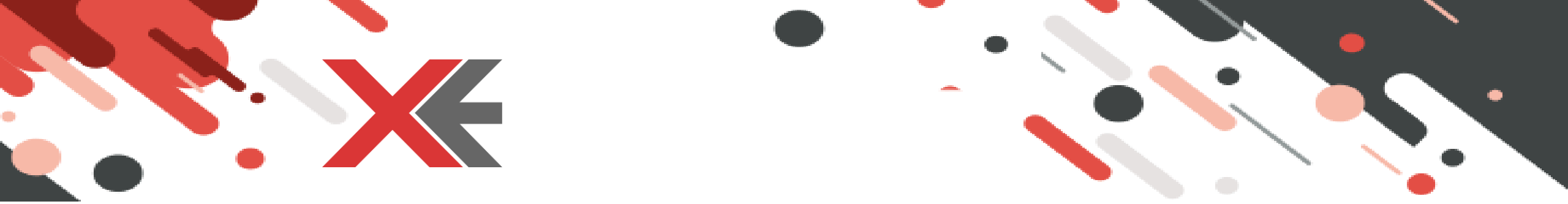


.NET Conf 2024 XE

Novità in ASP.NET Core

Andrea Dottor

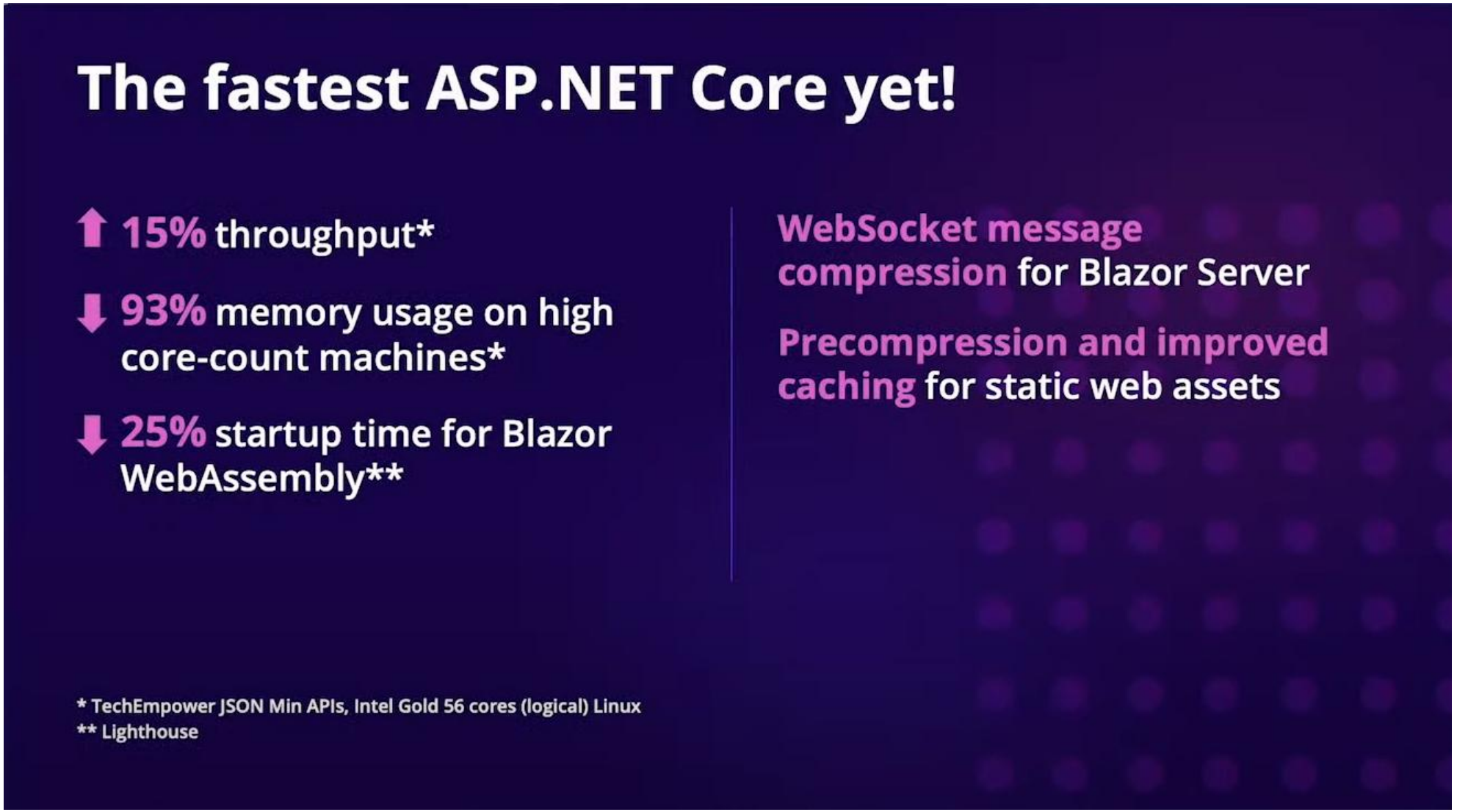
ASP.NET Core in .NET 9

Quality & fundamentals

Developer experience

Cloud native

https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-9.0


The fastest ASP.NET Core yet!
15% throughput*
93% memory usage on high core-count machines*
25% startup time for Blazor WebAssembly**
WebSocket message compression for Blazor Server
Precompression and improved caching for static web assets
* TechEmpower JSON Min APIs, Intel Gold 56 cores (logical) Linux
** Lighthouse

SignalR

La Notte Stellata (De sterrennacht), Vincent Van Gogh

Polymorphic Type Support in SignalR Hubs

Hub methods can now accept a base class instead of the derived class to enable polymorphic scenarios.

This feature allows for more flexible and reusable code in SignalR hubs.

Polymorphic Type Support in SignalR Hubs

