

What's new in ASP.NET Core 2.2

Article • 06/04/2022

This article highlights the most significant changes in ASP.NET Core 2.2, with links to relevant documentation.

OpenAPI Analyzers & Conventions

OpenAPI (formerly known as Swagger) is a language-agnostic specification for describing REST APIs. The OpenAPI ecosystem has tools that allow for discovering, testing, and producing client code using the specification. Support for generating and visualizing OpenAPI documents in ASP.NET Core MVC is provided via community driven projects such as [NSwag](#) and [Swashbuckle.AspNetCore](#). ASP.NET Core 2.2 provides improved tooling and runtime experiences for creating OpenAPI documents.

For more information, see the following resources:

- [Use web API analyzers](#)
- [Use web API conventions](#)
- [ASP.NET Core 2.2.0-preview1: OpenAPI Analyzers & Conventions](#)

Problem details support

ASP.NET Core 2.1 introduced `ProblemDetails`, based on the [RFC 7807](#) specification for carrying details of an error with an HTTP Response. In 2.2, `ProblemDetails` is the standard response for client error codes in controllers attributed with `ApiControllerAttribute`. An `ActionResult` returning a client error status code (4xx) now returns a `ProblemDetails` body. The result also includes a correlation ID that can be used to correlate the error using request logs. For client errors, `ProducesResponseType` defaults to using `ProblemDetails` as the response type. This is documented in OpenAPI / Swagger output generated using NSwag or Swashbuckle.AspNetCore.

Endpoint Routing

ASP.NET Core 2.2 uses a new *endpoint routing* system for improved dispatching of requests. The changes include new link generation API members and route parameter transformers.

For more information, see the following resources:

- [Endpoint routing in 2.2](#)
- [Route parameter transformers](#) (see **Routing** section)
- [Differences between IRouter- and endpoint-based routing](#)

Health checks

A new health checks service makes it easier to use ASP.NET Core in environments that require health checks, such as Kubernetes. Health checks includes middleware and a set of libraries that define an `IHealthCheck` abstraction and service.

Health checks are used by a container orchestrator or load balancer to quickly determine if a system is responding to requests normally. A container orchestrator might respond to a failing health check by halting a rolling deployment or restarting a container. A load balancer might respond to a health check by routing traffic away from the failing instance of the service.

Health checks are exposed by an application as an HTTP endpoint used by monitoring systems. Health checks can be configured for a variety of real-time monitoring scenarios and monitoring systems. The health checks service integrates with the [BeatPulse project](#), which makes it easier to add checks for dozens of popular systems and dependencies.

For more information, see [Health checks in ASP.NET Core](#).

HTTP/2 in Kestrel

ASP.NET Core 2.2 adds support for HTTP/2.

HTTP/2 is a major revision of the HTTP protocol. Notable features of HTTP/2 include:

- Support for header compression.
- Fully multiplexed streams over a single connection.

While HTTP/2 preserves HTTP's semantics (for example, HTTP headers and methods), it's a breaking change from HTTP/1.x on how data is framed and sent between the client and server.


As a consequence of this change in framing, servers and clients need to negotiate the protocol version used. Application-Layer Protocol Negotiation (ALPN) is a TLS extension that allows the server and client to negotiate the protocol version used as part of their TLS handshake. While it is possible to have prior knowledge between the server and the

client on the protocol, all major browsers support ALPN as the only way to establish an HTTP/2 connection.

For more information, see [HTTP/2 support](#).

Kestrel configuration

In earlier versions of ASP.NET Core, Kestrel options are configured by calling `UseKestrel`. In 2.2, Kestrel options are configured by calling `ConfigureKestrel` on the host builder. This change resolves an issue with the order of `IServer` registrations for in-process hosting. For more information, see the following resources:

- [Mitigate UseIIS conflict](#) 
- [Configure Kestrel server options with ConfigureKestrel](#)

IIS in-process hosting

In earlier versions of ASP.NET Core, IIS serves as a reverse proxy. In 2.2, the ASP.NET Core Module can boot the CoreCLR and host an app inside the IIS worker process (*w3wp.exe*). In-process hosting provides performance and diagnostic gains when running with IIS.

For more information, see [in-process hosting for IIS](#).

SignalR Java client

ASP.NET Core 2.2 introduces a Java Client for SignalR. This client supports connecting to an ASP.NET Core SignalR Server from Java code, including Android apps.

For more information, see [ASP.NET Core SignalR Java client](#).

