Feature Flags / Toggles

Often Hinted at. Seldom Introduced

Level: Beginner

Disclaimer

This talk isn’t sponsored by Unleash.

Unleash was picked due to the availability of a live web demo and the ability to self host.

$ whoami

SOAR Dev/Sec/Ops @ Desjardins
Formerly DecSecOps @ FOCUS

- 10 Hosts / 50 Services
- < 5 transactions/s
- 1 environment
- < 1 deployment/week
- ~5k IoT Devices
- 5th dev

- 100 Hosts / 400 Services
- 100 transactions/s
- 10 environments
- 200 deployments/day
- ~50k IoT Devices
- 15 devs

This is the talk I wish I'd have heard at ConFoo whilst we were speeding up

Table of contents

What are Feature Flags?

What are Feature Flags

> A condition within the code enables or disables a feature during runtime

Wikipedia

```
if (FLAG) {
    // Feature
}
```

Terminology: Features ...

Flags

Launchtime

Simpler

Toggles / Switches

Runtime

Features ++

Informally, all are used interchangeably

What aren’t Feature Flags

- User-Selectable Configuration
  - Country/Region Selectors
  - Dark Mode
  - Language Selectors
- Your Admin Dashboard

What shouldn’t Be Behind Feature Flags

- Bugfixes
- Everything
  - The Kitchen Sink

Homemade Alternatives to Feature Toggles Platforms

Configuration Options

"Compiler" Flags

- Once and Done
- Requires Better Artifacts Versionning

Environment Variables / Configuration Files

- No External Dependencies
- Allows for Environment-Specific Flags
- Might need to use IaC/CaC

Software

Access Control

Pros

- Works when developing *new* features/components
- Enables Canary Releases

Cons

- Loses its utility once the feature is released
- Can only handle feature overhauls by treating them as new features

In-House Platform

Pros

- Probably meets your current needs
- Might meet your **very complex** needs
- Might have to handle *sensitive* data

Cons

- Or Does it?
- **Are you in the business of selling Feature Toggles Platforms?**
- Does it support the Open Feature spec?

Why Use Feature Flags?

- Product / Feature Identification
- Continuous Integration
- Decouple Deployment from Release
- Environment-Specific Feature Sets
- Pre-release UATs

