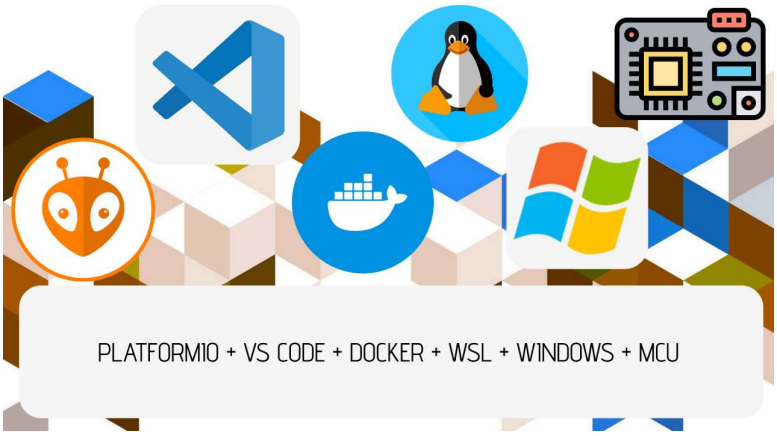


Connecting microcontrollers to PlatformIO running in a Linux Container on a Windows Host Computer

Emil Jenssen
Cybernetic and Robotic Engineer

11 de abril de 2024

Setting up a development environment for embedded systems often happens locally on your host OS computer, simply for convenience and easy access to any peripheral, such as USB devices. And like most of us, you might have forgotten to note down each configuration you do for your tools, compilers, and IDEs.

To solve this, we can look into the IT world for inspiration. Where containerization of development environments have gotten far with solutions such as Visual Studio Code Dev Containers. This allows the developer to isolate the development environment into preconfigured containers, ensuring consistency when changing computers or the host OS.

While in the embedded world these kinds of solutions meet challenges when you want to have access to your peripherals in an isolated development environment across different types of host OS. In our case, it is not possible to interact with any peripheral in a Linux container running on a Windows host OS.

> Interacting with peripherals inside a Linux container is possible for Linux containers running on Linux host OS, as shown by Nikul Padhya in his article about Utilizing Docker, Visual studio code, PlatformIO and "Dev container" extension for embedded system development.

Is there any existing solution to the problem we are experiencing?

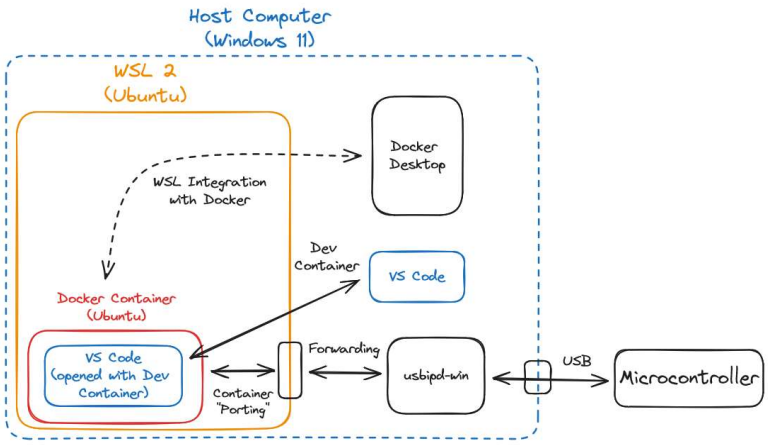
After a bit of searching online, I hatched down an idea on how to make it work. So, hear me out...

There exists a well-integrated compatibility layer of Linux for Windows that has existed since 2016, called **Windows Subsystem for Linux (WSL)**. In addition, **Docker** is a containerization tool that is well **integrated with WSL 2**. Which means, we might be able to make our host OS seem like a Linux OS from the point of view of the container itself.

...and after a bit more searching, I was able to find the last piece of the puzzle, **usbipd-win**! This software enables you to share locally any USB connection on the Windows host OS to the Linux OS running in **WSL 2.**

With all this combined, and with the **Dev Containers** extension in **VS Code**, we will have a smooth experience with a containerized environment for embedded development!

Let me show you the idea:



The idea is to have our development environment running inside a **Linux container**, running inside the **Linux OS in WSL**. Where **usbipd-win** forwards the **USB device** connected on the **Windows host OS** to the **Linux OS in WSL**, where it once more gets passed into the **Linux container**. And with **WSL integration in Docker**, we will be able to access the container from our **Windows host OS** as if it was running in the **host OS.**

With this combination it makes using **VS Code** as our IDE and the **Dev Containers** extension the smoothest experience for developing with containerized development environments and hardware connected on the **host OS**.

