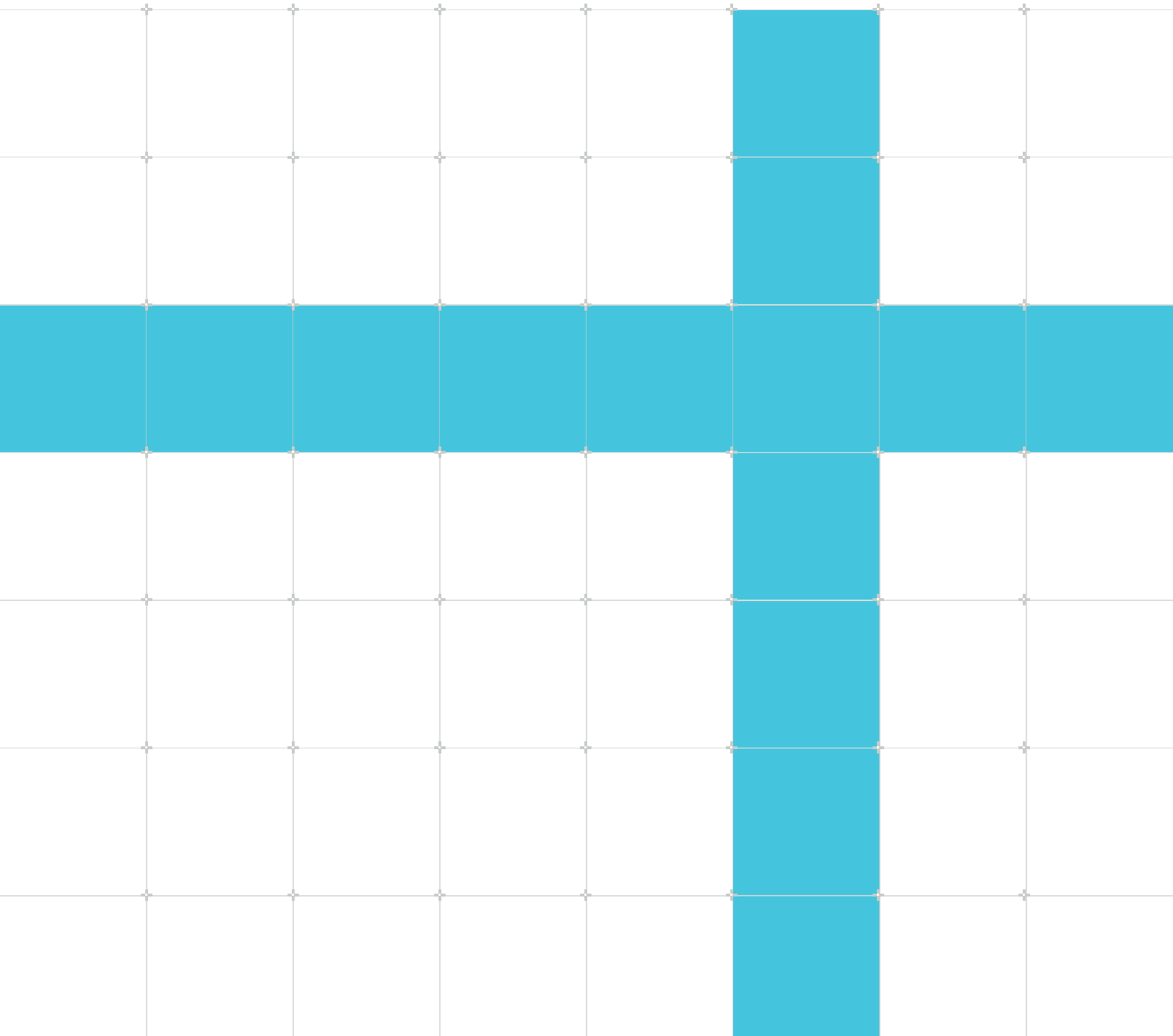
Arm Virtual Hardware Features

Version 1.0

Knowledge Base Article

Issue 01
107675_0100_01_en



Arm Virtual Hardware Features

Knowledge Base Article

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	21 September 2022	Non-Confidential	First release

Proprietary Notice

and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

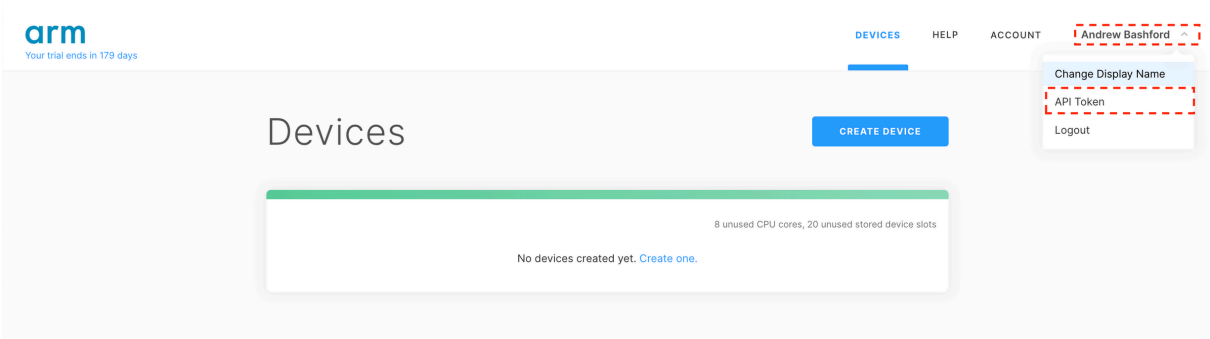
Contents

1. Generating an AVH API Token

This article demonstrates how to create and retrieve your API Token. This token can be used to authenticate your user for API access. Make sure to keep it secure. You can only have 1 token per user at any one time.

