WHITE PAPER

# INTRODUCTION TO FEATURE FLAGGING USING LAUNCHDARKLY


## Abstract

Applications with features continuously evolve. Modern SaaS based applications are multi-tenant and always have a frequent release cycle. But a big bang release of all features in production for all customers simultaneously is not the preferred approach. Identifying the market pulse and early feedback are important for any product. To support these scenarios, DevOps provides capabilities like the canary releases model. Feature flags also support this model by providing an alternative way of enabling or disabling a feature based on certain conditions. But sometimes switching on/off a feature may not be that simple, and the product team might want to enable/disable a feature based on several parameters and contextual information. Product teams usually build custom solutions to support these capabilities. But leveraging a centralized system that can help a rule-based evaluation model for feature flagging in different environments must be considered. This article discusses one of the leading services in this aspect - LaunchDarkly.


Infosys®
Navigate your next

## Capabilities

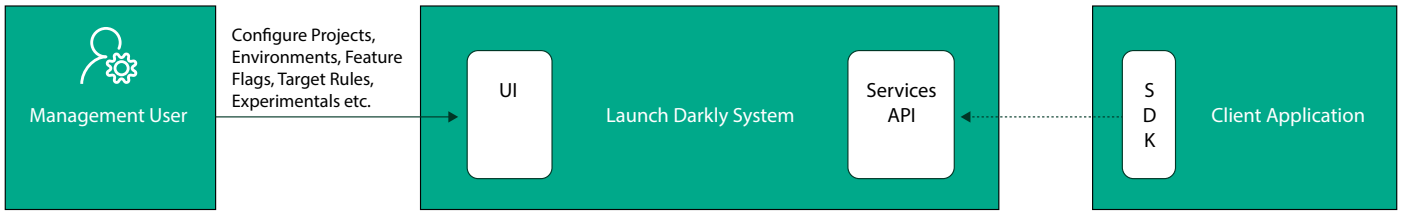
LaunchDarkly, a cloud based service, is highly scalable. It can manage the feature flags in any application that can access the service. Key concepts for using LaunchDarkly are Projects, Environments, Feature Flags, Experiments, SDKs and Rules.

LaunchDarkly allows the creation of feature flags that the application can evaluate during run time. This allows the application to enable or disable features. Features are assessed based on the context information provided by the application.

This helps to create rules based on the context attributes to evaluate a feature flag.

In addition to providing a true/false value as the result of a feature flag evaluation, LaunchDarkly also supports the evaluation result in other types such as int, float, double, string and JSON format. This can be used to get context-based configuration parameters for applications during run time.

**Figure 1 Conceptual model of LaunchDarkly**




In Figure 1, the management user accesses the LaunchDarkly System and configures the projects, environments, feature flags and rules. The client application uses the SDK to evaluate a feature flag for a context.

## Concepts

### Projects

Everything in LaunchDarkly is done through a project. The administrator for the LaunchDarkly account creates the project and adds management users. Users can create different environments in the project, such as Dev, QA, Staging and Production for an application. Projects can be configured to have some default values for flags created within the project and the kind of SDKs to evaluate the flags within the project.

### Environments

LaunchDarkly allows the creation of different environments within a project. All environments within a project have the same set of feature flags. When a new feature flag is created, it is created in every environment in the LaunchDarkly project. That flag is scoped to the entire project. Flags can have different rules for evaluation in different environments.

### Flags

Flags are the core of LaunchDarkly. Different flags can be created in each environment and enable or disable the targeting of the flag. With it, one can assign the evaluation result based on the flag targeting. i.e., the evaluation result of the flag when it is on and off. Usually, it is used to provide a Boolean value, i.e., provide true if the flag is enabled and false when the flag is disabled. But LaunchDarkly offers many options to consider for flag evaluation and the evaluation result. For example, a flag can be configured to provide an integer, string or JSON value as the evaluation result. So, one value can be set when the flag is enabled and another when it is disabled. Also, one can add a conditional evaluation result based on rules - say, when the flag is enabled/disabled, check additional conditions based on the parameters available as part of the evaluation context (like users, locations, or names, email addresses, or any custom parameters) and provide the configured evaluation value as the result.

