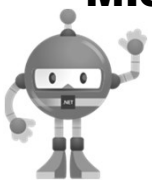
Welcome to the

Microsoft

.NET Core

B's Learning World

Microsoft .NET Core



Free. Cross-platform. Open source

A developer platform for building your all apps

Supported on Windows, Linux, and macOS

Presented by Ranjan Bhatnagar

bhatnagar_ranjan@outlook.com

Know Your Instructor

Ranjan Bhatnagar

- Azure and .NET Full Stack Trainer
- Microsoft Certified Trainer Since 2014.
- Freelance Trainer Imparts training in various organizations.
- Client-driven training professional offering 25+ years of progressive experience in training.
- More than 30 corporates & 30000+ Trainees
- Known for outstanding ability to create dynamic, eye-opening, highly interactive, and fun educational experiences for clients that exceed their highest expectations.

Copyright / Fair Use Disclaimers

This presentation may contain copyright material, the use of which may not have been specifically authorized by the copyright owner.

I have sourced these materials from various books, internet sites, in an effort to advance understanding and teaching. Most of the contents are owned, operated, and managed by Microsoft, who may have legal and teaching rights to use published materials (written, photo, video & social media) from the owner(s).

This presentation is created in an effort to explain topics relevant to Microsoft training program or to illustrate the use and benefits of Azure training program.

The material contained in this presentation is distributed without profit for learning and educational purposes. Only fair use, you must obtain permission from the respective copyright owner.

Further use of this presentation as in full or part is restricted.

If you are the owner of any copyrighted material and believe the use of

.NET Core at a Glance

Introduction

What is .NET?

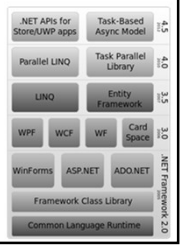
Overview of .NET Framework

Journey from .NET to Core

.NET Architectural Components

What we can Build?

Tools



What is .NET?

- .**NET** is a free, cross-platform, open source **developer platform**, created by Microsoft, for building many different types of applications.
- .**NET** is a developer platform made up of tools, **programming languages**, and **libraries** for building many different types of applications.
- With **.NET**, you can use multiple languages, editors, and libraries to build for web, mobile, desktop, gaming, and IoT.
- You can write .**NET** apps in C#, F#, or Visual Basic.
- .**NET** Framework is used to create and run software applications.
- .**NET** apps can run on many operating systems, using different implementations of .**NET**.
- .**NET** Framework is used for running .**NET** apps on Windows.

A Brief History of .NET

- The original .NET Framework was first released in early 2002.
- After 2002, Microsoft worked to make a version of .NET that had cross-platform compatibility.
- The goal was to allow developers to write one code base and use it across macOS, Linux, and Windows operating systems.
- .NET Core was introduced around 2014. Microsoft has maintained the original .NET Framework. But new features and improvements are reserved for .NET Core.
- Core was later dropped from the name. The next major versions are .NET 5, .NET 6, .NET 7, and so on. Versions are generally released each November.

Presented by Ranjan Bhatnagar

Why Choose .NET?

- **Productive:** .NET helps you develop high quality applications faster. Modern language constructs like generics, Language Integrated Query (LINQ), and asynchronous programming make developers productive.
- **Any app, any platform:** With .NET you can target any application type running on any platform. Developers can reuse skills and code across all of them in a familiar environment.
- **Performance where it matters:** .NET is fast. Really fast! That means applications provide better response times and require less compute power.
- **Trusted and secure:** .NET provides you with immediate security benefits via its managed runtime. A collection of services prevent critical issues like bad pointer manipulation or malicious attempts to alter compiled code. Microsoft takes security very seriously and releases updates quickly when threats are discovered

What is .NET Framework?

- The services that .NET Framework provides to running apps include the following:
  - Memory management
  - A common type system
  - An extensive class library
  - Development frameworks and technologies
    - Language interoperability
    - Version compatibility
    - Side-by-side execution
    - Multitargeting

.NET Framework Architecture

- The two major **components** of .NET Framework are:
  - The **Common Language Runtime (CLR)** is the execution engine that handles running applications. It provides services like thread management, garbage collection, type-safety, exception handling, and more.
  - The **Class Library** provides a set of APIs and types for common functionality. It provides types for strings, dates, numbers, etc. The Class Library includes APIs for reading and writing files, connecting to databases, drawing, and more.
- Code is compiled into a language-agnostic **Common Intermediate Language (CIL)**. Compiled code is stored in **assemblies**—files with a .dll or .exe file extension.
- When an app runs, the CLR takes the assembly and uses a **just-in-time compiler (JIT)** to turn it into machine code that can execute on the specific architecture of the computer it is running on.

