

INTRODUCCION A LA PROGRAMACION C#

1 - INTRODUCCIÓN A .NET CORE

UNIDAD: 1

PRESENTACIÓN: En esta unidad se explicarán los fundamentos y conceptos sobre los cuales está construido el framework de desarrollo .NET Core

OBJETIVOS

Que los participantes logren: Comprender los fundamentos de Microsoft .NET Core y el contexto de trabajo para un desarrollador que se inicia en C#.

TEMARIO

.NET Framework	4
Contexto histórico	4
Evolución	4
Que es .NET Core ?	5
Disponibilidad:	5
Estructura de .NET Core	6
El proceso de Compilación y Ejecución en .NET Core	7
Instalando .NET Core	9
Instalar Visual Studio 2019	9
Estructura de una aplicación .NET Core	9
Historial de Versiones de .NET Core	15
Futuro de .NET Core	15

.NET Framework

NET Framework es un entorno de ejecución runtime que administra aplicaciones cuyo destino es .NET Framework. Incorpora Common Language Runtime, que proporciona la administración de la memoria y otros servicios del sistema, y una biblioteca de clases completa, que permite a los programadores aprovechar el código estable y fiable de todas las áreas principales del desarrollo de aplicaciones

Contexto histórico

Lanzamiento Inicial: 13 de febrero de 2002

.NET podría considerarse una respuesta de Microsoft al creciente mercado de los negocios en entornos Web, como competencia a la plataforma Java de Oracle Corporation y a los diversos framework de desarrollo web basados en PHP.

Su propuesta es ofrecer una manera rápida y económica, a la vez que segura y robusta, de desarrollar aplicaciones –o como la misma plataforma las denomina, soluciones– permitiendo una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

Evolución

Año	Versión
2002	1.0
2005	2.0
2006	3.0
2007	3.5
2010	4.0
2012	4.5
2015	4.6
2017	4.7
2018	4.7.2

Mas informacion en: <https://docs.microsoft.com/en-us/dotnet/framework/migration-guide/versions-and-dependencies>

Que es .NET Core ?

.NET Core es la evolución natural de .NET.

Además de ser de código abierto, abarca no sólo sistemas operativos Windows, sino también Linux y Mac OS (Cross-platform).

Podemos desarrollar una aplicación en .NET Core, y ejecutarla en distintos sistemas operativos (Windows, Linux, Mac OS).

El lenguaje de natural de desarrollo para .NET Core es C#, favoreciendo a los desarrolladores .NET en su transición a .NET Core. Aunque también se puede desarrollar con Visual Basic y F#.

.NET Core es open source: <https://github.com/dotnet/core>

Disponibilidad:

.NET Framework está disponible para los sistemas operativos Windows. .NET Core es multiplataforma.

Con .NET Core se pueden desarrollar aplicaciones web, microservicios, bibliotecas y aplicaciones de consola que se ejecutan en diversas plataformas.

