Introduction to .NET Core Platform and Visual Studio.NET

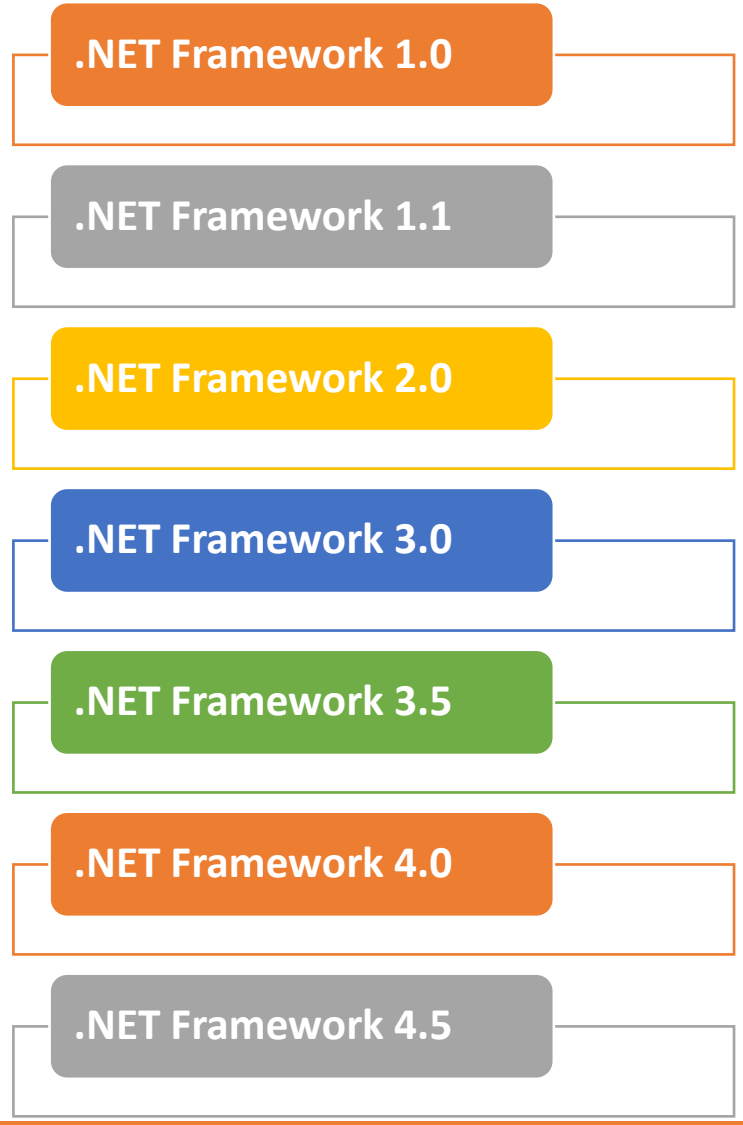
Objectives

- 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”