Common Language Runtime (CLR)

- The Common Language Runtime(CLR) manages memory, thread execution, code execution, code safety verification, compilation, and other system services.
- These features are intrinsic to the managed code that runs on the common language runtime.
- The runtime also enforces code robustness by implementing a strict type-and-code-verification infrastructure called Common Type System (CTS).
- The CTS ensures that all managed code is self-describing. The various Microsoft and third-party language compilers generate managed code that conforms to the CTS.
- This means that managed code can consume other managed types and instances, while strictly enforcing type fidelity and type safety.

Common Type System (CTS)

- Common Type System & Common Language Specification are two terms that are freely used in the .NET world.
- They actually help us to understand how a .NET implementation enables multi-language development and to understand how it works.
- There has to be a common way to describe all supported types. This is what the Common Type System (CTS) is in charge of doing.
- CTS defines two main kinds of types that should be supported: reference and value types. Their names point to their definitions.
- To enable full interoperability scenarios, all objects that are created in code must rely on some commonality in the languages that are consuming them.
- Since there are numerous different languages, .NET has specified those commonalities in something called the Common Language Specification (CLS).

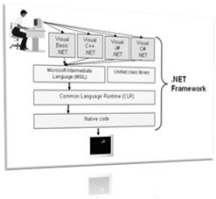
More on CLS

- CLS is a subset of the CTS.
- CLS defines a set of features that are needed by many common applications.
- It also provides a sort of recipe for any language that is implemented on top of .NET on what it needs to support.
- This means that all of the rules in the CTS also apply to the CLS, unless the CLS rules are more strict.
- If a component is built using only the rules in the CLS, that is, it exposes only the CLS features in its API, it is said to be CLS-compliant.
- For instance, the <framework-librares> are CLS-compliant precisely because they need to work across all of the languages that are supported on .NET.

Presented by Ranjan Bhatnagar

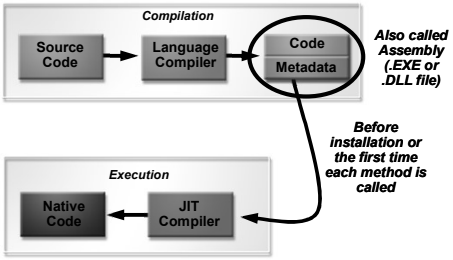
MSIL (Microsoft Intermediate Language)

- AKA Intermediate Language(IL) stands for Intermediate Language.
- All the .NET language such as C#, J#, VB uses their own compiler.
- The compiler compiles the code into IL and after that common runtime engine converts the IL code into native code with the help of the JIT compiler.



- .NET is shipped with a compiler of all programming languages to develop the program.
- All compilers produce an intermediate code after compiling the source code.
- The intermediate code is common for all languages and it is understandable in a .NET environment.
- This intermediate code is known as IL or MSIL.

Code Compilation and Execution



Managed and Unmanaged code

- **Managed Code:** Managed code is the code that executed directly by CLR instead of the operating system.
- The language compiler compiles the managed code to IL or MSIL.
- This code does not depend on the machine configuration and can be executed on a different machine.
- Managed code process is as follows:
- Choose Language Compiler --> Compile to MSIL, MSIL to Native Code, then Execute the code
- **Unmanaged Code:** Unmanaged code is the code that is executed directly by the operating system outside the CLR environment.
- It is directly compiled to native machine code which depends on the machine configuration.
- In unmanaged code, the allocation of memory, Type safety, and security are required to be taken care of by the developer. If unmanaged code is not properly handled it may result in a memory leak.

