C# 13: Neue Funktionen und Ausblick auf C# 14

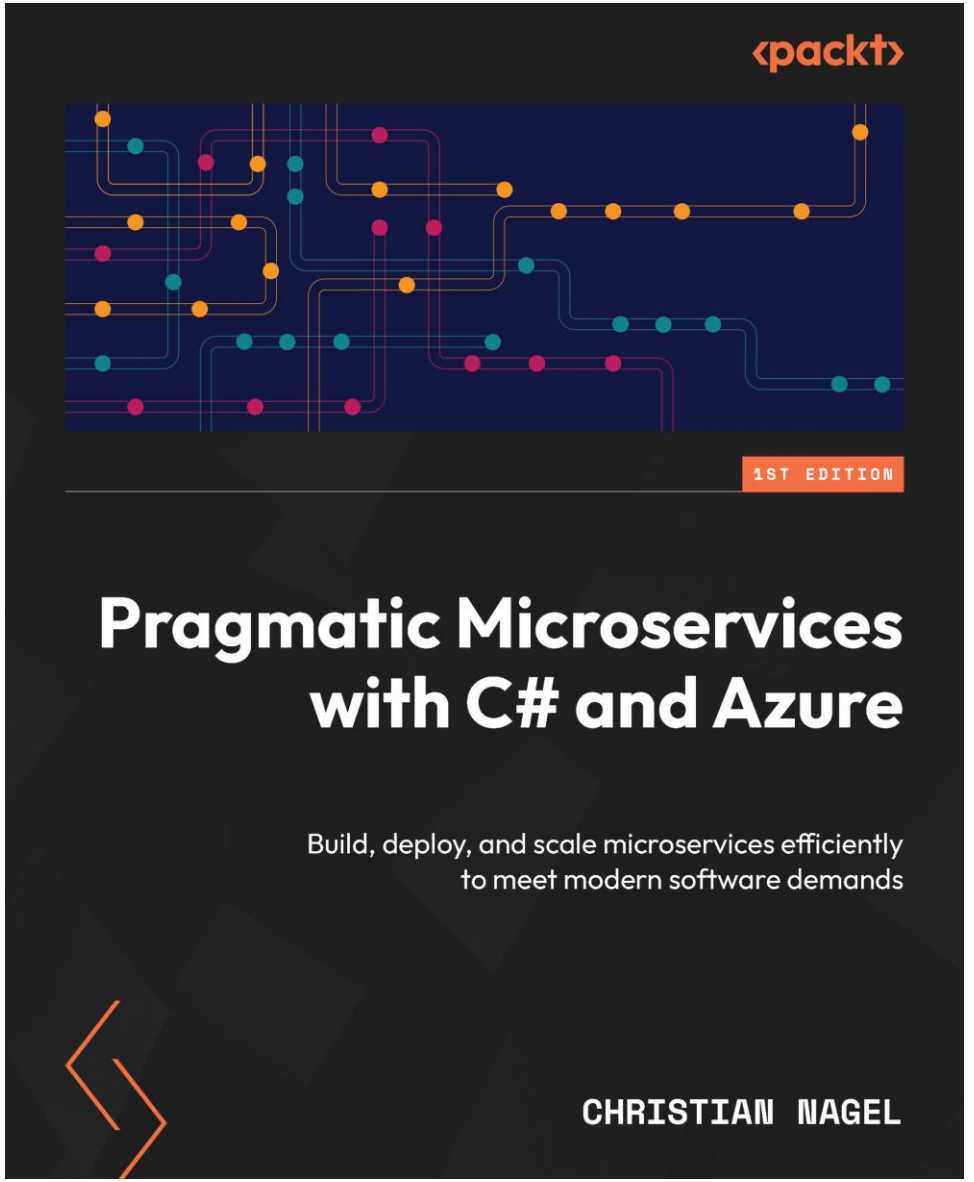
Christian Nagel

https://www.cninnovation.com

Christian Nagel

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

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



C# Updates!

