Key performance indicators

A short guide for PoV

Breno Gomes
Senior Solution Engineer

January 2022

Why it is important

Modern software has a large volume of interconnected components. In an ideal world the number and types of failures grow proportionally. Strong adoption of cloud functions and containers increases the complexity.

Distributed systems are updated regularly. Every change can create a new type of failure, instability or more “unknown unknowns”.

Key performance indicators are more than numbers you report during the software development lifecycle.

Metrics provide targets for teams to set goals, milestones to gauge progress, and insights that help people across the organization make informed decisions.

A good KPI, by definition, should be measurable and trackable.

KPI in the context of feature flags and experimentation

Existing and prospective customers normally go through a proof of value (PoV) before adopting or subscribing new products.

It is important to define pilot project to evaluate the feasibility of an implementation before it is widely incorporated in the software development lifecycle. It defines the processes to simplify a “minimum viable product”. A proof of value targets the financial and organisational benefits of such implementation.

Key performance indicators are fundamental to define the success criteria. They are used to compare metrics before, during and after the PoV.

Engineers are often caught off guard when they are asked to share the KPI for a PoV. That is perfectly natural unless they were briefed beforehand. This guide intends to share metrics frequently adopted during product evaluation. It is neither prescriptive nor exhaustive. Each company is unique, having freedom to choose what suits it best.

Use cases

Synchronously toggle features across platforms.

Moving from an old service to new service and using LaunchDarkly to incrementally accept traffic.

Migrating between databases by reading/writing to different datastores without multiple deployments.

Control the rollout of expensive operations during application updates.

Switching between different UI themes for given sets of client users, allowing controlled testing before release to the store.

Safely beta/alpha test features in production releases to gather feedback.

Flag for additional content to test click-through rate.

DevOps

Builds

Commits

Deployment frequency

Approving a feature release

Lead time for changes

Change volume

Change failure rate

Defect escape rate

Mean time to detect

Mean time to identify

Mean time to restore service

Percentage of code covered by automated tests

Application usage and traffic

Application availability

Support tickets

Business

Marketing conversion funnel

Cart abandonment

Cart optimisation

Revenue per customer

Customer retention rate

Profit margin

A/B tests

Unique users

Number of sessions

Page views

Session duration

Geography

Technical

Perceived page load time or mobile interaction

Error rate: frontend, backend and infrastructure

Mobile crash rate

DOM processing

Page rendering

Web browser

Mobile device

API timeout

API error

API latency

Web and Infrastructure

Availability

Throughput

Application response time

Database execution time

Error rate

Memory footprint

CPU utilisation

Network throughput

DORA

Aspect of Software Delivery Performance*	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per day and once per week	Between once per week and once per month	Between once per month and once every six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one day	Between one day and one week	Between one week and one month	Between one month and six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day[a]	Less than one day[a]	Between one week and one month
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0-15%[b,c]	0-15%[b,d]	0-15%[c,d]	46-60%

Source: https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance

Tooling

We can target sets of metrics to each audience, ideally covering a range of topics.

Observability tools play a key role, supporting the initiative and reporting the indicators before and after releases. That can maximise the integrations and differentiate from basic products.

Use case

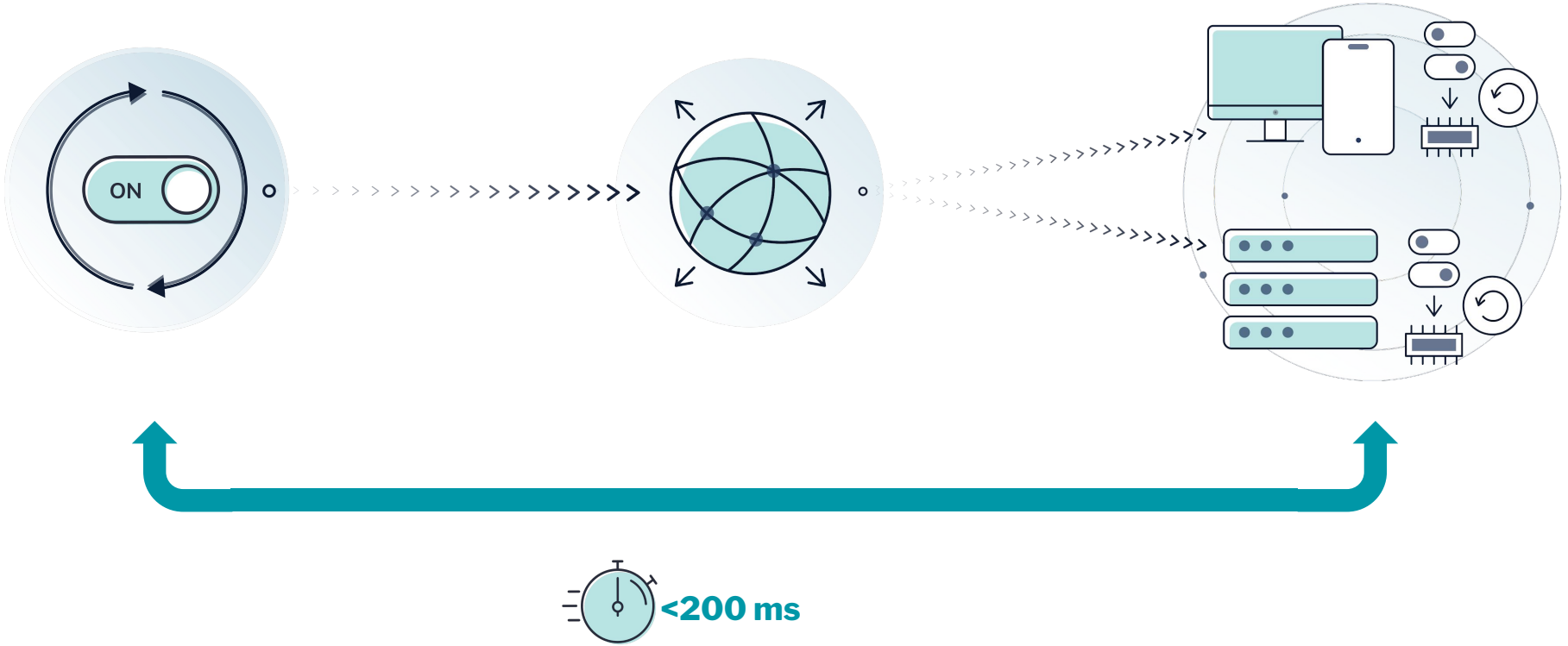
A prescriptive approach can create friction among operations, developers and service reliability engineers. The risk declines when engagements progress to PoV, implementation and enablement.

