Ubuntu on WSL

Canonical Ltd.

Feb 21, 2025

CONTENTS

Windows Subsystem for Linux (WSL) enables developers to run a GNU/Linux environment on Windows. The Ubuntu distribution for WSL is tightly integrated with the Windows OS, supporting features including remote development with popular IDEs and cross-OS file management. Ubuntu can be used as a terminal interface on Windows and can also launch Linux-native graphical applications.

Ubuntu Pro for WSL (UP4W) is an in-development automation tool for managing instances of Ubuntu on WSL. If you are responsible for a fleet of Windows devices, UP4W will help you to monitor, customise and secure Ubuntu WSL instances at scale.

Ubuntu on WSL provides a fully-featured Ubuntu experience on Windows, suitable for learning Linux, developing a personal open-source project or building for production in an enterprise environment.

CHAPTER

ONE

IN THIS DOCUMENTATION

Tutorials **Start here** and learn the basics of:

- *Developing with Ubuntu on WSL*
- *Securing WSL with the Ubuntu Pro app*

How-to guides **Follow guides** for common tasks, such as:

- *Installing an Ubuntu distro on WSL*
- *Installing Ubuntu Pro for WSL*

Reference **Find technical information**, including:

- *Ubuntu distributions available for WSL*

Explanation **Build an understanding** of:

- *The architecture of Ubuntu Pro for WSL*

CHAPTER

TWO

PROJECT AND COMMUNITY

Ubuntu on WSL is a member of the Ubuntu family. It's an open-source project that warmly welcomes community contributions, suggestions, fixes and constructive feedback. Check out our *contribution guidelines* on GitHub in order to bring ideas, report bugs, participate in discussions and much more!

Thinking about using Ubuntu on WSL for your next project? Get in touch!

2.1 Tutorials

These tutorials guide you through setting up Ubuntu on WSL and using the Ubuntu Pro for WSL (UP4W) application.

2.1.1 The Ubuntu distribution on WSL

Start by learning to set up a development environment with Ubuntu on WSL by building and testing a small web project.

Develop with Ubuntu on WSL

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 install WSL and Ubuntu on WSL from the terminal
- How to set up Visual Studio Code for remote development with Ubuntu on WSL
- How to create a basic Node.js webserver on Ubuntu using Visual Studio Code
- How to preview HTML served from an Ubuntu WSL instance in a native browser on Windows

What you will need

- A PC with Windows 10 or 11

Install Ubuntu on WSL2

Install WSL

Open a PowerShell prompt as an Administrator and run:

```
> wsl --install
```

This command will enable the features necessary to run WSL and also install the latest Ubuntu distribution available for WSL. It is recommended to reboot your machine after this initial installation to complete the setup.

Install Ubuntu on WSL

WSL supports a variety of Ubuntu releases. Check our *reference on the distributions* for more information.

There are multiple ways of installing Ubuntu on WSL, here we focus on using the terminal. For other installation methods, refer to your dedicated guide:

- *Install Ubuntu on WSL2*

In a PowerShell terminal, run `wsl --list --online` to see all available distros and versions:

```
The following is a list of valid distributions that can be installed.
The default distribution is denoted by '*'.
Install using 'wsl --install -d <Distro>'.

  NAME                                   FRIENDLY NAME
* Ubuntu                                 Ubuntu
  Debian                                 Debian GNU/Linux
  kali-linux                             Kali Linux Rolling
  Ubuntu-18.04                           Ubuntu 18.04 LTS
  Ubuntu-20.04                           Ubuntu 20.04 LTS
  Ubuntu-22.04                           Ubuntu 22.04 LTS
  Ubuntu-24.04                           Ubuntu 24.04 LTS
...
```

Your list may be different once new distributions become available.

You can install a version using a NAME from the output, for example:

```
> wsl --install -d Ubuntu-24.04
```

You'll see an indicator of the installation progress in the terminal:

```
Installing: Ubuntu 24.04 LTS
[==========================72,0%==========                ]
```

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

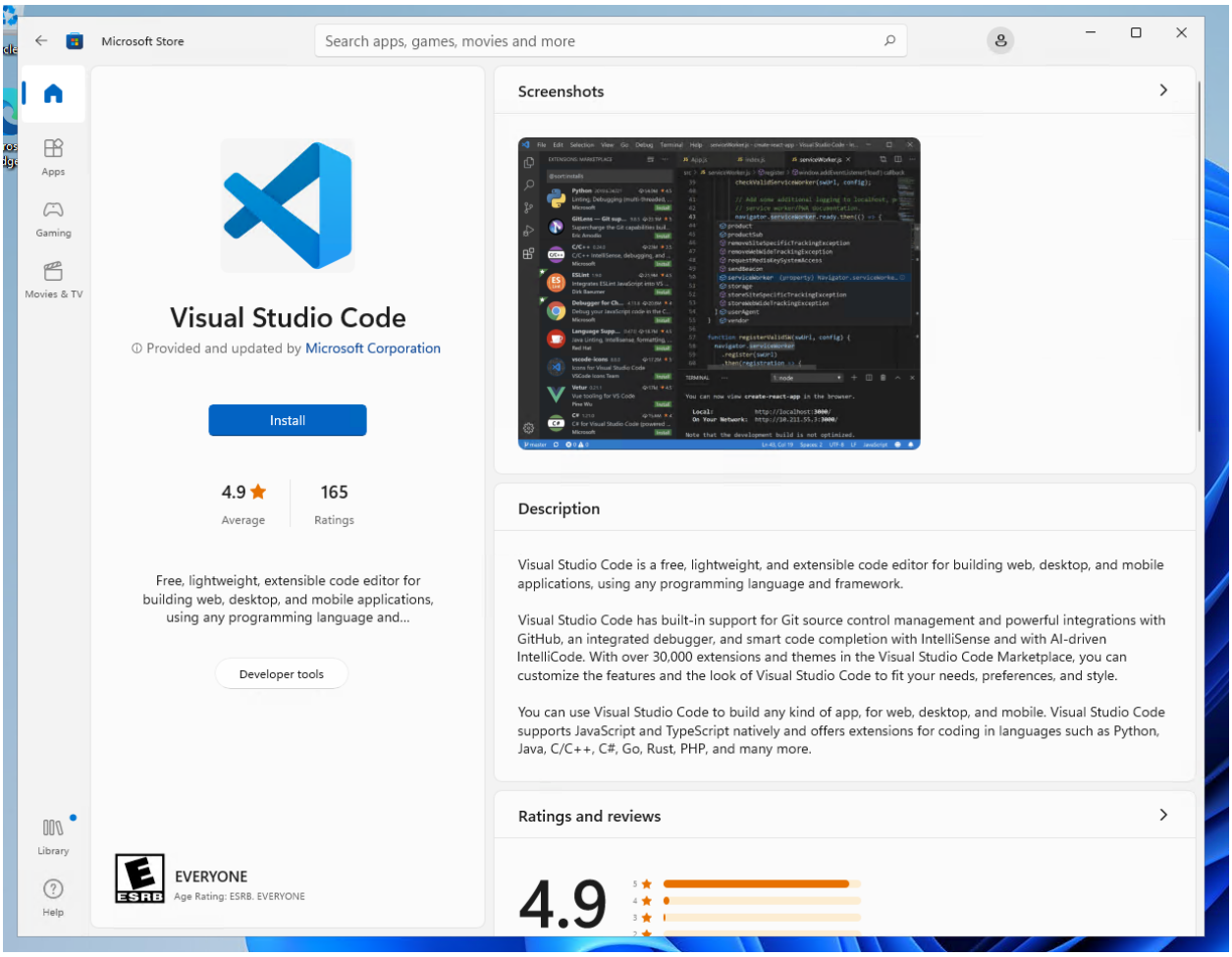
```
  NAME            STATE           VERSION
  Ubuntu-20.04    Stopped         2
* Ubuntu-24.04    Stopped         2
```

Install Visual Studio Code on Windows

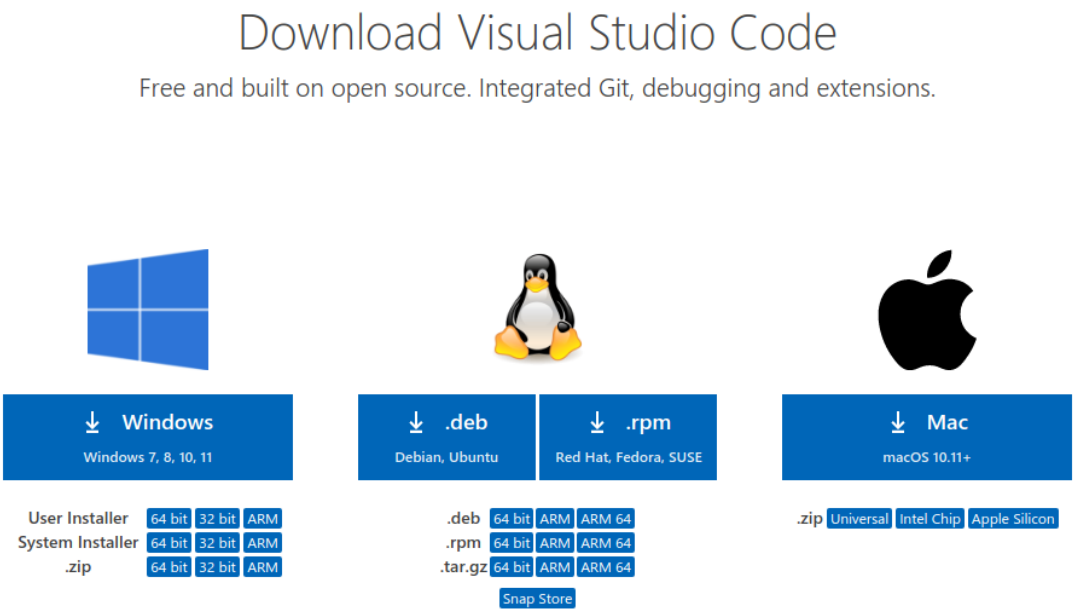
One of the advantages of WSL is that it can interact with the native Windows version of Visual Studio Code using its remote development extension.

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

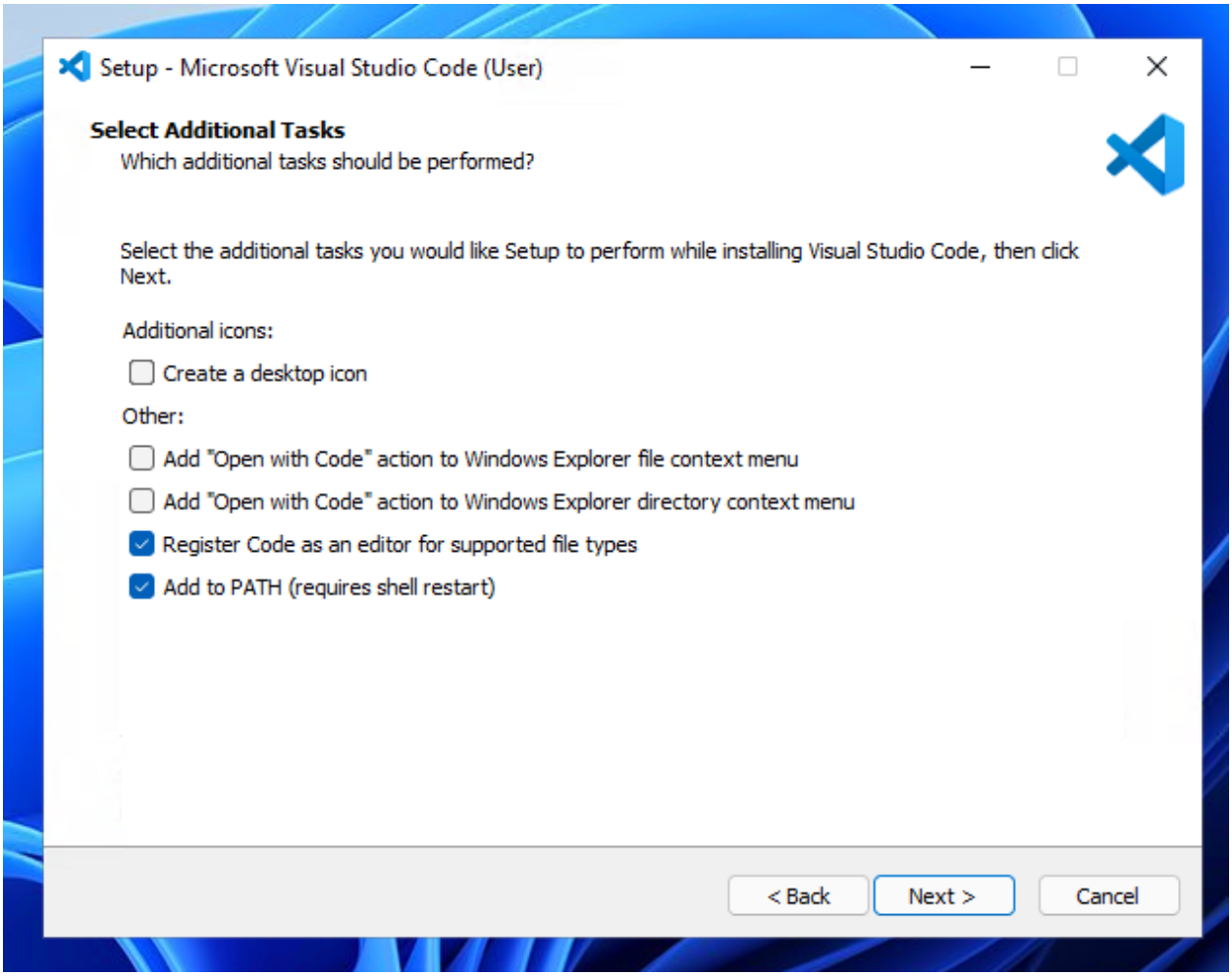
Then click **Install**.



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



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

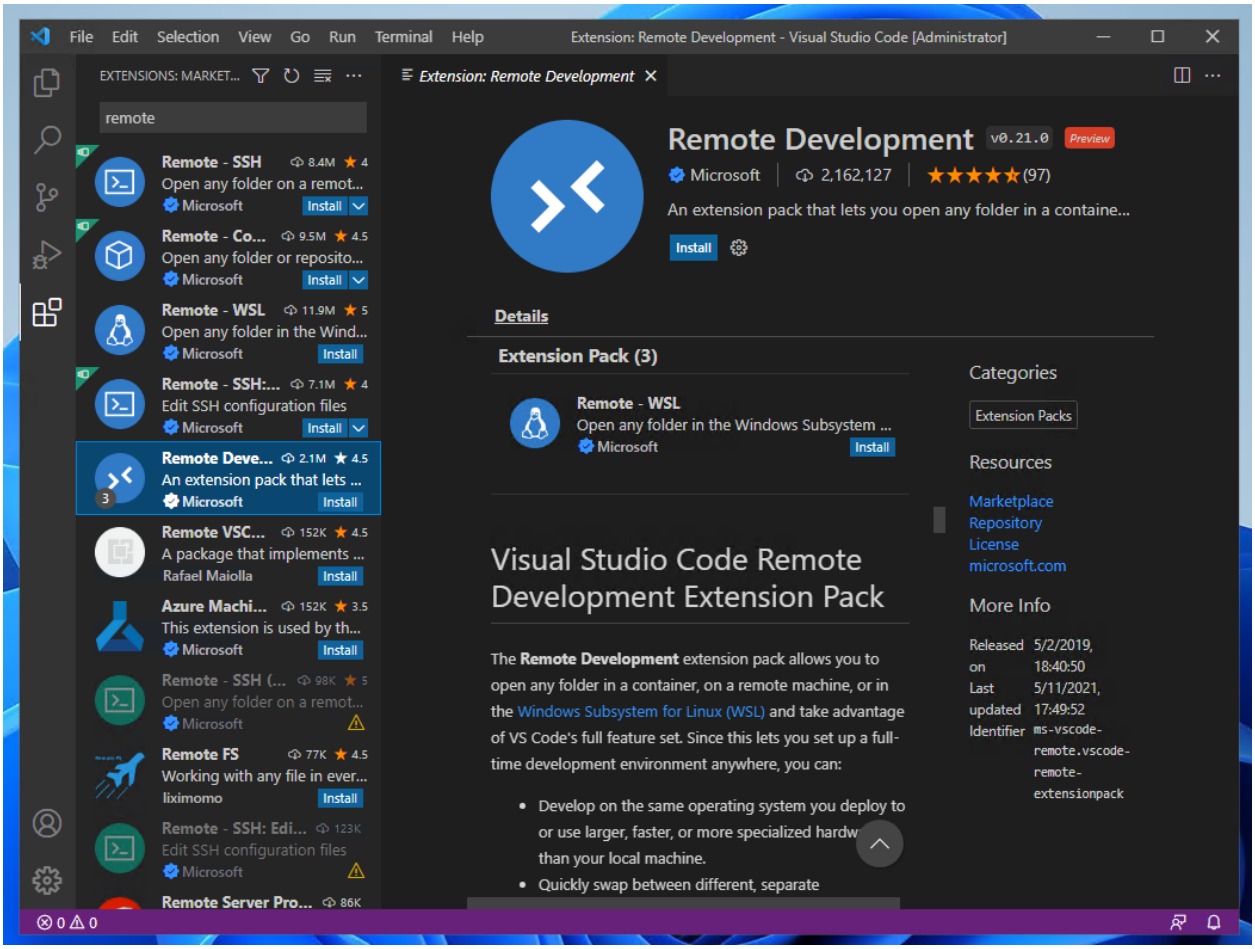


Once the installation is complete, open Visual Studio Code.

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`.



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

```
$ sudo apt update
```

Then:

```
$ sudo apt upgrade -y
```

Entering your password 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/
```

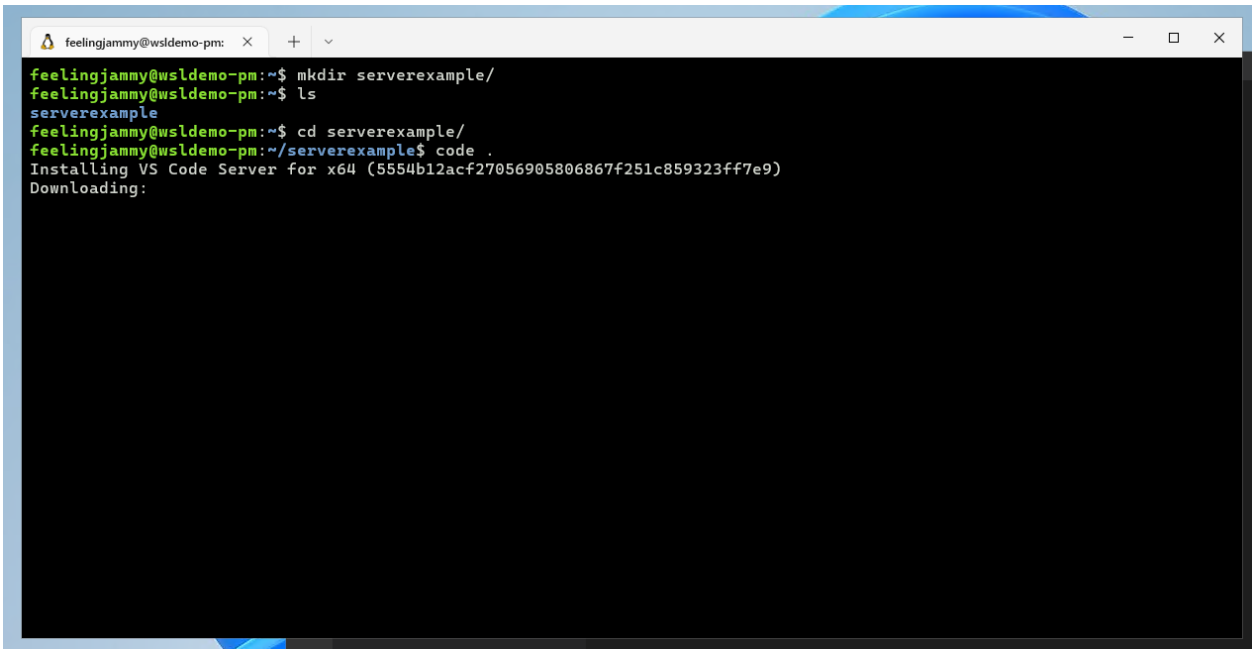
Then navigate into it:

```
$ cd serverexample/
```

Now, open up your folder in Visual Studio Code, with the following command:

```
$ code .
```

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



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 `package.json` file 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, then save and close:

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

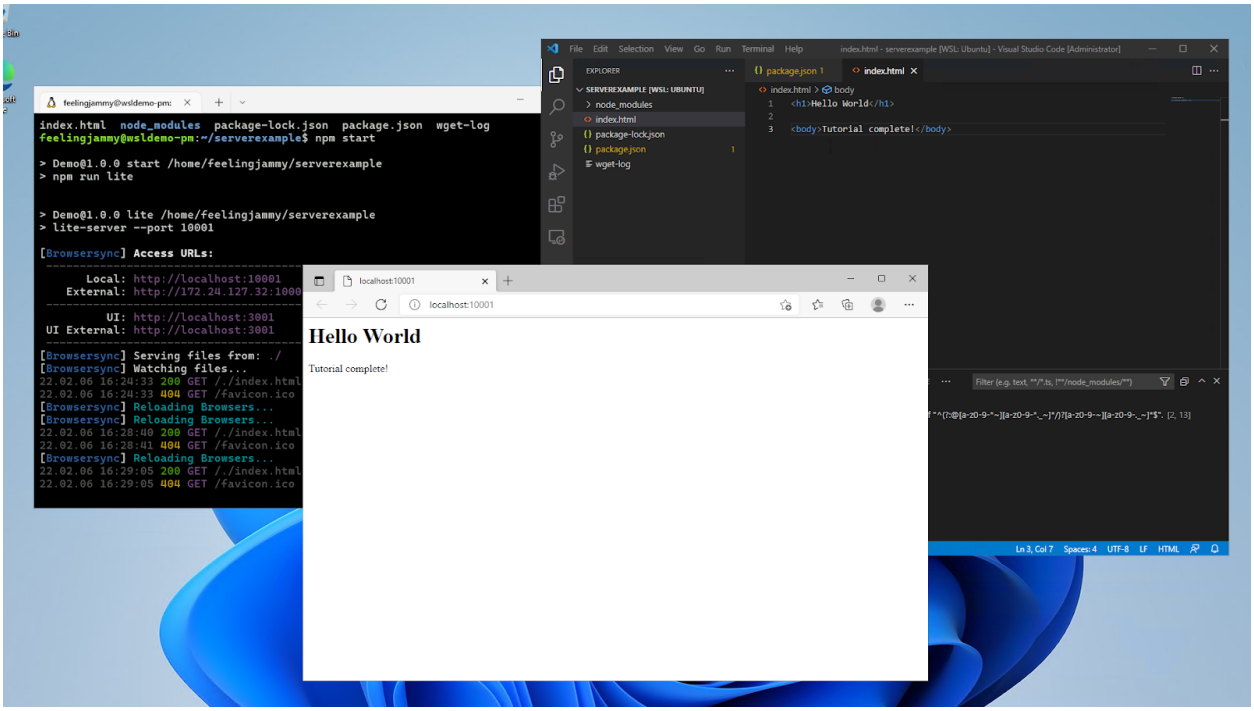
Now return to your Ubuntu terminal (or use the Visual Studio Code terminal window) and type the following to install a server defined by the above specifications detailed in `package.json`:

```
$ 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+LeftClick` on the terminal links.



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!

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 Ubuntu Pro for WSL

Then learn to automatically Pro-attach your Ubuntu WSL instances with the Ubuntu Pro for WSL application.

Get started with UP4W

> **Pro feature**
>
> This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

Windows Subsystem for Linux (WSL) is an easy and fast way to run Ubuntu on a Windows machine. Ubuntu Pro for WSL (UP4W) automatically attaches Ubuntu WSL instances to your Ubuntu Pro subscription. Developers get to use Ubuntu WSL while benefiting from the stability, security and compliance offered by Ubuntu Pro.

In this tutorial you will learn how to install UP4W on Windows and verify that Ubuntu WSL instances are Pro-attaching. You should then be ready for more advanced usage scenarios.

What you will do

- Install UP4W from the Microsoft Store
- Configure UP4W with a Pro token
- Test automatic Pro-attachment of WSL instances

> **Warning**
>
> **If you already have Ubuntu WSL pre-installed:**
>
> We recommend that any Ubuntu WSL installed is exported then deleted. You can then install it as described in this tutorial. At the end of the tutorial you can import and restore your data.
>
> Read our *how-to guide on backup and restore*.

What you will need

- A Windows 10 or 11 machine with a minimum of 16GB RAM and 8-core processor
- Some familiarity with commands for the Linux shell and PowerShell

> **Note**
>
> WSL enables using a Linux shell and Windows PowerShell side-by-side on the same machine. In this tutorial, commands will be prefixed by a prompt that indicates the shell being used, for example:
>
> - `PS C:\Users\me\tutorial>` is a PowerShell prompt where the current working directory is `C:\Users\me\tutorial`.
> - `u@mib:~/tutorial$` indicates a Linux shell prompt login as user "u" where the current working directory is `/home/ubuntu/tutorial/`
>
> Output logs are included in this tutorial when instructive but are sometimes omitted to save space.

Set up Ubuntu WSL

Install WSL

WSL can be installed directly from the Microsoft Store.

If you already have WSL installed, with `~\.wslconfig` on your system, you are advised to backup the file then remove it before continuing the tutorial.

To check if the file exists run:

```
PS C:\Users\me\tutorial> Test-Path -Path "~\.wslconfig"
```

If this returns `True` then the file exists and can be removed with:

```
PS C:\Users\me\tutorial> Remove-Item ~\.wslconfig
```

Install Ubuntu

Ubuntu 24.04 LTS is recommended for this tutorial and can be installed from the Microsoft Store:

> Install Ubuntu 24.04 LTS from the Microsoft Store

For other installation options refer to our install Ubuntu on WSL2 guide.

At this point, running `ubuntu2404.exe` in PowerShell will launch an Ubuntu WSL instance and log in to its shell.

To manually associate that Ubuntu instance with a Pro subscription you could run the `sudo pro attach` command from within the Ubuntu instance.

This, however, would need to be repeated manually for each new instance. UP4W solves this scalability problem by automating Pro-attachment. Next, let's take a look at how that works in practice.

Set up Ubuntu Pro for WSL

Get an Ubuntu Pro token

An active Ubuntu Pro subscription provides you with a token that can be added to the Ubuntu Pro client on WSL instances.

Your subscription token can be retrieved from the Ubuntu Pro Dashboard.

Visit the Ubuntu Pro page if you need a new subscription. The `Myself` option for a personal subscription is free for up to 5 machines.

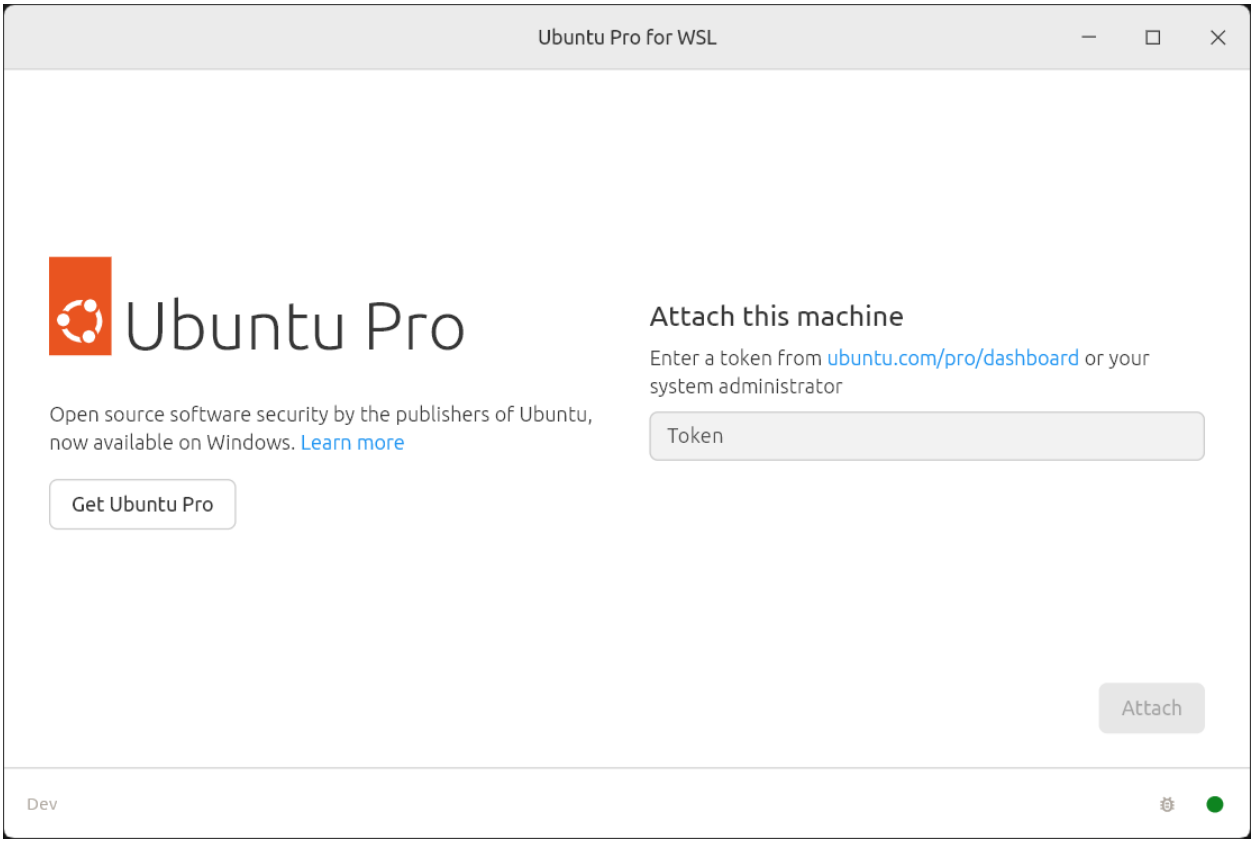
Once you have a token you are ready to install UP4W.

