Agenda: Building .NET Core Applications

- Building Console Based application
    - CLI tools
    - Visual Studio Code
    - Visual Studio 2017
- Debugging and Testing Application.
- Developing .NET Core on Linux

Visual Studio Code

Visual Studio Code (IDE):

Visual Studio Code is a **lightweight** but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity)

Download VS Code – for Windows, macOS and Linux.

Install the C# extension for Visual Studio Code.

Features of Visual Studio Code:

1. **Code smarter with IntelliSense** - completions for variables, methods, and imported modules.
2. **Streamlined Debugging:** Print debugging is a thing of the past. Debug in VS Code with your terminal tools.
3. **Fast, Powerful Editing**: Linting, multi-cursor editing, parameter hints, and other powerful editing features.
4. **Code Navigation and Refactoring**: Browse your source code quickly using peek and navigate to definition.
5. **In-Product Source Control**: Speed up your release cycle with SCM support inside your editor, including rich Git integration.

Production Platform Requirements

.NET Core Runtime https://www.microsoft.com/net/download/Windows/run

- To run applications built with .NET Core, a cross-platform .NET implementation.
- To run applications built with ASP.NET Core, you will also need the ASP.NET Core runtime.

Hello World .NET Core Application using Visual Studio Code

Open a Project

1. Execute the following command

    d:\DotNetCoreDemos\>**dotnet new console**
2. Create a Folder "d:\DotNetCoreDemos\HelloWorldDemo" in any location
3. Visual Studio Code → File → Open Folder → HelloWorldDemo

Initialize the C# Project

4. View → Integrated Terminal (Ctrl + `)
5. Type the following command

dotnet new console

Note: This command creates **Program.cs** file in your folder with simple "Hello World" program

HelloWorldDemo.csproj

```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>netcoreapp2.2</TargetFramework>
 </PropertyGroup>
</Project>
```

Run the "HelloWorldDemo" program

6. Type the following command in terminal

dotnet build (optional)

dotnet run

Note: **dotnet run** calls **dotnet build** to ensure that the build targets have been built, and then calls **dotnet <assembly.dll>** to run the target application.

This results in a compiled application as a DLL file that can be run with

dotnet ./bin/debug/netcoreapp2.2/HelloWorldDemo.dll

Command Line Arguments:

dotnet run <args1> <args2>

Handling Multiple Mains

To Specify the entry point method if multiple files with Main are added to project

7. Add another file to the project

```
public class Program2
{
    static void Main(string[] args)
    {
       Console.WriteLine("This is Program2");
    }
}
```

8. You can edit your **csproj** to define which class to use (inside a **<PropertyGroup>**):

    <StartupObject>HelloWorldDemo.Program2</StartupObject>

    OR

    $ dotnet build HelloWorldDemo.csproj **/p:StartupObject=HelloWorldDemo.Program2**

Note:

Using File Watcher

`dotnet watch` provides a file system watcher that waits for a file to change before executing a designated set of commands.

A file change can trigger compilation, test execution, or deployment.

For example, the following command automatically rebuilds the current project and generates verbose output whenever a file in it changes:

`dotnet watch --verbose build`

Following command automatically runs the application as soon as it changes the file

`dotnet watch run`

Debug the application

9. Open *Program.cs* by clicking on it. The first time you open a C# file in Visual Studio Code, **OmniSharp** loads in the editor.
10. Visual Studio Code should prompt you to ***add the missing assets to build and debug your app***. Select **Yes**.
11. To open the Debug view, click on the Debugging icon on the left side menu.
12. Locate the **green arrow** at the top of the pane or press F5. Make sure the drop-down next to it has `.NET Core Launch (console)` selected.
13. Add a breakpoint to your code.
14. To start debugging, select F5 or the green arrow.