Several prospects are caught off guard when they need to identify metrics during the PoV workshop. Some have clear indicators in mind. The majority brainstorm on the fly. It is like asking them to fill out a blank page with minimal or no previous notice and guidance. Deployment frequency, lead time to change, mean time to identify and recover are valuable and widely adopted. However there is no differentiation from any competitor.

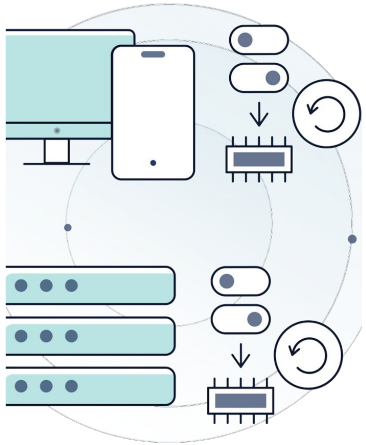
Emphasising use cases and less trivial metrics can generate additional revenue streams. it is important to go beyond the comfort zone, widening the distance from competitors.

Prescribing key performance indicators denotes knowledge and experience. That can beneficial when conversation is well timed during the pre-sales cycle.

How it works



Layers of resiliency



```
ldClient.boolVariation(
"feature", user, false);
```

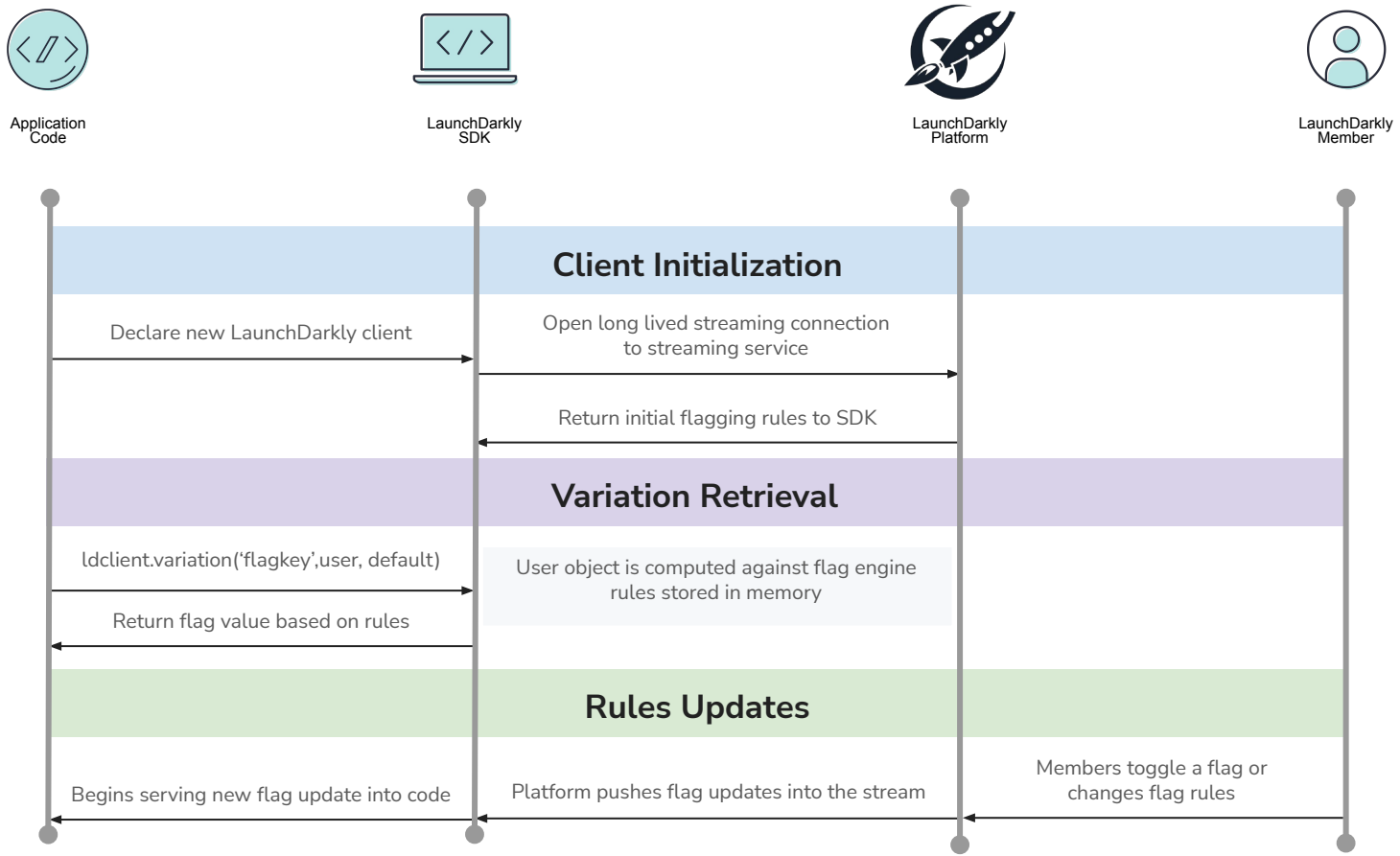
LaunchDarkly.com

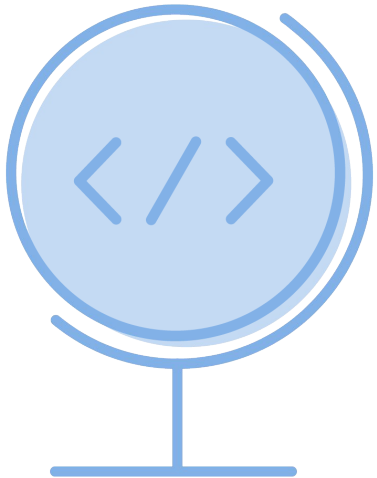
Optional:
LaunchDarkly Relay

Application Memory

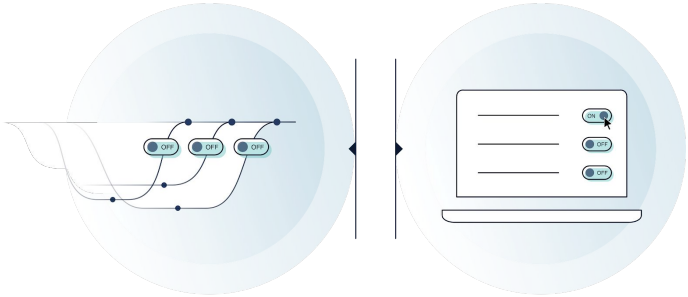
Fallback Values

Streaming API + Performance

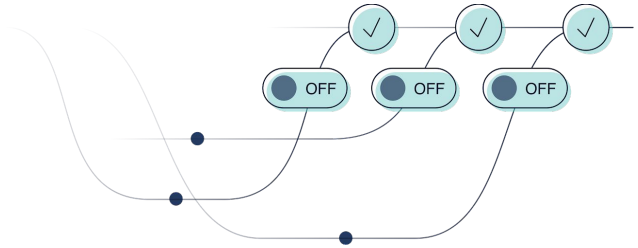




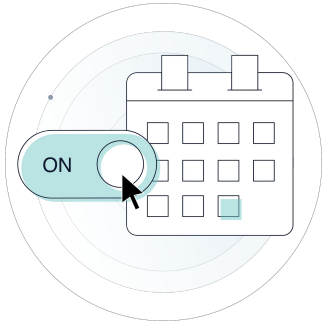
Building



Trunk-based Development



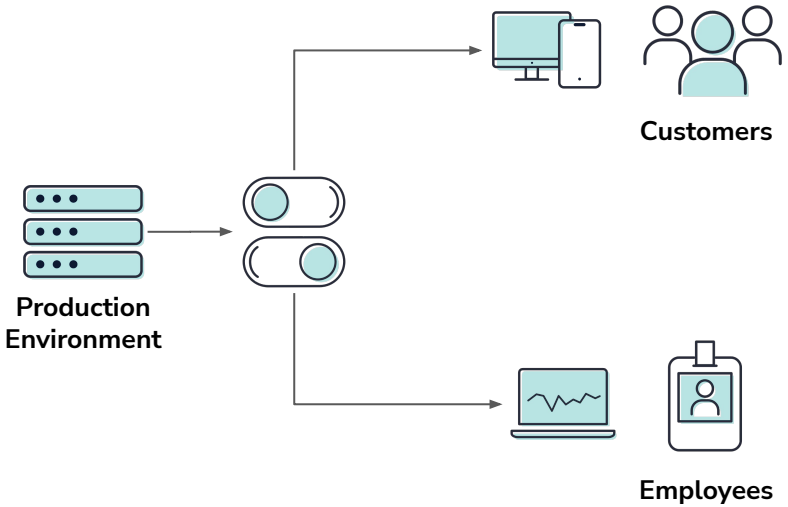
Test in Production

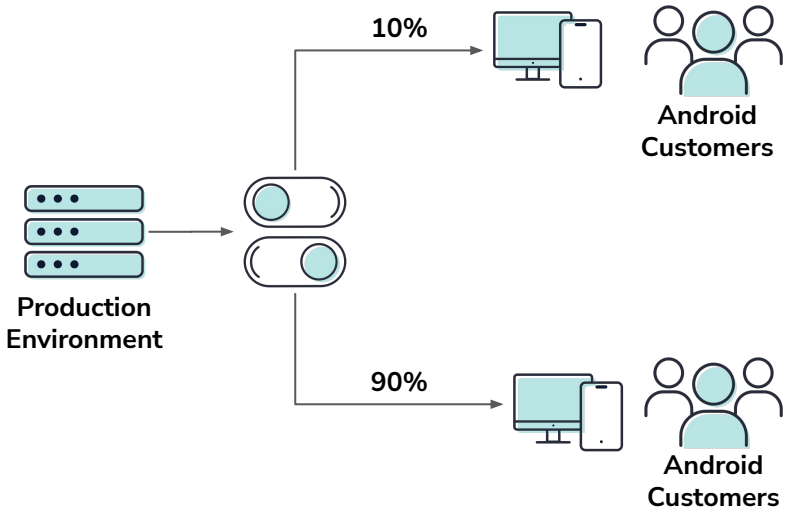


Targeted Rollouts

Canary Launches

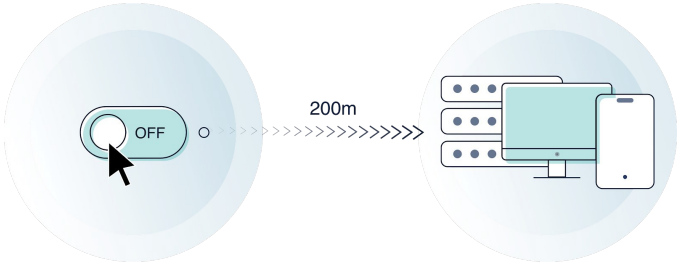
Testing in Production



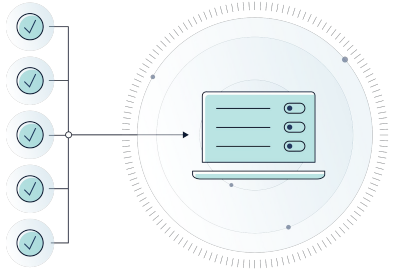


Netflix Testing in production: https://launchdarkly.com/blog/testing-in-production-the-netflix-way/