Install and configure UP4W

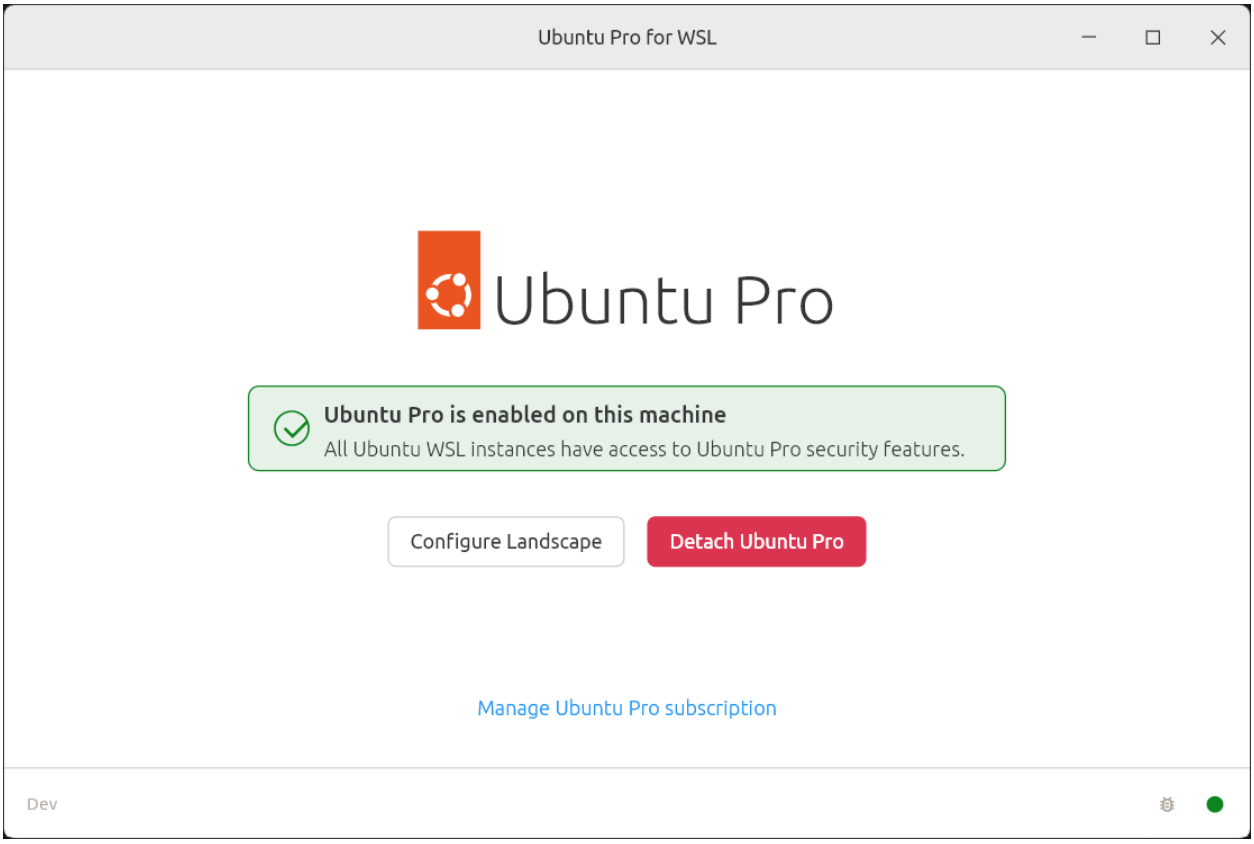
> **Warning**
>
> The install link below will work only if you're logged in to the Microsoft Store with an account for which access to the app has been enabled.

UP4W can be installed from this link to the Microsoft Store.

Open the application and paste the token you copied from the Ubuntu Pro dashboard:



After you confirm, a status screen will appear showing that configuration is complete:



Done! You can close the UP4W window before continuing. If at any time you want to detach your Pro subscription just open the UP4W application and select **Detach Ubuntu Pro**.

Your Ubuntu Pro subscription is now attached to UP4W on the Windows host. UP4W will automatically forward the subscription to the Ubuntu Pro client on your Ubuntu WSL instances.

Verify Pro-attachment

All Ubuntu WSL instances will now be automatically added to your Ubuntu Pro subscription.

Open Windows PowerShell and run the following command to create a new Ubuntu 24.04 instance, entering a user and password when prompted. For quick testing, set both to `u`:

```
PS C:\Users\me\tutorial> ubuntu2404.exe
```

You will now be logged in to the new instance shell and can check that UP4W has Pro-attached this instance with:

```
u@mib:~$ pro status
```

The output should indicate that services like ESM are enabled, with account and subscription information also shown:

```
SERVICE          ENTITLED  STATUS       DESCRIPTION
esm-apps         yes       enabled      Expanded Security Maintenance for Applications
esm-infra        yes       enabled      Expanded Security Maintenance for Infrastructure

NOTICES
Operation in progress: pro attach

```

(continues on next page)

(continued from previous page)

```
For a list of all Ubuntu Pro services, run 'pro status --all'
Enable services with: pro enable <service>

     Account: me@ubuntu.com
Subscription: Ubuntu Pro - free personal subscription
```

Packages can also be accessed from all the enabled services. Running `sudo apt update` will produce output like the following:

```
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://ppa.launchpad.net/ubuntu-wsl-dev/ppa/ubuntu noble InRelease
Hit:3 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:4 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:5 http://ppa.launchpad.net/landscape/self-hosted-beta/ubuntu noble InRelease
Hit:6 https://esm.ubuntu.com/apps/ubuntu noble-apps-security InRelease
Hit:7 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:8 http://ppa.launchpad.net/cloud-init-dev/proposed/ubuntu noble InRelease
Hit:9 https://esm.ubuntu.com/infra/ubuntu noble-infra-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
```

Now let's check that another Ubuntu instance will also Pro-attach.

Install Ubuntu 22.04 LTS directly from PowerShell:

```
PS C:\Users\me\tutorial> wsl --install Ubuntu-22.04
```

Once you are in the instance shell, enter a username and password then run `pro status`. You should again get confirmation of successful Pro-attachment for the new instance.

> If you want to uninstall UP4W after this tutorial refer to *our how-to guide*.

Next steps

This is only the start of what you can do with UP4W.

If you need to create and manage large numbers of Ubuntu WSL instances you will probably want to use the Windows registry. By using the Windows registry you can associate a Pro token with each new WSL instance using your organisation's own deployment solution.

> For detailed step-by-step instructions on using the Windows registry read our short guide on how to *install and configure UP4W*.

Landscape support is also built-in to UP4W. With a single configuration file, you can create and manage multiple WSL instances that will automatically be registered with your Landscape server:

> For more information, please refer to our tutorial on how to *deploy WSL instances with UP4W and Landscape*.

Our documentation includes several other *how-to guides* for completing specific tasks, *reference* material describing key information relating to UP4W.

If you are interested in remote management of WSL instances, you can also learn how UP4W's Landscape integration can be used to deploy Ubuntu WSL instances.

Deploy WSL instances with UP4W and Landscape

> **Pro feature**
>
> This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

In this tutorial you will develop an understanding of how UP4W can help you deploy and manage Ubuntu WSL instance using Landscape.

What you will do

- Deploy an Ubuntu WSL instance locally
- Deploy an Ubuntu WSL instance remotely
- Test automatic configuration of WSL instances by UP4W

What you need

- A Windows 10 or 11 machine with a minimum of 16GB RAM and 8-core processor
- The latest version of Landscape Server set up and configured on a physical or virtual machine
- WSL and Ubuntu 24.04 installed on Windows
- An UP4W installation configured with a Pro token

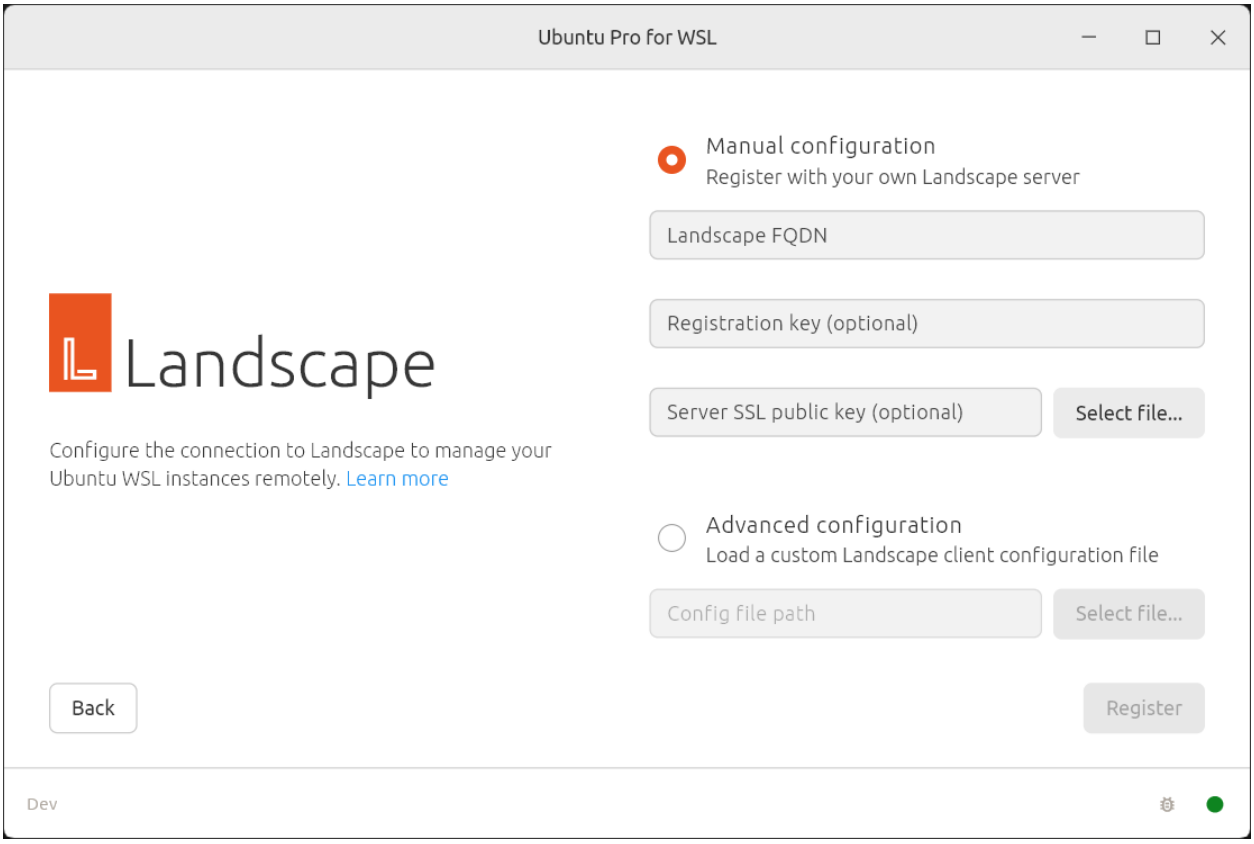
Before following this tutorial it is recommended that you complete the *getting started* tutorial to familiarise yourself with UP4W installation and configuration.

Set things up

To complete this tutorial you will need to have a Landscape server set up and you should be able access your Landscape dashboard in a browser. Please refer to the Landscape documentation for setup and configuration instructions.

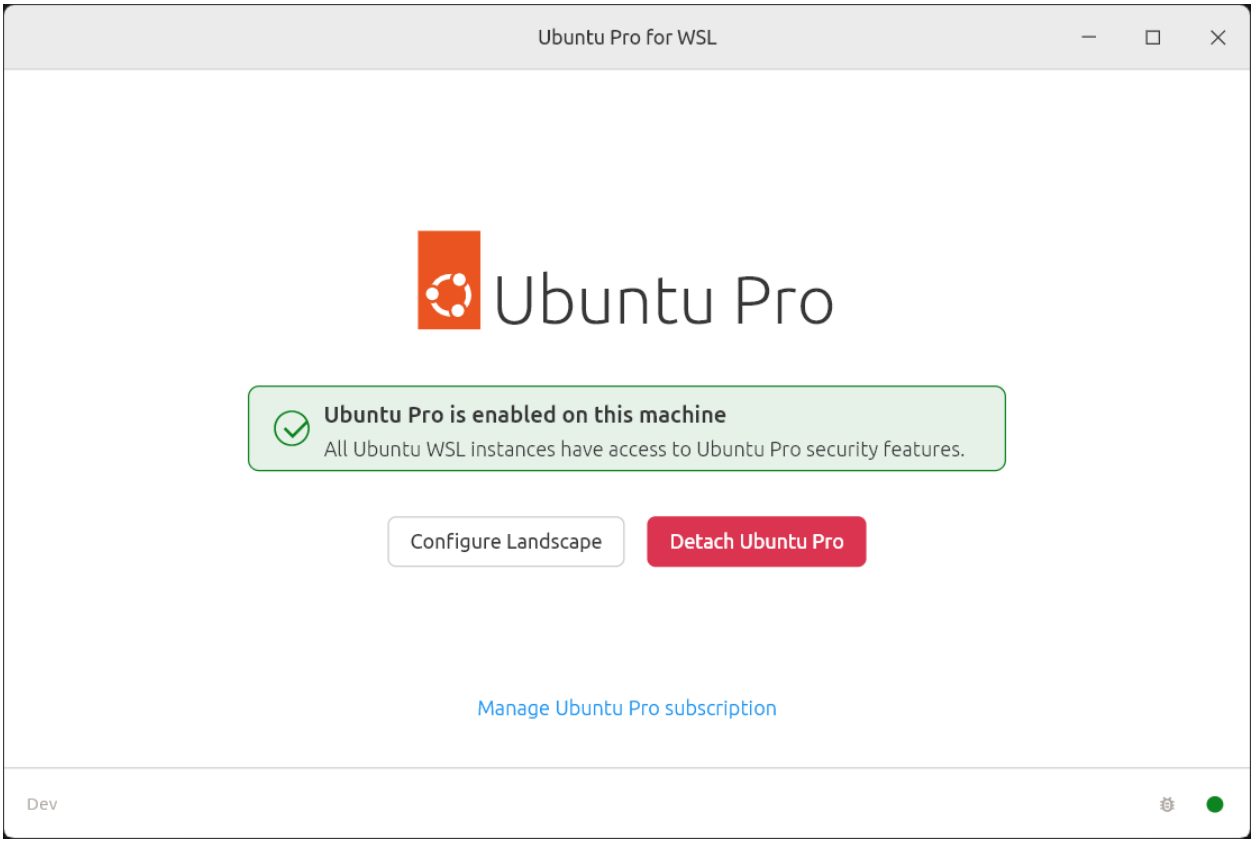
Configure Landscape in the UP4W app

In the UP4W app, after entering your Pro token, navigate to the Landscape configuration screen:



Choose your preferred configuration option and enter the required details.

When you continue a status screen will appear confirming that configuration is complete:



As well as your Ubuntu Pro subscription being attached to UP4W on the Windows host, this has also configured the Landscape client built into your UP4W Windows agent to know about your Landscape server. UP4W will forward this configuration to the Landscape client on your Ubuntu WSL instances as well, and all systems where the Landscape client has been configured this way are automatically registered with Landscape.

A dedicated how-to guide on configuring Landscape with UP4W can be found *here*.

Create an Ubuntu WSL instance locally

Open Windows PowerShell and run the following command to create a new Ubuntu 24.04 instance, creating a user and password when prompted. For quick testing, set both to `u`:

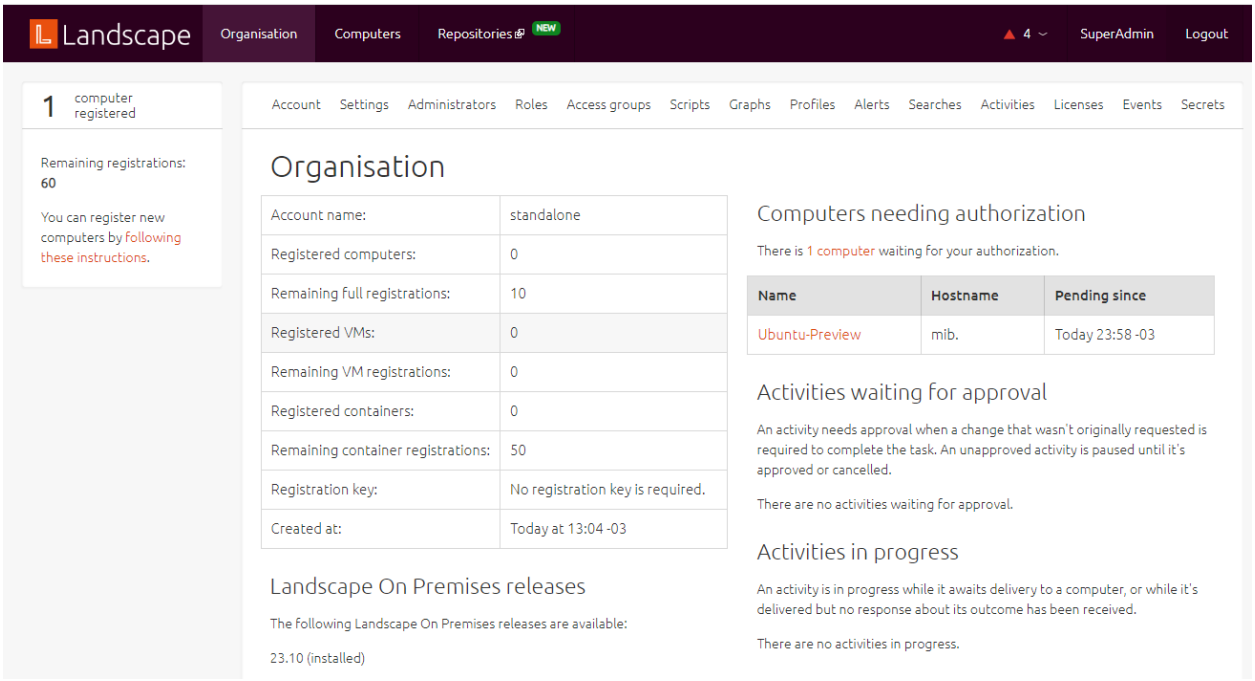
```
PS C:\Users\me\tutorial> ubuntu2404.exe
```

Verify Pro-attachment with:

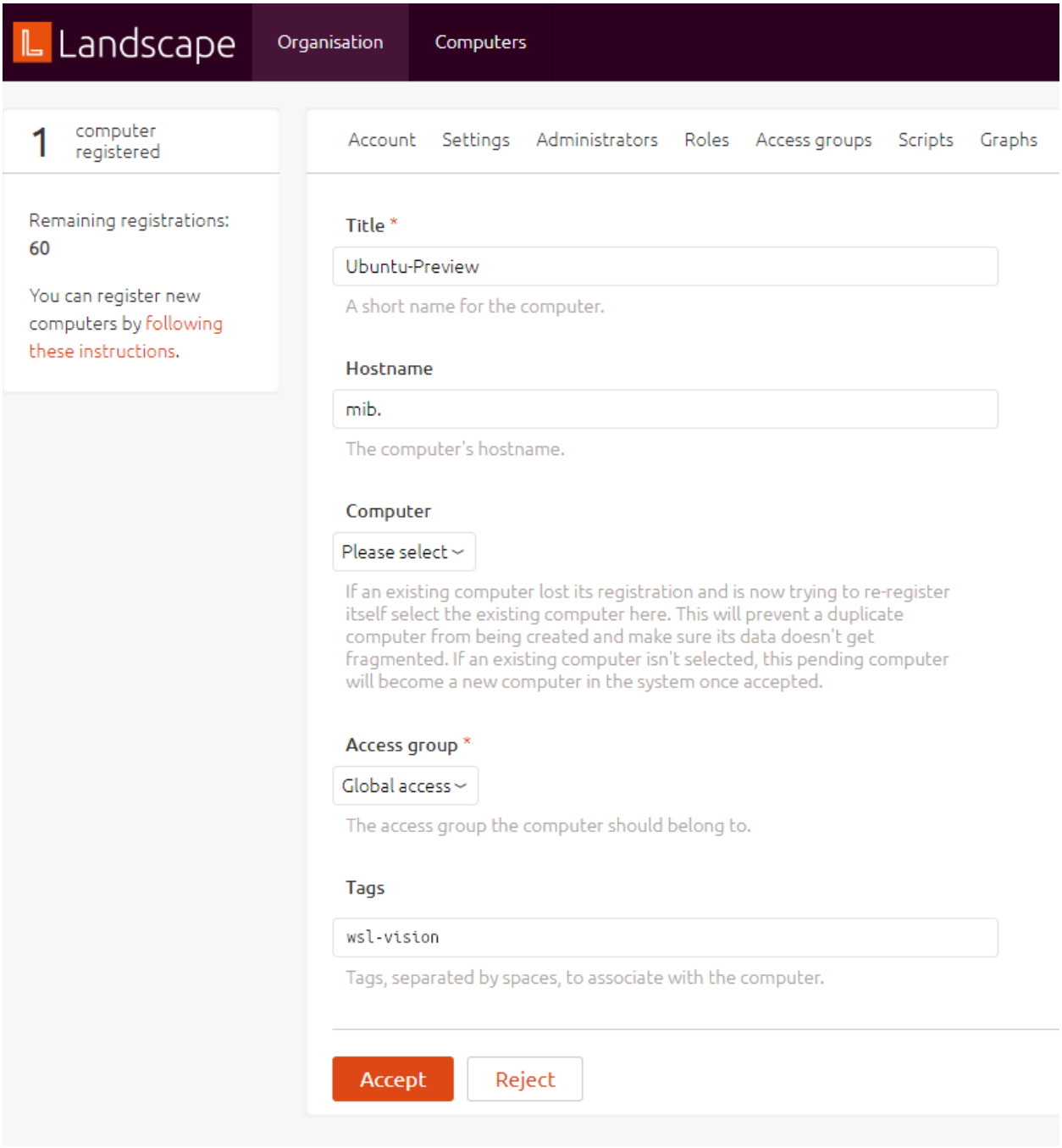
```
u@mib:~$ pro status
```

UP4W should have also Landscape-registered this instance.

To verify, refresh the Landscape server web page and the instance should be listed under “Computers needing authorisation”.

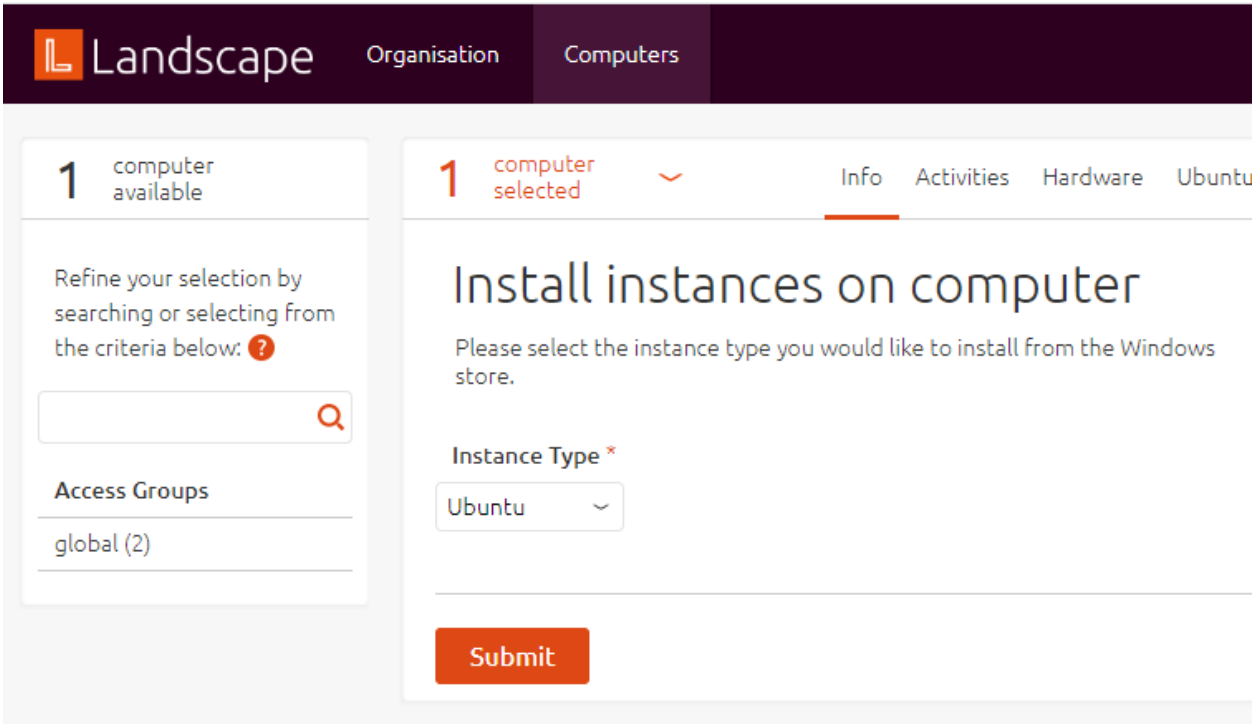


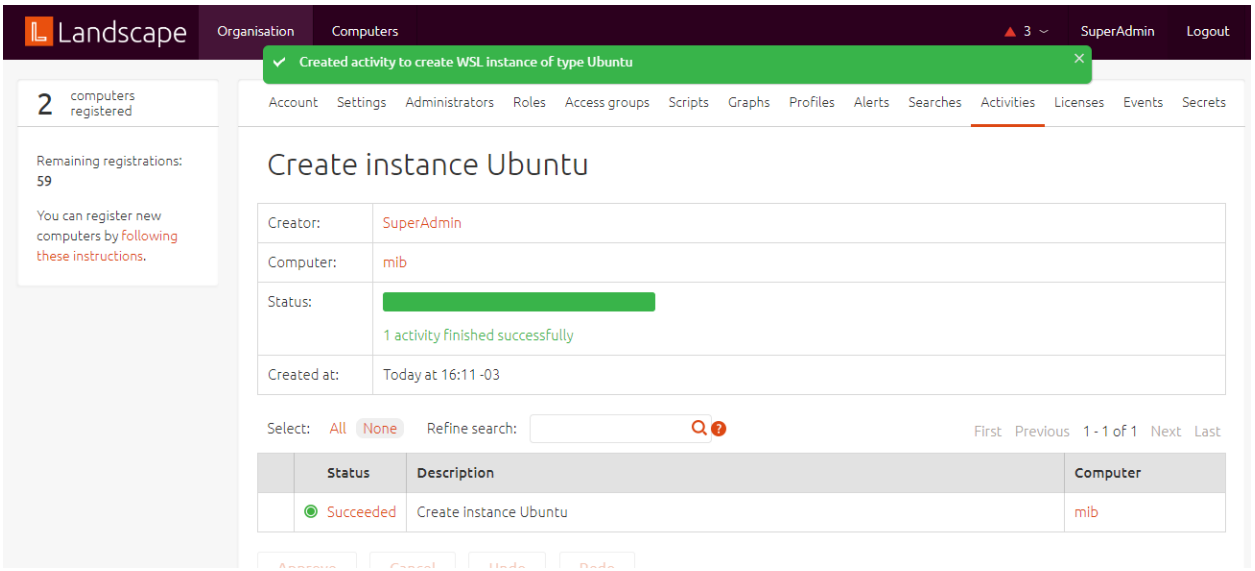
To accept the registration click on the instance name, set "Tags" to `wsl-vision` in the pop-up then click **Accept**. The `wsl-vision` tag will be used for all the instances accepted into Landscape.



Create an Ubuntu WSL instance remotely

Back on the Landscape page in your web browser, navigate to "Computers" and click on the Windows machine (below: `mib`). You will find "WSL Instances" on the right side of the page. Click on the "Install new" link then set "Instance Type" to "Ubuntu" and click **Submit**. A status page will appear showing the progress of the new instance creation.





The Landscape server will talk to the Landscape client built into your UP4W. UP4W will then install the `Ubuntu` application and create an Ubuntu WSL instance automatically. In PowerShell, run `ubuntu.exe` to log in to the new instance.

