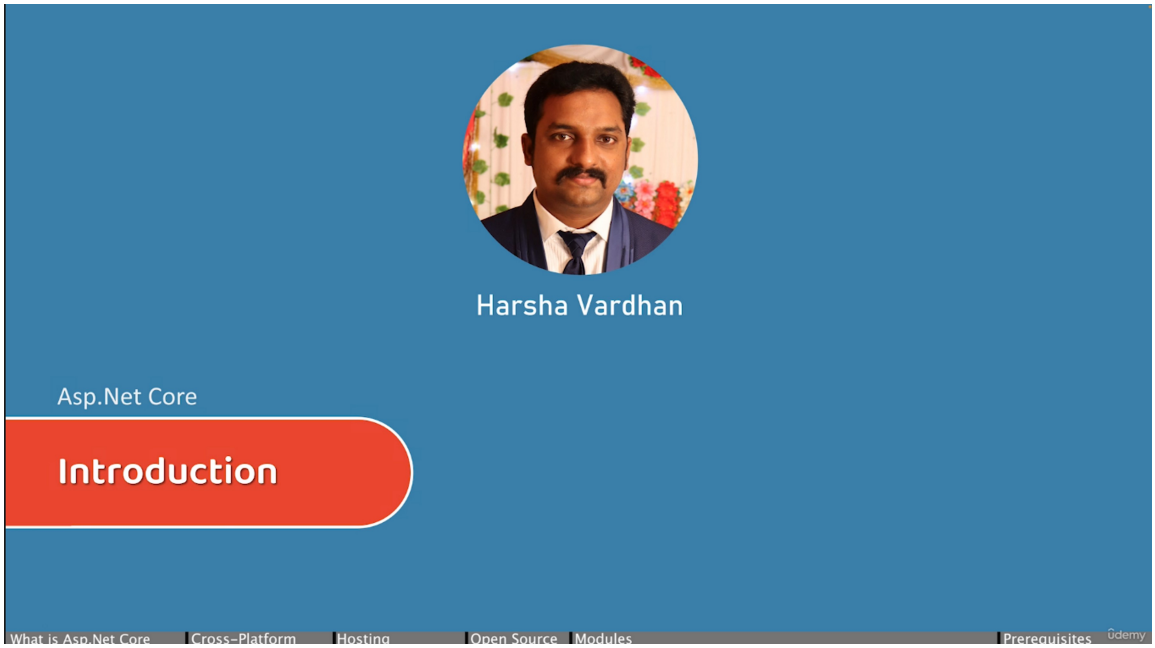
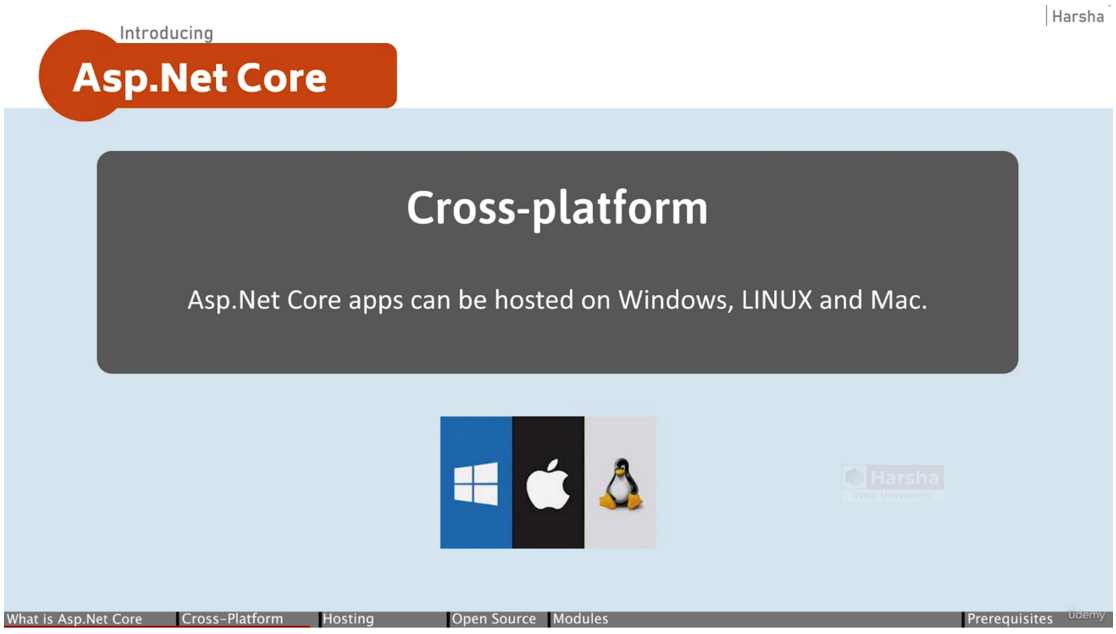
Section 1: Introduction