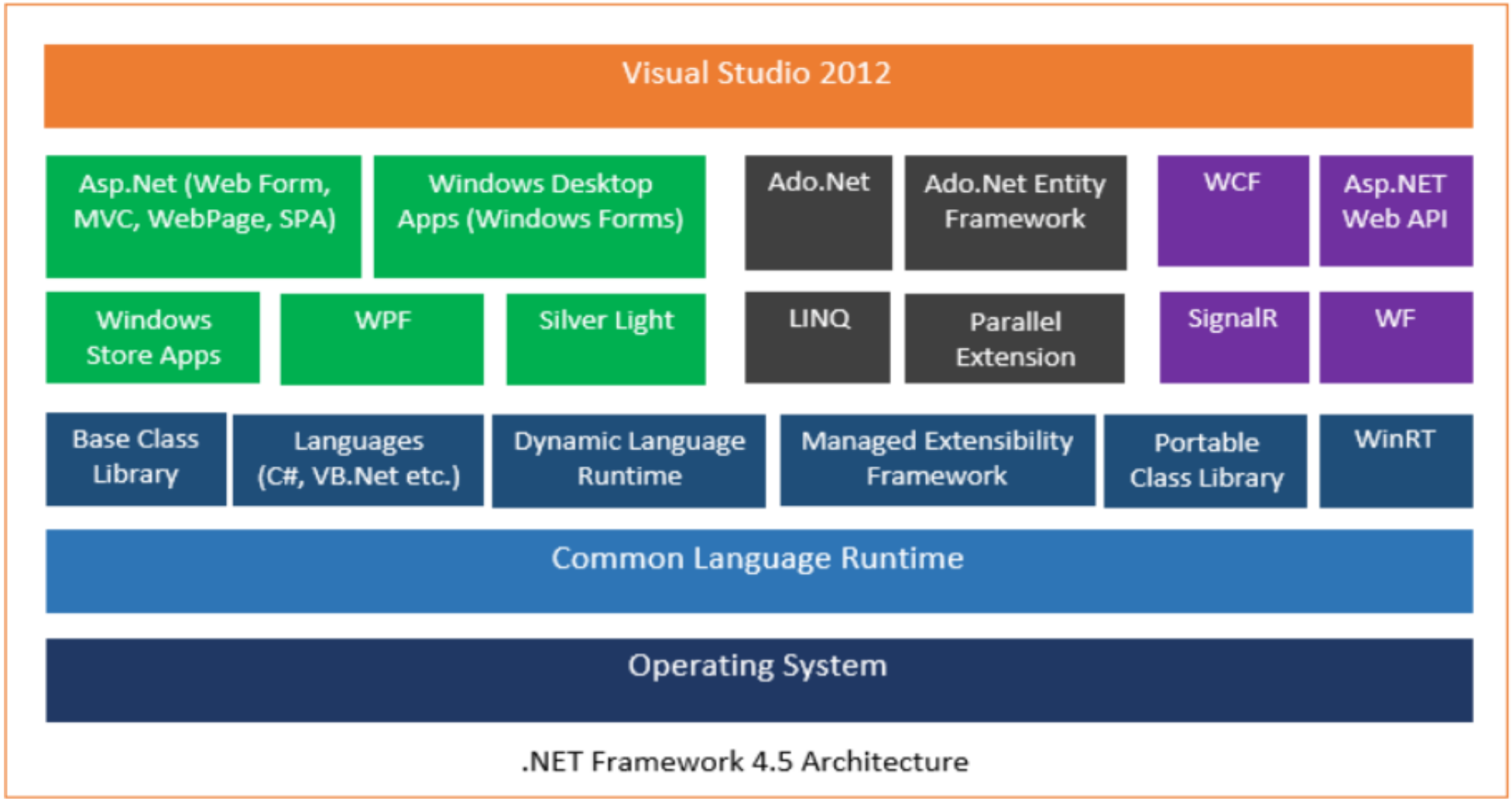
Overview .NET Framework

Introduction to .NET Framework

- Microsoft **.NET Framework 1.0** was released Feb 13, 2002 is the original implementation of .NET
- Developed and run-on Windows platform only
- Closed
- It supports ASP.NET Web Forms, WinForms, WCF, Silverlight, WPF, LINQ, ADO.NET Entity Framework, Parallel LINQ, Task Parallel Library, etc
- The **.NET Framework 4.8** version was released on April 18, 2019



.NET Framework 4.5 Architecture



.NET Framework 4.5 Architecture

The .NET Framework Architecture

