Ubuntu WSL

Canonical Group Ltd

Mar 11, 2024

CONTENTS

1 Windows Subsystem for Linux **3**

2 Ubuntu and WSL **5**

2.1 Tutorials 5

2.2 How-to guides 49

2.3 Reference 57

This documentation refers to the Ubuntu distribution for WSL.

CHAPTER

ONE

WINDOWS SUBSYSTEM FOR LINUX

Windows Subsystem for Linux (WSL) lets developers run a GNU/Linux environment – including most command-line tools, utilities, and applications – directly on Windows, unmodified, without the overhead of a traditional virtual machine or dual-boot setup. It is a Microsoft product, developed, maintained and documented by Microsoft. You can find their documentation here.

WSL can be installed from the Microsoft Store.

CHAPTER

TWO

UBUNTU AND WSL

While Microsoft provides the virtual machine, the kernel, and a few more utilities; Ubuntu provides the distribution that runs on top of them. We develop multiple flavours of Ubuntu for WSL, which you can read more about *here*. These flavours materialise as applications on the Microsoft Store.

2.1 Tutorials

2.1.1 Working with Visual Studio Code

Authored by Oliver Smith (oliver.smith@canonical.com)

The easiest way to access your Ubuntu development environment in WSL is by using Visual Studio Code via the built-in `Remote` extension.

What you will learn:

- How to set up Visual Studio Code for remote development on Ubuntu on WSL
- How to start a basic Node.js webserver on Ubuntu using Visual Studio Code

What you will need:

- A PC with Windows 10 or 11
- (Optional) This tutorial uses Windows Terminal Preview, which you can get from the Windows app store

Install Ubuntu on WSL2

This tutorial assumes that you already have WSL2 installed with the latest LTS release of Ubuntu.

If not, check out our getting started tutorials for Windows 10 and Windows 11:

- *Install Ubuntu on WSL2*

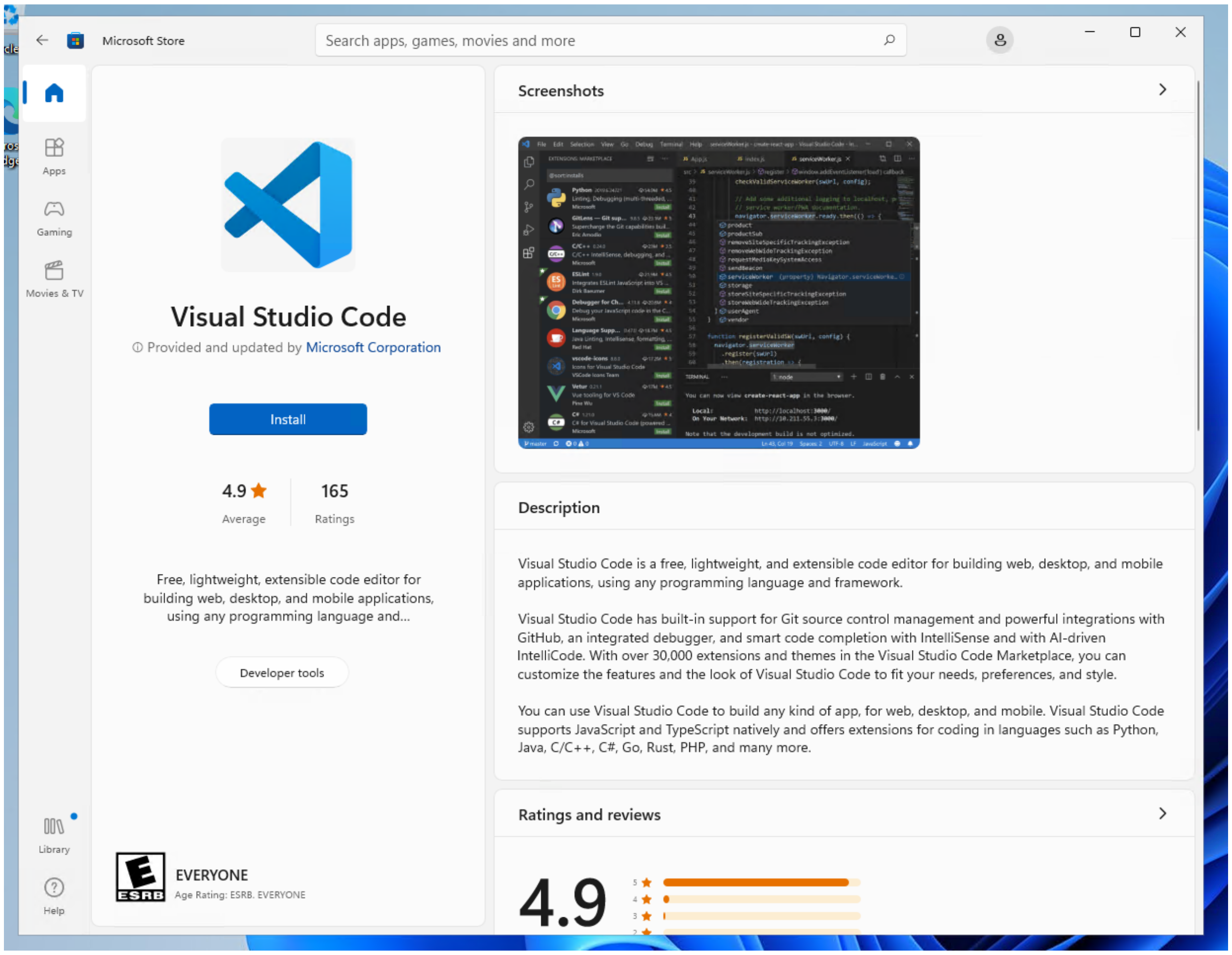
Once you have completed the relevant tutorial, the following steps will work on either Windows 10 or 11.

Install Visual Studio Code on Windows

One of the advantages of WSL is that it can interact with the native Windows version of Visual Studio Code using its remote development extension.

To install Visual Studio Code visit the Microsoft Store and search for `Visual Studio Code`.

Then click `Install`.



Visual Studio Code

Provided and updated by Microsoft Corporation

4.9 Average | 165 Ratings

Free, lightweight, extensible code editor for building web, desktop, and mobile applications, using any programming language and...

EVERYONE
Age Rating: ESRB. EVERYONE

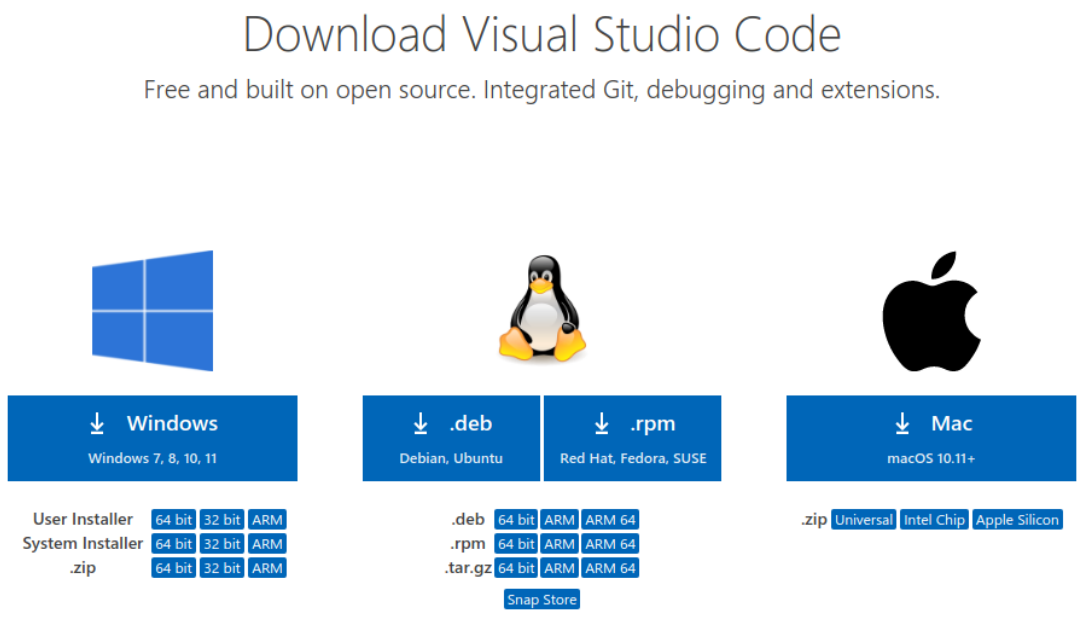
Visual Studio Code is a free, lightweight, and extensible code editor for building web, desktop, and mobile applications, using any programming language and framework.

Visual Studio Code has built-in support for Git source control management and powerful integrations with GitHub, an integrated debugger, and smart code completion with IntelliSense and with AI-driven IntelliCode. With over 30,000 extensions and themes in the Visual Studio Code Marketplace, you can customize the features and the look of Visual Studio Code to fit your needs, preferences, and style.

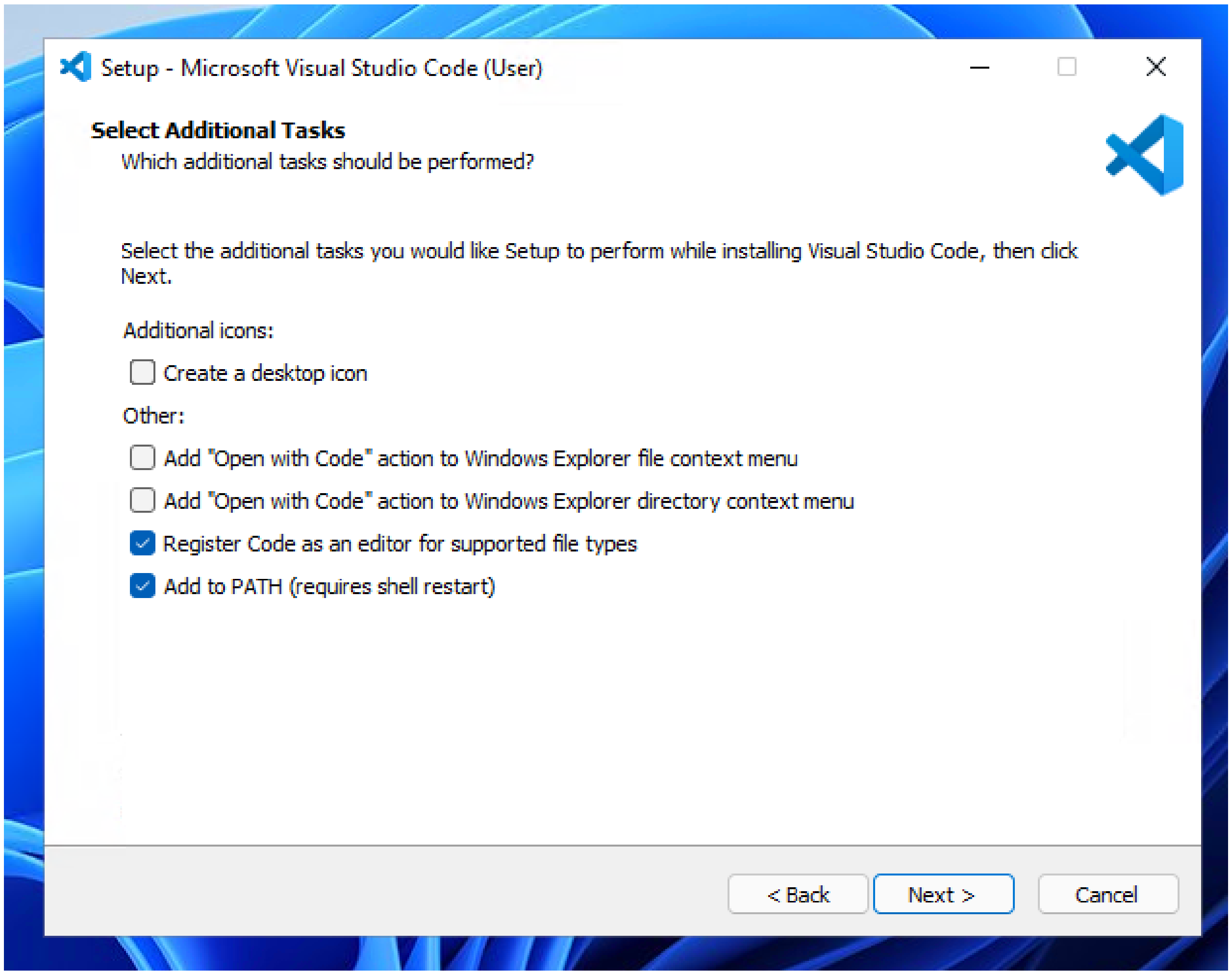
You can use Visual Studio Code to build any kind of app, for web, desktop, and mobile. Visual Studio Code supports JavaScript and TypeScript natively and offers extensions for coding in languages such as Python, Java, C/C++, C#, Go, Rust, PHP, and many more.

4.9

Alternatively, you can install Visual Studio Code from the web link here.


Download Visual Studio Code
Free and built on open source. Integrated Git, debugging and extensions.
Windows
Windows 7, 8, 10, 11
.deb
Debian, Ubuntu
.rpm
Red Hat, Fedora, SUSE
Mac
macOS 10.11+
User Installer 64 bit 32 bit ARM
System Installer 64 bit 32 bit ARM
.zip 64 bit 32 bit ARM
.deb 64 bit ARM ARM 64
.rpm 64 bit ARM ARM 64
.tar.gz 64 bit ARM ARM 64
Snap Store
.zip Universal Intel Chip Apple Silicon

During installation, under the `Additional Tasks` step, ensure the `Add to PATH` option is checked.



Select Additional Tasks

Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing Visual Studio Code, then click Next.

Additional icons:

- [ ] Create a desktop icon

Other:

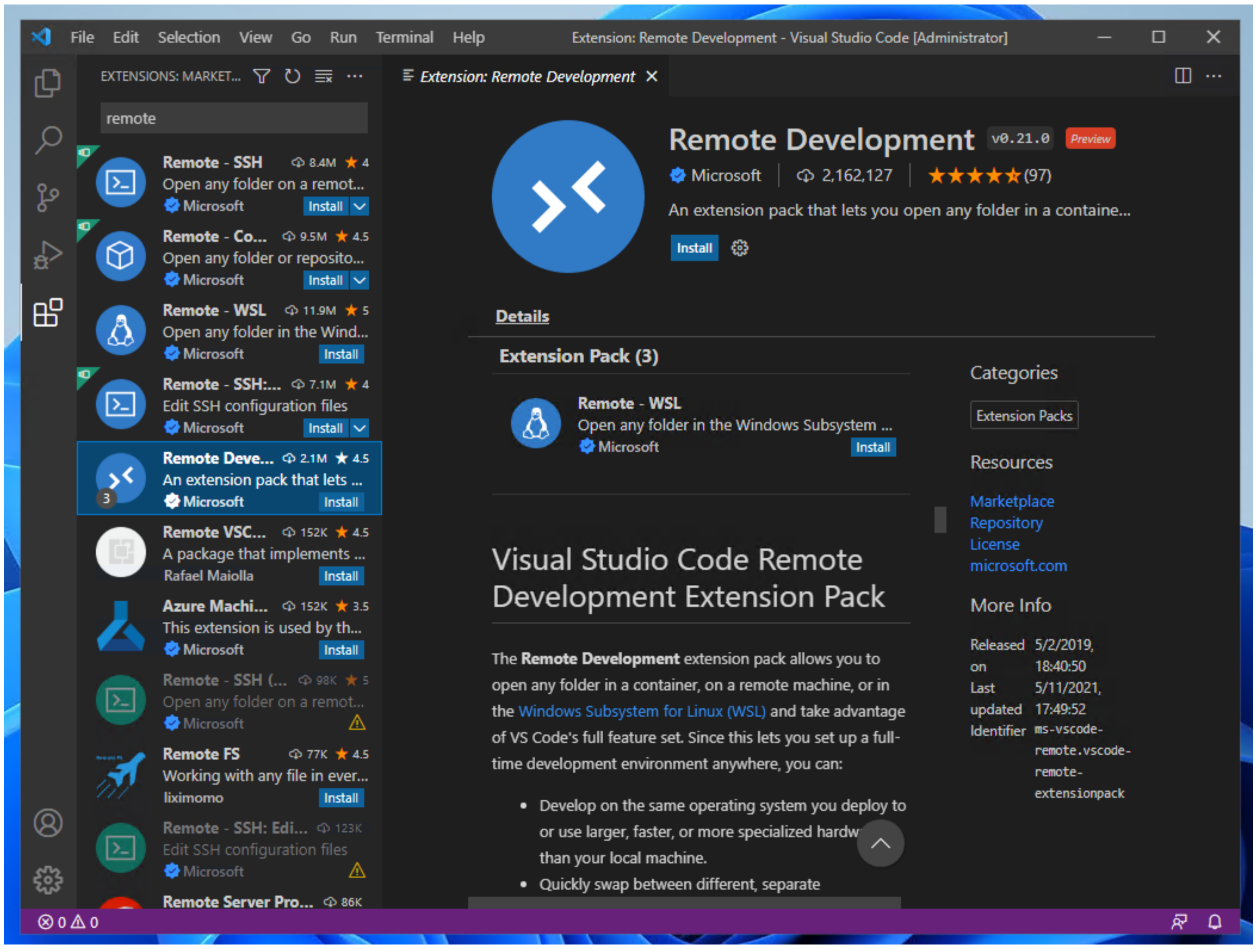
- [ ] Add "Open with Code" action to Windows Explorer file context menu
- [ ] Add "Open with Code" action to Windows Explorer directory context menu
- [✓] Register Code as an editor for supported file types
- [✓] Add to PATH (requires shell restart)

Once the installation is complete, open Visual Studio Code.

Install the Remote Development Extension

Navigate to the `Extensions` menu in the sidebar and search for `Remote Development`.

This is an extension pack that allows you to open any folder in a container, remote machine, or in WSL. Alternatively, you can just install `Remote - WSL`.


File Edit Selection View Go Run Terminal Help
Extension: Remote Development - Visual Studio Code [Administrator]
EXTENSIONS: MARKET...
remote
Extension: Remote Development
Install
Details
Categories
Extension Packs
Resources
Marketplace
Repository
License
microsoft.com
More Info

Remote - SSH 8.4M ★4 Open any folder on a remot... Microsoft Install
Remote - Co... 9.5M ★4.5 Open any folder or reposito... Microsoft Install
Remote - WSL 11.9M ★5 Open any folder in the Wind... Microsoft Install
Remote - SSH:... 7.1M ★4 Edit SSH configuration files Microsoft Install
Remote Deve... 2.1M ★4.5 An extension pack that lets ... Microsoft Install
Remote VSC... 152K ★4.5 A package that implements ... Rafael Maiolla Install
Azure Machi... 152K ★3.5 This extension is used by th... Microsoft Install
Remote - SSH (... 98K ★5 Open any folder on a remot... Microsoft
Remote FS 77K ★4.5 Working with any file in ever... liximomo Install
Remote - SSH: Edi... 123K Edit SSH configuration files Microsoft
Remote Server Pro... 86K

Remote Development v0.21.0 Preview
Microsoft | 2,162,127 | ★★★★★(97)
An extension pack that lets you open any folder in a containe...

Extension Pack (3)

Remote - WSL
Open any folder in the Windows Subsystem ...
Microsoft Install

Visual Studio Code Remote Development Extension Pack

The **Remote Development** extension pack allows you to open any folder in a container, on a remote machine, or in the Windows Subsystem for Linux (WSL) and take advantage of VS Code's full feature set. Since this lets you set up a full-time development environment anywhere, you can:

- Develop on the same operating system you deploy to or use larger, faster, or more specialized hardw than your local machine.
- Quickly swap between different, separate

Released on	5/2/2019, 18:40:50
Last updated	5/11/2021, 17:49:52
Identifier	ms-vscode-remote.vscode-remote-extensionpack

⊗ 0 ⚠ 0

Once installed we can test it out by creating an example local web server with Node.js

Install Node.js and create a new project

Open your WSL Ubuntu terminal and ensure everything is up to date by typing:

```
 sudo apt update
```

And then

```
sudo apt upgrade
```

Entering your password and pressing `Y` when prompted.

Next, install Node.js and npm:

```
sudo apt-get install nodejs

sudo apt install npm
```

Press `Y` when prompted.

Now, create a new folder for our server.

```
mkdir serverexample/
```

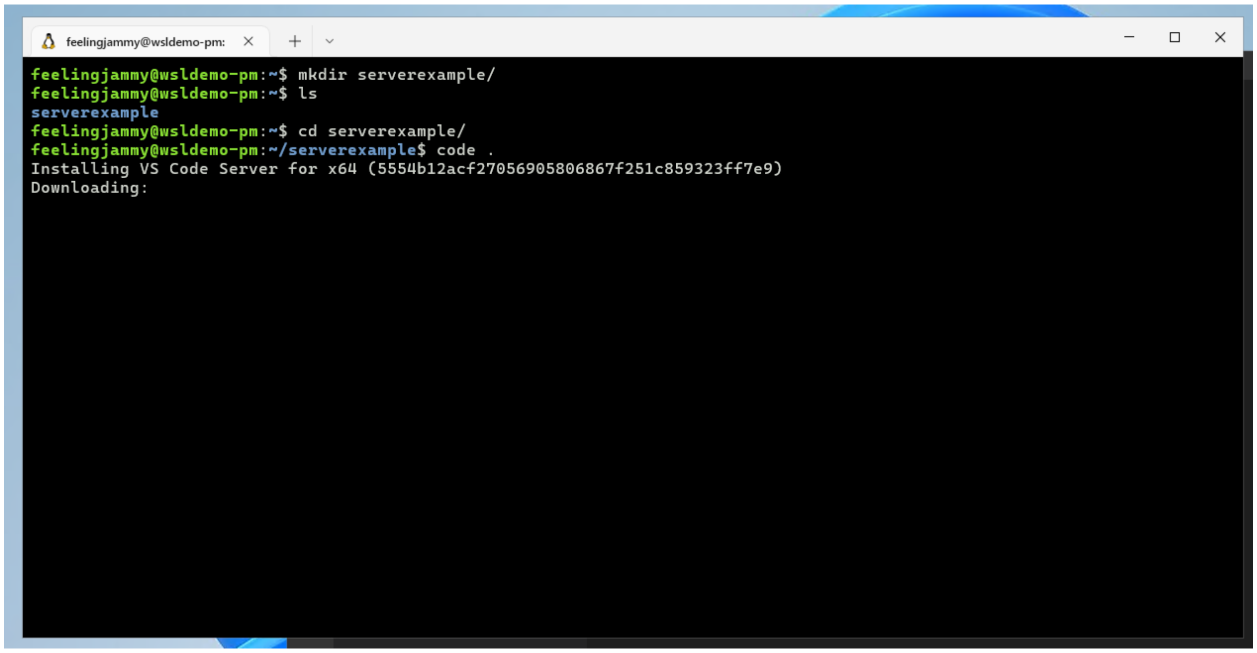
Then navigate into it:

```
cd serverexample/
```

Now, open up your folder in Visual Studio Code, you can do this by typing:

```
code .
```

The first time you do this, it will trigger a download for the necessary dependencies:



```
feelingjammy@wsldemo-pm:~$ mkdir serverexample/
feelingjammy@wsldemo-pm:~$ ls
serverexample
feelingjammy@wsldemo-pm:~$ cd serverexample/
feelingjammy@wsldemo-pm:~/serverexample$ code .
Installing VS Code Server for x64 (5554b12acf27056905806867f251c859323ff7e9)
Downloading:
```

Once complete, your native version of Visual Studio Code will open the folder.