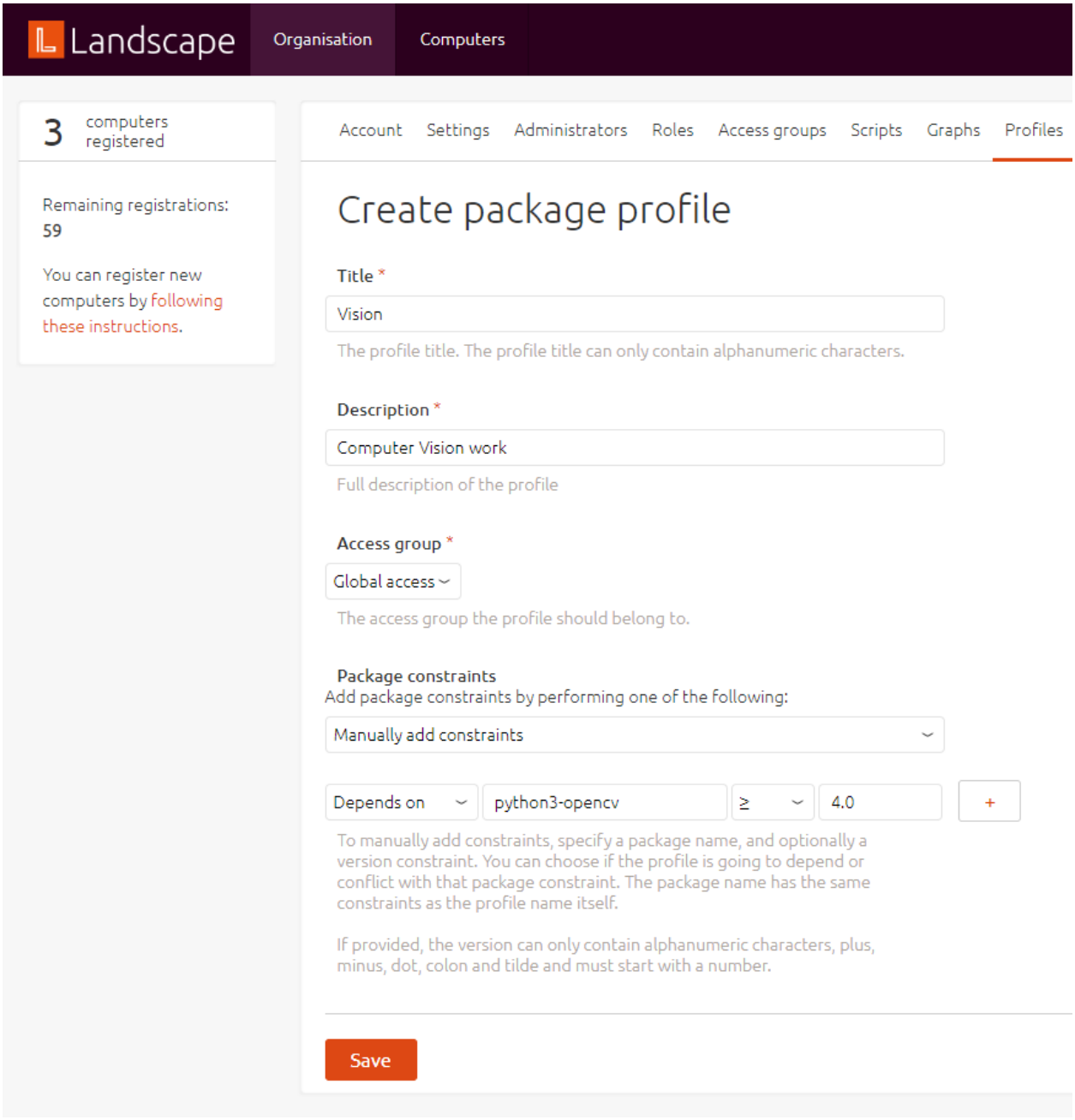
```
PS C:\Users\me\tutorial> ubuntu.exe

u@mib:~$
```

You can run `pro status` to verify pro-attachment and refresh your Landscape server page to verify and accept the registration. As before, apply the `wsl-vision` tag and click `Accept`.

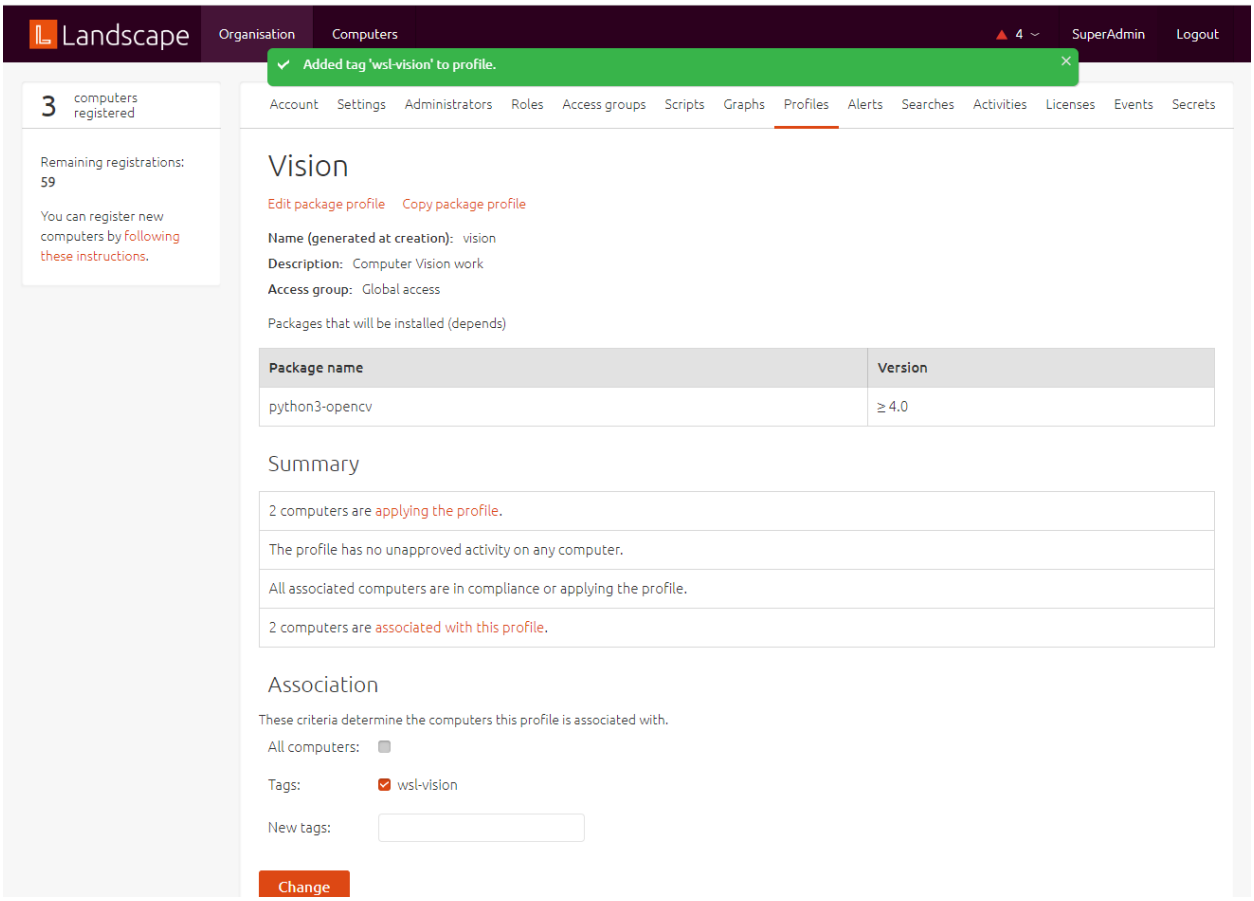
Deploy packages to all Ubuntu WSL instances

On your Landscape server page, navigate to `Organization > Profiles`, click on `Package Profiles` then `Add package profile`. Fill in the form with the following values and click "Save".

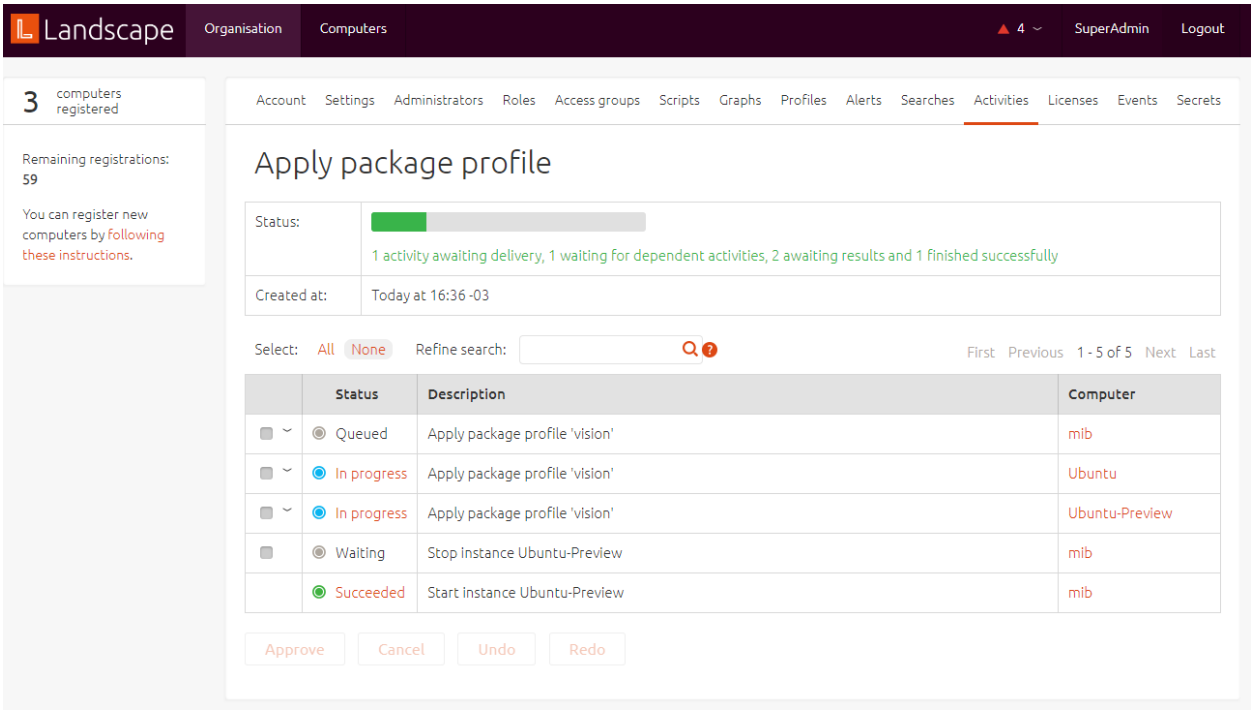
Field	Value
Title	Vision
Description	Computer Vision work
Access group	Global
Package constraints	Manually add constraints
	Depends on `python3-opencv` >= `4.0`



On the bottom of the "Vision" profile page, in the "Association" section, set the "New tags" field to `wsl-vision` and click **Change**.



In the "Summary" section in the middle of the page you will see a status message showing that two computers are `applying the profile`. Click on the `applying the profile` link and then, in the "Activities" list, click on **Apply package profile** to see the progress of the package deployment.



When this process has completed, use one of your instance shells to verify that the `python3-opencv` package has been installed. For example, in the `Ubuntu` instance the first three packages returned are:

```
u@mib:~$ apt list --installed | grep -m 3 opencv

libopencv-calib3d4.5d/jammy,now 4.5.4+dfsg-9ubuntu4 amd64 [installed,automatic]
libopencv-contrib4.5d/jammy,now 4.5.4+dfsg-9ubuntu4 amd64 [installed,automatic]
libopencv-core4.5d/jammy,now 4.5.4+dfsg-9ubuntu4 amd64 [installed,automatic]
```

You know how to leverage UP4W and Landscape to efficiently manage your Ubuntu WSL instances at scale.

Next steps

Our documentation includes several *how-to guides* for completing specific tasks and *reference* material describing key information relating to UP4W.

2.2 How-to guides

The guides in this section will help you to complete specific tasks with Ubuntu on WSL.

They include guides on the Ubuntu distribution and the Ubuntu Pro for WSL application.

2.2.1 Installation and setup

Get set up using Ubuntu on a Windows machine with WSL.

Installation and setup

These guides show you how to set up WSL with an Ubuntu distribution and the Ubuntu Pro for WSL application.

The Ubuntu distribution

Run a full Ubuntu development environment on Windows.

Install Ubuntu on WSL2

What you will learn

- How to enable and install WSL on Windows 10 and Windows 11
- How to install `Ubuntu 24.04 LTS` using the Microsoft Store or WSL commands in the terminal

What you will need

- Windows 10 or 11 running on either a physical device or virtual machine
- All of the latest Windows updates installed

Install WSL

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

```
> wsl --install
```

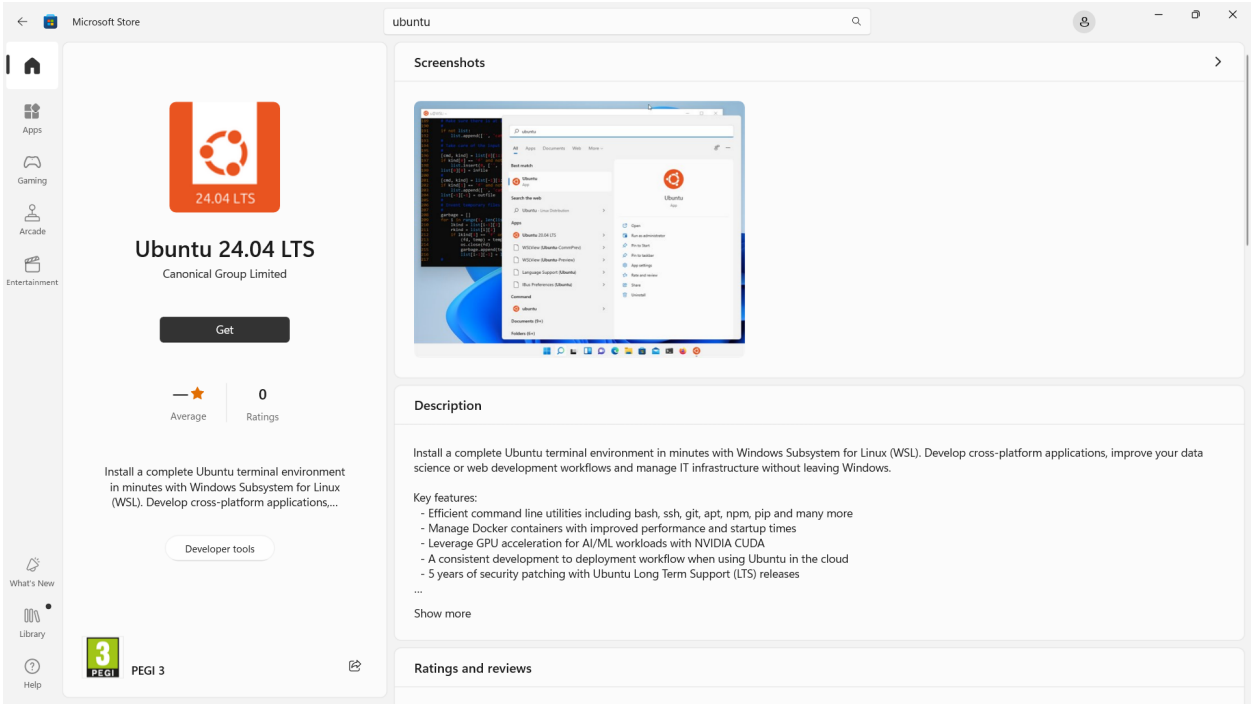
It is recommended to reboot your machine after this initial installation to complete the setup.

Install Ubuntu WSL

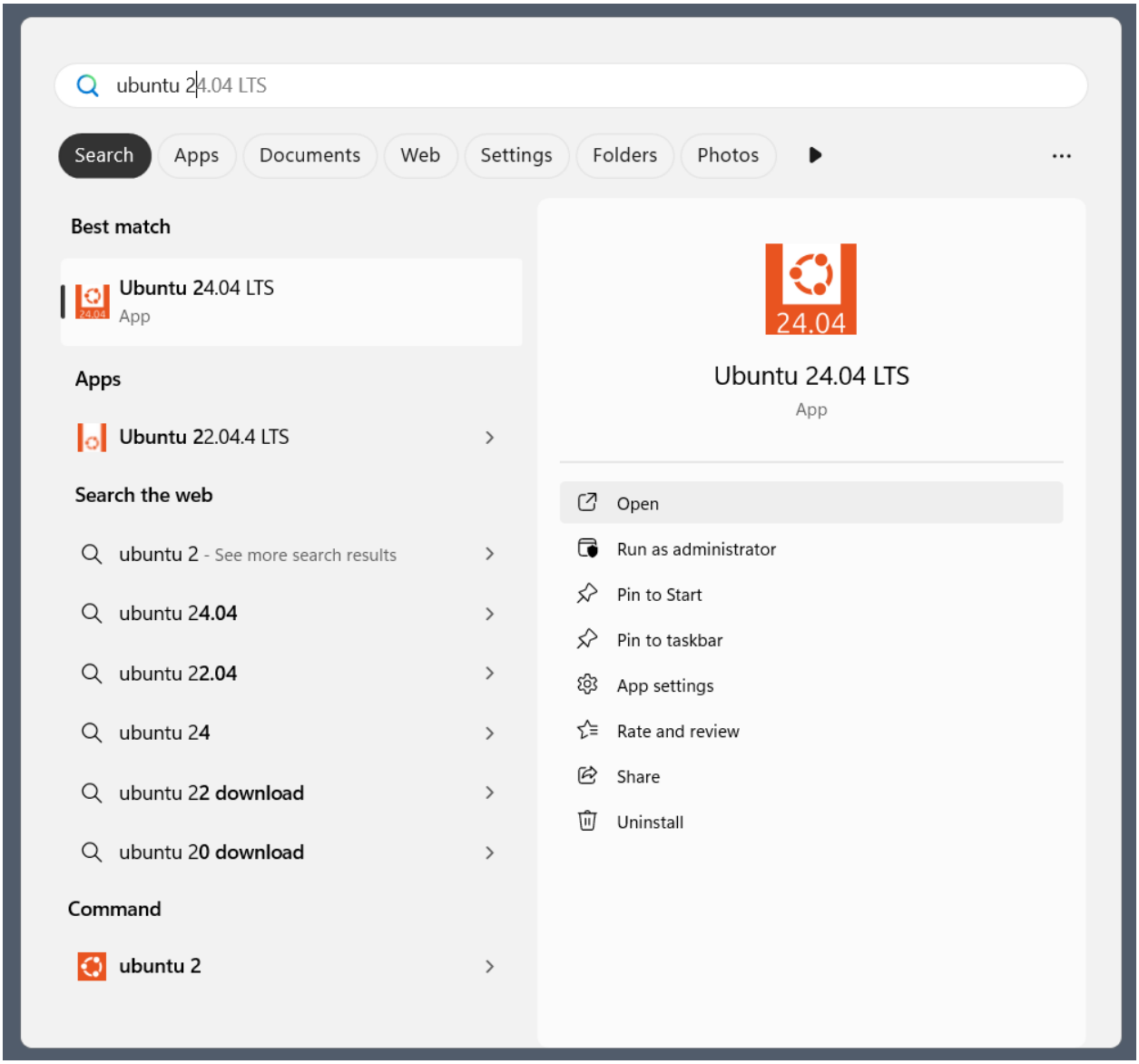
There are multiple ways of installing distros on WSL, here we focus on two: the Microsoft Store application and WSL commands run in the terminal. The result is the same regardless of the method.

Method 1: Microsoft Store application

Find the distribution you prefer on the Microsoft Store and then click **Get**.



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



Method 2: WSL commands in the terminal

In a PowerShell terminal, you can run `wsl --list --online` to see an output with all available distros and versions:

```
The following is a list of valid distributions that can be installed.
The default distribution is denoted by '*'.
Install using 'wsl --install -d <Distro>'.

  NAME                                   FRIENDLY NAME
* Ubuntu                                 Ubuntu
  Debian                                 Debian GNU/Linux
  kali-linux                             Kali Linux Rolling
  Ubuntu-18.04                           Ubuntu 18.04 LTS
  Ubuntu-20.04                           Ubuntu 20.04 LTS
  Ubuntu-22.04                           Ubuntu 22.04 LTS
  Ubuntu-24.04                           Ubuntu 24.04 LTS
```

(continues on next page)

(continued from previous page)

```
...
```

Your list may be different once new distributions become available.

You can install a version using a NAME from the output:

```
> wsl --install -d Ubuntu-24.04
```

You'll see an indicator of the installation progress in the terminal:

```
Installing: Ubuntu 24.04 LTS
[==========================72,0%==========                ]
```

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

```
  NAME            STATE           VERSION
  Ubuntu-20.04    Stopped         2
* Ubuntu-24.04    Stopped         2
```

Note on installing images without the Microsoft Store

If you do not have access to the Microsoft Store or need to install a custom image it is possible to import a distribution as a tar file:

```
> wsl --import <DistroName> <InstallLocation> <InstallTarFile>
```

Appx and MSIX packages for a given distro can also be downloaded and installed. Please refer to Microsoft's documentation for more detailed information on these installation methods:

- Importing Linux distributions
- Installing distributions without the Microsoft Store

> **Warning**
>
> You should always try to use the latest LTS release of Ubuntu, as it offers the best security, reliability and support when using Ubuntu WSL.
>
> Currently we do not have a recommended location from which to download tar and Appx/MSIX files for Ubuntu distros.

Run and configure Ubuntu

To open an Ubuntu 24.04 terminal run the following command in PowerShell:

```
> ubuntu2404.exe
```

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 by running the following commands within the Ubuntu terminal, entering your password when prompted:

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

Enjoy Ubuntu on WSL

In this guide, we've shown you how to install Ubuntu WSL on Windows 10 or 11.

We hope you enjoy working with Ubuntu in 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

Back up, restore and duplicate Ubuntu WSL instances

Motivation

You may need to backup one of your Ubuntu WSL instances, if you want to:

- Perform a clean installation without losing data
- Create a snapshot before experimenting with your instance
- Share a pre-configured instance between machines
- Duplicate an instance so it can be run and configured independently

Backing up

> **Note**
>
> For simplicity, PowerShell commands in this section will all be run from the home directory of the user `me`.

To backup an Ubuntu-24.04 instance first make a `backup` folder in your home directory:

```
PS C:\Users\me> mkdir backup
```

You then need to create a compressed version of the Ubuntu instance in that backup directory:

```
PS C:\Users\me> wsl --export Ubuntu-24.04 .\backup\Ubuntu-24.04.tar.gz
```

Removal and deletion

Once you have created a backup of your Ubuntu distro it is safe to remove it from WSL and delete all associated data.

This can be achieved with the following command:

```
PS C:\Users\me> wsl --unregister Ubuntu-24.04
```

Restoring

If you want to restore the Ubuntu-24.04 instance that you have previously backed up run:

```
PS C:\Users\me> wsl --import Ubuntu-24.04 .\backup\Ubuntu2404\ .\backup\Ubuntu-24.04.tar.
↪gz
```

This will import your previous data and if you run `ubuntu2404.exe` an Ubuntu WSL instance should be restored with your previous configuration intact.

To login as a user `k`, created with the original instance, run:

```
PS C:\Users\me> wsl -d Ubuntu-24.04 -u k
```

Alternatively, add the following to `/etc/wsl.conf` in the instance:

```
[user]
default=k
```

Without specifying a user you will be logged in as the root user.

Duplication

It is also possible to create multiple instances from a base instance. Below the restore process is repeated but the new instances are assigned different names than the original backup:

```
PS C:\Users\me> wsl --import ubuntu2404b .\backup\Ubuntu2404b\ .\backup\Ubuntu-24.04.tar.
↪gz
PS C:\Users\me> wsl --import ubuntu2404c .\backup\Ubuntu2404c\ .\backup\Ubuntu-24.04.tar.
↪gz
```

This will create two additional instances of Ubuntu 24.04 that can be launched and configured independently.

In PowerShell, running `wsl -l -v` will output the new instances in your list of installed distributions:

```
NAME            STATE           VERSION
Ubuntu-24.04    Stopped         2
ubuntu2404b     Stopped         2
ubuntu2404c     Stopped         2
```

To launch the first derived instance and login as the user `k` run:

```
PS C:\Users\me> wsl -d ubuntu2404b -u k
```

Automatic setup with cloud-init

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.

The latest release of Ubuntu 24.04 LTS (Noble Numbat) comes with cloud-init already preinstalled, so you'll need that specific application to follow this guide. Ubuntu 24.04 LTS can be installed from this link to the Microsoft Store. A previous version of this guide used Ubuntu (Preview), because that comes with the latest in-development features. You can still use it to follow the instructions below, if you prefer. This feature is now available in the default Ubuntu application as well as Ubuntu 22.04 LTS.

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 Ubuntu 24.04 LTS application from the 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-24.04.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]
    default=jdoe

packages: [ginac-tools, octave]

runcmd:
   - sudo git clone https://github.com/Microsoft/vcpkg.git /opt/vcpkg
   - sudo apt-get install zip curl -y
   - /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 guide - it requires an unusual setup).
>
> See more: WSL data source reference.

Register a new Ubuntu-24.04 instance

In PowerShell, run:

```
> ubuntu2404.exe
```

This command will register a new Ubuntu-24.04 instance that will be configured automatically by cloud-init. The process can take several minutes, depending on your computer and network speeds.

> If you want to be sure that there is now an Ubuntu-24.04 instance, run `wsl -l -v`. Notice that the application is named `Ubuntu24.04LTS` but the WSL instance created is named `Ubuntu-24.04`. See more about that naming convention in *our reference documentation*.

Verify automatic configuration by cloud-init

When the setup is complete, the WSL instance's shell will be logged in as the user `jdoe`. You should see the standard welcome text:

```
Installing, this may take a few minutes...
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.6.36.3-microsoft-standard-WSL2 x86_64)

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

 System information as of ter 01 out 2024 14:32:47 -03

  System load:  1.64               Processes:             63
  Usage of /:   0.2% of 1006.85GB  Users logged in:       0
  Memory usage: 4%                 IPv4 address for eth0: 172.22.8.90
  Swap usage:   0%


This message is shown once a day. To disable it please create the
/home/jdoe/.hushlogin file.
jdoe@mib01:~$
```

Once logged into 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
```

This should be verified with the output message:

```
jdoe
```

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

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

Verified with the output:

```
Valid schema user-data
```

3. The locale is set

```
jdoe@mib:~$ locale
```

Verified with:

```
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'
```

Verified:

```
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. Lastly, 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
```

This should also be verified with:

```
vcpkg package management program version 2024-01-11-
↪710a3116bbd615864eef5f9010af178034cb9b44

