Feature flags: the toggle, the A/B test, and, the canary

Santosh Hari

Agenda

- Feature Flags
- Difference between Deployments and Release
- Feature Flag Frameworks and Services
- Feature Flag Use Cases
- Lifecycle and technological considerations

Let’s say you

Feature Flags

Feature Toggles (often also referred to as Feature Flags) are a powerful technique, allowing teams to **modify system behavior without changing code.**

https://martinfowler.com/articles/feature-toggles.html

In software, a flag is "one or more bits used to store binary values"
aka a Boolean that can either be true or false.
In the same context in software, a feature is a chunk of functionality that delivers some kind of value.
Thus, a feature flag, in the simplest terms, is an IF statement surrounding some chunk of functionality in your software.
Feature flags, in reality, can be and are more complex than that.
A feature flag is a way to change your software's functionality without changing and re-deploying your code.
Feature Flags, also referred to as "Feature Toggles" is a set of patterns which can help a team to deliver new functionality to users rapidly but safely.

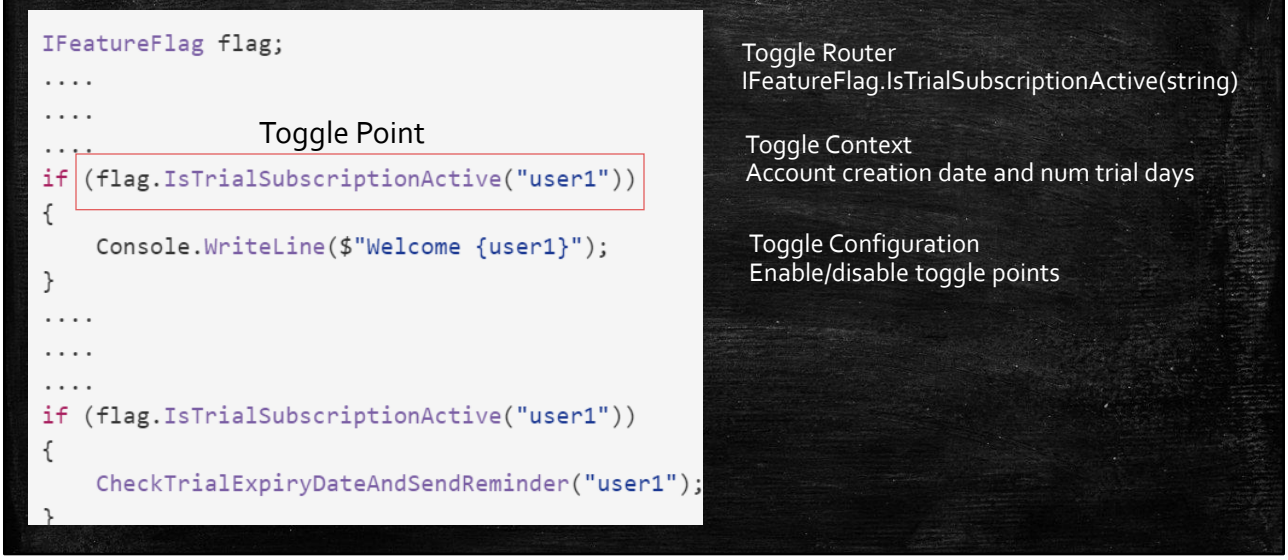
Simple Feature Flag

```
{
  "MyFeature": {
    "Flag1": true
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

```
public class TestModel : PageModel
{
    // requires using Microsoft.Extensions.Configuration;
    private readonly IConfiguration Configuration;

    public TestModel(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public ContentResult OnGet()
    {
        bool flag1 = bool.Parse(Configuration["MyFeature:Flag1"]);

        string featureState = "not available";
        if (flag1)
        {
            featureState = "available";
        }