Creating a basic web server

In Visual Studio Code, create a new file called `package.json` and add the following text (original example)

```
{
    "name": "Demo",
    "version": "1.0.0",
    "description": "demo project.",
    "scripts": {
        "lite": "lite-server --port 10001",
        "start": "npm run lite"
    },
    "author": "",
    "license": "ISC",
    "devDependencies": {
        "lite-server": "^1.3.1"
    }
}
```

Save the file and then, in the same folder, create a new one called `index.html`

Add the following text, and then save and close:

```
<h1>Hello World</h1>
```

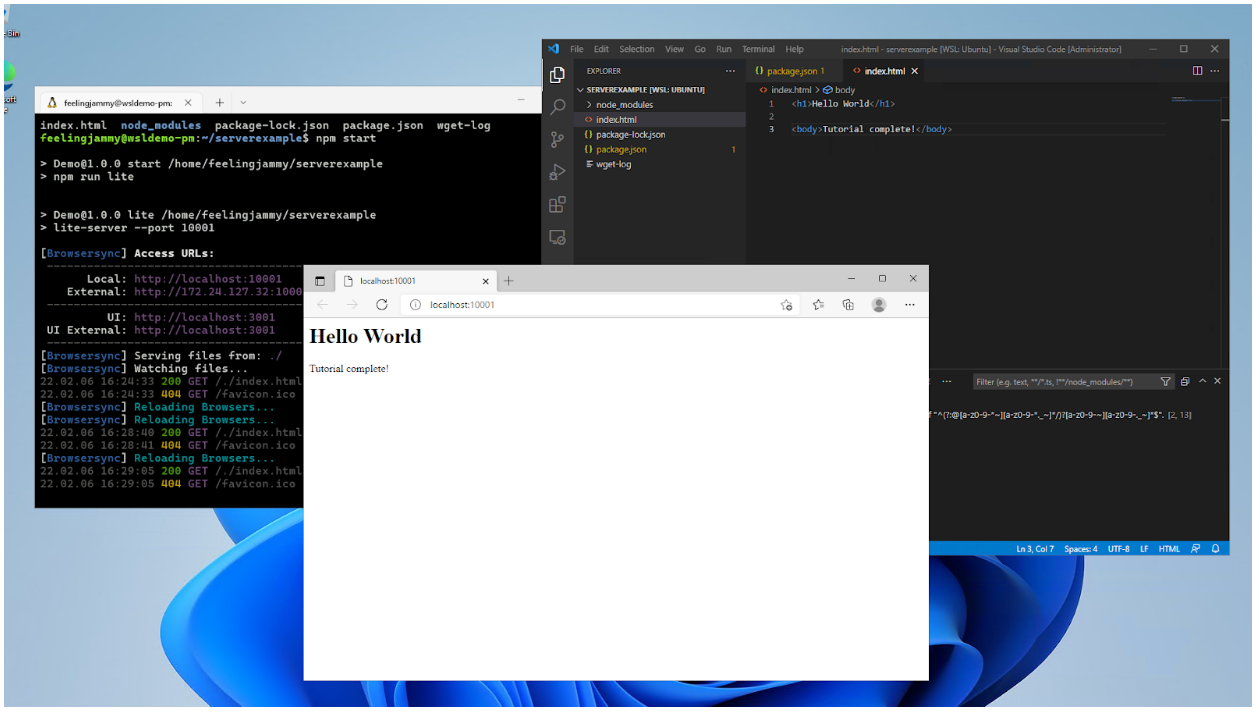
Now return to your Ubuntu terminal (or use the Visual Studio Code terminal window) and type the following to install a server defined by the above specifications detailed in `package.json`:

```
npm install
```

Finally, type the following to launch the web server:

```
npm start
```

You can now navigate to `localhost:10001` in your native Windows web browser by using `CTRL+Left Click` on the terminal links.



```
index.html  node_modules  package-lock.json  package.json  wget-log
feelingjammy@wsldemo-pm:~/serverexample$ npm start

> Demo@1.0.0 start /home/feelingjammy/serverexample
> npm run lite


> Demo@1.0.0 lite /home/feelingjammy/serverexample
> lite-server --port 10001

[Browsersync] Access URLs:
 ----------------------------------------
       Local: http://localhost:10001
    External: http://172.24.127.32:1000
 ----------------------------------------
          UI: http://localhost:3001
 UI External: http://localhost:3001
 ----------------------------------------
[Browsersync] Serving files from: ./
[Browsersync] Watching files...
22.02.06 16:24:33 200 GET /./index.html
22.02.06 16:24:33 404 GET /favicon.ico
[Browsersync] Reloading Browsers...
[Browsersync] Reloading Browsers...
22.02.06 16:28:40 200 GET /./index.html
22.02.06 16:28:41 404 GET /favicon.ico
[Browsersync] Reloading Browsers...
22.02.06 16:29:05 200 GET /./index.html
22.02.06 16:29:05 404 GET /favicon.ico
```

```
<h1>Hello World</h1>

<body>Tutorial complete!</body>
```

feelingjammy@wsldemo-pm:
index.html - serverexample [WSL: Ubuntu] - Visual Studio Code [Administrator]
File Edit Selection View Go Run Terminal Help
EXPLORER
SERVEREXAMPLE [WSL: UBUNTU]
node_modules
index.html
package-lock.json
package.json
wget-log
localhost:10001
Hello World
Tutorial complete!
Ln 3, Col 7 Spaces: 4 UTF-8 LF HTML

That's it!

By using Ubuntu on WSL you're able to take advantage of the latest Node.js packages available on Linux as well as the more streamlined command line tools.

Enjoy Ubuntu on WSL!

That's all folks! In this tutorial, we've shown you how to connect the Windows version of Visual Studio Code to your Ubuntu on WSL filesystem and launch a basic Node.js webserver.

We hope you enjoy using Ubuntu inside WSL. Don't forget to check out our other tutorials for tips on how to optimise your WSL setup for Data Science.

Further Reading

- *Install Ubuntu on WSL2*
- Microsoft WSL Documentation
- Setting up WSL for Data Science
- Ask Ubuntu

2.1.2 Windows and Ubuntu interoperability

Authored by Didier Roche-Tolomelli (didier.roche@canonical.com)

Some tools used during the development process are only available or are performed best on one platform and transferring data from one system to another to visualise or share can be tedious. WSL solves this problem with a feature called `interoperability`.

Interoperability is the ability to transparently execute commands and applications, share files, network and environment variables across Windows and Ubuntu.

We'll illustrate all these notions by generating data from your Ubuntu WSL instance using your Windows user profile directory, perform some transformations via PowerShell scripts, and finally, visualise those on Windows. We are going to cross the chasm between the two worlds not just once, but many times, seamlessly!

What you will learn:

- How to access a service provided by a web server running on your Ubuntu WSL instance from Windows.
- Share environment variables between Windows and Ubuntu, back and forth.
- Access files across filesystems, and discover where they are located on both sides.
- Run Windows commands (command line and graphical) from your WSL instance and chain them.

What you will need:

- Know how to use command line tools on Windows or Linux.
- A PC with Windows 10 or 11.
- Optional: LibreOffice or MS Excel to visualise and manipulate generated data from Ubuntu.

Install Ubuntu on WSL2

This tutorial assumes that you already have WSL2 installed with the latest LTS release of Ubuntu.

If not, check out our getting started tutorial for Windows 10 and Windows 11:

- *Install Ubuntu on WSL2*

For this tutorial, we will assume that you have installed the Ubuntu main WSL application.

Once you have completed the relevant tutorial, the following steps will work on either Windows 10 or 11.

Note: in this tutorial, we consider that interoperability is turned on in WSL.conf, which is the default behaviour. If you have disabled it, you can either use the ubuntu reconfiguration tool on Ubuntu 22.04+, or, for earlier versions, directly modify back `wsl.conf` yourself as described on wsl.conf documentation. The settings you are interested in are `[Interop]`: `enabled` and `appendWindowsPath` both set to true (or not being present, which defaults to true).

Share ports between WSL and Windows

Install Jupyter notebook on WSL.

Let's install Jupyter notebook, a web-based interactive computing platform where we will generate some statistics.

1. Start our WSL instance, on a terminal, using Ubuntu:

   $ ubuntu.exe

2. Install the python package manager pip:

```
$ sudo apt update

$ sudo apt install python3-pip
```

3. Get Jupyter notebook installed via pip:

   $ pip install notebook

Executing Jupyter notebook.

Finally, let's start Jupyter, by adding it to the user PATH first:

```
$ export PATH=$PATH:~/.local/bin
$ jupyter notebook --no-browser
[I 10:52:23.760 NotebookApp] Writing notebook server cookie secret to /home/u/.local/
↪share/jupyter/runtime/notebook_cookie_secret
[I 10:52:24.205 NotebookApp] Serving notebooks from local directory: /home/u
[I 10:52:24.205 NotebookApp] Jupyter Notebook 6.4.10 is running at:
[I 10:52:24.205 NotebookApp] http://localhost:8888/?
↪token=1d80ee69da6238f22bb683a4acd00025d32d15dde91cbdf4
[I 10:52:24.205 NotebookApp] or http://127.0.0.1:8888/?
↪token=1d80ee69da6238f22bb683a4acd00025d32d15dde91cbdf4
[I 10:52:24.205 NotebookApp] Use Control-C to stop this server and shut down all kernels␣
↪(twice to skip confirmation).
[C 10:52:24.209 NotebookApp]
To access the notebook, open this file in a browser:
file:///home/u/.local/share/jupyter/runtime/nbserver-5744-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=1d80ee69da6238f22bb683a4acd00025d32d15dde91cbdf4
or http://127.0.0.1:8888/?token=1d80ee69da6238f22bb683a4acd00025d32d15dde91cbdf4
[I 10:53:03.628 NotebookApp] 302 GET / (127.0.0.1) 0.600000ms
[I 10:53:03.633 NotebookApp] 302 GET /tree? (127.0.0.1) 1.040000ms
```

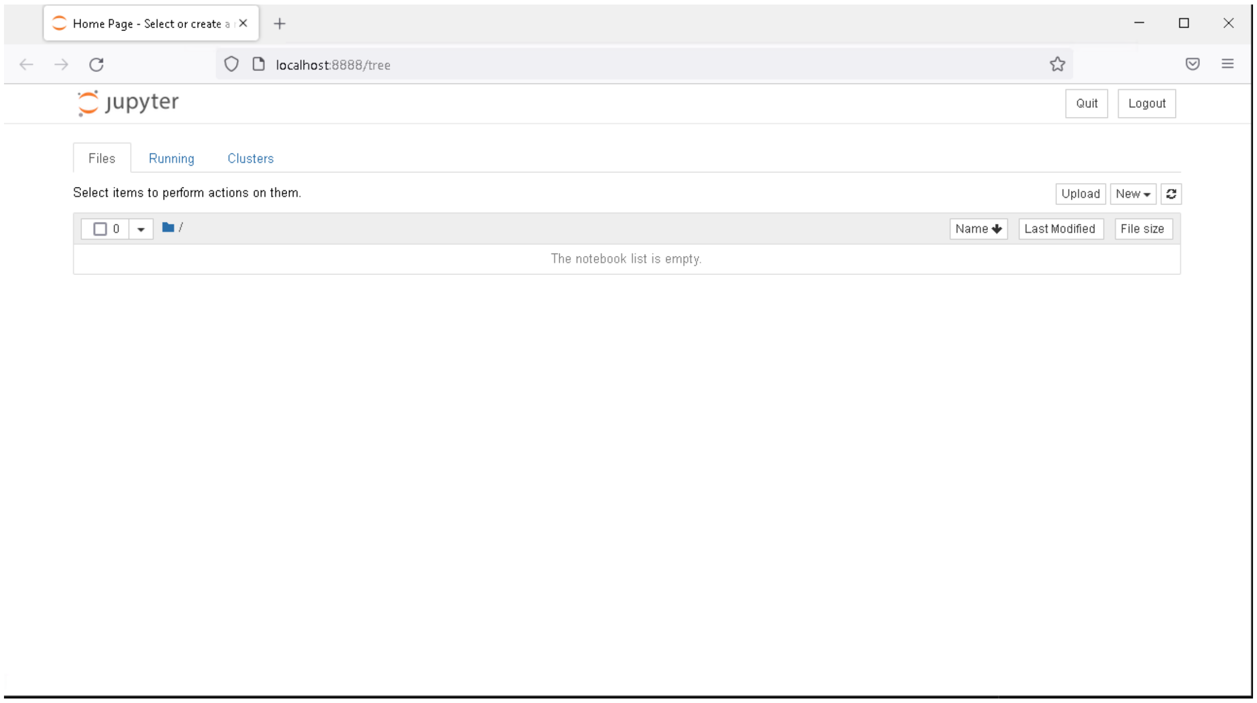
If you look closely at the output, you will see that the Jupyter notebook is now running, and its interface is exposed via its internal web server on localhost, port 8888.

Accessing port 8888

This port can be accessed directly from our Ubuntu WSL instance via curl, lynx (a command line web browser), or any GUI web browser executed in WSL.

However, this tutorial is not about running Linux GUI applications from WSL (which you can do), or visualising data in the command line, but interoperability! So, as you can guess, any network port exposed locally is also available from Windows directly, if no conflict occurs.

Let's try this right away: from Windows, launch a web browser and enter the URL `printed above` with the corresponding token, for example: `http://localhost:8888/?token=1d80ee69da6238f22bb683a4acd00025d32d15dde91cbdf4`.



Select items to perform actions on them.

The notebook list is empty.

And it works! You can thus easily expose and share any services that are using network ports between your Windows machine and WSL instances!

> **Note:** you need to keep this command line Window opened to have your Jupyter instance running. If you close it, the service will shut down and you won't have access to it anymore. Other command-line operations in the same WSL instance should be done on another terminal.

Get access to Windows environment variables on WSL

Our next step is to be able to generate some statistics on our Windows user personal directory. For Jupyter to access those, we need to know how to get access from our Ubuntu instance to the Windows partition(s). You may already know about it by reading documentation or blog posts, but let's do something even cooler: let's discover it by using environment variables!

On another terminal, under PowerShell, let's first check our Windows user profile directory:

```
PS C:\Users\myuser> echo $env:USERPROFILE
C:\Users\myuser
```

Let's share it with Ubuntu by setting `WSLENV`:

```
PS C:\Users\myuser> $env:WSLENV="USERPROFILE"
PS C:\Users\myuser> ubuntu.exe
$ echo $USERPROFILE
C:\Users\myuser
```

Awesome! Setting `WSLENV="ENVVAR1:ENVVAR2:..."` allows us to share multiple environment variables from Windows to our Ubuntu instance. We can then retrieve the value under Ubuntu.

However, you may notice that the environment variable value was shared as is, which is fine in most cases but not for path-related content. Let's check:

```
$ ls 'C:\Users\myuser'
ls: cannot access 'C:\Users\myuser': No such file or directory
```

Indeed, `C:\Users\myuser` is not a compatible Linux-path where the Windows file system is located under WSL.

But we haven't done all that for nothing! `WSLENV` variable declaration can be suffixed with `/p`, which then translates any paths between Windows and your Linux instance.

Let's try again:

```
$ exit
PS C:\Users\myuser> $env:WSLENV="USERPROFILE/p"
PS C:\Users\myuser> ubuntu.exe
$ echo $USERPROFILE
/mnt/c/Users/myuser
$ ls /mnt/c/Users/myuser
AppData
'Application Data'
Contacts
Cookies
Desktop
[...]
```

And here we go! We now know where our user profile data is accessible on WSL thanks to environment variables sharing. But more generally, environment variables could be used in your scripts, or any services in your WSL instance, where parameters are controlled from Windows.

Going further:

- There are many other flags available for environment variables sharing via `WSLENV`. Check out the reference section to have a look at some handy links explaining each of them.
- The place where your Windows filesystems are mounted can vary depending on your configuration. This can be changed with our configuration tool on Ubuntu 22.04+ or by modifying the `automount` section in wsl.conf.

With this, we are now ready to generate some statistics on your Windows user profile directory from our WSL instance!

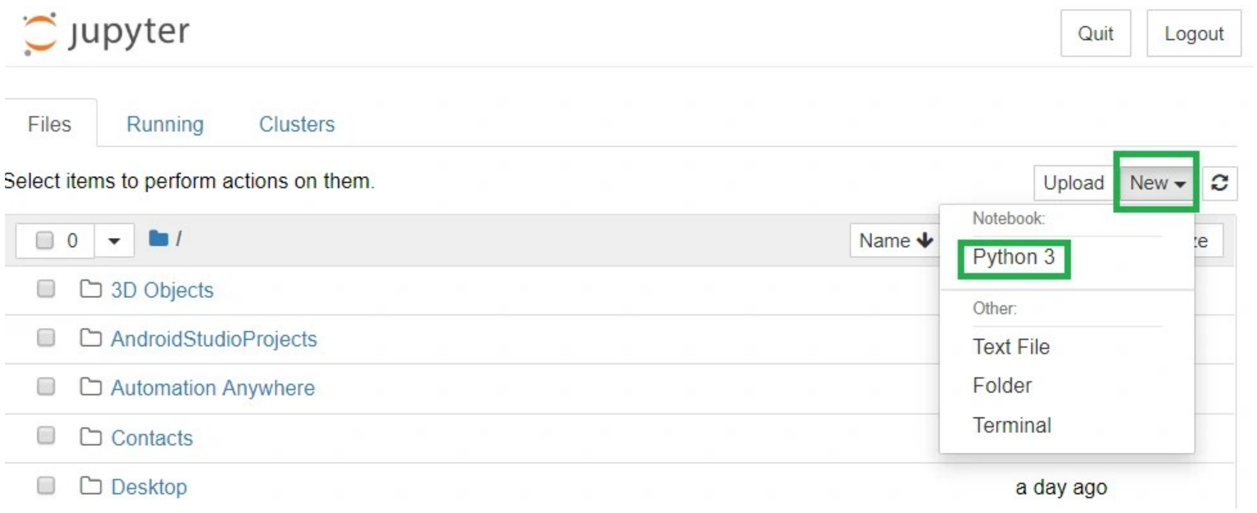
Accessing Windows file from Ubuntu to run our script

After this little detour into the command line world to discover which path to use, let's go back to our Jupyter interface in our browser running on Windows.

We are going to create a `stats-raw.csv` file, containing statistics about our user profile directory.

Some preliminary warnings: accessing Windows filesystem from Ubuntu is using the 9P protocol, which might mean slower access and indexing of files than native performance. So that this section doesn't take too long to complete, we are advising you to choose a subdirectory or your Windows user profile directory, with fewer files and directories to run over. We will call it here `/mnt/c/Users/mysuser/path/my/subdirectory`.

From the Jupyter main screen, create a new notebook to start developing an interactive Python solution. You can do this by clicking on the `New` button, and then clicking on the `Python 3` option, as we can see below.



Copy this to the first cell, adapting the input directory:

```
import os
import mimetypes
import csv

data = {}

for root, dirs, files in os.walk("/mnt/c/Users/mysuser/path/my/subdirectory"):
  for f in files:
    mime_type, encoding = mimetypes.guess_type(os.path.join(root, f))
    if not mime_type:
      continue
    if mime_type not in data:
      data[mime_type] = 0
    data[mime_type] += 1

csv_cols = ["mime_type", "count"]
with open("stats-raw.csv", "w") as f:
  writer = csv.writer(f)
  writer.writerow(csv_cols)
  for mime_type, count in data.items():
    writer.writerow([mime_type, count])
```

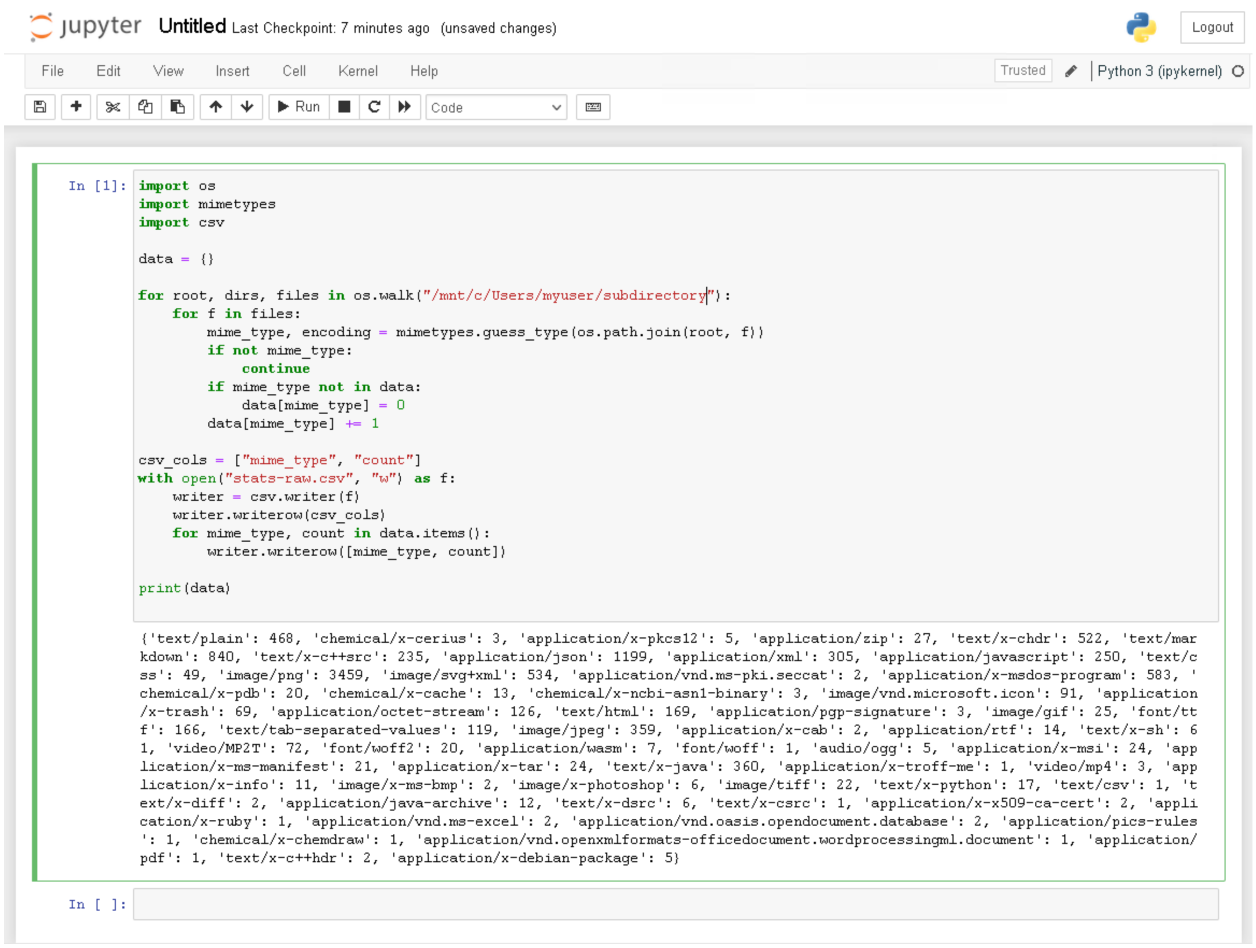
(continues on next page)

(continued from previous page)

```
print(data)
```

This script will enumerate all files under `/mnt/c/Users/mysuser/path/my/subdirectory`, get the mime types of each entry, and count them. It will write the result in your Ubuntu user home directory as a Comma-separated Values file named `stats-raw.csv` and print it as well for your viewing pleasure. :)

Let's execute it by clicking on the "Run" button in the web interface.


jupyter
Untitled Last Checkpoint: 7 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [1]: import os
        import mimetypes
        import csv

        data = {}

        for root, dirs, files in os.walk("/mnt/c/Users/myuser/subdirectory"):
            for f in files:
                mime_type, encoding = mimetypes.guess_type(os.path.join(root, f))
                if not mime_type:
                    continue
                if mime_type not in data:
                    data[mime_type] = 0
                data[mime_type] += 1

        csv_cols = ["mime_type", "count"]
        with open("stats-raw.csv", "w") as f:
            writer = csv.writer(f)
            writer.writerow(csv_cols)
            for mime_type, count in data.items():
                writer.writerow([mime_type, count])

        print(data)
```

```
{'text/plain': 468, 'chemical/x-cerius': 3, 'application/x-pkcs12': 5, 'application/zip': 27, 'text/x-chdr': 522, 'text/mar
kdown': 840, 'text/x-c++src': 235, 'application/json': 1199, 'application/xml': 305, 'application/javascript': 250, 'text/c
ss': 49, 'image/png': 3459, 'image/svg+xml': 534, 'application/vnd.ms-pki.seccat': 2, 'application/x-msdos-program': 583, '
chemical/x-pdb': 20, 'chemical/x-cache': 13, 'chemical/x-ncbi-asn1-binary': 3, 'image/vnd.microsoft.icon': 91, 'application
/x-trash': 69, 'application/octet-stream': 126, 'text/html': 169, 'application/pgp-signature': 3, 'image/gif': 25, 'font/tt
f': 166, 'text/tab-separated-values': 119, 'image/jpeg': 359, 'application/x-cab': 2, 'application/rtf': 14, 'text/x-sh': 6
1, 'video/MP2T': 72, 'font/woff2': 20, 'application/wasm': 7, 'font/woff': 1, 'audio/ogg': 5, 'application/x-msi': 24, 'app
lication/x-ms-manifest': 21, 'application/x-tar': 24, 'text/x-java': 360, 'application/x-troff-me': 1, 'video/mp4': 3, 'app
lication/x-info': 11, 'image/x-ms-bmp': 2, 'image/x-photoshop': 6, 'image/tiff': 22, 'text/x-python': 17, 'text/csv': 1, 't
ext/x-diff': 2, 'application/java-archive': 12, 'text/x-dsrc': 6, 'text/x-csrc': 1, 'application/x-x509-ca-cert': 2, 'appli
cation/x-ruby': 1, 'application/vnd.ms-excel': 2, 'application/vnd.oasis.opendocument.database': 2, 'application/pics-rules
': 1, 'chemical/x-chemdraw': 1, 'application/vnd.openxmlformats-officedocument.wordprocessingml.document': 1, 'application/
pdf': 1, 'text/x-c++hdr': 2, 'application/x-debian-package': 5}
```

```
In [ ]:
```

Note that while the entry is running, you will have a `In [*]` with the star marker. This will be replaced by `In [1]:` when completed. Once done and the results are printed, let's ensure that the CSV file is present on disk using an Ubuntu terminal:

```
$ cat stats-raw.csv
mime_type,count
text/plain,468
chemical/x-cerius,3
application/x-pkcs12,5
application/zip,27
text/x-chdr,522
text/markdown,840
text/x-c++src,235
application/json,1199
```

(continues on next page)

(continued from previous page)

```
application/xml,305
application/javascript,250
text/css,49
image/png,3459
image/svg+xml,534
application/vnd.ms-pki.seccat,2
application/x-msdos-program,583
chemical/x-pdb,20
chemical/x-cache,13
chemical/x-ncbi-asn1-binary,3
image/vnd.microsoft.icon,91
[...]
```

Accessing Ubuntu files from Windows

So, we now have a stat file on Ubuntu, which is the result of computation on files stored on the Windows partition. We now want to analyse this file using Windows tools, but first, can we access it from Windows?

Of course, interoperability goes both ways, and we already know exactly how to discover where those are available on Windows: introducing sharing environment variable round 2!

Start PowerShell from Ubuntu and share the HOME directory

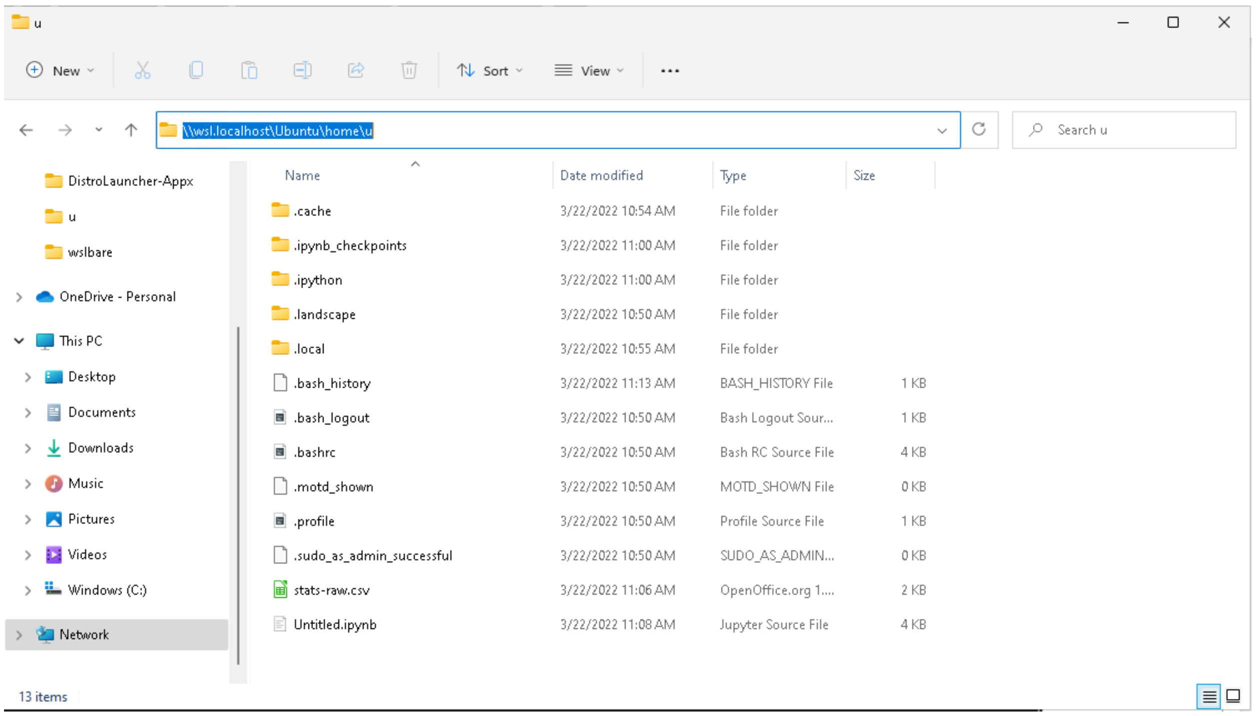
Similarly to `USERPROFILE`, we want, this time, to share the user `HOME` variable with Windows, and let interoperability translate it to a Windows-compatible path. Let's do this right away from an Ubuntu terminal:

```
$ export WSLENV=HOME/p
$ cmd.exe
C:\Windows> set HOME
HOME=\\wsl.localhost\Ubuntu\home\u
C:\Windows> exit
```

First, we are able to export the `HOME` variable to subprocess, telling us that we want to translate the path back to Windows compatible paths by appending `/p` as we previously saw. But this is not all: we are running `cmd.exe` from an Ubuntu terminal (which is itself running inside a PowerShell terminal), and get the corresponding Windows path! Even if that sounds a little bit like inception, using this feature is just seamless: you are launching any process, Linux or Windows, from your Ubuntu terminal. Inputs and outputs are connected and this complex machinery works flawlessly!

Accessing Linux files from Windows

Open Windows Explorer and navigate to that path to confirm they are visible there:



\\wsl.localhost\Ubuntu\home\u

Name	Date modified	Type	Size
.cache	3/22/2022 10:54 AM	File folder	
.ipynb_checkpoints	3/22/2022 11:00 AM	File folder	
.ipython	3/22/2022 11:00 AM	File folder	
.landscape	3/22/2022 10:50 AM	File folder	
.local	3/22/2022 10:55 AM	File folder	
.bash_history	3/22/2022 11:13 AM	BASH_HISTORY File	1 KB
.bash_logout	3/22/2022 10:50 AM	Bash Logout Sour...	1 KB
.bashrc	3/22/2022 10:50 AM	Bash RC Source File	4 KB
.motd_shown	3/22/2022 10:50 AM	MOTD_SHOWN File	0 KB
.profile	3/22/2022 10:50 AM	Profile Source File	1 KB
.sudo_as_admin_successful	3/22/2022 10:50 AM	SUDO_AS_ADMIN...	0 KB
stats-raw.csv	3/22/2022 11:06 AM	OpenOffice.org 1....	2 KB
Untitled.ipynb	3/22/2022 11:08 AM	Jupyter Source File	4 KB

13 items

Let's now create a PowerShell script, from Windows, on this Ubuntu filesystem and save it there:

You can open any editor, from notepad to a full-fledged IDE. Create a file named `filter-less-than-five.ps1` under `\\wsl.localhost\Ubuntu\home\<youruser>` (with the following content:

```
$csvImport = $input | ConvertFrom-CSV

# Create Array for Exporting out data
$csvArray = [System.Collections.ArrayList]@()

# Filter lines where count less than 5
Foreach ($csvImportedItem in $csvImport){
  if([int]$csvImportedItem.count -ge 5){
      $csvArray += $csvImportedItem
  }
}

$csvArray | convertTo-CSV -NoTypeInformation
```

This script will take a CSV-formatted content as input, filter any item which has less than 5 occurrences and will export it to the standard output as another CVS-formatted content.

After saving, let's check it's available on the WSL side:

```
$ cat filter-less-than-five.ps1
```

You should see the file content there.

This PowerShell script, written from Windows on your Linux instance will be quite handy to create a pipeline between applications.

Execute and connect Linux and Windows executables.

This is all very impressive, we have been able to share network, environment variables, paths, and files, and execute processes interchangeably between Ubuntu and Windows. Let's go one step further by chaining all of this together in a single, but effective line:

```
$ cat stats-raw.csv | powershell.exe -ExecutionPolicy Bypass -File $HOME/filter-less-
↪than-five.ps1 | tee stats.csv
"mime_type","count"
"text/plain","468"
"application/x-pkcs12","5"
"application/zip","27"
"text/x-chdr","522"
"text/markdown","840"
"text/x-c++src","235"
"application/json","1199"
"application/xml","305"
[...]
```

Let's figure out what happens here:

1. We are executing a Linux application `cat` to display file content, hosted on Ubuntu.
2. We are then executing `powershell.exe` from Ubuntu which:
3. Takes as input the content piped from Ubuntu.
4. This PowerShell application uses a script, hosted on Ubuntu (that we wrote from Windows in the previous section), converted transparently to a Windows path for PowerShell to be able to consume it.
5. Then, the script proceeds with some operations on the content and prints on stdout the filtered CSV data.
6. This is then piped back to Ubuntu, to be processed by the `tee` command, which writes `stats.csv` to the Ubuntu filesystem, and displays the resulting output.

This simple command exercises many concepts of interoperability we saw in previous sections. However, as you can see, this is completely transparent to us!

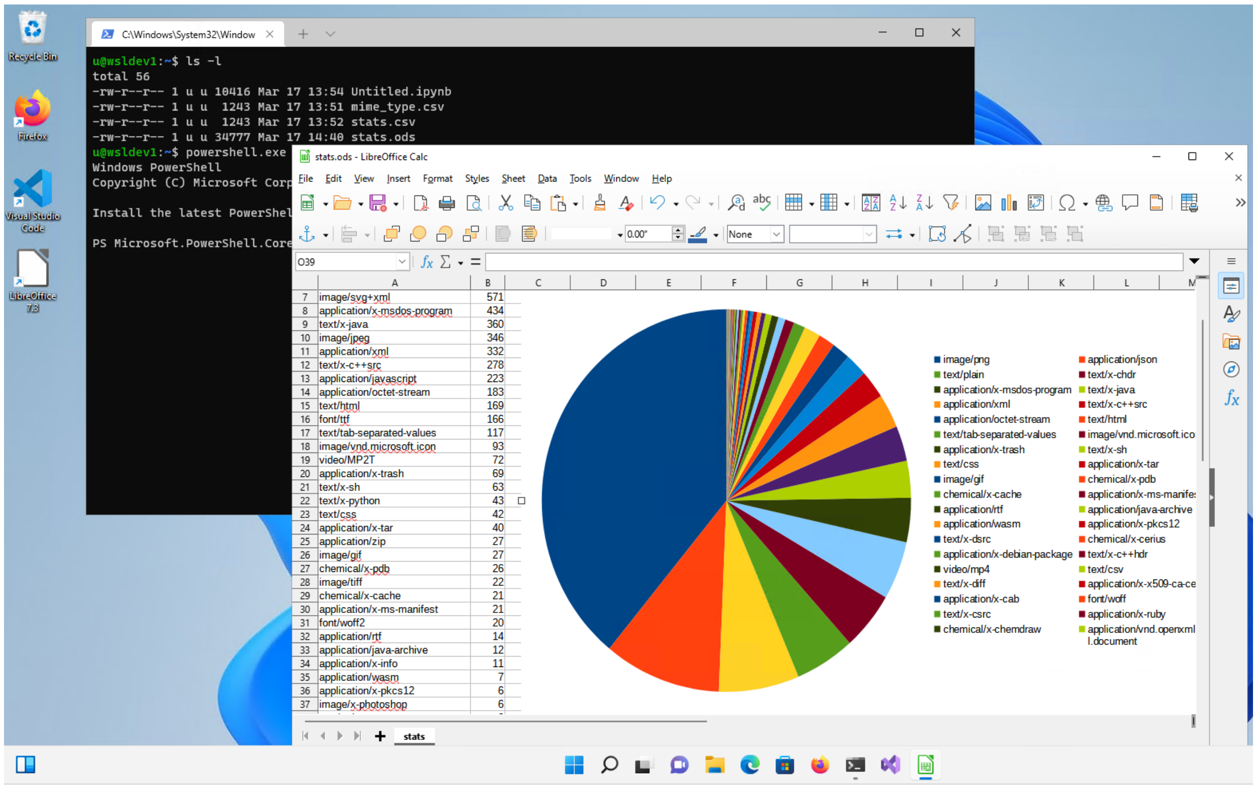
This deep integration for back-and-forth access between systems allows users to create awesome pipelines, taking the best tool that is available, independent of their host operating system. To make that easy, WSL transparently converts any paths and does the heavy lifting for you so that you don't need to do the manual conversion!

And finally, we can even run the default associated GUI Windows application associated with those files, from Ubuntu:

```
$ explorer.exe stats.csv
```

Note: You can't deny it's really amazing to be able to execute explorer.exe from Ubuntu. :)

This will open LibreOffice, Microsoft Excel, or any other tool you may have associated with CSV files. From there, you will be able to draw beautiful charts, make further analyses, and so on. But that's another story…



```
u@wsldev1:~$ ls -l
total 56
-rw-r--r-- 1 u u 10416 Mar 17 13:54 Untitled.ipynb
-rw-r--r-- 1 u u  1243 Mar 17 13:51 mime_type.csv
-rw-r--r-- 1 u u  1243 Mar 17 13:52 stats.csv
-rw-r--r-- 1 u u 34777 Mar 17 14:40 stats.ods
u@wsldev1:~$ powershell.exe
Windows PowerShell
Copyright (C) Microsoft Corp

Install the latest PowerShel

PS Microsoft.PowerShell.Core
```

	A	B
7	image/svg+xml	571
8	application/x-msdos-program	434
9	text/x-java	360
10	image/jpeg	346
11	application/xml	332
12	text/x-c++src	278
13	application/javascript	223
14	application/octet-stream	183
15	text/html	169
16	font/ttf	166
17	text/tab-separated-values	117
18	image/vnd.microsoft.icon	93
19	video/MP2T	72
20	application/x-trash	69
21	text/x-sh	63
22	text/x-python	43
23	text/css	42
24	application/x-tar	40
25	application/zip	27
26	image/gif	27
27	chemical/x-pdb	26
28	image/tiff	22
29	chemical/x-cache	21
30	application/x-ms-manifest	21
31	font/woff2	20
32	application/rtf	14
33	application/java-archive	12
34	application/x-info	11
35	application/wasm	7
36	application/x-pkcs12	6
37	image/x-photoshop	6

Enjoy Ubuntu on WSL!

That's all folks! In this tutorial, we've shown you many aspects of interoperability on WSL. To sum it up, we can:

- **Run Ubuntu commands from a Windows PowerShell prompt** such as cut, grep, or awk
- **Run Windows commands from an Ubuntu Terminal** such as explorer.exe or notepad.exe
- **Share network ports** between Ubuntu and Windows systems.
- **Share environment variables** between Ubuntu and Windows systems.
- **Open files** on the `Windows` file system from `Ubuntu`.
- Browse the Ubuntu file system **from Windows Explorer**.
- **Pipe and connect any commands or applications** between Ubuntu and Windows systems.

We hope you enjoy using Ubuntu inside WSL. Don't forget to check out our other tutorials for tips on how to optimise your WSL experience.

Further Reading

- WSL ENV documentation
- WSL.conf documentation
- Jupyter notebook
- *Install Ubuntu on WSL2*
- Microsoft WSL Documentation

- Ask Ubuntu

2.1.3 Run a .Net Echo Bot as a systemd service on Ubuntu WSL

Authored by Oliver Smith (oliver.smith@canonical.com)

.Net is an open-source development platform from Microsoft that enables developers to build multi-platform applications from a single codebase.

In this tutorial we demonstrate how easy it is to start working with .Net on Ubuntu WSL by creating a simple chatbot accessible from your Windows host. We also take advantage of WSL's new systemd support to run our bot as a systemd service for easier deployment.

Requirements

- A PC running Windows 11
- The latest version of the WSL Windows Store application
- Ubuntu, Ubuntu 22.04 LTS or Ubuntu Preview installed

Systemd support is a new feature of WSL and only available on WSL version XX or higher.

You can check your current WSL version by running:

```
> wsl -- version
```

In your PowerShell terminal.

To enable systemd on your Ubuntu distribution you need to add the following content to `/etc/wsl.conf`:

```
[boot] systemd=true
```

Make sure to restart your distribution after you have made this change.

Install .Net

.Net has recently been added to the Ubuntu repositories. This means you can now quickly install a bundle with both the SDK and runtime with a single command:

```
$ sudo apt install dotnet6
```

Run `dotnet --version` to confirm that the package was installed successfully.

It is recommended to create a new directory for this project, do so now and navigate to it before proceeding:

```
$ sudo mkdir ~/mybot $ cd mybot
```

Install and run the Bot Framework EchoBot template

Create a new directory for the project and navigate to it before proceeding:

```
$ sudo mkdir ~/mybot $ cd mybot
```

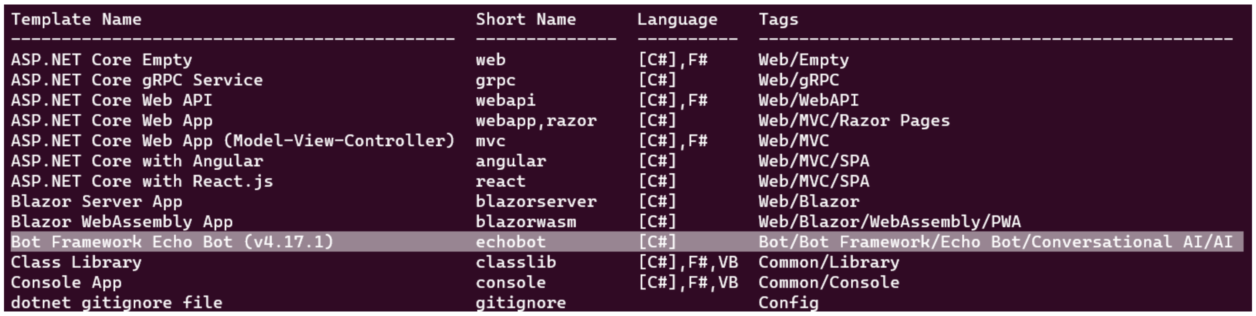
Once inside we can install the EchoBot C# template by running:

```
$ dotnet new -i Microsoft.Bot.Framework.CSharp.EchoBot
```

We can verify the template has been installed correctly by running:

```
$ dotnet new --list
```

And looking for the `Bot Framework Echo Bot` template in the list.



```
Template Name                                 Short Name      Language    Tags
--------------------------------------------  --------------  ----------  ----------------------------------------------
ASP.NET Core Empty                            web             [C#],F#     Web/Empty
ASP.NET Core gRPC Service                     grpc            [C#]        Web/gRPC
ASP.NET Core Web API                          webapi          [C#],F#     Web/WebAPI
ASP.NET Core Web App                          webapp,razor    [C#]        Web/MVC/Razor Pages
ASP.NET Core Web App (Model-View-Controller)  mvc             [C#],F#     Web/MVC
ASP.NET Core with Angular                     angular         [C#]        Web/MVC/SPA
ASP.NET Core with React.js                    react           [C#]        Web/MVC/SPA
Blazor Server App                             blazorserver    [C#]        Web/Blazor
Blazor WebAssembly App                        blazorwasm      [C#]        Web/Blazor/WebAssembly/PWA
Bot Framework Echo Bot (v4.17.1)              echobot         [C#]        Bot/Bot Framework/Echo Bot/Conversational AI/AI
Class Library                                 classlib        [C#],F#,VB  Common/Library
Console App                                   console         [C#],F#,VB  Common/Console
dotnet gitignore file                         gitignore                   Config
```

Create a new Echo Bot project using the following command. We use `echoes` as the name for our bot.

```
$ dotnet new echobot -n echoes
```

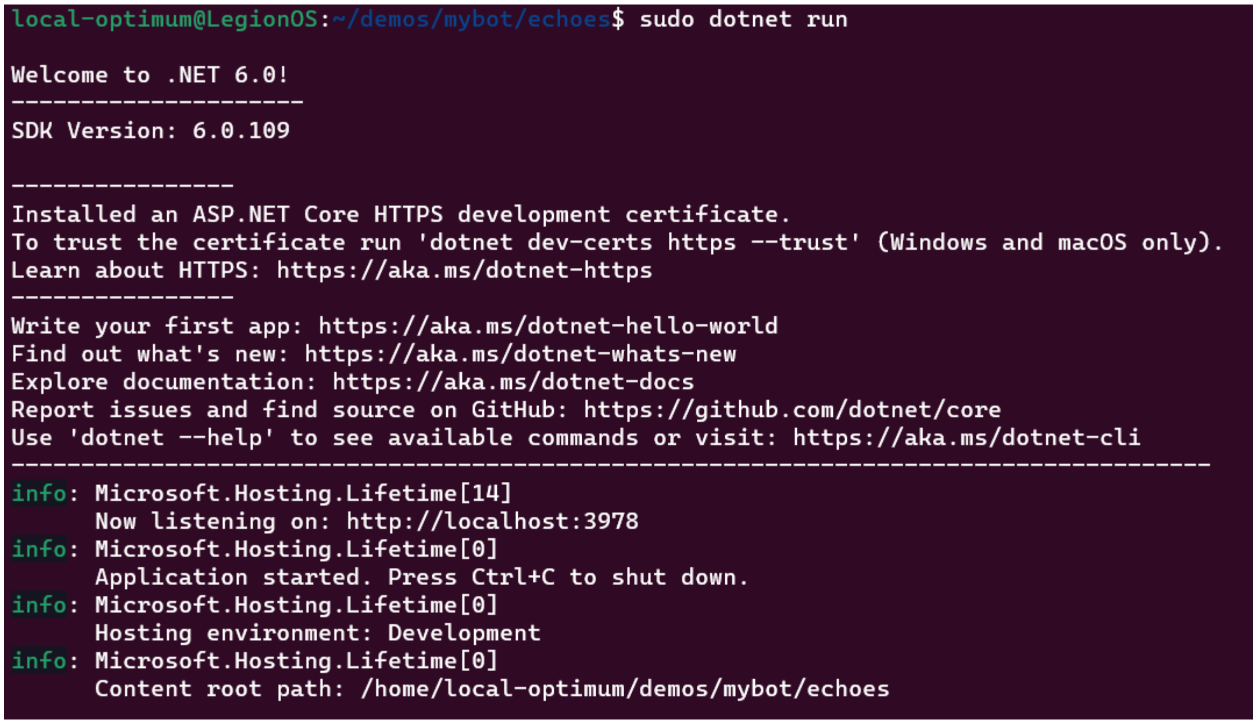
Once this has completed we can navigate into the new directory that has been created.

```
$ cd ~/mybot/echoes
```

Once inside, the project should be ready to run. Test it with:

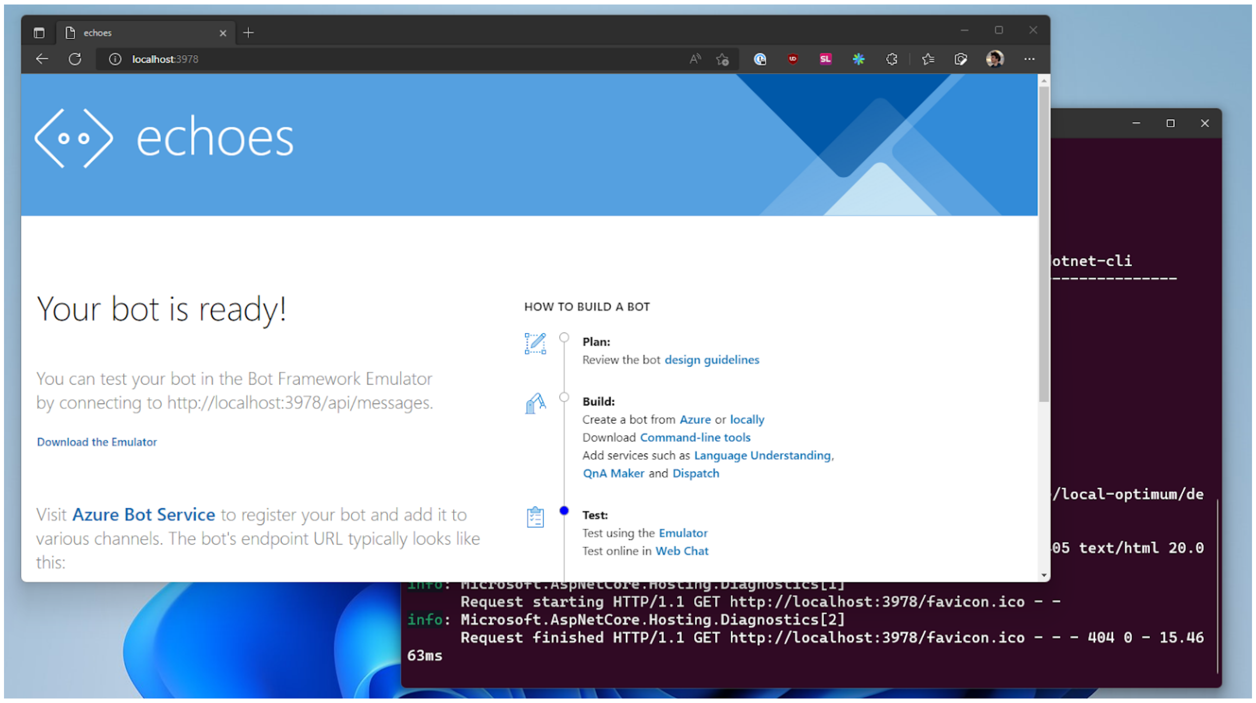
```
$ sudo dotnet run
```

If everything was set up correctly you should see a similar output to the one below:



```
local-optimum@LegionOS:~/demos/mybot/echoes$ sudo dotnet run

Welcome to .NET 6.0!
---------------------
SDK Version: 6.0.109

----------------
Installed an ASP.NET Core HTTPS development certificate.
To trust the certificate run 'dotnet dev-certs https --trust' (Windows and macOS only).
Learn about HTTPS: https://aka.ms/dotnet-https
----------------
Write your first app: https://aka.ms/dotnet-hello-world
Find out what's new: https://aka.ms/dotnet-whats-new
Explore documentation: https://aka.ms/dotnet-docs
Report issues and find source on GitHub: https://github.com/dotnet/core
Use 'dotnet --help' to see available commands or visit: https://aka.ms/dotnet-cli
--------------------------------------------------------------------------------------
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:3978
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /home/local-optimum/demos/mybot/echoes
```

Leave the EchoBot App running in WSL for now. Open a new browser window on your Windows host and navigate to `localhost:3978` where you should see the following window:



echoes

Your bot is ready!

You can test your bot in the Bot Framework Emulator by connecting to http://localhost:3978/api/messages.

Download the Emulator

Visit Azure Bot Service to register your bot and add it to various channels. The bot's endpoint URL typically looks like this:

HOW TO BUILD A BOT

Plan:
Review the bot design guidelines

Build:
Create a bot from Azure or locally
Download Command-line tools
Add services such as Language Understanding, QnA Maker and Dispatch

Test:
Test using the Emulator
Test online in Web Chat

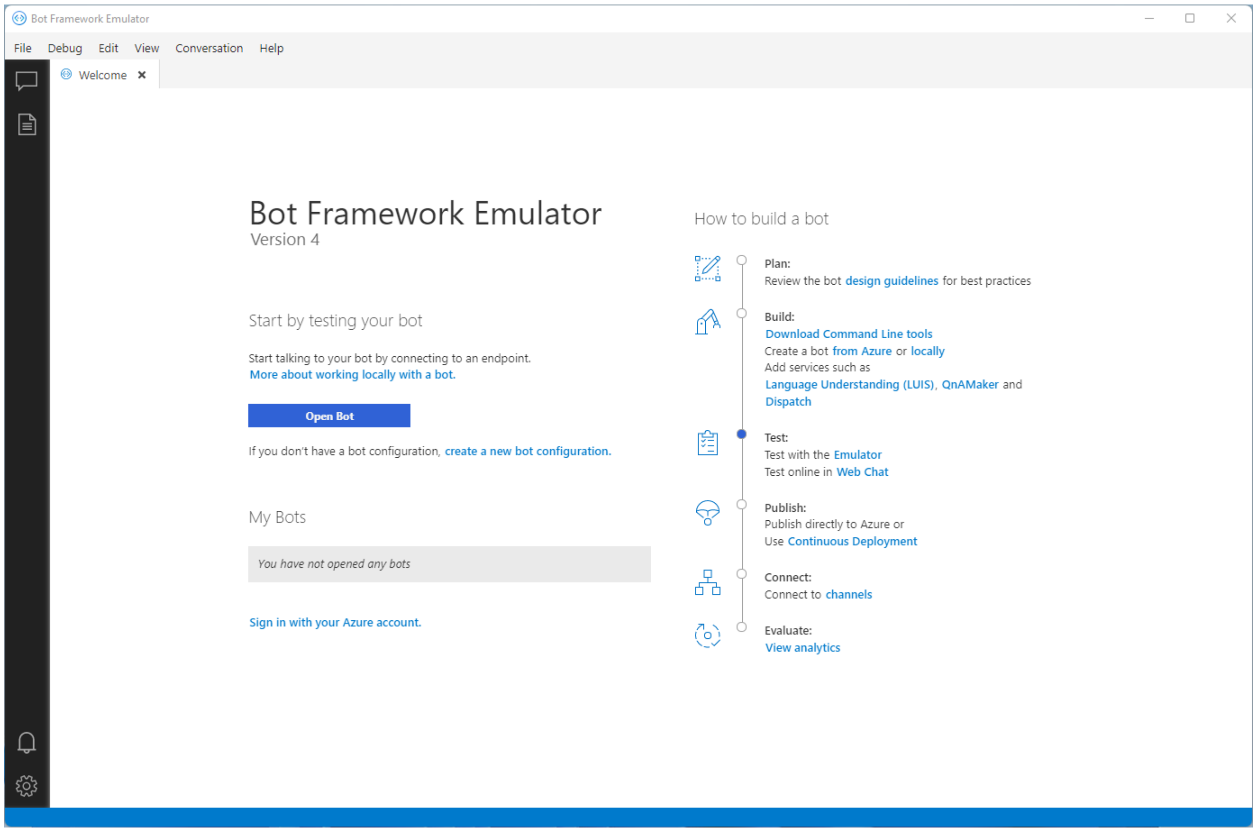
```
otnet-cli
--------------
/local-optimum/de
05 text/html 20.0
info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
      Request starting HTTP/1.1 GET http://localhost:3978/favicon.ico - -
info: Microsoft.AspNetCore.Hosting.Diagnostics[2]
      Request finished HTTP/1.1 GET http://localhost:3978/favicon.ico - - - 404 0 - 15.46
63ms
```

Leave everything running as we move to the next step.

Install the Bot Emulator on Windows and connect to your bot

Download the Bot Emulator from the official Microsoft GitHub, taking care to select BotFramework-Emulator-4.14.1-windows-setup.exe and install.

Running it will present you with the following screen, but before you can connect to your bot you need to change a few settings.



Bot Framework Emulator

File Debug Edit View Conversation Help

Welcome

Bot Framework Emulator
Version 4

Start by testing your bot

Start talking to your bot by connecting to an endpoint.
More about working locally with a bot.

Open Bot

If you don't have a bot configuration, create a new bot configuration.

My Bots

You have not opened any bots

Sign in with your Azure account.

How to build a bot

Plan:
Review the bot design guidelines for best practices

Build:
Download Command Line tools
Create a bot from Azure or locally
Add services such as
Language Understanding (LUIS), QnAMaker and Dispatch

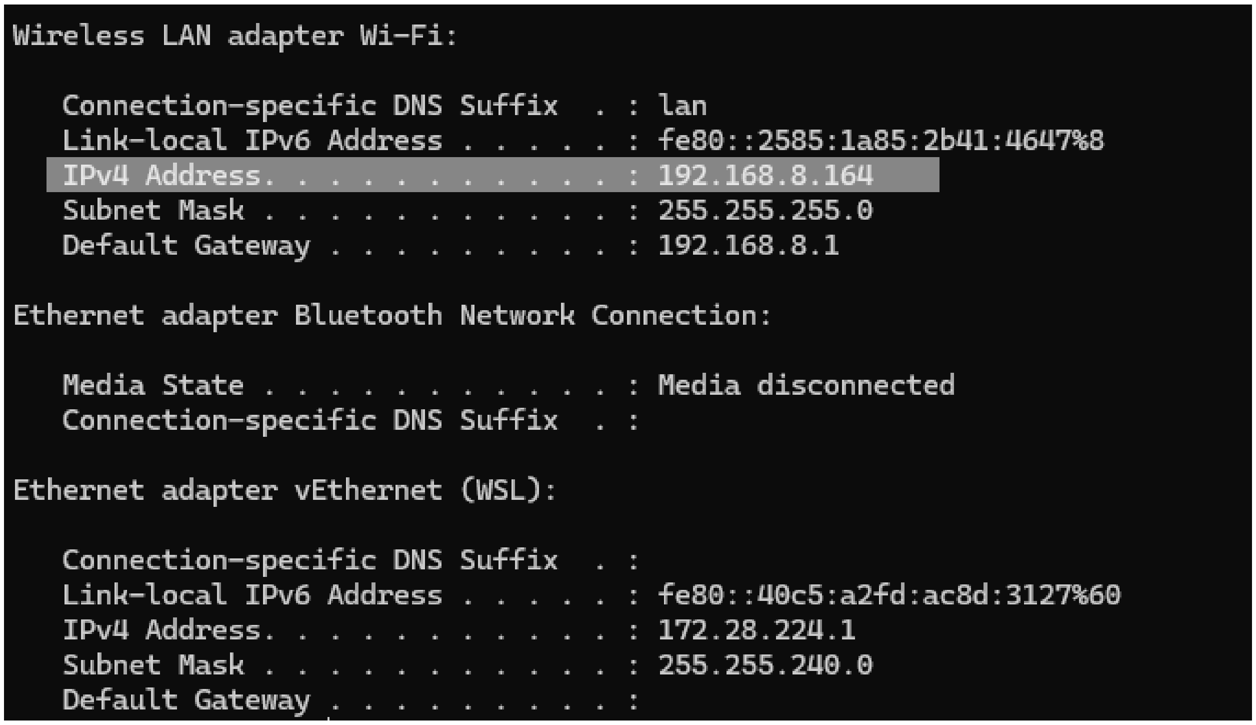
Test:
Test with the Emulator
Test online in Web Chat

Publish:
Publish directly to Azure or
Use Continuous Deployment

Connect:
Connect to channels

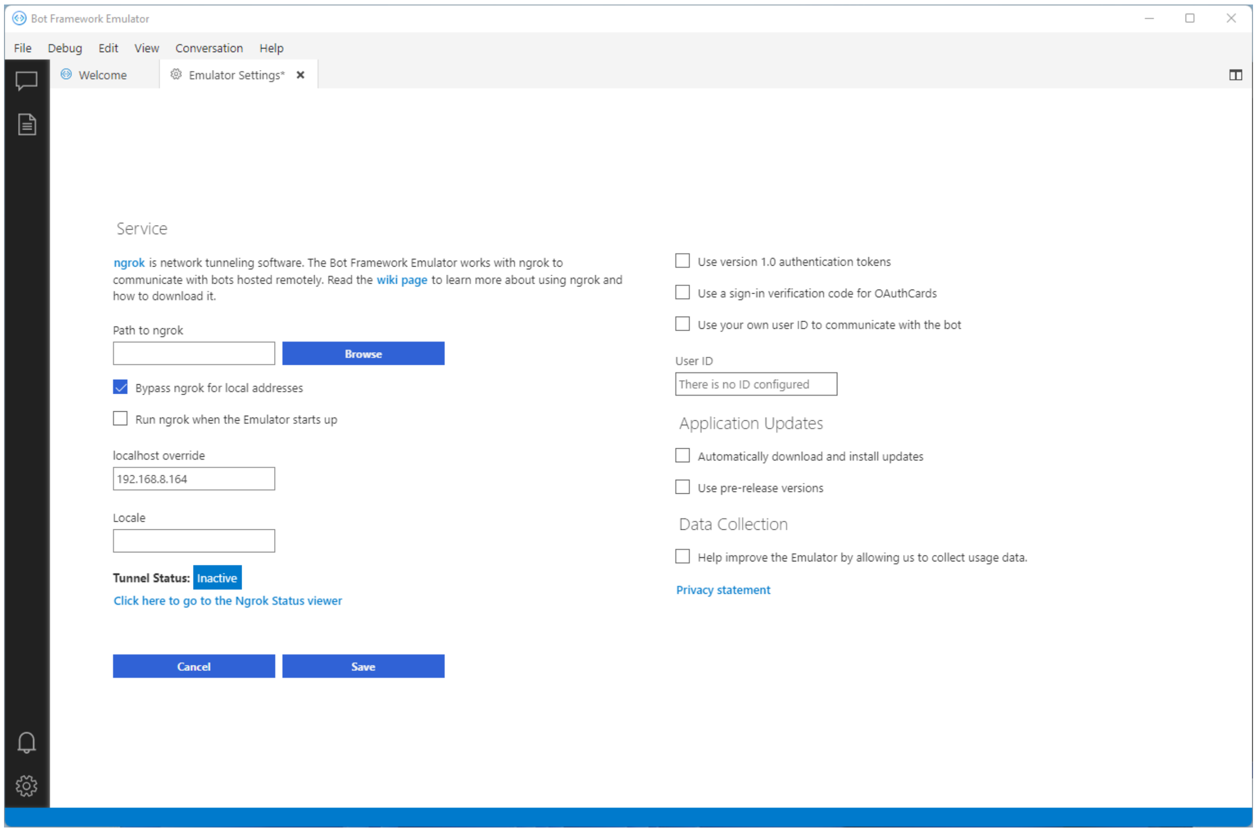
Evaluate:
View analytics

First, get the IP address of your machine by running `ipconfig` in a PowerShell terminal.



```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : lan
   Link-local IPv6 Address . . . . . : fe80::2585:1a85:2b41:4647%8
   IPv4 Address. . . . . . . . . . . : 192.168.8.164
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.8.1

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter vEthernet (WSL):

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::40c5:a2fd:ac8d:3127%60
   IPv4 Address. . . . . . . . . . . : 172.28.224.1
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . :
```

Then select the setting icon in the corner of the Bot Framework Emulator and enter your IP under ‘localhost override’.



Service

ngrok is network tunneling software. The Bot Framework Emulator works with ngrok to communicate with bots hosted remotely. Read the wiki page to learn more about using ngrok and how to download it.

Path to ngrok

[✓] Bypass ngrok for local addresses

[ ] Run ngrok when the Emulator starts up

localhost override

192.168.8.164

Locale

Tunnel Status: Inactive

Click here to go to the Ngrok Status viewer

[ ] Use version 1.0 authentication tokens

[ ] Use a sign-in verification code for OAuthCards

[ ] Use your own user ID to communicate with the bot

User ID

There is no ID configured

Application Updates

[ ] Automatically download and install updates

[ ] Use pre-release versions

Data Collection

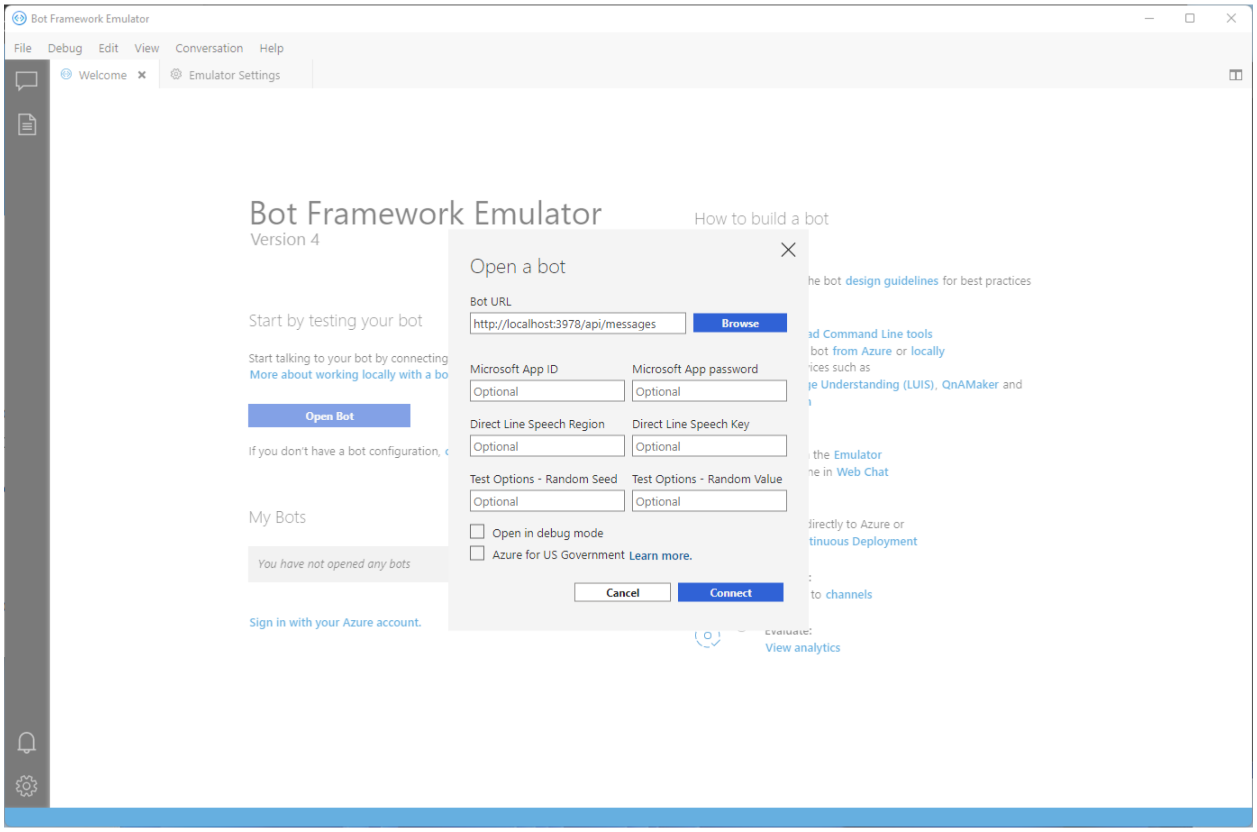
[ ] Help improve the Emulator by allowing us to collect usage data.

Privacy statement

Click save and navigate back to the Welcome tab.

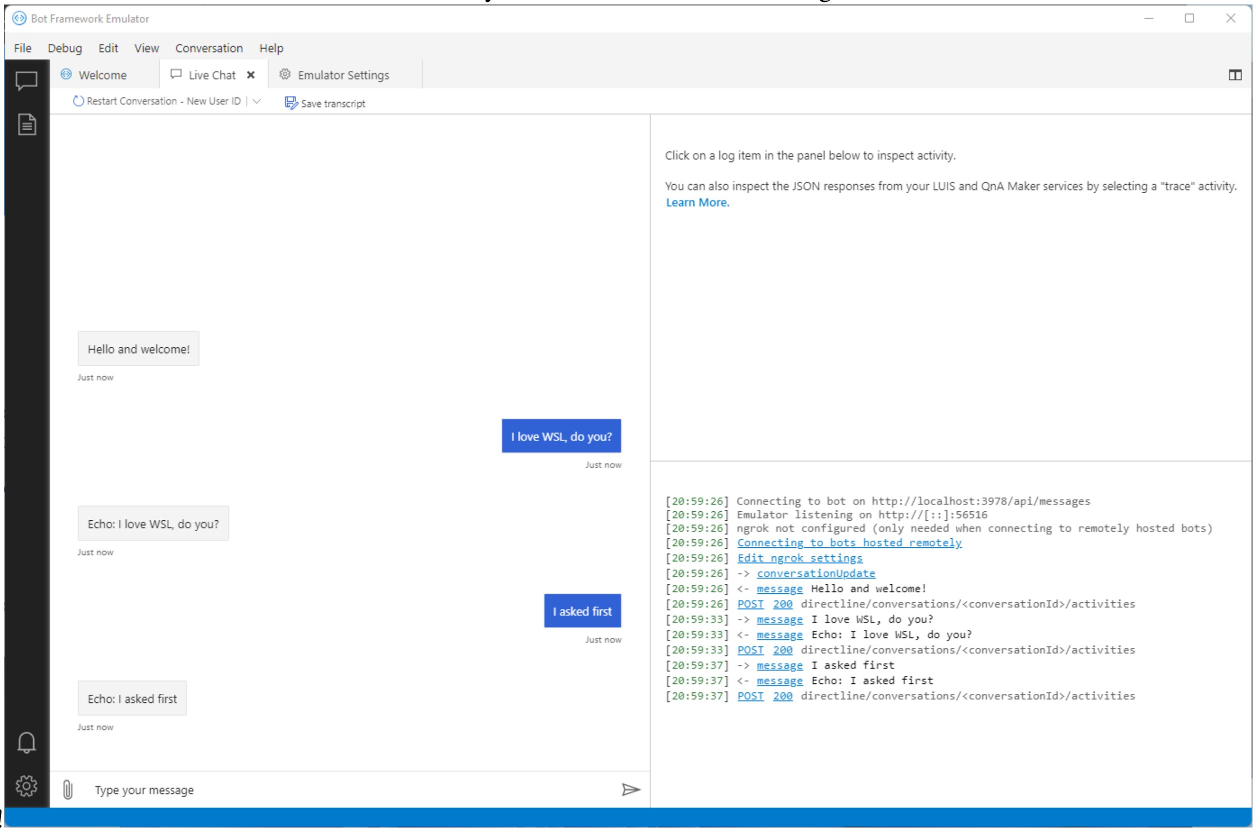
Click ‘Open Bot’ and under ‘Bot URL’ input:

```
http://localhost:3978/api/messages
```


Bot Framework Emulator
File Debug Edit View Conversation Help
Welcome
Emulator Settings
Bot Framework Emulator
Version 4
Start by testing your bot
Start talking to your bot by connecting
More about working locally with a bo
Open Bot
If you don't have a bot configuration,
My Bots
You have not opened any bots
Sign in with your Azure account.
How to build a bot
Open a bot
Bot URL
Browse
Microsoft App ID
Microsoft App password
Direct Line Speech Region
Direct Line Speech Key
Test Options - Random Seed
Test Options - Random Value
Optional
Open in debug mode
Azure for US Government Learn more.
Cancel
Connect

Bot URL: http://localhost:3978/api/messages

And click 'Connect' to connect to your Echo Bot running in WSL and start chatting!



Hello and welcome!
Just now

I love WSL, do you?
Just now

Echo: I love WSL, do you?
Just now

I asked first
Just now

Echo: I asked first
Just now

Click on a log item in the panel below to inspect activity.

You can also inspect the JSON responses from your LUIS and QnA Maker services by selecting a "trace" activity. Learn More.

```
[20:59:26] Connecting to bot on http://localhost:3978/api/messages
[20:59:26] Emulator listening on http://[::]:56516
[20:59:26] ngrok not configured (only needed when connecting to remotely hosted bots)
[20:59:26] Connecting to bots hosted remotely
[20:59:26] Edit ngrok settings
[20:59:26] -> conversationUpdate
[20:59:26] <- message Hello and welcome!
[20:59:26] POST 200 directline/conversations/<conversationId>/activities
[20:59:33] -> message I love WSL, do you?
[20:59:33] <- message Echo: I love WSL, do you?
[20:59:33] POST 200 directline/conversations/<conversationId>/activities
[20:59:37] -> message I asked first
[20:59:37] <- message Echo: I asked first
[20:59:37] POST 200 directline/conversations/<conversationId>/activities
```

Congratulations, your Echo Chat Bot App is running on Ubuntu WSL as an App. Now it is time to make it run as a service.

Running your Echo Bot as a systemd service

Return to your running WSL distro and end the app with `Ctrl+C`.

Then install the .Net systemd extension with:

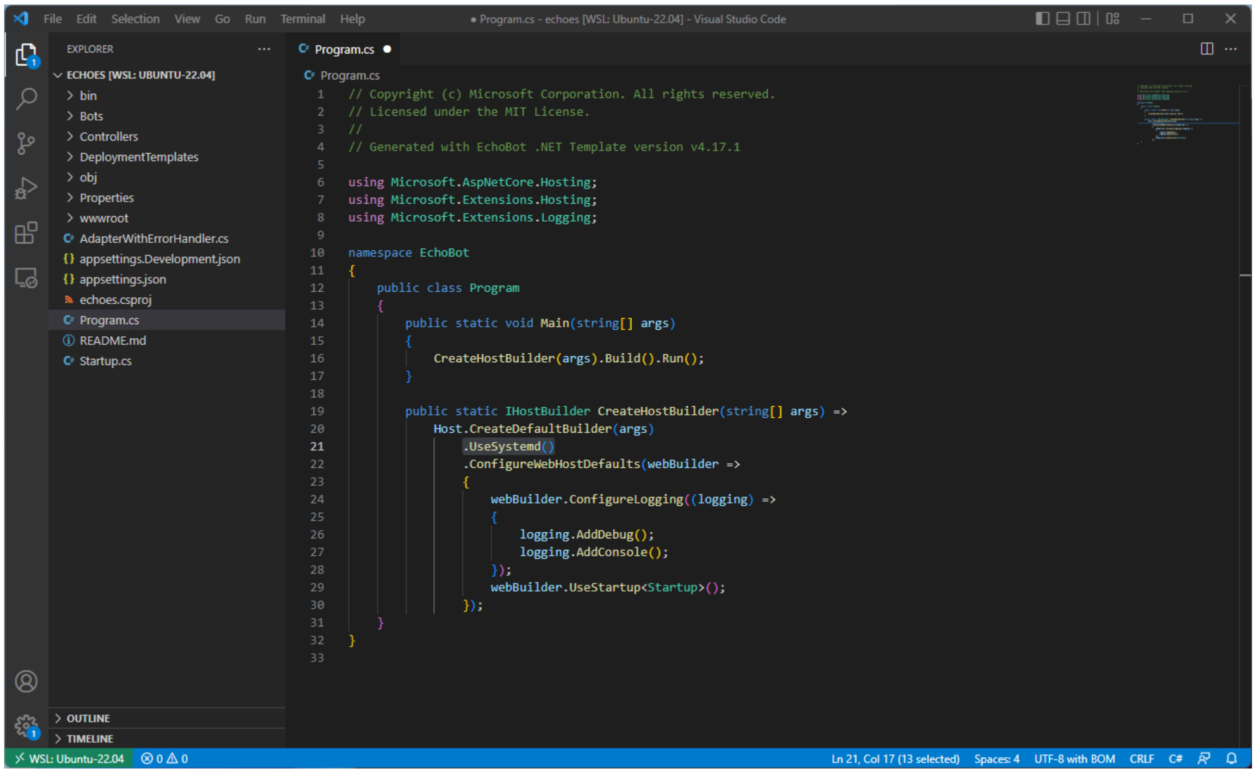
```
$ sudo dotnet add package Microsoft.Extensions.Hosting.Systemd
```

Now we need to open our 'echoes' project in VS Code by running:

```
$ code .
```

From the 'echoes' directory.

Navigate to 'Program.cs' and insert `.UserSystemd()` as a new line in the location shown in the screenshot.


File Edit Selection View Go Run Terminal Help
● Program.cs - echoes [WSL: Ubuntu-22.04] - Visual Studio Code
EXPLORER
ECHOES [WSL: UBUNTU-22.04]
bin
Bots
Controllers
DeploymentTemplates
obj
Properties
wwwroot
AdapterWithErrorHandler.cs
appsettings.Development.json
appsettings.json
echoes.csproj
Program.cs
README.md
Startup.cs
OUTLINE
TIMELINE
Program.cs
WSL: Ubuntu-22.04
Ln 21, Col 17 (13 selected) Spaces: 4 UTF-8 with BOM CRLF C#

```
// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT License.
//
// Generated with EchoBot .NET Template version v4.17.1

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace EchoBot
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .UseSystemd()
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.ConfigureLogging((logging) =>
                    {
                        logging.AddDebug();
                        logging.AddConsole();
                    });
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

Save and close the project in VS code and return to your WSL terminal.

