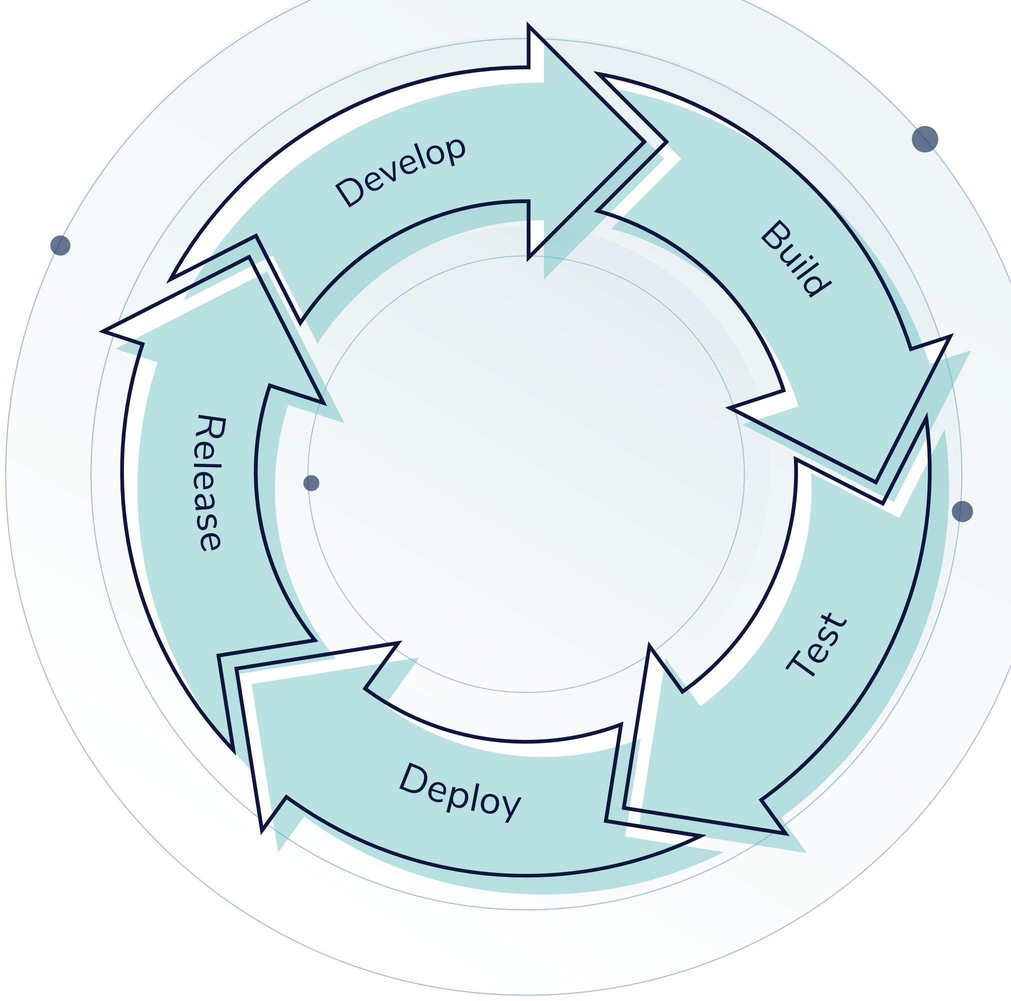
LaunchDarkly

PROGRESSIVE DELIVERY

The next iteration of Continuous Delivery centered on risk reduction, business outcomes, and control.

Table of Contents



A new approach to software delivery

Elite software development teams share this in common: they've all mastered Continuous Delivery. Companies like Amazon, Google, and Netflix get small changes—features, bug fixes, etc.—into production or users' hands thousands of times a day.[1,2] They move fast, yes. But they also learn fast.

Each change generates data on user engagement, system performance, and other key metrics. This data, in turn, fuels a virtuous cycle of iteration, wherein the quality of the product constantly improves. In this way, elite companies reap the benefits of Continuous Delivery.

Despite the clear benefits of this approach, many large organizations are still reluctant to adopt it. Why?

For one thing, Continuous Delivery often implies shipping incomplete code. That's a scary proposition when talking about a production environment for millions of users. Moreover, it embraces the reality of

[1] Companies that deploy hundreds or thousands of times a day often are engaging in Continuous Deployment in addition to Continuous Delivery. This means they have automated the deployment process. Whereas, Continuous Delivery entails keeping code in a deployable state at all times but does not refer to automatically deploying code once it's been committed. But, of course, code deployments cannot be automated unless the code is in a deployable state. All that to say, when a team engages in Continuous Deployment, it's implied that they also are doing Continuous Delivery.

[2] Forsgren, Nicole, Dr., Frazelle, Jessie, Humble, Jez, Smith, Dustin, Dr. "Accelerate: State of DevOps 2019." DevOps Research & Assessment, Google Cloud. 2019: 22.

bugs and small system failures. It accepts that some application changes will inevitably cause problems.

And such problems are acceptable, so long as they happen early and on a small scale.

This approach contrasts sharply with the Waterfall delivery method, in which developers toil for months, or years, on a set of features before finally shipping them in one big heap. Many teams cling to this approach because, in their minds, it allows them to retain control over the deployment process.

Of course, Waterfall attempts to provide such control by reducing the deployment and release frequency.

Many software organizations fear that Continuous Delivery will rob them of their sense of control over deployments and releases.

And in any case, such control is an illusion given how frequently bugs still occur in a Waterfall context. Nevertheless, Waterfall practitioners fear that Continuous Delivery will rob them of their sense of control.

Early proponents of Continuous Delivery introduced techniques to quell such fears. Practices like blue-green deployments, canary launches, and ring deployments[3] enable teams to continuously deliver in a controlled manner. But many teams either don't know about these techniques or fail to employ them.

Moreover, while such practices were born in a Continuous Delivery context, they were not stressed as heavily as other aspects of the methodology. As a result, large organizations are left with the false impression that Continuous Delivery is dangerous.