Testing the Project

15. Create a Folder "**HelloSolution"** and execute the following command in it.

    d:\HelloSolution> dotnet new sln

16. Create a Folder "**HelloSolution/HelloWorldDemo**" and execute following command

    d:\HelloSolution> dotnet new **console** -o HelloWorldDemo

    d:\HelloSolution>dotnet sln HelloSolution.sln **add** HelloWorldDemo/HelloWorldDemo.csproj

17. Add a new folder "HelloSolution/**HelloWorldTest**" and execute the following command

    d:\HelloSolution>dotnet new **xunit** -o HelloWorldTest

    d:\HelloSolution>dotnet sln HelloSolution.sln **add** HelloWorldTest/HelloWorldTest.csproj

18. To add a project reference, use the `dotnet add reference` command:

d:\HelloSolution\HelloWorldTest> dotnet **add reference** ../HelloWorldDemo/HelloWorldDemo.csproj

OR

We can manually add the following to **HelloWorldTest.csproj**

```
<ItemGroup>
  <ProjectReference Include="../HelloWorldDemo/HelloWorldDemo.csproj" />
</ItemGroup>
```

List the project references for the project in the current directory:

dotnet list reference

Remove multiple project references from the project in the current directory:

dotnet remove reference lib1/lib1.csproj lib2/lib2.csproj

19. Edit Unit1.test

```
namespace HelloWorldTest
{
   public class UnitTest1
   {
      [Fact]
      public void Test1()
      {
         int expected = 10;
         int actual = HelloWorldDemo.Math.Add(5, 5);
         Assert.Equal(expected, actual);
      }
   }
}
```

20. At terminal execute the following

    dotnet **build**

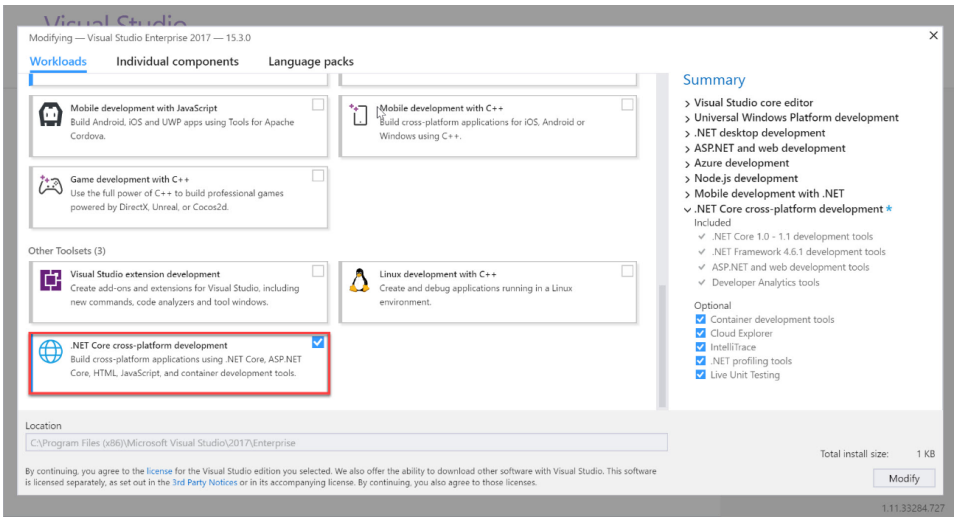
    dotnet **test**

Note: This command starts the test runner specified in the project file.

Hello World application with .NET Core in Visual Studio 2017

To develop .NET Core 2.x apps in Visual Studio 2017:

1. Download and install Visual Studio 2017 with the **.NET Core cross-platform development** workload (in the **Other Toolsets** section) selected.



2. After the **.NET Core cross-platform development** toolset is installed, Visual Studio 2017 uses .NET Core 1.x by default. Install the .NET Core 2.x SDK to get .NET Core 2.x support in Visual Studio 2017. Install the .NET Core 2.x SDK.
3. Create a Project and Execute the same.
    1. File → New → Project → Template → .NET Core → Console App (.NET Core)
    2. Name = HelloWorldInVS2017DemoApp, Solution= HelloWorldInVS2017DemoSolution→ OK
    3. Build → Build Solution
    4. Run: F5 or Debug Run

.NET Core Application Development on Linux

.NET Core 2.1 treats Linux as a single operating system. There is a single Linux build (per chip architecture) for supported Linux distros.