- The two core components of the .NET Framework integral to any application or service development are:

Common Language Runtime (CLR)

- Is a backbone of .NET Framework
- Performs various functions such as:
  - Memory management, Code execution
  - Error handling, Code safety verification
  - Garbage collection(GC)

.NET Framework Class Library (FCL)

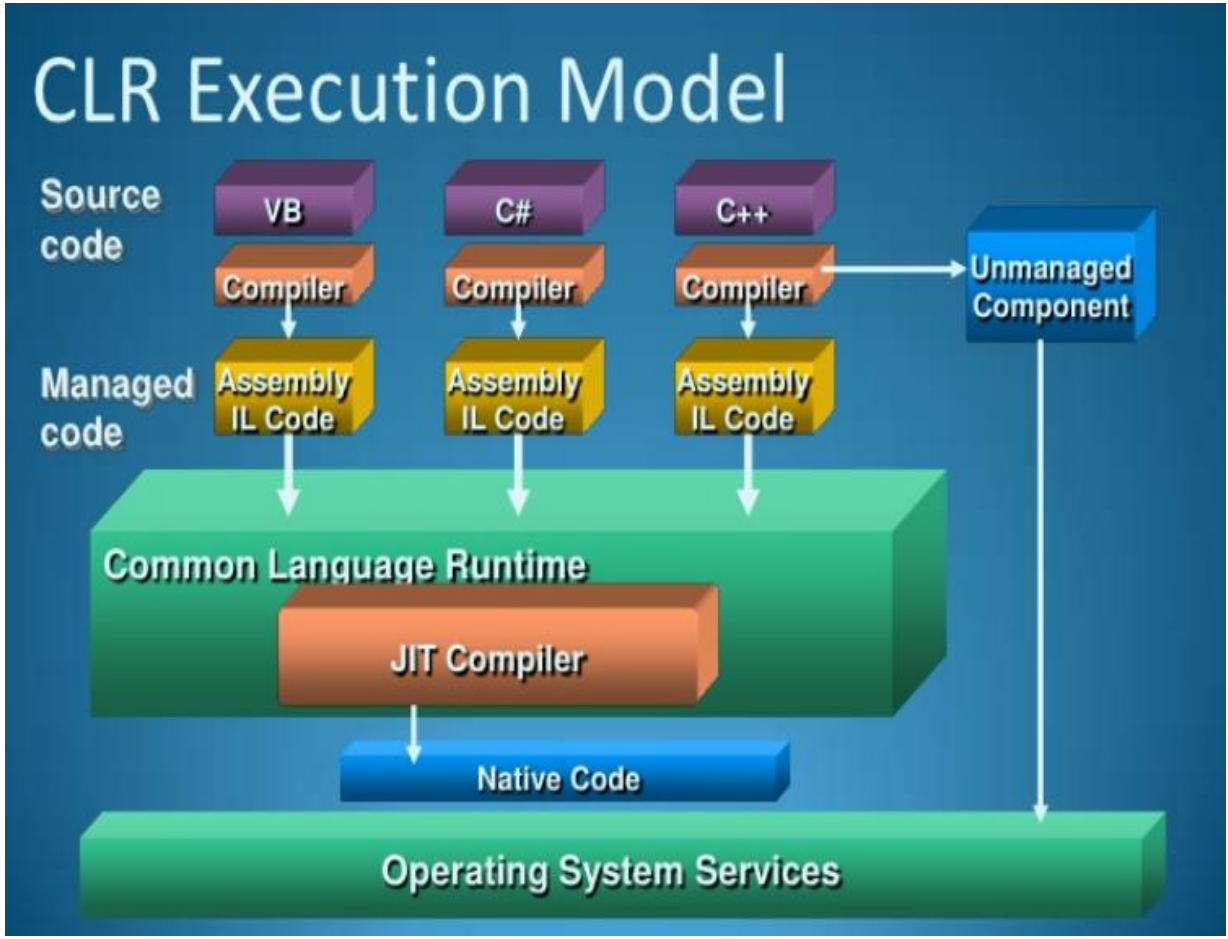
- Is a comprehensive object-oriented collection of reusable types.
- Used to develop applications ranging from traditional command-line to GUI applications that can be used on the Web.