Other practices have since emerged that build upon the initial risk-reducing techniques. Collectively, these

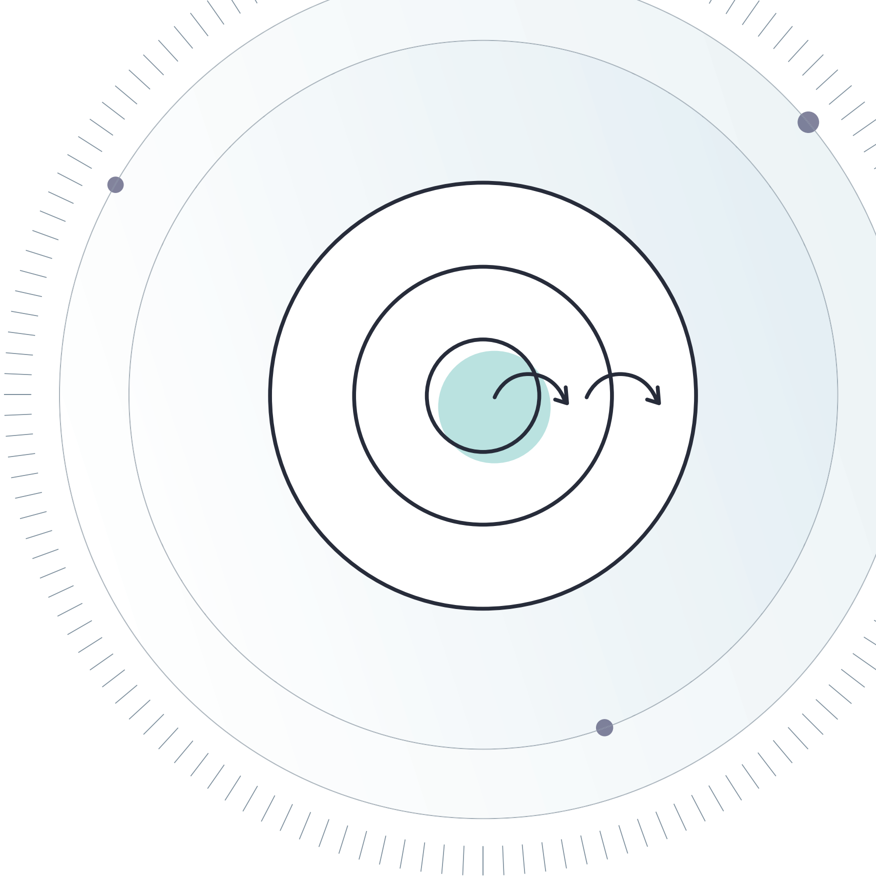
[3] Jez Humble discusses deployment rings and limiting impact in his book *Continuous Delivery*.

techniques are like different strands of the same rope, all aimed at giving teams greater control when delivering software. When woven together, they become a new approach to software delivery, one that iterates on Continuous Delivery and that is essential to modern software development. Enterprises gain the control they desire and thus pursue the core benefits of Continuous Delivery safely, at their own pace, and on their own terms.

We call this new approach Progressive Delivery.[4]

[4] Teams at IBM, Microsoft, and Target have written and talked about how they are making Progressive Delivery (sometimes referred to as "Continuous Delivery ++") work for them. Microsoft has been speaking about this approach for a while and helping teams adopt the right tools to benefit from this approach. James Governor of RedMonk was one of the first to popularize the term "progressive delivery."

Progressive Delivery gives you the control you need to pursue Continuous Delivery safely, at your own pace, and on your own terms.



What is Progressive Delivery?

Progressive Delivery has two defining characteristics.

Release Progression

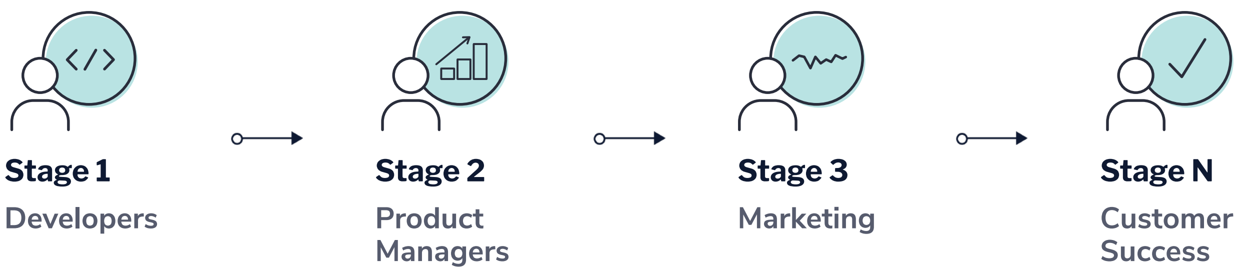
The practice of releasing software/features to end-users at a cadence that is appropriate for the business. This can be done continuously and gradually, or it can even include the long-term goal of only shipping certain features to a fraction of users. This practice builds upon the core tenet of Continuous Delivery that requires the separation of the “deployment of code” from the “release of features.”