29 December 2024 11:45

1. Introduction to Asp.Net Core



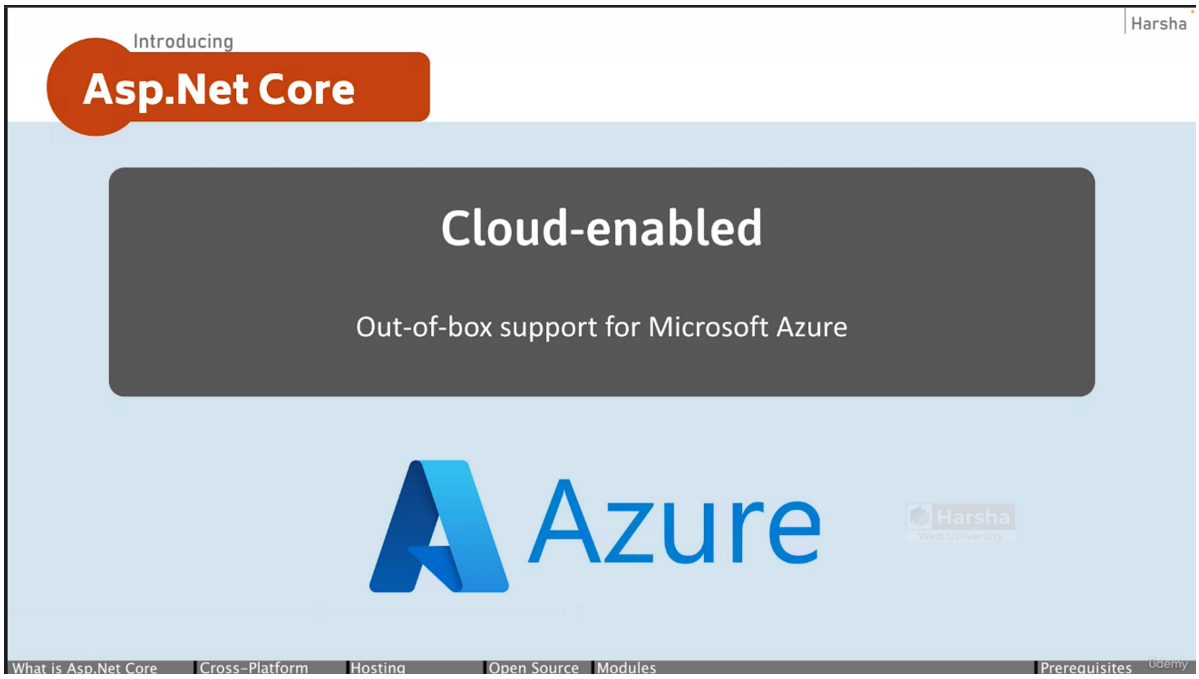
Welcome to the true ASP.NET Core Ultimate Guide.
Before getting started with ASP.NET Core application development, I would like to introduce what ASP.NET Core is.
ASP.NET Core is a cross-platform, high-performance, open-source web application development framework.
Using ASP.NET Core, you can create web applications and services. This means you can build web applications of any scale—small, medium, or complex web applications.
Additionally, you can create REST API services, which can act as backends for web applications and mobile applications.

