FPT Education
FPT UNIVERSITY

Microsoft .NET

Introduction to .NET Core Platform and Visual Studio.NET

Objectives

- Overview .NET Framework Architecture
- Overview .NET Core and .NET
- Introduction to Cross-platform application with .NET
- Explain why .NET Core and C# Language is selected as develop application?
- Explain meaning "dotnet CLI”
- Explain about NuGet package
- Demo create and run C# Console Application on Windows, Mac and Linux using “dotnet CLI”

.NET Framework and .NET

The History of .NET Framework and .NET Core

- ***.NET Framework:*** Introduced in 2002, .NET Framework was Microsoft's primary platform for building and running Windows applications. It provided a comprehensive framework for developing various types of applications including desktop, web, and server applications. .NET Framework was tightly coupled with Windows and tied to the Windows operating system.

- ***.NET Core Announcement:*** In 2014, Microsoft announced .NET Core, a modular, open-source, cross-platform version of the .NET Framework. This was a significant departure from the Windows-only focus of the .NET Framework. .NET Core was designed to be lightweight, modular, and able to run on different operating systems, including Windows, macOS, and Linux.

The History of .NET Framework and .NET Core

- ***Initial Releases of .NET Core*** (2016): The first preview release of .NET Core was in 2016. It included a subset of the full .NET Framework functionality but with cross-platform support. Initially, it was targeted mainly for building web applications and services.

- ***.NET Core Becomes .NET 5*** (2020): Microsoft announced that .NET Core 3.0 would be the last release under the ".NET Core" name. Starting with version 3.0, it would simply be called ".NET." Alongside this change, Microsoft announced a new versioning scheme, with the next release being ".NET 5." The decision to skip versions 4.x was to align the platform versions across .NET Framework and .NET Core.

The History of .NET Framework and .NET Core

- Unified Platform with .NET 5 and Beyond: .NET 5, released in November 2020, marked a significant milestone in the unification of the .NET platform. It merged the capabilities of .NET Core, .NET Framework, and Xamarin (which was used for building mobile applications) into a single platform. This consolidation aimed to simplify development and provide a consistent experience across different types of applications.

- .NET 6 and Beyond: .NET 6, released in November 2021, continued the trend of enhancing performance, improving developer productivity, and expanding support for more workloads and platforms. Microsoft's roadmap includes further improvements in areas such as cloud-native development, IoT, AI, and machine learning.
- .NET

The History of .NET Framework and .NET Core

- .NET 8 (Nov 2023) allows developers to build a wide range of applications, including Windows, web, mobile, and console ones.

- Typically, each new version of .NET brings enhancements in performance, features, and tooling to improve the development experience and support a wider range of application scenarios. If .NET 8 has been released, you can expect it to continue this trend of innovation and improvement within the .NET ecosystem.

Introducing .NET Core

- Open-Source: Open-source framework, meaning its source code is freely available and can be modified and distributed by anyone.
- Cross-Platform: Emphasize that .NET Core is designed to run on various operating systems, including Windows, macOS, and Linux.
- Modular Framework: Describe the modular nature of .NET Core, where functionalities are divided into individual NuGet packages.
- Performance and Scalability: Highlight the performance benefits of .NET Core, such as its efficient runtime and optimized libraries.

.NET 8: Enhanced Performance and Productivity

- **Native AOT Compilation:** Introduce Native AOT as a new deployment option for creating self-contained applications with *faster startup times* and *smaller memory footprints*.
- **Container Improvements:** Discuss enhancements for containerized applications, such as improved image size and startup time optimizations.
- **Performance Enhancements:** Briefly mention various performance improvements across the framework, including optimizations in core libraries and runtime.
- **Observability and Diagnostics:** Introduce new tools and APIs for monitoring and diagnosing .NET applications, allowing for better insights into application behavior and troubleshooting.

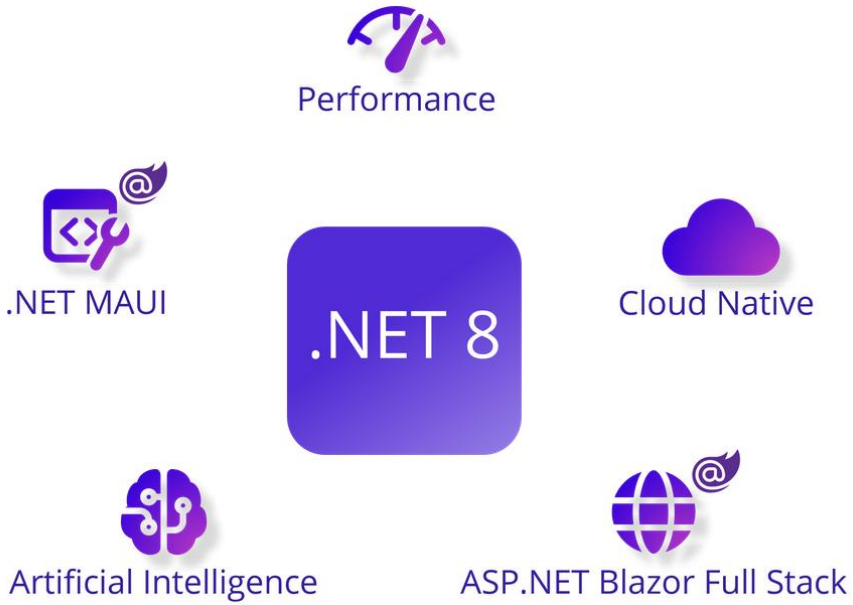
.NET 8: Modern Language Features and APIs

