Arm Virtual Hardware

Version 2.0

User Guide

Non-Confidential

Copyright © 2022 Arm Limited (or its affiliates).
All rights reserved.

Issue 02
107660_0200_02_en

Arm Virtual Hardware

User Guide

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	21 September 2022	Non-Confidential	First release
0200-02	1 December 2022	Non-Confidential	Combined all AVH documentation into a single guide.

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND

REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws 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.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at https://www.arm.com/company/policies/trademarks.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.

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.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Contents

1. Getting started.... 9
1.1 Quickstart.... 9
1.1.1 Quickstart for Raspberry Pi 4.... 9
1.1.2 Quickstart for STM32U5 IoT Discovery Kit.... 12
1.1.3 Quickstart for iMX8m Arm Cortex Complex.... 18
1.1.4 Quickstart for the API Docs.... 23
1.2 Supported Devices and Components.... 30
1.2.1 Supported Hardware Components for the Raspberry Pi 4.... 30
1.2.2 Supported Hardware Components for the STM32U5.... 31
1.2.3 Supported Hardware Components for the i.M X8M.... 34
1.3 Next Steps.... 38
1.3.1 Taking and restoring snapshots.... 38
1.3.2 Quick Connect.... 43
1.3.3 Connecting to the VPN.... 50
1.3.4 Connect via SSH.... 59
1.3.5 Connect to the Console.... 61

2. Examples.... 64
2.1 Capture Input from the Console.... 64
2.1.1 Retrieving Character Input from the User.... 64
2.1.2 Detecting When the User Presses Return.... 64
2.1.3 Modify How the Terminal Handles “newline”.... 64
2.2 iMX8m Arm Cortex Complex Examples.... 65
2.2.1 Blink the Led on the iMX8m Arm Cortex Complex device.... 65
2.3 STM32U5 IoT Discovery Kit Examples.... 66
2.3.1 STM32CubeU5 TFM example.... 66
2.3.2 STM32CubeU5 WiFiBasics example.... 75
2.4 Raspberry Pi 4 Example.... 80
2.4.1 Raspberry Pi vMMIO example.... 80
2.5 API Examples.... 88
2.5.1 Upload Custom Firmware with the Python API.... 89
2.5.2 Python and Javascript API Examples.... 93

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2.5.3 GitHub Actions Example - Building Firmware, Uploading, and Testing.....93

3. Features.....99

3.1 Generating an AVH API Token.....99
3.2 Running Docker.....100
3.3 Service IP and LAN IP.....103
3.3.1 Services IP.....103
3.3.2 LAN IP.....103
3.4 Power and Action Buttons.....104
3.4.1 Power and Action Buttons.....105
3.4.2 Power On and Off.....105
3.4.3 Pause.....106
3.4.4 Restart.....106
3.4.5 Delete.....106
3.5 Connect to your virtual devices.....106
3.5.1 Connect via VPN.....107
3.5.2 Connect via GDB.....107
3.5.3 Connect via Serial Console.....107
3.5.4 Connect via other methods.....108
3.6 Sensors.....108
3.6.1 Environment.....108
3.6.2 Motion and position.....108
3.7 Console.....108
3.8 CoreTrace.....109
3.8.1 What is CoreTrace?.....109
3.8.2 Setting up CoreTrace.....109
3.8.3 Threads and Processes.....110
3.8.4 Understanding the results.....110
3.9 Connect to Wi-Fi on the Raspberry Pi 4.....111
3.10 Communicate Using Bluetooth Low Energy.....122
3.10.1 Use BLE Between Two Raspberry Pi Devices.....123
3.10.2 Set Up the Raspberry Pi 4 BLE Node.....123
3.10.3 Connect to the Node with the Raspberry Pi 4 BLE Master.....136
3.10.4 Use BLE Between a Raspberry Pi 4 Board and an Android Device.....148
3.10.5 Set Up the Raspberry Pi 4 BLE Node.....148
3.10.6 Connect to the Node with the Android BLE Master.....150

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.11 Detect Audio through the Microphone with the STM32U5 Board....160
3.11.1 Build the ADF_AudioSoundDetector Firmware File....161
3.11.2 Boot the Custom Firmware and Detect Audio....167

4. Custom Firmware....170
4.1 Firmware....170
4.1.1 Understanding Firmware on AVH....170
4.1.2 Package Ubuntu Server Firmware for AVH....173
4.1.3 Upload custom firmware using the UI....178
4.1.4 Package Ubuntu Desktop Firmware for AVH....182
4.2 Storage Files....188
4.2.1 Storage Files for Raspberry Pi 4....188
4.2.2 Storage Files for iMX8m Arm Cortex Complex....188
4.2.3 Storage Files for STM32U5 IoT Discovery Kit....189

5. Advanced Features....191
5.1 Play Video Files on Virtual i.M X8M Boards....191
5.1.1 Table of Contents....191
5.1.2 Open the GStreamer Multimedia Application....191
5.1.3 Play Video Using the Web Interface....194
5.1.4 Use Your Webcam....194
5.1.5 Upload a File - Web Interface....195
5.1.6 Link to a URL - Web Interface....196
5.1.7 Play Videos Using API and POST Requests....197
5.1.8 Upload a File - API and POST Requests....197
5.1.9 Link to a URL - API and POST Requests....198
5.2 Use USB/IP to Attach Peripherals over the Internet....198
5.2.1 Table of Contents....198
5.2.2 Use Your Local Linux Computer as the USB/IP Host....199
5.2.3 Set Up the USB/IP Daemon on Your Computer....199
5.2.4 Attach a Physical Device to the Virtual Board Client....205
5.2.5 (Optionally) Close the Connection When Finished....209
5.2.6 Use a Virtual STM32U5 Board as the USB/IP Host....212
5.2.7 Build USB Firmware Files with the STM32CubeIDE Tool....212
5.2.8 Attach a Virtual USB Peripheral to a Virtual Raspberry Pi 4 Board....218
5.2.9 Attach a Virtual USB Peripheral to a Physical Computer Using a VPN Connection....226
5.2.10 Attach a Virtual USB Peripheral to a Physical Computer Using SSH Port Forwarding....234

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.3 vMMIO Introduction.... 235
5.3.1 Background.... 236
5.3.2 The CoreIO Library API.... 236
5.3.3 Connecting to a target VM.... 236
5.3.4 Registering a vMMIO Range.... 237
5.3.5 Registering a vMMIO Handler.... 237
5.3.6 Processing Read/Write Events.... 238
5.3.7 Interrupts.... 239
5.3.8 Direct Memory Access (DMA).... 239
5.3.9 Examples.... 240
5.4 Kernel debugging.... 240
5.4.1 Using GDB/LLDB.... 241
5.4.2 Monitor Commands.... 242
5.5 Custom device tree.... 243

6. Troubleshooting and FAQs.... 245
6.1 CPU Cores and Device Slot Limits.... 245
6.2 Using the console interactively.... 247
6.3 FAQ - My VPN keeps disconnecting.... 247
6.4 FAQ - What is a core?.... 247
6.5 Deleting your AVH account.... 247
6.5.1 Deleting your account.... 248
6.6 Updating the Raspberry Pi 4 Kernel.... 250
6.6.1 Table of Contents.... 250
6.6.2 Updating the Raspberry Pi 4 Kernel.... 251
6.6.3 Update Using the Console Command Line.... 251
6.6.4 Update Using the Welcome to Raspberry Pi GUI on the Desktop Firmware.... 252
6.6.5 Identifying Issues with the Device Kernel.... 253
6.6.6 List the Loaded Kernel Modules.... 253
6.6.7 Try to Install and Run Docker.... 254
6.6.8 Fixing the Updated Kernel File.... 256

7. Updates.... 260
7.1 Release 3.10.... 260
7.2 Release 3.11.... 260
7.3 Release 3.14 and 3.14.1.... 261

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1. Getting started

The following articles are available:

- Quickstart
- Supported Devices and Components
- Next Steps

1.1 Quickstart

Quickstart guides:

- Quickstart for Raspberry Pi 4
- Quickstart for STM32U5 IoT Discovery Kit
- Quickstart for iMX8m Arm Cortex Complex
- Quickstart for the API Docs

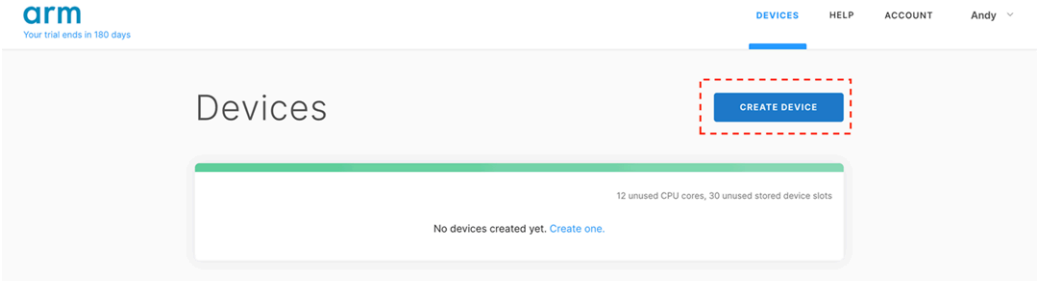
1.1.1 Quickstart for Raspberry Pi 4

This sections shows you how to create a virtual Raspberry Pi 4 device along with some basic functionalities of the device's interface.

1.1.1.1 Creating your device

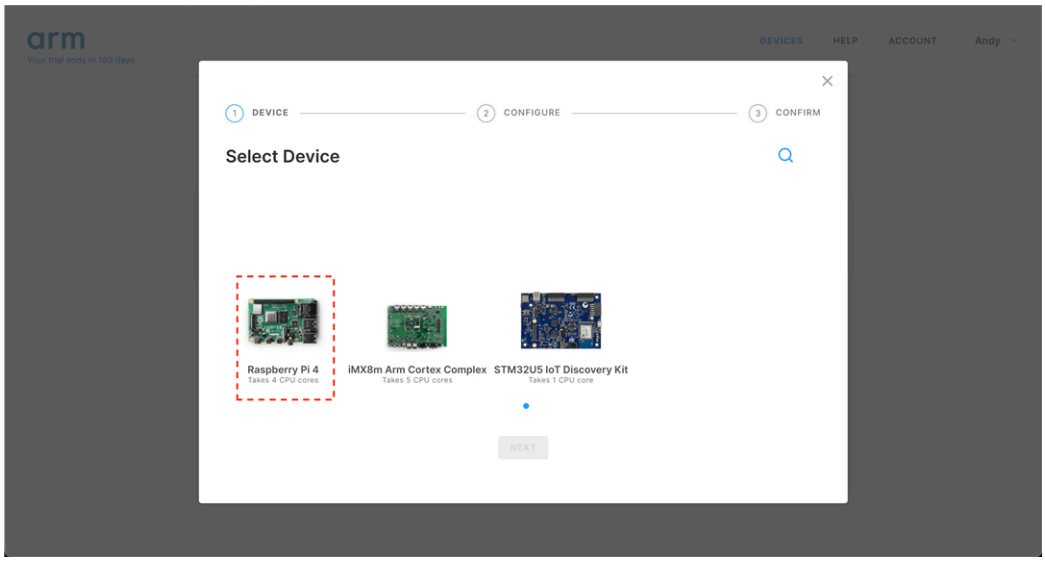
To create your device, do the following:

1. Log in to AVH with your Arm user account at https://app.avh.arm.com/. If you do not have an Arm account you can register for it at Arm Developer.
2. When you reach the AVH Devices page, click CREATE DEVICE.

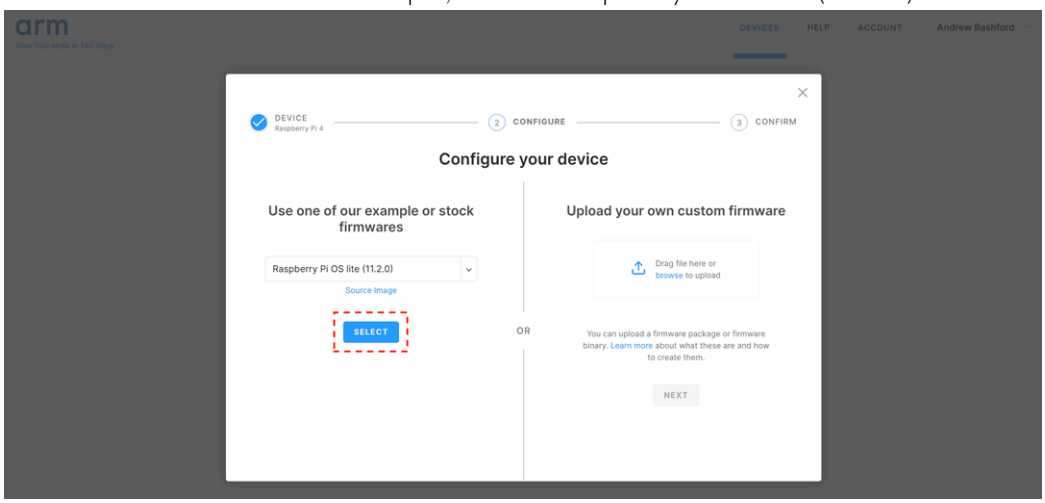


Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. Select the Raspberry Pi 4 device, then click NEXT.

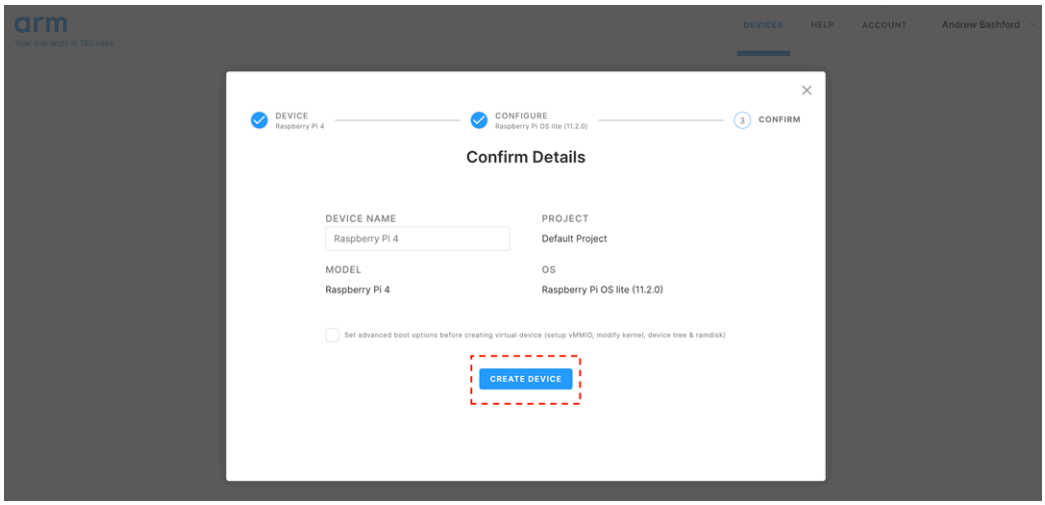


4. On the *Configure Your Device* page, you will see a dropdown with stock firmware options. You can upload your own custom firmware, see Upload custom firmware using the UI for more information. For this example, choose Raspberry Pi OS lite (11.2.0) and click SELECT.

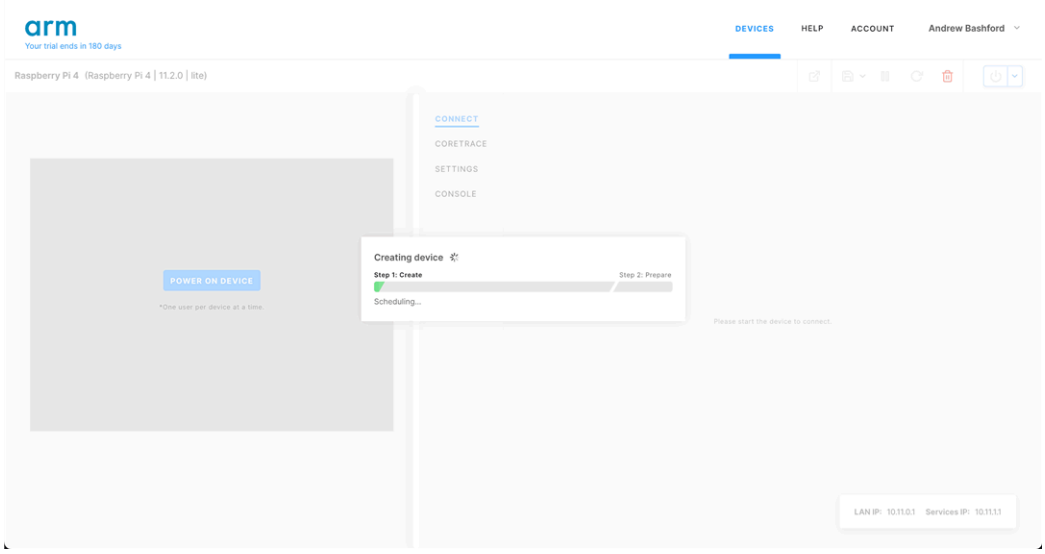


Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5. On the *Confirm Details* page, verify your selections then click CREATE DEVICE. Leave the device name as the default and do not set advanced boot options.

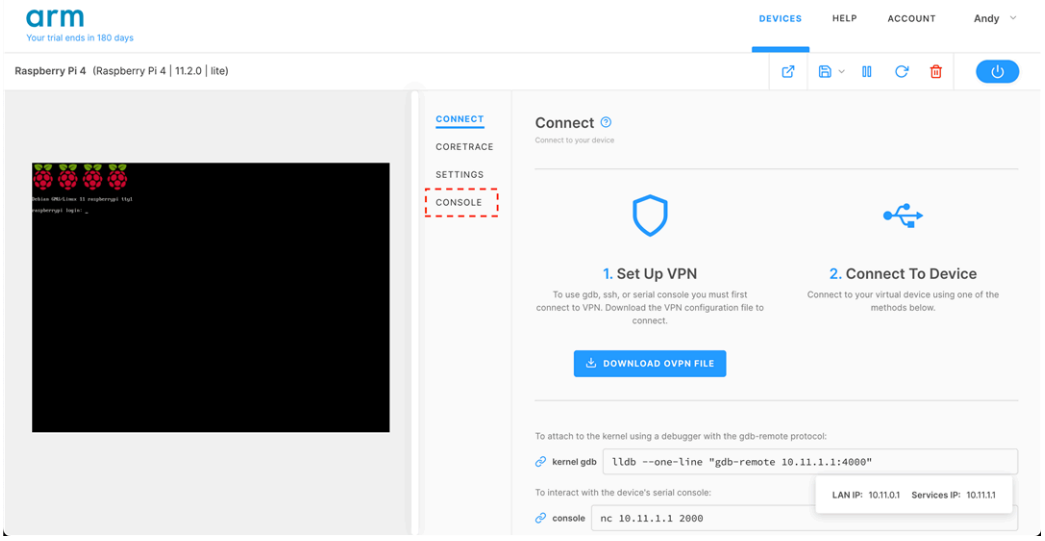


6. Please be patient while the virtualized device loads. It may take a few minutes.

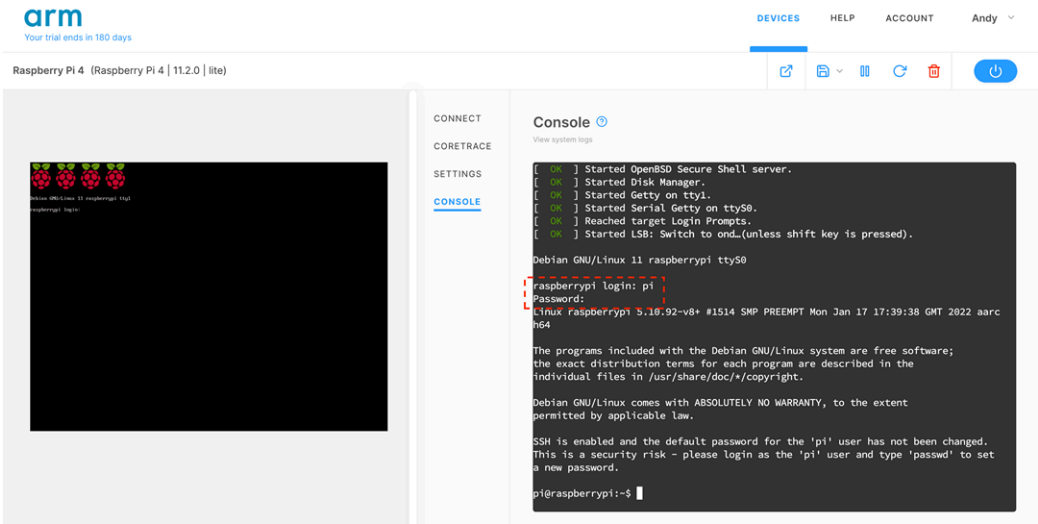


Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7. Your Raspberry Pi device is now ready to go! Next, let's log in. Click CONSOLE in the devices menu.



8. Type in the default username: `pi` and password: `raspberry`, then hit `return`. When you see the display name change to `pi@rasberrypi`, you are logged in. Alternatively, you could use the graphical console on the right to log in.



9. Your virtualized Raspberry Pi 4 device is now ready to use!

You can test out your new Raspberry Pi 4 device by creating and restoring Snapshots in the next tutorial, Taking and restoring snapshots.

1.1.2 Quickstart for STM32U5 IoT Discovery Kit

Set up a STM32U5 IoT Discovery Kit device and run the `IOT_HTTP_WebServer` example.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.1.2.1 Introduction

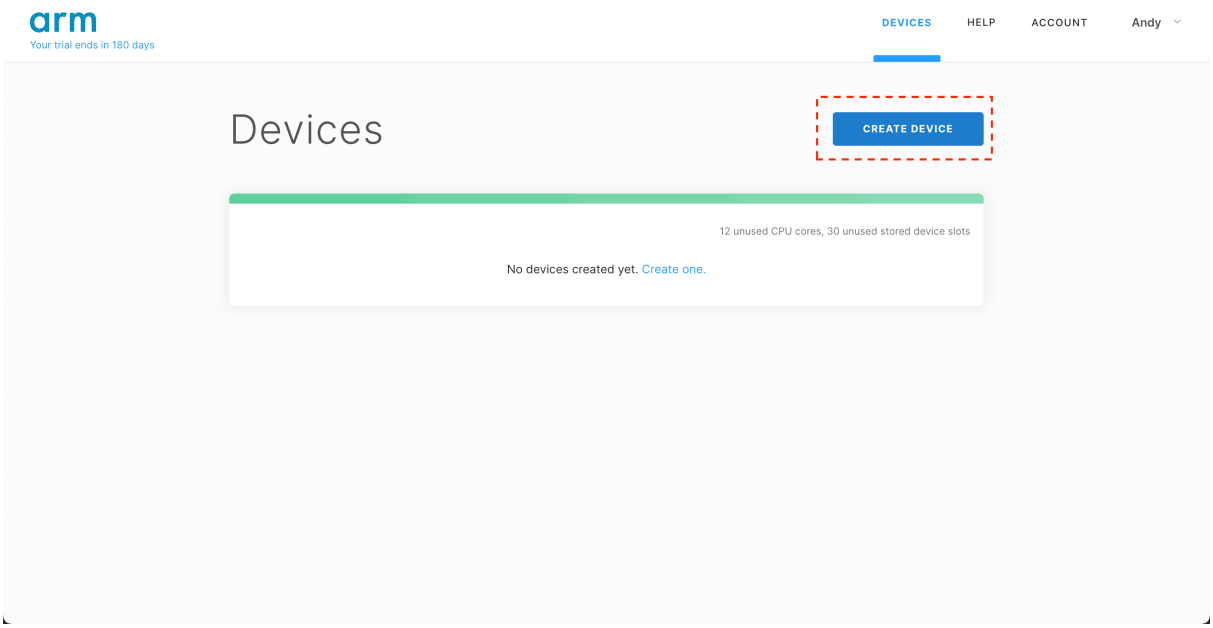
This example demonstrates how to create a STM32U5 IoT Discovery Kit device with an example firmware. In this example kernel component in the archive is an `IOT_HTTP_WebServer.elf` firmware binary produced by STMCubeIDE. Find out more at the ST website.

1.1.2.2 Creating your device

To create your device, do the following:

1. Log in to AVH with your Arm user account. https://avh.arm.com/ If you do not have create an Arm account by registering at https://developer.arm.com/
2. When logged in you land on the AVH Devices page. Click CREATE DEVICE.

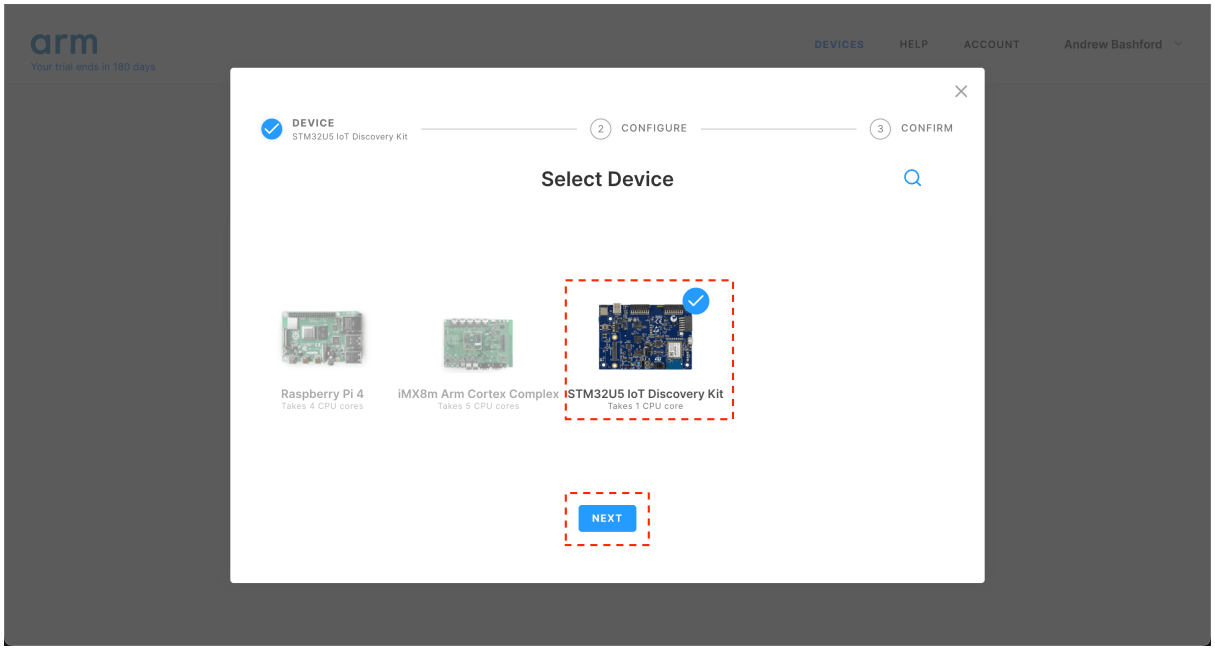
Figure 1-1: Create Device



3. Click on the STM32U5 IoT Discovery Kit to select it. Then click NEXT.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-2: Select STM32U5 IoT Discovery Kit



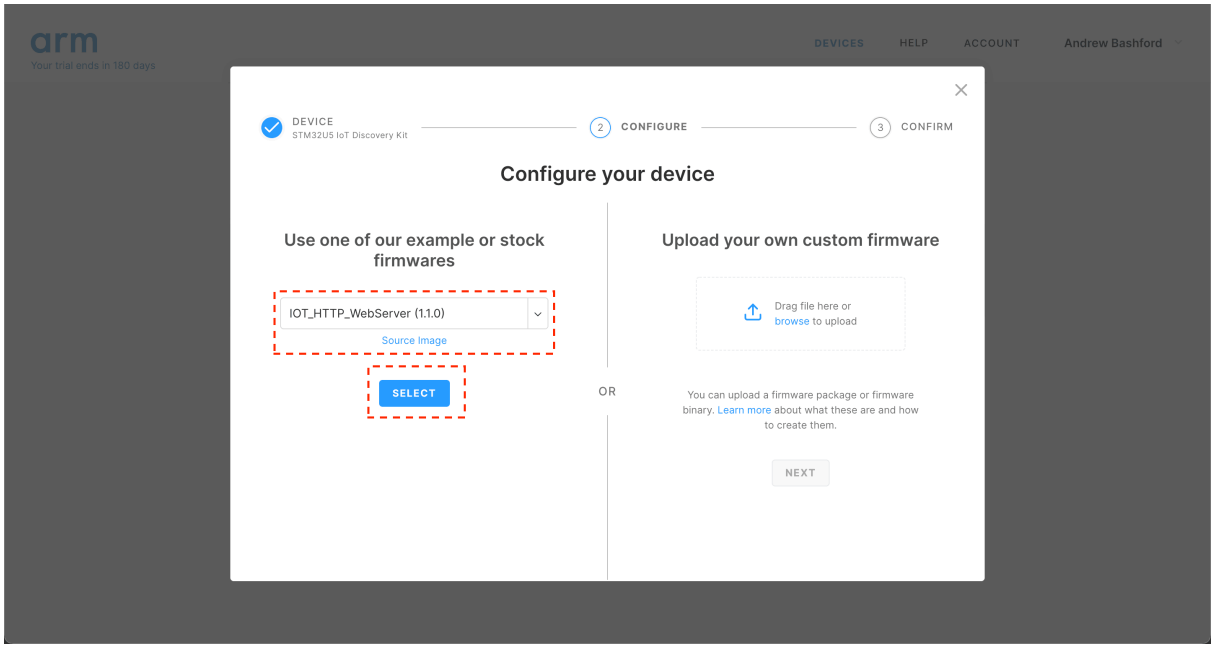
Note

You require 1 free CPU core to build this device or it will be disabled.

4. In this next step, you will see a list of pre-configured examples/stock firmware or you can upload your own custom firmware. In this example, we will use the selection `IOT_HTTP_WebServer (1.1.0)` in the stock firmware section. Click SELECT.

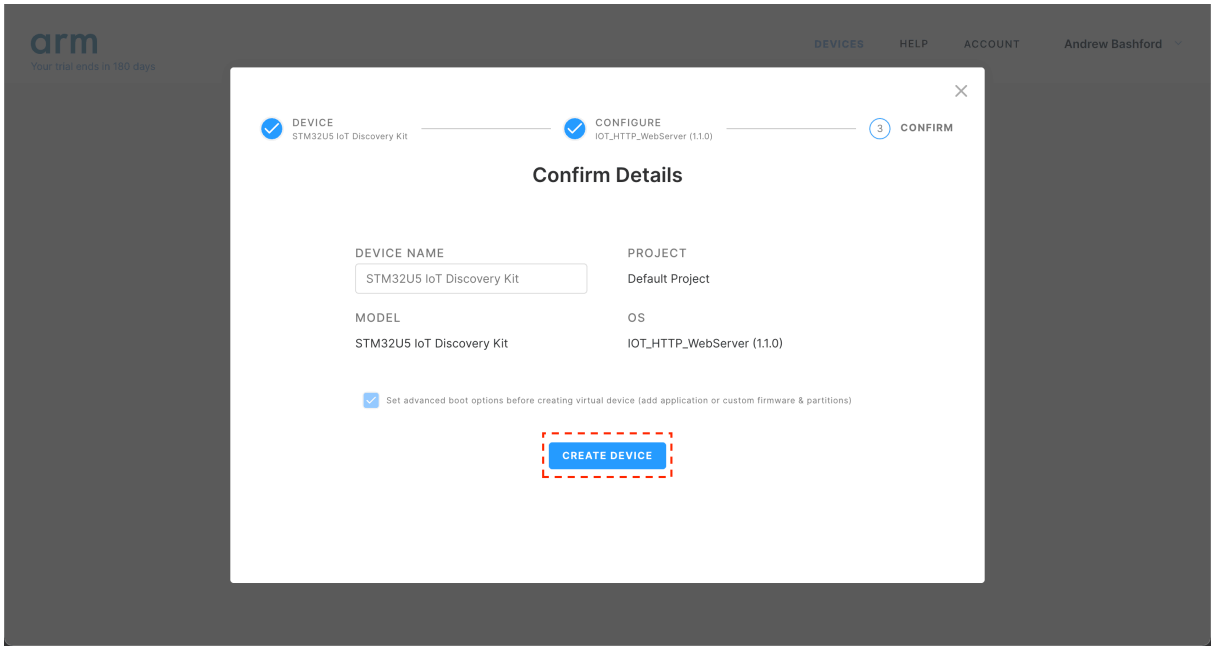
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-3: See a list of pre-configured examples/stock



5. In the confirmation screen, click CREATE DEVICE using the default name and the advance boot options will be checked by default.

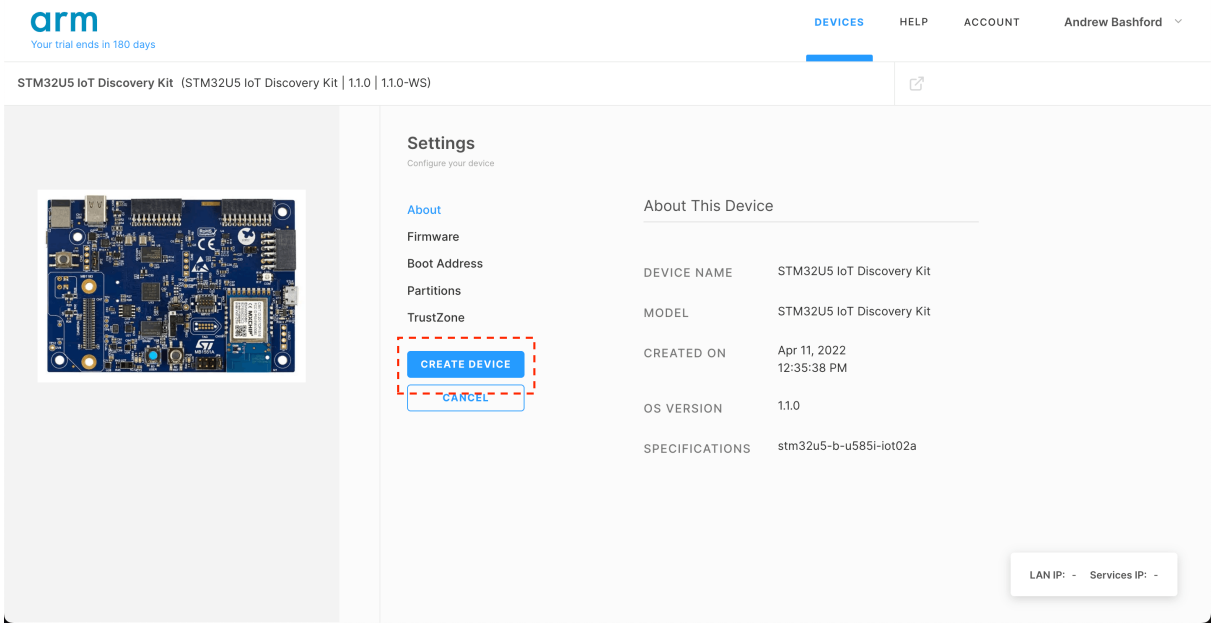
Figure 1-4: Confirmation screen



6. In the advanced boot options, we will just use the default. Click CREATE DEVICE.

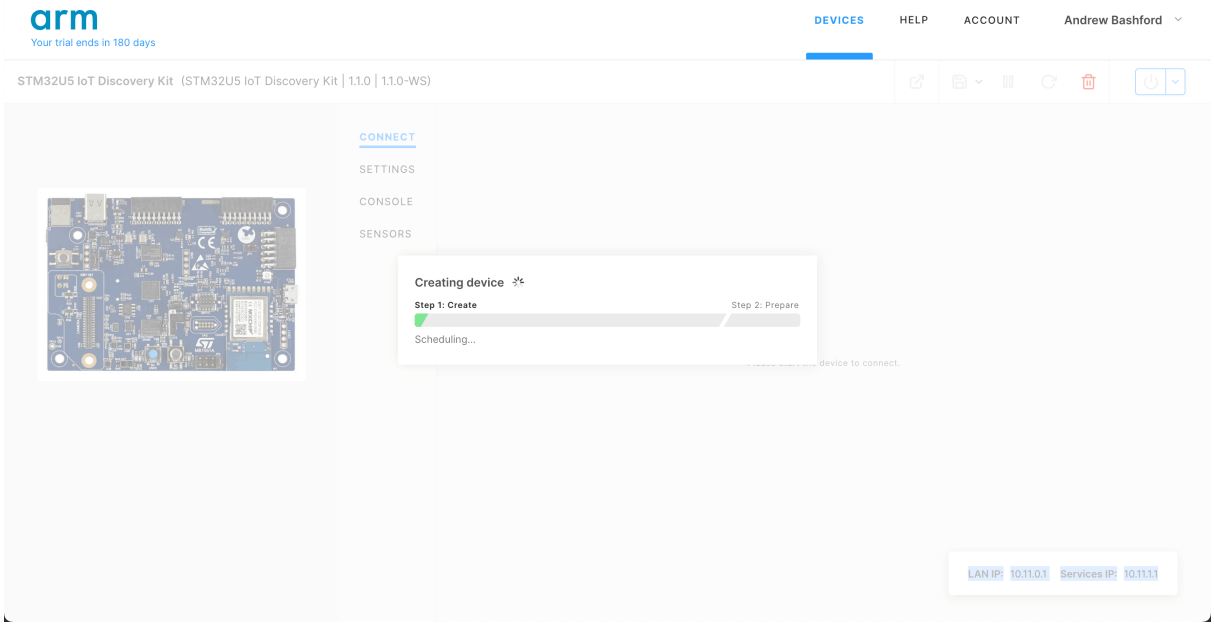
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-5: Advanced boot options



7. The device will then be created. It takes a few moments.

Figure 1-6: Creating devices



8. Your STM32U5 IoT Discovery Kit is now ready to go!

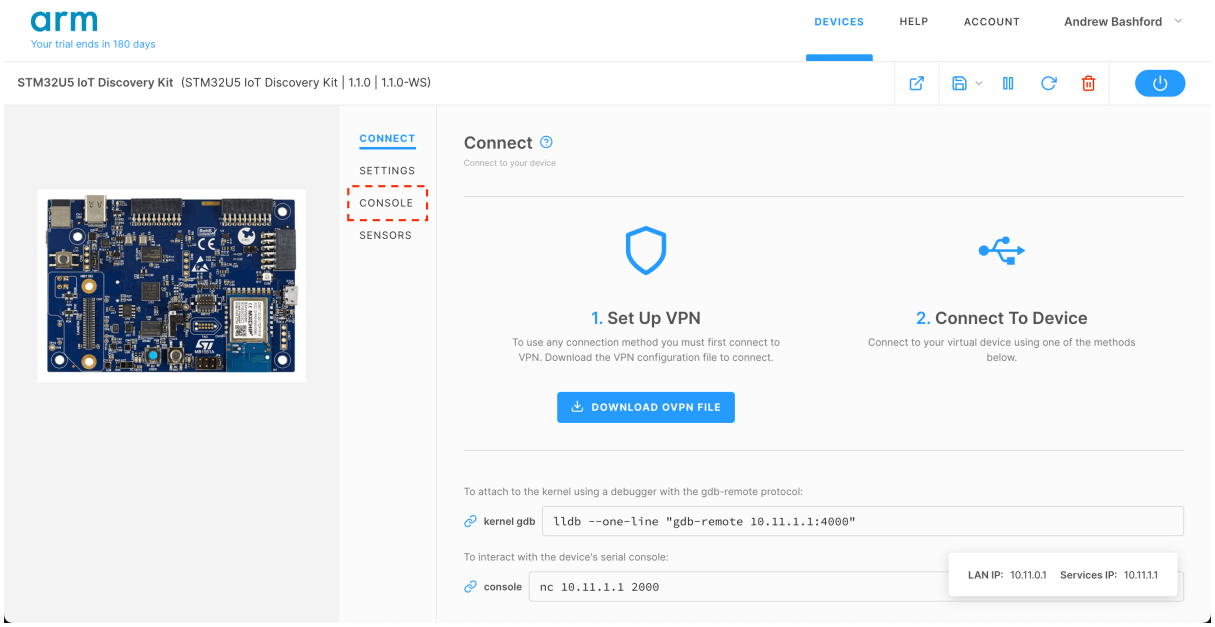
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.1.2.3 Running the example

To run the example, do the following:

1. Click CONSOLE on the devices menu.

Figure 1-7: Running the example



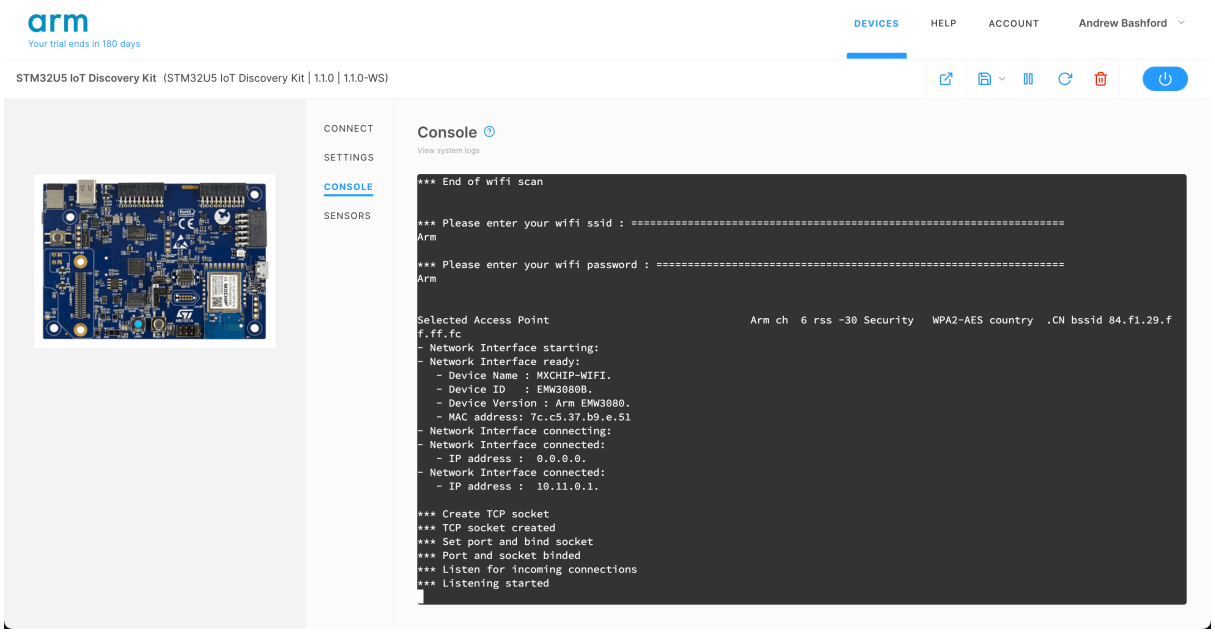
2. Type SSID from the list (`Arm` in this case) and anything as a password (`Arm` in this case).

Note

The SSID and password are case-sensitive.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-8: Type SSID



3. Now the STM32U5 IoT Discovery Kit virtual device is running a sensor web-server available through VPN or other virtual devices (like Raspberry Pi). It is possible to set sensor values using API or from the sensor tab.

1.1.2.4 Next steps

Further reading on the STM board page.

Other STM32U5 IoT Discovery Kit examples:

- STM32CubeU5 WiFiBasics example
- STM32CubeU5 TFM example

1.1.3 Quickstart for iMX8m Arm Cortex Complex

Set up an iMX8m Arm Cortex Complex device and log in to the console.

1.1.3.1 Introduction

This example demonstrates how to create an iMX8m Arm Cortex Complex device and Log in to the Console.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

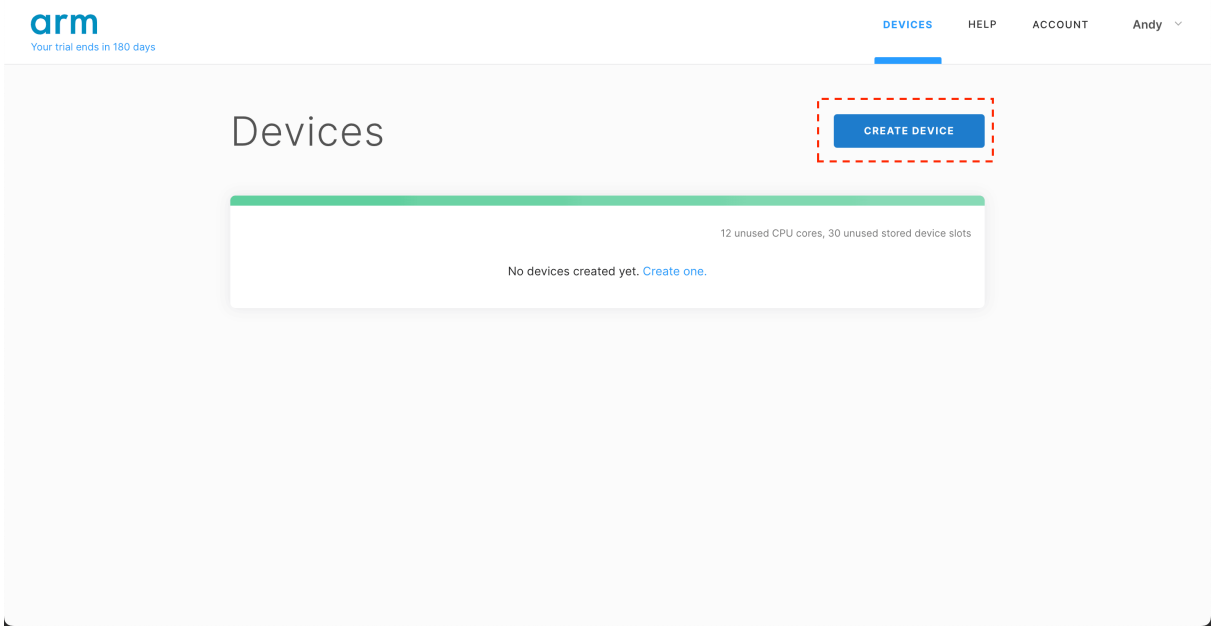
1.1.3.2 Creating your device

To create your device, do the following:

1. Log in to AVH with your Arm user account. https://avh.arm.com/

   If you do not have or need to create an Arm account register at https://developer.arm.com/
2. When logged in you land on the AVH Devices page. Click CREATE DEVICE.

   Figure 1-9: AVH Devices page



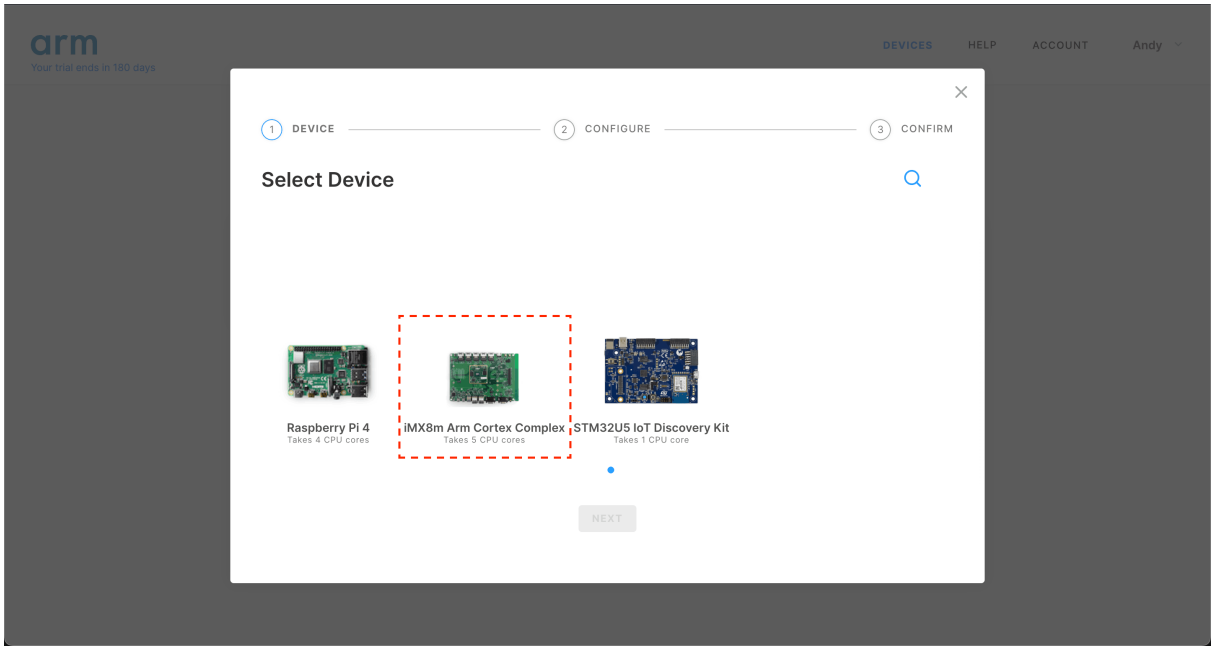
3. Click on the iMX8m Arm Cortex Complex to select it, then click NEXT.

Note

You require 5 free CPU cores to build this device or it will be disabled.

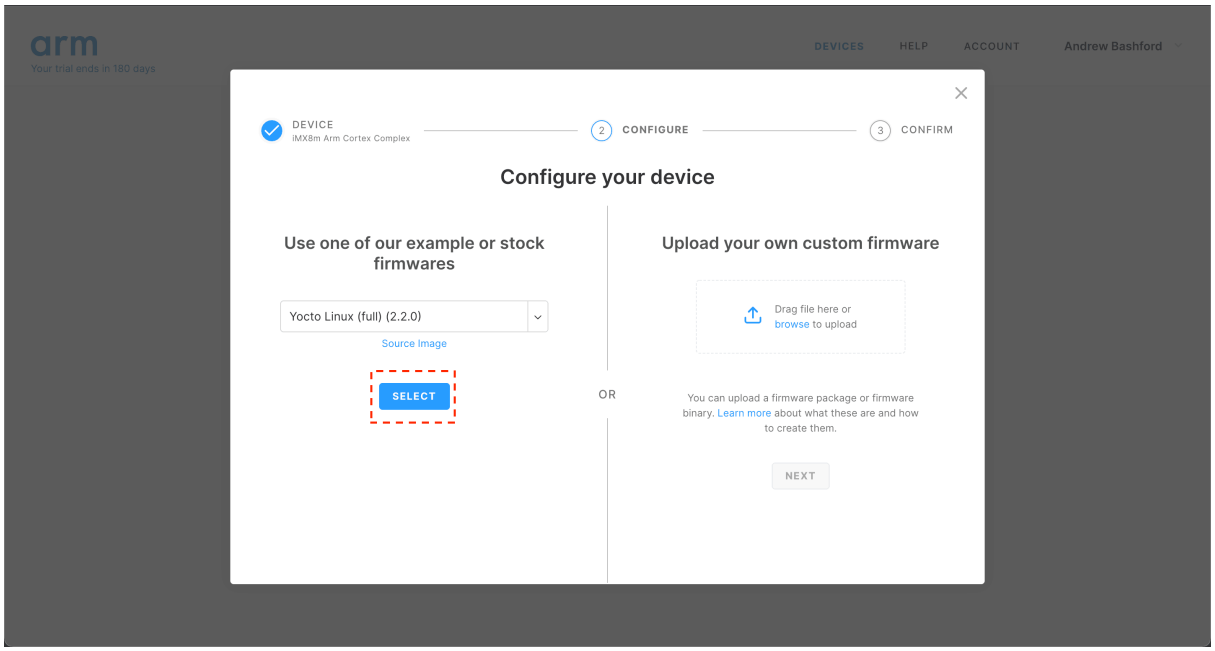
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-10: Select iMX8m Arm Cortex Complex



4. In this next step, you will see a list of pre-configured examples/stock firmware or you can upload your own custom firmware, see Upload custom firmware using the UI for more information. In this example, we will use the default selection `Yocto Linux (2.2.0)` in the stock firmware section. Click SELECT.

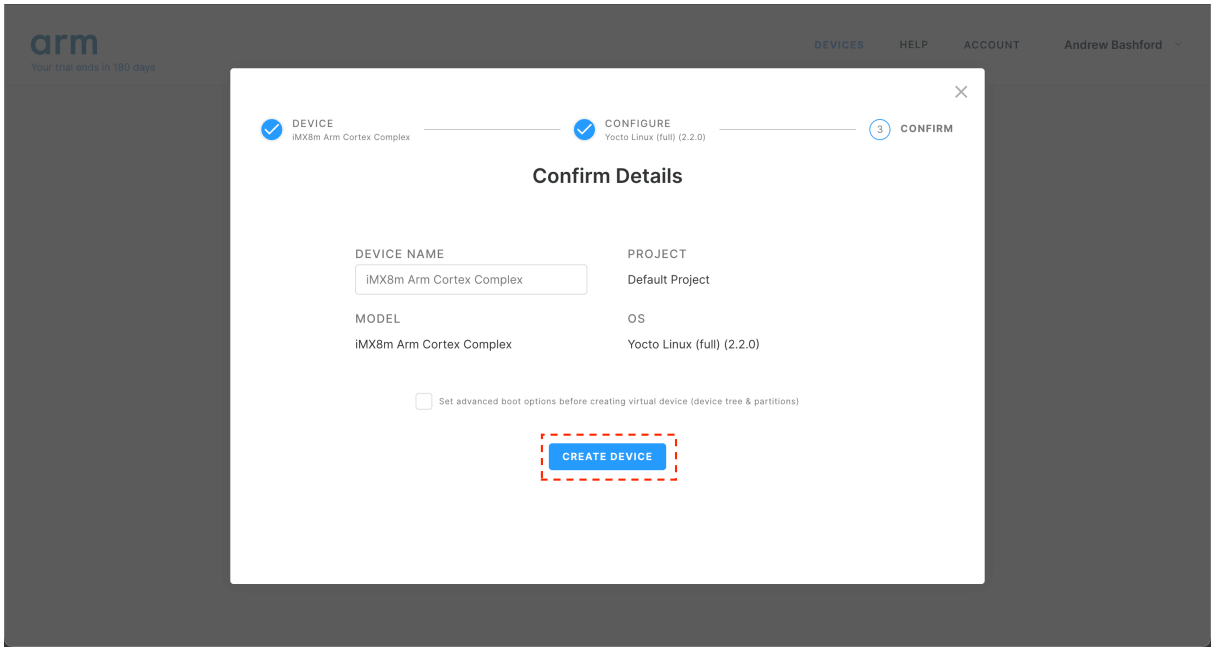
Figure 1-11: Yocto Linux (2.2.0)



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

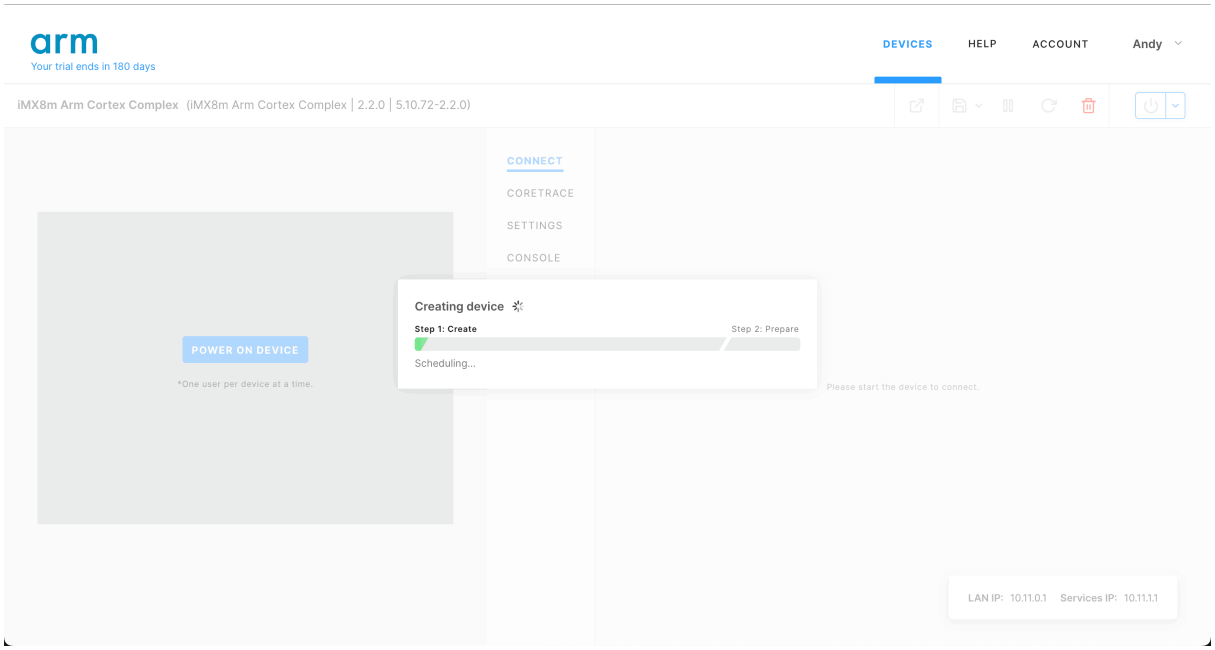
5. In the confirmation screen, click CREATE DEVICE using the default name and the advance boot options unchecked.

Figure 1-12: Confirmation screen



6. The device will then be created. It takes a few moments.

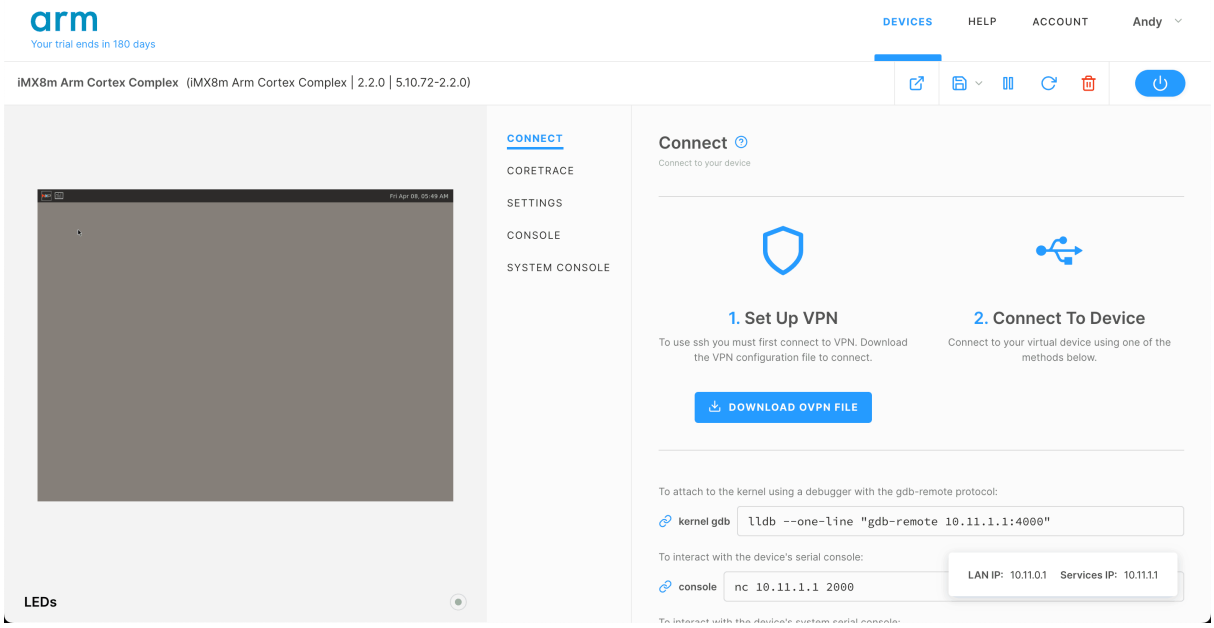
Figure 1-13: Creating device



7. Your iMX8m Arm Cortex Complex is now ready to go!

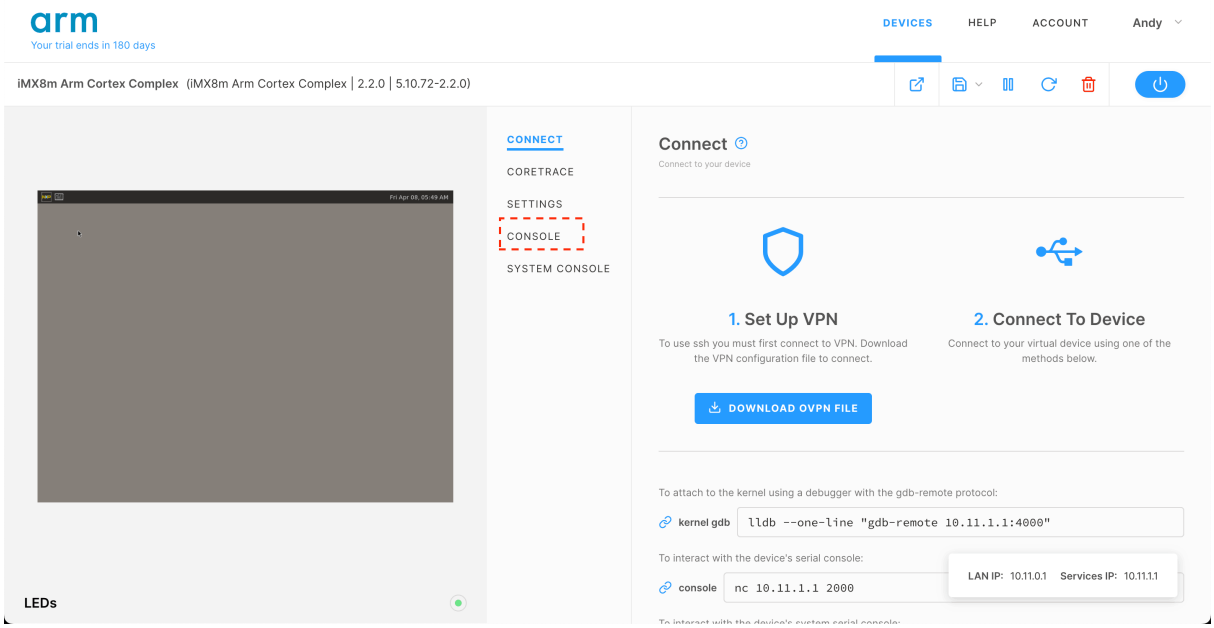
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-14: iMX8m Arm Cortex Complex



8. Next, let's log in. Click CONSOLE on the devices menu.

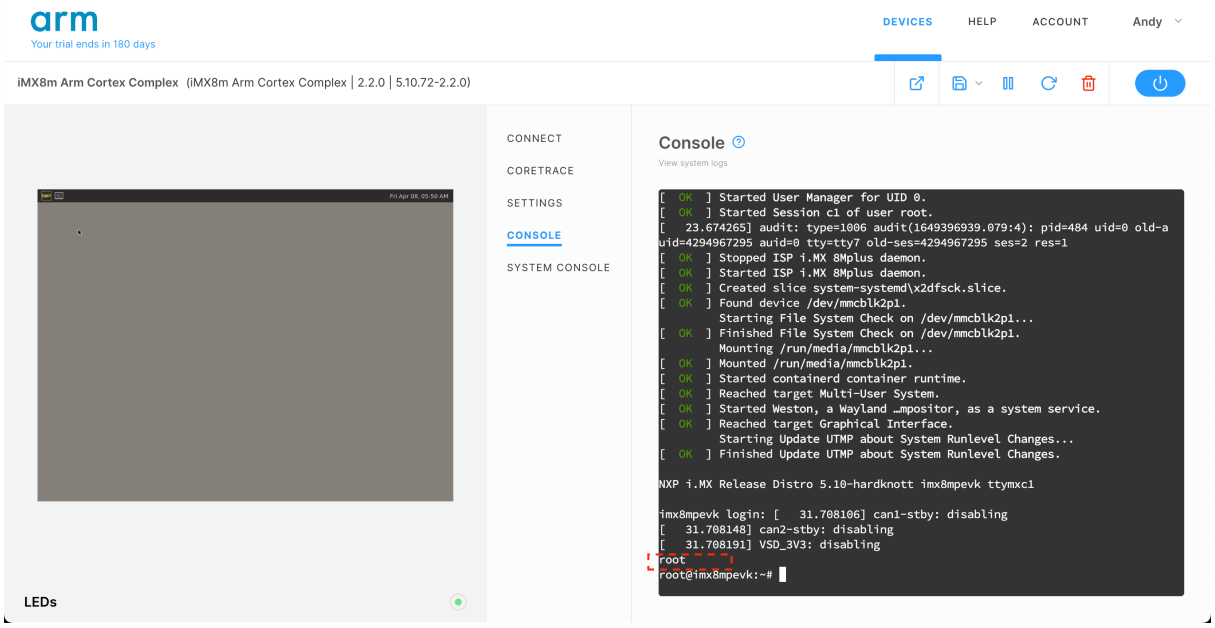
Figure 1-15: Select Console



9. Type into Login `root` then hit `return`, and you are logged in.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-16: Login



Note

The text after the login in the screenshot is normal as this is the device putting the camera to sleep.

10. Now your device is set up, logged in and ready to use.

1.1.3.3 Next steps

With your new device, you could follow the Blink the Led on the iMX8m Arm Cortex Complex device tutorial to blink the LED.

1.1.4 Quickstart for the API Docs

This quickstart will demonstrate using the API Docs to create a new device. It will include logging in, setting the API Token, finding your project ID, and finally creating the device.

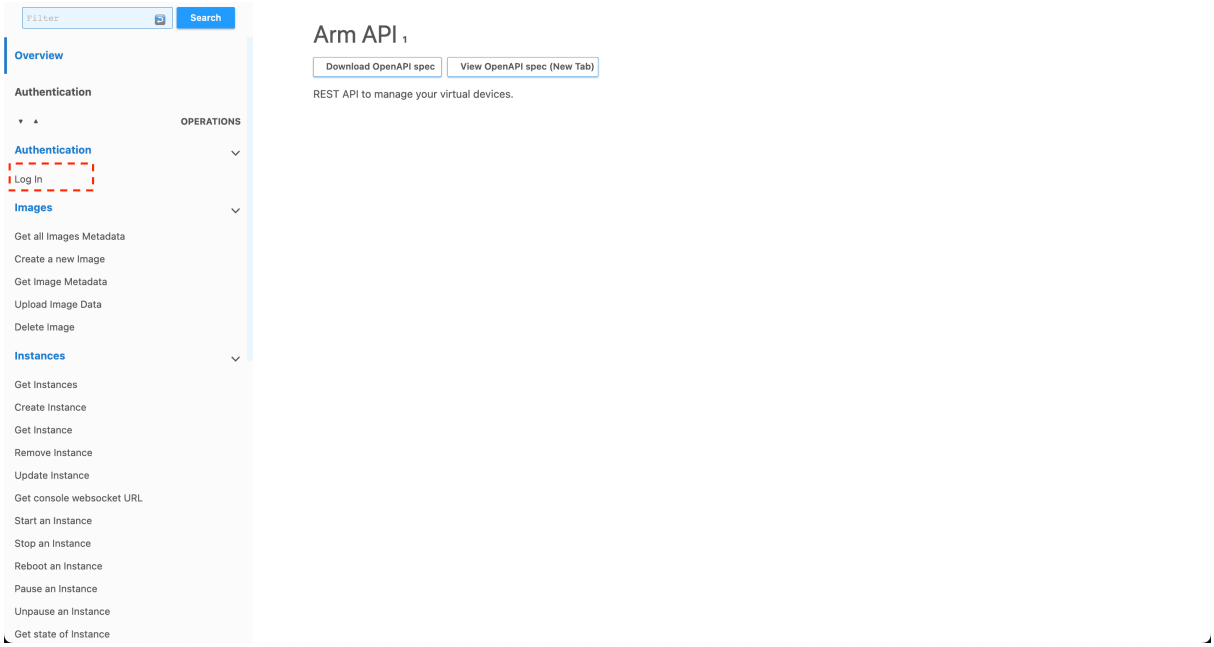
1. Log in to AVH with your Arm user account.

   https://avh.arm.com/ If you do not have or need to create an Arm account register at https://developer.arm.com/
2. To retrieve your API Token, follow the steps in the Generating an AVH API Token article.
3. Next, go to the AVH API Docs https://app.avh.arm.com/api/docs

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

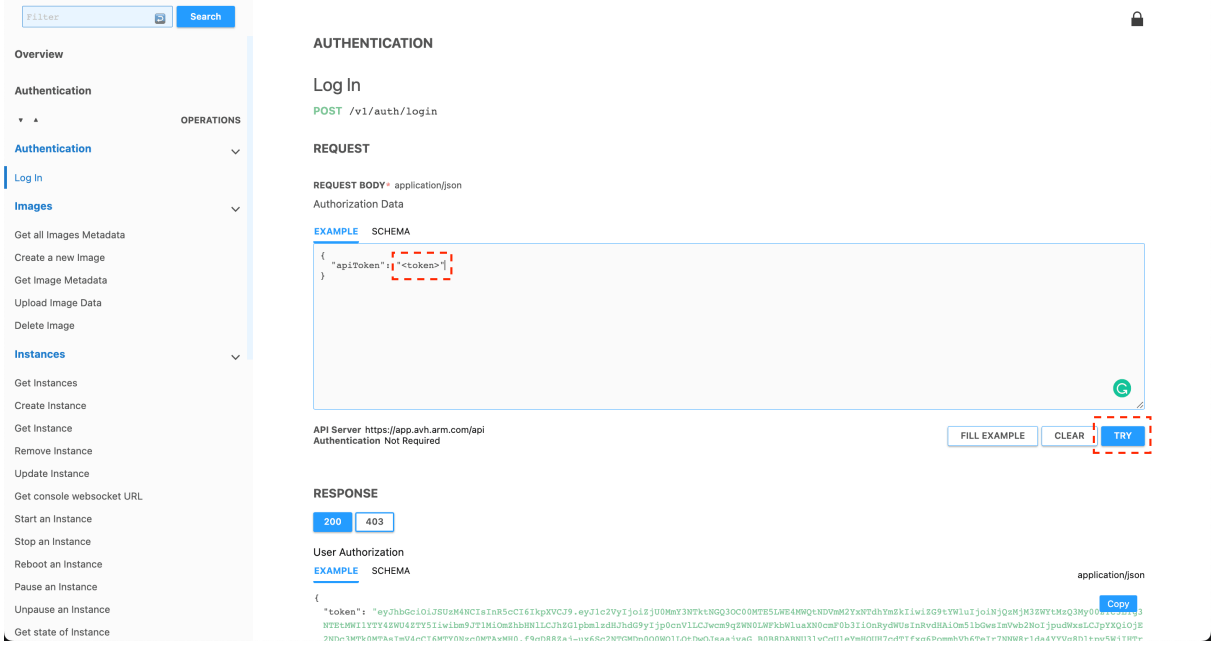
4. Click on “Log In” on the menu.

Figure 1-17: Click “Log In”



5. Paste the API Token you generated into the example request replacing `<token>`, then click “TRY”.

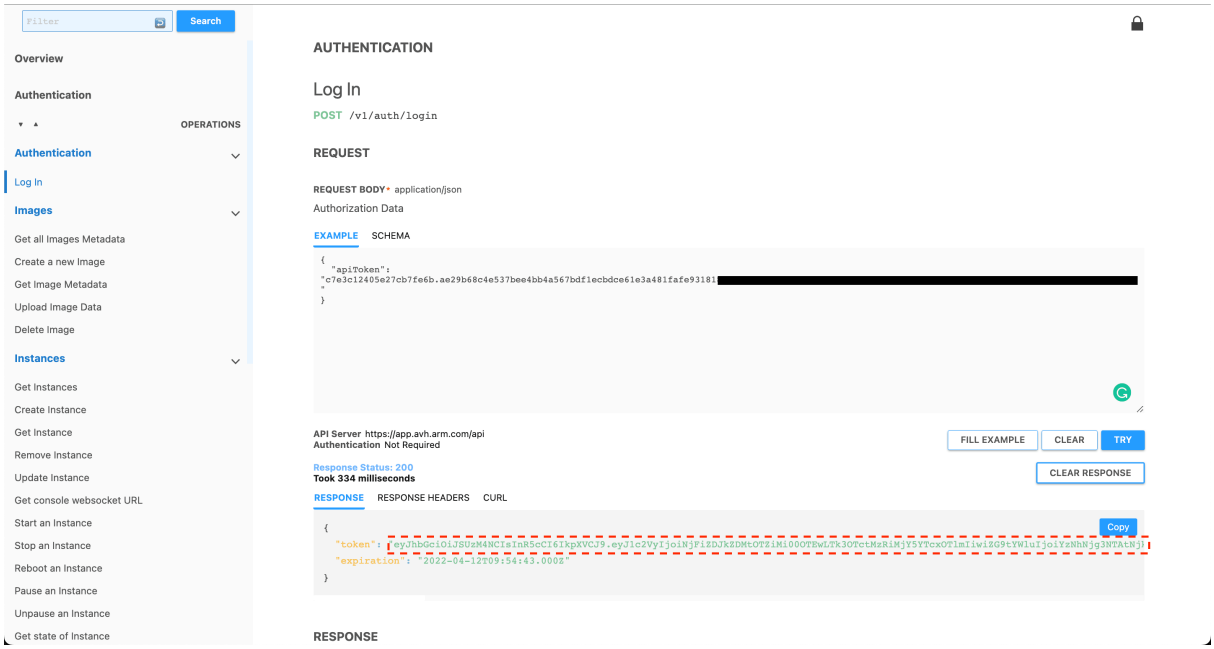
Figure 1-18: Paste the API Token



6. You can now see your auth token in the response. Copy just the value of the token. We are going to use this in the HTTP Bearer.

