Using USB/IP with Docker Desktop

- Setup and use
  - Step one: Run a USB/IP server
  - Step two: Start a privileged Docker container
  - Step three: Enter the mount namespace of PID 1
  - Step four: Use the USB/IP tools
    - List USB devices
    - Attach a USB device
    - Verify device attachment
  - Step five: Access the device from another container

{{< summary-bar feature_name="USB/IP support" >}}

USB/IP enables you to share USB devices over the network, which can then be accessed from within Docker containers. This page focuses on sharing USB devices connected to the machine you run Docker Desktop on. You can repeat the following process to attach and use additional USB devices as needed.

> Docker Desktop includes built-in drivers for many common USB devices but Docker can't guarantee every possible USB device works with this setup.

Setup and use

Step one: Run a USB/IP server

To use USB/IP, you need to run a USB/IP server. For this guide, the implementation provided by jiegec/usbip will be used.

1. Clone the repository.

```
$ git clone https://github.com/jiegec/usbip
$ cd usbip
```

2. Run the emulated Human Interface Device (HID) device example.

```
$ env RUST_LOG=info cargo run --example hid_keyboard
```

Step two: Start a privileged Docker container

To attach the USB device, start a privileged Docker container with the PID namespace set to `host`:

```
$ docker run --rm -it --privileged --pid=host alpine
```

`--privileged` gives the container full access to the host, and `--pid=host` allows it to share the host's process namespace.

Step three: Enter the mount namespace of PID 1

Inside the container, enter the mount namespace of the `init` process to gain access to the pre-installed USB/IP tools:

```
$ nsenter -t 1 -m
```

Step four: Use the USB/IP tools

Now you can use the USB/IP tools as you would on any other system:

List USB devices

To list exportable USB devices from the host:

```
$ usbip list -r host.docker.internal
```

Expected output:

```
Exportable USB devices
======================
 - host.docker.internal
      0-0-0: unknown vendor : unknown product (0000:0000)
           : /sys/bus/0/0/0
           : (Defined at Interface level) (00/00/00)
           :  0 - unknown class / unknown subclass / unknown protocol (03/00/00)
```

Attach a USB device

To attach a specific USB device, or the emulated keyboard in this case:

```
$ usbip attach -r host.docker.internal -d 0-0-0
```

Verify device attachment

After attaching the emulated keyboard, check the `/dev/input` directory for the device node:

```
$ ls /dev/input/
```

Example output:

```
event0  mice
```

Step five: Access the device from another container

While the initial container remains running to keep the USB device operational, you can access the attached device from another container. For example:

1. Start a new container with the attached device.

   ```
   $ docker run --rm -it --device "/dev/input/event0" alpine
   ```

2. Install a tool like `evtest` to test the emulated keyboard.

```
$ apk add evtest
$ evtest /dev/input/event0
```

3. Interact with the device, and observe the output.
   Example output:

```
Input driver version is 1.0.1
Input device ID: bus 0x3 vendor 0x0 product 0x0 version 0x111
...
Properties:
Testing ... (interrupt to exit)
Event: time 1717575532.881540, type 4 (EV_MSC), code 4 (MSC_SCAN), value 7001e
Event: time 1717575532.881540, type 1 (EV_KEY), code 2 (KEY_1), value 1
Event: time 1717575532.881540, -------------- SYN_REPORT ------------
...
```

> The initial container must remain running to maintain the connection to the USB device. Exiting the container will stop the device from working.

□□□ #2
□ □□□ □□□ 2026-10-31 19:12:10 UTC
□ □□□ □□□ 2026-10-31 19:12:12 UTC