Product Identification

image: NVidia

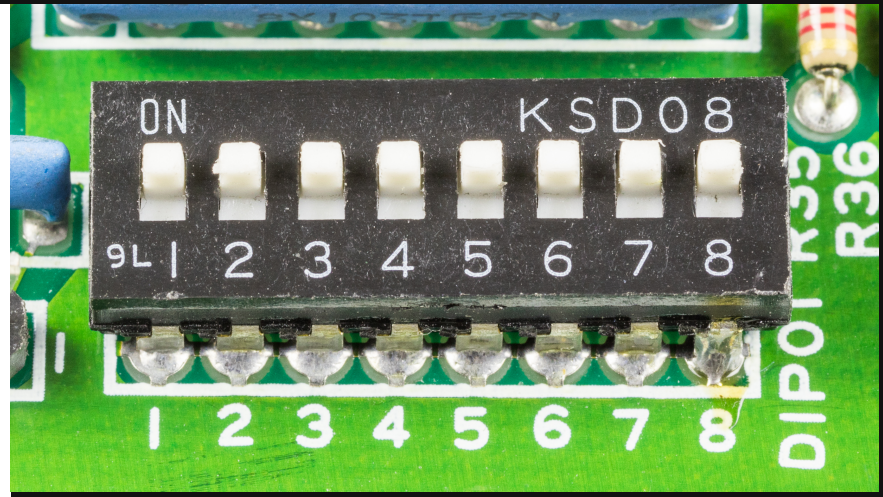


Image: Raymond Spekking

Decouple Deployment from Release

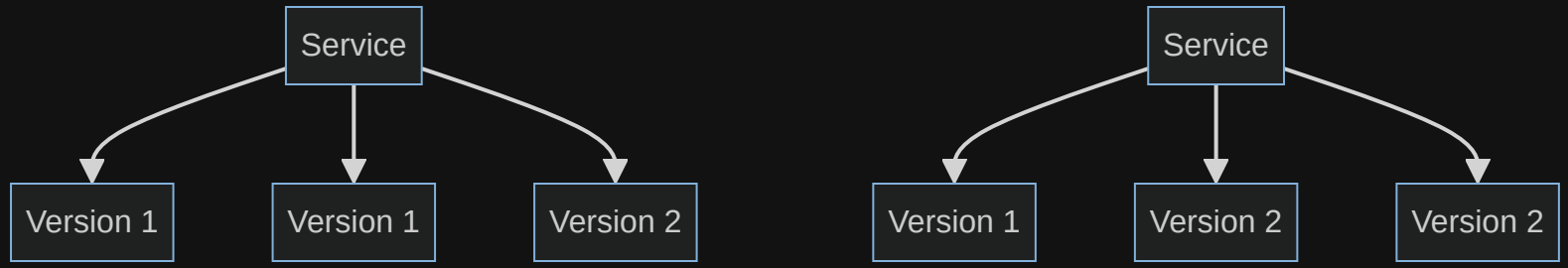
Going past *big bang* releases

- Rolling Release
- Canaries
- A/B Testing
- ...

Business Constraints

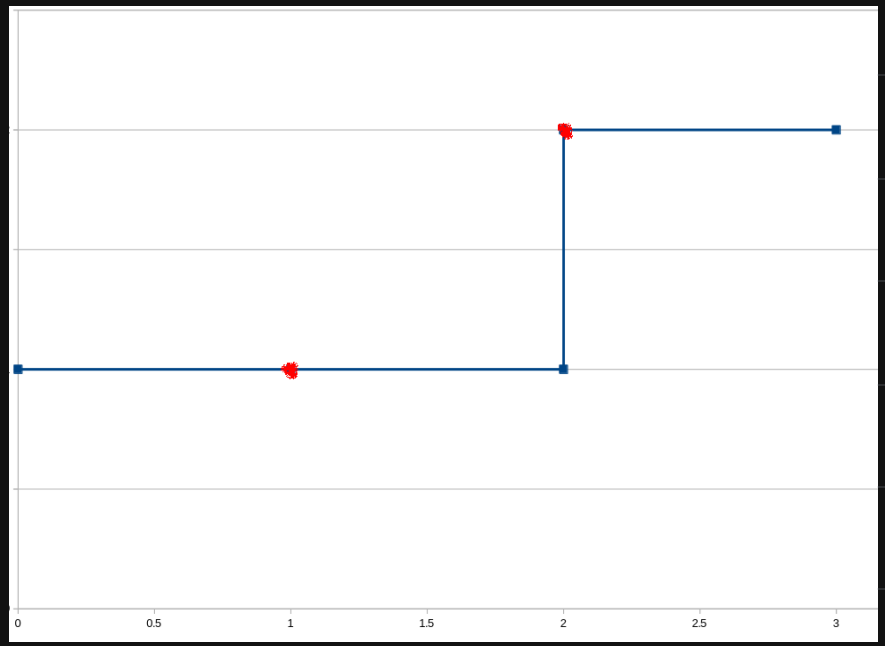
- If you have to release a feature at a given date.
  - Murphy's Law will fail your merge/deployment
  - Better to have the code ready in prod
- If you have regulatory frameworks to follow

Rolling Releases



Pitfall: There’s a special kind of emergent failure modes in your changes when using rolling release

Example Transient Rollover Failure Window



id	Server	X	Y	Odo	Expected Odo
0	2	0	1	0	0
1	1	1	1	null	1
2	2	2	1	2	2
3	1	2	2	null	3
4	2	3	2	3.44	4

On distributed systems, non-breaking changes may cause unexpected behaviors when being progressively released, **if they rely on data processed by other instances**