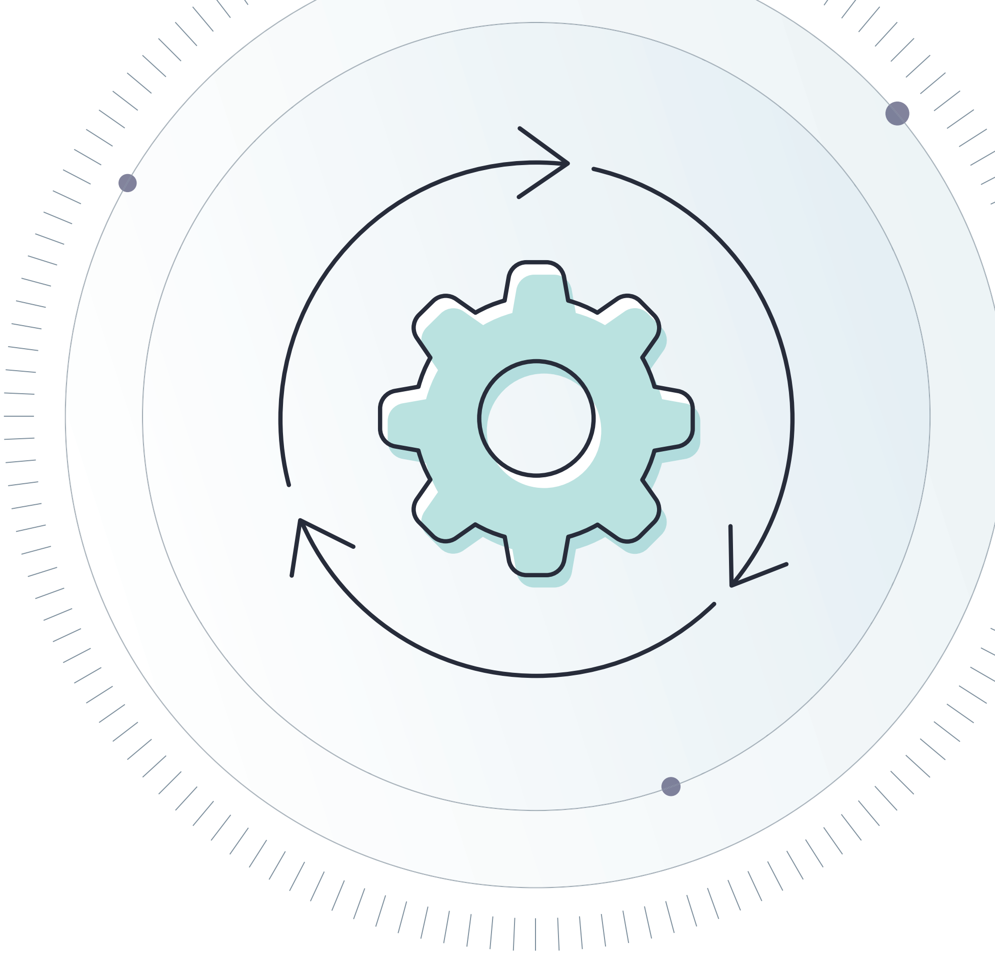
Progressive Delegation

This refers to delegating the control of a feature to the team most responsible for the outcome. This might look like transferring the ownership of a feature from Engineering to Product Management, then from Product Management to Marketing, and so on.

Together, release progressions and progressive delegation reduce the risks associated with Continuous Delivery and empower teams with more control throughout their release cycles.

Stages of feature delegation





How does Progressive Delivery work?

Progressive Delivery gives all teams control over which users see which code changes, and when. It is a transformative cultural shift that enables your entire organization to spend more time creating value and less time managing risk.

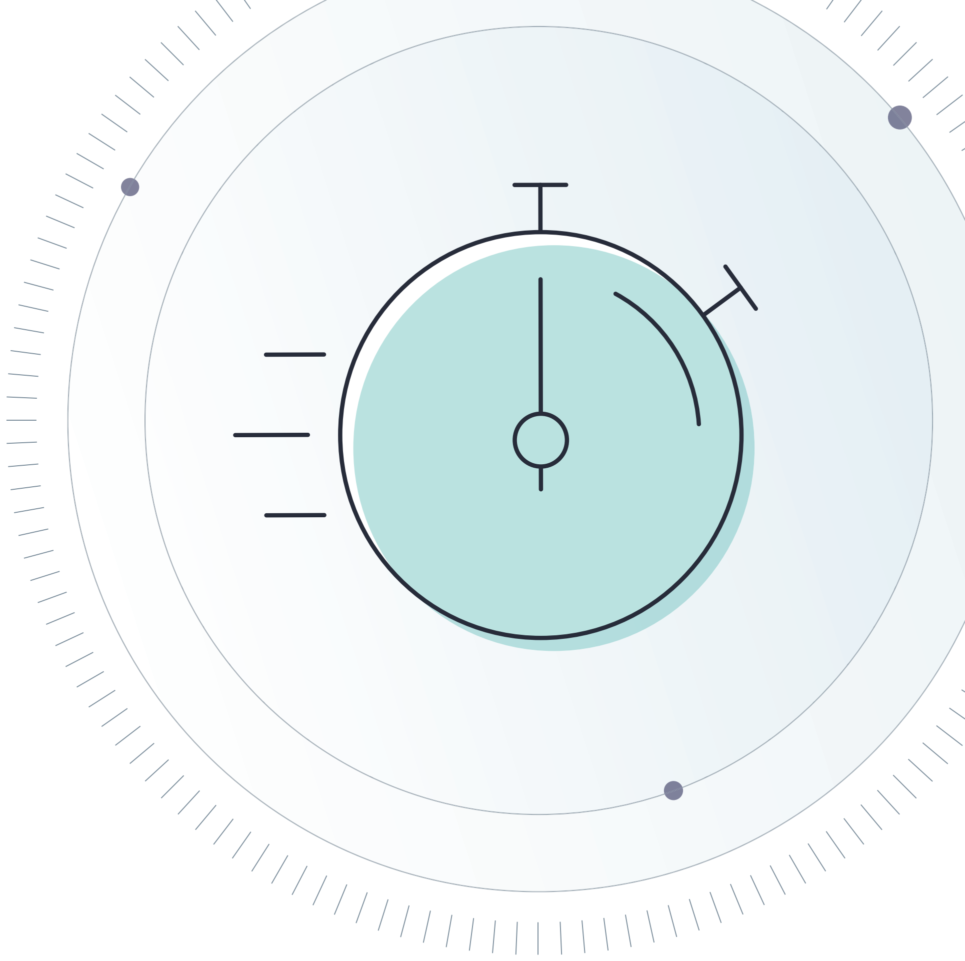
This starts with how teams build. Many teams that engage in Progressive Delivery do trunk-based development. Or, at a minimum, they do continuous integration and unit testing as a part of their code merge process. Teams also use feature management (a topic we'll discuss later) as a best practice to maintain control over features at varying states of readiness. Giving engineers the freedom to continuously integrate code changes, with the power to hide those features selectively, means they can safely deliver faster.

After that is validation. Testing in production, canary launches, beta groups—these are ways teams can provide access to a feature, starting with a small group and then progressively increasing the size of the audience as they build confidence in the feature's stability and functionality.

And then releasing. This means releasing features to user groups based on their tolerance for new features, especially features that may be less stable. It also includes providing non-engineering teams (Product, Marketing, Sales, or others) the ability to release features to users based on business timelines.

Whether running tests with internal and beta users or performing percentage-based rollouts, making changes progressively helps teams deliver software with confidence.

Progressive Delivery gives teams the confidence to ship faster.



Why is Progressive Delivery important?