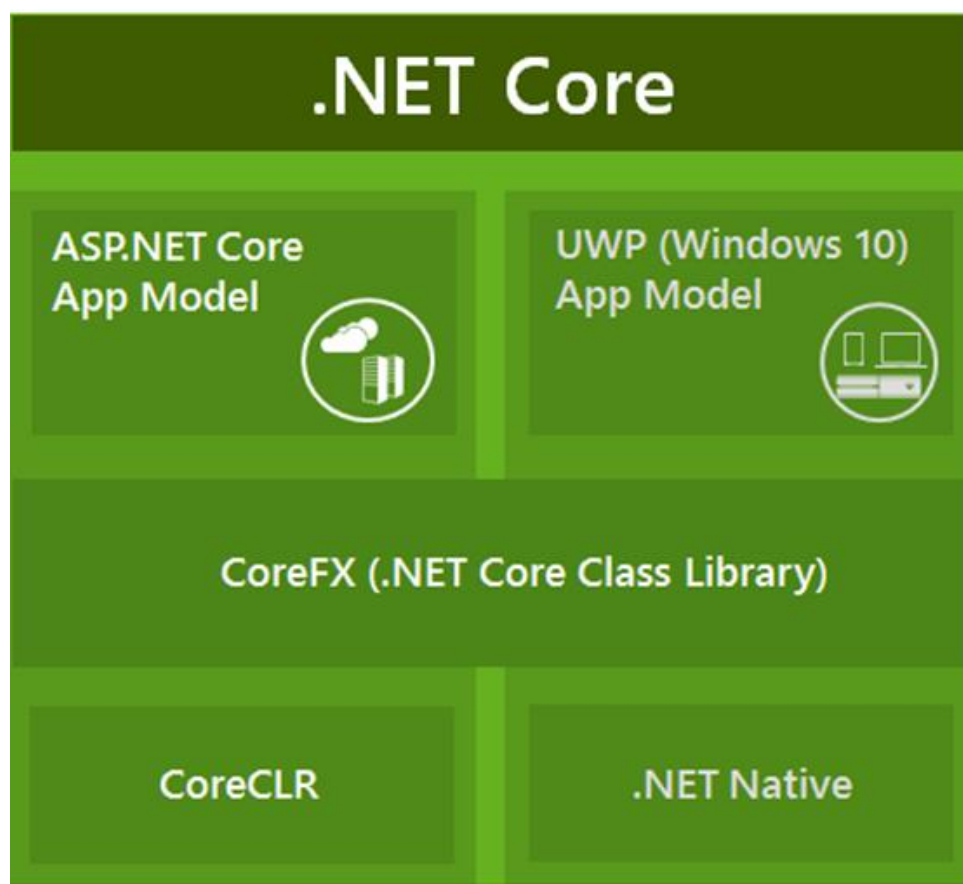
Estructura de .NET Core

A diferencia de .NET Framework tradicional, que es un solo paquete de instalación y es parte del sistema operativo Windows, .NET Core Platform se empaqueta e instala de una manera diferente.

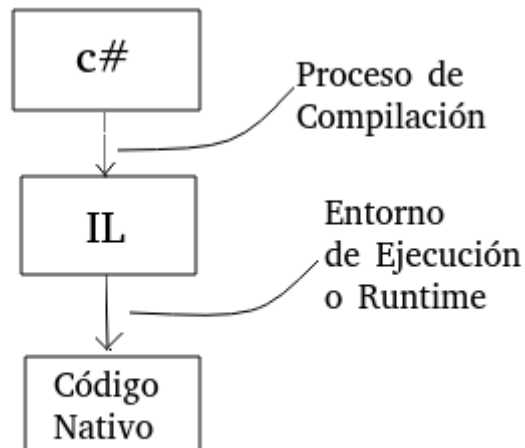
.NET Core se compone de paquetes NuGet y se compila directamente en una aplicación o se coloca en una carpeta dentro de la aplicación. Esto significa que las aplicaciones pueden llevar .NET Core dentro de ellas.

.NET Core, fundamentalmente, consiste de:

- Un lenguaje de tiempo de ejecución: CoreCLR.
- CoreFX, una colección modular de bibliotecas.
- Un compilador de Código Fuente a un Código Intermedio



El proceso de Compilación y Ejecución en .NET Core



- Los desarrolladores escriben el código fuente en un lenguaje compatible con .NET Core. Como C# , F# o Visual Basic .NET.
- En tiempo de compilación, un compilador .NET Core convierte el código a IL.
- En tiempo de ejecución, el Core CLR convierte el código IL en código nativo para el sistema operativo por medio de un compilador JIT.

Compilador

El compilador de C# a Código intermedio, se llama Roslyn

CoreCLR

El Common Language Runtime o CLR ("entorno en tiempo de ejecución de lenguaje común") de .NET Core es un entorno de ejecución para los códigos de los programas que corren en la plataforma .NET Core.

Core CLR es el **encargado de compilar** una forma de código intermedio llamado Intermediate Language IL, al código de máquina nativo, mediante un compilador en tiempo de ejecución. Esto lo hace por medio de un compilador JIT denominado RyuJIT

La existencia de un CLR permite a los programadores ignorar muchos detalles específicos del microprocesador que estará ejecutando el programa. El CLR también permite otros servicios importantes, incluyendo los siguientes:

- Administración de la memoria
- Administración de hilos
- Manejo de excepciones
- Recolección de basura
- Seguridad

CoreFX

Es la biblioteca de clases base de .NET Core.

Basicamente es la reimplementacion de la libreria de clases de .NET Framework.