Transient Rollover Failure Window

Killswitches

Think of Feature Flags.
Used the Other Way Around

image: Stahlkocher

UATs

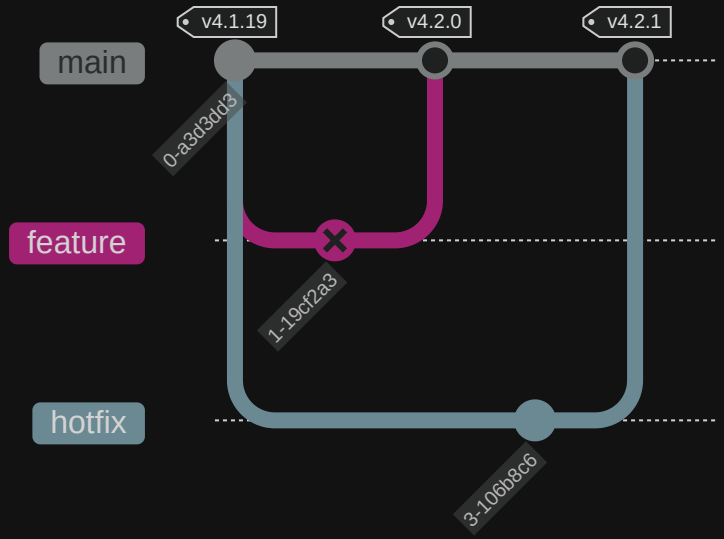
Most modern Feature Flags System Implementation offer environment management.

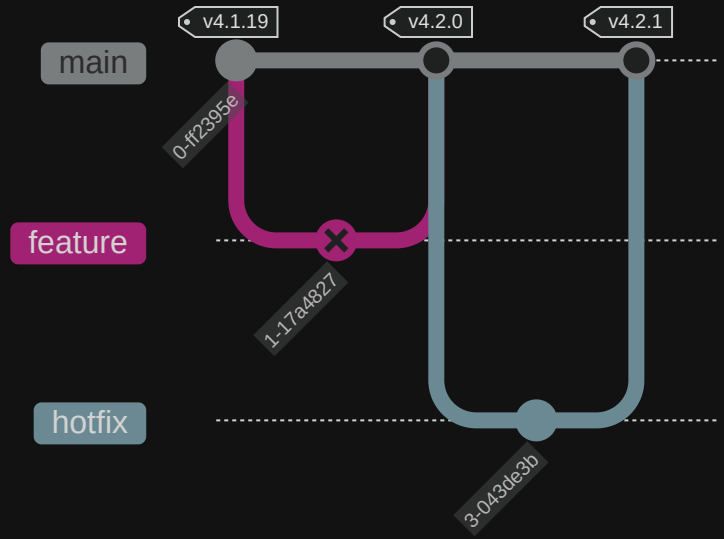
It is trivial to flag an environment for UAT for a given feature.

Note: Unleash Free/Self-Hosted is limited to Dev/Prod Environments

Storytime: When CD gets ahead of the Team!

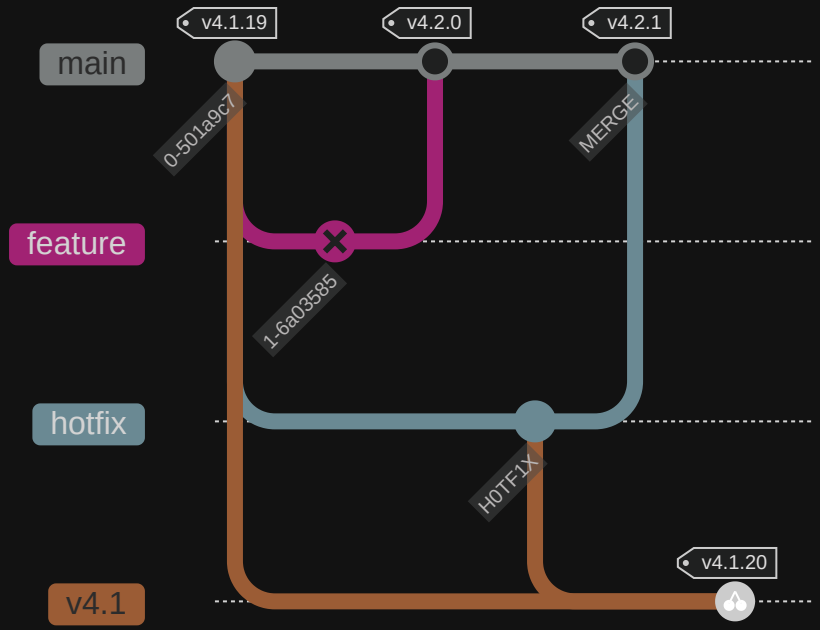
- You're releasing a new feature, say `v4.2.0`.
- It's promply rolled back.
- Customer Support escalates an *unrelated* important bug that was present prior to `v4.1.19` releases
- Which version will you hotfix?