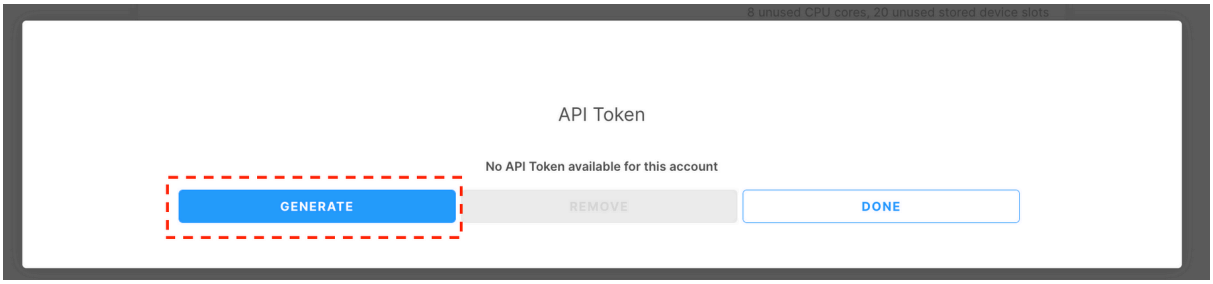
1. Log in to AVH with your Arm user account.
2. To retrieve your API Token, click on your name at the top right corner. Then click “API Token”

Figure 1-1: API token



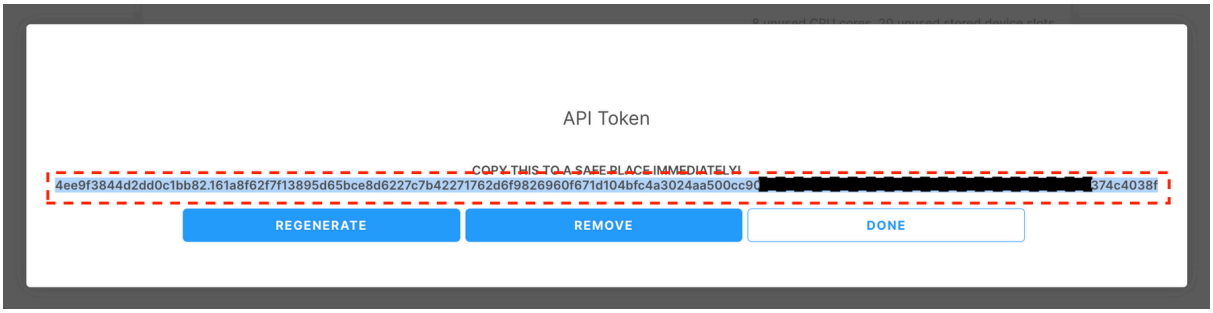
3. Click “GENERATE” this will generate your API Token.

Figure 1-2: Generate API token



4. You can now see your API Token. You will not be shown this token again, copy it and put it somewhere secure to use. Click “DONE”