Su fin es brindar herramientas al desarrollador para manejar las operaciones básicas que se encuentran involucradas en el desarrollo de aplicaciones.

Instrucciones IL

El código intermedio de IL incluye un conjunto de instrucciones para las siguientes grupos de tareas:

- Carga y almacenamiento
- Aritméticas
- Conversión de tipos
- Creación y manipulación de objetos
- Operadores de pila (push / pop)
- Transferencia de control (saltos)
- Invocación y retorno de métodos
- Manejo de excepciones
- Concurrencia

Modularidad

.NET Core está construido con un diseño modular, lo que permite que las aplicaciones incluyan solo las bibliotecas y las dependencias de .NET Core que son necesarias, evitando conflictos con los componentes compartidos.

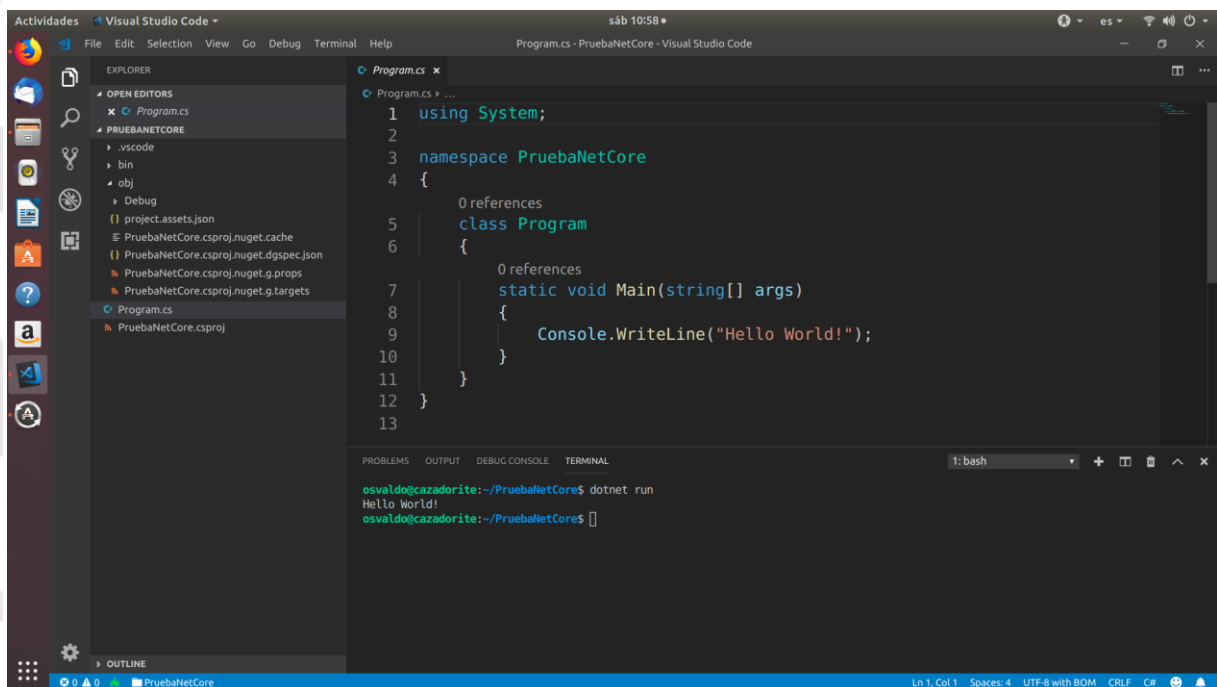
Instalando .NET Core

Vamos a guiarnos con la [Documentación oficial de Microsoft](#)

[Instalar VS Code](#)

[Instalar Visual Studio 2019](#)

