LaunchDarkly

9 best practices for release management

Using feature flags for successful software rollouts

Contents

Release management feature flags use cases	02
Deploying release management feature flags	03
Ring deployment	03
Percentage deployment	04
Release management best practices	05
Planning	06
Implementation	07
Deploying	10
1 best practice for leveraging best practices	11

Release management feature flags use cases

When most people hear “feature flag,” they probably think of a flag that controls who can see a feature and when. In fact though, release management is one of the most common use cases for feature flags.

Whether you call it early access, a canary release, or a beta, giving a select group of users access to features prior to release helps you fine-tune a feature before releasing it to everyone. Starting small and rolling out to larger groups over time helps you:

- Observe the behavior of the systems and services under increasing load
- Collect user feedback and make changes if you need to
- Limit the blast radius if something goes wrong

A release management flag is intentionally short-lived. Once 100% of users have access to the feature, remove the flag.

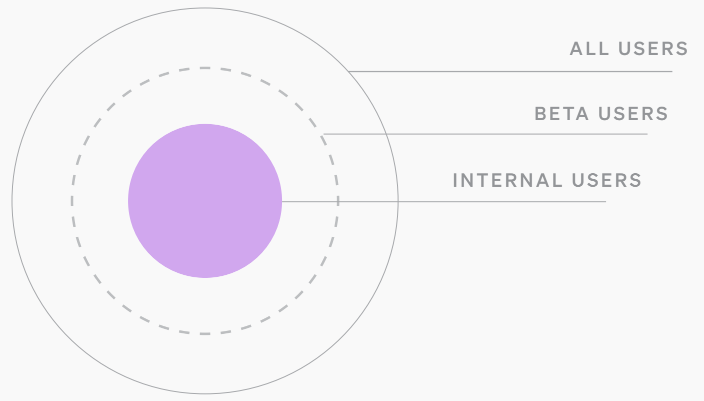
Deploying release management feature flags

The two most common deployment models to use are ring deployments and percentage-based deployments.

1 Ring deployment

Jez Humble introduced the concept of a ring deployment in his book “Continuous Delivery.” In this scenario, different groups gradually receive the feature to manage the risk of deployment. Users for each group are selected based on a set of similar attributes or an opt-in process. Then make features available to the selected groups.

For example, release to internal users first, then beta users, and then to all users:



2 Percentage deployment

The selection of who receives a new feature occurs randomly in a percentage-based deployment. The new feature rolls out to 10% of users, then 25%, then 50%, until all users receive the new feature. These deployments are useful when you cannot run a beta program or have little variation in your targeted user base.

Suppose you run a small, B2B application with the same 1,000 monthly active users. In that case, a percentage-based release is likely to give you an accurate estimate of whether the full rollout will have a positive or negative impact on your user base.

Some companies automate percentage-based rollouts, gradually increasing the number of users receiving the new feature. The rollout is terminated if there are too many errors or if a response time exceeding a set threshold occurs.

You can combine ring and percentage-based deployments. You first would deploy with selected groups and then use a percentage-based deployment to roll out to the remainder of your user base.

Release management best practices

Creating release management flags is a three-part process: planning, implementation, and deployment. Consider the following best practices for each phase in the process.

1 Planning

Make feature flags a priority from the start

Again, the right time to think about whether a feature needs a flag is during the design process. The design of a feature may vary based on whether or not it is behind a flag. When feature flags are an afterthought, you end up with flags that don't work as expected, don't target the right users, or have unexpected behaviors.

Thinking about a feature flag during the design phase requires you to consider who will use the feature. If you determine the feature is for larger enterprises during the design phase, you know that targeting customers at small or mid-sized businesses would not be appropriate.

Understand the purpose of a feature flag

A flag can serve many purposes. Release flags can control how a feature rolls out to users as well as serve as a kill switch if something goes wrong. If a flag will serve multiple purposes, notify all parties and make sure everyone has access to the flag.

Establish naming conventions

As mentioned earlier, a consistent naming strategy ensures everyone understands the purpose of the flag and ultimately helps avoid technical debt.

2 Implementation

You've planned the feature; now you're ready to implement the flag. Questions to ask to determine the deployment path:

- Do features get deployed to staging, pre-production, or test environments before rolling out to production?
- How will you roll out the feature to users?
- Is it first used by developers, then internal staff, then external users?
- Will there be a beta or a canary launch?
- Will release occur through a controlled rollout or to all users at once?

There is not a right or wrong answer when it comes to how you deploy, but to create an appropriate flag, you need to have a deployment plan in place.

Test the behavior of the feature flag

We are big proponents of testing in production. However, this doesn't mean you only test in production before you deploy a flag—you should test the behavior of the application with the flag toggled on and off. This is especially important if the feature is related to a schema or database change. What happens when a feature is on and then toggled off? Is data lost or corrupted?

Another aspect of testing is dogfooding—using your own product or service internally. This lets you test out changes on internal employees first. Put yourself in your customer's shoes. With dogfood environments, you can see firsthand the pains that users might experience with a feature. Using the application as a user would use it is a step towards empathy-driven development.

You can test all you want before deployment, but until customers use the feature in production, you won't know if it is working and what needs to be changed. This is why we regularly endorse testing in production.

Configure a fallback value

What happens with a flag if a user is not able to access the most recent settings? What happens if a user accesses your application in offline mode? Configuring a fallback value can eliminate unexpected experiences or errors with interrupted connections. Every flag should have a fallback value describing the expected behavior of the feature in case failures occur during evaluation. Should the flag be on or off by default?

Keep features small, but not too small

Not every feature needs a flag, and sometimes a feature needs multiple flags. You don’t need to wrap a tiny change in a feature flag unless you’re very concerned about its impact to users. Large features such as a new dashboard might require multiple flags.

Create segments or cohorts

Before you deploy, you need to determine how to segment your users. Identify which attributes determine who sees a feature when.

3 Deploying

You've planned and implemented the flag; all that's left is to cross your fingers and deploy to production. Wait, not so fast! As you deploy, consider the following:

Track metrics

What metrics do you need to track regarding features? Availability, reliability, and response time of an application should not change with the deployment of a new feature. Set up monitoring to track performance before, during, and after you deploy code to ensure performance does not get worse. Tracking metrics such as page load time, errors, uptime, conversions, or user registrations can tell you whether a feature is performing well or causing problems. The metrics you track will vary based on your business and objectives.

Avoid technical debt

Release management flags should be short-term. This means you should remove a flag after it has rolled out to 100% of users, as mentioned above. Technical debt accumulates when you forget to remove a flag from the code. Put processes and reminders in place to remove the flag.

1 best practice for leveraging best practices

Lastly? It’s always a best practice to modify any advice for your team. Recommendations may change over time, and what works well for one company or team may not work well for another. That said, we hope you’ve found these tips helpful and if you’re interested in staying updated on the latest best practice, check out the LaunchDarkly blog.

If you’re interested in taking the plunge with LaunchDarkly, see why consulting firm Forrester said you’ll get a 245% ROI from our platform in their report, “The Total Economic Impact of LaunchDarkly,” or go here to view survey results and learn more about the ROI LaunchDarkly has provided our users.

Build your
best software.

launchdarkly.com

sales@launchdarkly.com

LaunchDarkly