```
[JsonPolymorphic]
[JsonDerivedType(typeof(PersonExtended), nameof(PersonExtended))]
[JsonDerivedType(typeof(PersonExtended2), nameof(PersonExtended2))]
public class Person
{
    public string Name   { get; set; }
    public Person Child  { get; set; }
    public Person Parent { get; set; }
}

public class PersonExtended : Person
{
    public int Age { get; set; }
}

public class PersonExtended2 : Person
{
    public string Location { get; set; }
}
```

Polymorphic Type Support in SignalR Hubs

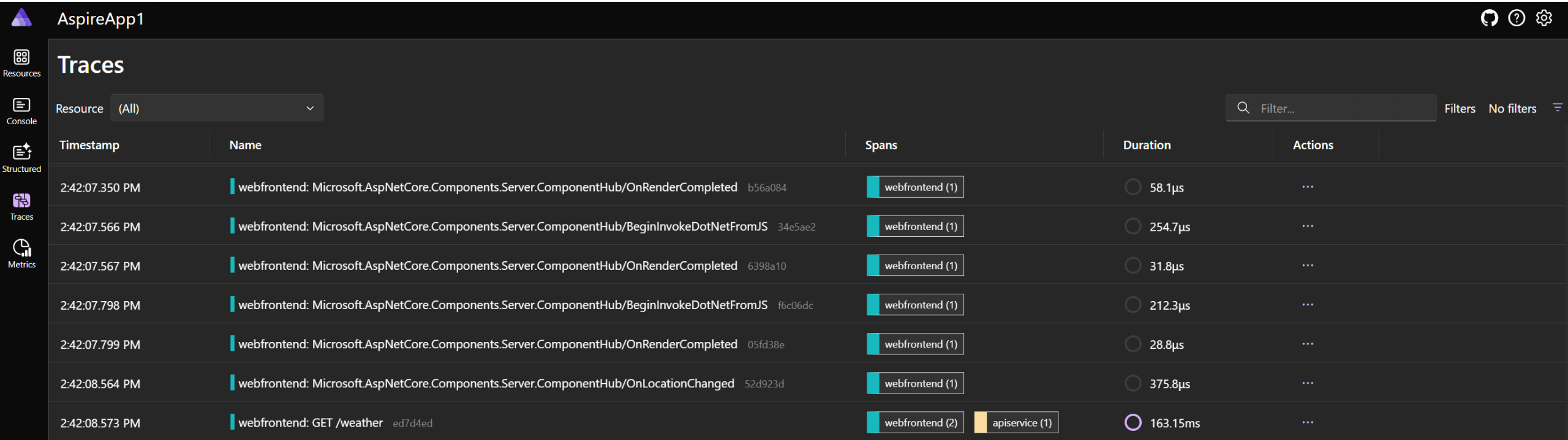
```
public class MyHub : Hub
{
    public void Method(Person person)
    {
        if (person is PersonExtended)
        {
        }
        else if (person is PersonExtended2)
        {
        }
        else
        {
        }
    }
}
```

Improved SignalR Activities

SignalR now has an ActivitySource named *Microsoft.AspNetCore.SignalR.Server* that emits events for hub method calls.