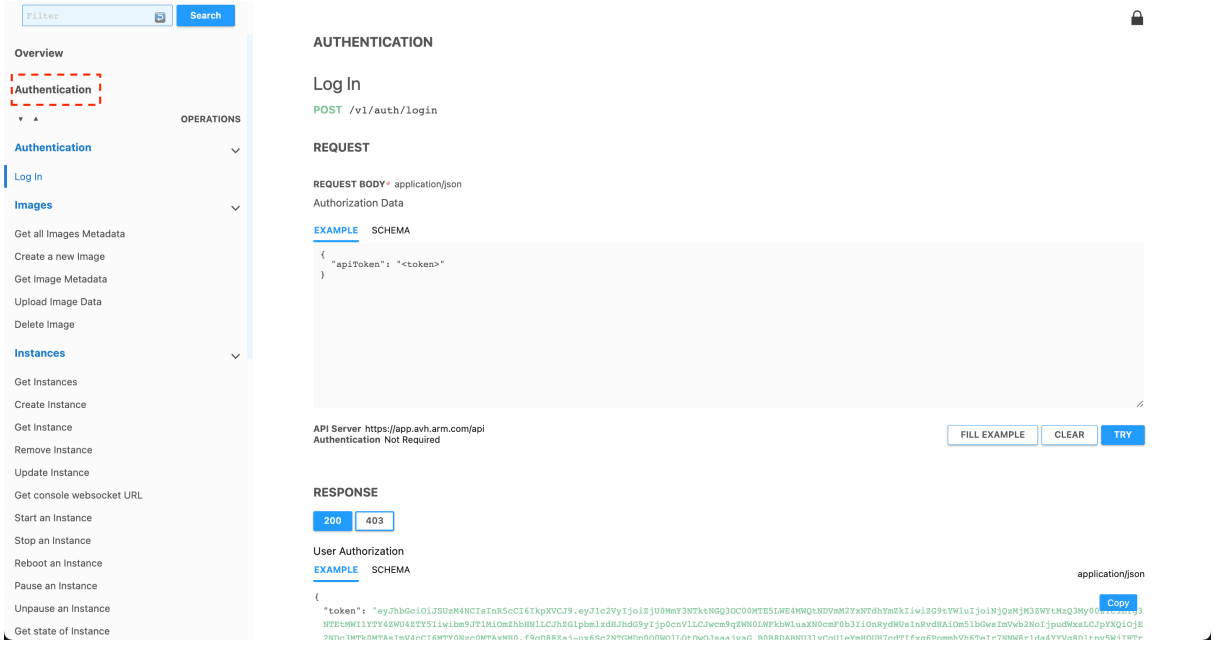
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-19: The auth token



7. Click “Authentication” in the menu.

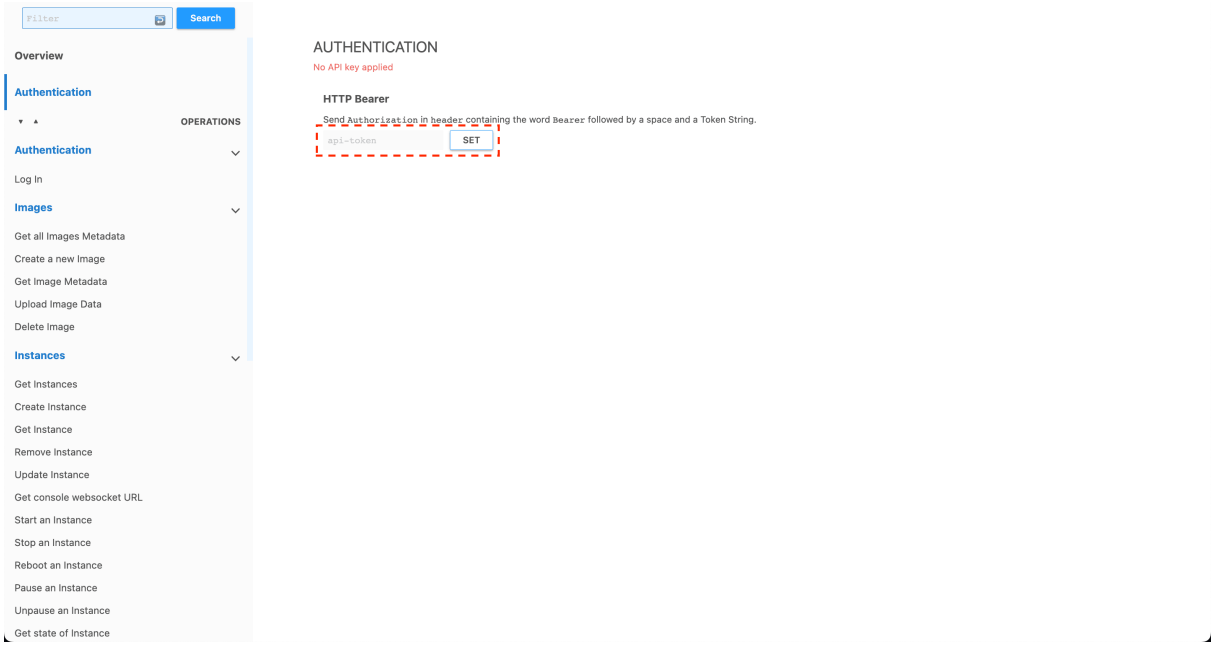
Figure 1-20: Click “Authentication”



8. Paste your auth token in the input and click “SET”

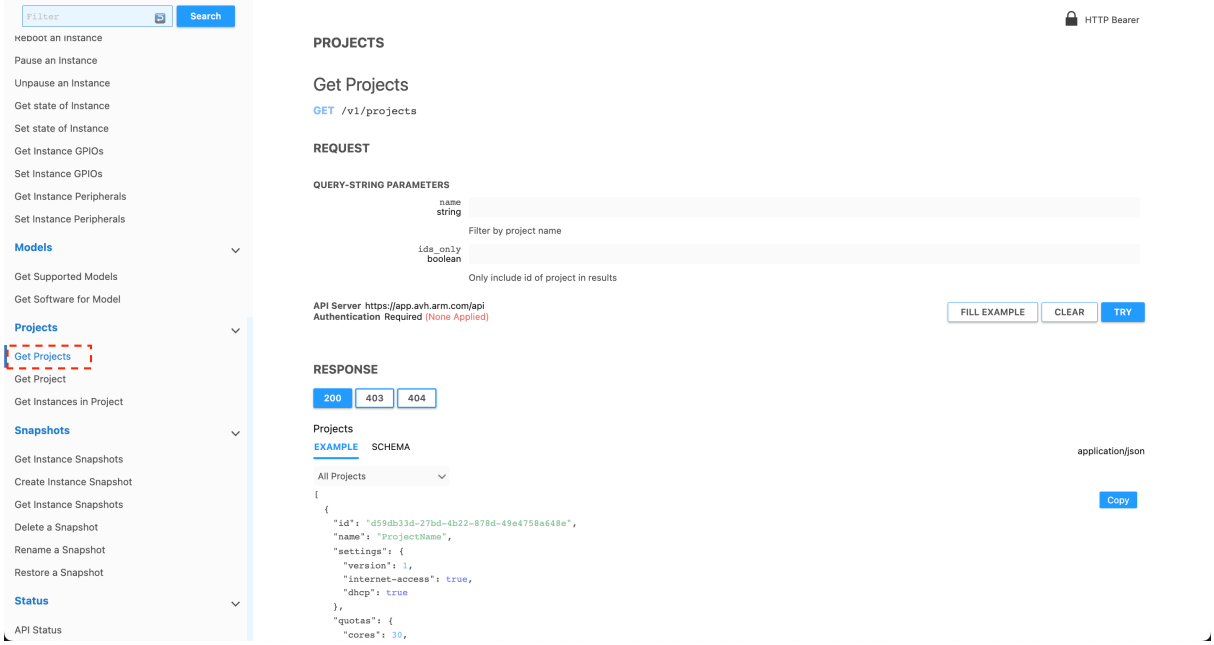
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-21: Paste the auth token



9. Click “Get Projects”

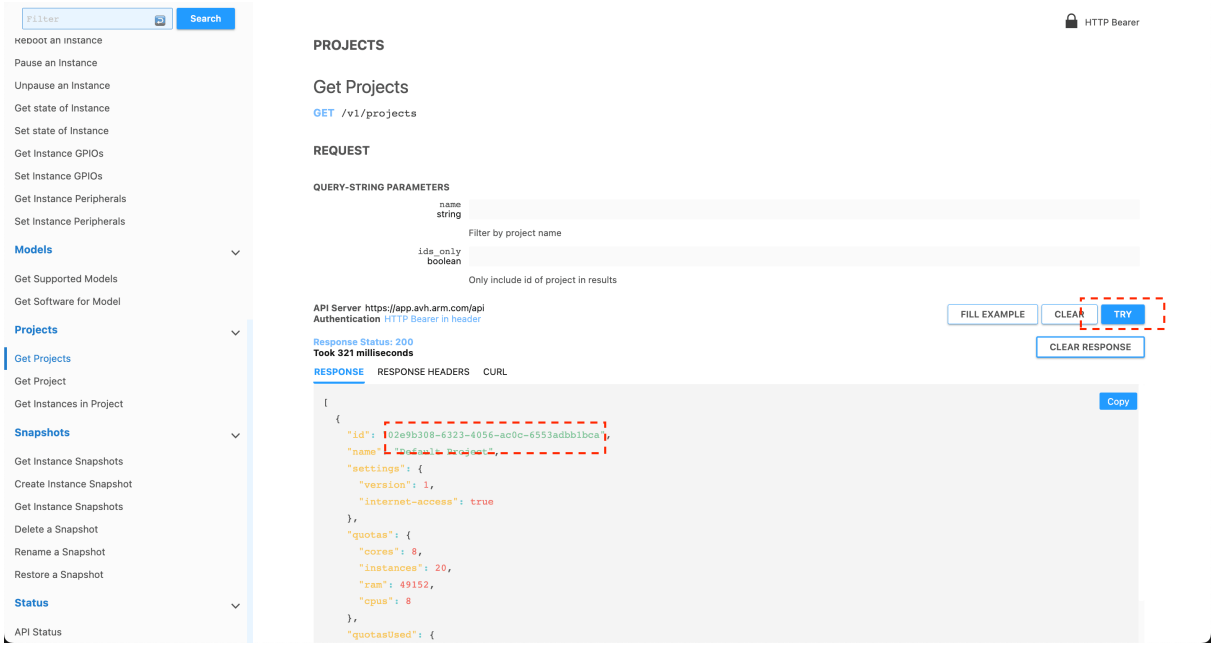
Figure 1-22: Click “Get Projects”



10. Click “TRY” to get your project’s information. Copy the `id` from the JSON, you will need this to create the device.

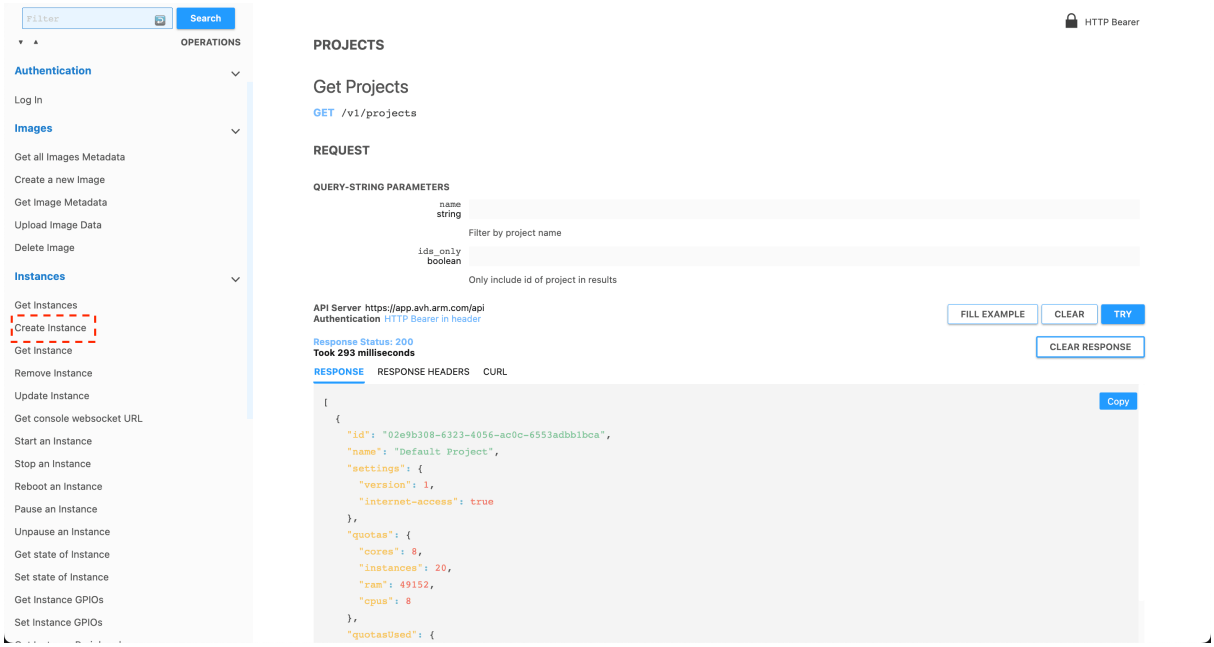
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-23: Copy the id from the JSON



11. In the menu click “Create Instance”

Figure 1-24: Click “Create Instance”



12. We are going to create a STM32U5 IoT Discovery Kit device and name it “STM32U5 created via API”. Paste the object into the example replacing the sample object with your project id.

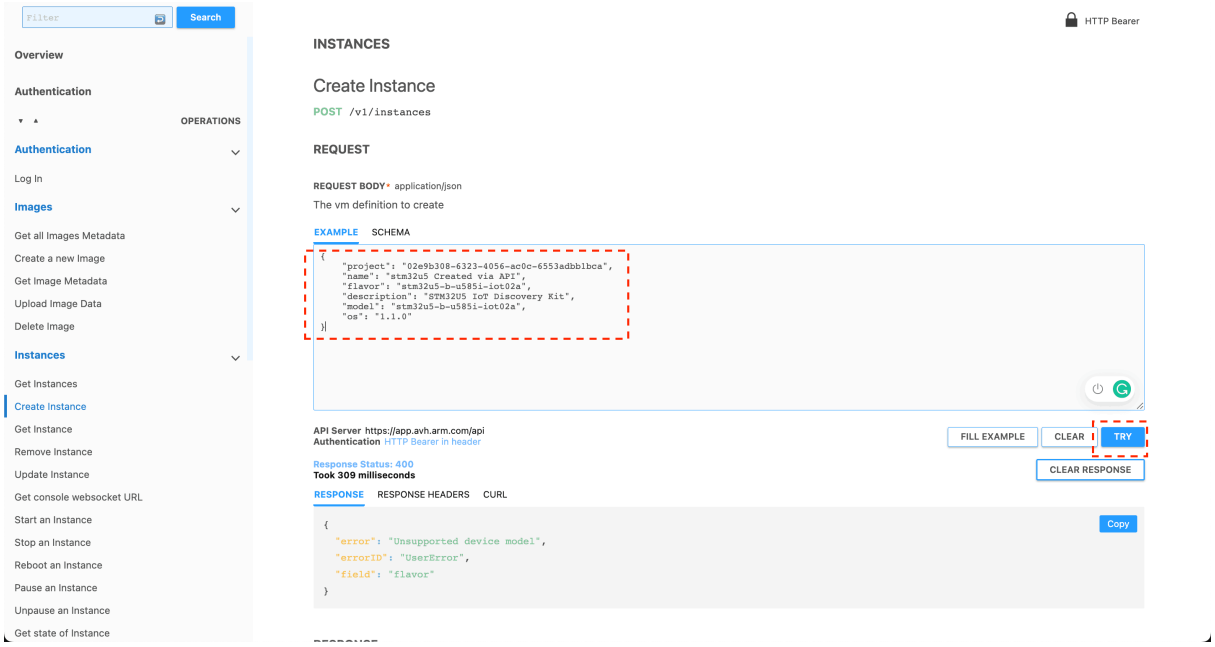
```
{
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
  "project": "<your_project_id>",
  "name": "stm32u5 Created via API",
  "flavor": "stm32u5-b-u585i-iot02a",
  "description": "STM32U5 IoT Discovery Kit",
  "model": "stm32u5-b-u585i-iot02a",
  "os": "1.1.0"
}
```

Once you have added this click “Try”

Figure 1-25: Click “Try”



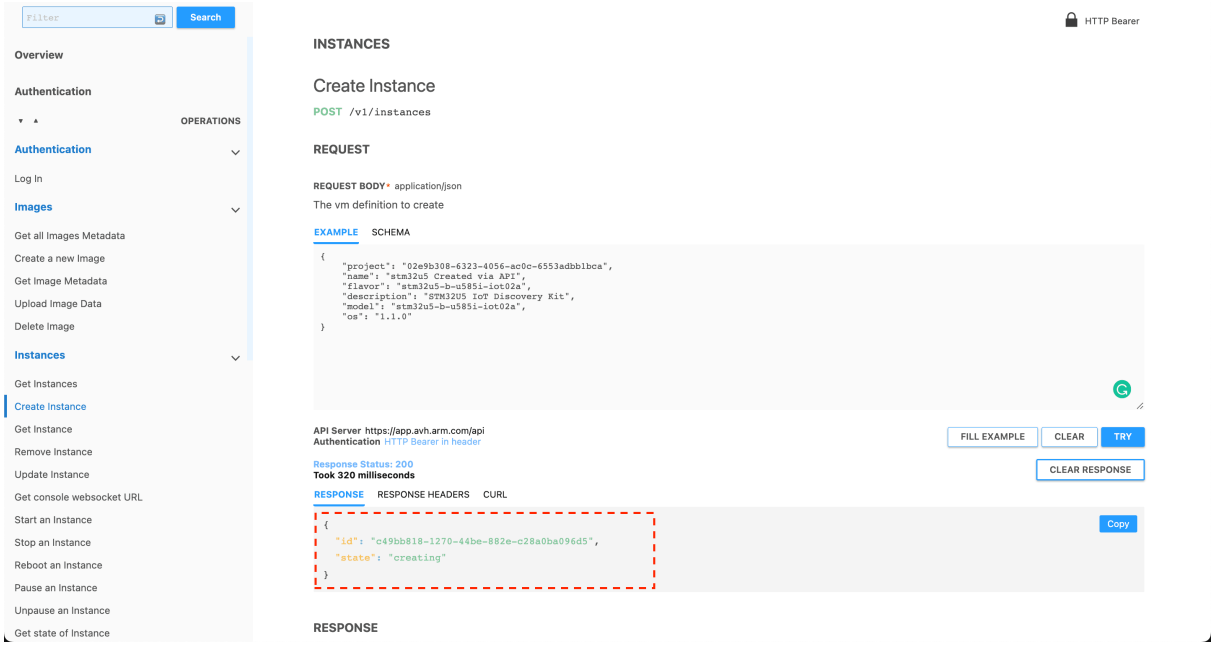
Note

You can find information about the models and the software for the models in the API Doc

13. The response will show that the device status is being created.

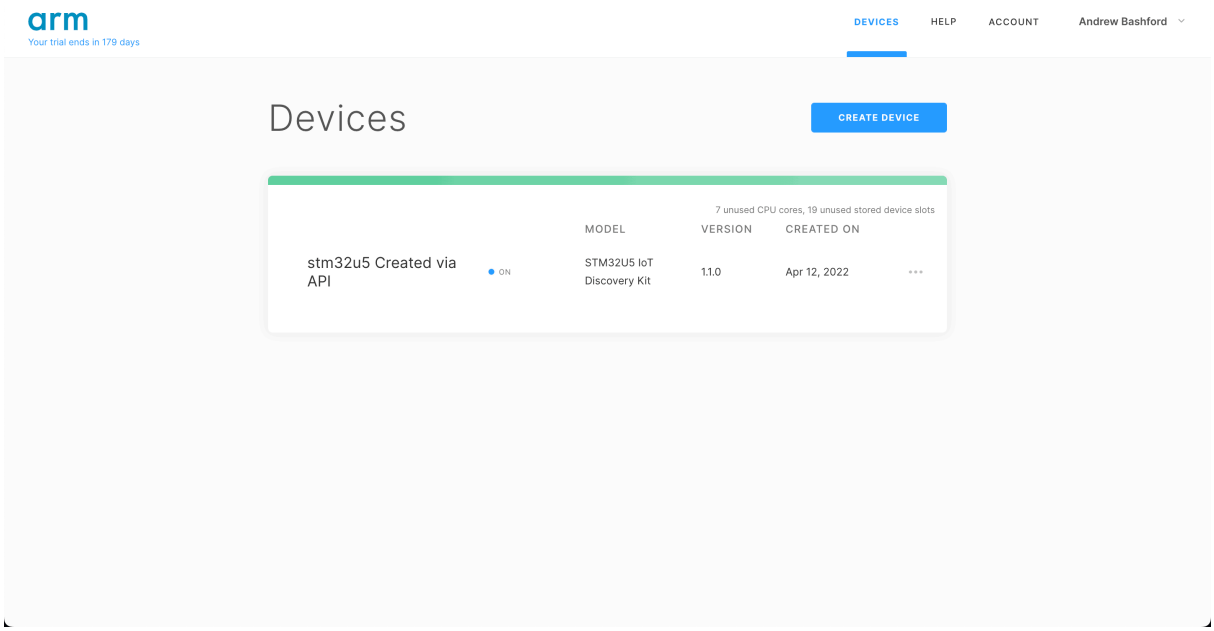
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-26: Response showing device status



14. When you go back to AVH the devices list will be the device we created in this quickstart guide.

Figure 1-27: AVH devices list



The API Docs has a rich set of features to programmatically use AVH devices for your use cases.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.2 Supported Devices and Components

See which hardware components our virtual devices support:

- Supported Hardware Components for the Raspberry Pi 4
- Supported Hardware Components for the STM32U5
- Supported Hardware Components for the i.M X8M

1.2.1 Supported Hardware Components for the Raspberry Pi 4

See which hardware components of the Raspberry Pi 4 our virtual devices support.

Note

This information is up-to-date as of 6/6/2022.

1.2.1.1 High-Level Summary

The virtualized Raspberry Pi 4 board models the following:

- SoC
  - Broadcom BCM2711B0 quad-core A72 (ARMv8-A) 64-bit, runs as Neoverse-N1
- GPU
  - Broadcom VideoCore VI (partially modeled)
- Networking
  - Gigabit Ethernet
- RAM
  - 1GB, 2GB, or 4GB LPDDR4 SDRAM
- GPIO
  - 40-pin GPIO header, populated (user can model using vMMIO)
- Storage
  - microSD
- Ports
  - 2 × micro-HDMI 2.0
  - 3.5 mm analog audio-video jack
  - 2 × USB 2.0
  - 2 × USB 3.0

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Camera Serial Interface (CSI)

1.2.2 Supported Hardware Components for the STM32U5

Table of Contents

- High-Level Summary
- Detailed List of Components

Note

This information is up-to-date as of 6/6/2022.

1.2.2.1 High-Level Summary

The virtualized STM325U5 board models the following:

- 802.11 b/g/n compliant Wi-Fi
- MEMS sensors:
  - 2 digital microphones
  - Relative humidity and temperature sensor
  - 3-axis magnetometer
  - 3D accelerometer and 3D gyroscope
  - Pressure sensor, 260-1260 hPa absolute digital output barometer
- Ambient-light sensor
- 2 user LEDs
- User push-button
- Reset push-button
- USB Type-C

1.2.2.2 Detailed List of Components

The detailed list of components is as follows:

- CPU
  - ICACHE
  - DCACHE
  - RAMCFG

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

    - EXTI
- Accelerators
    - CORDIC model
    - FMAC model
    - CRC model
- Power
    - PWR model
- Clocks and Watchdogs
    - RCC
    - RTC
    - CRS
    - IWDG
    - WWDG
- Timers
    - TIM6/7
    - TIMx
    - LPTIM
- DMA
    - LPDMA
    - GPDMA
    - DMA2D
        - Stub only since the board has no LCD
- ADC, DAC, Analog
    - ADC4
    - ADC1
    - DAC
    - VREFBUF
    - COMP
    - OPAMP
- Communication, Interconnect, IO
    - USART/UART
    - I2C
    - LPUART
    - SPI
    - OTG_FS

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

  - OCTOSPI
  - FDCAN
  - UCPD
  - GPIO
- Storage
  - FLASH
  - SDMMC
  - FSMC
  - DLYB
- Security
  - GTZC
  - TAMP
  - RNG
  - AES / SAES
  - HASH
  - PKA
- Audio/Video/Touch
  - MDF
  - ADF
  - TSC
- External
  - EMW3080 WLAN
  - TCPP03 USB type-C PD controller
  - M24256 EEPROM
  - IIS2MDCTR magnetometer
  - ISM330DHCX accelerometer and gyro
  - LPS22HH barometer
  - HTS221 thermohydrometer
  - VEML6030 ambient light sensor
  - MX25LM51245G
  - APS6408L-3OBM

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.2.3 Supported Hardware Components for the i.M X8M

See which hardware components of the NXP i.M X8M our virtual devices support.

Table of Contents

- High-Level Summary
- Detailed List of Components
  - On-Chip Resources
  - Off-Chip Resources

Note

This information is up-to-date as of 6/6/2022.

1.2.3.1 High-Level Summary

The virtualized NXP iMX8MPlus board models the following:

- Main CPU Platform
  - Quad Core Arm Cortex-A53 (Trustzone not implemented)
- Low Power, Security CPU
  - Arm Cortex-M4
- External Memory
  - 3 SD/eMMC
- Connectivity and I/O Peripherals
  - MIPI CSI (4-lanes each)
  - 18 I2S TDM (32 b @ 384 kHz), ASRC, 8-ch. PDM DMIC input, eARC
  - 2 USB 2.0/3.0 Type C
  - 2 GbE (1x TSN)
  - UART x4

Note

The following interfaces/IP are not modeled:

- DSP and NPU
- ISP
- PCIe Gen 3
- Raw NAND

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.2.3.2 Detailed List of Components

In this section read about on-chip and off-chip resources.

1.2.3.2.1 On-chip resources

In this section read about on-chip resources.

- Security
  - Resource Domain Controller (RDC)
  - CAAM
- Arm Platform and Debug
  - Arm Cortex A53 Platform (A53)
  - Arm Cortex M7 Platform (CM7)
  - Messaging Unit (MU)
  - Semaphore (SEMA4)
  - On-Chip RAM Memory Controller (OCRAM)
  - Shared Peripheral Bus Arbiter (SPBA)
  - System Counter (SYS_CTR)
- Clocks and Power Management
  - Clock Control Module (CCM)
    - PLL and clock root selection modeled for peripherals that use clock frequencies.
  - General Power Controller (GPC)
  - Crystal Oscillator (XTALOSC)
  - Thermal Monitoring Unit (TMU)
    - Returns constant but valid temperature.
- SNVS, Reset, Fuse, and Boot
  - On-Chip OTP Controller (OCOTP_CTRL)
  - Secure Non-Volatile Storage (SNVS)
  - 6.6 Watchdog Timer (WDOG)
    - Watchdog will cause system reboot / power off.
- Interrupts and DMA
  - 7.1 Interrupts and DMA Events
    - GICv3 and NVIC modeled.
  - Smart Direct Memory Access Controller (SDMA)
    - Documented features modeled. Model includes ROM.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

    - Interrupt Request Steering (IRQ_STEER)
- Chip I/O and Pinmux
    - IOMUX Controller (IOMUXC)
        - Necessary features modeled (registers, INITVTOR for M7).
    - General Purpose Input/Output (GPIO)
- External Memory
    - DDR Controller (DDRC)
        - Modeled as block of RAM, no registers.
    - AHB-to-APBH Bridge with DMA (APBH-Bridge-DMA
    - 62BIT Correcting ECC Accelerator (BCH)
        - Modeled for GPMI + APBH_DMA.
    - General Purpose Media Interface (GPMI)
        - Modeled for APBH_DMA with two x8 NAND chips.
- Mass Storage
    - Enhanced Configurable SPI (ECSPI)
    - FlexSPI Controller (FlexSPI)
    - Ultra Secured Digital Host Controller (uSDHC)
- Connectivity
    - HSIO BLK_CTRL
    - Universal Serial Bus Controller (USB)
        - Both host and device modes supported.
    - Universal Serial Bus PHY (USB_PHY)
    - PCI Express (PCIe)
    - PCI Express PHY (PCIe_PHY)
        - PHY model only as needed for PCIe operation.
    - Ethernet MAC (ENET)
        - No QoS support, otherwise complete.
    - Ethernet Quality Of Service (ENET_QOS)
        - No QoS/PTP support, otherwise complete.
    - FlexCAN
- Timers
    - General Purpose Timer (GPT)
    - Pulse Width Modulation (PWM)
- Display, Imaging, and Camera
    - LCD Interface (LCDIF)

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

        - Supports RGB output modes in this version.
    - MIPI DSI Host Controller (MIPI_DSI)
    - MIPI D-PHY (MIPI_DPHY)
    - LVDS Display Bridge (LDB)
    - HDMI TX Controller
    - HDMI TX PHY
        - PLL modeled, required for audio operation.
    - HDMI TX BLK_CTRL
    - HDMI TX Parallel Audio Interface (HTX_PAI)
        - Minimal register model
    - HDMI TX Parallel Video Interface (HTX_PVI)
        - The HDMI video output is the AVH video output
- Audio
    - AUDIO BLK_CTRL
    - PDM Microphone Interface (MICFIL)
        - The PDM microphone input is the AVH audio input.
        - Stereo input is provided. Other channels are silent.
    - Synchronous Audio Interface (SAI)
    - Enhanced Audio Return Channel (eARC)
    - AUD2HTX HDMI audio output
        - Block is undocumented but required in Linux.
        - The HDMI audio output is the AVH audio output.
    - Low-Speed Communication and Interconnects
        - I2C Controller (I2C)
        - Universal Asynchronous Receiver/Transmitter (UART)

1.2.3.2.2 Off-chip resources

In this section read about off-chip resources.

- ONFI NAND chips for GPMI
- HDMI monitor (1024×768, 60Hz)
- USB keyboard + mouse
- PCA9450 PMIC
- PCA6416 GPIO expander
- Wolfson WM8960 audio codec

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- PLL frequency computation is required for SAI3 operation.

1.3 Next Steps

The next steps you can take are as follows:

- Taking and restoring snapshots
- Quick Connect
- Connecting to the VPN
- Connect via SSH
- Connect to the Console

1.3.1 Taking and restoring snapshots

In this article, we will use the Raspberry Pi 4 device that was created in Quickstart for Raspberry Pi 4 tutorial.

1.3.1.1 What are snapshots?

Snapshots give you the ability to save the state of a device and then restore or clone that state at a moment's notice. Reduce valuable setup time by instantly returning to a previously saved state, create clones to run multiple tests from the same starting environment, or provide identical starting VMs to all students in a virtual classroom. Snapshots can save you significant time reproducing bugs or repeating tests.

Every virtual device automatically captures a new snapshot right after the initial creation, providing a quick and easy way to revert back to that fresh image. You can take up to five snapshots (in addition to the snapshot from initial creation) per virtual device.

The virtual device must be turned off to take a snapshot.

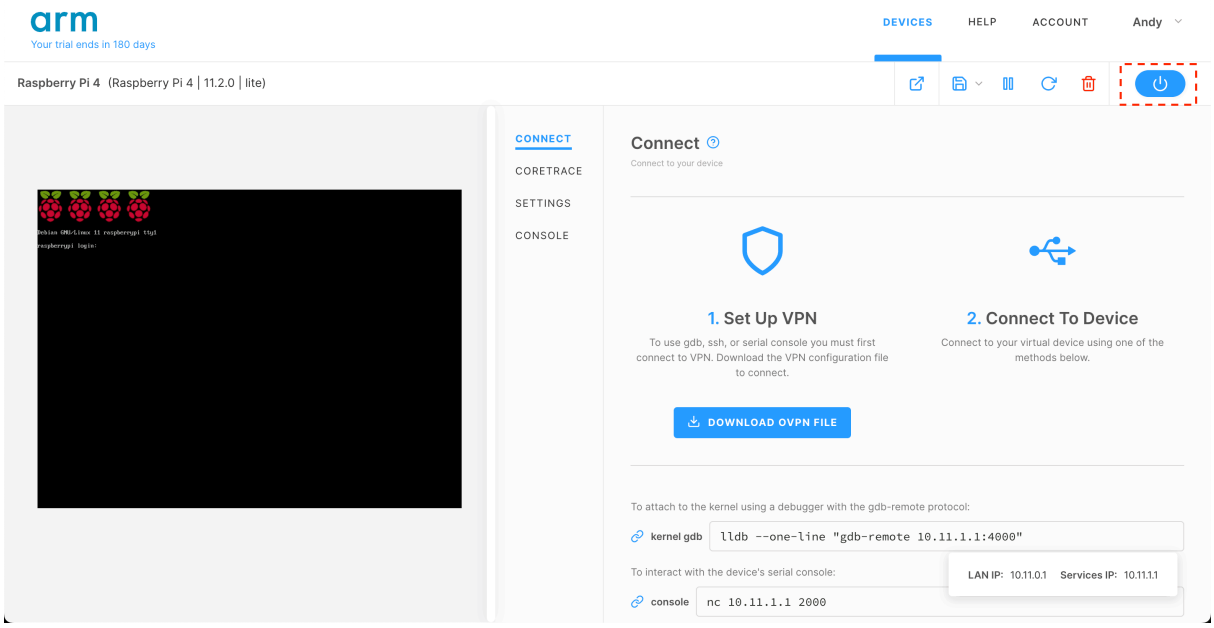
1.3.1.2 Taking snapshots

This tutorial will demonstrate Snapshots by creating and restoring them.

1. Find the device you created in the Quickstart for Raspberry Pi 4 tutorial.
2. Power off the device by clicking the power symbol on the top-right. Wait until you see POWER ON DEVICE on the left to confirm it is shut down.

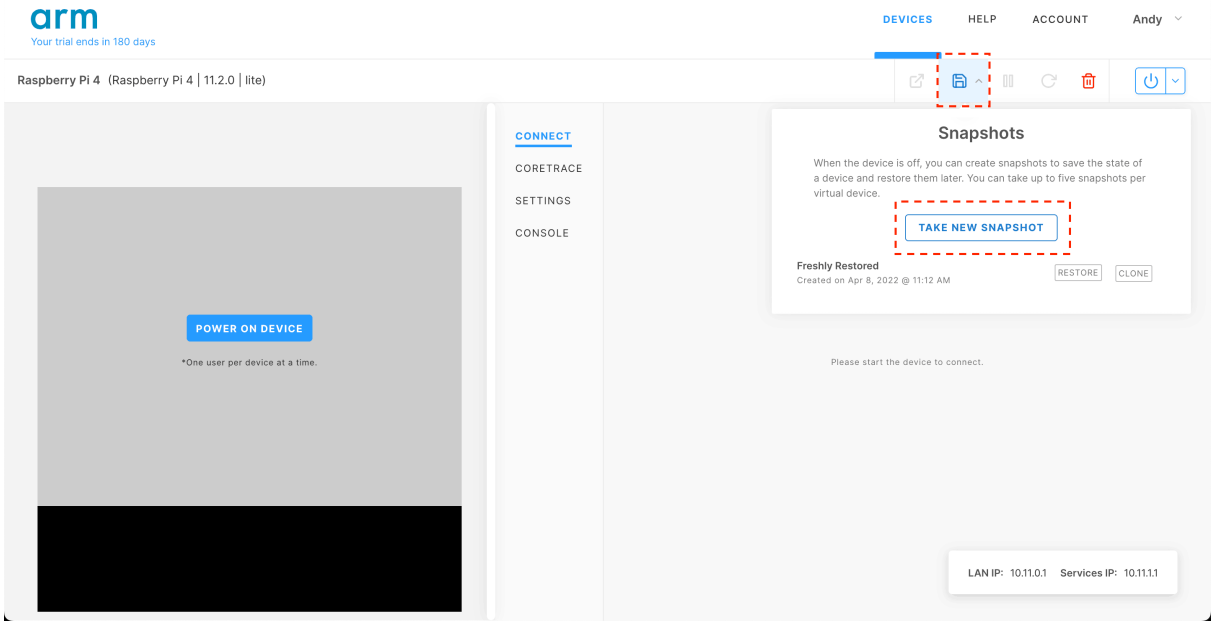
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-28: Power off the device



3. Click the Snapshot icon, then click TAKE NEW SNAPSHOT. You will see Snapshot # 1 appear below the Freshly Restored snapshot.

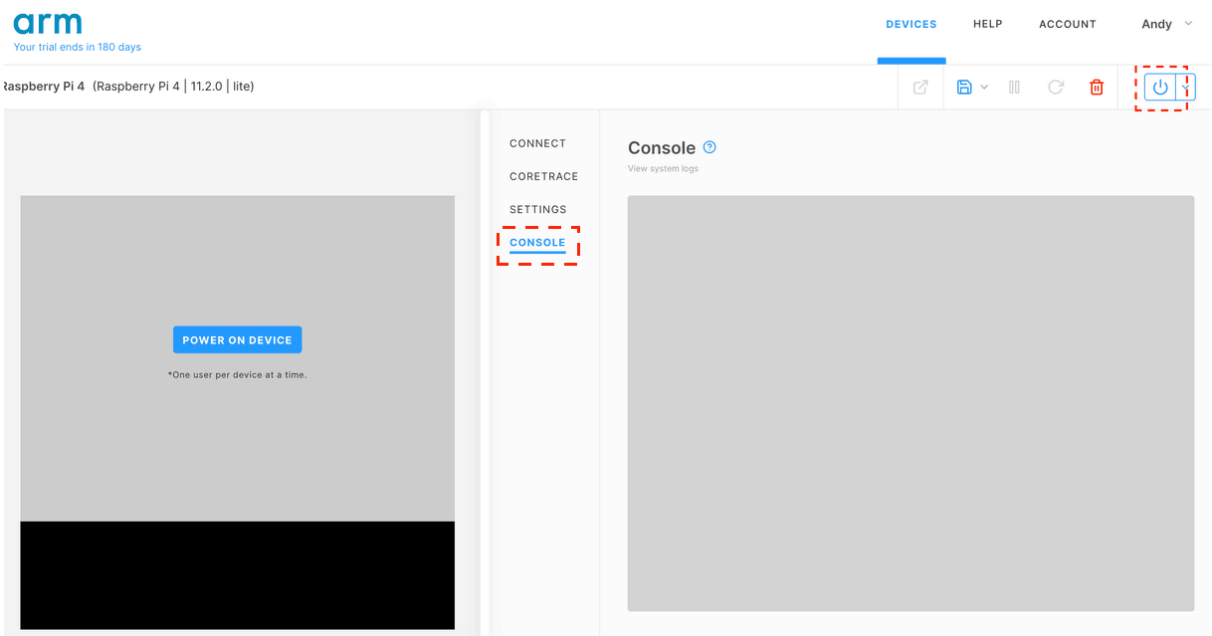
Figure 1-29: Snapshot # 1



4. Now that we saved a snapshot, let's break the device to practice restoring the saved state. Open the CONSOLE and POWER ON the device.

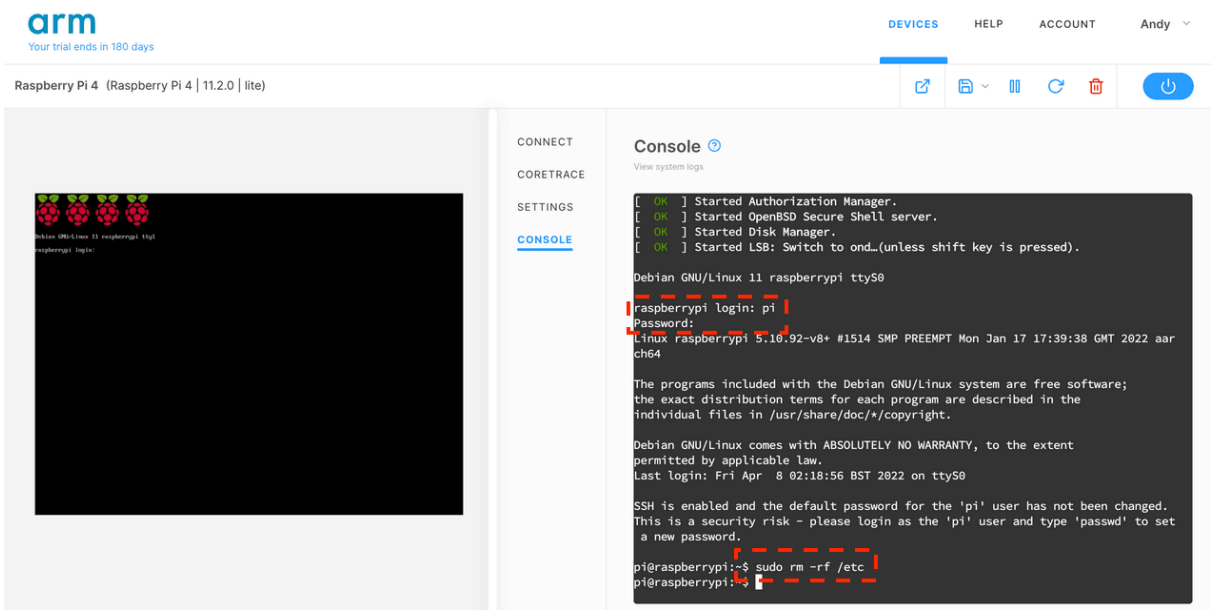
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-30: Open CONSOLE



5. Login with username: `pi` and password:`raspberry`, type `sudo rm -rf /etc`, then hit `Return`.

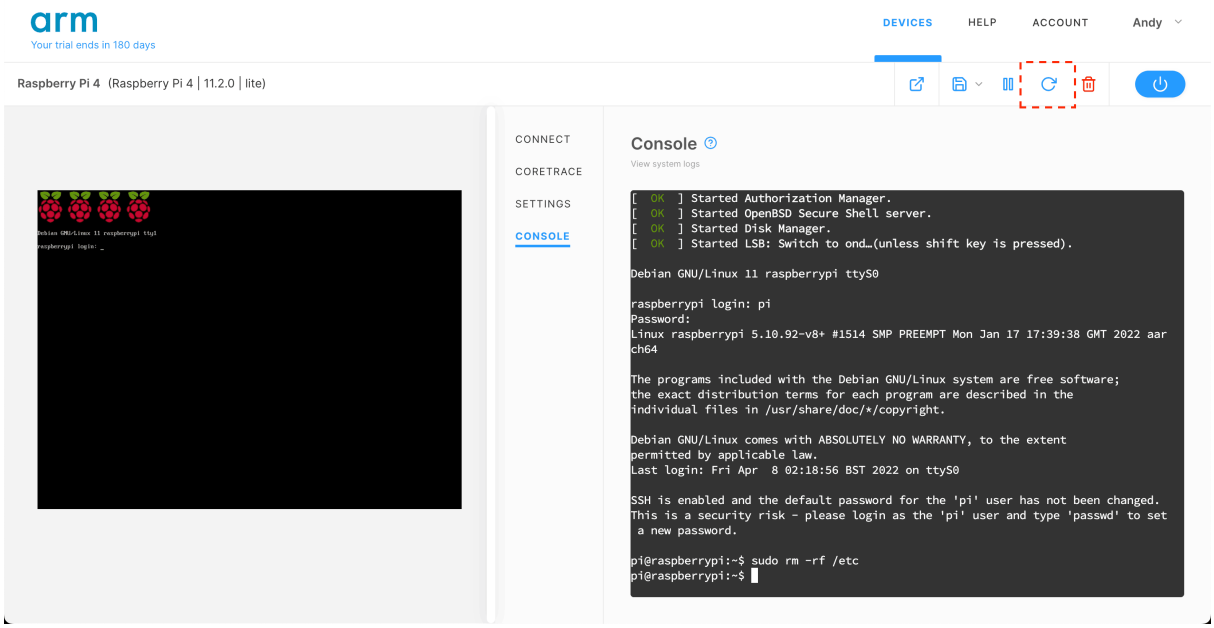
Figure 1-31: Login



6. Click Restart on the top-right.

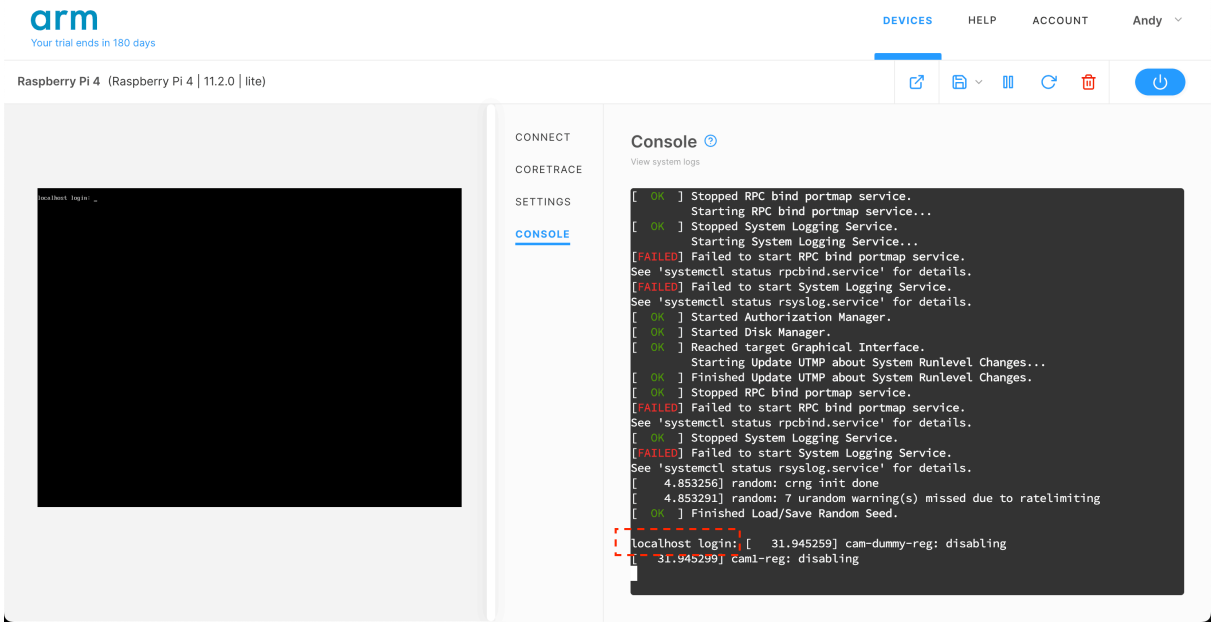
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-32: Restart



7. The Console should display `localhost login:`. This will validate the device is broken, as it is unable to log in normally.

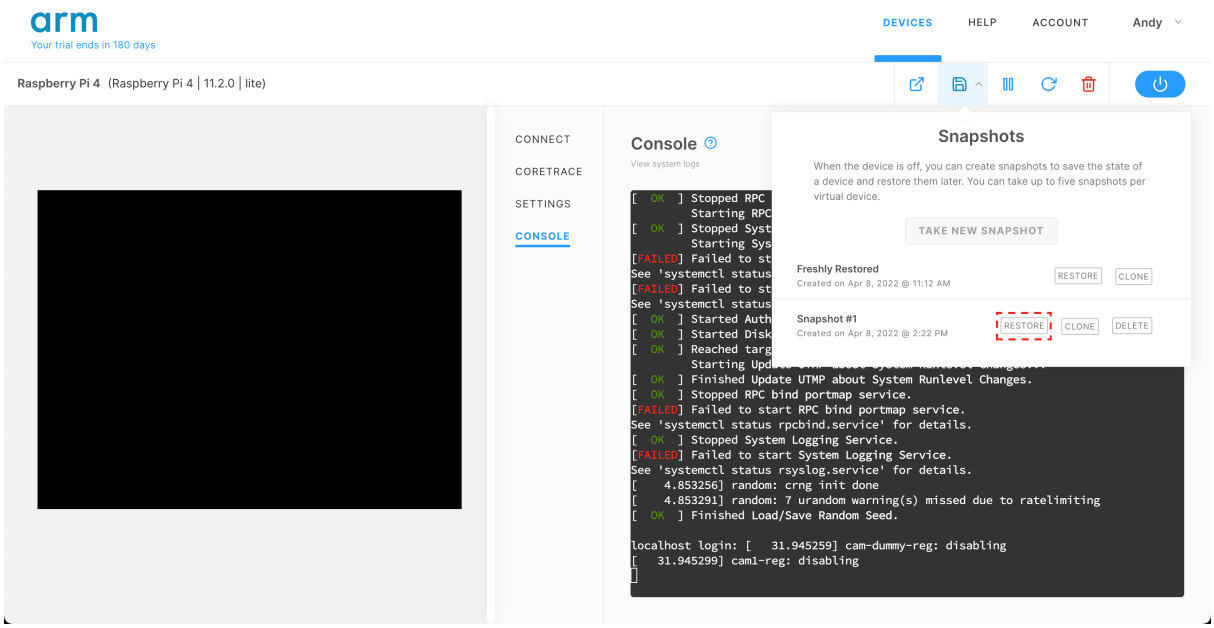
Figure 1-33: Validation



8. We will now restore the snapshot. With the device still powered on, click the Snapshot button then click RESTORE next to Snapshot #1. Wait while the snapshot state is loaded.

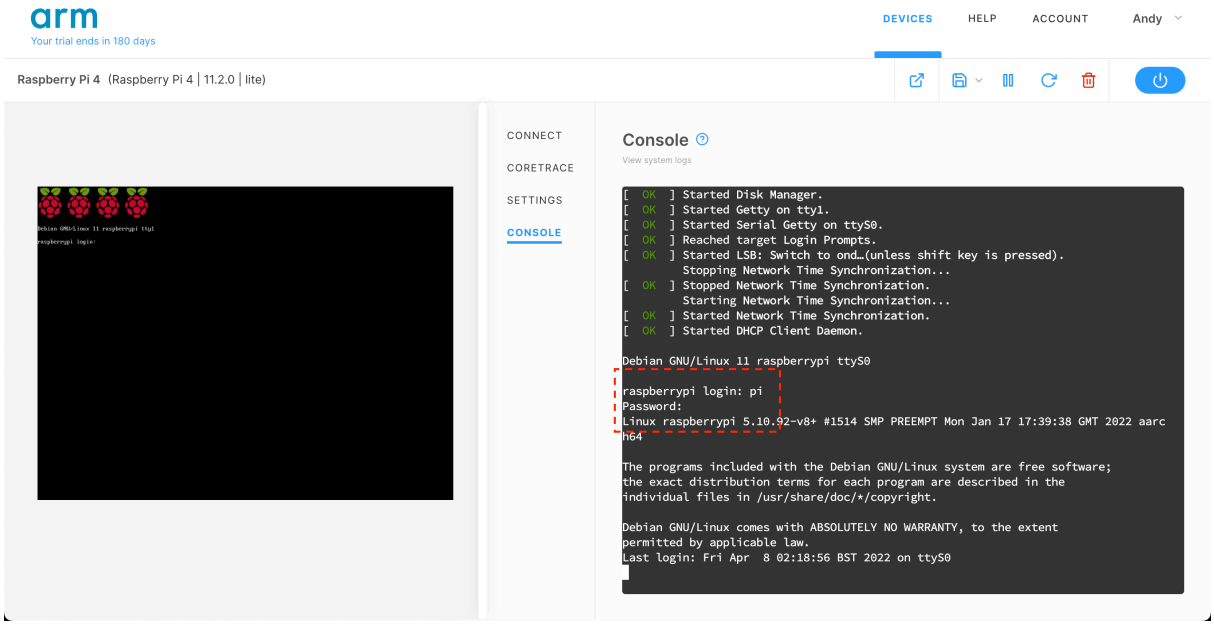
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-34: Restore snapshots



9. After the device boots up, validate that the device is working by logging in with username: `pi` and password: `raspberry`.

Figure 1-35: Validate the device



In the next tutorial, Connecting to the VPN, we will connect our device to a VPN.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.3.2 Quick Connect

Learn how to use the Quick Connect feature that enables you to connect without a VPN via an SSH Tunnel.

Table of Contents

- Note About Your Network Firewall Configuration
- Step 1: Create a New Public-Private Key Pair
  - Generate a Key Pair with Terminal
  - Generate a Key Pair with PowerShell
- Step 2: Add Your SSH Public Key to Your Project
- Step 3: Create Your Device
- Step 4: Use Quick Connect
- Specifying the Private Key File for SSH Authentication (Optional)

AVH's Quick Connect feature allows you to connect to your virtual devices and use features without requiring a VPN connection.

This "Quick Connect" feature uses standard SSH, which comes pre-installed on Mac, Windows, and Linux.

Quick Connect is available on Raspberry Pi4 and iMX8m Arm Cortex Complex at present.

No more downloading a VPN client and installing the .ovpn profile to gain root access!

1.3.2.1 Note About Your Network Firewall Configuration

Your network firewall must allow outbound connections on Port 22 to AVH's proxy server.

You can find the address for the proxy server in the `SSH -J ... or SSH -M ...` command under your device's "Connect -> Quick Connect" section (for example `proxy.avh.com`).

1.3.2.2 Step 1: Create a New Public-Private Key Pair

AVH supports the use of standard SSH keys, which make use of public-key cryptography for authentication when connecting your local computer with virtual devices.

We will utilize the ssh-keygen tool, which is installed by default on all three major operating systems, and the Ed25519 signature scheme.

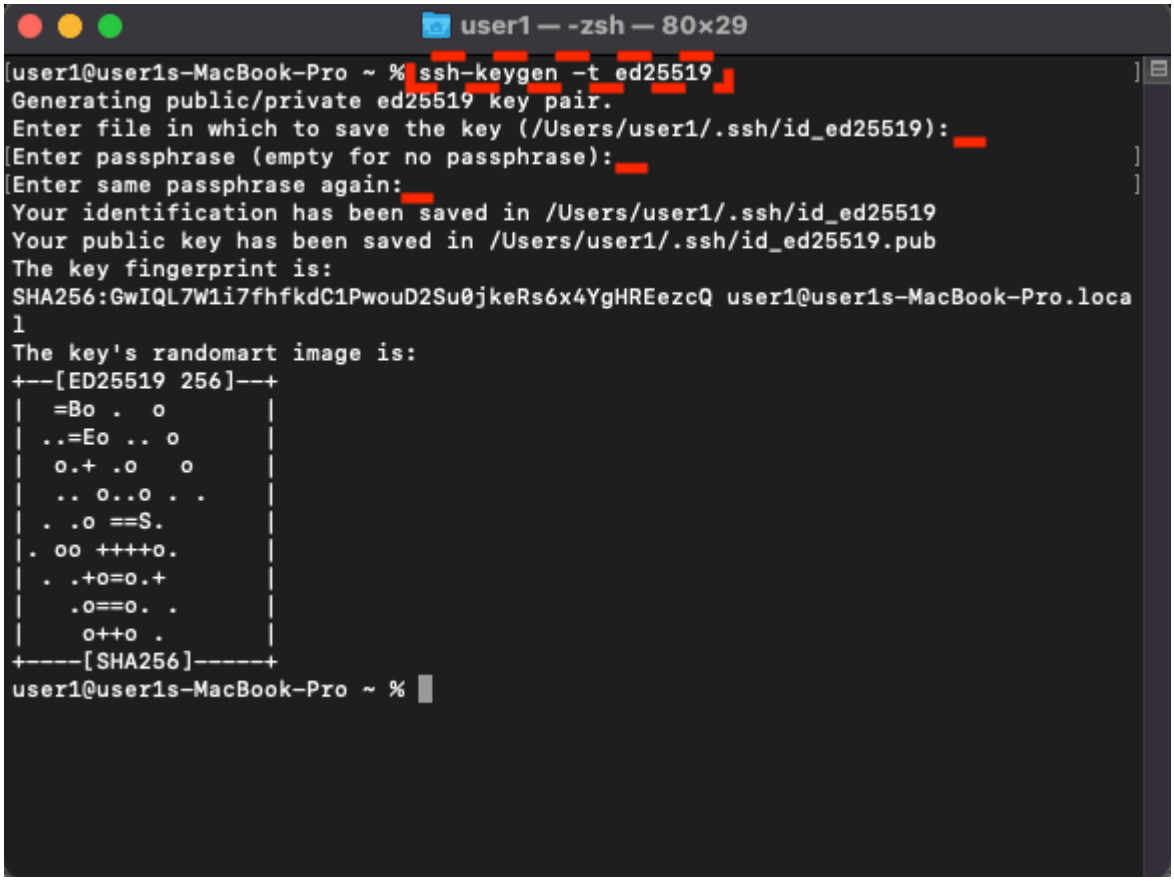
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.3.2.2.1 Generate a Key Pair with Terminal

To generate a key pair with Terminal, do the following:

1. Open Terminal and enter `ssh-keygen -t ed25519`.
2. Use the default file location (`/Users/<user>/.ssh/id_ed25519`) by pressing the `return` key.
3. Enter an optional passphrase for added security. In this example, we will leave the passphrase field blank.
4. Confirm your optional passphrase.
5. The terminal will show the location of your identification (private key) and public key files, the key fingerprint, and the randomart image.

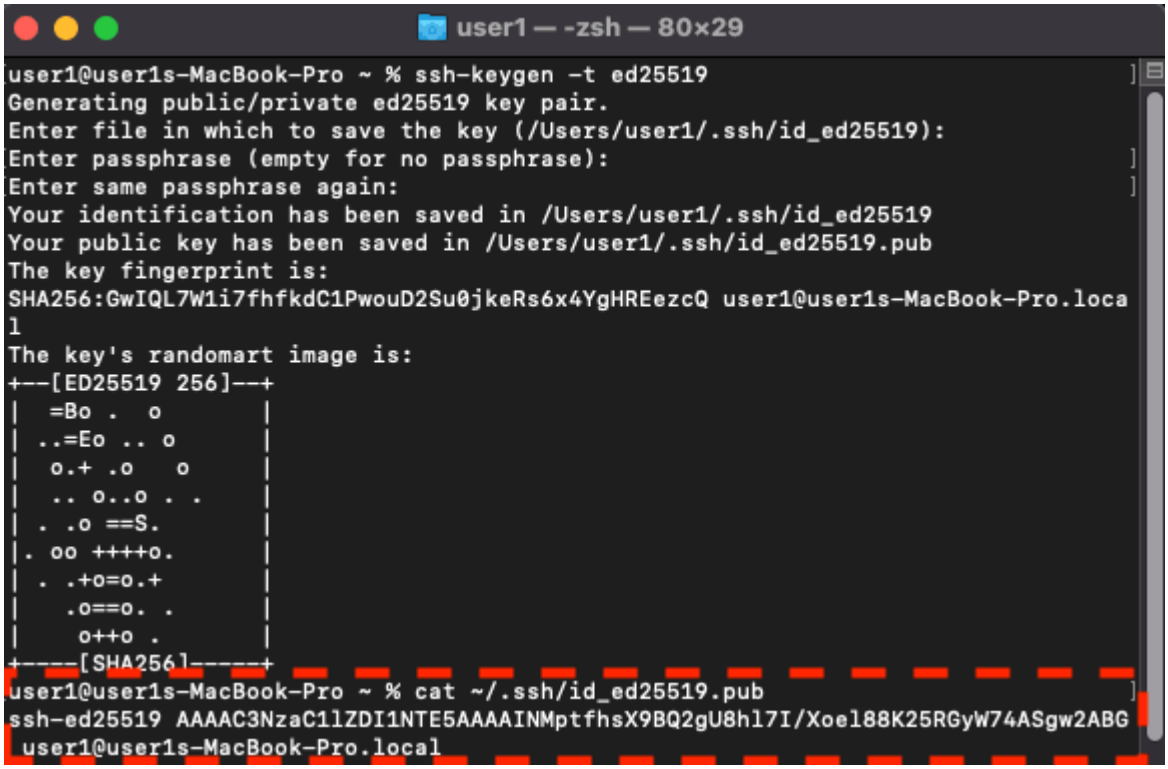
Figure 1-36: ssh-keygen output



6. View the contents of the file using `cat ~/.ssh/id_ed25519.pub`.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-37: File contents



7. Copy the output to use later in Step 2. In our example, we will copy the following:

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINMptfhsX9BQ2gU8hl7I/Xoel88K25RGyW74ASgw2ABG
 user1@user1s-MacBook-Pro.local
```

1.3.2.2.2 Generate a Key Pair with PowerShell

To generate a key pair with PowerShell, do the following:

1. Open PowerShell and enter `cd ~/.ssh`.
2. Start the key generation process using `ssh-keygen -t ed25519`.
3. Use the default file location (`c:\Users\<user>\.ssh\id_ed25519`) by pressing the `Enter` key.
4. Enter an optional passphrase for added security. In this example, we will leave the passphrase field blank.
5. Confirm the passphrase.
6. PowerShell will show the location of your identification (private key) and public key files, the key fingerprint, and the random art image.
7. View the contents of the file using `cat .\id_ed25519.pub`.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8. Copy the output, which we will use in Step 2. In our example, we will copy:

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIIURGEJFM5/YvlWysZT4nSnT5bjovGK/Uj8d6fUBsBUp
 <user>@Lenovo-Laptop
```

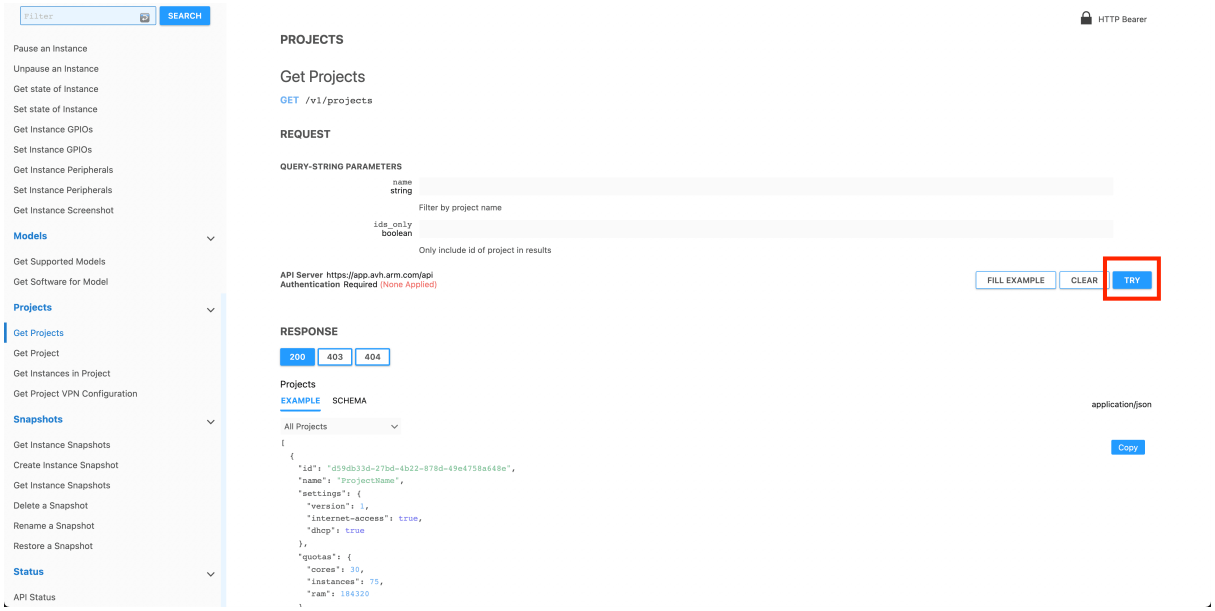
1.3.2.3 Step 2: Add Your SSH Public Key to Your Project

Once added this SSH key will be included in each new device created.

The AVH REST API supports adding SSH keys to your project using the command line.

1. Authenticate to the Rest API with your API Token. See Quickstart for the API Docs for guidance.
2. Go to the Get Projects in the API and click “Try”.

Figure 1-38: Get Projects



3. Copy the ID of the project named “Default Project” in the response.
4. Go to “Add Project Authorized Key”
5. Add the Project ID you have just copied into the “project Id string” input and add the SSH key you created in Step 1 into the “key” value pair key in the JSON. Now click “TRY”.

   The response will show the newly created key. Please note you can only have one SSH key per project.

   You can check the current SSH key with “Get Project Authorized Keys”.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.3.2.4 Step 3: Create Your Device

After adding you add the public key to your project, all devices you create under that project will automatically include the public key.

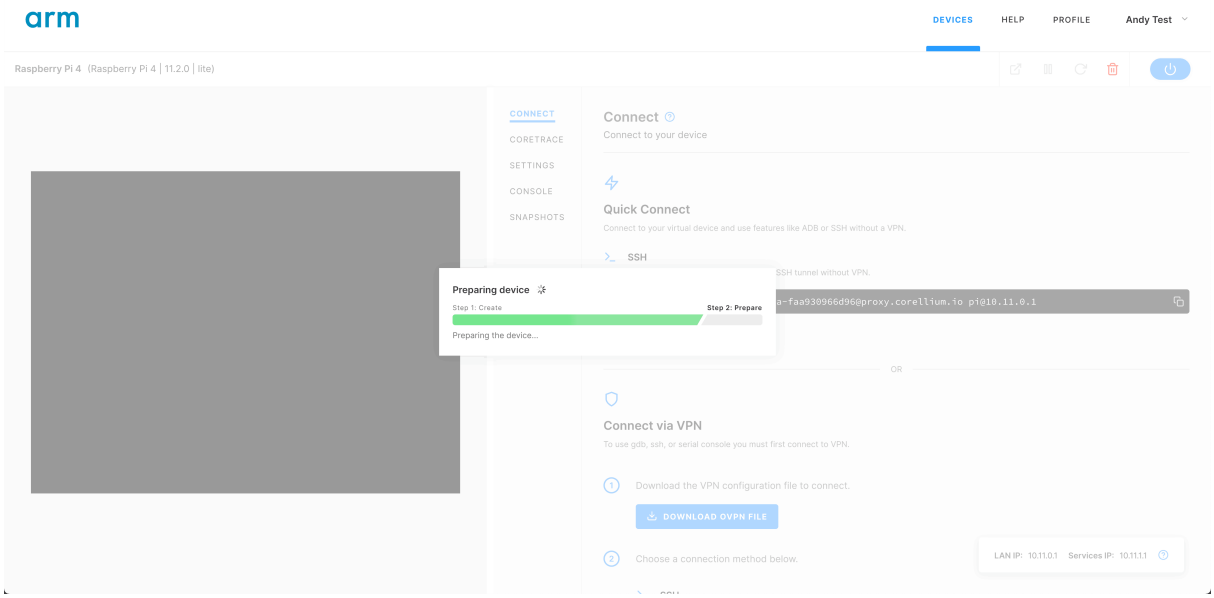
Note

You cannot add SSH keys to an existing device.

---

1. Create a new device and wait for it to boot. In this instance, we will create a Raspberry Pi.

   Figure 1-39: New Raspberry Pi device



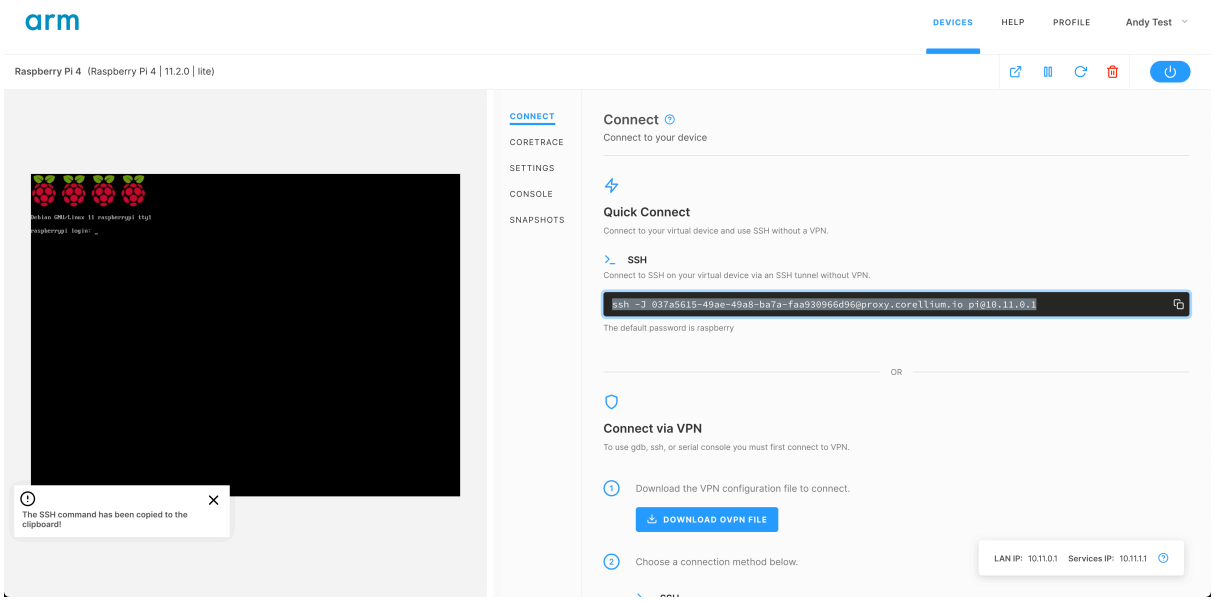
1.3.2.5 Step 4: Use Quick Connect

To use Quick Connect, do the following:

1. Once booted, copy the Quick Connect SSH command.

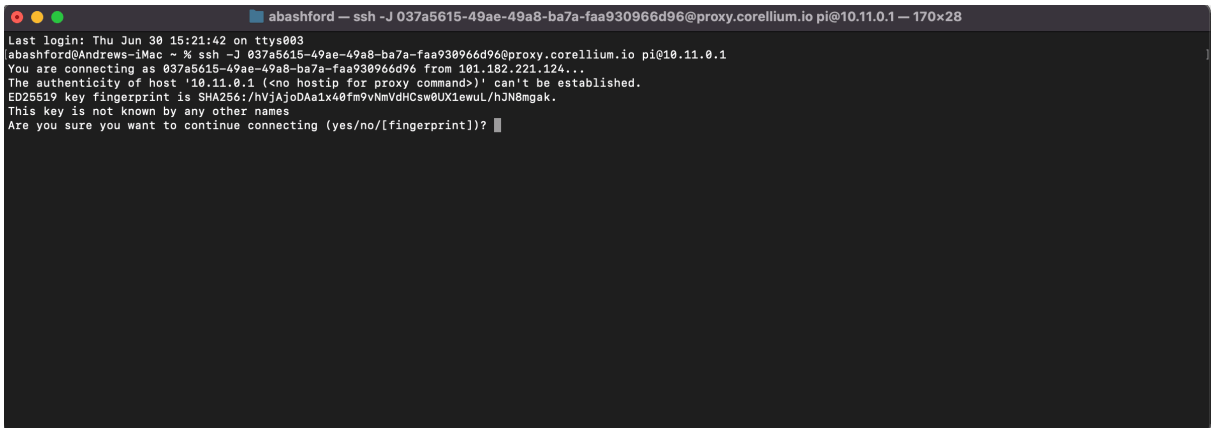
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-40: Quick Connect SSH command



2. Paste and run the copied command into Terminal or Powershell.

Figure 1-41: Paste and run the command



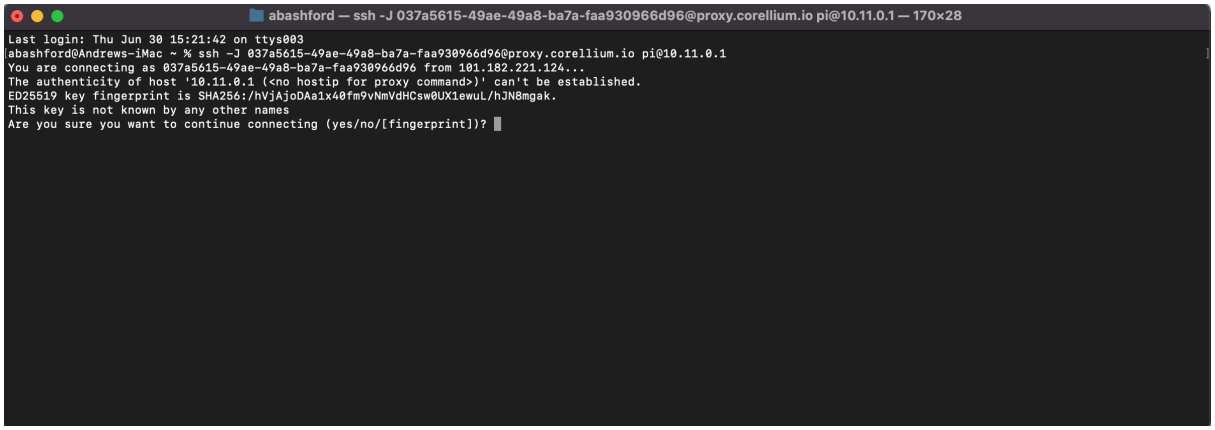
Note

Please note that if you would like to specify the private key file to use for SSH authentication rather than use the default filename, refer to Specifying the Private Key File for SSH Authentication (Optional) at the end of the article.

3. The first time you connect, you will need to confirm the proxy host fingerprint. Type `yes` when asked about connecting to the proxy server (for example `proxy.avh.com`). This step only happens once. You should see a `You are connecting as ...` confirmation message.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-42: Confirm the proxy host fingerprint



4. If there is a password required you can enter it now.

Figure 1-43: Enter the password

5. Now you are connected without the need for a VPN.

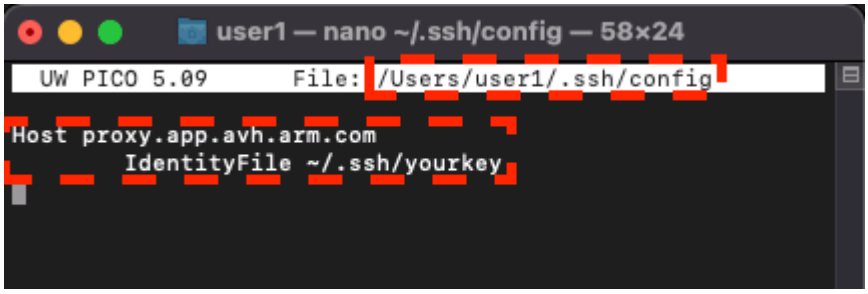
1.3.2.6 Specifying the Private Key File for SSH Authentication (Optional)

As an advanced option, if you normally use a "named" ssh file key using the `-i` argument, as opposed to using the default filename `id_ed25519` detailed in this article, you have two options:

- Include `-i` arguments in the ssh command. If you plan to use separate keys for the jump proxy host and for the target destination, you will need to include two `-i` arguments. You can read more about this method here.
- Specify the `IndentityFile` in your `~/.ssh/config`. (The private key file must exist in `~/.ssh`, and the associated public key will also need to be added to your AVH project.)

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-44: Specify the IndentityFile



1.3.3 Connecting to the VPN

In this article, we will use the Raspberry Pi 4 device that was created in Quickstart for Raspberry Pi 4 tutorial.

- How to Connect with Mac OS
- How to Connect with Windows OS

1.3.3.1 Why Use the VPN?

Using a VPN will put your Virtual Device and the computer you are accessing AVH from (typically a desktop or laptop computer) on the same network.

Common use cases for this are:

- SSH access to the device
- Access your application via various TCP/IP or UDP ports and protocols from your desktop/laptop computer.
- Kernel debugging
- Run a vmmio peripheral over TCP/IP on your local computer to feed stimuli data through a peripheral running locally on your computer into the Virtual Device for application and algorithm validation.

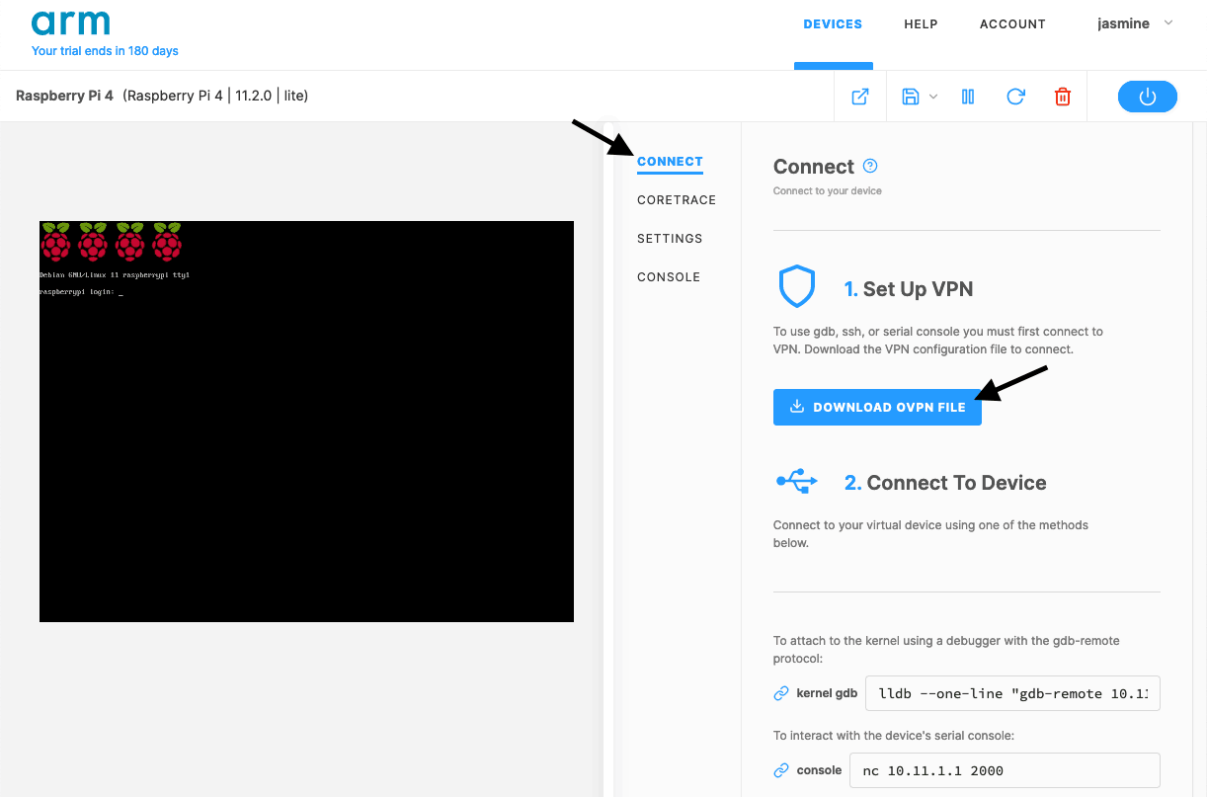
1.3.3.2 How to Connect with Mac OS

To connect with Mac OS, do the following:

1. Go to the device created in the Quickstart for Raspberry Pi 4 tutorial.
2. First, click CONNECT and then DOWNLOAD OVPN FILE.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

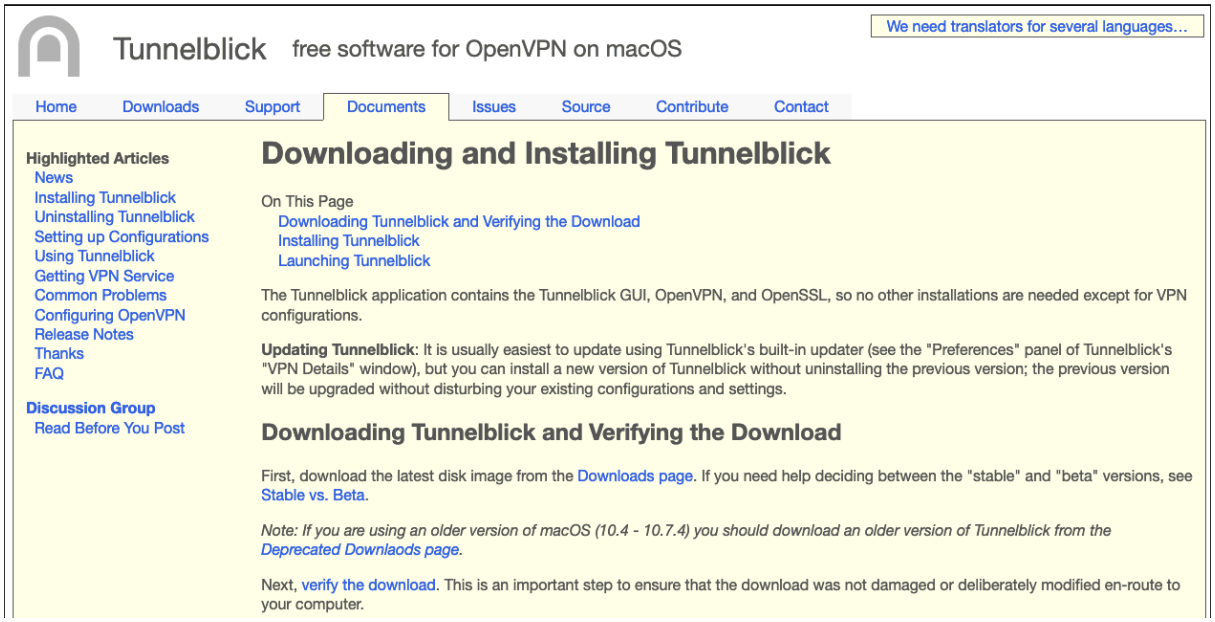
Figure 1-45: DOWNLOAD OVPN FILE



3. Download the appropriate version of Tunnelblick for your Machine from this URL
   - If you are on a new version of MacOS you need to use a patched version of Tunnelblick as MacOS no longer allows kext loading. That version is available from https://github.com/sbingner/Tunnelblick/releases/download/v3.8.8beta03%2Biouserethernet1/Tunnelblick.dmg
4. Navigate to this URL and follow the instructions to install Tunnelblick.