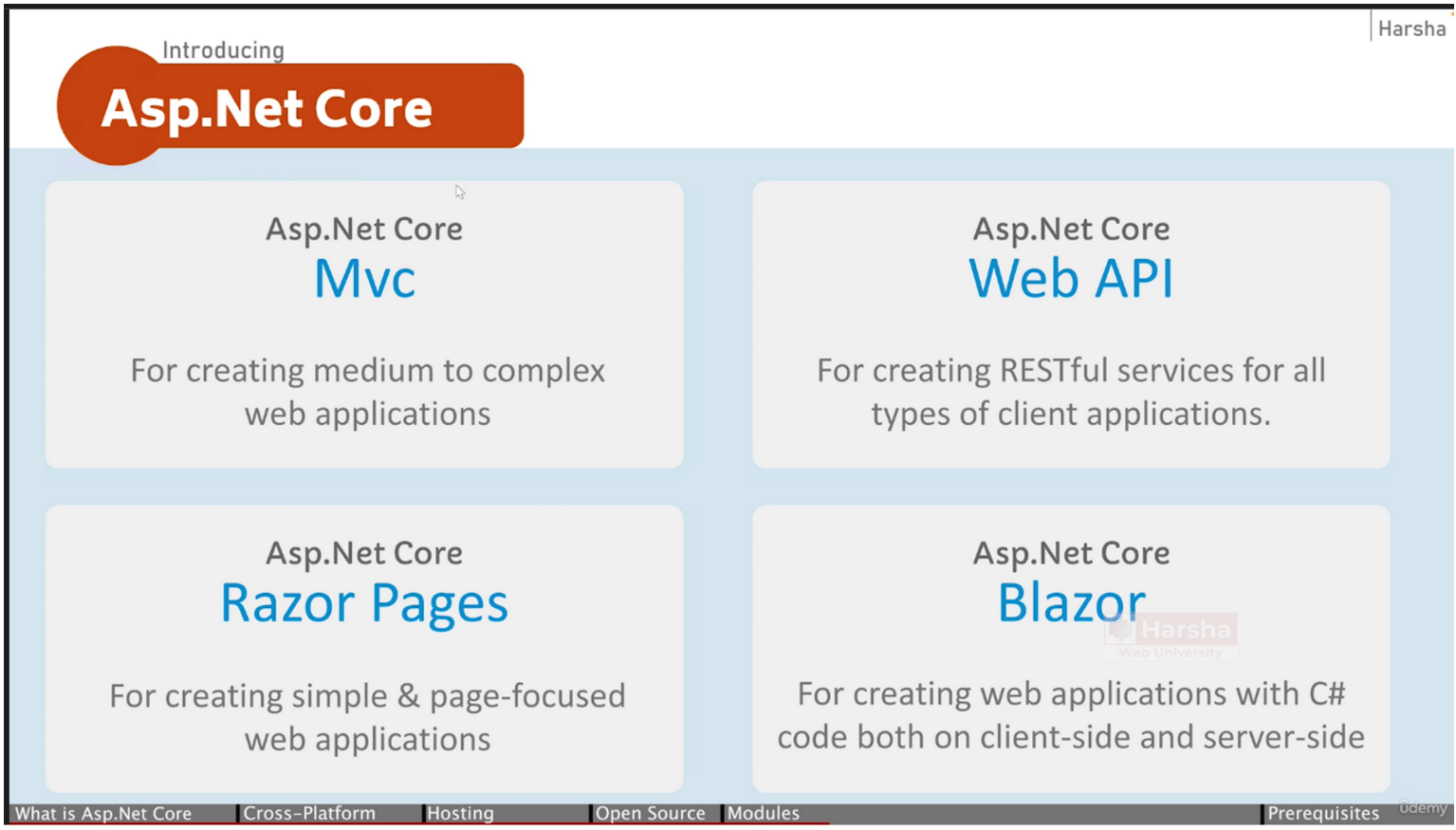


Let us explore the features of ASP.NET Core.
ASP.NET Core is cross-platform.
After completing your ASP.NET Core application development, you will need to host it. This means you must install it on your server machine so it can start receiving requests from clients and providing responses.
What kind of operating system is required on the server machine?
ASP.NET Core supports three main operating systems: Windows, Linux, and Mac.
Generally, in most cases, Windows or Mac is used on the developer machine, and Linux is used on the server machine.

ASP.NET Core supports three main operating systems: **Windows**, **Linux**, and **Mac**.
Generally, **Windows** or **Mac** is used on the developer machine, while **Linux** is commonly used on the server machine.
On the server machine, there must be software that can receive requests and provide responses. This is referred to as a **reverse proxy**.
ASP.NET Core supports **Kestrel** by default as the application server, and additional reverse proxies like **IIS**, **NGINX**, and **Docker**.
Docker, in particular, supports various operating systems. You can run **Windows** or **Linux** on Docker, and your ASP.NET Core application can run seamlessly on any of these servers.

ASP.NET Core is **open source**, which means its source code is freely available on **GitHub**.
The benefit of being open source is that you receive frequent updates for ASP.NET Core.
ASP.NET Core is developed with the **cloud** in mind. This means it provides built-in support for **Microsoft Azure Cloud**.





There are four parts of ASP.NET Core:

1. ASP.NET Core MVC
2, ASP.NET Core Web API
1. Razor Pages
2. Blazor

Most web applications are developed using ASP.NET Core MVC, which should not be confused with ASP.NET MVC that was part of the older .NET Framework.

When using the Model-View-Controller (MVC) pattern in ASP.NET Core, it is referred to as ASP.NET Core MVC.