        return Content($"Feature {featureState}");
    }
}
```

The simplest version of a config flag in .NET Core involves two steps:
1. Add a config setting from appsettings.json files
2. Read Config settings from C# code and use the flag in code flow

Components of a Feature Flag



While it's not advisable to daisychain feature flag, one flag being used in multiple loc
ation is often the norm.
These are the various components of the feature flag based on the above code:
Toggle Point: Each check of the feature flag is a toggle point.
For instance, `if (flag.IsTrialSubscriptionActive("user1"))`.
There may be multiple toggle points.
Toggle Router: The `IFeatureFlag.IsTrialSubscriptionActive(string)` method rep
resents a toggle router.
Toggle Context: This is the set of conditions that the router takes into account
while computing if the user is in a Trial Period,
or instance, account creation date and number of trial days.
Toggle Configuration:
Contains informations about enable/disable toggle points.
Usually, toggle configurations, are environment-specific.

Deployment versus Release

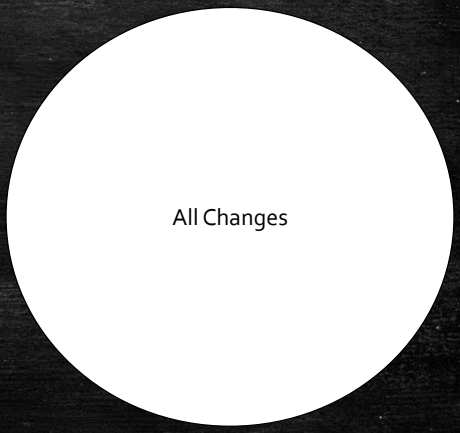
- Used interchangeably
- Different risk levels

Common misconception

Software teams often use "deployment", and "release" loosely, even interchangeably leading to a lot of confusion.
Throw in the term "ship" often used by business facing entities and that complicates the situation even further.

Deployment versus Release

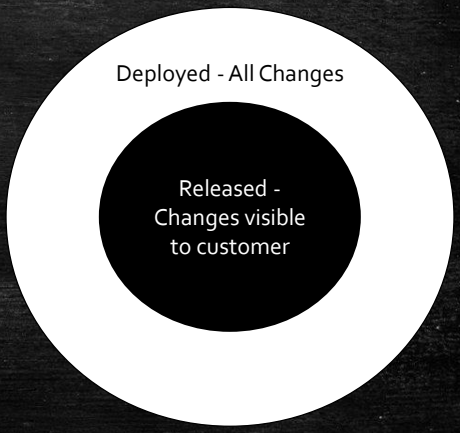
- Deployed
  - Installed on prod
  - Not exposed to customers
  - Minimal impact
  - Software team decision



When we use the term "Deployment"
we're talking about your team's process for installing the new version of your code o n production infrastructure.
When we say a new version of software is deployed,
we mean it is running somewhere in your production infrastructure.
Deployment does not mean customers have access to this new version.
Deployments are low risk because even if code does not work.
Deployments should have minimal impact on the customer.

Deployment versus Release

- Released
  - Receiving prod traffic
  - Exposed to customers
  - High impact

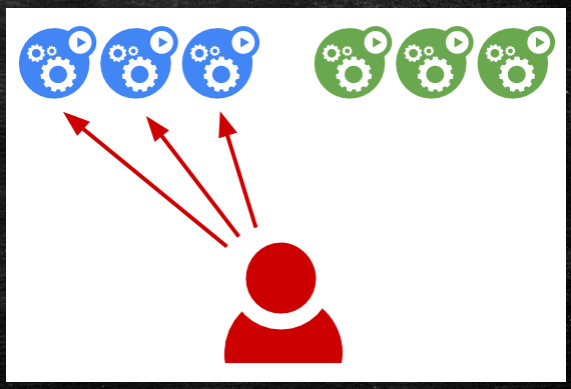


When we say a new feature is released, this means that production traffic if hitting th e code for this new feature.
Releasing is the process of pointing production workflow to point to the code related to the new feature.
When production traffic hits this new code, all the risks associated with releases – outages, crashes,performance issues, bugs - are exposed to customers.
The decision to release new features could involve various stakeholders including no n-software team members.

Depending on your team's practices, this could be considered a rollout of changes.
Released changes may need to be rolled back or rolled forward ie fixed ASAP.

Deployment versus Release

- Blue-Green Deploy
  - Blue == current
  - Green == new
  - High risk profile
  - Beyond scope



While this talk does not cover, blue-
green deployments this is an useful concept to discuss anyway since it involves separ
ating deployments from releases.

Blue-green deployments can be used instead of or with Feature Flags.
Green (new) version is deployed to fresh set of machines or containers