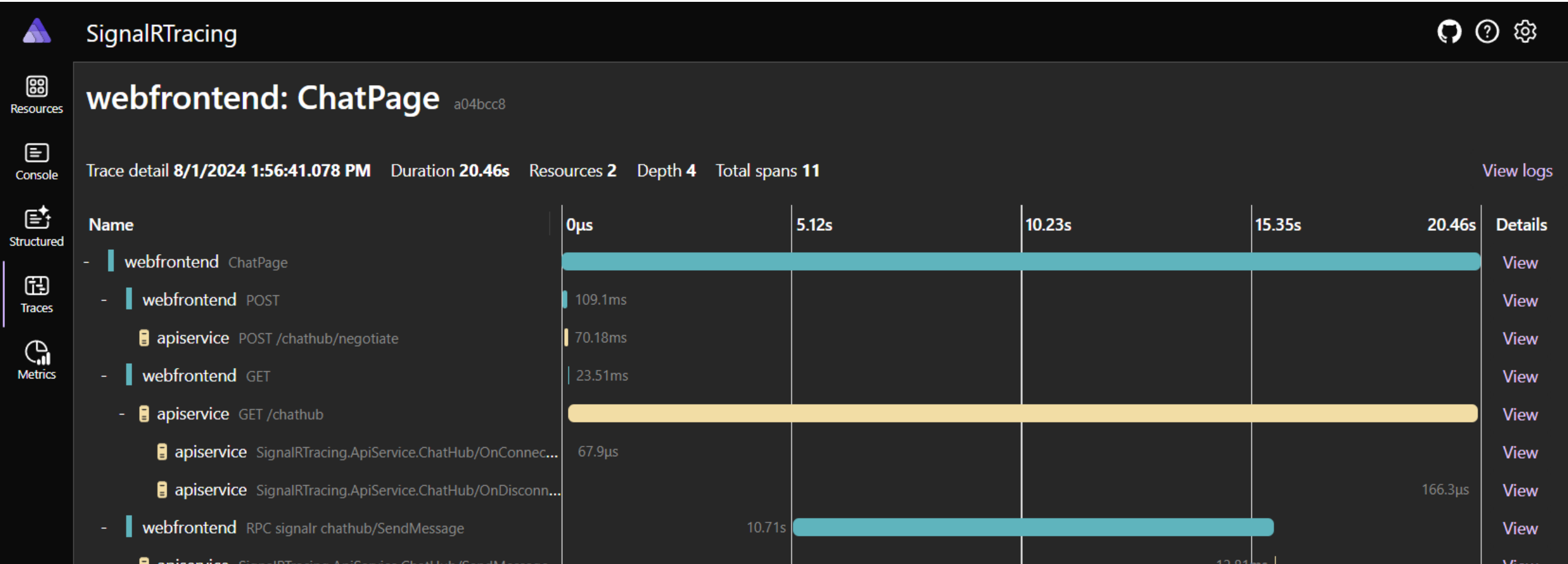
Each method is its own activity, which helps in monitoring and tracking activities during hub method calls.

This feature enhances the observability and debugging capabilities of SignalR applications.

