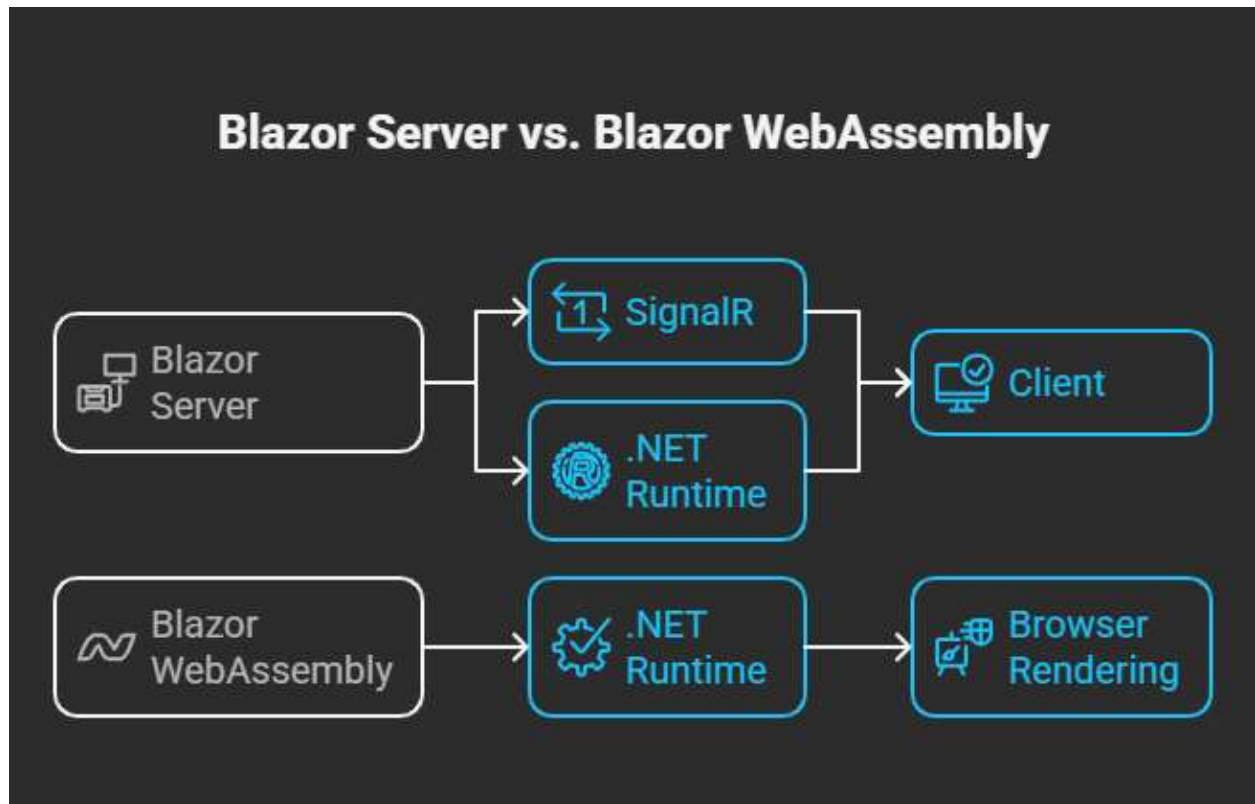


Blazor Architecture with ASP.NET Core and .NET

1. Positioning Blazor in the ASP.NET Core | .NET Ecosystem



Blazor is a **UI framework** that runs on top of **ASP.NET Core**, using the **.NET runtime** to build interactive web applications with C#.

At a high level:

- **ASP.NET Core** provides:
 - Web hosting
 - Middleware pipeline
 - Dependency Injection
 - Security (AuthN/AuthZ)

- Configuration & logging
 - **Blazor** provides:
 - Component-based UI model
 - Client-side routing
 - State-driven rendering
 - SPA behavior
-

2. High-Level Blazor Architecture

```
Browser
├─ Blazor UI (Components)
│   └─ Razor (.razor)
│   └─ Routing
│   └─ State Management
├─ Communication Layer
│   └─ HTTP (WebAssembly)
│   └─ SignalR (Server)
└─ ASP.NET Core Host
    └─ Middleware Pipeline
    └─ Dependency Injection
    └─ Authentication / Authorization
    └─ Controllers / Minimal APIs / gRPC
    └─ Data Access (EF Core, Dapper)
```

Blazor is **not a replacement** for ASP.NET Core—it is **a UI layer that plugs into it**.

3. Core Architectural Building Blocks

3.1 Razor Components

- .razor files combine:
 - HTML (markup)
 - C# (logic)
 - Components are:
 - Reusable
 - Stateful
 - Event-driven
-

3.2 Rendering Engine

Blazor uses a **diff-based rendering model**:

- UI is represented as a **render tree**
- On state change:
 - Blazor computes the diff
 - Only minimal DOM updates are applied

This is similar in concept to React's virtual DOM but implemented in .NET.

3.3 Dependency Injection (DI)

Blazor fully relies on **ASP.NET Core's built-in DI container**.

```
builder.Services.AddScoped<ProductService>();
```

Scopes depend on hosting model:

- **Server**: per user connection (circuit)
 - **WebAssembly**: per browser session
-

4. Blazor Hosting Models in ASP.NET Core



4.1 Blazor Server Architecture

Execution Location

- UI logic runs on the **server**
- Browser holds a **SignalR connection**

Flow

```
Browser Event
→ SignalR
→ Server-side Component Logic
→ Render Diff
→ SignalR
→ DOM Update
```

4.2 Blazor WebAssembly Architecture

Execution Location

- Entire app runs in the **browser**
- .NET runtime executes via **WebAssembly**

Flow

```
Browser Loads:
- .NET Runtime (WASM)
- Blazor App DLLs

UI Event:
→ Client-side C# Execution
→ HTTP API Call (if needed)
→ UI Update
```

5. ASP.NET Core Middleware Pipeline Integration

Blazor apps are hosted inside the **ASP.NET Core request pipeline**.

```
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.MapBlazorHub();           // Server
app.MapFallbackToPage("/_Host");
```

6. Communication Patterns

Blazor Server

- Uses **SignalR**
- Persistent, stateful connection
- Real-time UI updates

Blazor WebAssembly

- Uses **HttpClient**
- Stateless REST / gRPC-Web calls
- Same architecture as modern SPAs

7. Security Architecture

- Authentication handled by ASP.NET Core:
 - Cookies
 - JWT
 - OAuth / OpenID Connect
- Authorization:

- [Authorize]
 - Policy-based
 - Role-based
 - Blazor enforces security **both at route and component level**
-

8. Deployment Architecture

Blazor Server

- Deployed as an ASP.NET Core app
- Runs on:
 - IIS
 - Azure App Service
 - Containers

Blazor WebAssembly

- Static files (CDN-friendly)
 - Backend APIs hosted separately or together
 - Ideal for cloud-native architectures
-

9. Architectural Summary

Layer	Responsibility
Browser	UI rendering & events
Blazor	Component model & SPA behavior
ASP.NET Core	Hosting, DI, middleware

Layer	Responsibility
.NET Runtime	Execution & memory management
Backend APIs	Business logic & data