Figure 1-3: Copy API token

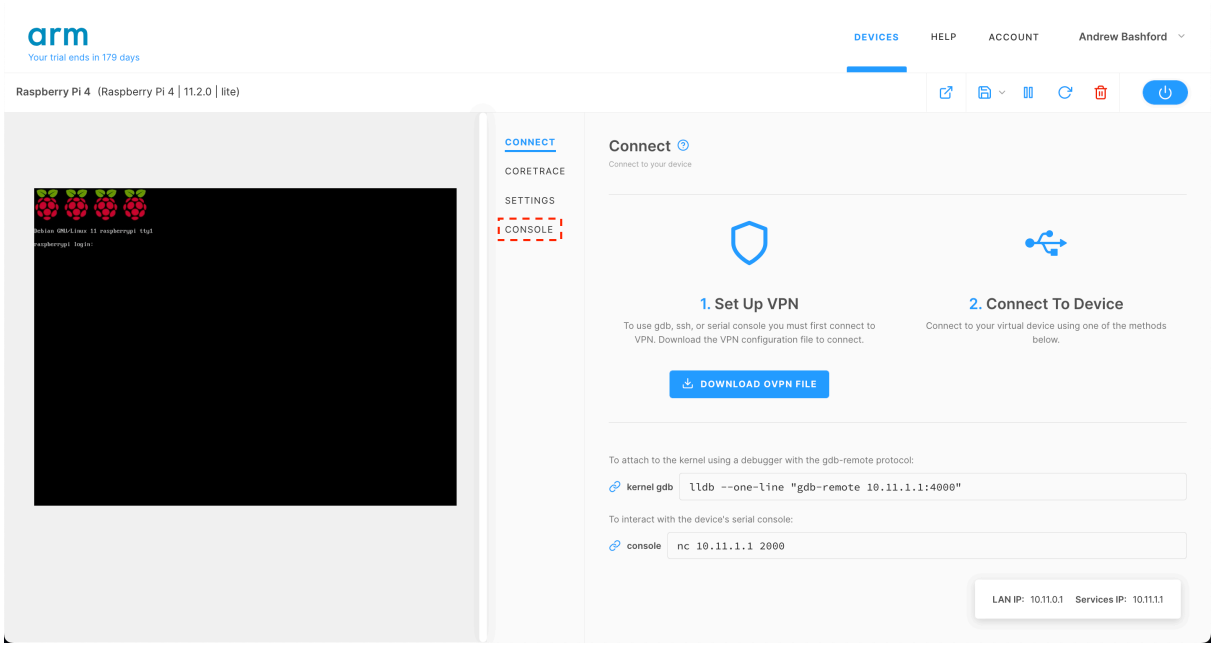


2. Running Docker

This tutorial will demonstrate running Docker on a Raspberry Pi 4.

1. Log in to AVH.
2. Select your Raspberry Pi Device. If you haven't got one you can use the Quickstart for AVH (Raspberry Pi4) to create a Raspberry Pi Device.
3. Select CONSOLE tab.

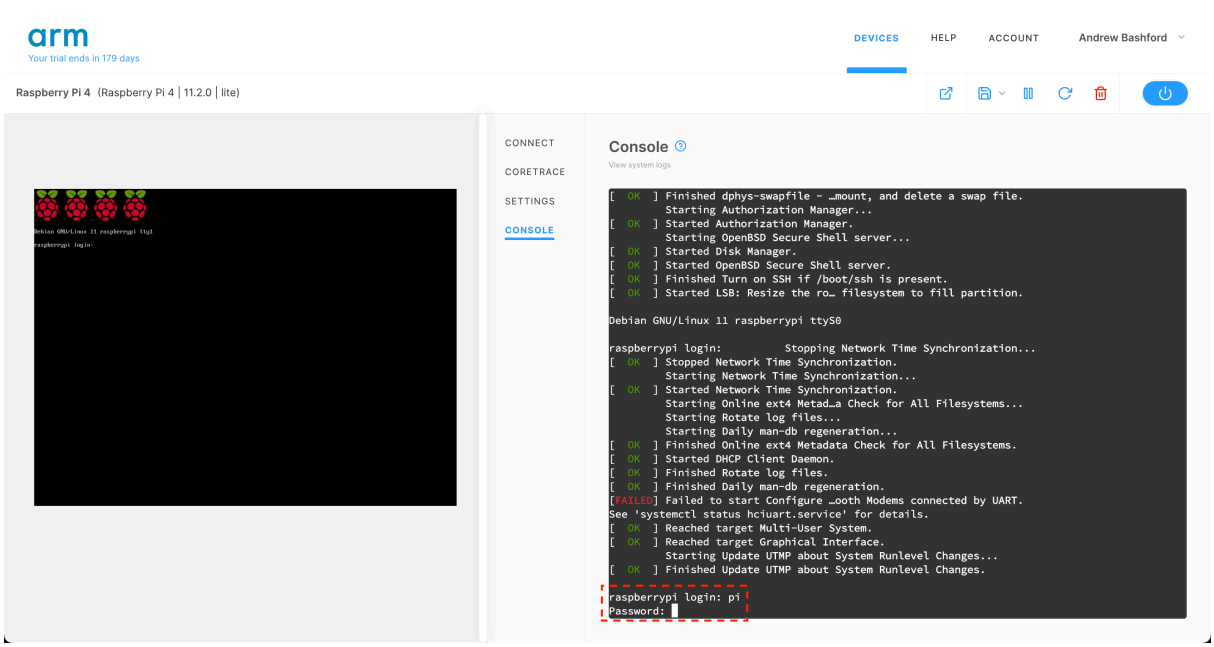
Figure 2-1: Select CONSOLE tab



4. Log in with the user: `pi` and password: `raspberry`

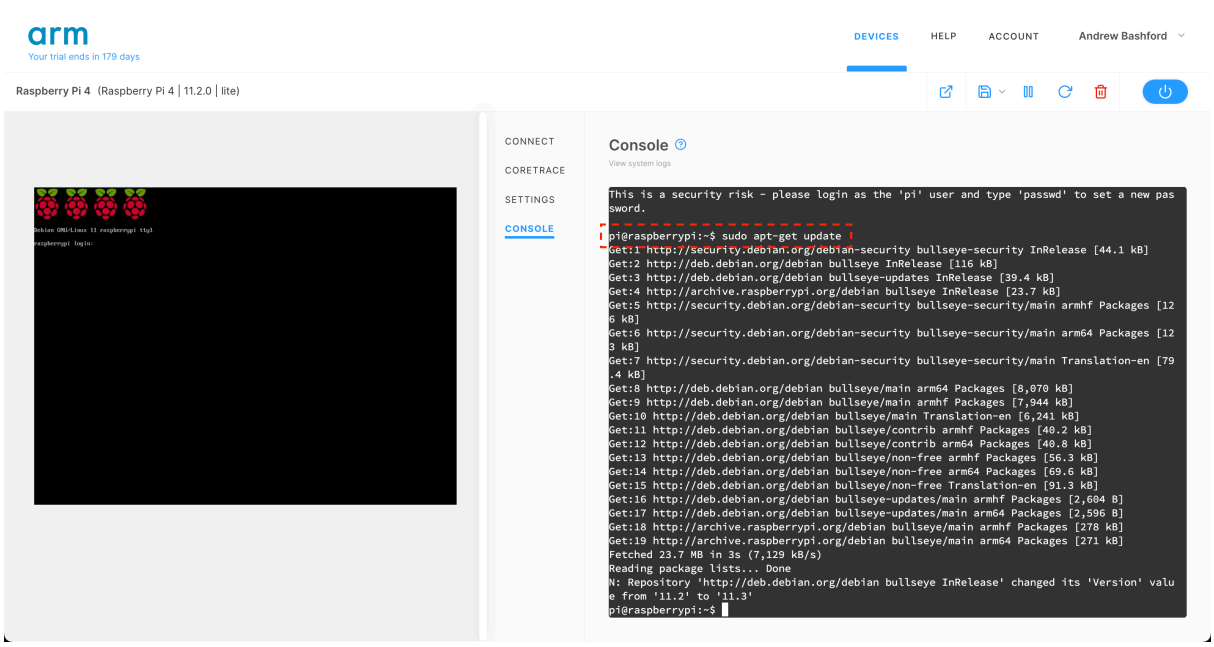
   If it is not prompting you for the user, press your `Return` key.