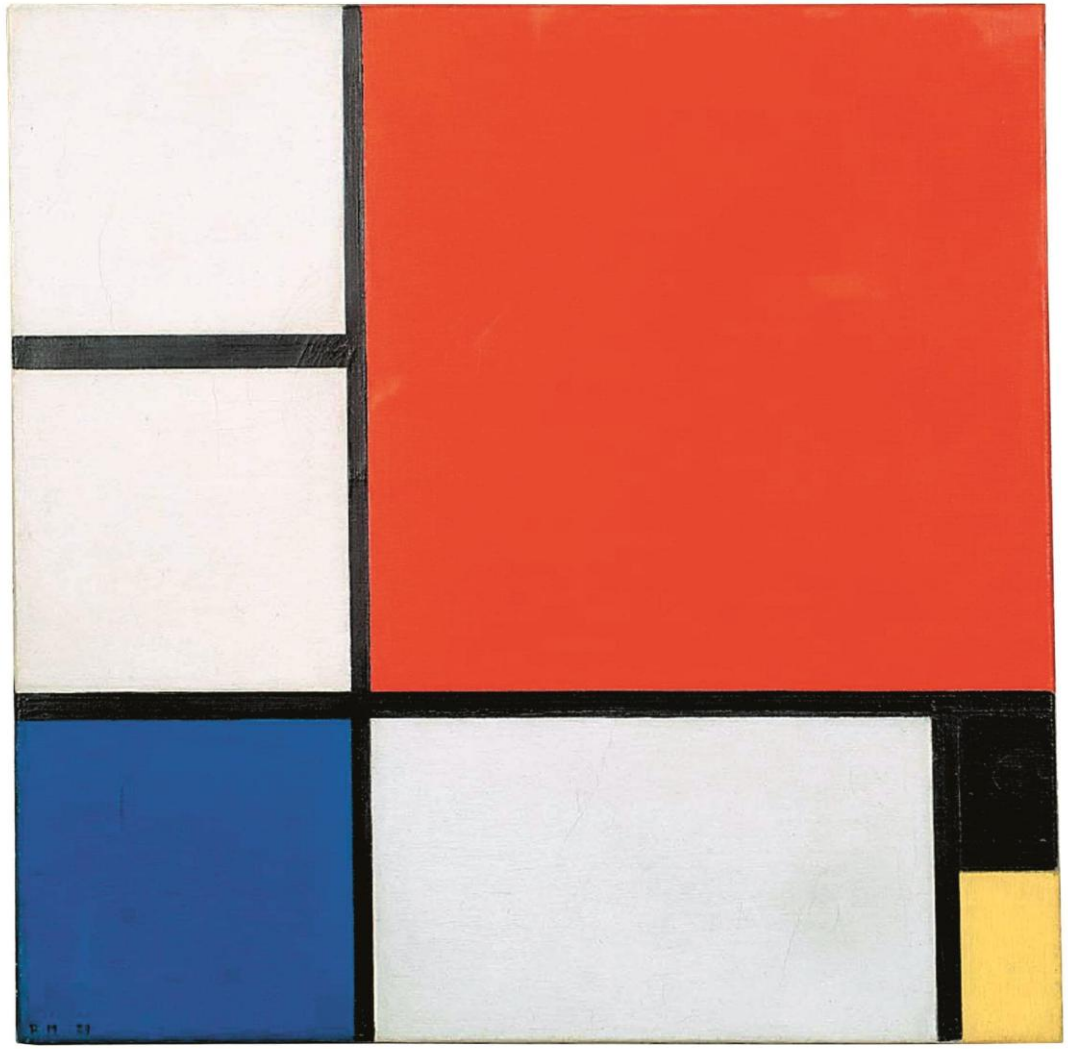


Improved SignalR Activities

Hub invocations from the client to the server now support *context propagation*. Propagating the trace context enables true distributed tracing. It's now possible to see invocations flow from the client to the server and back.



Minimal APIs



Piet Mondrian (1872-1944): Composizione II, 1929.

New Methods in TypedResults

The TypedResults class now includes factory methods and types for returning 500 Internal Server Error responses from endpoints.

```
var app = WebApplication.Create();

app.MapGet("/",
       () => TypedResults.InternalServerError("Something went wrong!"));

app.Run();
```

Support for ProducesProblem and ProducesValidationProblem

These extension methods have been updated to support their use on route groups.

They indicate that all endpoints in a route group can return ProblemDetails or ValidationProblemDetails responses for OpenAPI metadata purposes.

This feature improves error handling and documentation in minimal APIs.

```
var app = WebApplication.Create();

var todos = app.MapGroup("/todos")
               .ProducesProblem();
```

OpenAPI

Pablo Picasso – Guernica

OpenAPI Document Generation

Built-in support for generating OpenAPI documents representing controller-based or minimal APIs via the *Microsoft.AspNetCore.OpenApi* package.

Adding *builder.Services.AddOpenApi()* and *app.MapOpenApi()* in the app's configuration to enable OpenAPI document generation.

OpenAPI Document Generation

```
var builder = WebApplication.CreateBuilder();

builder.Services.AddOpenApi();

var app = builder.Build();

app.MapOpenApi();

app.MapGet("/hello/{name}", (string name) => $"Hello {name}"!);
app.Run();
```

Generatation at build-time

OpenAPI documents can also be generated at build-time by adding the *Microsoft.Extensions.ApiDescription.Server* package

```
<PropertyGroup>
  <OpenApiDocumentsDirectory>$(MSBuildProjectDirectory)</OpenApiDocumentsDirectory>
  <OpenApiGenerateDocuments>true</OpenApiGenerateDocuments>
</PropertyGroup>
```

DEMO

Blazor

Leonardo da Vinci – La Gioconda

Static asset delivery optimization

MapStaticAssets is a new middleware that helps optimize the delivery of static assets in any ASP.NET Core app, including Blazor apps.