Estructura de una aplicación .NET Core

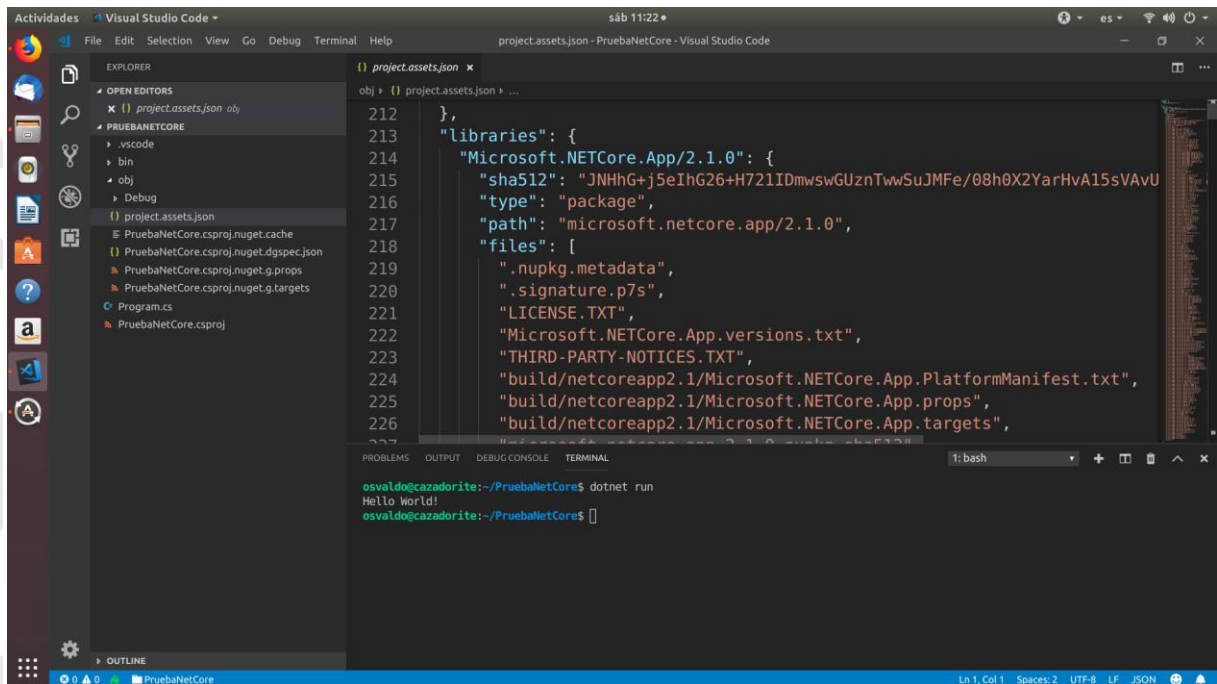


The screenshot shows the Visual Studio Code interface with a project named 'PruebaNetCore'. The Explorer pane on the left shows the project structure, including files like 'project.assets.json', 'PruebaNetCore.csproj.nuget.cache', and 'Program.cs'. The main editor displays the code in 'Program.cs', which is a simple console application. The code includes the 'System' namespace, defines a 'Program' class, and has a 'Main' method that writes 'Hello World!' to the console. The bottom panel shows the terminal output, where the command 'dotnet run' has been executed, resulting in 'Hello World!' being printed.