Figure 2-2: Log in



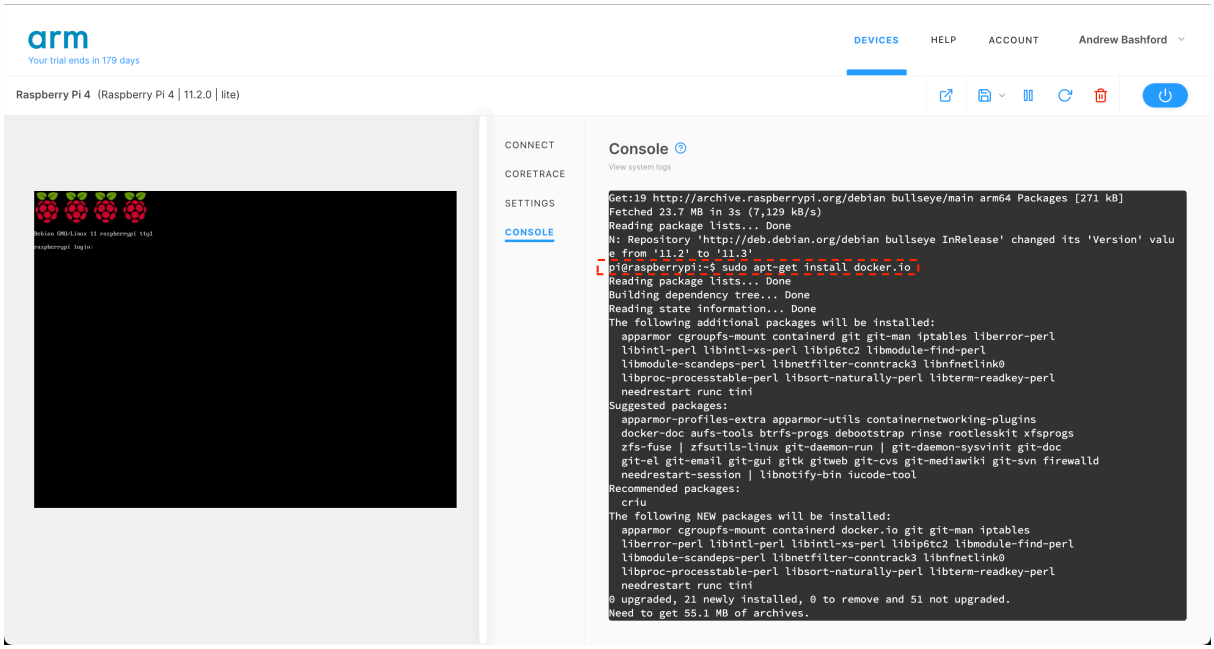
5. Type `sudo apt-get update` then press `return`.