C# Enhancements

Type member enhancements

Span

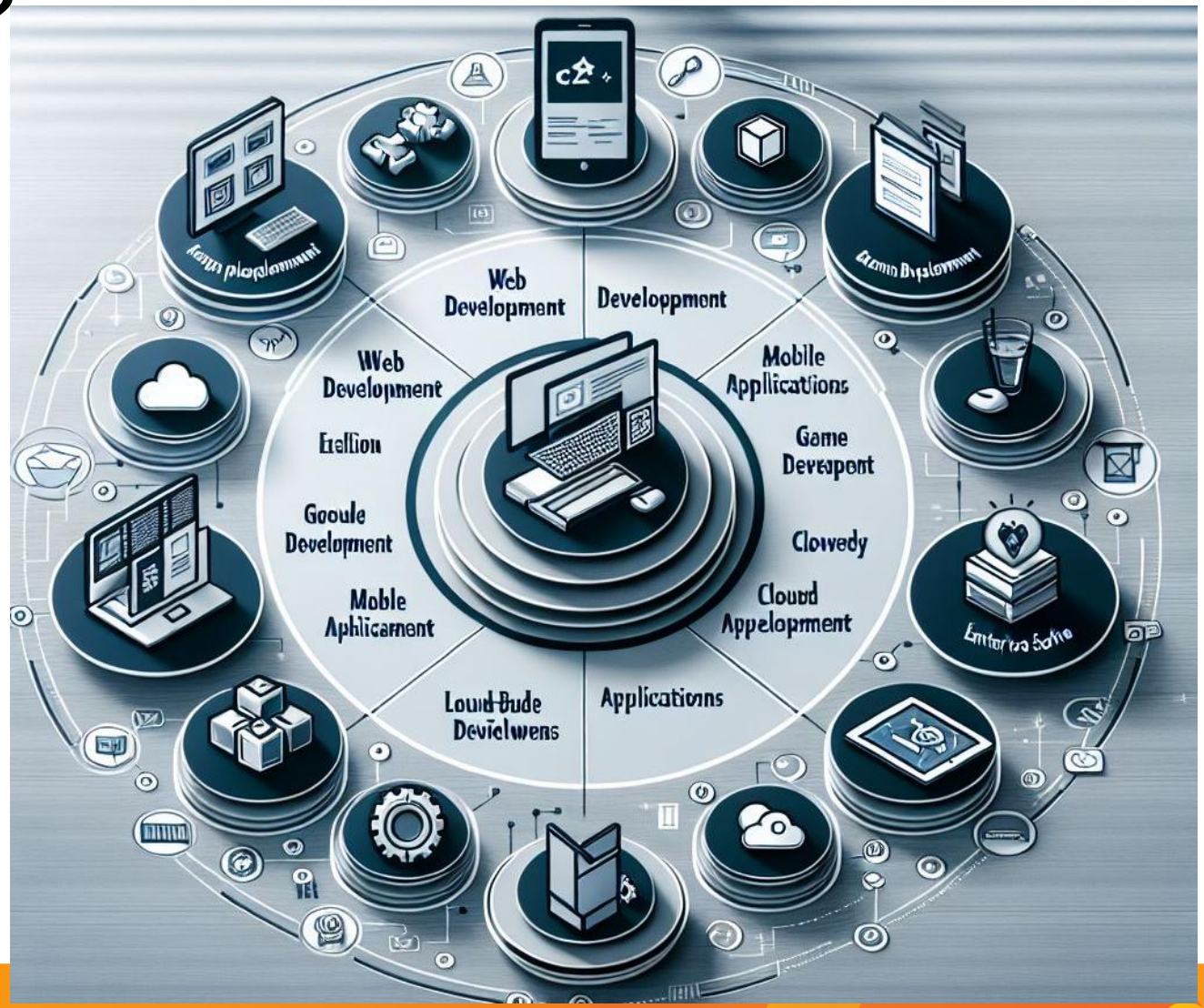
Async and native

What’s new with C# 13

What’s coming with C# 14

Where is C# used?

- Windows Desktop
- Linux, Mac
- Web
- Mobile
- Cloud
- IoT
- Games
- AI
- Native
- ...



Warmup

Simple, and practical...

