Feature Flags

DevOps Advanced

Agenda

- What is Feature Flag?
- Types of Feature Flags
- Feature Flags and CI/CD
- How to use
- Tooling

What is Feature Flag?

Feature Flags

What is Feature Flags?

- Feature flags are a software development concept that allow you to enable or disable a feature without modifying the source code or requiring a redeploy
- They are commonly referred to as feature toggles, release toggles, or feature flippers
- At a basic level, feature flags take the form of simple conditionals (e.g., if-else statements) that determine the code path that will be executed
- Feature flags determine at runtime which portions of code are executed
- This lets you deploy new features without making them visible to users, or you can make features visible only to specific subsets of users and environments.

Decoupling deploy from release

- Historically, deploy and release were synonymous because code that was deployed to the production environment was automatically running in production
- This meant that every new deploy carried significant risks to the existing application
- As a way to mitigate these risks, teams would bundle multiple features into releases that might happen once a week, once a month, or even once a quarter
- However, this actually raised the stakes of a deploy significantly and, if any single feature was broken in production, the entire bundled release needed to be rolled back, dramatically slowing down the velocity of development teams.

Decoupling deploy from release

- Feature flags change the traditional deployment workflow by decoupling deploy and release
- Allowing new code to exist in a production deploy but not be executed and, therefore, not released (aka a dark launch)
- This provides a safety net that enables teams to both lower risk and deploy more frequently
- Teams increase speed and stability, a hallmark of high-performing engineering teams, according to DORA metrics

How feature flags change software delivery

- Trunk-based development
  - With flags, you don't have to maintain multiple long-lived feature branches
  - Feature flags enable trunk-based development, a practice in which development teams all work in the main branch and/or frequently merge short-lived branches
  - This removes the complexity of merges and enables teams to move faster by allowing them to deploy code changes continuously
- Manage the feature lifecycle
  - Controlling who has access to a feature enables teams to control the full lifecycle of that feature
  - For example, a feature flag can make new features available only to a subset of users for testing and feedback before being released publicly
  - Or a release can be timed to align to with a marketing campaign without requiring a deploy or even intervention by engineering at all
  - Simply flip the flag and the feature is live with the full confidence that the success of the campaign will not be undermined by a failed or delayed deployment.

How feature flags change software delivery

- Remove the stress and risk around software releases
  - Feature flags reduce the stress around releases by allowing faster resolution of incidents
  - If a feature is causing problems, turning off the corresponding flag will disable it, solving the problem immediately without the need to roll back a complex release that may contain multiple critical features or fixes
  - Feature flags even enable Progressive Delivery, whereby a feature is gradually released to larger populations of users to ensure that it performs as expected.

Feature flags are NOT configuration files

- Generally, when you use configuration files to change the behavior of your software, you have to manually go into a file, change an environment variable, and then push the change through your deployment pipeline
- But using feature flags platforms make to evaluate in runtime. You flip a switch, and the change occurs instantly without you needing to redeploy, restart, or wait
- Also, config files act as blunt on-off switches for all users. Whereas, feature flags enable precise targeting and control, letting you roll out specific changes to specific audiences
- Point being, feature flags are more versatile, sophisticated, and powerful than config files

Types of Feature Flags

Feature Flags

Types of Feature Flags

- De-risk software releases (**release management**)
- Improve **reliability** and operational health
- **Target** and personalize software experiences
- Learn through product **experimentation** and A/B testing
- Reduce the **risk** of migrations

Feature Flags: Release Management

- Using flags as part of your build or feature release process is one of the most common uses of feature flags.
- Release management includes early access programs, canary releases, and beta programs—anywhere you give specific users access to features before releasing the feature 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

Feature Flags: Reliability

- Feature flags can manage and control the behavior of a system toggling a feature on/off to minimize the impact of incidents
- Using operational feature flags you can:
  - Deploy circuit breakers or kill switches to programmatically or manually disable a feature if it negatively impacts the user experience and alerts are triggered.
  - Limit API requests to ensure API reliability.
  - Switch to a less feature-rich version of a page under heavy load.
  - Test migrations to new microservices or third-party dependencies in production for interoperability testing.
  - Change log-levels on the fly for debugging purposes.

Feature Flags: Targeting

- If you use feature flags with a targeting engine you can release specific features and experiences to specific audiences
- With targeting, you can:
  - Personalize software experiences based on user preferences
  - Tailor features to users' mobile device and app version
  - Deliver targeted experiences based on region, geography, etc.
  - Gradually roll out features to target segments, starting with internal testers
  - Automate feature entitlements

Feature Flags: Experiments

- Monitoring and observability tools can tell you if a feature isn't working correctly, but they can't tell you if you built the right feature
- Experimentation or A/B testing can offer insights into things like how users are using your feature and/or whether it is performing as intended to ensure you release the correct feature
- Using feature flags, you can test different configurations of a feature to validate or disprove hypotheses
- Experiments provide concrete reporting and real world measurements to ensure that you are launching the best version of a feature that positively impacts company metrics

Feature Flags: Risk

- Infrastructure, cloud or software migrations are inherently high risk, so much so that often teams will avoid or postpone necessary migrations
- Feature flags can be used to help mitigate the risk of infrastructure migrations or avoid downtime in a cloud migration
- Combining infrastructure migrations using feature flags with progressive rollouts can help ensure that the change doesn't degrade performance
- Some feature flag services can even integrate directly with monitoring and observability platforms and automatically trigger a kill switch if specific performance thresholds are crossed

Feature Flags and CI/CD

Feature Flags

Feature Flags and CI/CD

- Feature flags and continuous integration/continuous delivery go hand in hand
- With continuous integration, development teams are encouraged to merge code into a main branch of the repository
- Traditionally, the number of feature branches tended to grow exponentially as the development team expanded
- The longer each branch lived, the more out of sync it would become with the main branch and with feature branches belonging to other team members. Integrating these branches was frequently extremely painful
- Continuous integration helps teams solve this problem by enabling them to merge frequently, with automated testing processes that helped prevent “breaking the build.”

