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

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

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"}}}}
```

Always use :app–version

Only for mobile

```
{:rules
 #{{:types   :app-version
    :content {:min-version #{{:platform :android
                               :code     1000000}
                              {:platform :ios
                               :code     2000000}}}}}}
```

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