- **C# 12 Enhancements:** Briefly showcase new language features introduced in C# 12, such as primary constructors for non-record types, aliasing any type, and improved interop capabilities.
- **.NET Libraries Updates:** Highlight updates to key .NET libraries.
- **Blazor Unification:** Introduce the new unified Blazor hosting model, allowing developers to choose between different rendering modes (server-side, WebAssembly, or hybrid) on a per-component basis, enhancing flexibility and performance.
- **Minimal APIs Advancements:** Discuss new capabilities in Minimal APIs for building lightweight and efficient web services, including improved route tooling and support for streaming scenarios.

.NET Core Architecture

- **.NET CLI:** Introduce the dotnet command-line interface as the primary tool for managing .NET Core projects and applications. Explain its role in tasks like project creation, building, running, and publishing applications.
- **.NET Runtime:** Explain the role of the .NET runtime environment (CoreCLR) for executing .NET Core applications. Discuss features like just-in-time (JIT) compilation and garbage collection.
- **Base Class Library (BCL):** Describe the BCL as a collection of essential libraries providing fundamental functionalities like file I/O, networking, collections, and more.
- **ASP.NET Core:** Briefly introduce ASP.NET Core as a framework for building web applications and services on top of .NET Core. Mention key components like MVC, Razor Pages, and Blazor.

.NET 8



.NET Aspire

A cloud ready stack for building observable, production ready, distributed applications