See LICENSE.txt for license information.
```

Enjoy!

That's all folks! In this guide, 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!

The Ubuntu Pro for WSL application

Secure and manage Ubuntu on WSL with an Ubuntu Pro subscription.

Install UP4W and add a Pro token

> **Pro feature**
>
> This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

To install and configure UP4W you will need:

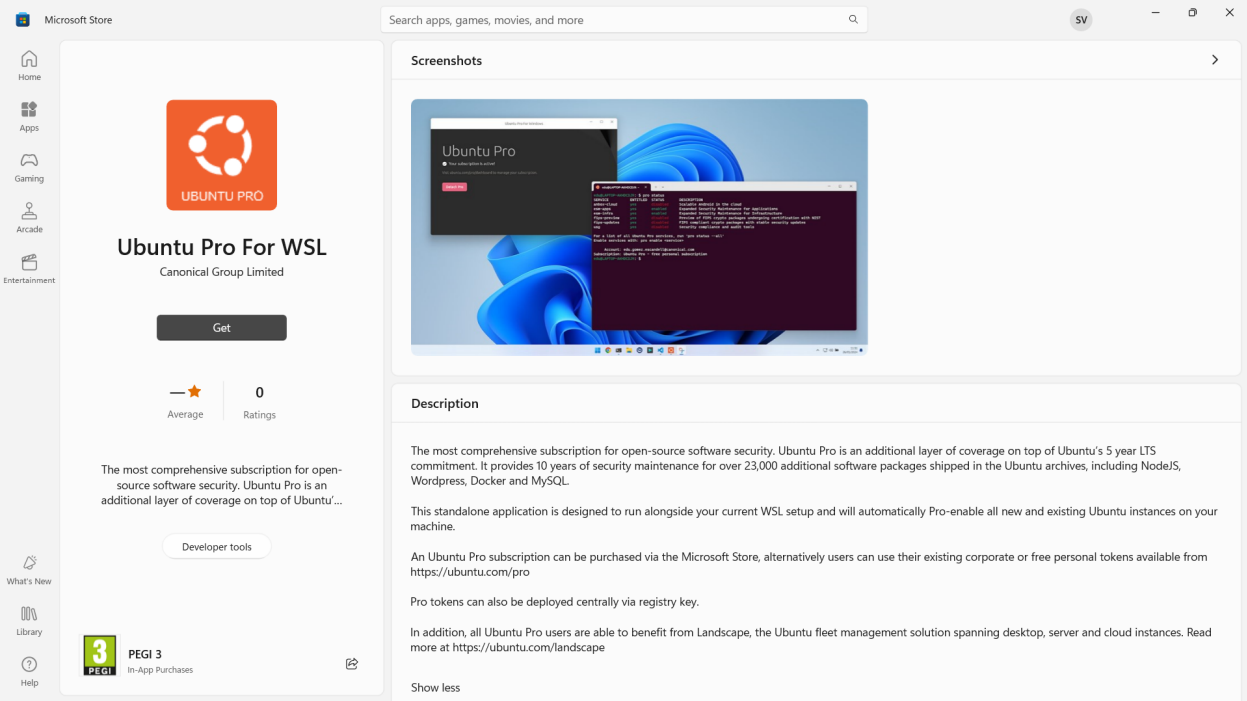
- A Windows 10 or 11 machine
- An Ubuntu Pro token

You should also verify that the *firewall rules are correctly set up*.

Install UP4W

> **Warning**
>
> The install link below will work only if you're logged in to the Microsoft Store with an account for which access to the app has been enabled.

You can install UP4W on this page of the Microsoft Store:



Choose a configuration method

After installation has finished you can start configuring UP4W in two ways:

- Windows registry: easier to automate and deploy at scale
- Graphical Windows application: convenient option for individual users

Click the appropriate tab to read more.

Windows registry

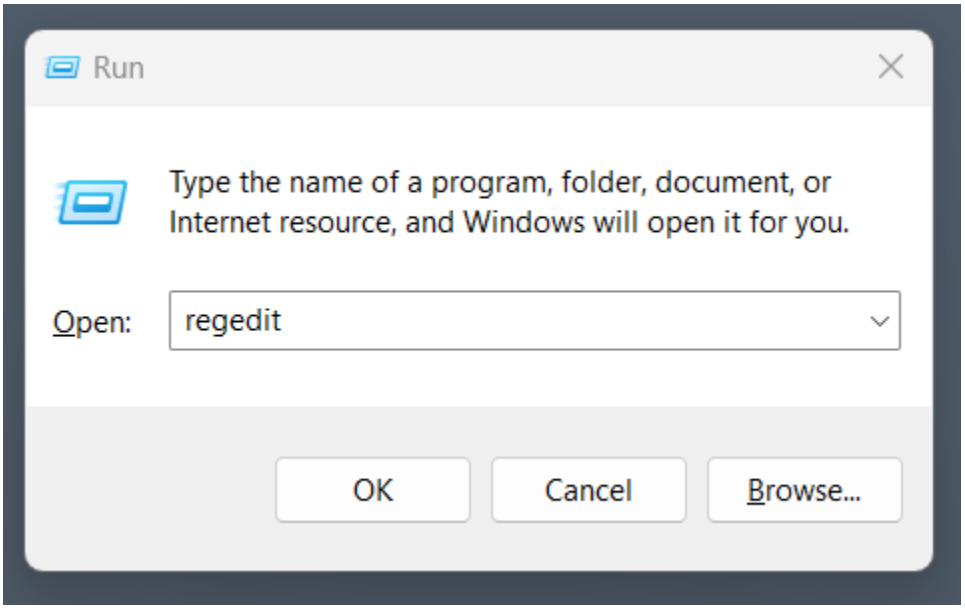
Graphical Windows application

Access the registry

First, ensure that UP4W has run at least once after installation. This ensures that the key and values necessary for configuration will be set up in the registry.

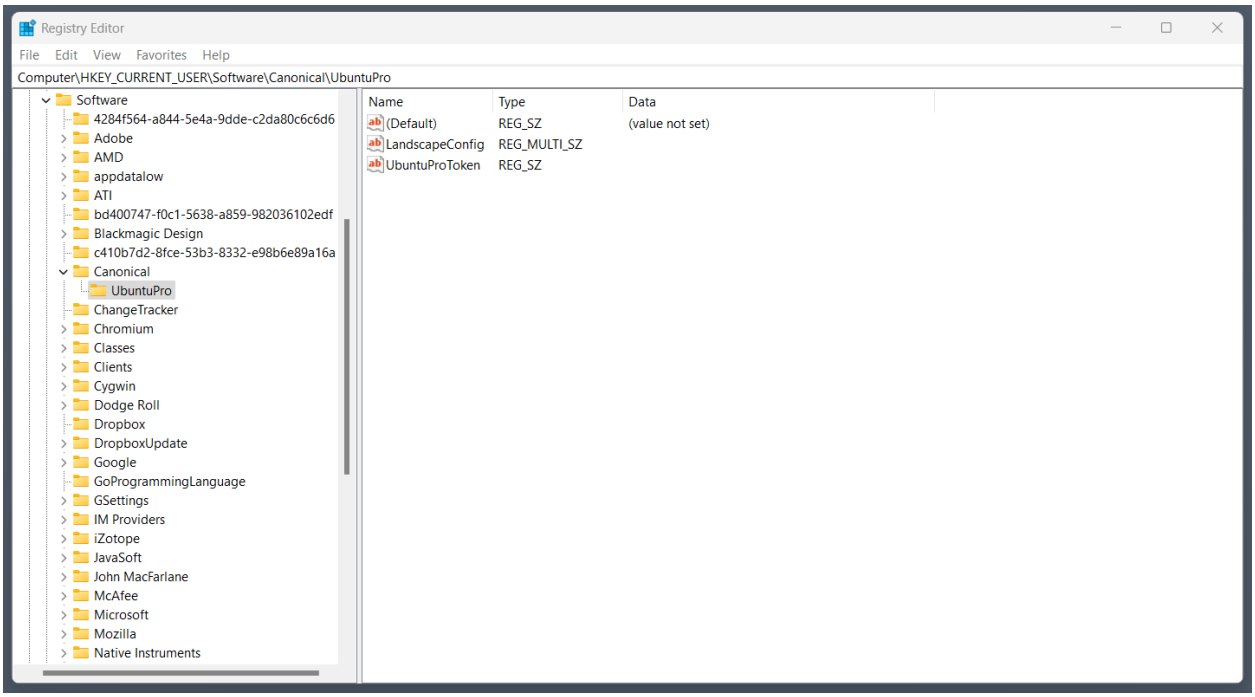
Advanced users of the registry can find relevant information in the Microsoft documentation about alternative methods for modifying the registry data.

To open the registry type `Win+R` and enter `regedit`:

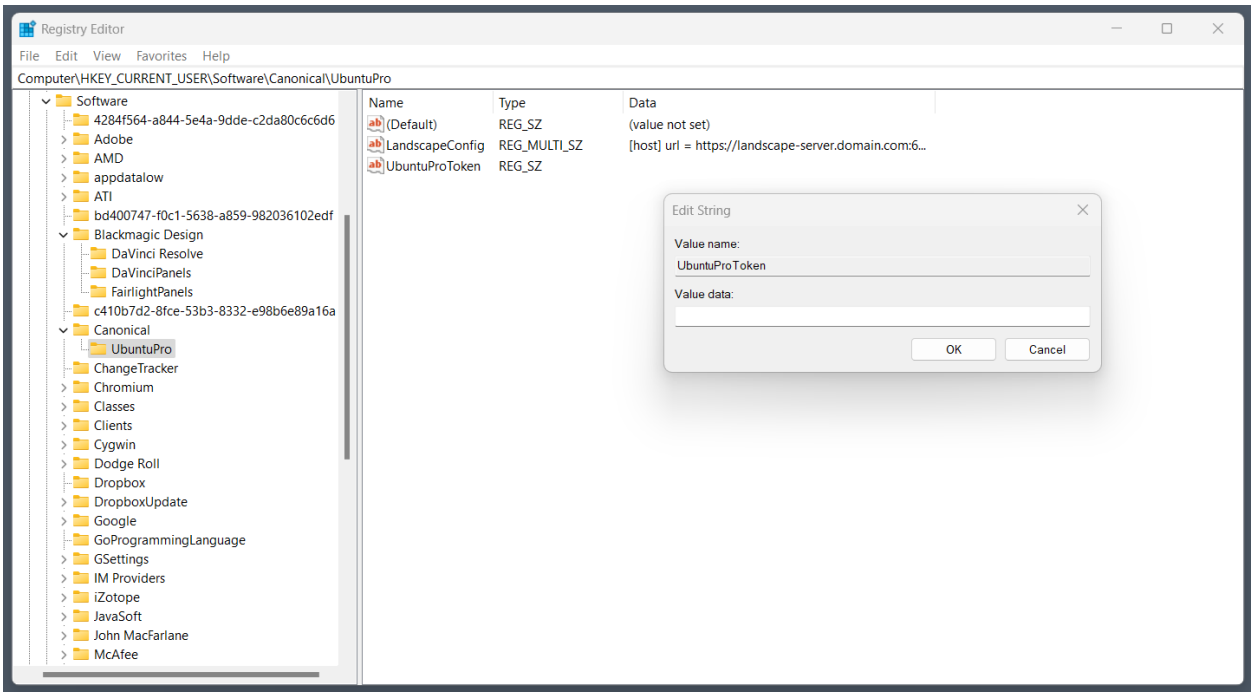


Add Pro token in the registry

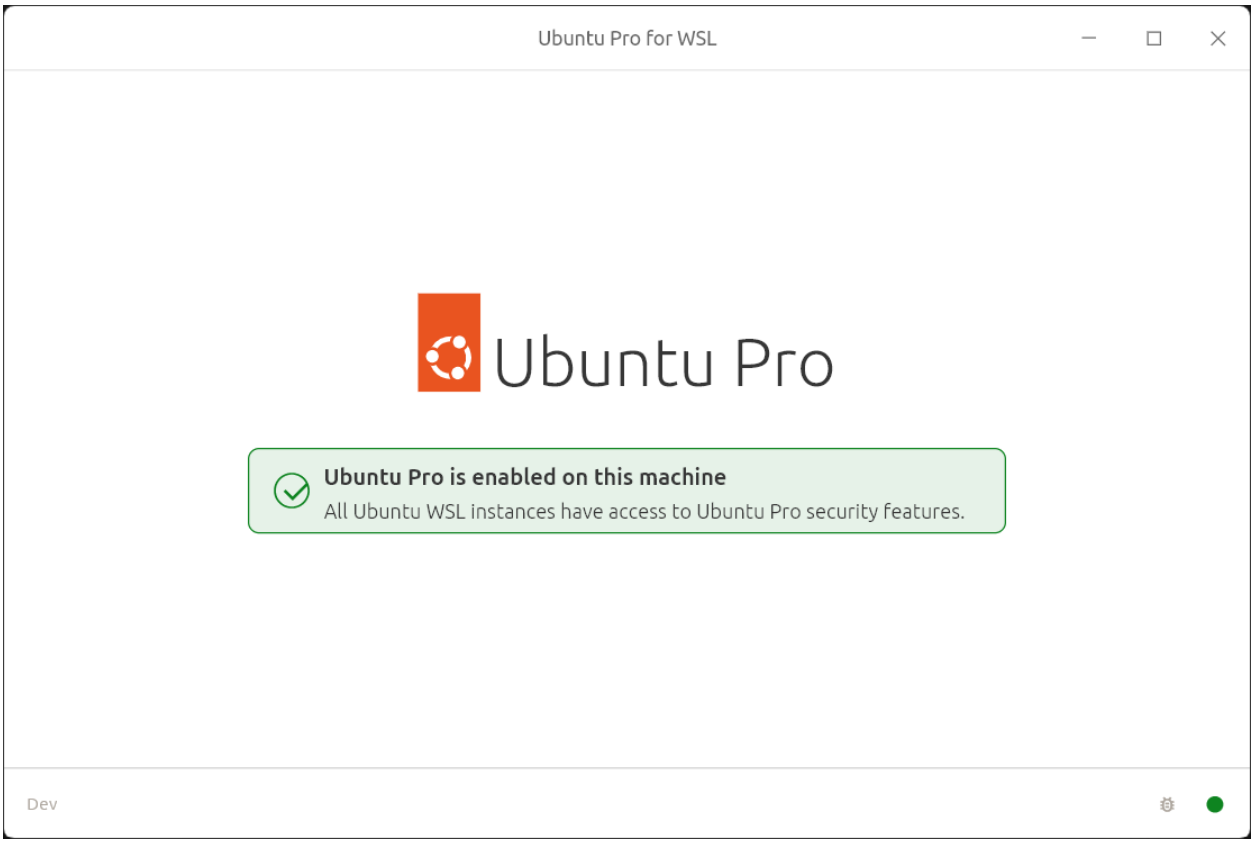
Navigate to `HKEY_CURRENT_USER\Software\Canonical\UbuntuPro`:



Input your Ubuntu Pro token in `UbuntuProToken > Modify > Write the Ubuntu Pro token`:

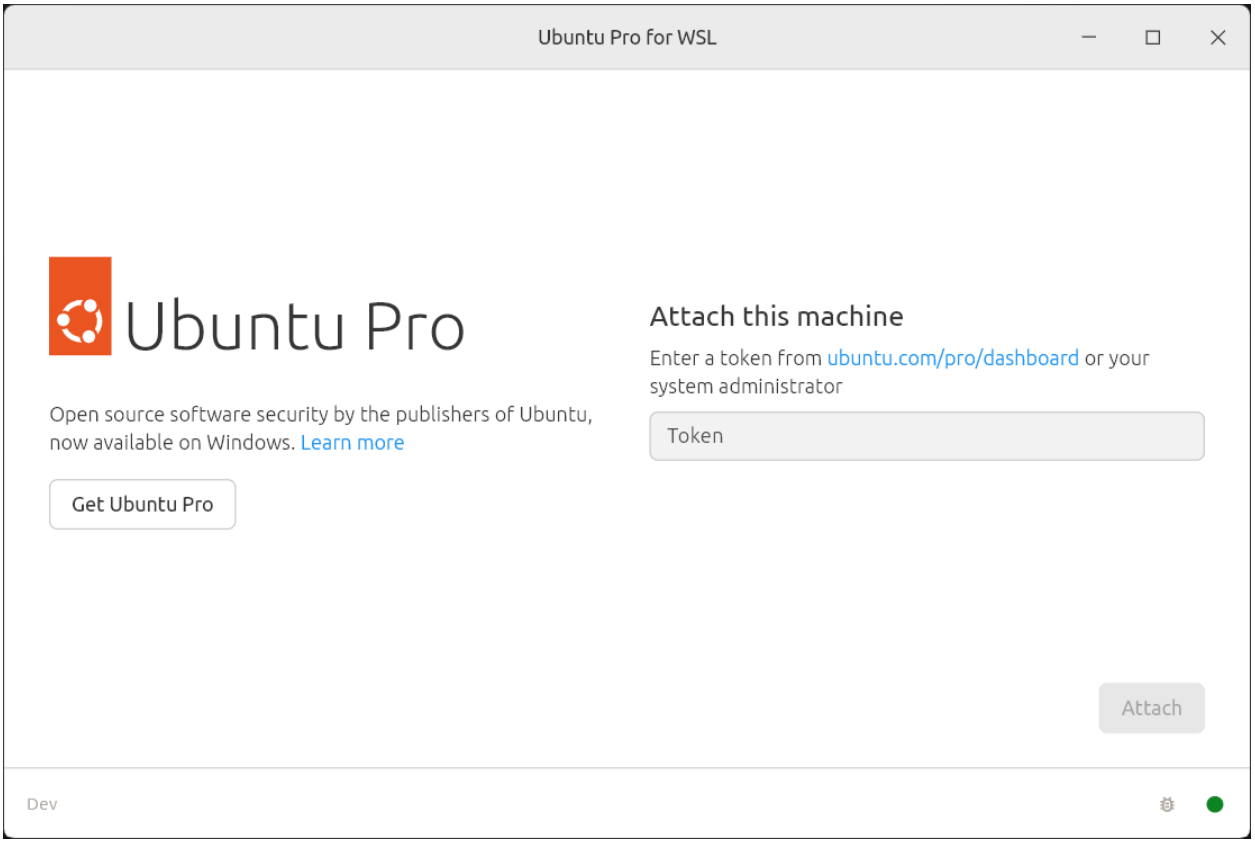


After configuration using the Windows Registry the status in the UP4W Windows application will show that the Pro subscription is active and managed by the user's organisation. Unlike installation through the graphical Windows application, there is no option to detach the Pro subscription in the application interface when the registry is used:



Enter your Pro token

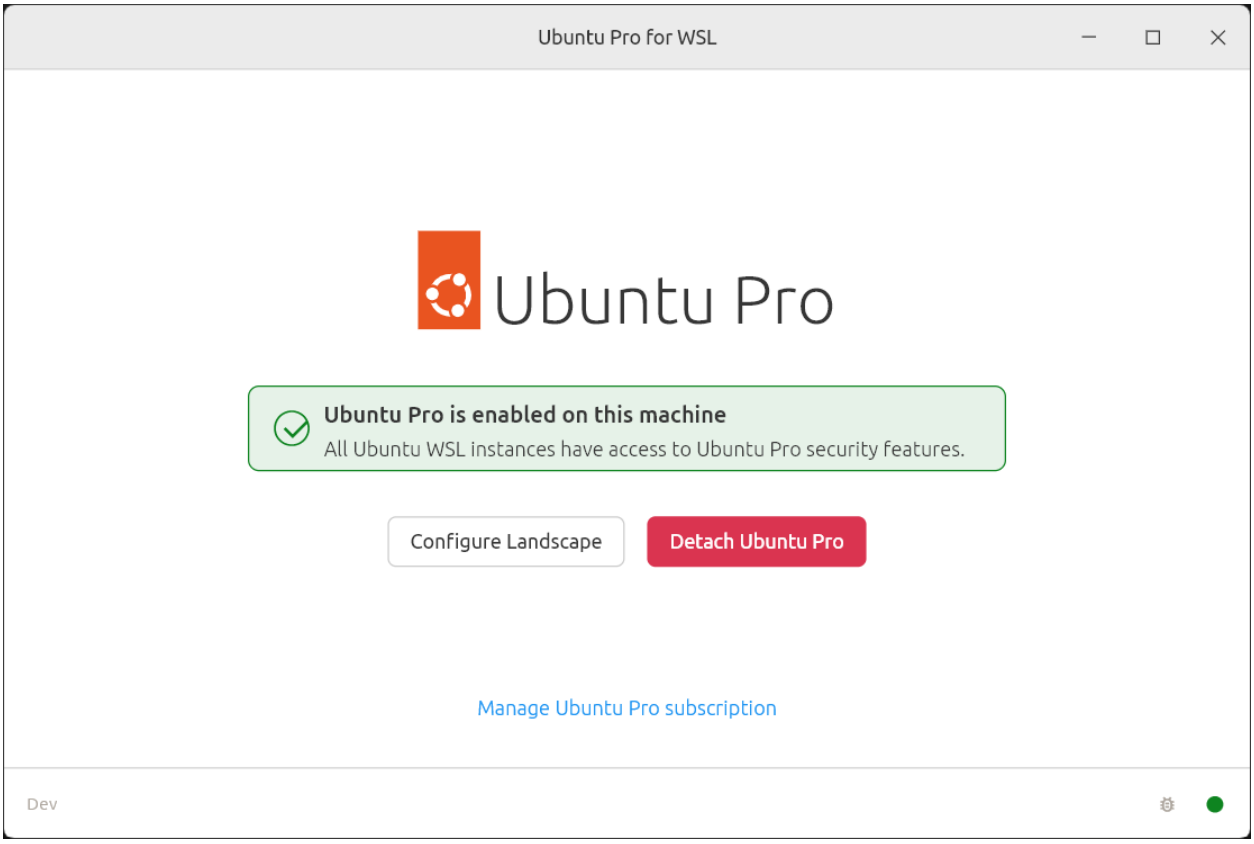
Enter your Ubuntu Pro token in the space provided:



Continue to the confirmation screen.

Confirm subscription is active

You should now see that your Pro subscription is active:



Opening the application again at any point will show this screen, confirm the subscription is active and enable detaching of the subscription.

For additional verification steps refer to *our guide*.

Verify active Pro subscription and Pro attachment

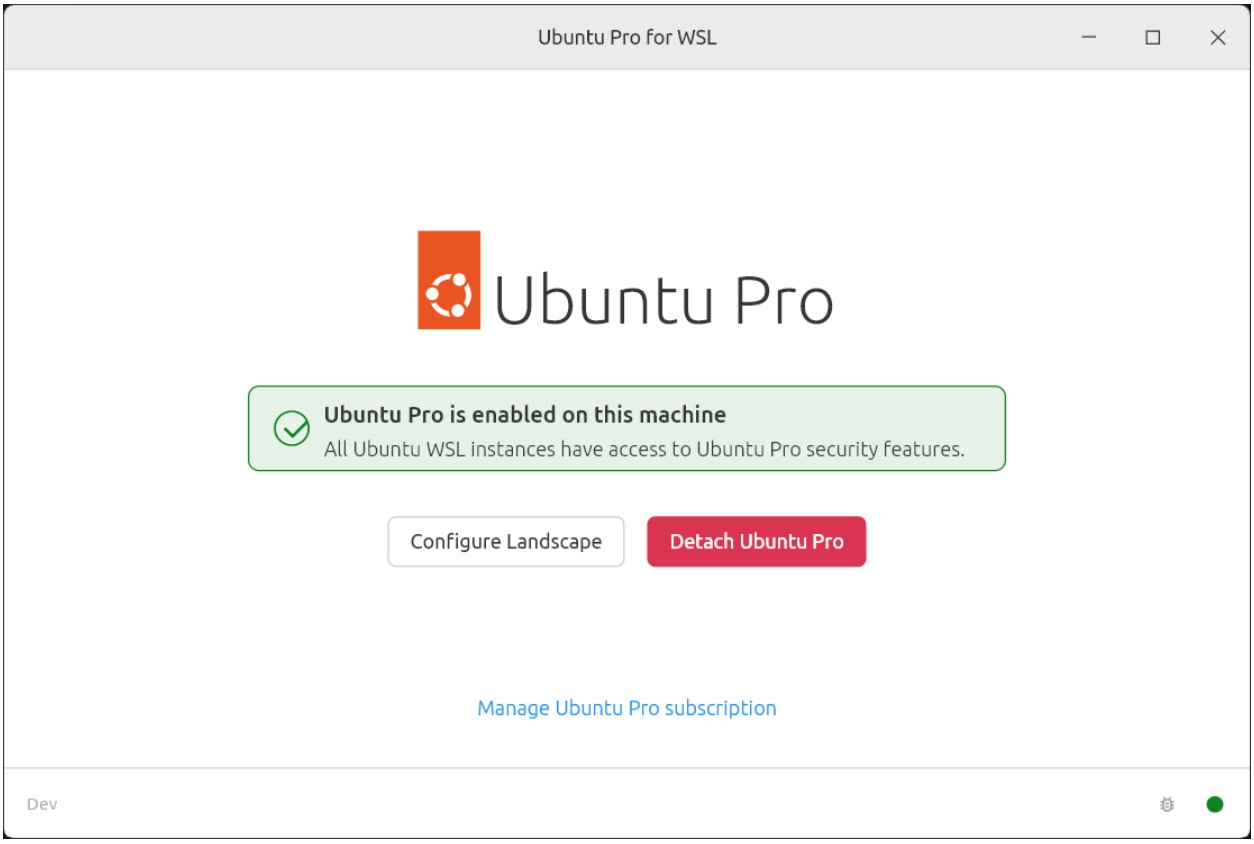
Pro feature

This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

If you have just installed and configured UP4W and a verification step is failing, wait for a few seconds and try again. The process should not take longer than a minute.

Pro subscription

After installing UP4W on a Windows machine and entering your token you should see a confirmation that your Pro subscription is active:



Find and run *Ubuntu Pro for WSL* from the Windows start menu at any time and the app will confirm whether you are subscribed.

Pro-attachment

> **Note**
>
> To verify Pro-attachment WSL should be installed on the Windows machine along with an Ubuntu distro — Ubuntu 24.04 LTS will be used in this example.

To verify Pro-attachment a new Ubuntu instance needs to be created. Running the following command in PowerShell will create a new Ubuntu-24.04 instance and prompt you to create a username and password for the machine:

```
PS C:\Users\me\up4wInstall> ubuntu2404.exe
```

You will now be logged in to the new instance shell and can check that UP4W has Pro-attached this instance with:

```
u@mib:~$ pro status
```

The output will confirm the following:

- Services like ESM are enabled
- Account and subscription information for Ubuntu Pro
- Verification of Pro-attachment

```
SERVICE          ENTITLED  STATUS       DESCRIPTION
esm-apps         yes       enabled      Expanded Security Maintenance for Applications
esm-infra        yes       enabled      Expanded Security Maintenance for Infrastructure

NOTICES
Operation in progress: pro attach

For a list of all Ubuntu Pro services, run 'pro status --all'
Enable services with: pro enable <service>

     Account: me@ubuntu.com
Subscription: Ubuntu Pro - free personal subscription
```

Each new Ubuntu WSL instance that is created should automatically now be Pro-attached.

To find other useful Ubuntu pro commands run:

```
u@mib:~$ pro status
```

Landscape

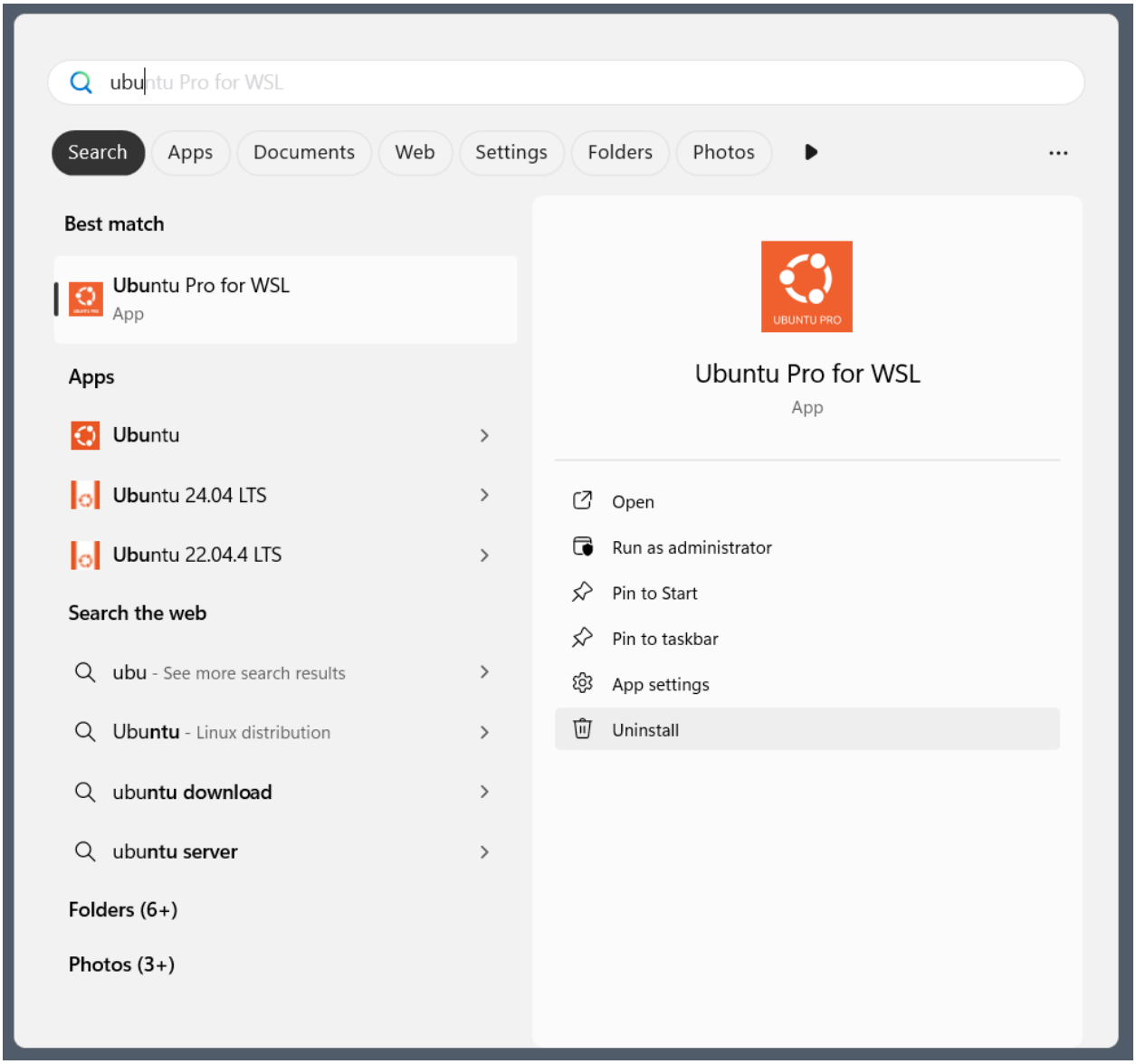
For verification and troubleshooting of Landscape server and client configuration please refer to Landscape | View WSL host machines and child computers.

Uninstall UP4W, Ubuntu WSL and WSL

Uninstalling UP4W, Ubuntu WSL apps and WSL generally only requires finding the relevant application in the Windows Start Menu and clicking **Uninstall**, although in some cases a few additional steps are required.

UP4W

In the Windows Start Menu, locate the "Ubuntu Pro for WSL" application and right-click on it, then click **Uninstall**.



You should also remove the `.ubuntupro` directory from your Windows user profile directory.