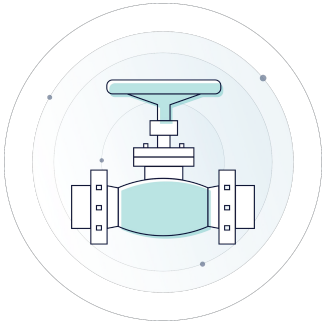
Operational



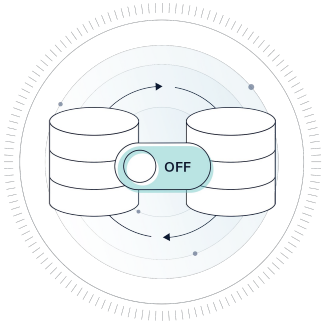
Kill Switch



Service Metrics

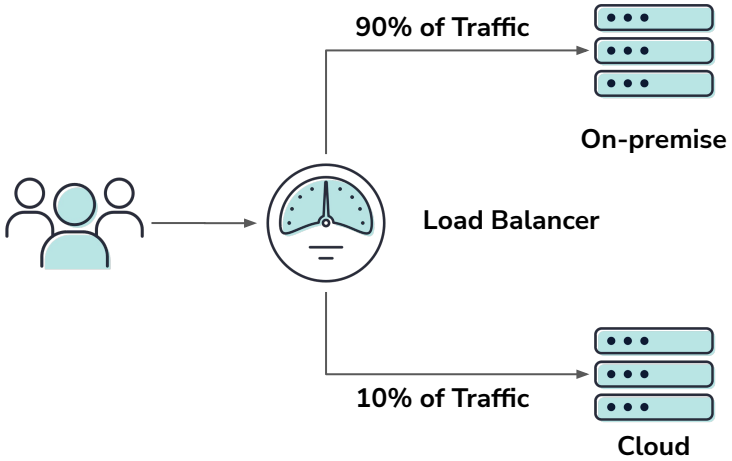


Dynamic Configuration

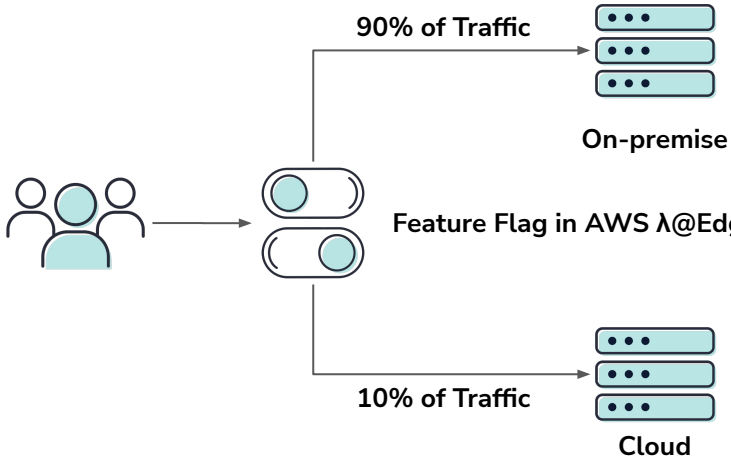


Safe Migrations

TrueCar Cloud Migration



- Simple
- Traffic-based
- Broad

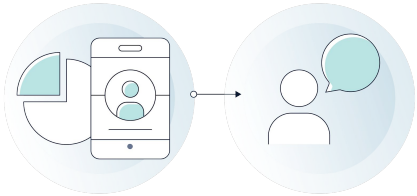


- Complex
- User/site/Attribute based
- Granular (e.g., site searches for new cars vs. used ones)

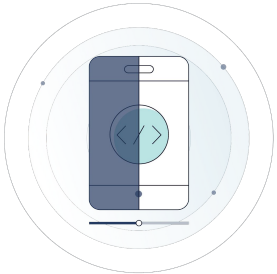
Database Migration Example: https://docs.launchdarkly.com/guides/best-practices/infrastructure-migration

Business Alignment

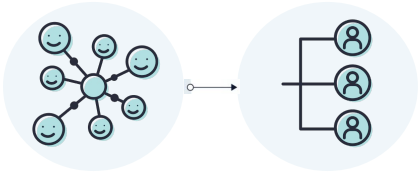
Beta Testing



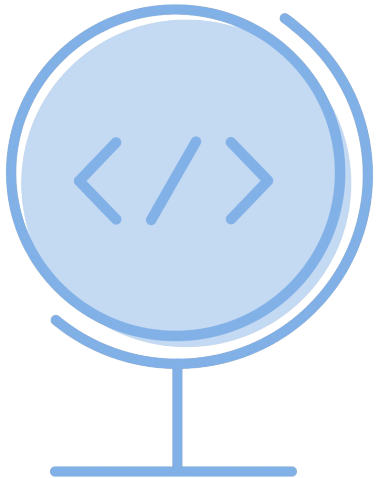
Entitlements



A/B/n testing



Personalisation

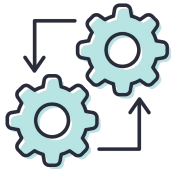


Make flag planning a part of feature design.

- Minimise the focus of your flag.
- Temporary // Permanent?
- Hierarchical Deploys.

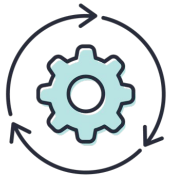
Naming Conventions are your friend!

- Prefix a Team or Project name.
- Behaviour?
- Temporary // Permanent?
- Creation Date.
- Avoid Redundancy: ‘New’, ‘Flag’



Understand the deployment path.

- How will you rollout?
  - Ring or Percentage.
  - Canary or Beta.
  - Progressive or Binary.
- Where will Testing happen?
- What are my Fallback Values?



Cleaning up.

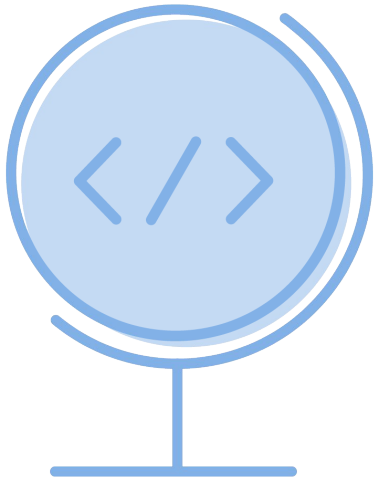
- Regular Review Cycles.
  - Status
  - Age
  - Variation volume
  - Evaluations
- RBAC: Promote & Delete.
- Alignment: Flags / Repos.
- Wrappers & Abstraction.


//Typical conditional
if flag.variation("recommendation-algo", false, user) {
// do stuff
} else {
// do other stuff
}

```
// caller
get_recommendation(user);

// somewhere else
function get_recommendation(user) {
    if flag.variation("recommendation-algo", false, user) {
    new_method(user);
} else {
   old_method(user)
}
```


//remove the flag
function get_recommendation(user) {
new_method(user);
}



Keep in Mind

LaunchDarkly is a developer-focused tool, it is important to keep your architecture in mind when trying to establish the best model for your organization.

It's important to remember that there are no right or wrong answers to this, merely that you should try to consider the opportunity cost for the model that you apply.

We'll cover three models below:

- **Single project.**
- **One project per product line.**
- **Shared services.**

Single Project.

If you find yourself with a smaller team, or are only rolling out LaunchDarkly for a specific project, there isn't a need to try to break it up to more projects. This is also great if you have a monolith and are using feature flags throughout the entire stack.

Pros.

- Easy to implement.
- One SDK to rule them all.

Cons.

- Can quickly become complex with multiple teams making thousands of flags in your environment.
- Challenging permission model as you start to manage who has access to what.
- Large amount of mental overhead as you try to grapple with the complexities of finding flags.