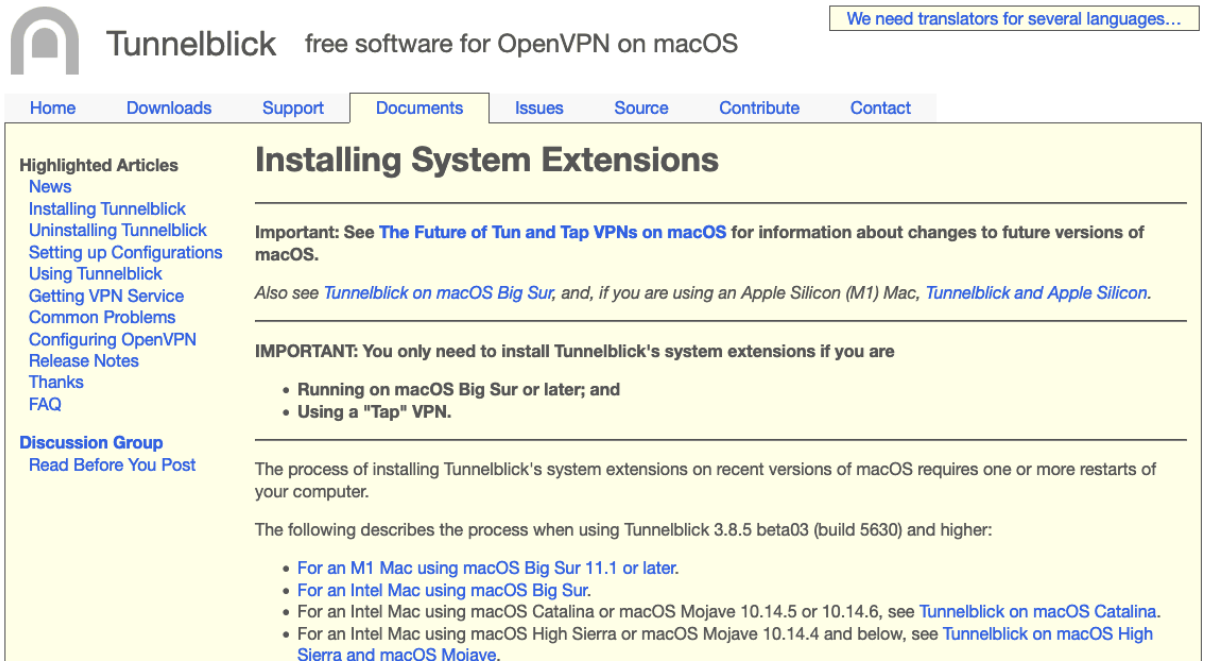
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-46: Downloading and installing Tunnelblick



5. Navigate to this URL to install the system extensions for Tunnelblick.

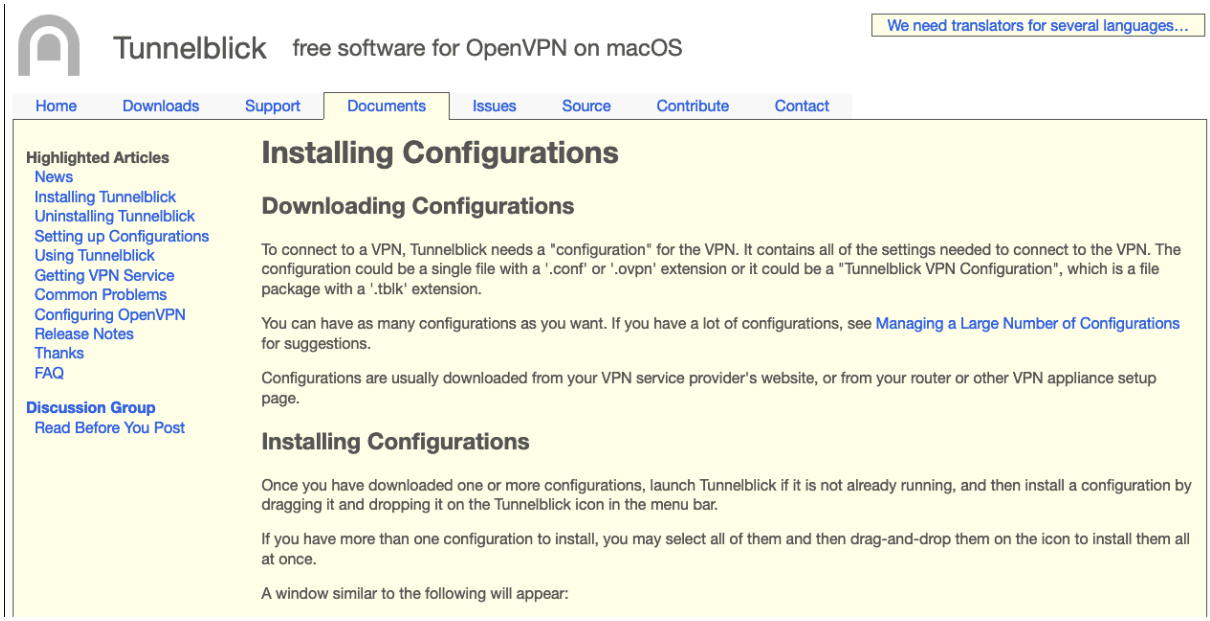
Figure 1-47: Install System Extensions



6. Add the VPN Configuration file to Tunnelblick by dragging the file over the Tunnelblick Application Icon. Navigate to this URL for detailed instructions.

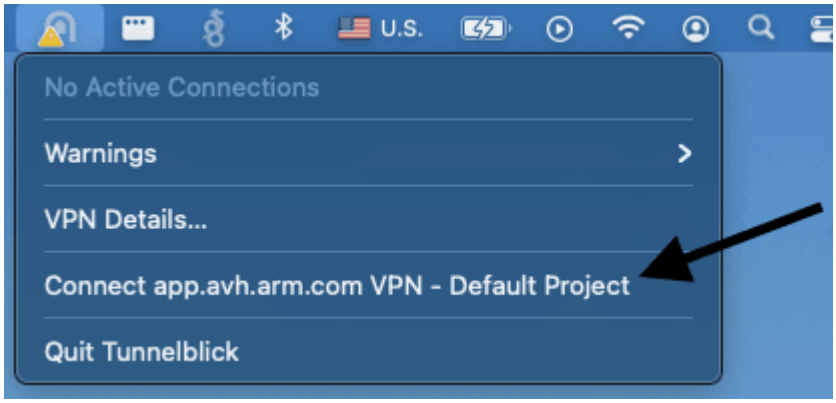
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-48: Installing Configurations



7. Click the TunnelBlick icon, then Connect ....

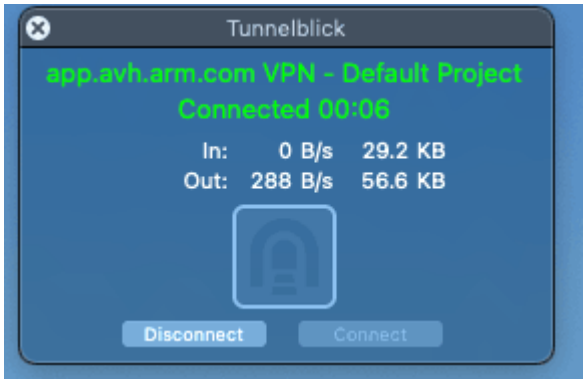
Figure 1-49: Connect



8. The text will turn green and you are now connected to the VPN!

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-50: Connected message

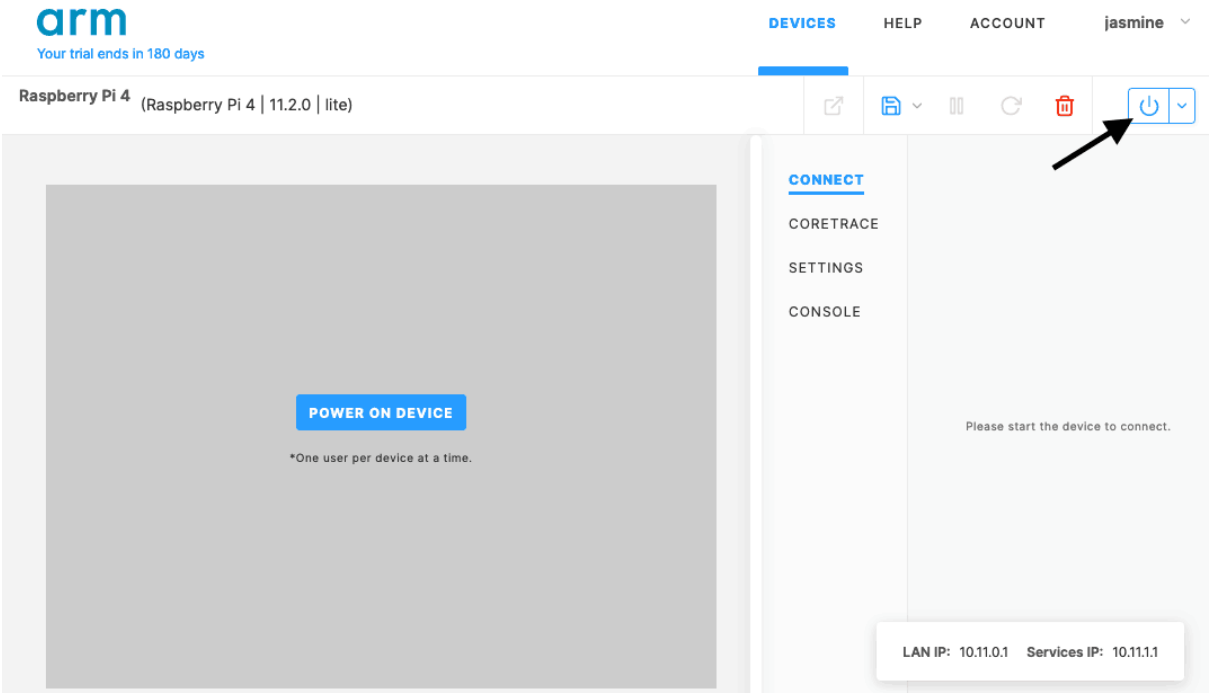


1.3.3.3 How to Connect with Windows OS

To connect with Windows OS, do the following:

1. If the device is off, power it on.

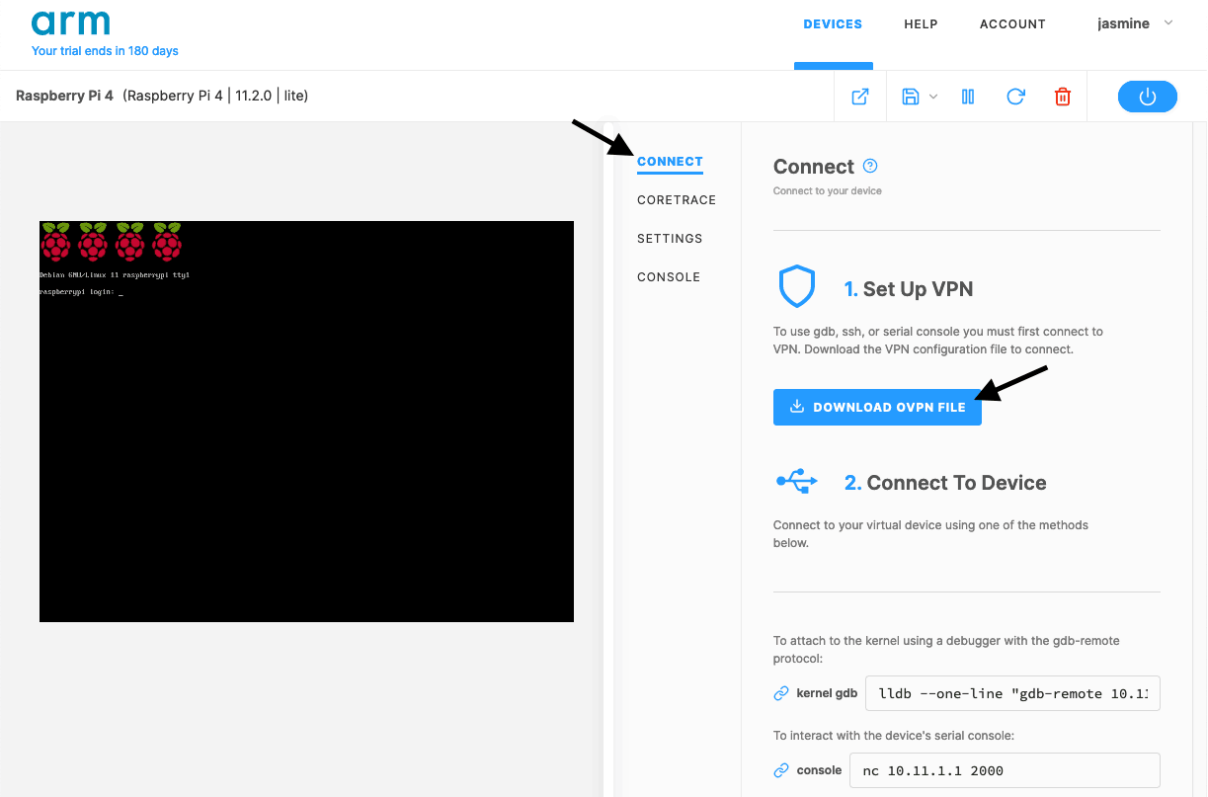
Figure 1-51: Turn on the device



2. Click CONNECT and then DOWNLOAD OVPN FILE.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

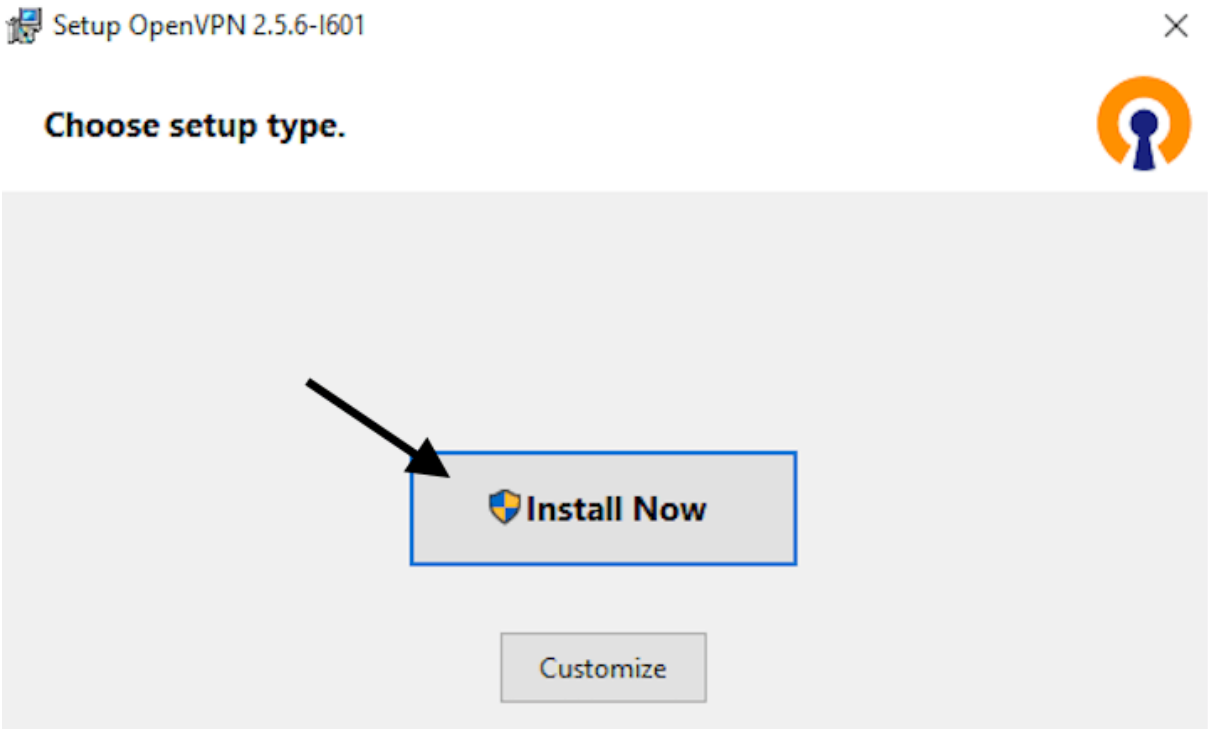
Figure 1-52: DOWNLOAD OVPN FILE



3. Install the appropriate version of OpenVPN for your machine from this URL.
4. Double click the downloaded file, then click Install Now.

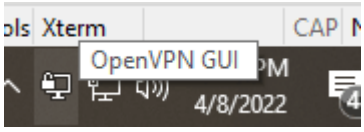
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-53: Install now



5. The OpenVPN GUI icon should appear in the taskbar by the clock.

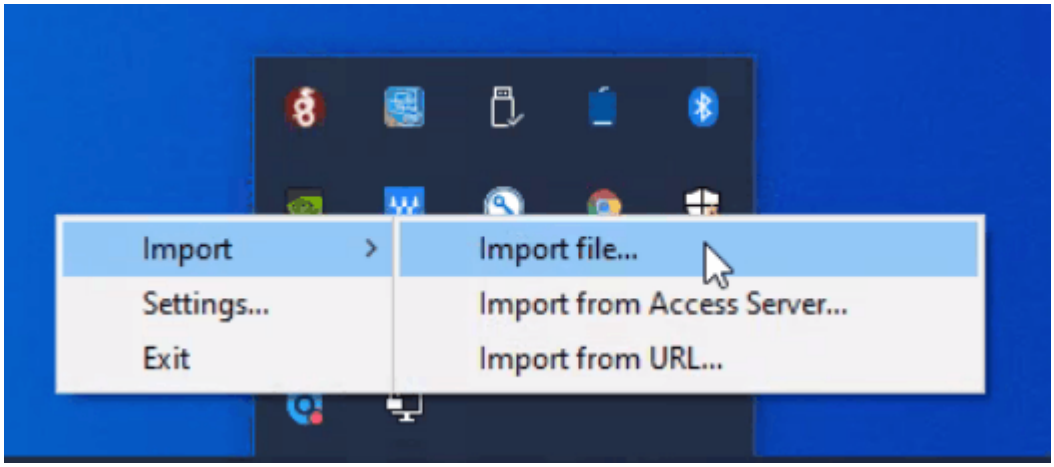
Figure 1-54: OpenVPN GUI



6. Right-click on the OpenVPN GUI icon in the taskbar and select Import → Import File.

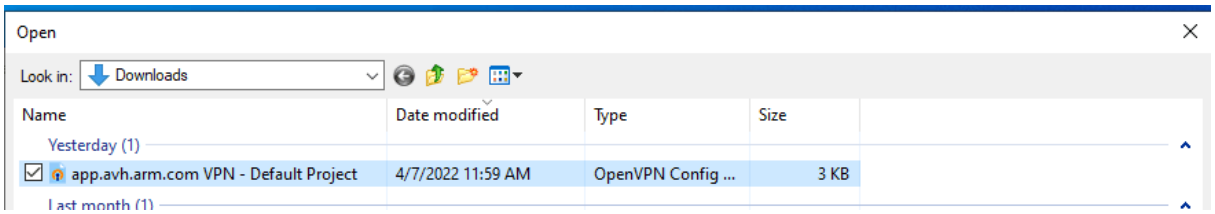
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-55: Imort file



7. Then, select the downloaded OVPN file.

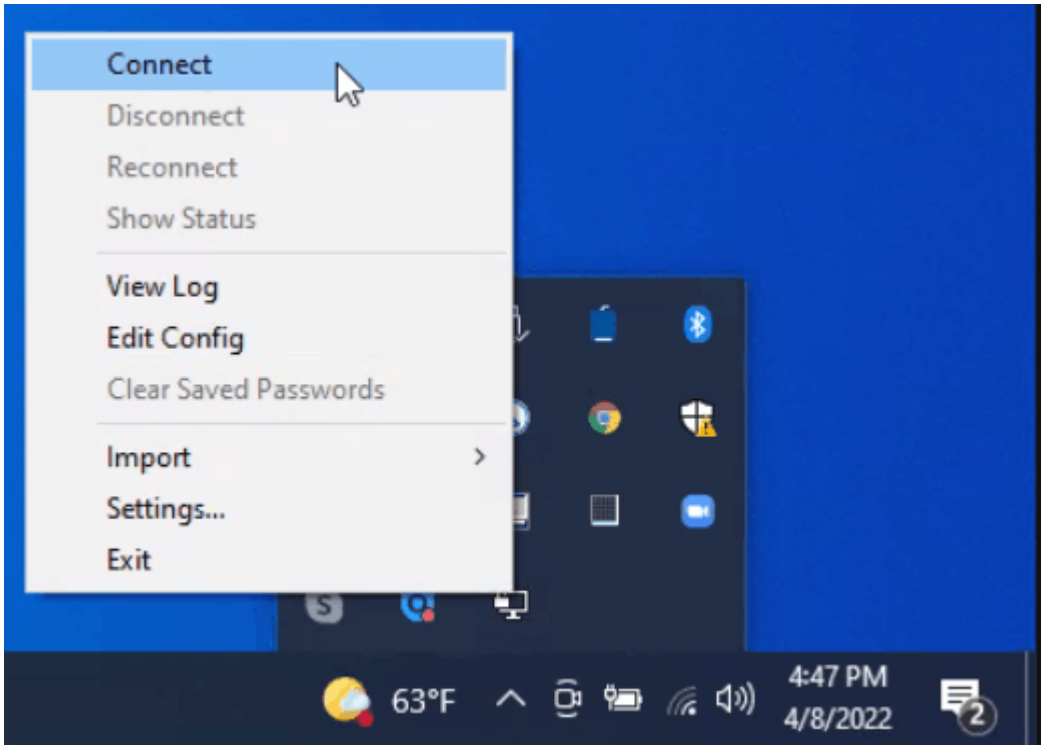
Figure 1-56: Select the OVPN file



8. Right-click on the OpenVPN GUI icon in the taskbar and select Connect.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

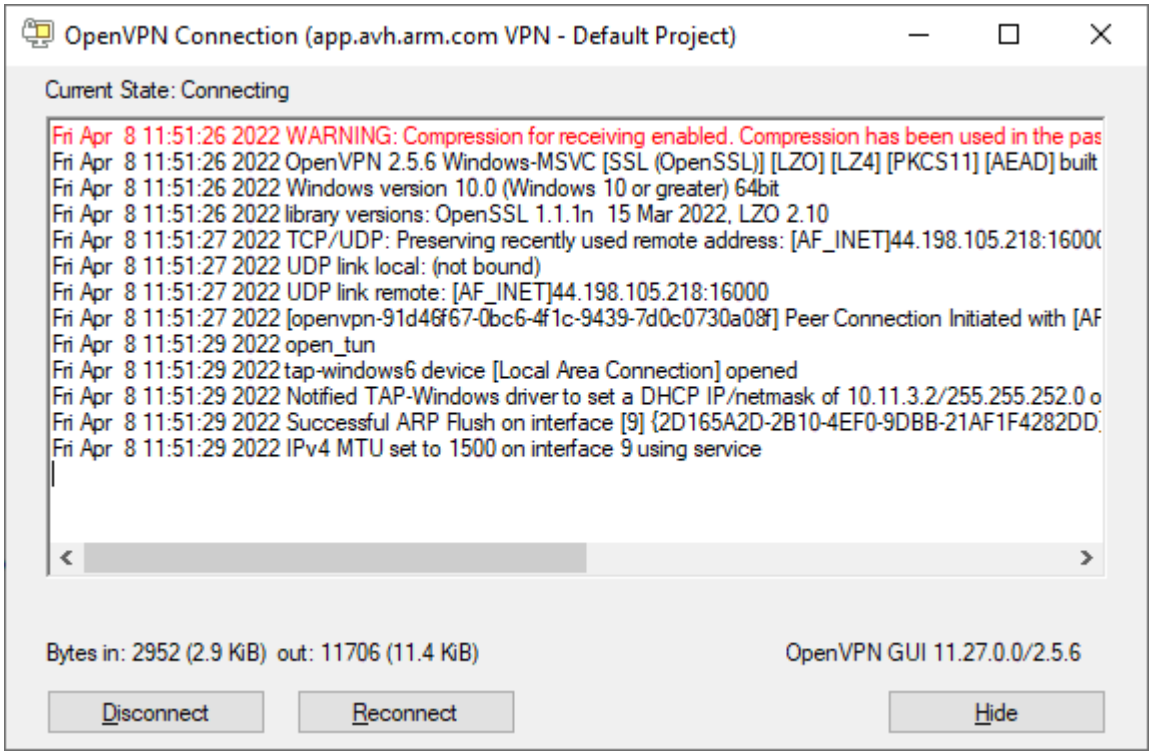
Figure 1-57: Connect



9. This window will appear and it will close when OpenVPN has connected to the device.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-58: Open VPN Connection



10. So now you have connected the VPN to your own host network you can SSH to the device or access the webserver.

Next try connecting to your local machine with SSH with the Connect via SSH tutorial.

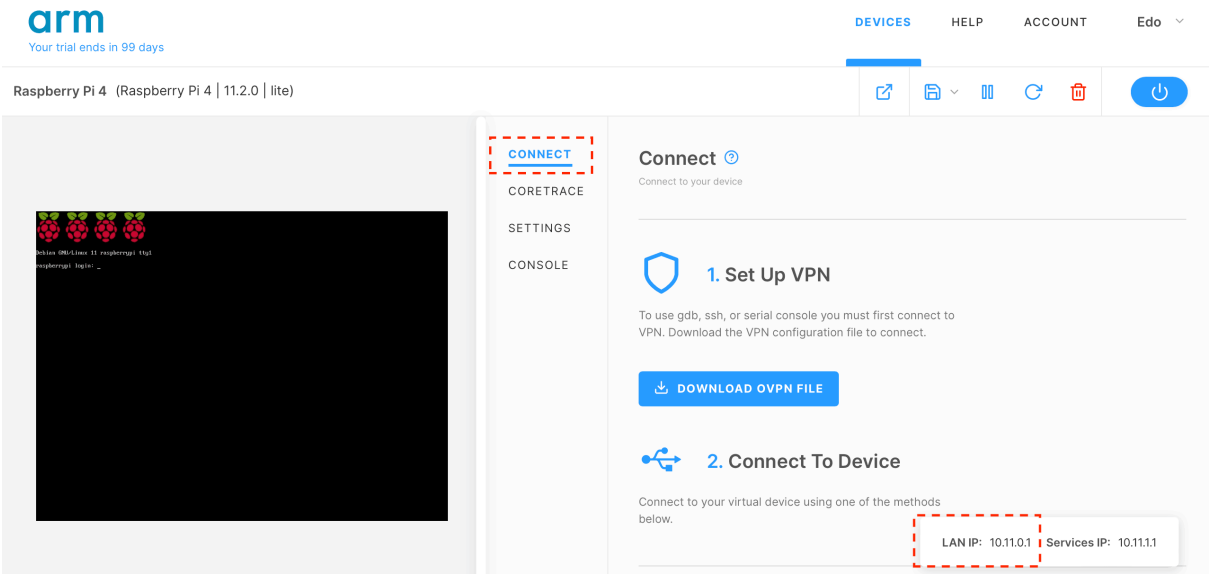
1.3.4 Connect via SSH

This tutorial will demonstrate how to connect a device with SSH.

1. To follow this tutorial we need a device set up with a VPN. If you haven't got a device set up, start with the Quickstart for Raspberry Pi 4 tutorial then follow the Connecting to the VPN tutorial. Otherwise use a device with the VPN already set up.
2. On your device page, click on "CONNECT" and copy the "LAN IP" address from the bottom right. In this example, it is `10.11.0.1`.

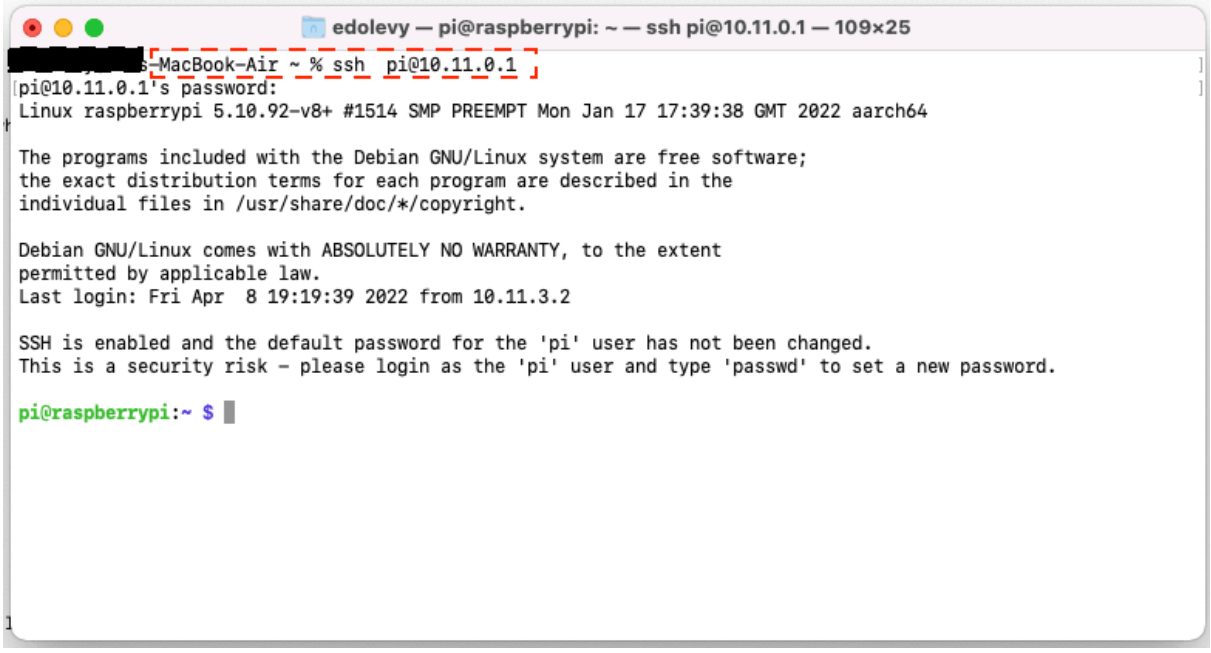
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-59: LAN IP address



3. Open a local terminal and paste the following command to connect: `ssh <Username>@<LAN IP>` In our example: `ssh pi@10.11.0.1` Then use your password to complete the connection, in this example the password is `raspberry`

Figure 1-60: Connect to SSH



4. You have now connected your virtual device to SSH.

Next, try Connect to the Console.

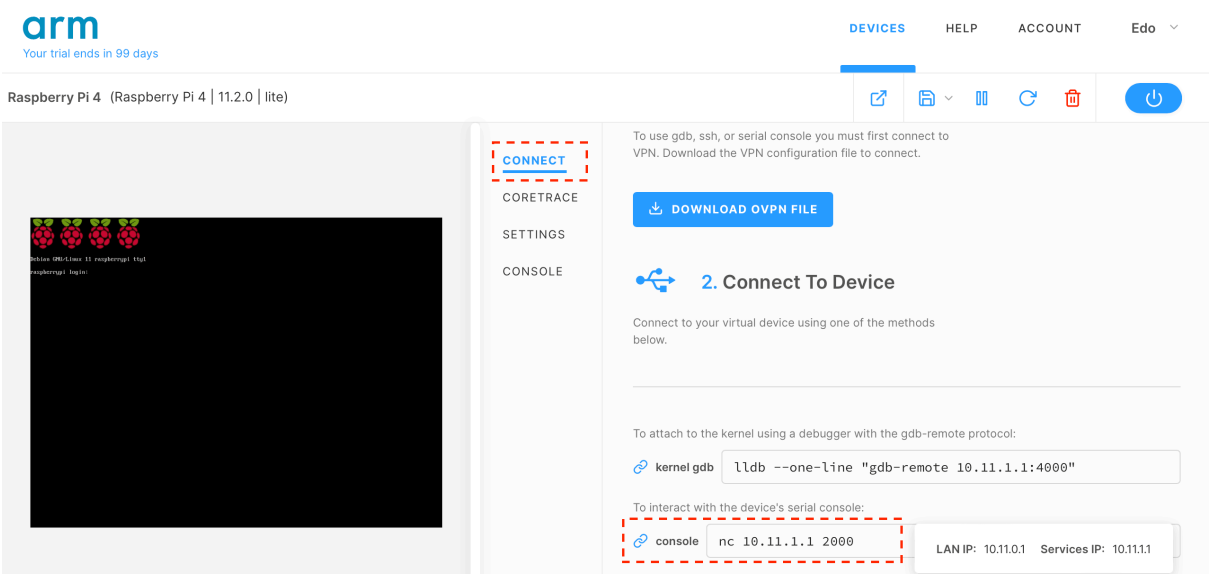
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.3.5 Connect to the Console

This tutorial will demonstrate how to connect to the console with a device with the VPN setup.

1. To follow this tutorial we need a device set up with a VPN. If you haven't created a device yet, start with the Quickstart for Raspberry Pi 4 tutorial then follow the Connecting to the VPN tutorial. Otherwise use a device with the VPN already set up.
2. On your device page, click on CONNECT copy the `console command: nc <device IP> <port>` in the example below it is `nc 10.11.1.1 2000`

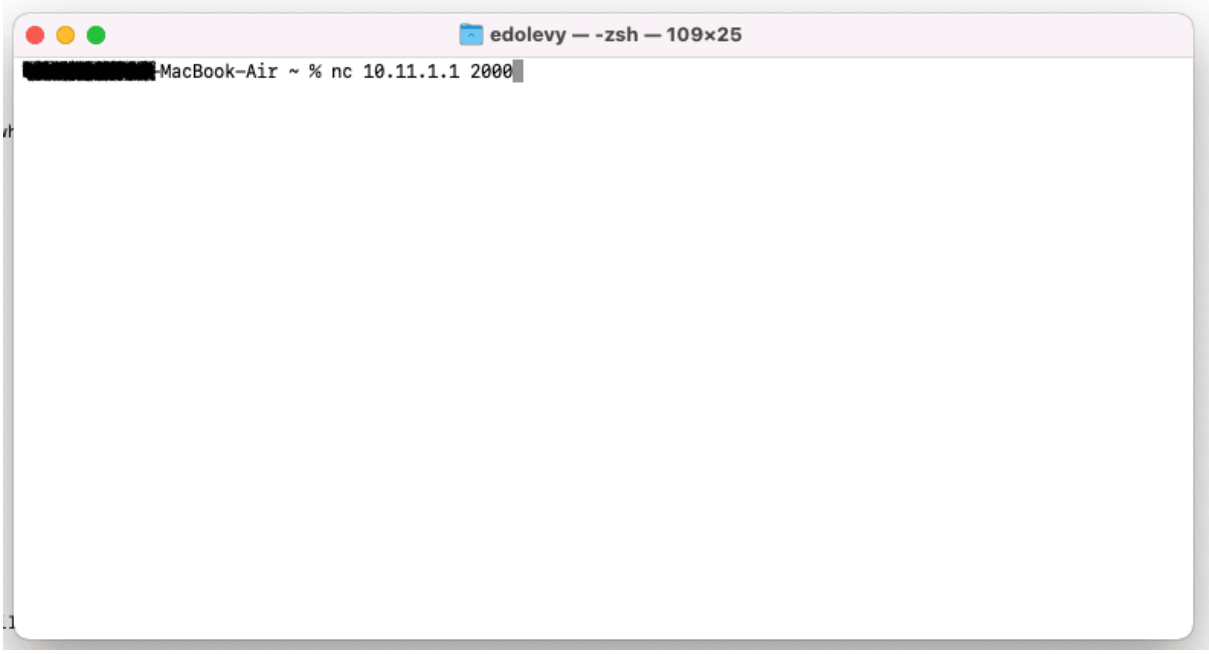
Figure 1-61: Connect



3. Paste the command into your shell (or shell application) and hit `return`.

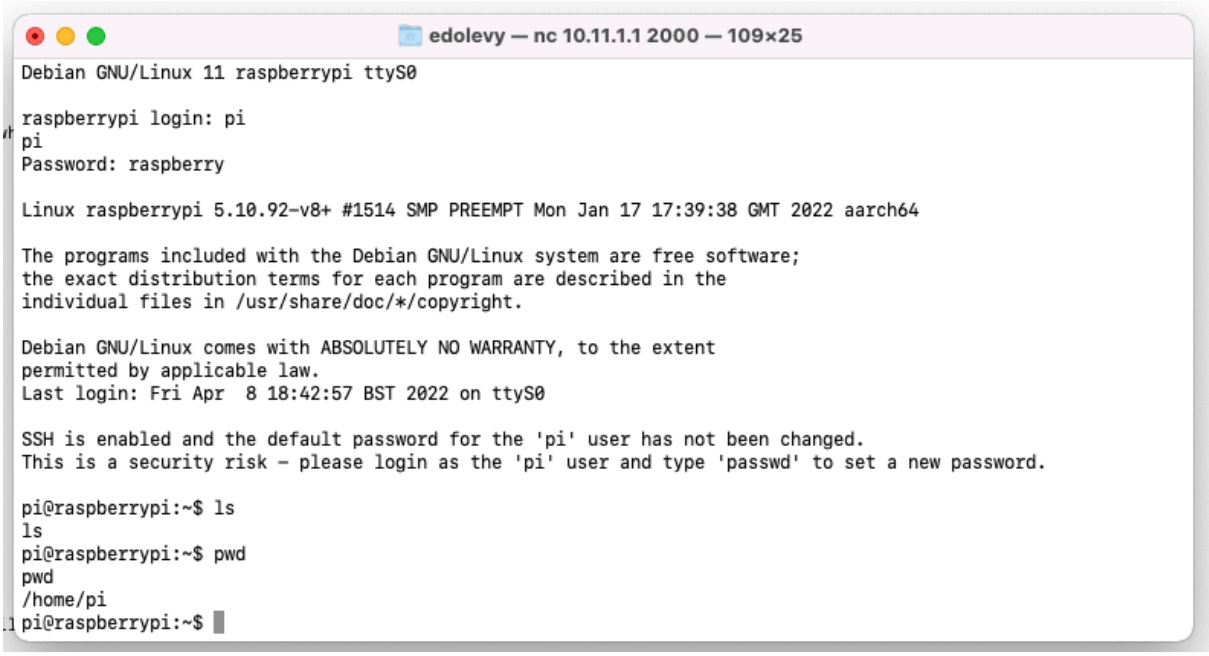
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-62: Paste the command



4. The first time logging (and after reboot) in you will need to enter the login credentials. In this example, we are using a Raspberry Pi4 device with the default credentials. User: `pi` Password: `raspberry`
5. You are now logged in to the device's console!

Figure 1-63: Console



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1.3.5.1 Next Steps

You can explore the AVH Examples within this knowledgebase.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2. Examples

The following articles are available:

- Capture Input from the Console
- iMX8m Arm Cortex Complex Examples
- STM32U5 IoT Discovery Kit Examples
- Raspberry Pi 4 Example
- API Examples

2.1 Capture Input from the Console

Learn how to capture character and newline console input.

2.1.1 Retrieving Character Input from the User

It is possible to capture character input from the console using the scanf C library function.

For example, capture a character input using

```
scanf("%c", &input_char);
```

2.1.2 Detecting When the User Presses Return

Our UI console sends a new line (`\n`) when you hit return but doesn't send a carriage return (`\r`). When detecting a new line in code, use `\n` rather than `\r`.

For example the following will run, whereas using `\r` will hang:

```
while ( getchar() != '\n');
```

2.1.3 Modify How the Terminal Handles "newline"

Alternatively, you can adjust the terminal settings to not map CR to NL using the command `stty -icrnl`.

You can then send a `\r` by pressing Ctrl+M or `\n` by pressing Ctrl+J.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2.2 iMX8m Arm Cortex Complex Examples

iMX8m Arm Cortex Complex examples:

- Blink the Led on the iMX8m Arm Cortex Complex device

2.2.1 Blink the Led on the iMX8m Arm Cortex Complex device

This tutorial will demonstrate how to blink the LED in the iMX8m Arm Cortex Complex device. It follows from the Quickstart for iMX8m Arm Cortex Complex.

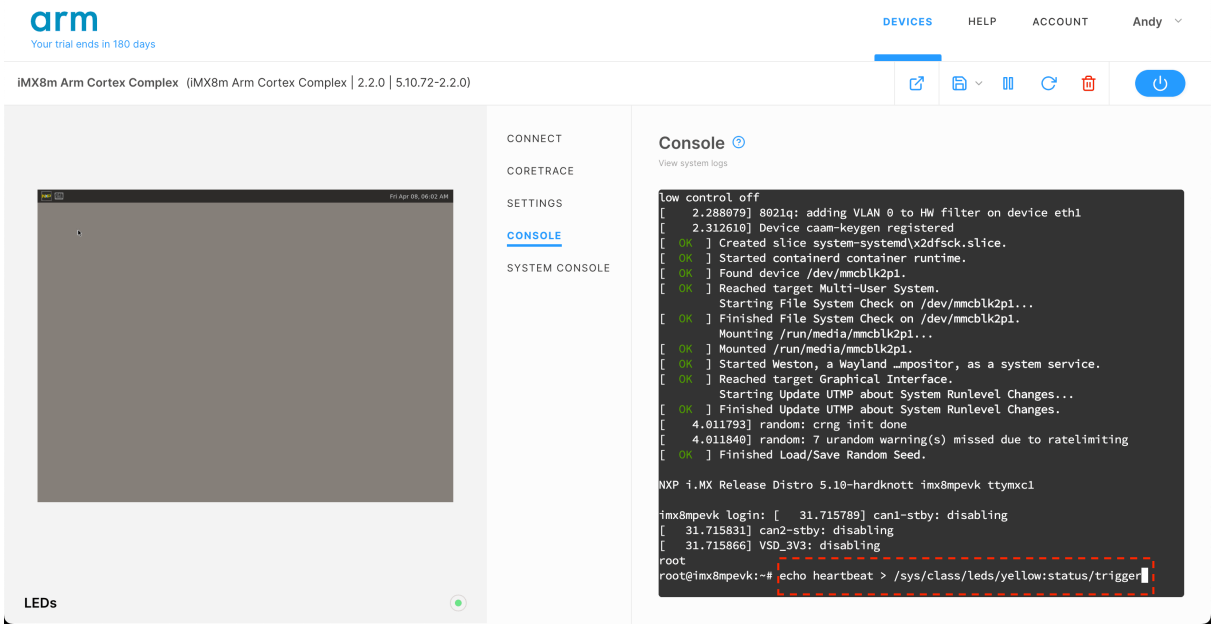
2.2.1.1 Blink the LED

To blink the LED, do the following:

1. Following on from the Quickstart for iMX8m Arm Cortex Complex tutorial OR go to a iMX8m Arm Cortex Complex device that has the console logged in.
2. To blink the LED type `echo heartbeat > /sys/class/leds/yellow:status/trigger` into the console.

   The LED representation is at the bottom of the desktop. In the current state it is on and solid.

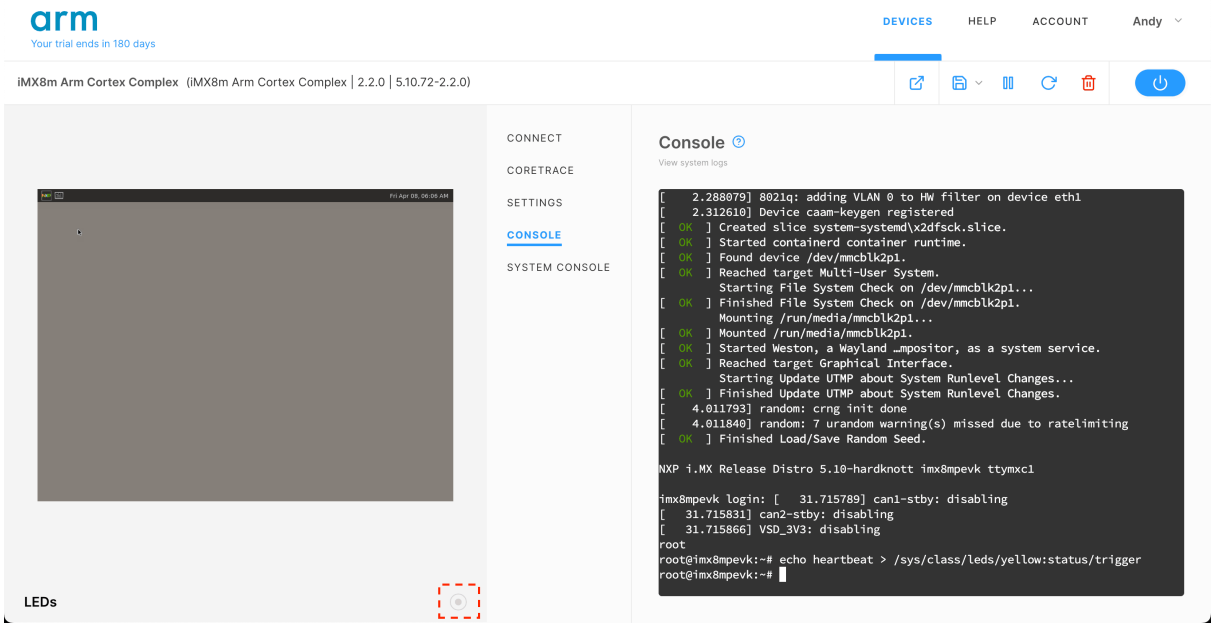
   Figure 2-1: LED on and solid



3. The LED will now be blinking.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-2: LED blinking



Find out more

2.3 STM32U5 IoT Discovery Kit Examples

STM32U5 IoT Discovery Kit examples:

- STM32CubeU5 TFM example
- STM32CubeU5 WiFiBasics example

2.3.1 STM32CubeU5 TFM example

STM32U5 Trusted Firmware M tests firmware package example.

For this example the kernel component in the firmware package is itself a ZIP archive, containing various binaries and `load.txt` file describing their location in memory:

```
name:tfm_s_init.bin load:0xc024000
name:tfm_ns_init.bin load:0xc052000
name:loader.bin load:0xc1fa000
name:TFM_SBSFU_Boot.bin load:0xc002000
```

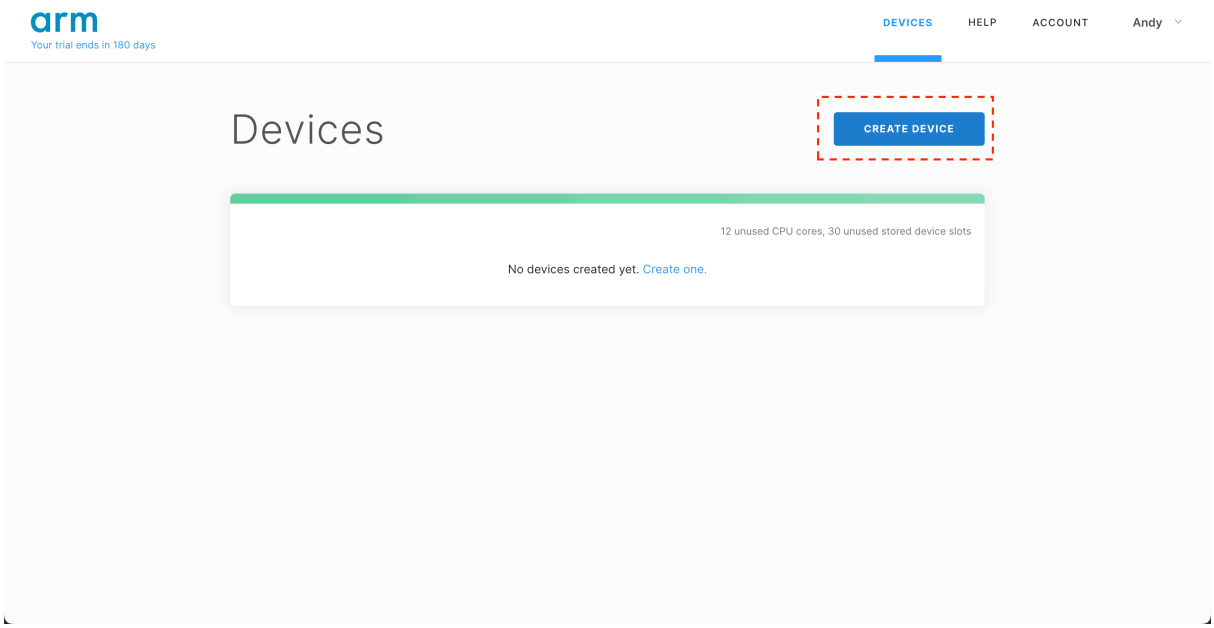
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2.3.1.1 Creating your device

To create your device:

1. Log in to AVH with your Arm user account. If you do not have or need to create an Arm account, register at https://developer.arm.com/.
2. When logged in you land on the AVH Devices page. Click CREATE DEVICE.

Figure 2-3: Click Create Device



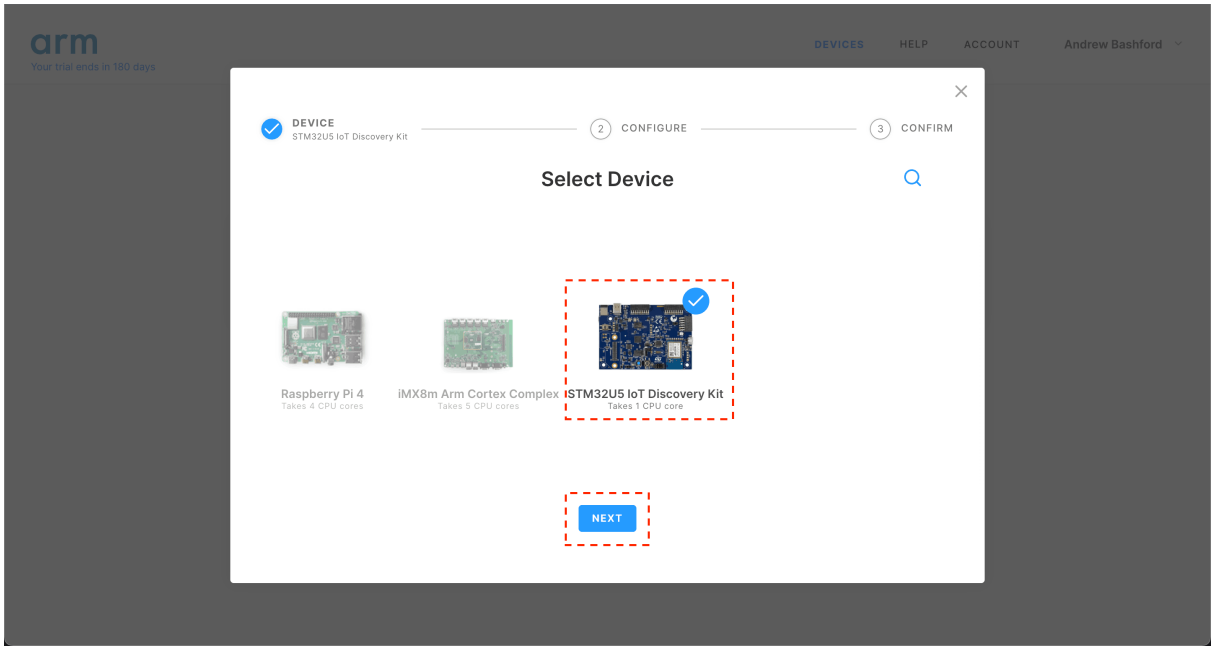
3. Click on the STM32U5 IoT Discovery Kit to select it. Then click NEXT.

Note

You require 1 free CPU core to build this device or it will be disabled.

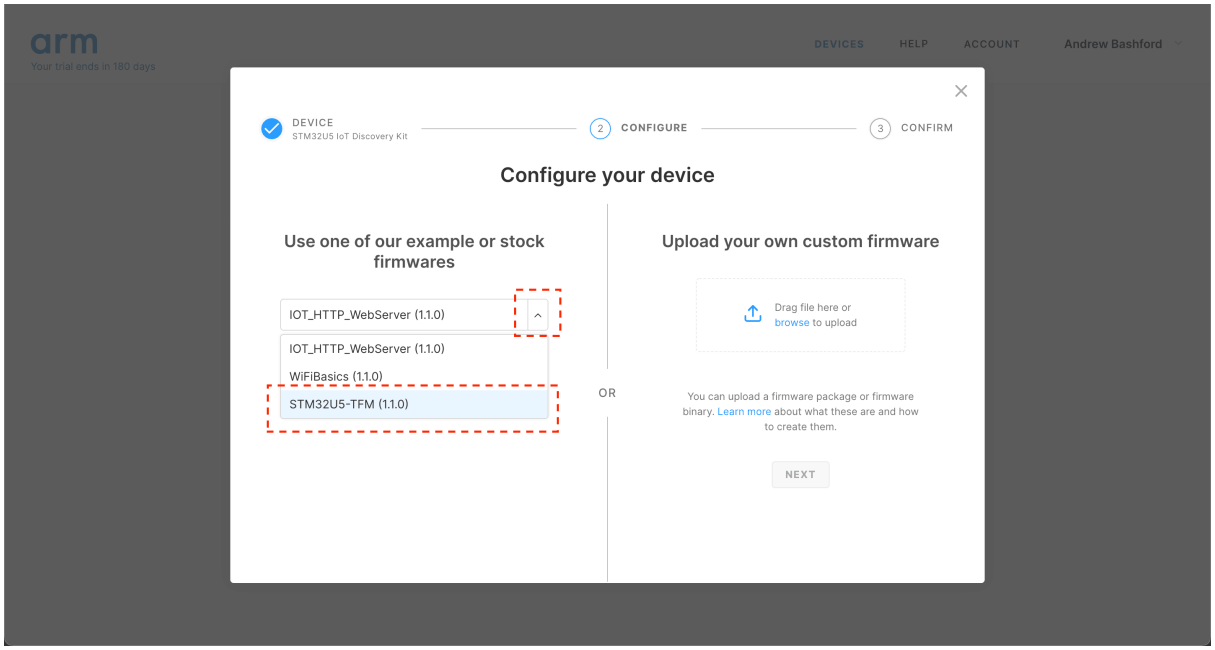
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-4: Select device



4. In this next step, you'll see a list of pre-configured examples/stock firmware or you can upload your own custom firmware, see Understanding Firmware on AVH for more information. In this example, we will use the selection `STM32U5-TFM (1.1.0)` in the stock firmware section.

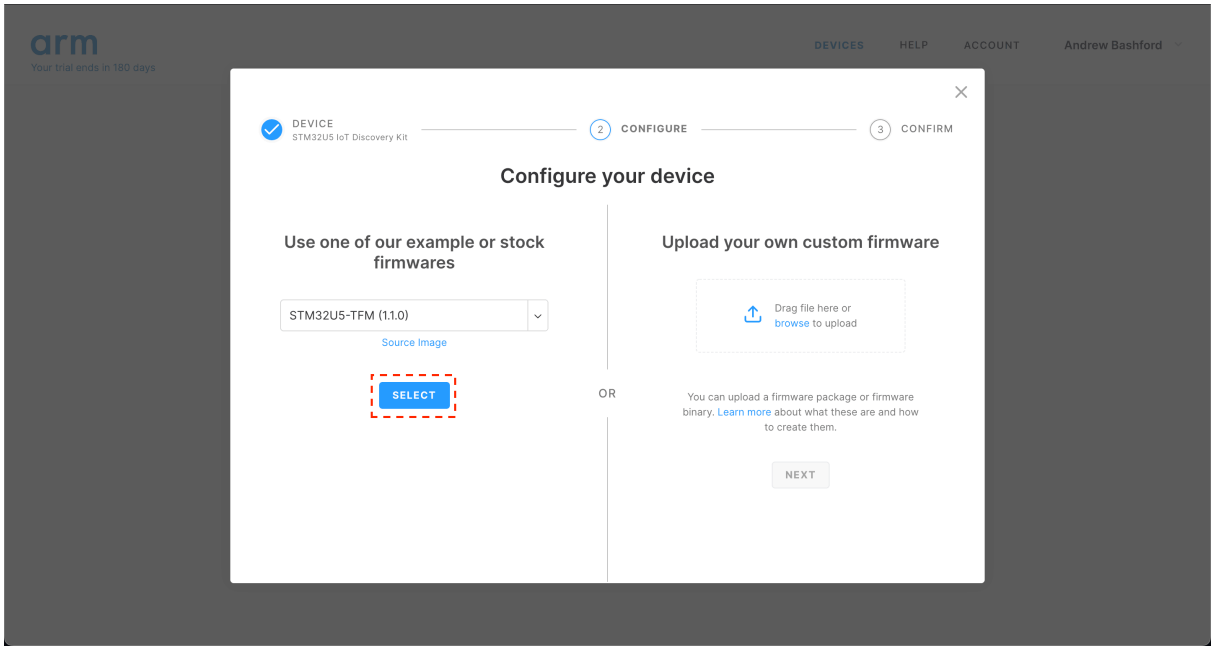
Figure 2-5: Configure your device



5. Click SELECT.

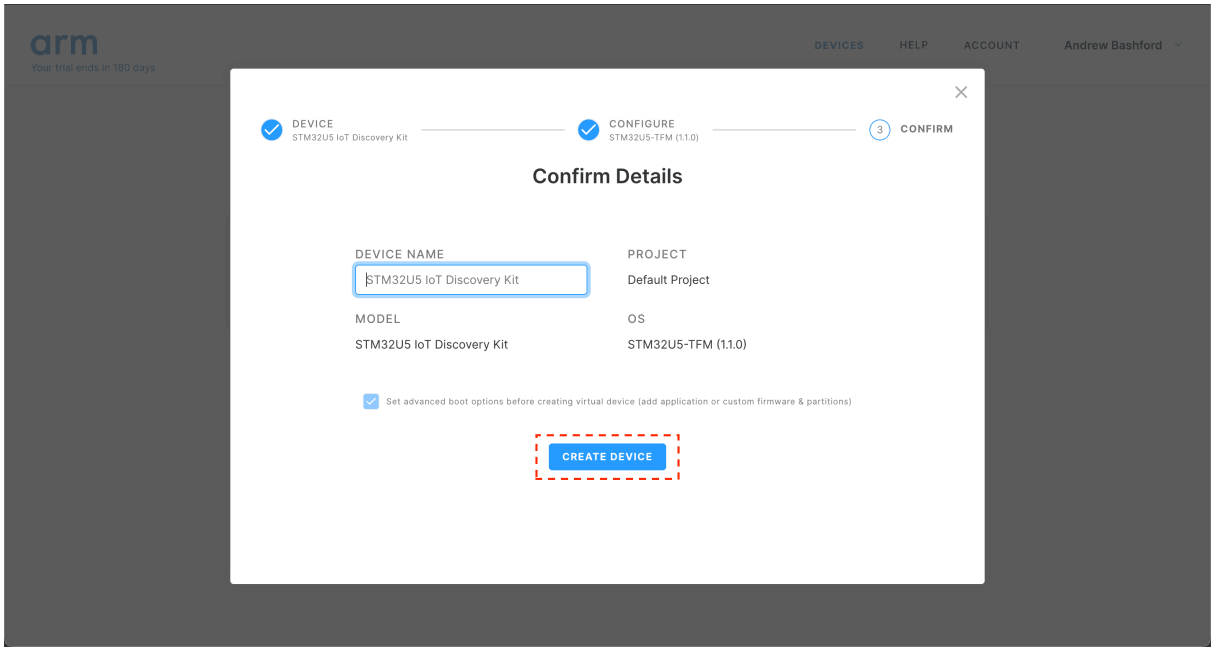
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-6: Click Select



6. In the confirmation screen, click CREATE DEVICE using the default name and the advance boot options will be checked by default.

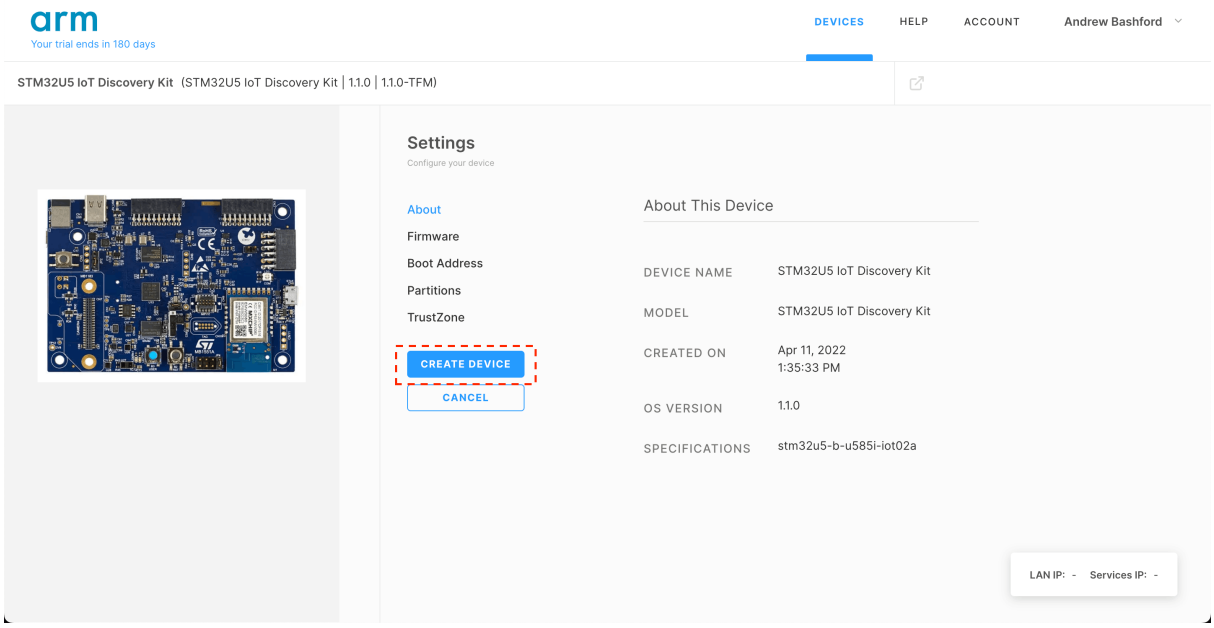
Figure 2-7: Confirm details



7. In the advanced boot options, we will just use the default. Click CREATE DEVICE.

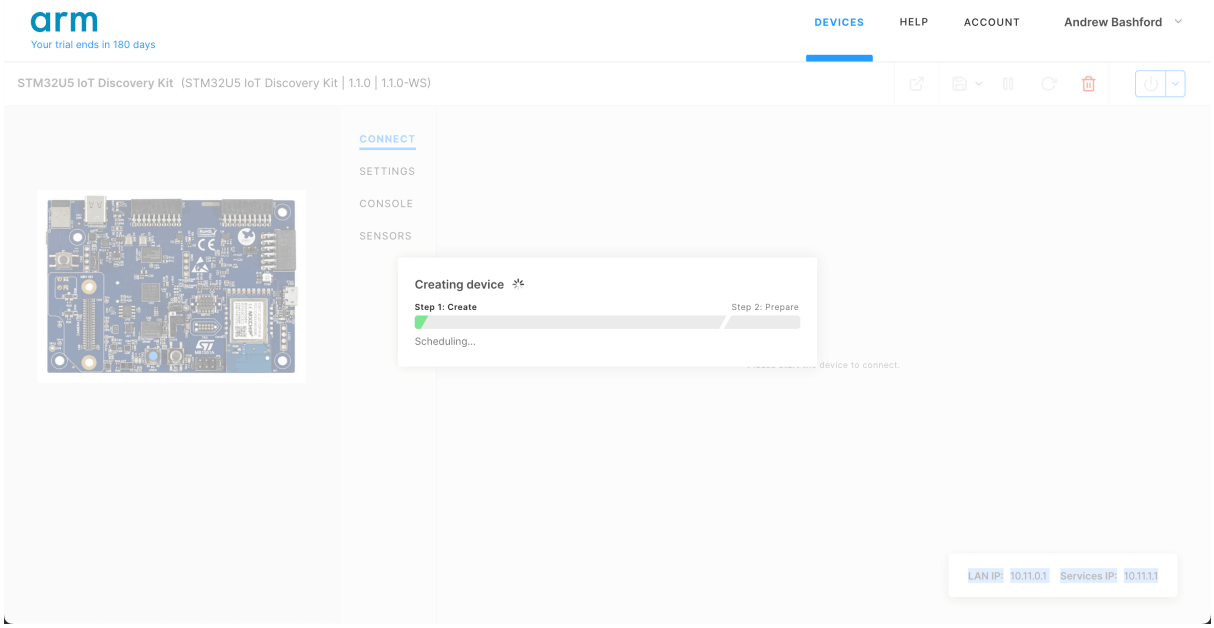
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-8: Settings



8. The device will then be created. It takes a few moments...

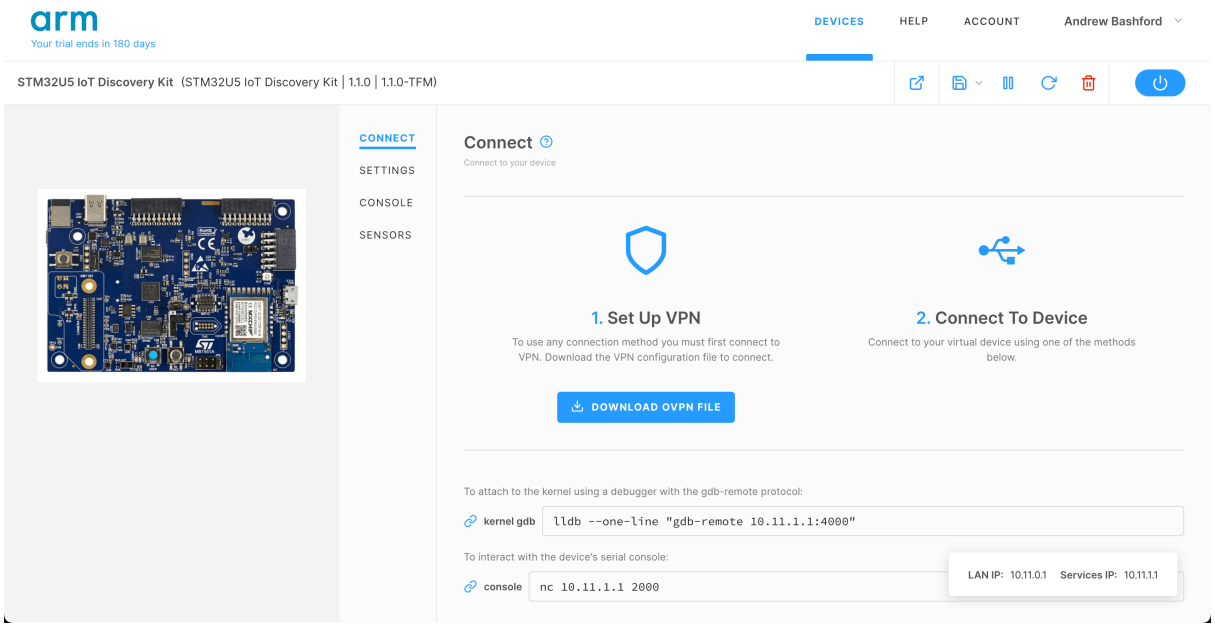
Figure 2-9: Creating device



9. Your STM32U5 IoT Discovery Kit is now ready to go!

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-10: Connect



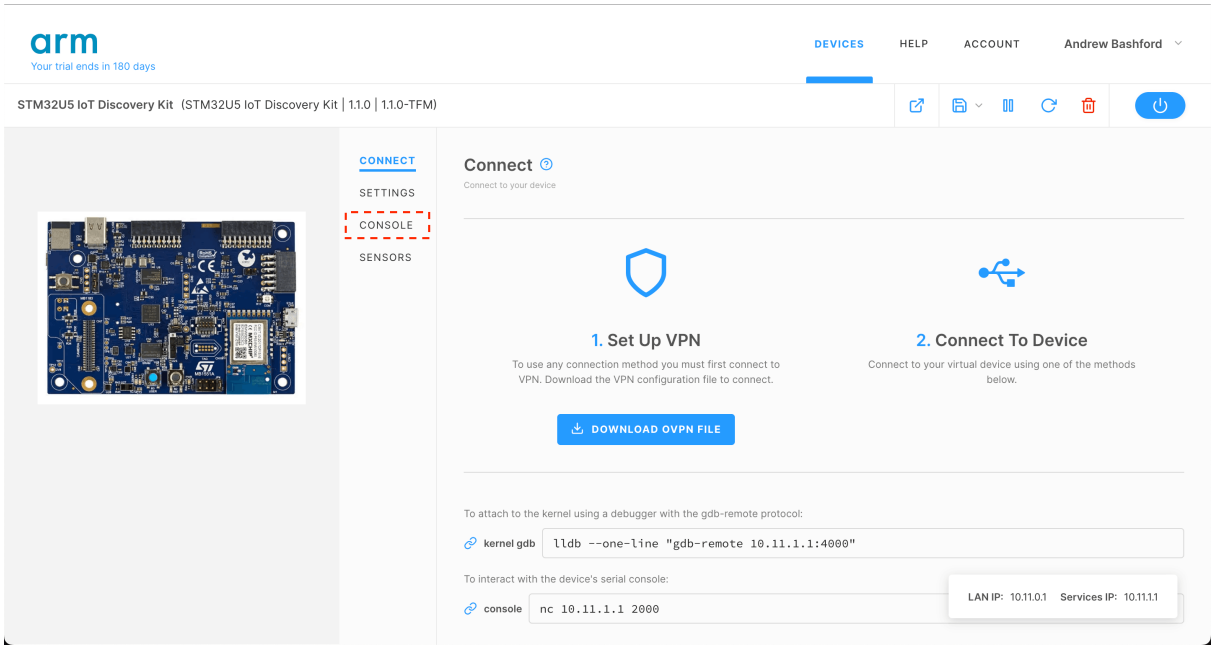
2.3.1.2 Running the example

To run the example:

1. Click on CONSOLE.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-11: Click Console



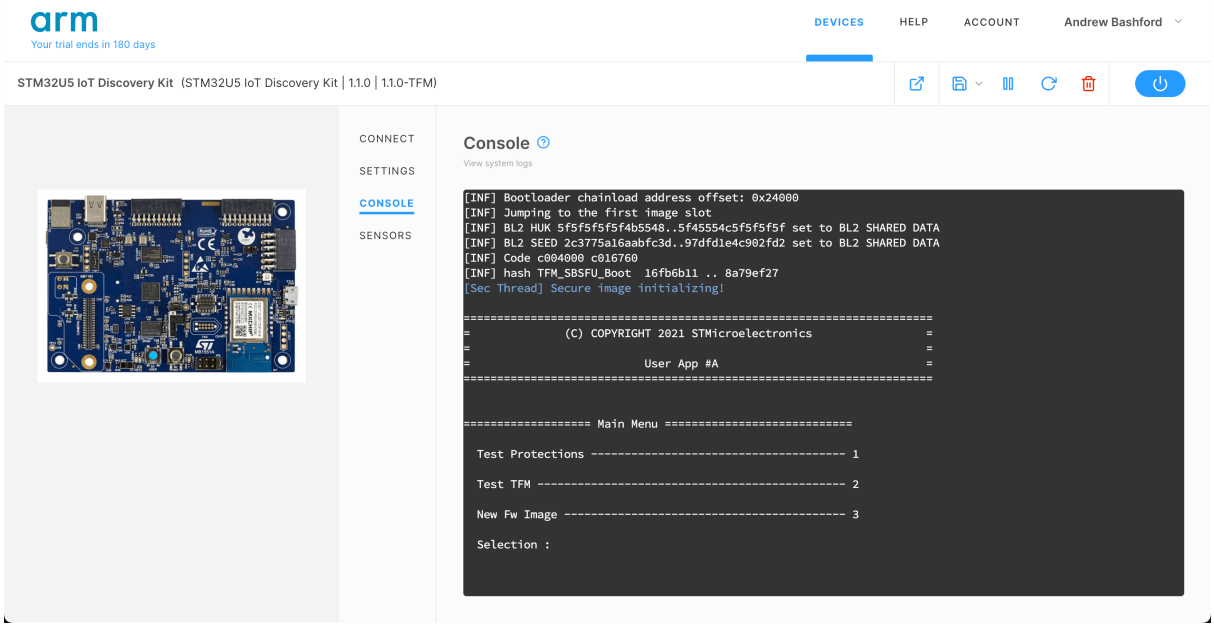
2. There is a console menu to run various tests using Trusted Firmware. For example type 2 (Test TFM) followed by return.

Note

You may see `Invalid Number !` also menus printed twice this is because the device is interpreting the return keypress as a number. Please ignore this.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-12: Running tests on the console



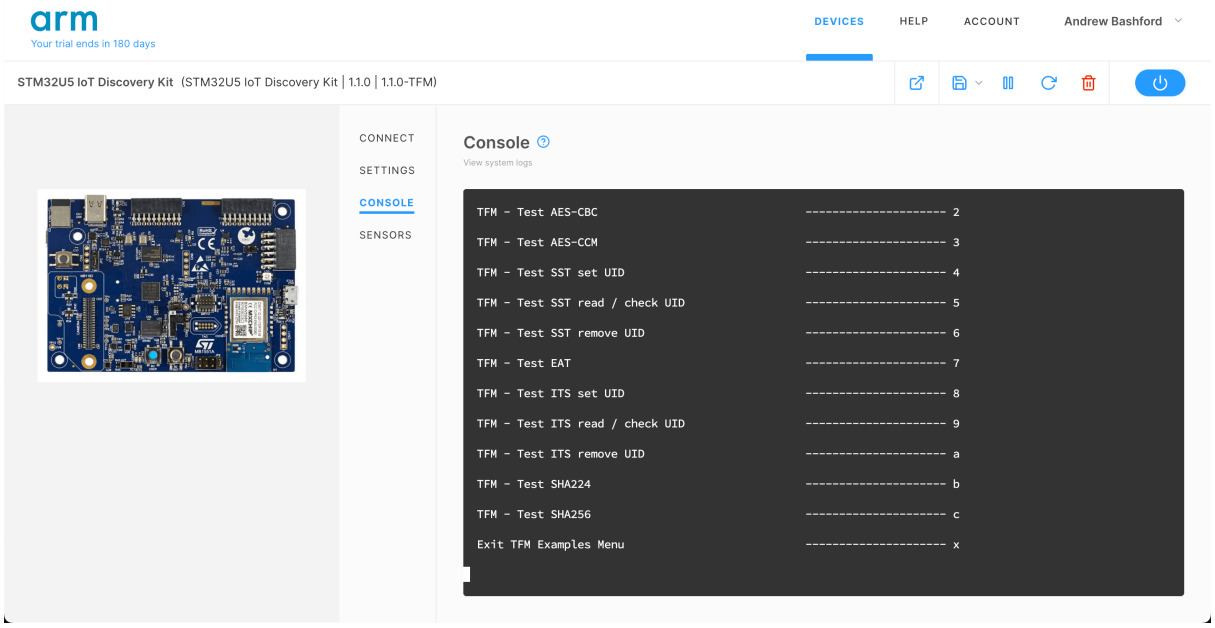
3. Then type `0` ("TFM - Test All") followed by `return`. to start tests.

Note

You may need to scroll up to see all of the menu.

