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

In today's talk we'll focus on Feature Flags.
I will tell you about feature flags, what are they and what they look like and use cases
Then we'll talk about the difference between deployments and release
I will then touch on some useful feature flag frameworks and services that enable them
Then we get into the meat of the talk with discussion on Toggles, A/B Testing and Canary releases
I will then end with some tech considerations – pros, cons, best practices and gotchas while using feature flags

Santosh Hari

Sr Consultant @ Microsoft

Azure MVP 2016-2020

President, Orlando .NET UG

Organizer, Orlando Codecamp

Microsoft

santoshhari.wordpress.com

santosh.hari@microsoft.com

@_s_hari

/in/santoshhari

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

```
IFeatureFlag flag;
....
....
....
if (flag.IsTrialSubscriptionActive("user1"))
{
    Console.WriteLine($"Welcome {user1}");
}
....
....
....
if (flag.IsTrialSubscriptionActive("user1"))
{
    CheckTrialExpiryDateAndSendReminder("user1");
}
```

Toggle Point

Toggle Router
IFeatureFlag.IsTrialSubscriptionActive(string)

Toggle Context
Account creation date and num trial days

Toggle Configuration
Enable/disable toggle points

While it's not advisable to daisychain feature flag, the
same flag being used in multiple location 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

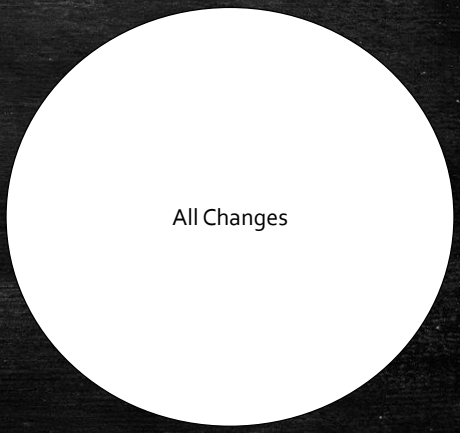
- Used interchangeably
- Different risk levels

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

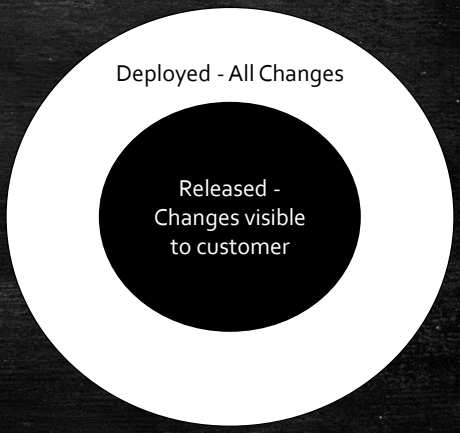
- Deployed
  - Installed on prod
  - Not exposed to customers
  - Minimal impact
  - Software team decision



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.
Deployments should be driven by software teams

Deployment versus Release

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

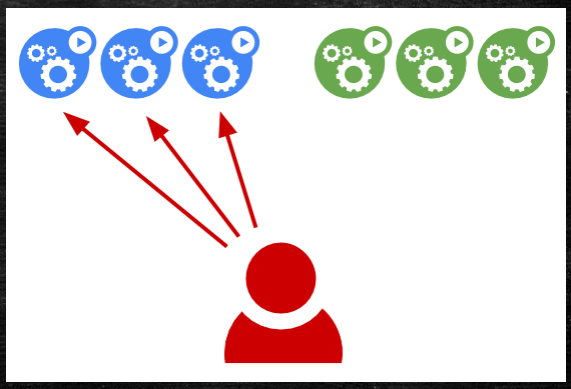


When we say a new feature is released, this means that production traffic if hitting th e code for this new feature.
Releasing is the process of pointing production workflow to point to the code related to the new feature.
When production traffic hits this new code, all the risks associated with releases – 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
  - Green 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 changes 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.

Deployment versus Release

- Deployments
  - On going
  - Low risk
- Releases
  - Business decision
  - High risk

Takeaway:
Deployments should be continuous and low impact to customer
Release should be a mindful business decision and should be high impact to the customer (hopefully positive)

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.
Uses configuration files to actually configure the actual toggle
But overall it gives you strongly typed toggles in your C# code
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.
So the application won't continue to run in an unknown state.

the FeatureToggle library aims to have a flexible provider model.
while some toggles are pre-defined
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

No in-built telemetry
Project still active?