Progressive Delivery helps teams move faster, in part, by reducing risk. As teams shift to a Continuous Delivery model, they require infrastructure that mitigates the risks of shipping buggy code. Progressive

Delivery acts as this risk-mitigating agent by controlling the audience that is exposed to each new change in your application. This control makes you feel safe when shipping code, in turn, giving you the freedom to deploy faster.

The importance of Progressive Delivery also comes into view when you consider the shift to microservices and global distribution. Increasingly, teams want to isolate changes and control the population impacted by those changes. Moreover, development and operations teams have begun looking to data scientists for insight into how partial rollouts are being received and adopted. Progressive Delivery meets all these various needs of modern software teams.

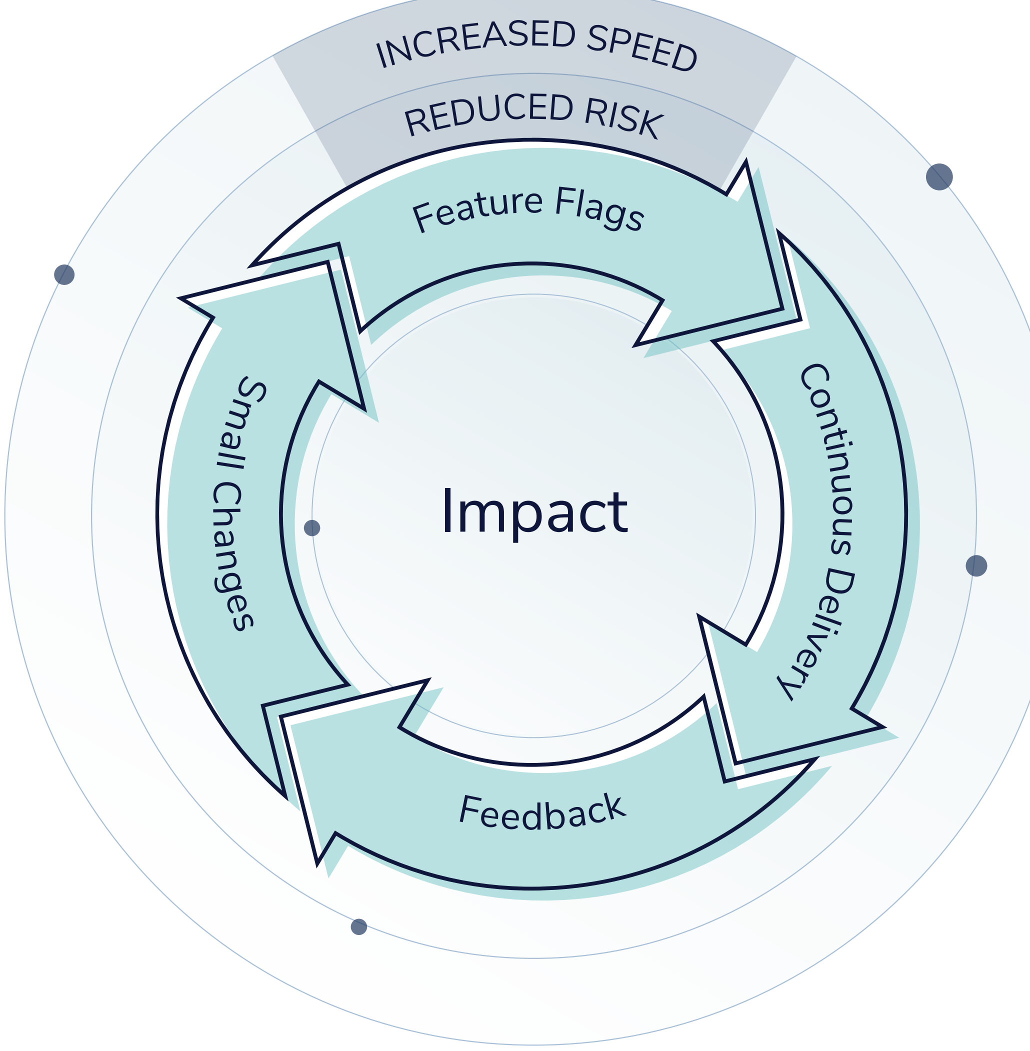
Ultimately, Progressive Delivery enables teams to achieve Continuous Delivery safely and with greater control.

Control your releases

Roll out a new feature to Australia first and see what adoption looks like.

Make a change available to 10% of users, then gradually expand to 100%.

Update a critical service but limit the impact of that change to a small cohort of users.



Progressive Delivery vs. Continuous Delivery

Continuous Integration and Continuous Delivery (CI/CD) emphasizes the need to keep code in a deployable state at all times. This entails continuously integrating the code of new features with the master branch (or trunk) of your application, rather than waiting several weeks to check the code of long-lived feature branches back into master. The goal of CI/CD is to accelerate the development and delivery of software to users, while reducing risk relative to Waterfall.