One project per ‘product line’.

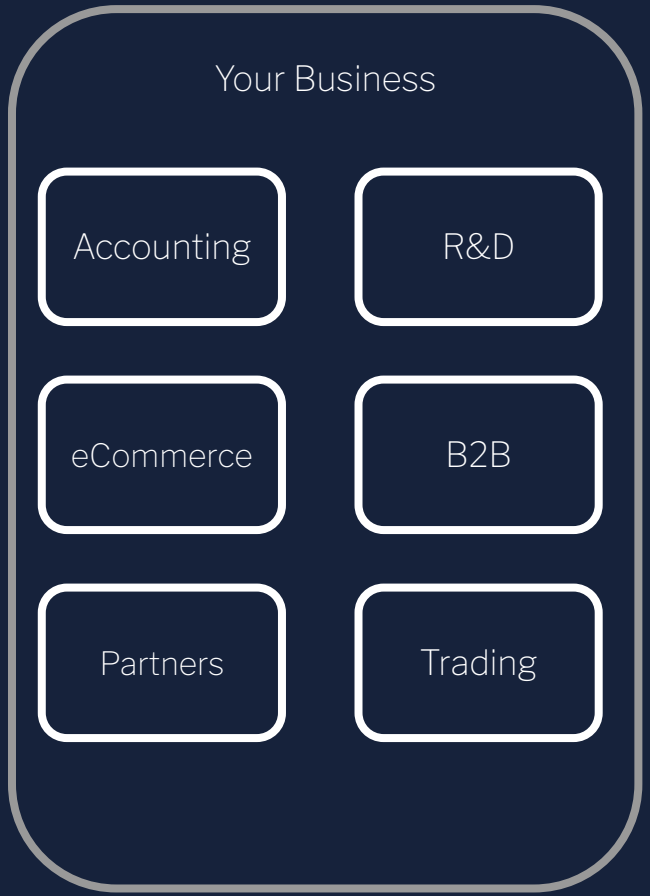
Product line is in quotes because you can abstract these however you want—from lines of business to actual project lines. One project per product line is best when you are rolling out LaunchDarkly to the whole organization, or you have microservices that are functionally distinct from one another or share very few dependencies.

Pros.

- Easy Permissioning as you can scale to teams and just one line of business.
- Less likely to get out of hand due to the limited scope of those teams.

Cons.

- You can’t have a flag in one project apply to a different project.
- You may also need to manage multiple SDK instances within the same project, and this adds complexity.



Shared Services (single project).

This is the most popular model, but also the most complex.

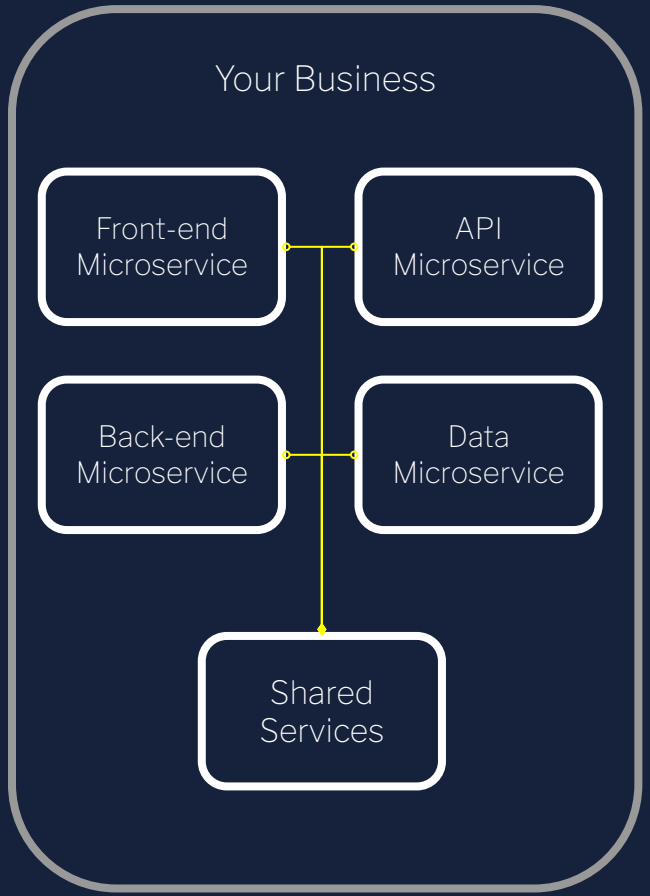
There are situations where different microservices depend on common services. In these situations, you should try a mixed implementation model.

Pros.

- Lets you manage the flags in a centralized model

Cons.

- You can't share flags across the various separate microservices



Shared Services (Multiple projects).

When using multiple projects in the shared services model, you have one project that refers to the product line, and a second project that supports the shared service.

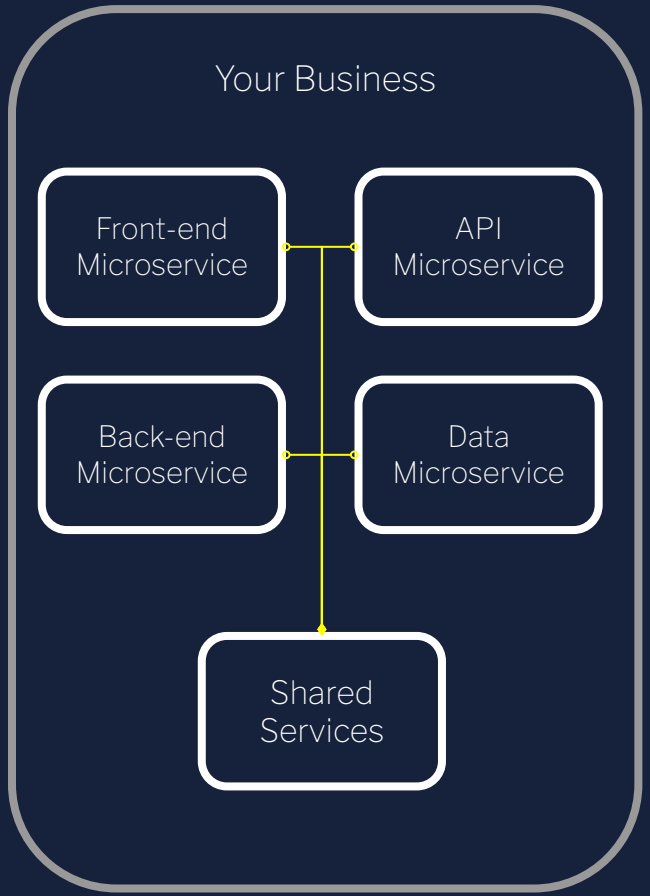
You would then want to initialize multiple instances of the SDK and write a wrapper around them to abstract the complexity away from your developers.