Escape (C# 13)

Make escape codes easier

\e instead of \u001b

VT100 escape characters

Implicit Index access (C# 13)

- In object initializers

```
TimerRemaining countdown = new()
{
  Buffer =
  {
    [^1] = 0,
    [^2] = 1,
    [^3] = 2,
  }
};
```

Simple lambda parameters with modifiers (C# 14)

- Instead of:

```
TryParse<int> tryParseInt =
  (string text, out int result) ⇒ int.TryParse(text, out result);
```

- Do this:

```
TryParse<int> tryParseInt =
  (text, out result) ⇒ int.TryParse(text, out result);
```

Nameof with unconstrained generics (C# 14)

```
string name = nameof(List<>);
Console.WriteLine(name); // List
```

String literals in data sections (C# 14)

```
string s1 = "the quick brown fox jumped";
ReadOnlySpan<byte> s2 = "the quick brown fox jumped"u8;
```

- string literals in UserString heap (default)
- Limited to 2^24 bytes
- Enable with feature flag

```
<PropertyGroup>
  <Features>$(Features);
    experimental-data-section-string-literals=20
  </Features>
</PropertyGroup>
```

Method group natural type improvements (C# 13)

- C# 10 added weak natural types
- C# 13 – type inference is more robust
- Find matching candidate methods scope by scope
  - Instance methods first
  - Extension methods later
- Prune candidates that don’t succeed
  - Prune generic methods when no type arguments are provided
  - Prune generic extension methods on constraints

Type Members

Semi-Auto Properties (C# 13 + preview)

- field keyword

```
// full property
private int _x;
public int X
{
  get ⇒ _x;
  set ⇒ _x = value;
}
```

```
// auto property
public int X { get; set; }
```

```
// semi-auto property
public int X
{
  get ⇒ field;
  set ⇒ field = value;
}
```

```
public MyType X ⇒ field ??= ComputeValue();
```

Partial properties and indexers (C# 13)

```
partial class Book
{
  public partial string Title { get; set; }
}
```

```
partial class Book
{
  private string _title;
  public partial string Title
  {
    get ⇒ _title;
    set ⇒ _title = value;
  }
}
```

Partial events and constructors (C# 14)

- Extensions are “transparent wrappers”
- Augmented with additional members and interfaces

```
partial class Publisher
{
  partial event Action<int, string> MyEvent
  {
    add { }
    remove { }
  }
}
```

Ref Struct & Span Enhancements

Ref struct enhancements (C# 13, .NET 9)

- What is a ref struct?
- Compare struct .vs. class .vs. ref struct
- Before C# 13 – ref struct can’t implement interfaces
- C# 13
  - ref struct implement interfaces
  - Generic anti constraint: ***allows ref struct***

Params collections (C# 13)

- Params modifier not limited to arrays

```
void Foo(params IEnumerable<T> items)
{ }
```