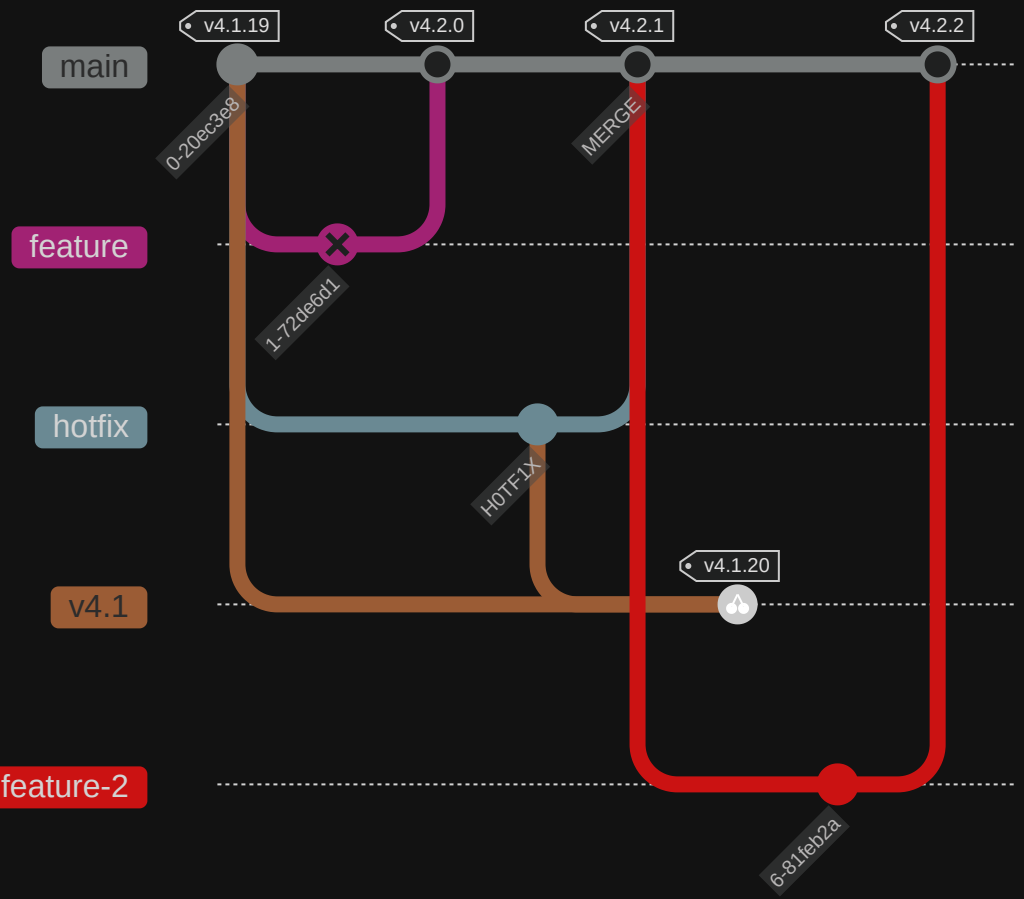


I sense a meeting in your near future...

