Be ready for C# 12 and C# 13

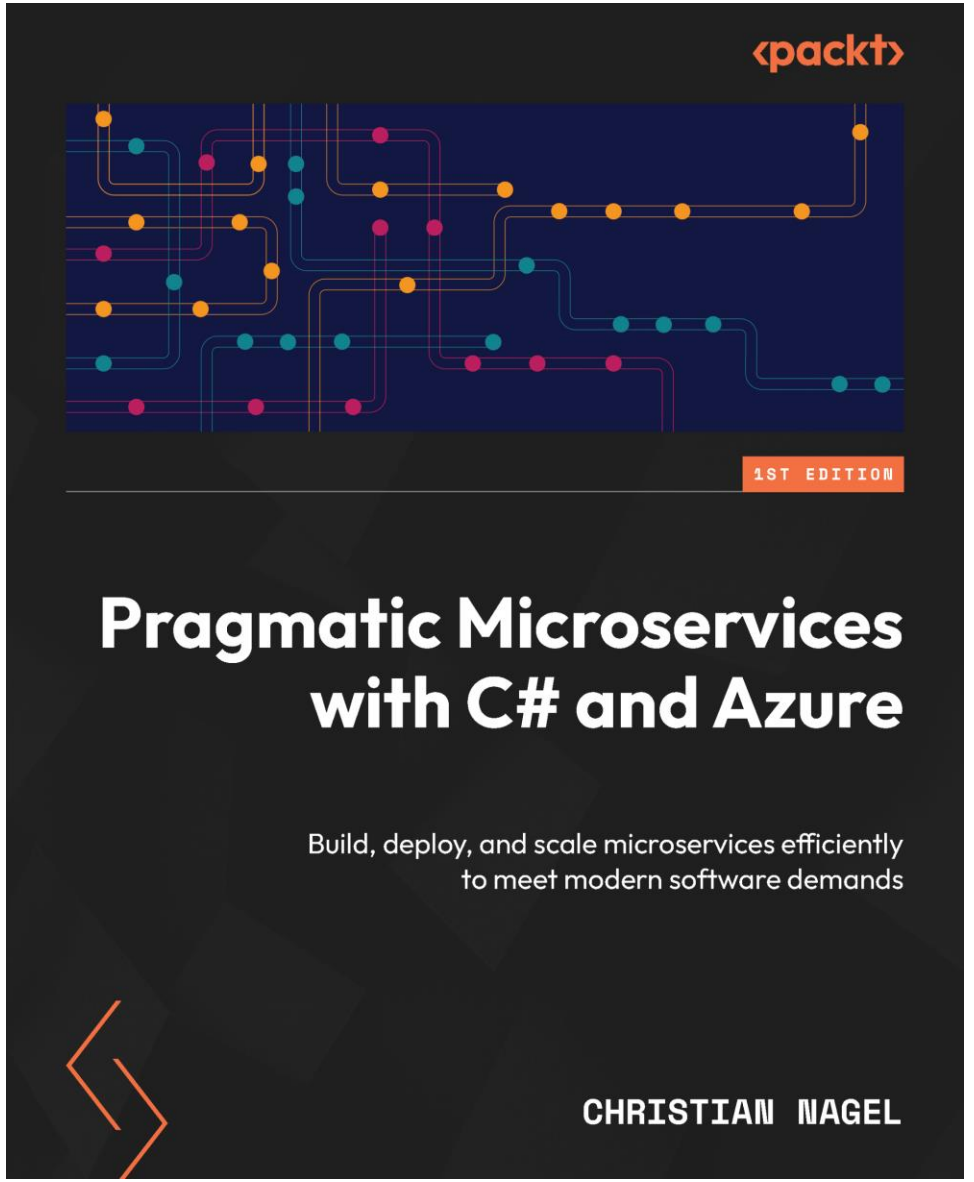
Christian Nagel

https://www.cninnovation.com

Christian Nagel

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

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



What's new with...

- Types
- Arrays and Collections
- Lambda Expressions
- Something special...

Escape

Make escape codes easier

\e instead of \u001b

VT100 escape characters


Writing code today...
Sample: minimal APIs with scaffolding...

Types and members Enhancements

Alias any type

- using alias relaxed with C# 12
- alias tuples, pointers, arrays, generic types…

Primary Constructors

- Class records
  - get & init accessors
- Struct records
  - get & set accessors
- Readonly struct records
  - get & init accessors
- Classes and structs
  - Parameters

Parameter ref readonly

- ref
  - Needs initialization before calling the method
- out
  - Initialization not required
  - Method must assign a value
- ref readonly
  - Must be initialized
  - Method cannot assign a new value
- in
  - Must be initialized
  - Method cannot assign a new value
  - Compiler can use a temporary variable within the method

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: *allow ref struct*


Arrays and Collections

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)
{ }
```

Inline Arrays

- Optimized creation for fixed sizes
- Directly assign *Span<T>*
- *InlineArray* attribute
- Performance optimization

```
[InlineArray(10)]
public struct Buffer
{
  private int _x;
}
```

Method	Mean	Error	StdDev	Gen0	Allocated
UseNormalArray	3.9942 ns	0.0401 ns	0.0375 ns	0.0032	40 B
UseBuffer	0.1927 ns	0.0038 ns	0.0034 ns	-	-

Collection Expressions (Collection Literals)

- Conversion to many different collection types using square brackets *[]*

```
int[] arr = [1, 2];
List<int> list1 = [3, 4];
IEnumerable<int> list2 = [5, 6];
```

Spread Operator

- Expand elements without manual iteration
- Can be used together with the range operator

```
int[] arr = [1, 2];
List<int> list1 = [3, 4];
IEnumerable<int> list2 = [.. arr, .. list1];
```

CollectionBuilder Attribute

- Allow collection expressions with custom collection types

```
[CollectionBuilder(typeof(MyCustomCollection),
  nameof(MyCustomCollection.Create))]
internal class MyCustomCollection<T> : Collection<T>
{
}
```

```
internal static class MyCustomCollection
{
  public static MyCustomCollection<T> Create<T>(ReadOnlySpan<T> items)
  {
    MyCustomCollection<T> collection = new();
    foreach (T item in items)
    {
      collection.Add(item);
    }
    return collection;
  }
}
```


Lambda Expressions

Natural delegate type (C# 10)

- Natural type of lambda expression
- Doesn’t require to declare a delegate type (e.g. Func<>)

Default lambda parameters (C# 12)

- Default values for parameters on lambda expressions
- Convenient with minimal APIs


Something special...

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*

Unsafe Accessor

- With reflection it is possible to access private members of a type
- *UnsafeAccessor* doesn't need reflection!
- Serialization, EF Core…
- Compiler-Feature
- Access private members

```
internal class ChangeIt
{
  [UnsafeAccessor(UnsafeAccessorKind.Field,
    Name = "_title")]
  public extern static ref string GetTitle(Book @this);
}
```

Interceptors

- Replace implementation
- Usually used by source generators
- Pre-release with .NET 8
- Release 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 For Action

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

C# next

- First-class Span type
- Field keyword in properties
- Default in deconstruction
- Roles/extensions

Summary

Productivity

- Primary constructors
- Collection expressions
- Escape sequence

Performance

- Span enhancements
- Inline Array
- Native AOT
- Source generators

Thank you for joining!

Questions?

- https://github.com/cnilearn/bastamainz2024
- https://blogs.cninnovation.com
- https://www.cninnovation.com