Transform your .NET Applications with AI Today

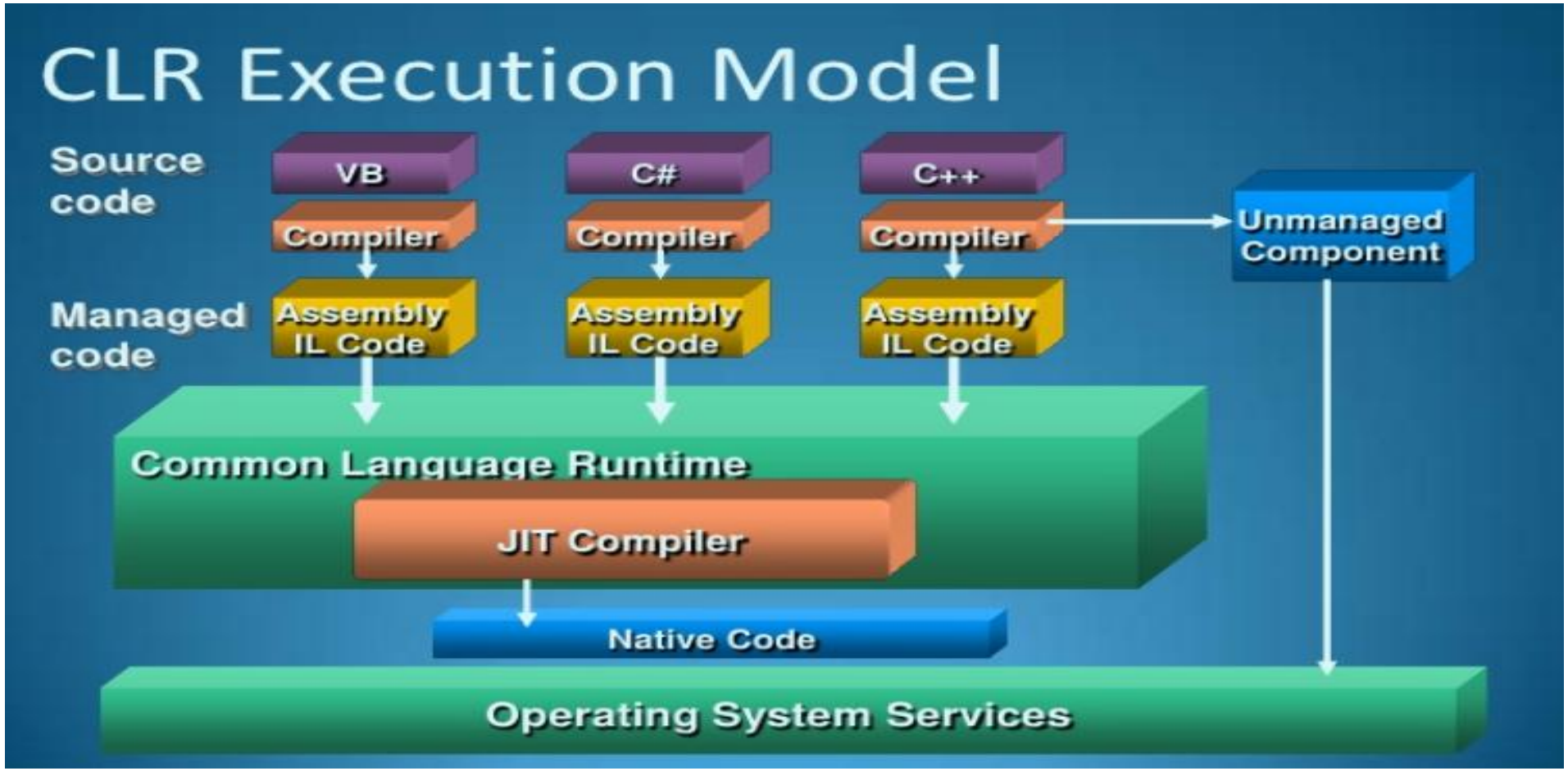
.NET AI

- Out of the box AI features
- Easy to get started with Azure Open AI and Azure Cognitive Search SDKs
- Large Language Model integration with Semantic Kernel
- Fast growing AI ecosystem

Core Common Language Runtime(CoreCLR)

- A common runtime for all .NET languages
  - Common type system
  - Common metadata
  - Intermediate Language (IL) to native code compilers
  - Memory allocation and garbage collection
  - Code execution and security
- Over 15 languages supported today
  - C#, VB, Jscript, Visual C++ from Microsoft
  - Perl, Python, Smalltalk, Cobol, Haskell, Mercury, Eiffel, Oberon, Oz, Pascal, APL, CAML, Scheme, etc.

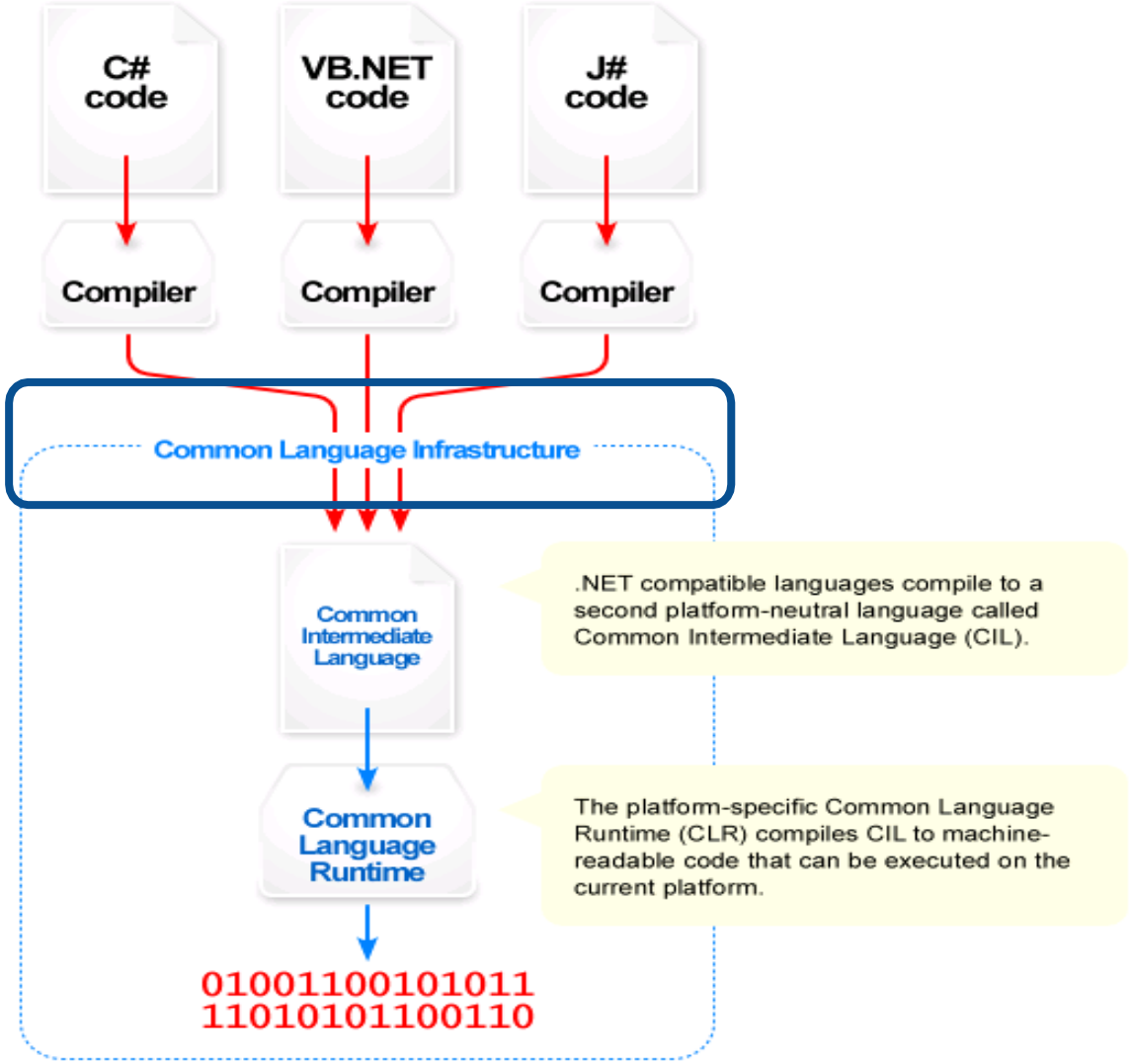
Common Language Runtime (CLR)