```
PS C:\Users\me> Remove-Item -Recurse -Force C:\Users\me\.ubuntupro
```

Ubuntu WSL apps

In PowerShell run the following command to stop WSL:

```
PS C:\Users\me> wsl --shutdown
```

Then, in the Windows Start Menu, locate the "Ubuntu 24.04 LTS" application, right-click on it, and select "Uninstall". The instances will be removed automatically.

WSL app

Only do this if you no longer need WSL on your Windows machine.

In the Windows Start Menu locate the “WSL” application, right-click on it then select “Uninstall”.

2.2.2 Remote deployment

Use the Ubuntu Pro for WSL application to remotely manage Ubuntu on WSL.

Remote deployment

These guides help you deploy and manage WSL instances using Ubuntu Pro for WSL with remote management tools.

Landscape

Manage Ubuntu on WSL with Canonical’s Landscape.

Configure the Landscape client with UP4W

> **Pro feature**
>
> This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

Choose a configuration method

The Landscape client can be configured in two ways:

- Windows registry: easier to automate and deploy at scale
- Graphical Windows application: convenient option for individual users

Click the appropriate tab to read more.

Windows registry

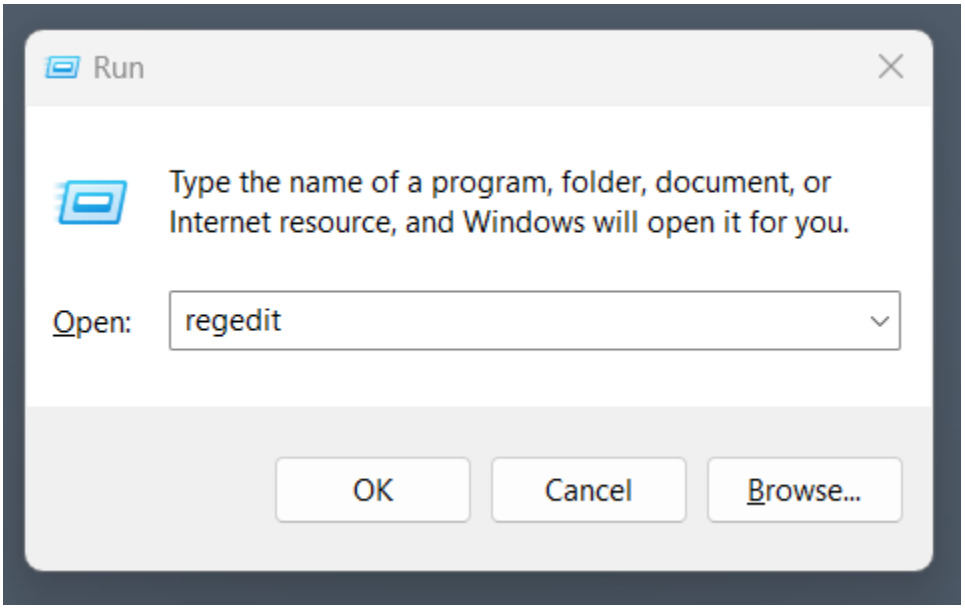
Graphical Windows application

Access the registry

First, ensure that UP4W has run at least once after installation. This ensures that the key and values necessary for configuration will be set up in the registry.

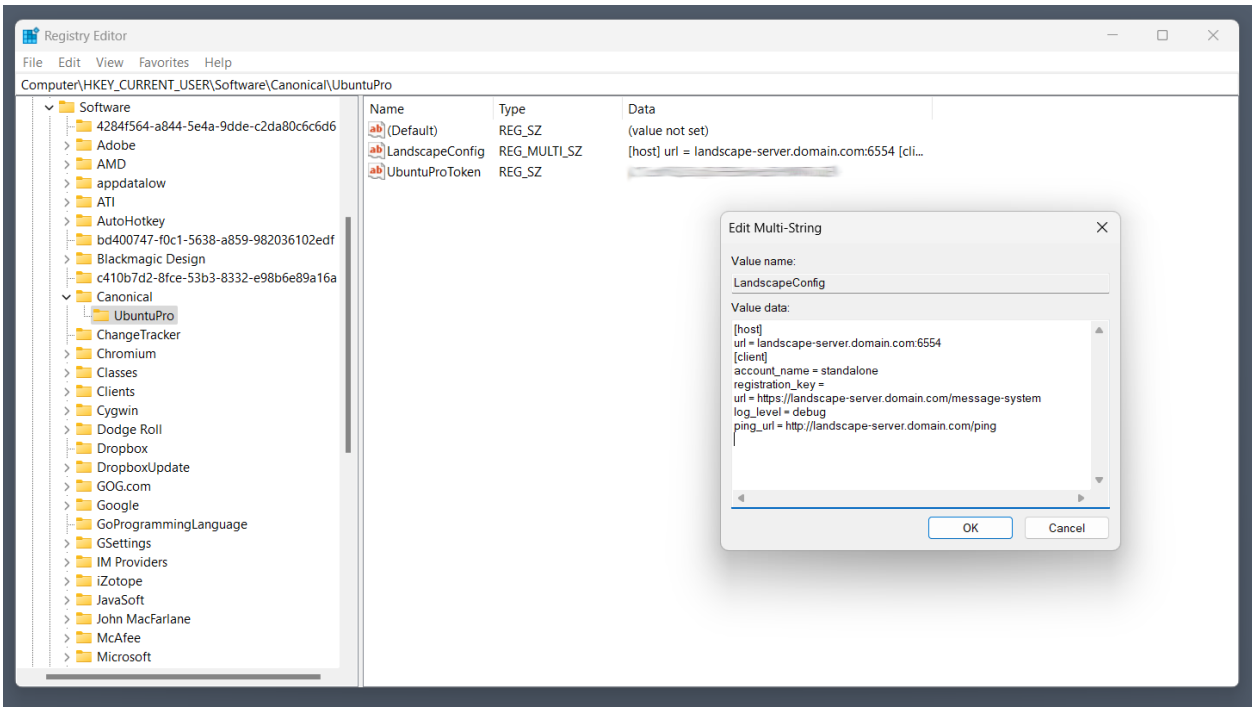
Advanced users of the registry can find relevant information in the Microsoft documentation about alternative methods for modifying the registry data.

To open the registry type `Win+R` and enter `regedit`:



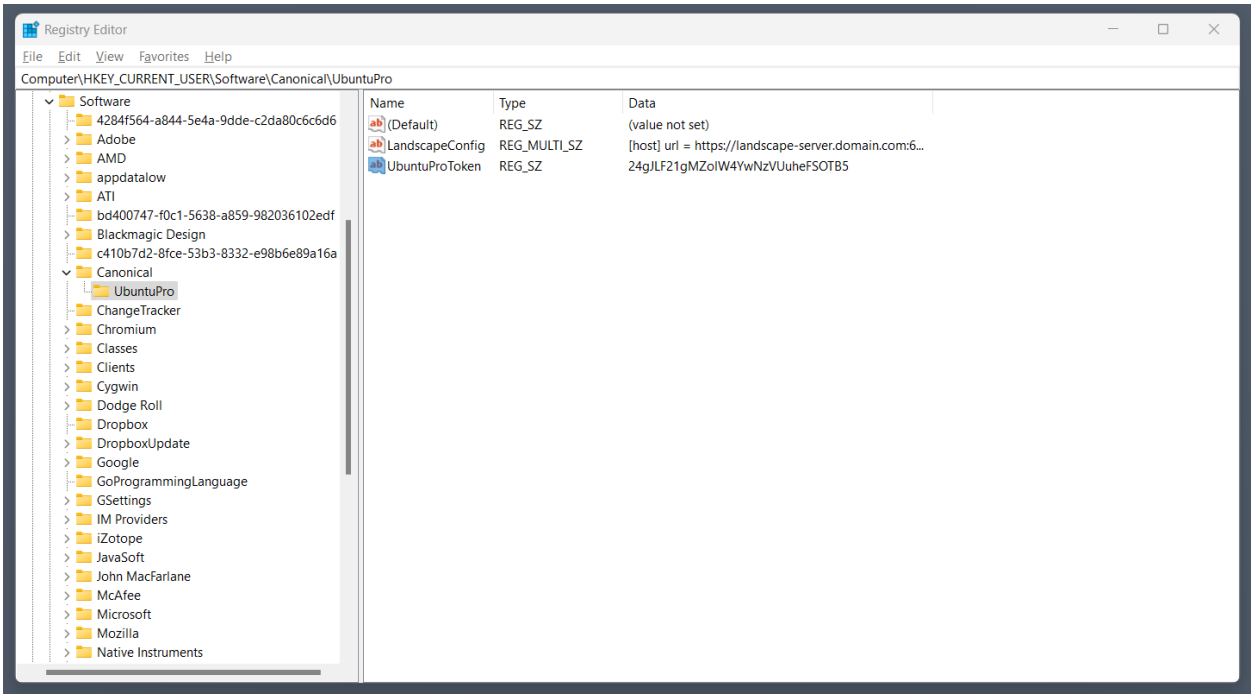
Configure Landscape in the registry

If you are using Landscape you can input your configuration in `LandscapeConfig` > `Modify` > `Write the Landscape config`:

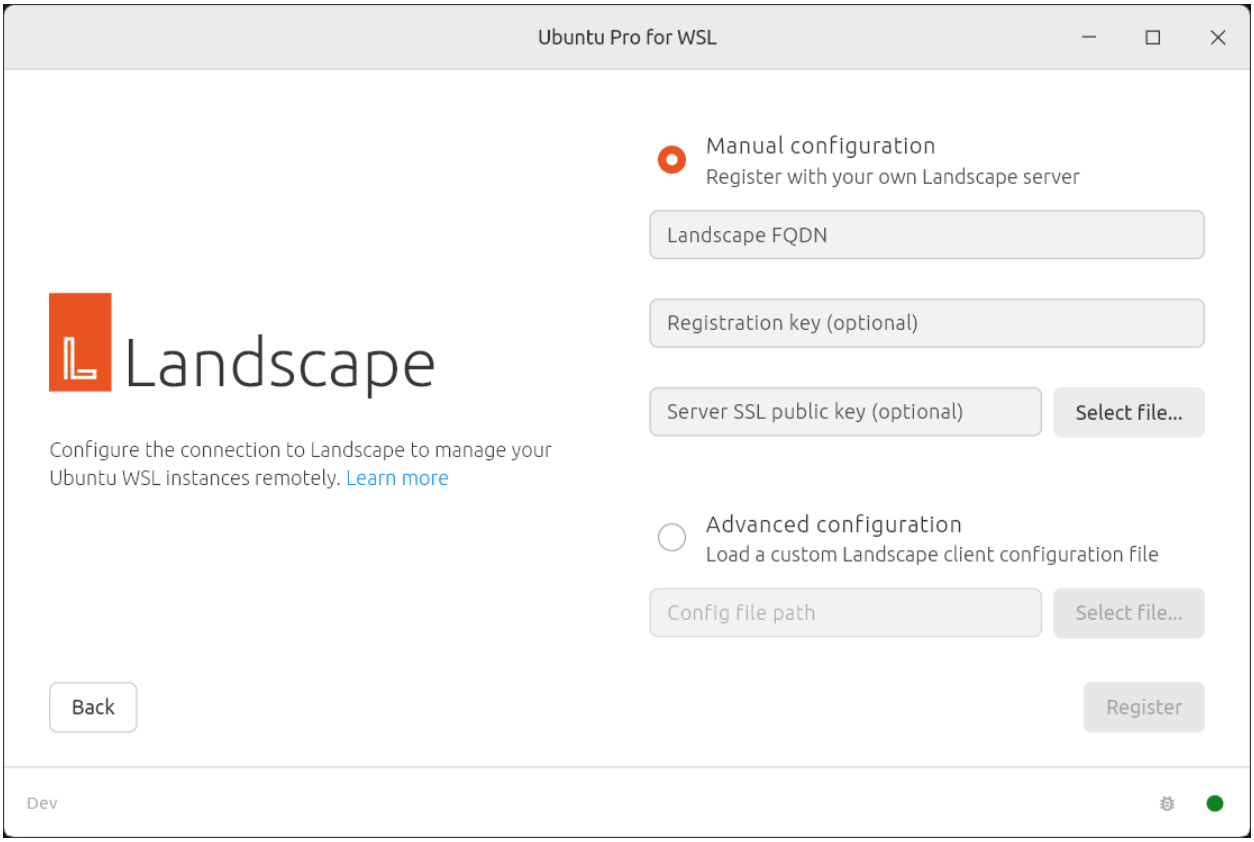


Refer to the section on *Landscape client configuration* for an example.

After you have populated the configuration with data you should be ready to create and manage automatically Pro-attaching WSL instances through Landscape:



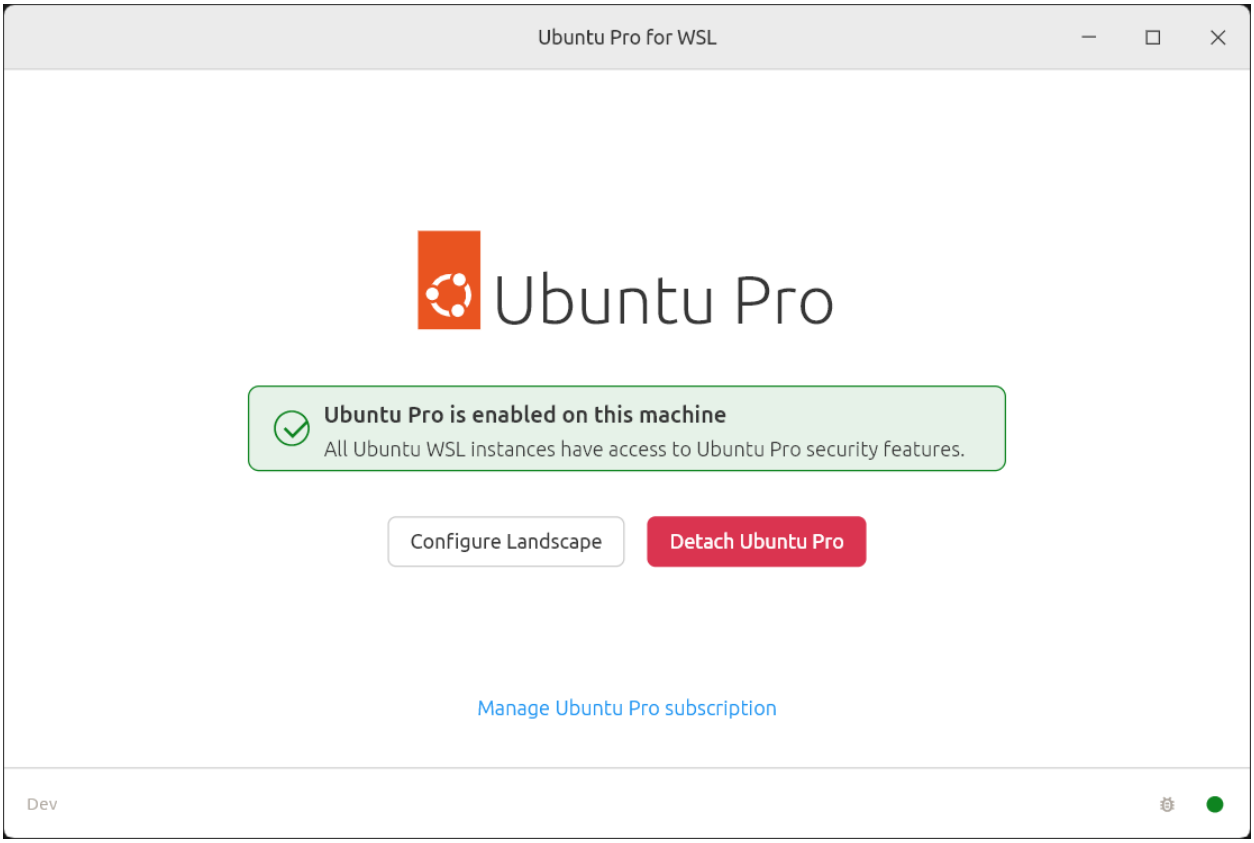
In the UP4W app navigate to the Landscape configuration screen:



Choose your preferred configuration option and enter the required details.

The “Advanced Configuration” option requires you to specify a `landscape.conf`. Refer to the section on *Landscape client configuration* for an example.

When you continue a status screen will appear confirming that configuration is complete:



Warning

Until version 0.1.15 of Ubuntu Pro for WSL, the app explicitly requires referencing a path to the SSL certificate on a Windows host machine. Newer versions completely follow the Windows OS certificate stores, only requiring reference to that certificate if the machine running the Landscape server is not trusted on your network.

For example, if you followed the Landscape Quickstart installation, the auto-generated self-signed certificate can be found at `/etc/ssl/certs/landscape_server.pem`.

This can be copied to a Windows machine:

C:\Users\<YOUR_WINDOWS_USER_NAME>\landscape_server.pem

The path can then be referenced during Landscape configuration in the UP4W Windows app.

Configuring the landscape client

Both the `LandscapeConfig` data in the Windows registry and the Advanced Configuration option in the graphical Windows application can be configured as follows:

```
[host]
url = landscape-server.domain.com:6554

[client]
url = https://landscape-server.domain.com/message-system
```

(continues on next page)

(continued from previous page)

```
ping_url  = http://landscape-server.domain.com/ping
account_name = standalone
log_level = debug
ssl_public_key = C:\Users\user\Downloads\landscape_server.pem
```

> **Warning**
>
> The `ping_url` must be a `http` address. A `https` address will not work.

A more comprehensive example of the configuration options is provided here.

How-to deploy a custom rootfs across multiple Windows machines with the Landscape API

> **Pro feature**
>
> This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

This guide shows how to use the Landscape API to automate the deployment of a custom rootfs across multiple Windows machines. Scaled deployment is enabled by Ubuntu Pro for WSL, which ensures that Ubuntu WSL instances on Windows machines are automatically registered with Landscape. Cloud-init is used for initialisation and final configuration of the instances. To follow the steps outlined in this guide you can use either:

- Bash scripting on Linux, or
- PowerShell scripting on Windows

Prerequisites

- A running self-hosted Landscape server version `24.10~beta.5` or later.
- Multiple Windows machines *already registered with Landscape* via Ubuntu Pro for WSL.
- Make sure you have installed `curl` and `jq`, if you're following this guide using Bash.
- Familiarity with Bash and/or PowerShell.

Prepare the environment

For convenience when writing subsequent commands, first export the following environment variables, modifying the values that are assigned as needed:

Bash

PowerShell

```
# Credentials to authenticate the API requests
export LANDSCAPE_USER_EMAIL=admin@mib.com
export LANDSCAPE_USER_PASSWORD=mib
export LANDSCAPE_URL=https://landscape.mib.com

# The URL of the custom rootfs to be deployed
export ROOTFS_URL="http://landscape.mib.com:9009/ubuntu-24.04-custom.tar.gz"
```

(continues on next page)

(continued from previous page)

```
# The list of IDs of the different Windows machines on which we are going to deploy WSL␣
↪instances
export PARENT_COMPUTER_IDS=(26 30 31)

# The name of the WSL instance to be created
export COMPUTER_NAME=Carbonizer

# Path to the cloud-config file whose contents will be used to initialize the WSL␣
↪instances
export CLOUD_INIT_FILE="~/Downloads/init.yaml"
```

```
# Credentials to authenticate the API requests
$LANDSCAPE_USER_EMAIL="admin@mib.com"
$LANDSCAPE_USER_PASSWORD="mib"
$LANDSCAPE_URL="https://landscape.mib.com"

# The URL of the custom rootfs to be deployed
$ROOTFS_URL="http://landscape.mib.com:9009/ubuntu-24.04-custom.tar.gz"

# The list of IDs of the different Windows machines on which we are going to deploy WSL␣
↪instances
$PARENT_COMPUTER_IDS=@(26, 30, 31)

# The name of the WSL instance to be created
$COMPUTER_NAME="Carbonizer"

# Path to the cloud-config file whose contents will be used to initialize the WSL␣
↪instances
$CLOUD_INIT_FILE="~\Downloads\init.yaml"
```

Generate a Base64-encoded string with the cloud-config data:

Bash

PowerShell

```
BASE64_ENCODED_CLOUD_INIT=$(cat $CLOUD_INIT_FILE | base64 --wrap=0)
```

```
$content = Get-Content -Path $CLOUD_INIT_FILE -Raw
$bytes = [System.Text.Encoding]::UTF8.GetBytes($content)
$BASE64_ENCODED_CLOUD_INIT = [System.Convert]::ToBase64String($bytes)
```

Authenticate against the Landscape API

Build the authentication payload of the form: `{"email": "admin@mib.com", "password": "mib"}` using the values exported in prior steps:

Bash

PowerShell

```
LOGIN_JSON=$( jq -n \
    --arg em "$LANDSCAPE_USER_EMAIL" \
    --arg pwd "$LANDSCAPE_USER_PASSWORD" \
    '{email: $em, password: $pwd}' )
```

```
$LOGIN_JSON = @{
 email = "$LANDSCAPE_USER_EMAIL"
 password = "$LANDSCAPE_USER_PASSWORD"
} | ConvertTo-Json
```

Issue an authenticate request and retrieve the JSON web token (JWT) to be used in the subsequent API requests.

Bash

PowerShell

```
LOGIN_RESPONSE=$( curl -s -X POST "$LANDSCAPE_URL/api/v2/login" \
    --data "$LOGIN_JSON"                                        \
    --header "Content-Type: application/json"                   \
    --header "Accept: application/json" )

JWT=$( echo $LOGIN_RESPONSE | jq .token | tr -d '"')
```

```
$LOGIN_RESPONSE = Invoke-WebRequest -Method POST `
    -URI "$LANDSCAPE_URL/api/v2/login"           `
    -Body "$LOGIN_JSON" -ContentType "application/json"

$JWT = ConvertTo-SecureString -AsPlainText -Force $( $LOGIN_RESPONSE.Content |␣
↪ConvertFrom-Json).token
```

Send the Install request

Build the payload with information about the WSL instance to be deployed. In this case it would look like:

```
{"rootfs_url": "http://landscape.mib.com:9009/ubuntu-24.04-custom.tar.gz", "computer_name
↪": "Carbonizer", "cloud_init": "<base64 encoded material>"}
```

Bash

PowerShell

```
WSL_JSON=$( jq -n                           \
    --arg rf "$ROOTFS_URL"                  \
    --arg cn "$COMPUTER_NAME"               \
    --arg b64 "$BASE64_ENCODED_CLOUD_INIT"  \
    '{rootfs_url: $rf, computer_name: $cn, cloud_init: $b64}' )
```

```
$WSL_JSON = @{
 rootfs_url = "$ROOTFS_URL"
 computer_name = "$COMPUTER_NAME"
 cloud_init = "$BASE64_ENCODED_CLOUD_INIT"
} | ConvertTo-Json
```

At the moment of this writing there is no specific API endpoint to trigger installation of WSL instances on multiple Windows machines at once. Instead we send one request per target machine.

Bash

PowerShell

```
for COMPUTER_ID in "${PARENT_COMPUTER_IDS[@]}"; do
    API_RESPONSE=$( curl -s -X POST                                    \
        "$LANDSCAPE_URL/api/v2/computers/$COMPUTER_ID/children" \
        --data "$WSL_JSON"                                             \
        --header "Authorization:Bearer $JWT"                           \
        --header "Content-Type: application/json"                      \
        --header "Accept: application/json" )

    # show the response
    echo $API_RESPONSE
    echo
done
```

```
foreach ($COMPUTER_ID in $PARENT_COMPUTER_IDS) {
    $API_RESPONSE = Invoke-WebRequest -Method POST -Body "$WSL_JSON" `
        -Uri "$LANDSCAPE_URL/api/v2/computers/$COMPUTER_ID/children" `
        -Authentication Bearer -Token $JWT -ContentType "application/json"

    # show the response
    Write-Output $API_RESPONSE
}
```

When that completes, you'll be able to find activities in the Landscape dashboard about the installation of a new WSL instance for each of the Windows machines listed.

Summarising the steps in a single script

The steps above can be made into a single script:

Bash

PowerShell

```
#!/usr/bin/env bash

# Base64-encoding the cloud-config file contents
BASE64_ENCODED_CLOUD_INIT=$(cat $CLOUD_INIT_FILE | base64 --wrap=0)

# Build the auth payload
LOGIN_JSON=$( jq -n                          \
    --arg em "$LANDSCAPE_USER_EMAIL"     \
    --arg pwd "$LANDSCAPE_USER_PASSWORD" \
    '{email: $em, password: $pwd}' )

# Issue an auth request and retrieve the JWT
LOGIN_RESPONSE=$( curl -s -X POST "$LANDSCAPE_URL/api/v2/login" \
    --data "$LOGIN_JSON"                                        \
    --header "Content-Type: application/json"                   \
```

(continues on next page)

(continued from previous page)

```
    --header "Accept: application/json" )

JWT=$( echo $LOGIN_RESPONSE | jq .token | tr -d '"')

# Build the installation payload
WSL_JSON=$( jq -n                           \
    --arg rf "$ROOTFS_URL"                  \
    --arg cn "$COMPUTER_NAME"               \
    --arg b64 "$BASE64_ENCODED_CLOUD_INIT"  \
    '{rootfs_url: $rf, computer_name: $cn, cloud_init: $b64}' )

# Issue the command for each Windows machine
for COMPUTER_ID in "${PARENT_COMPUTER_IDS[@]}"; do
    API_RESPONSE=$( curl -s -X POST                                 \
        "$LANDSCAPE_URL/api/v2/computers/$COMPUTER_ID/children" \
        --data "$WSL_JSON"                                          \
        --header "Authorization:Bearer $JWT"                        \
        --header "Content-Type: application/json"                   \
        --header "Accept: application/json" )

    # show the response
    echo $API_RESPONSE
    echo
done
```

```
# Base64-encoding the cloud-config file contents
$content = Get-Content -Path $CLOUD_INIT_FILE -Raw
$bytes = [System.Text.Encoding]::UTF8.GetBytes($content)
$BASE64_ENCODED_CLOUD_INIT = [System.Convert]::ToBase64String($bytes)

# Build the auth payload
$LOGIN_JSON = @{
 email = "$LANDSCAPE_USER_EMAIL"
 password = "$LANDSCAPE_USER_PASSWORD"
} | ConvertTo-Json

# Issue an auth request and retrieve the JWT
$LOGIN_RESPONSE = Invoke-WebRequest -Method POST `
    -URI "$LANDSCAPE_URL/api/v2/login"           `
    -Body "$LOGIN_JSON" -ContentType "application/json"

$JWT = ConvertTo-SecureString -AsPlainText -Force $( $LOGIN_RESPONSE.Content |␣
↪ConvertFrom-Json).token

# Build the installation payload
$WSL_JSON = @{
 rootfs_url = "$ROOTFS_URL"
 computer_name = "$COMPUTER_NAME"
 cloud_init = "$BASE64_ENCODED_CLOUD_INIT"
} | ConvertTo-Json

# Issue the command for each Windows machine
```

(continues on next page)

(continued from previous page)

```
foreach ($COMPUTER_ID in $PARENT_COMPUTER_IDS) {
    $API_RESPONSE = Invoke-WebRequest -Method POST -Body "$WSL_JSON" `
        -Uri "$LANDSCAPE_URL/api/v2/computers/$COMPUTER_ID/children" `
        -Authentication Bearer -Token $JWT -ContentType "application/json"

    # show the response
    Write-Output $API_RESPONSE
}
```

Further reading

- Visit the Landscape API documentation to learn more about it.
- Landscape documentation about WSL integration contains more information about this and other methods of creating WSL instances on Windows machines registered with Landscape via its REST API.

Intune

Manage Ubuntu on WSL with Microsoft's Intune.

How to enforce the UP4W background agent startup remotely using the Windows Registry

> **Pro feature**
>
> This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

Ubuntu Pro for WSL, being a Microsoft Store application, cannot ship user services as of the time of writing (late 2024), but can deploy startup tasks instead, programs that run with user permissions when the user logs into the Windows device. The UP4W background agent runs as a startup task, which is only enabled by the operating system when the user interacts with the application for the first time. While this behaviour is a feature for end-users it presents a source of friction for deployments at scale, when system administrators expect zero-touch deployment of UP4W to just work.

This guide shows how sysadmins can use the Windows Registry to enforce the enablement of the UP4W background agent startup task without depending on end-user interaction. While this guide uses Intune, it should be reproducible with any remote management solution capable of deploying MS Store (or MSIX-packaged) applications and registry keys.

Pre-requisites

- At least one managed Windows device.
- A Windows remote management solution.
- If using Intune, an Enterprise E3 or E5 or Education A3 or A5 licenses.

Overview

1. The registry path `"HKCU:\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppModel\SystemAppData\CanonicalGroupLimited.UbuntuPro_79rhkp1fndgsc"` holds configuration information specific about UP4W and is created or overwritten when the MSIX package is installed.

2. A sub-key named `UbuntuProAutoStart` governs the startup task state.
3. Setting the DWORD value named `State` to `4` makes the operating system interpret it as "Enabled by Policy".
4. When the user logs in, Windows executes the UP4W startup task, even if the user has not interacted with the application.
5. A Windows remote management solution can monitor that registry key value and proactively fix it, thus enforcing the UP4W startup task to be always enabled.

Using Intune Remediations

Remediations are script packages that can detect and fix common issues on a user's device before they realise there's a problem. Each script package consists of a detection script, a remediation script, and metadata. Through Intune, you can deploy these script packages and monitor reports of their effectiveness. Visit the Microsoft Intune documentation to learn more. Those scripts run on a predefined schedule and if the detection script reports a failure (by `exit 1`) then the remediation script will run. That allows system administrators to watch for the specific Registry key value that represents the enablement of the UP4W background agent startup task. The contents of both scripts are presented below. **Make sure to save them encoded in UTF-8**, as required by Intune.

Detection script

```
$Path = "HKCU:\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\
↪AppModel\SystemAppData\CanonicalGroupLimited.UbuntuPro_79rhkp1fndgsc\UbuntuProAutoStart
↪"
$Name = "State"
$Value = 4

Try {
    $Registry = Get-ItemProperty -Path $Path -Name $Name -ErrorAction Stop | Select-
↪Object -ExpandProperty $Name
    If ($Registry -eq $Value){
        Write-Output "Compliant"
        Exit 0
    }
    Write-Warning "Not Compliant"
    Exit 1
}
Catch {
    Write-Warning "Not Compliant"
    Exit 1
}
```

Remediation script

```
$Path = "HKCU:\Software\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\
↪AppModel\SystemAppData\CanonicalGroupLimited.UbuntuPro_79rhkp1fndgsc\UbuntuProAutoStart
↪"
$Name = "State"
$KeyFormat = "DWORD"
$value = 4


try{
```

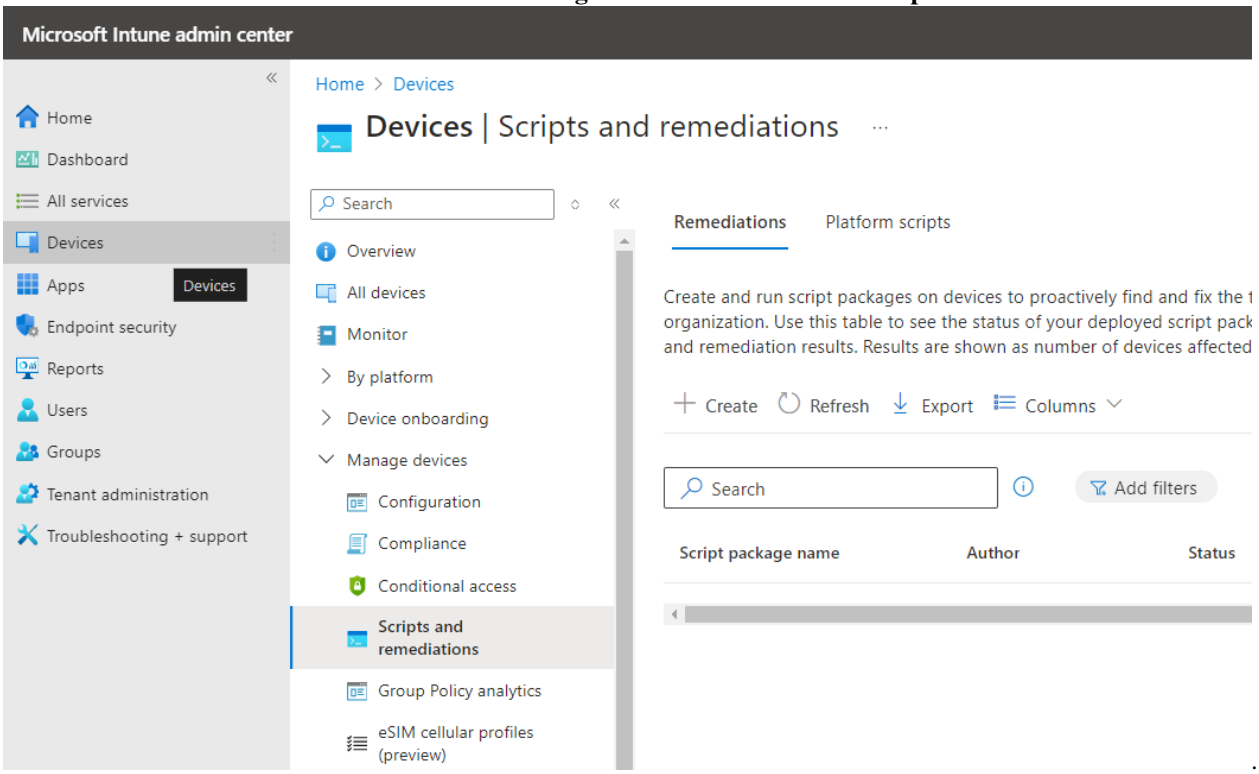
(continues on next page)

(continued from previous page)

