Engineer

Feature-Flag

⚑ Code ohne erneuten Build der Anwendung ein- oder ausschalten

Was es ist

Ein Feature-Flag ist eine Entscheidungsstelle im Code, die eine Einstellung liest, die das Team ohne erneuten Build der Anwendung ändern kann. Dahinter liegt ein neuer Codepfad, der bereits in Produktion ist, aber ruht, bis die Einstellung ihn einschaltet, für alle oder für eine ausgewählte Nutzergruppe. Pete Hodgson beschrieb die Technik ausführlich auf Martin Fowlers Website, in mehreren Teilen ab Januar 2016 und mit einer Überarbeitung im Oktober 2017. Er nennt sie Feature Toggles und führt Feature Flags, Feature Bits und Feature Flippers als weitere Bezeichnungen für dasselbe auf.

Hodgson zerlegt ein Flag in Bestandteile. Der Toggle Point ist die Stelle im Code, die fragt, ob der neue Pfad eingeschaltet ist. Der Toggle Router beantwortet die Frage anhand der Toggle-Konfiguration und, wenn die Antwort vom Fragenden abhängt, anhand eines Toggle-Kontexts: meist des aktuellen Nutzers oder eines speziellen Cookies oder HTTP-Headers, der stellvertretend für ihn steht. Hodgson warnt davor, beide zu koppeln. Der Code am Toggle Point sollte eine eng gefasste Frage stellen, etwa ob diese Bestellbestätigung den neuen Stornierungslink enthält. Die entscheidende Regel liegt dagegen zentral hinter dem Router, damit sie sich ändern lässt, ohne jede Aufrufstelle anzufassen.

Flags unterscheiden sich nach Lebensdauer und Häufigkeit wechselnder Antworten. Hodgson ordnet sie vier Arten zu. Release Toggles lassen unfertigen, ungetesteten Code als ruhenden Code in Produktion gelangen. Sie sollten nach seinen Worten „im Allgemeinen nicht viel länger als ein oder zwei Wochen bestehen bleiben“. Experiment Toggles teilen Nutzer für einen A/B-Test in Gruppen ein; jeder Nutzer landet jedes Mal in derselben Gruppe. Ops Toggles ermöglichen dem Betriebspersonal, das Verhalten des produktiven Systems zu ändern. Viele sind kurzlebig, doch einige langlebige Notausschalter ermöglichen bei ungewöhnlich hoher Last, verzichtbare Funktionalität zurückzufahren, etwa ein teuer zu erzeugendes Empfehlungsfeld. Hodgson vergleicht sie mit einem

manuell verwalteten Circuit Breaker. Permissioning Toggles entscheiden, welche Funktionen ein bestimmter Nutzer sieht, etwa Premium-Funktionen für zahlende Kunden oder Alpha-Funktionen für eigene Beschäftigte. Sie können jahrelang bestehen bleiben.

Ein Feature-Flag ist kein Canary Release, auch wenn beide oft zusammenwirken. Ein Canary betreibt zwei Versionen eines Dienstes nebeneinander und leitet einen Teil des Verkehrs zur neuen. Ein Flag liefert eine Version mit beiden Pfaden und wählt bei jeder Anfrage zwischen ihnen. Auch ein langlebiger Branch in der Versionsverwaltung hält neue Arbeit getrennt. Der Code gelangt dann aber erst beim Zusammenführen des Branches in Produktion, und zwar auf einmal.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Feature-Flag?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Ein Team führt eine Idee von der Skizze bis zu einem Produkt, bei dem Ihre Kunden sich anmelden können, statt sie zwischen drei Dienstleistern weiterzureichen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird.
GIS-Kartenprodukt zu 10-fachem Umsatz geführt — 10x Umsatzwachstum durch die Leitung des Kartenprodukts

GIS-Kartenprodukt zu 10-fachem Umsatz geführt Kontakt aufnehmen →

Warum es wichtig ist

Der Hauptgrund für Flags ist die Trennung von Bereitstellung und Freigabe. Hodgson beschreibt Release Toggles als häufigste Umsetzung eines Prinzips von Continuous Delivery: Die Freigabe einer Funktion ist ein eigener Schritt, getrennt von der Bereitstellung ihres Codes. Code kann täglich aus einem einzigen Hauptbranch in Produktion gelangen, während eine Funktion, deren Entwicklung einen Monat dauert, ausgeschaltet bleibt, bis sie fertig ist. Das macht Trunk-Based Development praktikabel.

Weil ein Flag verschiedenen Nutzern verschiedene Antworten geben kann, lässt sich eine Freigabe schrittweise ausweiten. Hodgson beschreibt, neue Funktionen zuerst für interne Nutzer einzuschalten. Das nennt er Champagne Brunch, weil das Unternehmen zuerst seinen eigenen Champagner trinken darf. Danach folgt eine Canary-Kohorte: In seinem Beispiel wird eine zufällige Gruppe von 1 Prozent der Nutzer durch eine Modulo-Operation auf der Nutzer-ID ausgewählt, damit

dieselben Personen in der Gruppe bleiben. Die übrigen 99 Prozent behalten das alte Verhalten. Meldet die Kohorte Probleme, wird das Flag ausgeschaltet, ohne dass jemand eine Bereitstellung zurückrollen muss.