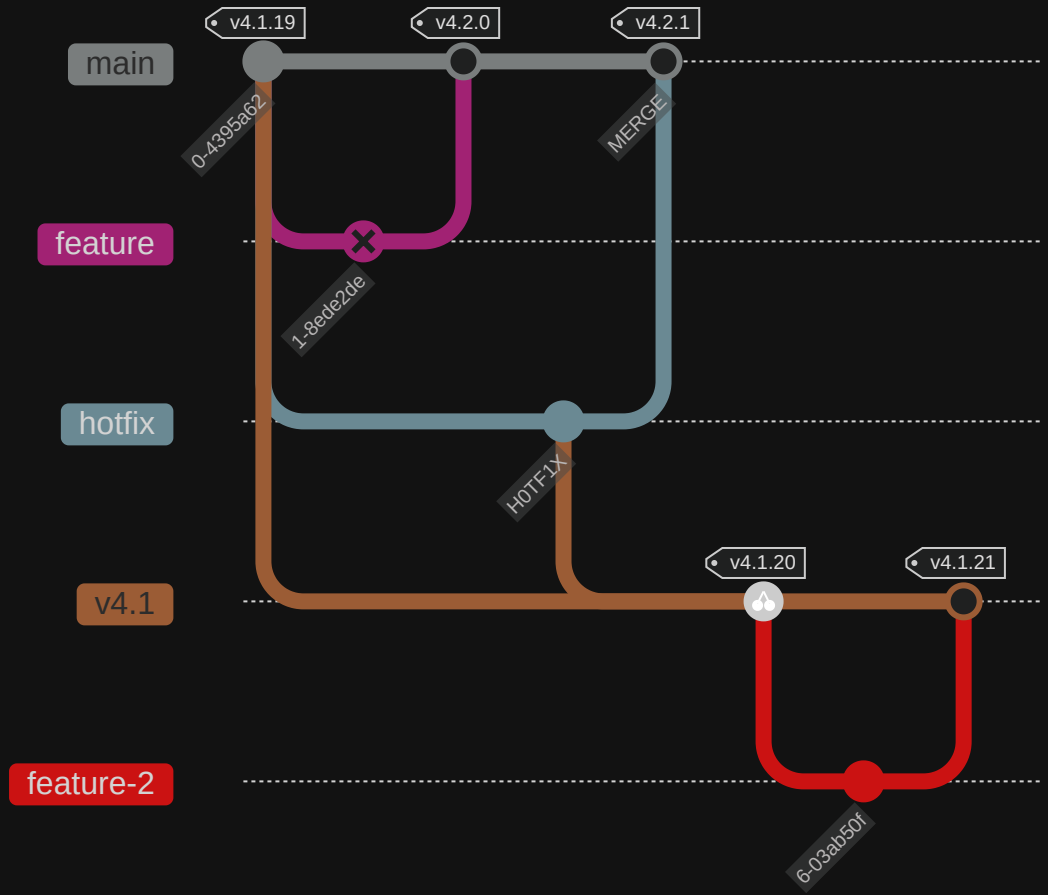
You could create a version branch



Your CD is still primed to release



Now, your team has to work on the version branch...



Using Unleash Platform

Install Dependencies

```
npm i koa unleash-client
npm i -D @types/koa typescript
```

Sample KoaFoo App

```
import Koa from 'koa'

const app = new Koa();

// response
app.use(ctx ⇒ {
  ctx.body = 'Hello ConFoo YUL 2024!';
});

app.listen(3000);
```

Plug in Unleash

& **Definitely** Functionnal New Feature

```
import Koa from 'koa'
import { startUnleash } from 'unleash-client';

const app = new Koa();

const unleash = await startUnleash({
    url: 'https://YOUR-API-URL',
    appName: 'my-node-name',
    customHeaders: { Authorization: 'SOME-SECRET' }
})

// response
app.use(ctx => {
  ctx.body = 'Hello ConFoo YUL 2024!';

  if (unleash.isEnabled("I-Did-Not-Break-This-Demo")) {
    throw new Error("Works On My Machine(TM)!")
  }
});

app.listen(3000);
```

Demo

- Self-Hosted Portal
- Add Flag
- http://localhost:3000

Going Forward

Feature Flags Lifetime

Release Flags

As Short-Lived As Possible

Killswitches

Permanent

Best Pratices:

- Do not put business logic in your Toggles
- Do not put Confidential Information (eg: PII) in your toggles
- Instrument your feature branches
- Remove the dead branch
  - Or at least have a new ticket to do so at a later date
  - Also Prune / Archive old flags in your tracker

Release Strategies

- On/Off
- Progressive Rollout
- Custom Logic
  - Application
  - User
  - Environment
  - ...

Conclusion

- Make sure the solution you're using meets your needs
- Feature Flag Systems will help you smooth your release process when scaling
- Use as few features at a time as possible

Thank You!

Questions?
Comments?
Insults?

Slides Deck: https://github.com/carboneater/confoo-2024-feature-flags