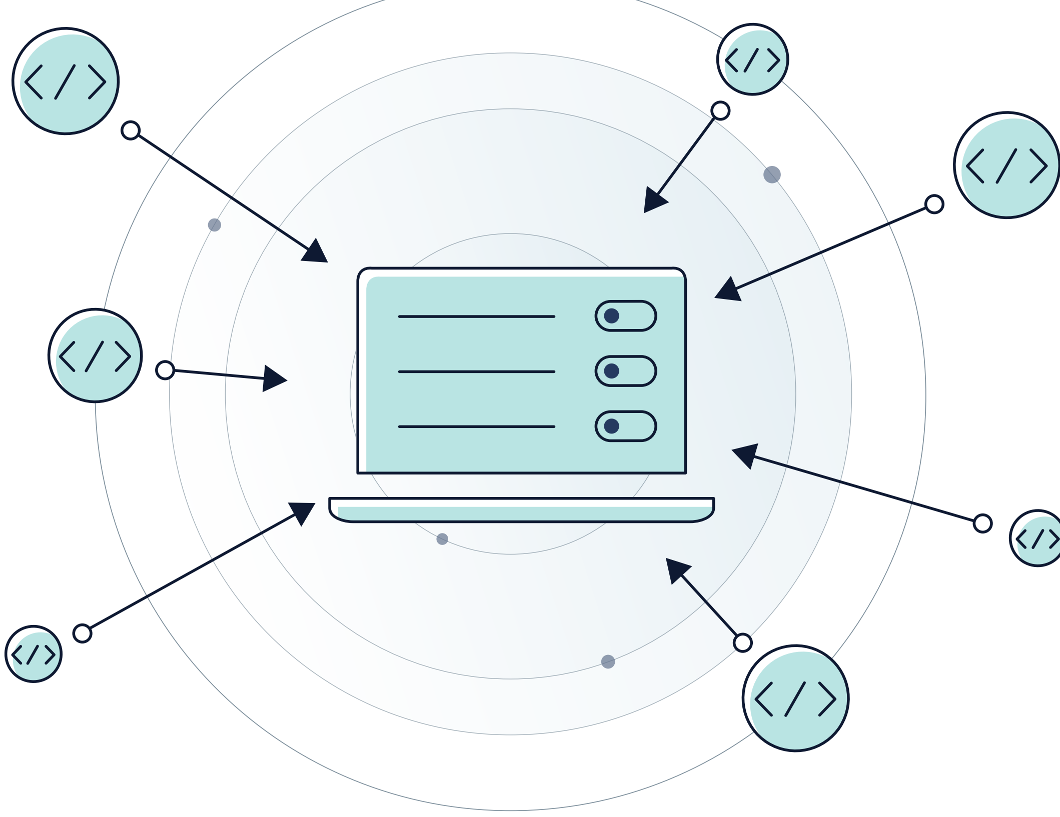
While CI/CD enables teams to move fast, the tools associated with it do not have the inherent safety valves and control points to protect the service and the consumer when failures occur. When teams deploy to production, this can mean releasing changes to all users at the same time. This can be catastrophic in cases where software changes contain major bugs.

And while canary deploys and percentage rollouts are a part of the CI/CD model, greater protections need to be put into place to address the risks that come with moving faster.

Continuous Delivery
Emphasizes the need to keep code in a deployable state at all times. It lacks inherent safety valves and control points needed to protect the customer when failures occur.

Progressive Delivery
Emphasizes control. It is built for failure, and it includes kill switches, safety valves, and control points that enable you to limit the blast radius of failures.

Progressive Delivery incorporates a “built-for-failure” model through the use of feature flags and a feature management platform that consolidates all the control points into a single interface. This is a key distinction of Progressive Delivery. While Continuous Delivery can be done with or without the use of a feature management platform, Progressive Delivery requires this aspect of control.



How feature management enables Progressive Delivery

Many in the industry have chosen feature flags as a tool for implementing Progressive Delivery. When feature flags are combined with robust access controls, collaboration, reliability, and compliance, you get a feature management platform.

At its most basic, a **feature flag** is a control point, or if-then statement, you add to your code. These control points are quite powerful in that they separate deployments from releases. In other words, they let you deploy a new feature to your production environment without releasing it to users.

Among other things, this allows you to safely test new features in production and turn off problematic features without having to redeploy your entire application. Attaching flags to every new feature enables you to deliver faster, iterate more extensively, and virtually eliminate risk when shipping code. At least in theory.

Feature flags come with a catch. They add complexity to your codebase. The more feature flags proliferate, the

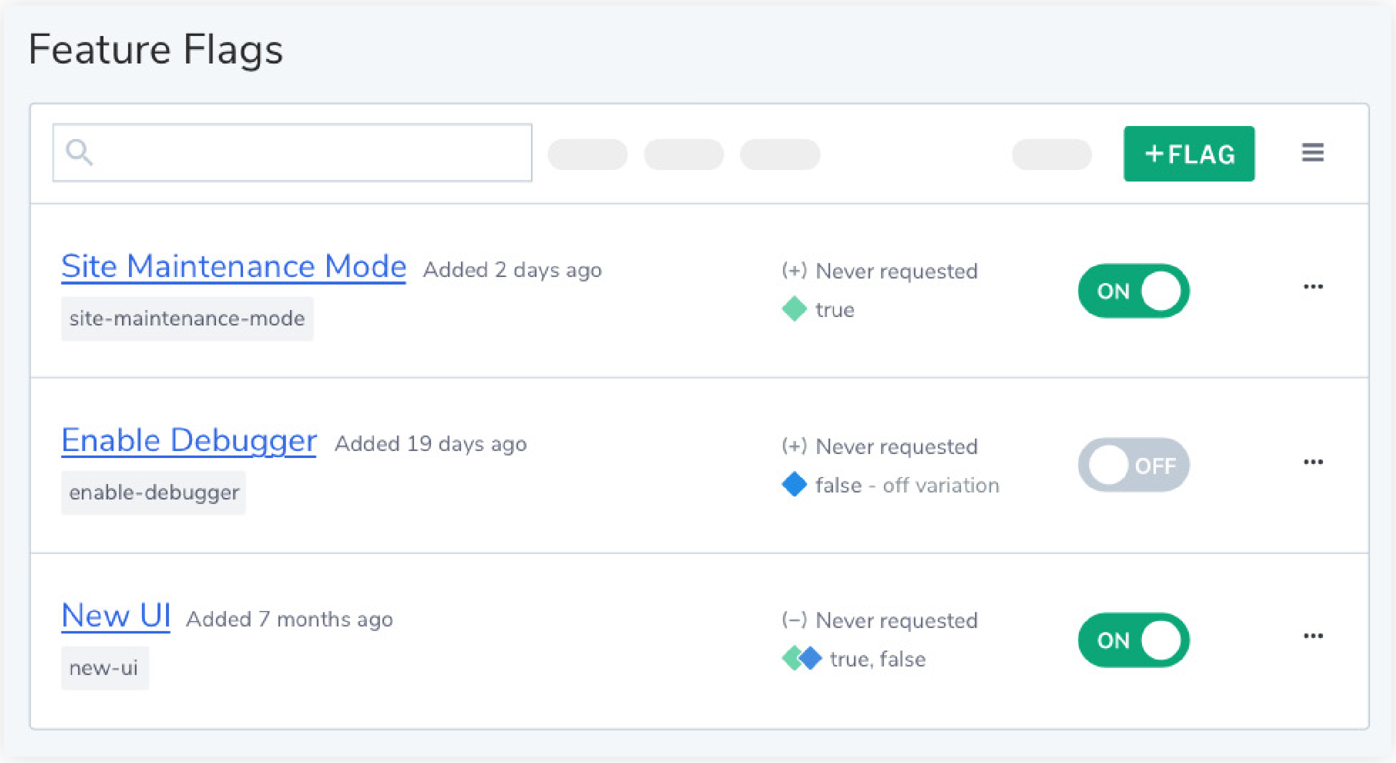
Feature flags are quite powerful in that they separate code deployments from feature releases.

more difficult they become to manage. What's more, when engineers manually create flags, they often can only employ them for simple true-false (Boolean) scenarios.