Common Language Runtime(CLR)

- 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 20 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)

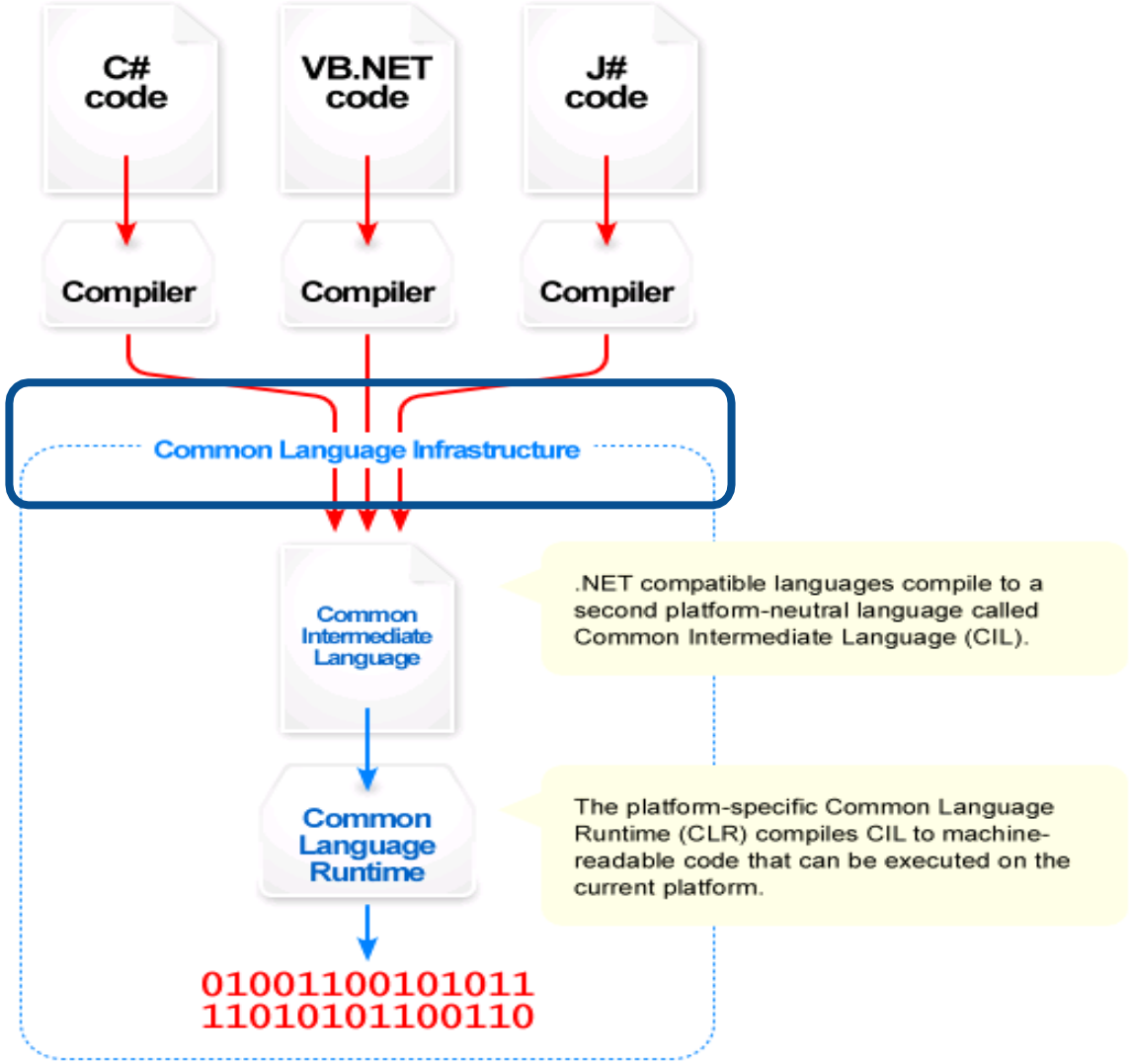
- Using the .NET Framework:
  - The code of a program is compiled into CIL (formerly called MSIL) and stored in a file called assembly
  - This assembly is then compiled by the CLR to the native code at run-time
- In traditional Windows applications:
  - Codes were directly compiled into the executable native code of the operating system