Improves performance of Blazor apps

Benefits:

- Build time compression for all the assets in the app:
  - gzip during development and gzip + brotli during publish.
  - All assets are compressed with the goal of reducing the size of the assets to the minimum.
- Content based ETags: The ETags for each resource are the Base64 encoded string of the SHA-256 hash of the content. This ensures that the browser only redownloads a file if its contents have changed.

Static asset delivery optimization

File	Original	Compressed	% Reduction
bootstrap.min.css	163	17.5	89.26%
jquery.js	89.6	28	68.75%
bootstrap.min.js	78.5	20	74.52%
Total	**331.1**	**65.5**	**80.20%**

File	Original	Compressed	% Reduction
fluent.js	384	73	80.99%
fluent.css	94	11	88.30%
Total	**478**	**84**	**82.43%**

Detect rendering location, interactivity, and assigned render mode at runtime

New API designed to simplify the process of querying component states at runtime

- Determine the current execution location of the component
- Check if the component is running in an interactive environment
- Retrieve the assigned render mode for the component

Determine the current execution location

RendererInfo.Name returns the location where the component is executing:

- **Static**: On the server (SSR) and incapable of interactivity.
- **Server**: On the server (SSR) and capable of interactivity after prerendering.
- **WebAssembly**: On the client (CSR) and capable of interactivity after prerendering.
- **WebView**: On the native device and capable of interactivity after prerendering.

Check if the component is running in an interactive environment

RendererInfo.IsInteractive indicates if the component supports interactivity at the time of rendering. The value is true when rendering interactively or false when prerendering or for static SSR (Platform.Name of Static).

Retrieve the assigned render mode

ComponentBase.AssignedRenderMode exposes the component's assigned render mode:

- **InteractiveServer** for Interactive Server.
- **InteractiveAuto** for Interactive Auto.
- **InteractiveWebAssembly** for Interactive WebAssembly.

Improved Server-Side Reconnection Experience

Immediate Reconnection Attempt

- When a user navigates back to an app with a disconnected circuit, reconnection is attempted immediately rather than waiting for the next reconnect interval.
- This enhancement improves the user experience when navigating to an app in a browser tab that has gone to sleep.

Automatic Page Refresh

- If a reconnection attempt reaches the server but the server has already released the circuit, a page refresh occurs automatically.
- This prevents the user from having to manually refresh the page, increasing the likelihood of a successful reconnection.

Computed Backoff Strategy

- Reconnect timing uses a computed backoff strategy.
- The first several reconnection attempts occur in rapid succession without a retry interval before computed delays are introduced between attempts.
- Customizable retry interval behavior by specifying a function to compute the retry interval.

Modernized Reconnect UI

- The styling of the default reconnect UI has been modernized to provide a better user experience.

DEMO

Blazor

Simplified Authentication State Serialization for Blazor Web Apps

Simplified Authentication State Serialization for Blazor Web Apps

New APIs for Authentication

- Easier to add authentication to existing Blazor Web Apps
- Simplifies the process of serializing and deserializing authentication state

Simplified Code Implementation

- Reduces the amount of code needed to implement authentication state serialization
- Easier to add authentication to existing projects

Improved User Experience

- Ensures authentication state is available during prerendering
- Enhances security by handling authentication on the server

Authentication State Provider

Custom AuthenticationStateProvider

- Included in both server and client projects when creating a new Blazor Web App with authentication using Individual Accounts
- Flows the user's authentication state to the browser
- Allows the app to access authentication state during prerendering and before the .NET WebAssembly runtime is initialized

Persistent Component State Service

PersistentComponentState Service

- Used to serialize the authentication state into HTML comments
- Reads it back from WebAssembly to create a new AuthenticationState instance
- Works well if starting from the Blazor Web App project template with Individual Accounts option

New APIs

AddAuthenticationStateSerialization

- Adds necessary services to serialize the authentication state on the server

AddAuthenticationStateDeserialization

- Adds necessary services to deserialize the authentication state in the browser
- By default, only serializes the server-side name and role claims for access in the browser
- Option to include all claims

Add static server-side rendering (SSR) pages to a globally-interactive Blazor Web App

Simplify the addition of static SSR pages to Blazor Web Apps that use global interactivity.

ExcludeFromInteractiveRouting Attribute

- This attribute causes navigation to the page to exit from interactive routing, forcing a full-page reload.

