

ROLLOUT, FEATURE FLAG, EXPERIMENT, OPERATIONAL TOGGLE

Different use cases for backend, frontend
and mobile

**"FEATURE FLAGS" TEND
TO COME UP WHEN
TALKING ABOUT
CONTINUOUS
DEPLOYMENT**

CI: continuous integration

CD: continuous delivery

CD: continuous deployment

TYPES

1. rollout
2. feature flag
3. experiment
4. operational toggle

ROLLOUT

FOR **ROLLING OUT** A NEW VERSION OF SOFTWARE

Short-lived using percentages

- a new deployment of kubernetes
- new APK released to the Play Store

FEATURE FLAG

FOR TURNING A FEATURE **ON** OR **OFF**

Medium-lived using allow list, A/B test, percentage, app version, etc.

- :new-chargeback-flow
- :new-debit-card-activation-screen

EXPERIMENT

FOR ANALYSING BEHAVIOUR

Medium-lived using allow list and A/B test
– :debit-withdrawal-test

OPERATIONAL TOGGLE

FOR DISABLING FEATURES IN #CRASH-LIKE SITUATIONS

Long-lived using percentage

- :bank-barcode-payment
- :savings-bank-barcode-query-provider

We know know about the types

**BUT THEY HAVE
DIFFERENT RELEVANCE
FOR BACKEND,
FRONTEND AND MOBILE**

BACKEND

1. rollout: k8s blue/green, canary and common-rollout
common-xp
2. feature flag: common-rollout common-xp and datasets
3. experiment: common-xp
4. operational toggle: common-rollout common-xp

FRONTEND

1. rollout: CDN and page refreshes
2. feature flag: percentages and maybe IPs (no :customer/id on the website)
3. experiment: via dynamic backend control
4. operational toggle: via dynamic backend control

BACKEND

1. rollout: app stores
2. feature flag: via dynamic backend control
3. experiment: via dynamic backend control
4. operational toggle: via dynamic backend control

Key differentiator is

**HOW MUCH CONTROL WE
HAVE OVER THE
ENVIRONMENT**

BACKEND

FULL CONTROL

FRONTEND

PARTIAL CONTROL

We choose when to make a new version available

MOBILE

VERY LIMITED CONTROL

- app stores can restrict updates (worse for iOS)
- customers still have to download new versions

COSTS

- more complex code
- compatibility with old app versions
- nesting is exponential

BENEFITS

- dynamicity

**WEIGHTING COSTS ×
BENEFITS**

**THE LESS CONTROL WE HAVE, THE
MORE WE VALUE DYNAMICITY**

WEIGHTING COSTS × BENEFITS

- backend: sometimes worth the cost
- frontend: almost always worth the cost
- mobile: **always** worth the cost

BEST PRACTICES

DYNAMIC CONTENT > FEATURE FLAG

Always true for mobile, almost always for frontend

USE :INCLUDE-LIST FOR NAMED GROUPS

Always true for backend, frontend and mobile

```
{:rules  
  #{{:types    :include-list  
    :content  {:filename "debit-team-members.txt"}}}}
```


EXTEND COMMON-ROLLOUT COMMON-XP IF REQUIRED

That's how :include-list, :app-version, etc. were born

BEWARE OF MANY NESTED FEATURE FLAGS

True for backend, frontend and mobile

DON'T DELETE APP-FACING FEATURE FLAGS

True for mobile

**INCLUDE A FEATURE
FLAG ON THE
WHITEBOARDING PHASE**

**INCLUDE
DELETING/RETIRING THE
FEATURE FLAG AT THE
END**

AVOID RENAMING A FEATURE FLAG

Use :app-version with :min-version instead

**AND MOST
IMPORTANTLY...**

ALWAYS RELY ON A FEATURE FLAG ON THE APP

Never do a hotfix, avoid expedited releases at all costs

REFERENCES

1. "Feature Toggles (aka Feature Flags)", by Pete Hodgson
2. "Continuous integration vs. delivery vs. deployment", by Sten Pittet
3. Accelerate, by N. Forsgren, J. Humble and G. Kim
4. these slides: euandre.org/slide/
5. prose version of this presentation
6. view source