Advantages of 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



Common Language Infrastructure

- CLI allows for cross-language development
- Four components:
  - Common Type System (CTS)
  - Meta-data in a language agnostic fashion
  - Common Language Specification – behaviors that all languages need to follow
  - A Virtual Execution System (VES)

Common Type System (CTS)

- The common type system defines how types are declared, used, and managed in the common language runtime, and is also an important part of the runtime's support for cross-language integration. The common type system performs the following functions:
  - Establishes a framework that helps enable cross-language integration, type safety, and high-performance code execution
  - Provides an object-oriented model that supports the complete implementation of many programming languages
  - Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other
  - Provides a library that contains the primitive data types (such as Boolean, Byte, Char, Int32, and UInt64) used in application development

Common Type System (CTS)

- The common type system in .NET supports the following five categories of types:
  - Classes
  - Structures
  - Enumerations
  - Interfaces
  - Delegates

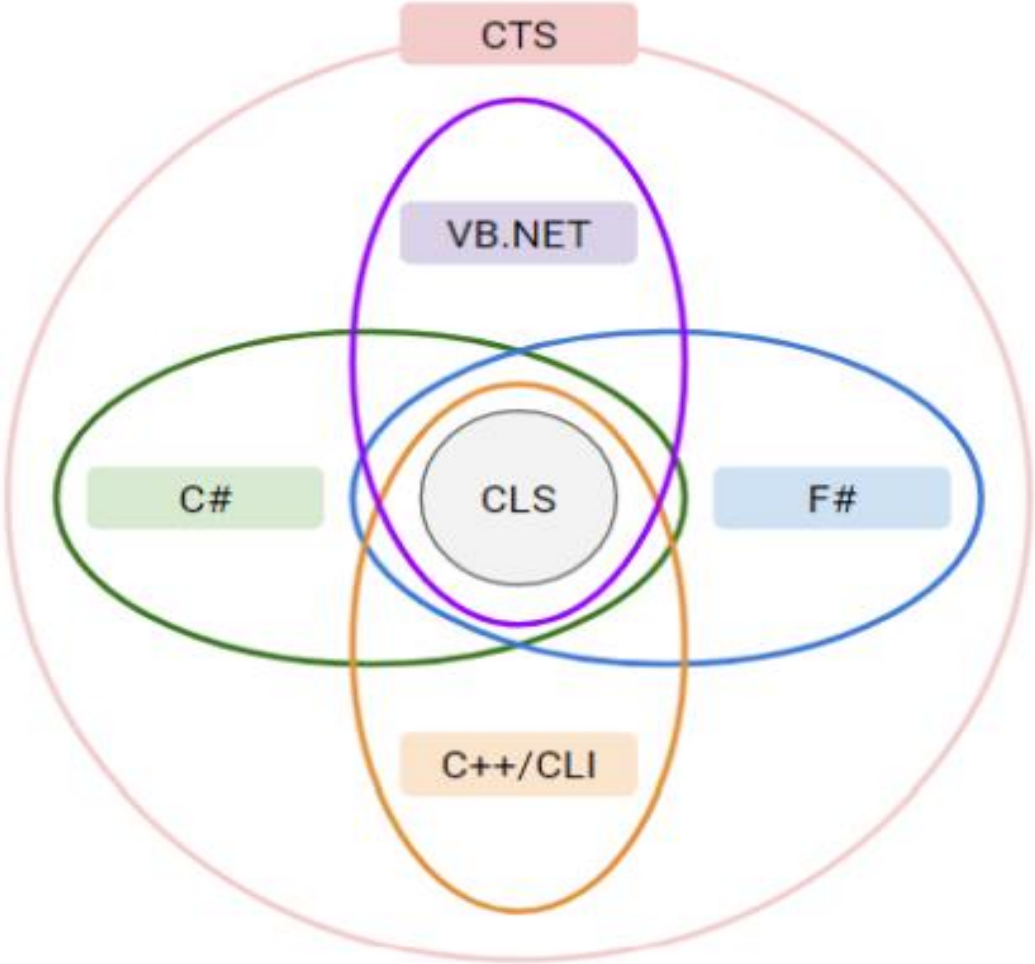
CTS Data Types

CTS Data Type	VB .NET Keyword	C# Keyword	Managed Extensions for C++ Keyword
System.Byte	Byte	byte	unsigned char
System.SByte	SByte	sbyte	signed char
System.Int16	Short	short	short
System.Int32	Integer	int	int or long
System.Int64	Long	long	__int64
System.UInt16	UShort	ushort	unsigned short
System.UInt32	UInteger	uint	unsigned int or unsigned long
System.UInt64	ULong	ulong	unsigned __int64
System.Single	Single	float	Float
System.Double	Double	double	Double
System.Object	Object	object	Object^
System.Char	Char	char	wchar_t
System.String	String	string	String^
System.Decimal	Decimal	decimal	Decimal
System.Boolean	Boolean	bool	Bool

The .NET Framework Architecture

- **Common Language Specification (CLS)**
  - The CLS comprises a set of rules that any language that targets the CLI needs to adhere to, to be able to interoperate with other CLS-compliant languages
  - CLS rules fall into the broader rules of the CTS and therefore it can be said that the CLS is a subset of CTS