Feature Flags and CI/CD

- The issue becomes, how do you combine frequent merges with feature development that may take days, weeks or even months without breaking the build?
- This is where feature flags come in. They allow you to define which code paths will be executed and which will be ignored. This allows teams to combine short-lived feature branches with longer running feature development.

Who uses feature flags?

- Since feature flags are implemented in code, they require a development or engineering team to configure them within the application
- As a result, software developers are typically the first to adopt feature flags for managing feature releases
- Feature flags are the underpinning of a feature management solution, which enables teams across the company to perform tasks that previously required engineering intervention. For example, utilizing a feature management platform with feature flags can enable:
  - QA to test functionality in production and validate changes.
  - Sales and Customer Support to provision entitlements.
  - Product Management to manage which end users get beta program access or align releases with non-engineering business priorities.
  - Marketing to run A/B tests or time releases to campaigns.
- Pairing feature flags with a feature management platform makes toggling flags easy to understand can enable release stakeholders across the company to contribute to the release lifecycle.

How to use

Feature Flags

How to use feature flags

- Many companies start using feature flags via a configuration file, often created in environment variables, a markup language like YAML, or configuration values in a service like Azure AppConfiguration
- This type of homegrown solution works well for flags that are either "on" or "off" for your entire user base
- However, it does require developer intervention to change a flag and, in the case of a physical config file, it may require a redeploy of changes
- Also, because using config files can get too long, it tends to be a better solution for project that only want a small number of flags.

Phases to use Feature Flags

- Design
- Development
- Configuration and management
- Testing and Verification
- Deployment and Monitoring
- Ongoing Management

Phases: Design

- Identify features to be controlled by feature flags.
- Determine the granularity and scope of each feature flag (e.g., per user, per group, globally).
- Plan for flag lifecycle management (creation, modification, retirement).
- When Feature Flags are something normal on development team, all features are covered with flags

Phases: Development

- Implement feature flags in the codebase
- Use conditional statements to check the status of feature flags
- Ensure fallback mechanisms are in place for disabled features
- Integrate feature flag libraries or services (e.g., Azure App Configuration, LaunchDarkly, Unleash).

Phases: Configuration

- Set up the feature flag management service
- Define and add feature flags in the service's interface.
- Configure the application to connect to the feature flag service
- Retrieve feature flag values during application startup or at runtime.

Phases: Testing

- Develop and execute unit and integration tests to ensure feature flags work as expected
- Crucial to have automated testing for taking the best from feature flags
- Verify that feature toggles enable and disable features correctly in different environments (development, staging, production).

Phases: Monitoring

- Deploy the application with feature flag checks in place
- Use the feature flag service's dashboard to monitor the status and usage of feature flags
- Gradually roll out features using flags and monitor application performance and user feedback
- Crucial to have a proper monitoring process in place

Phases: Ongoing

- Continuously monitor and update feature flags based on operational needs and user feedback
- Clean up and remove deprecated or unused feature flags from the codebase and configuration service
- Document feature flag usage and management procedures for the development team

Tooling

Feature Flags

Tooling

- There are several tools available on the market for Feature Flags
- A great initiative was the creation of OpenFeature on CNCF to bring some standardization for Feature Flagging
- This project wants to create standard for Feature Flags platforms
- Some platforms with stable usage:
  - Azure App Configuration (equivalence on other clouds)
  - LaunchDarkly
  - FeatureToggle
  - Unleash

Comparison

Feature / Platform	Azure App Configuration	LaunchDarkly	FeatureToggle	Unleash
Integration	Seamless with Azure services, .NET, Java, Python, Node.js	Broad language and framework support (Java, .NET, Node.js, Python, Ruby)	Simple .NET-based integration	Wide range of SDKs (Java, .NET, Node.js, Python, Go, Rust)
UI & Management	User-friendly portal, integrated with Azure ecosystem	Intuitive UI, extensive feature management capabilities	Basic UI, focused on simplicity	User-friendly dashboard, suitable for tech and non-tech users
Real-time Updates	Near real-time updates via push notifications	Real-time flag changes and analytics	Limited real-time capabilities	Near real-time updates
Scalability	High scalability, Azure-backed infrastructure	Highly scalable, suitable for large enterprises	Suitable for small to medium projects	Scalable, open-source flexibility

Comparison

Feature / Platform	Azure App Configuration	LaunchDarkly	FeatureToggle	Unleash
Granular Targeting	Limited advanced targeting options	Advanced targeting rules, A/B testing, user segments	Basic targeting options	Granular targeting with strategies
Pricing	Pay-as-you-go, integrated with Azure subscription	Subscription-based, tiered pricing	Free for basic usage, paid plans for advanced features	Open-source (free), commercial support available
Analytics & Insights	Basic metrics, Azure Monitor integration	Advanced analytics and insights, experiment results	Limited analytics	Basic analytics, with plugins for more insights
Security & Compliance	Azure security standards, compliance with major regulations	Strong security features, SOC2, GDPR, HIPAA compliant	Basic security features	Good security practices, open-source community-driven improvements

Usng Feature Flags

Demo