From Transient to Singleton: Secrets of .NET Dependency Injection

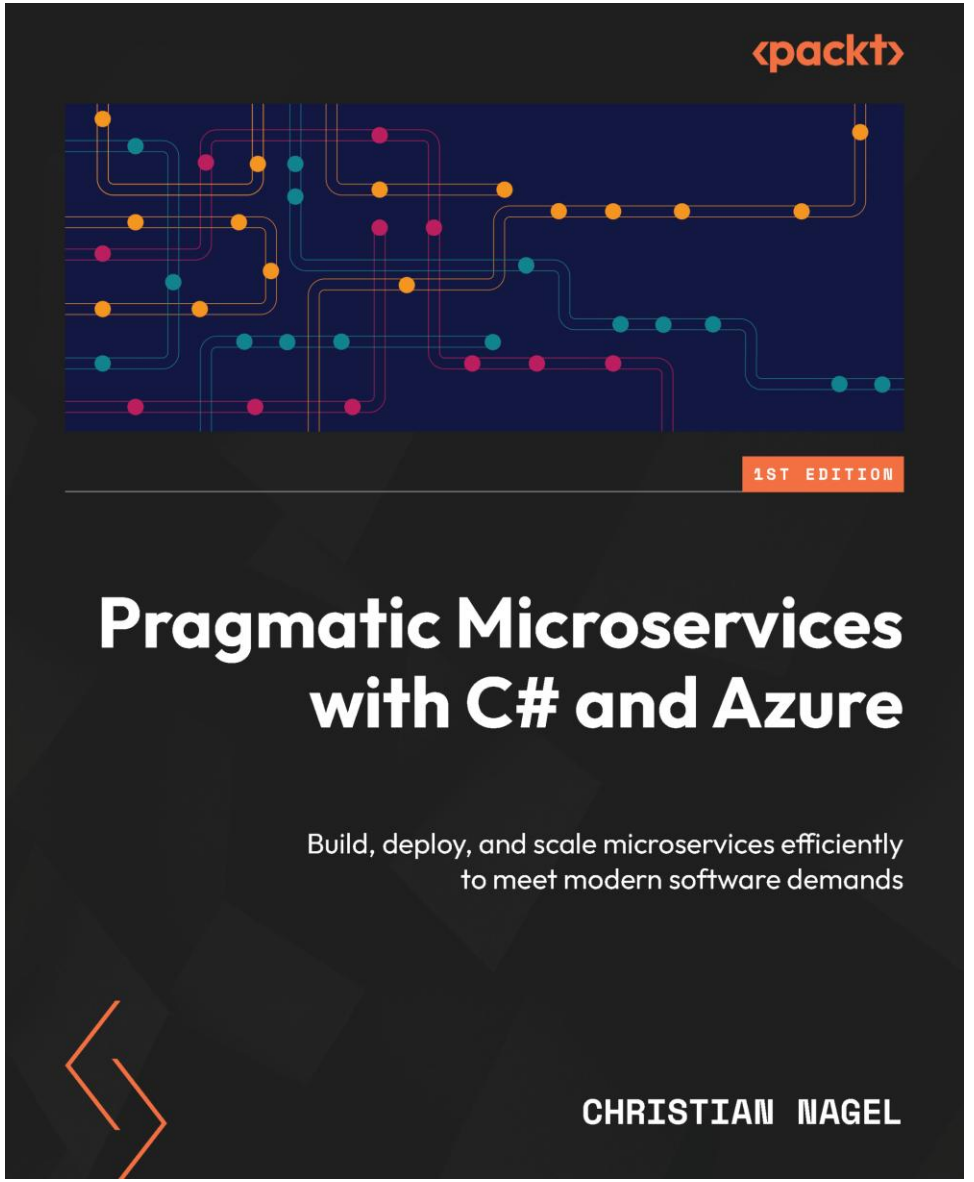
Christian Nagel

https://www.cninnovation.com

Christian Nagel

- Training
- Coaching
- Consulting
- Development
- New book: **Pragmatic Microservices**

- Microsoft MVP
- www.cninnovation.com
- csharp.christiannagel.com
- @christiannagel



Agenda

- DI Foundations
- App Builder Pattern
- Lifetime of services
- More…

Many code samples!

Not really “secrets”, but often “not known”

Foundations

Why Dependency Injection?

- Decoupling
- Unit tests
- Platform independence
- Simplify implementations
- Versioning