```
    if(!(Test-Path $Path)){New-Item -Path $Path -Force}
    if(!$Name){Set-Item -Path $Path -Value $Value}
    else{Set-ItemProperty -Path $Path -Name $Name -Value $Value -Type $KeyFormat}
    Write-Output "Key set: $Name = $Value"
}catch{
    Write-Error $_
}
```

Running the scripts with Intune

Access your organisation's Intune Admin Center and when logged in go to **Devices > Monitor > Manage Devices > Scripts and remediations**.



Click on the "Create" button to create a new script package. Fill in the **Basics** form with name, description and other details and proceed to **Settings**. During that step, upload the scripts in the correct boxes and use the following options:

- Run this script using the logged-on credentials (important because the script refers to a registry path under `HKCU` a.k.a `HKEY_CURRENT_USER`)
- Enforce script signature check: No (unless otherwise required by your company's policies)
- Run script in 64-bit PowerShell: No

Finally, select "Next" and assign "Scope tags" (if relevant) and in the "Assignments" select the device or user groups that are required.

Follow this guide if you need more detail on the steps outlined above.

When users covered by the assignment next log in, Intune executes the detection script and then runs the remediation script if the device is found to be non-compliant.

Remarks

Careful readers may notice that there is an inherent race condition between setting the registry value and installing the MSIX (if remotely deployed): when the MSIX is installed the registry sub-key that is referenced is recreated, overwriting any previous value that the remote management solution would have deployed if that happened before the package installation.

An advantage of Intune Remediation scripts in this scenario is that eventually Intune finds the non-compliant state and fixes it automatically. A potential disadvantage is that the fix doesn't start the UP4W background agent, The fix enables the startup task and the agent will start at next user login.

How to start the agent remotely

> **Pro feature**
>
> This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

Ubuntu Pro for WSL, being a Microsoft Store application, cannot ship user services as of the time of writing (late 2024), but can deploy startup tasks instead, programs that run with user permissions when the user logs into the Windows device. The UP4W background agent runs as a startup task, which is only enabled by the operating system when the user interacts with the application for the first time. While this behaviour is a feature for end-users it presents a source of friction for deployments at scale, when system administrators expect zero-touch deployment of UP4W to just work.

This guide shows how system administrators can leverage Windows remote management solutions to start the UP4W background agent once with user credentials, thus interacting with the application on the user's behalf. Subsequent logins will have the UP4W background agent started automatically as a consequence of the interaction.

While this guide uses Intune, readers can translate the steps for the remote management solution of their choice, as long as the said solution can run scripts with current user's credentials.

Pre-requisites

- At least one managed Windows device.
- A Windows remote management solution.

Overview

1. Running a script as user to start the UP4W background agent makes it immediately available.
2. Remote management solutions can be used to start the agent.
3. The startup task is then also enabled on the operating system.
4. On subsequent logins, the UP4W background agent starts automatically, as expected.

Using Intune to run the UP4W background agent

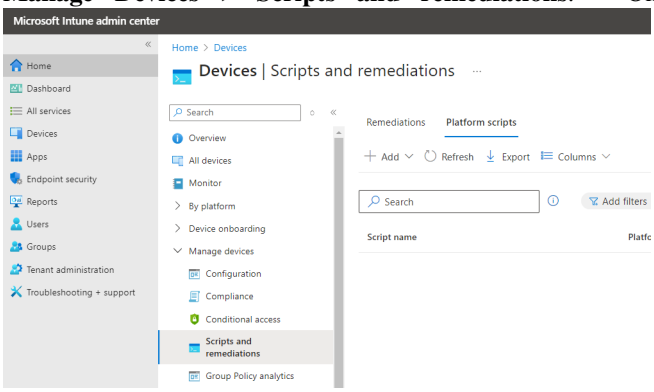
The contents of the script can be far more elaborate, but for the purposes of this guide the following is enough:

```
Write-Output "Starting the UP4W background agent remotely from Intune"
ubuntu-pro-agent.exe
```

Make sure to save that as UTF-8, as required by Intune.

> **Note**
>
> Follow this section from Intune documentation if you need more detailed step-by-step guide on how to create and assign script policies.

Access your organisation's Intune Admin Center and when logged in go to **Devices > Monitor > Manage Devices > Scripts and remediations**. On that page, click on the **Platform scripts** tab.



Click on the "Add" button to create a new script policy and select the platform **Windows 10 and later**.

Fill in the **Basics** form with a name and description for the script being created.

In the **Settings** tab, browse your machine to the PowerShell script to be deployed, and select the following options:

- Run this script using the logged on credentials: Yes (the default). UP4W must run with user credentials.
- Enforce script signature check: No (unless required otherwise by your company's policies).
- Run script in 64-bit PowerShell host: No (the default).

Apply the "Scope tags" according to your company's practices and, in "Assignments" make sure to select one or more groups encompassing the users that must receive and run the script.

You can then monitor the execution of this script in the Intune Admin Center.

When the selected users log in to their devices, Intune will eventually start the UP4W background agent. A terminal window will appear to the user showing its regular outputs. This will only happen once.

Remarks

Careful readers might have noticed that if the script is deployed in conjunction with a policy installing the UP4W, it would be theoretically possible to have the script running before the application gets installed. A more elaborate solution is required if that scenario is possible, like looping in the script or using a more proactive solution such as Intune Remediations.

This guide shows how to use that tool to enforce the UP4W startup task state.

2.2.3 GPU and graphics

Leverage GPU and graphical support with Ubuntu on WSL.

GPU and graphics

These how-to guides explain how to use GPU acceleration and graphical apps with Ubuntu on WSL.

GPU acceleration

If you are working on ML or AI, you can enable GPU acceleration for your projects:

Enabling GPU acceleration with the NVIDIA CUDA Platform

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.

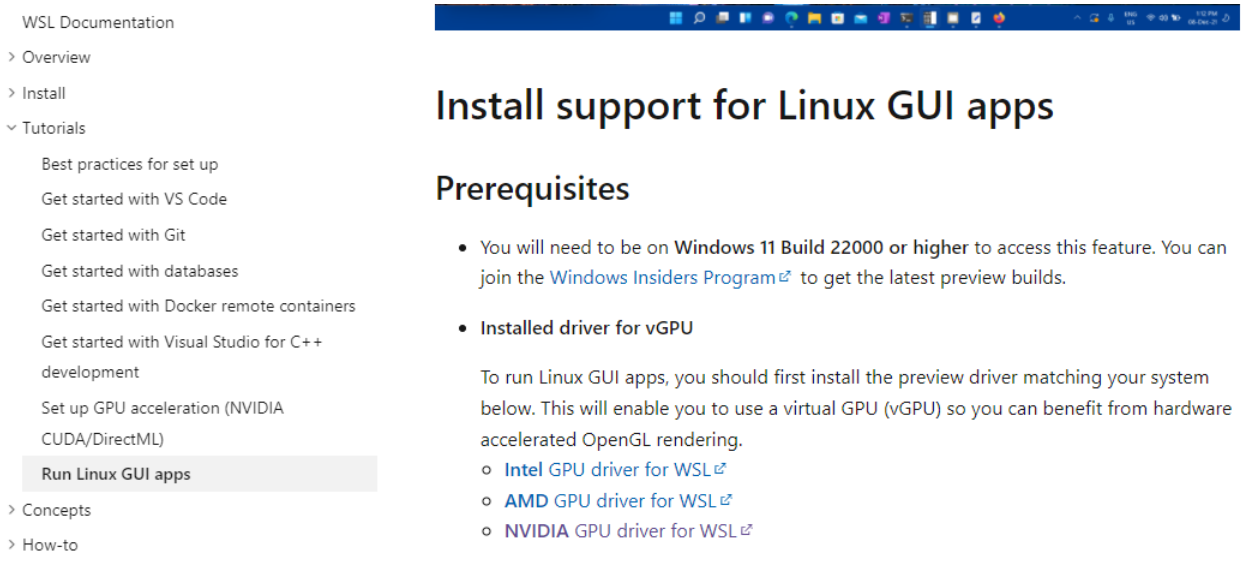
Make sure the following prerequisites are met before moving forward:

- 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

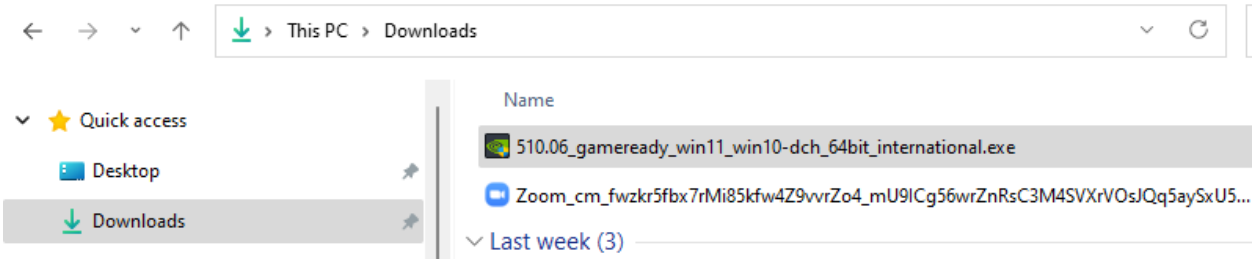
Specific drivers are needed to enable use of a virtual GPU, which is how Ubuntu applications are able to access your GPU hardware, so you'll need to follow this step even if your system drivers are up-to-date.

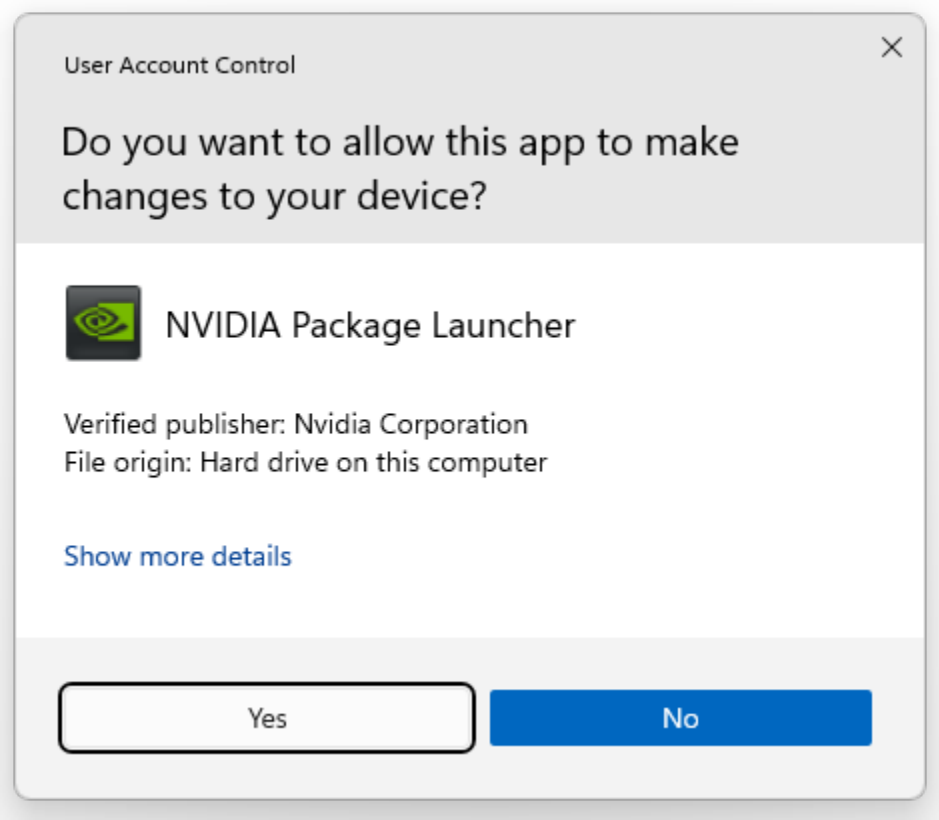
Please refer to the official WSL documentation for up-to-date links matching your specific GPU vendor. You can find these in `Install support for Linux GUI apps > Prerequisites`. For this example, we will download the `NVIDIA GPU Driver for WSL`.



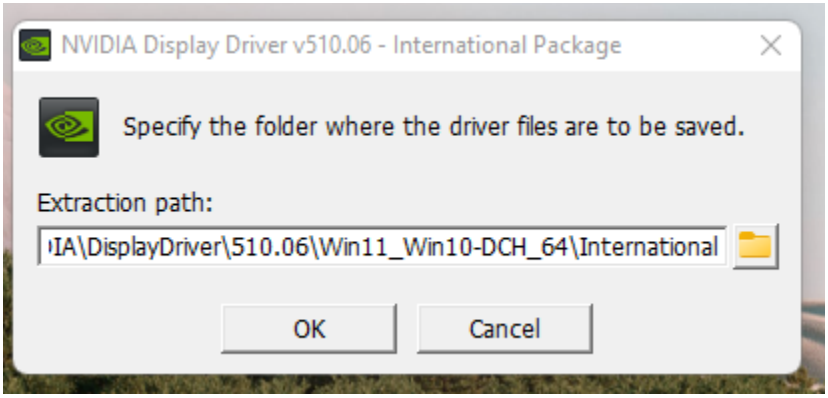
Note: This is the only device driver you'll need to install. Do not install any display driver on Ubuntu.

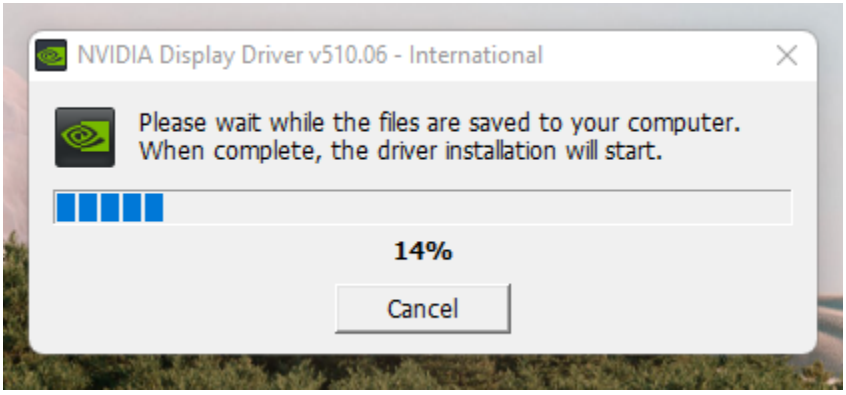
Once downloaded, double-click on the executable file and click `Yes` to allow the program to make changes to your computer.



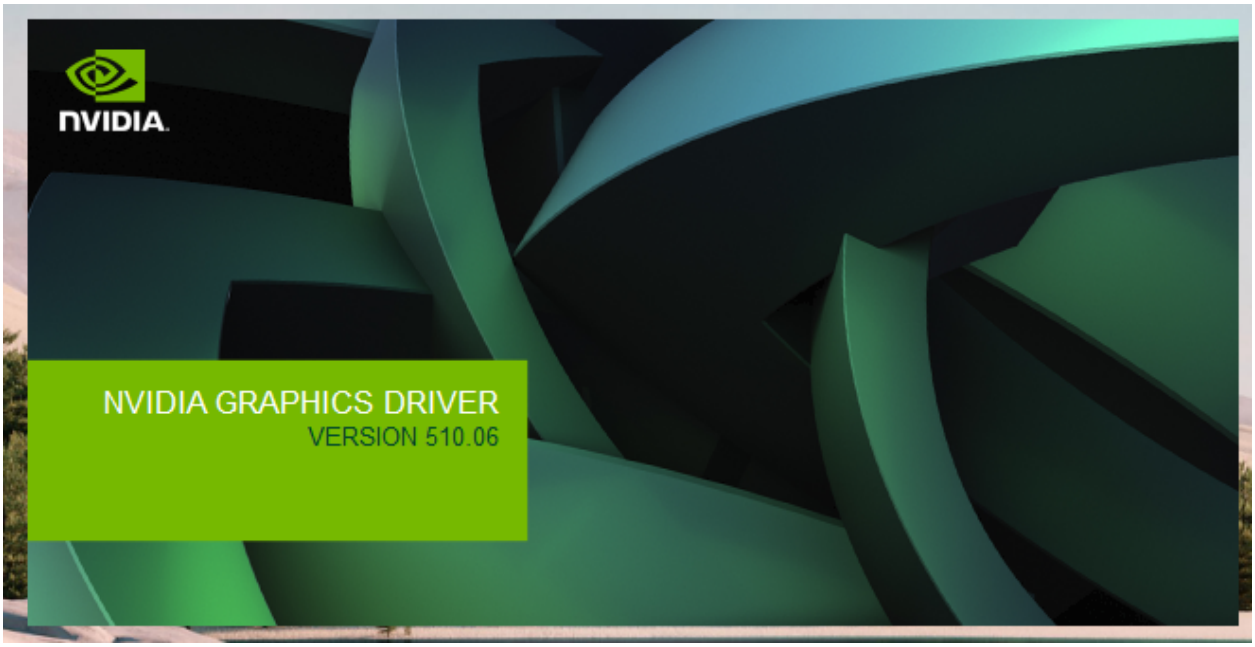


Confirm the default directory and allow the self-extraction process to proceed.



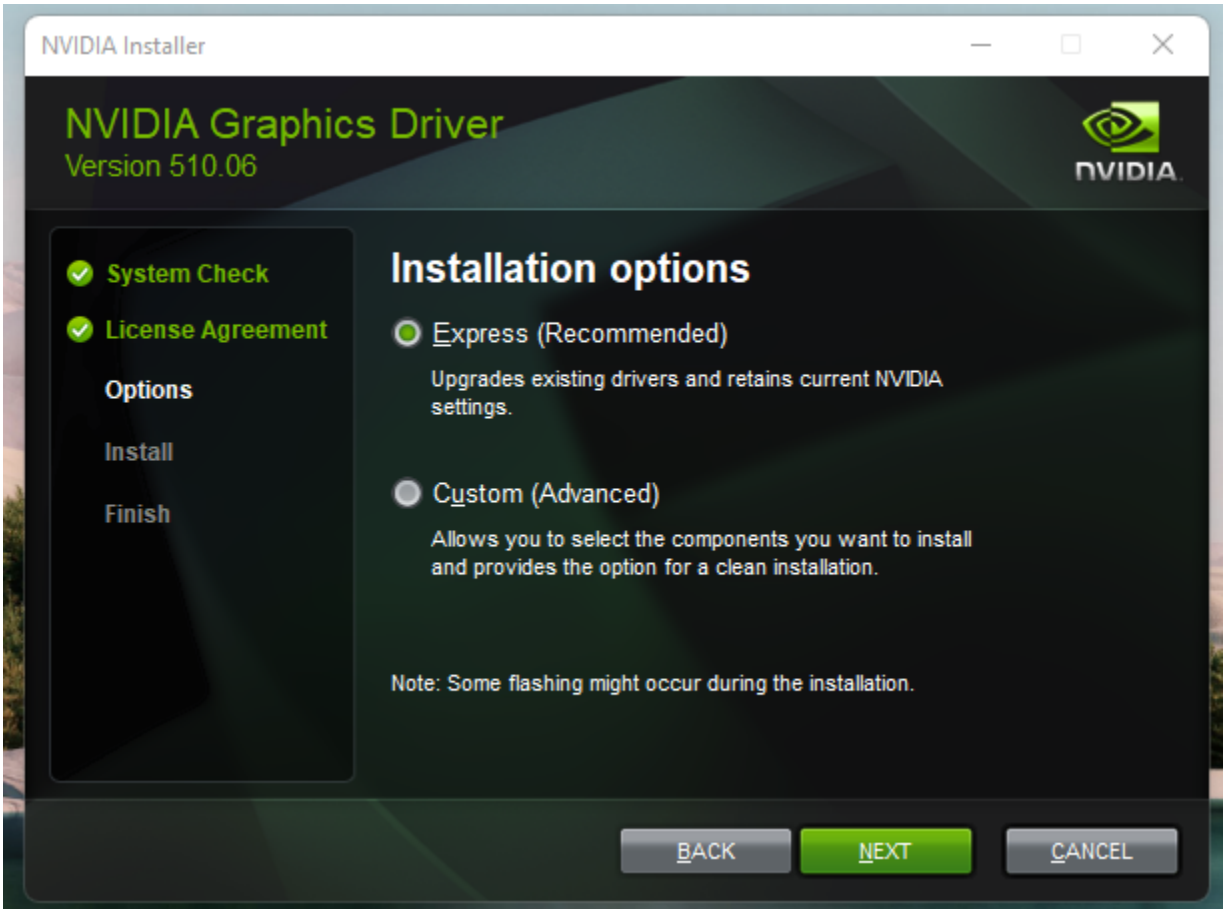


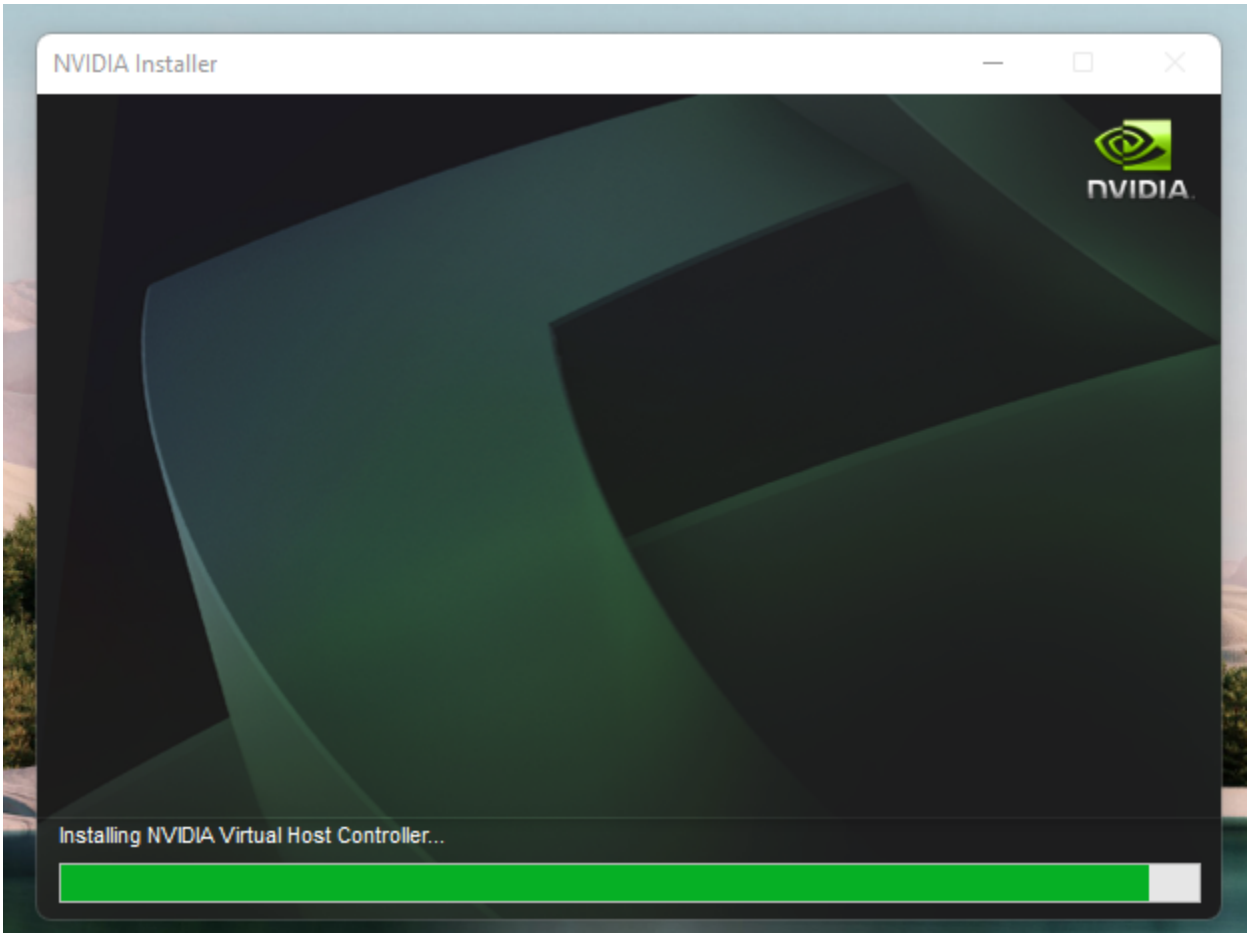
A splash screen appears with the driver version number and quickly turns into the main installer window. Read and accept the license terms to continue.



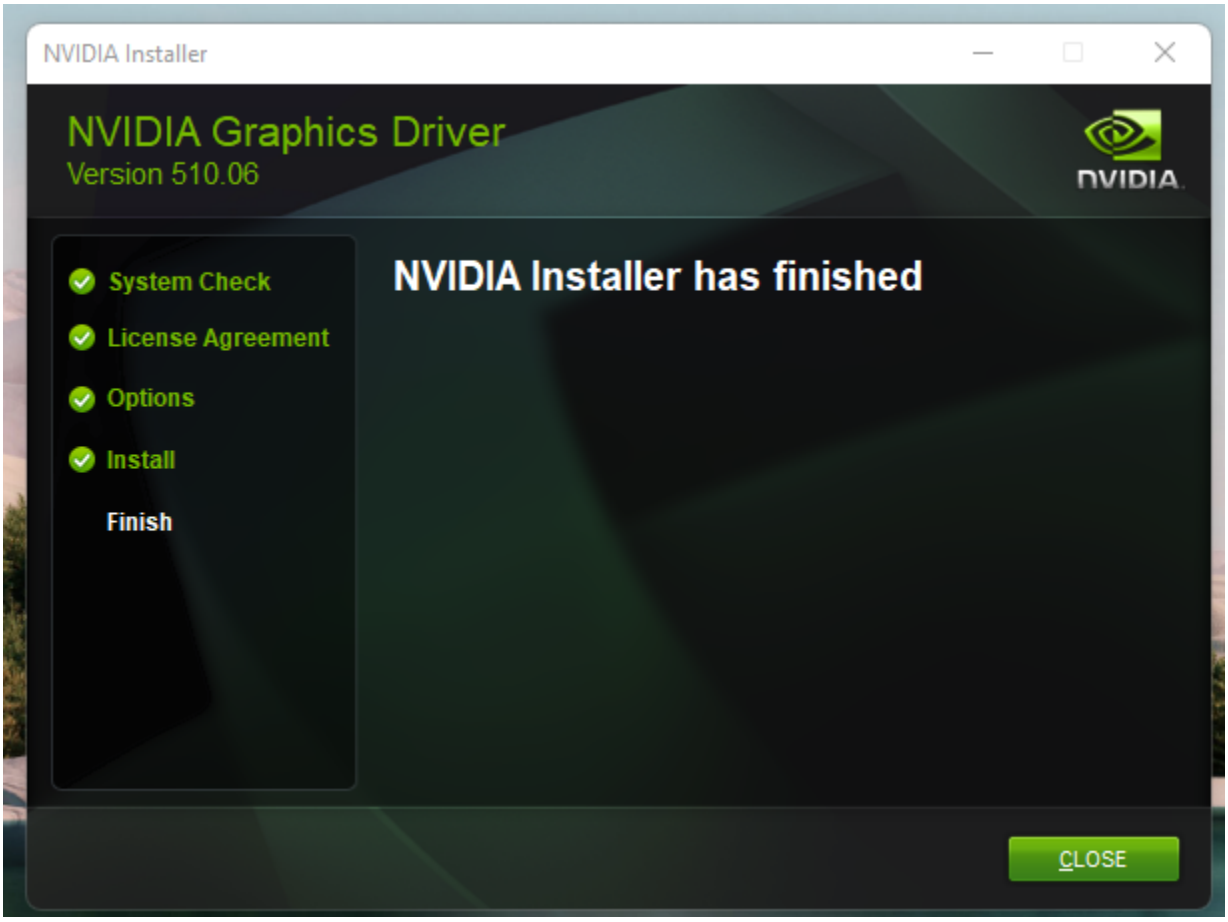
Confirm the wizard defaults by clicking `Next` and wait until the end of the installation. You might be prompted to

restart your computer.





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



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 with the following commands:

```
$ 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/
```

(continues on next page)

(continued from previous page)

```
↪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 run `make`:

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

A successful build will look like the screenshot below.

```
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
```

Once complete, run the application with:

```
$ ./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

Graphical applications

Ubuntu on WSL is a terminal environment but it can also launch Linux-native graphical applications.

Use WSL for data science and engineering

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.

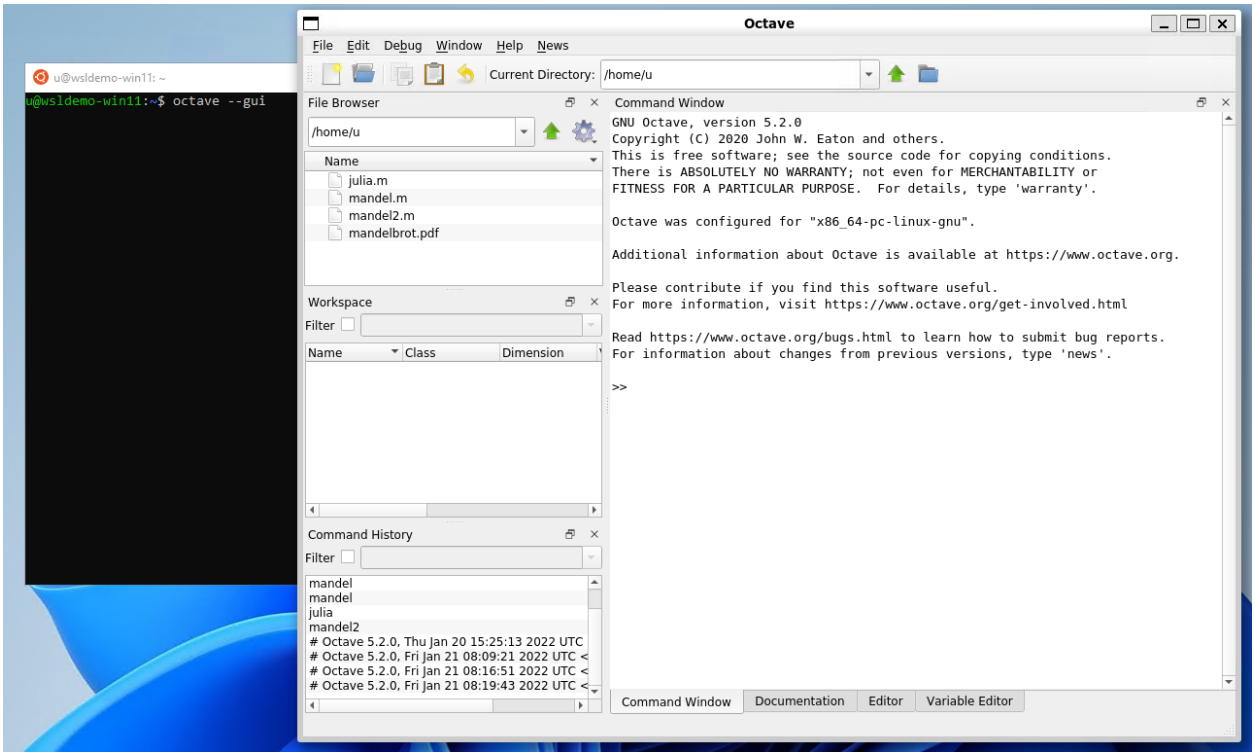
From an Ubuntu WSL terminal prompt run:

```
$ 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.