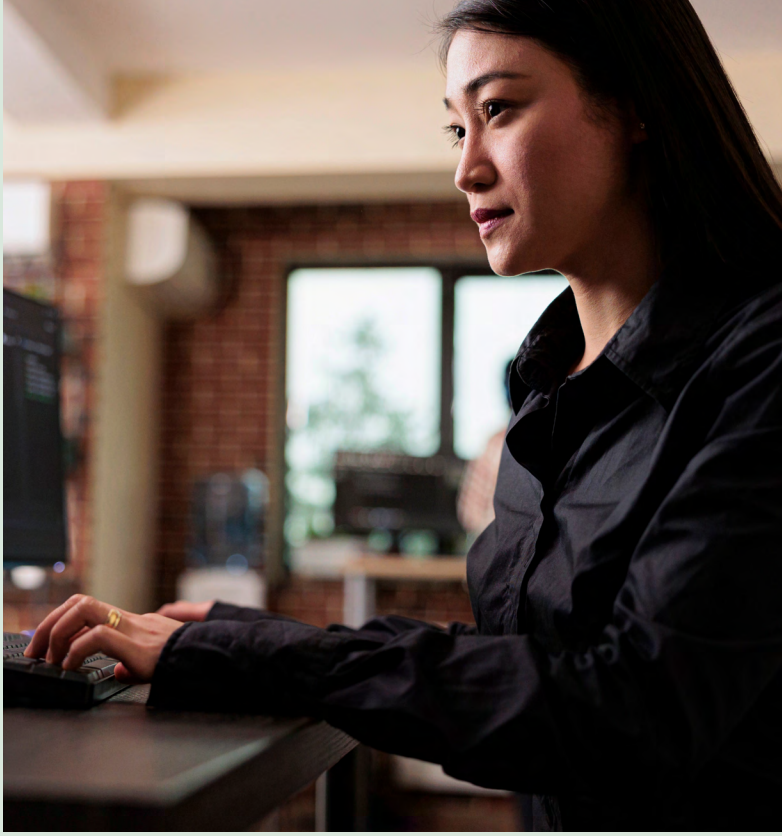
### Segments

LaunchDarkly allows the creation of multiple segments so users can be added to different segments. This enables other flag evaluations to be targeted to different segments of users, which can be used for A/B testing kind of scenarios. LaunchDarkly also supports segments created from existing rules, allowing users who satisfy a rule to be moved to that segment. The specific value can then be returned as the flag value for those users in the segment.

## Experiments

Experiments allow testing the acceptability of features to:

- Validate new ideas by testing multiple variations of a feature.
- Determine the user base's interest in a feature before development.



- Gather performance data for a feature, service or API.
- Increase product adoption by determining the features users prefer.
- Drive revenue and conversion rate by rolling out successful variations to the rest of the user base.

Through experiments, metrics on features can be collected so that analysis of the acceptance of the features becomes possible.

The chief supported metrics types are:

- Click conversion metrics: Tracks user interface (UI) element clicks. For example, the frequency of the user clicks on the save button. It is only compatible with JavaScript and React SDKs.
- Custom conversion metrics: Tracks events for arbitrary events, such as whether a user search called a service.
- Custom numeric metrics: Tracks increases or decreases in numeric value against a set baseline - for example, the number of items in a user's cart when they check out your online store.
- Page view conversion metrics: Tracks how many times a page is viewed. For example, the number of views of blog posts based on three different titles. This is only compatible with JavaScript and React SDKs.

The usual types of experiments are:

- Validating a feature and its acceptance
- Mitigating risk by getting early results of new features, like performance and new issues introduction
- Optimization based on feedback
- Can be used for Chaos engineering by introducing failures based on some feature rules and studying the system behavior.

## SDK

LaunchDarkly provides different kinds of SDKs for client applications based on the application type. SDKs capitalize on LaunchDarkly's streaming APIs under the hood. This allows the applications to have the latest feature flag information available, thus enabling work in offline scenarios.

The key SDKs available are:

## Server SDK

Server-side SDKs are usually used with server-side languages. SDKs are available for server-side frameworks like .NET, Java, Node and Python Got. These SDKs can be installed through the respective package manager tools supported by frameworks like Nuget and Maven. The primary object used in the server-side SDK is an LDClient object which is initialized using an SDK key taken from the admin portal. Each environment in a project will have a separate SDK key. When initialized, the server-side SDK establishes a connection with the LaunchDarkly endpoint, which is used to send requests, get a response and send events. The client, a heavy object, keeps an internal state, and having a single instance per application is advisable. The client object allows us to get the feature key variation by passing the feature key and a context. The client also supports providing a default value for a feature flag which is returned if an error occurs while evaluating the feature flag.