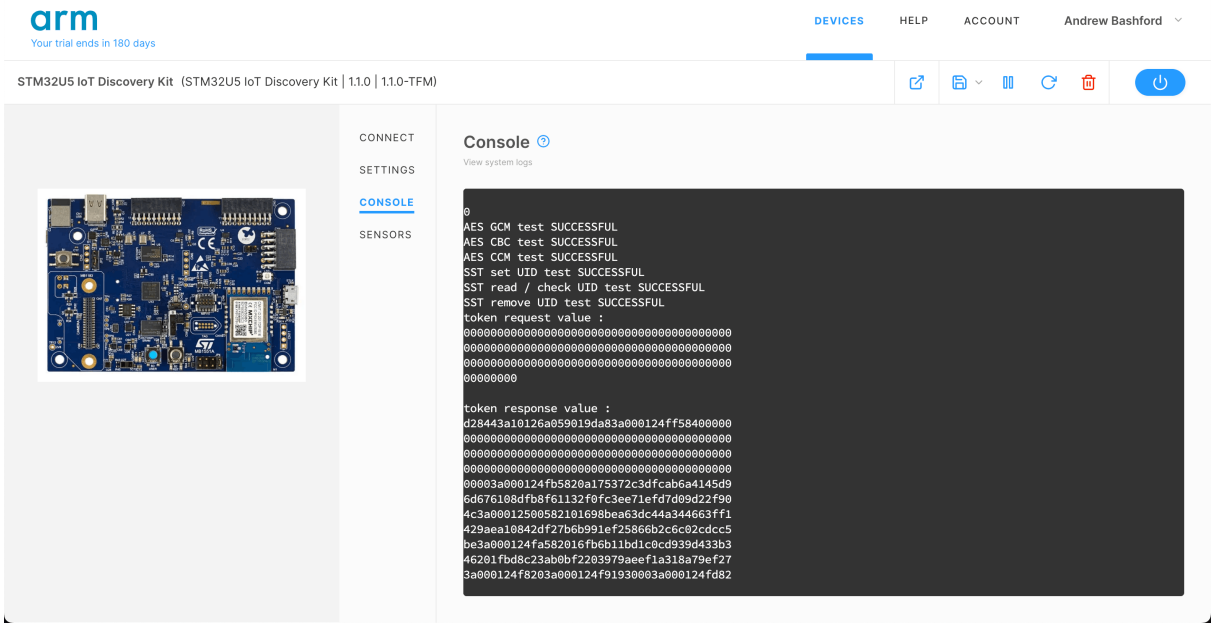
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-13: TFM - Test All



4. You will be able to observe the result in the Console by scrolling up.

Figure 2-14: Test results



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2.3.1.3 Next steps

Other STM32U5 IoT Discovery Kit examples;

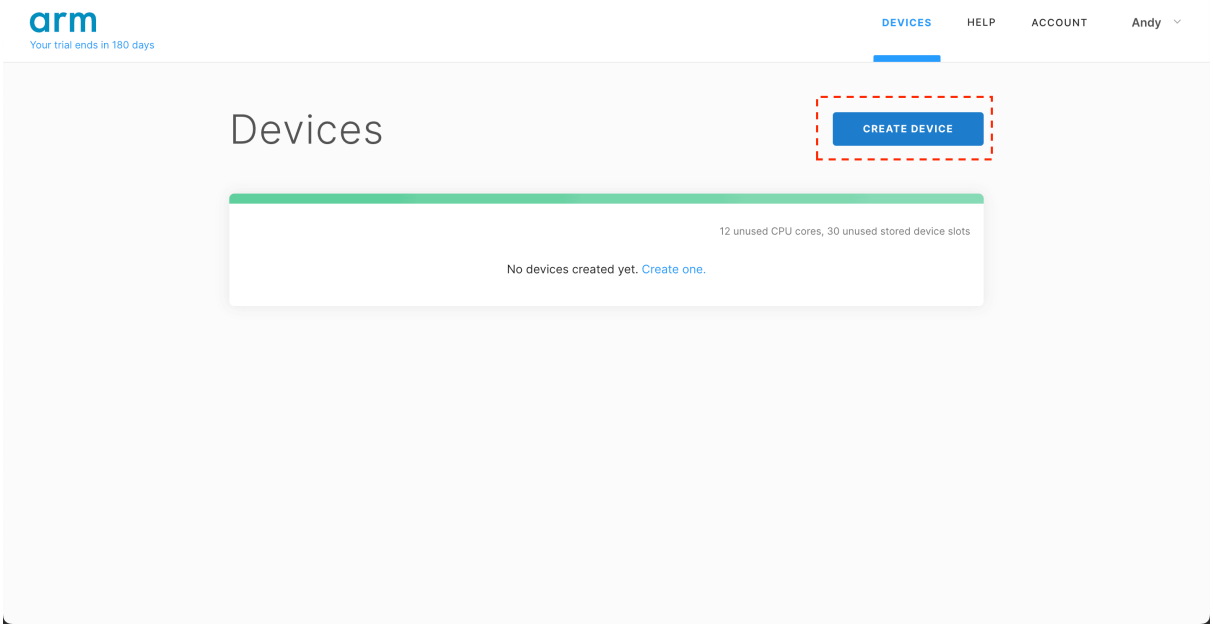
- STM32CubeU5 WiFiBasics example
- STM32CubeU5 IOT_HTTP_WebServer (This is the Quickstart for STM32U5 IoT Discovery Kit)

2.3.2 STM32CubeU5 WiFiBasics example

This example will test basic WiFi networking functions:

1. Log in to AVH with your Arm user account. If you do not have or need to create an Arm account, register at https://developer.arm.com/.
2. When logged in you land on the AVH Devices page. Click CREATE DEVICE.

Figure 2-15: Create device



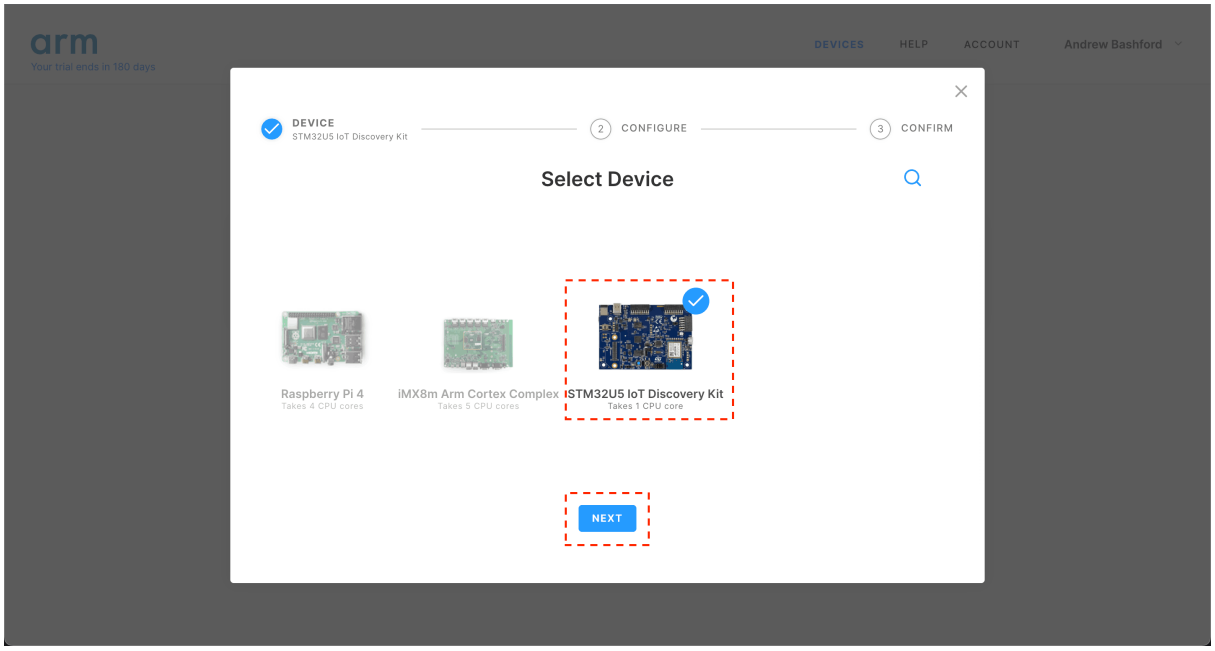
3. Click on the STM32U5 IoT Discovery Kit to select it. Then click NEXT.

Note

You require 1 free CPU core to build this device or it will be disabled.

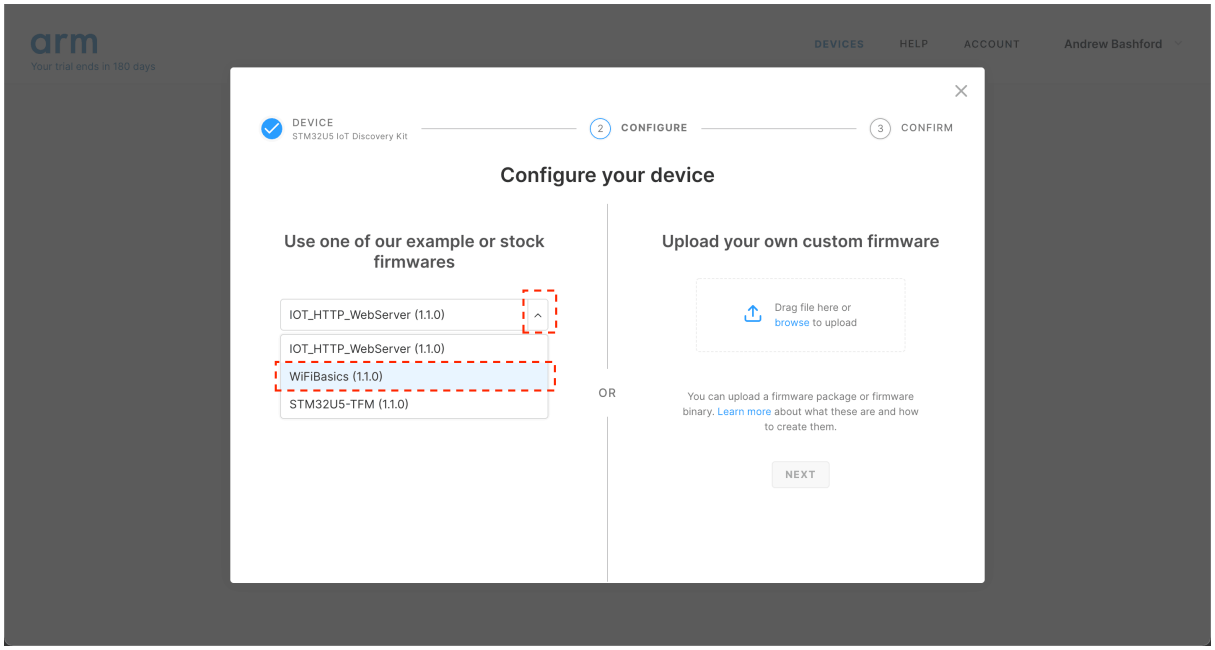
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-16: Select device



4. In this next step, you will see a list of pre-configured examples/stock firmware or you can upload your own custom firmware, see Understanding Firmware on AVH for more information. In this example, we will use the selection `WiFiBasics (1.1.0)` in the stock firmware section.

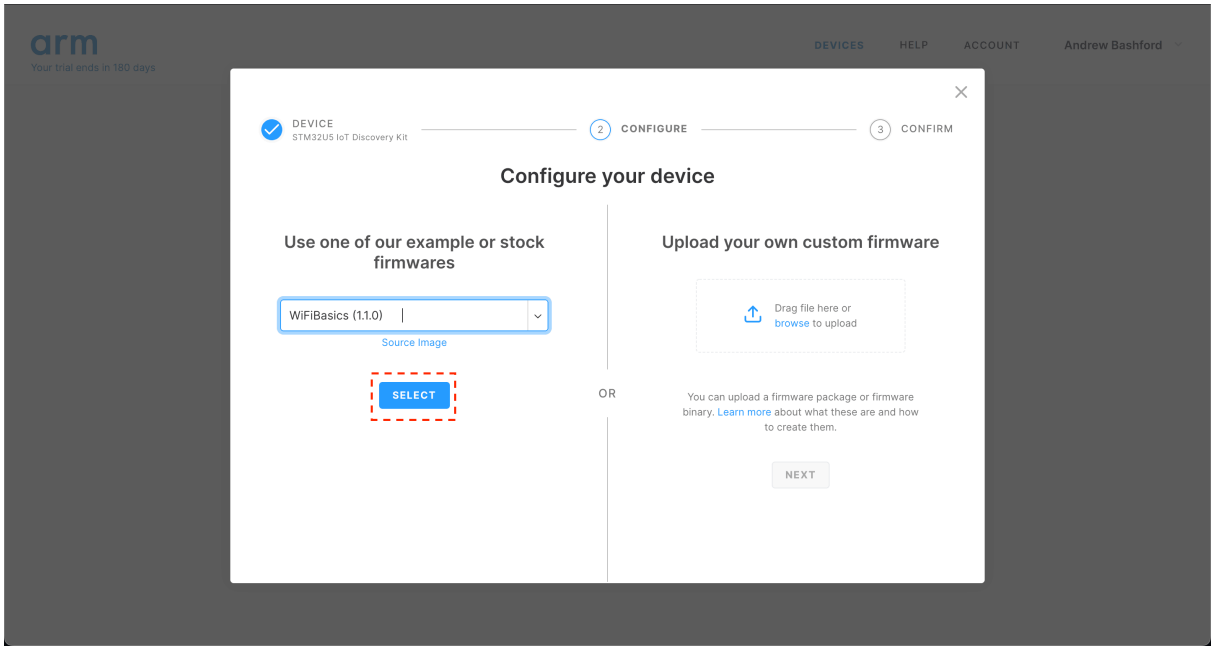
Figure 2-17: Configure your device



5. Click SELECT.

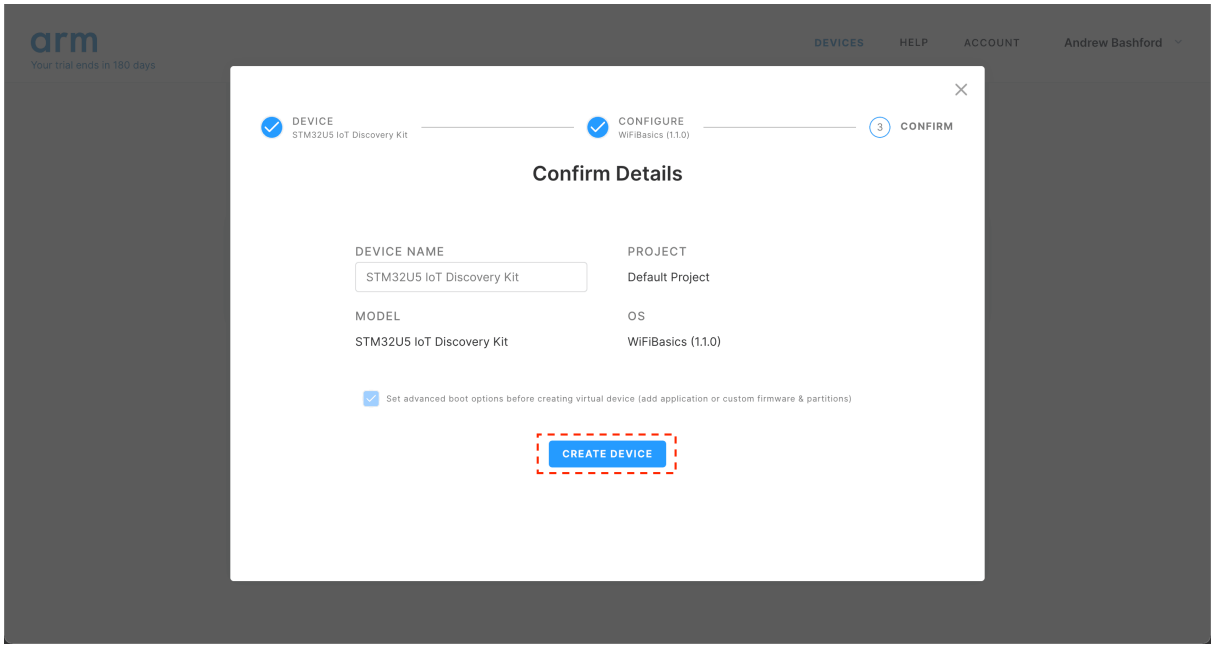
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-18: Configure your device



6. In the confirmation screen, click CREATE DEVICE using the default name and the advance boot options will be checked by default.

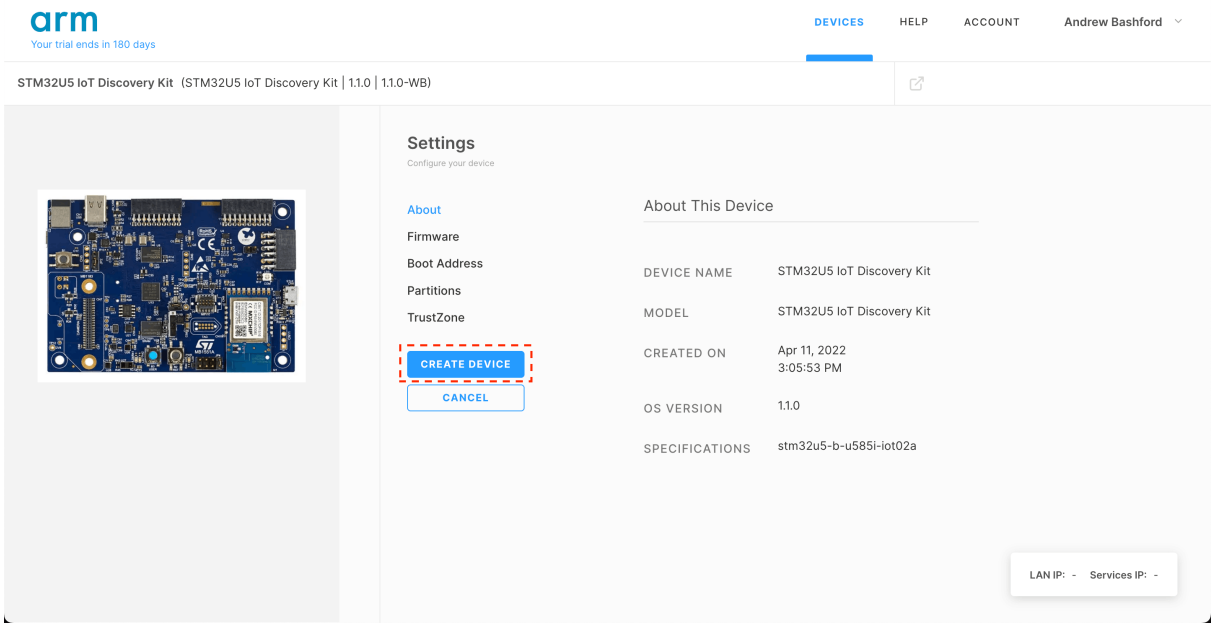
Figure 2-19: Confirm Details



7. In the advanced boot options, we will just use the default. Click CREATE DEVICE.

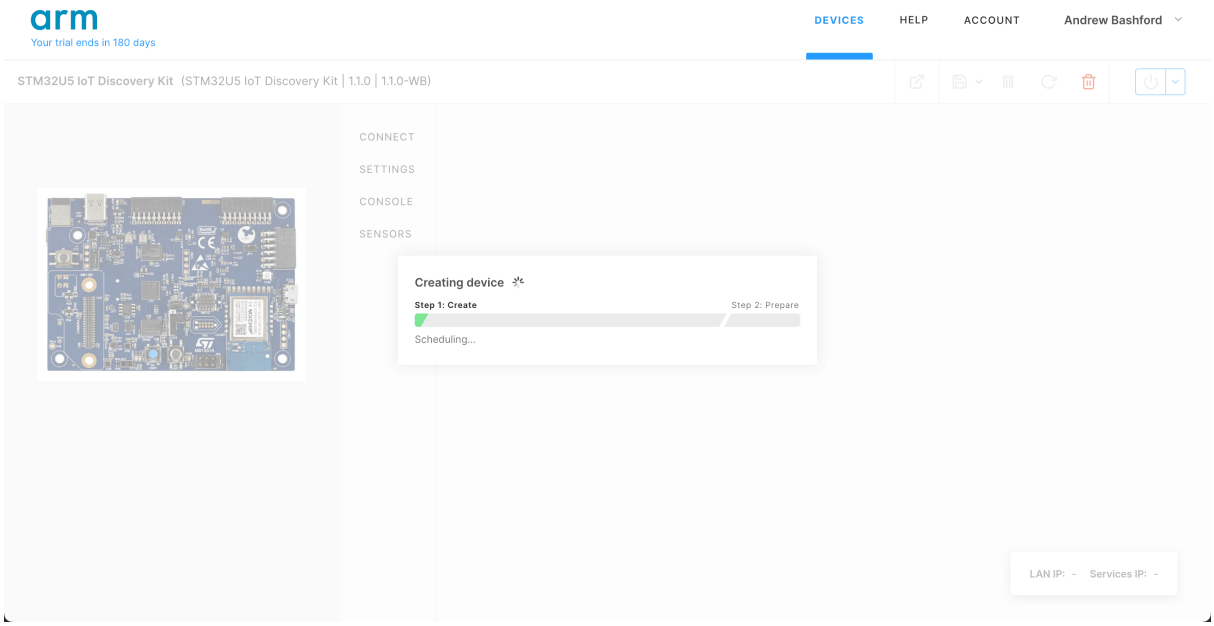
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-20: Settings



8. The device will then be created. It takes a few moments...

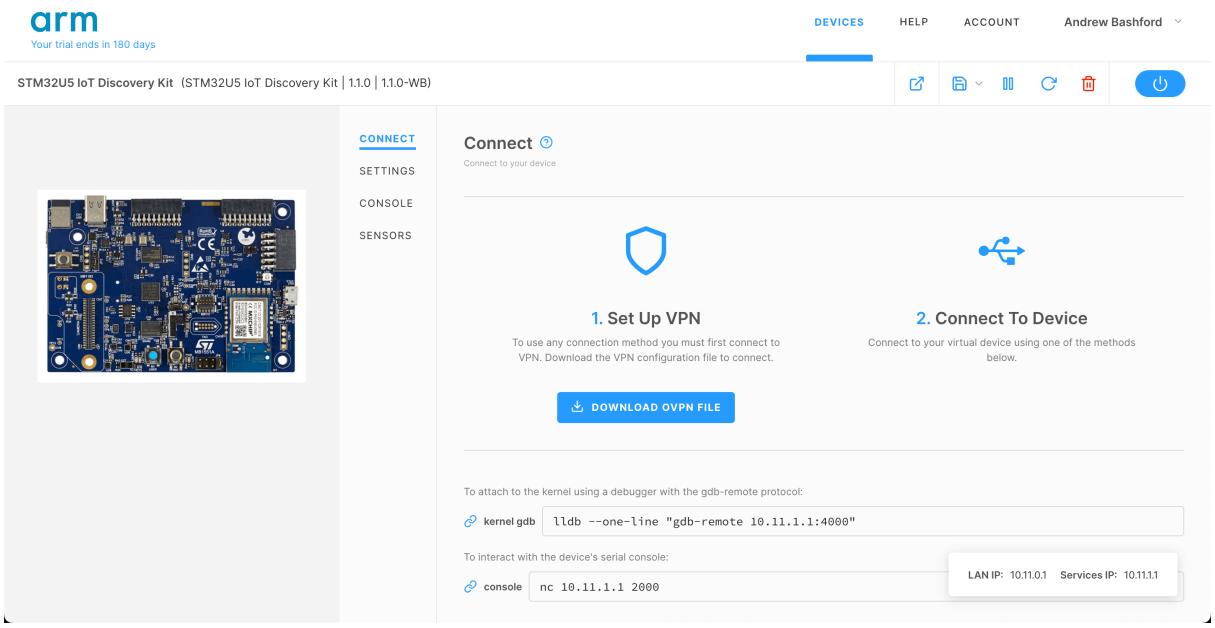
Figure 2-21: Creating device



9. Your STM32U5 IoT Discovery Kit is now ready to go!

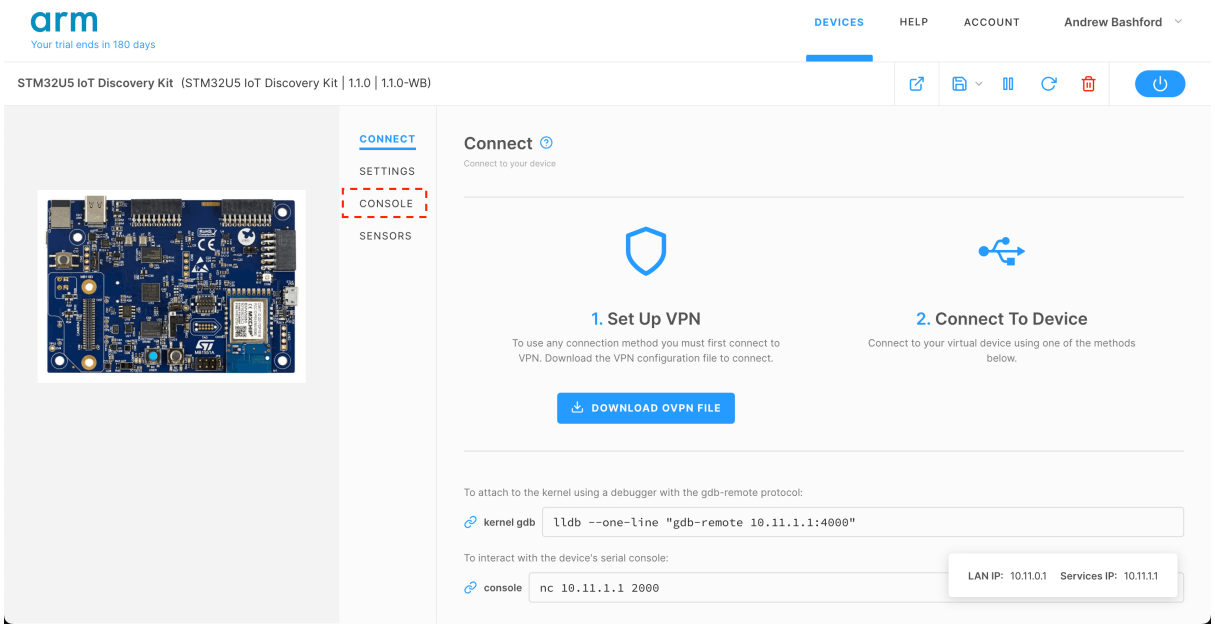
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-22: Connect



10. Click CONSOLE to go to the Console.

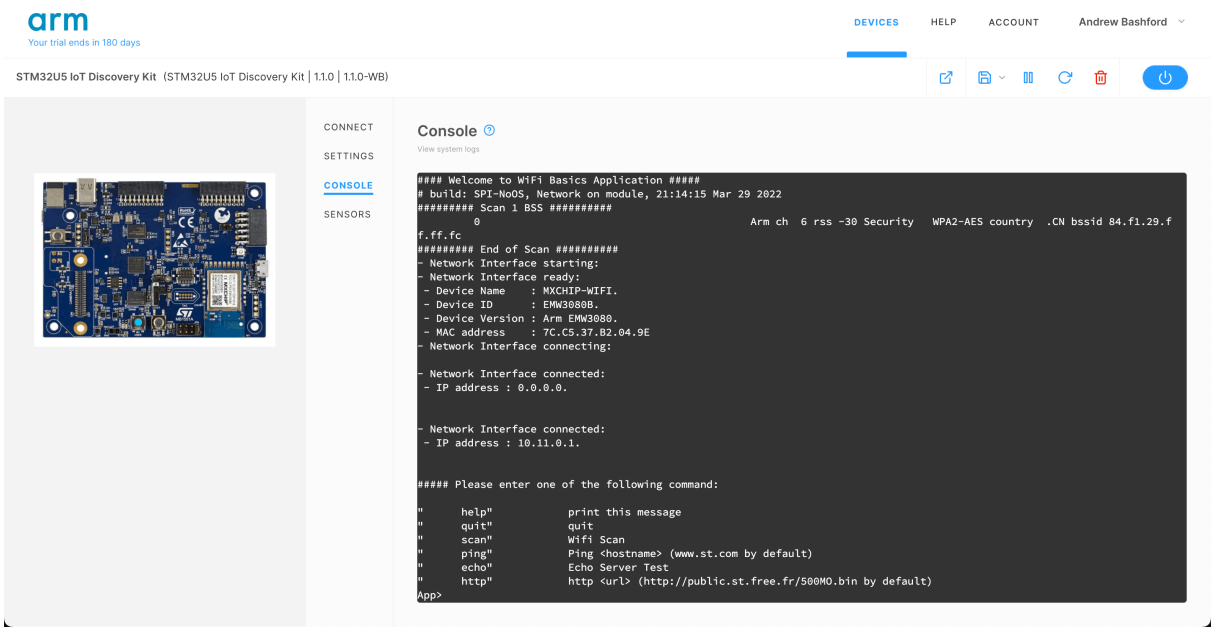
Figure 2-23: Click CONSOLE



11. Model scans for the available WiFi network and connects to the default one ("arm" in this case). There is a console menu to test various functions. For example, `http` command downloads binary from the ST website.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-24: Console



2.4 Raspberry Pi 4 Example

Raspberry Pi examples:

- Raspberry Pi vMMIO example

2.4.1 Raspberry Pi vMMIO example

This example will exercise vMMIO using the RaspberryPi model by demonstrating an external (over the network) MMIO handler and a local (RaspberryPi userspace) driver.

Read more in the AVH vMMIO Introduction article.

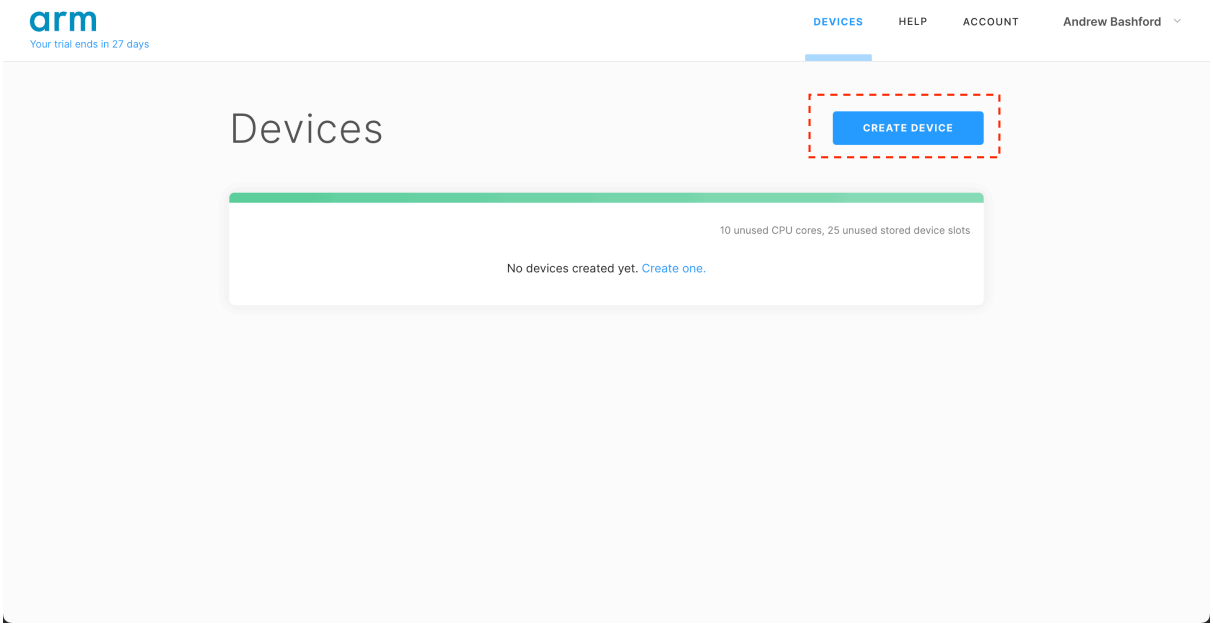
2.4.1.1 Creating the VM

To create the VM:

1. Create a Raspberry Pi 4 Virtual Machine. In DEVICES in the menu click on CREATE DEVICE.

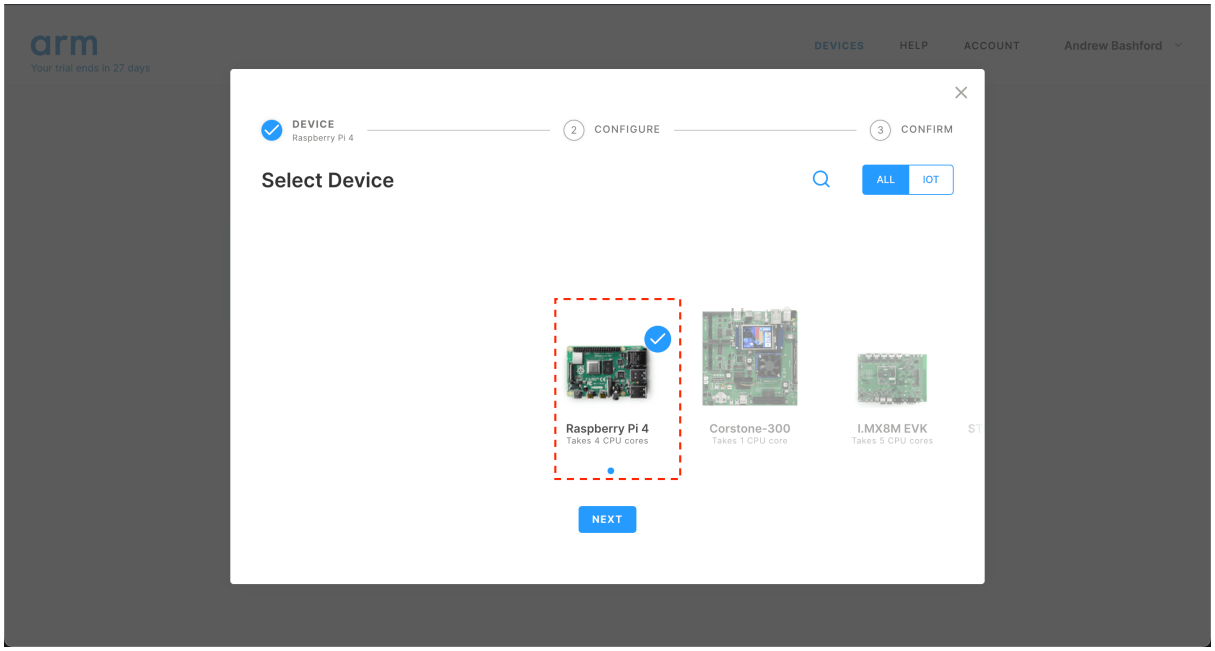
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-25: Create device



2. Click on the RPi VM device then click NEXT.

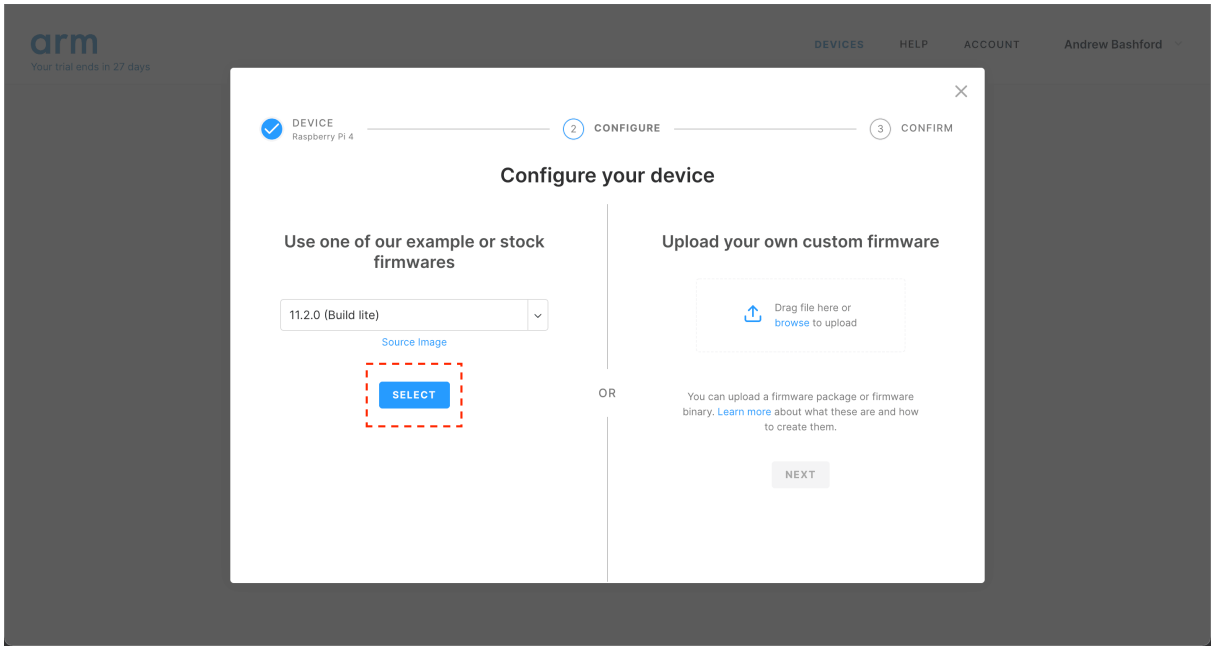
Figure 2-26: Select device



3. Select the `lite` firmware option if not already selected. Then click SELECT.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

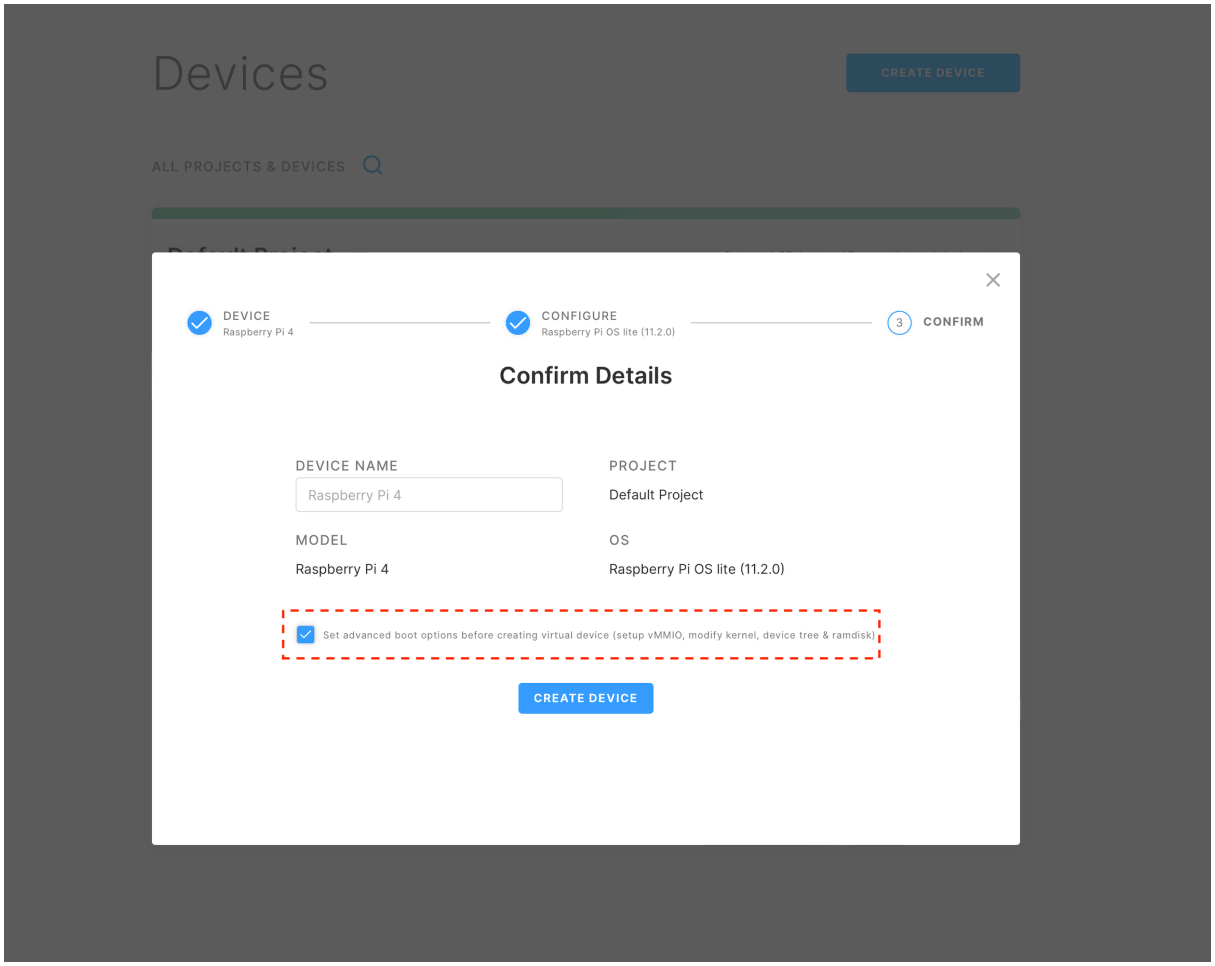
Figure 2-27: Configure your device



4. Select the `Set advanced boot options ...` checkbox and click CREATE DEVICE.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

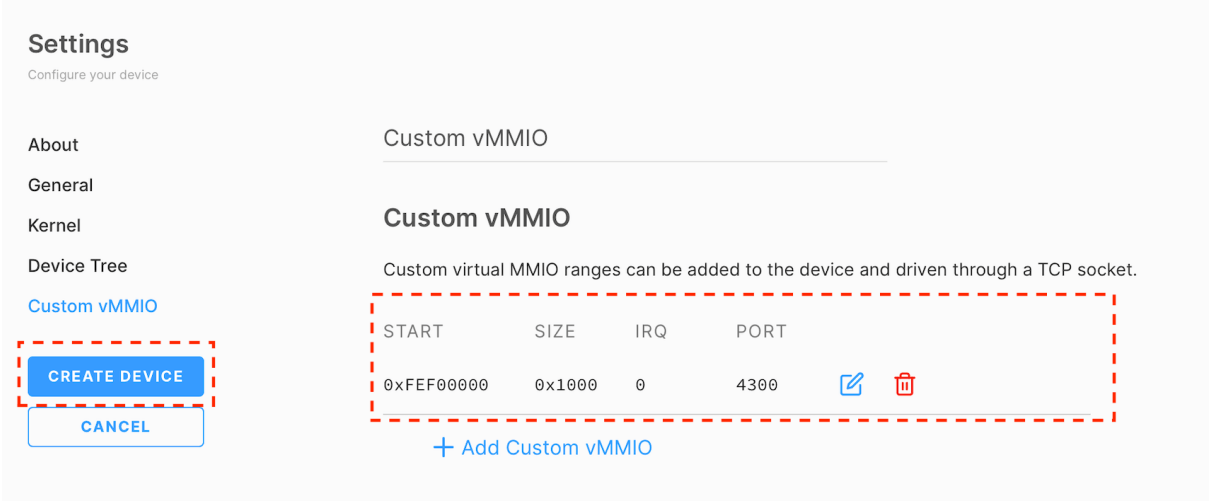
Figure 2-28: Confirm details



5. Select Custom vMMIO and enter `0xfef00000` for `start`, `0x1000` for `size`, 0 for `IRQ` (not used in this example), and 4300 for `port` (arbitrary). Then click CREATE DEVICE one final time.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-29: Custom vMMIO



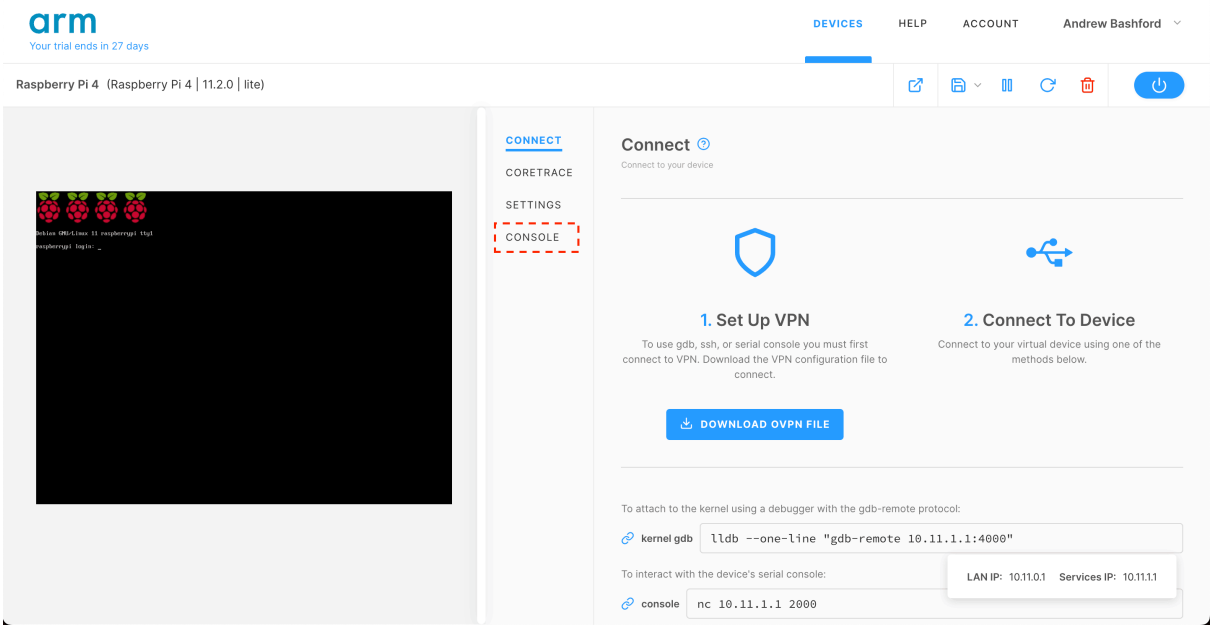
Note

`0xfef00000` was selected because the Raspberry Pi hardware does not have anything mapped at this address and as such it is safe to use for our purposes.

---

6. The device will take a few moments to build. When complete click on CONSOLE.

Figure 2-30: Connect



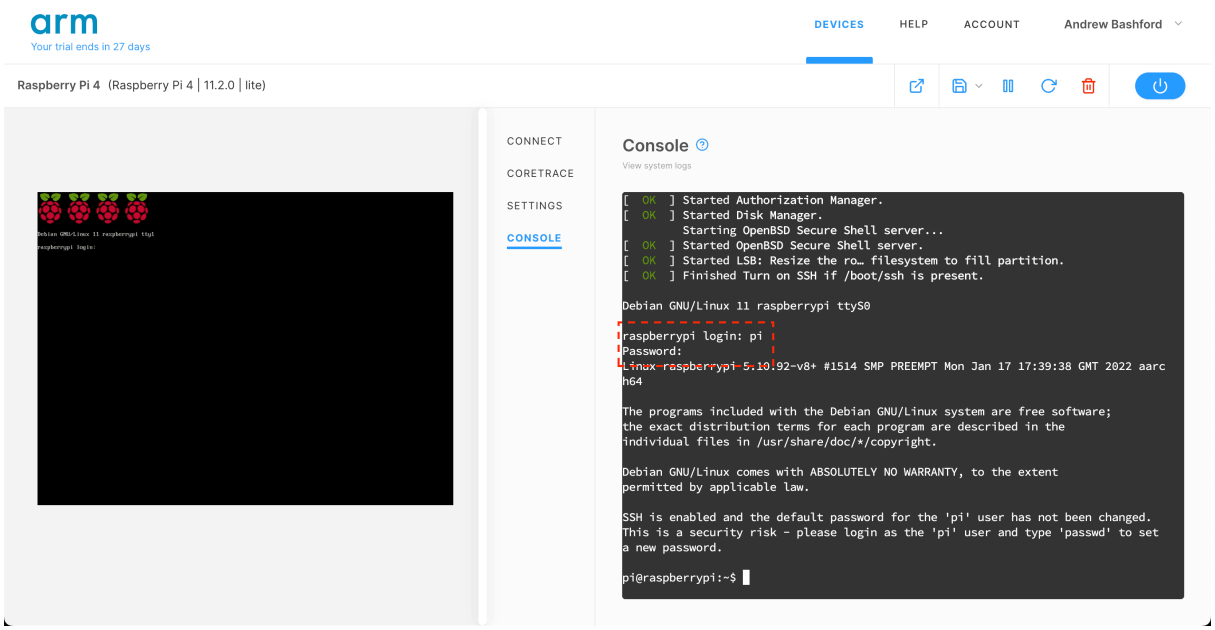
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2.4.1.2 Writing to MMIO Without a vMMIO Handler Registered

To write to MMIO without a vMMIO handler registered:

1. Log in using the default credentials of `pi/raspberry`.

   Figure 2-31: Log in



2. Download project sources to your Raspberry Pi, extract, and build it.

```
pi@raspberrypi:~ $ wget https://avh.arm.com/downloads/coreio.tar.gz
...
2022-04-12 14:28:55 (18.0 MB/s) - 'coreio.tar.gz' saved [23940/23940]
pi@raspberrypi:~ $ tar -xf coreio.tar.gz
pi@raspberrypi:~ $ cd coreio/examples/rpi-test/
pi@raspberrypi:~/coreio/examples/rpi-test $ make
gcc -g -Wall -I../../ -shared -fPIC -o libcoreio.so ../../corehdl-base.c ../../
coreio.c
gcc -g -Wall -I../../ -c -o corehdl-base.o ../../corehdl-base.c
gcc -g -Wall -I../../ -c -o coreio.o ../../coreio.c
ar cr libcoreio.a corehdl-base.o coreio.o
gcc -g -Wall -I../../ -ovmmio-test-inside vmmio-test-inside.c
gcc -g -Wall -I../../ -ovmmio-test-outside vmmio-test-outside.c libcoreio.a
```

3. Run the `vmmio-test-inside` build product

   This example source will open `/dev/mem` at offset `0xfef00000` (recall this is the address of the registered vMMIO) and attempt to read and write to it.

   However, without any registered vMMIO handlers running, AVH's default behavior, as mentioned earlier, is to return `0xaa` in the registers receiving the read contents.

```
$ sudo ./vmmio-test-inside
0: 0xaa
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
1: 0xaa
2: 0xaa
3: 0xaa
...
0: 0xaaaa
1: 0xaaaa
2: 0xaaaa
3: 0xaaaa
...
0: 0xaaaaaaaa
1: 0xaaaaaaaa
2: 0xaaaaaaaa
3: 0xaaaaaaaa
...
0: 0xaaaaaaaaaaaaaaaa
1: 0xaaaaaaaaaaaaaaaa
2: 0xaaaaaaaaaaaaaaaa
3: 0xaaaaaaaaaaaaaaaa
```

In order to implement something more meaningful we need to handle those read and write requests by registering a handler

2.4.1.3 Writing to MMIO With a vMMIO Handler Registered

To write to MMIO with a vMMIO handler registered:

1. Download project sources to your local machine, extract, and build it.

```
$ wget https://avh.arm.com/downloads/coreio.tar.gz
...
2022-04-12 14:28:55 (18.0 MB/s) - 'coreio.tar.gz' saved [23940/23940]
$ tar -xf coreio.tar.gz
$ cd coreio/examples/rpi-test/
$ make
gcc -g -Wall -I../../ -shared -fPIC -o libcoreio.so ../../corehdl-base.c ../../
coreio.c
gcc -g -Wall -I../../ -c -o corehdl-base.o ../../corehdl-base.c
gcc -g -Wall -I../../ -c -o coreio.o ../../coreio.c
ar cr libcoreio.a corehdl-base.o coreio.o
gcc -g -Wall -I../../ -ovmmio-test-inside vmmio-test-inside.c
gcc -g -Wall -I../../ -ovmmio-test-outside vmmio-test-outside.c libcoreio.a
```

2. Run the `vmmio-test-outside` build product, targeting the Services IP of the VM and vMMIO port (4300) as arguments

---

Note

This step registers the vMMIO handler, overriding the default behavior just observed.

```
$ ./vmmio-test-outside 10.11.1.1:4300
tick
tick
tick
...
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. Re-run `vmmio-test-inside` within the RaspberryPi VM. Observe that the `vmmio-test-outside` binary will now emit state information for each read and write event happening inside the VM by `vmmio-test-inside`.

```
$ ./vmmio-test-outside 10.11.1.1:4300
tick
tick
tick
write fef00000 1 0000000000000000 0
write fef00001 1 0000000000000001 0
write fef00002 1 0000000000000002 0
write fef00003 1 0000000000000003 0
write fef00004 1 0000000000000004 0
write fef00005 1 0000000000000005 0
write fef00006 1 0000000000000006 0
tick
write fef00007 1 0000000000000007 0
write fef00008 1 0000000000000008 0
...
```

Meanwhile `vmmio-test-inside` will emit the contents it reads from the MMIO address after attempting to modify its contents with a write.

```
$ sudo ./vmmio-test-inside
0: 0x00
1: 0x01
2: 0x02
3: 0x03
...
0: 0x0100
1: 0x0201
2: 0x0302
3: 0x0403
...
0: 0x03020100
1: 0x04030201
2: 0x05040302
3: 0x06050403
...
0: 0x0706050403020100
1: 0x0807060504030201
2: 0x0908070605040302
3: 0x0a09080706050403
```

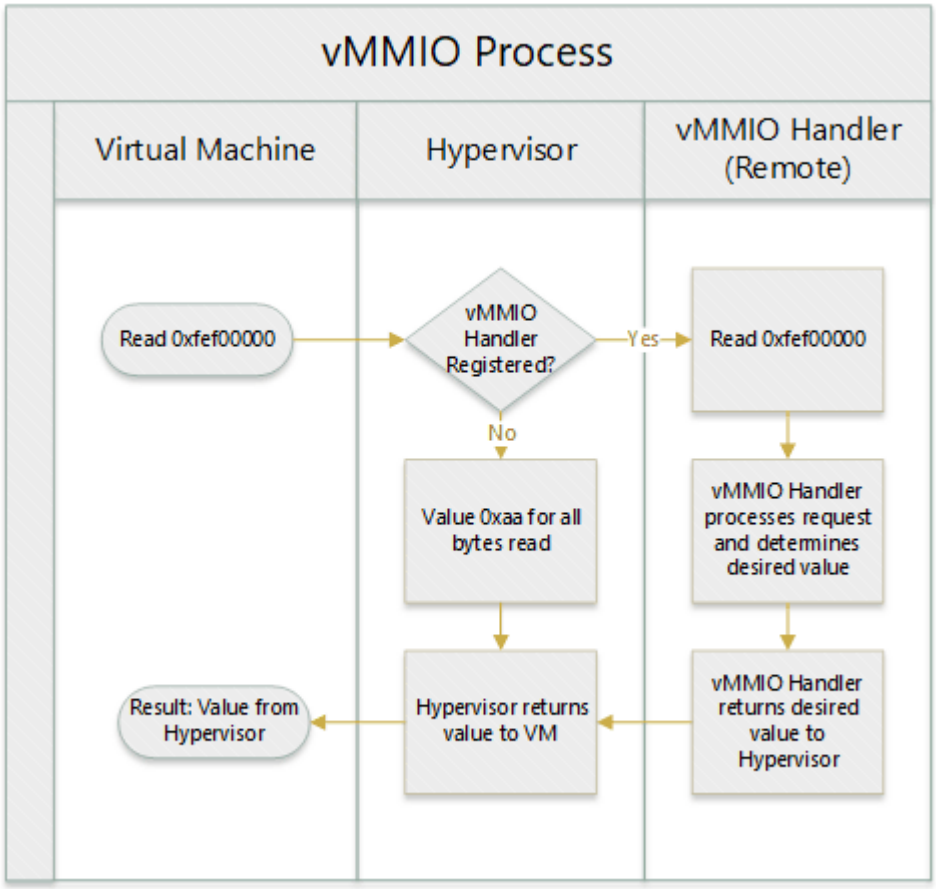
2.4.1.4 Flow Diagram

The flow followed in this example is illustrated below. The Writing to MMIO With a vMMIO Handler Registered example follows the “vMMIO Handler Registered” No path while the Writing to

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

MMIO With a vMMIO Handler Registered example has a remote handler registered and follows the Yes path.

Figure 2-32: Flow diagram



2.4.1.5 Summary

In summary, `vmmio-test-outside` is registering as an vMMIO read and write handler for a specified address range within the VM. When the physical address range specified is read from or written to, this handler receives a callback allowing it to handle the request. In this example, the `vmmio-test-inside` binary opened `/dev/mem` and simulates the OS or some other entity reading or writing to physical memory in order to exercise the callbacks in the registered vMMIO handler.

2.5 API Examples

API examples:

- Upload Custom Firmware with the Python API
- Python and Javascript API Examples

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- GitHub Actions Example - Building Firmware, Uploading, and Testing

2.5.1 Upload Custom Firmware with the Python API

Learn how to upload a custom firmware package using the AVH Asynchronous API.

Table of Contents

- Overview of the Python Script
- Python Script
- How to Run the Script

The AVH platform offers many different ways to interact with your virtual devices.

In this article, we demonstrate how to upload custom firmware and connect the the Raspberry Pi 4 boad's console using a WebSocket using the AVH Asynchronous Python API.

2.5.1.1 Overview of the Python Script

The script performs the following tasks:

1. Log in using your API token.
2. Find your project and save the VPN details.
3. Upload your custom firmware package.
4. Create a device using your uploaded image.
5. Wait for the device to boot.
6. Find the SSH connection details.
7. Stream the board's console in UTF-8 via a WebSocket.

Example of the script's output.

```
user1@comp1-2 ~ % python3 /Users/user1/api_example.py https://app.avh.arm.com/api
 7bf64295bfe46191f7bf.d735bb3128d2903c5b19f4d443f424ecb12cdce22ddbcf43aef3adf473e635702ef3f6923af
Starting the script ...
Logging in ...
Logged in.
Finding the project ...
Found project 18370783-d77c-4382-bf20-61c024d4eabf.
Finding the project VPN configuration ...
Saved the VPN configuration to /Users/user1/Desktop/AVH_VPN.ovpn.
Uploading the custom firmware package ...
Uploaded image cafcb47f-8e48-4ca8-8f00-df2b07793671.
Creating an instance named Ubuntu Desktop via Python API ...
Instance 71849293-74ff-4dbf-85f1-1aa5cb8ee41c is creating.
Waiting for the instance to boot up ...
After 30 seconds, the instance is creating.
After 60 seconds, the instance is creating.
After 90 seconds, the instance is creating.
After 120 seconds,j the instance is on.
Getting the connection details ...
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
To SSH, connect to the VPN then use `ssh ubuntu@10.11.0.13`.
Connecting to the WebSocket ...
{Begin stream of your Raspberry Pi 4 board's console output}
...
g/HQLlFVcJqnetdzkWXnPXF1GX7XSZqIazZ2/2E= root@ubuntu
-----END SSH HOST KEY KEYS-----
[   22.338550] cloud-init[1333]: Cloud-init v. 22.2-0ubuntu1~22.04.3 finished at
 Mon, 27 Jun 2022 18:29:02 +0000. Datasource DataSourceNoCloud [seed=/dev/mmcblk0p1]
[dsmode=net].  Up 22.33 seconds
Finished running main() due to AsyncIO timeout.
Finished the script.
```

Note

This script will fail for packages larger than 2.00 gigabytes. For larger firmware packages, we recommend using the Create a New Image call from our REST API, which you can run with a cURL command from within Python.

2.5.1.2 Python Script

The Python script is as follows:

```
import asyncio
import time
import sys
import websockets
#from pprint import pprint # Optional for testing and development purposes.
import avh_api_async as AvhAPI
from avh_api_async.rest import ApiException as AvhAPIException

if len(sys.argv) < 3:
  print('Usage: %s <ApiEndpoint> <ApiToken>', sys.argv[0])
  exit(-1)

apiEndpoint = sys.argv[1]
apiToken = sys.argv[2]

instanceFlavor = 'rpi4b' # use flavor=rpi4b for the Raspberry Pi 4b board
ubuntuImageType = 'fwpackage' # use type=fwpackage for the Ubuntu Server firmware
 package
ubuntuImageEncoding = 'plain' # use encoding=plain
ubuntuImageVersion = '22.04.1' # the OS version that matches Info.plist
ubuntuImagePath = '/Users/user1/Desktop/' # directory that contains your firmware
 package
ubuntuImageFilename = 'rpi4b-ubuntu-server.zip' # name of your firmware package
ubuntuInstanceName = 'Ubuntu Server via Python API' # arbitraty name the instance

async def main():
  configuration = AvhAPI.Configuration(host = apiEndpoint)

  # Enter a context with an instance of the API client
  async with AvhAPI.ApiClient(configuration=configuration) as api_client:
    # Create an instance of the API class
    api_instance = AvhAPI.ArmApi(api_client)

    # Log in
    print('Logging in ...')
    try:
        token_response = await api_instance.v1_auth_login({
          "apiToken": apiToken
        })
        configuration.access_token = token_response.token
    except AvhAPIException as e:
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
        print('Exception when calling v1_auth_login: %s\n' % e)
        exit(1)
    print('Logged in.')

    # Get the project
    print('Finding the project ...')
    api_response = await api_instance.v1_get_projects()
    projectId = api_response[0].id
    print('Found project ' + projectId + '.')

    # Get the VPN config
    print('Finding the project VPN configuration ...')
    try:
      api_response = await api_instance.v1_get_project_vpn_config(projectId, 'ovpn')
      #pprint(api_response)
      projectVPNConfigFile = open(ubuntuImagePath+'AVH_VPN.ovpn', "w")
      projectVPNConfigFile.write(api_response)
      projectVPNConfigFile.close()
    except AvhAPIException as e:
      print('Exception when calling v1_get_project_vpn_config: %s\n' % e)
      exit(1)
    print('Saved the VPN configuration to ' + ubuntuImagePath + 'AVH_VPN.ovpn.')

    # Upload the image
    print('Uploading the custom firmware package ...')
    try:
      api_response = await api_instance.v1_create_image(_request_timeout=3600,
type=ubuntuImageType, encoding=ubuntuImageEncoding, name=ubuntuImageFilename,
project=projectId, file=ubuntuImagePath+ubuntuImageFilename)
      # The timeout for v1_create_image() is set to 1 hour.
      #pprint(api_response)
      ubuntuImageId = api_response.id
    except AvhAPIException as e:
      print('Exception when calling v1_create_image: %s\n' % e)
      exit(1)
    print('Uploaded image ' + ubuntuImageId + '.')

    # Create the instance
    print('Creating an instance named ' + ubuntuInstanceName + ' ...')
    try:
      api_response = await api_instance.v1_create_instance({
        'project': projectId,
        'name': ubuntuInstanceName,
        'flavor': instanceFlavor,
        'os': ubuntuImageVersion,
        'fwpackage': ubuntuImageId
      })
      #pprint(api_response)
      ubuntuInstanceId = api_response.id
      ubuntuInstanceState = api_response.state
    except AvhAPIException as e:
      print('Exception when calling v1_create_instance: %s\n' % e)
      exit(1)
    print('Instance ' + ubuntuInstanceId + ' is ' + ubuntuInstanceState + '.')

    # Wait for the instance to turn on
    count = 0
    secondsToPause = 30
    print('Waiting for the instance to boot up ...')
    while ubuntuInstanceState == 'creating':
      count+=1
      time.sleep(secondsToPause)
      api_response = await api_instance.v1_get_instance_state(ubuntuInstanceId)
      ubuntuInstanceState = api_response
      print('After ' + str(secondsToPause*count) + ' seconds, the instance is ' +
ubuntuInstanceState + '.')

    # Get the SSH details
    print('Getting the connection details ...')
    try:
      api_response = await api_instance.v1_get_instance(ubuntuInstanceId)
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
      #pprint(api_response)
      wifiIP = api_response.wifi_ip
    except AvhAPIException as e:
      print('Exception when calling v1_get_instance: %s\n' % e)
      exit(1)
    print('To SSH, connect to the VPN then use `ssh ubuntu@' + wifiIP + '`.')

    # Connect to the console WebSocket
    print('Requesting the instance console WebSocket URL ...')
    try:
      api_response = await api_instance.v1_get_instance_console(ubuntuInstanceId)
      ubuntuConsoleWebSocketURL = api_response.url
    except AvhAPIException as e:
      print('Exception when calling v1_get_instance_console: %s\n' % e)
      exit(1)
    print('Connecting to the WebSocket ...')
    async with websockets.connect(ubuntuConsoleWebSocketURL) as ws:
      await handler(ws)
      await asyncio.Future()  # keep WebSocket connection open

async def handler(websocket):
  while True:
    message = await websocket.recv() # encoded in Unicode
    print(message.decode(encoding='UTF-8',errors='replace')) # decoded to UTF-8

if __name__ == '__main__':
  print('Starting the script ...')
  try:
    asyncio.run(asyncio.wait_for(main(), 7200)) # The timeout is 2 hours.
  except Exception as e:
    print('Finished running main() due to AsyncIO timeout.')
  print('Finished the script.')
  exit(0)
```

2.5.1.3 How to Run the Script

In this example, we will be uploading the `rpi4b-ubuntu-server.zip` custom firmware package from our Package Ubuntu Server Firmware for AVH article.

Follow the steps below to run the Python script. For more details, refer to the Python API Readme and the Reference for avh-api-async Methods documents.

1. Ensure you are running at least Python 3.4 and install the required packages including `avh-api-async`. Refer to the Python API Readme.md for more details.

   ```
   pip3 install avh-api avh-api-async websockets python-dateutil aiohttp six urllib3
   ```

2. Save the Python script to a local file on your computer ending in `.py`. For our example, we save the script to the local user's desktop.
3. Update the hardcoded `ubuntuImagePath` and `ubuntuImageFilename` in the script to reference your local firmware package. The script will save a VPN configuration file to this path.

   ```
   ubuntuImagePath = '/Users/user1/Desktop/'
   ubuntuImageFilename = 'rpi4b-ubuntu-server.zip'
   ```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4. Run the script by referencing the AVH endpoint and your AVH API token, see Generating an AVH API Token for more information.

```
python3 /Users/user1/Destkop/api_example.py https://app.avh.arm.com/api
 7bf64295bfe46191f7bf.d735bb3128d2903c5b19f4d443f424ecb12cdce22ddbcf43aef3adf473e635702ef3f69
```

2.5.2 Python and Javascript API Examples

Visiting the AVH GitHub where you will find a repo with examples for both Python and Javascript.

2.5.2.1 Python Examples

The Python examples are as follows:

- `instance_create_snapshot_delete.py`
  - This example demonstrates creating a STM32U5 device, taking a snapshot and then deleting it.
- `stm32_bsp_test.py`
  - This example demonstrates using GPIO input/output of a virtual machine
- `create_github_runner.py`
  - This example demonstrates using GPIO input/output of a virtual machine

2.5.2.2 Javascript Examples

The Javascript examples are as follows:

- `arm_api`
  - ArmApi - JavaScript client for arm_api REST API to manage your virtual devices.
- `sensordemo-docker`
  - This example will execute a build on your runner and create a new device on AVH to test the output of the build on.
- `sensordemo-generate-api`

2.5.3 GitHub Actions Example - Building Firmware, Uploading, and Testing

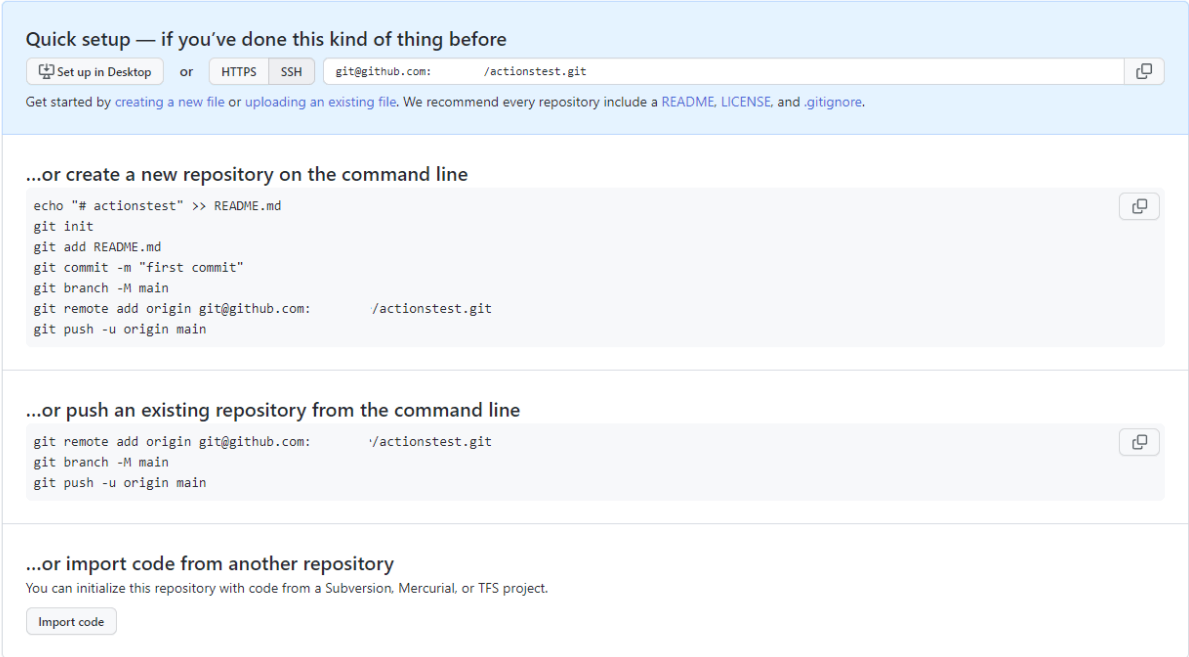
The following steps help you to set up GitHub Actions.

1. To get a VM ready for the GitHub runner, follow the steps in the Running Docker on AVH Pi article.
2. Ensure you have an AVH API Token by following the steps in the Generating an AVH API Token article.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

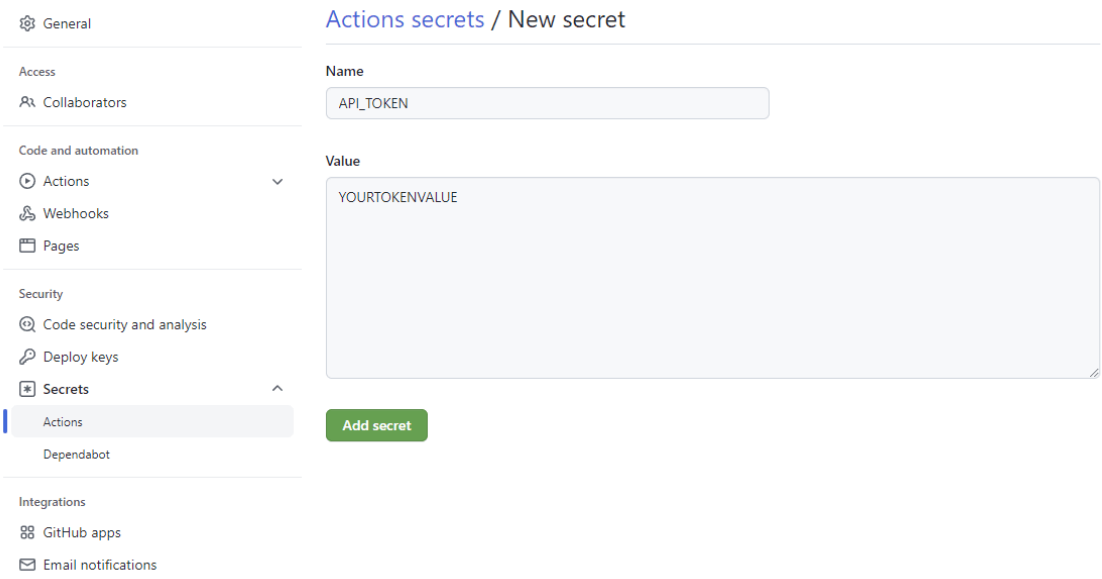
3. Create an empty GitHub repository for your runner to accept jobs from by following the steps in this GitHub Article.

Figure 2-33: Create an empty GitHub repository



4. Add a "API_TOKEN" secret to your repository with a value of your API Token by following the steps in this GitHub Article.

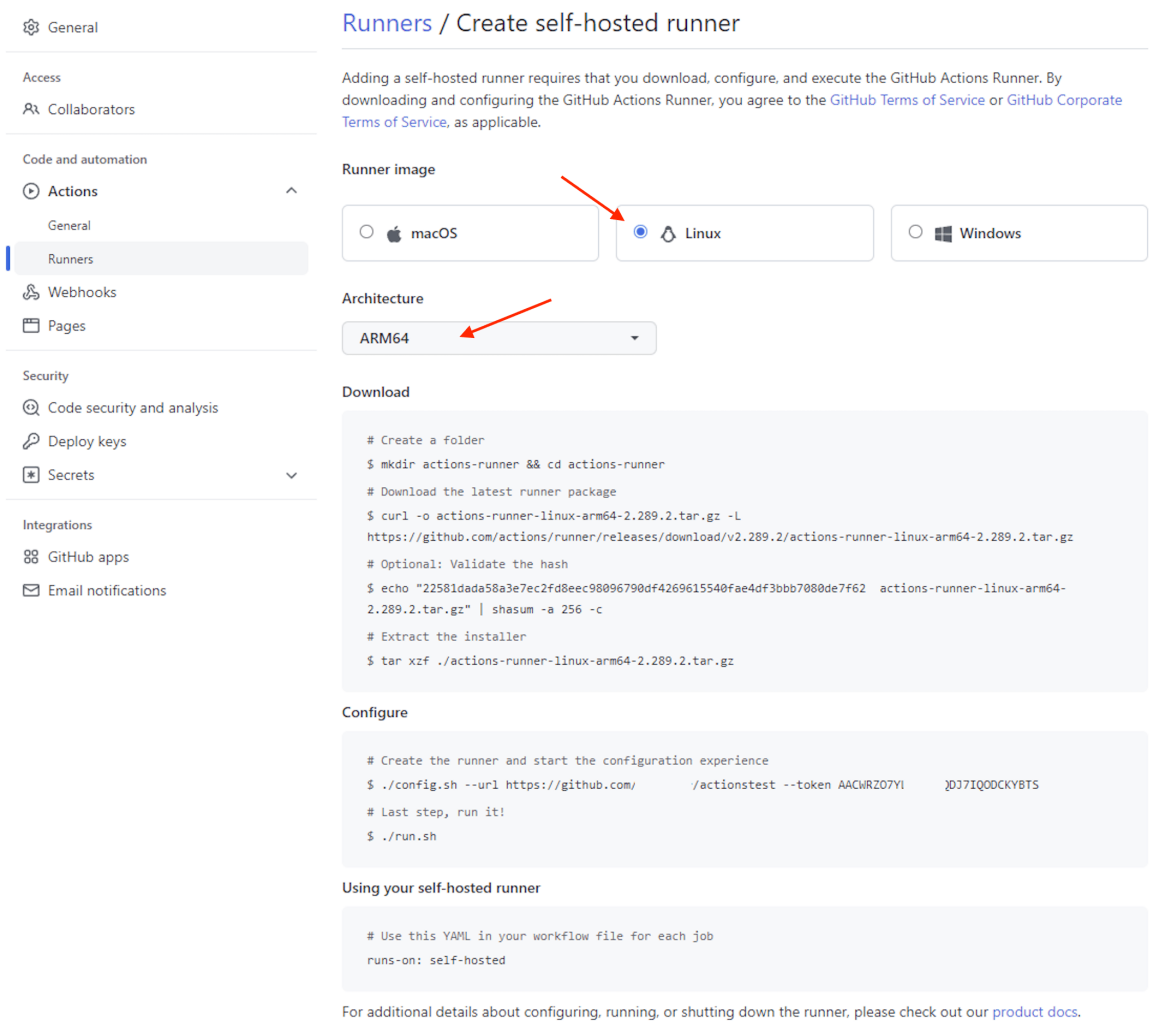
Figure 2-34: API_TOKEN



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5. To configure and start the runner, follow the steps in this Adding Self Hosted Runners article. Select “Linux” as the OS and “ARM64” as the architecture.

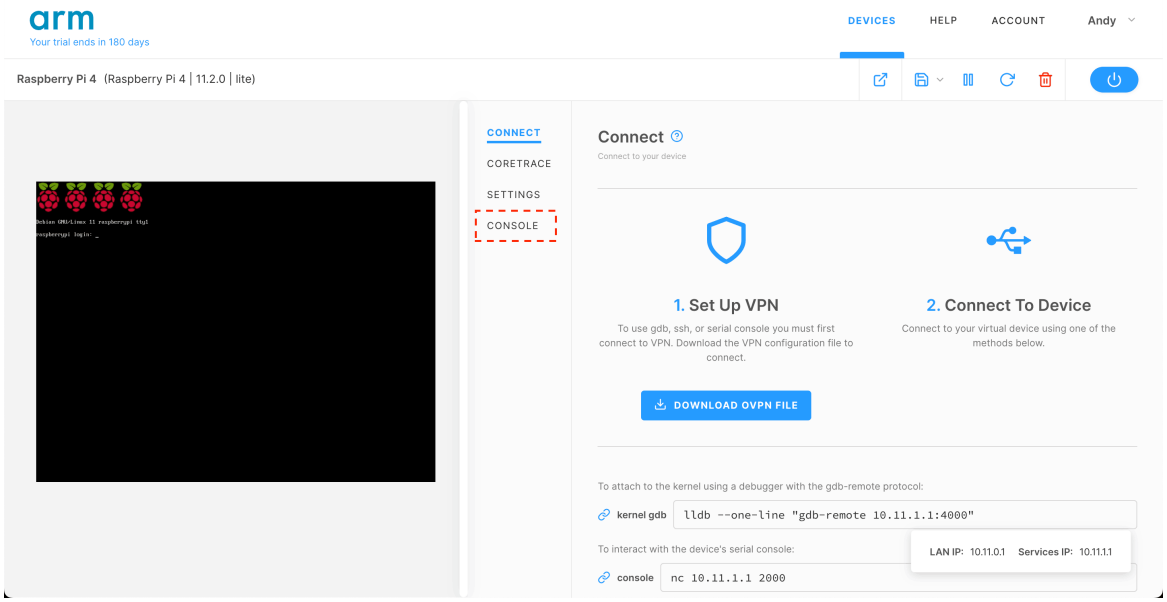
Figure 2-35: Adding self hosted runners



a. In AVH, select the Raspberry Pi VM created previously. Then select “CONSOLE” to access a shell to run the commands referenced above.

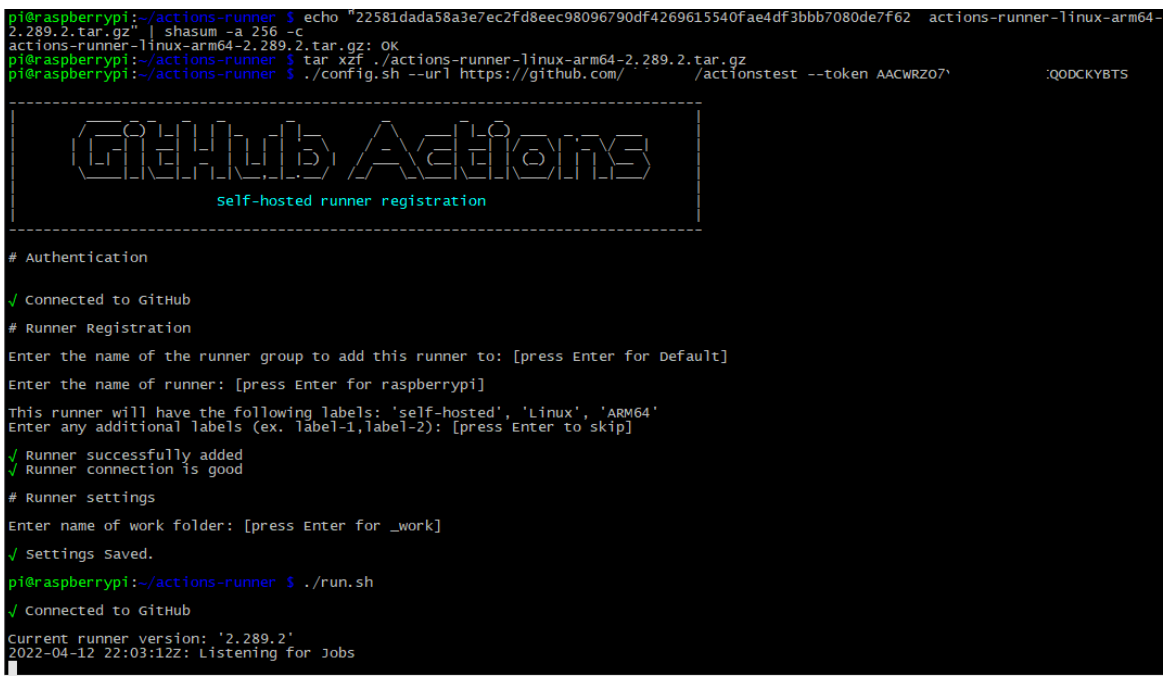
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-36: Select Console



b. Upon completion of runner setup you should see something similar to:

Figure 2-37: Console output



c. Do not cancel the running process or your runner will be unavailable until it is run again. The runner can also be installed as a service, but that is not covered in this article.

6. Clone the avh-api repository on your usual local development system from GitHub with: `git clone https://github.com/ARM-software/avh-api.git`

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-38: Clone the avh-api repository