Assemblies

- When you compile managed code what you get is called as an assembly. (DLL or EXE better known as Portable EXE file)
- Theoretically, assembly can contain multiple modules
- Although Visual Studio supports single-module assemblies
- It is a smallest deployable unit in the CLR
- Have unique version number and no version conflicts (known as DLL hell)
- Contains Compiled IL code to be executed
- Security boundary – permissions are granted at the assembly level
- Type boundary – all types include the assembly name they are a part of
- Self-describing manifest – metadata that describes the types in the assembly

Assembly Manifest

- Assembly manifest stores the assembly metadata. It contains all the metadata needed to do the following things:
  - Version of assembly
  - Security Identity
  - Scope of assembly
  - Resolve reference to the resource of class
  - Assembly manifest contains PE file either .exe or .dll

Type Descriptions
- **Classes**
- **Base classes**
- **Implemented interfaces**
- **Data members**
- **Methods**

Assembly Description
- **Name**
- **Version**
- **Culture**
  - **Other assemblies**
  - **Security Permissions**
  - **Exported Types**

What's Developer Platform?

- An **application** is computer software designed to help a user perform specific tasks.
- To build an application, a **programming language** and a **computing platform** is required.
- Languages such as **C**, **C++**, **C#**, **Java** etc. provides syntax whereas a computing platform includes a hardware architecture and a software framework that allow application software to run.
- Overall a **Developer Platform** is a combination of **Languages and Libraries**.
- **Developer Platforms**, sometimes referred to as APIs and SDKs — but there's much more to them that this, allow you to open up your product to a marketplace of 3rd party developers and companies.

Presented by Ranjan Bhatnagar

.NET – Languages Not a Barrier

- You can write your **.NET apps** in C#, F#, or Visual Basic.
- **C#** is a simple, modern, object-oriented, and type-safe programming language.
- **Visual Basic** is an approachable language with a simple syntax for building type-safe, object-oriented apps.
- **F#** is a cross-platform, open-source, functional programming language for .NET. It also includes object-oriented and imperative programming.

Cross Platform

- Whether you're working in C#, F#, or Visual Basic, your code will run natively on any compatible OS. Below is the list of various **.NET implementations**:
  - **.NET Core** is a cross-platform .NET implementation for websites, servers, and console apps on Windows, Linux, and macOS.
  - **.NET Framework** supports websites, services, desktop apps, and more on Windows.
  - **Xamarin/Mono** is a .NET implementation for running apps on all the major mobile operating systems.
- **.NET Standards**, is a base set of APIs that are common to all .NET implementations.

.NET is open-source

- NET Core is an open-source and cross-platform version of .NET that is maintained by Microsoft and the .NET community on GitHub.
- All aspects of .NET Core are open-source including class libraries, runtime, compilers, languages, ASP.NET Core web framework, Windows desktop frameworks, and Entity Framework Core data access library.
- Microsoft do accept contributions! As with any open-source project they don't just blindly accept everything. The pull requests they receive are reviewed for quality and to ensure it aligns with the goals of .NET.

- They have already accepted contributions from over 60,000 developers and 3,700 companies.
- The various parts of .NET Core are maintained in different GitHub repositories. These repositories typically use the MIT or Apache 2 licenses.

.NET Core Composed of

- The .NET Core runtime, which provides a type system, assembly loading, a garbage collector, native interop and other basic services.
- .NET Core framework libraries provide primitive data types, app composition types and fundamental utilities.
- The ASP.NET Core runtime, which provides a framework for building modern cloud based internet connected applications, such as web apps, IoT apps and mobile backends.
- The .NET Core SDK and language compilers (Roslyn and F#) that enable the .NET Core developer experience.
- The dotnet command, which is used to launch .NET Core apps and CLI tools. It selects and hosts the runtime, provides an assembly loading policy and launches apps and tools.

History – What Next?