## Client SDK

Client-side SDKs are provided for major client-side frameworks, which are native and web, like iOS, Android, Flutter, Xamarin, React and Angular Vue. By default, flags are only available to server-side SDKs. When a flag is created, we can choose to expose the flag to SDKs that use client-side IDs, SDKs that use mobile keys, or both. If we use a client-side or mobile SDK, one must expose the feature flags for the client-side or mobile SDKs to evaluate them.

## Targeting – Prerequisites, Rules

Targeting allows configuring what variations must be delivered based on conditions when the feature flag is switched on. Targeting consists of Prerequisites (set of other feature flags and its values which will be required to enable the feature) and Rules which can be used to provide a conditional evaluation of feature flags. Parameters of the rules can be context-specific information passed as part of feature evaluation. Rules allow configuring a percentage rollout, where a percentage of users can have one variation and another percentage has a different variation of the feature flags. This helps to roll out new features in an incremental fashion to users.

**Figure 2 shows a sample UI to configure a project - here, instead of true/false variations, different variations are provided and serve a variation with value error.**

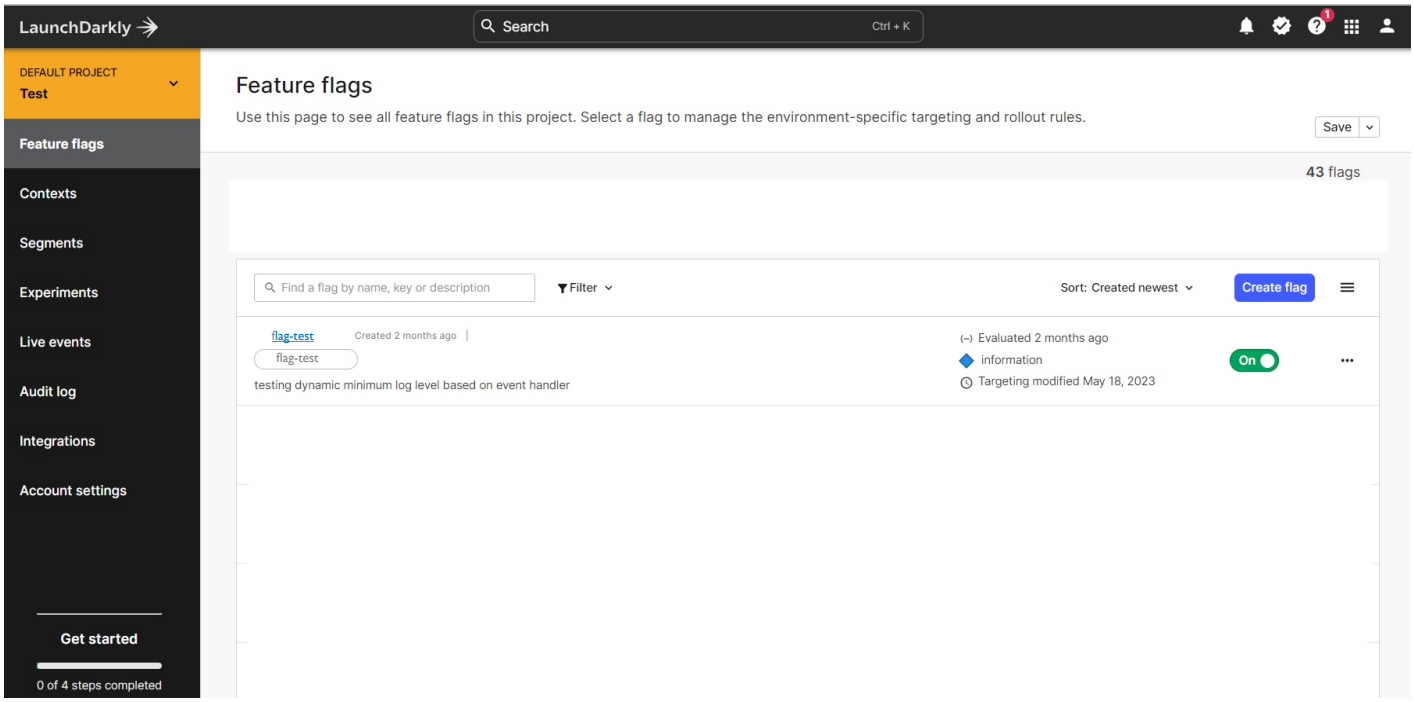



Figure 2 Sample UI

## Relay proxy

LaunchDarkly also supports a proxy based model to access its API. It provides a Go based application called relay proxy which acts as the proxy between client applications and the LaunchDarkly endpoint. It is a small application that connects to the LaunchDarkly streaming API and proxies that connects to clients within an organization's network. LaunchDarkly recommends using a Relay proxy based deployment model if the number of servers accessing the system is huge (in thousands).