Advantages of Core CLR

- Interoperation between managed code and unmanaged code (COM, DLLs)
- Managed code environment
- Improved memory handling
- JIT (**Just-In-Time**) Compiler allows code to run in a protected environment as managed code
- JIT allows the IL code to be hardware independent
- CLR also allows for enforcement of code access security
- Verification of type safety
- Access to Metadata (enhanced Type Information)

Common Language Infrastructure



Cross-Platform Application

"Write once, run anywhere" seems to be the mantra that finds favor with application developers nowadays. This reduces the need for developers to write a lot of redundant code. .NET, an open source offering from Microsoft, is just the tool for writing code for a cross-platform application that will work on Windows, Linux and macOS systems.

Cross-Platform Application

- A platform is a computer hardware and software combination on which a program runs. A platform is a combination of both hardware resources: CPU frequency, RAM size, HDD space, GPU capacity,…and also the software platform being provided to install on such as Operating system; Third-party or extended framework(.NET or JVM,..)
- Cross-platform support runs on multiple platforms. In a sense, it means that a code can run on multiple frameworks, platforms, operating systems, and machine architectures.
- A cross-platform programming language is one that can run on multiple frameworks, operating systems, and machine architectures. Many factors cause the language or tool to be able to run on multiple machines and platforms.

What is the .NET Standard?

- .NET Standard is a specification that can be used across all .NET implementations. It is used for developing library projects only. This means if we are creating a **library** in .NET Standard we can use those in .NET Framework and .NET Core.
- To create uniformity means to allow usage in all the .NET implementations. .NET Standard has support for Mono platform, Xamarin, Universal Windows Platform, and Unity.

Comparisons Table

.NET Core	.NET Framework	.NET Standard
For New Application Development.	For Maintenance of Existing Applications only.	For Developing Library Projects only.
Cross-Platform	Windows Only	Cross-Platform
High Performance	Average Performance	-
Open Source https://github.com/dotnet/core-sdk	Private	Open Source https://github.com/dotnet/standard
CoreCLR and CoreFX	CLR and BCL	-
Visual Studio / Visual Studio Code	Visual Studio	Visual Studio / Visual Studio Code
Free	Free	Free

New features in .NET 8

- Java interoperability will be available on all platforms.
- Objective-C and Swift interoperability will be supported on multiple operating systems.
- CoreFX will be extended to support static compilation of .NET (ahead-of-time – AOT), smaller footprints and support for more operating systems.

Benefits of using .NET

- **Open Source**: Open source and community-oriented on GitHub.
- **Cross-Platform**: .NET Core can run on Windows, Linux, and macOS
- **Command-line tools**: Create, build, and run projects from the command line
- **Modular:** Ships as NuGet packages
- **Host Agnostic:**
  - .NET Core on the server side is not dependent on IIS and, with two lightweight servers: Kestrel and WebListener
  - It can be self-hosted as a Console application and can be also gelled with mature servers such as IIS, Apache, and others through a reverse proxy option

Benefits of using .NET

- Support for leveraging platform-specific capabilities, such as **Windows Forms** and **WPF(Windows Presentation Foundation)** on Windows and the native bindings to each native platform from **Xamarin**.
- High performance.
- Side-by-side installation.
- Small project files (SDK-style).
- Visual Studio, Visual Studio for Mac, and Visual Studio Code integration.

Why C# is selected as develop application?

- C# was developed by Anders Hejlsberg and his team during the development of .NET
- C# is a modern, object-oriented, and type-safe programming language. C# enables developers to build many types of secure and robust applications that run in the .NET ecosystem. C# has its roots in the C family of languages and will be immediately familiar to C, C++, Java, and JavaScript programmers
- C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures

Why C# is selected as develop application?