Version	Release date	Released with	Latest update	Latest update date	Support ends
.NET Core 1.0	2016-06-27	Visual Studio 2015 Update 3	1.0.16	2019-05-14	June 27, 2019
.NET Core 1.1	2016-11-16	Visual Studio 2017 Version 15.0	1.1.13	2019-05-14	June 27, 2019
.NET Core 2.0	2017-08-14	Visual Studio 2017 Version 15.3	2.0.9	2018-07-10	October 1, 2018
.NET Core 2.1	2018-05-30	Visual Studio 2017 Version 15.7	2.1.29 (LTS)	2021-08-10	August 21, 2021
.NET Core 2.2	2018-12-04	Visual Studio 2019 Version 16.0	2.2.8	2019-11-19	December 23, 2019
.NET Core 3.0	2019-09-23	Visual Studio 2019 Version 16.3	3.0.3	2020-02-18	March 3, 2020
.NET Core 3.1	2019-12-03	Visual Studio 2019 Version 16.4	3.1.19 (LTS)	2021-09-14	December 3, 2022
.NET 5	2020-11-10	Visual Studio 2019 Version 16.8	5.0.10	2021-09-14	3 months after .NET6 release (around February 2022)
.NET 6	2021-11-09		6.0.0 RC 1	2021-09-14	November 2024 (projected)
.NET 7	2022-11 (projected)				February 2025 (projected)
.NET 8	2023-11 (projected)		(will be LTS)		November 2026 (projected)

Latest Update was released on 2021-09-14 SDK 5.0.401

- The next release after .NET Core 3.1 is.NET 5.

.NET 6.0.0-rc.1 is available now

- The .NET Framework will not receive any further major versions, and .NET 5 will be the only .NET meant for new applications going forward – hence the removal of the "Core" branding and skipping of version 4 to avoid confusion with the .NET Framework 4.x. The first preview of .NET 5 was released on March 16, 2020.

.NET Framework is a Windows-only version

- .NET Framework is a Windows-only version of .NET for building any type of app that runs on Windows.

Version	Release date	End of support
.NET Framework 4.8 (recommended)	April 18, 2019	
.NET Framework 4.7.2	April 30, 2018	
.NET Framework 4.7.1	October 17, 2017	
.NET Framework 4.7	April 05, 2017	
.NET Framework 4.6.2	August 02, 2016	
.NET Framework 4.6.1	November 30, 2015	April 26, 2022
.NET Framework 4.6	July 20, 2015	April 26, 2022
.NET Framework 4.5.2	May 05, 2014	April 26, 2022
.NET Framework 3.5 SP1	November 18, 2008	October 10, 2028

Presented by

Ranjan Bhatnagar

.NET 5.0 = .NET Core vNext

- As on November 10, 2020, **.NET 5.0.0** is available for download and usage in your environment.
- This release includes .NET 5.0.0 Runtime and .NET SDK 5.0.100.
- After .NET 5.0, there will be just one .NET.
- You will be able to use it to target Windows, Linux, macOS, iOS, Android, tvOS, watchOS and WebAssembly and more.

.NET Core Characteristics

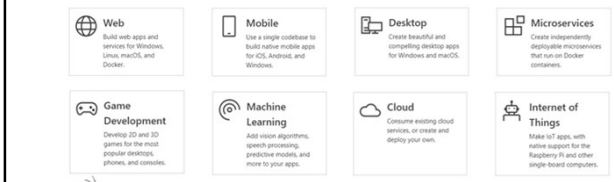
- .NET Core is an open-source, general-purpose development platform maintained by Microsoft and the .NET community on GitHub.
- It's cross-platform (supporting Windows, macOS, and Linux) and can be used to build device, cloud, and IoT applications.
- .NET Core has the following characteristics:
  - Cross-platform
  - Consistent across architectures
  - Command-line tools
  - Flexible deployment
  - Compatible
  - Open source
  - Supported by Microsoft

.NET Core Characteristics at a glance

- **Cross-platform:** Runs on Windows, macOS and Linux operating systems.
- **Consistent across architectures:** Runs your code with the same behavior on multiple architectures, including x64, x86, and ARM.
- **Command-line tools:** Includes easy-to-use command-line tools that can be used for local development and in continuous-integration scenarios.
- **Flexible deployment:** Can be included in your app or installed side-by-side (user-wide or system-wide installations). Can be used with Docker containers.
- **Compatible:** .NET Core is compatible with .NET Framework, Xamarin and Mono, via .NET Standard.
- **Open source:** The .NET Core platform is open source, using MIT and Apache 2 licenses. .NET Core is a .NET Foundation project.
- **Supported by Microsoft:** .NET Core is supported by Microsoft, per .NET Core Support.

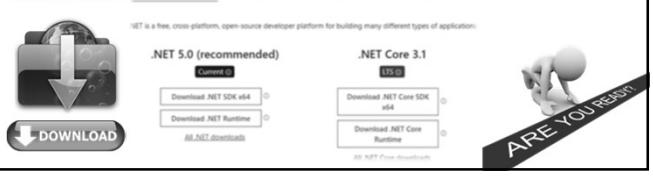
What Can I build?