For simplicity, we will be using the **PlatformIO** extension to show case the idea.

Now let's set this up

First, we will need to go through some prerequisites:

- A **microcontroller board** and an USB cable that you can connect to your host computer with. In my case, I'm using an **STM32f401 Nucleo-64 board.**
- A **host computer** with an **x64 processor** is required (for **usbipd-win**).
- The **host OS** must be **Windows 11** or **Windows 10 version 2004 or higher** (Build 19041 and higher).
- A **Linux distro** (Recommend the default, **Ubuntu**) **installed and enabled with WSL 2** on the **Windows host OS**, **link.**
- **Docker Desktop** installed and up and running on **Windows host OS.** With **Use WSL 2 instead of Hyper-V** option enabled in the settings.
- **VS Code** installed with the **Dev Containers** extension.

Setup the USB device connection to WSL 2

To setup the USB device connection to our Linux OS running in WSL 2, we'll be using this **tutorial**. Which requires us to **download and install usbipd-win** (download the .msi file and install it).

Open PowerShell on the Windows host OS and run it as an administrator to be allowed to share the connected USB devices. In the terminal give the following command:

```
usbipd list
```

This gives us a list of all the USB devices connected to the host computer.

As you see on the list, we have the **ST-Link Debugger** enlisted, which is on the **STM32f401 Nucleo-64 board**. Then, we will bind the USB device to **usbipd-win**:

```
usbipd bind --busid 8-4
```

If you get the following error: *usbipd: error: Access denied; this operation requires administrator privileges.* It is because you are not running the terminal (PowerShell) as administrator.

And if you see this warning: *usbipd: warning: Unknown USB filter 'tdevflt' may be incompatible with this software; 'bind --force' may be required.* You might need to use the *--force* argument:

```
Usbipd bind --force --busid 8-4
```

We are ready to attach/forward the USB device to WSL:

```
Usbipd attach --wsl --busid 8-4
```

NB! Every time you disconnect your USB device, you will need to rerun this command.

Now we will continue with **Docker Desktop**. If you haven't done it before, open up **Docker Desktop** and go to **settings > Resources > WSL integration**. Then check **Enable integration with my default WSL distro.** This allows us to interact with the docker container as if it was running directly in the host OS.

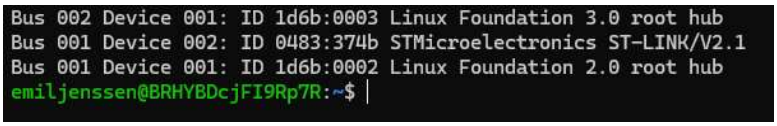
Minimize docker desktop, and open **Ubuntu** (application). You should see the app on your **Windows host OS**, which prompts you to a WSL command line. In this terminal you are now inside the Linux distribution running as an WSL.

Note: If you do not see the **Ubuntu** application, set it up with WSL (check prerequisites).

To see if the the USB device was forwarded correctly to the Linux OS in WSL 2, in the WSL command line (Ubuntu) give it the following command:

```
lsusb
```

It should look like this:



Spin up the Linux Container

Since **Docker** is well integrated with **WSL 2**, we will be able to work with the container from both the **Linux OS in WSL 2** and in the **Windows host OS**. The following steps can be done either from a **WSL command line** or a terminal in your **Windows host OS**:

Let's begin with preparing an image of an **Ubuntu** container:

```
docker pull ubuntu:latest
```

Spin up the container (**NB!** This is will work, because the WSL handles the relation to */dev/bus/usb* from the host side, automatically for us):

```
docker run -it -d --name ubuntu --privileged -v
/dev/bus/usb:/dev/usb/usb ubuntu /bin/bash
```

Let's check if the USB device is passed down correctly to the device. Now we will enter into the Linux container:

```
docker exec -it ubuntu /bin/bash
```

Install an USB utility in the container first:

```
apt-get update && apt-get install usbutils
```

Then, check if the USB device is accessible to the container:

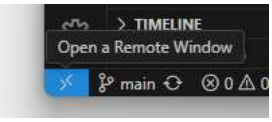
```
lsusb
```

```
root@e364eed9343c:/# lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux 5.15.146.1-microsoft-standard-WSL
Bus 001 Device 002: ID 0483:374b STMicroelectronics STM32 STLink
Bus 001 Device 001: ID 1d6b:0002 Linux 5.15.146.1-microsoft-standard-WSL
root@e364eed9343c:/#
```