Das alles hat seinen Preis. Jedes Flag ergänzt eine Verzweigung im Code und eine zu testende Kombination. Die dahinterliegende Konfiguration wird zu einem Teil der Produktion, der sich ohne Build ändern kann. Hodgson empfiehlt, Toggle-Konfigurationen in der Versionsverwaltung zu halten und sie durch erneute Bereitstellungen zu ändern, wo das Flag dies zulässt. Die Konfiguration durchläuft dann dieselbe Pipeline wie der Code, und ihre Änderungen werden geprüft und aufgezeichnet. Bei Flags, die sich innerhalb von Minuten ändern müssen, etwa Ops Toggles, rechtfertigt sich eine Laufzeitkonfiguration.

Wie es in der Praxis aussieht

flagd ist eine quelloffene Engine zur Flag-Auswertung nach der OpenFeature-Spezifikation. Sie hat keine Benutzeroberfläche und speichert selbst nichts. Sie liest Flag-Definitionen aus einer JSON-Datei oder einer anderen Quelle und beantwortet Anfragen, entweder als Sidecar, im Anwendungsprozess oder als zentraler Dienst. Ein Flag, das etwa jeden zwanzigsten Nutzer zu einem neuen Kaufabschluss leitet, sieht so aus:

```
{
  "$schema": "https://flagd.dev/schema/v0/flags.json",
  "flags": {
    "new-checkout": {
      "state": "ENABLED",
      "variants": {
        "on": true,
        "off": false
      },
      "defaultVariant": "off",
      "targeting": {
        "fractional": [
          ["on", 5],
          ["off", 95]
        ]
      },
      "metadata": {
        "owner": "checkout",
        "expires": "2027-01-31"
      }
```

```
    }
  }
}
```

Jedes Flag hat einen Zustand und eine Menge von Varianten, die alle denselben Typ haben müssen; die Standardvariante ist optional. Die Regel `fractional` verteilt Nutzer nach relativen Gewichten auf Gruppen. 5 und 95 setzen daher etwa jeden zwanzigsten Nutzer auf `on`. Standardmäßig wird die Gruppe aus dem Flag-Schlüssel und dem `targetingKey` im Auswertungskontext berechnet. Derselbe Nutzer erhält dadurch immer dieselbe Antwort, und das Umbenennen des Flags verteilt alle neu. `metadata` enthält beliebige vom Team angehängte Angaben, hier einen Verantwortlichen und das Datum, bis zu dem das Flag entfernt sein sollte. Wird `state` auf `DISABLED` gesetzt, antwortet flagd ganz ohne Wert. Der Aufrufer fällt dann auf seinen eigenen Standardwert zurück.

flagd mit der Datei starten und über HTTP eine Antwort anfordern:

```
flagd start --uri file:./flags.json
curl -X POST "localhost:8013/flagd.evaluation.v1.Service/ResolveBoolean" \
  -H "Content-Type: application/json" \
  -d '{"flagKey":"new-checkout","context":{"targetingKey":"user-4711"}}'
```

Die Antwort enthält den Wert, die Variante und den Grund `TARGETING_MATCH`. In der Anwendung erfolgt der Aufruf über einen OpenFeature-Client mit dem Flag-Schlüssel, dem Standardwert `false` und dem Auswertungskontext. Der Toggle Point sieht nur wahr oder falsch. Den Anteil von 5 auf 20 Prozent zu erhöhen bedeutet eine Änderung der Flag-Datei, keine Änderung am Build der Anwendung.

Die Arbeitsbereiche für Personalgewinnung und die Laufbahn von Seeleuten aus der seit 2025 laufenden Tätigkeit zeigen ein Release-Flag im dokumentierten Implementierungsstand: Der Bereich für Personalgewinnung wird als live beschrieben, der Laufbahnbereich als per Feature-Flag gesteuert und unvollständig freigegeben. Das ist eine historische Momentaufnahme, keine Aussage über den heutigen Bereitstellungsstand. Im Backoffice für den Betrieb aus derselben Tätigkeit werden solche Flags geändert. Das Betriebspersonal schaltet Flags um und aktiviert Funktionen über eine Oberfläche nur für

bestimmte Nutzer, statt dass eine technische Fachkraft die Produktionsdatenbank von Hand ändert. Jede Änderung wird dort wie jede andere Aktion autorisiert und aufgezeichnet.