A **feature management platform** like LaunchDarkly solves both of these problems. It enables anyone to create and manage feature flags on a large scale across a wide range of complex use cases.

Create and manage feature flags at scale

LaunchDarkly's feature management platform enables you to easily manage the entire lifecycle of a feature flag, whether you're using short-term flags (e.g., for releases) or long-term flags (e.g., for operating your application infrastructure). The platform shows you all the flags in your codebase and lets you trace the origins of those flags—who created the flag, what was the purpose behind the flag, etc. This comes in handy when a feature is throwing a bug in production, as you can quickly find the flag in LaunchDarkly and then, well, turn it off.


Feature Flags

+FLAG

Site Maintenance Mode Added 2 days ago site-maintenance-mode	(+) Never requested ◆ true	ON	···
Enable Debugger Added 19 days ago enable-debugger	(+) Never requested ◆ false - off variation	OFF	···
New UI Added 7 months ago new-ui	(–) Never requested ◆◆ true, false	ON	···

Manage several feature flags in a single view
On LaunchDarkly's dashboard, you can view a list of your feature flags and filter them in a variety of ways, take action (e.g., archive a flag or switch it on or off), manage your account, and drill into an individual flag.

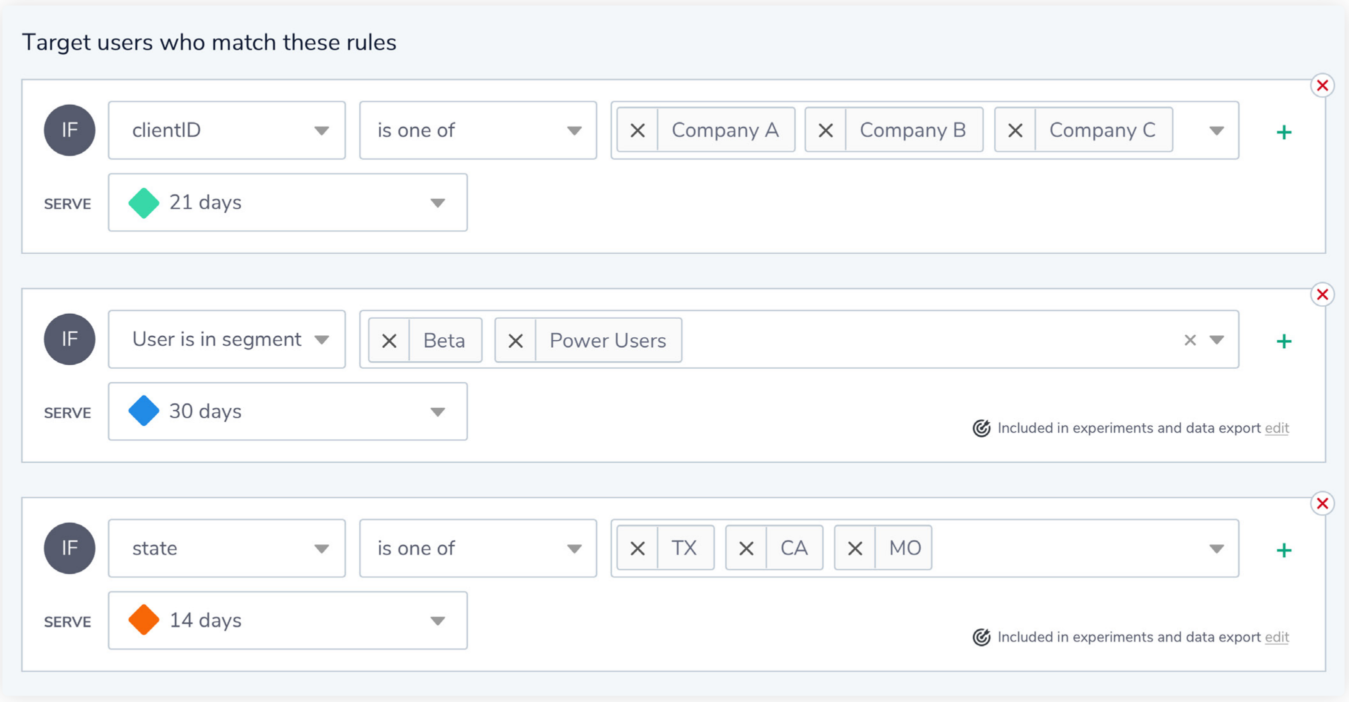
As a platform, LaunchDarkly is designed for everyone. That's why it is equipped with SDKs for all the major programming languages, a growing list of integrations with top developer tools, and API-first features that enable a custom feature flagging experience.

Underpinning the platform is a best-in-class streaming architecture. Among other things, this means that when you make a change to a feature flag, the change registers instantly. And it has no effect on system performance whatsoever.

Feature flags for complex use cases and experimentation

Besides enabling teams to use feature flags at scale, LaunchDarkly also provides a way to do sophisticated user targeting. For instance, teams can create a user cohort based on a set of attributes (e.g., age, country of residence, favorite Lord of the Rings character, etc.) and then release certain features to that group exclusively.

Again, this is the kind of control that defines Progressive Delivery.