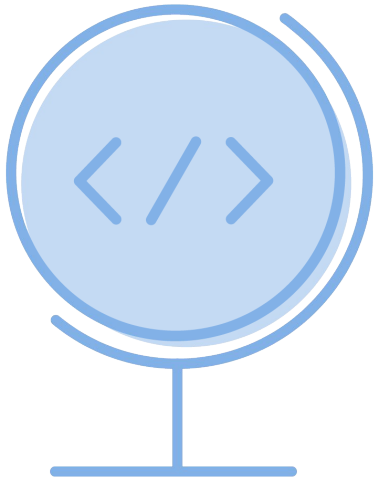
Pros.

- Let you abstract the complexity away from developers while still having features flags for all your services.

Cons.

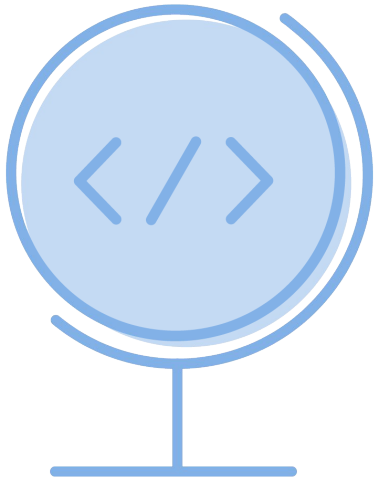
- It can be complex to integrate.





Things to Consider

1. **What will the future look like?**
    a. How many flags will you have long-term?
    b. How can you make this scalable?
2. **How do you disseminate information to your teams?**
    a. Confluence, town halls, google drive, trainings...
3. **At what level will the accountability be?**
    a. Per person, squad, whole team, etc.
4. **What tools are involved in your release process now?**
    a. Slack/MS Teams, Jira/Azure DevOps, ServiceNow, AWS Kinesis/Azure Events Hub
5. **What's a realistic cleanup process?**
    a. Every month, after every flag is fully released, etc.



Whitepapers Etc.