  - Language constructs that make it impossible to easily verify the type safety of the code were excluded from the CLS so that all languages that work with the CLS can produce verifiable code

The .NET Framework Architecture



The relationship between the CTS and CLS

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

Overview .NET Core

What is the .NET Core?

- It is a cross-platform, open-source framework that implements .NET Standard
- It includes **JIT(Just-In-Time) Compiler**, **GC(Garbage Collection)** and several low-level classes
- It provides a runtime known as **.NET Core CLR**, framework class libraries, which are primitive libraries known as **CoreFX**, and APIs that are similar to **CLR** and **BCL(Base Class Library)** of **.NET Framework**, but have a smaller footprint (lesser dependencies on other assemblies)

What is the .NET Core?

- CoreFX is made of foundation class libraries. These come as alternatives to CLR and BCL of .NET Framework. It comes integrated with .NET Core CLI
- It supports modern application frameworks such as gRPC, ML.NET for Machine Learning, ASP.NET Core Razor Pages, Blazor (for WebAssembly), UWP(Universal Windows Platform), etc
- The **.NET Core 5(.NET)** version was released November 10, 2020

.NET 5 (.NET) = .NET Core vNext

- Released on November 10, 2020 (Visual Studio 2019 and C# 9.0)

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 5 (.NET)

- 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

.NET components

- **Language compilers:** These turn source code written with languages such as C#, F#, and Visual Basic into intermediate language (IL) code stored in assemblies. NET language compilers for C# and Visual Basic, also known as **Roslyn**
- **Common Language Runtime (CoreCLR):** This runtime loads assemblies, compiles the IL code stored in them into native code instructions for computer's CPU, and executes the code within an environment that manages resources such as threads and memory
- **Base Class Libraries (BCLs)** of assemblies in NuGet packages (CoreFX): These are prebuilt assemblies of types packaged and distributed using NuGet for performing common tasks when building applications

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

- 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

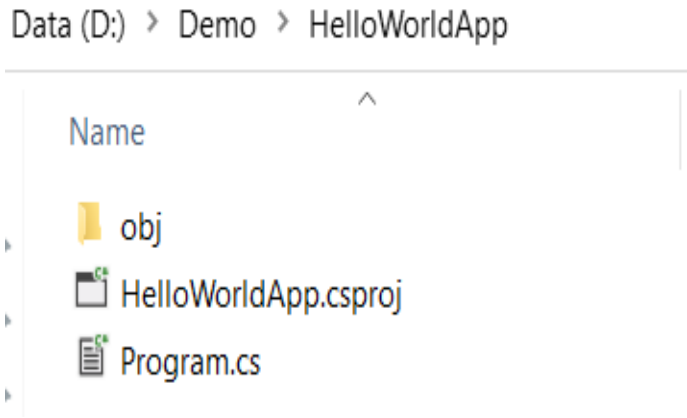
On Windows OS

- Install package: **dotnet-sdk-5.0.102-win-x64.exe** and open Command Prompt dialog