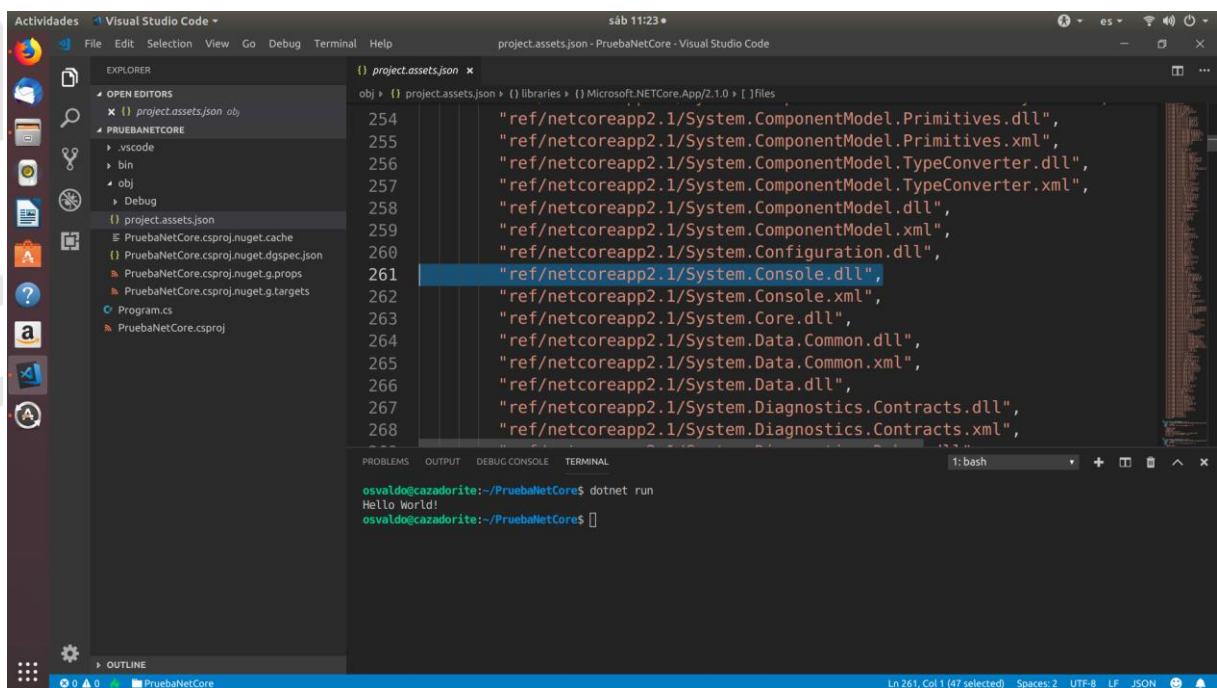
```
1 using System;
2
3 namespace PruebaNetCore
4 {
5     0 references
6     class Program
7     {
8         0 references
9         static void Main(string[] args)
10        {
11            Console.WriteLine("Hello World!");
12        }
13 }
```

```
osvaldo@cazadorite:~/PruebaNetCore$ dotnet run
Hello World!
osvaldo@cazadorite:~/PruebaNetCore$
```

Donde están las referencias ?



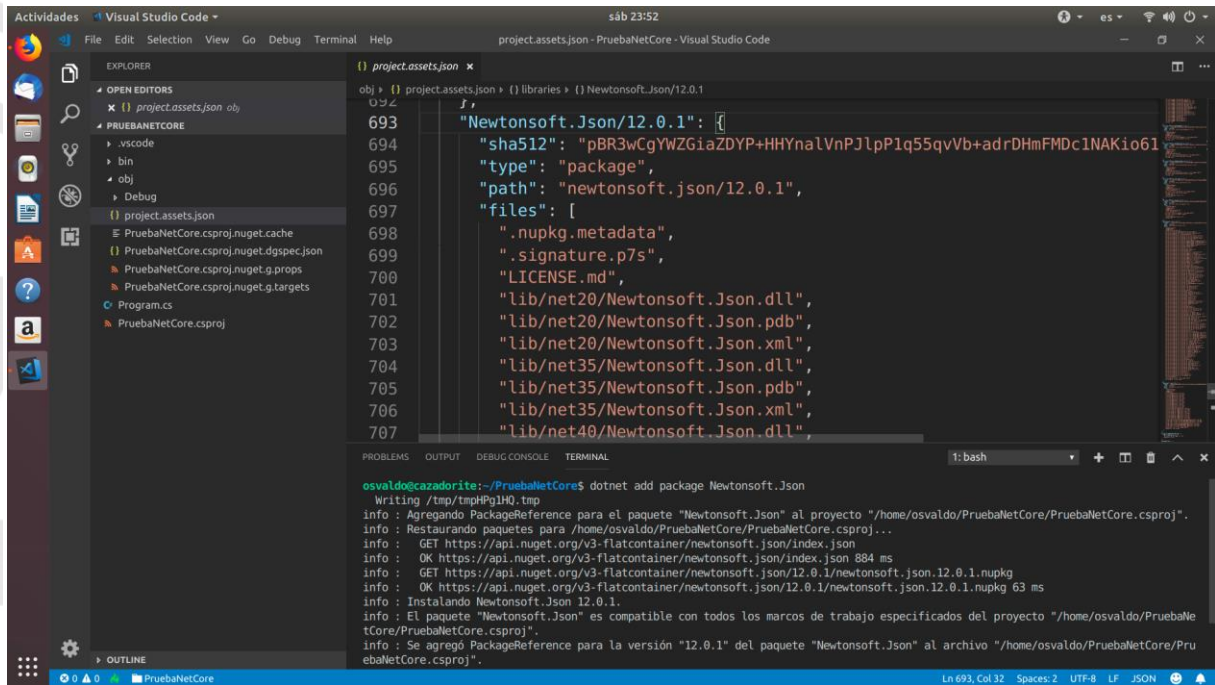
Nuestro programa imprime en consola. Busquemos la referencia a esa librería.