Effective Feature Management

By John Kodumal, LaunchDarkly Co-Founder

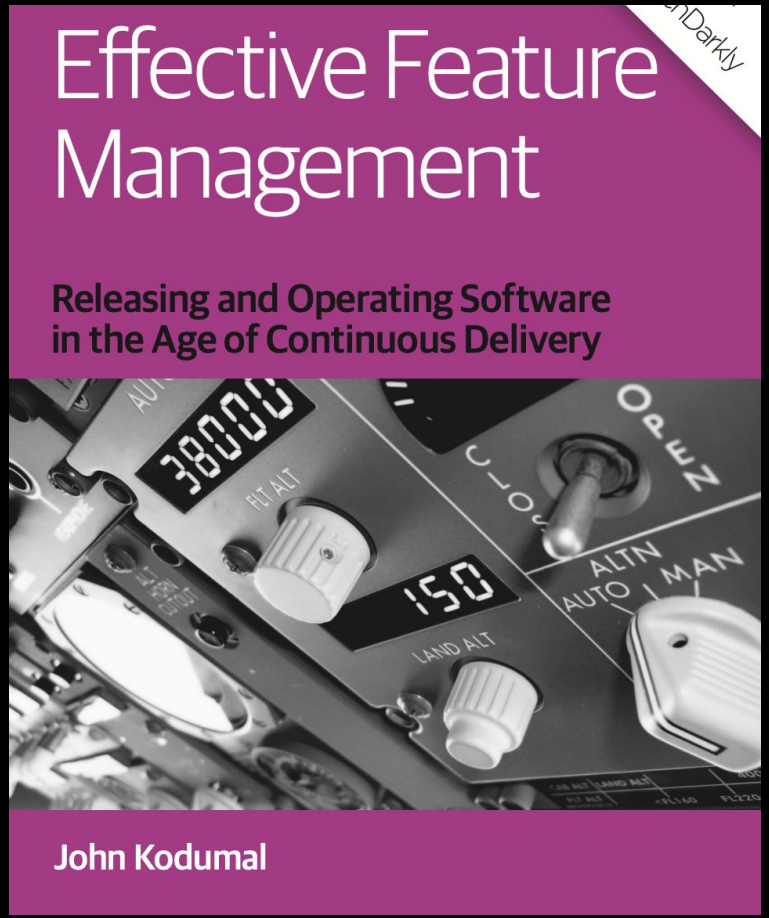
2019 DORA Report

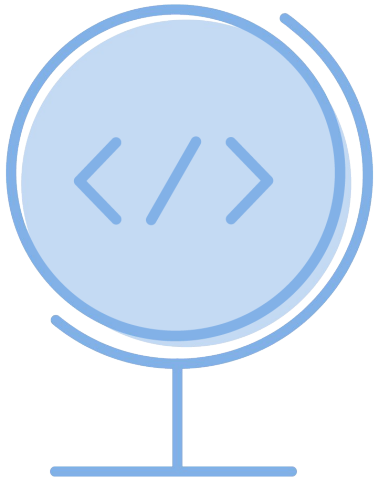
Provided by Google

DevOps Enterprise Journal 2020

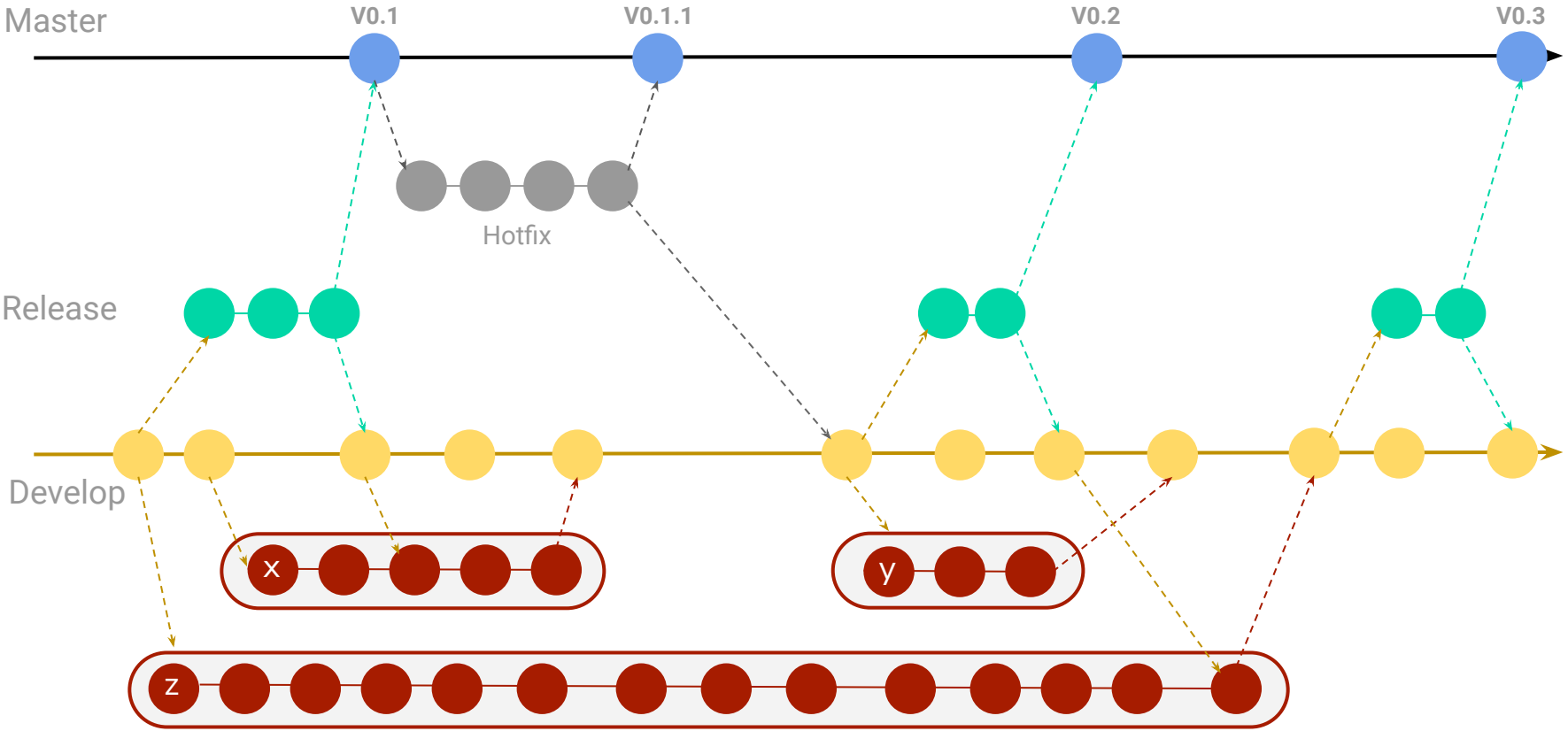
By Gene Kim and Team

Martin Fowler Blog Post