FeatureManagement

- Build on .NET Core IConfiguration
- Toggle with config files
- API extensions ASP.NET Core and MVC
- Name + Feature Filter

FeatureManagement provide comprehensive feature flag support in .NET Core
These libraries are built on top of the .NET Core configuration system.
- Supports local JSON file feature flag setup
- Configuration values can change in real-
time, feature flags can be consistent across the entire request
- Toggle on/off features through declarative configuration file
- Dynamically evaluate state of feature based on call to server
-
API extensions for ASP.NET Core and MVC framework: Routing, Filters and Action At
tributes

Feature Flags consists of a name and a feature filter.
Feature Filters are scenarios for enabling/disabling flags.
For instance, a filter that enables the flag for only 10% of the requests.

FeatureManagement with Azure App Config

- PaaS
- Manage app settings and feature flags
- Feature Flags auto update
- Easy to setup and manage flags
- Security (MSI, encryption – REST+transit)
- Key Vault integration (app settings)
- IaC friendly
- Multiple SDKs

Azure App Configuration provides a PaaS to centrally manage application settings and feature flags.
You can create an end-to-end implementation of feature management in an ASP.NET Core application
using Azure App Configuration using the App Configuration service to centrally store all your feature flags and control their states.

feature flags are automatically updated without redeploying your app
the .NET Core SDK automatically pulls feature flags every 30 seconds by default)

A fully managed service that can be set up in minutes
Flexible key representations and mappings
Tagging with labels
Dedicated UI for feature flag management
Comparison of two sets of configurations on custom-defined dimensions
Enhanced security through Azure-managed identities
Encryption of sensitive information at rest and in transit
IaC friendly with ARM templates and Terraform like any other azure service
Native integration with popular frameworks

LaunchDarkly

- SaaS
- Flag and Toggle Management
- Fast
- Multiple environments
- Easy to manage
- Multiple SDKs

LaunchDarkly is a SaaS Feature Flag and Toggle Management system
serves over 200 billion feature flags/day.

It uses a streaming architecture to service feature flags in microseconds without mak ing remote requests.
All flags are served locally and backed up using a globally distributed CDN provider which makes them fast
You can easily configure and manage TEST/PROD style feature flag deployments and even sync across environemtns

It is also pretty easy to get started with, with some very nice tutorials that get you up and going in minutes.
The toggling is easy for a non-tech user to manage

Has multiple SDKs for various frameworks and languages

Which one should I use?

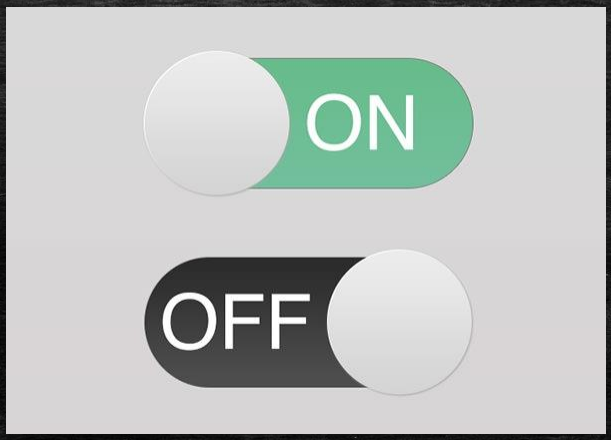
- It Depends
- Greenfield .NET Core = FeatureManagement + Azure App Configuration
- Existing app
  - LaunchDarkly = if non-technical users are flipping toggles
  - .NET Core = FeatureManagement or FeatureToggle
  - .NET Framework = FeatureToggle
- Extreme customization = FeatureToggle