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.

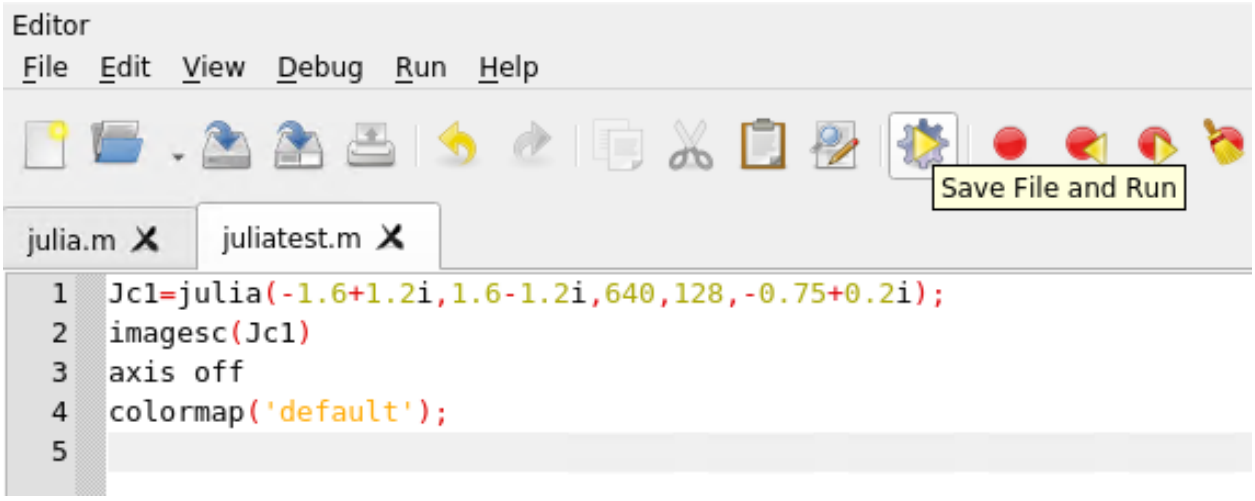
Open a second editor window with the New Script button and copy and paste the following code:

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

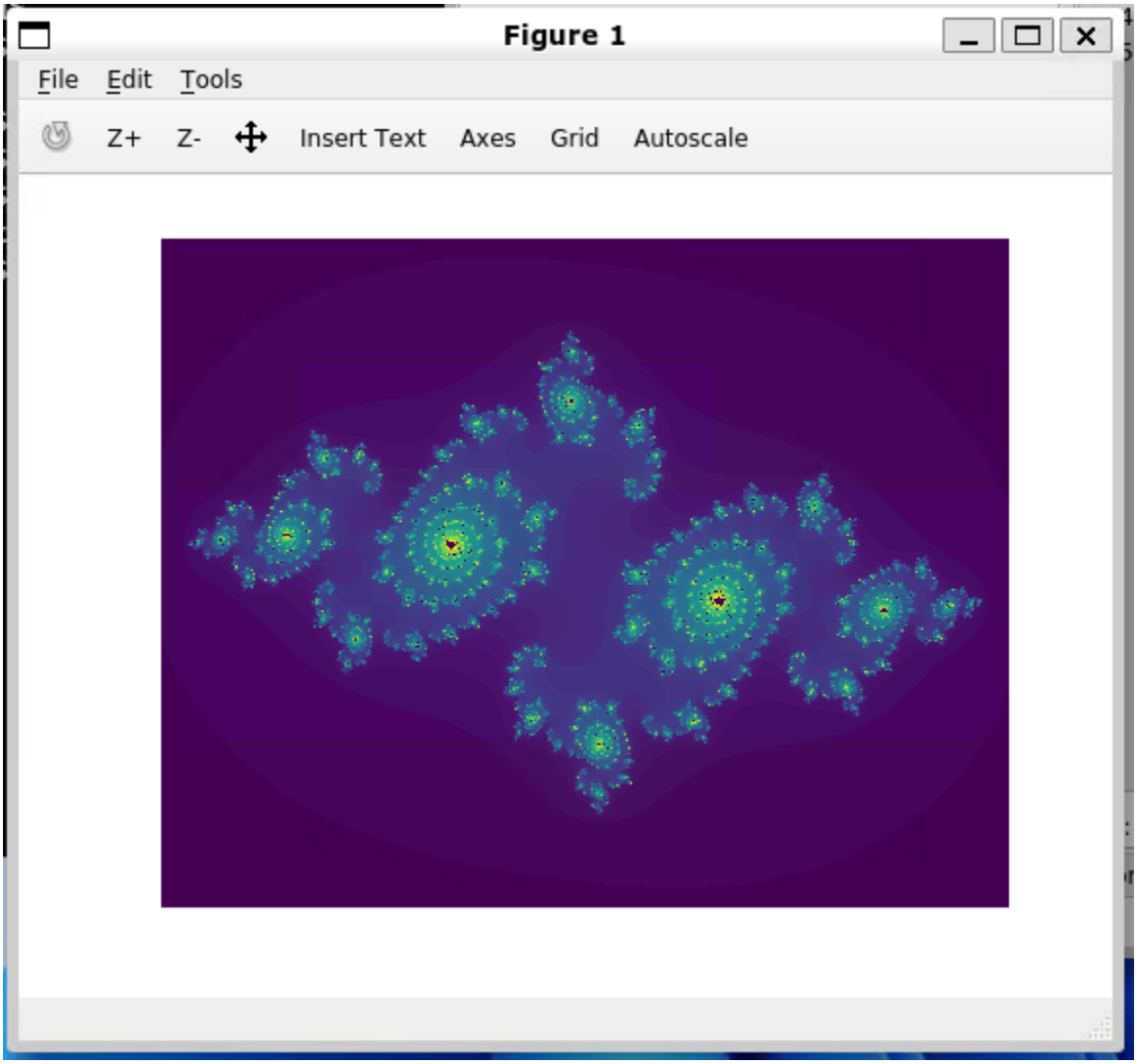
This code calls the function defined in `julia.m`. You can later change the parameters if you want to explore the Julia fractal.

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

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



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

2.2.4 Contributing

Contribute to the development of Ubuntu on WSL.

Contributing

These how-to guides help you contribute to Ubuntu on WSL.

General guidelines for contributors

If you are interested in contributing to code or docs, please read our guidelines.

General contribution guidelines

Ubuntu WSL and Ubuntu Pro for WSL

To ensure that making a contribution is a positive experience for both contributor and reviewer we ask that you read and follow these community guidelines.

This communicates that you respect our time as developers. We will return that respect by addressing your issues, assessing proposed changes and finalising your pull requests, as helpfully and efficiently as possible.

These are mostly guidelines, not rules. Use your best judgement and feel free to propose changes to this document in a pull request.

Code of conduct

We take our community seriously and hold ourselves and other contributors to high standards of communication. By participating and contributing, you agree to uphold the Ubuntu Community Code of Conduct.

Getting started

The source code for Ubuntu WSL and Ubuntu Pro for WSL can be found on GitHub:

- Ubuntu WSL repo
- Ubuntu Pro for WSL repo

Issues with WSL should be directed to Microsoft's WSL project

We accept any contributions relating to Ubuntu WSL and Ubuntu Pro for WSL. However, we do not directly maintain WSL itself, which is a Microsoft product. If you have identified a problem or bug in WSL then file an issue in Microsoft's WSL project repository.

If you are unsure whether your problem relates to an Ubuntu project or the Microsoft project then familiarise yourself with their documentation.

- Ubuntu WSL docs
- Microsoft WSL docs

At this point, if you are still not sure, try to contact a maintainer of one of the projects who will advise you where best to submit your Issue.

How to contribute

Contributions are made via Issues and Pull Requests (PRs). A few general guidelines that cover both:

- Use the advisories page of the repository and not a public bug report to report security vulnerabilities.
- Search for existing Issues and PRs before creating your own.

- Give a friendly ping in the comment thread to the submitter or a contributor to draw attention if your issue is blocking — while we work hard to makes sure issues are handled in a timely manner it can take time to investigate the root cause.
- Read this Ubuntu discourse post for resources and tips on how to get started, especially if you've never contributed before

Issues

Issues should be used to report problems with the software, request a new feature or to discuss potential changes before a PR is created. When you create a new Issue, a template will be loaded that will guide you through collecting and providing the information that we need to investigate.

If you find an Issue that addresses the problem you're having, please add your own reproduction information to the existing issue rather than creating a new one. Adding a reaction can also help be indicating to our maintainers that a particular problem is affecting more than just the reporter.

Pull requests

PRs are always welcome and can be a quick way to get your fix or improvement slated for the next release. In general, PRs should:

- Only fix/add the functionality in question **OR** address wide-spread whitespace/style issues, not both.
- Add unit or integration tests for fixed or changed functionality.
- Address a single concern in the least number of changed lines as possible.
- Include documentation in the repo or on our docs site.
- Use the complete Pull Request template (loaded automatically when a PR is created).

For changes that address core functionality or would require breaking changes (e.g. a major release), it's best to open an Issue to discuss your proposal first. This is not required but can save time creating and reviewing changes.

In general, we follow the "fork-and-pull" Git workflow:

1. Fork the repository to your own GitHub account.
2. Clone the fork to your machine.
3. Create a branch locally with a succinct yet descriptive name.
4. Commit changes to your branch.
5. Follow any formatting and testing guidelines specific to this repo.
6. Push changes to your fork.
7. Open a PR in our repository and follow the PR template so that we can efficiently review the changes.

   PRs will trigger unit and integration tests with and without race detection, linting and formatting validations, static and security checks, freshness of generated files verification. All the tests must pass before anything is merged into the main branch.

Once merged to the main branch, `po` files will be automatically updated and are therefore not necessary to update in the pull request itself, which helps minimise diff review.

Contributing to the code

For testing and debugging Ubuntu WSL or Ubuntu Pro for WSL you will need a Windows Machine with WSL installed.

About the test suite

The source code includes a comprehensive test suite made of unit and integration tests. All the tests must pass with and without the race detector.

Each module has its own package tests and you can also find the integration tests at the appropriate end-to-end (e2e) directory.

The test suite must pass before merging the PR to our main branch. Any new feature, change or fix must be covered by corresponding tests.

Additional dependencies for UP4W

- Ubuntu-24.04
- Visual Studio Community 2019 or above
- Go
- Flutter
- An Ubuntu Pro token

Building and running the binaries for UP4W

For building, you can use the following two scripts:

- Build the Windows Agent
- Build the Wsl Pro Service

Note that you'll need to create a self-signing certificate to build the Windows Agent.

Contributor License Agreement

It is requirement that you sign the Contributor License Agreement in order to contribute. You only need to sign this once and if you have previously signed the agreement when contributing to other Canonical projects you will not need to sign it again.

An automated test is executed on PRs to check if it has been accepted.

Please refer to the licences for Ubuntu WSL and Ubuntu Pro for WSL below:

- Ubuntu WSL.
- Ubuntu Pro for WSL.

Contributing to the docs

The documentation for Ubuntu WSL and Ubuntu Pro for WSL is maintained here.

Our goal is to provide documentation that gives users the information that they need to get what they need from Ubuntu WSL.

You can contribute to the documentation in various different ways. If you are not a developer but want to help make the product better then helping us to improve the documentation is a way to achieve that.

At the top of each page in the documentation, you will find a feedback button. Clicking this button will open an Issue submission page in the Ubuntu WSL GitHub repo. A template will automatically be loaded that you can modify before submitting the Issue.

You can also find a pencil icon for editing the page on GitHub, which will open up the source file in GitHub so that you can make changes before committing them and submitting a PR. This can be a good option if you want to make a small change, e.g., fixing a single typo.

Lastly, at the bottom of the page you will find various links, including a link to the Discourse forum for Ubuntu WSL, where you can ask questions and participate in discussions.

Types of contribution

Some common contributions to documentation are:

- Add or update documentation for new features or feature improvements by submitting a PR
- Add or update documentation that clarifies any doubts you had when working with the product by submitting a PR
- Request a fix to the documentation, by opening an issue on GitHub.
- Post a question or suggestion on the forum.

Automatic documentation checks

Automatic checks will be run on any PR relating to documentation to verify the spelling, the validity of links, correct formatting of the Markdown files and the use of inclusive language.

You should run these tests locally before submitting a PR by running the following commands:

- Check the spelling: `make spelling`
- Check the validity of links: `make linkcheck`
- Check for inclusive language: `make woke`

Doing these checks locally is good practice. You are less likely to run into failed CI checks after your PR is submitted and the reviewer of your PR can more quickly focus on the contribution you have made.

Note on using code blocks

In the Ubuntu WSL docs, code blocks are used to document:

- Ubuntu terminal commands
- PowerShell terminal commands
- Terminal outputs
- Code and config files

We follow specific conventions when including code blocks so that they are readable and functional.

Include prompts when documenting terminal commands

It is common that Ubuntu and PowerShell terminal commands are included in the same page. We use prompts to ensure that the reader can distinguish between them.

Here are some examples:

- PowerShell prompt symbol: >

- PowerShell prompt symbol with path: `C:\Users\myuser>`
- PowerShell prompt symbol with path and PowerShell prefix: `PS C:\Users\myuser>`
- Ubuntu prompt symbol: `$`
- Ubuntu prompt symbol with user and host: `user@host:~$`

Whether to include the path or user@host depends on whether it is useful in the context of the documentation being written. For example, if demonstrating the use of multiple WSL instances, including the user and host can make it easier to tell the instances apart.

Exclude prompts from clipboard when using copy button

The WSL docs automatically strips prompts when a user clicks the **copy** button on a code block. This is to prevent errors when a reader pastes the full content of a copy block into their terminal.

We use a solution based on regular expressions, which identifies the first instance of a prompt symbol followed by whitespace on a particular line before removing the text before that symbol.

There may be edge-cases when this creates problems; for example, you should include whitespace after a prompt but if you don't it may not be removed during copying.

Always test code blocks when you include them to ensure that the correct text is captured during the copy operation. If you encounter a problem or edge-case contact the maintainers or file an issue.

Separate input from output and remove copy button from output blocks

Terminal commands are separated from the output that they generate. Explanatory text can be included to explain to the reader what is being presented:

- "Run the following command…"
- "This will generate the following output…"

Copy buttons are not included in output blocks. This is to prevent an output being confused for an input. There are also few reasons why someone would copy an output from documentation.

To exclude a copy button from an output block the `no-copy` CSS class must be included within the code block:

```
:class: no-copy
```

Note: a code-block must be labelled with the code-block directive for this to work.

Getting Help

Join us in the Ubuntu Community and post your question there with a descriptive tag.

Guides for developing Ubuntu Pro for WSL

Useful guides when developing and testing Ubuntu Pro for WSL.

How to install UP4W

This guide will show you how to install UP4W for local development and testing.

Requirements:

- A Windows machine with access to the internet
- Appx from the Microsoft Store:

  - Windows Subsystem For Linux
  - Either Ubuntu, Ubuntu 22.04, or Ubuntu (Preview)
- The Windows Subsystem for Windows optional feature enabled

1. Download the Windows Agent and the WSL Pro Service

1. Go to the repository actions page.
2. Click the latest successful workflow run.
3. Scroll down past any warnings or errors, until you reach the Artifacts section.
4. Download:
   - Windows agent: UbuntuProForWSL+...-production
   - wsl-pro-service: Wsl-pro-service_...

Notice that, for the step above, there is also an alternative version of the MSIX bundle enabled for end-to-end testing. Most likely, that's not what you want to download.

2. Install the Windows Agent

This is the Windows-side agent that manages the distros.

1. Uninstall Ubuntu Pro for WSL if you had installed previously:

```
Get-AppxPackage -Name CanonicalGroupLimited.UbuntuPro | Remove-AppxPackage
```

2. Follow the download steps to download UbuntuProForWSL
3. Unzip the artefact
4. Find the certificate inside. Install it into `Local Machine/Trusted people`.
5. Double click on the MSIX bundle and complete the installation.
6. The Firewall may ask for an exception. Allow it.
7. The GUI should show up. You're done.

3. Install the WSL Pro Service

This is the Linux-side component that talks to the agent. Choose one or more distros Jammy or greater, and follow the instructions.

1. Uninstall the WSL-Pro-Service from your distro if you had it installed previously:

```
sudo apt remove wsl-pro-service
```

2. Follow the download steps to download the WSL-Pro-Service.
3. Unzip the artifact.
4. Navigate to the unzipped directory containing the .deb file. Here is a possible path:

```
cd /mnt/c/Users/WINDOWS-USER/Downloads/wsl-pro-service_*
```

5. Install the deb package.

```
sudo apt install ./wsl-pro-service_*.deb
```

6. Ensure it works via systemd:

```
systemctl status wsl-pro.service
```

How to restart UP4W

Some configuration changes only apply when you restart UP4W. Here is a guide on how to restart it. There are two options.

Option 1: Restart your UP4W host machine

This is the simple one. If you're not in a hurry to see the configuration updated, just wait until next time you boot your machine.

Option 2: Restart only UP4W

1. Stop the agent:

```
Get-Process -Name Ubuntu-Pro-Agent | Stop-Process
```

2. Stop the distro, or distros you installed WSL-Pro-Service in:

```
wsl --terminate DISTRO_NAME_1
wsl --terminate DISTRO_NAME_2
# etc.

# Alternatively, stop all distros:
wsl --shutdown
```

3. Start the agent again:
    1. Open the start Menu and search for "Ubuntu Pro for WSL".
    2. The GUI should start.
    3. Wait a minute.
    4. Click on "Click to restart it".
4. Start the distro, or distros you installed WSL-Pro-Service in.

How to access UP4W logs

At some point you may want to read the UP4W logs, most likely for debugging purposes. The agent and the service store their logs separately. This guide shows you where to find each of the logs.

Access the logs for the WSL Pro service

To access the logs of a specific distribution's WSL-Pro-Service, you must first launch the distribution and then query the journal:

```
journalctl -u wsl-pro.service
```

For more information on using the journal, you can check out its man page with `man journalctl` or online.

These logs may be insufficient for proper debugging, so you may be interested in looking at the agent's logs as well.

Access the logs for the Windows Agent

To access the logs for the Windows Agent:

1. Go to your home directory
   - Open the file explorer
   - Write `%USERPROFILE%` at the address
2. In the home directory, find the `.ubuntupro` directory and double-click on it.
3. In the `.ubuntupro` folder, find file `log` and open it with any text editor.
   - This file contains the logs sorted with the oldest entries at the top and the newest at the bottom.

How to enable opt-in features

Some features in UP4W are opt-in or can be toggled on and off via the Windows Registry. While the code is arranged such that CI always tests with those features enabled, when running UP4W on your machine, you may need to toggle them on and off explicitly via the Windows registry. This guide shows you how to do that.

Enable subscribing via the Microsoft Store

1. Open the Windows registry editor: press `Win + R`, type `regedit` and press `Enter`.
2. Navigate to the following key: `HKEY_CURRENT_USER\Software\Canonical\UbuntuPro\`.
3. Create a new DWORD value named `AllowStorePurchase` and set it to `1`.

The next time you open the GUI you'll find the button to subscribe to Ubuntu Pro via the Microsoft Store.

> **Warning**
>
> Beware that can incur in real charges if you proceed with the purchase.

Disable Landscape configuration in the GUI

Landscape configuration page and related buttons are enabled by default, but can be disabled via registry.

1. Open the Windows registry editor: press `Win + R`, type `regedit` and press `Enter`.
2. Navigate to the following key: `HKEY_CURRENT_USER\Software\Canonical\UbuntuPro\`.
3. Create a new DWORD value named `LandscapeConfigVisibility` and set it to `0`.

The next time you open the GUI, you'll find that the Landscape configuration page can be shown via the set up wizard or by clicking on the 'Configure Landscape' button. If that value is not present or set to anything other than `0`, Landscape configuration page and related buttons will be visible.

How to reset UP4W back to factory settings

You can reset Ubuntu Pro for WSL to factory settings following these steps:

1. Shut down WSL

```
wsl --shutdown
```

2. Uninstall the package and shut down WSL:

```
Get-AppxPackage -Name "CanonicalGroupLimited.UbuntuPro" | Remove-AppxPackage`
```

3. Remove the public directory:

```
Remove-Item -Recurse -Force "${env:UserProfile}\.ubuntupro\"
```

4. Remove the registry key:
    1. Press Win+R
    2. Type `regedit.exe` and click OK
    3. Write `HKEY_CURRENT_USER\Software\Canonical\UbuntuPro` at the address bar
        - If this fails, you are done (the key does not exist).
    4. Find the `UbuntuPro` key on the left
    5. Right-click on it
    6. Click delete
5. Install the Windows Agent package again (see the section on *how to install*). You do not need to re-install the WSL-Pro-Service.
6. You're done. Next time you start the GUI it'll be like a fresh install.

2.2.5 Testing

Automate the testing of applications for Ubuntu on WSL with GitHub workflows.

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

This section contains concise references relating to how Ubuntu on WSL is designed, configured and developed.

2.3.1 The Ubuntu distribution on WSL

Information on the Ubuntu distros that are available and supported on WSL.

Distributions

Our flagship distribution (distro) is Ubuntu. It is the default option when you install WSL for the first time. Several releases of the Ubuntu distro are available for WSL.

Each release of Ubuntu for WSL is available as an application from the Microsoft Store. Once a release is installed, it will be available to use in your WSL environment.

Releases of Ubuntu on WSL

These are the releases of Ubuntu that we support and that are available on the Microsoft Store:

- Ubuntu ships the latest stable LTS (Long Term Support) release of Ubuntu. When new LTS versions are released, this release of Ubuntu can be upgraded once the first point release is available.
- Numbered releases – for example, Ubuntu 22.04 LTS – refer to specific Long Term Stability (LTS) releases that receive standard support for five years. For more information on LTS releases, support and timelines, visit the Ubuntu releases page. Numbered releases of Ubuntu on WSL will not be upgraded unless configured to upgrade in `etc/update-manager/release-upgrades`.
- Ubuntu (Preview) is a daily build of the latest development version of Ubuntu, which previews 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.

Interim releases

Interim releases of Ubuntu are currently not supported on WSL.

Naming

Depending on context, releases of Ubuntu are referred to by different names:

1. **App name** is the name of the application for a specific Ubuntu release that you will see in the Microsoft Store.
2. **AppxPackage name** is the name that can be passed to `Get-AppxPackage -Name` in PowerShell to get information about an installed package.
3. **Distro name** is the name logged when you invoke `wsl -l -v` to list installed releases of Ubuntu.
4. **Executable name** is the program you need to run to start the Ubuntu distro.

These naming conventions are summarised in the table below:

App name	AppxPackage name	Distro name	Executable name
`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`

The WSL kernel

The kernel used in WSL environments is maintained by Microsoft. Bug reports and support requests for the WSL kernel should be directed to the official repository for the WSL kernel.

If you are interested in automated testing of applications in Ubuntu WSL, read:

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:

```
- 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:

```
- 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:

```
- 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:

```
- 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
```

2.3.2 Ubuntu Pro for WSL

A glossary of key UP4W system components.

Glossary of UP4W components

The architecture of UP4W and how its components integrate together is covered in our detailed *explanation article*. This glossary includes concise descriptions of the components for reference.

GUI front end

UP4W has a small GUI that helps users provide an Ubuntu Pro token and *configure Landscape*.

When the GUI starts, it attempts to establish a connection to the *UP4W Windows Agent*. If this fails, the agent is restarted. For troubleshooting purposes, you can restart the agent by first stopping the Windows process `ubuntu-pro-agent-launcher.exe` in Windows Task Manger or by issuing the following command in a PowerShell terminal:

```
Stop-Process -Name ubuntu-pro-agent.exe
```

You can then launch the GUI to complete the restart.

Landscape client

The Landscape client is a `systemd` unit running inside every Ubuntu WSL instance. It sends information about the system to the Landscape server. The server, in turn, sends instructions that the client executes.

The Landscape client comes pre-installed in your distro as part of the package `landscape-client`.

You can check the status of the Landscape client in any particular Ubuntu WSL instance by starting a shell in that instance and running:

```
systemctl status landscape-client.service
```

Ubuntu Pro client

The Ubuntu Pro client is a command-line utility that manages the different offerings of your Ubuntu Pro subscription. In UP4W, this executable is used within each of the managed WSL distros to enable Ubuntu Pro services within that distro.

This executable is provided as part of the `ubuntu-pro-client` package, which comes pre-installed in Ubuntu WSL instances since Ubuntu 24.04 LTS.

Windows agent

UP4W's Windows agent is a Windows application running in the background. It starts automatically when the user logs in to Windows. If it stops for any reason, it can be started by launching the UP4W GUI or running the executable from the terminal, optionally with `-vvv` for verbose logging:

```
ubuntu-pro-agent.exe -vvv
```

The Windows agent is UP4W's central hub that communicates with all the components to coordinate them.

WSL Pro service

This is a `systemd` unit running inside every Ubuntu WSL instance. The *Windows agent* running on the Windows host sends commands that the WSL Pro Service executes, such as pro-attaching or configuring the *Landscape client*.

You can check the current status of the WSL Pro Service in any particular distro with:

```
systemctl status wsl-pro.service
```

Details on firewall configuration, Landscape setup and using the Windows registry.

UP4W configuration

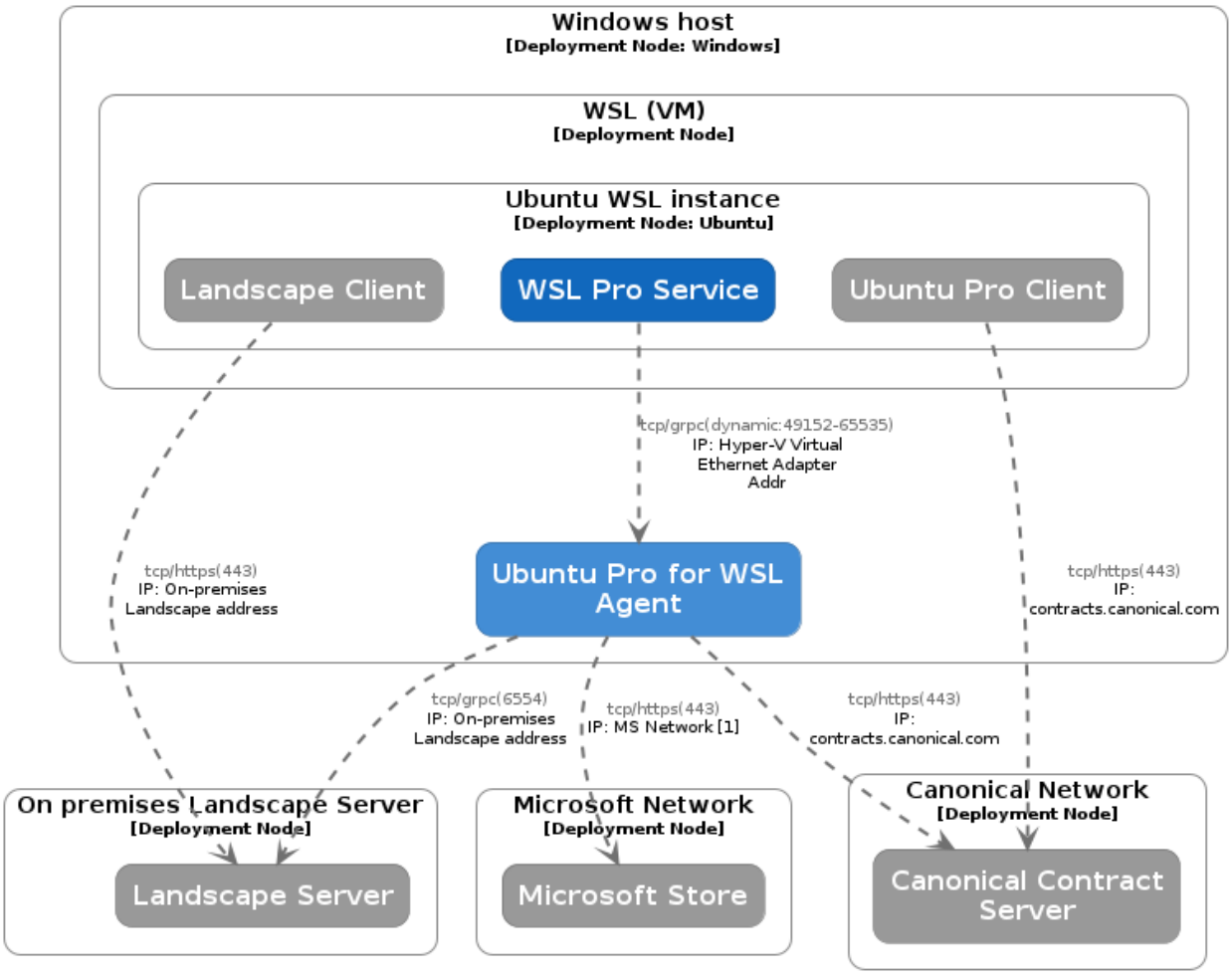
When configuring firewall settings, Landscape clients or the Windows registry, the following references may be useful:

Firewall requirements

> **Pro feature**
>
> This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

Firewall rules must be configured for Ubuntu Pro for WSL to operate fully.

The following figure shows the possible connections between the different components and their default ports and protocols:



The following table lists the default ports and protocols used by Ubuntu Pro for WSL:

Description	Client System	Server System	Protocol	Default Port	Target address
Required for online installation of WSL instances[1].	Windows Host / Pro Agent	MS Store	tcp	https (443)	See Microsoft documentation for a list of addresses to allow.
Ubuntu Pro enablement[2]	Windows Host / Pro Agent	Canonical Contract Server	tcp	https (443)	`contracts.canonical.com`
Landscape management[2]	Windows Host / Pro Agent	Landscape Server	tcp	grpc (6554)	On-premise Landscape address
WSL instance management on the Windows host. Firewall rules set up at installation time of the WSL Pro agent.	WSL Instance / wsl-pro-service	Windows Host / Pro Agent	tcp	grpc (dynamic:49 65535)	Hyper-V Virtual Ethernet Adapter IP
Ubuntu Pro[2][3].	WSL Instance / Ubuntu Pro client	Canonical Contract Server	tcp	https (443)	`contracts.canonical.com`
Landscape[2].	WSL Instance / Ubuntu Pro client	Landscape Server	tcp	https (443)	On-premise Landscape address

If the client system is behind a proxy, ensure that the proxy is configured to allow the required connections.

Landscape configuration schema

Pro feature

This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

Both Landscape clients are configured via a single, plain text configuration file (e.g., `landscape.conf` or `landscape.ini`). This file is provided to the Windows host.

The schema for this file is the same as Landscape for Ubuntu desktop or server, with a few additional keys specific to the WSL settings, which can be grouped into keys that affect just the Windows-side client and keys that affect both the Windows-side client and the Ubuntu WSL-side client(s). These additions are documented below.

> See more: Landscape | Configure Ubuntu Pro for WSL for Landscape

Here is an example of what the configuration looks like:

[1] Access to the Microsoft Store is required for the online installation of WSL instances. Without it Ubuntu Pro for WSL will still be functional but it will only be possible to install WSL instances centrally from Landscape from custom tarballs, not using the official Ubuntu releases.

[2] Access to the contract server and Landscape server is required for proper operation of Ubuntu Pro for WSL.

[3] For air-gapped installation refer to the Ubuntu Pro documentation.