Target users who match these rules

IF clientID is one of Company A, Company B, Company C
SERVE 21 days

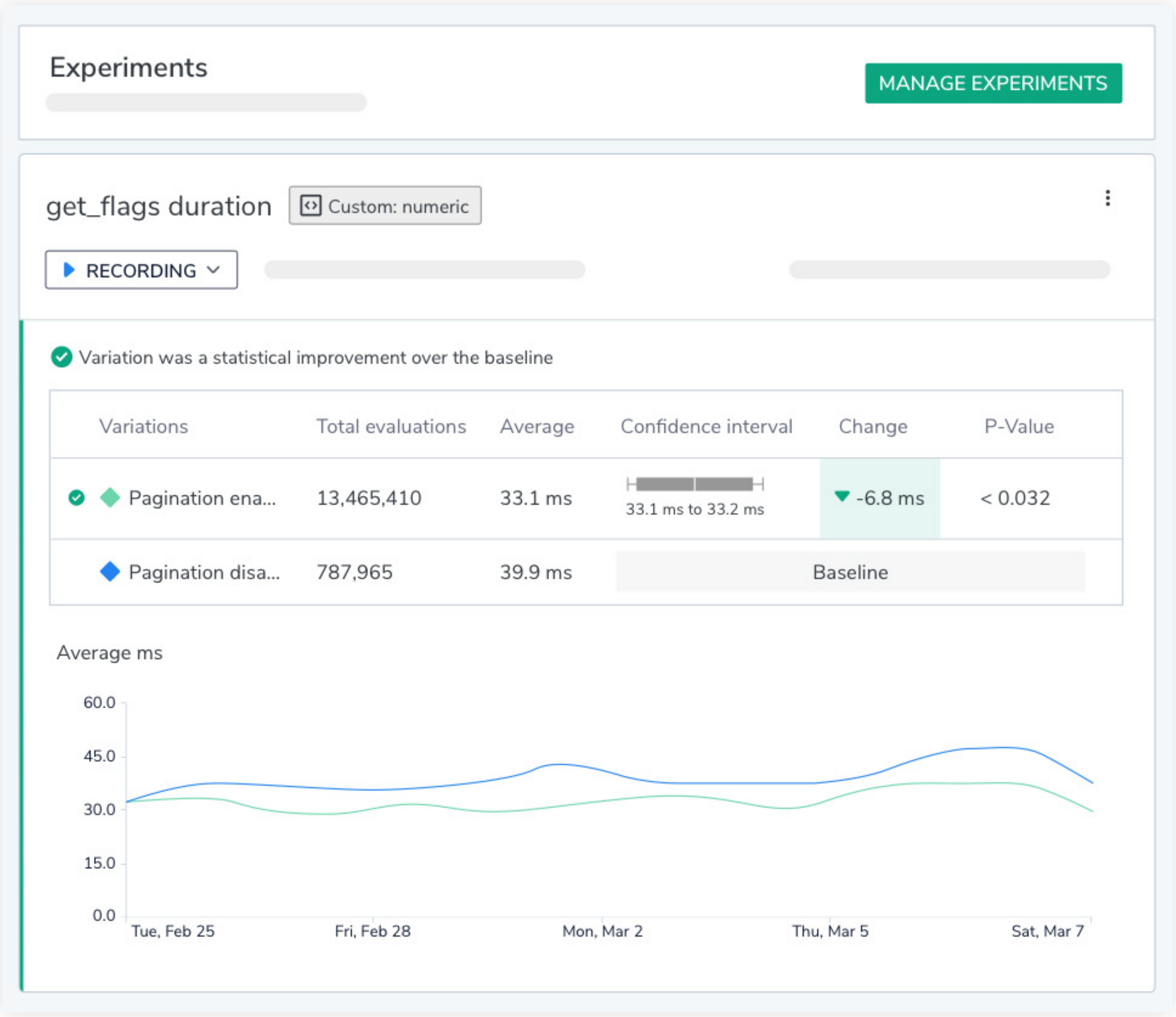
IF User is in segment Beta, Power Users
SERVE 30 days
Included in experiments and data export edit

IF state is one of TX, CA, MO
SERVE 14 days
Included in experiments and data export edit

Detailed user targeting (segmentation) for a targeted rollout
This screen shows a flag that delivers different feature variations to users depending on the company name (ClientID), type of user (User segment), and region (State) associated with the user's account.

LaunchDarkly also enables teams to test multiple variations of a feature. This alone marks a big advantage over traditional binary uses of feature flags. And you can use these multivariate flags for experiments.

LaunchDarkly has experimentation capabilities baked into the platform. You can set baseline metrics, run A/B tests, and conduct in-depth experiments for operational and product engagement purposes—all within the same platform you use for release management.