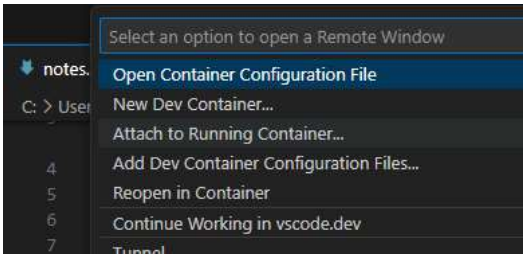
It seems to work well! Time to see if we can load up a simple example to the microcontroller from the container.

Setting up PlatformIO with VS Code and Dev Container

Open **VS Code** (on your host OS) and make sure you have the **Dev Container** extension installed. Then, down at the bottom left corner of VS Code, click on **Open a Remote Window.**

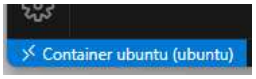


You will then be prompted to the **Dev Container** menu, click on **Attach to Running Container...**



And click on the container, in our case it will be **ubuntu**:

You are now prompted to a **VS Code** instance that is running in your **Linux container**:

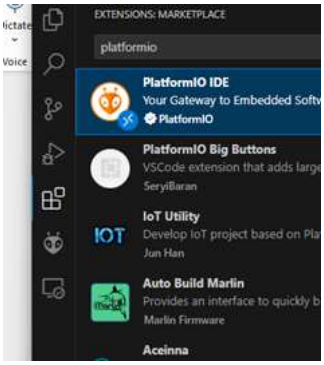


Before we add the **PlatformIO** extension to the container, we need to install **Python**. You can run these commands in the container by opening a new terminal in **VS Code**, and install **Python**:

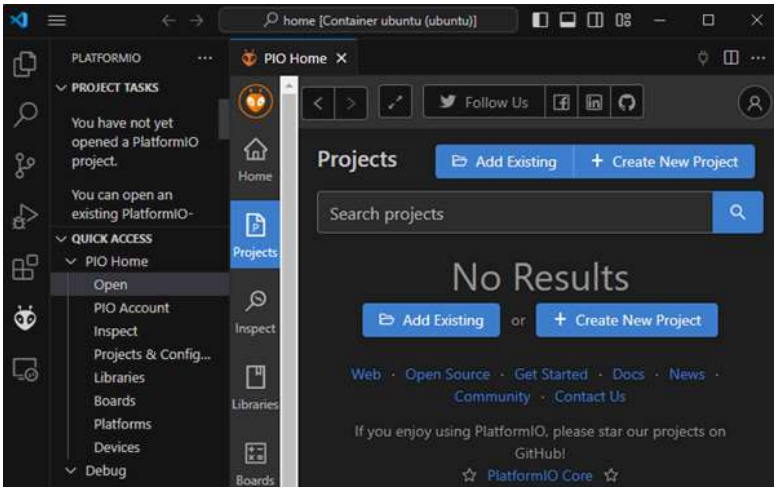
```
apt-get update && apt-get install python3 -y && apt-
get install python3-venv -y && apt-get install
```

```
python-is-python3
```

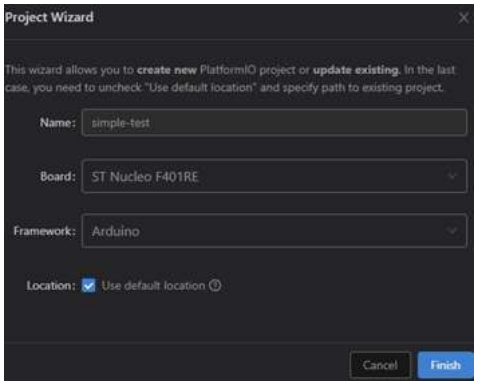
Next, install the **PlatformIO** extension from the extensions tab on the left side:



You will be prompted to restart **VS Code**. After the restart you will be able to open up **PlatformIO** from the extension button on the left side (looks like an ant's head). In the extension menu, under **Quick Access** > **PIO Home**, click on **Open**. Then in the **PIO Home Menu**, click on **Projects** and **Create New Project**:

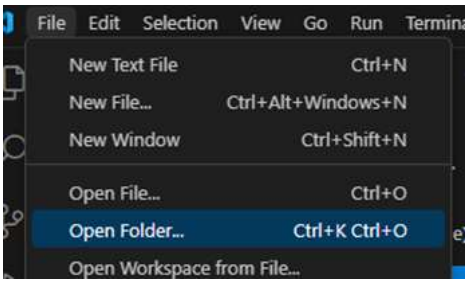


In our case we will setup a simple test project with the **ST Nucleo F401RE** board and the **Arduino** framework:

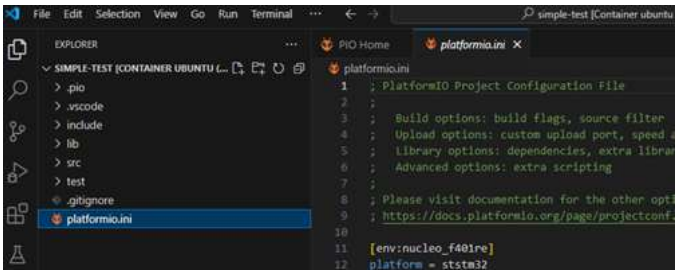


NB! Because **PlatformIO** doesn't distinguish the **Windows Host OS** from working inside the **Linux Container**, it will generate the folder name as following *project\simpletest*. Giving us the following project folder path */root/Documents/projects\simpletest*, which is an invalid path.

A quick fix is to **Open Folder...** and open from */root/*. Then rename it from *projects\simpletest* to *projects\simpletest*. Now we have a valid path.



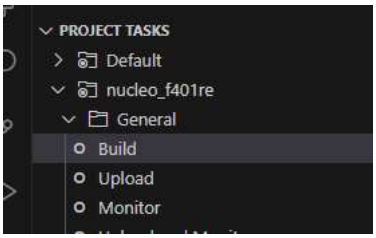
Once more, **Open Folder...** and open */root/Documents/PlatformIO/Projects/simple-test/*



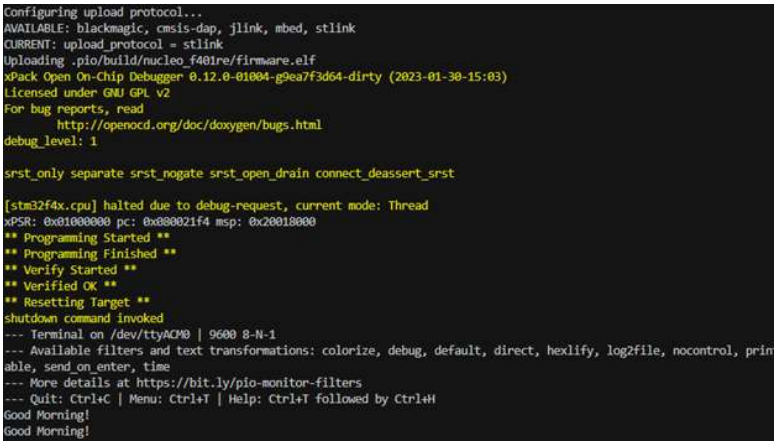
Let's add a simple example code in *src/main.cpp*:

```
#include <Arduino.h>

void setup(){
    Serial.begin(9600);
}

void loop(){
    delay(5000);
    Serial.println("Good morning!");
}
```

Open the **PlatformIO** extension, under *nucleo_f40re* (if that's your board) then *General*, and click **build**:



When the build is complete. Click **Upload and Monitor**:



Success! Now we can write, build/compile, and upload the code to the microcontroller, and monitor the serial communication, all from a container!

Conclusion

We have been able to see how we can work with microcontrollers from a Linux container, when the container is running on a Windows host OS computer. Which enables us to develop with a consistent development environment on cross-platforms. To work on a Linux host OS computer, you simply set up a Docker container as normal by binding the */dev/bus/usb* volumes together.

If you use different Linux distros, you might need to check privileges and permissions, as mentioned by Nikul Padhya in his **article.**

Additional thoughts

You might ask yourself, for what reason do I need to set up such a complex system to work with **PlatformIO**? Which is a valid question, and for **PlatformIO** this might seem like an overkill...

However! If you plan to set up unique embedded development environments with specific tools, compilers and personalized configurations, you can finally predefine this with the use of *Dockerfiles* (for containers) and *devcontainer.json* (for VS Code).

This way you will have your development environment predefined and indirectly documented, which both saves time and ensures consistency in your development.

¿Te ha gustado este artículo?

Sigue para no perderte ninguna novedad.

Emil Jenssen

Cybernetic and Robotic Engineer

Seguir