Full-Page Reload

- Inbound navigation performs a full-page reload, allowing the app to switch to a different top-level render mode.
- The top-level root component (typically App.razor) rerenders from the server.

HttpContext.AcceptsInteractiveRouting Extension Method

- Detects whether [ExcludeFromInteractiveRouting] is applied to the current page.

Constructor injection

Razor components support constructor injection.

```
public partial class MyCustomComponent(NavigationManager navigation)
{
    private void HandleClick()
    {
        navigation.NavigateTo("/counter");
    }
}
```

.NET MAUI Blazor Hybrid and Web App solution template

A new solution template makes it easier to create .NET MAUI native and Blazor web client apps that share the same UI.

Maximizes code reuse

Key features include the ability to choose a Blazor interactive render mode for the web app and automatic creation of the appropriate projects, including a Blazor Web App and a .NET MAUI Blazor Hybrid app.

Blazor WebAssembly improvements

Investigate Blazor WebAssembly startup performance characteristics #54353

- https://github.com/dotnet/aspnetcore/issues/54353

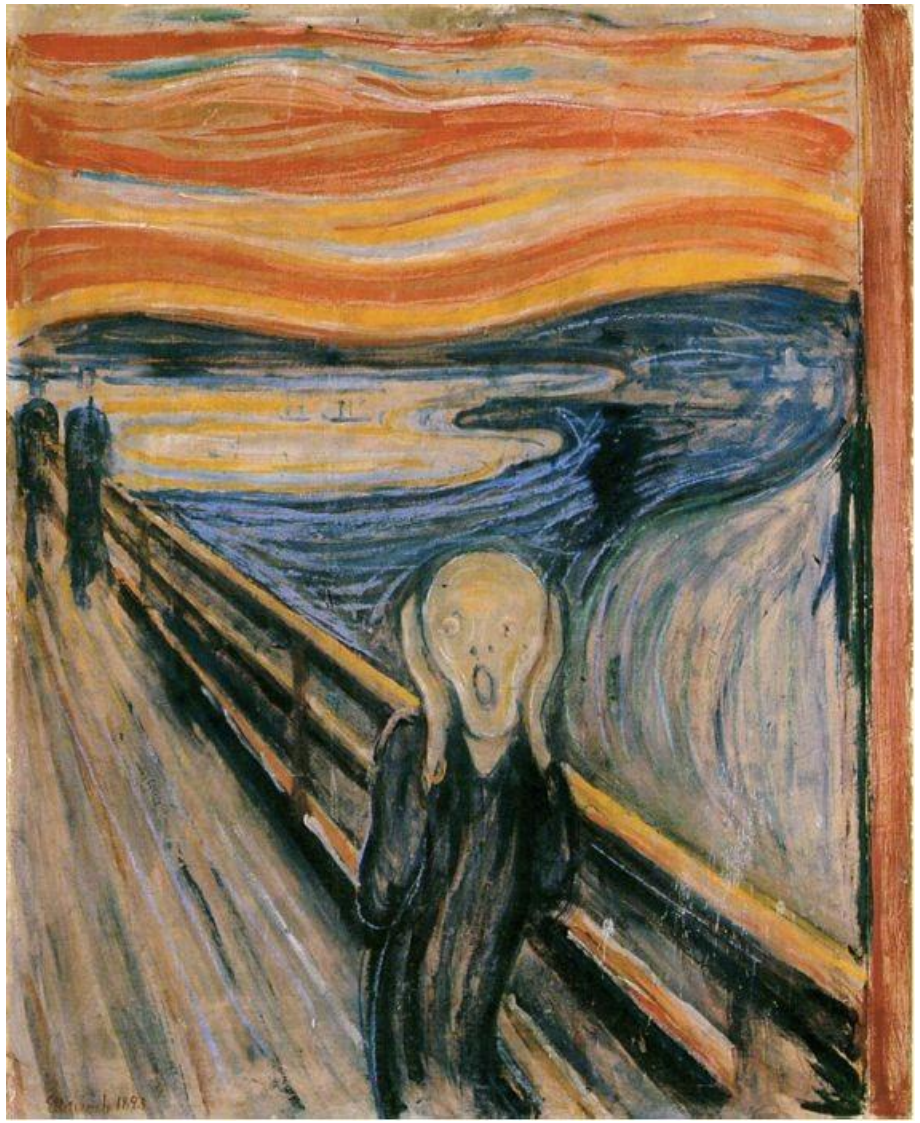
"*After introducing a change that utilized the STJ source generator during WebAssembly startup, we noticed a WebAssembly startup time improvement of 15-30%. These numbers may vary for larger apps, or apps that do significant work in user code during startup.*

Finding a way to eliminate JSON serialization altogether during startup may result in further improvement of startup time (I would conservatively estimate an additional 15% improvement, possibly higher for Blazor Web scenarios because it performs more serialization on startup)."