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

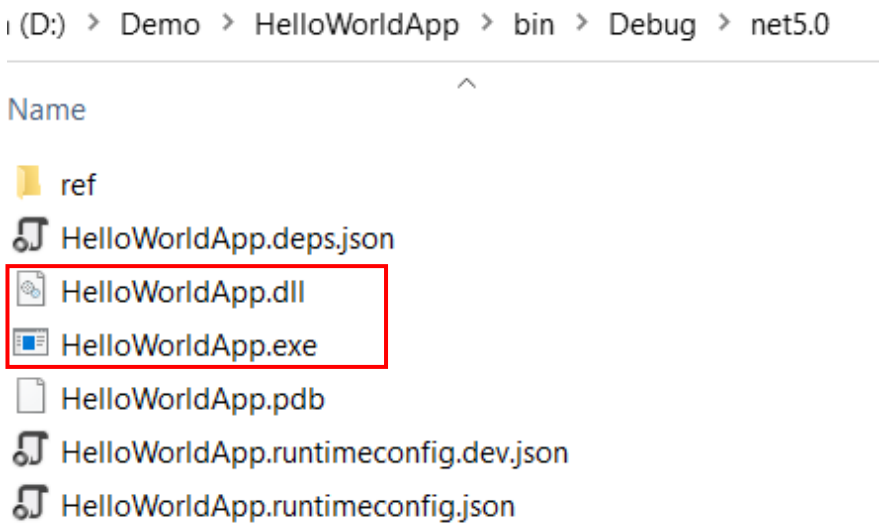
Command Prompt

```
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

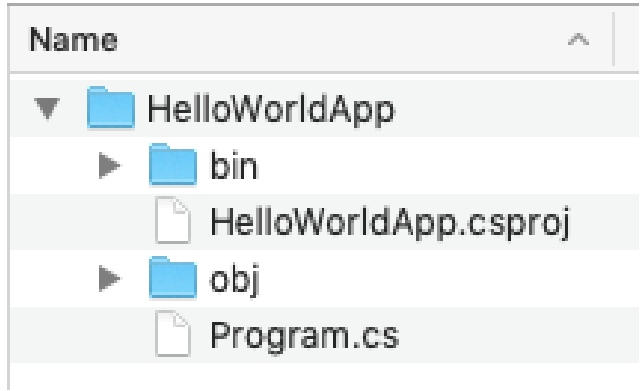
```
D:\Demo>dotnet run -p HelloWorldApp
Hello World!
```

On macOS 10.14 "Mojave"

- Install package: **dotnet-sdk-5.0.102-osx-x64.pkg** 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

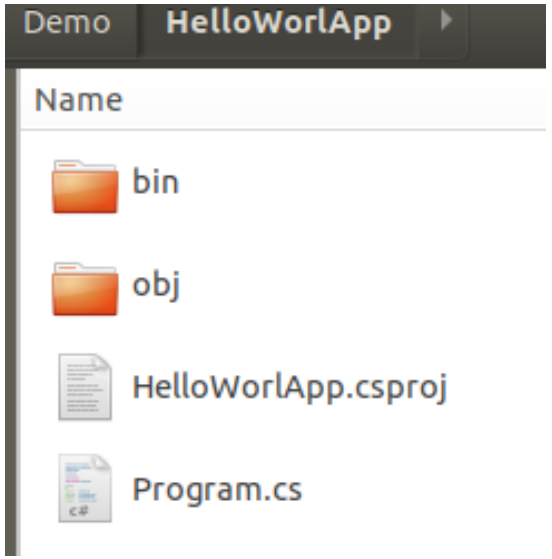
```
Demo — -bash — 80×16
Swords-Mac:demo swordlake$ dotnet run -p HelloWorldApp
Hello World!
Swords-Mac:demo swordlake$
```

On Linux(Ubuntu 14.05) OS

- Install package: **dotnet-sdk-5.0** and open **Terminal** dialog

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

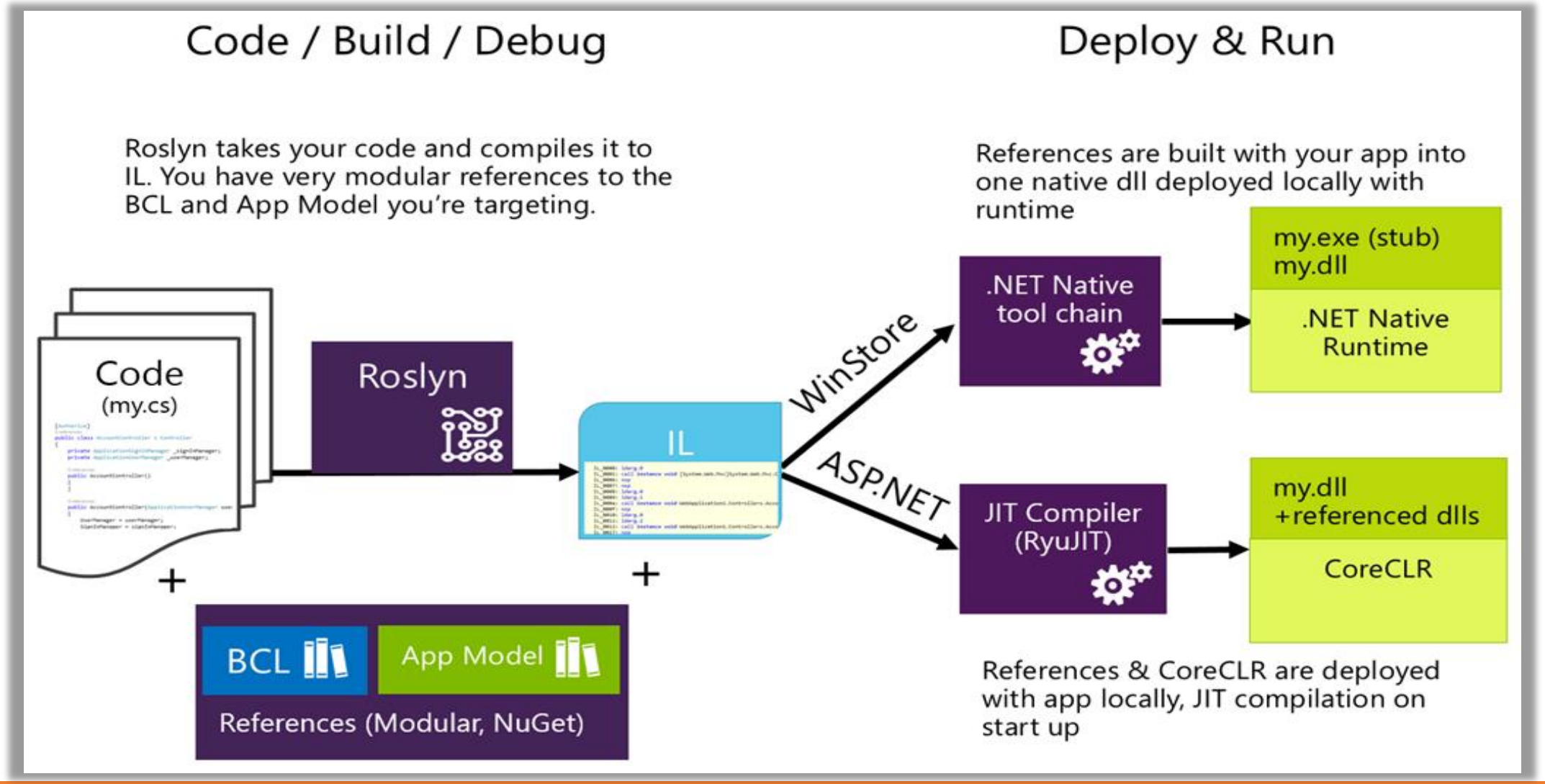
```
ubuntu@ubuntu1804: ~/Demo
File  Edit  View  Search  Terminal  Help
ubuntu@ubuntu1804:~/Demo$ dotnet new console -n HelloWorlApp
The template "Console Application" was created successfully.