Como agregar dependencias ?

Por ejemplo, vamos a agregar una librería de utilidades para trabajar JSON

<https://www.nuget.org/packages/Newtonsoft.Json/>



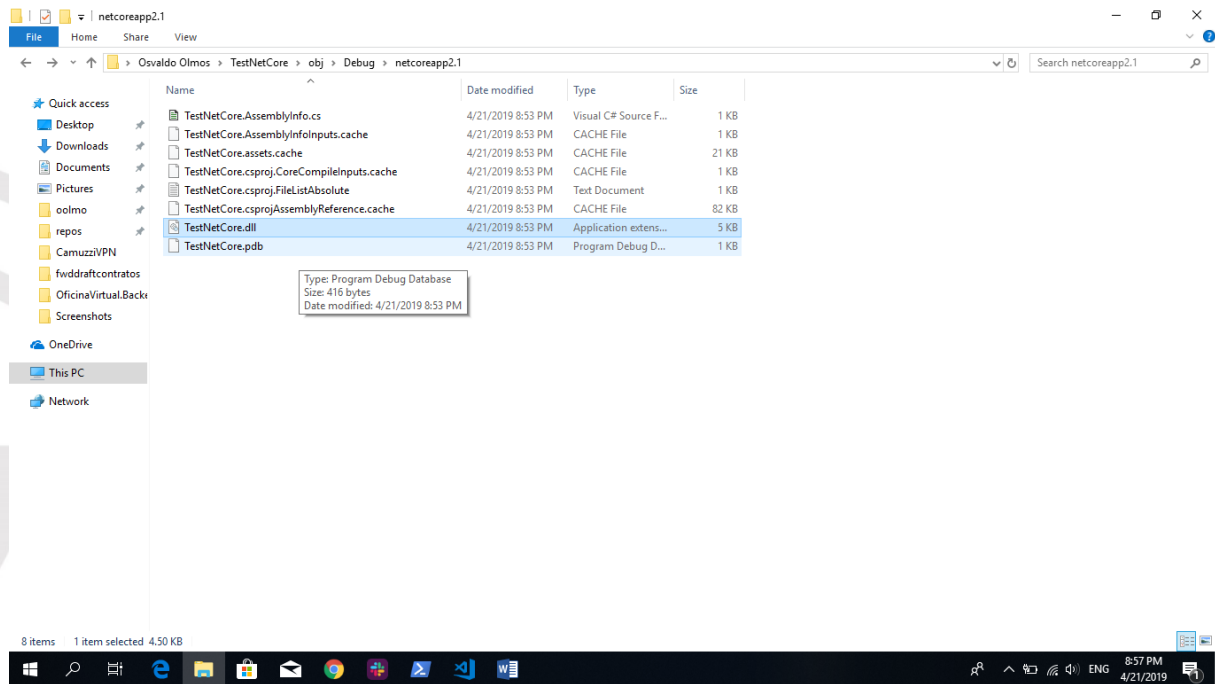
The screenshot shows the Visual Studio Code interface with the Explorer, Search, and Run and Debug views on the left. The main editor displays the `project.assets.json` file, which contains the JSON configuration for the project. The terminal at the bottom shows the command `dotnet add package Newtonsoft.Json` being executed, followed by the output of the command, which includes the package name, version, and the path to the package files.

```
obj : project.assets.json > {} libraries > {} Newtonsoft.Json/12.0.1
693 "Newtonsoft.Json/12.0.1": {
694   "sha512": "pBR3wGyWZGiaZDYP+HHYnaLVnPJlpP1q55qvVb+adrDHmFMDc1NAKio61
695   "type": "package",
696   "path": "newtonsoft.json/12.0.1",
697   "files": [
698     ".nupkg.metadata",
699     ".signature.p7s",
700     "LICENSE.md",
701     "lib/net20/Newtonsoft.Json.dll",
702     "lib/net20/Newtonsoft.Json.pdb",
703     "lib/net20/Newtonsoft.Json.xml",
704     "lib/net35/Newtonsoft.Json.dll",
705     "lib/net35/Newtonsoft.Json.pdb",
706     "lib/net35/Newtonsoft.Json.xml",
707     "lib/net40/Newtonsoft.Json.dll",