DEMO

HybridCache

HybridCache is currently still in preview but will be fully released after .NET 9.0 in a future minor release of .NET Extensions.



Edvard Munch – L'urlo

New HybridCache library

A new caching solution that combines the benefits of in-memory and distributed caching.

In-Memory Caching:

- Provides fast access to frequently used data.
- Reduces latency by storing data in memory.

Distributed Caching:

- Ensures data consistency across multiple instances.
- Supports scalability by distributing the cache across a cluster.

Automatic Synchronization

- Automatically synchronizes data between in-memory and distributed caches.
- Ensures data consistency and reliability.

HybridCache benefits

Performance: Combines the speed of in-memory caching with the scalability of distributed caching.

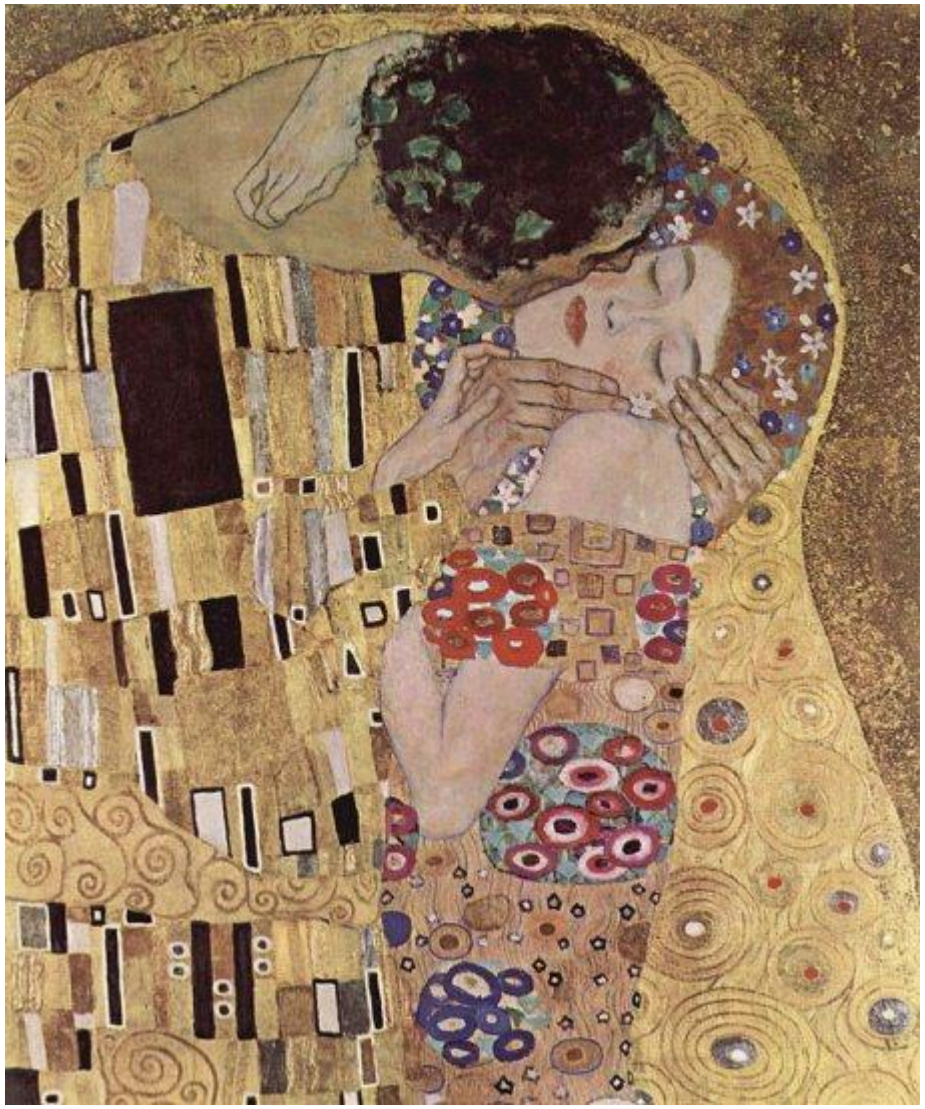
Scalability: Supports large-scale applications with high availability requirements.