Experiments

MANAGE EXPERIMENTS

get_flags duration — Custom: numeric

RECORDING

Variation was a statistical improvement over the baseline

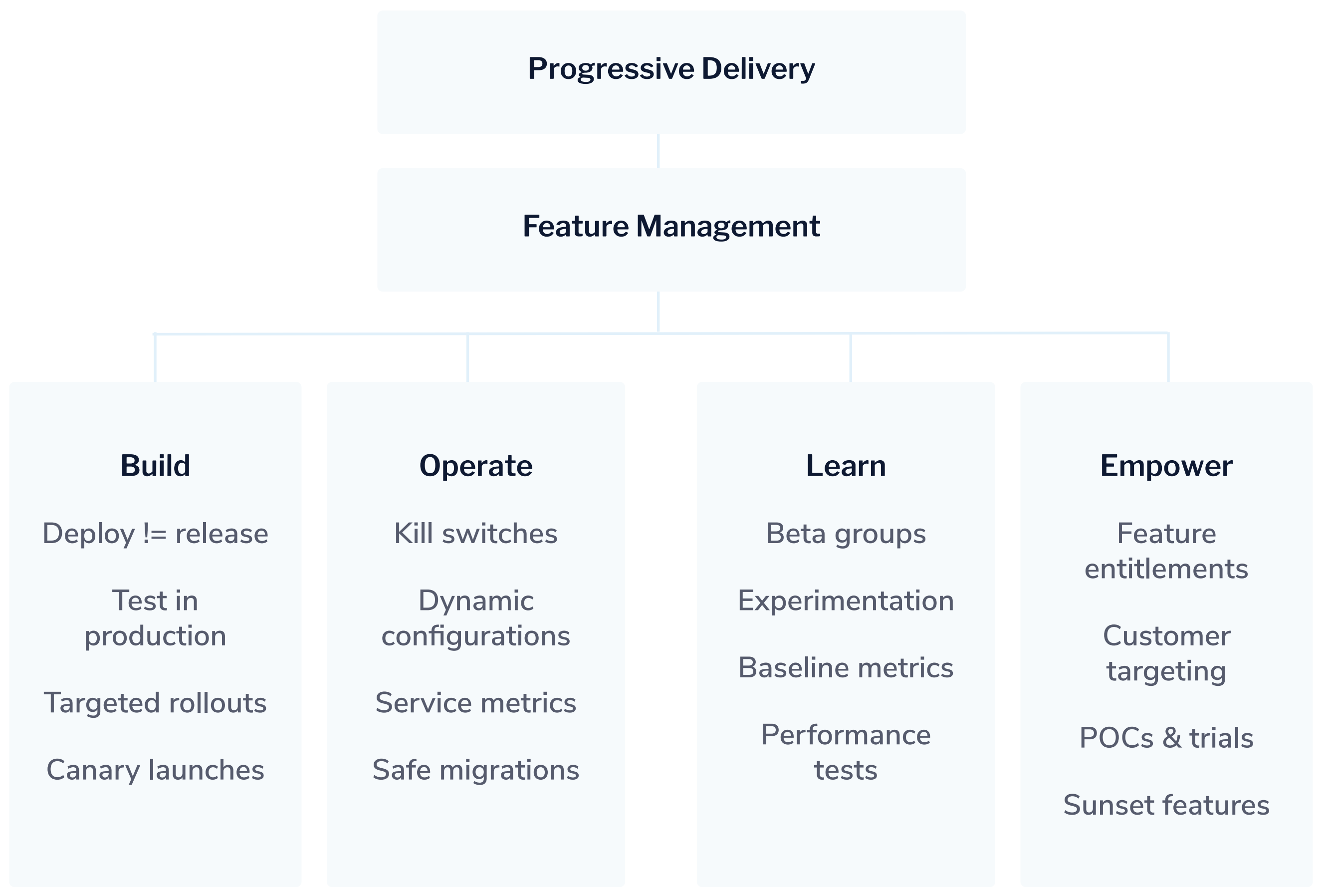
	Variations	Total evaluations	Average	Confidence interval	Change	P-Value
✓	◆ Pagination ena...	13,465,410	33.1 ms	33.1 ms to 33.2 ms	▼ -6.8 ms	< 0.032
	◆ Pagination disa...	787,965	39.9 ms	Baseline		

Run experiments in a feature management platform
Manage an experiment (stop, start, pause, reset, delete), add/remove metrics, and view the performance of the experiment in numbers and a visualization. In this example, we're looking at the performance impact (in ms) of a pagination feature.

Build, Operate, Learn, and Empower

We've briefly explained how a feature management platform enables you to both manage feature flags at scale and employ them for complex use cases. Again, the reason we bring feature management up at all is that it enables Progressive Delivery.

And it does so across four key areas of software development and delivery: Build, Operate, Learn, and Empower.



Build

Build and deliver software faster and with less risk.

Deploy != release. Engineers can deploy code—even incomplete code—to production whenever they want; the marketing team can then release when they are ready.

Test in production. Rather than letting features pile up in QA, you can test code directly in production without exposing it to the wrong users.

Targeted rollouts. Progressively deliver features via targeted rollouts, canary launches, ring deployments, and other controlled techniques while gathering valuable user feedback and engagement data along the way.

Operate

Improve your application’s resilience and reliability.

Kill switches. Instantly shut off features when they are hurting the user experience and your application’s performance (no rollbacks or redeploys).

Dynamic configurations. Change configurations on the fly for things like adjusting logging levels or rate limiting API calls without having to redeploy.

Service metrics. Measure how new features affect your key service metrics.

Safe migrations. Use feature flags to perform safer migrations when swapping databases, switching to microservices, and moving on-premises systems to the cloud.

Learn

Learn about your software and users by experimenting with everything.

Beta groups. Conduct beta tests more seamlessly and gain feedback from real users earlier in the development process.

Experimentation. Run A/B/n experiments not only for front-end features but also for testing large infrastructure changes, new algorithms, and more.

Baseline metrics and performance tests. Set baseline metrics to compare the performance of one feature variant to another while also measuring the impact of certain features on system performance.

Empower

Empower teams outside of Engineering to play a bigger role in delivering software.

Feature entitlements. Reduce the burden on development teams by granting feature control to customer-facing teams (e.g., Product Management).

Customer targeting. Let Product, Marketing, Support, and Sales control exactly which features a customer has access to, and when.

POCs and trials. Allow sales teams to issue and run their own POCs and trials, so as to create a better customer experience and further reduce the burden on engineers.

Sunset features. Reduce technical debt and gracefully remove old features without needing much intervention from the engineering team.

The four areas of development and delivery—Build, Operate, Learn, and Empower—articulate the core value drivers of Progressive Delivery. You can, of course, still do Progressive Delivery without adhering to every facet described in this model. But we do see a consistent pattern of teams adopting the practices in the four areas of Progressive Delivery over time.

If you're progressively releasing features and progressively delegating the control of those features in some fashion, then you are engaging in Progressive Delivery, or at least moving toward it.

To reiterate, a feature management platform supports the core use cases of Progressive Delivery. Many teams, for example, use feature management to perform all types of release progressions.

Developers may create a feature flag to expose certain features only to users in, say, Egypt (i.e., a targeted rollout). This same team could then use the platform to progressively delegate the control of that flag. For instance, the developers may transfer control to a product manager, who can then regulate the rollout in a way that aligns with broader business objectives.

In sum, a feature management platform offers the tools needed to put Progressive Delivery into practice. It empowers software teams to ship faster, stay safer, and continuously improve both their product and the customer experience. And it enables them to take control of their releases, once and for all.

A feature management platform offers the tools needed to put Progressive Delivery into practice.

Start profiting from Progressive Delivery

Those who continuously deliver, continuously improve. In doing so, they eat up market share and leave competitors chasing at their heels. Indeed, Continuous

Delivery is critical to success for modern software development teams. But Continuous Delivery without control is scary.

Progressive Delivery gives teams the control they need to do CI/CD safely and in a way that better supports critical business priorities. Feature management is a key enabler of Progressive Delivery.

Enterprises that leverage a feature management platform reduce risk, vastly improve the quality of their software, and accelerate their release cycles—all in increasing measure. Most importantly, they deliver greater value to their customers.

Ship fast, stay safe, and stay in control. That is the way of Progressive Delivery.

LaunchDarkly

Empowering all teams to deliver and control their software.

launchdarkly.com
sales@launchdarkly.com