Next we need to create a service file for your bot using your favourite editor, for example.

```
$ sudo nano /etc/systemd/system/echoes.service
```

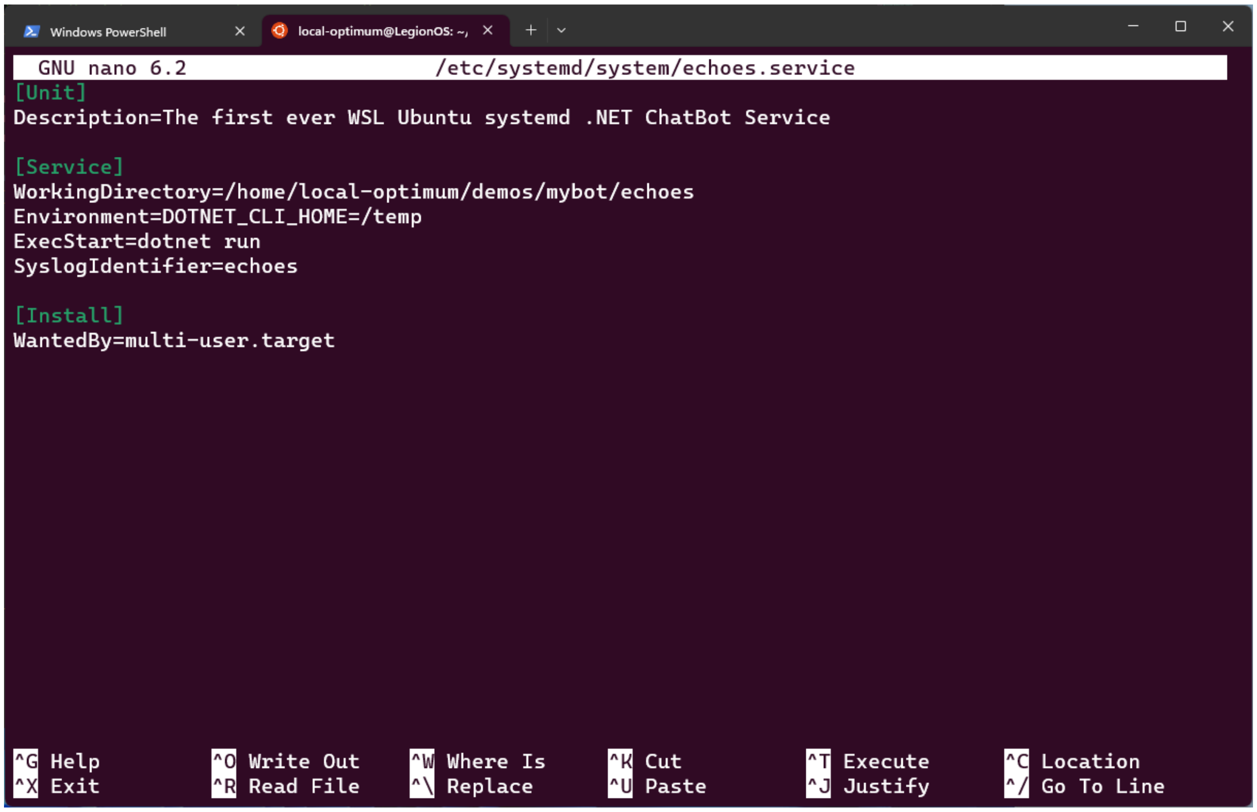
Then paste the below taking care to change the path of `WorkingDirectory` to the location of your project folder.

```
[Unit]
Description=The first ever WSL Ubuntu systemd .NET ChatBot Service

[Service]
WorkingDirectory=/home/local-optimum/demos/mybot/echoes
Environment=DOTNET_CLI_HOME=/temp
```

(continues on next page)

(continued from previous page)

```
ExecStart=dotnet run
SyslogIdentifier=echoes

[Install]
WantedBy=multi-user.target
```



```
  GNU nano 6.2                    /etc/systemd/system/echoes.service
[Unit]
Description=The first ever WSL Ubuntu systemd .NET ChatBot Service

[Service]
WorkingDirectory=/home/local-optimum/demos/mybot/echoes
Environment=DOTNET_CLI_HOME=/temp
ExecStart=dotnet run
SyslogIdentifier=echoes

[Install]
WantedBy=multi-user.target

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line
```

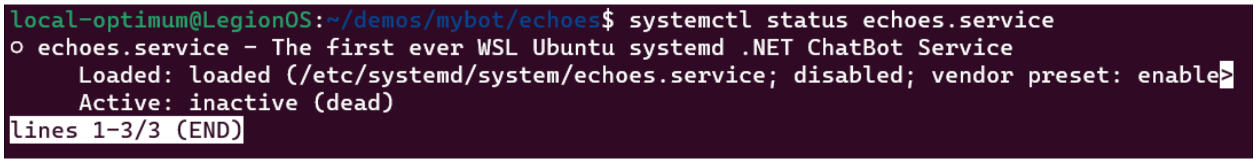
Save your file and reload your services with:

```
$ sudo systemctl daemon-reload
```

To reload the services. You can check if your service is ready by running:

```
$ systemctl status echoes.service
```

You should get the following output:

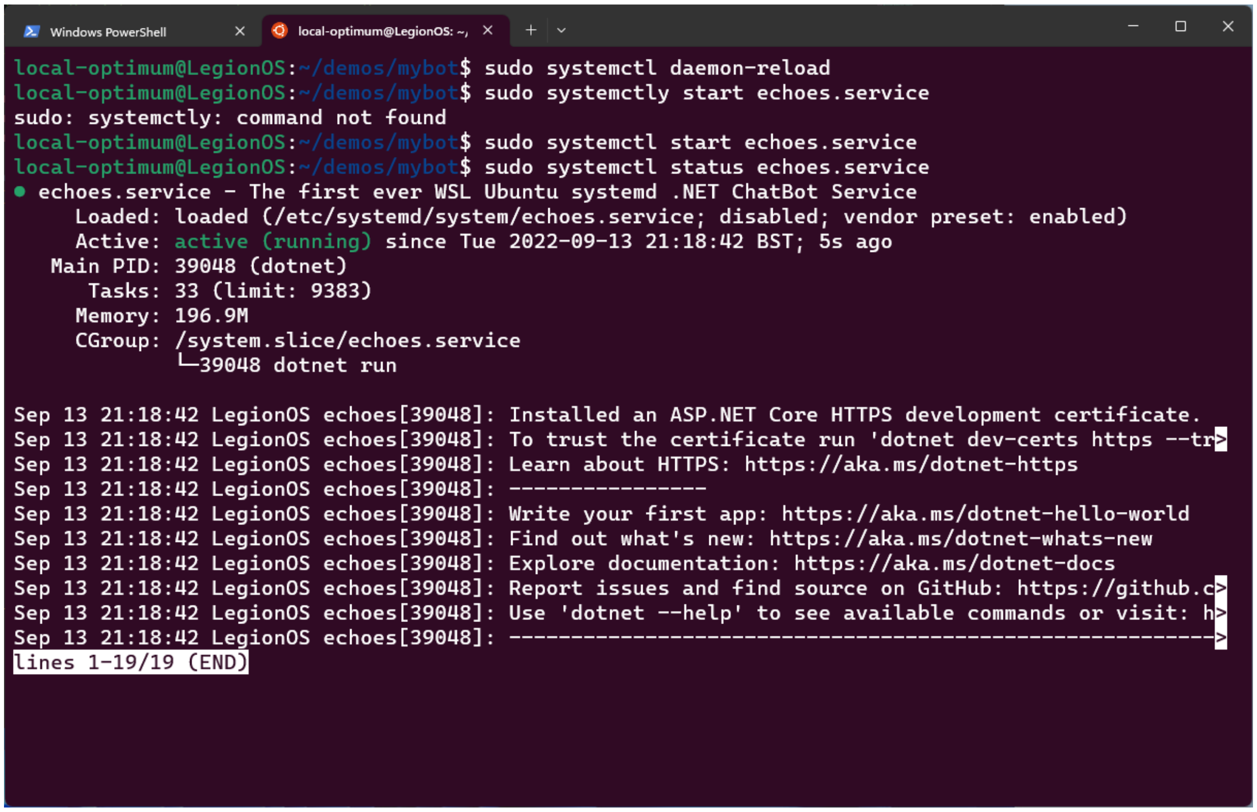


```
local-optimum@LegionOS:~/demos/mybot/echoes$ systemctl status echoes.service
○ echoes.service - The first ever WSL Ubuntu systemd .NET ChatBot Service
     Loaded: loaded (/etc/systemd/system/echoes.service; disabled; vendor preset: enable>
     Active: inactive (dead)
lines 1-3/3 (END)
```

Now start your service and then check its status again.

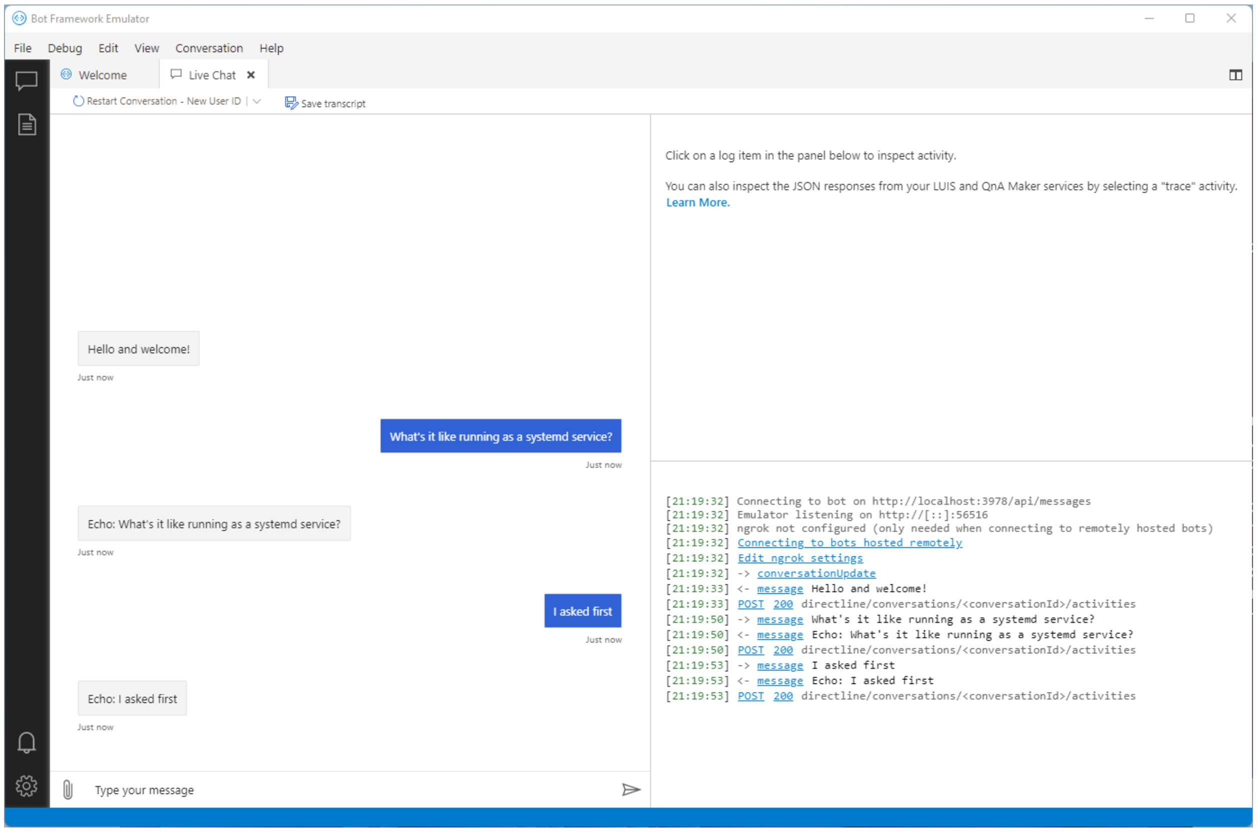
```
$ sudo systemctl start echoes.service $ sudo systemctl status echoes.service
```

If everything has been configured correctly you should get an output similar to the below.



```
local-optimum@LegionOS:~/demos/mybot$ sudo systemctl daemon-reload
local-optimum@LegionOS:~/demos/mybot$ sudo systemctly start echoes.service
sudo: systemctly: command not found
local-optimum@LegionOS:~/demos/mybot$ sudo systemctl start echoes.service
local-optimum@LegionOS:~/demos/mybot$ sudo systemctl status echoes.service
● echoes.service - The first ever WSL Ubuntu systemd .NET ChatBot Service
     Loaded: loaded (/etc/systemd/system/echoes.service; disabled; vendor preset: enabled)
     Active: active (running) since Tue 2022-09-13 21:18:42 BST; 5s ago
   Main PID: 39048 (dotnet)
      Tasks: 33 (limit: 9383)
     Memory: 196.9M
     CGroup: /system.slice/echoes.service
             └─39048 dotnet run

Sep 13 21:18:42 LegionOS echoes[39048]: Installed an ASP.NET Core HTTPS development certificate.
Sep 13 21:18:42 LegionOS echoes[39048]: To trust the certificate run 'dotnet dev-certs https --tr>
Sep 13 21:18:42 LegionOS echoes[39048]: Learn about HTTPS: https://aka.ms/dotnet-https
Sep 13 21:18:42 LegionOS echoes[39048]: ----------------
Sep 13 21:18:42 LegionOS echoes[39048]: Write your first app: https://aka.ms/dotnet-hello-world
Sep 13 21:18:42 LegionOS echoes[39048]: Find out what's new: https://aka.ms/dotnet-whats-new
Sep 13 21:18:42 LegionOS echoes[39048]: Explore documentation: https://aka.ms/dotnet-docs
Sep 13 21:18:42 LegionOS echoes[39048]: Report issues and find source on GitHub: https://github.c>
Sep 13 21:18:42 LegionOS echoes[39048]: Use 'dotnet --help' to see available commands or visit: h>
Sep 13 21:18:42 LegionOS echoes[39048]: --------------------------------------------------------->
lines 1-19/19 (END)
```

Return to your Windows host and reconnect to your Bot Emulator using the same information as before and confirm that your bot is running, but this time as a systemd service!


Bot Framework Emulator
File Debug Edit View Conversation Help
Welcome
Live Chat
Restart Conversation - New User ID
Save transcript
Type your message

Click on a log item in the panel below to inspect activity.

You can also inspect the JSON responses from your LUIS and QnA Maker services by selecting a "trace" activity. Learn More.

Hello and welcome!
Just now

What's it like running as a systemd service?
Just now

Echo: What's it like running as a systemd service?
Just now

I asked first
Just now

Echo: I asked first
Just now

```
[21:19:32] Connecting to bot on http://localhost:3978/api/messages
[21:19:32] Emulator listening on http://[::]:56516
[21:19:32] ngrok not configured (only needed when connecting to remotely hosted bots)
[21:19:32] Connecting to bots hosted remotely
[21:19:32] Edit ngrok settings
[21:19:32] -> conversationUpdate
[21:19:33] <- message Hello and welcome!
[21:19:33] POST 200 directline/conversations/<conversationId>/activities
[21:19:50] -> message What's it like running as a systemd service?
[21:19:50] <- message Echo: What's it like running as a systemd service?
[21:19:50] POST 200 directline/conversations/<conversationId>/activities
[21:19:53] -> message I asked first
[21:19:53] <- message Echo: I asked first
[21:19:53] POST 200 directline/conversations/<conversationId>/activities
```

You can stop your bot from running at any time with the command:

```
$ sudo systemctl stop echoes.service
```

Tutorial complete!

You now have a simple Echo Bot running as a systemd service on WSL that you can access from your host Windows machine.

If you would like to expand on this example try reviewing some of the more advanced Bot Framework samples on the Microsoft GitHub.

To read more about how Ubuntu supports .Net developers, making it easier than ever to build multi-platform services and applications, read our recent announcement.

Further Reading

- .Net on Ubuntu
- Bot Framework samples
- *Enabling GPU acceleration on Ubuntu on WSL2 with the NVIDIA CUDA Platform*
- *Windows and Ubuntu interoperability on WSL2*
- Microsoft WSL Documentation
- Ask Ubuntu

2.1.4 Enabling GPU acceleration with the NVIDIA CUDA Platform

Authored by Carlos Nihelton (carlos.santanadeoliveira@canonical.com)

While WSL's default setup allows you to develop cross-platform applications without leaving Windows, enabling GPU acceleration inside WSL provides users with direct access to the hardware. This provides support for GPU-accelerated AI/ML training and the ability to develop and test applications built on top of technologies, such as OpenVINO, OpenGL, and CUDA that target Ubuntu while staying on Windows.

What you will learn:

- How to install a Windows graphical device driver compatible with WSL2
- How to install the NVIDIA CUDA toolkit for WSL 2 on Ubuntu
- How to compile and run a sample CUDA application on Ubuntu on WSL2

What you will need:

- A Windows 10 version 21H2 or newer physical machine equipped with an NVIDIA graphics card and administrative permission to be able to install device drivers
- Ubuntu on WSL2 previously installed
- Familiarity with Linux command line utilities and interacting with Ubuntu on WSL2

  Note: If you need more introductory topics, such as how to install Ubuntu on WSL, refer to previous tutorials that can be found *here*.

Prerequisites

The following steps assume a specific hardware configuration. Although the concepts are essentially the same for other architectures, different hardware configurations will require the appropriate graphics drivers and CUDA toolkit.

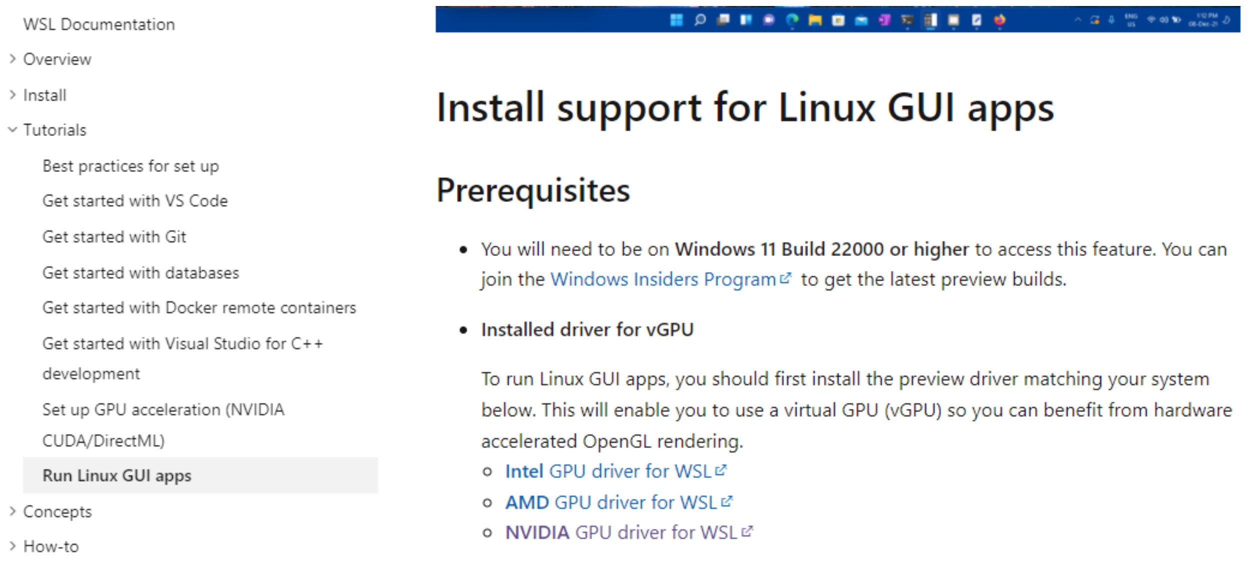
Make sure the following prerequisites are met before moving forward:

- A physical machine with Windows 10 version 21H2 or higher
- NVIDIA's graphic card
- Ubuntu 20.04 or higher installed on WSL 2
- Broadband internet connection able to download a few GB of data

Install the appropriate Windows vGPU driver for WSL

Specific drivers are needed to enable use of a virtual GPU, which is how Ubuntu applications are able to access your GPU hardware, so you'll need to follow this step even if your system drivers are up-to-date.

Please refer to the official WSL documentation for up-to-date links matching your specific GPU vendor. You can find these in `Install support for Linux GUI apps > Prerequisites`. For this example, we will download the `NVIDIA GPU Driver for WSL`.



WSL Documentation
Overview
Install
Tutorials
Best practices for set up
Get started with VS Code
Get started with Git
Get started with databases
Get started with Docker remote containers
Get started with Visual Studio for C++ development
Set up GPU acceleration (NVIDIA CUDA/DirectML)
Run Linux GUI apps
Concepts
How-to

Install support for Linux GUI apps

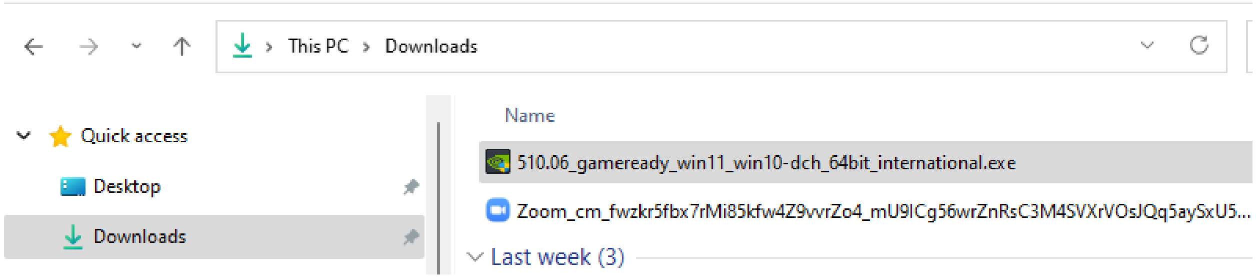
Prerequisites

- You will need to be on **Windows 11 Build 22000 or higher** to access this feature. You can join the Windows Insiders Program to get the latest preview builds.
- **Installed driver for vGPU**

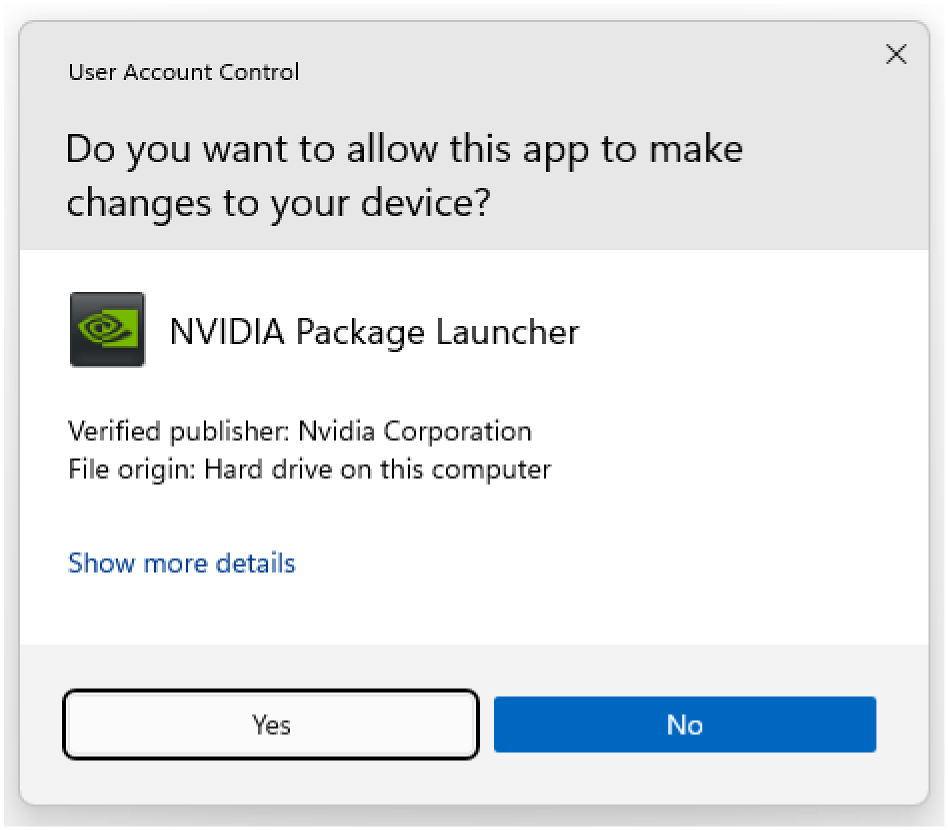
  To run Linux GUI apps, you should first install the preview driver matching your system below. This will enable you to use a virtual GPU (vGPU) so you can benefit from hardware accelerated OpenGL rendering.
  - **Intel** GPU driver for WSL
  - **AMD** GPU driver for WSL
  - **NVIDIA** GPU driver for WSL

Note: This is the only device driver you'll need to install. Do not install any display driver on Ubuntu.

Once downloaded, double-click on the executable file and click `Yes` to allow the program to make changes to your computer.



510.06_gameready_win11_win10-dch_64bit_international.exe
Zoom_cm_fwzkr5fbx7rMi85kfw4Z9vvrZo4_mU9ICg56wrZnRsC3M4SVXrVOsJQq5aySxU5...


User Account Control

Do you want to allow this app to make changes to your device?

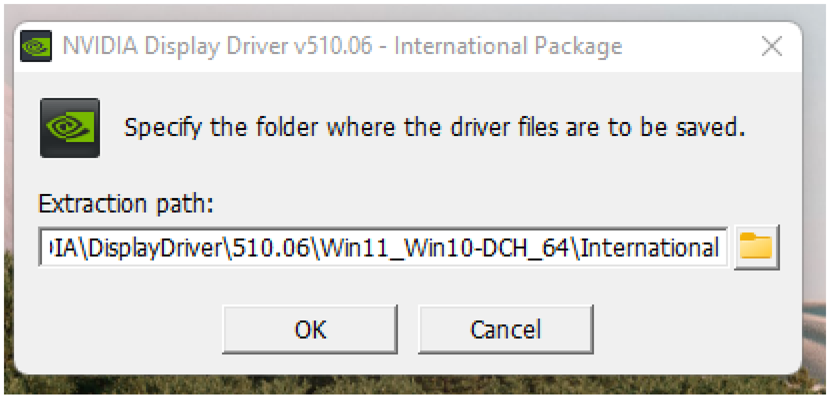
NVIDIA Package Launcher

Verified publisher: Nvidia Corporation
File origin: Hard drive on this computer

Show more details

Yes | No

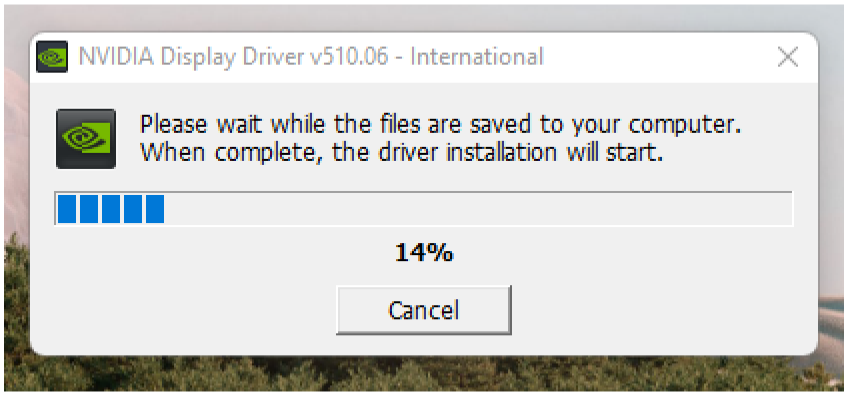
Confirm the default directory and allow the self-extraction process to proceed.


NVIDIA Display Driver v510.06 - International Package

Specify the folder where the driver files are to be saved.