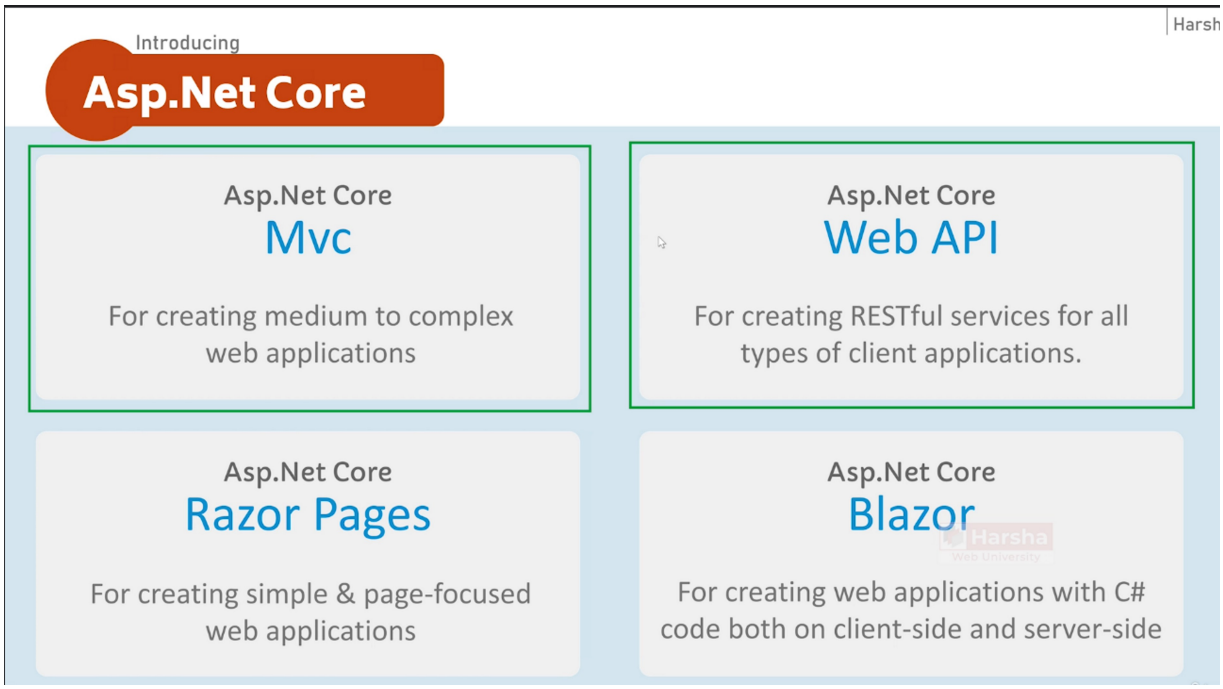
If you create only controllers with models but no views, it is referred to as ASP.NET Core Web API.

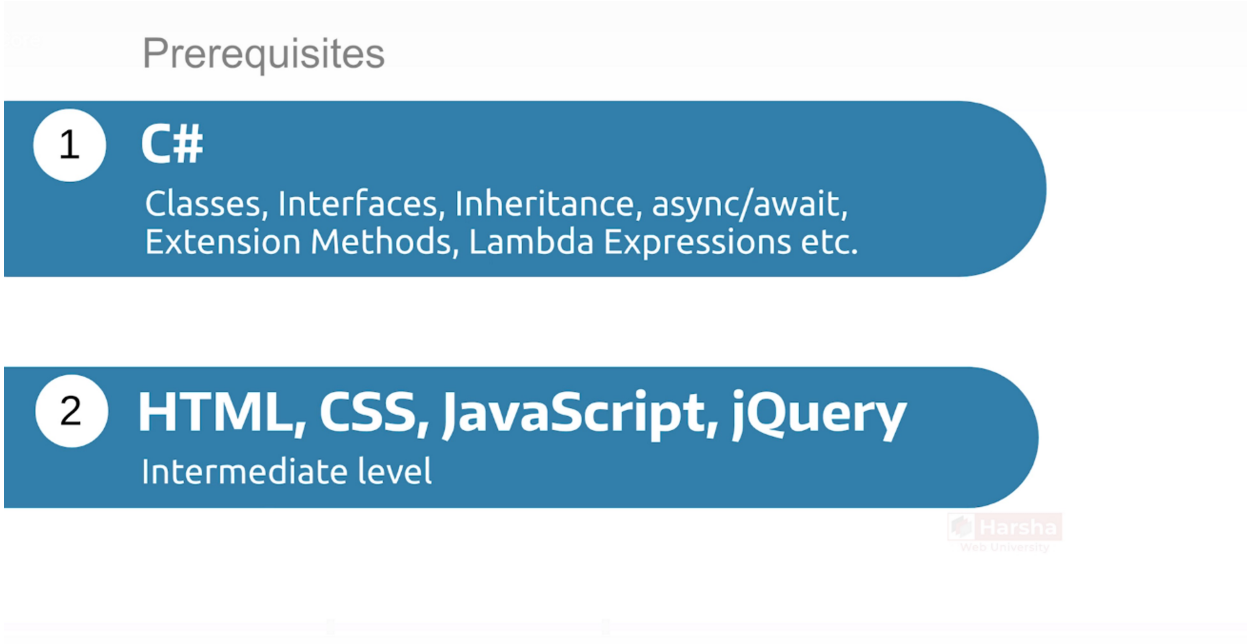
ASP.NET Core Web API is typically used to create RESTful services.
These services receive requests and provide only data (not views) as a response.
This allows the front end to be created using any web or mobile application framework, such as:
Web frameworks: React, Angular
Mobile frameworks: Xamarin, Ionic
The Web API acts as a backend for these front ends.
For simple and page-focused scenarios, Razor Pages is available.