- The following reasons make C# a widely used professional language
  - It is a modern, general-purpose programming language
  - It is object oriented.
  - It is component oriented.
  - It is easy to learn.
  - It is a structured language.
  - It produces efficient programs.
  - It can be compiled on a variety of computer platforms.
  - It is a part of .Net
- More C# features :

  https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/

Introduction to dotnet CLI

- Download: https://dotnet.microsoft.com/en-us/download/dotnet

- Check SDK versions
  dotnet --list-sdks

- Check runtime versions
  dotnet --list-runtimes

- The global.json file allows you to define which .NET SDK version is used when you run .NET CLI commands.
  dotnet new globaljson

Introduction to dotnet CLI

- The .NET command-line interface(CLI) is a cross-platform for developing, building, running, and publishing .NET applications.
- More dotnet CLI :
  https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet/

```
dotnet new <TEMPLATE> [--dry-run] [--force] [-i|--install {PATH|NUGET_ID}]
    [-lang|--language {"C#"|"F#"|VB}] [-n|--name <OUTPUT_NAME>]
    [--nuget-source <SOURCE>] [-o|--output <OUTPUT_DIRECTORY>]
    [-u|--uninstall] [--update-apply] [--update-check] [Template options]

dotnet new <TEMPLATE> [-l|--list] [--type <TYPE>]

dotnet new -h|--help
```

Introduction to dotnet CLI

Command	Description
new	Initialize .NET projects.
restore	Restore dependencies specified in the .NET project.
build	Builds a .NET project.
publish	Publishes a .NET project for deployment (including the runtime).
run	Compiles and immediately executes a .NET project.
test	Runs unit tests using the test runner specified in the project.
pack	Creates a NuGet package.
migrate	Migrates a project.json based project to a msbuild based project
clean	Clean build output(s).
sln	Modify solution (SLN) files.
Project modification commands	
add	Add items to the project
remove	Remove items from the project
list	List items in the project
NuGet packages	

Introduction to dotnet CLI

Templates	Short name
Console Application	console
Class library	classlib
WPF Application	wpf
Windows Forms (WinForms) Application	winforms
Unit Test Project	mstest
NUnit 3 Test Project	nunit
xUnit Test Project	xunit
Razor Page	page
MVC ViewImports	viewimports

Templates	Short name
ASP.NET Core Empty	web
ASP.NET Core Web App (Model-View-Controller)	mvc
ASP.NET Core Web App	webapp, razor
ASP.NET Core with Angular	angular
ASP.NET Core with React.js	react
ASP.NET Core with React.js and Redux	reactredux
Razor Class Library	razorclasslib
ASP.NET Core Web API	webapi
ASP.NET Core gRPC Service	grpc

Demo Create a C# Console App using dotnet CLI

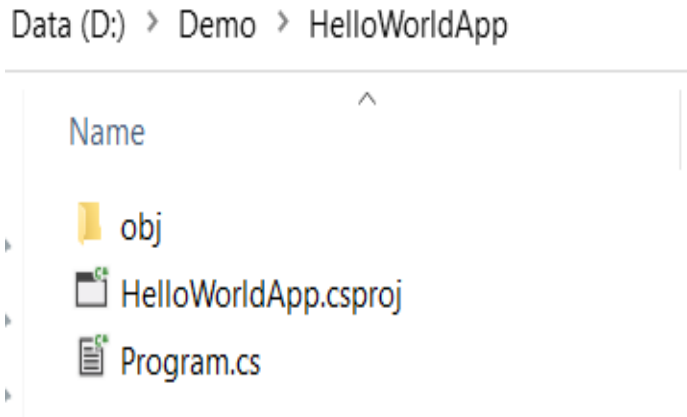
On Windows OS

- Install package: **dotnet-sdk-xxx.exe** and open Command Prompt dialog

1. Create Console App named ***HelloWorldApp*** with C# language

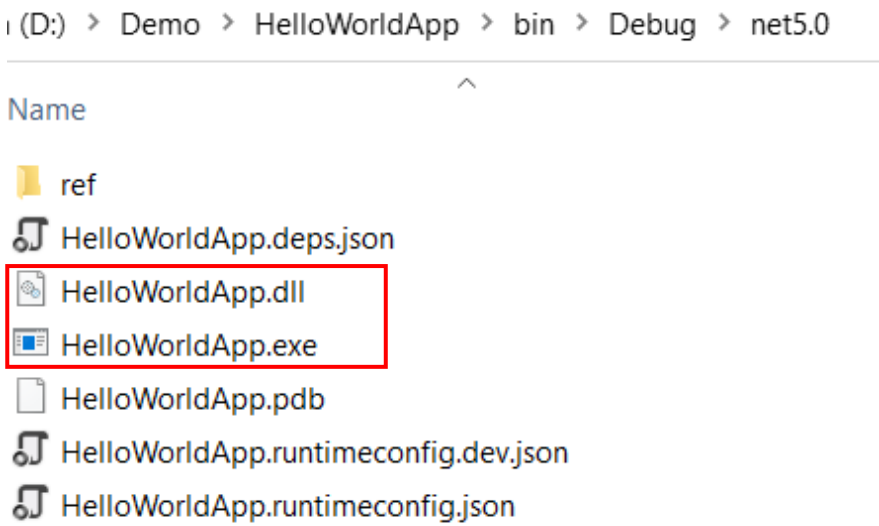
Command Prompt

```
D:\Demo>dotnet new console -lang c# -n HelloWorldApp
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on HelloWorldApp\HelloWorldApp.csproj...
  Determining projects to restore...
  Restored D:\Demo\HelloWorldApp\HelloWorldApp.csproj (in 98 ms).
Restore succeeded.
```