- You can build many types of apps with .NET. Some are cross-platform, and some target a specific OS or .NET implementation.



Get started

- There are a number of ways to get started with .NET. Because **.NET** is a massive platform.
- While installing **.NET** on you system you can download it from https://dotnet.microsoft.com/download
- Downloads for **.NET Framework** and **.NET Core**, including **ASP.NET** and **ASP.NET Core**



.NET Frameworks v/s .NET Core

.NET Framework	.NET Core
• .NET Framework is the original .NET implementation that has existed since 2002. • Versions 4.5 and later implement .NET Standard, so code that targets .NET Standard can run on those versions of .NET Framework. • It contains additional Windows-specific APIs, such as APIs for Windows desktop development with Windows Forms and WPF. .NET Framework is optimized for building Windows desktop applications.	• .NET Core is a cross-platform implementation of .NET and designed to handle server and cloud workloads at scale. • It runs on Windows, macOS, and Linux. It implements the .NET Standard, so code that targets the .NET Standard can run on .NET Core. • ASP.NET Core, Windows Forms, and Windows Presentation Foundation (WPF) all run on .NET Core.

Presented by Ranjan Bhatnagar

Our Agenda : .NET Core

- Because this course is specifically concentrating over .NET Core we assume that trainees will have idea of .NET architecture which was used for development in .NET framework and .NET Standard.
- I request all to review the .NET architecture.
- You can go through with "**Tour of .NET**" using following url:

  https://docs.microsoft.com/en-us/dotnet/standard/tour
- Also I recommend visiting "**.NET architectural components**" using following url:

  https://docs.microsoft.com/en-us/dotnet/standard/components

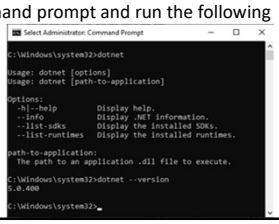
Start Digging

- Let's create a simple application written in C# that prints Hello, World! to the console.
- To start building .NET apps, download and install the .NET SDK (Software Development Kit).
- Once you've installed, open a new command prompt and run the following command:

```
> dotnet
```

- If the command runs, printing out information about how to use dotnet, you're good to go.
- Check version using

```
> dotnet --version
```

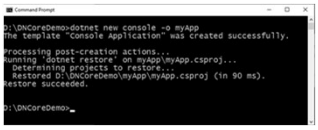


Create your app

- Before creating a console app, create one folder in your working directory as in my case it is: d:\DNCoreDemo
- Now, in your command prompt, run the following commands:

```
> dotnet new console -o myApp
```

- The dotnet command creates a new application of type console for you. The -o parameter creates a directory named myApp where your app is stored, and populates it with the required files.



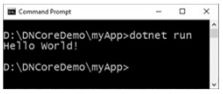
App → Create, Build and Run

- The main file in the myApp folder is Program.cs. By default, it already contains the necessary code to write "Hello World!" to the Console.
- If you want and know the syntax of C#, you can edit this file.
- If so, save changes and follow steps to execute application.
- In your command prompt, run the following command:

```
D:\DNCoreDemo\myApp> dotnet run
```

- You've built and run your first .NET app!

```
using System;
namespace myApp
{
        class Program
        {
                static void Main(string[] args)
                {
                        Console.WriteLine("Hello World!");
                }
        }
}
```

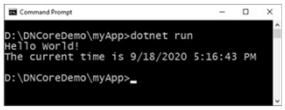


Edit your code

- Open Program.cs in any text editor, such as Notepad, and add a new line of code below the one that prints "Hello World!", like the following:

```
Console.WriteLine("Hello World!");
Console.WriteLine("The current time is " + DateTime.Now);
```

- Save the Program.cs file, and run your code again.



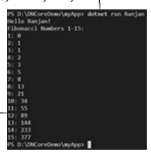
- You can also use Visual Studio, Visual Studio (Windows only), or Visual Studio for Mac (macOS only), to create a .NET Core application.