Flexibility: Allows developers to choose the best caching strategy for their needs.

Adds new capabilities, such as:

- "Stampede" protection to prevent parallel fetches of the same work.
- Configurable serialization.

Native AOT



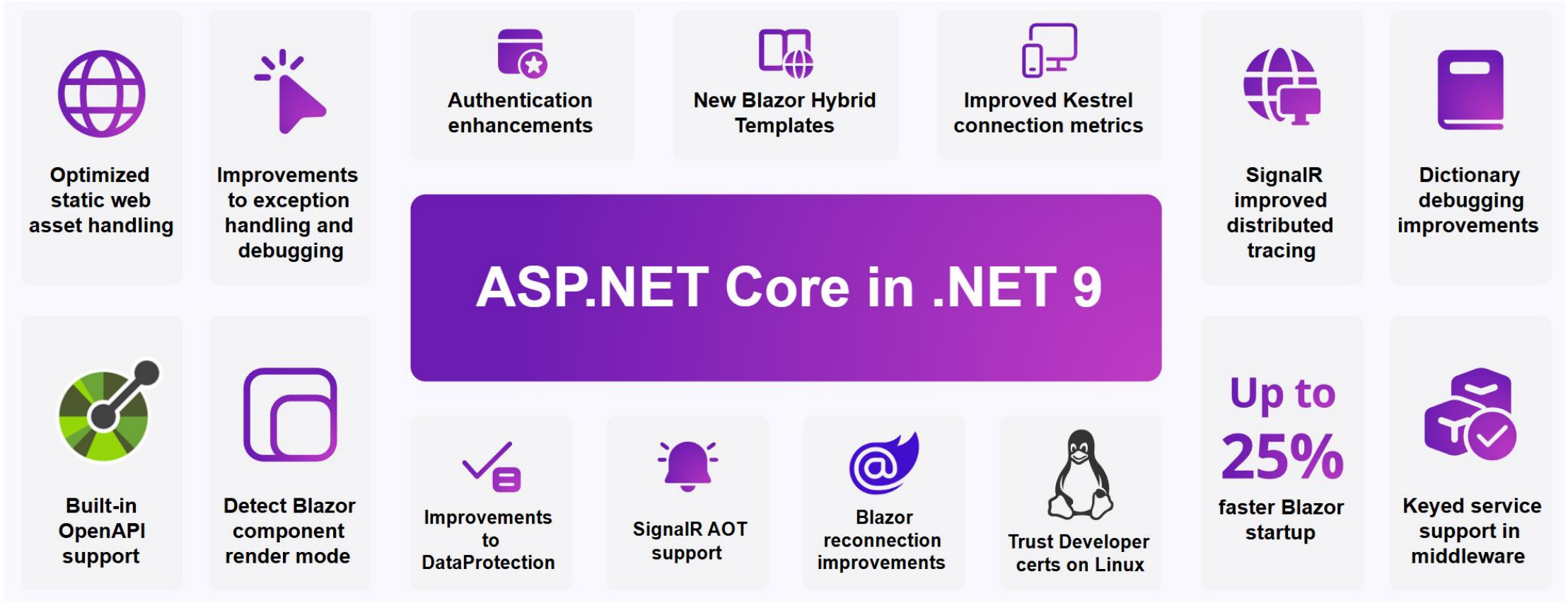
Gustav Klimt – Il bacio

Native AOT

The new built-in OpenAPI supports trimming and Native AOT.

SignalR supports trimming and Native AOT*

- Only the JSON protocol is currently supported:
  - Apps that use JSON serialization and Native AOT must use the System.Text.Json Source Generator.
- Hub method parameters of type IAsyncEnumerable<T> and ChannelReader<T> where T is a ValueType (struct) aren't supported.
- Strongly typed hubs aren't supported with Native AOT
- Only Task, Task<T>, ValueTask, or ValueTask<T> are supported for async return types.


Optimized static web asset handling
Improvements to exception handling and debugging
Authentication enhancements
New Blazor Hybrid Templates
Improved Kestrel connection metrics
SignalR improved distributed tracing
Dictionary debugging improvements
ASP.NET Core in .NET 9
Built-in OpenAPI support
Detect Blazor component render mode
Improvements to DataProtection
SignalR AOT support
Blazor reconnection improvements
Trust Developer certs on Linux
Up to 25% faster Blazor startup
Keyed service support in middleware

Contatti

Andrea Dottor

Microsoft MVP Developer Technologies

www.dottor.net

andrea@dottor.net

@dottor

il podcast