Extraction path:
IA\DisplayDriver\510.06\Win11_Win10-DCH_64\International

OK | Cancel

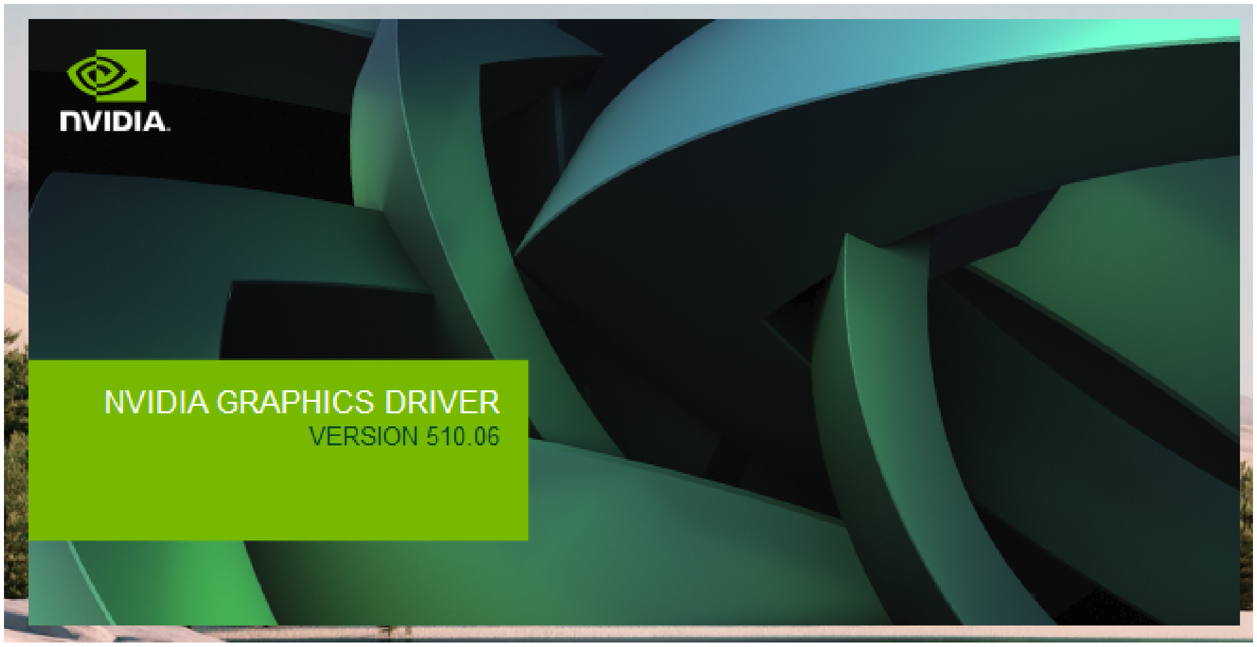

NVIDIA Display Driver v510.06 - International

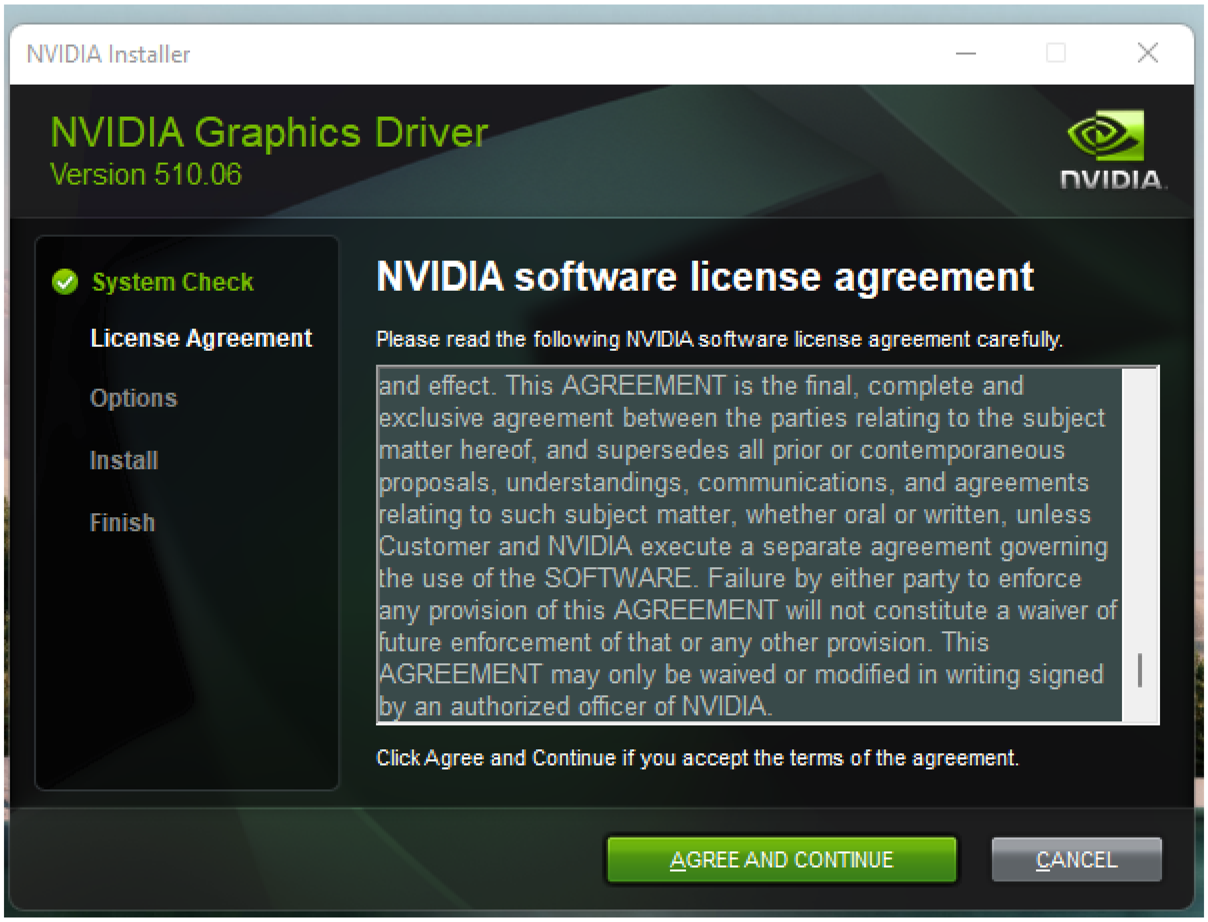
Please wait while the files are saved to your computer.
When complete, the driver installation will start.

14%

Cancel

A splash screen appears with the driver version number and quickly turns into the main installer window. Read and accept the license terms to continue.





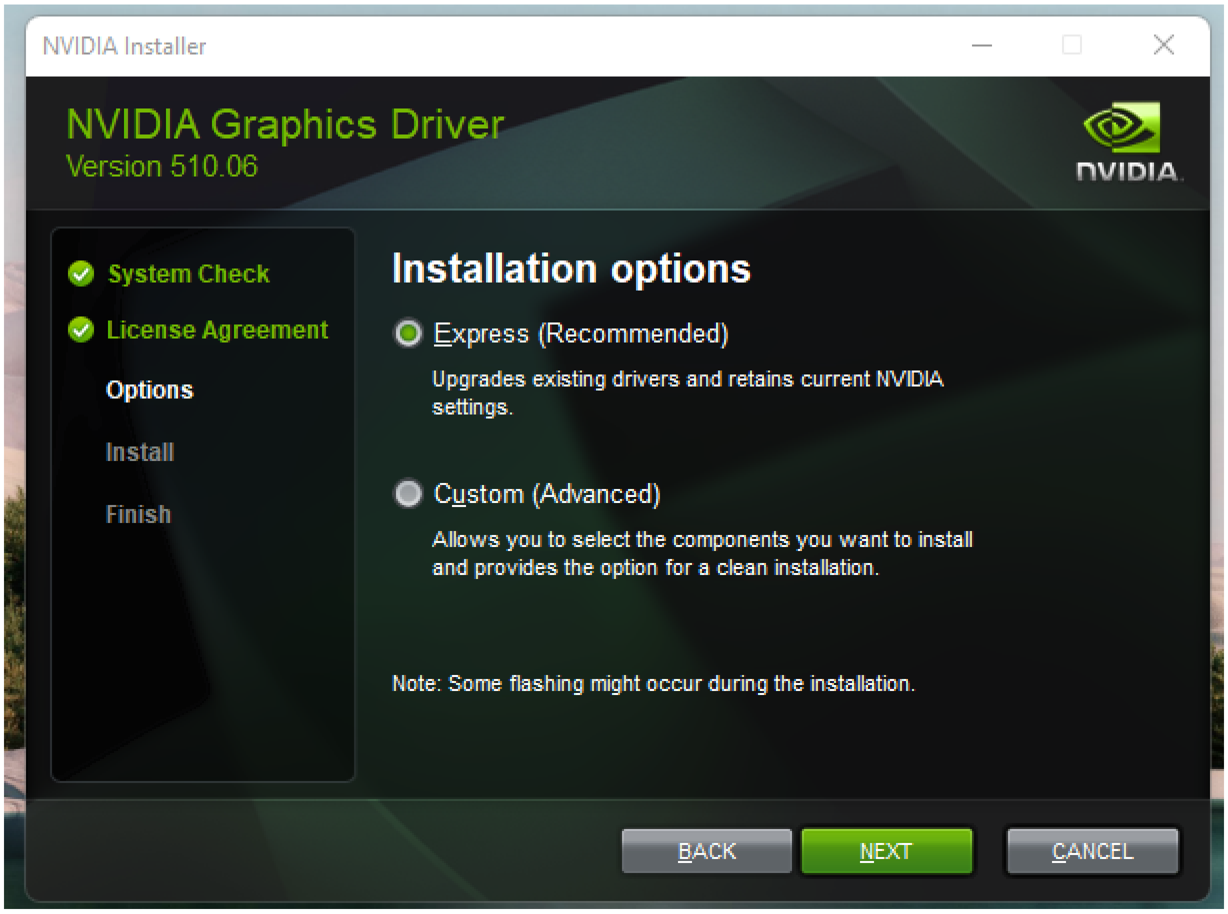
NVIDIA software license agreement

Please read the following NVIDIA software license agreement carefully.

and effect. This AGREEMENT is the final, complete and exclusive agreement between the parties relating to the subject matter hereof, and supersedes all prior or contemporaneous proposals, understandings, communications, and agreements relating to such subject matter, whether oral or written, unless Customer and NVIDIA execute a separate agreement governing the use of the SOFTWARE. Failure by either party to enforce any provision of this AGREEMENT will not constitute a waiver of future enforcement of that or any other provision. This AGREEMENT may only be waived or modified in writing signed by an authorized officer of NVIDIA.

Click Agree and Continue if you accept the terms of the agreement.

Confirm the wizard defaults by clicking `Next` and wait until the end of the installation. You might be prompted to restart your computer.



Installation options

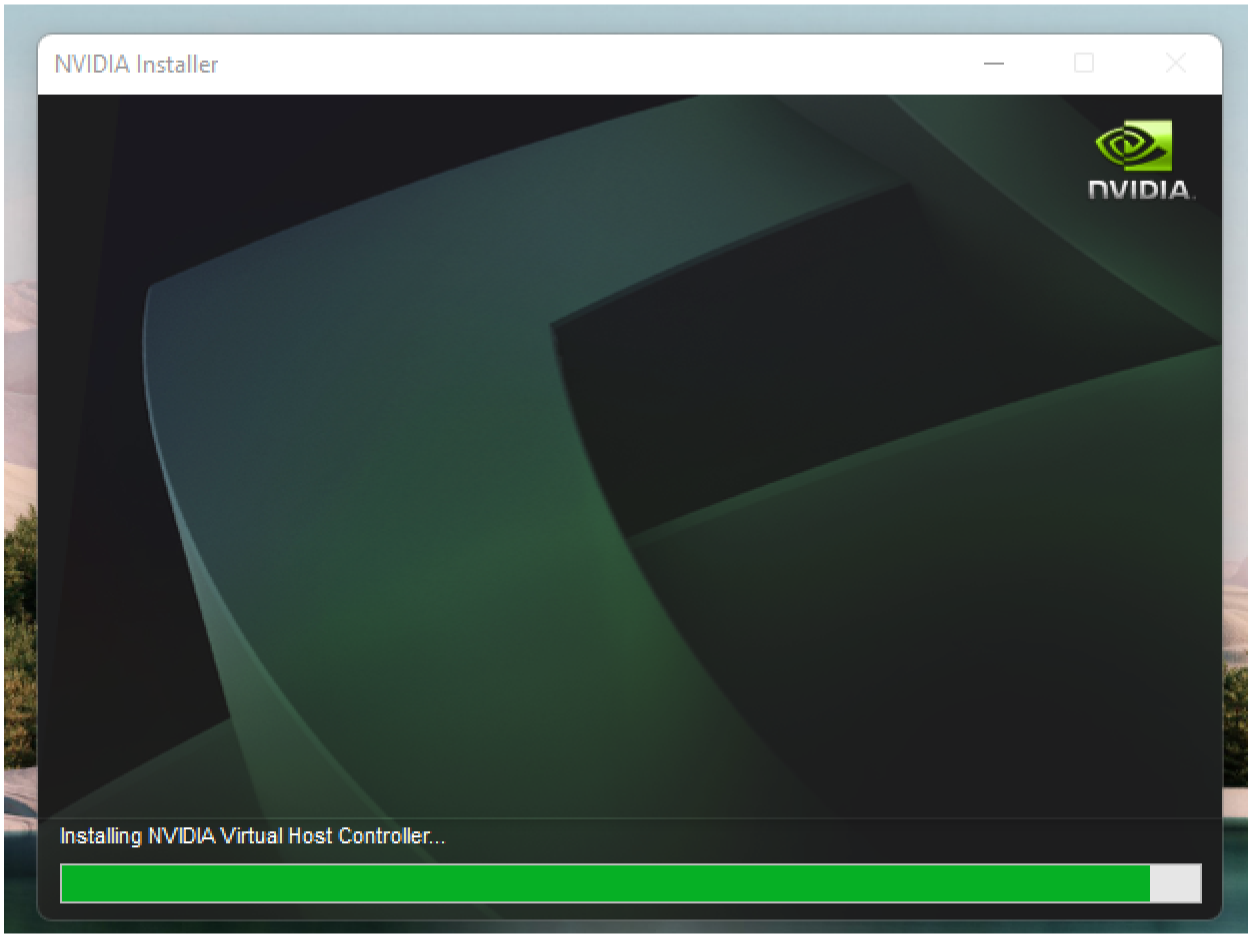
[✓ Express (Recommended)]

Upgrades existing drivers and retains current NVIDIA settings.

Custom (Advanced)

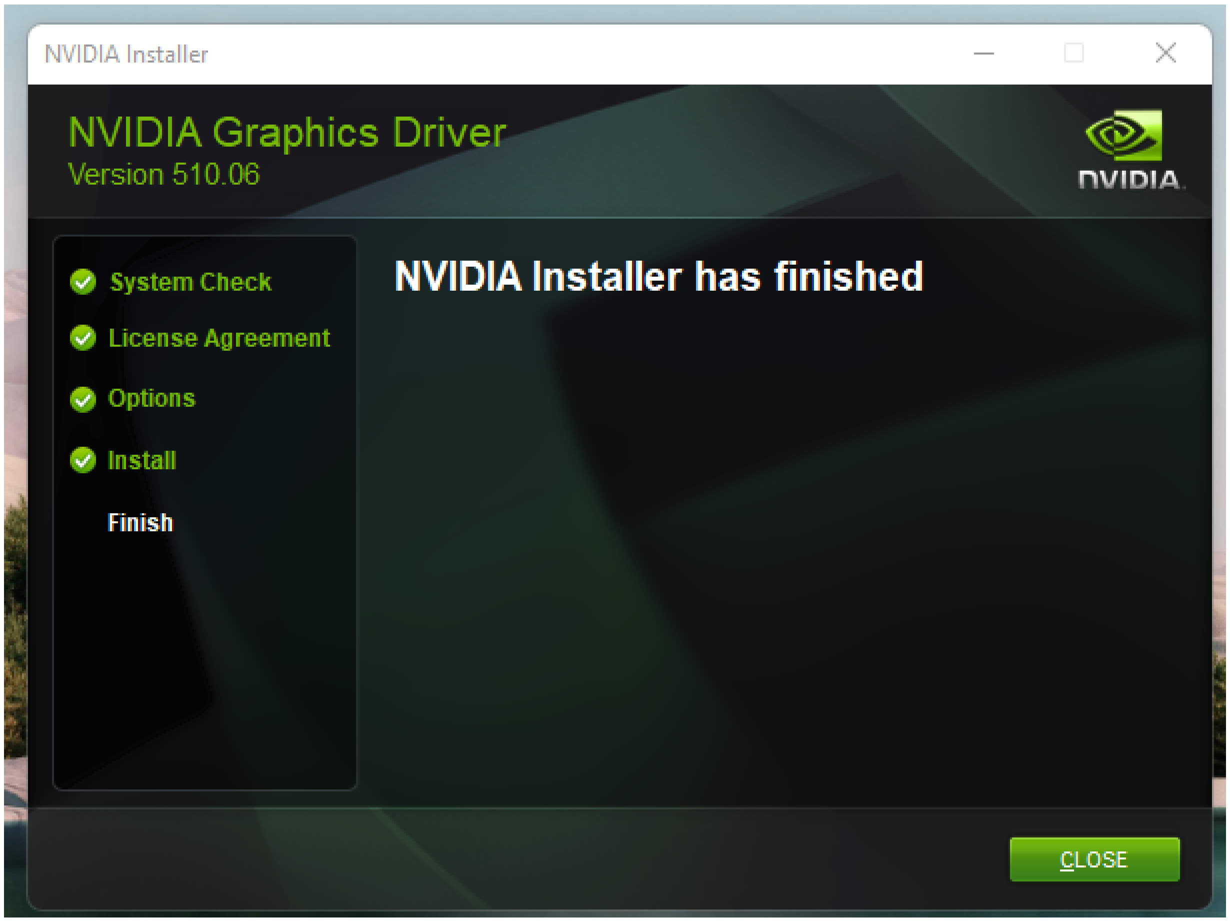
Allows you to select the components you want to install and provides the option for a clean installation.

Note: Some flashing might occur during the installation.



Installing NVIDIA Virtual Host Controller...

This step ends with a screen similar to the image below.



NVIDIA Installer has finished

Install NVIDIA CUDA on Ubuntu

> Normally, CUDA toolkit for Linux will have the device driver for the GPU packaged with it. On WSL 2, the CUDA driver used is part of the Windows driver installed on the system, and, therefore, care must be taken `not` to install this Linux driver as previously mentioned.

The following commands will install the WSL-specific CUDA toolkit version 11.6 on Ubuntu 22.04 AMD64 architecture. Be aware that older versions of CUDA (<=10) don't support WSL 2. Also notice that attempting to install the CUDA toolkit packages straight from the Ubuntu repository (`cuda`, `cuda-11-0`, or `cuda-drivers`) will attempt to install the Linux NVIDIA graphics driver, which is not what you want on WSL 2. So, first remove the old GPG key:

```
sudo apt-key del 7fa2af80
```

Then setup the appropriate package for Ubuntu WSL:

```
wget https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/
x86_64/cuda-wsl-ubuntu.pin
```

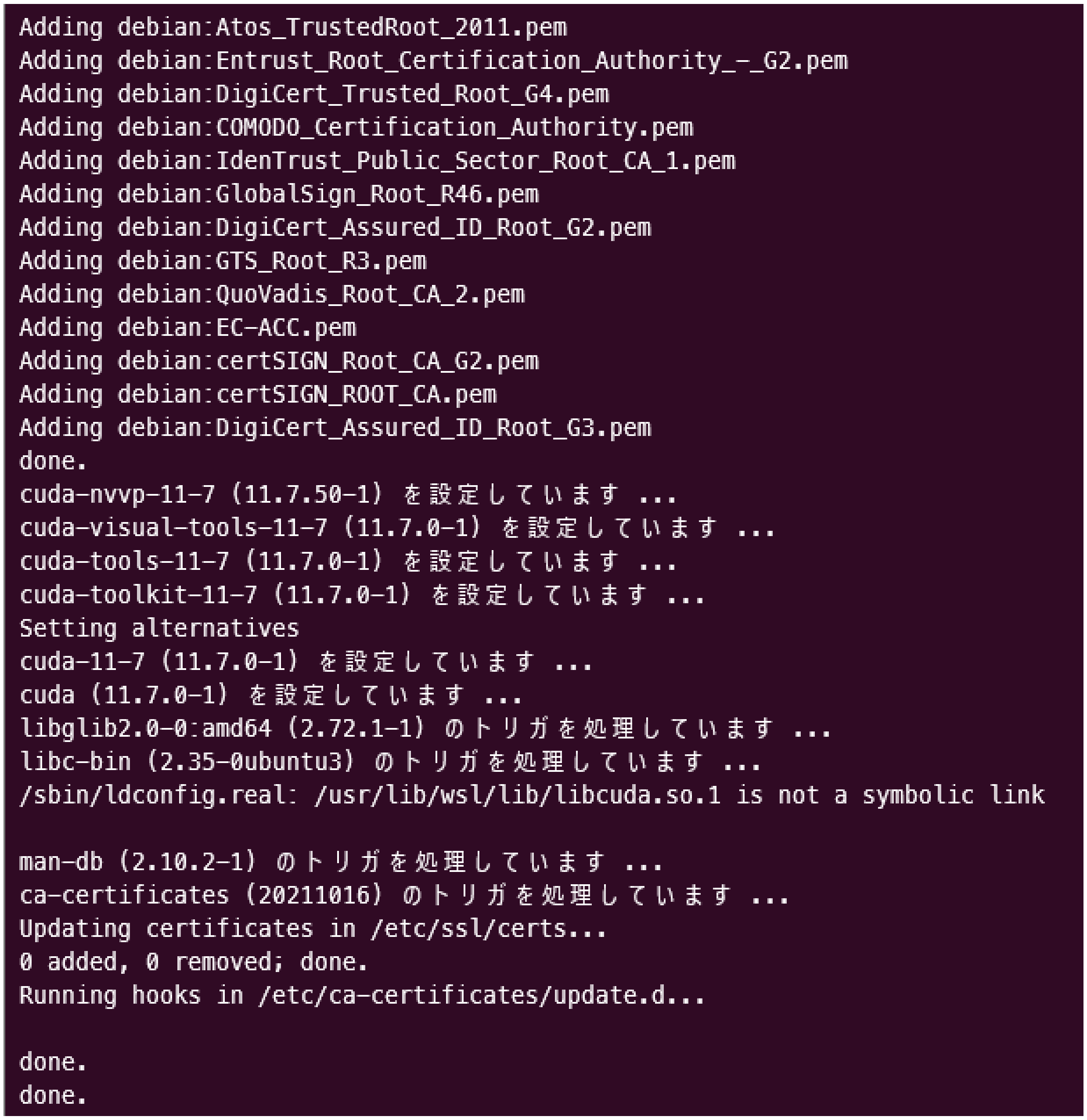
```
sudo mv cuda-wsl-ubuntu.pin /etc/apt/preferences.d/cuda-repository-pin-600
```

```
sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/
cuda/repos/wsl-ubuntu/x86_64/3bf863cc.pub
```

```
sudo add-apt-repository 'deb https://developer.download.nvidia.com/compute/
cuda/repos/wsl-ubuntu/x86_64/ /'
```

```
sudo apt-get update
sudo apt-get -y install cuda
```

Once complete, you should see a series of outputs that end in `done.`:



```
Adding debian:Atos_TrustedRoot_2011.pem
Adding debian:Entrust_Root_Certification_Authority_-_G2.pem
Adding debian:DigiCert_Trusted_Root_G4.pem
Adding debian:COMODO_Certification_Authority.pem
Adding debian:IdenTrust_Public_Sector_Root_CA_1.pem
Adding debian:GlobalSign_Root_R46.pem
Adding debian:DigiCert_Assured_ID_Root_G2.pem
Adding debian:GTS_Root_R3.pem
Adding debian:QuoVadis_Root_CA_2.pem
Adding debian:EC-ACC.pem
Adding debian:certSIGN_Root_CA_G2.pem
Adding debian:certSIGN_ROOT_CA.pem
Adding debian:DigiCert_Assured_ID_Root_G3.pem
done.
cuda-nvvp-11-7 (11.7.50-1) を設定しています ...
cuda-visual-tools-11-7 (11.7.0-1) を設定しています ...
cuda-tools-11-7 (11.7.0-1) を設定しています ...
cuda-toolkit-11-7 (11.7.0-1) を設定しています ...
Setting alternatives
cuda-11-7 (11.7.0-1) を設定しています ...
cuda (11.7.0-1) を設定しています ...
libglib2.0-0:amd64 (2.72.1-1) のトリガを処理しています ...
libc-bin (2.35-0ubuntu3) のトリガを処理しています ...
/sbin/ldconfig.real: /usr/lib/wsl/lib/libcuda.so.1 is not a symbolic link

man-db (2.10.2-1) のトリガを処理しています ...
ca-certificates (20211016) のトリガを処理しています ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...

done.
done.
```

Congratulations! You should have a working installation of CUDA by now. Let's test it in the next step.

Compile a sample application

NVIDIA provides an open source repository on GitHub with samples for CUDA Developers to explore the features available in the CUDA Toolkit. Building one of these is a great way to test your CUDA installation. Let's choose the simplest one just to validate that our installation works.