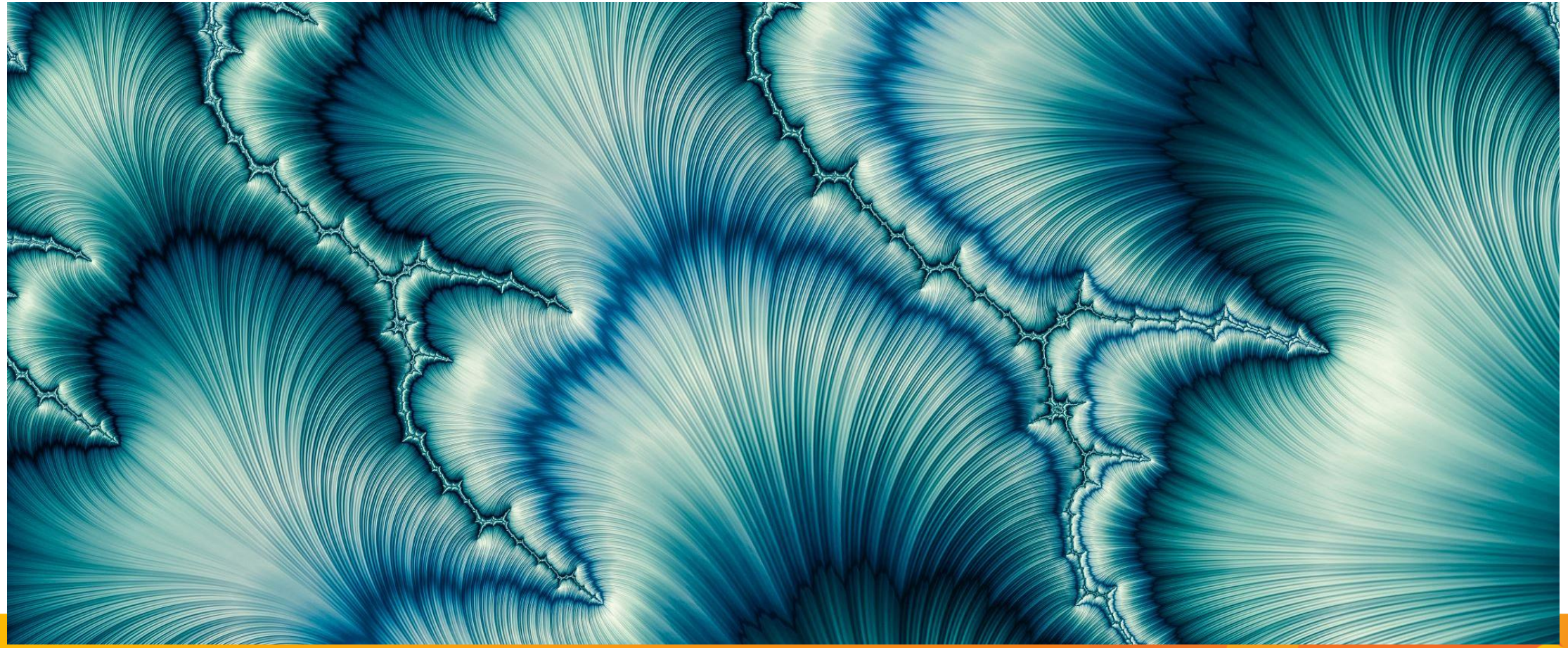
```
void Foo(params Span<T> items)
{ }
```

```
void Foo(params MyCollection items)
{ }
```

First class Span<T> (C# 14)

- T[] → Span<T>
- T1[] → ReadOnlySpan<T2>
  - Covariance-compatible
- Span<T1> → ReadOnlySpan<T2>
- string → ReadOnlySpan<char>

Async & Native



Lock Object

- .NET 9 includes *System.Threading.Lock* type
- First-class lock-type
- Simpler and faster

- The ***lock*** keyword is enhanced to not only support *Monitor*, but also *Lock*

Runtime async (probably after .NET 10)

- Compiler rewrite
- Methods can “yield” control back to their caller
- Specific suspension points
- Improvements in performance

Interceptors

- Replace implementation
- Usually used by source generators
- Pre-release with .NET 8
- Released with .NET 9 (with changes)
- Used from source generators
- *InterceptsLocation* Attribute
- .NET 9: Roslyn *GetInterceptableLocation*

Native AOT

- Compile .NET to native code
- Self-contained
- Quick startup, less memory usage
- Can run where JIT is not allowed
- Compilation to a single file

Native AOT Restrictions

- No dynamic loading
- No reflection emit
- No C++/CLI
- Trimming required
- Many libraries don’t support native AOT (yet)

Native AOT .NET 9 Updates

- Improved performance
- Trimming enhancements - smaller application size
- Enhanced platform support
- Microsoft.AspNetCore.OpenAPI
- SignalR, gRPC support, .NET MAUI
- WinUI in progress to fully support native AOT
- EF Core in progress

Native AOT For Action

- Make libraries AOT compatible
  - if possible
  - *IsAotCompatible* adds checks
- Create native AOT services
  - if useful and possible

Extensions (C# 14?)

- Methods, properties, indexers, operators

```
public static class Enumerable
{
    // New extension declaration
    extension(IEnumerable source) { ... }

    // Classic extension method
    public static IEnumerable<TResult> Cast<TResult>(this IEnumerable source) { ... }

    // Non-extension member
    public static IEnumerable<int> Range(int start, int count) { ... }
}
```

Summary

Productivity

- Natural type enhancements
- More partial members – source generator support!
- Span enhancements

Performance

- Span enhancements
- Native AOT

Thank you for joining!

Questions?

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