NET Core 2.x is supported on the following Linux 64-bit (`x86_64` or `amd64`) distributions/versions:

- Red Hat Enterprise Linux 7
- CentOS 7
- Oracle Linux 7
- Fedora 25, Fedora 26
- Debian 8.7 or later versions
- Ubuntu 17.04, Ubuntu 16.04, Ubuntu 14.04
- Linux Mint 18, Linux Mint 17
- openSUSE 42.2 or later versions
- SUSE Enterprise Linux (SLES) 12 SP2 or later versions

1. **Use Hyper-V**
    - Create a New VM

- Install Ubuntu from ISO

OR

2. Create a VM in Azure or Linux Ubuntu 16.04 LTS (Long Term Support)

Download Putty from www.putty.org

Login using Putty: Hostname = dssadmin@<IP>, Port=22

Configuring Linux Machine:

3. Ensure that Latest Linux Libraries are installed

```
sudo apt-get update                          #This updates the repository
sudo apt-get upgrade                         #Fetches all the updates for the installed softwares
sudo apt-get install curl                    #curl is used for fetch the o/p of URL
sudo apt-get install openssh-server          #required for remote login (required when Hyper-V is used)
sudo raspi-config → Interfacing Options → P2 SSH → Yes        #to enable SSH Server on RaspberryPi
sudo apt-get install apt-transport-https #For HTTPS support
```

4. Ensure that Linux distribution dependencies for .NET Core are installed.

```
sudo apt-get install liblttng-ust0 libcurl3 libssl1.0.0 libkrb5-3 zlib1g libicu55
```

5. Register the Microsoft Product key as trusted.

   wget -q https://packages.microsoft.com/config/ubuntu/16.04/packages-microsoft-prod.deb

   sudo dpkg -i packages-microsoft-prod.deb Set up the desired version host package feed. (for 16.04)

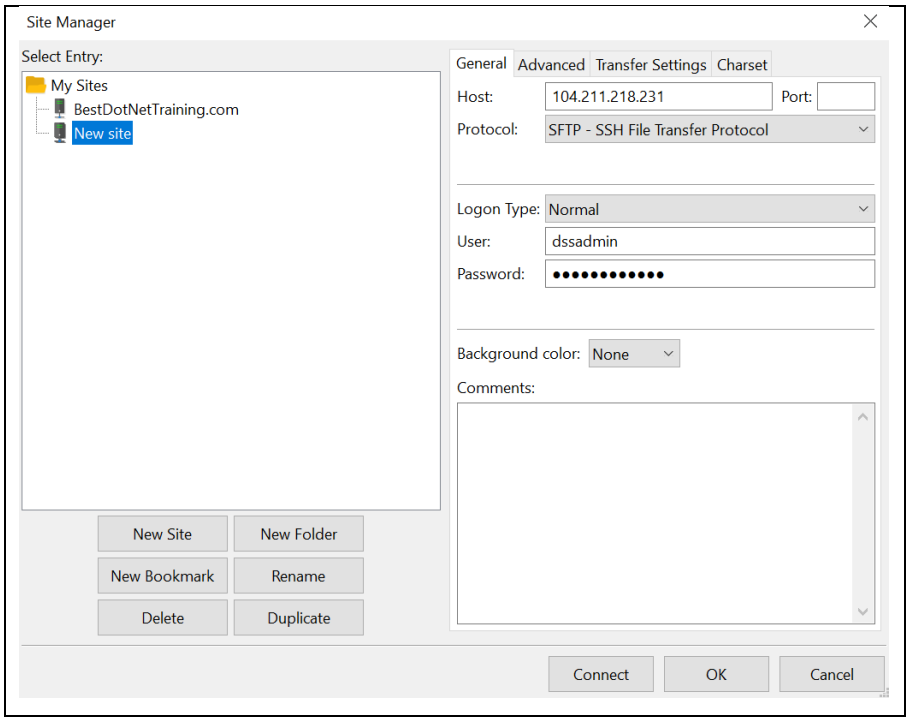
```
sudo apt-get update
```

6. Install .NET Core.

```
sudo apt-get install dotnet-sdk-2.1
```

7. Run the **dotnet --version** command to prove the installation succeeded.

Filezilla → Use SFTP (SSH File Transfer Protocol) to connect



Some Useful Commands for Linux

- General Commands: ls, clear, pwd,
- cd /home/<username>
- ifconfig – to know the IP address of Machine
- Ctrl + Alt + T to start the terminal window.
- sudo apt install curl
- sudo apt-get install opensh-server (To connect using FTP Client which supports ssh protocol)
- sudo apt-get install apt-transport-https (To support https protocol)
- sudo inti 0 (its zero)
- sudo nano /etc/default/grub (to edit resolution)