Let's say you have a `~/Dev/` directory where you usually put your working projects. Navigate inside the directory and `git clone` the cuda-samples repository:

```
cd ~/Dev
```

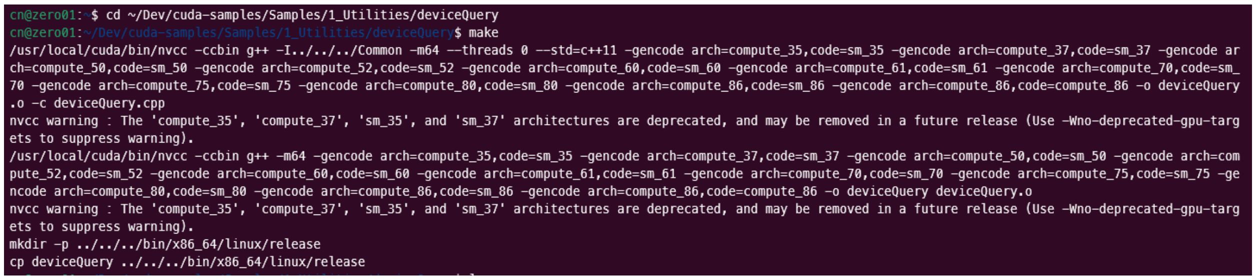
```
git clone https://github.com/nvidia/cuda-samples
```

To build the application, go to the cloned repository directory and type `make`:

```
cd ~/Dev/cuda-samples/Samples/1_Utilities/deviceQuery
```

```
make
```

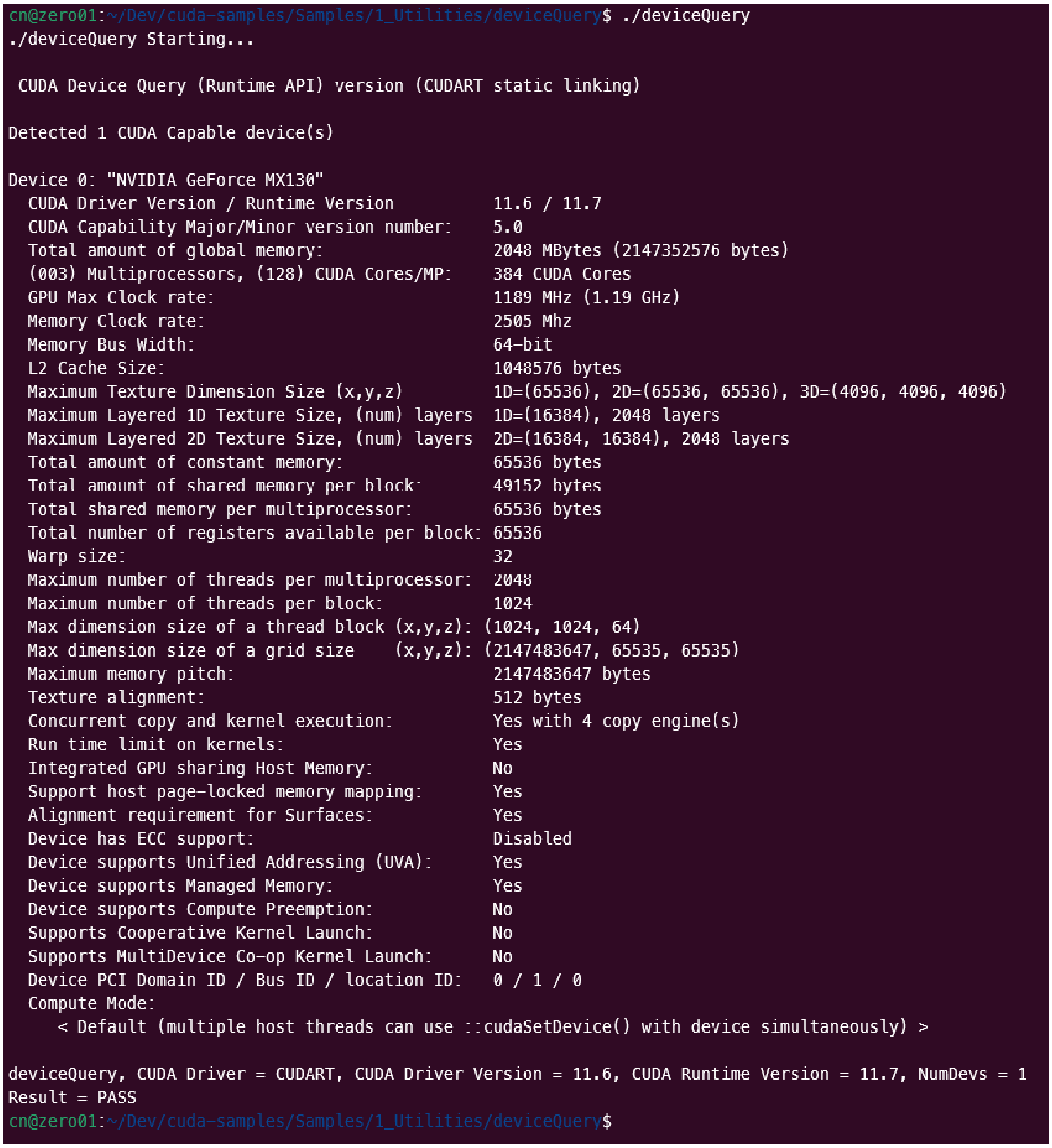
A successful build will look like the screenshot below. Once complete, run the application:



```
cn@zero01:~$ cd ~/Dev/cuda-samples/Samples/1_Utilities/deviceQuery
cn@zero01:~/Dev/cuda-samples/Samples/1_Utilities/deviceQuery$ make
/usr/local/cuda/bin/nvcc -ccbin g++ -I../../../Common -m64 --threads 0 --std=c++11 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode ar
ch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_60,code=sm_60 -gencode arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_
70 -gencode arch=compute_75,code=sm_75 -gencode arch=compute_80,code=sm_80 -gencode arch=compute_86,code=sm_86 -gencode arch=compute_86,code=compute_86 -o deviceQuery
.o -c deviceQuery.cpp
nvcc warning : The 'compute_35', 'compute_37', 'sm_35', and 'sm_37' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targ
ets to suppress warning).
/usr/local/cuda/bin/nvcc -ccbin g++ -m64 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_50 -gencode arch=com
pute_52,code=sm_52 -gencode arch=compute_60,code=sm_60 -gencode arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_70 -gencode arch=compute_75,code=sm_75 -ge
ncode arch=compute_80,code=sm_80 -gencode arch=compute_86,code=sm_86 -gencode arch=compute_86,code=compute_86 -o deviceQuery deviceQuery.o
nvcc warning : The 'compute_35', 'compute_37', 'sm_35', and 'sm_37' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targ
ets to suppress warning).
mkdir -p ../../../bin/x86_64/linux/release
cp deviceQuery ../../../bin/x86_64/linux/release
```

```
./deviceQuery
```

You should see a similar output to the following detailing the functionality of your CUDA setup (the exact results depend on your hardware setup):



```
cn@zero01:~/Dev/cuda-samples/Samples/1_Utilities/deviceQuery$ ./deviceQuery
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA GeForce MX130"
  CUDA Driver Version / Runtime Version          11.6 / 11.7
  CUDA Capability Major/Minor version number:    5.0
  Total amount of global memory:                 2048 MBytes (2147352576 bytes)
  (003) Multiprocessors, (128) CUDA Cores/MP:    384 CUDA Cores
  GPU Max Clock rate:                            1189 MHz (1.19 GHz)
  Memory Clock rate:                             2505 Mhz
  Memory Bus Width:                              64-bit
  L2 Cache Size:                                 1048576 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total shared memory per multiprocessor:        65536 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 4 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            No
  Supports Cooperative Kernel Launch:            No
  Supports MultiDevice Co-op Kernel Launch:      No
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.6, CUDA Runtime Version = 11.7, NumDevs = 1
Result = PASS
cn@zero01:~/Dev/cuda-samples/Samples/1_Utilities/deviceQuery$
```

Enjoy Ubuntu on WSL!

That's all folks! In this tutorial, we've shown you how to enable GPU acceleration on Ubuntu on WSL 2 and demonstrated its functionality with the NVIDIA CUDA toolkit, from installation through to compiling and running a sample application.

We hope you enjoy using Ubuntu inside WSL for your Data Science projects. Don't forget to check out our blog for the latest news on all things Ubuntu.

Further Reading

- Setting up WSL for Data Science
- Ubuntu WSL for Data Scientists Whitepaper
- *NVIDIA's CUDA Post Installation Actions*
- *Install Ubuntu on WSL2*
- Microsoft WSL Documentation
- Ask Ubuntu

2.1.5 Use WSL for data science and engineering

Authored by Oliver Smith (oliver.smith@canonical.com) and edited by Edu Gómez Escandell (edu.gomez.escandell@canonical.com)

WSL is an ideal platform to run your Linux workflows while using your Windows machines. Here we show an example of how to set up GNU octave and run a toy program.

First, you'll need to set up Ubuntu on WSL, see *here*.

GNU octave

> GNU Octave is software featuring a high-level programming language, primarily intended for numerical computations. Octave helps in solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB. [GNU / Octave]

We will use it to calculate and draw a beautiful Julia fractal. The goal here is to use Octave to demonstrate how WSLg works, not to go through the theory of fractals.

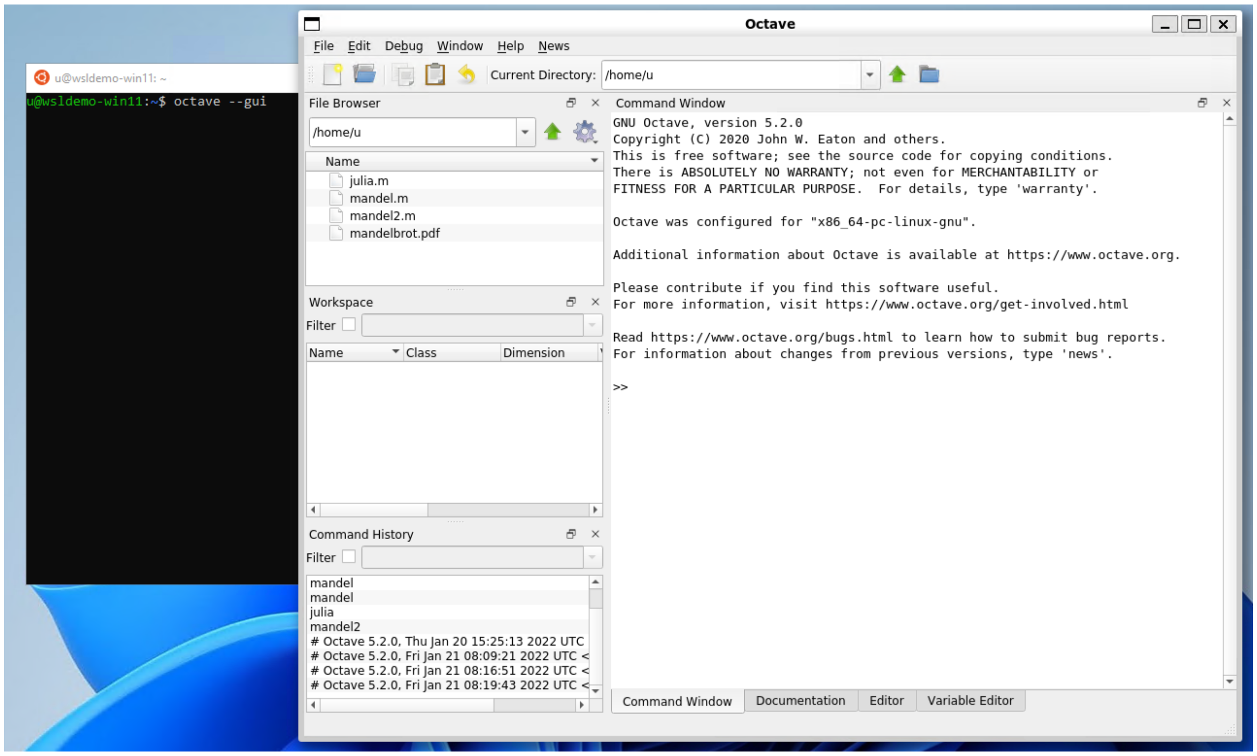
First thing is to install the software like we did for x11-apps, from the terminal prompt run:

```
sudo apt update
sudo apt install -y octave
```

Then start the application:

```
octave --gui &
```

Do not forget the ampersand & at the end of the line, so the application is started in the background and we can continue using the same terminal window.



```
u@wsldemo-win11:~$ octave --gui
```

```
julia.m
mandel.m
mandel2.m
mandelbrot.pdf
```

```
GNU Octave, version 5.2.0
Copyright (C) 2020 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-pc-linux-gnu".

Additional information about Octave is available at https://www.octave.org.

Please contribute if you find this software useful.
For more information, visit https://www.octave.org/get-involved.html

Read https://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

>>
```

```
mandel
mandel
julia
mandel2
# Octave 5.2.0, Thu Jan 20 15:25:13 2022 UTC
# Octave 5.2.0, Fri Jan 21 08:09:21 2022 UTC <
# Octave 5.2.0, Fri Jan 21 08:16:51 2022 UTC <
# Octave 5.2.0, Fri Jan 21 08:19:43 2022 UTC <
```

In Octave, click on the `New script` icon to open a new editor window and copy/paste the following code:

```
#{
Inspired by the work of Bruno Girin ([Geek Thoughts: Fractals with Octave: Classic Mandelbrot and Julia](http://brunogirin.blogspot.com/2008/12/fractals-with-octave-classic-mandelbrot.html))
Calculate a Julia set
zmin: Minimum value of c
zmax: Maximum value of c
hpx: Number of horizontal pixels
niter: Number of iterations
c: A complex number
#}
function M = julia(zmin, zmax, hpx, niter, c)
    %% Number of vertical pixels
    vpx=round(hpx*abs(imag(zmax-zmin)/real(zmax-zmin)));
    %% Prepare the complex plane
    [zRe,zIm]=meshgrid(linspace(real(zmin),real(zmax),hpx),
    linspace(imag(zmin),imag(zmax),vpx));
    z=zRe+i*zIm;
    M=zeros(vpx,hpx);
    %% Generate Julia
    for s=1:niter
        mask=abs(z)<2;
        M(mask)=M(mask)+1;
        z(mask)=z(mask).^2+c;
    end
    M(mask)=0;
end
```

This code is the function that will calculate the Julia set. Save it to a file named `julia.m`. Since it is a function definition, the name of the file must match the name of the function.

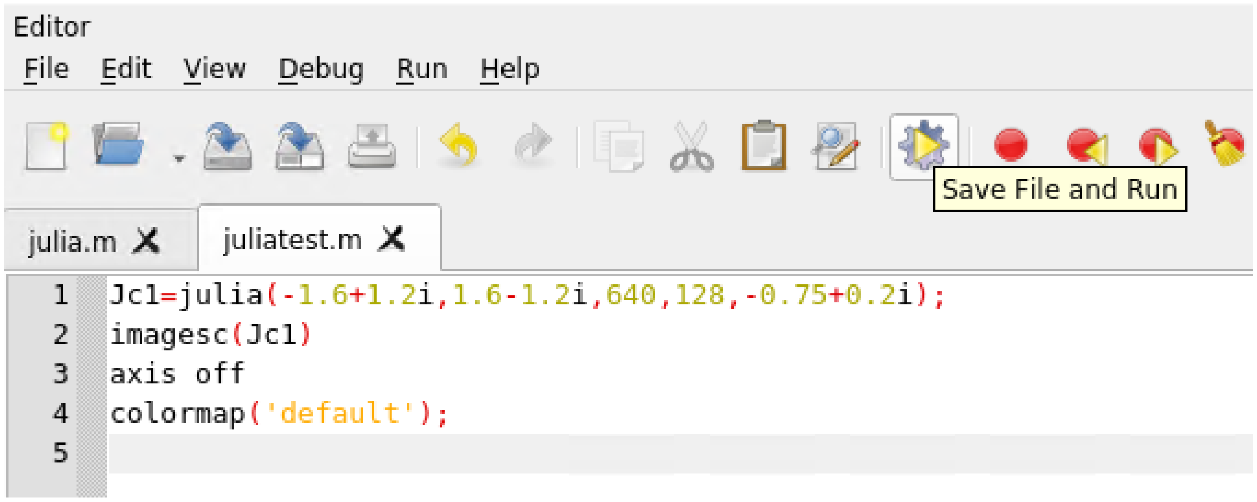
Open a second editor window with the New Script button and copy and paste the following code:

```
Jc1=julia(-1.6+1.2i, 1.6-1.2i, 640, 128, -0.75+0.2i);
imagesc(Jc1)
axis off
colormap('default');
```

This code calls the function defined in `julia.m`. You can later change the parameters if you want to explore the Julia fractal.

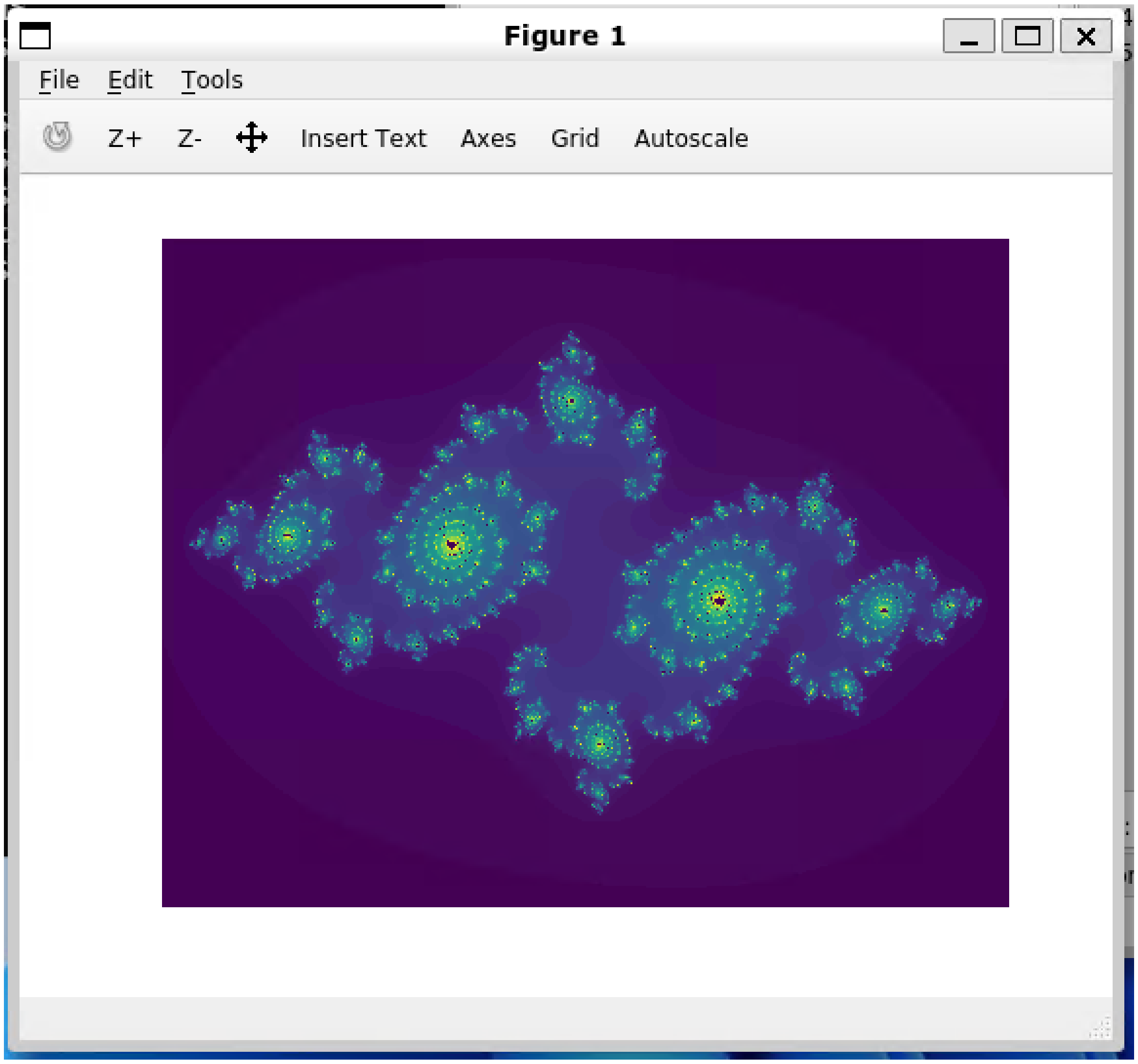
Save it to a file named `juliatest.m`.

And finally, press the button `Save File and Run`.



```
Jc1=julia(-1.6+1.2i,1.6-1.2i,640,128,-0.75+0.2i);
imagesc(Jc1)
axis off
colormap('default');
```

After a few seconds, depending on your hardware and the parameters, a Julia fractal is displayed.



Like Octave, this window is displayed using WSLg completely transparently to the user.

Enjoy!

Further Reading

- An introduction to numerical computation applications using Ubuntu WSL
- Setting up WSL for Data Science
- Whitepaper: Ubuntu WSL for Data Scientists
- Microsoft WSL Documentation
- Ask Ubuntu

2.1.6 Automatic setup with cloud-init

Authored by Carlos Nihelton (carlos.santanadeoliveira@canonical.com)

Cloud-init is an industry-standard multi-distribution method for cross-platform cloud instance initialisation. Ubuntu WSL users can now leverage it to perform an automatic setup to get a working instance with minimal touch.

See more: cloud-init official documentation.

Note: **Coming soon*:

That feature is currently in development and will be available soon on the UbuntuPreview app first, then gradually released for the latest LTS applications.

What you will learn:

- How to write cloud-config user data to a specific WSL instance.
- How to automatically set up a WSL instance with cloud-init.
- How to verify that cloud-init succeeded with the configuration supplied.

What you will need:

- Windows 11 with WSL 2 already enabled
- The latest UbuntuPreview application from Microsoft Store.

Write the cloud-config file

Locate your Windows user home directory. It typically is `C:\Users\<YOUR_USER_NAME>`.

You can be sure about that path by running `echo $env:USERPROFILE` in PowerShell.

Inside your Windows user home directory, create a new folder named `.cloud-init` (notice the `.` à la Linux configuration directories), and inside the new directory, create an empty file named `Ubuntu-Preview.user-data`. That file name must match the name of the distro instance that will be created in the next step.

Open that file with your text editor of choice (`notepad.exe` is just fine) and paste in the following contents:

```
#cloud-config
locale: pt_BR
users:
- name: jdoe
  gecos: John Doe
  groups: [adm,dialout,cdrom,floppy,sudo,audio,dip,video,plugdev,netdev]
  sudo: ALL=(ALL) NOPASSWD:ALL
  shell: /bin/bash

write_files:
- path: /etc/wsl.conf
  append: true
  content: |
    [user]
```

(continues on next page)

(continued from previous page)

```
    default=jdoe

packages: [ginac-tools, octave]

runcmd:
   - git clone https://github.com/Microsoft/vcpkg.git /opt/vcpkg
   - /opt/vcpkg/bootstrap-vcpkg.sh
```

Save it and close it.

> That example will create a user named `jdoe` and set it as default via `/etc/wsl.conf`, install the packages `ginac-tools` and `octave` and install `vcpkg` from the git repository, since there is no deb or snap of that application (hence the reason for being included in this tutorial - it requires an unusual setup).
>
> See more: WSL data source reference.

Register a new Ubuntu-Preview instance

In PowerShell, run:

```
ubuntupreview.exe install --root
```

We skip the user creation since we expect cloud-init to do it.

> If you want to be sure that there is now an Ubuntu-Preview instance, run `wsl -l -v`. Notice that the application is named `UbuntuPreview` but the WSL instance created is named `Ubuntu-Preview`. See more about that naming convention in *our reference documentation*.

Check that cloud-init is running

In PowerShell again run:

```
ubuntupreview run cloud-init status --wait
```

That will wait until cloud-init completes configuring the new instance we just created. When done, you should see an output similar to the following:

```
...............................................................................
...............................................................................
...............................................................................
...................
status: done
```

Verify that it worked

Restart the distro just to make sure the changes in `/etc/wsl.conf` made by cloud-init will take effect, as listed below:

```
> wsl -t Ubuntu-Preview
The operation completed successfully.
> ubuntupreview
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu Noble Numbat (development branch) (GNU/Linux 5.15.137.3-microsoft-
↪standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro


This message is shown once a day. To disable it please create the
/home/cn/.hushlogin file.
jdoe@mib:~$
```

Once logged in the new distro instance's shell, verify that:

1. The default user matches what was configured in the user data file (in our case `jdoe`).

```
jdoe@mib:~$ whoami
jdoe
```

2. The supplied cloud-config user data was approved by cloud-init validation.

```
jdoe@mib:~$ sudo cloud-init schema --system
Valid schema user-data
```

3. The locale is set

```
jdoe@mib:~$ locale
LANG=pt_BR
LANGUAGE=
LC_CTYPE="pt_BR"
LC_NUMERIC="pt_BR"
LC_TIME="pt_BR"
LC_COLLATE="pt_BR"
LC_MONETARY="pt_BR"
LC_MESSAGES="pt_BR"
LC_PAPER="pt_BR"
LC_NAME="pt_BR"
LC_ADDRESS="pt_BR"
LC_TELEPHONE="pt_BR"
LC_MEASUREMENT="pt_BR"
LC_IDENTIFICATION="pt_BR"
LC_ALL=
```

4. The packages were installed and the commands they provide are available

```
jdoe@mib:~$ apt list --installed | egrep 'ginac|octave'

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

ginac-tools/noble,now 1.8.7-1 amd64 [installed]
libginac11/noble,now 1.8.7-1 amd64 [installed,automatic]
octave-common/noble,now 8.4.0-1 all [installed,automatic]
octave-doc/noble,now 8.4.0-1 all [installed,automatic]
octave/noble,now 8.4.0-1 amd64 [installed]
```

5. Verify that the commands requested were also run. In this case we set up `vcpkg` from git, as recommended by its documentation (there is no deb or snap available for that program).

```
jdoe@mib:~$ /opt/vcpkg/vcpkg version
vcpkg package management program version 2024-01-11-
↪710a3116bbd615864eef5f9010af178034cb9b44

See LICENSE.txt for license information.
```

Enjoy!

That's all folks! In this tutorial, we've shown you how to use cloud-init to automatically set up Ubuntu on WSL 2 with minimal touch.

This workflow will guarantee a solid foundation for your next Ubuntu WSL project.

We hope you enjoy using Ubuntu inside WSL!

2.2 How-to guides

2.2.1 Install Ubuntu on WSL2

Authored by Oliver Smith (oliver.smith@canonical.com) and edited by Edu Gómez Escandell (edu.gomez.escandell@canonical.com)

What you will learn:

- How to enable and install WSL on Windows 10 and Windows 11
- How to install and run a simple graphical application that uses WSLg
- How to install and run a much more advanced application that uses WSLg

What you will need:

- A Windows 10 or Windows 11 physical or virtual machine with all the updates installed

Install WSL

WSL can be installed from the command line. Open a PowerShell prompt as an Administrator (we recommend using Windows Terminal) and run:

```
wsl --install
```

This command will enable the features necessary to run WSL and also install the default Ubuntu distribution of Linux available in the Microsoft Store. It is recommended to reboot your machine after this initial installation to complete the setup. You can also install WSL from the Microsoft Store.