The relay proxy lets multiple servers connect to a local stream instead of making many outbound connections to LaunchDarkly's streaming service. Relay proxy is an Open-Source project accessed in the URL- LaunchDarkly Relay Proxy (github.com). Relay proxy is also available as a docker image (Relay Proxy Docker).

Relay proxy is recommended in the following scenarios -

- Reducing the app's outbound connections – Relay proxy resides inside the organization's network
- Keeping user data private
- Facilitating faster connections
- Meeting continuation of service requirements
- Reducing firewall configuration complexity for customers
- Increasing startup speed for serverless functions.

**Figure 3 provides a typical deployment model using a Relay proxy.**

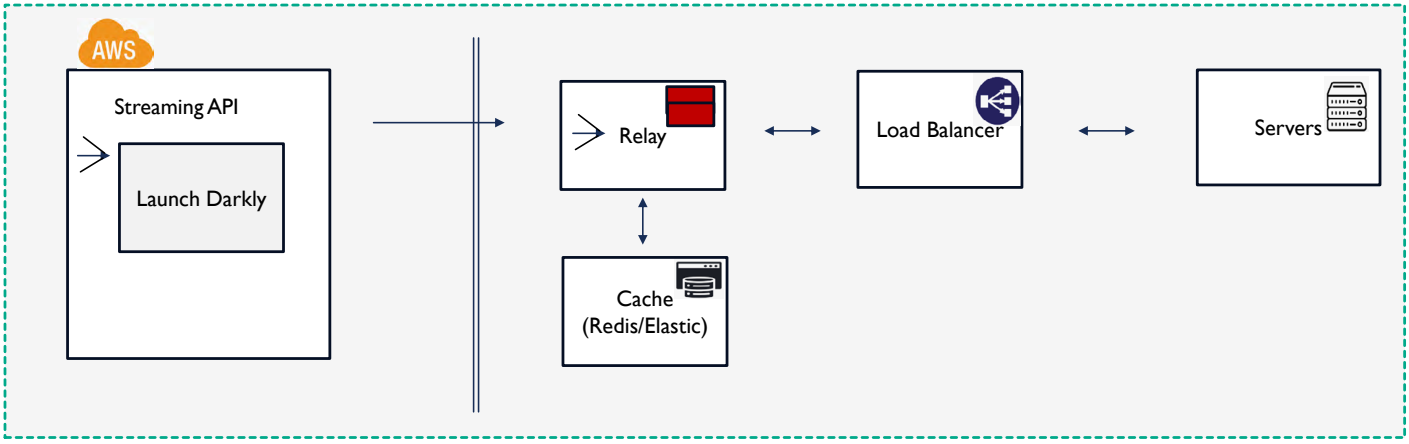



Figure 3 Deployment model using Relay proxy (Ref : docs.launchdarkly.com)

Note that a Relay proxy of server-side SDKs can store the flag evaluations in a local storage system like Redis so that each time the client application evaluates a flag, there is no external traffic to LaunchDarkly. The Relay proxy/SDK coordinates with LaunchDarkly to make the data in local storage up to date.