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, etc

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

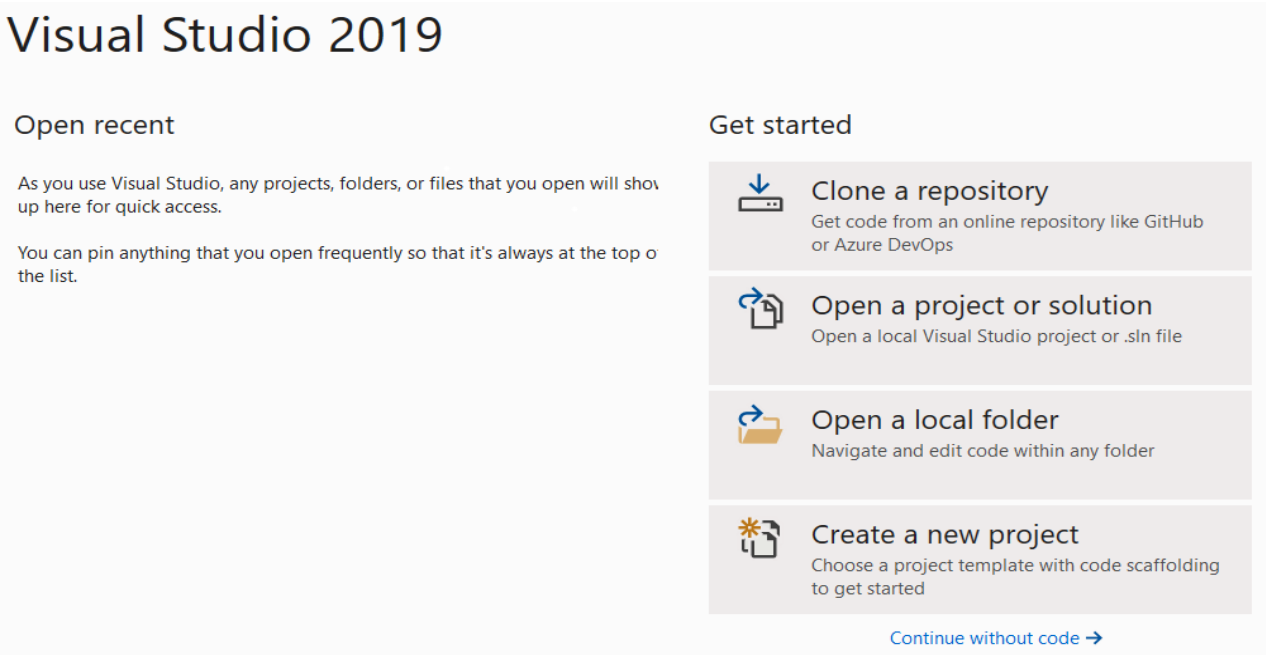
```
.method private hidebysig static default void Main(string[] args) cil managed
{
  // Method begins at Relative Virtual Address (RVA) 0x2050
  .entrypoint
  // Code size 13 (0xD)
  .maxstack 8
  IL_0000: nop
  IL_0001: ldstr "Hello World!"
  IL_0006: call void class [System.Console]System.Console::WriteLine(string)
  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 2019



Introduction to Visual Studio.NET

Read by yourself

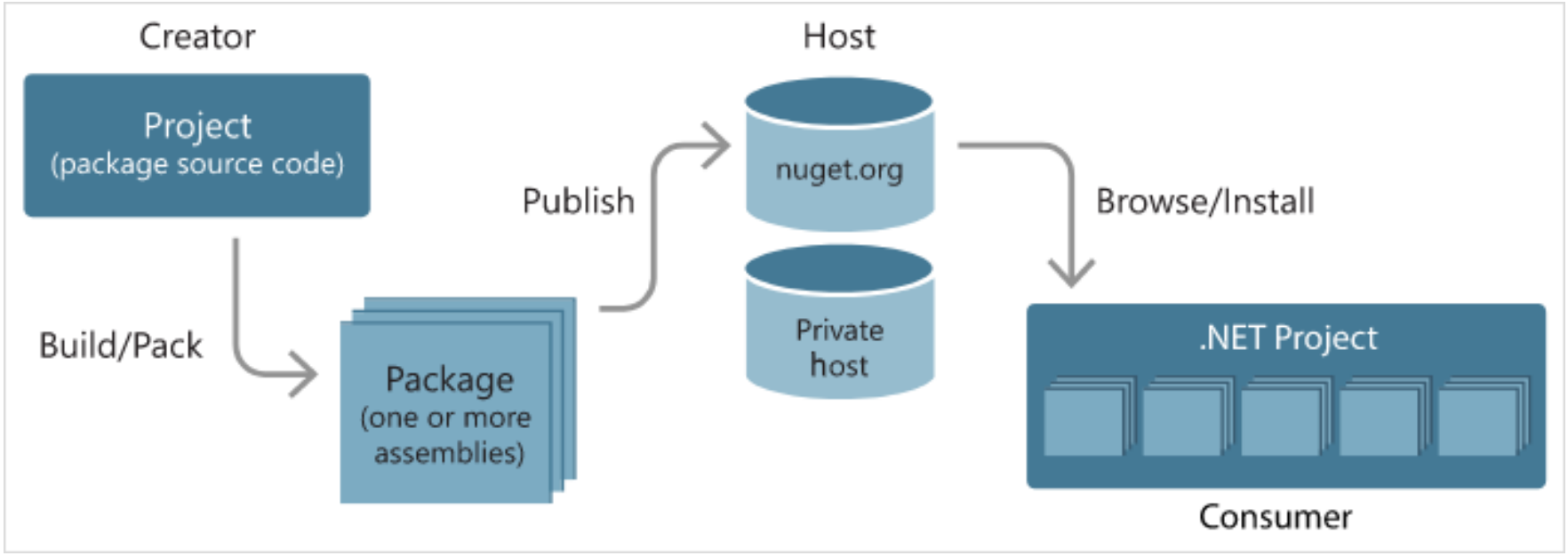
- **New User Experienced Start Window**: Check out the code, Open a project, Open a folder and Create a new project
- **Visual Studio Live Share**: Live Share is a developer service in Visual Studio 2019. This feature directly enables to share code context and debugging process with your teammates and 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 Visual Studio.NET

- Enhanced Search Experience
- Search Feature While in Debugging
- Visual Studio IntelliCode
- Code cleanup in One Click
- Integrated Code Reviews in Development
- Per Monitor Aware Rendering(PMA)
- New Delivery Model for SQL Server Data Tools

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, .NET 5(.NET) and .NET Framework
  - Overview .NET Framework and .NET 5(.NET) 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#