platform.

Jenkins

Use Split's Jenkins plugin to create, update, and delete code splits as part of test automation and build workflow.

Achieving continuous delivery with Split means speeding up your development cycles, lowering the risk of releases, and renewing your focus on the features that matter.

*Learn how **GoodRx** uses Split to get advanced insight on feature changes and iterate new ideas faster — or **schedule a demo** to discover what Split can do for you.*