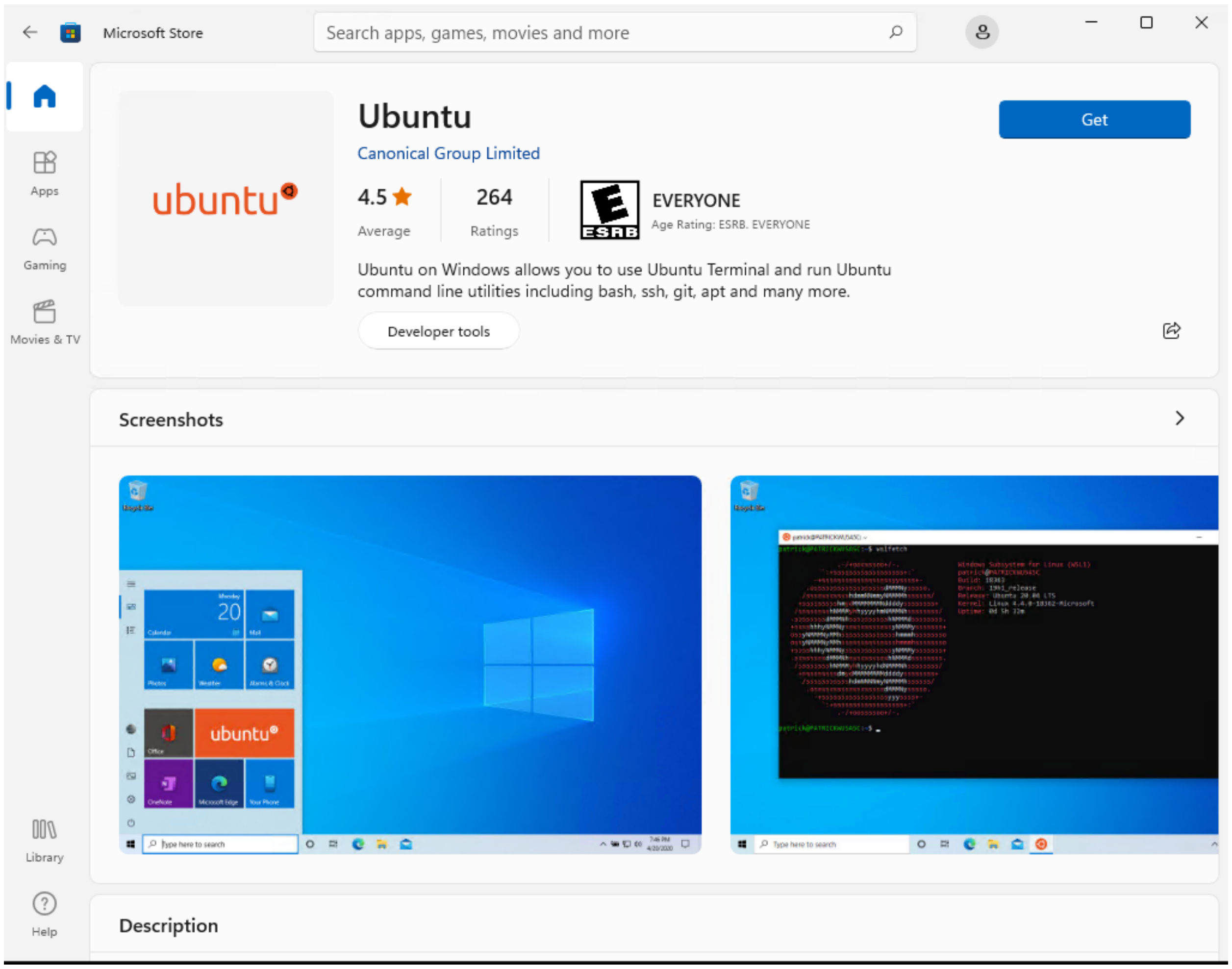
Install Ubuntu WSL

WSL supports a variety of Linux distributions including the latest Ubuntu release. Check out *the documentation* to see which one you prefer. For the rest of this tutorial we'll use `Ubuntu` as the example.

There are multiple ways of installing distros on WSL, here we show three: via the Microsoft store, via Winget, and the WSL CLI. The result is equivalent.

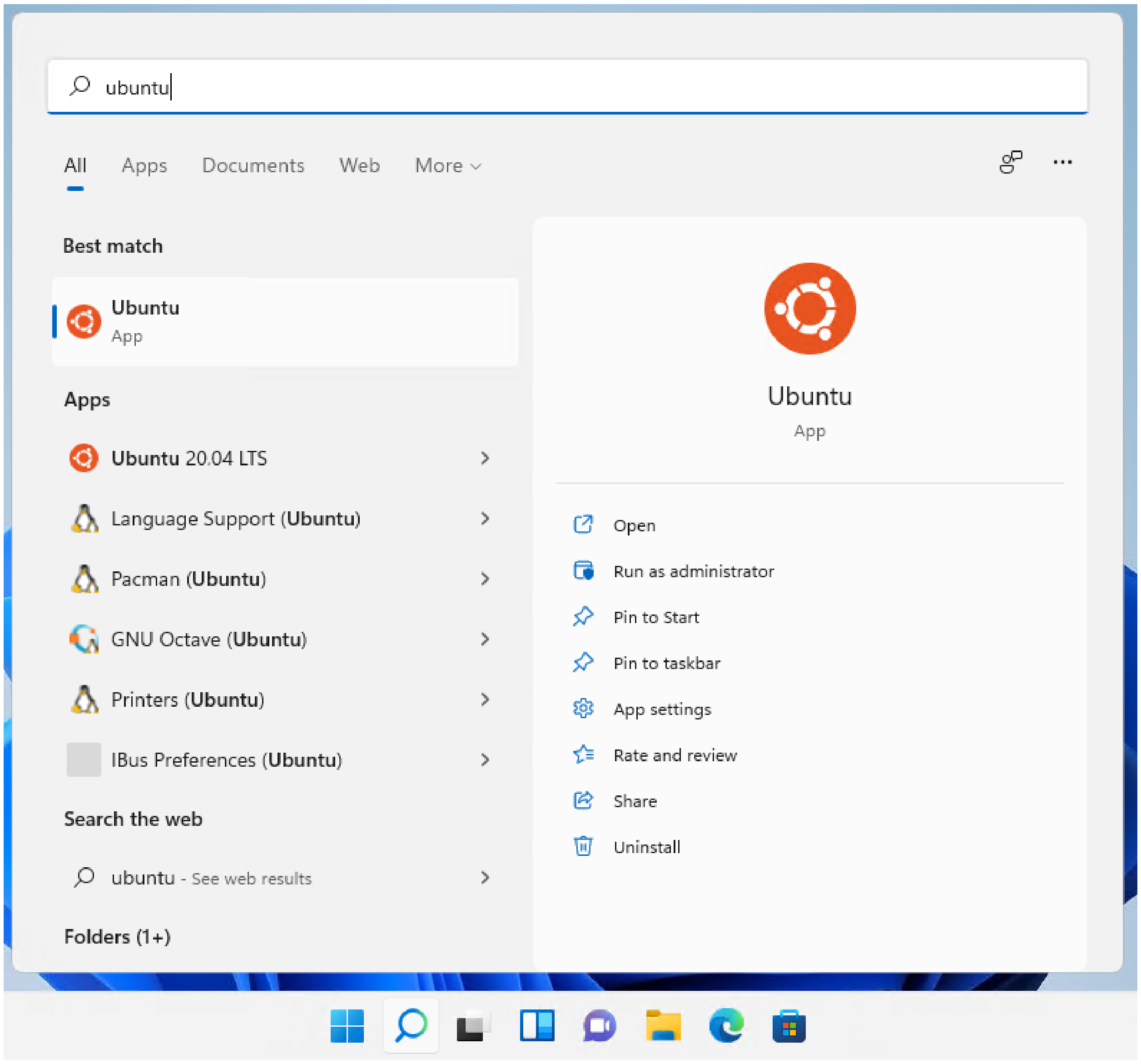
Method 1: Microsoft store

Find the distribution you prefer on the Microsoft Store and then select `Get`.



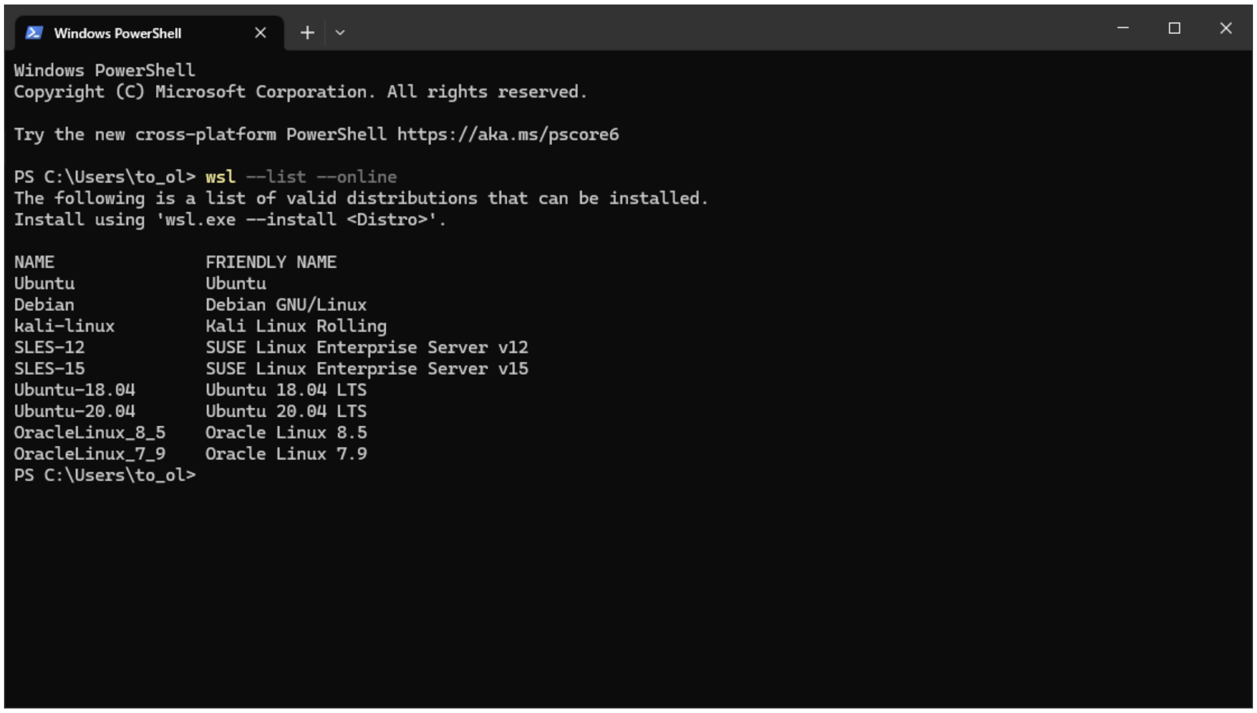
Ubuntu on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

Ubuntu will then be installed on your machine. Once installed, you can either launch the application directly from the store or search for Ubuntu in your Windows search bar.



Method 2: WSL Command line interface

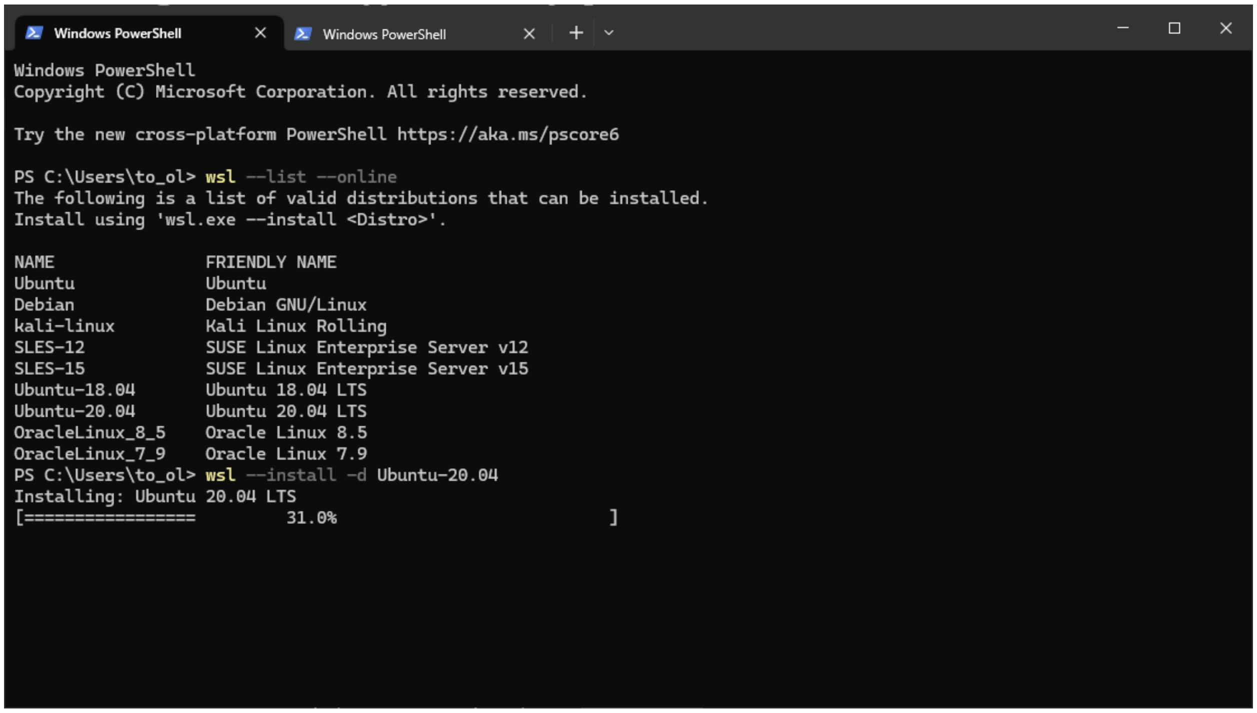
It is possible to install the same Ubuntu applications available on the Windows Store directly from the command line. In a PowerShell terminal, you can run `wsl --list --online` to see all available distros.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\to_ol> wsl --list --online
The following is a list of valid distributions that can be installed.
Install using 'wsl.exe --install <Distro>'.

NAME                   FRIENDLY NAME
Ubuntu                 Ubuntu
Debian                 Debian GNU/Linux
kali-linux             Kali Linux Rolling
SLES-12                SUSE Linux Enterprise Server v12
SLES-15                SUSE Linux Enterprise Server v15
Ubuntu-18.04           Ubuntu 18.04 LTS
Ubuntu-20.04           Ubuntu 20.04 LTS
OracleLinux_8_5        Oracle Linux 8.5
OracleLinux_7_9        Oracle Linux 7.9
PS C:\Users\to_ol>
```

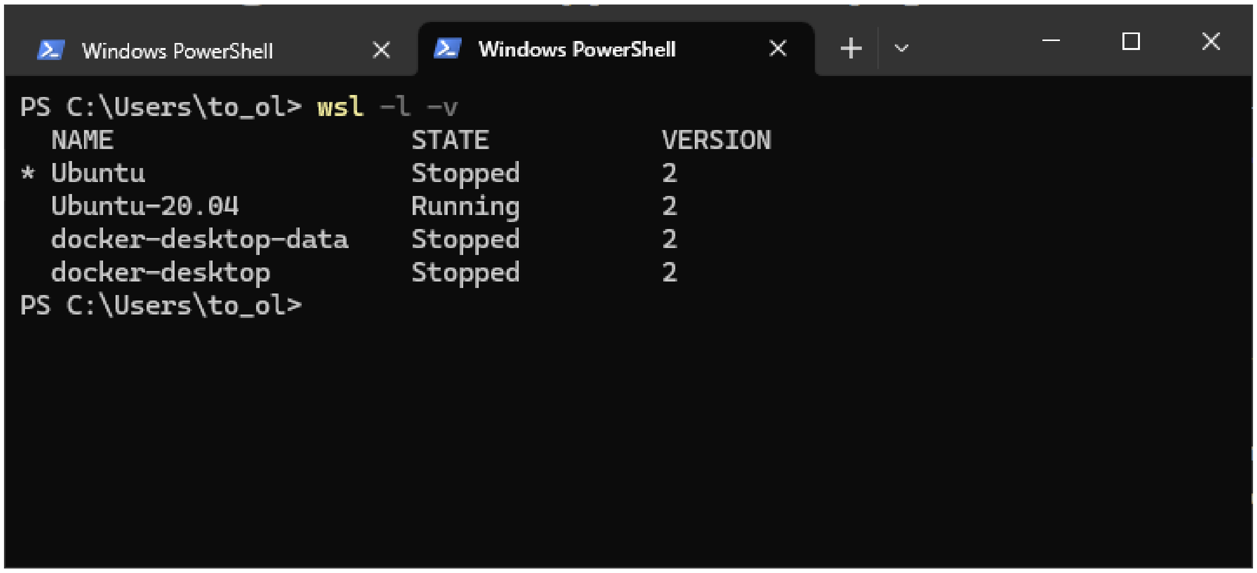
You can install a distro using the NAME by running:

```
wsl --install -d Ubuntu-20.04
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\to_ol> wsl --list --online
The following is a list of valid distributions that can be installed.
Install using 'wsl.exe --install <Distro>'.

NAME                   FRIENDLY NAME
Ubuntu                 Ubuntu
Debian                 Debian GNU/Linux
kali-linux             Kali Linux Rolling
SLES-12                SUSE Linux Enterprise Server v12
SLES-15                SUSE Linux Enterprise Server v15
Ubuntu-18.04           Ubuntu 18.04 LTS
Ubuntu-20.04           Ubuntu 20.04 LTS
OracleLinux_8_5        Oracle Linux 8.5
OracleLinux_7_9        Oracle Linux 7.9
PS C:\Users\to_ol> wsl --install -d Ubuntu-20.04
Installing: Ubuntu 20.04 LTS
[==================          31.0%                            ]
```

Use `wsl -l -v` to see all your currently installed distros and which version of WSL they are using:



```
PS C:\Users\to_ol> wsl -l -v
  NAME                   STATE           VERSION
* Ubuntu                 Stopped         2
  Ubuntu-20.04           Running         2
  docker-desktop-data    Stopped         2
  docker-desktop         Stopped         2
PS C:\Users\to_ol>
```

Method 3: Winget

Open a PowerShell terminal and type:

```
winget show --name Ubuntu --source msstore
```

You'll see a list of distros available and their ID. Choose the one you prefer and install it. For instance, Ubuntu:

```
winget install --Id "9PDXGNCFSCZV" --source msstore
```

You'll be prompted to accept the source and package agreements before installing. You need to accept them in order to proceed.

Check out *the documentation* to see which executable matches your application and run it.

```
ubuntu.exe
```

Configure Ubuntu

Congratulations, you now have an Ubuntu terminal running on your Windows machine!

Once it has finished its initial setup, you will be prompted to create a username and password. They don't need to match your Windows user credentials.

Finally, it's always good practice to install the latest updates with the following commands, entering your password when prompted.

```
sudo apt update
sudo apt full-upgrade -y
```

Enjoy Ubuntu on WSL!

That's it! In this tutorial, we've shown you how to install WSL and Ubuntu on Windows 11, set up your profile, install a few packages, and run a graphical application.

We hope you enjoy working with Ubuntu inside WSL. Don't forget to check out our blog for the latest news on all things Ubuntu.

Further Reading

- Setting up WSL for Data Science
- Whitepaper: Ubuntu WSL for Data Scientists
- Microsoft WSL Documentation
- Ask Ubuntu

2.2.2 Using the Auto-installation feature

Warning: This feature has been deprecated and is no longer supported. Stay tuned for upcoming support for Cloud-init.

Note: See more: Cloud init | Home

2.2.3 How to run your WSL GitHub workflow on Azure

Read more: How we improved testing Ubuntu on WSL – and how you can too!

Most of the time, what works on Ubuntu desktop works on WSL as well. However, there are some exceptions. Furthermore, you may want to test software that lives both on Windows and inside WSL. In these cases, you may want to run your automated testing on a Windows machine with WSL rather than a regular ubuntu machine.

There exist Windows GitHub runners, but they do not support the latest version of WSL. The reason is that WSL is now a Microsoft Store application, which requires a logged-in user. GitHub runners, however, run as a service. This means that they are not on a user session, hence they cannot run WSL (or any other store application).

Read more: What's new in the Store version of WSL?

Summary

We propose you run your automated tests on a Windows virtual machine hosted on Azure. This machine will run the GitHub actions runner not as a service, but as a command-line application.

Step-by-step

This guide will show you how to set up an Azure VM to run your WSL workflows.

1. Create a Windows 11 VM on Azure: follow Azure's instructions, no special customisation is necessary.

   Note: You can use any other hosting service. We use Azure in this guide because that is what we use for our CI.

2. Install WSL with `wsl --install`.
3. Enable automatic logon: use the registry to set up your machine to log on automatically. Explanation here.
4. Add your runner to your repository: head to your repository's page on GitHub > Settings > Actions > Runners > New self-hosted runner. Follow the instructions. Make sure you do not enable running it as a service.
5. Set up your runner as a startup application:
   1. Go to the directory you installed the GitHub runner.
   2. Right-click on the `run.cmd` file, and click *Show more options* > *Send to* > *Desktop (create shortcut)*.
   3. Press Win+R, type `shell:startup` and press **OK**. A directory will open.
   4. Find the shortcut in the desktop and drag it to the startup directory.
6. Set up your repository secrets To add a new secret, head to your repository's page on GitHub > Settings > Secrets > Actions > New repository secret. You'll need the following secret:
   - `AZURE_VM_CREDS`: See the documentation here.
7. Create your GitHub workflow. This workflow must have at least three jobs which depend each on the previous one.
   1. Start up the VM
   2. Your workflow(s)
   3. Stop the VM

   It is also recommended to add a `concurrency` directive to prevent different workflows from interleaving steps 1 and 3.
8. Use our actions. We developed some actions to help you build your workflow. They are documented in the *WSL GitHub actions reference*.

Example repositories

The following repositories use some variation of the workflow explained here.

- Ubuntu/WSL-example hello world example
- Ubuntu/WSL-example cloud-init testing
- Ubuntu/WSL end-to-end tests

2.3 Reference

2.3.1 Distributions

Our flagship distribution is Ubuntu. This is the one that is installed by default when you install WSL for the first time. However, we develop several flavours. It may be the case that one or more of these flavours fits your needs better.

Each of these flavours corresponds to a different application on the Microsoft Store, and once installed, they'll create different distros in your WSL. These are the applications we develop and maintain:

- Ubuntu ships the latest stable LTS release of Ubuntu. When new LTS versions are released, Ubuntu can be upgraded once the first point release is available.
- Ubuntu 18.04.6 LTS, Ubuntu 20.04.6 LTS, and Ubuntu 22.04.3 LTS are the LTS versions of Ubuntu and receive updates for five years. Upgrades to future LTS releases will not be proposed.
- Ubuntu (Preview) is a daily build of the latest development version of Ubuntu previewing new features as they are developed. It does not receive the same level of QA as stable releases and should not be used for production workloads.

Naming

Due to different limitations in different contexts, these applications will have different names in different contexts. Here is a table matching them.

1. App name is the name you'll see in the Microsoft Store and Winget.
2. AppxPackage is the name you'll see in `Get-AppxPackage`.
3. Distro name is the name you'll see when doing `wsl -l -v` or `wsl -l --online`.
4. Executable is the program you need to run to start the distro.

App name	AppxPackage name	Distro name	Executable
`Ubuntu`	`CanonicalGroupLimited.Ubuntu`	`Ubuntu`	`ubuntu.exe`
`Ubuntu (Preview)`	`CanonicalGroupLimited.UbuntuPreview`	`Ubuntu-Preview`	`ubuntupreview.exe`
`Ubuntu XX.YY.Z LTS`	`CanonicalGroupLimited.UbuntuXX.YYLTS`	`Ubuntu-XX.YY`	`ubuntuXXYY.exe`

2.3.2 GitHub actions

Download rootfs

Download the latest Rootfs tarball for a particular release of Ubuntu WSL. This can be used when you need better granularity than what is offered by *wsl-install*, or you want to cache the rootfs.

Its arguments are:

- `distros`: a comma-separated list of distros to download. Use the names as shown in WSL. Read more: *Ubuntu WSL distributions*. Defaults to `Ubuntu`.
- `path`: the path where to store the tarball. The tarball will end up as `${path}\${distro}.tar.gz`. PowerShell-style environment variables will be expanded. If there already exists a tarball at the download path, a checksum comparison will be made to possibly skip the download.

Example usage:

```yaml
- name: Download Jammy rootfs
  uses: Ubuntu/WSL/.github/actions/download-rootfs@main
  with:
   distro: Ubuntu-22.04
   path: '${env:UserProfile}\Downloads\rootfs'
```

WSL install

See also: *download-rootfs*

This action installs the Windows Subsystem for Linux application, and optionally an Ubuntu WSL application.

Its arguments are:

- `distro`: Optional argument
  - Blank (default): don't install any Ubuntu WSL distro
  - Distro name: any of the available distros in the Microsoft store. Write its name as shown in WSL. Read more: *Ubuntu WSL distributions*

Example usage:

```yaml
- name: Install or update WSL
  uses: Ubuntu/WSL/.github/actions/wsl-install@main
  with:
   distro: Ubuntu-20.04
```

WSL checkout

This action checks out your repository in a WSL distro. If you want to check it out on the Windows file system, use the regular `actions/checkout` action instead. Example usage:

Its arguments are:

- `distro`: an installed WSL distro. Write its name as it would appear on WSL. Read more: *Ubuntu WSL distributions*
- `working-dir`: the path where the repository should be cloned. Set to ~ by default.
- `submodules:`: Whether to fetch sub-modules or not. False by default.

Example usage:

```yaml
- name: Check out the repository
  uses: Ubuntu/WSL/.github/actions/wsl-checkout@main
  with:
   distro: Ubuntu-20.04
   working-dir: /tmp/github/
   submodules: true
```

WSL bash

This action runs arbitrary bash code in your distro.

Its arguments are:

- `distro`: an installed WSL distro. Write its name as it would appear on WSL. Read more: *Ubuntu WSL distributions*
- `exec`: the script to run.
- `working-dir`: path to the WSL directory to run the script in. Set to ~ by default.

Example usage:

```yaml
- name: Install pip
  uses: Ubuntu/WSL/.github/actions/wsl-bash@main
  with:
   distro: Ubuntu-20.04
   working-dir: /tmp/github/
   exec: |
       DEBIAN_FRONTEND=noninteractive sudo apt update
       DEBIAN_FRONTEND=noninteractive sudo apt install python3-pip
```