but since it's not yet exposed to production it's considered "Not released".

Blue (current) version is running on currently dedicated prod infrastructure

Release in a blue-green setup usually involves making changes at the load balancer
to redirect production traffic from the blue (current) machines to the green (new) ma
chines.

The operation equates to add hosts running the new version and remove hosts runni
ng the known-good version.

Blue-
green deployment involve infrastructure level changes. Once switched over new cha
nges have same high risk profile. These deployments are easy to rollback.

>**Note:** Since blue-green deployments often involve production infrastructure,

this might be an useful technique to use when infrastructure level changes occur. Feature Flags could still handle code level changes.

Feature Flag Frameworks and Services

- FeatureToggle
- FeatureManagement
- FeatureManagement with Azure App Configuration
- Launch Darkly

FeatureToggle

- Strongly-typed flags
- Several in-built options: simple, random, always on/off, date
- Easy to customize
- Supports: Code, Config files, Centralized (SQL Server or RavenDB)
- No in-built telemetry
- Project still active?

FeatureToggle by Jason Roberts is an open source library licensed under the Apache version two license.
To install FeatureToggle into your application, we can simply use NuGet
supports multiple. NET platforms.
doesn't use magic strings to represent toggles in code.
strings in the configuration files to actually configure the values, inside our code base but we're working with strongly typed toggles.
if the value of a toggle is missing from the configuration file, rather than defaulting to true or false, the application will throw an exception.
In this way, the application won't continue to run in an unknown state.
the FeatureToggle library aims to have a flexible provider model.
while some of the toggles have an idea of a toggle value provider.
the library aims to make this provider swappable so potentially a custom provider could be written for a toggle.
The FeatureToggle library supports a number of different configuration options.

- Code (needs redeploy of code)
- App or web config (edit file)
- Centralized (SQL or RavenDB) - .NET Framework only

FeatureManagement

Feature Flag Use Cases

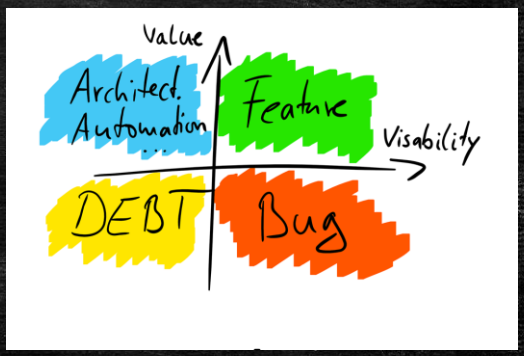
- Toggles/Kill switches
- A/B Testing
- Canary releases
- Test in production

Lifecycle & technological considerations

- Feature Flags are technical debt
- Do not daisy chain Feature Flags
- Measure everything
- Use frameworks
- Useful in CAB situations
- No long lived branches

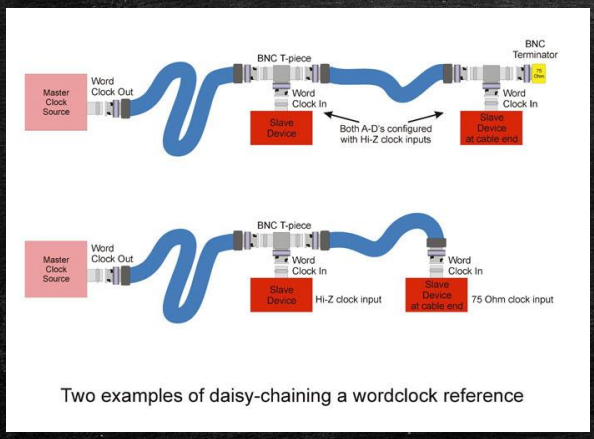
Feature Flags are technical debt

- Periodic review
- Remove older
- Remove after final release
- Permanent flags?



Feature flags is essentially new code you're adding which means they should be treat ed as technical debt.
You need to have a review process in place to periodically review all feature flags.
Older flags with deprecated or unused features should be removed ASAP.
If a feature is released to all users, the feature flag is no longer needed, ie, should be removed.
Permanent feature flags, for instance, ones used to handle subscription type access s hould be reviewed and maintained for as long as they're in use.