2. Build ***HelloWorldApp*** application

```
D:\Demo>dotnet build HelloWorldApp
Microsoft (R) Build Engine version 16.8.3+39993bd9d for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

  Determining projects to restore...
  All projects are up-to-date for restore.
  HelloWorldApp -> D:\Demo\HelloWorldApp\bin\Debug\net5.0\HelloWorldApp.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:03.16
```



3. Run ***HelloWorldApp*** application

Command Prompt

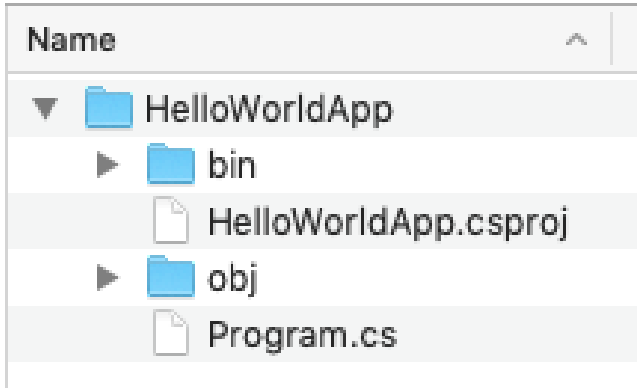
```
D:\Demo>dotnet run -p HelloWorldApp
Hello World!
```

On macOS 10.14 "Mojave"

- Install package: **dotnet** and open **Terminal** dialog

1. Create Console App named ***HelloWorldApp*** with C# language

```
Demo — -bash — 80×16
Swords-Mac:demo swordlake$ dotnet new console -lang C# -n HelloWorldApp
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on HelloWorldApp/HelloWorldApp.csproj...
  Determining projects to restore...
  Restored /Users/swordlake/Documents/KiemHH/Demo/HelloWorldApp/HelloWorldApp.cs
proj (in 122 ms).
Restore succeeded.
```



2. Run ***HelloWorldApp*** application

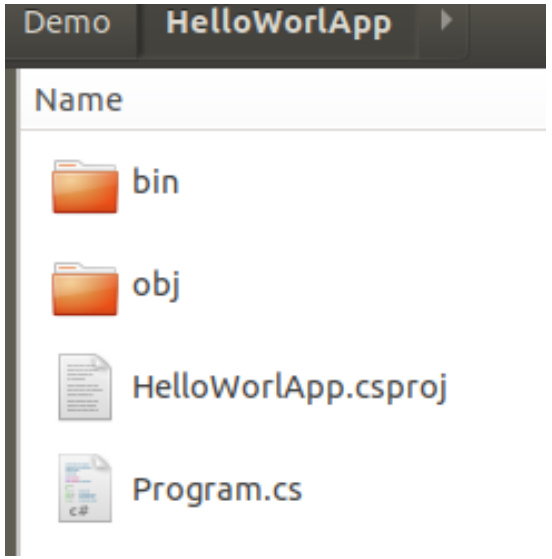
```
Demo — -bash — 80×16
Swords-Mac:demo swordlake$ dotnet run -p HelloWorldApp
Hello World!
Swords-Mac:demo swordlake$
```