\

Feature Flag Use Cases

- Toggles/Kill switches - demo
- A/B Testing – demo
- Canary releases – demo

Toggle/Kill Switch

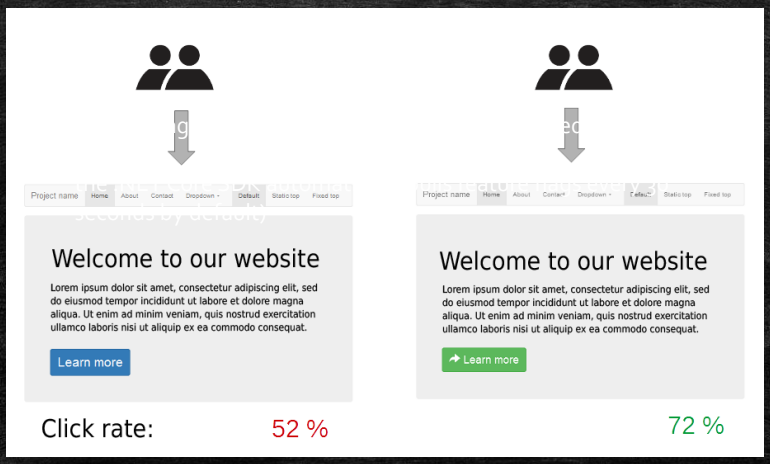


In simplest terms, a toggle is a ON/OFF button.
You can basically turn a feature on and off with a toggle by redirecting code flow in production.
A well-wrapped feature means you can quickly turn it off if it's performing poorly.
This can be the difference between a public relations disaster and minimal impact.
When you combine this with percentage rollouts, it's even more powerful.
If you're doing this right, you can essentially take risk off the table.

Toggle/Kill Switch DEMO

- Feature Toggle

A/B Testing

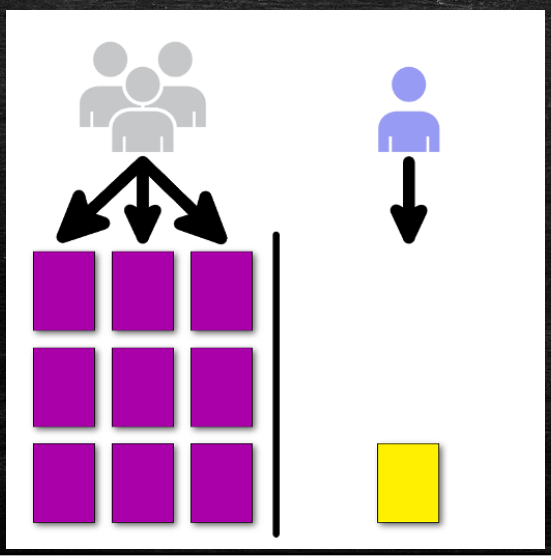


A/B testing (also known as split testing) is a process of showing two variants of the feature
to different segments of customers at the same time and comparing which variant drives more conversions.
the variant that gives higher conversions is the winning one

A/B Testing DEMO

- FeatureManagement
- Azure App Configuration

Canary Release



Story of a canary in a coalmine – when it stopped singing get the heck out
Similar to blue green but scope and lifetime of the newly deployed instance is vastly different than blue green
Also does not involve the use of load balancer

When you are happy with the new version, you can start routing a few selected users to it.
Strategies – select users randomly, beta users,
Microsoft has the concept of ring releases for Windows and Azure DevOps for instance

As new version proves itself, rollout to more users
Migrate phase could last for weeks
Phased rollout helps smooth out new version – bugs, performance, functionality, etc

Easy to rollback

Downside? Manage multiple versions

Canary Releases DEMO

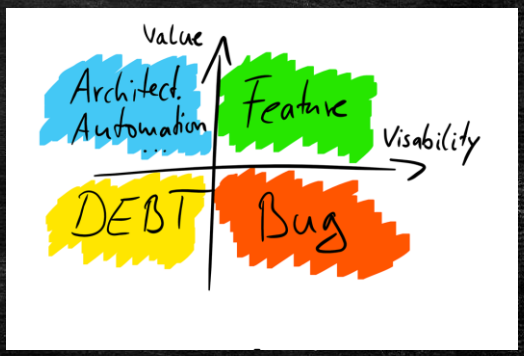
- LaunchDarkly

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

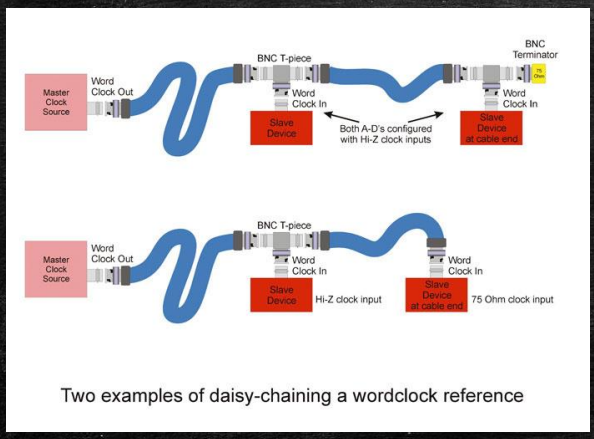
Feature Flags are technical debt

- 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.