```
pi@raspberrypi:~$ git clone https://github.com/ARM-software/avh-api.git
Cloning into 'avh-api'...
remote: Enumerating objects: 1256, done.
remote: Counting objects: 100% (1256/1256), done.
remote: Compressing objects: 100% (638/638), done.
remote: Total 1256 (delta 600), reused 1227 (delta 573), pack-reused 0
Receiving objects: 100% (1256/1256), 25.91 MiB | 31.97 MiB/s, done.
Resolving deltas: 100% (600/600), done.
pi@raspberrypi:~$
```

Note

We used a raspberry pi for this in this example, this will likely be your laptop or desktop system.

7. Enter the `javascript/sensordemo-docker` directory

Figure 2-39: Enter the directory

```
pi@raspberrypi:~$ cd avh-api/javascript/sensordemo-docker/
pi@raspberrypi:~/avh-api/javascript/sensordemo-docker$
```

8. Run `make clean gitrepo GITREMOTE='<url to Github repo you created>'` in the console.

Note

This will overwrite any contents of the repo passed as GITREMOTE=.

Figure 2-40: Run GITREMOTE=

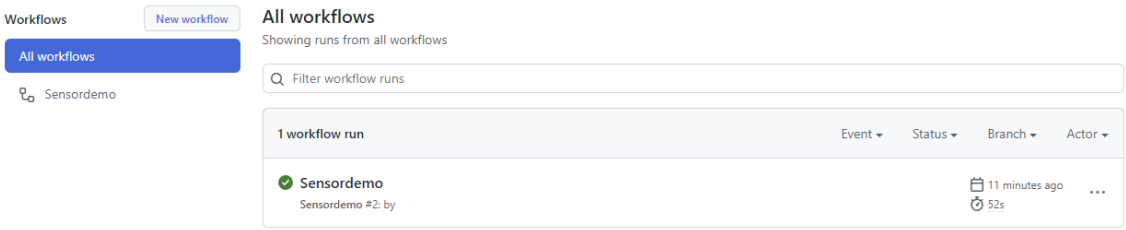
```
pi@raspberrypi:~/avh-api/javascript/sensordemo-docker $ make clean gitrepo GITREMOTE='https://github.com/        /actionstest.git'
rm -rf gitrepo
git archive HEAD --prefix="gitrepo/" | tar -xf -
cp -a ../api gitrepo/api
cd gitrepo; \
  git init; \
  mkdir -p .github/workflows; \
  mv github_workflow.yml .github/workflows/avh_build_and_test.yml; \
```

```
 create mode 100755 run_docker.sh
Example repository created in /home/pi/avh-api/javascript/sensordemo-docker/gitrepo
cd gitrepo; \
  git remote add origin https://github.com/        /actionstest.git; \
  git push -fu origin $(git rev-parse --abbrev-ref HEAD)
Username for 'https://github.com':
Password for 'https://        @github.com':
Enumerating objects: 750, done.
Counting objects: 100% (750/750), done.
Delta compression using up to 4 threads
Compressing objects: 100% (717/717), done.
Writing objects: 100% (750/750), 24.39 MiB | 8.82 MiB/s, done.
Total 750 (delta 215), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (215/215), done.
To https://github.com/        /actionstest.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
pi@raspberrypi:~/avh-api/javascript/sensordemo-docker $
```

9. You should see an action run on your repository executed from your AVH VM.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

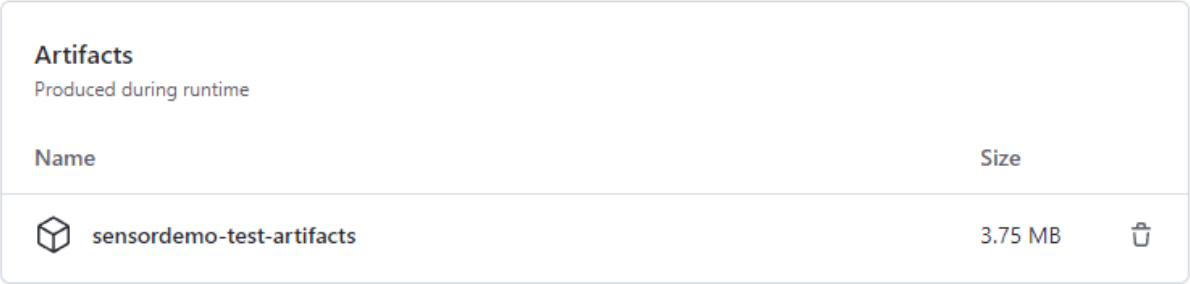
Figure 2-41: Executed action



This will compile a new firmware for STM32u5 on the pi VM, then upload it to a new STM32 VM in Arm Virtual Hardware, and run a test against it to ensure it behaved properly.

10. You will get a built firmware and the test results as the artifacts downloadable from GitHub when the action completes.

Figure 2-42: Artifacts



11. Any changes made to the repository, such as changing how the firmware behaves and pushes to GitHub, will trigger a new run.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. Features

The following articles are available:

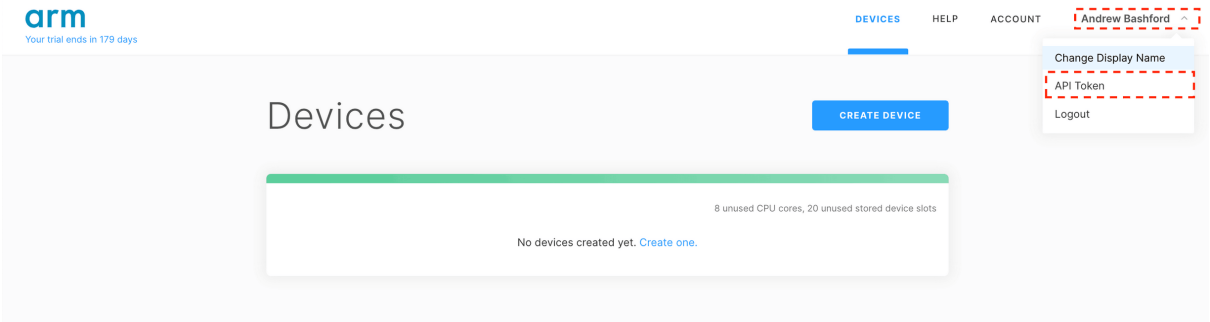
- Generating an AVH API Token
- Running Docker
- Service IP and LAN IP
- Detect Audio through the Microphone with the STM32U5 Board
- Power and Action Buttons
- Communicate Using Bluetooth Low Energy
- Connect to Wi-Fi on the Raspberry Pi 4
- Connect to your virtual devices
- Sensors
- Console
- CoreTrace

3.1 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”

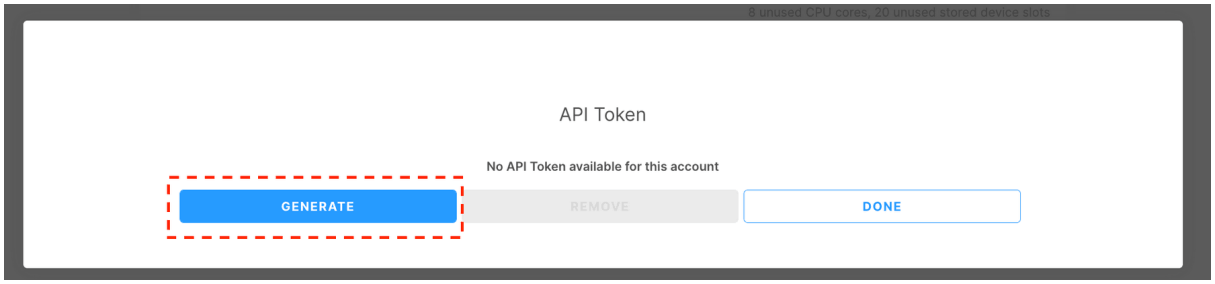
Figure 3-1: API token



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

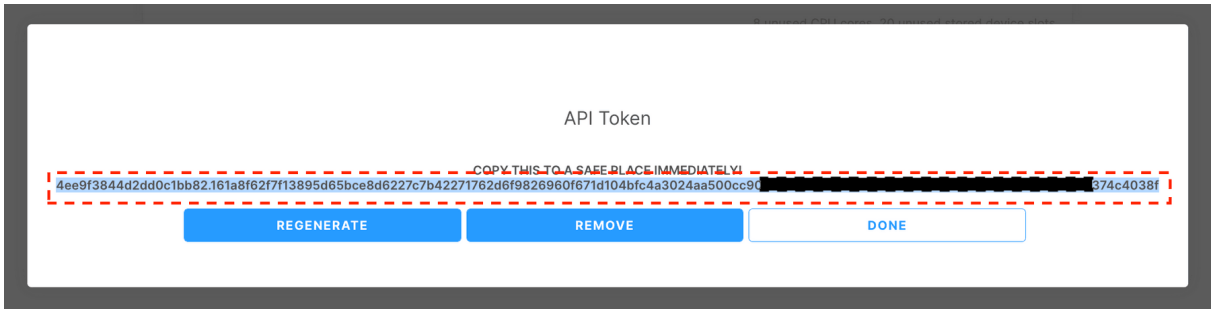
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-2: 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 3-3: Copy API token



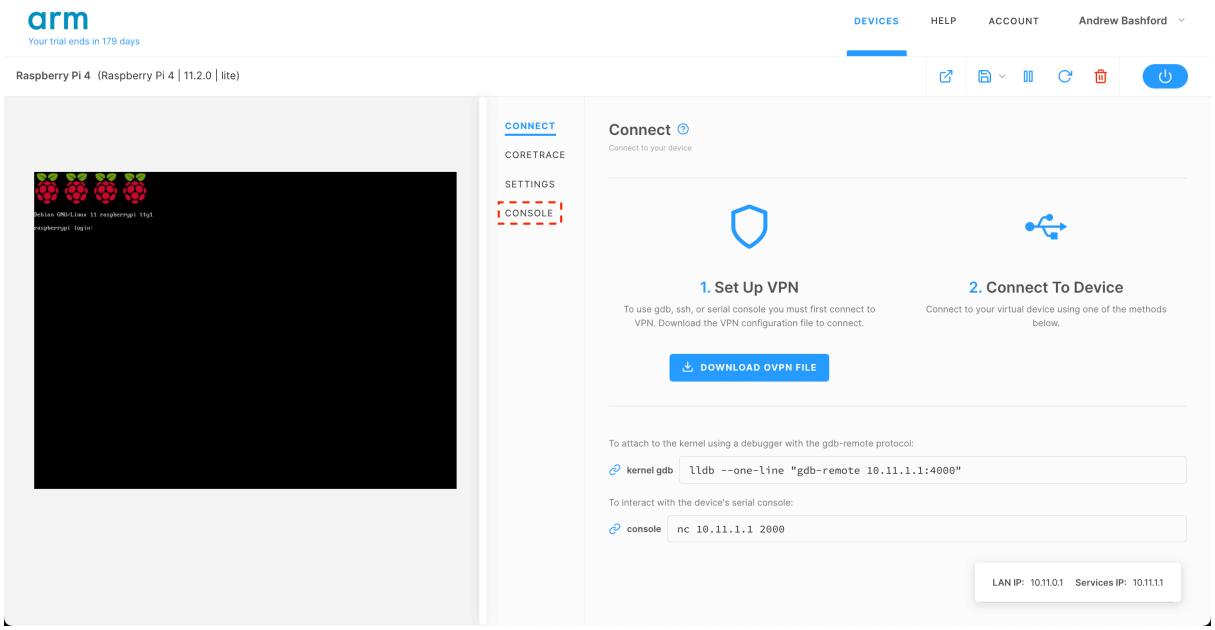
3.2 Running Docker

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 Raspberry Pi 4 to create a Raspberry Pi Device.
3. Select CONSOLE tab.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

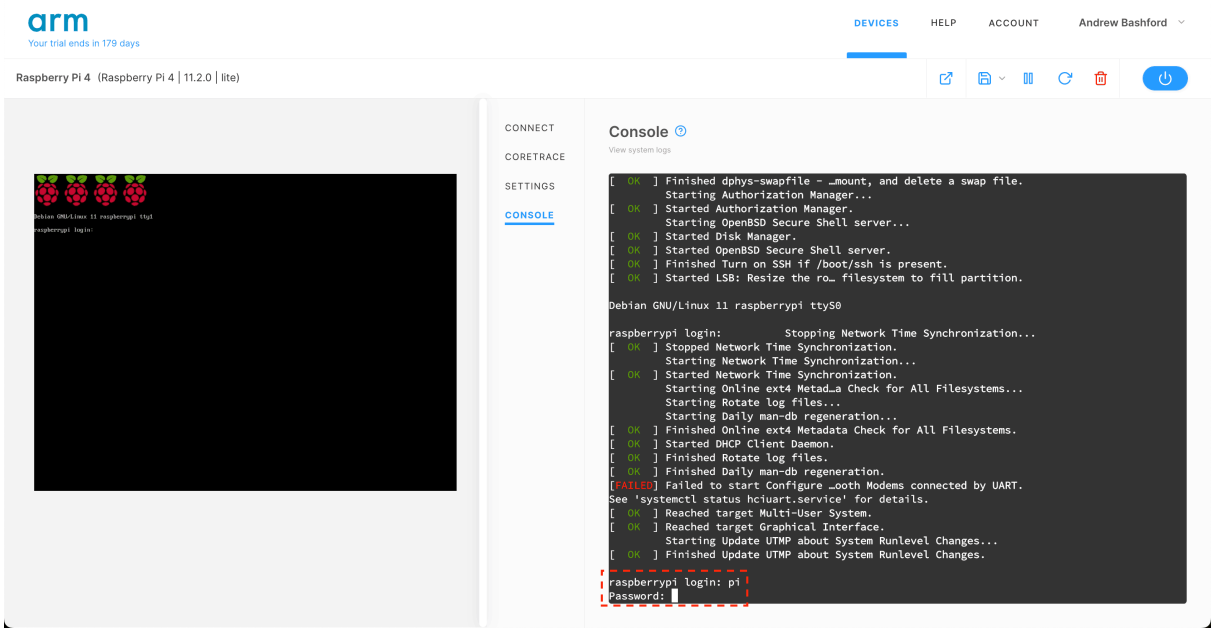
Figure 3-4: Select CONSOLE tab



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

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

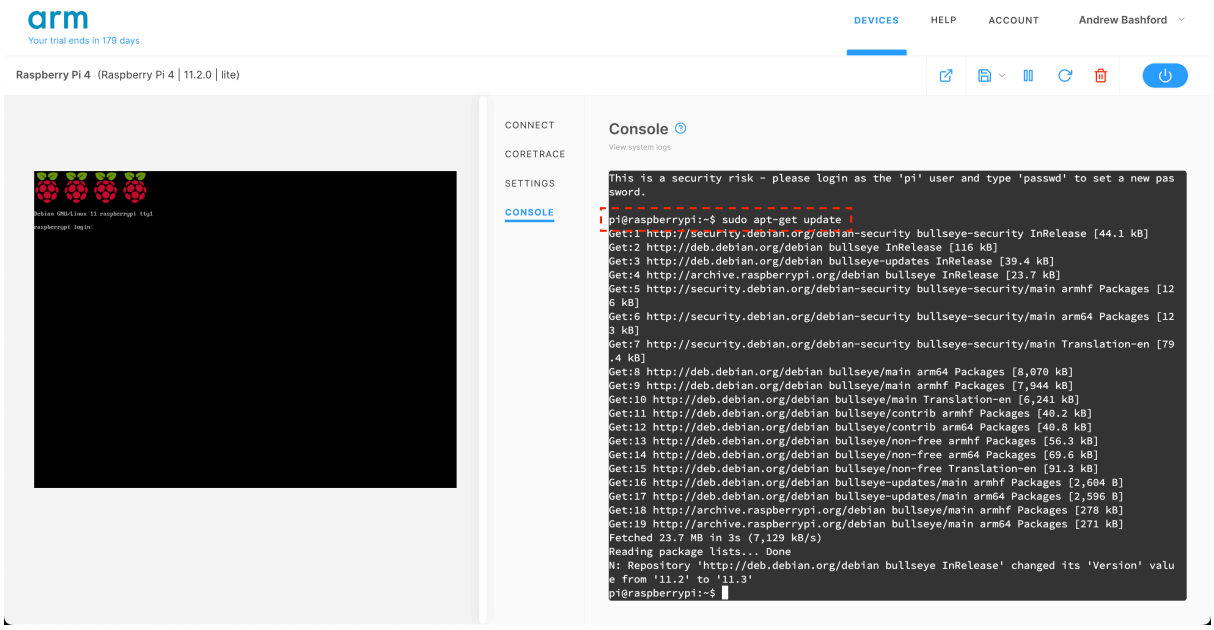
Figure 3-5: Log in



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

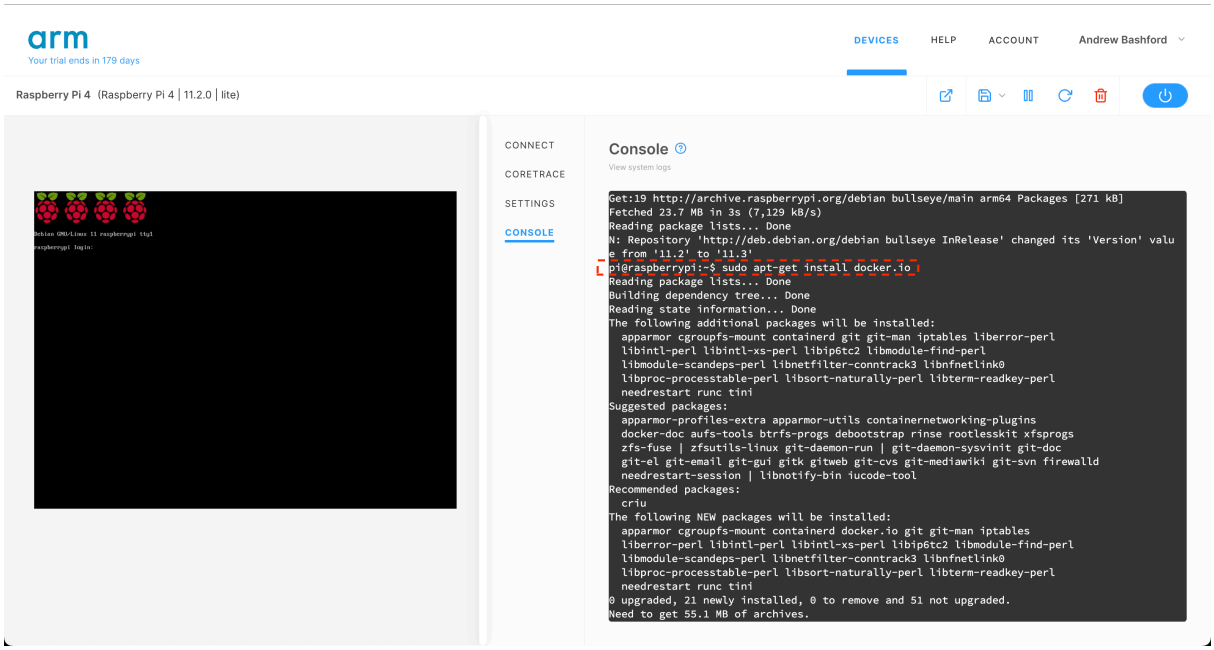
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-6: sudo apt-get update



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

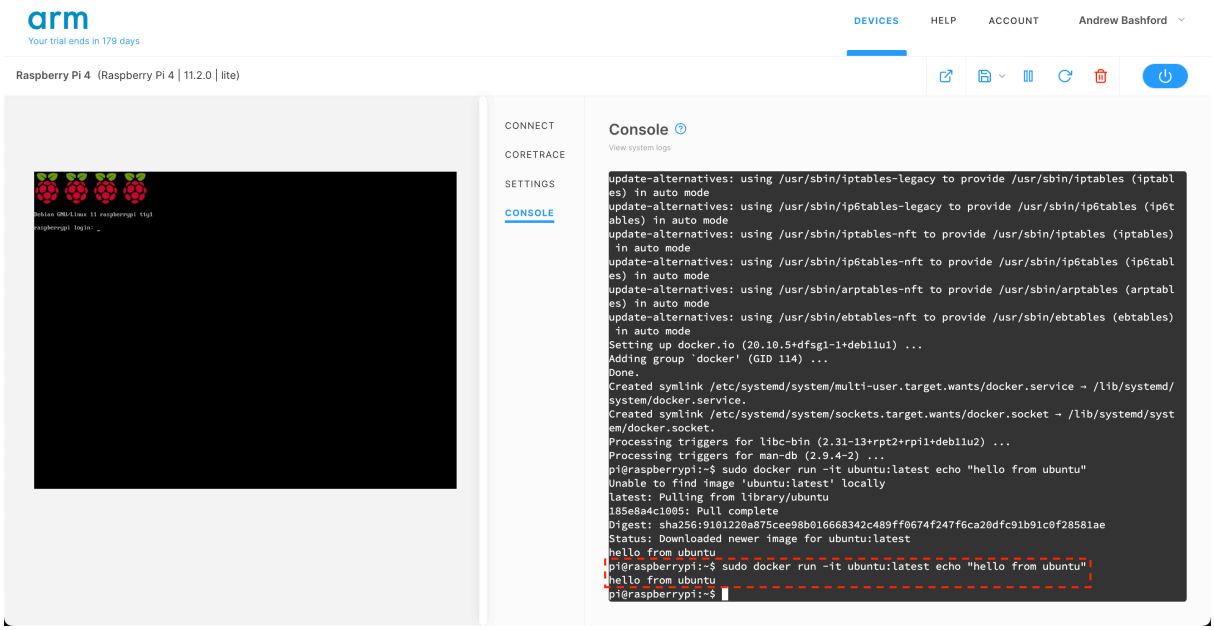
Figure 3-7: sudo apt-get install -y jq docker.io



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

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-8: Try Docker



3.3 Service IP and LAN IP

This article describes the Service IP and LAN IP.

3.3.1 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.

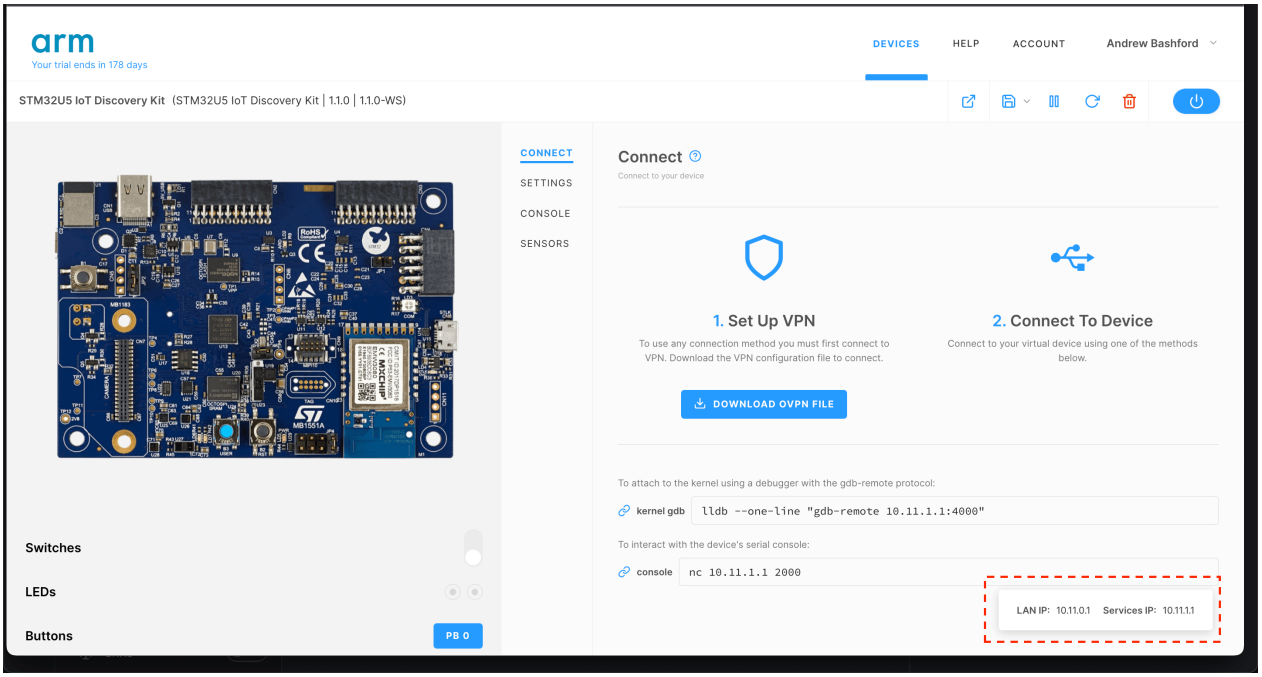
3.3.2 LAN IP

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.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-9: LAN IP and Services IP



3.4 Power and Action Buttons

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

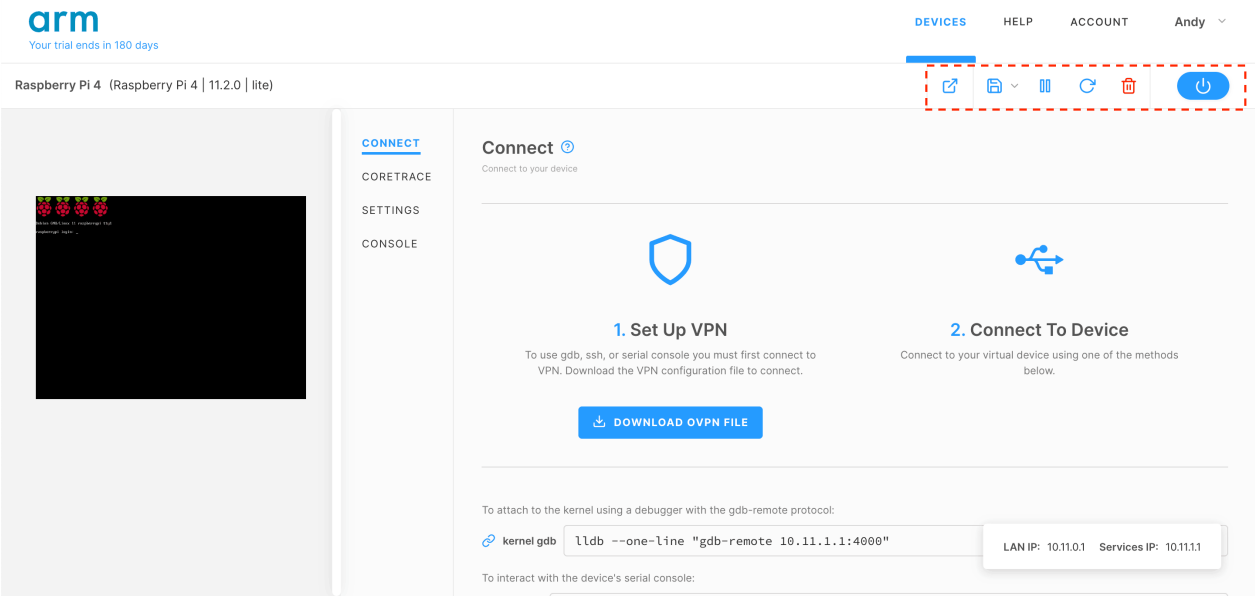
- Power and Action Buttons
  - Power On and Off
  - Pause
  - Restart
  - Delete

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.4.1 Power and Action Buttons

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

Figure 3-10: Power and action buttons



3.4.2 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 3-11: Power On / Off button

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.4.3 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 3-12: Pause button

3.4.4 Restart

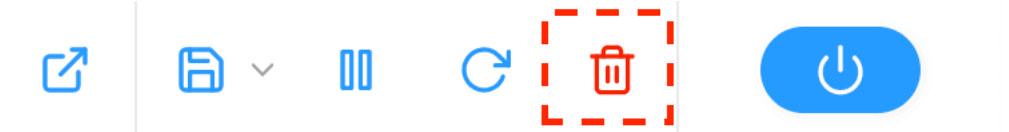
The restart button is akin to rebooting a physical device. It's the same as powering the device off and on again.

Figure 3-13: Restart button

3.4.5 Delete

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

Figure 3-14: Delete button



3.5 Connect to your virtual devices

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

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.5.1 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.

3.5.2 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.

3.5.3 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.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.5.4 Connect via other methods

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

3.6 Sensors

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

3.6.1 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)

3.6.2 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)

3.7 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.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.8 CoreTrace

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

3.8.1 What is CoreTrace?

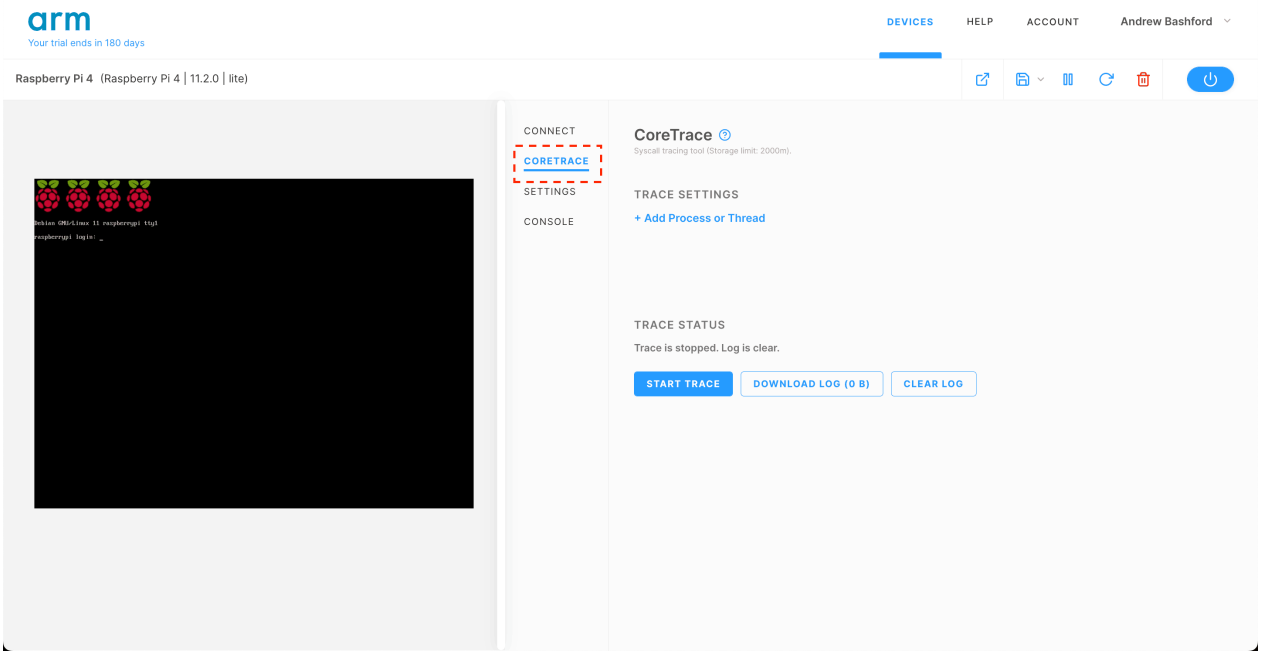
You can trace system calls using either strace, a standard command-line Linux tool, or our proprietary CoreTrace tool. strace is included in our virtual devices, and it is implemented with ptrace. Applications can employ anti-debugging techniques to detect and prevent ptrace-based tracing. However, these 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.

3.8.2 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 3-15: CoreTrace tab



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.8.3 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:

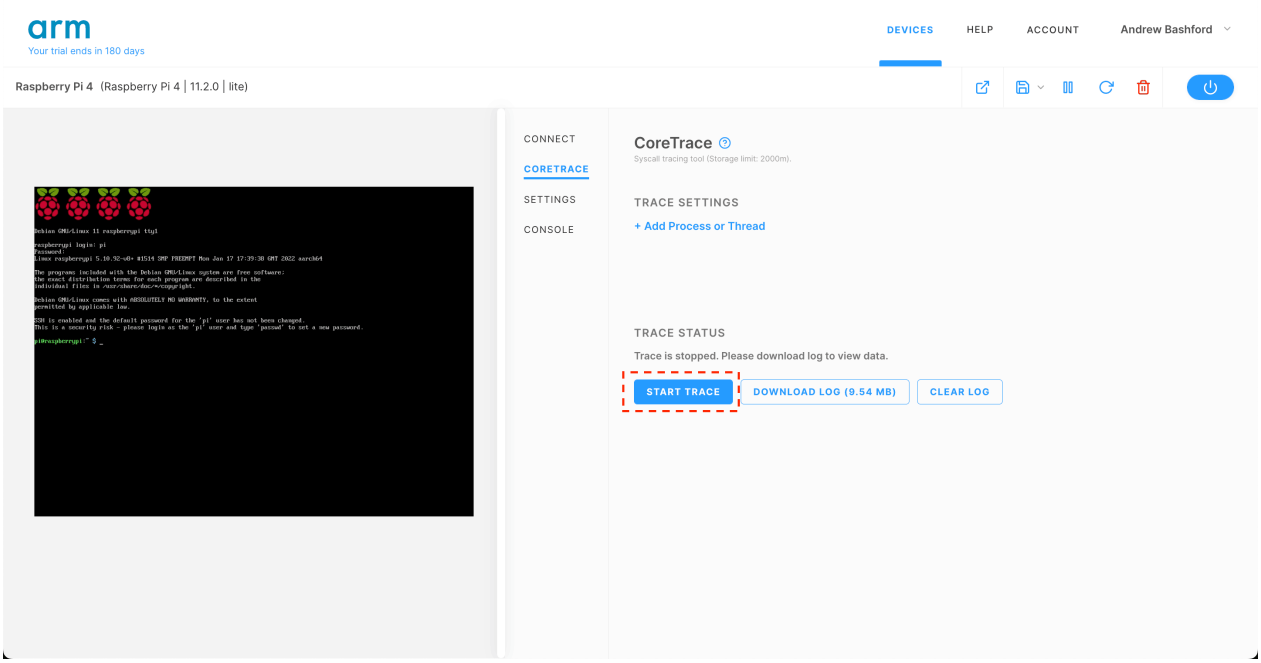
The Processes dialog displays all processes and threads in the system. To examine the threads inside a process, click the "THREADS" button in the process' row.

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 3-16: Start Trace



3.8.4 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
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
 0000006e5f6f9840 0000006e5f6f948c 0000006e5f6f7f90 0000006e5f6f7b54
 0000006e5f6f7434 00000070efc2088c 00000070efbc0e0c ]
```

or like this:

```
1<1> [00248.864656648] fffff806401e040-0/337: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.

3.9 Connect to Wi-Fi on the Raspberry Pi 4

The virtual Raspberry Pi 4 board connects by default over the ethernet interface.

To set up a connection over Wi-Fi, perform the following:

1. Create a Raspberry Pi 4 device with the "lite" firmware version and log in using the default credentials `(pi/raspberry)`.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-17: Raspberry Pi login

```
[  OK  ] Finished Update UTMP about System Runlevel Changes.
[  OK  ] Finished Daily man-db regeneration.

raspberrypi login: pi
Password:
Linux raspberrypi 5.10.92-v8+ #1514 SMP PREEMPT Mon Jan 17 17:39:38 GMT 202
2 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

SSH is enabled and the default password for the 'pi' user has not been chan
ged.
This is a security risk - please login as the 'pi' user and type 'passwd' t
o set a new password.


Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~$
```

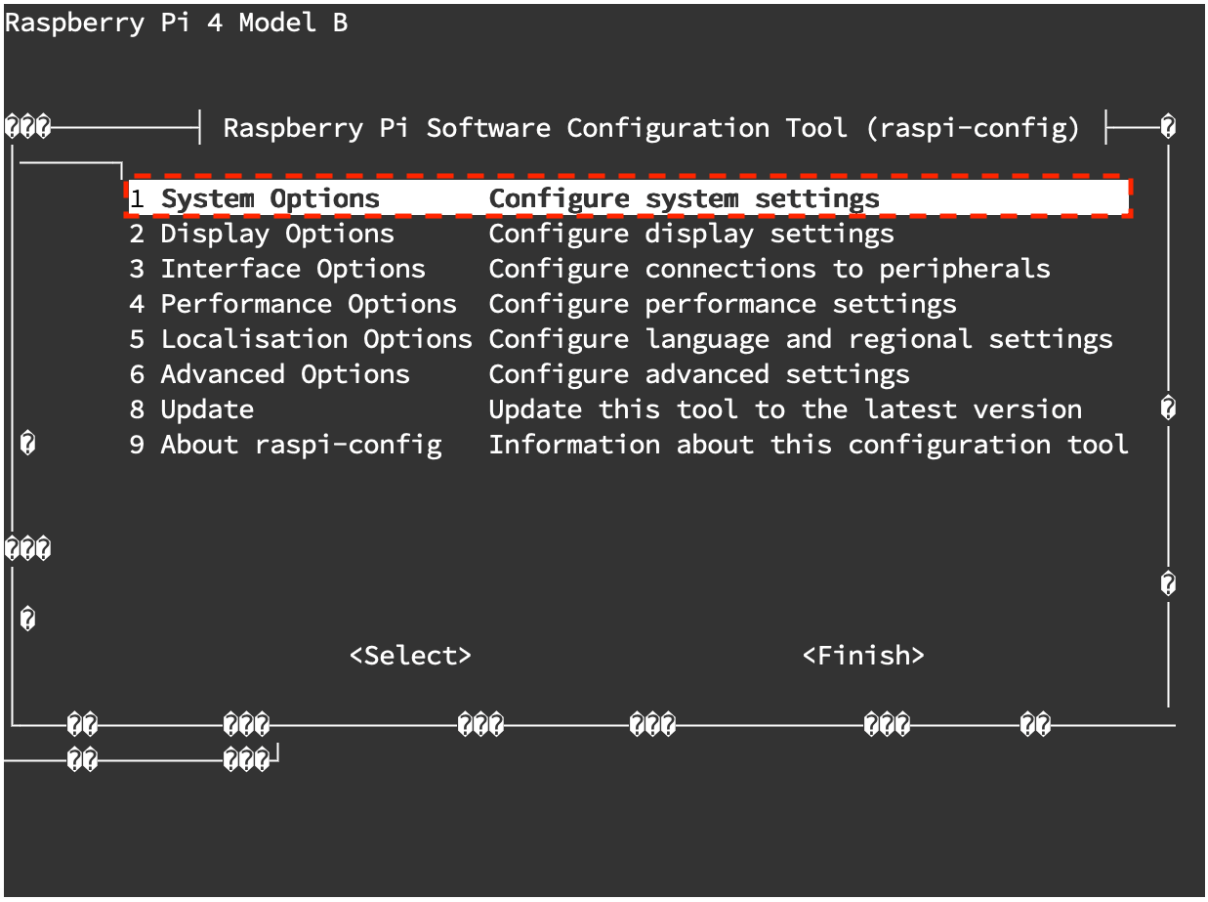
2. Open the configuration tool.

   ```
   sudo raspi-config
   ```

3. Select `System Options` (menu item 1).

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

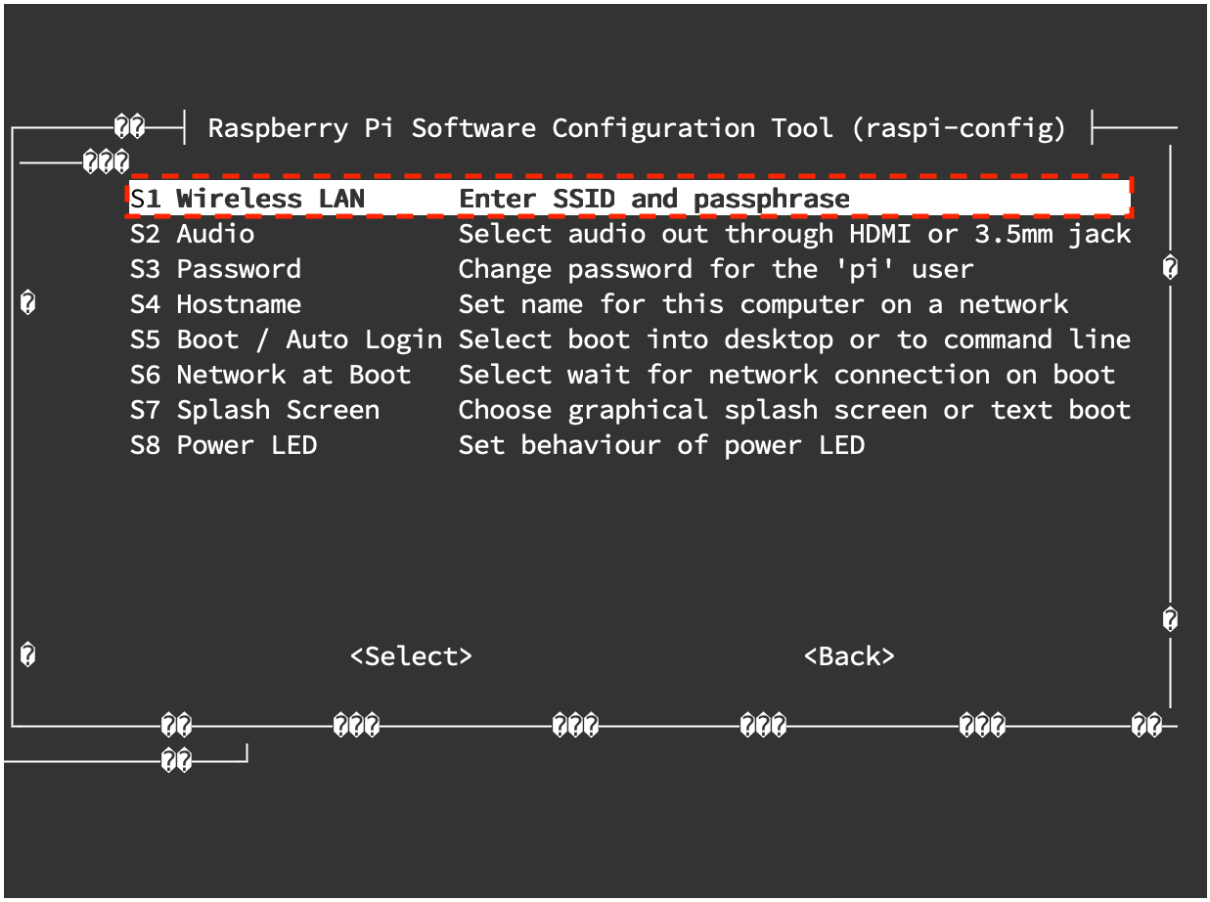
Figure 3-18: Configure main screen



4. Select `Wireless LAN` (menu item S1).

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

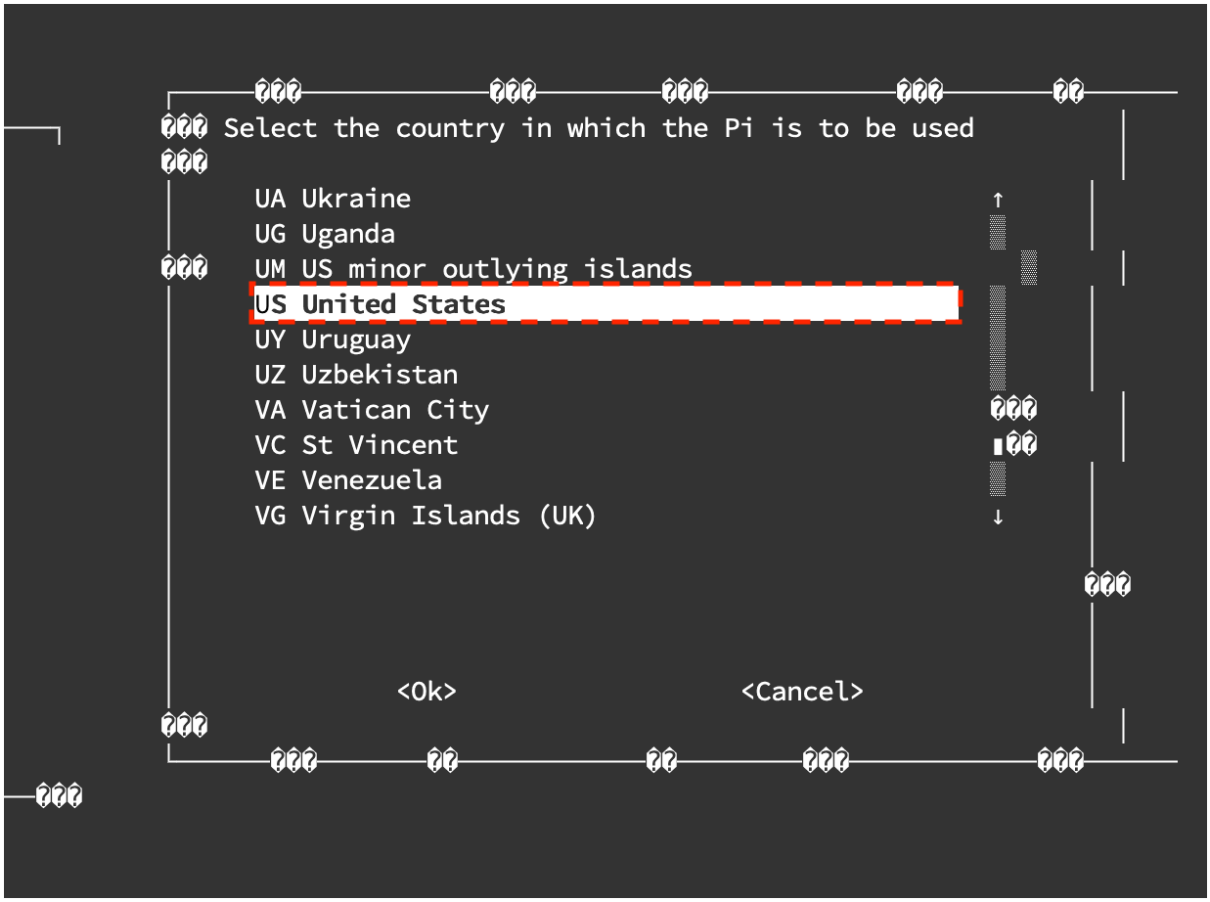
Figure 3-19: Configure system options



5. Set the country. In our example, we will use the United States. (Type “u” to quickly navigate the menu).

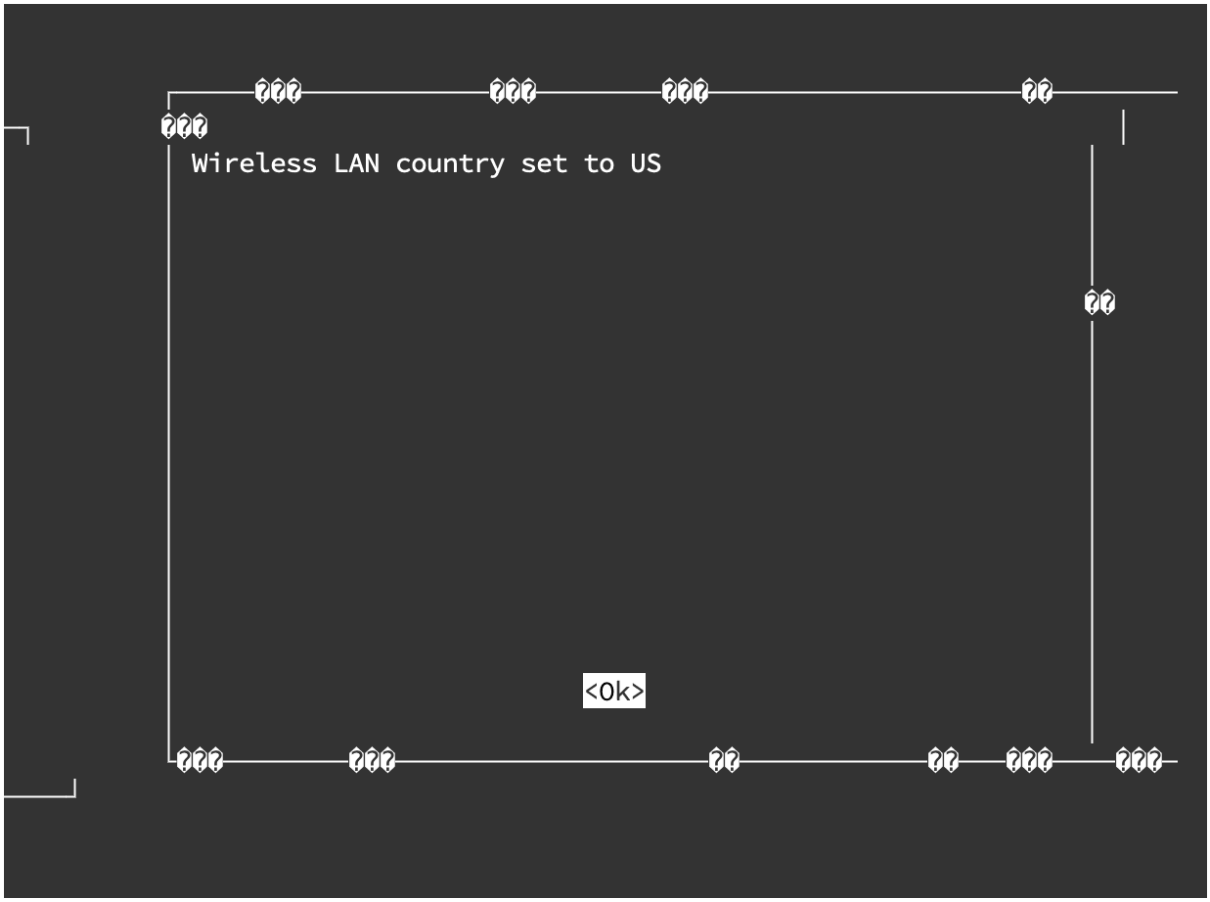
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-20: Set coutry



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

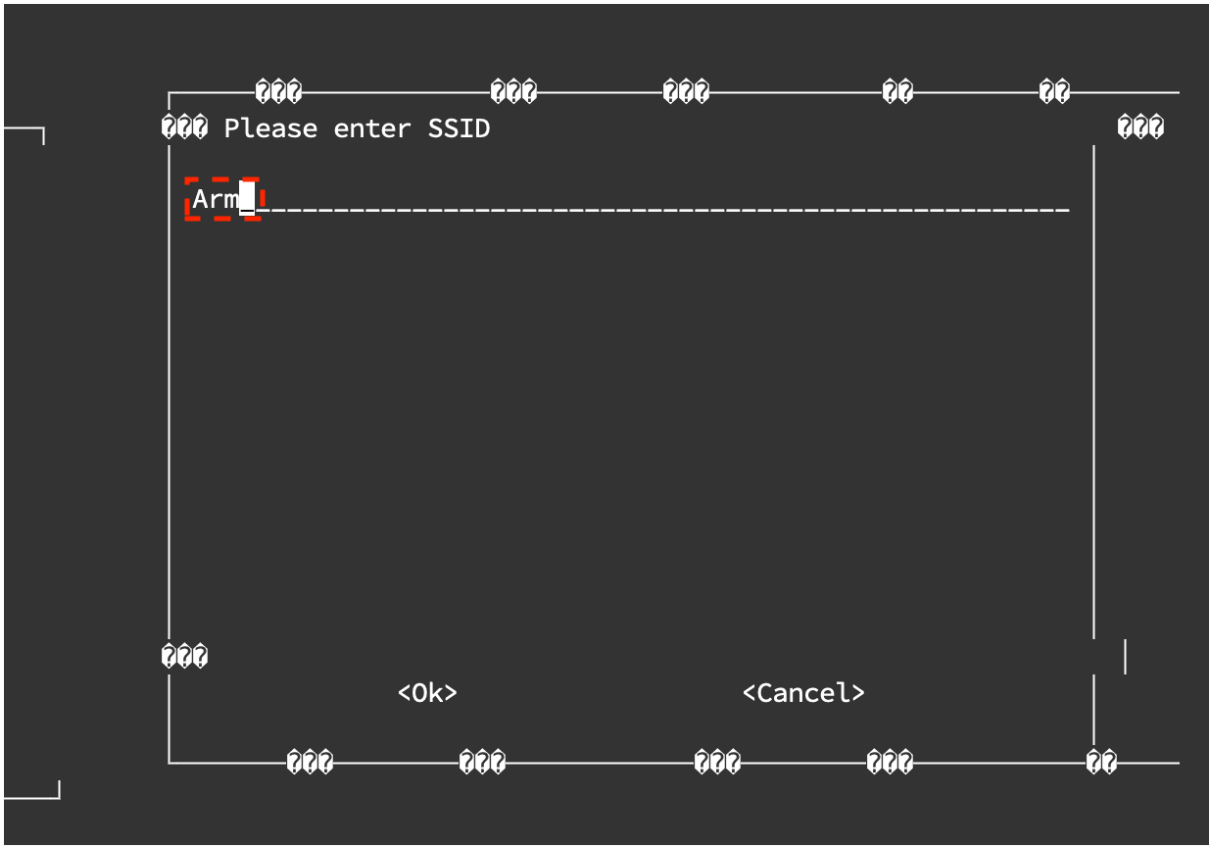
Figure 3-21: Set coutry confirmation



6. Set the SSID to Arm and the passphrase to `password`.

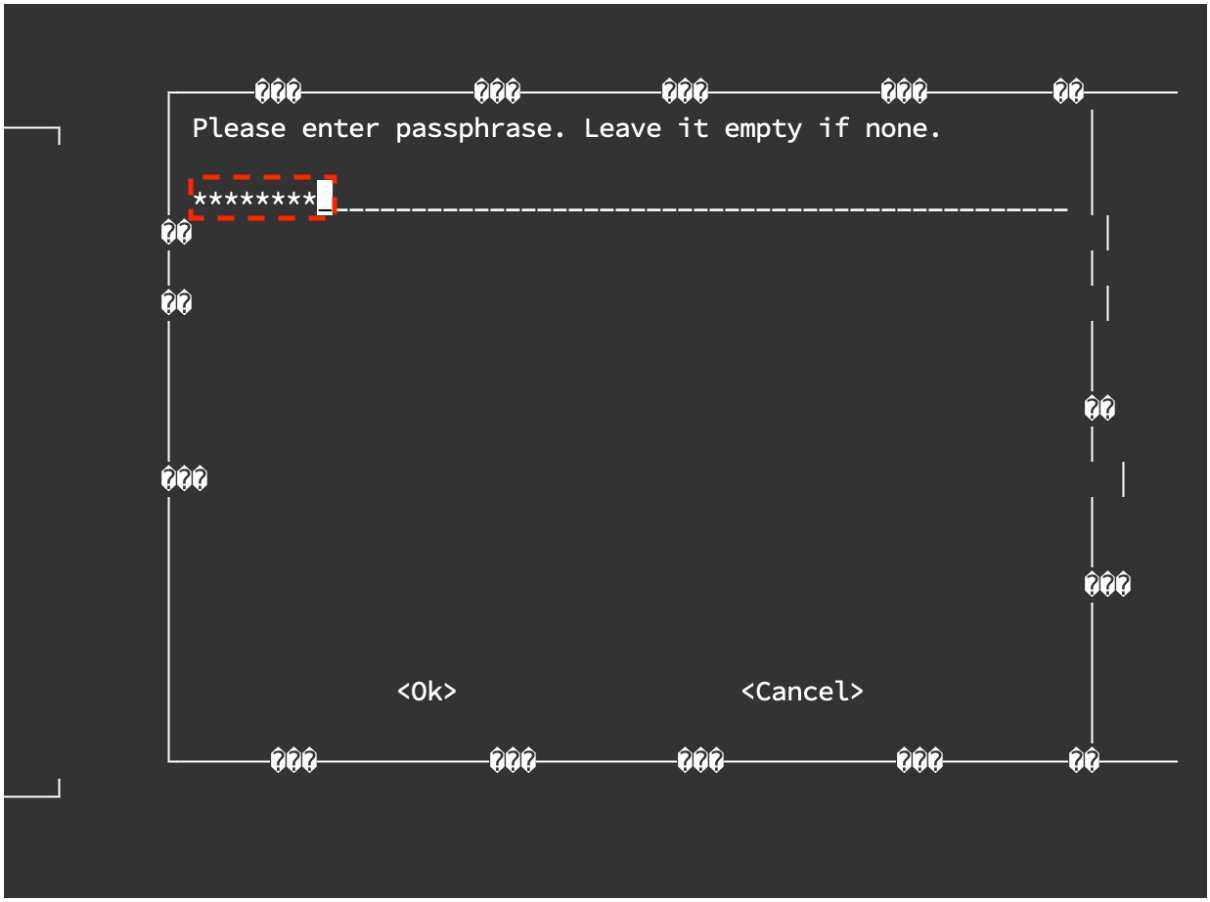
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-22: Set SSID



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

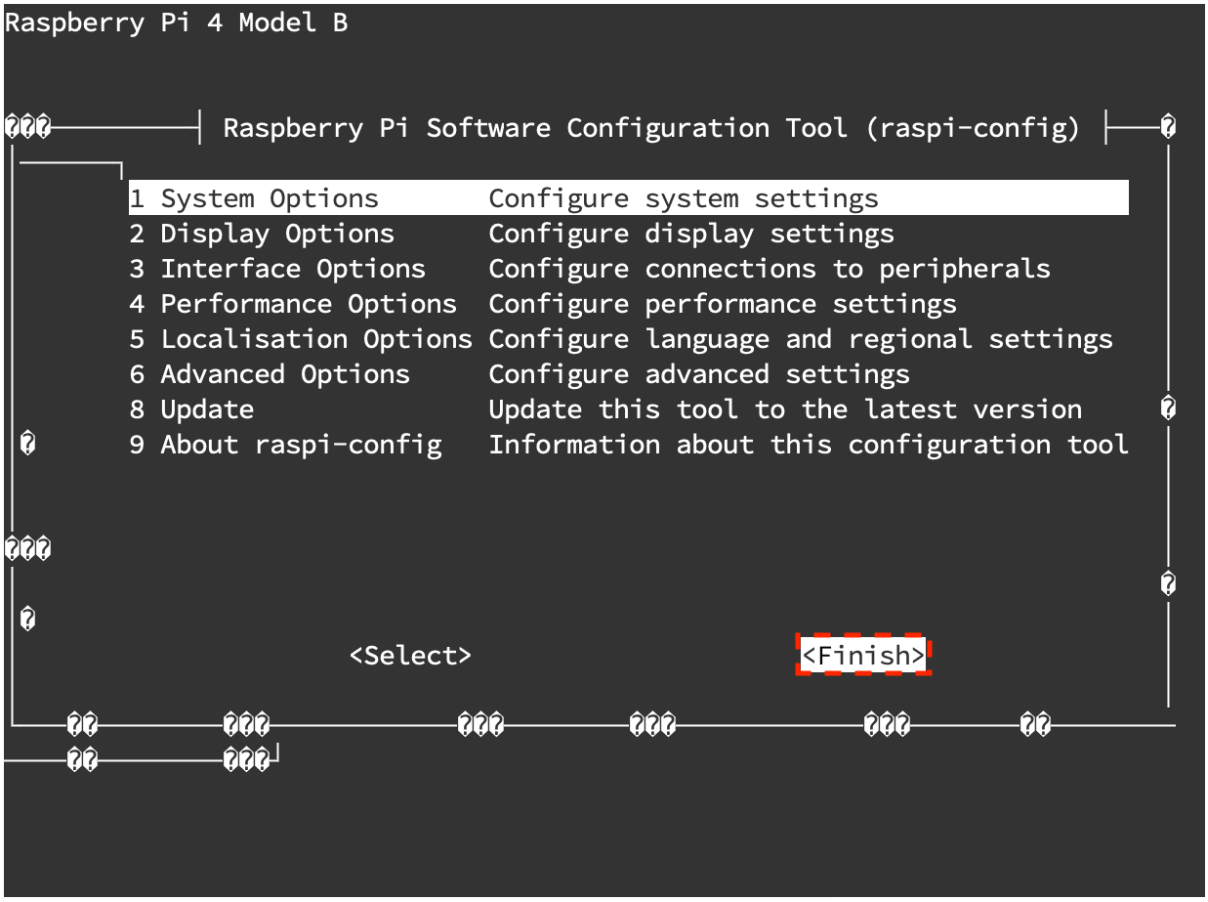
Figure 3-23: Set passphrase



7. At the main configuration screen, choose Finish. (Push the right arrow on your keyboard twice).

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

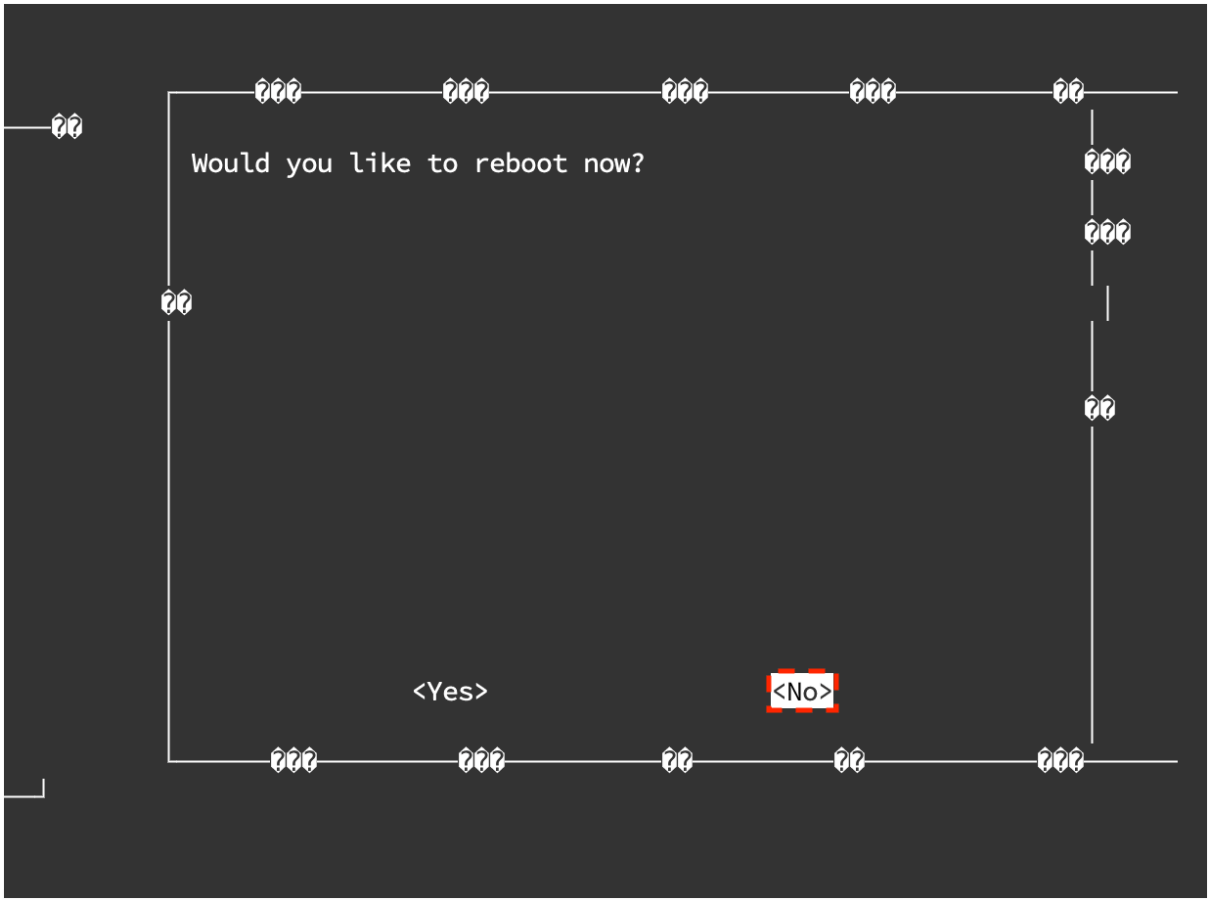
Figure 3-24: Choose finish



8. Do not reboot the machine.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-25: Decline to reboot

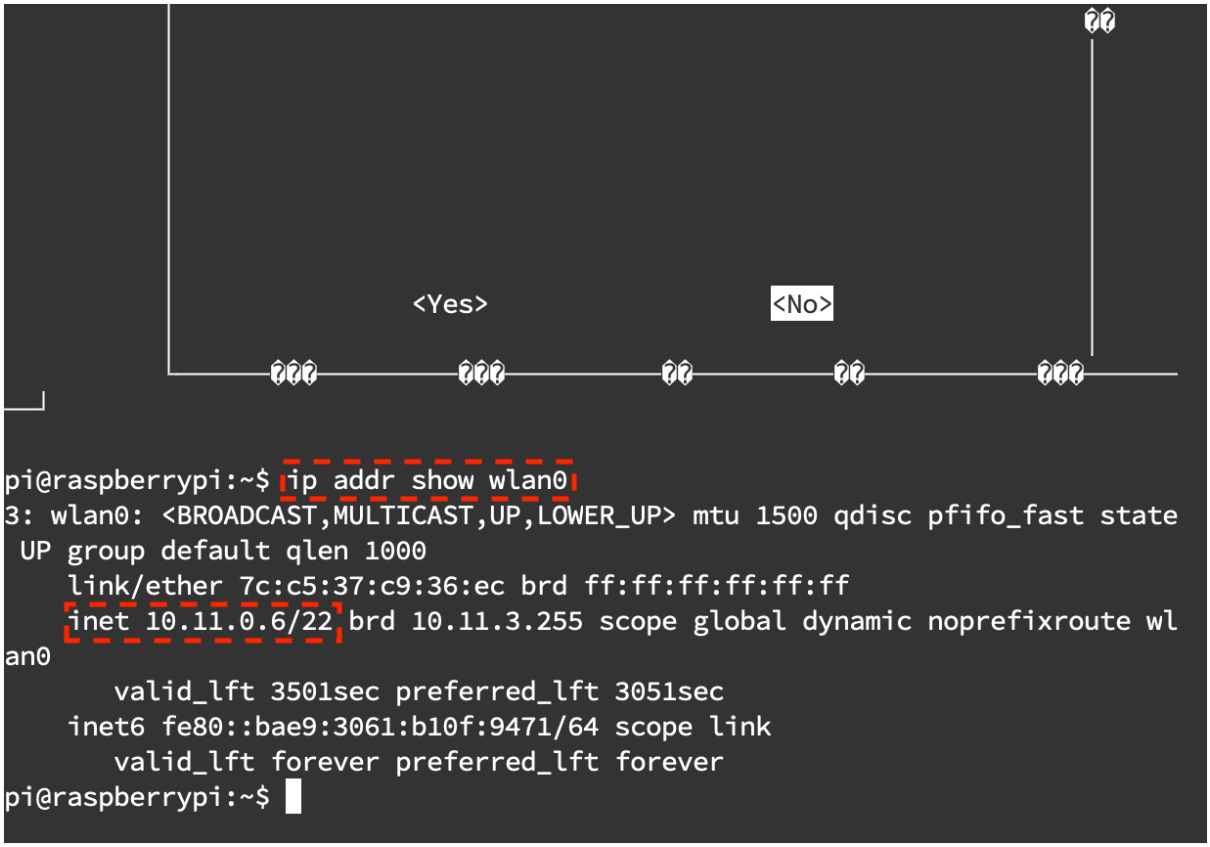


9. Verify that the wlan0 interface shows a valid IPv4 address that does not start with 169. Please note: the IPv6 address for this interface is a local-link auto address and not a real IPv6 address.

```
ip addr show wlan0
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-26: Show wlan0



10. Test the Wi-Fi connection to the internet by turning off the ethernet interface and trying to ping a public domain. Look for a result of `0% packet loss`.

```
sudo ifconfig eth0 down
ping -c5 aws.com
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-27: Zero packet loss

```
    inet 10.11.0.4/22 brd 10.11.3.255 scope global dynamic noprefixroute wl
an0
       valid_lft 3091sec preferred_lft 2641sec
    inet6 fe80::3ccb:6336:3c38:d42c/64 scope link
       valid_lft forever preferred_lft forever
pi@raspberrypi:~$ sudo ifconfig eth0 down
pi@raspberrypi:~$ ping -c5 aws.com
PING aws.com (18.67.65.31) 56(84) bytes of data.
64 bytes from server-18-67-65-31.iad89.r.cloudfront.net (18.67.65.31): icmp
_seq=1 ttl=238 time=0.949 ms
64 bytes from server-18-67-65-31.iad89.r.cloudfront.net (18.67.65.31): icmp
_seq=2 ttl=238 time=1.02 ms
64 bytes from server-18-67-65-31.iad89.r.cloudfront.net (18.67.65.31): icmp
_seq=3 ttl=238 time=0.979 ms
64 bytes from server-18-67-65-31.iad89.r.cloudfront.net (18.67.65.31): icmp
_seq=4 ttl=238 time=1.02 ms
64 bytes from server-18-67-65-31.iad89.r.cloudfront.net (18.67.65.31): icmp
_seq=5 ttl=238 time=1.01 ms

--- aws.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 0.949/0.994/1.021/0.026 ms
pi@raspberrypi:~$
```

11. After the test, you can bring the ethernet interface back up.

```
sudo ifconfig eth0 up
```

3.10 Communicate Using Bluetooth Low Energy

Learn how to use BLE on virtual Raspberry Pi 4 boards and virtual Android 12 devices.

- Use BLE Between Two Raspberry Pi Devices
  - Set Up the Raspberry Pi 4 BLE Node
  - Connect to the Node with the Raspberry Pi 4 BLE Master
- Use BLE Between a Raspberry Pi 4 Board and an Android Device
  - Set Up the Raspberry Pi 4 BLE Node
  - Connect to the Node with the Android BLE Master

AVH virtual devices support communicating through virtual Bluetooth Low Energy.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

In this guide, we show how to communicate between two virtual Raspberry Pi 4 virtual boards as well as between a virtual Raspberry Pi 4 board and a virtual Android device.

3.10.1 Use BLE Between Two Raspberry Pi Devices

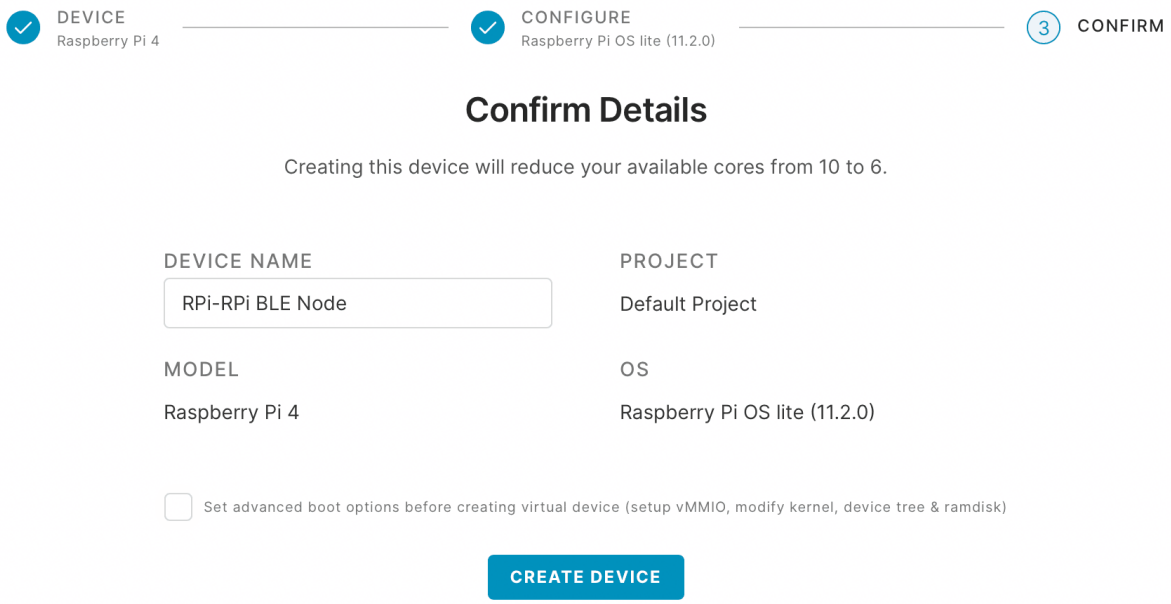
In this example, we will set up a Raspberry Pi 4 BLE Node and a Raspberry Pi 4 BLE Master.

3.10.2 Set Up the Raspberry Pi 4 BLE Node

To set up the Raspberry Pi 4 BLE node, do the following:

1. Create a new Raspberry Pi 4 instance using the lite firmware. In our example, we will use the name “RPi-RPi BLE Node”.

   Figure 3-28: Create RPi-RPi node



2. Log in using the default credentials (`pi/raspberry`) and open the Bluetooth tool.

```
bluetoothctl1
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-29: Raspberry Pi Login

```
Password:
Linux raspberrypi 5.10.92-v8+ #1514 SMP PREEMPT Mon Jan 17 17:39:38 GMT 2022 aa
rch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to se
t a new password.


Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~$ bluetoothctl
Agent registered
[CHG] Controller 3E:A3:EC:CE:B1:61 Pairable: yes
[bluetooth]#
```

3. Turn on Bluetooth.

```
power on
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-30: Power on Bluetooth

```
rch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to se
t a new password.


Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~$ bluetoothctl
Agent registered
[CHG] Controller 3E:A3:EC:CE:B1:61 Pairable: yes
[bluetooth]# power on
Changing power on succeeded
[bluetooth]#
```

4. Open the advertise menu.

```
menu advertise
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-31: Menu advertise

```
[bluetooth]# menu advertise
Menu advertise:
Available commands:
-------------------
uuids [uuid1 uuid2 ...]                          Set/Get advertise uuids
service [uuid] [data=xx xx ...]                  Set/Get advertise service dat
a
manufacturer [id] [data=xx xx ...]               Set/Get advertise manufacture
r data
data [type] [data=xx xx ...]                     Set/Get advertise data
discoverable [on/off]                            Set/Get advertise discoverabl
e
discoverable-timeout [seconds]                   Set/Get advertise discoverabl
e timeout
tx-power [on/off]                                Show/Enable/Disable TX power
to be advertised
name [on/off/name]                               Configure local name to be ad
vertised
appearance [on/off/value]                        Configure custom appearance t
o be advertised
duration [seconds]                               Set/Get advertise duration
timeout [seconds]                                Set/Get advertise timeout
secondary [1M/2M/Coded]                          Set/Get advertise secondary c
```

5. Set the manufacturer to `0xffff 0x12 0x34` then read back the values.

```
manufacturer 0xffff 0x12 0x34
manufacturer
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-32: Manufacturer node

```
tx-power [on/off]                                    Show/Enable/Disable TX power
to be advertised
name [on/off/name]                                   Configure local name to be ad
vertised
appearance [on/off/value]                            Configure custom appearance t
o be advertised
duration [seconds]                                   Set/Get advertise duration
timeout [seconds]                                    Set/Get advertise timeout
secondary [1M/2M/Coded]                              Set/Get advertise secondary c
hannel
clear [uuids/service/manufacturer/config-name...]    Clear advertise config
back                                                 Return to main menu
version                                              Display version
quit                                                 Quit program
exit                                                 Quit program
help                                                 Display help about this progr
am
export                                               Print environment variables
[bluetooth]# manufacturer 0xffff 0x12 0x34
[bluetooth]# manufacturer
Manufacturer: 65535
  12 34                                               .4
[bluetooth]#
```

6. Set the name to `avh-demo-1` then read back the value.