If the development team prefers to use C# exclusively on both the server side and the client side, ASP.NET Blazor is an ideal choice.

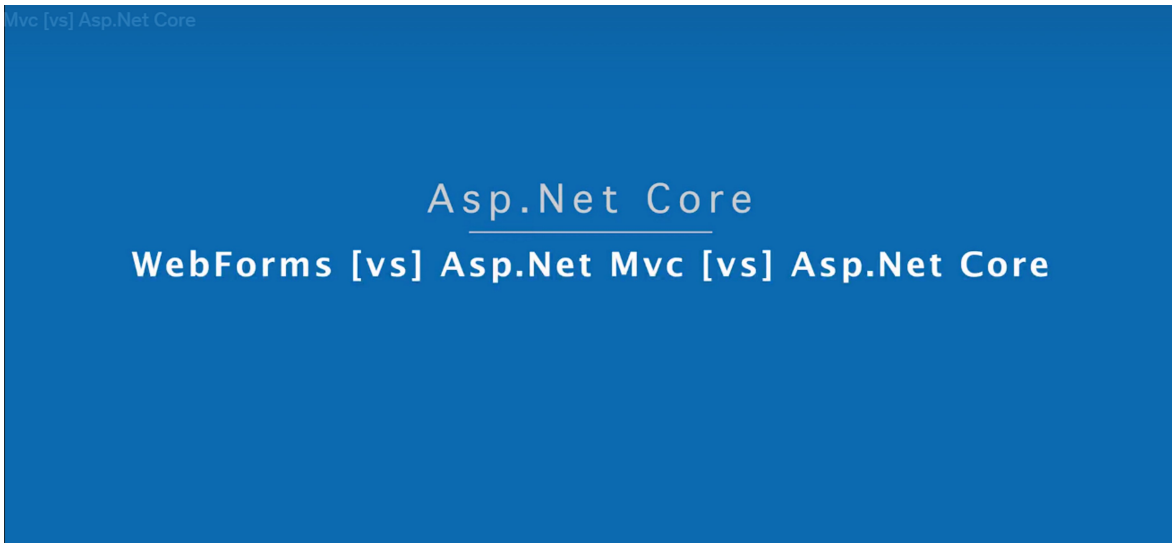
Among these four parts, the first two (ASP.NET Core MVC and ASP.NET Core Web API) are the most commonly used in real-world projects.

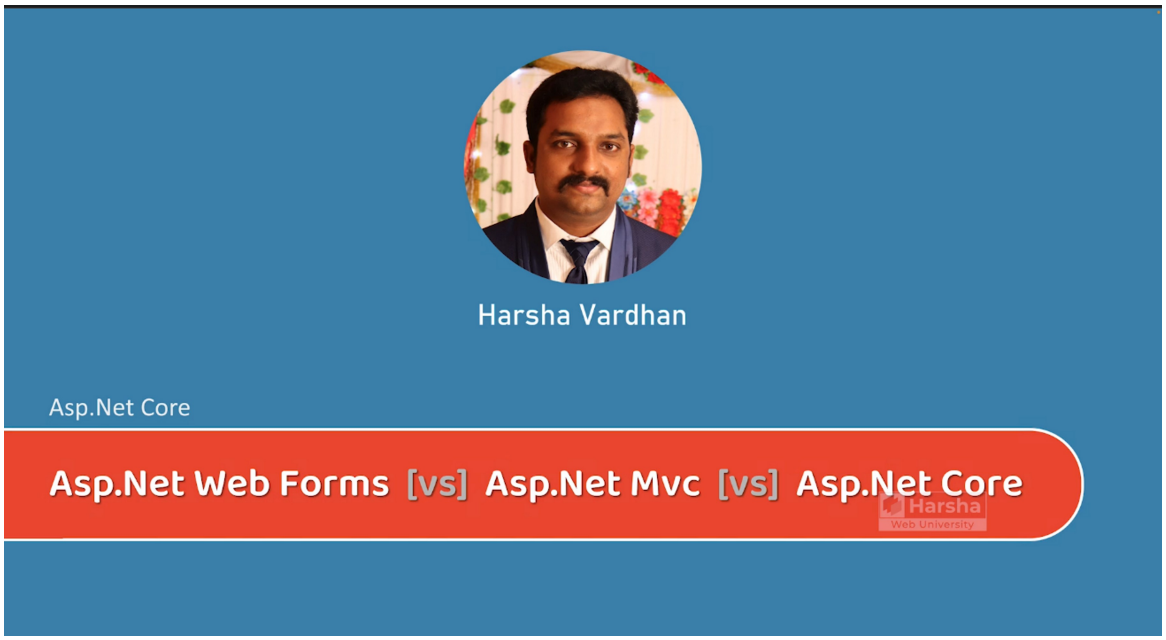
This course will focus on the first two parts, providing a comprehensive understanding of all essential ASP.NET Core concepts with practical examples.