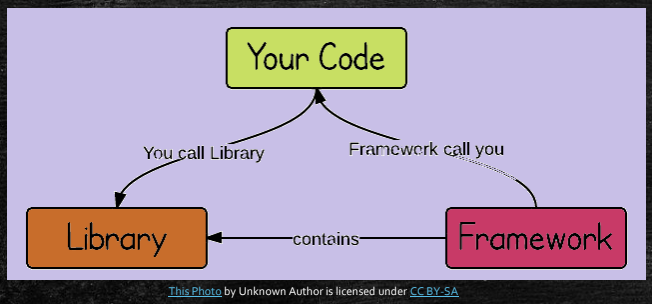
Microsoft has the story in 2013 when they did a TFS release and flipped all feature flags on at once

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 to 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.

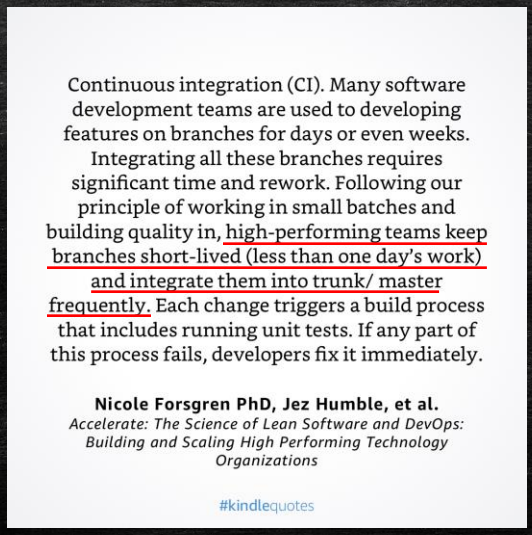
Useful in CAB Situations

- 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.

No long lived branches



This is a quote from Accelerate
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.

As promised, we talked about
In today's talk we'll focus on Feature Flags.
I will tell you about feature flags, what are they and what they look like and use cases
Then we'll talk about the difference between deployments and release
I will then touch on some useful feature flag frameworks and services that enable them
Then we get into the meat of the talk with discussion on Toggles, A/B Testing and Canary releases
I will then end with some tech considerations – pros, cons, best practices and gotchas while using feature flags

Call to action

- Contact me with the following info
  - Which framework will you use?
  - Which service will you use?
- Please use Feature Flags!

Santosh Hari

Sr Consultant @ Microsoft

Azure MVP 2016-2020

President, Orlando .NET UG

Organizer, Orlando Codecamp

Microsoft

santoshhari.wordpress.com

santosh.hari@microsoft.com

@_s_hari

/in/santoshhari

Resources

- Deploy != Release (Part 1) From <https://blog.turbinelabs.io/deploy-not-equal-release-part-one-4724bc1e726b>
- Choosing a Feature Flag Platform From <https://newsignature.com/articles/choosing-a-feature-flag-platform/>
- Anatomy of a Feature Flag From <https://rollout.io/blog/ultimate-feature-flag-guide/>
- DevOps tech: Trunk-based development From <https://cloud.google.com/solutions/devops/devops-tech-trunk-based-development>
- Canary release From <https://martinfowler.com/bliki/CanaryRelease.html>
- Quickstart: Add feature flags to an ASP.NET Core app From <https://docs.microsoft.com/en-us/azure/azure-app-configuration/quickstart-feature-flag-aspnet-core?tabs=core3x>
- FeatureToggle Documentation From <http://jason-roberts.github.io/FeatureToggle.Docs/pages/usage.html>
- Feature Toggles (aka Feature Flags) From <https://martinfowler.com/articles/feature-toggles.html>
- Introducing Microsoft.FeatureManagement From <https://andrewlock.net/introducing-the-microsoft-featuremanagement-library-adding-feature-flags-to-an-asp-net-core-app-part-1/>
- What is Azure App Configuration? From <https://docs.microsoft.com/en-us/azure/azure-app-configuration/overview>
- LaunchDarkly SDK in your .NET application From <https://docs.launchdarkly.com/sdk/server-side/dotnet>