Figure 2-3: sudo apt-get update



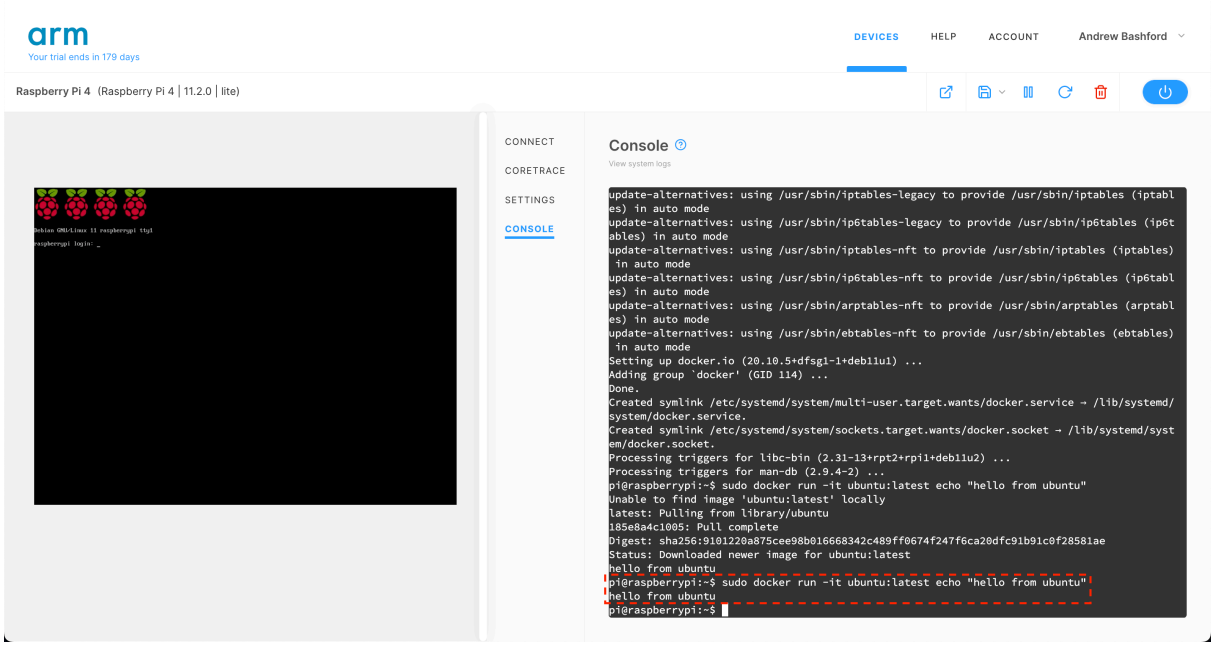
6. Now type `sudo apt-get install -y jq docker.io` then `return`.

Figure 2-4: sudo apt-get install -y jq docker.io



7. Try Docker, type `sudo docker run -it ubuntu:latest echo "hello from ubuntu"` then `return`.

Figure 2-5: Try Docker



3. Service IP and LAN IP

This article describes the Service IP and LAN IP.

3.1 Services IP

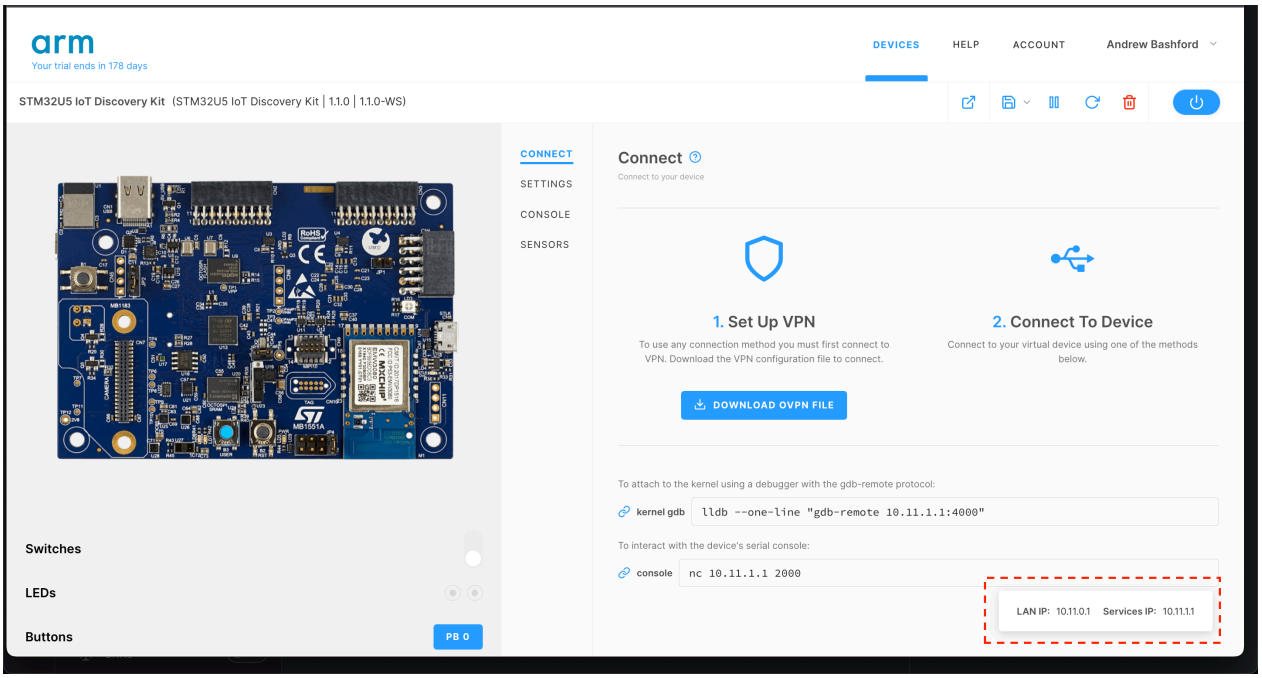
The services IP is a network address used by the Virtual Hardware manager. Virtual services such as serial port (console) pass-through or vMMIO use it. The service IP is used by the Virtual Hardware infrastructure.

3.2 LAN IP

The LAN IP is a network address used by programs running inside the Virtual Hardware. For instance, if there is a web server or SSH server running on the virtual device, it will use the LAN IP.

Both can be found on either the CONNECT or SETTINGS menus of a device.