```

```
osvaldo@cazadorite:~/PruebaNetCore$ dotnet add package Newtonsoft.Json
Writing /tmp/tpHPglH0.tmp
info : Agregando PackageReference para el paquete "Newtonsoft.Json" al proyecto "/home/osvaldo/PruebaNetCore/PruebaNetCore.csproj".
info : Restaurando paquetes para /home/osvaldo/PruebaNetCore/PruebaNetCore.csproj...
info : GET https://api.nuget.org/v3-flatcontainer/newtonsoft.json/index.json
info : OK https://api.nuget.org/v3-flatcontainer/newtonsoft.json/index.json 884 ms
info : GET https://api.nuget.org/v3-flatcontainer/newtonsoft.json/12.0.1/newtonsoft.json.12.0.1.nupkg
info : OK https://api.nuget.org/v3-flatcontainer/newtonsoft.json/12.0.1/newtonsoft.json.12.0.1.nupkg 63 ms
info : Instalando Newtonsoft.Json 12.0.1.
info : El paquete "Newtonsoft.Json" es compatible con todos los marcos de trabajo especificados del proyecto "/home/osvaldo/PruebaNetCore/PruebaNetCore.csproj".
info : Se agregó PackageReference para la versión "12.0.1" del paquete "Newtonsoft.Json" al archivo "/home/osvaldo/PruebaNetCore/PruebaNetCore.csproj".
```

Donde esta el ejecutable ?



Solo vemos un archivo .dll. Este archivo es el que ejecuta el runtime de netcore

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Processing post-creation actions...
Running 'dotnet restore' on C:\Users\oolmo\TestNetCore\TestNetCore.csproj...
  Restoring packages for C:\Users\oolmo\TestNetCore\TestNetCore.csproj...
  Generating MSBuild file C:\Users\oolmo\TestNetCore\obj\TestNetCore.csproj.nuget.g.props.
  Generating MSBuild file C:\Users\oolmo\TestNetCore\obj\TestNetCore.csproj.nuget.g.targets.
  Restore completed in 275.17 ms for C:\Users\oolmo\TestNetCore\TestNetCore.csproj.

Restore succeeded.

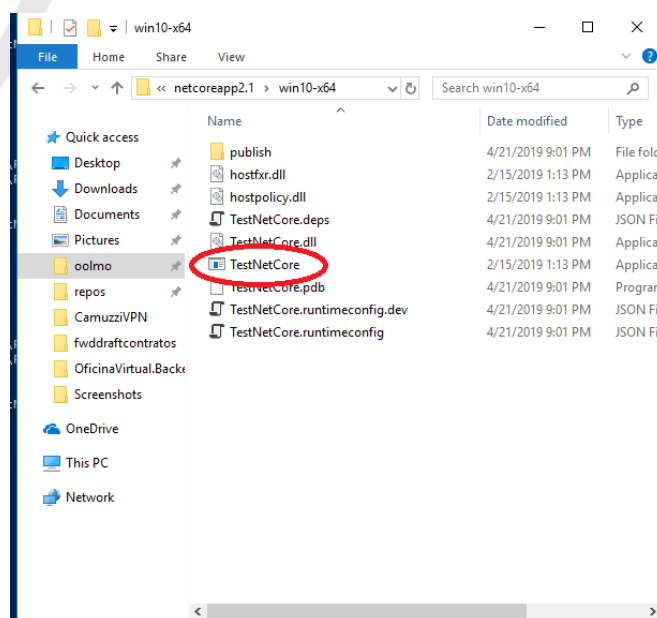
PS C:\Users\oolmo\TestNetCore> dotnet run
Hello World!
PS C:\Users\oolmo\TestNetCore> |
```

Si ejecutamos lo siguiente

```
PS C:\Users\oolmo\TestNetCore> dotnet publish -c Release -r win10-x64
Microsoft (R) Build Engine version 15.9.20+g88f5fadfbc for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