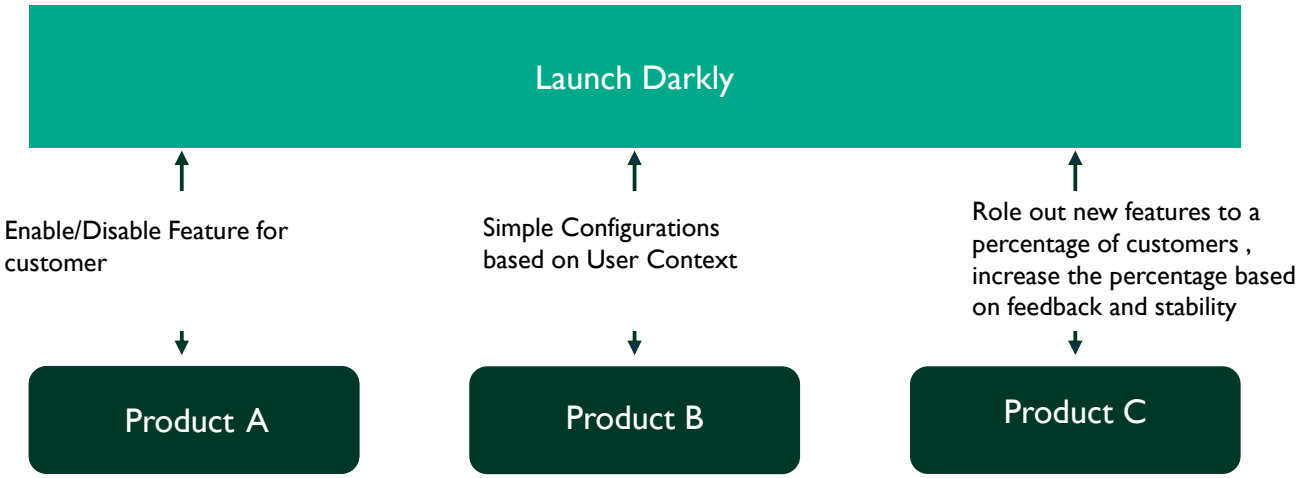
## Case study

Different product teams at a SaaS company, with many products in its portfolio, use LaunchDarkly in various capacities. Figure 4 shows the key use cases where LaunchDarkly is used now.

- Most product teams leverage LaunchDarkly to turn on and off certain features for different customers based on the product features in their subscriptions. This helps with a seamless rollout of new features to customers whenever they upgrade their subscriptions.
- Some product teams use the capability to provide non-Boolean value as feature flag evaluation for simple configurational parameters, which need to be different for diverse customers.
- Few products use the flagging capability to test new features in production by rolling out the features to a limited percentage of users to get early feedback.

Different product teams plan to use LaunchDarkly for more use cases by harnessing features like user segmentation and experimentation.

**Figure 4 LaunchDarkly Use Cases**




## Integrations

LaunchDarkly also supports integrations with different CICD tools, including -

- Ansible Collection
- AWS CloudTrail Lake
- AWS CloudWatch RUM
- Azure DevOps
- Bitbucket Pipelines
- Cloudflare
- Compass
- GitHub Actions Flag Evaluations
- Heap
- LogDNA
- Pendo
- Release
- Terraform
- Zapier
- Zendesk

For example, the Azure DevOps integration allows controlled rollouts to manage feature releases. With the integration, one can define a percentage rollout for the feature flags as part of a release task. More details on the specific integration functionalities are available at Integrations (launchdarkly.com).

## Why LaunchDarkly

There are different ways to manage feature flagging. Most organizations build custom solutions to provide feature flagging capability. Also, other solutions offer the feature flagging capability, like Azure based feature management. Available as part of Azure Cloud, it uses Azure App Configuration to manage feature flags. Azure portal gives a Feature Manager with a UI to configure feature flags. Azure based feature flags also support different filters for features to target a percentage of users, specific time windows or custom rules. But this does not support comprehensive features provided by LaunchDarkly, like user segmentation, context-based feature flagging and dynamic rules. Azure Feature flag only supports Boolean variations. Also, the SDK provided by Azure is for C# only, and code level changes are expected if the context needs to be updated.

Other competing products in this area are worth exploring as well. They include kameloon (kameleoon.com), a comprehensive AB testing platform with feature flagging support. Others are AB Tasty, Adobe Target and Optimizely (Optimizely). Most of it is in the AB testing and user segment analysis category.

## Conclusion

This document briefly introduces LaunchDarkly – a service that can be used to manage feature flagging efficiency. Application or product teams can rely on its capabilities to build and release software features in a controlled manner, targeting different user segments. As the ecosystem grows, it might add more capabilities to get insights into the usage of certain released features by utilizing the flags. Flags can also be used for a minimal level of configuration to different segments of users based on the user context or attributes. More detailed documentation is available on the LaunchDarkly website, which is given in the references section.

## References


1 LaunchDarkly docs

2 Server-side SDKs (launchdarkly.com)

3 LaunchDarkly REST API Documentation.


## Author


### Jereesh Thomas

Senior Technology Architect, Infosys



For more information, contact askus@infosys.com


Infosys®
Navigate your next