**ASP.NET Core for Linux Setup**

**Windows Subsystem for Linux (WSL)**

Linux is great for development, because many of the tools needed for a programmer are part of the operating system. However, for general office use that involves applications, such as Outlook and Word, Windows is generally a better choice for most people. The natural follow-up to this is that it would be great if you could have both Linux and Windows at the same time.

Windows has supported virtualization in different forms for a long time, and because Linux runs on the same hardware as Windows (as well as being available for free in many cases), it has been a common option to run a virtual machine with Linux, if you need it. However, the point of virtual machines is having something separate from the host machine. Thus, even minor things, such as getting files into and out of the Linux virtual machine, has been a less than smooth experience.

In 2016, Microsoft brought the Linux operating system closer to being a part of Windows by introducing **Windows Subsystem for Linux** (**WSL**), where you could install special builds of selected distributions into Windows 10. This was further improved with WSL2, which was introduced with Windows 10 2004, where Linux can be made an integrated part of Windows. (The current release of Windows 10 is named 20204 to signify that the release was first released in the year 2020 and the fourth month, April.)

Let's install WSL2 before we proceed with running code on Linux.

Note that this is the install procedure as of the May 2020 version of Windows 10. Things may change in future versions.

Your computer will need to be capable of running Hyper-V and Windows 10 2004 (or later). Most modern computers will be able to run Hyper-V, but if your developer machine is virtualized, then there may be issues enabling WSL2.

To install WSL2, perform the following steps:

1. Open Command Prompt as an admin.
2. Run the following command to install WSL:

dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

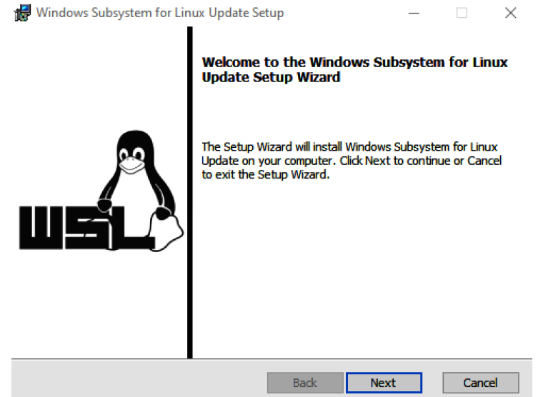
1. Enable Virtual Machine Platform by using the following command:

dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

1. Reboot your computer.
2. Download the latest WSL2 kernel from

https://wslstorestorage.blob.core.windows.net/wslblob/wsl\_update\_x64.msi

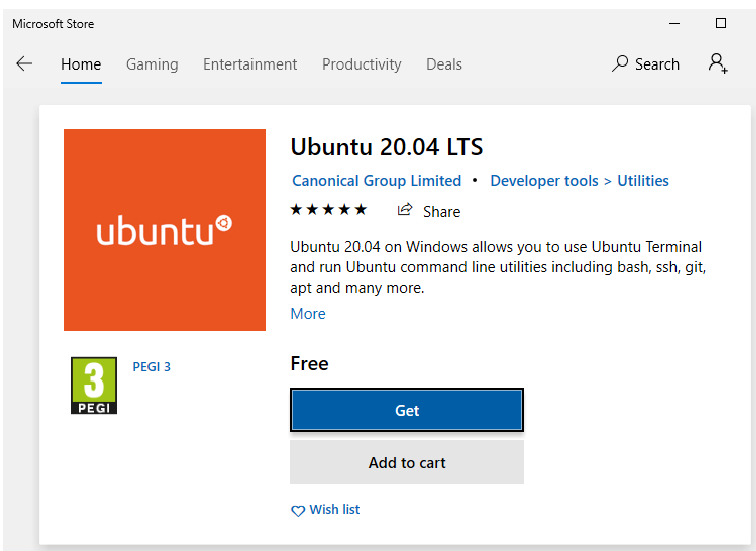
1. Run the installer, as shown in figure below:



1. Make WSL2 your default version by using the following command:

wsl –-set-default-version 2

1. Download a Linux distribution from the Microsoft Store. For this book, we have used Ubuntu 20.04 LTS (see figure below):



1. Click Launch to start Linux for the first time:



1. Define a username and password for your Linux installation (see figure above). (This is not related to your Windows credentials and can be something different.) You should now find yourself in a regular Linux shell.
2. Since this operating system lives its own life, it is suggested to start by updating to the latest patches by running sudo apt update && sudo apt upgrade, as shown in figure below:



1. Press Y to continue, and you should be good to go.

Windows should also have automatically configured integration with your non-Linux hard drive partition. So, if you open Windows Explorer, you should find Tux (the Linux mascot) there:



You can also browse the Linux filesystem from Windows and copy files to and from your Linux partition (see figure below):



Note that, under the hood, the Linux filesystem is treated differently to the Windows filesystem, so only place files that you intend to run inside Linux in these folders, and vice versa. If you have applications that run inside Linux, these should not be placed in the Windows partition. It will not cause corruption to do so, but the performance might be degraded.

The Ubuntu installer automatically started up a command line, but if you followed the instructions in the previous chapter for setting up Windows Terminal, Ubuntu 20.04 should have been added automatically. This book uses Windows Terminal going forward in this chapter, but both options should work.

**Installing .NET on Linux**

We recommend you install .NET on Ubuntu by using APT:

1. Run the following commands to add Microsoft's repositories:

wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb

sudo dpkg -i packages-microsoft-prod.deb

1. Install the SDK:

sudo apt-get update; \

sudo apt-get install -y apt-transport-https && \

sudo apt-get update && \

sudo apt-get install -y dotnet-sdk-5.0

**Note**

There are a couple of different ways to install .NET on Linux, and things may

change over time. If you experience issues while installing .NET, check the

instructions online at https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu

Everything should now be in place for creating and running a .NET application. It is time to test the theory in practice:

1. Create a new directory and change into it:

mkdir LinuxHelloWorld && cd LinuxHelloWorld

1. Linux running in WSL2 does not support a graphical UI yet, so we need to do the editing via non-graphical utilities:

sudo vi View/Home/Index.cshtml

1. Vi is not exactly intuitive, but press Insert and edit the code to look like this:

@{

ViewData["Title"] = "LinuxHelloWorld";

}

<div class="text-center">

<h1 class="display-4">Running on @Environment.OSVersion</h1>

</div>

1. To save and exit, press **Esc** followed by **:wq**, and then hit **Enter**.
2. Test the app with **sudo dotnet run**. You should see the output indicate that it is running. See figure below:



1. You can test this with some more cross-platform magic. You do not have a browser running on this Ubuntu. You most likely do have one in Windows 10, so you can open that and browse to https://locahost:5001. See figure below for an example of browsing a web app that's running on Linux:



1. Return to the Linux shell and terminate the running app with **Ctrl+C**.

We saw that the vi utility was sufficient for the minor edits that we made to the code, but not everyone will want to go all-in on Vi as an editor for writing C# code.

Vi "exit strategy"

If you are new to Vi, it can be confusing, because it works differently to most text editors that you might be used to in the Windows world. You might end up being unsure about what you have actually edited, or how to correct it. The exit strategy (if you feel like a mistake was made) is to quit Vi without saving

the changes. This is done by pressing the Esc key, pressing : (the colon) (you should see it appear in the lower-left corner), and then typing q! (include the exclamation mark), followed by Enter. You can then re-attempt editing with a clean slate.

Fortunately, there is another option here as well. In the previous chapter, we showed you how useful Visual Studio Code is, so if you haven't already installed it, please do so.

We will step through how to use Visual Studio Code (VS Code) as the editor for your code on Linux:

1. Open Visual Studio Code (in Windows 10).
2. Install the Remote – WSL extension from within VS Code. See figure below:



1. Go back to your Linux shell in WSL and type code . (including the punctuation mark).
2. After an initial bit of setup work, Visual Studio Code will load in Windows 10. You will observe that there's an indicator in the lower-left corner referring to WSL. See figure below:



1. If you have the C# Extension installed in VS Code, you can go to the debug pane (at the bottom). See figure below:



1. Click the little green arrow to start the debugger. When things have finished building, you should see the same output as before with the LinuxHelloWorld app running in the browser. (VS Code launches the browser for you.) If you take a look in the Terminal window, you will see the application starting in WSL. See figure below:



This session is separate to the one you are running in the Windows Terminal shell, so you can work in parallel there if you like.

Now you can develop code in Windows, which executes on Linux running on Windows. This can take a little while to digest, but the takeaway from this section is that the cross-platform story for Linux is powerful.