Do not daisy chain Feature Flags



Two examples of daisy-chaining a wordclock reference

- Leads to unpredictable behavior
- Small batches principle

In electrical and electronic engineering, a **daisy chain** is a wiring scheme
in which multiple devices are wired together in sequence or in a ring

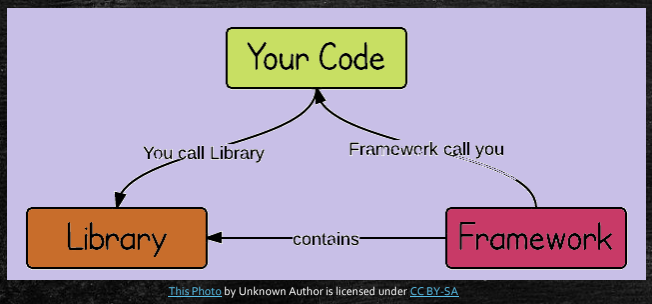
Combining multiple feature flags could lead to multiple features being toggled on or off simultaneously
which could lead to unpredictable system behavior
including increased system load, decrease in system performance and crashes.
Using DevOps principle of small batches,
limit the number of Feature Flags you're enabling or disabling at the same time.

Measure everything

- Is it working as expected?
- Metrics versus logs

Let's say you turned on a new feature - how do you know it's working as expected?
Is it throwing any errors?
How is the system performance?
Are people using the feature and if so, how many and what paths are they taking?
Metrics are numerical units and collected constantly,
logs are recorded events aka collected only when the event occurs.
Use metrics and logs to measure how the Feature is doing once the flag is turned ON
.
For instance, click throughs, page views, errors

Use frameworks



This Photo by Unknown Author is licensed under CC BY-SA

- Focus on solving business problems
- Well tested and maintained frameworks

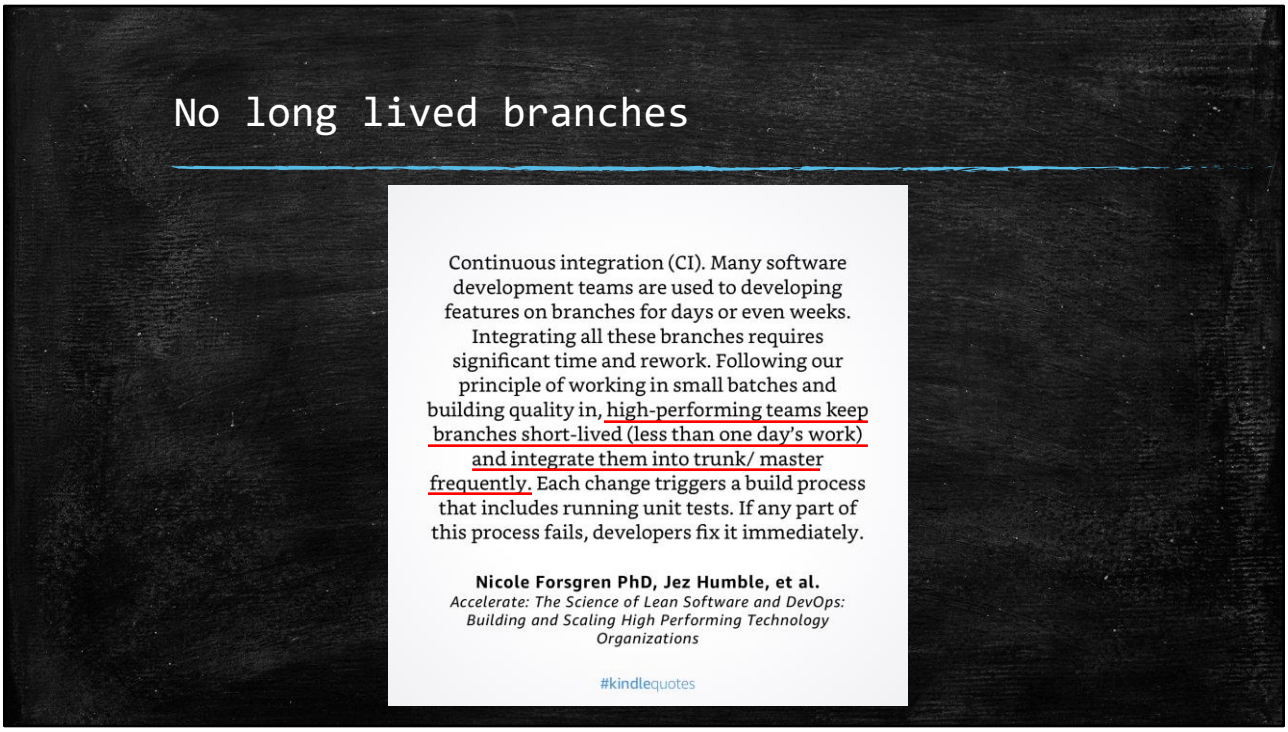
As we discussed earlier, feature flags are additional code you add to your system. This means additional maintenance.
The purpose of feature flags is to control exposure of features to the customers aka separating deployments from releases.
Since feature flags are largely used for achieve team/process/business goals, they're not central to solving your actual business problems.
Do not spend time writing your own implementation unless absolutely necessary. Use mature frameworks that we discussed in earlier sections.

Useful in CAB Situations

- Leverage feature flags to tackle bureaucracy
- Deploy code without invoking it
- Last step: add feature flag and turn on

A change-advisory board ("CAB") delivers support to a change-management team by advising on requested changes,
assisting in the assessment and prioritization of changes.
This body is generally made up of IT and Business representatives that include:
a change manager, user managers and groups, technical experts and, possible third parties and customers (if required).

Since the CAB members are largely concerned with changes impacting customer,
you could add code that does not hit any production paths (think: o code references i n C#)
over a period of time and continue pushing the code to deployment.
When time comes for a CAB review, you can simply add a Feature Flag and send it fo r review.
An additional advantage of this approach is that most,
CAB members are concerned about rollback in case the code is not working and feat ure flags can handle this situation with ease.



With feature flags, the release of software to users becomes a business decision inst ead of a technical decision.
The execution of a section of code or feature no longer needs a dedicated source cod e branch,
but becomes an execution branch instead, controlled by the feature flag.