```
[host]
url = landscape-server.domain.com:6554

[client]
url = https://landscape-server.domain.com/message-system
ping_url  = http://landscape-server.domain.com/ping
account_name = standalone
log_level = debug
ssl_public_key = C:\Users\user\Downloads\landscape_server.pem
```

Host

This section contains settings unique to the Windows-side client. Currently these consist of a single key:

- `url`: The URL of your Landscape account followed by a colon (`:`) and the port number. Port 6554 is the default for Landscape Quickstart installations.

Client

This section contains settings used by both clients. Most keys in this section behave the same way they would on a traditional Landscape setup. Only the following keys behave differently:

- `ssl_public_key`: This key must be a Windows path. The WSL instances will have this path translated automatically.
- `computer_title`: This key will be ignored. Instead, each WSL instance will use its Distro name as computer title.
- `hostagent_uid`: This key will be ignored.

Warning

The certificate referred to by the `ssl_public_key` key is used for both the Landscape client inside the WSL instances as well as the Windows background agent. Until version 0.1.15 of Ubuntu Pro for WSL, the app explicitly requires referencing a path to the SSL certificate on a Windows host machine. Newer versions completely follow the Windows OS certificate stores, only requiring reference to that certificate if the machine running the Landscape server is not trusted on your network.

See more: GitHub | Landscape client configuration schema

Windows registry

Pro feature

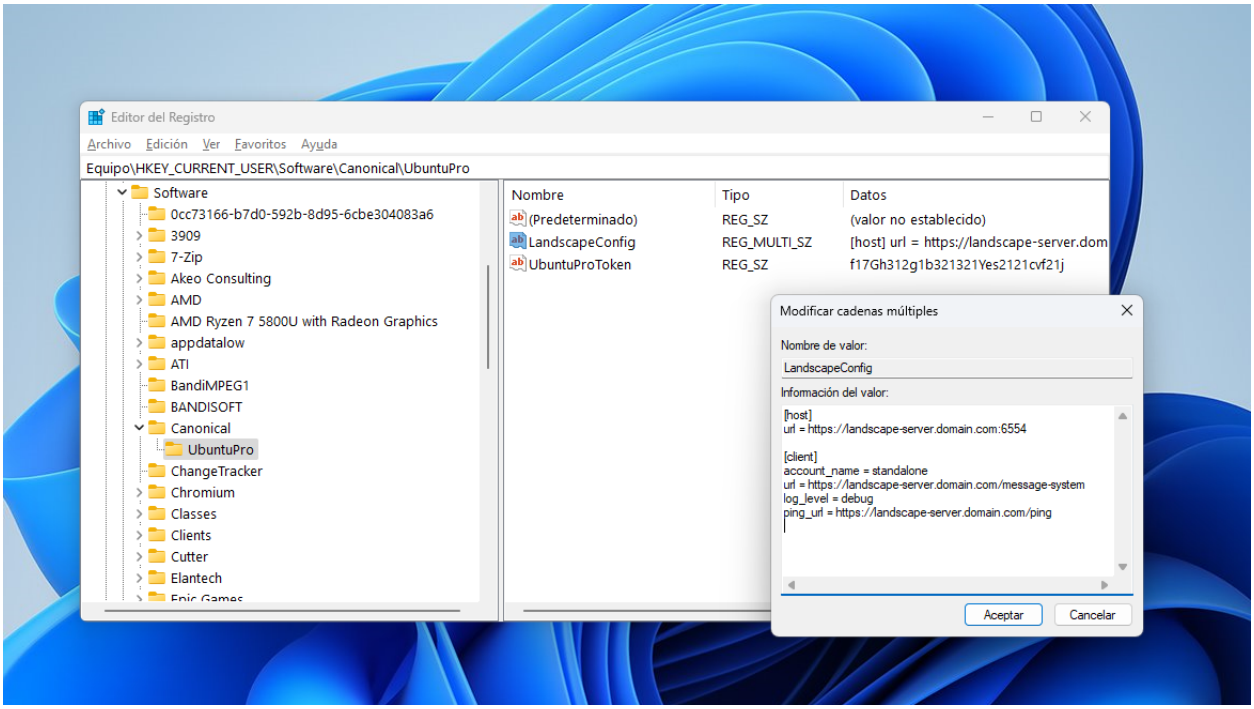
This page refers to features that require an Ubuntu Pro subscription and access to the Ubuntu Pro for WSL application, which is currently under development.

The Windows registry is a database provided by Windows where programs can read and write information. UP4W uses it as a read-only source of configuration.

See more: Microsoft Learn | Windows registry information for advanced users

In UP4W, you can use the Windows registry to supply the configuration for Ubuntu Pro and Landscape to the *Windows Agent*.

See more: *install UP4W and add a Pro token*



Expected contents of the UbuntuPro registry key

The Windows agent will read the following values from the key at `HK_CURRENT_USER\Software\Canonical\UbuntuPro`:

- Value `UbuntuProToken` (type `String`) expects the Ubuntu Pro token for the user.
- Value `LandscapeConfig` (type `String` or `Multi-line string`) expects the *Landscape configuration*.

Information helpful for developers working on Ubuntu Pro for WSL.

Debugging and testing

The reference material in this section is helpful for contributors when debugging the application and contributing code.

Windows Agent CLI

See first: *UP4W - Windows Agent*

Usage

User commands

ubuntu-pro-agent

Ubuntu Pro for WSL agent

Synopsis

Ubuntu Pro for WSL agent for managing your pro-enabled distro.

```
ubuntu-pro-agent COMMAND [flags]
```

Options

```
  -c, --config string     configuration file path
  -h, --help              help for ubuntu-pro-agent
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

ubuntu-pro-agent clean

Removes all the agent's data and exits

```
ubuntu-pro-agent clean [flags]
```

Options

```
  -h, --help   help for clean
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

ubuntu-pro-agent completion

Generate the autocompletion script for the specified shell

Synopsis

Generate the autocompletion script for ubuntu-pro-agent for the specified shell. See each sub-command's help for details on how to use the generated script.

Options

```
  -h, --help   help for completion
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

ubuntu-pro-agent completion bash

Generate the autocompletion script for bash

Synopsis

Generate the autocompletion script for the bash shell.

This script depends on the 'bash-completion' package. If it is not installed already, you can install it via your OS's package manager.

To load completions in your current shell session:

```
source <(ubuntu-pro-agent completion bash)
```

To load completions for every new session, execute once:

Linux:

```
ubuntu-pro-agent completion bash > /etc/bash_completion.d/ubuntu-pro-agent
```

macOS:

```
ubuntu-pro-agent completion bash > $(brew --prefix)/etc/bash_completion.d/ubuntu-pro-
↪agent
```

You will need to start a new shell for this setup to take effect.

```
ubuntu-pro-agent completion bash
```

Options

```
  -h, --help              help for bash
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

ubuntu-pro-agent completion fish

Generate the autocompletion script for fish

Synopsis

Generate the autocompletion script for the fish shell.

To load completions in your current shell session:

```
ubuntu-pro-agent completion fish | source
```

To load completions for every new session, execute once:

```
ubuntu-pro-agent completion fish > ~/.config/fish/completions/ubuntu-pro-agent.fish
```

You will need to start a new shell for this setup to take effect.

```
ubuntu-pro-agent completion fish [flags]
```

Options

```
  -h, --help              help for fish
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

ubuntu-pro-agent completion powershell

Generate the autocompletion script for powershell

Synopsis

Generate the autocompletion script for powershell.

To load completions in your current shell session:

```
ubuntu-pro-agent completion powershell | Out-String | Invoke-Expression
```

To load completions for every new session, add the output of the above command to your powershell profile.

```
ubuntu-pro-agent completion powershell [flags]
```

Options

```
  -h, --help              help for powershell
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

ubuntu-pro-agent completion zsh

Generate the autocompletion script for zsh

Synopsis

Generate the autocompletion script for the zsh shell.

If shell completion is not already enabled in your environment you will need to enable it. You can execute the following once:

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
```

To load completions in your current shell session:

```
source <(ubuntu-pro-agent completion zsh)
```

To load completions for every new session, execute once:

Linux:

```
ubuntu-pro-agent completion zsh > "${fpath[1]}/_ubuntu-pro-agent"
```

macOS:

```
ubuntu-pro-agent completion zsh > $(brew --prefix)/share/zsh/site-functions/_ubuntu-pro-
↪agent
```

You will need to start a new shell for this setup to take effect.

```
ubuntu-pro-agent completion zsh [flags]
```

Options

```
  -h, --help              help for zsh
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

ubuntu-pro-agent version

Returns version of agent and exits

```
ubuntu-pro-agent version [flags]
```

Options

```
  -h, --help   help for version
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

Hidden commands

Those commands are hidden from help and should primarily be used by the system or for debugging.

WSL Pro Service CLI

See first: *UP4W - WSL Pro Service*

Usage

User commands

wsl-pro-service

WSL Pro Service

Synopsis

WSL Pro Service connects Ubuntu Pro for WSL agent to your distro.

```
wsl-pro-service COMMAND [flags]
```

Options

```
  -c, --config string     configuration file path
  -h, --help              help for wsl-pro-service
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

wsl-pro-service completion

Generate the autocompletion script for the specified shell

Synopsis

Generate the autocompletion script for wsl-pro-service for the specified shell. See each sub-command's help for details on how to use the generated script.

Options

```
  -h, --help   help for completion
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

wsl-pro-service completion bash

Generate the autocompletion script for bash

Synopsis

Generate the autocompletion script for the bash shell.

This script depends on the 'bash-completion' package. If it is not installed already, you can install it via your OS's package manager.

To load completions in your current shell session:

```
source <(wsl-pro-service completion bash)
```

To load completions for every new session, execute once:

Linux:

```
wsl-pro-service completion bash > /etc/bash_completion.d/wsl-pro-service
```

macOS:

```
wsl-pro-service completion bash > $(brew --prefix)/etc/bash_completion.d/wsl-pro-service
```

You will need to start a new shell for this setup to take effect.

```
wsl-pro-service completion bash
```

Options

```
  -h, --help              help for bash
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

wsl-pro-service completion fish

Generate the autocompletion script for fish

Synopsis

Generate the autocompletion script for the fish shell.

To load completions in your current shell session:

```
wsl-pro-service completion fish | source
```

To load completions for every new session, execute once:

```
wsl-pro-service completion fish > ~/.config/fish/completions/wsl-pro-service.fish
```

You will need to start a new shell for this setup to take effect.

```
wsl-pro-service completion fish [flags]
```

Options

```
  -h, --help              help for fish
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

wsl-pro-service completion powershell

Generate the autocompletion script for powershell

Synopsis

Generate the autocompletion script for powershell.

To load completions in your current shell session:

```
wsl-pro-service completion powershell | Out-String | Invoke-Expression
```

To load completions for every new session, add the output of the above command to your powershell profile.

```
wsl-pro-service completion powershell [flags]
```

Options

```
  -h, --help              help for powershell
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

wsl-pro-service completion zsh

Generate the autocompletion script for zsh

Synopsis

Generate the autocompletion script for the zsh shell.

If shell completion is not already enabled in your environment you will need to enable it. You can execute the following once:

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
```

To load completions in your current shell session:

```
source <(wsl-pro-service completion zsh)
```

To load completions for every new session, execute once:

Linux:

```
wsl-pro-service completion zsh > "${fpath[1]}/_wsl-pro-service"
```

macOS:

```
wsl-pro-service completion zsh > $(brew --prefix)/share/zsh/site-functions/_wsl-pro-
↪service
```

You will need to start a new shell for this setup to take effect.

```
wsl-pro-service completion zsh [flags]
```

Options

```
  -h, --help              help for zsh
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

wsl-pro-service version

Returns version of agent and exits

```
wsl-pro-service version [flags]
```

Options

```
  -h, --help   help for version
```

Options inherited from parent commands

```
  -c, --config string     configuration file path
  -v, --verbosity count   issue INFO (-v), DEBUG (-vv) or DEBUG with caller (-vvv) output
```

Hidden commands

Those commands are hidden from help and should primarily be used by the system or for debugging.

QA Process

Generalities

> **Note**
>
> We always use the latest Go version available. Information about specific Go versions on this page may be outdated, as the version used is periodically updated.

`wsl-pro-service` is seeded only on WSL images.

```
Build-dep: golang-go (\>= 2:1.21\~)
```

At any point in time, only the latest two versions of the Go toolchain receive security patches. Hence, we need to keep backporting new releases to fix vulnerabilities. They follow an approximate 6-month release cycle, so Go 1.21 should fall out of support by August 2024.

Process

WSL Pro Service follows a robust continuous integration and testing process. It is covered by a comprehensive automated test suite.

The team applies the following quality criteria:

- All changes are thoroughly reviewed and approved by core team members before integration.
- Each change is thoroughly tested at the unit, integration and system levels. All the tests pass in all supported architectures.
- Releases are reviewed as part of the SRU exception.

The test plan is **completely automated** and runs **every time a change is merged**, as well as **during packaging**. This covers integration and end-to-end tests. Integration tests run on each LTS affected by the SRU to ensure compatibility.

Testing also covers the upgrade from the current version to the proposed version.

Tests are not executed on different versions of Windows due to testing environment limitations.

Packaging QA

To prepare the release to LTS, the following procedure is being completed to ensure quality:

- All autopkgtests pass. Unit tests are executed as autopkgtests. Running higher-level tests would require a Windows VM. It is not available in autopkgtest at the moment. Even if wsl-pro-service tests could run in a VM, they wouldn't test anything real.
- The package does not break when upgrading.
- The binary is identical to the CI build, with only Debian packaging changes.
- The copyrights and changelog are up to date.
- An upgrade test from the previous package version has been performed using apt install/upgrade.

Code sanity

Code sanity checks are performed automatically on each build. They verify:

- Code linting.
- Go module files are up to date.
- Generated files are up to date.
- Any binary in the project builds.
- The Debian package builds.
- Vulnerabilities. It is a run of `govulncheck`.

All the layers are tested from APIs to mocks to the service itself

Example reports

- Code sanity and unit testing: QA workflow
- Integration tests: end-to-to-end tests workflow

Go Quality checks (ubuntu, wsl-pro-service) summary

Code sanity summary on /wsl-pro-service

Job	Status
Linting	🟢
Go module files up to date	🟢
Generated files up to date	🟢
Build	🟢
Vulnerability scanning	🟢

Job summary generated at run-time

Go Quality checks (ubuntu, common) summary

Code sanity summary on /common

Job	Status
Linting	🟢
Go module files up to date	🟢
Generated files up to date	🟢
Build	🟢
Vulnerability scanning	🟢

Job summary generated at run-time

Code coverage

There is no Codecov report due to the limitations of private projects. However, code coverage is calculated and displayed at testing time. Coverage is manually reviewed by the engineers.

Bug reporting

The main bug tracker remains on GitHub. GitHub Templates are available to help the user with the bug-reporting process and provide the right information.

Wsl-pro supports ubuntu-bug reporting to Launchpad with an apport hook but we are not collecting any data at the moment.

References

- Project Documentation
- Ubuntu Pro for WSL SRU exception
- Ubuntu Pro tools SRU exception

2.4 Explanation

This section contains explanations that will help you develop a deeper understanding of how Ubuntu Pro for WSL is designed.

2.4.1 UP4W architecture

This page describes the different components of UP4W and how they integrate together to form the software architecture.

Background

What is Ubuntu WSL?

With the Windows Subsystem for Linux (WSL), Linux distributions can be run on Windows with minimal overhead. Ubuntu WSL is an image that is optimised for running Ubuntu WSL. A user can create an Ubuntu WSL instance on a Windows machine and use that instance as a production-ready development environment. Ubuntu WSL also benefits from tight integration with Ubuntu Pro and Landscape, which makes it easier to secure Ubuntu WSL instances and manage them at scale.

Why is managing WSL instances at scale difficult?

An individual user can create and configure multiple, independent instances of Ubuntu WSL on their machine. For a system administrator, who is managing fleets of Windows machines with multiple users, the potential number of Ubuntu WSL instances increases dramatically. While the system administrator has tools for managing Windows machines, WSL instances can be a black box that cannot be directly monitored or patched. This can result in WSL instances that do not follow system administration policies.

How does Ubuntu Pro for WSL solve this problem?

Ubuntu Pro for WSL (UP4W) helps automate the management of Ubuntu WSL. For each new instance that is discovered or created, UP4W will automatically:

- Attach them to your Ubuntu Pro subscription
- Enrol them with your Landscape server

The integration with Ubuntu Pro keeps instances secure, while the integration with Landscape enables remote deployment and management.

Without UP4W, these configurations would be manual, time-consuming and error-prone.

The components of UP4W

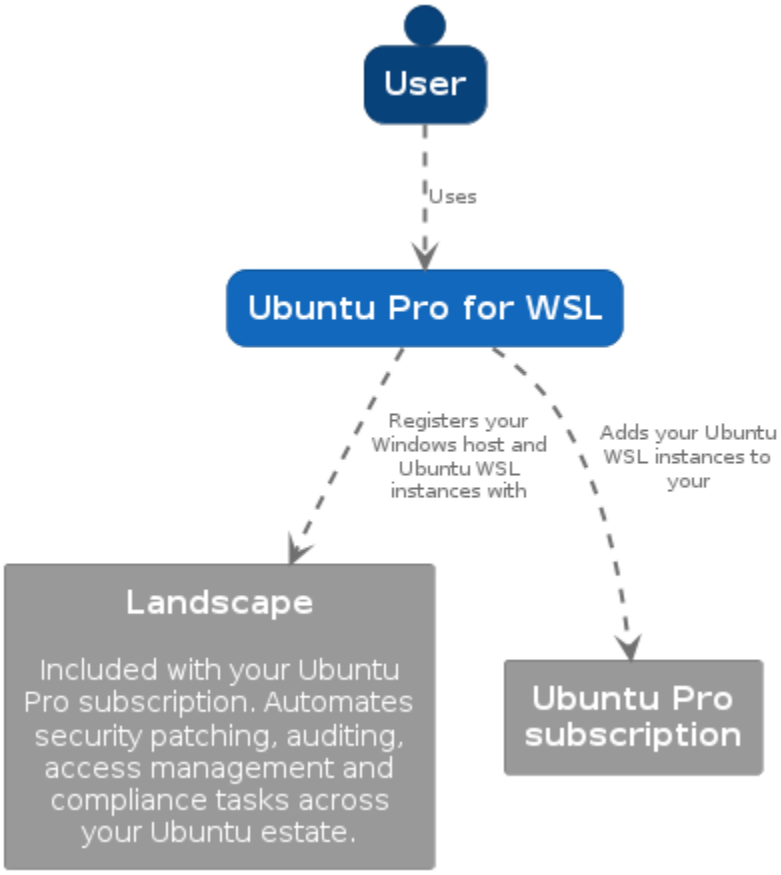
> **WSL architecture**
>
> WSL is maintained by Microsoft and its architecture is outside the scope of this page.
>
> A good overview of WSL architecture is provided in this blog from Microsoft.

Overview

UP4W consists of some components that run on a Windows host and others that run within instances of Ubuntu WSL.

A user interacts with the UP4W application. UP4W then automatically pro-attaches and Landscape-enrols each new instance of Ubuntu WSL that is created on the Windows host.

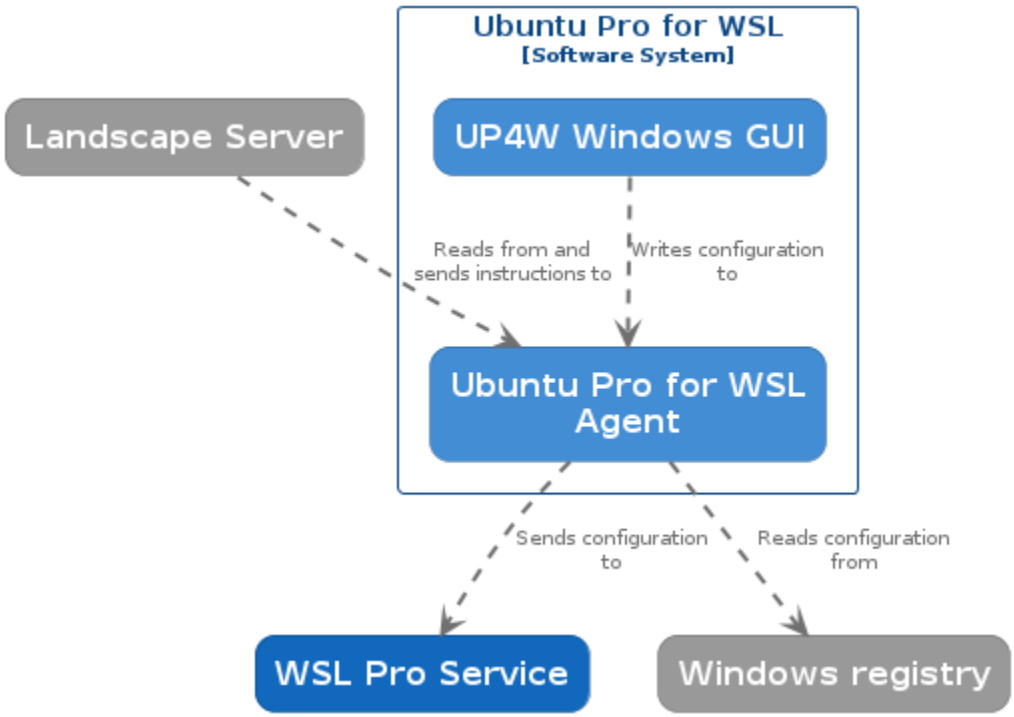
In an organisation, multiple users of Windows machines can create Ubuntu WSL instances, which are secured by Ubuntu Pro and that can be managed by Landscape.



Components on the Windows host

The Windows host is a single machine running the Windows OS. WSL instances can be created and instanced on this host. The UP4W application that is installed on the Windows host consists of a GUI front end and an agent that runs in the background.

A user enters a Pro token and Landscape configuration using the GUI. When the GUI is launched it starts the Windows Agent, if it's not already running. The agent runs in the background on the Windows host and manages communication with other components, including the remote Landscape server and the Pro service running within each instance of Ubuntu WSL. The agent is responsible for managing the state of instances and acts as a bridge between those instances and Landscape. If the configuration details are valid, all new instances will have Ubuntu Pro enabled and will be able to communicate with the Landscape server.



It is possible to bypass the GUI and instead configure UP4W using the Windows registry. This may be the preferred option for those operating at scale. When UP4W is launched, a registry path is created that can be used to store a Pro token and a Landscape configuration. A system administrator can use a remote management solution like Intune to configure the registry on fleets of devices.

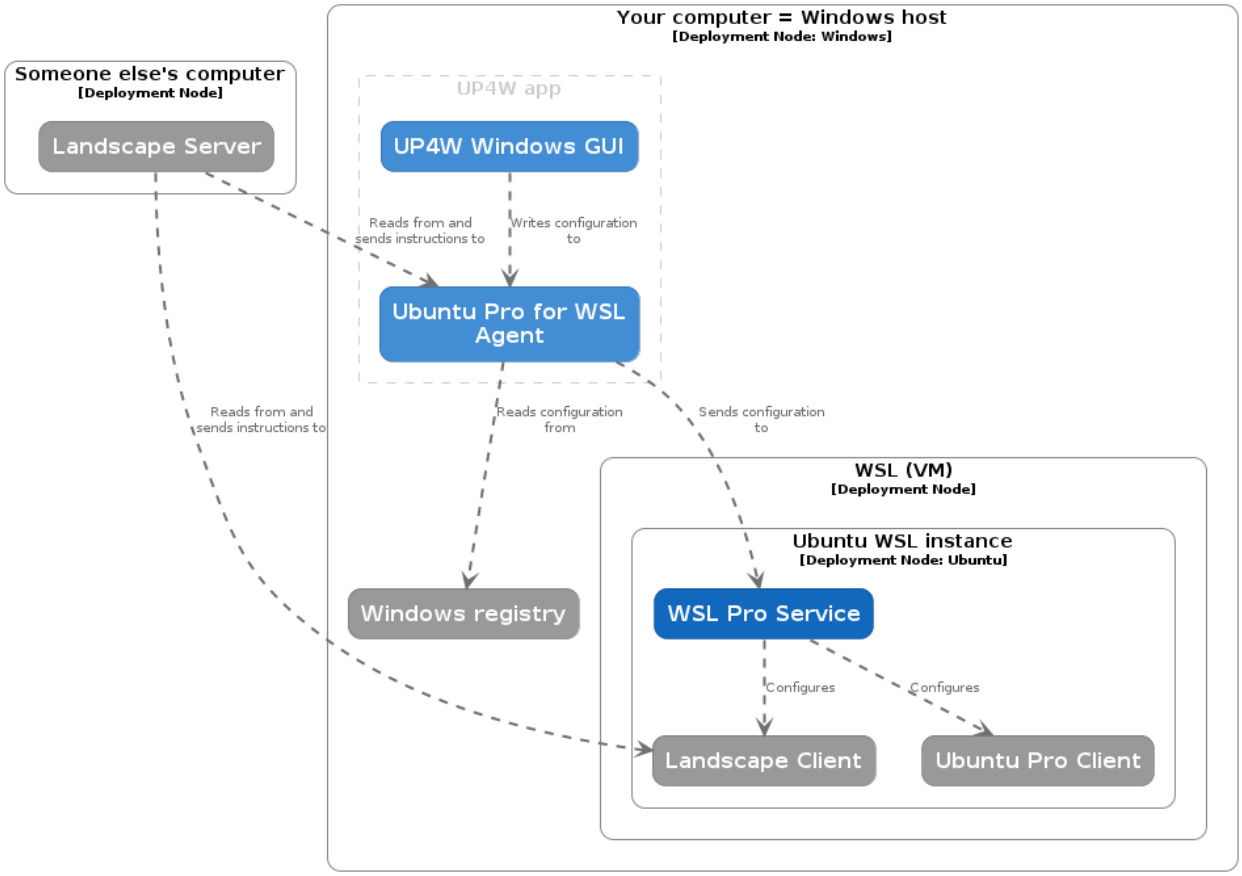
Components on Ubuntu WSL instances

The WSL Pro service runs in each instance of Ubuntu WSL. From the host, the Windows agent communicates with this service. This allows commands to be sent from the host, which are then executed by the Pro service on each instance. When an Ubuntu WSL instance is started, the WSL Pro service runs and queries the Windows agent on the host for the status of the Pro subscription. If the Pro token is valid, it is retrieved from the Windows agent and passed to the Pro service running on Ubuntu WSL instances. If not, Ubuntu Pro is disabled on the instances.

Pre-installed on each instance of Ubuntu WSL is an Ubuntu Pro client and a Landscape client. After a Pro token is provided, the Windows agent can send a command to the Ubuntu Pro client to execute pro-attach on active instances. Similarly, when a Landscape configuration is provided, the Windows agent can send a command to configure the Landscape client in each instance.

The administrator of the Landscape server can also send commands to the agent to deploy new instances or delete

existing instances.



Ubuntu WSL instances that are deployed at scale can be extensively customised. The Landscape API can be used to automate the deployment of a custom rootfs. As of Ubuntu 24.04 LTS, cloud-init is pre-installed on Ubuntu WSL instances, which makes it possible to automate the configuration of instances created from that release.

Source code

UP4W is open-source software. You can look at the code in the GitHub repo.

The following technologies are used to build UP4W:

- **Go**: Windows agent and WSL Pro service
- **Flutter**: GUI front end
- **gRPC**: communication between back end and front end