On Linux(Ubuntu 14.05) OS

- Install package: **dotnet** and open **Terminal** dialog

1. Create Console App named ***HelloWorldApp*** with C# language

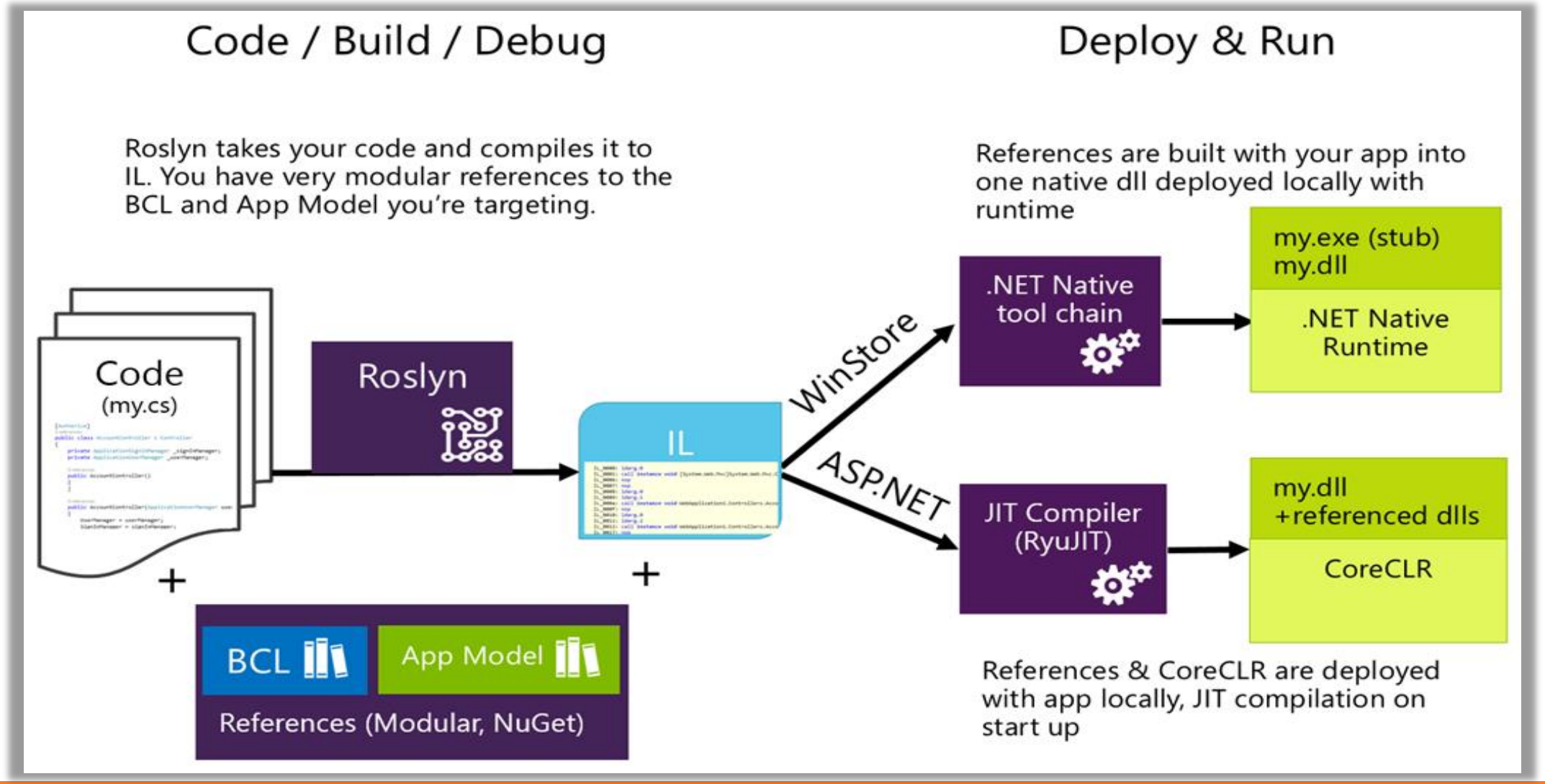
```
ubuntu@ubuntu1804: ~/Demo
File  Edit  View  Search  Terminal  Help
ubuntu@ubuntu1804:~/Demo$ dotnet new console -n HelloWorlApp
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on HelloWorlApp/HelloWorlApp.csproj...
  Determining projects to restore...
  Restored /home/ubuntu/Demo/HelloWorlApp/HelloWorlApp.csproj (in 94 ms).
Restore succeeded.
```



2. Run ***HelloWorldApp*** application

```
ubuntu@ubuntu1804:~/Demo$ dotnet run -p  HelloWorlApp
Hello World!
ubuntu@ubuntu1804:~/Demo$
```

Compilation Process .NET Application



Compilation Process .NET Application

- The compiler used by the dotnet CLI tool converts .NET source code(C#/VB/C++,..) into **Intermediate Language** (IL) code, and stores the IL in an assembly (a **DLL** or **EXE** file).
  - IL code statements are like assembly language instructions, but they are executed by .NET Core's virtual machine, known as the **CoreCLR**.
- At runtime, the **CoreCLR** loads the IL code from the assembly, JIT compiles it into native CPU instructions, and then it is executed by the CPU on your machine.
- The benefit of this two-step compilation process is that Microsoft can create CLRs for Linux and macOS as well as for Windows. The same IL code runs everywhere because of the second compilation process that generates code for the native operating system and CPU ...

Common Intermediate Language (CIL)

- CIL is a platform-neutral intermediate language (formerly called Microsoft Intermediate Language or MSIL) that represents the intermediate language binary instruction set defined by the CLI. It is a stack-based object-oriented assembly language that represents the code in byte-code format

```
using System;

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

C# source code