Restoring packages for C:\Users\oolmo\TestNetCore\TestNetCore.csproj...
Installing Microsoft.NETCore.DotNetAppHost 2.1.9.
Installing Microsoft.NETCore.DotNetHostResolver 2.1.9.
Installing Microsoft.NETCore.DotNetHostPolicy 2.1.9.
Installing Microsoft.NETCore.Platforms 2.1.3.
Installing Microsoft.NETCore.Targets 2.0.0.
Installing Microsoft.NETCore.App 2.1.9.
Installing runtime.win-x64.Microsoft.NETCore.DotNetAppHost 2.1.9.
Installing runtime.win-x64.Microsoft.NETCore.DotNetHostResolver 2.1.9.
Installing runtime.win-x64.Microsoft.NETCore.DotNetHostPolicy 2.1.9.
Installing runtime.win-x64.Microsoft.NETCore.App 2.1.9.
Generating MSBuild file C:\Users\oolmo\TestNetCore\obj\TestNetCore.csproj.nuget.g.props.
Generating MSBuild file C:\Users\oolmo\TestNetCore\obj\TestNetCore.csproj.nuget.g.targets.
Restore completed in 15.68 sec for C:\Users\oolmo\TestNetCore\TestNetCore.csproj.
TestNetCore -> C:\Users\oolmo\TestNetCore\bin\Release\netcoreapp2.1\win10-x64\TestNetCore.dll
TestNetCore -> C:\Users\oolmo\TestNetCore\bin\Release\netcoreapp2.1\win10-x64\publish\
PS C:\Users\oolmo\TestNetCore>
```

Vamos a generar el exe



Y podemos ejecutarlo de la siguiente manera

```
Windows PowerShell (x86)
PS C:\Users\oolmo\TestNetCore\bin\Release> cd .\netcoreapp2.1\
PS C:\Users\oolmo\TestNetCore\bin\Release\netcoreapp2.1> ls

Directory: C:\Users\oolmo\TestNetCore\bin\Release\netcoreapp2.1

Mode                LastWriteTime         Length Name
----                -
d-----          4/21/2019   9:01 PM             win10-x64

PS C:\Users\oolmo\TestNetCore\bin\Release\netcoreapp2.1> cd .\win10-x64\
PS C:\Users\oolmo\TestNetCore\bin\Release\netcoreapp2.1\win10-x64> ls

Directory: C:\Users\oolmo\TestNetCore\bin\Release\netcoreapp2.1\win10-x64

Mode                LastWriteTime         Length Name
----                -
d-----          4/21/2019   9:01 PM             publish
-a-----         2/15/2019   1:13 PM        402512 hostfxr.dll
-a-----         2/15/2019   1:13 PM        585288 hostpolicy.dll
-a-----         4/21/2019   9:01 PM        41748 TestNetCore.deps.json
-a-----         4/21/2019   9:01 PM         4096 TestNetCore.dll
-a-----         2/15/2019   1:13 PM       137728 TestNetCore.exe
-a-----         4/21/2019   9:01 PM         412 TestNetCore.pdb
-a-----         4/21/2019   9:01 PM        236 TestNetCore.runtimeconfig.dev.json
-a-----         4/21/2019   9:01 PM         28 TestNetCore.runtimeconfig.json

PS C:\Users\oolmo\TestNetCore\bin\Release\netcoreapp2.1\win10-x64> .\TestNetCore.exe
Hello World!
PS C:\Users\oolmo\TestNetCore\bin\Release\netcoreapp2.1\win10-x64>
```

Conclusiones:

La app esta buildeada para correr en una plataforma específica. Por eso lo natural no es tener un “exe” de salida.

Historial de Versiones de .NET Core

https://en.wikipedia.org/wiki/ASP.NET_Core

Futuro de .NET Core

<https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-core-3-0>

BIBLIOGRAFÍA RECOMENDADA:

OBLIGATORIA

<https://docs.microsoft.com/es-es/dotnet/core/>

COMPLEMENTARIA

<https://www.loginworks.com/blogs clr-work-dot-net-framework/>

https://en.wikipedia.org/wiki/.NET_Framework_version_history

Fecha	Versión	Observaciones
20/06/2018	1.0	Versión Original
20/04/2019	2.0	Adaptación a NET Core