```
name avh-demo-1
name
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-33: Set name node

```
vertised
appearance [on/off/value]                          Configure custom appearance t
o be advertised
duration [seconds]                                 Set/Get advertise duration
timeout [seconds]                                  Set/Get advertise timeout
secondary [1M/2M/Coded]                            Set/Get advertise secondary c
hannel
clear [uuids/service/manufacturer/config-name...] Clear advertise config
back                                               Return to main menu
version                                            Display version
quit                                               Quit program
exit                                               Quit program
help                                               Display help about this progr
am
export                                             Print environment variables
[bluetooth]# manufacturer 0xffff 0x12 0x34
[bluetooth]# manufacturer
Manufacturer: 65535
  12 34                                             .4
[bluetooth]# name avh-demo-1
[bluetooth]# name
LocalName: avh-demo-1
[bluetooth]#
```

7. Go back to the main menu.

```
back
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-34: Main menu

```
[bluetooth]# back
Menu main:
Available commands:
-------------------
advertise                                          Advertise Options Submenu
scan                                               Scan Options Submenu
gatt                                               Generic Attribute Submenu
list                                               List available controllers
show [ctrl]                                        Controller information
select <ctrl>                                      Select default controller
devices                                            List available devices
paired-devices                                     List paired devices
system-alias <name>                                Set controller alias
reset-alias                                        Reset controller alias
power <on/off>                                     Set controller power
pairable <on/off>                                  Set controller pairable mode
discoverable <on/off>                              Set controller discoverable m
ode
discoverable-timeout [value]                       Set discoverable timeout
agent <on/off/capability>                          Enable/disable agent with giv
en capability
default-agent                                      Set agent as the default one
advertise <on/off/type>                            Enable/disable advertising wi
```

8. Open the generic attributes menu.

```
menu gatt
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-35: Generic attributes menu

```
[bluetooth]# menu gatt
Menu gatt:
Available commands:
-------------------
list-attributes [dev/local]                    List attributes
select-attribute <attribute/UUID>              Select attribute
attribute-info [attribute/UUID]                Select attribute
read [offset]                                  Read attribute value
write <data=xx xx ...> [offset] [type]         Write attribute value
acquire-write                                  Acquire Write file descriptor
release-write                                  Release Write file descriptor
acquire-notify                                 Acquire Notify file descripto
r
release-notify                                 Release Notify file descripto
r
notify <on/off>                                Notify attribute value
clone [dev/attribute/UUID]                     Clone a device or attribute
register-application [UUID ...]                Register profile to connect
unregister-application                         Unregister profile
register-service <UUID> [handle]               Register application service.
unregister-service <UUID/object>               Unregister application servic
e
register-includes <UUID> [handle]              Register as Included service
```

9. Register a new service. At the prompt, enter `yes`.

```
register-service e2d36f99-8909-4136-9a49-d825508b297b
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-36: Register a new service

```
unregister-includes <Service-UUID><Inc-UUID>      Unregister Included service.
register-characteristic <UUID> <Flags=read,write,notify...> [handle] Register a
pplication characteristic
unregister-characteristic <UUID/object>           Unregister application charac
teristic
register-descriptor <UUID> <Flags=read,write...> [handle] Register application
descriptor
unregister-descriptor <UUID/object>               Unregister application descri
ptor
back                                              Return to main menu
version                                           Display version
quit                                              Quit program
exit                                              Quit program
help                                              Display help about this progr
am
export                                            Print environment variables
[bluetooth]# register-service e2d36f99-8909-4136-9a49-d825508b297b
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/app/service0
        e2d36f99-8909-4136-9a49-d825508b297b
        Vendor specific
[/org/bluez/app/service0] Primary (yes/no): yes
[bluetooth]#
```

10. Register a new read-only characteristic. In our example, we will enter `07` at the prompt.

```
register-characteristic 0x1234 read
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-37: Register a read-only characteristic

```
descriptor
unregister-descriptor <UUID/object>              Unregister application descri
ptor
back                                             Return to main menu
version                                          Display version
quit                                             Quit program
exit                                             Quit program
help                                             Display help about this progr
am
export                                           Print environment variables
[bluetooth]# register-service e2d36f99-8909-4136-9a49-d825508b297b
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/app/service0
        e2d36f99-8909-4136-9a49-d825508b297b
        Vendor specific
[/org/bluez/app/service0] Primary (yes/no): yes
[bluetooth]# register-characteristic 0x1234 read
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/app/service0/chrc0
        0x1234
        Unknown
[/org/bluez/app/service0/chrc0] Enter value: 07
[bluetooth]#
```

11. Register a new read-write characteristic. In our example, we will enter `13` at the prompt.

```
register-characteristic 0x5678 read,write
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-38: Register a read-write characteristic

```
exit                                                Quit program
help                                                Display help about this progr
am
export                                              Print environment variables
[bluetooth]# register-service e2d36f99-8909-4136-9a49-d825508b297b
[NEW] Primary Service (Handle 0x0000)
        /org/bluez/app/service0
        e2d36f99-8909-4136-9a49-d825508b297b
        Vendor specific
[/org/bluez/app/service0] Primary (yes/no): yes
[bluetooth]# register-characteristic 0x1234 read
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/app/service0/chrc0
        0x1234
        Unknown
[/org/bluez/app/service0/chrc0] Enter value: 07
[bluetooth]# register-characteristic 0x5678 read,write
[NEW] Characteristic (Handle 0x0000)
        /org/bluez/app/service0/chrc1
        0x5678
        Unknown
[/org/bluez/app/service0/chrc1] Enter value: 13
[bluetooth]#
```

12. Register the application. You should see a response `Application Registered`.

```
register-application
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-39: Register the application

```
[bluetooth]# register-application
[CHG] Primary Service (Handle 0x0013)
        /org/bluez/app/service0
        e2d36f99-8909-4136-9a49-d825508b297b
        Vendor specific
[CHG] Characteristic (Handle 0x0015)
        /org/bluez/app/service0/chrc1
        0x5678
        Unknown
[CHG] Characteristic (Handle 0x0017)
        /org/bluez/app/service0/chrc0
        0x1234
        Unknown
[CHG] Controller 3E:A3:EC:CE:B1:61 UUIDs: 00001801-0000-1000-8000-00805f9b34fb
[CHG] Controller 3E:A3:EC:CE:B1:61 UUIDs: 00001800-0000-1000-8000-00805f9b34fb
[CHG] Controller 3E:A3:EC:CE:B1:61 UUIDs: e2d36f99-8909-4136-9a49-d825508b297b
[CHG] Controller 3E:A3:EC:CE:B1:61 UUIDs: 00001200-0000-1000-8000-00805f9b34fb
[CHG] Controller 3E:A3:EC:CE:B1:61 UUIDs: 0000110c-0000-1000-8000-00805f9b34fb
[CHG] Controller 3E:A3:EC:CE:B1:61 UUIDs: 0000110e-0000-1000-8000-00805f9b34fb
[CHG] Controller 3E:A3:EC:CE:B1:61 UUIDs: 0000180a-0000-1000-8000-00805f9b34fb
Application registered
[CHG] Controller 3E:A3:EC:CE:B1:61 UUIDs: 00001801-0000-1000-8000-00805f9b34fb
[CHG] Controller 3E:A3:EC:CE:B1:61 UUIDs: 00001800-0000-1000-8000-00805f9b34fb
```

13. Go back to the main menu again.

```
back
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-40: Main menu

```
[bluetooth]# back
Menu main:
Available commands:
-------------------
advertise                                          Advertise Options Submenu
scan                                               Scan Options Submenu
gatt                                               Generic Attribute Submenu
list                                               List available controllers
show [ctrl]                                        Controller information
select <ctrl>                                      Select default controller
devices                                            List available devices
paired-devices                                     List paired devices
system-alias <name>                                Set controller alias
reset-alias                                        Reset controller alias
power <on/off>                                     Set controller power
pairable <on/off>                                  Set controller pairable mode
discoverable <on/off>                              Set controller discoverable m
ode
discoverable-timeout [value]                       Set discoverable timeout
agent <on/off/capability>                          Enable/disable agent with giv
en capability
default-agent                                      Set agent as the default one
advertise <on/off/type>                            Enable/disable advertising wi
```

14. Turn on advertising.

```
advertise on
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-41: Advertise node

```
block [dev]                                          Block device
unblock [dev]                                        Unblock device
remove <dev>                                         Remove device
connect <dev>                                        Connect device
disconnect [dev]                                     Disconnect device
menu <name>                                          Select submenu
version                                              Display version
quit                                                 Quit program
exit                                                 Quit program
help                                                 Display help about this progr
am
export                                               Print environment variables
[bluetooth]# advertise on
[CHG] Controller 3E:A3:EC:CE:B1:61 SupportedInstances: 0x04 (4)
[CHG] Controller 3E:A3:EC:CE:B1:61 ActiveInstances: 0x01 (1)
Advertising object registered
Manufacturer: 65535
  12 34                                               .4
Tx Power: off
LocalName: avh-demo-1
Appearance: off
Discoverable: on
[bluetooth]#
```

15. Your Bluetooth node is now set up. Proceed to the next step to connect to this node using a second Raspberry Pi 4 board.

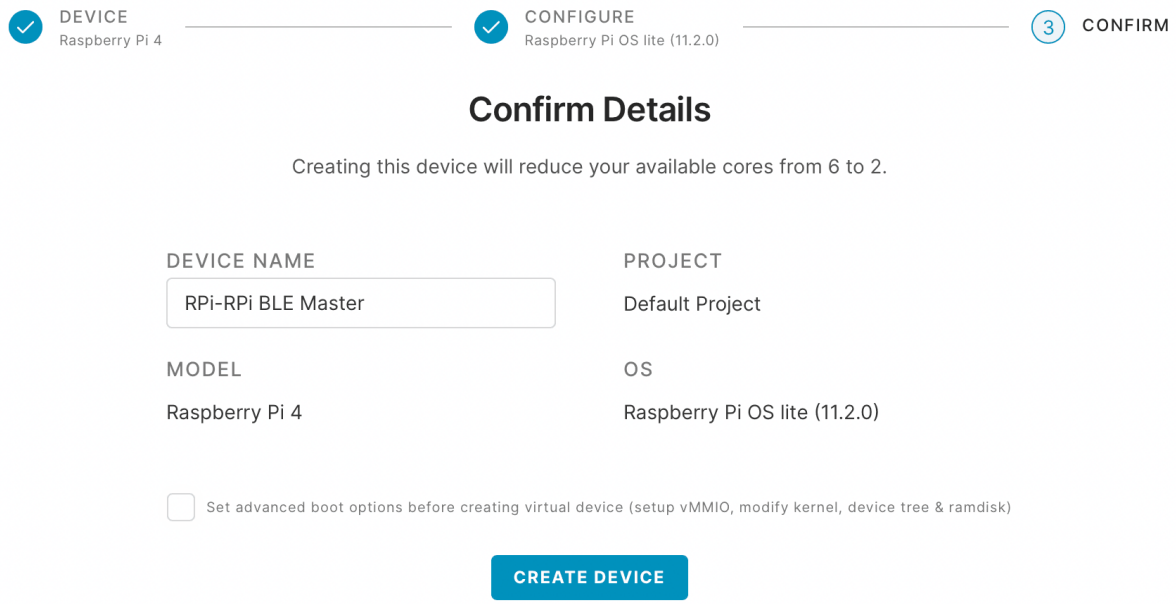
3.10.3 Connect to the Node with the Raspberry Pi 4 BLE Master

To connect to the node with the Raspberry Pi 4 BLE master, do the following:

1. Create a new Raspberry Pi 4 instance using the lite firmware. In our example, we will use the name “RPi-RPi BLE Master”.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-42: Create Raspberry Pi 4 instance



2. Log in using the default credentials (`pi/raspberry`) and open the Bluetooth tool.

```
bluetoothctl1
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-43: Raspberry Pi Login

```
Password:
Linux raspberrypi 5.10.92-v8+ #1514 SMP PREEMPT Mon Jan 17 17:39:38 GMT 2022 aa
rch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to se
t a new password.


Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~$ bluetoothctl
Agent registered
[CHG] Controller 3E:A3:EC:CE:B1:61 Pairable: yes
[bluetooth]#
```

3. Turn on Bluetooth.

```
power on
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-44: Power on Bluetooth

```
rch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to se
t a new password.


Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~$ bluetoothctl
Agent registered
[CHG] Controller 3E:A3:EC:CE:B1:61 Pairable: yes
[bluetooth]# power on
Changing power on succeeded
[bluetooth]#
```

4. Scan for Bluetooth devices.

```
scan on
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-45: Scan node

```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to se
t a new password.


Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~$ bluetoothctl
Agent registered
[CHG] Controller 3E:A3:EC:DF:D6:A4 Pairable: yes
[bluetooth]# power on
Changing power on succeeded
[bluetooth]# scan on
Discovery started
[CHG] Controller 3E:A3:EC:DF:D6:A4 Discovering: yes
[NEW] Device 3E:A3:EC:CE:B1:61 avh-demo-1
[bluetooth]#
```

5. Connect to the `avh-demo-1` device. Make note of the two IDs starting with `00001234` and `00005678`, referring to the custom characteristics that you set up.

```
connect 3E:A3:EC:CE:B1:61
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-46: Connect node

```
[bluetooth]# connect 3E:A3:EC:CE:B1:61
Attempting to connect to 3E:A3:EC:CE:B1:61
[CHG] Device 3E:A3:EC:CE:B1:61 Connected: yes
Connection successful
[NEW] Primary Service (Handle 0xb964)
        /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0006
        00001801-0000-1000-8000-00805f9b34fb
        Generic Attribute Profile
[NEW] Characteristic (Handle 0x0d48)
        /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0006/char0007
        00002a05-0000-1000-8000-00805f9b34fb
        Service Changed
[NEW] Descriptor (Handle 0x0d90)
        /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0006/char0007/desc0009
        00002902-0000-1000-8000-00805f9b34fb
        Client Characteristic Configuration
[NEW] Characteristic (Handle 0x0d48)
        /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0006/char000a
        00002b29-0000-1000-8000-00805f9b34fb
        Client Supported Features
[NEW] Characteristic (Handle 0x0d48)
        /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0006/char000c
        00002b2a-0000-1000-8000-00805f9b34fb
        Database Hash
[NEW] Characteristic (Handle 0x0d48)
        /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0006/char000e
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-47: Connect node

```
        Device Information
[NEW] Characteristic (Handle 0x0d48)
        /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0010/char0011
        00002a50-0000-1000-8000-00805f9b34fb
        PnP ID
[NEW] Primary Service (Handle 0xcd04)
        /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013
        e2d36f99-8909-4136-9a49-d825508b297b
        Vendor specific
[NEW] Characteristic (Handle 0x0d48)
        /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0014
        00005678-0000-1000-8000-00805f9b34fb
        Unknown
[NEW] Characteristic (Handle 0x0d48)
        /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016
        00001234-0000-1000-8000-00805f9b34fb
        Unknown
[CHG] Device 3E:A3:EC:CE:B1:61 UUIDs: 00001800-0000-1000-8000-00805f9b34fb
[CHG] Device 3E:A3:EC:CE:B1:61 UUIDs: 00001801-0000-1000-8000-00805f9b34fb
[CHG] Device 3E:A3:EC:CE:B1:61 UUIDs: 0000180a-0000-1000-8000-00805f9b34fb
[CHG] Device 3E:A3:EC:CE:B1:61 UUIDs: e2d36f99-8909-4136-9a49-d825508b297b
[CHG] Device 3E:A3:EC:CE:B1:61 ServicesResolved: yes
[CHG] Device 3E:A3:EC:CE:B1:61 Name: raspberrypi
[CHG] Device 3E:A3:EC:CE:B1:61 Alias: raspberrypi
[CHG] Device 3E:A3:EC:CE:B1:61 Modalias: usb:v1D6Bp0246d0537
[avh-demo-1]#
```

6. Open the generic attributes menu.

```
menu gatt
```

7. Select the read-only attribute with an ID starting with `00001234`. The prompt should change to the appropriate characteristic.

```
select-attribute 00001234-0000-1000-8000-00805f9b34fb
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-48: Select attribute

```
clone [dev/attribute/UUID]                          Clone a device or attribute
register-application [UUID ...]                     Register profile to connect
unregister-application                              Unregister profile
register-service <UUID> [handle]                    Register application service.
unregister-service <UUID/object>                    Unregister application servic
e
register-includes <UUID> [handle]                   Register as Included service
in.
unregister-includes <Service-UUID><Inc-UUID>        Unregister Included service.
register-characteristic <UUID> <Flags=read,write,notify...> [handle] Register a
pplication characteristic
unregister-characteristic <UUID/object>             Unregister application charac
teristic
register-descriptor <UUID> <Flags=read,write...> [handle] Register application
descriptor
unregister-descriptor <UUID/object>                 Unregister application descri
ptor
back                                                Return to main menu
version                                             Display version
quit                                                Quit program
exit                                                Quit program
help                                                Display help about this progr
am
export                                              Print environment variables
[avh-demo-1]# select-attribute 00001234-0000-1000-8000-00805f9b34fb
[raspberrypi:/service0013/char0016]#
```

8. Read the value. You should see the value you set previously, represented in hexidecimal.

```
read
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-49: Read node

```
register-includes <UUID> [handle]                  Register as Included service
in.
unregister-includes <Service-UUID><Inc-UUID>       Unregister Included service.
register-characteristic <UUID> <Flags=read,write,notify...> [handle] Register a
pplication characteristic
unregister-characteristic <UUID/object>            Unregister application charac
teristic
register-descriptor <UUID> <Flags=read,write...> [handle] Register application
descriptor
unregister-descriptor <UUID/object>                Unregister application descri
ptor
back                                               Return to main menu
version                                            Display version
quit                                               Quit program
exit                                               Quit program
help                                               Display help about this progr
am
export                                             Print environment variables
[avh-demo-1]# select-attribute 00001234-0000-1000-8000-00805f9b34fb
[raspberrypi:/service0013/char0016]# read
Attempting to read /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016
[CHG] Attribute /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016 Valu
e:
  07                                                .
  07                                                .
[raspberrypi:/service0013/char0016]#
```

9. Attempt to write a new value. You should see an error because the characteristic is read-only.

```
write 08
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-50: Write error

```
pplication characteristic
unregister-characteristic <UUID/object>          Unregister application charac
teristic
register-descriptor <UUID> <Flags=read,write...> [handle] Register application
descriptor
unregister-descriptor <UUID/object>              Unregister application descri
ptor
back                                             Return to main menu
version                                          Display version
quit                                             Quit program
exit                                             Quit program
help                                             Display help about this progr
am
export                                           Print environment variables
[avh-demo-1]# select-attribute 00001234-0000-1000-8000-00805f9b34fb
[raspberrypi:/service0013/char0016]# read
Attempting to read /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016
[CHG] Attribute /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016 Valu
e:
  07                                              .
  07                                              .
[raspberrypi:/service0013/char0016]# write 08
Invalid value at index 0
Attempting to write /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016
Failed to write: org.bluez.Error.NotSupported
[raspberrypi:/service0013/char0016]#
```

10. Select the read-write characteristic with an ID starting with `00005678`. The prompt should change to the appropriate characteristic.

```
select-attribute 00005678-0000-1000-8000-00805f9b34fb
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-51: Read-write characteristic

```
teristic
register-descriptor <UUID> <Flags=read,write...> [handle] Register application
descriptor
unregister-descriptor <UUID/object>              Unregister application descri
ptor
back                                             Return to main menu
version                                          Display version
quit                                             Quit program
exit                                             Quit program
help                                             Display help about this progr
am
export                                           Print environment variables
[avh-demo-1]# select-attribute 00001234-0000-1000-8000-00805f9b34fb
[raspberrypi:/service0013/char0016]# read
Attempting to read /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016
[CHG] Attribute /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016 Valu
e:
  07                                              .
  07                                              .
[raspberrypi:/service0013/char0016]# write 08
Invalid value at index 0
Attempting to write /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016
Failed to write: org.bluez.Error.NotSupported
[raspberrypi:/service0013/char0016]# select-attribute 00005678-0000-1000-8000-0
0805f9b34fb
[raspberrypi:/service0013/char0014]#
```

11. Read the value. You should see the value you set previously, represented in hexidecimal.

```
read
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-52: Read node

```
version                                          Display version
quit                                             Quit program
exit                                             Quit program
help                                             Display help about this progr
am
export                                           Print environment variables
[avh-demo-1]# select-attribute 00001234-0000-1000-8000-00805f9b34fb
[raspberrypi:/service0013/char0016]# read
Attempting to read /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016
[CHG] Attribute /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016 Valu
e:
  07                                              .
  07                                              .
[raspberrypi:/service0013/char0016]# write 08
Invalid value at index 0
Attempting to write /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016
Failed to write: org.bluez.Error.NotSupported
[raspberrypi:/service0013/char0016]# select-attribute 00005678-0000-1000-8000-0
0805f9b34fb
[raspberrypi:/service0013/char0014]# read
Attempting to read /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0014
[CHG] Attribute /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0014 Valu
e:
  0d                                              .
  0d                                              .
[raspberrypi:/service0013/char0014]#
```

12. Write a new value then read it back. In our example, we will use `14`, which is `0d` in hexidecimal.

```
write 14
read
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-53: New value node

```
Attempting to read /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016
[CHG] Attribute /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016 Valu
e:
  07                                                .
  07                                                .
[raspberrypi:/service0013/char0016]# write 08
Invalid value at index 0
Attempting to write /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0016
Failed to write: org.bluez.Error.NotSupported
[raspberrypi:/service0013/char0016]# select-attribute 00005678-0000-1000-8000-0
0805f9b34fb
[raspberrypi:/service0013/char0014]# read
Attempting to read /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0014
[CHG] Attribute /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0014 Valu
e:
  0d                                                .
  0d                                                .
[raspberrypi:/service0013/char0014]# write 14
Attempting to write /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0014
[raspberrypi:/service0013/char0014]# read
Attempting to read /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0014
[CHG] Attribute /org/bluez/hci0/dev_3E_A3_EC_CE_B1_61/service0013/char0014 Valu
e:
  0e                                                .
  0e                                                .
[raspberrypi:/service0013/char0014]#
```

3.10.4 Use BLE Between a Raspberry Pi 4 Board and an Android Device

In this example, we will set up a Raspberry Pi 4 BLE Node and an Android BLE Master.

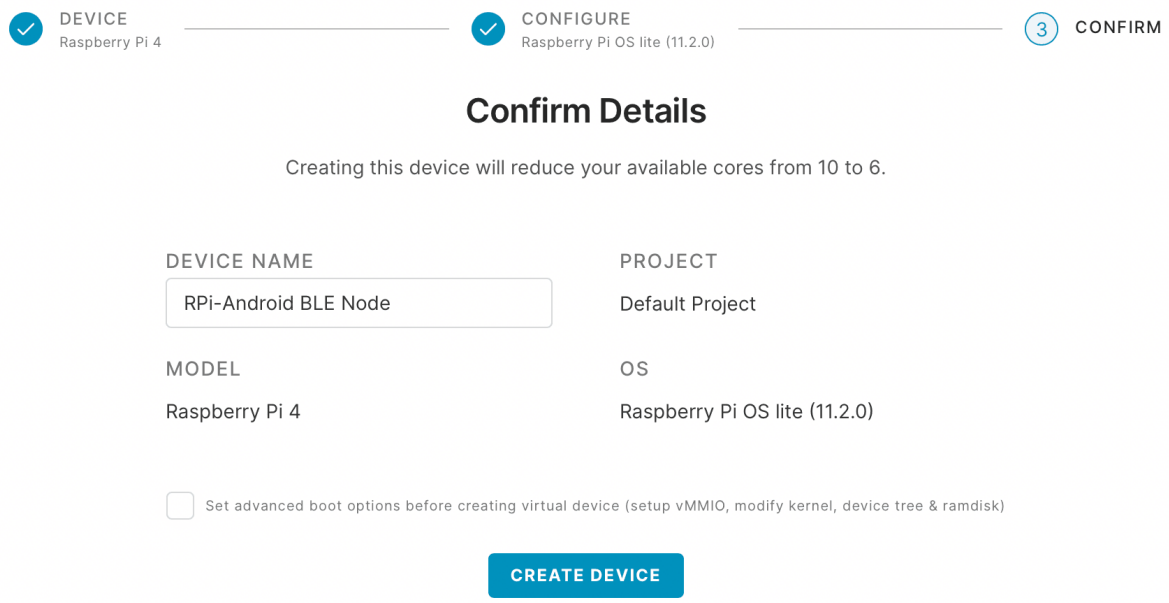
3.10.5 Set Up the Raspberry Pi 4 BLE Node

To set up the Raspberry Pi 4 BLE node, do the following:

1. Create a new Raspberry Pi 4 board. In our example, we will use the name “RPi-Android BLE Node”.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-54: Create Raspberry Pi 4 instance



2. Follow the steps in the previous section to set up a Raspberry Pi BLE Node. For our example, we will set the name of the Bluetooth node to "avh-demo-2'.

```
bluetoothctl
power on
menu advertise
manufacturer 0xffff 0x12 0x34
name avh-demo-2
back
menu gatt
register-service e2d36f99-8909-4136-9a49-d825508b297b
register-characteristic 0x1234 read
register-characteristic 0x5678 read,write
register-application
back
advertise on
```

3. When the node is advertising, make note of the node's MAC address (`3E:A3:EC:7B:47:66` in our example).

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-55: Node MAC address

```
unblock [dev]                                    Unblock device
remove <dev>                                     Remove device
connect <dev>                                    Connect device
disconnect [dev]                                 Disconnect device
menu <name>                                      Select submenu
version                                          Display version
quit                                             Quit program
exit                                             Quit program
help                                             Display help about this
program
export                                           Print environment variab
les
[bluetooth]# advertise on
[CHG] Controller 3E:A3:EC:7B:47:66 SupportedInstances: 0x04 (4)
[CHG] Controller 3E:A3:EC:7B:47:66 ActiveInstances: 0x01 (1)
Advertising object registered
Manufacturer: 65535
  12 34                                           .4
Tx Power: off
LocalName: avh-demo-2
Appearance: off
Discoverable: on
[bluetooth]#
```

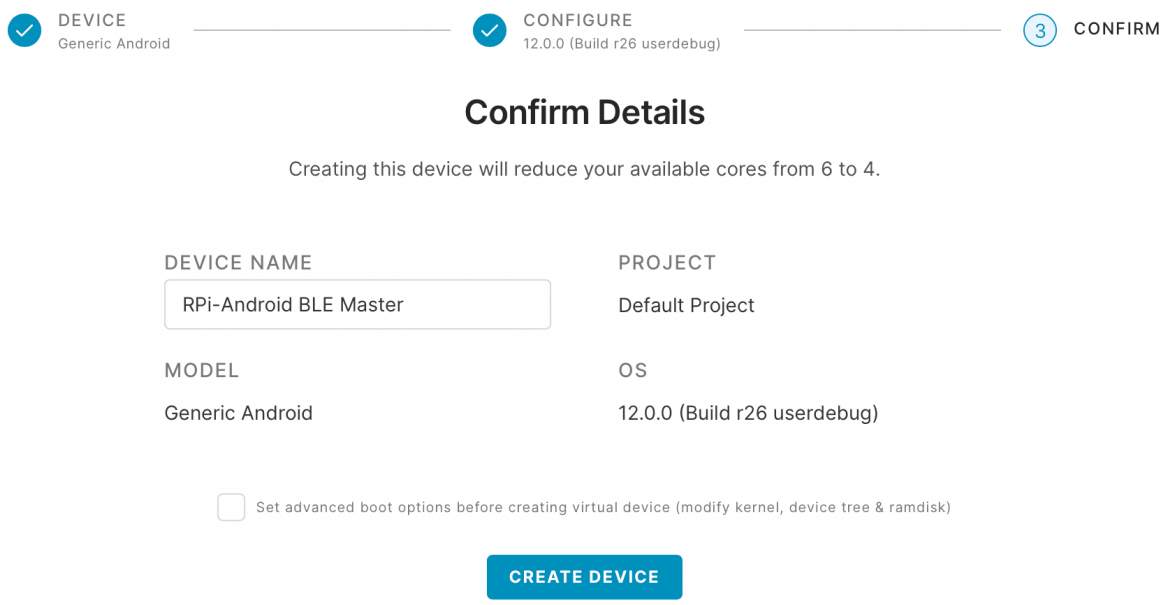
3.10.6 Connect to the Node with the Android BLE Master

To connect to the node with the Android BLE master, do the following:

1. Create a new Android device running firmware 12.0.0. For our example, we will use the name “RPi-Android BLE Master”.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

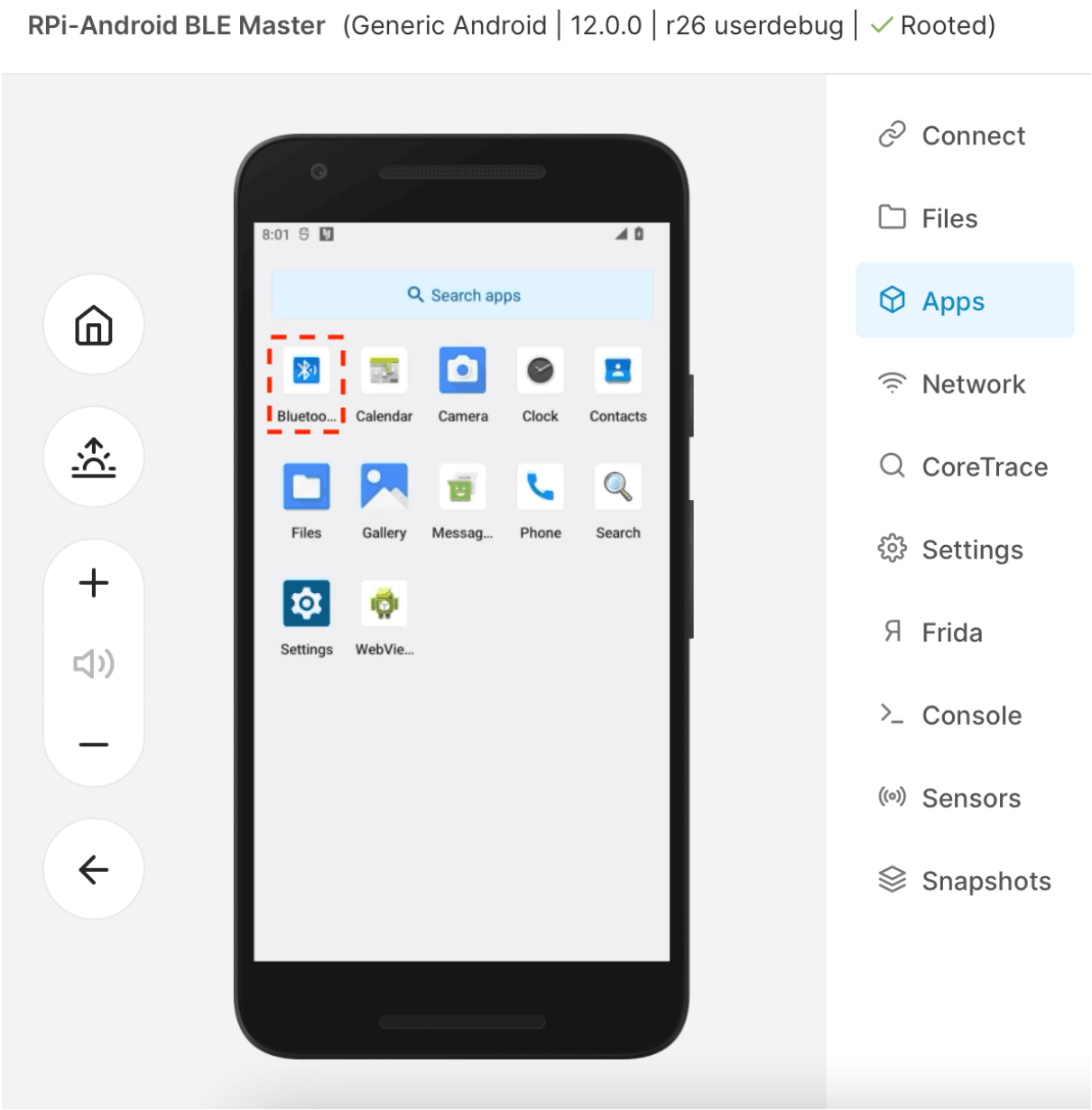
Figure 3-56: Create Raspberry Pi 4 instance



2. Download the Bluetooth LE Scanner APK to your local computer then install the APK through the Apps tab.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

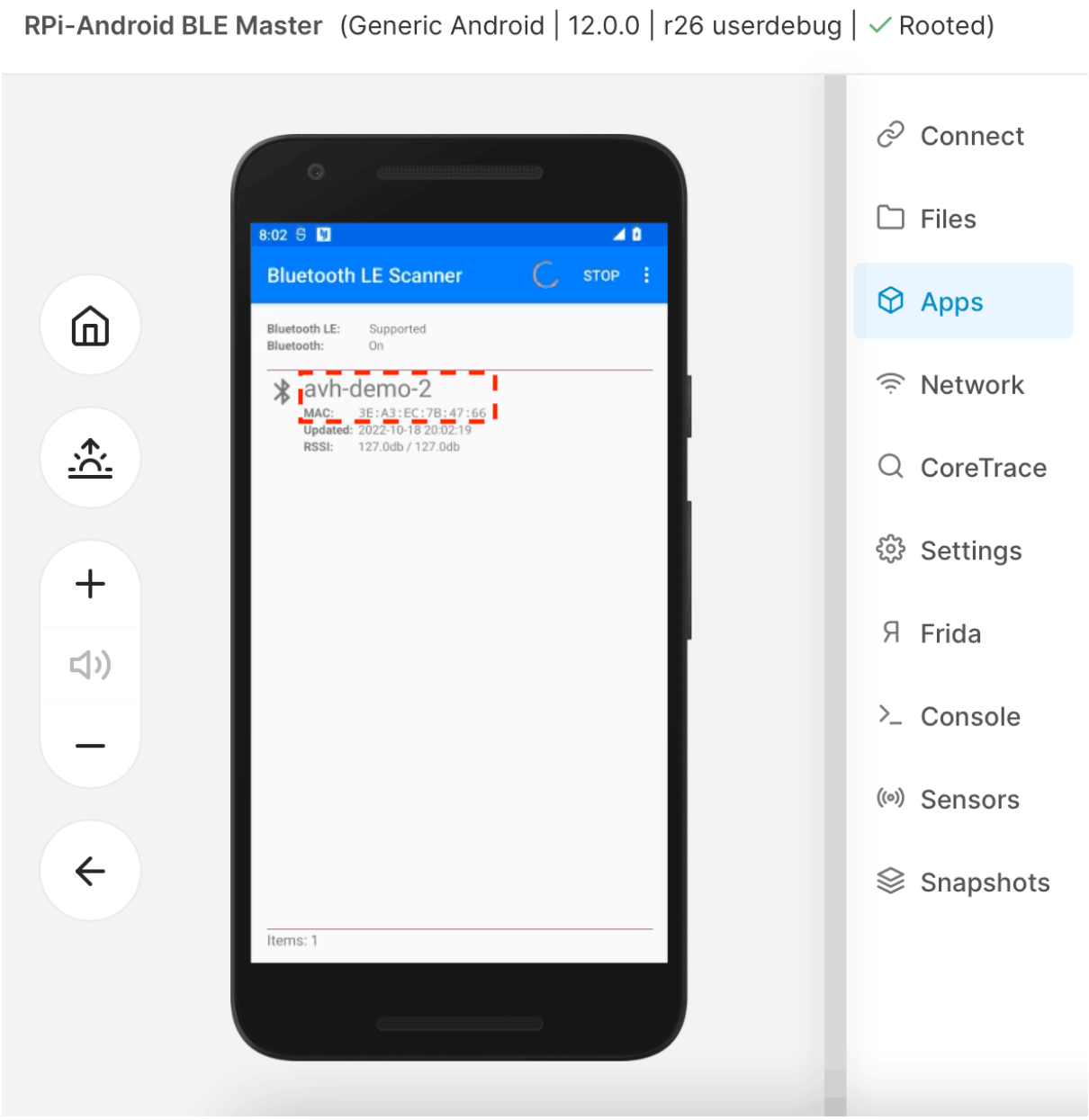
Figure 3-57: Install Bluetooth LE scanner



3. Open the Bluetooth LE Scanner app and click Scan. You should see `avh-demo-2` appear with the matching MAC address.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

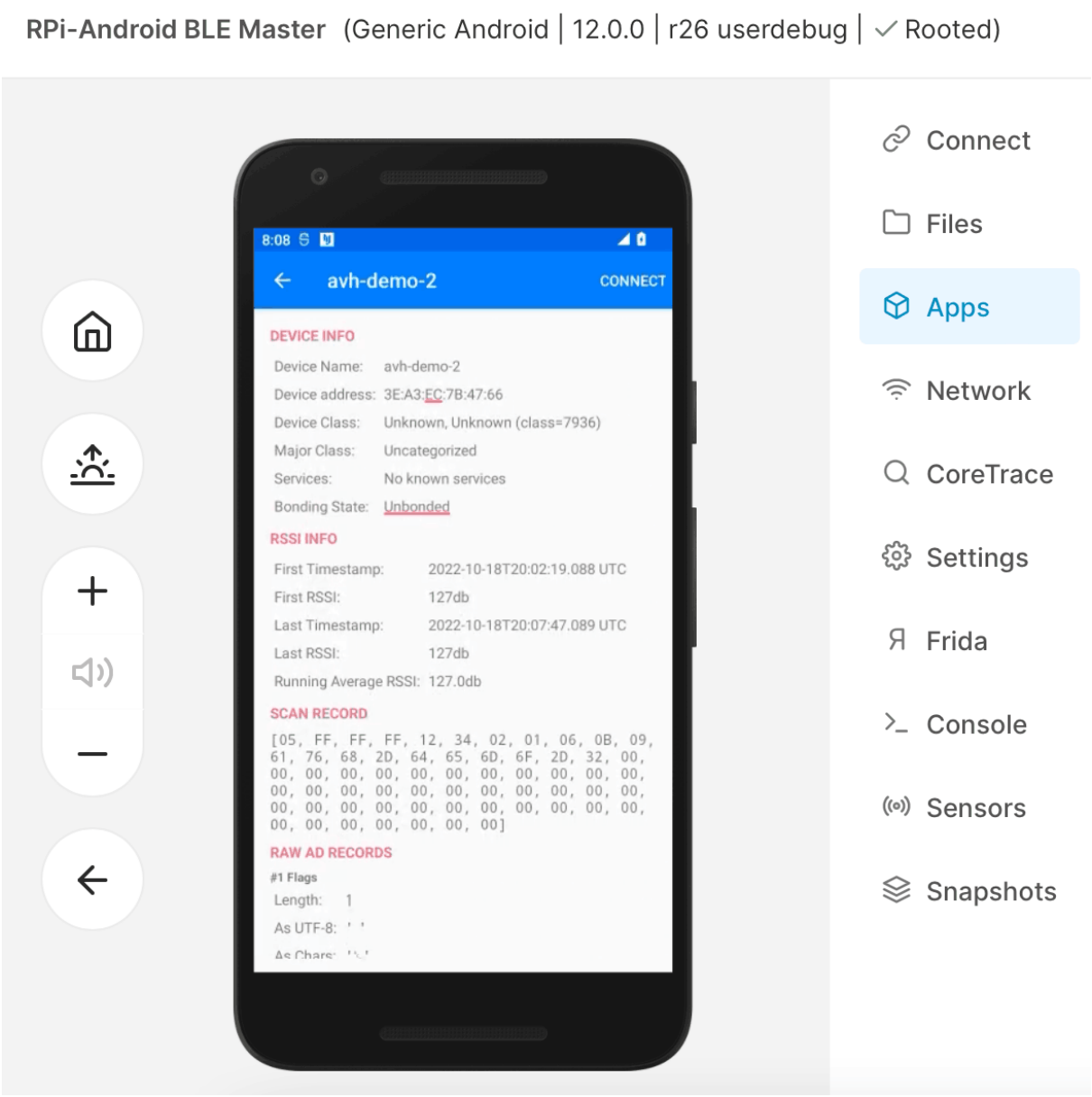
Figure 3-58: Scan for node



4. Click on avh-demo-2 to open up additional details. Scroll to the bottom of the page to see the manufacturer [ff, ff, 12, 34] we set previously.

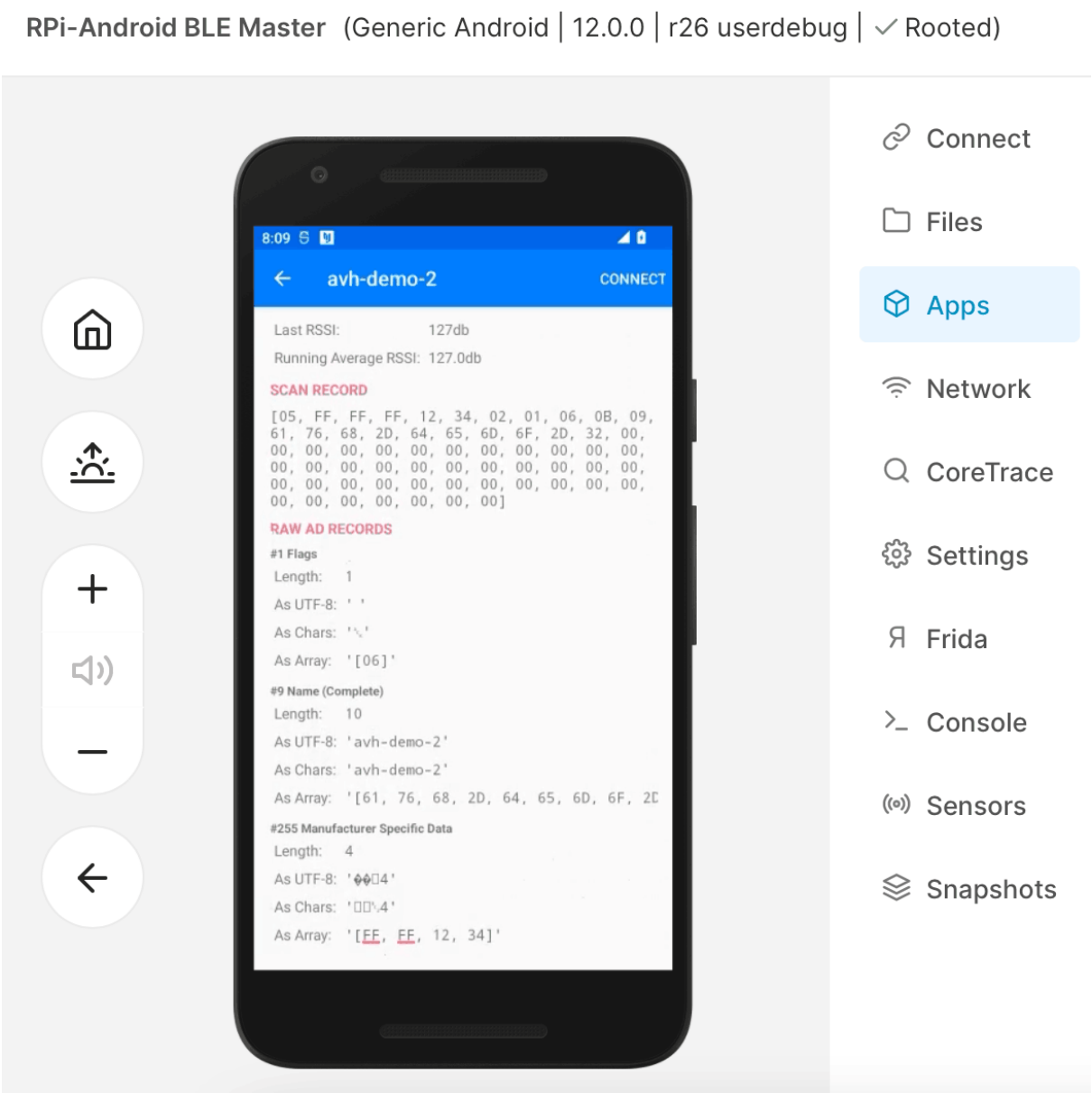
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-59: Additional details view



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

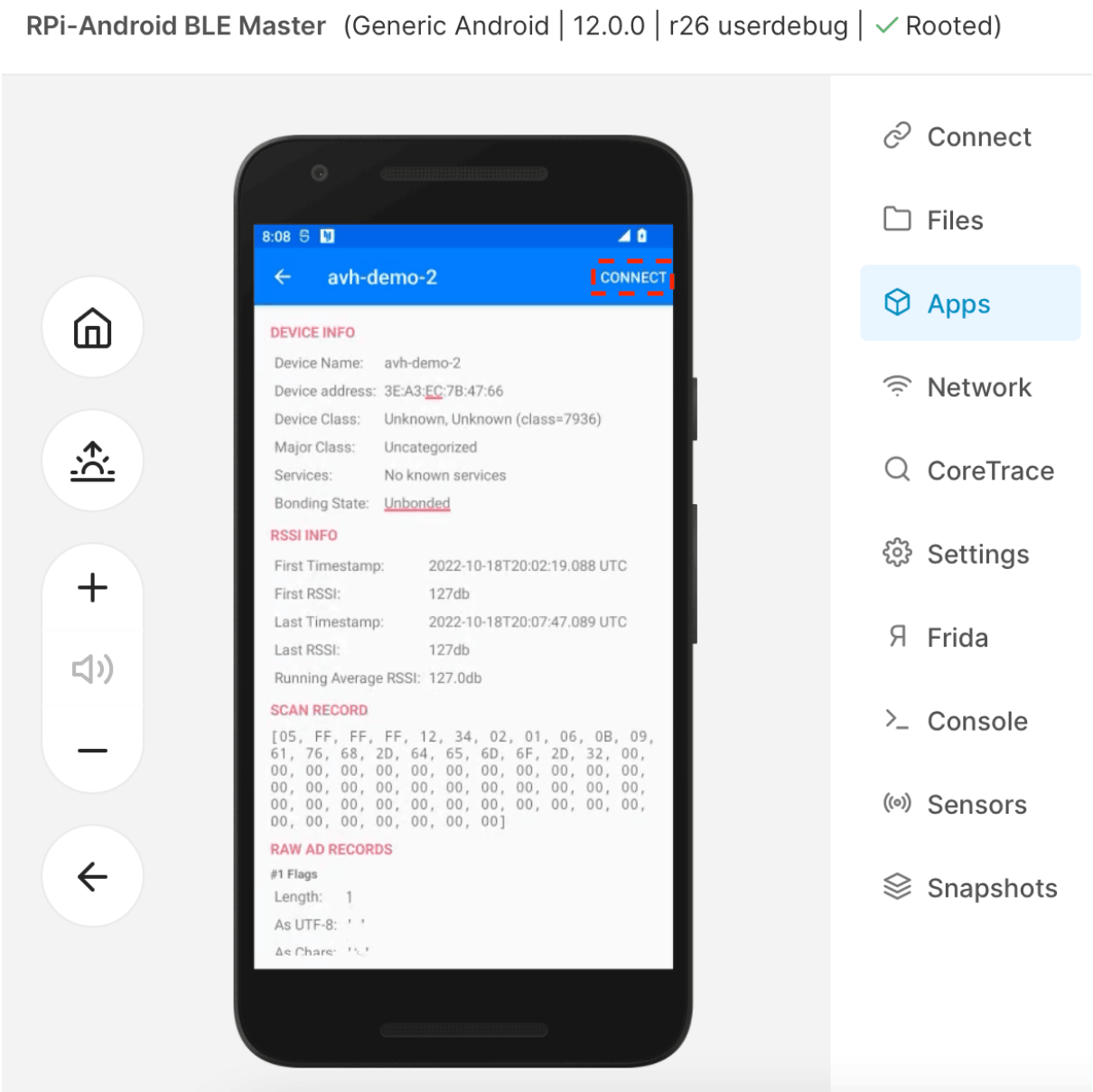
Figure 3-60: Additional details view



5. Click the CONNECT button in the app.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

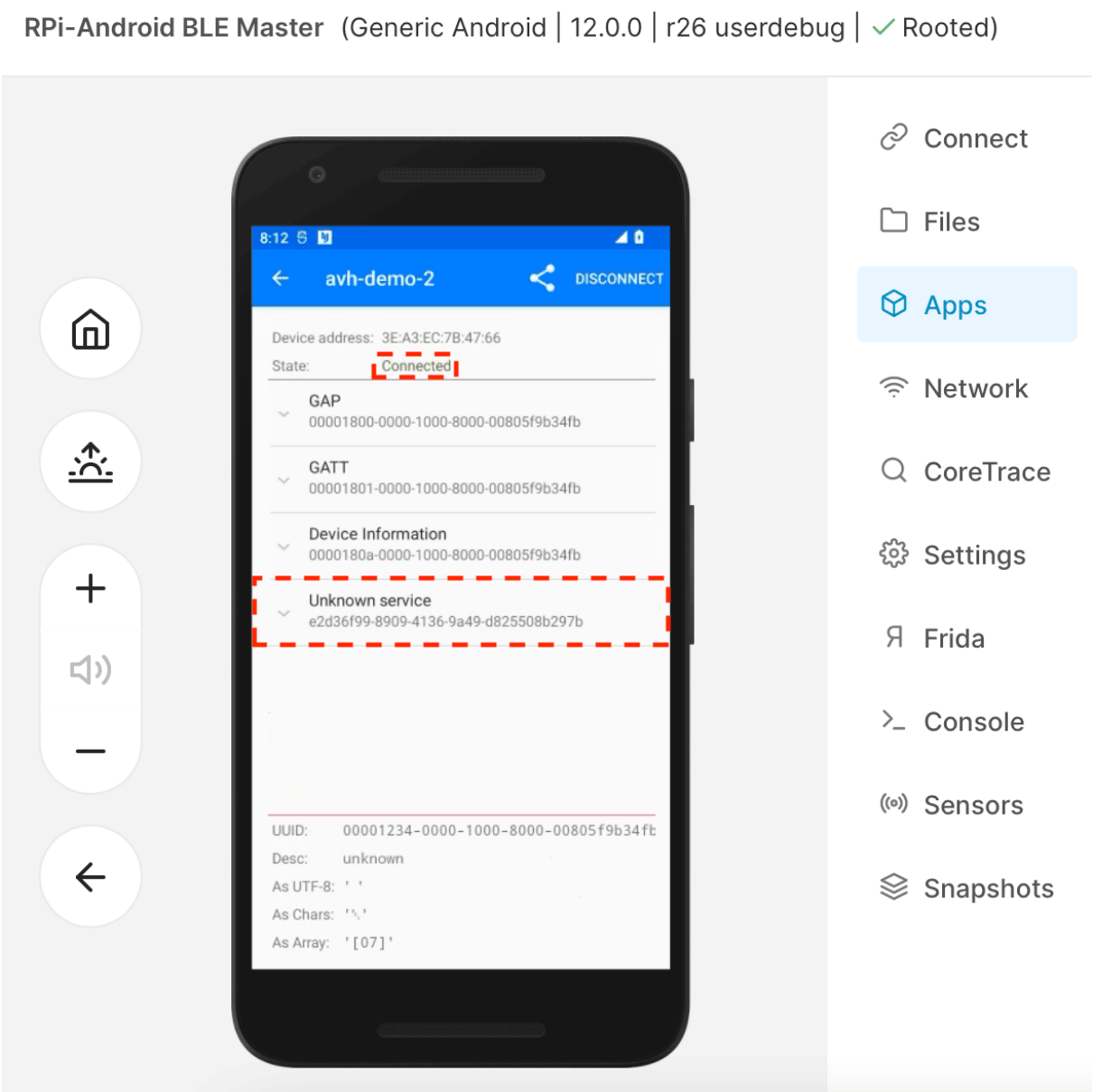
Figure 3-61: Connect to AVH demo



6. After the device connects successfully, you will see an `Unknown Service` with the ID matching what we set up. Expand the service.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

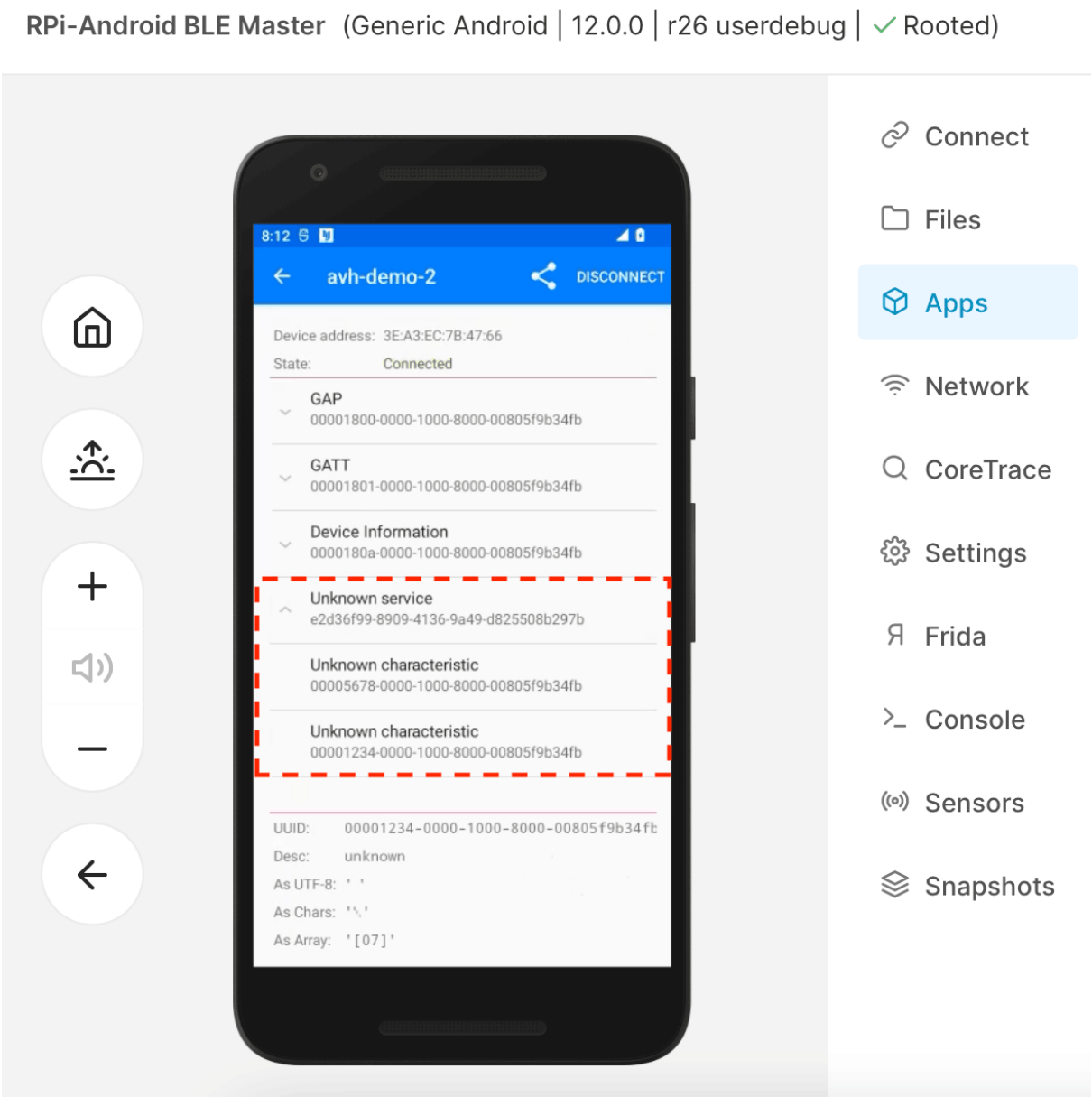
Figure 3-62: Unknown service screen



7. You will see the two custom characteristics `1234` and `5678` that we set up.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

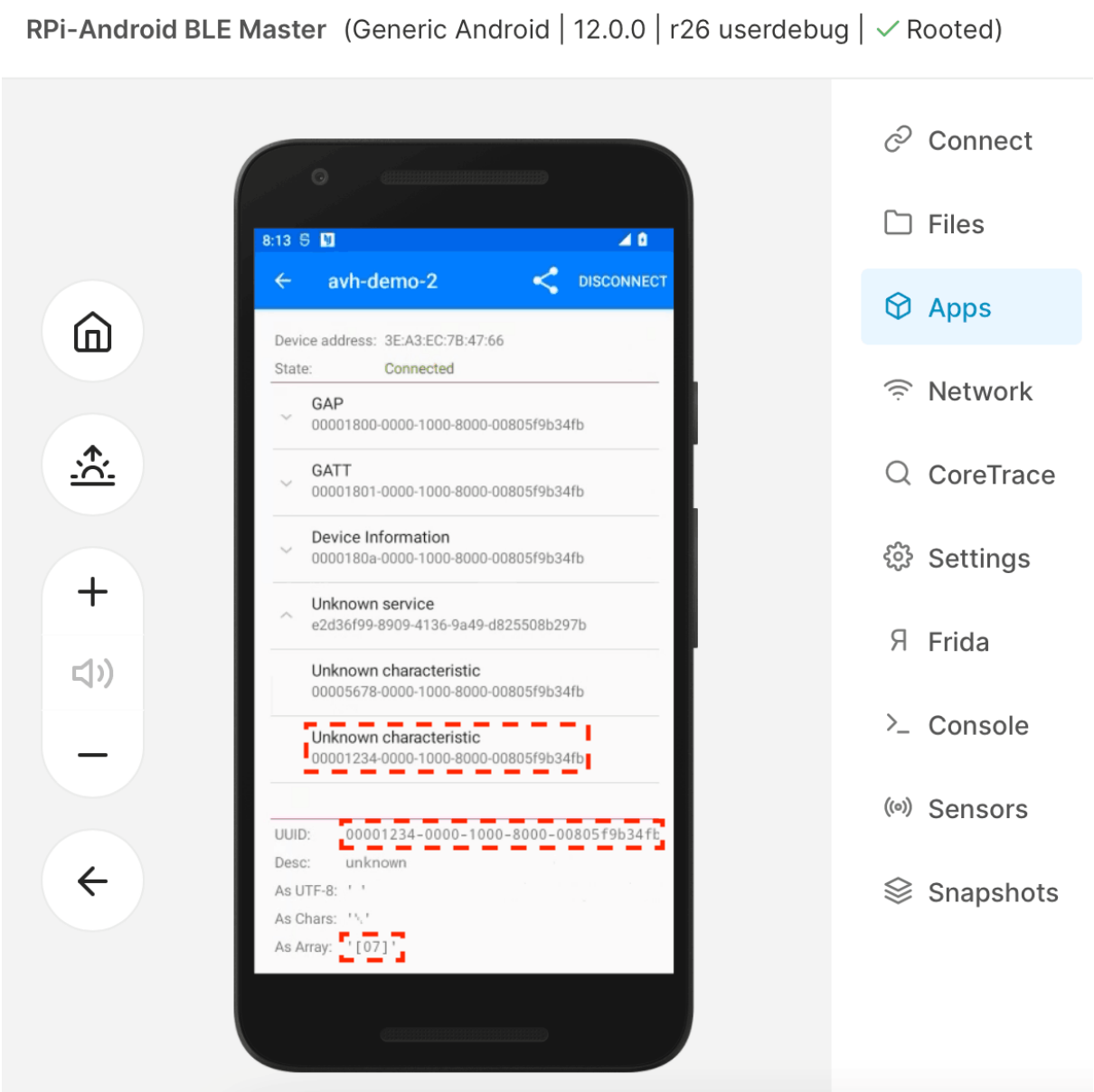
Figure 3-63: Custom characteristics



8. Click on the `1234` and the `5678` services to see the values we had set previously (`07` and `0d`, respectively).

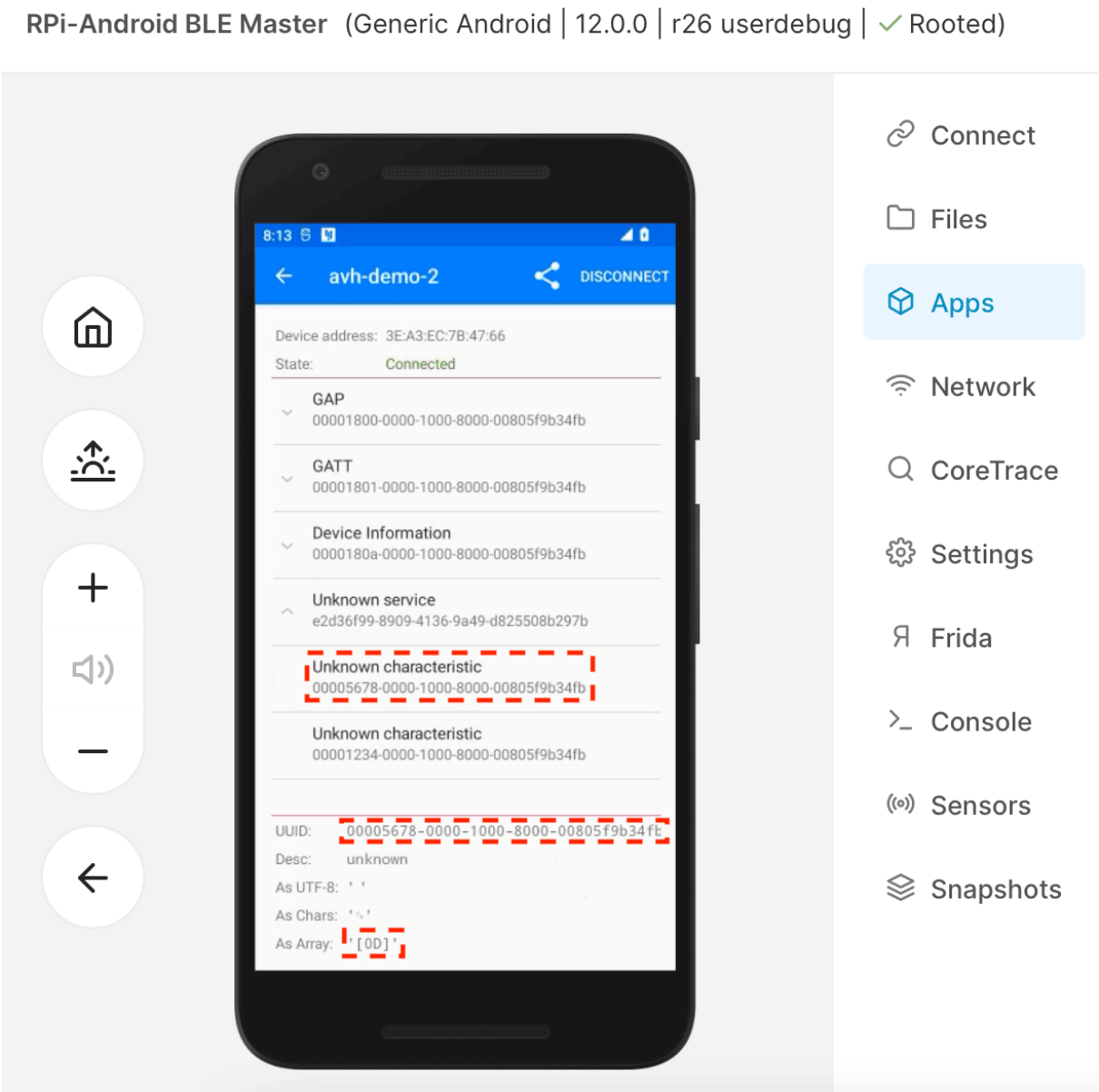
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-64: Custom characteristics values



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-65: Custom characteristics values



3.11 Detect Audio through the Microphone with the STM32U5 Board

AVH includes support for connecting your computer's microphone to virtual devices. This article will demonstrate how to detect audio using the virtual STM32U5 board.

- Build the ADF_AudioSoundDetector Firmware File

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

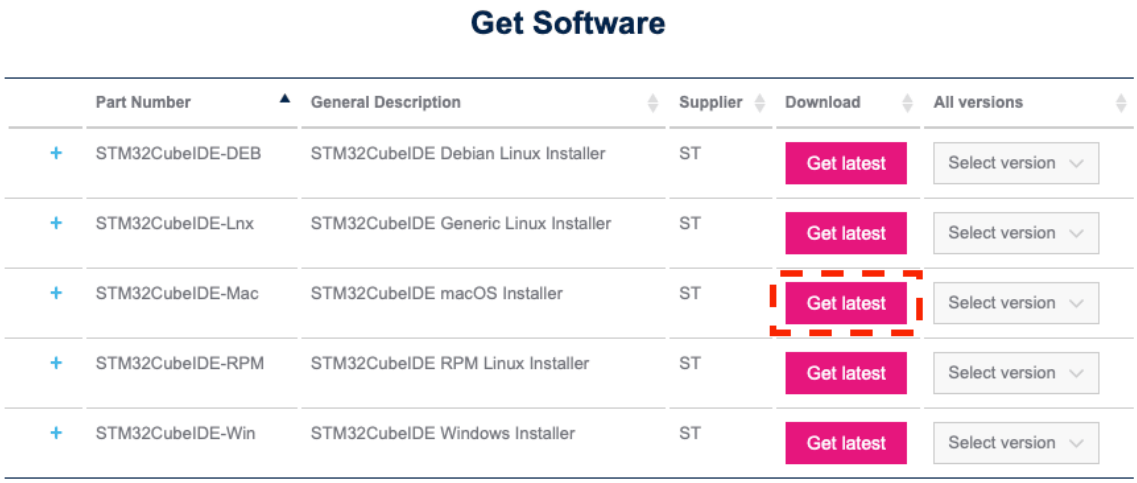
- Boot the Custom Firmware and Detect Audio

3.11.1 Build the ADF_AudioSoundDetector Firmware File

Follow this process to build the required `.elf` custom firmware files using the free STM32CubeIDE tool from STMicroelectronics.

1. Install the STM32CubeIDE tool. In our example, we will use the MacOS version.

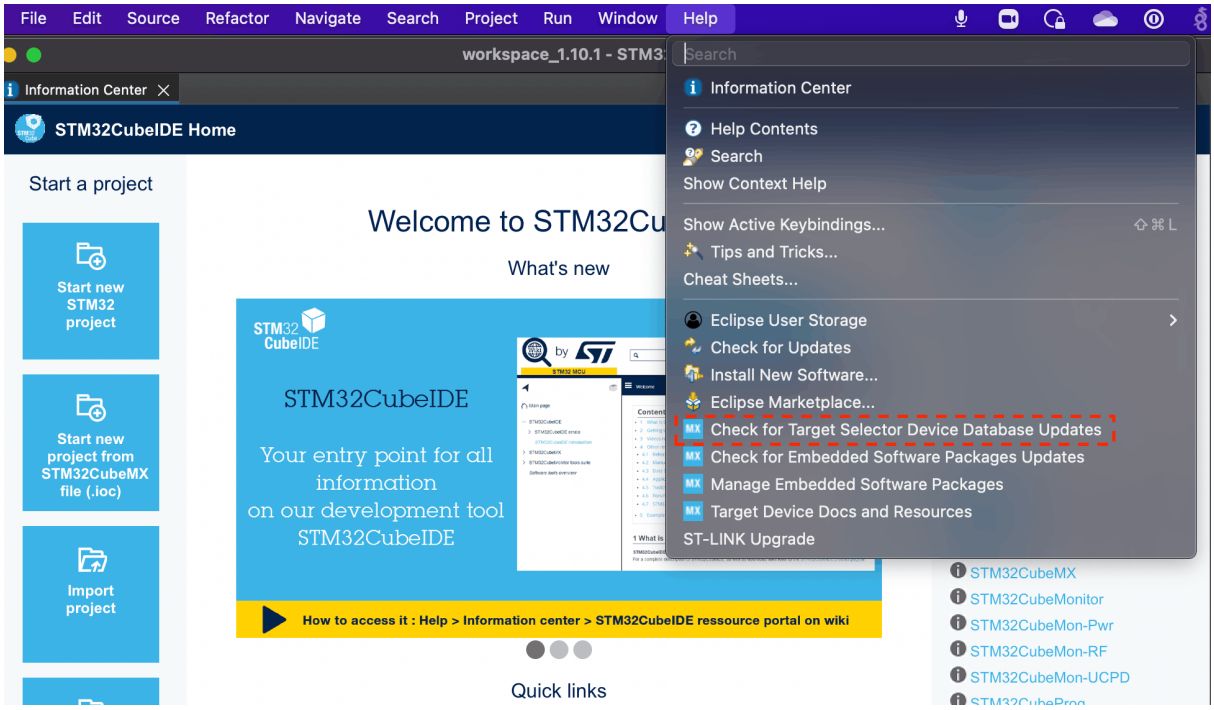
   Figure 3-66: Download CubeIDE



2. On the top menu, click Help then `Check for Target Selector Device Database Updates`.

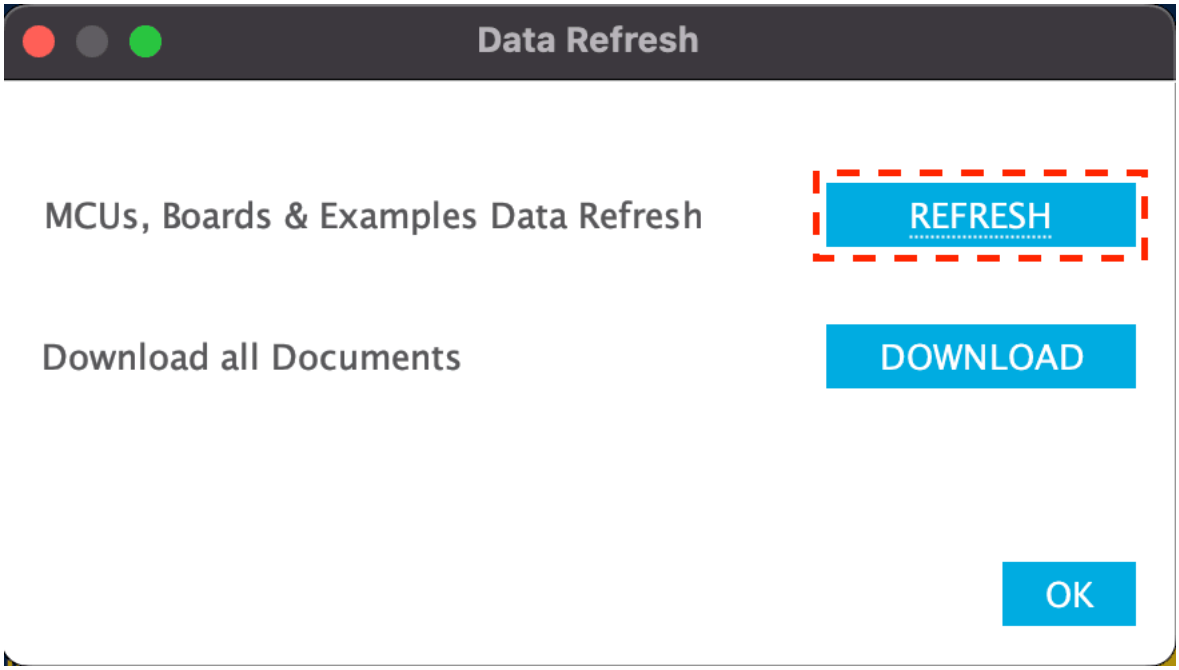
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-67: Check for updates



3. On the popup window click REFRESH. Your project files will update.

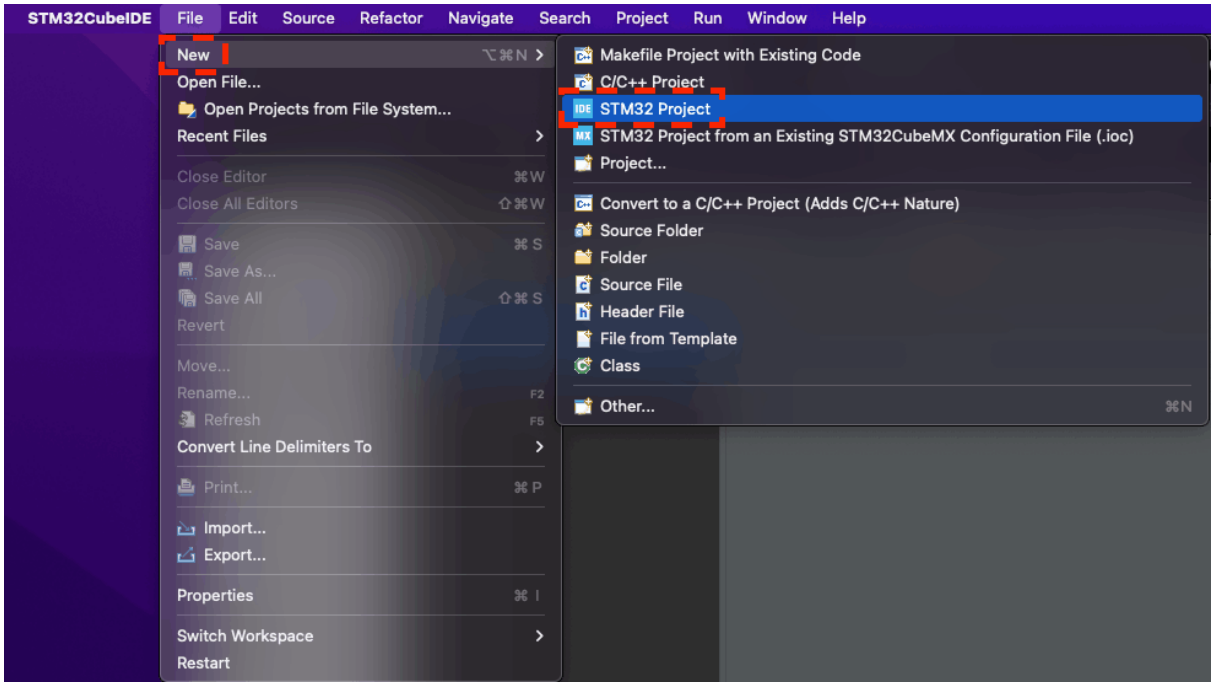
Figure 3-68: Refresh button



4. Create a new project by clicking New then STM32 Project from the File menu.

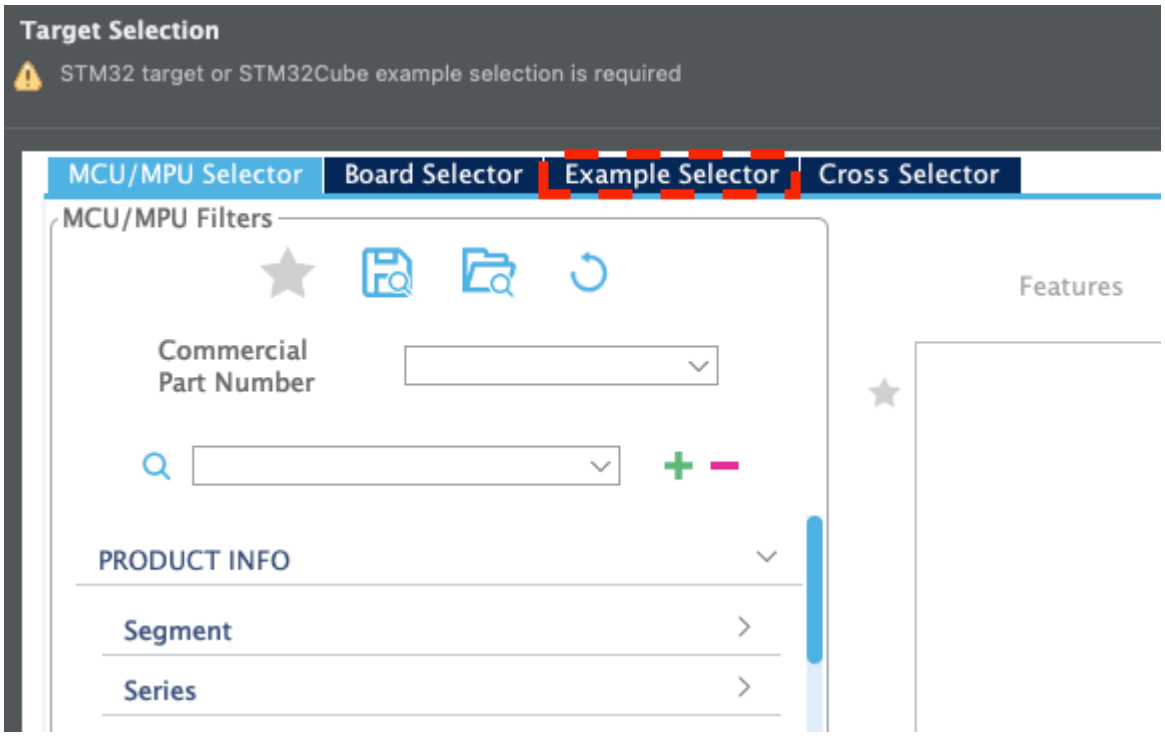
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-69: New project screen



5. Click on Example Selector.

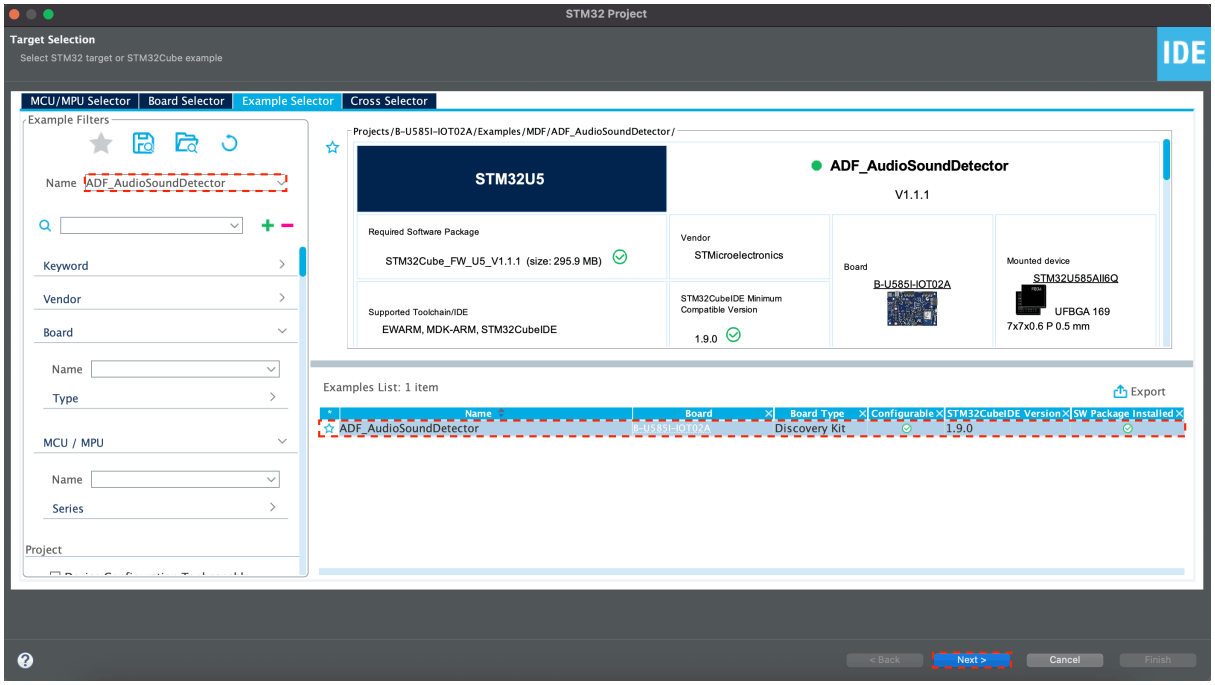
Figure 3-70: Target selection



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6. Set the Name to ADF_AudioSoundDetector, click on the ADF_AudioSoundDetector example to the right, and click Next.

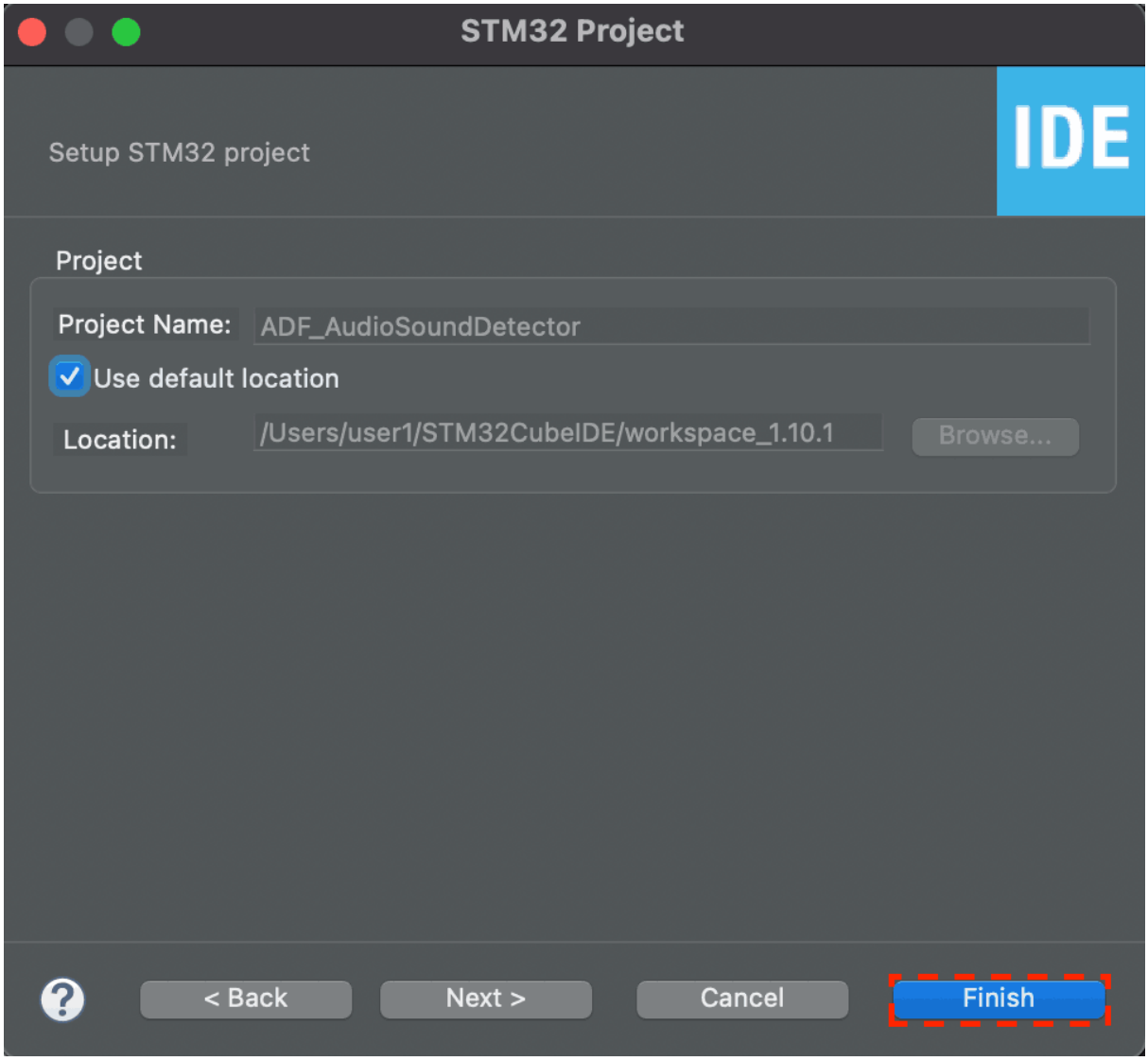
Figure 3-71: ADF_AudioSoundDetector example



7. Click Finish to create the project.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

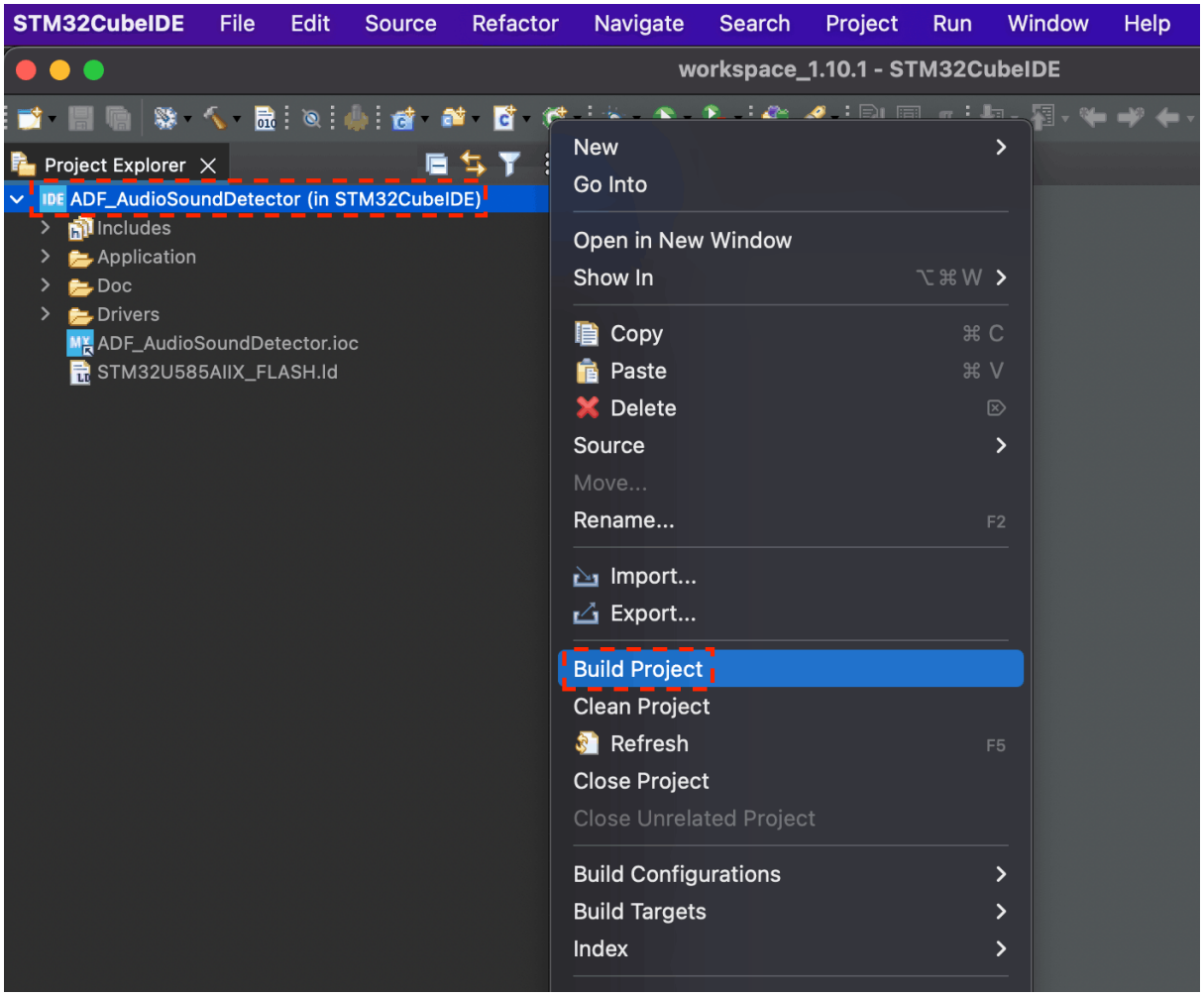
Figure 3-72: Finish project setup



8. In the Project Explorer window, right-click on the ADF_AudioSoundDetector project and choose Build Project.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

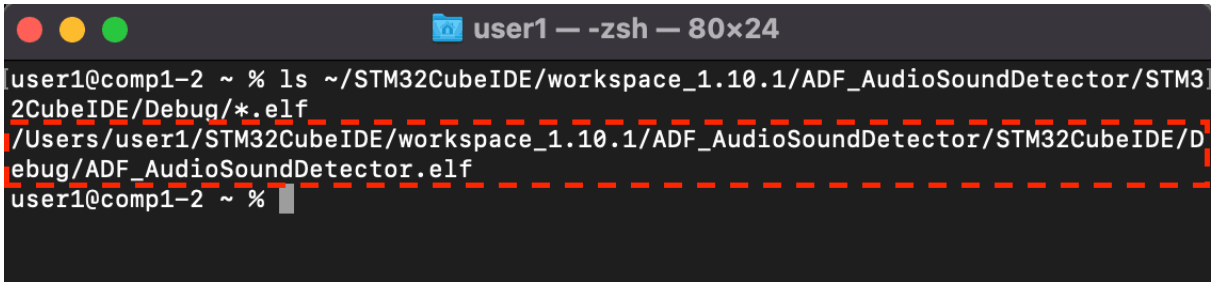
Figure 3-73: Build project



9. The default path for your firmware file is: `~/STM32CubeIDE/workspace_1.0.1/ADF_AudioSoundDetector/STM32CubeIDE/Debug/ADF_AudioSoundDetector.elf`.