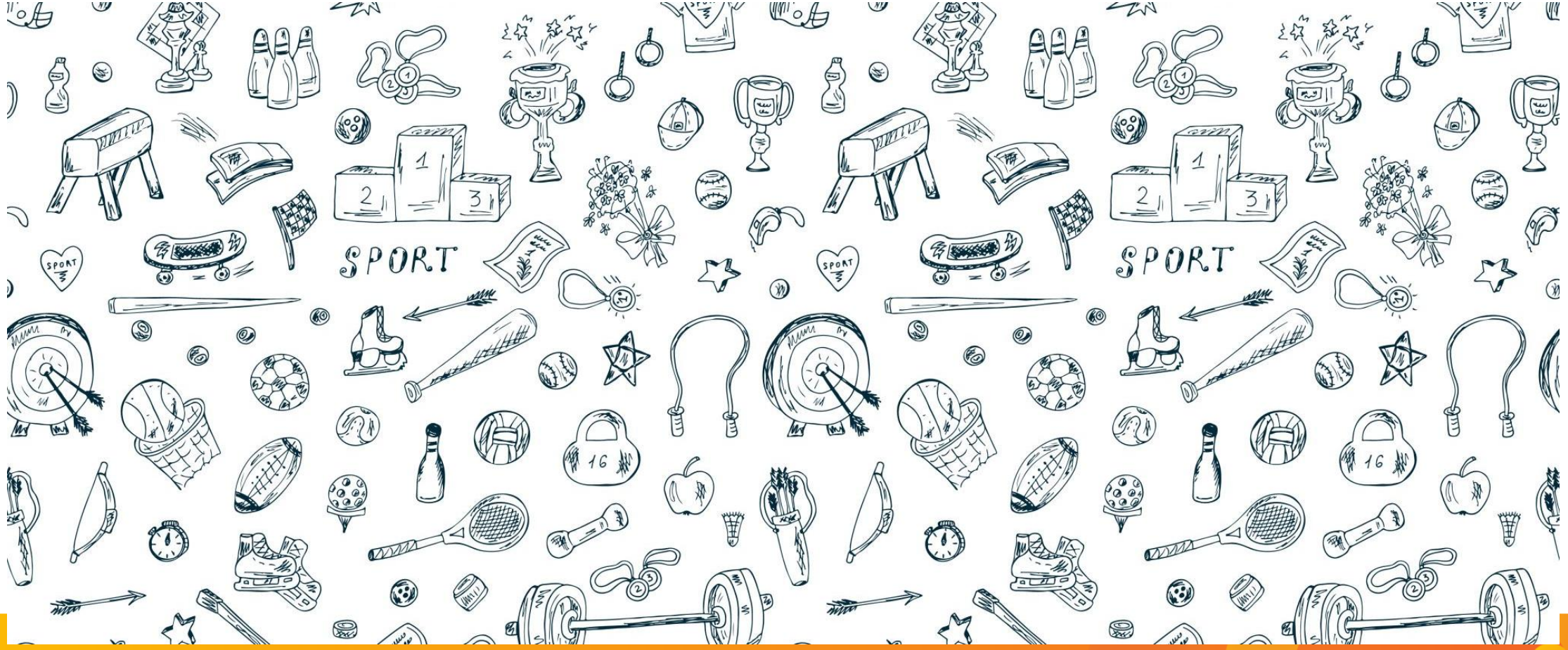
Dependency Injection Container

- Microsoft.Extensions.DependencyInjection
- Simplify DI
- High performance
- Used by many .NET frameworks
- Integration with third party containers

Host class

- Originates from ASP.NET Core WebHost
- Functionality needed by all applications
- DI Container
- Logging
- Configuration

App Builder Pattern



(Web) App Builder Pattern

- Simpler APIs without passing delegates
- Web with middleware

Hosted Service

- IHostedService
- Run

- Windows Service / System Daemon
- AddWindowsService
- AddSystemd

App Builders

- ASP.NET Core
  - WebApplication.CreateBuilder
- Blazor WASM
  - WebAssemblyHostBuilder.CreateDefault
- .NET MAUI
  - MauiApp.CreateBuilder

Lifetime

Lifetime of Services

- Transient
  - A new instance for every injection
- Scoped
  - Injecting in the same scope?
- Singleton
  - Only one instance created

Lifetime rules for injecting services

- Inject services with equal or longer lifespans!

- Inject a singleton into a transient service

- ~~Inject a transient into a singleton service~~

Rules for IDisposable/IAsyncDisposable Services

Service are automatically disposed at…

- **Scoped** services
- **Singleton** services
- **Transient** services

When

- ➢ End of **scope**
- ➢ End of **root scope**
- ➢ End of **scope**

Avoid registering *IDisposable/IAsyncDisposable* services as **transient**!

What’s a scope?

- ASP.NET Core Web Application
- Blazor Server / SignalR
- Blazor WASM
- WPF/WinUI/…

➢Created per request

➢Created per connection

➢Created per browser session

➢Custom

More...

Multiple registrations - who wins?

- The last one wins!
- Core functionality can easily be updated
  (ASP.NET Core, EF Core…)

Collections

- Register multiple implementations for the same contract
- Inject IEnumerable<T>
- Use multiple services with the same contract
- Decide based on additional information

Keyed services

- AddKeyed[Transient|Scoped|Singleton]Service
  - Specify a key name
- Inject with [FromKeyedService("Key")]
- Example: Meter

Open Generics

- Register open generic services
  - `builder.Services.AddSingleton(typeof(IMath<>), typeof(Math<>));`
- Inject as needed
  - `internal class DoubleCalculator(IMath<double> math)`
- Used with .NET: IOptions<T>, ILogger<T>

Summary

- DI Container
- App Builder
- Logging, Configuration, Middleware
- Can be used in every application type

Thank you for joining!

Questions?

- https://github.com/cnilearn/bastaspring2025
- https://csharp.christiannagel.com
- https://www.cninnovation.com