```
.method private hidebysig static default void Main(string[] args) cil managed
{
  // Method begins at Relative Virtual Address (RVA) 0x2050
  .entrypoint
  // Code size 13 (0xD)
  .maxstack 8
  IL_0000: nop
  IL_0001: ldstr "Hello World!"
  IL_0006: call void class [System.Console]System.Console::WriteLine(string)
  IL_000b: nop
  IL_000c: ret
} // End of method System.Void Create_ConsoleApp_CLI.Program::Main(System.String[])
```

MSIL code

Introduction to Visual Studio.NET

Read by yourself

- Visual Studio is one of the most famous IDE's has been using for the last few years.
- Microsoft developed it. It is used to create a computer program, web applications, and EXE files, etc.
- The first version of its kind was launched in 1997.
- And now the latest version available in the market is Visual Studio 2022.

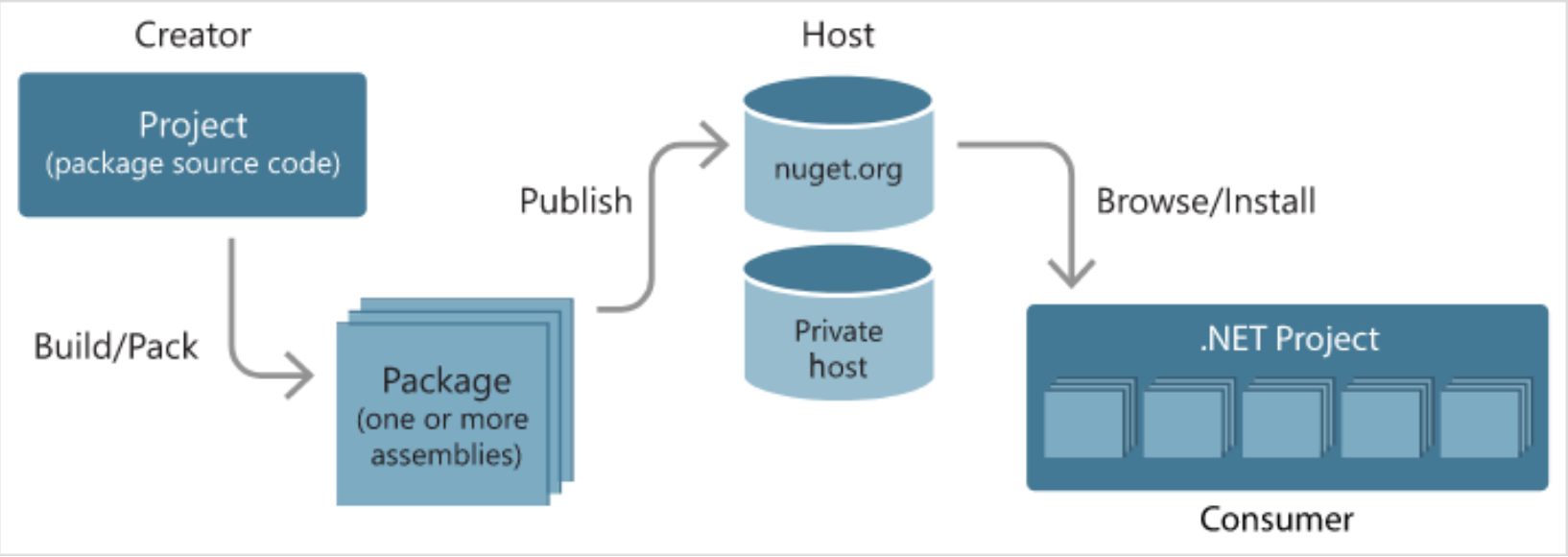
Introduction to Visual Studio.NET

Read by yourself

- **New User Experienced Start Window**: Check out the code, Open a project, Open a folder and Create a new project
- **Visual Studio Live Share**: Live Share is a developer service in Visual Studio 2019. This feature directly enables to share code context and debugging process with your teammates and get live access within Visual Studio itself like Google document services.
- **Improved Refactoring:** Refactoring in any IDE will highly helpful for developers. In Visual Studio 2019 these refactorings will come up with new advanced features, and these are used to organize your code in a structured manner.

Introduction to Nuget packages

- .NET is split into a set of packages, distributed using a Microsoft supported package management technology named NuGet. Each of these packages represents a single assembly of the same name.
  - For example, the System.Collections package contains the System.Collections.dll assembly.



Introduction to Nuget packages

Read by yourself

- Install and use a package for .NET project in Visual Studio
  - Using **NuGet Package Manager**
  - (For **Windows:** https://docs.microsoft.com/en-us/nuget/quickstart/install-and-use-a-package-in-visual-studio)
  - (For **Mac:** https://docs.microsoft.com/en-us/nuget/quickstart/install-and-use-a-package-in-visual-studio-mac)
  - Using the **dotnet CLI**
  - (https://docs.microsoft.com/en-us/nuget/quickstart/install-and-use-a-package-using-the-dotnet-cli)

Summary

- Concepts were introduced:
  - Overview about .NET Core
  - Overview .NET 5 Core Architecture
  - Overview new features of Visual Studio.NET
  - Explain about Cross-platform application with .NET
  - Why .NET Core and C# Language is selected as develop application?
  - Explain and demo using “dotnet CLI” to create C# Console App
  - Overview NuGet package
  - Create and Run cross-platform Console application with C#