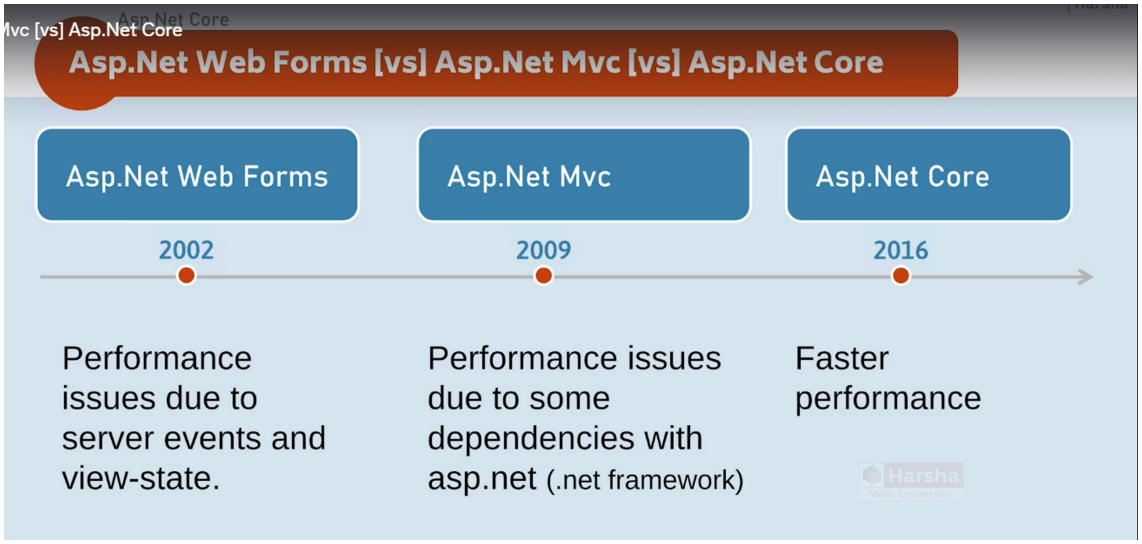


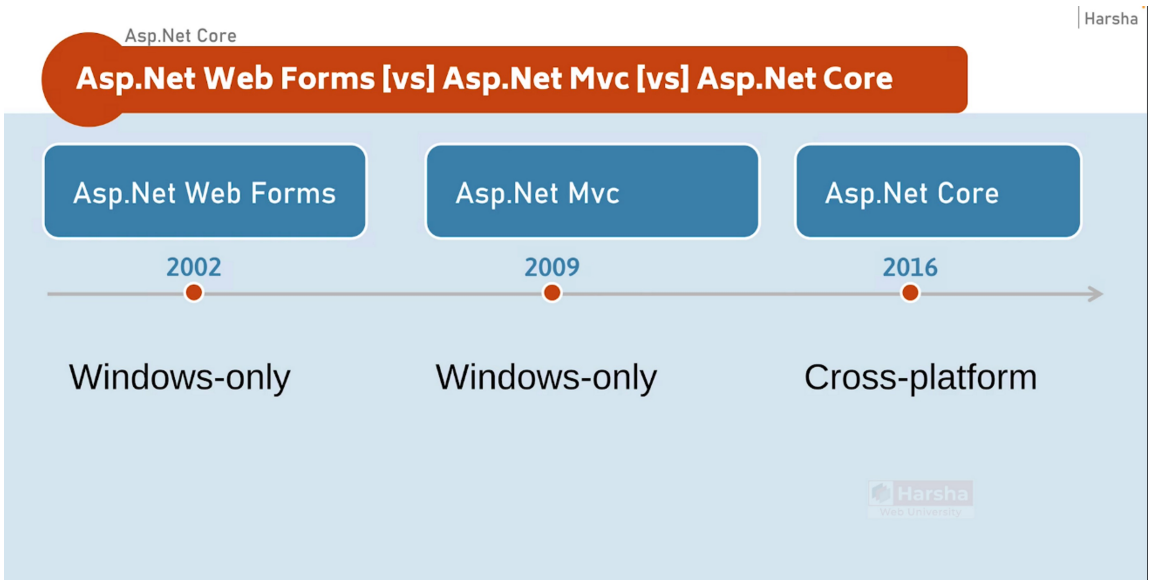


2. WebForms [vs] Asp.Net Mvc [vs] Asp.Net Core



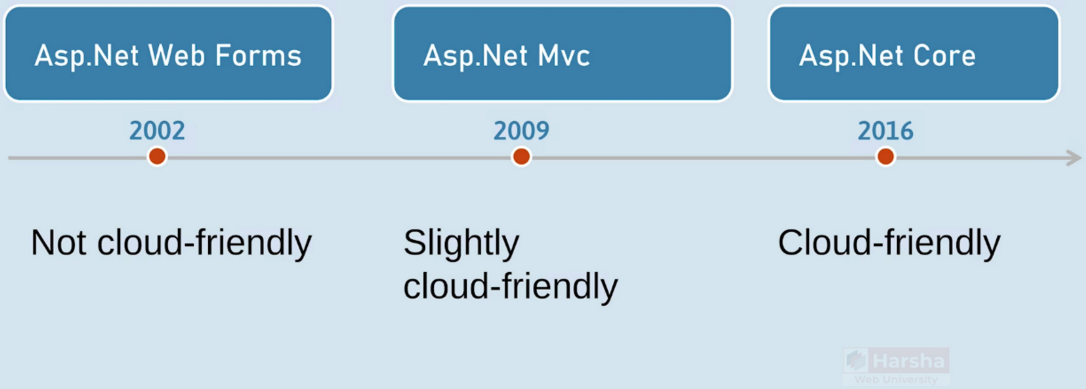

Harsha Vardhan
Asp.Net Core
Asp.Net Web Forms [vs] Asp.Net Mvc [vs] Asp.Net Core
Harsha
Web University


Asp.Net Core
Asp.Net Web Forms [vs] Asp.Net Mvc [vs] Asp.Net Core
Asp.Net Web Forms
Asp.Net Mvc
Asp.Net Core
2002
2009
2016
Performance issues due to server events and view-state.
Performance issues due to some dependencies with asp.net (.net framework)