```
ls ~/STM32CubeIDE/workspace_1.10.1/ADF_AudioSoundDetector/STM32CubeIDE/Debug/*.el
```

Figure 3-74: .elf File



10. Proceed to the next section to boot the custom firmware file.

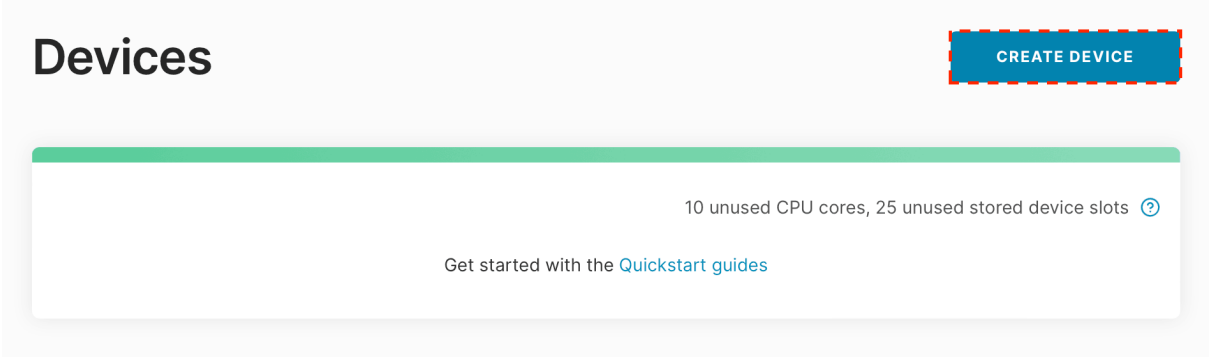
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.11.2 Boot the Custom Firmware and Detect Audio

Follow this process to boot the custom firmware and detect audio:

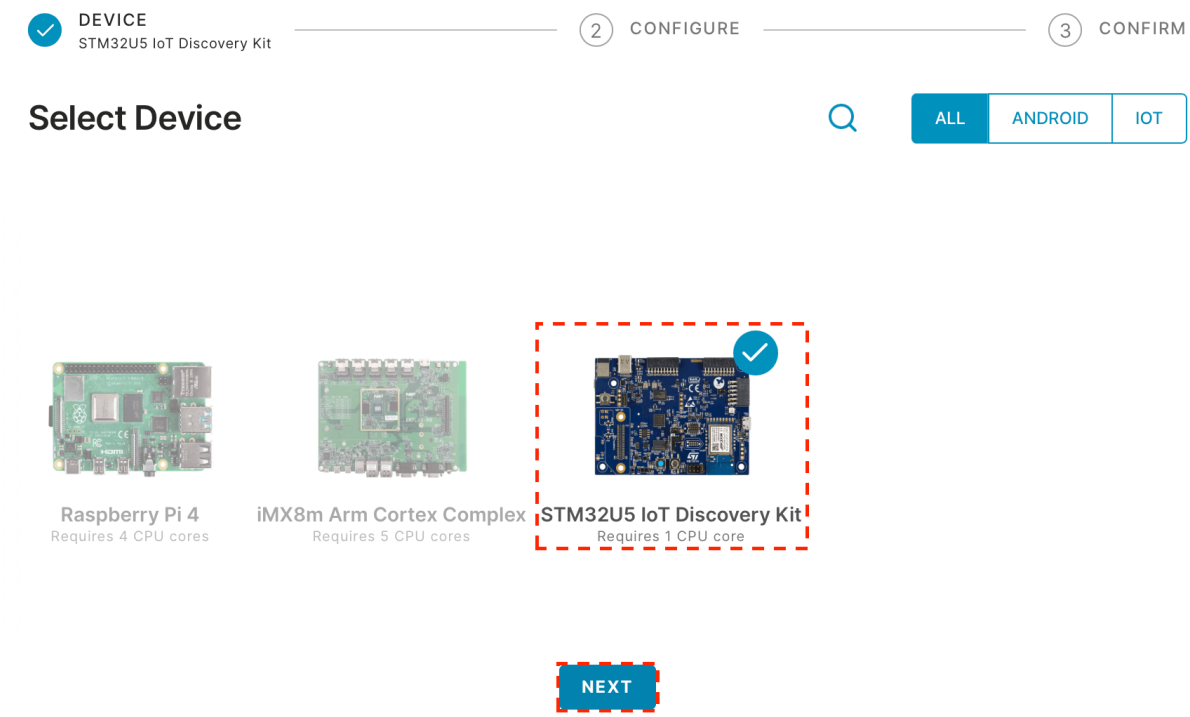
1. Log in to the AVH platform and click CREATE DEVICE.

Figure 3-75: Create device



2. Select the STM32U5 device and click NEXT.

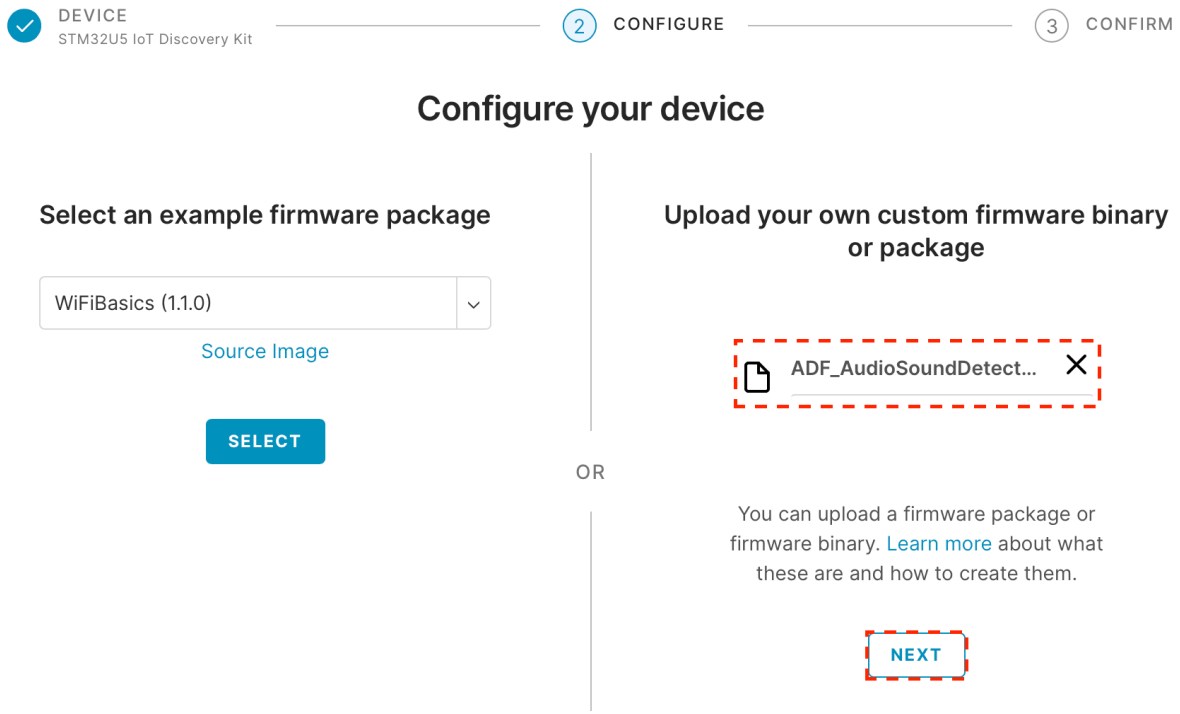
Figure 3-76: Select device



3. Upload your ADF_AudioSoundDetector.elf custom firmware file click NEXT, then choose CREATE DEVICE.

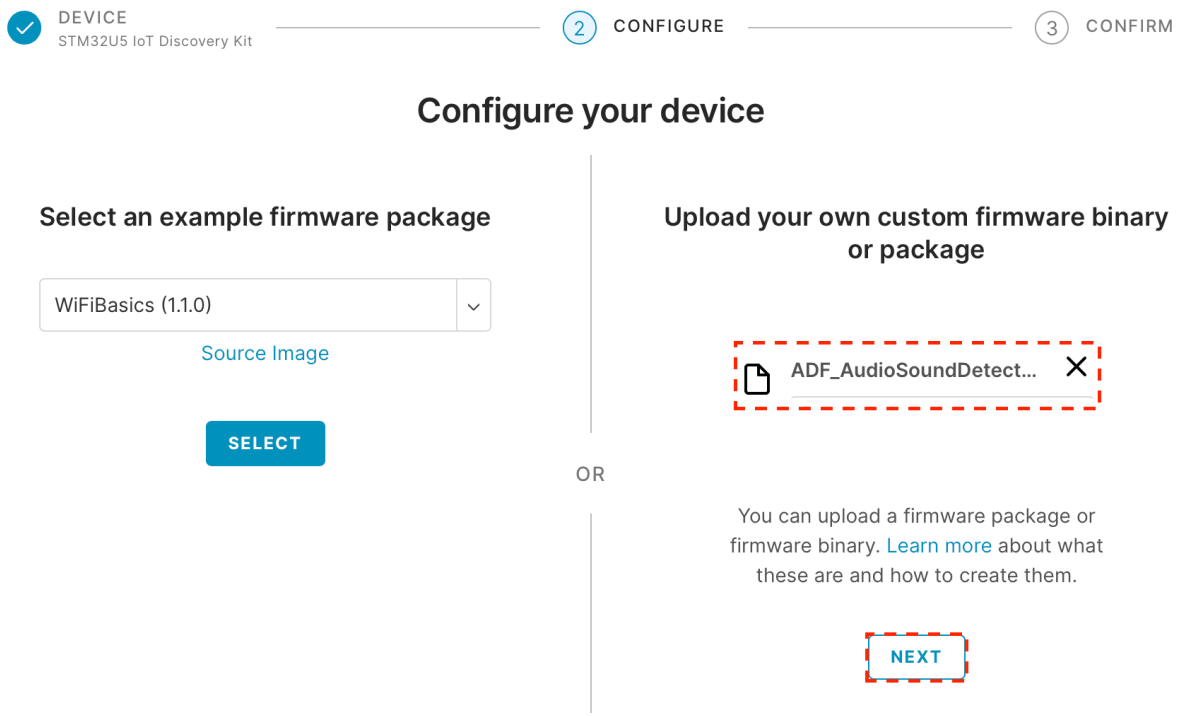
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-77: Upload firmware



4. After your device boots, open the Sensors tab and enable the microphone.

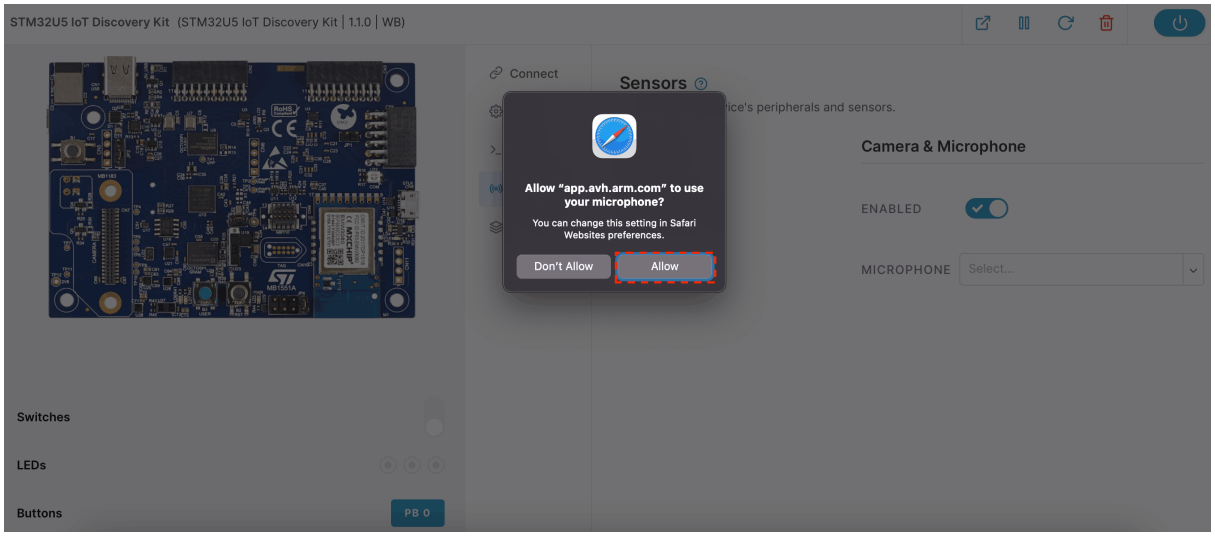
Figure 3-78: Enable microphone



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

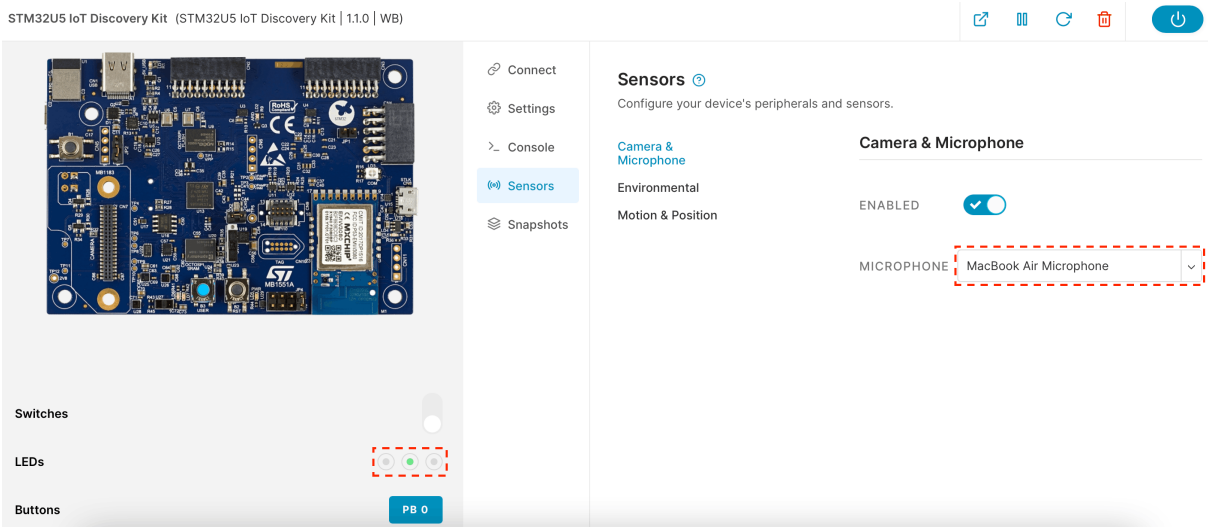
5. Confirm on your browser that you would like to share the microphone.

Figure 3-79: Share microphone with browser



6. The device should begin automatically detecting sound. You will see LED #2 lighting green to indicate audio is detected.

Figure 3-80: Audio detection



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4. Custom Firmware

The following articles are available:

- Firmware
- Storage Files

4.1 Firmware

Information about firmware on AVH:

- Understanding Firmware on AVH
- Package Ubuntu Server Firmware for AVH
- Upload custom firmware using the UI

4.1.1 Understanding Firmware on AVH

Learn how firmwares work on AVH.

4.1.1.1 Overview

AVH supports two types of firmware images.

1. A zip file containing multiple images such as a disk image, kernel or binary firmware etc
2. A raw binary file, ELF executable or kernel loaded into RAM

4.1.1.2 Zip package

The format of the zip package looks like this -

Required files:

- `Info.plist` (meta information)
- If what you are booting is Linux -
  - `kernel` (a Linux kernel in the `Image` format)
  - `devicetree` (the device tree for Linux in binary `.dtb` format)
- Or if it's a raw firmware (e.g. used by Cortex-M machines)
  - `firmware` (Binary, ELF executable file, ZIP archive with load instructions)

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.1.1.3 Storage

Most devices have flash specific files that are required. See the relevant storage files page for that specific device.

- Storage Files for STM32U5 IoT Discovery Kit
- Storage Files for iMX8m Arm Cortex Complex
- Storage Files for Raspberry Pi 4

4.1.1.4 What does the Info.plist look like?

An `Info.plist` file containing the version, type, build, unique identifier and device identifier.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
dict>
<key>Type</key>
<string>iot</string>
<key>UniqueIdentifier</key>
<string>WifiBasics</string>
<key>DeviceIdentifier</key>
<string>stm32u5-b-u585i-iot02a</string>
<key>Version</key>
<string>WifiBasics</string>
<key>Build</key>
<string>WB</string>
</dict>
</plist>
```

4.1.1.5 Formats of raw firmware

There are three supported formats for executable firmware:

- ELF executable: 32-bit ELF program files (fully linked, no relocation required) typically produced by IoT vendor tools: recommended - they contain all information needed to load the program, even if it has multiple memory ranges;
- binary: loaded at a predefined location in memory (typically start of Flash for devices with built-in executable Flash, otherwise start of RAM);
- ZIP archive: an archive containing a few binary files, and a file called `LOAD.TXT` that instructs the firmware loader to distribute them to different load locations in memory; a `LOAD.TXT` file could look like this:

```
load:0x00000000 name:bl2.bin
load:0x01000000 name:tfm_s_ns_signed.bin
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

This `LOAD.TXT` file would cause file `bl2.bin` in the same ZIP archive to be loaded at address `0x00000000`, and `tfm_s_ns_signed.bin` at `0x01000000`. This format is required when all that's available is binary files, and there's more than one of them.

4.1.1.6 Sample Firmware Packaging Script for Raspberry Pi

Sample firmware packaging script for Raspberry Pi:

```
#!/bin/bash
set -e
[ -d pi ] || mkdir pi
cd pi
# Grab the raspberry pi firmware
[ -f 2022-01-28-raspios-bullseye-arm64.zip ] || wget https://
downloads.raspberrypi.org/raspios_arm64/images/raspios_arm64-2022-01-28/2022-01-28-
raspios-bullseye-arm64.zip
rm -rf {nand,devicetree,kernel,Info.plist,boot,rootfs}
unzip 2022-01-28-raspios-bullseye-arm64.zip
mv 2022-01-28-raspios-bullseye-arm64.img nand
# Mount the firmware image and extract the kernel and device tree
LO="$(losetup -f)"
mkdir boot
losetup -P "${LO}" nand
mount "${LO}p1" boot
# Enable ssh
touch boot/ssh
cp boot/bcm2711-rpi-4-b.dtb devicetree
zcat boot/kernel8.img > kernel
umount boot
rm -rf boot
mkdir rootfs
mount "${LO}p2" rootfs
# Don't run dhcpcd on docker interfaces
echo 'denyinterfaces veth*' >> rootfs/etc/dhcpcd.conf
umount rootfs
rm -rf rootfs
losetup -d "${LO}"
# create the Info plist that describes the model image
cat << EOF > Info.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Type</key>
    <string>iot</string>
    <key>UniqueIdentifier</key>
    <string>Raspberry Pi OS Desktop</string>
    <key>DeviceIdentifier</key>
    <string>rpi4b</string>
    <key>Version</key>
    <string>11.2.0</string>
    <key>Build</key>
    <string>desktop</string>
</dict>
</plist>
EOF
# zip image and its ready for use
zip -r ../rpi4b-11.2-desktop.zip Info.plist nand devicetree kernel ramdisk.img
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.1.2 Package Ubuntu Server Firmware for AVH

Follow this guide to package, install, and run custom Ubuntu Server firmware on a virtual Raspberry Pi 4 board.

AVH supports running custom Linux firmware packages.

This guide outlines the contents of a valid Ubuntu firmware package, provides a script for building the Ubuntu Server firmware, and walks through the installation on a virtual Raspberry Pi 4 board.

This packaging process follows the Understanding Firmware on AVH knowledge base article and uses the Raspberry Pi image of Ubuntu Server available directly from Ubuntu.

Please also refer to our companion guide Package Ubuntu Desktop Firmware for AVH.

4.1.2.1 Firmware Package Contents

A proper Ubuntu Server firmware package file contains the following:

- `Info.plist` - the version, type, build, unique identifier, and device identifier
- `nand` - the preinstalled Ubuntu Server arm64 image file
- `devicetree` - hardware components data for the Linux kernel
- `kernel` - the Linux kernel file
- `ramdisk.img` (optional) - the initrd root file system image

Although specifying a `ramdisk.img` is generally optional, we need to add a reboot command in this case because of how this virtual device handles the first pass.

4.1.2.2 Firmware Packaging Script

The following script creates a custom firmware package in your working directory:

```
#!/bin/bash
set -e
mkdir rpi_ubuntu_server_firmware
cd rpi_ubuntu_server_firmware
# Download and extract the "Ubuntu 22.04.1 Server for RPi" image
wget https://cdimage.ubuntu.com/releases/22.04.1/release/ubuntu-22.04.1-
preinstalled-server-arm64+raspi.img.xz
xz -dv ubuntu-22.04.1-preinstalled-server-arm64+raspi.img.xz
mv ubuntu-22.04.1-preinstalled-server-arm64+raspi.img nand
# Attach the image file
LO="$(losetup -f)"
losetup -P "${LO}" nand
# Mount partition 1 to directory boot
mkdir boot
mount "${LO}p1" boot
# Copy the rpi4b devicetree
cp boot/bcm2711-rpi-4-b.dtb devicetree
# Extract the Linux kernel
zcat boot/vmlinuz > kernel
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
# Extract initrd
lz4 -d boot/initrd.img initrd.cpio
umount boot
rm -r boot/
losetup -d "${LO}"
mkdir ramdisk
cd ramdisk
# Extract initrd
cat ../initrd.cpio | cpio -idm
rm ../initrd.cpio
# Add reboot conditional to init before matching string
sed -i '/Move virtual filesystems over to the real filesystem/i \
if /bin/grep init_resize /proc/cmdline; then\
/bin/reboot -f\
fi\
' init
# Create ramdisk.img from directory
find . | cpio -o -H newc -R root:root | lz4 -l > ../ramdisk.img
cd ..
rm -r ramdisk/
# Create the Info.plist file
cat << EOF > Info.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Type</key>
<string>iot</string>
<key>UniqueIdentifier</key>
<string>Ubuntu Server on RPi</string>
<key>DeviceIdentifier</key>
<string>rpi4b</string>
<key>Version</key>
<string>22.04.1</string>
<key>Build</key>
<string>Ubuntu Server</string>
</dict>
</plist>
EOF
zip -rm ../rpi4b-ubuntu-server.zip Info.plist nand devicetree kernel ramdisk.img
cd ..
rm -r rpi_ubuntu_server_firmware/
```

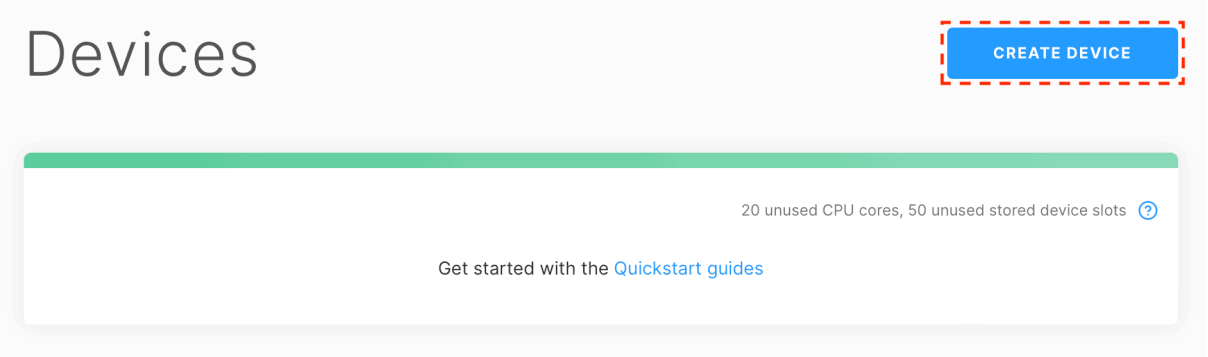
4.1.2.3 Install the Package

To install the package:

1. Run the above shell script on your local Linux environment to create the firmware package `rpi4b-ubuntu-server.zip`.
2. On the AVH web interface, click CREATE DEVICE.

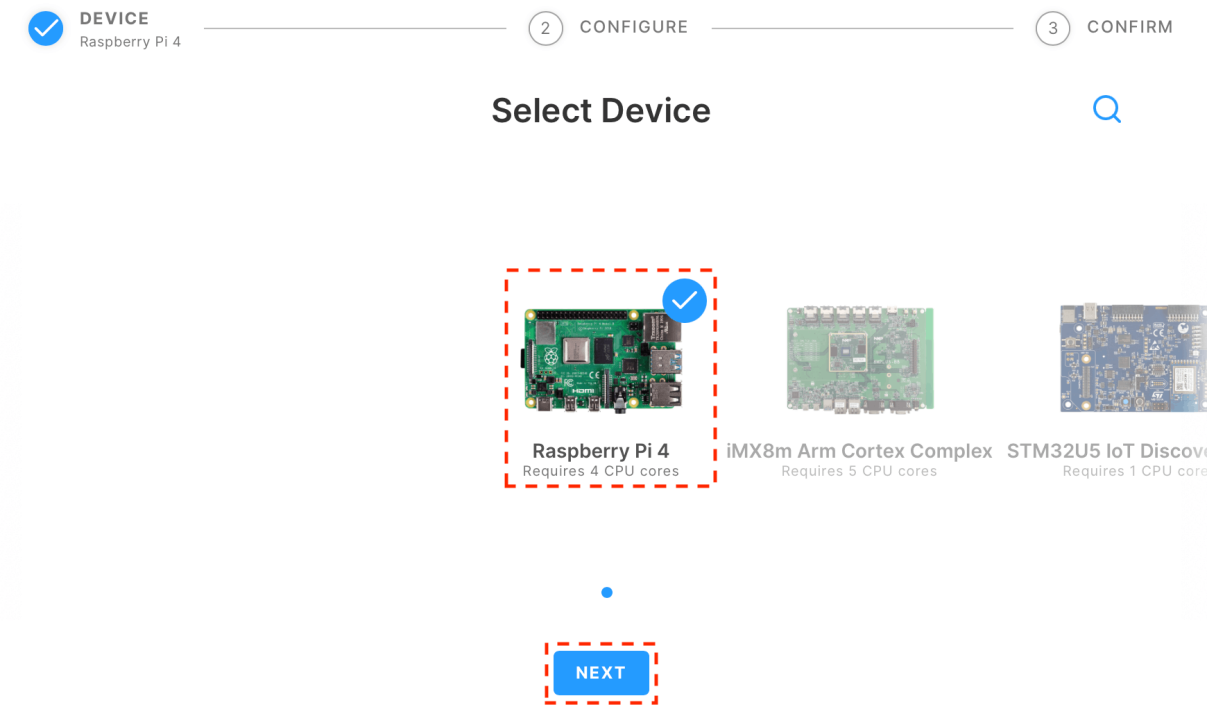
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-1: Create device



3. In Step 1, choose the Raspberry Pi 4 and click NEXT.

Figure 4-2: Select device



4. In Step 2, upload the custom firmware package. When the process is complete, click NEXT.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-3: Upload the custom firmware package

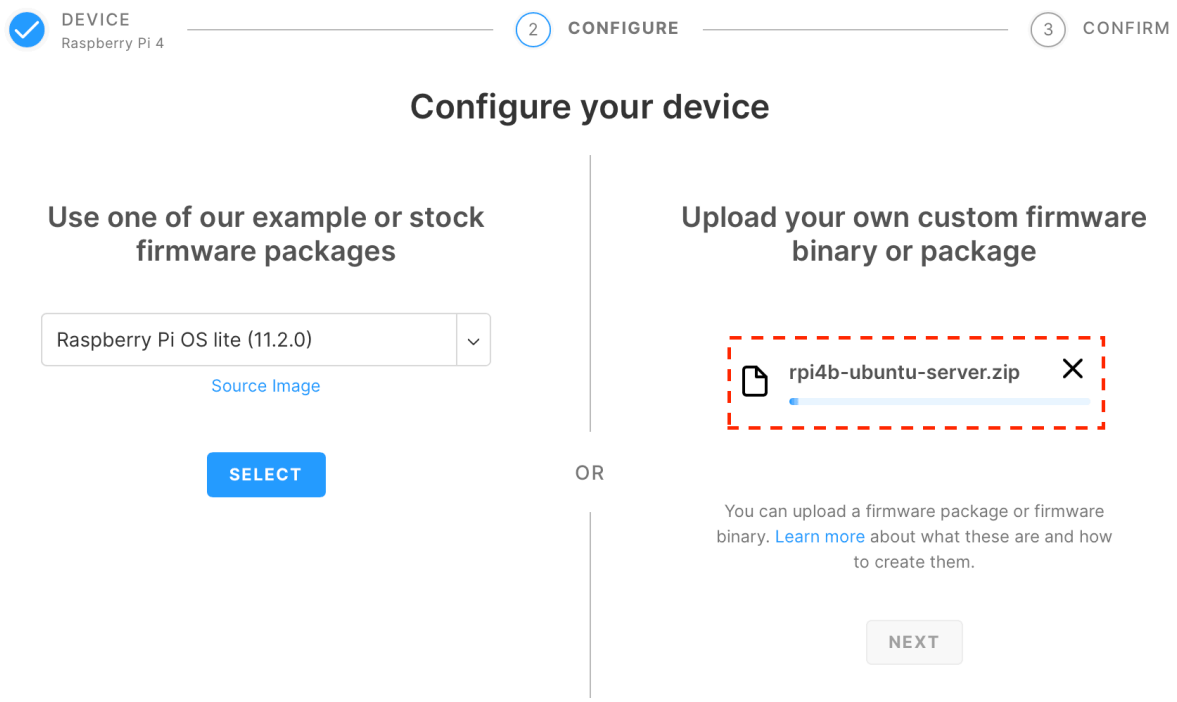
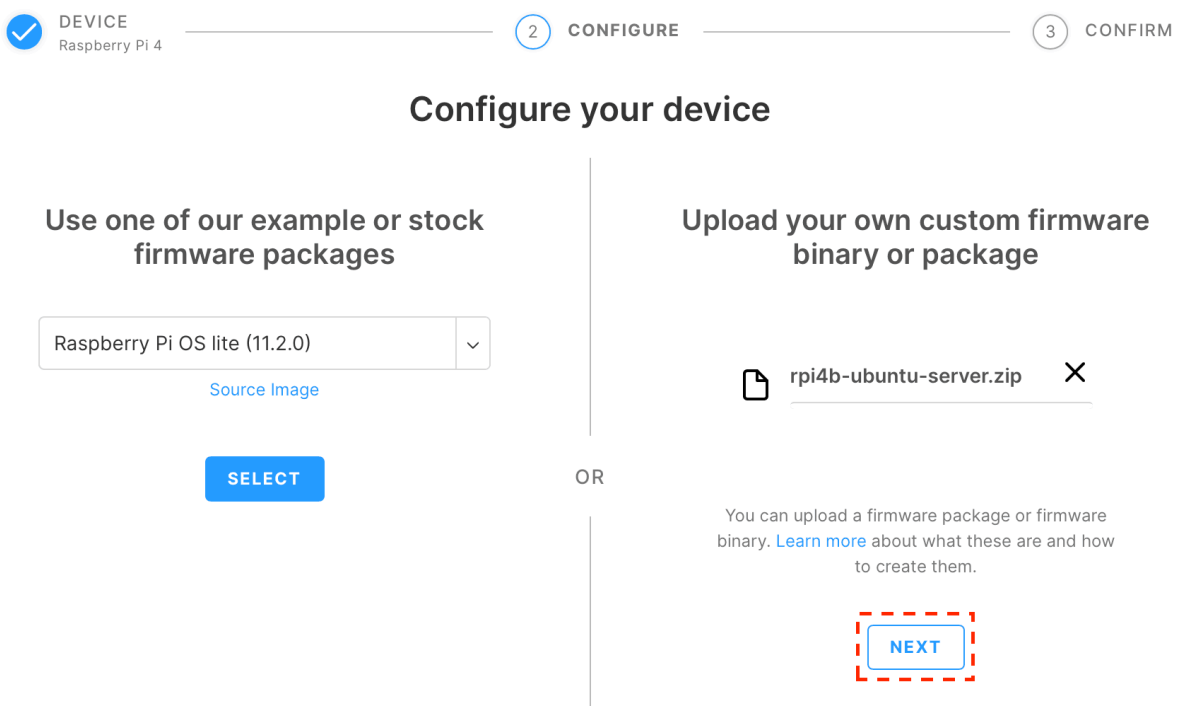


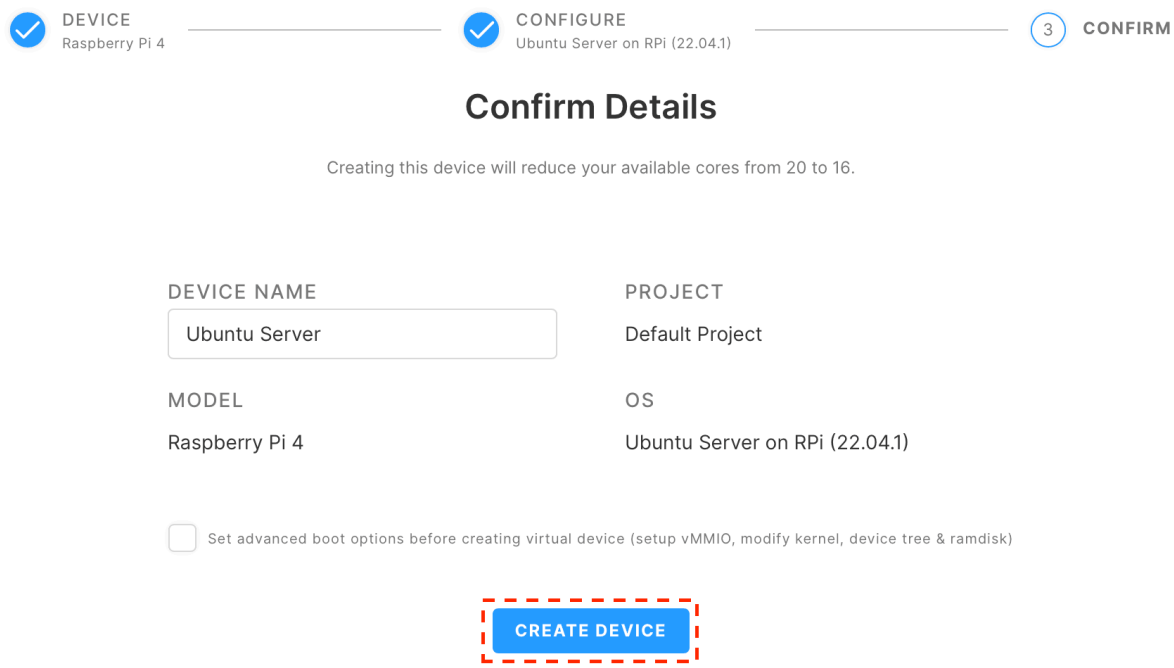
Figure 4-4: Click Next



5. Create the device without enabling advanced boot options.

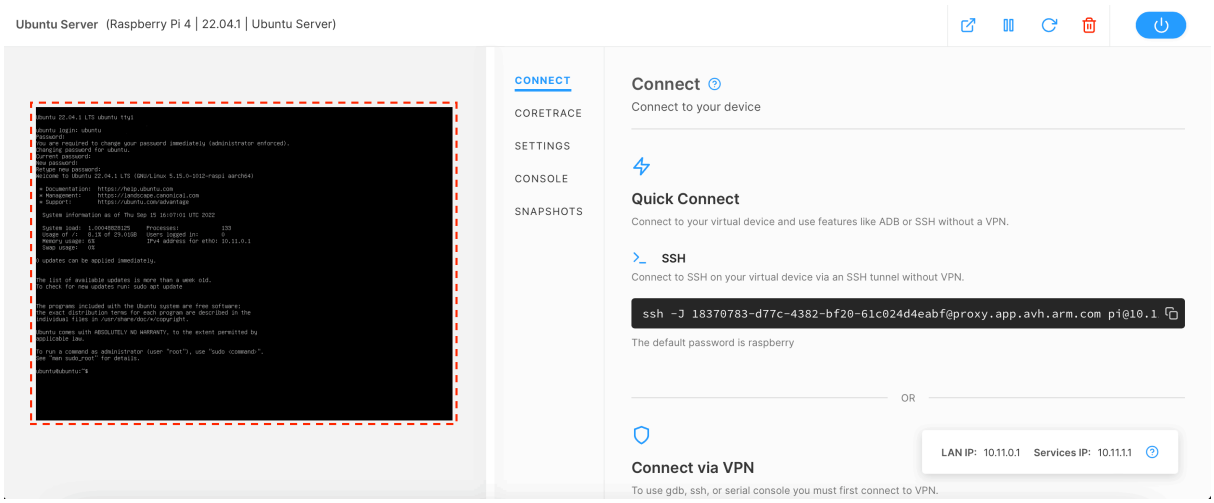
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-5: Create device



6. The virtual board will boot to the Ubuntu Server login screen. Login with the default credentials (`ubuntu/ubuntu`) and change your admin password.

Figure 4-6: Login with the default credentials



Note

Since we are using a custom configuration, you will need to adjust the SSH commands under the CONNECT tab to use `ubuntu` as the username instead of the default (`pi`).

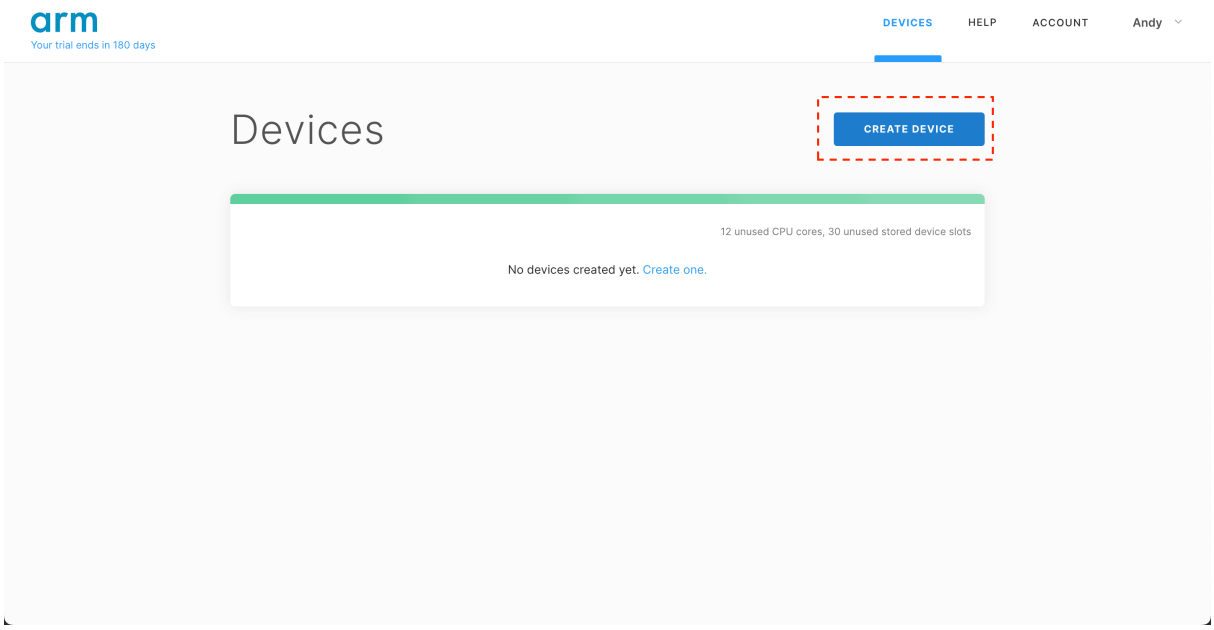
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.1.3 Upload custom firmware using the UI

This tutorial will demonstrate to you how to upload firmware to a new device.

1. Log in to AVH with your Arm user account. Log in to https://avh.arm.com/ If you do not have or need to create an Arm account register at https://developer.arm.com/
2. When logged in you land on the AVH Devices page. Click CREATE DEVICE.

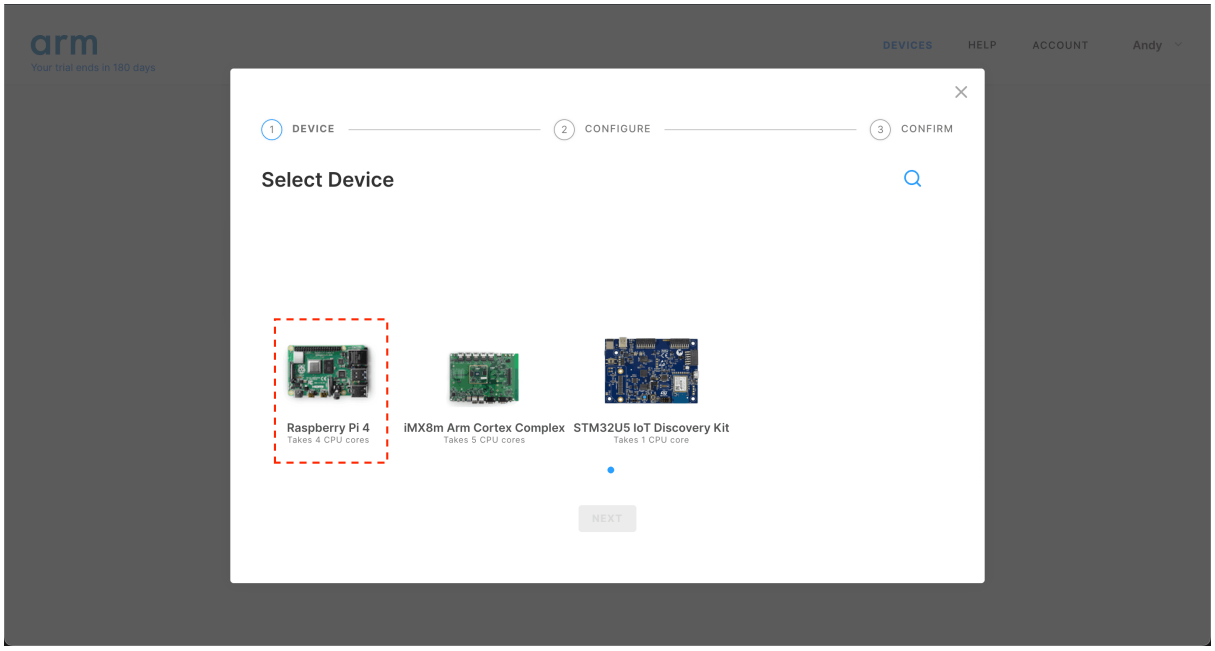
Figure 4-7: Create device



3. Click on the Raspberry Pi 4 to select it. Then click NEXT.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

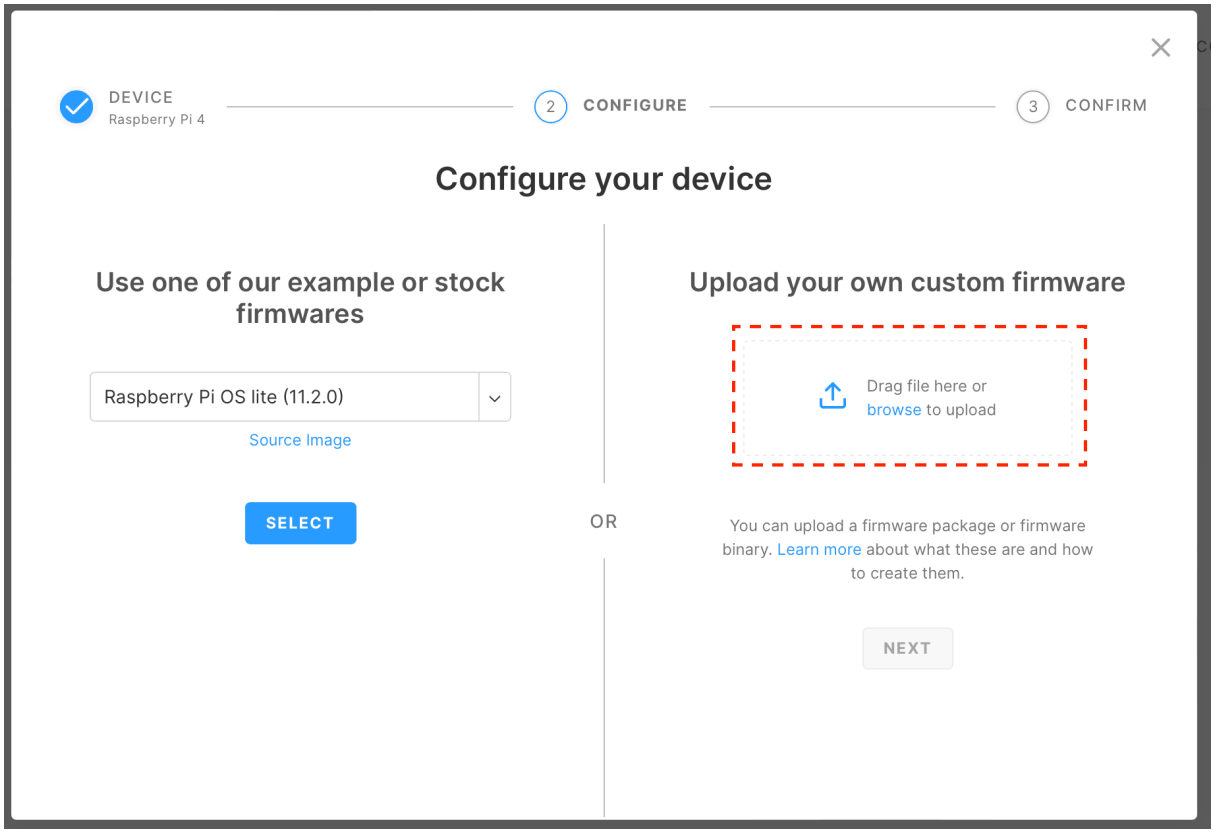
Figure 4-8: Select device



4. In this next step, you'll see a list of pre-configured examples/stock firmware or you can upload your own custom firmware. See Understanding Firmware on AVH for more information. Either drag&drop your custom firmware into the Drag file here box or use the browse option, navigate and select the desired custom firmware.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

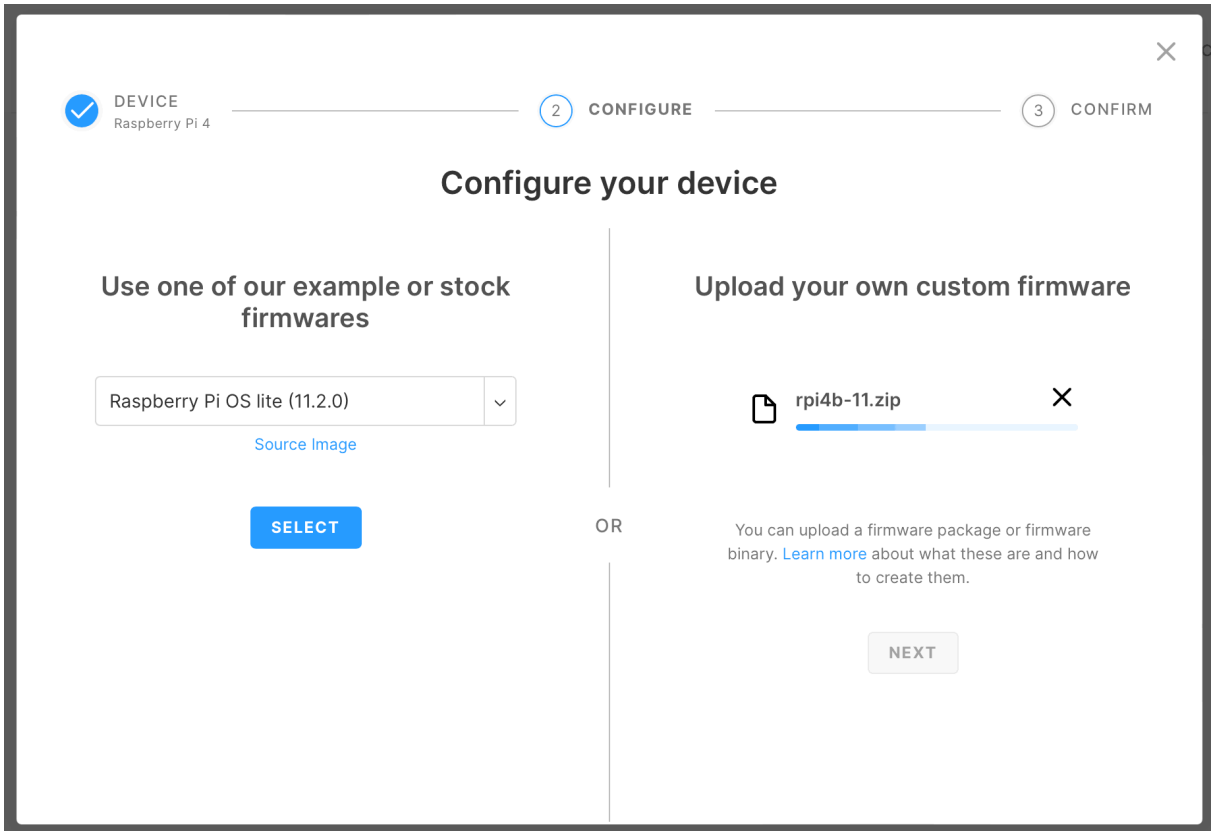
Figure 4-9: Configure your device



5. You will see the progress of your custom firmware uploading. This might take a while, depending on the size of custom firmware you're uploading. Once the upload is complete, click NEXT.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

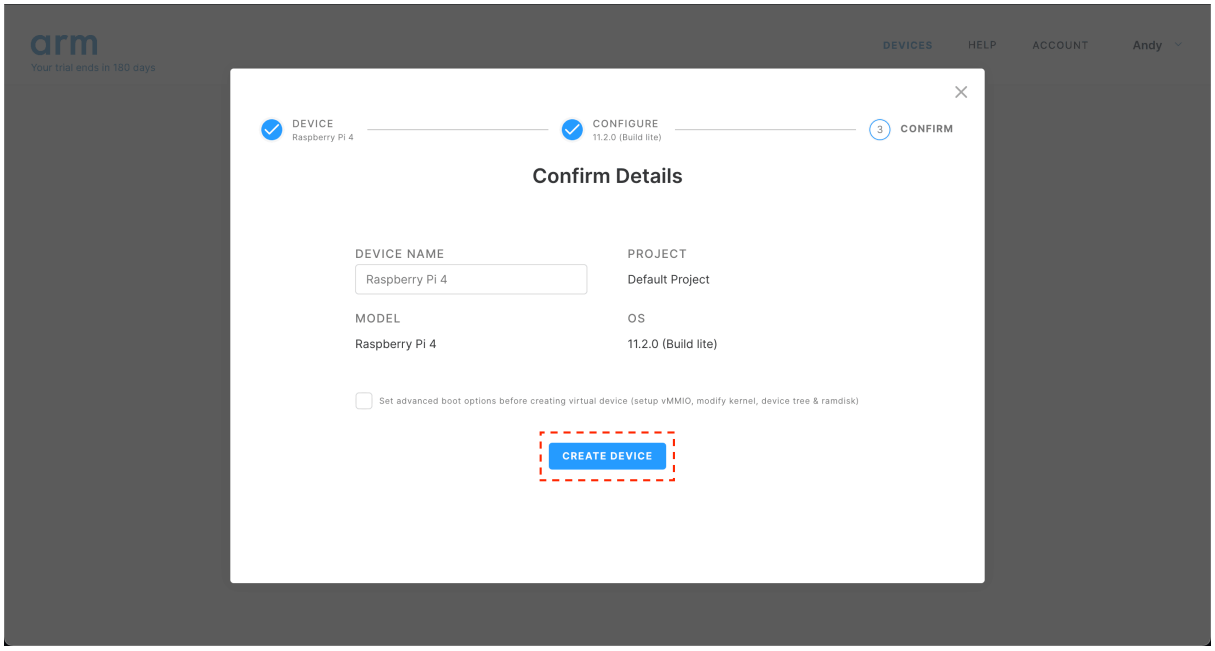
Figure 4-10: Configure your device



6. In the confirmation screen, click CREATE DEVICE. You can update the default name and use advanced boot options if required.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-11: Confirm details



7. The device will then be created. It takes a few moments...

4.1.4 Package Ubuntu Desktop Firmware for AVH

Follow this guide to package, install, and run custom Ubuntu Desktop firmware on a virtual Raspberry Pi 4 board.

AVH supports running custom Linux firmware packages.

This guide outlines the contents of a valid Ubuntu firmware package, provides a script for building the Ubuntu Desktop firmware, and walks through the installation on a virtual Raspberry Pi 4 board.

This packaging process follows the Understanding Firmware on AVH knowledge base article and uses the Raspberry Pi image of Ubuntu Desktop available directly from Ubuntu.

Please also refer to our companion guide Package Ubuntu Server Firmware for AVH.

4.1.4.1 Firmware Package Contents

A proper Ubuntu Desktop firmware package file contains the following:

- `Info.plist` - the version, type, build, unique identifier, and device identifier
- `nand` - the preinstalled Ubuntu Desktop arm64 image file
- `devicetree` - hardware components data for the Linux kernel

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- `kernel` - the Linux kernel file
- `ramdisk.img` (optional) - the initrd root file system image

Although specifying a `ramdisk.img` is generally optional, we need to add a reboot command in this case because of how this virtual device handles the first pass.