Wo es schiefgeht

Flags, die nie verschwinden. Ein Release-Flag, das seinen Zweck erfüllt hat, verursacht nur noch Kosten: eine unnötige Verzweigung, einen Pfad, den niemand testet, und eine Einstellung, die jemand versehentlich umlegen kann. Hodgson schlägt vor, Flags als Bestand zu behandeln, der Kosten verursacht, und ihn klein zu halten. Zu seinen Gegenmitteln gehören eine Aufgabe zum Entfernen im Backlog, sobald ein Release-Flag eingeführt wird, ein Ablaufdatum, eine Zeitbombe, die nach Ablauf einen Test scheitern lässt oder den Anwendungsstart verhindert, und eine feste Höchstzahl von Flags, sodass für ein neues ein anderes entfernt werden muss. Ein Flag zu entfernen heißt, neben der Prüfung auch den verworfenen Codepfad zu löschen.

Ein altes Flag erhält eine neue Bedeutung. Am 1. August 2012 schickte Knight Capitals automatischer Order-Router SMARS innerhalb von 45 Minuten Millionen Aufträge an den Markt. Das Unternehmen verlor mehr als 460 Millionen US-Dollar. Die SEC-Verfügung beschreibt den Ablauf. Neuer Code für das Retail Liquidity Program der New York Stock Exchange verwendete ein auf eingehenden Aufträgen mitgeführtes Flag erneut. Es hatte früher Power Peg eingeschaltet, eine seit 2003 nicht mehr genutzte, aber nie entfernte Funktion. Bei der Bereitstellung wurde einer von acht Servern ausgelassen. Dort weckte das wiederverwendete Flag die alte Funktion. Als Beschäftigte den neuen Code anschließend von den anderen sieben entfernten, aktivierte das Flag auch dort Power Peg. Geben Sie einem ausgemusterten Flag nie eine neue Bedeutung, löschen Sie toten Code zusammen mit seinem Schutz-Flag und prüfen Sie vor dem Einschalten, ob alle Server denselben Build ausführen.

Kombinationen, die niemand getestet hat. Zwei Flags ergeben vier Kombinationen, zehn ergeben über tausend. Hodgson nennt das eine kombinatorische Explosion. Alle zu testen ist selten möglich. Testen Sie daher die Kombinationen, die in Produktion gelangen können: die aktuelle Produktionskonfiguration mit eingeschalteten, kurz vor der Freigabe stehenden Flags, den Rückfallzustand mit diesen Flags ausgeschaltet und, wie viele Teams zusätzlich, alles eingeschaltet. Eine feste Konvention hilft: „Aus“ bedeutet immer das bestehende Verhalten, „Ein“ das neue.

Ein Standardwert, den niemand gewählt hat. Laut OpenFeature-Spezifikation darf die Flag-Auswertung keine Ausnahme auslösen. Geht etwas schief, gibt der Aufruf den vom Aufrufer übergebenen Standardwert zurück. Die ausführliche Ergebnisform enthält einen Fehlercode. Auch der Client sollte nichts ins Log schreiben, weil eine vor der Definition ihres Flags bereitgestellte Anwendung das Log sonst mit `FLAG_NOT_FOUND` fluten würde. Das hält die Anwendung am Laufen, bedeutet aber auch, dass ein Tippfehler im Flag-Schlüssel unbemerkt für alle Nutzer den Standardwert ergibt. Übergeben Sie das alte, sichere Verhalten als Standard und hängen Sie einen Hook an, an dem die Spezifikation das Logging vorsieht. Zählen Sie damit Auswertungen, die mit einem Fehler enden, und lösen Sie bei deren Auftreten einen Alarm aus.

Überschreibungen, die jeder setzen kann. Tester können ein Flag bequem mit einem speziellen Cookie, einem URL-Parameter oder einem HTTP-Header einschalten. Hodgson weist darauf hin, dass neugierige oder böswillige Nutzer dieselbe Überschreibung setzen und damit den Zustand eines Flags für sich ändern können. Berücksichtigen Sie solche Überschreibungen nur bei internem Verkehr oder signieren Sie sie kryptografisch. Das entschärft die Sorge, erhöht aber, wie Hodgson anmerkt, selbst Komplexität und Angriffsfläche.

Weiterführende Lektüre

- Feature Toggles (auch Feature Flags) — Pete Hodgsons Artikel auf Martin Fowlers Website: Bestandteile eines Toggles, die vier Kategorien, Tests und das Kleinhalten des Bestands.
- API zur Flag-Auswertung — die OpenFeature-Spezifikation zum Auswerten von Flags, einschließlich Standardwerten, Fehlercodes und dem Grund, warum die Auswertung keine Ausnahmen auslöst.
- flagd: Flag-Definitionen — das von flagd gelesene JSON-Format: Zustand, Varianten, Standardvariante, Zielgruppenauswahl und Metadaten.
- flagd: Fractional-Operation — deterministische prozentuale Aufteilungen und die Auswahl der Gruppe eines Nutzers.
- In der Sache Knight Capital Americas LLC — die SEC-Verfügung von 2013 zum Handelsverlust im August 2012, einschließlich des wiederverwendeten Flags.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Feature-Flag?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/feature-flag/