Faster performance
Harsha
Web University


Harsha
Asp.Net Core
Asp.Net Web Forms [vs] Asp.Net Mvc [vs] Asp.Net Core
Asp.Net Web Forms
Asp.Net Mvc
Asp.Net Core
2002
2009
2016
Windows-only
Windows-only
Cross-platform
Harsha
Web University

Asp.Net Core

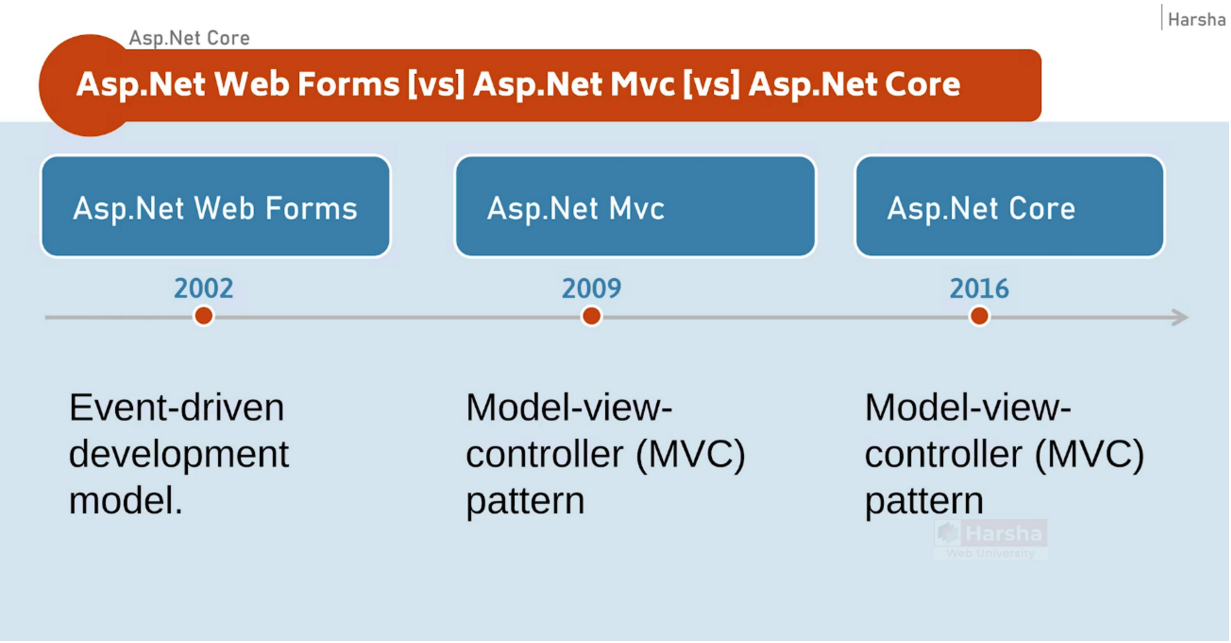
Asp.Net Web Forms [vs] Asp.Net Mvc [vs] Asp.Net Core



Harsha

Asp.Net Core

Asp.Net Web Forms [vs] Asp.Net Mvc [vs] Asp.Net Core



Let us explore the features of ASP.NET Core.