Command Line Arguments

- Let's change the program a bit. Fibonacci numbers are fun, so let's add that in addition to use the argument to greet the person running the app.
- Replace the contents of your Program.cs file with the following code:

```
using System;
namespace Hello
{
  class Program
  {
    static void Main(string[] args)
    {
      if (args.Length > 0)
      { Console.WriteLine($"Hello {args[0]}!"); }
      else
      { Console.WriteLine("Hello! Anonymous"); }
      Console.WriteLine("Fibonacci Numbers 1-15:");
      for (int i = 0; i < 15; i++)
      {
        Console.WriteLine($"{i + 1}: {FibonacciNumber(i)}");
      }
    }
```

```
    static int FibonacciNumber(int n)
    {
      int b = 1;
      int tmp;
      int a = 0;
      for (int i = 0; i < n; i++)
      {
        tmp = a;
        a = b;
        b += tmp;
      }
      return a;
    }
  }
}
```

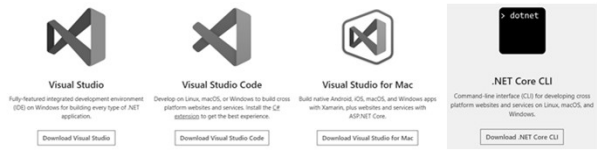


Presented by

Ranjan Bhatnagar

.NET Tools & Editors

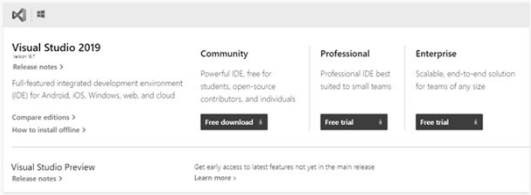
- The Visual Studio product family provides a great .NET development experience on Windows, Linux, and macOS.
- The Visual Studio Marketplace has thousands of editor extensions from Microsoft and others.
- If you prefer to use a different editor, there are .NET command line tools and plugins for many popular editors.



Install Visual Studio

- Visual Studio 2019 provides a full-featured development environment for building .NET Core applications.
- You can install Visual Studio 2019 Community edition from:

https://visualstudio.microsoft.com/downloads/



Hello World using Visual Studio

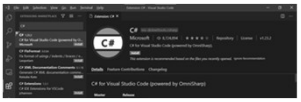
- Let's begin creating a simple "Hello World" console application using Visual Studio, follow these steps:
  - Launch Visual Studio.
  - From dialog box, select the Create a new Project.
  - Select Console App (.NET Core) template.
  - Don't forget to choose C# as language.
  - Give project name
    - for example "MyfirstCoreApp"
  - Select appropriate location
    - as "D:\DNCoreDemo\"
  - System will create project with all required files.
  - The Program.cs file will be opened in an Editor with wrapper Code in it.
  - It is ready to run project.

Using Visual Studio Code

- Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux.
- You can download it from URL:

https://code.visualstudio.com/

- Apart from .NET Core 3.1 SDK or later You must have Visual Studio Code with the C# extension installed.



- VS Code will prompt you to install the extension as soon as you open a C# file.

Create a "Hello World" app

- Launch Visual Studio Code.
- Open a terminal (Ctrl+Shift+`) and create or navigate to the folder, in which you'd like to create the app.
- Enter the following command in the command shell:

```
D:\DNCoreDemo\myApp>dotnet new console
```

- Another option to open existing folder in VS Code, use on command prompt.

```
D:\DNCoreDemo\myApp>code .
```

- When the project folder is first opened in VS Code, a "Would you like to add the required assets to build and debug your project?" notification appears at the top of the window. Select Yes.
- Run the app by entering the following command in the Terminal:

```
D:\DNCoreDemo\myApp> dotnet run
```

What Next?

- Now you are capable to start working with .NET Core environment.
- From here you can choose what type of application you want to develop.
- As a .NET developer you must have a good control over OOPS using C#.
- In coming sections we will be covering:

• C# with .NET Core	• Microservices
• Advanced C# with LINQ	• Introduction to Containers (Docker)
• Entity Framework Core	• Blazer – Client Web Apps
• ASP.NET Core MVC	• Xamarin – Android and iOS
• Web API	• Azure – Cloud Computing
• Unit Testing	• DevOps

You can also learn Apache Spark, ML.NET, IOT, and Game Development to enhance your career.

Presented by

Ranjan Bhatnagar

Presented by

Ranjan Bhatnagar