Figure 3-1: LAN IP and Services IP



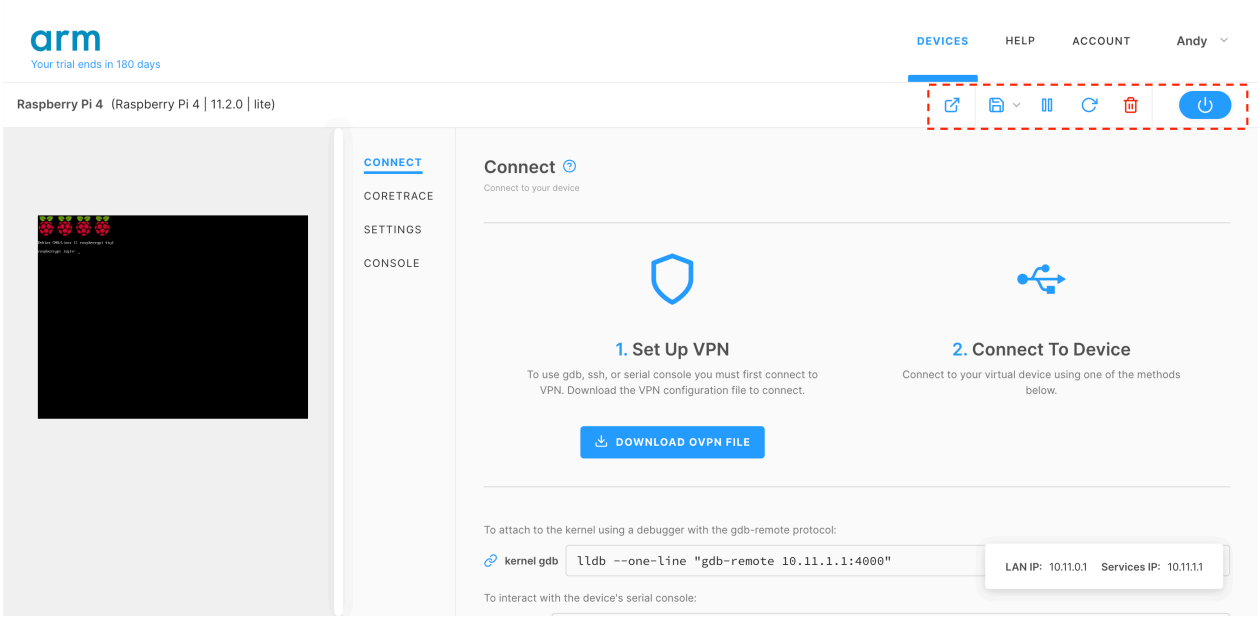
4. Power and Action Buttons

This tutorial will demonstrate how to use your device's action buttons.

4.1 Power and Action Buttons

Power and action buttons can be found at the top right when you are in a Device.

Figure 4-1: Power and action buttons



4.2 Power On and Off

Powering the virtual device off is like powering off a physical device. All applications are shut down, and information about the current state of the device is not preserved. When a device is in the Off

state, it becomes Stored, and it no longer occupies CPU cores. When you are not using the devices it is best practice to turn them off.

Figure 4-2: Power On / Off button

4.3 Pause

Unlike a physical device, your virtual device can be paused. This action suspends device activity but does not shut down the device. Simply click to pause, and click again to unpause.

Figure 4-3: Pause button

4.4 Restart

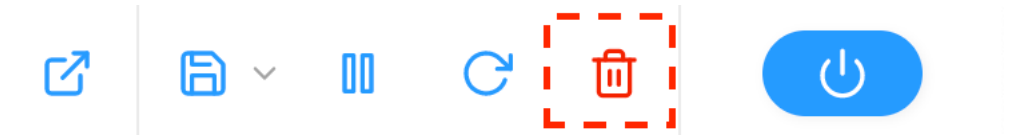
The restart button is akin to rebooting a physical device. It's the same as powering the device off and on again.

Figure 4-4: Restart button

4.5 Delete

Deleting a device will permanently delete the device along with all of its snapshots.

Figure 4-5: Delete button



5. Connect to your virtual devices

You can connect to your virtual devices via a few methods.

5.1 Connect via VPN

For security, some connections require that you first connect to the virtual device via VPN. Just download the provided configuration file, install it, and you're ready to go.

To connect to VPN, you will need to have a VPN client installed. If you don't already have a VPN client, we recommend an OpenVPN client. Once your VPN client is installed, you can simply download the OpenVPN configuration file from the Connect tab and install it to configure your VPN.

Important Notes

- VPN will only establish connection if you have at least one device in your project that is in the on state. If all your devices are off, or if you have no devices, the VPN will not establish connection.
- Running other VPNs may interfere with your VPN connection. If you are having trouble connecting to the VPN and are already running your own VPN, you may need to disable the other VPN and then reconnect to the VPN.

Troubleshooting Tips

- If you're having trouble connecting, try re-installing your VPN Profile.
- Make sure you're connecting to the right profile for your project, and that at least one device in your project is On.
- Close any other VPNs.
- Check that your network doesn't have a firewall preventing your connection. Learn more about configuring Windows Defender Firewall.

5.2 Connect via GDB

You can connect to the stub via GDB with the address and port for your particular virtual device, located at the bottom of the virtual device page.

Make sure to use the correct version of GDB for your device. 32-bit devices, including Cortex-M based ones, require 32-bit Arm GDB, and 64-bit devices require a 64-bit one.

You can also use other debuggers compatible with the GDB protocol, for instance lldb and IDA debugger.