ASP.NET Core is cross-platform.
After completing your ASP.NET Core application development, you will need to host it. This means you must install it on your server machine so it can start receiving requests from clients and providing responses.
What kind of operating system is required on the server machine?
ASP.NET Core supports three main operating systems: Windows, Linux, and Mac. Generally, in most cases, Windows or Mac is used on the developer machine, and Linux is used on the server machine.

ASP.NET Core supports three main operating systems: Windows, Linux, and Mac. Generally, Windows or Mac is used on the developer machine, while Linux is commonly used on the server machine.
On the server machine, there must be software that can receive requests and provide responses. This is referred to as a reverse proxy.
ASP.NET Core supports Kestrel by default as the application server, and additional reverse proxies like IIS, NGINX, and Docker.
Docker, in particular, supports various operating systems. You can run Windows or Linux on Docker, and your ASP.NET Core application can run seamlessly on any of these servers.

ASP.NET Core is open source, which means its source code is freely available on GitHub.
The benefit of being open source is that you receive frequent updates for ASP.NET Core.
ASP.NET Core is developed with the cloud in mind. This means it provides built-in support for Microsoft Azure Cloud.

There are four parts of ASP.NET Core:

1. ASP.NET Core MVC
2. ASP.NET Core Web API
1. Razor Pages
2. Blazor

Most web applications are developed using ASP.NET Core MVC, which should not be confused with ASP.NET MVC that was part of the older .NET Framework.

When using the Model-View-Controller (MVC) pattern in ASP.NET Core, it is referred to as ASP.NET Core MVC.

If you create only controllers with models but no views, it is referred to as ASP.NET Core Web API.

ASP.NET Core Web API is typically used to create RESTful services.
These services receive requests and provide only data (not views) as a response.
This allows the front end to be created using any web or mobile application framework, such as:
Web frameworks: React, Angular
Mobile frameworks: Xamarin, Ionic
The Web API acts as a backend for these front ends.
For simple and page-focused scenarios, Razor Pages is available.

If the development team prefers to use C# exclusively on both the server side and the client side, ASP.NET Blazor is an ideal choice.

Among these four parts, the first two (ASP.NET Core MVC and ASP.NET Core Web API) are the most commonly used in real-world projects.

This course will focus on the first two parts, providing a comprehensive understanding of all essential ASP.NET Core concepts with practical examples.

2. WebForms [vs] Asp.Net Mvc [vs] Asp.Net Core

Key Benefits of ASP.NET Core

1. Performance
   - ASP.NET Core offers high performance compared to ASP.NET Web Forms and ASP.NET MVC:
     - Web Forms:
       - § Slower due to ViewState (stores page state, increasing payload).
       - § Server events and page lifecycle add complexity and overhead.
     - MVC:
       - § Performance issues arise from dependencies on legacy components like System.Web.dll.
2. Cross-Platform Compatibility
   - ASP.NET Core is cross-platform from the ground up, working seamlessly on Windows, Linux, and macOS.
   - In contrast, both Web Forms and MVC are tightly coupled with the .NET Framework, which is Windows-specific.
3. Cloud-Readiness
   - ASP.NET Core is cloud-friendly, designed with cloud computing in mind.
   - It has first-class support for Microsoft Azure and other cloud platforms.
   - Web Forms and MVC were developed before cloud computing became mainstream, making them less suited for modern cloud-hosted applications.
4. Open Source
   - ASP.NET Core is fully open source, with its codebase available on GitHub, enabling frequent updates and improvements.
   - While ASP.NET MVC is open source, its dependency on the proprietary .NET Framework limits its flexibility.
   - ASP.NET Web Forms is not open source.
5. Lightweight and Modular
   - ASP.NET Core is built on .NET Core, making it lightweight and modular.
   - It eliminates unnecessary dependencies like System.Web.dll, improving performance.
6. Separation of Concerns
   - Like ASP.NET MVC, ASP.NET Core follows the Model-View-Controller (MVC) design pattern, enabling clean separation of concerns.
   - This approach enhances maintainability and allows for unit testing of controllers, models, and views independently.
7. Event-Driven Programming
   - ASP.NET Web Forms uses an event-driven programming model, which adds complexity.
   - ASP.NET Core simplifies development by using modern patterns such as dependency injection and middleware.
8. Future-Proof
   - ASP.NET Core continuously evolves, with new versions such as ASP.NET Core 6 and upcoming releases (ASP.NET Core 7, 8, etc.) providing improved features and capabilities.
   - Microsoft actively supports and updates ASP.NET Core, ensuring its relevance in modern application development.
9. Popularity in the Open Source Community
   - ASP.NET Core has gained significant traction and popularity in the open-source community due to its flexibility and modern design.