CORS improvements

In earlier versions of ASP.NET Core, CORS Middleware allows `Accept`, `Accept-Language`, `Content-Language`, and `Origin` headers to be sent regardless of the values configured in `CorsPolicy.Headers`. In 2.2, a CORS Middleware policy match is only possible when the headers sent in `Access-Control-Request-Headers` exactly match the headers stated in `WithHeaders`.

For more information, see [CORS Middleware](#).

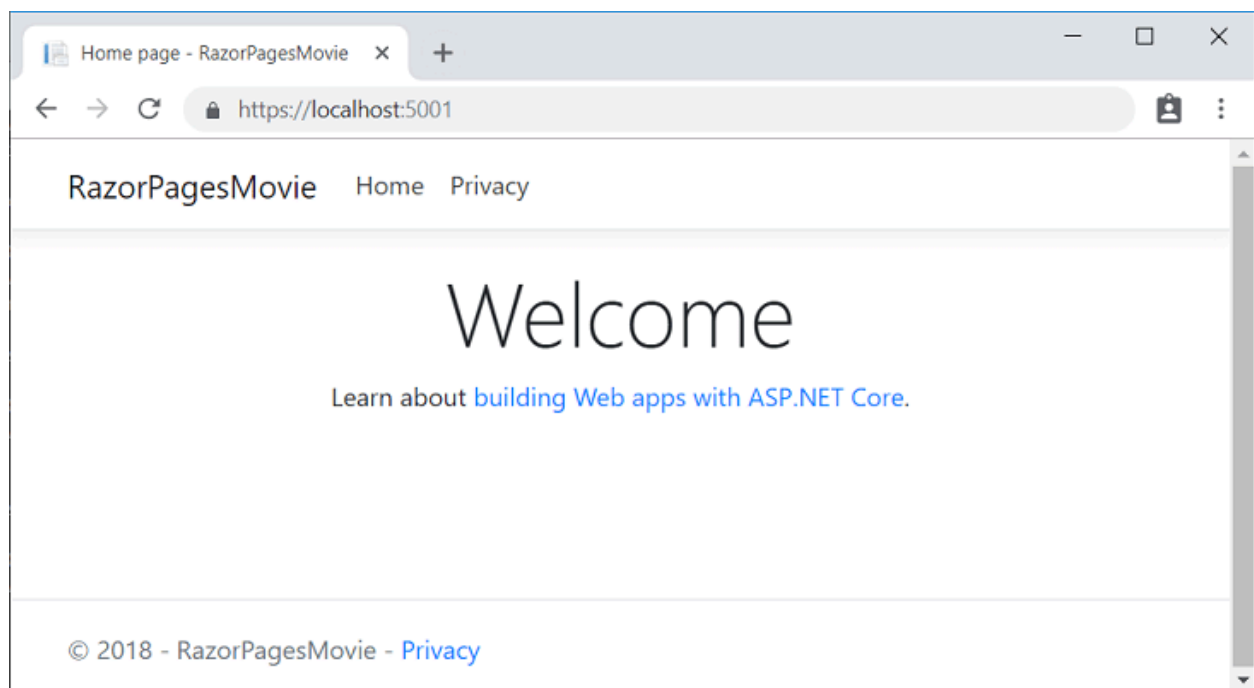
Response compression

ASP.NET Core 2.2 can compress responses with the [Brotli compression format](#).

For more information, see [Response Compression Middleware supports Brotli compression](#).

Project templates

ASP.NET Core web project templates were updated to [Bootstrap 4](#) and [Angular 6](#). The new look is visually simpler and makes it easier to see the important structures of the app.



Validation performance

MVC's validation system is designed to be extensible and flexible, allowing you to determine on a per request basis which validators apply to a given model. This is great for authoring complex validation providers. However, in the most common case an application only uses the built-in validators and don't require this extra flexibility. Built-in validators include DataAnnotations such as [Required] and [StringLength], and `IValidatableObject`.

In ASP.NET Core 2.2, MVC can short-circuit validation if it determines that a given model graph doesn't require validation. Skipping validation results in significant improvements when validating models that can't or don't have any validators. This includes objects

such as collections of primitives (such as `byte[]`, `string[]`, `Dictionary<string, string>`), or complex object graphs without many validators.

HTTP Client performance

In ASP.NET Core 2.2, the performance of `SocketsHttpHandler` was improved by reducing connection pool locking contention. For apps that make many outgoing HTTP requests, such as some microservices architectures, throughput is improved. Under load, `HttpClient` throughput can be improved by up to 60% on Linux and 20% on Windows.

For more information, see [the pull request that made this improvement](#).

Additional information

For the complete list of changes, see the [ASP.NET Core 2.2 Release Notes](#).

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).

.NET

ASP.NET Core feedback

ASP.NET Core is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)