5.3 Connect via Serial Console

If you prefer to view the device's serial console input / output in your local console program, such as Terminal, you can copy and paste the Console link from the Connect tab to your local console app. This can be convenient for enabling advanced console features, such as search.

5.4 Connect via other methods

Some virtual devices also support connect methods such as Iris and Cadi.

6. Sensors

Test devices in real world conditions by configuring a wide variety of environmental sensors and controls.

6.1 Environment

We've added new realism to our virtual devices with an array of new environmental sensors. While you can't hold a cloud device in your hands, you can still make the device behave as if you can.

- Configure the ambient light level (in lux)
- Configure the atmospheric pressure (in hectopascals)
- Configure the temperature (degrees Celsius) and humidity (% RH)
- Simulate presence of objects in front of a proximity sensor (distance in millimeters)

6.2 Motion and position

Simulate your device in motion as it moves around the virtual world.

- Set instantaneous accelerometer readings readings (3-axis, in meters/second^2)
- Set instantaneous gyroscope readings (3-axis, in radians/second)
- Set instantaneous magnetic compass readings (3-axis, in microteslas)

7. Console

Console provides convenient access to the virtual device's console.

You can use the Console to quickly see system and kernel logs and to quickly run commands without needing to connect over various methods.

The console can sometimes be flooded with system and kernel logs, making it difficult to use interactively. If you're using the console interactively, you may want to run `dmesg -n 1` to prevent all messages, except emergency (panic) messages. Run `dmesg -n 4` to restore the default behavior, and run `dmesg -n 7` to log all messages to the console.

8. CoreTrace

CoreTrace is our system call capture tool that offers a quick way to understand a program's behavior.

8.1 What is CoreTrace?

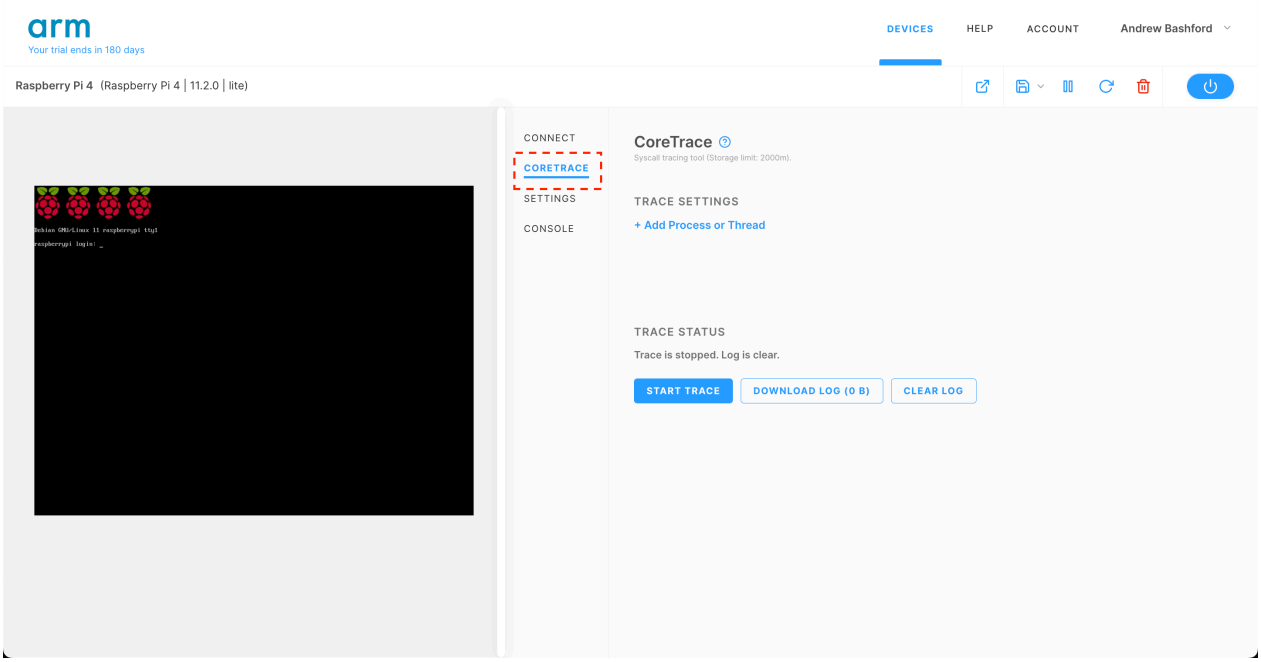
You can trace system calls using either strace, a standard command-line Linux tool, or our proprietary CoreTrace tool. strace is included in our virtual devices, and it is implemented with ptrace. Applications can employ anti-debugging techniques to detect and prevent ptrace-based tracing. However, these techniques cannot prevent, or even easily detect, hypervisor-based tracing. Additionally, you may often be interested in a particular target. CoreTrace makes it easy to filter by specific processes and threads for more targeted analysis.

Tracing system calls is a dynamic analysis reverse engineering technique that can offer a quick way to understand a program's behavior.

8.2 Setting up CoreTrace

To access CoreTrace, open the CoreTrace tab in the device screen. The CoreTrace UI allows you to start and stop a trace, download the log generated by tracing, and clear the log.