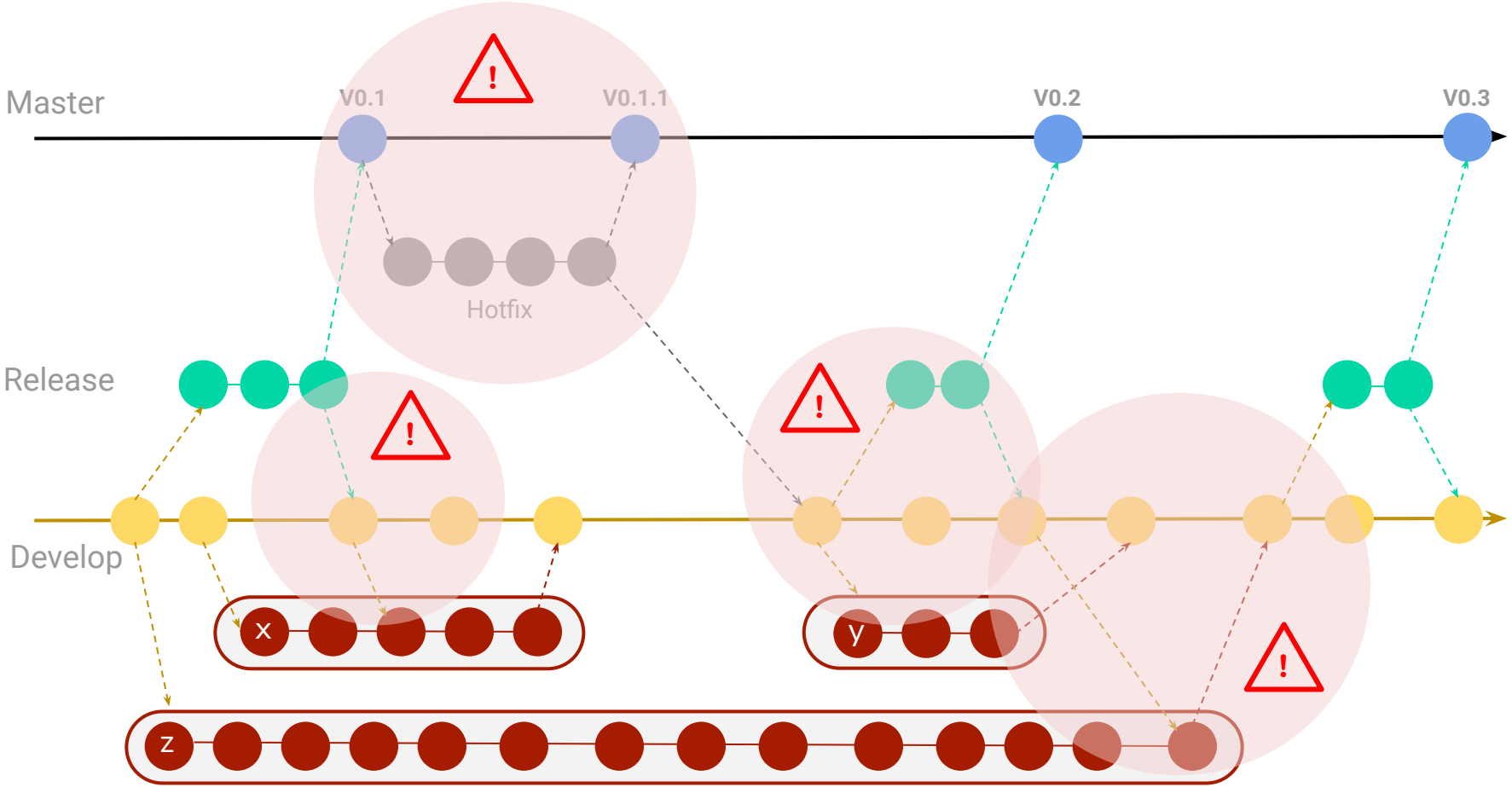




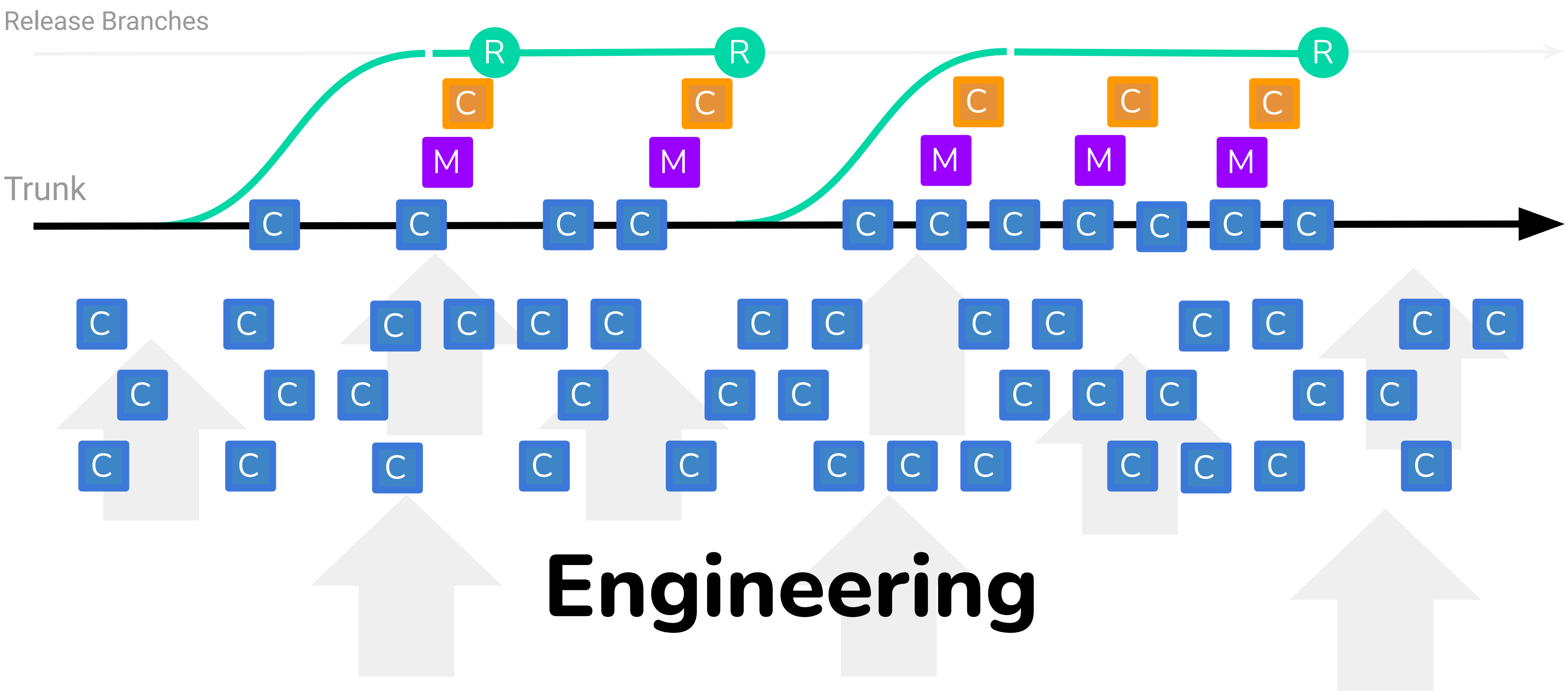
Feature Branch development. (GitFlow)



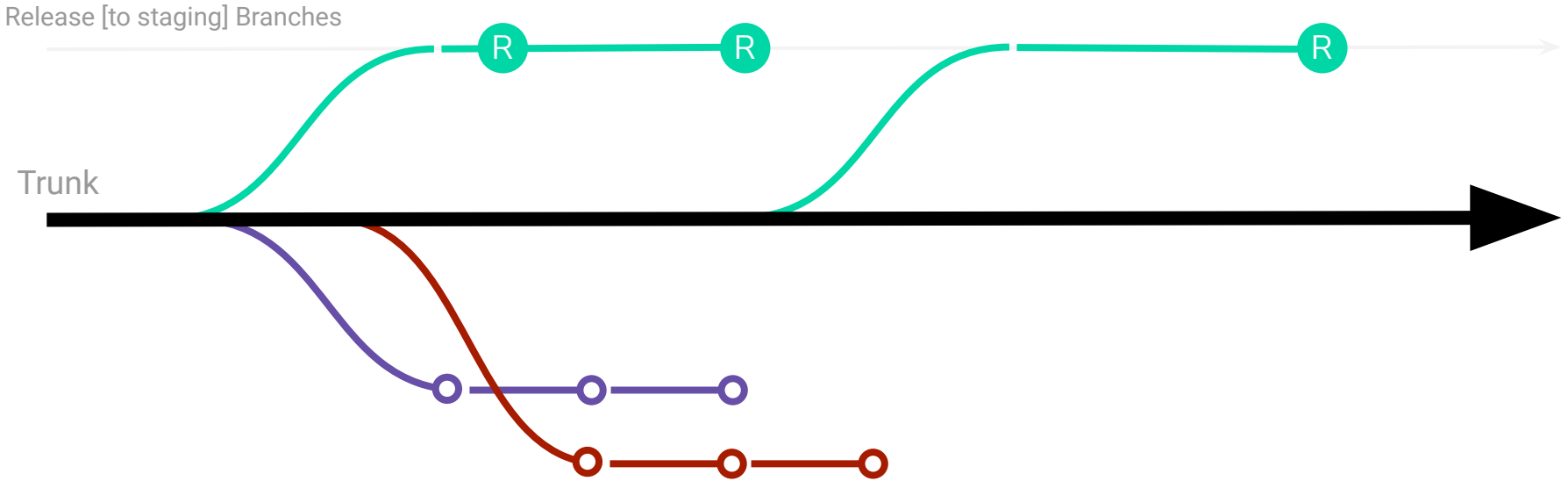
Feature Branch development. (GitFlow)



Trunk-based development.



Trunk-based development.



Key performance indicators

Breno Gomes
Senior Solution Engineer

January 2022