Figure 8-1: CoreTrace tab



8.3 Threads and Processes

By default, CoreTrace traces all threads in the system. This rapidly produces a huge amount of data. Often you'll be interested in a particular target. To apply a filter to the results, click "Add a process or thread" to display the Processes dialog:

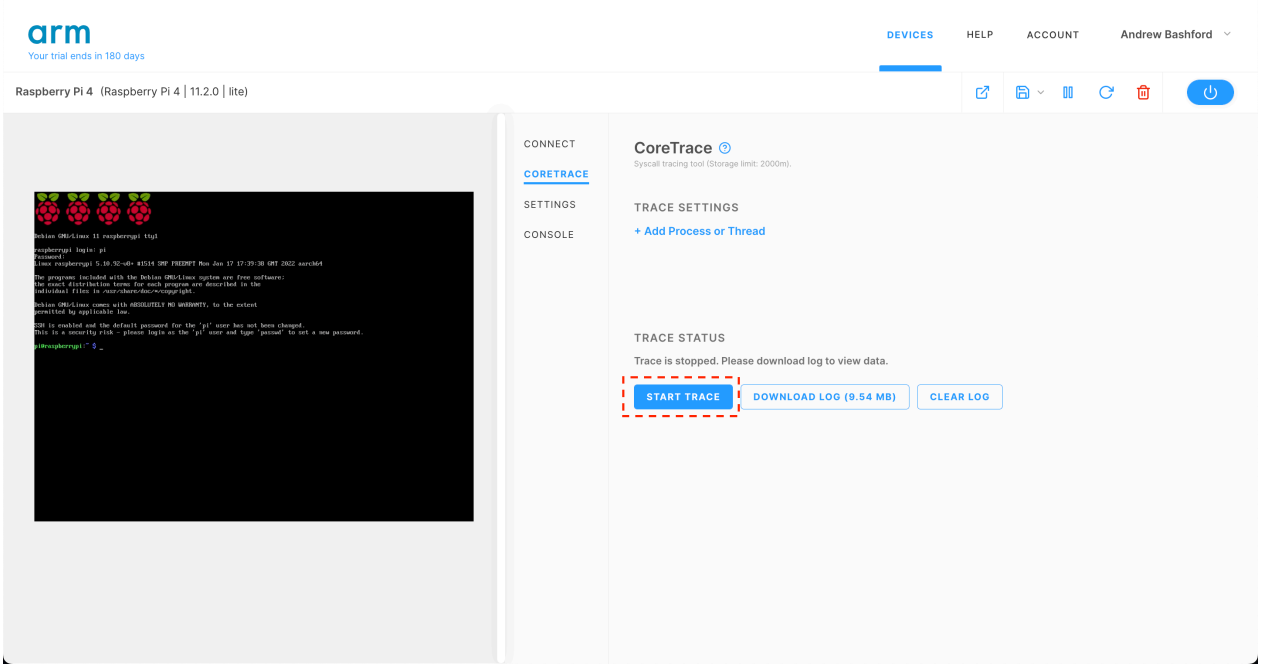
The Processes dialog displays all processes and threads in the system. To examine the threads inside a process, click the "THREADS" button in the process' row.

To add a filter, click the "ADD" button in the process' or thread's row. Alternatively, specify a filter manually. Trace will log traces as long as they match at least one filter.

There are often many processes running. To more easily find the processes and threads you're interesting in, click the magnifying glass in the top-right corner of the dialog and type a phrase. Only rows that contain the phrase will be displayed.

Then, you are ready to click Start Trace.

Figure 8-2: Start Trace



8.4 Understanding the results

After you have captured the trace (or during the capture) you can download the log file. Each line of the log will look like this:

```
1<1> [00248.864651618] ffffff806401e040-0/337:sensors@2.1-ser.379/ @00000070efc0b834
 read ( fd: 5, buf: 0x6e5f6ec980, count: 4 ) ... @[ 0000006e5f6f9778
 0000006e5f6f9840 0000006e5f6f948c 0000006e5f6f7f90 0000006e5f6f7b54
 0000006e5f6f7434 00000070efc2088c 00000070efbc0e0c ]
```

or like this:

```
1<1> [00248.864656648] ffffff806401e040-0/337:sensors@2.1-ser.379/
 @00000070efc0b834 ... read ( result: 4, buf: 0x6e5f6ec980 -> [s"001e"] )
```

The fixed line header contains the following information:

```
1<cpu> [time.nsec] threadid-sigid/pid:comm.tid/ @pc
```

Where:

- *cpu* is the processor core the log comes from,
- *time.nsec* is time the entry was captured by the hypervisor,
- *threadid* is the internal kernel thread ID (usually address of a task or thread structure)
- *sigid* is the signal state (if a signal happens, a thread could execute in a different signal state before it's done with the signal, then return to the original signal state),
- *pid* is the process ID (PID of the process on Linux),
- *comm* is the short process name, which may be the original command but may also be set by process itself,
- *tid* is the thread ID (or, PID of the thread),
- *pc* is the PC where the syscall happened in EL0 (userland).

After the header, lines that end with ... are syscall invocations and lines that start with ... are syscall returns. On syscall invocation lines, if the environment permits it, there will be an additional trailer of the form:

```
1@[ lr ret1 ret2 ret3 ... ]
```

This trailer contains the EL0 return stack of the function that invoked the syscall.