4.1.4.2 Firmware Packaging Script

The following script creates a custom firmware package in your working directory:

```
#!/bin/bash
set -e
mkdir rpi_ubuntu_desktop_firmware
cd rpi_ubuntu_desktop_firmware
# Download and extract the "Ubuntu 22.04.1 Desktop for RPi" image
wget https://cdimage.ubuntu.com/releases/22.04.1/release/ubuntu-22.04.1-
preinstalled-desktop-arm64+raspi.img.xz
xz -dv ubuntu-22.04.1-preinstalled-desktop-arm64+raspi.img.xz
mv ubuntu-22.04.1-preinstalled-desktop-arm64+raspi.img nand
# Attach the image file
LO="$(losetup -f)"
losetup -P "${LO}" nand
# Mount partition 1 to directory boot
mkdir boot
mount "${LO}p1" boot
# Copy the rpi4b devicetree
cp boot/bcm2711-rpi-4-b.dtb devicetree
# Extract the Linux kernel
zcat boot/vmlinuz > kernel
# Extract initrd
lz4 -d boot/initrd.img initrd.cpio
umount boot
rm -r boot/
mkdir rootfs
# Mount partition 2 to directory rootfs
mount "${LO}p2" rootfs
# Use aarch64 Linux sssd.conf and set permission to 600
cp rootfs/usr/lib/aarch64-linux-gnu/sssd/conf/sssd.conf rootfs/etc/sssd/.
chmod 600 rootfs/etc/sssd/sssd.conf
umount rootfs
rm -r rootfs/
losetup -d "${LO}"
mkdir ramdisk
cd ramdisk
# Extract initrd
cat ../initrd.cpio | cpio -idm
rm ../initrd.cpio
# Add reboot conditional to init before matching string
sed -i '/Move virtual filesystems over to the real filesystem/i \
if  /bin/grep init_resize /proc/cmdline; then\
/bin/reboot -f\
fi\
' init
# Create ramdisk.img from directory
find . | cpio -o -H newc -R root:root | lz4 -l  > ../ramdisk.img
cd ..
rm -r ramdisk/
# Create the Info.plist file
cat << EOF > Info.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

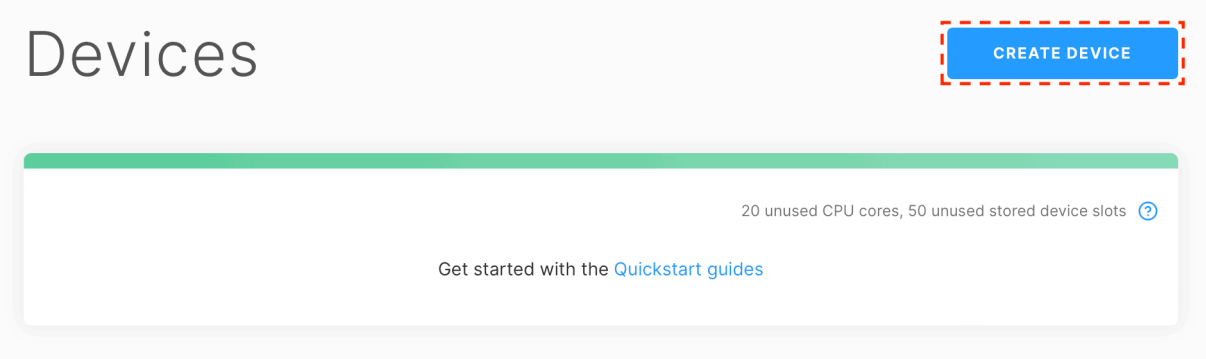
```
<key>Type</key>
<string>iot</string>
<key>UniqueIdentifier</key>
<string>Ubuntu Desktop on RPi</string>
<key>DeviceIdentifier</key>
<string>rpi4b</string>
<key>Version</key>
<string>22.04.1</string>
<key>Build</key>
<string>Ubuntu Desktop</string>
</dict>
</plist>
EOF
zip -rm ../rpi4b-ubuntu-desktop.zip Info.plist nand devicetree kernel ramdisk.img
cd ..
rm -r rpi_ubuntu_desktop_firmware/
```

4.1.4.3 Install the Package

To install the package:

1. Run the above shell script on your local Linux environment to create the firmware package `rpi4b-ubuntu-desktop.zip`.
2. On the AVH web interface, click CREATE DEVICE.

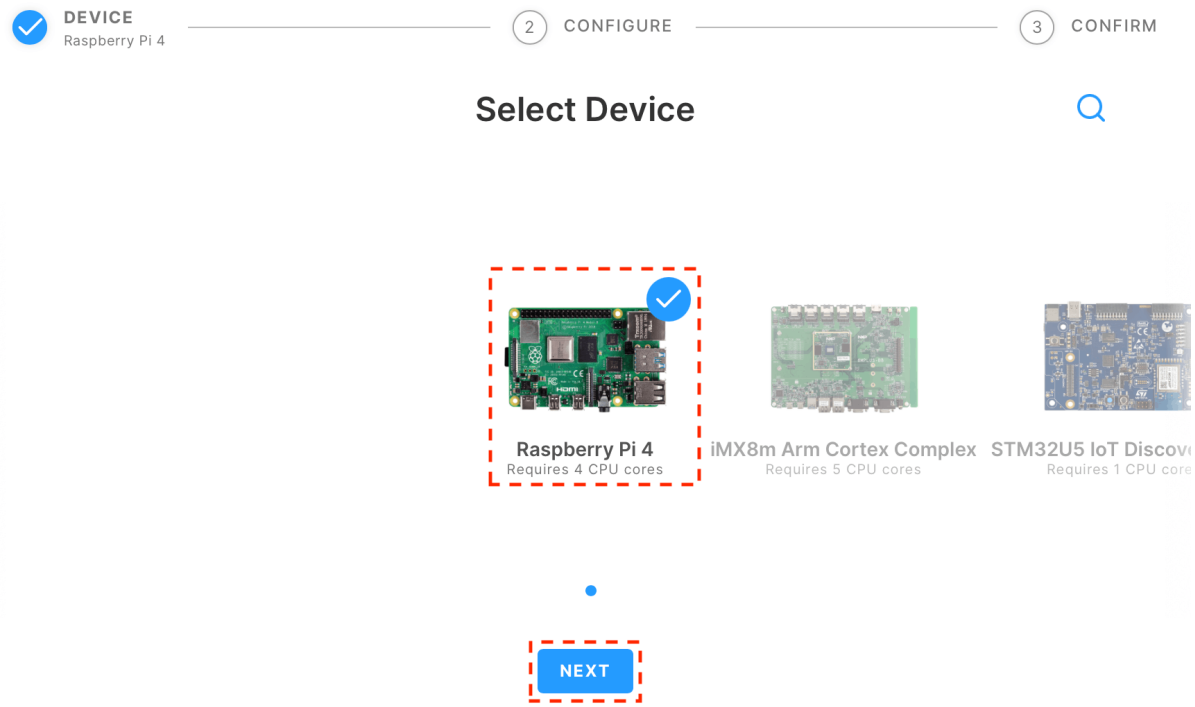
   Figure 4-12: Create device



3. In Step 1, choose the Raspberry Pi 4 and click NEXT.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-13: Select device



4. In Step 2, upload the custom firmware package. When the process is complete, click NEXT.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-14: Upload the custom firmware package

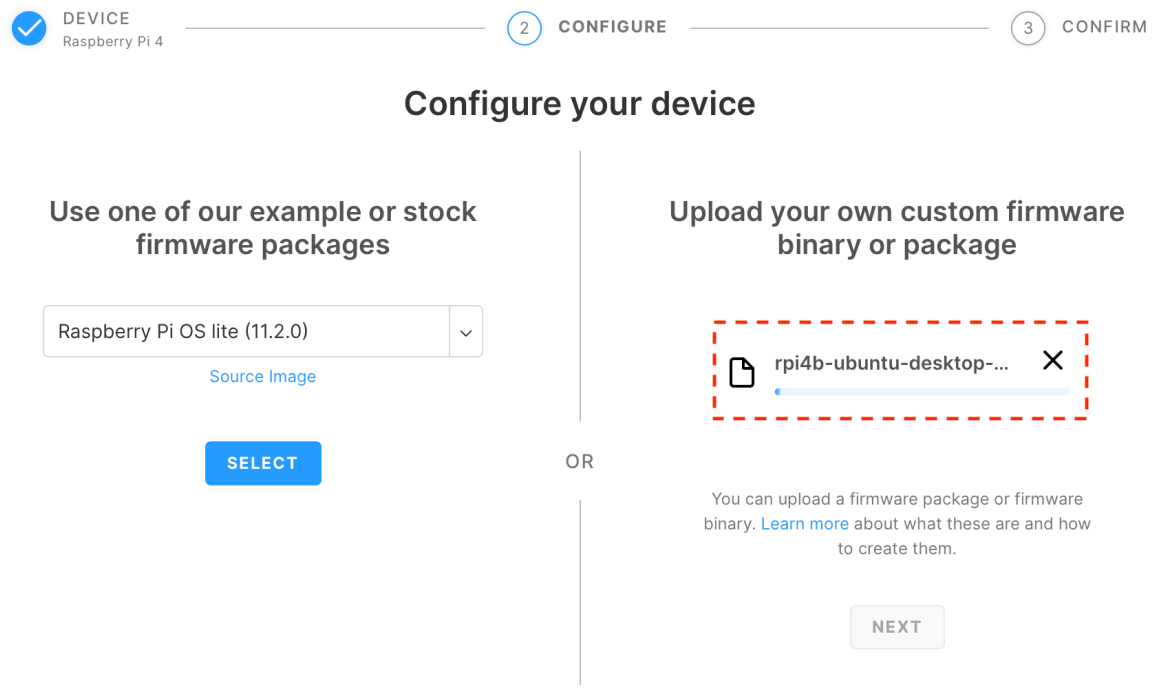
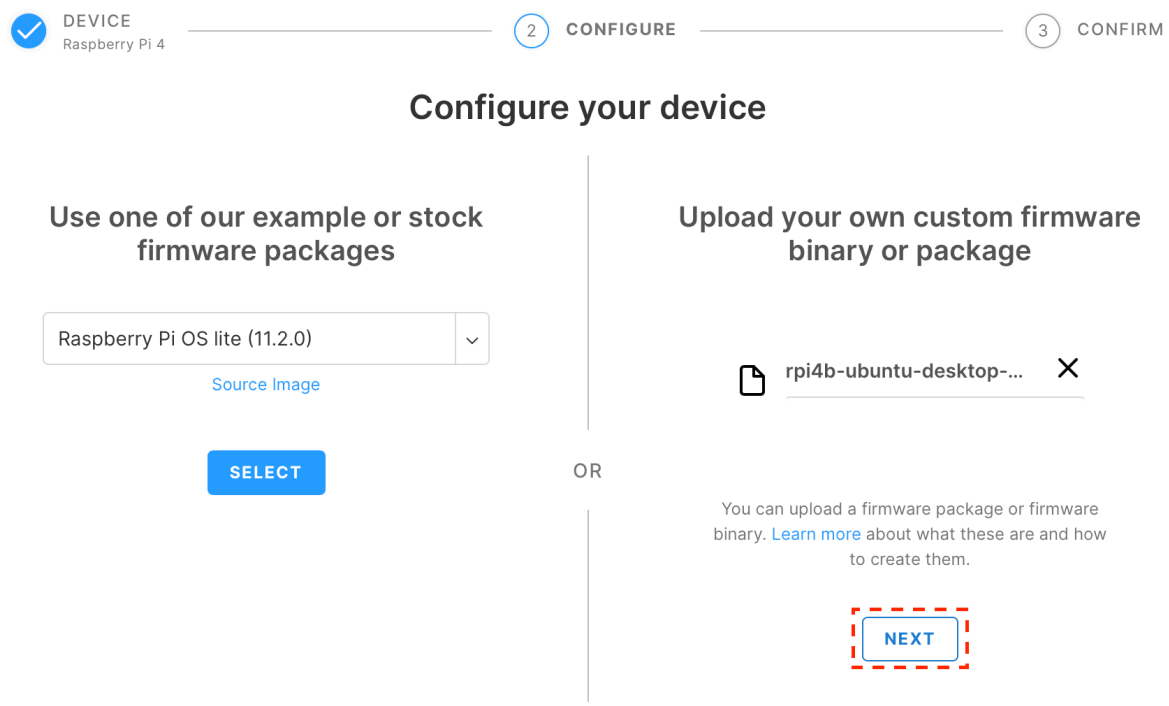


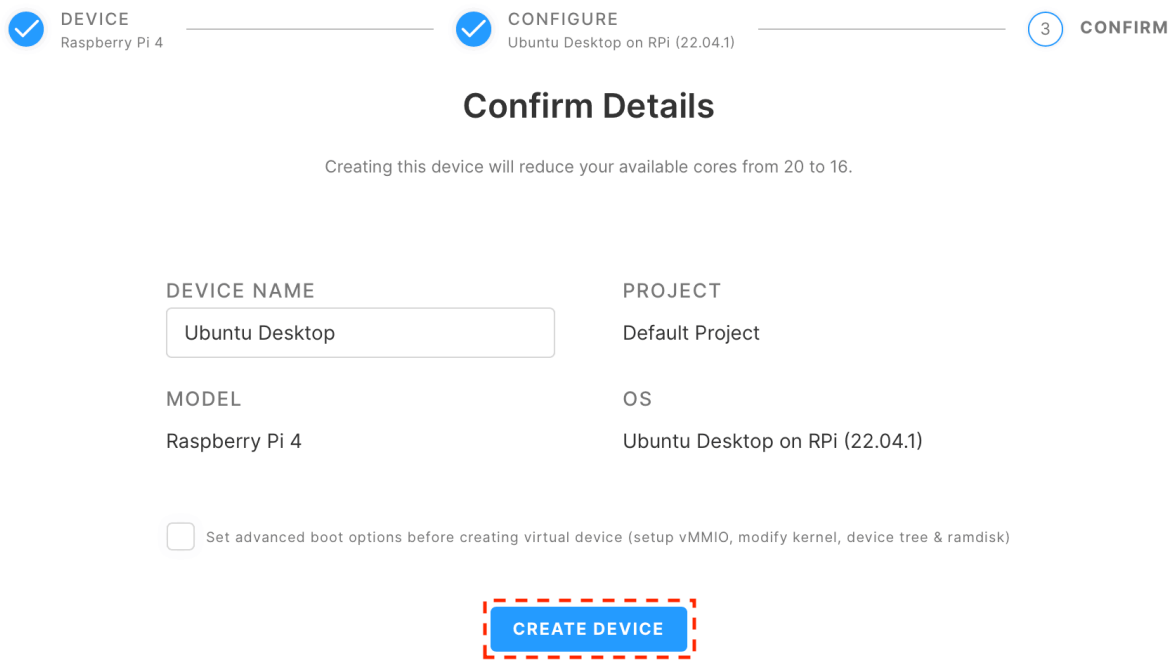
Figure 4-15: Click Next



5. Create the device without enabling advanced boot options.

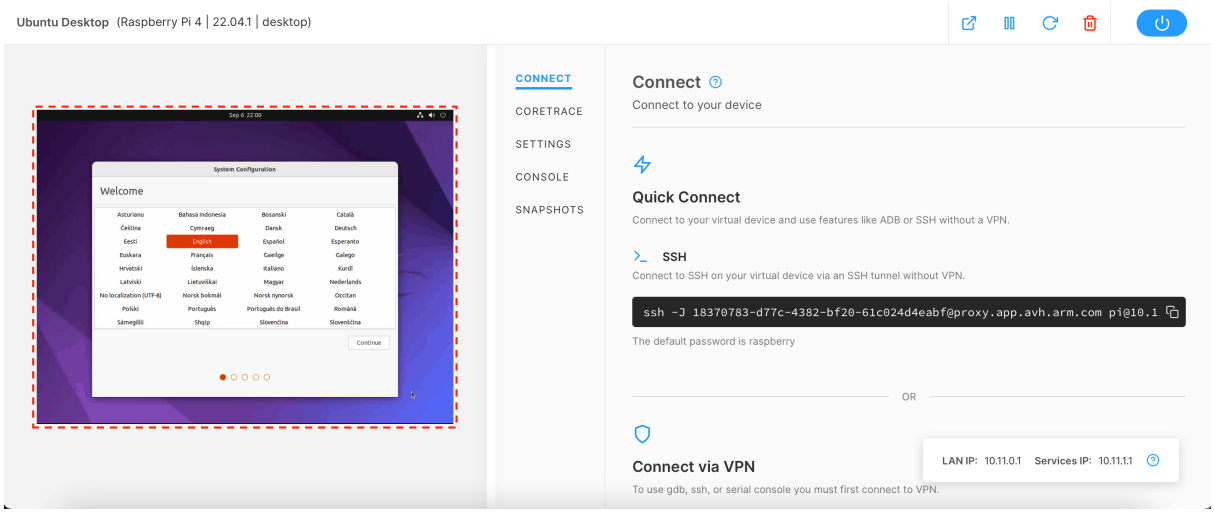
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-16: Create device



6. The virtual board will boot to the Ubuntu Desktop setup screen.

Figure 4-17: Login with the default credentials



Note

After initial setup and device restart, enable SSH using `sudo apt install openssh-server`. Since we are using a custom configuration, you will need to adjust the SSH commands under the CONNECT tab to use the username and password you specified during setup instead of the default credentials (`pi/raspberry`).

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.2 Storage Files

Information about storage files for different platforms:

- Storage Files for Raspberry Pi 4
- Storage Files for iMX8m Arm Cortex Complex
- Storage Files for STM32U5 IoT Discovery Kit

4.2.1 Storage Files for Raspberry Pi 4

This document outlines the required storage files information for the Raspberry Pi 4.

4.2.1.1 Required files

The following files are required:

- nand - SD card
  - block_size: 512
  - blocks: 31250000
  - max: 16000000000 bytes

Find out more about Raspberry Pi at https://www.raspberrypi.com/documentation/.

4.2.2 Storage Files for iMX8m Arm Cortex Complex

This document outlines the required storage files information for iMX8m Arm Cortex Complex

4.2.2.1 Required files

The following files are required:

- nand - eMMC NAND
  - block_size: 512
  - blocks: 31250000
  - max: 16000000000 bytes
- nand_flash0 - raw NAND (chip select 0)
  - block_size: 2048
  - blocks: 67584
  - max: 138412032 bytes

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

Includes 4M for ECC

- nand_flash1 - raw NAND (chip select 1)
  - block_size: 2048
  - blocks: 67584
  - max: 138412032 bytes

Note

Includes 4M for ECC

Find out more at the NXP website.

4.2.3 Storage Files for STM32U5 IoT Discovery Kit

This document outlines the required and optional storage files information for STM32U5 IoT Discovery Kit.

4.2.3.1 Required files

The following files are required:

- flash - on-chip main Flash
  - block_size: 512
  - blocks: 4096
  - max: 2097152 bytes
- flash-option - Boot options in flash
  - block_size: 512
  - blocks: 1
  - max: 512 bytes

Note

For more information about the flashoptions can be found at https://www.st.com/resource/en/reference_manual/rm0456-stm32u575585-armbased-32bit-mcus-stmicroelectronics.pdf in section 7.4

- flash-otp - OTP memory in Flash block

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

  - block_size: 512
  - blocks: 1
  - max: 512 bytes
- sdcard - SD card
  - block_size: 512
  - blocks: 1953125
  - max: 1000000000 bytes

4.2.3.2 Optional files

The following files are optional:

- spinor = SPI NOR Flash on QSPI bus
  - block_size: 512
  - blocks: 131073
  - max: 67109376 bytes

> **Note**
> If not supplied this will be filled with `0xff`.

- eeprom = EEPROM
  - block_size: 512
  - blocks: 65
  - max: 33280 bytes

> **Note**
> If not supplied this will be filled with `0xff`.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5. Advanced Features

The following articles are available:

- Play Video Files on Virtual i.M X8M Boards
- Use USB/IP to Attach Peripherals over the Internet
- vMMIO Introduction
- Kernel debugging
- Custom device tree

5.1 Play Video Files on Virtual i.M X8M Boards

Upload videos to your device or provide a URL to play videos in a loop on your virtual iMX8m board.

5.1.1 Table of Contents

This section contains the following:

- Open the GStreamer Multimedia Application
- Play Video Using the Web Interface
  - Use Your Webcam
  - Upload a File - Web Interface
  - Link to a URL - Web Interface
- Play Videos Using API and POST Requests
  - Upload a File - API and POST Requests
  - Link to a URL - API and POST Requests

The virtual iMX8m board supports playing `.mp4` video files directly on your device.

You can use either the AVH web interface or a set of API and POST requests.

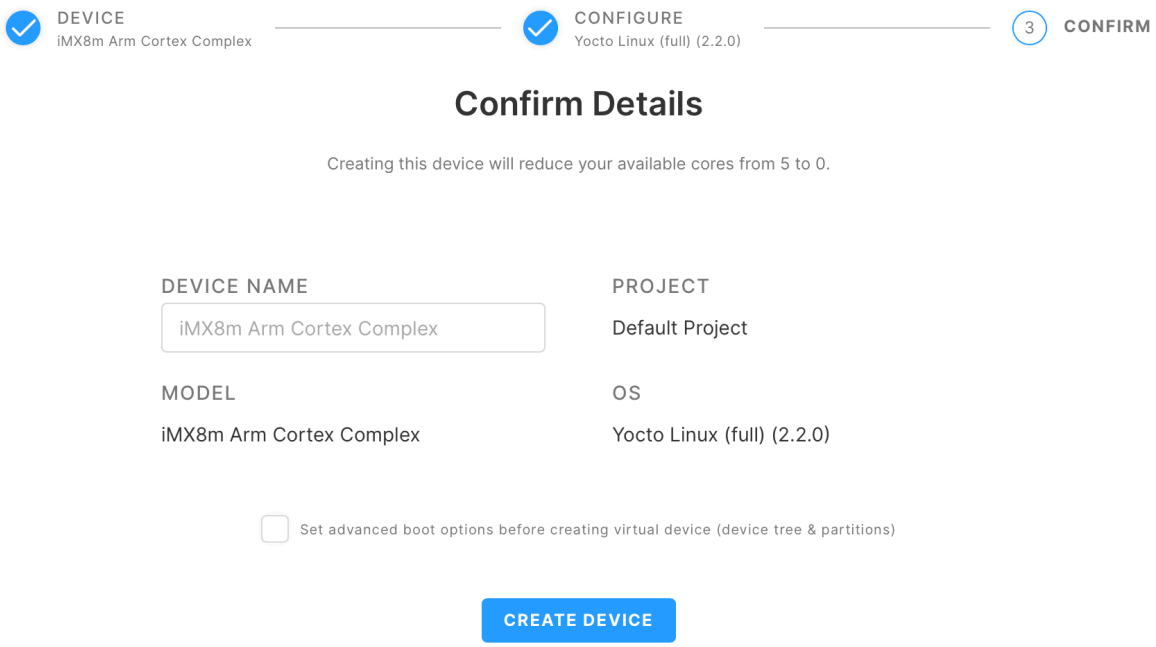
5.1.2 Open the GStreamer Multimedia Application

To open GStreamer:

1. Create a virtual iMX8M Arm Cortex Complex board running Yocto Linux OS version 2.2.0.

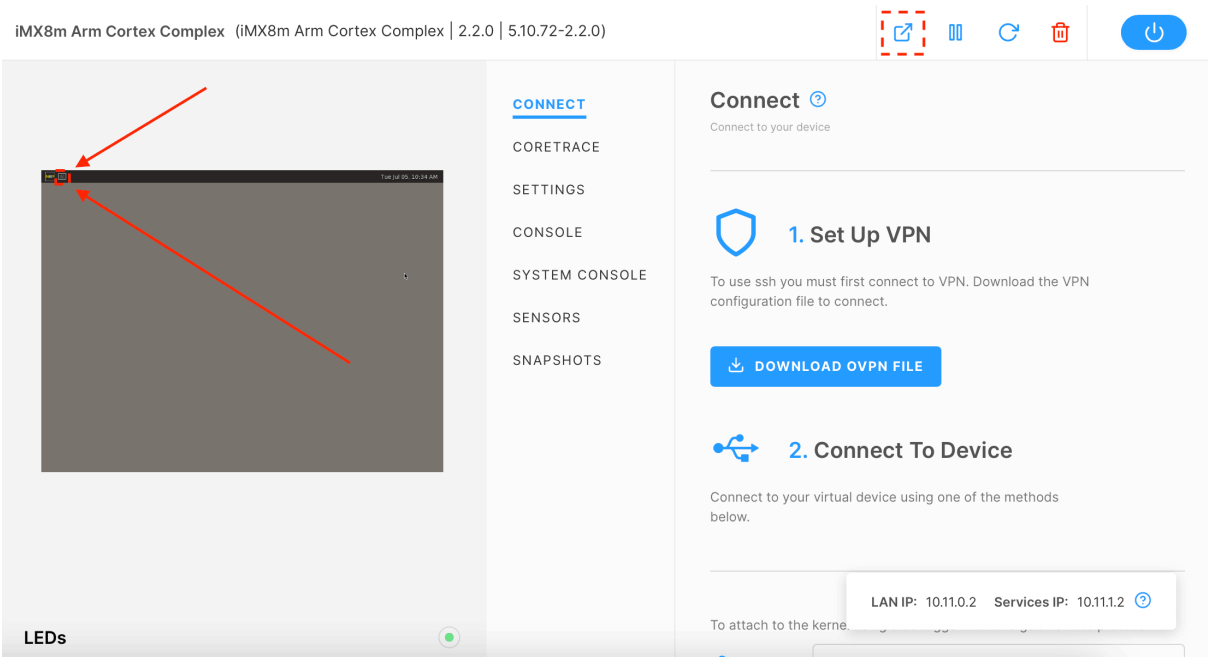
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-1: Create iMX8m Board



2. Open the Terminal by clicking the second icon in the top-left of the device's display. You may prefer to open the display to a new window by clicking Pop Out towards the top-right.

Figure 5-2: Open Terminal

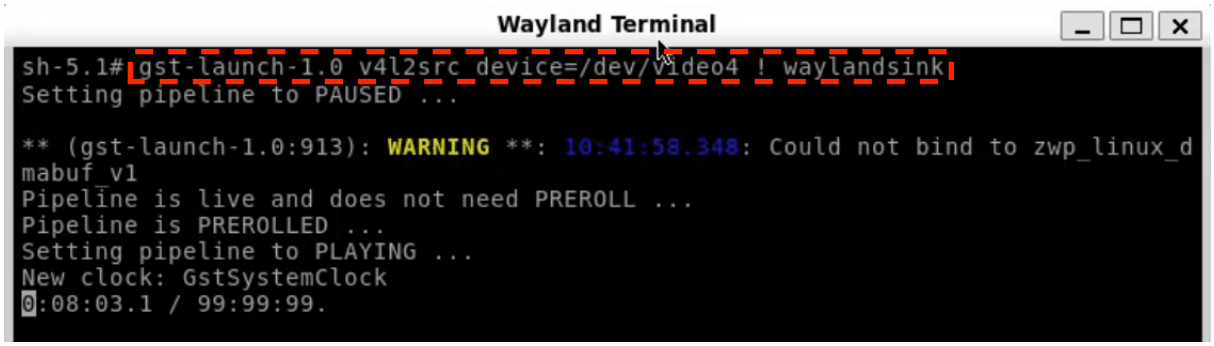


Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. Run the following command in Terminal to launch GStreamer.

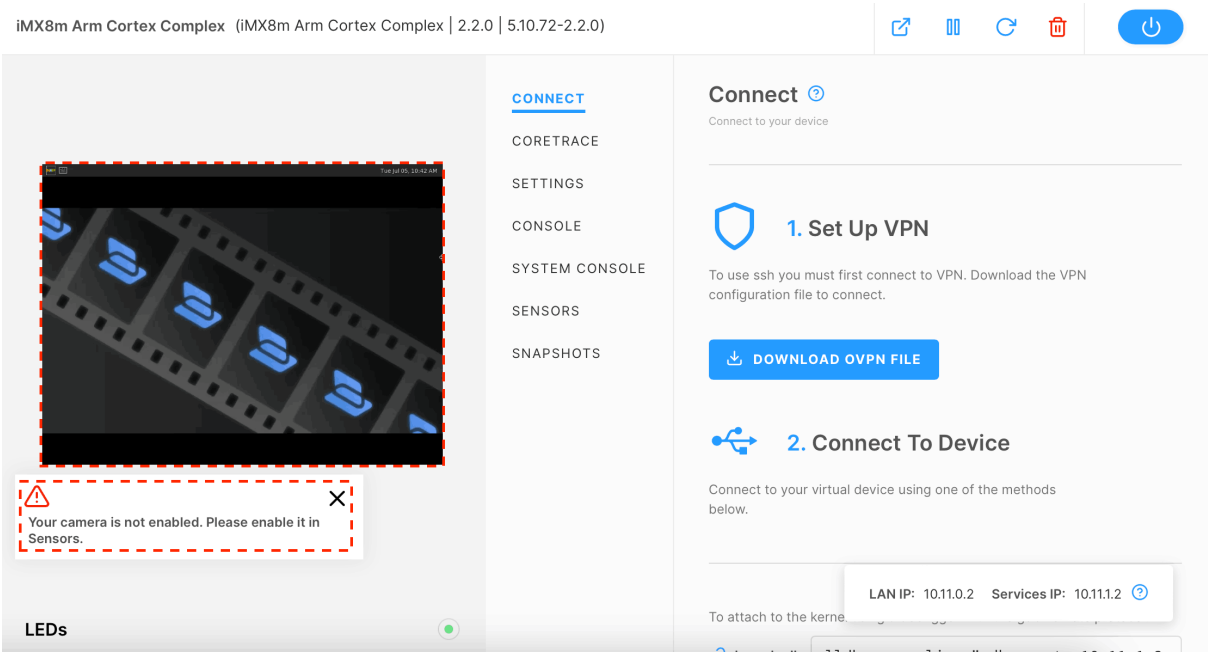
```
gst-launch-1.0 v4l2src device=/dev/video4 ! waylandsink
```

Figure 5-3: Terminal Command



4. After a few seconds, you should see the video player open along with a message that your camera is not enabled.

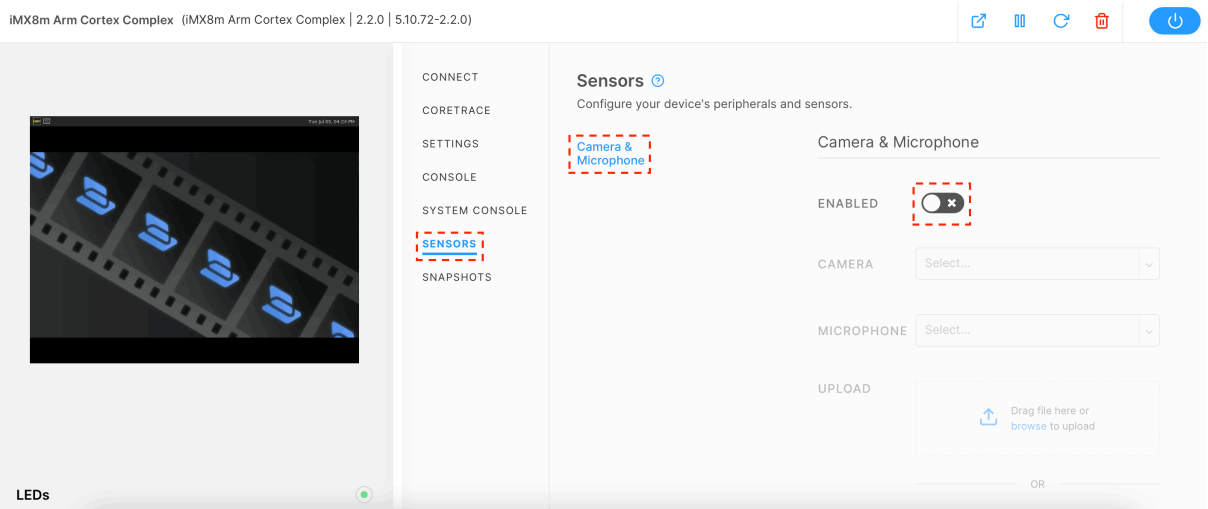
Figure 5-4: GStreamer Opened



5. Open the Camera & Microphone settings under the Sensors tab and enable the sensors.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-5: Enable the sensors



5.1.3 Play Video Using the Web Interface

The iMX8m device supports video playback from your webcam, by uploading a file, or by sharing the video's URL.

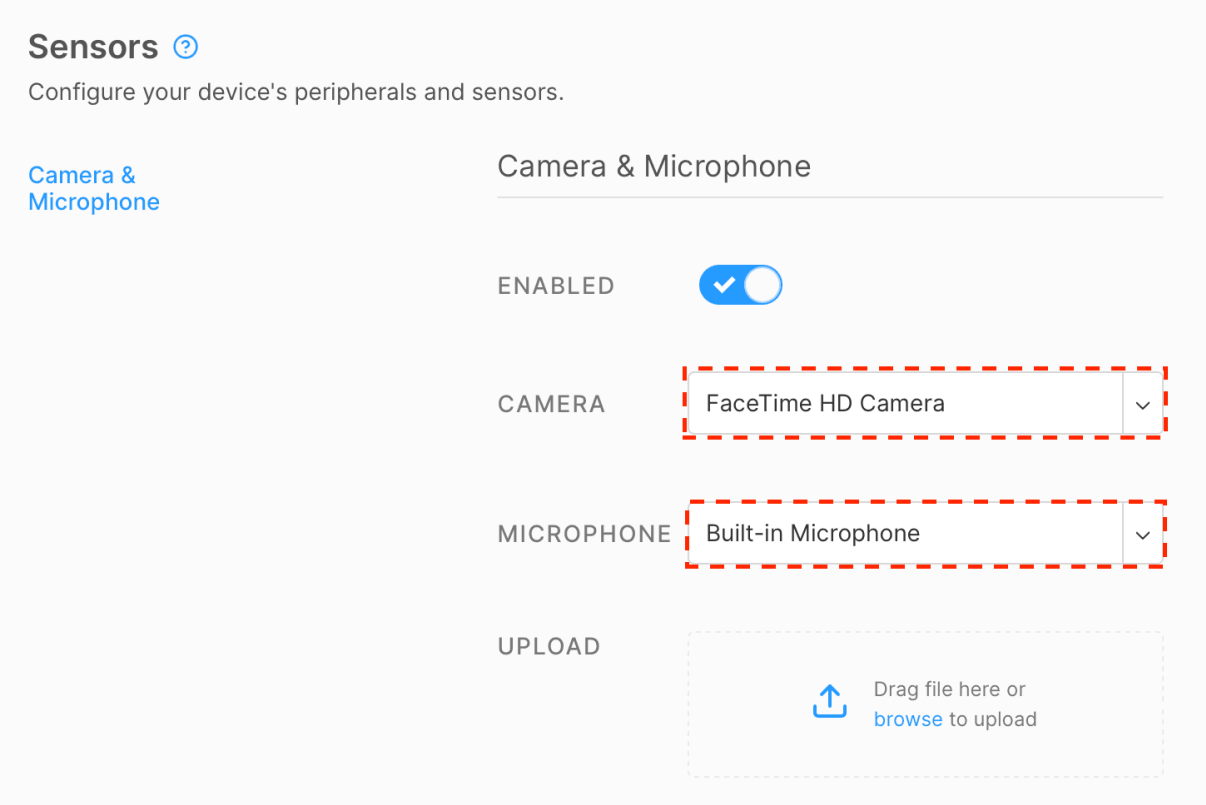
5.1.4 Use Your Webcam

To use your webcam:

1. After you enable the camera and microphone sensors, allow the browser to access your computer's built-in webcam and microphone. (This process will vary depending on your browser.)
2. Confirm that you can see your computer's sensors appear under CAMERA and MICROPHONE.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-6: Confirm sensors



3. You should see your webcam video appear on GStreamer.

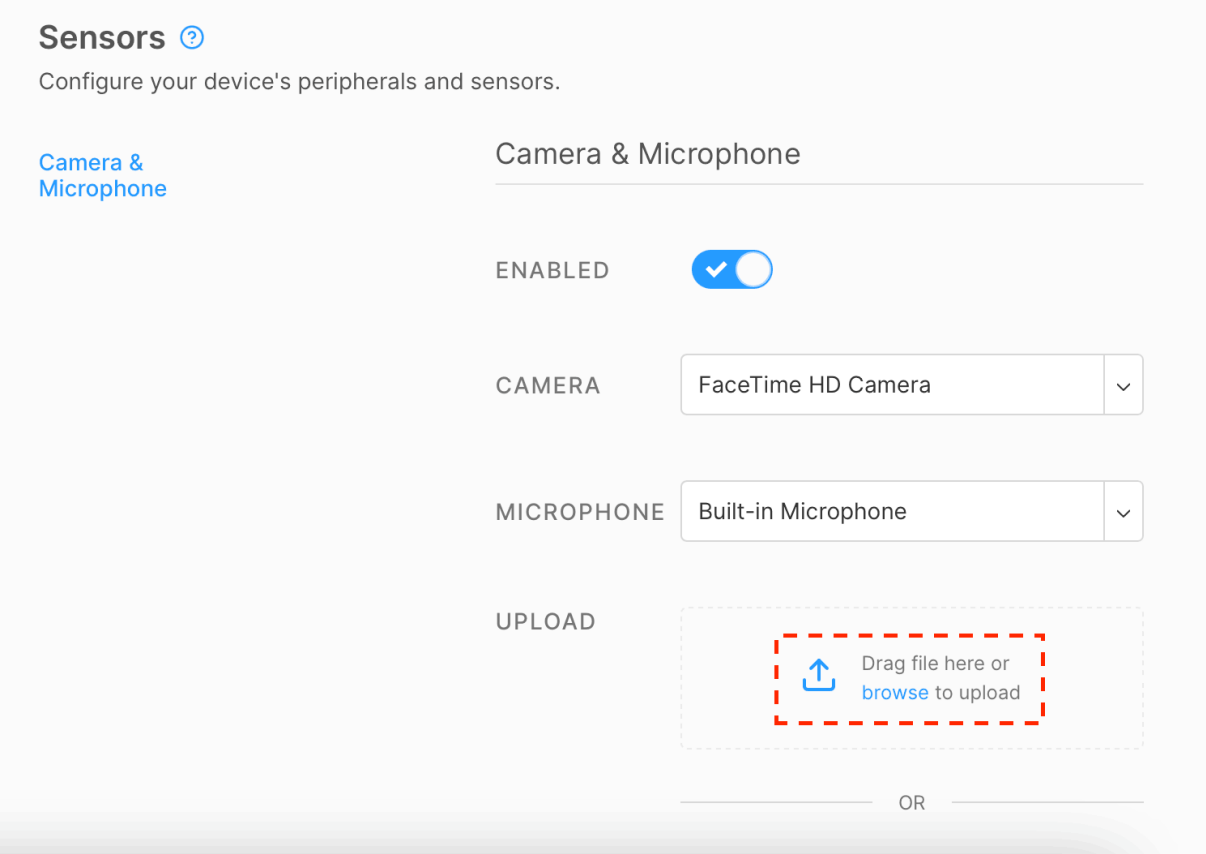
5.1.5 Upload a File - Web Interface

To upload a file using the web interface:

1. After enabling the virtual camera and microphone in the Sensors tab, upload the `.mp4` video file. You do not need to allow the browser to access your local computer's physical camera and microphone.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-7: Upload the `.mp4` video file



2. When the upload completes, click PLAY VIDEO. The video should play on GStreamer in a loop.

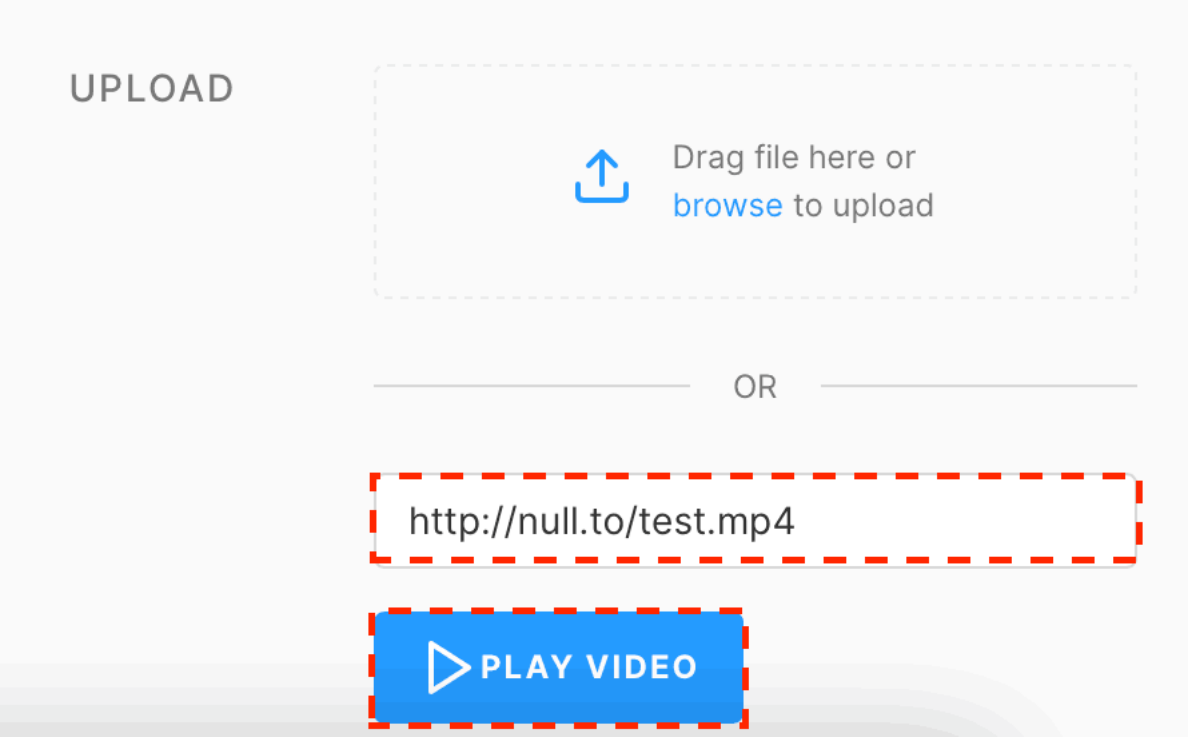
5.1.6 Link to a URL - Web Interface

To link to a URL using the web interface:

1. After enabling the camera and microphone in the Sensors tab, enter a URL to a `.mp4` video and click PLAY VIDEO. You do not need to allow the browser to access your local computer's physical camera and microphone.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-8: Enter a URL



2. The video should play on GStreamer in a loop.

5.1.7 Play Videos Using API and POST Requests

You can also use the command line to play multimedia files.

Start with GStreamer opened (`gst-launch-1.0 v4l2src device=/dev/video4 ! waylandsink`).

5.1.8 Upload a File - API and POST Requests

To upload a file using the AVH API:

1. Use the AVH API to acquire the device instance.
2. Upload the `.mp4` file to that instance.

```
const videoImage = await instance.uploadImage('media',
 $LOCATION_OF_AN_MP4_ON_YOUR_LOCAL_DISK, '$ANY_ARBITRARY_STRING')
```

3. Note the `imageId` that is returned.
4. Make a POST request to `/api/v1/instances/${instanceId}/play` with the following body.

```
{
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
  instanceId,
  imageId
}
```

5. Verify that you can see the video playing.

5.1.9 Link to a URL - API and POST Requests

To link to a URL using the AVH API:

1. Use the AVH API to acquire the device instance.
2. Make a POST request to `/api/v1/instances/${instanceId}/play` with the video URL in the body.

```
{
  instanceId,
  url: 'http://null.to/test.mp4'
}
```

3. Verify that you can see the video playing.

5.2 Use USB/IP to Attach Peripherals over the Internet

The USB/IP protocol allows you to share USB devices between your virtual machine and another device.

5.2.1 Table of Contents

This section contains the following:

- Use Your Local Linux Computer as the USB/IP Host
  - Set Up the USB/IP Daemon on Your Computer
  - Attach a Physical Device to the Virtual Board Client
  - (Optionally) Close the Connection When Finished
- Use a Virtual STM32U5 Board as the USB/IP Host
  - Build USB Firmware Files with the STM32CubeIDE Tool
  - Attach a Virtual USB Peripheral to a Virtual Raspberry Pi 4 Board
  - Attach a Virtual USB Peripheral to a Physical Computer Using a VPN Connection
  - Attach a Virtual USB Peripheral to a Physical Computer Using SSH Port Forwarding

USB/IP is an open-source tool for encapsulating USB I/O messages into TCP/IP messages. The host device shares a USB peripheral that the client device can use as if it were directly attached to the client. Read more about the protocol on the USB/IP SourceForge page.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

In this article, we will show an example of how to Use Your Local Linux Computer as the USB/IP Host to directly control the cursor on a virtual Raspberry Pi Desktop device over a VPN connection.

We will also show examples of how to Use a Virtual STM32U5 Board as the USB/IP Host to share both input and output USB peripherals.

5.2.2 Use Your Local Linux Computer as the USB/IP Host

We will be using a physical computer with a fresh install of Ubuntu Desktop 22.04 LTS and a Raspberry Pi 4 running the Desktop version of firmware 11.2.0.

Start with the USB peripheral disconnected from your device.

Note

The USB device we use will stop working on your host, so make sure to have a second mouse (or laptop trackpad) to control your computer.

5.2.3 Set Up the USB/IP Daemon on Your Computer

Start with a fresh install of the most recent LTS version of Ubuntu Desktop as the host and a virtual Raspberry Pi 4 Desktop board as the client.

1. Install the required packages.

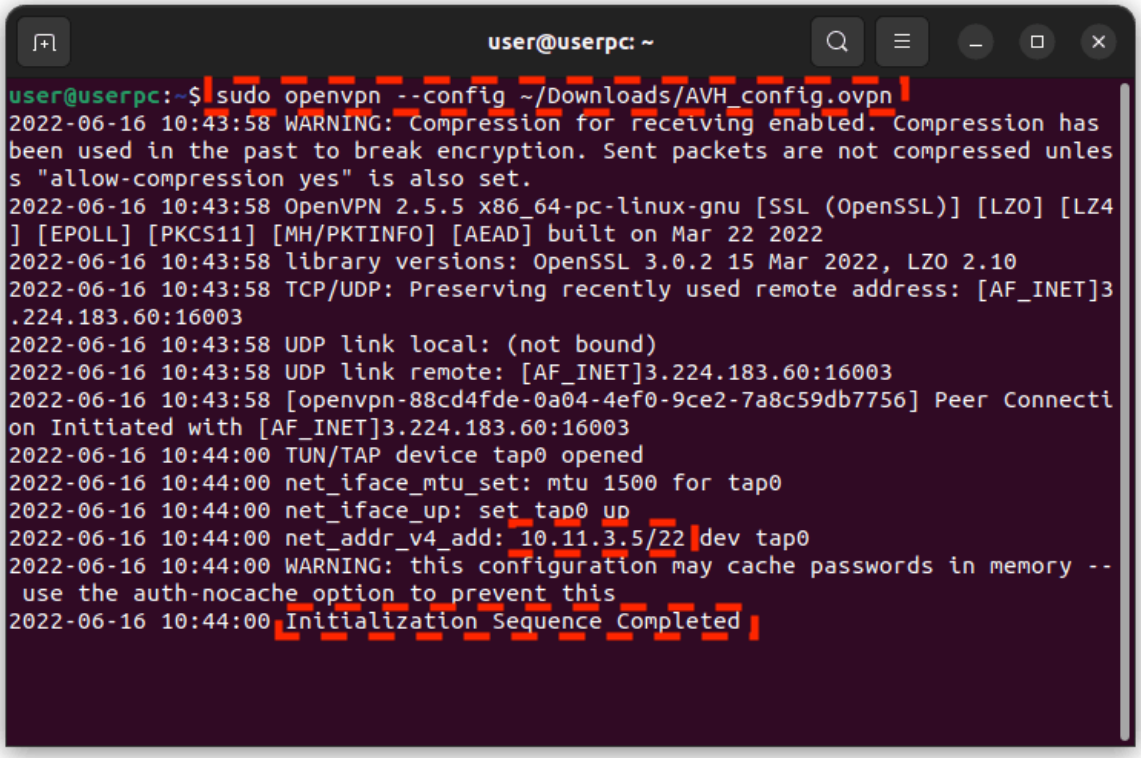
```
sudo apt update && sudo apt install -y linux-tools-$(uname -r) hwdata ubuntu-restricted-extras
```

2. Download the `.ovpn` file from the Raspberry Pi 4's Connect tab and connect to your AVH project using openvpn. Make note of the VPN's IP address, which in our example is `10.11.3.5`.

```
sudo openvpn --config ~/Downloads/AVH_config.ovpn
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-9: Connect to AVH VPN

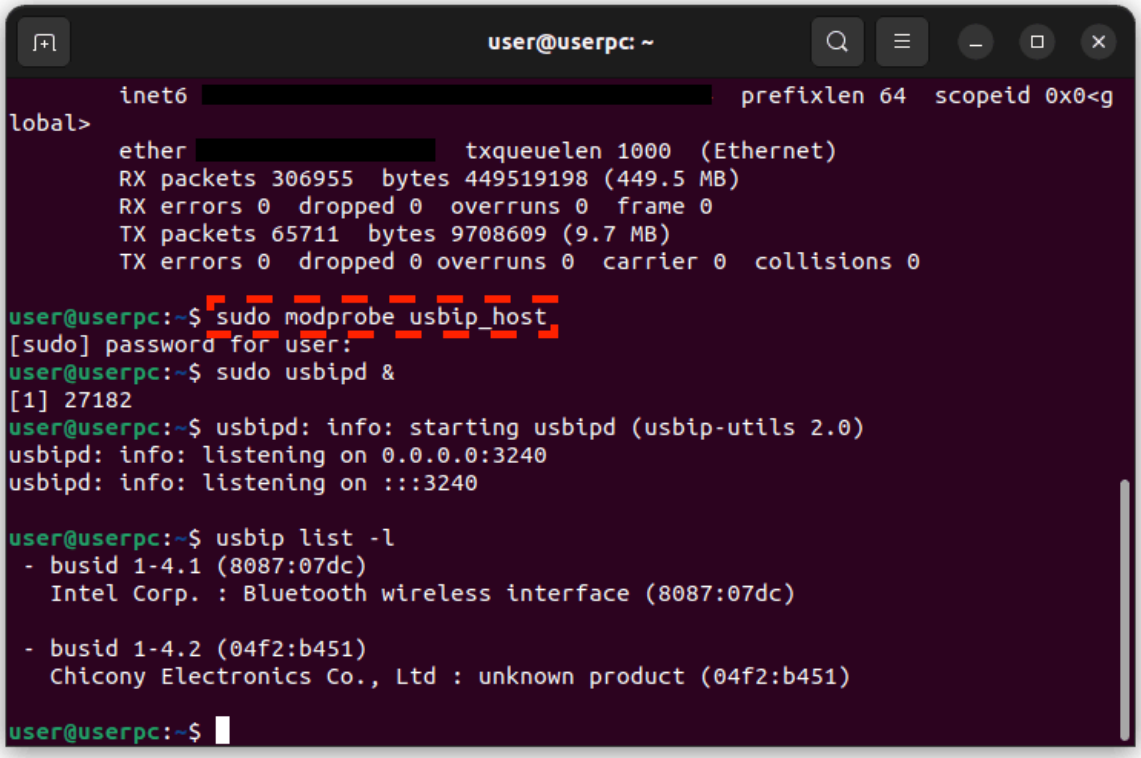


3. Open a new Terminal window and load the USB/IP Host kernel module.

```
sudo modprobe usbip_host
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-10: Load the USBIP host kernel package

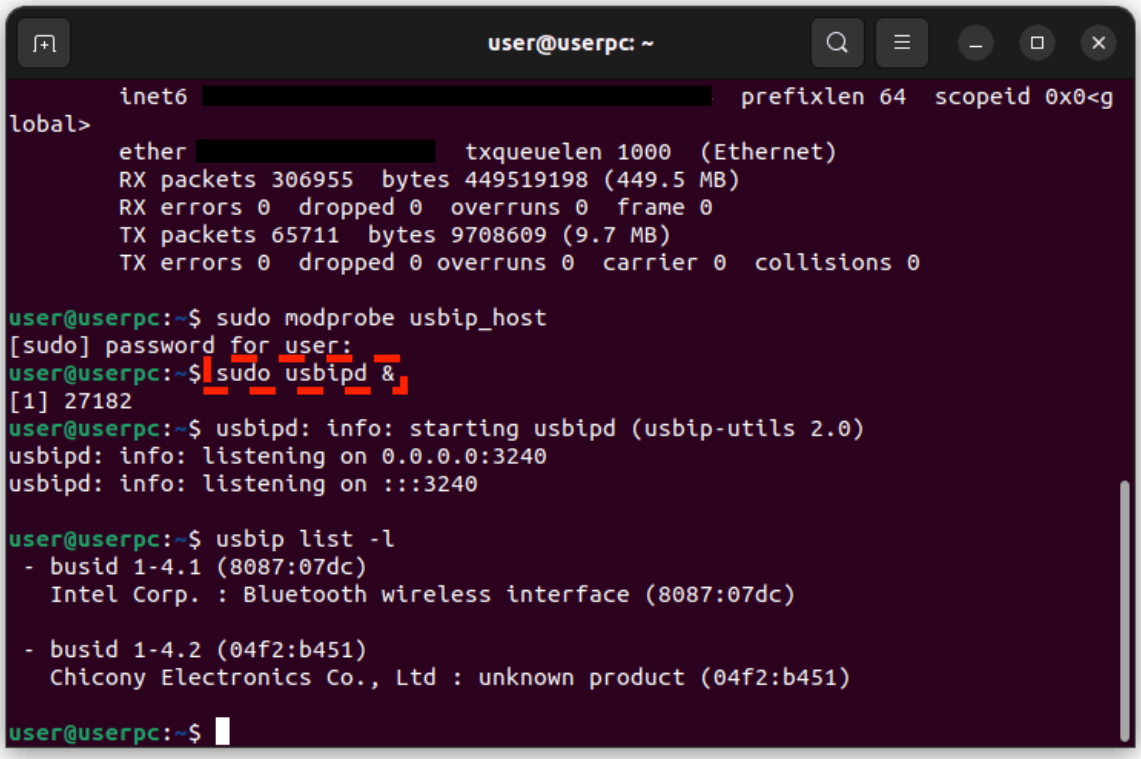


4. Start the USB/IP daemon in the background.

```
sudo usbipd &
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-11: Run the USBIP daemon



5. With the USB mouse dongle still disconnected, list the available peripherals.

```
usbip list -l
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

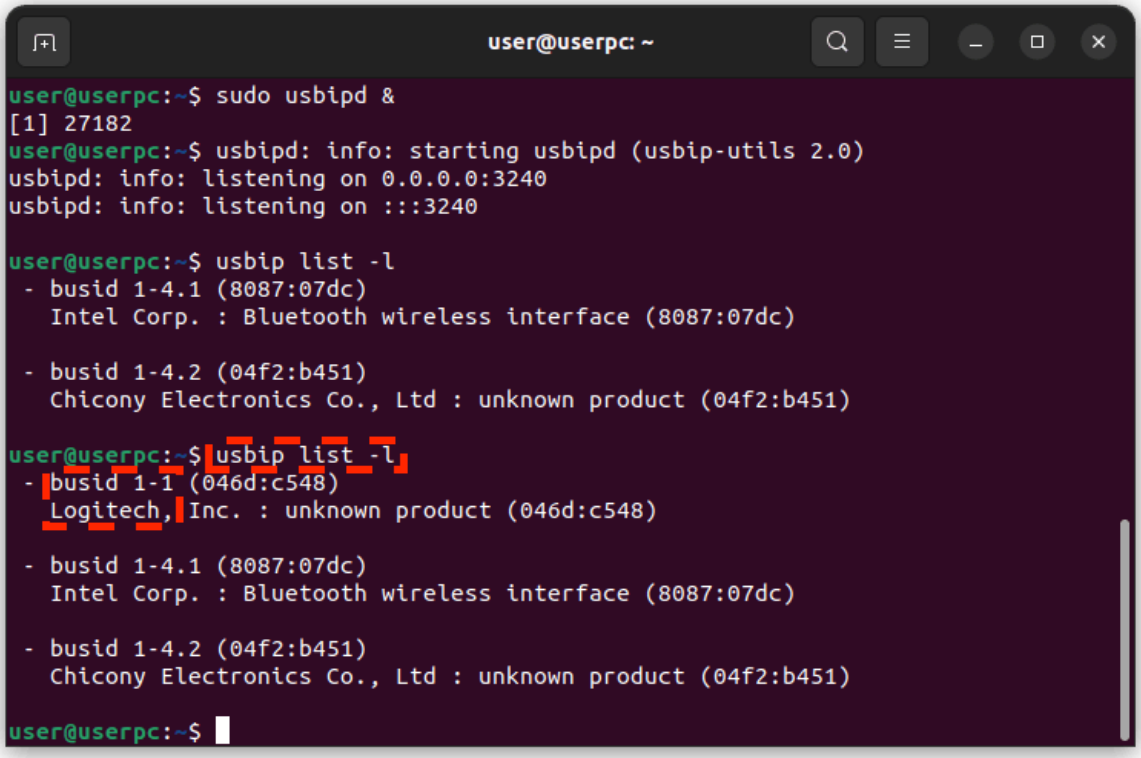
Figure 5-12: USBIP list without device

```
user@userpc: ~
        inet6 [redacted]  prefixlen 64  scopeid 0x0<g
lobal>
        ether [redacted]  txqueuelen 1000  (Ethernet)
        RX packets 306955  bytes 449519198 (449.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 65711  bytes 9708609 (9.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

user@userpc:~$ sudo modprobe usbip_host
[sudo] password for user:
user@userpc:~$ sudo usbipd &
[1] 27182
user@userpc:~$ usbipd: info: starting usbipd (usbip-utils 2.0)
usbipd: info: listening on 0.0.0.0:3240
usbipd: info: listening on :::3240

user@userpc:~$ usbip list -l
 - busid 1-4.1 (8087:07dc)
   Intel Corp. : Bluetooth wireless interface (8087:07dc)

 - busid 1-4.2 (04f2:b451)
   Chicony Electronics Co., Ltd : unknown product (04f2:b451)

user@userpc:~$
```

6. Connect the physical USB mouse dongle to your computer and run the `list` command again. You should see a new device on a new bus number. In our example, we have a new device on bus `1-1`.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-13: USBIP list with device

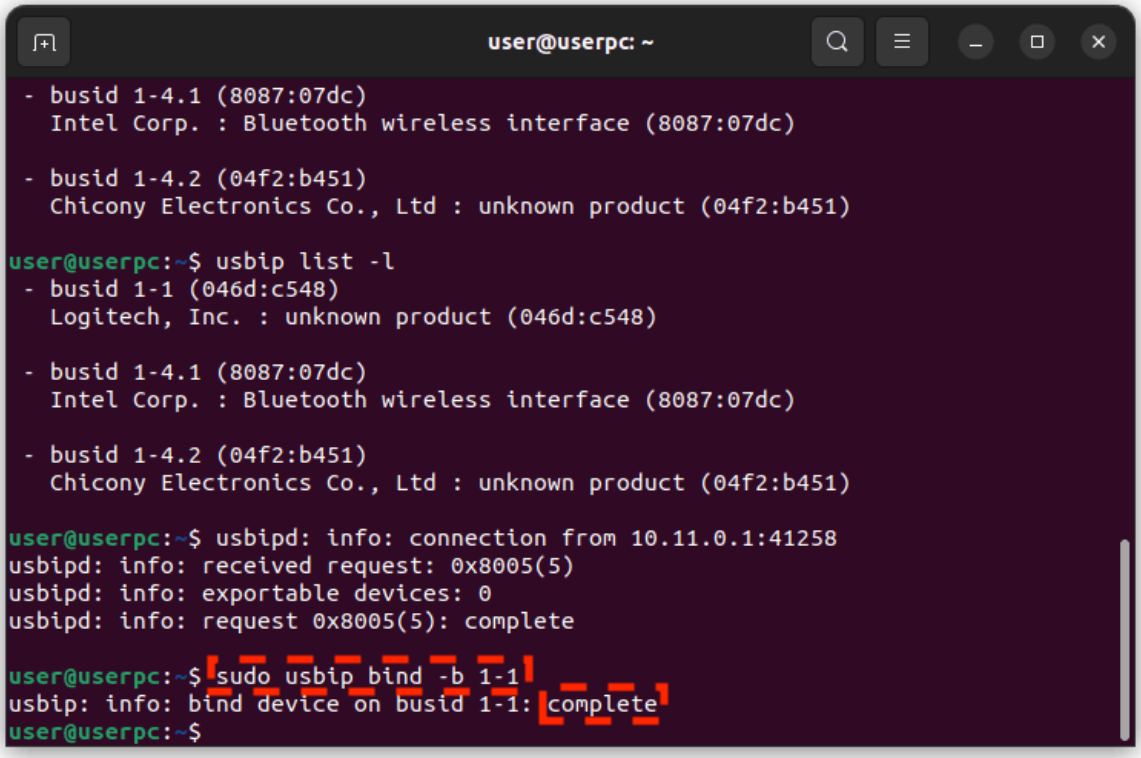


7. Bind the appropriate USB bus, `1-1` in our case. (This will cause the device to stop functioning on your host computer, so make sure to have a backup in the case of a mouse.)

```
sudo usbip bind -b 1-1
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-14: USBIP bind



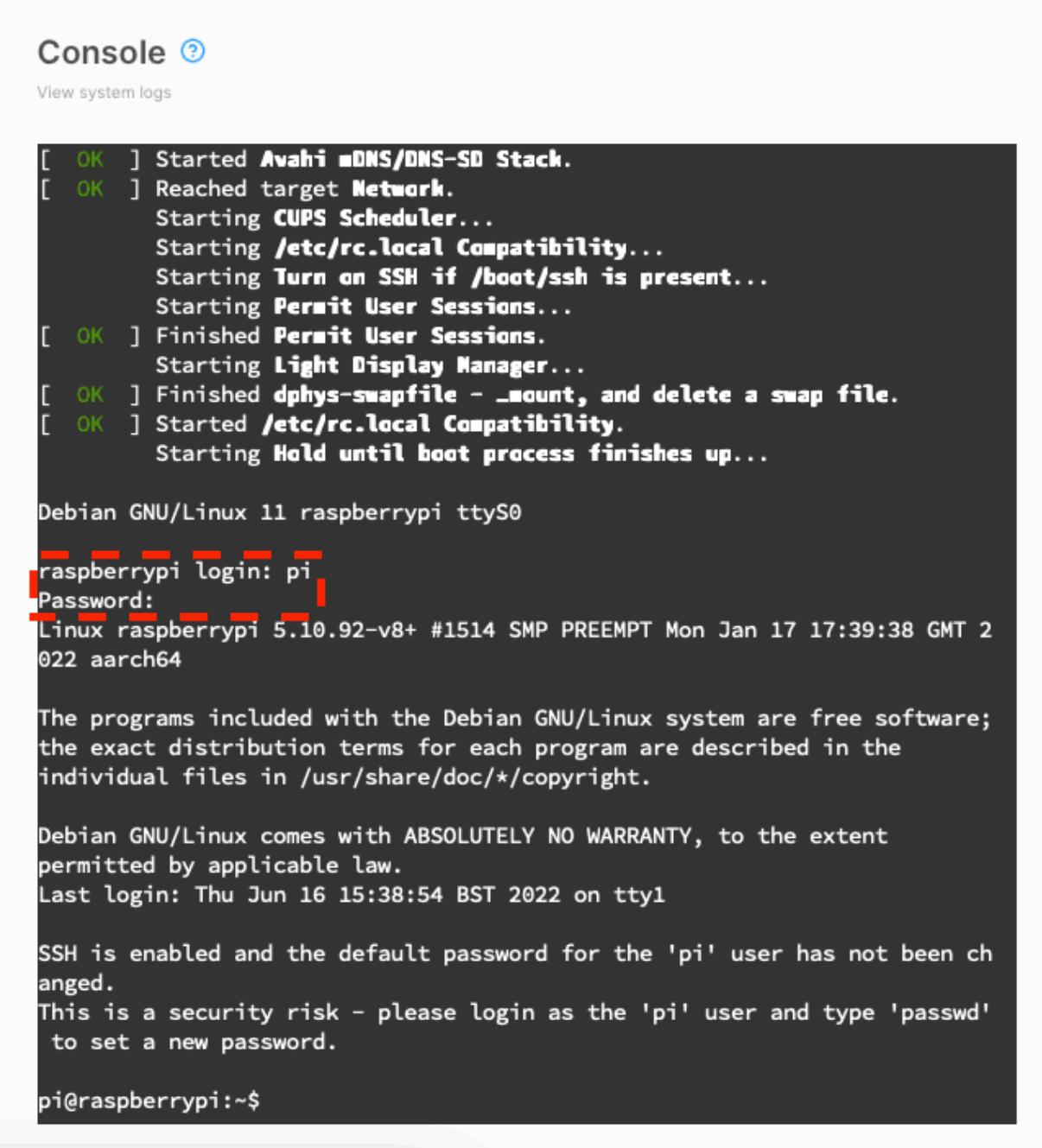
5.2.4 Attach a Physical Device to the Virtual Board Client

To attach a physical device to the virtual board client:

1. On your newly created Raspberry Pi 4 virtual board, log in with the credentials `pi` and `raspberry`.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-15: Log in to RPi 4 Desktop



2. Install the USB/IP package.

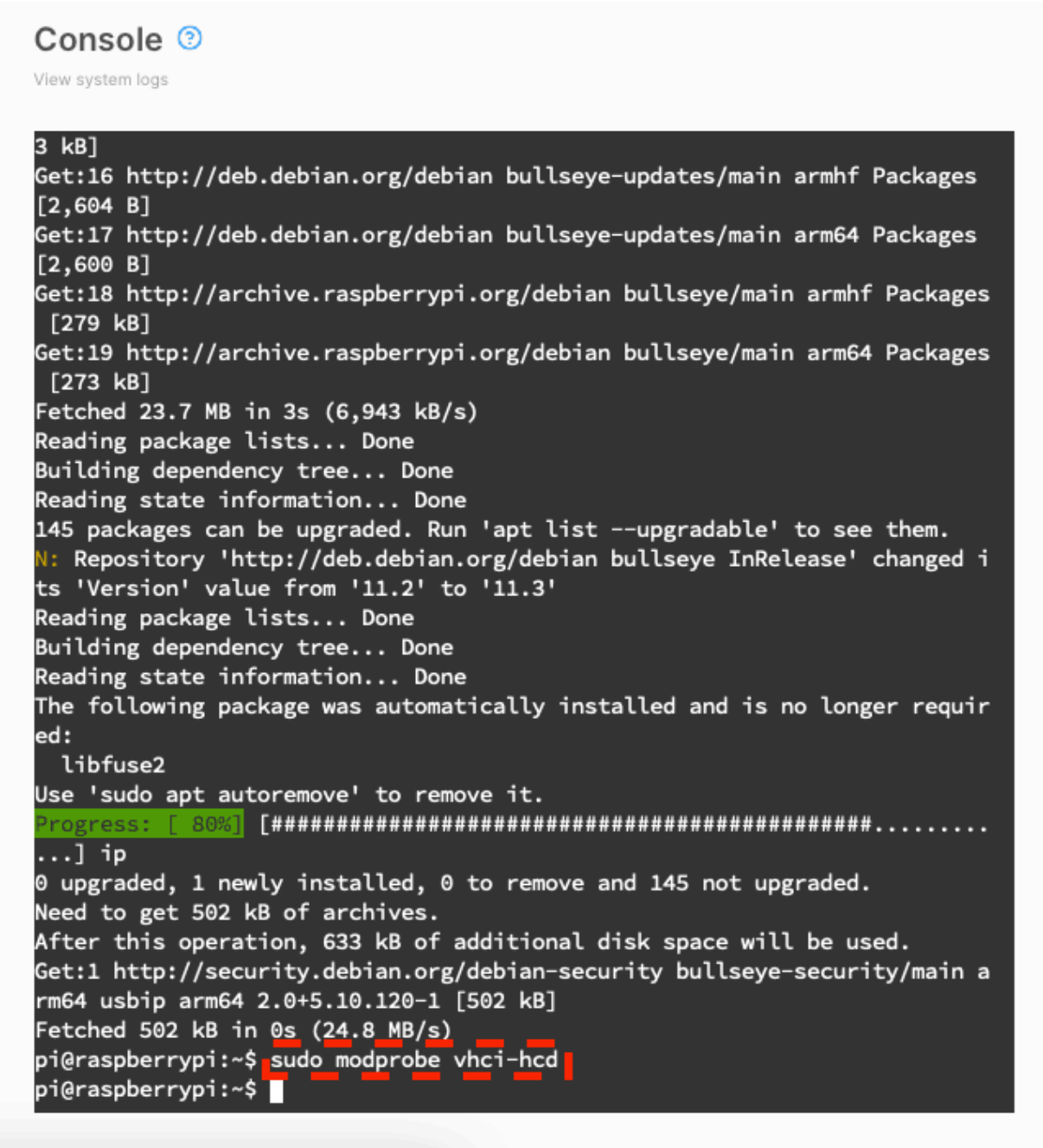
```
sudo apt update && sudo install usbip
```

3. Load the Virtual Host Controller Interface kernel module.

```
sudo modprobe vhci-hcd
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-16: Load the VHCI kernel module

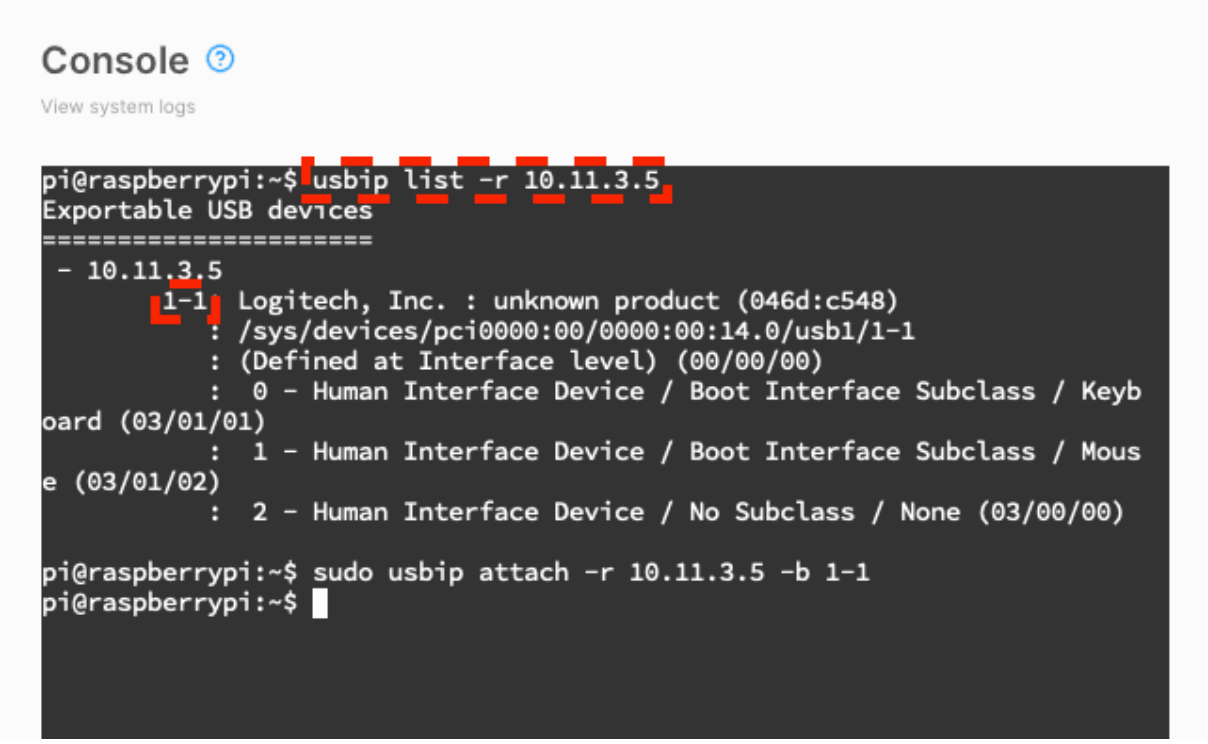


4. Using the VPN IP address from openvpn, list the exportable USB devices on the host. Look for the same bus number from the previous section.

```
usbip list -r 10.11.3.5
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-17: USBIP list on RPi4 Desktop

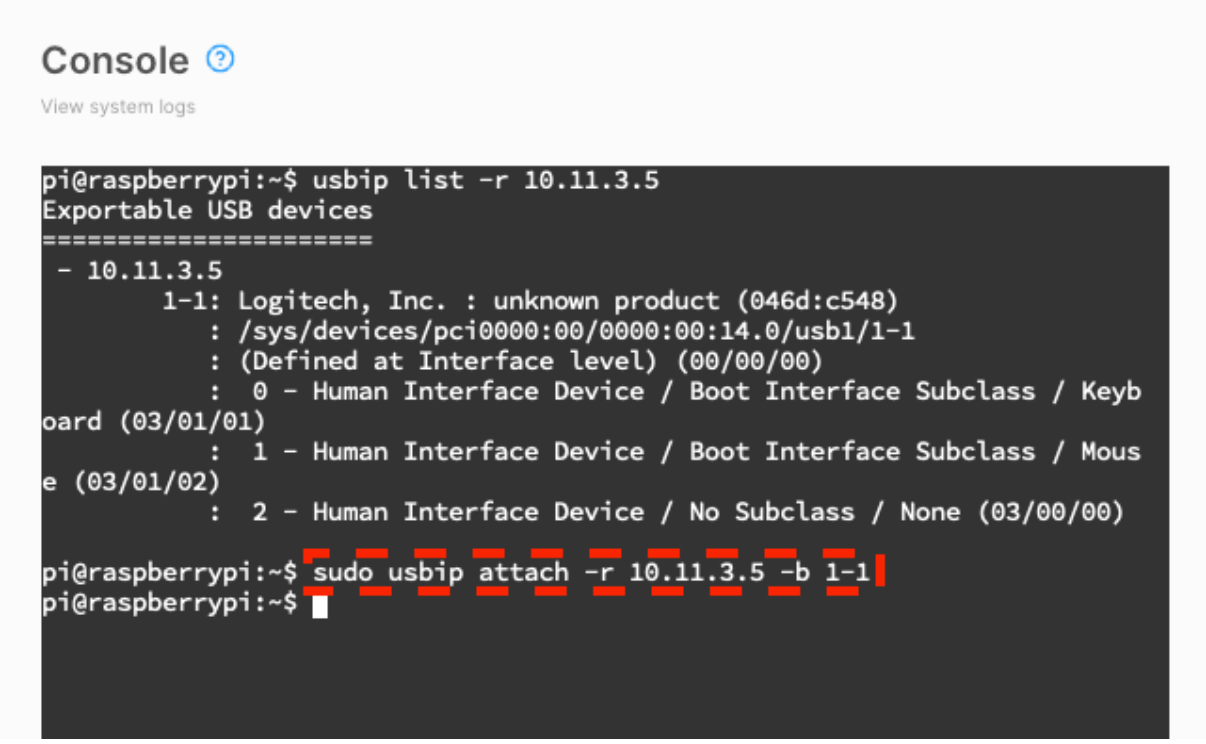


5. Attach to USB bus `1-1`, which corresponds to our mouse.

```
sudo usbip attach -r 10.11.3.5 -b 1-1
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-18: USBIP attach on RPi4 Desktop



6. You should now be able to control the cursor on the Raspberry Pi 4 Desktop.

5.2.5 (Optionally) Close the Connection When Finished

When you are finished, you can optionally close the connection by detaching on the client and unbinding the host's USB bus.

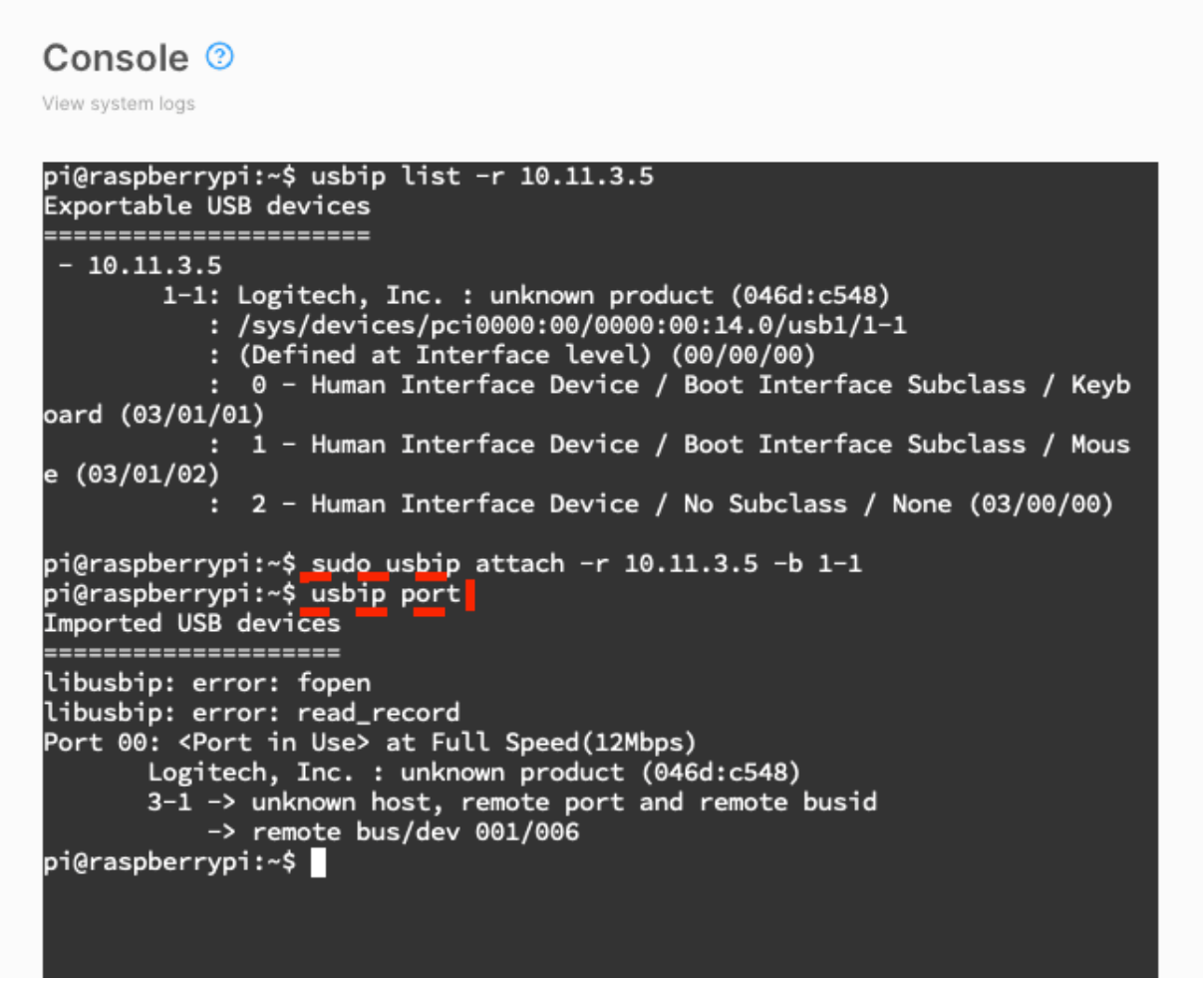
This process is necessary if you want to access the USB resource again on your host device.

1. On the USB/IP client, which in our case is the virtual Raspberry Pi 4 Desktop board, list the imported USB devices. Look for the appropriate port number.

```
usbip port
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-19: USBIP port on RPi4 Desktop

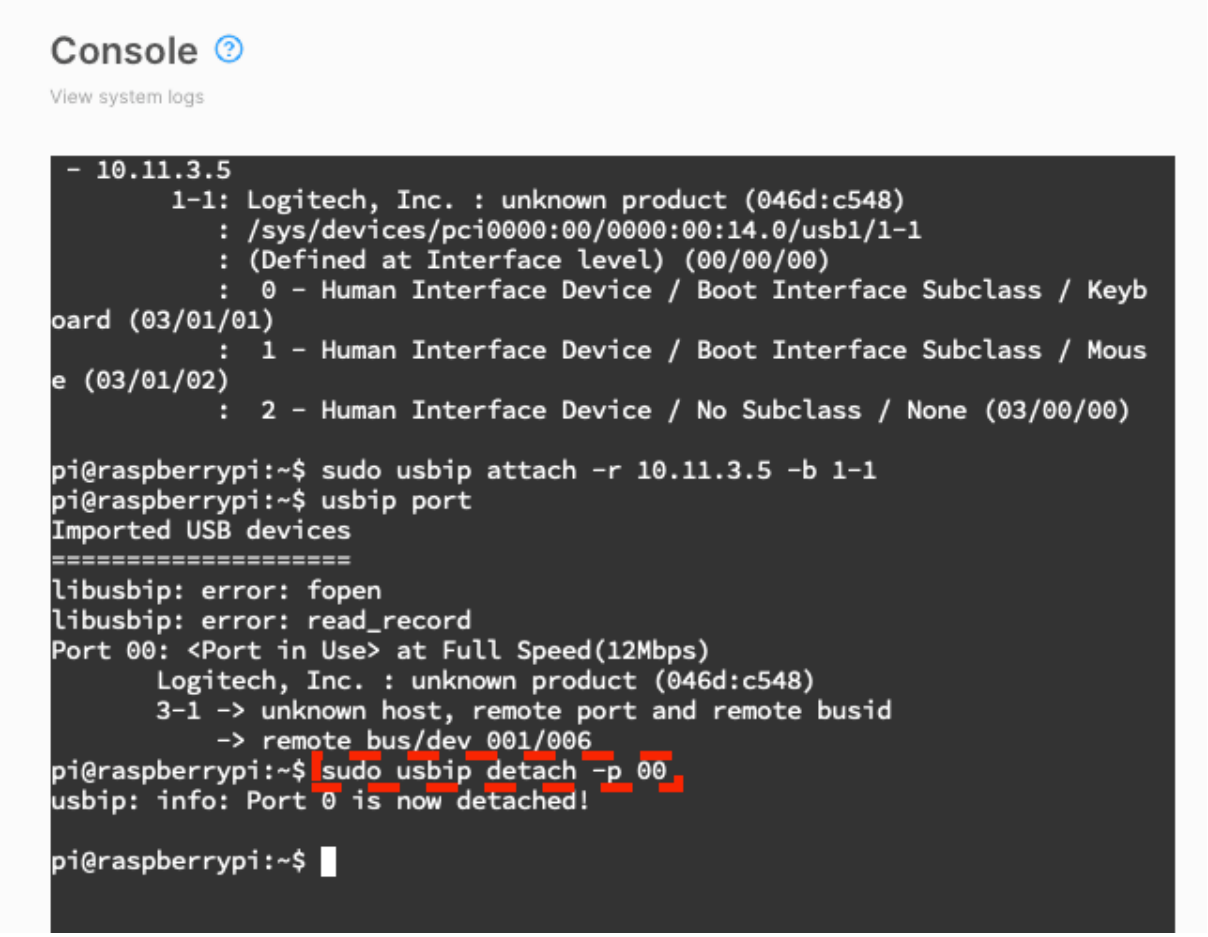


2. Detach port `00`, which corresponds to our mouse. Look for a response showing `detached`.

```
sudo usbip detach -p 00
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-20: USBIP detach on RPi4 Desktop

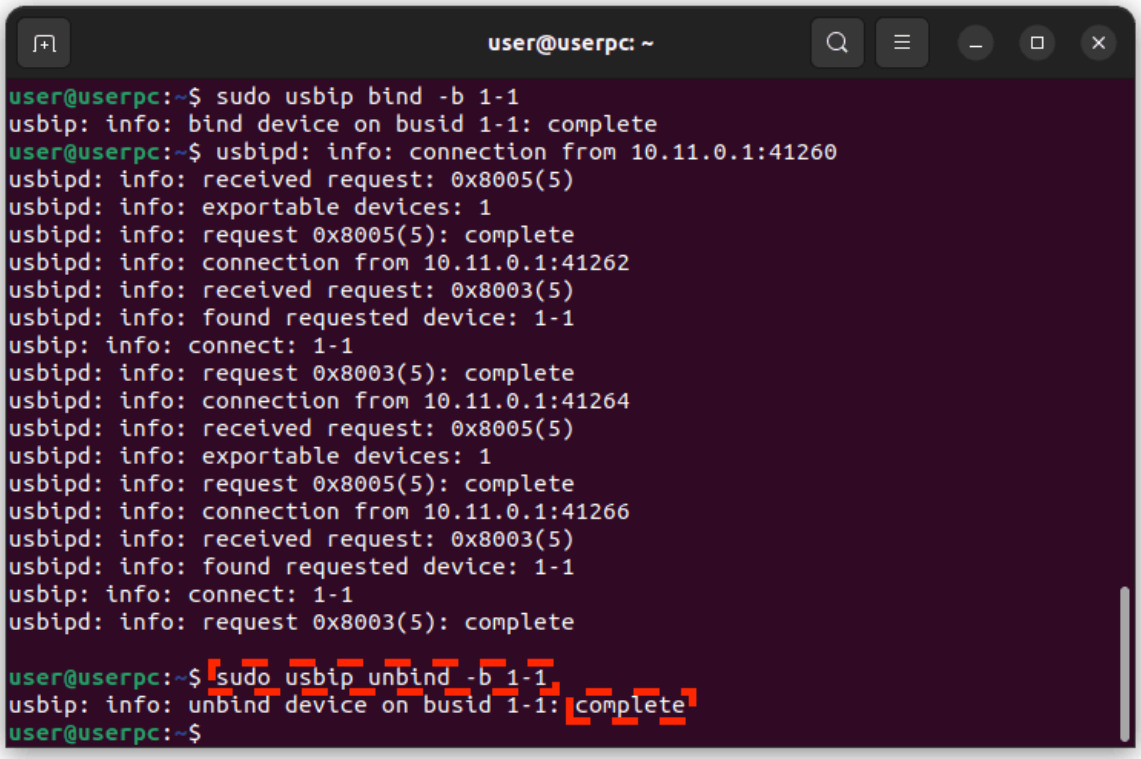


3. Switch over to your USB/IP host, which in our case is the Ubuntu machine, and unbind the appropriate USB bus. Look for a response showing `complete`.

```
usbip unbind -b 1-1
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-21: USBIP unbind



5.2.6 Use a Virtual STM32U5 Board as the USB/IP Host

We will be using firmware files that are offered as examples in STMicroelectronics's STM32CubeIDE tool.

You will need to build the `Ux_Device_HID.elf` and `Ux_Device_HID_CDC_ACM.elf` firmware files.

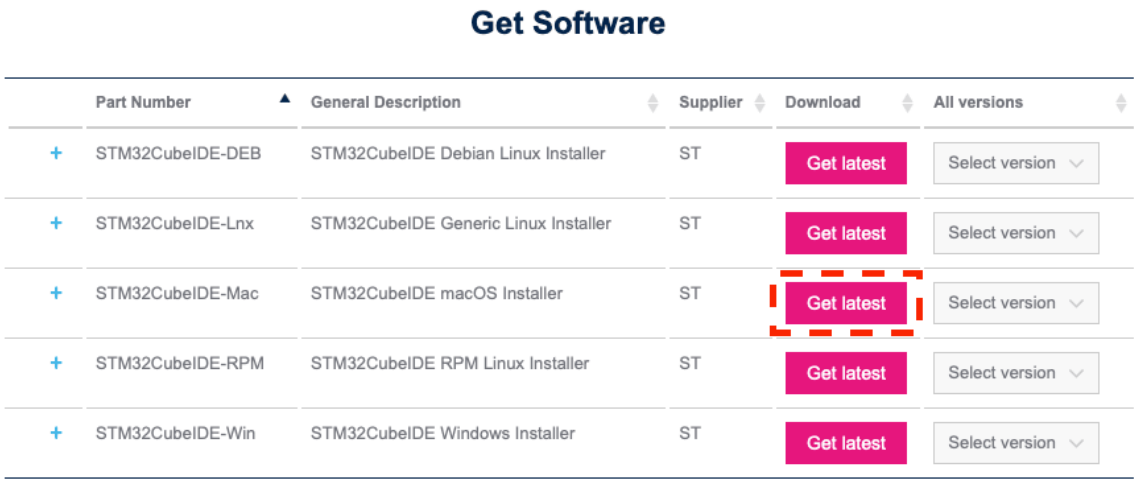
5.2.7 Build USB Firmware Files with the STM32CubeIDE Tool

Follow this process to build the required `.elf` custom firmware files using free tools from STMicroelectronics.

1. Install the STM32CubeIDE tool. In our example, we will use the MacOS version.

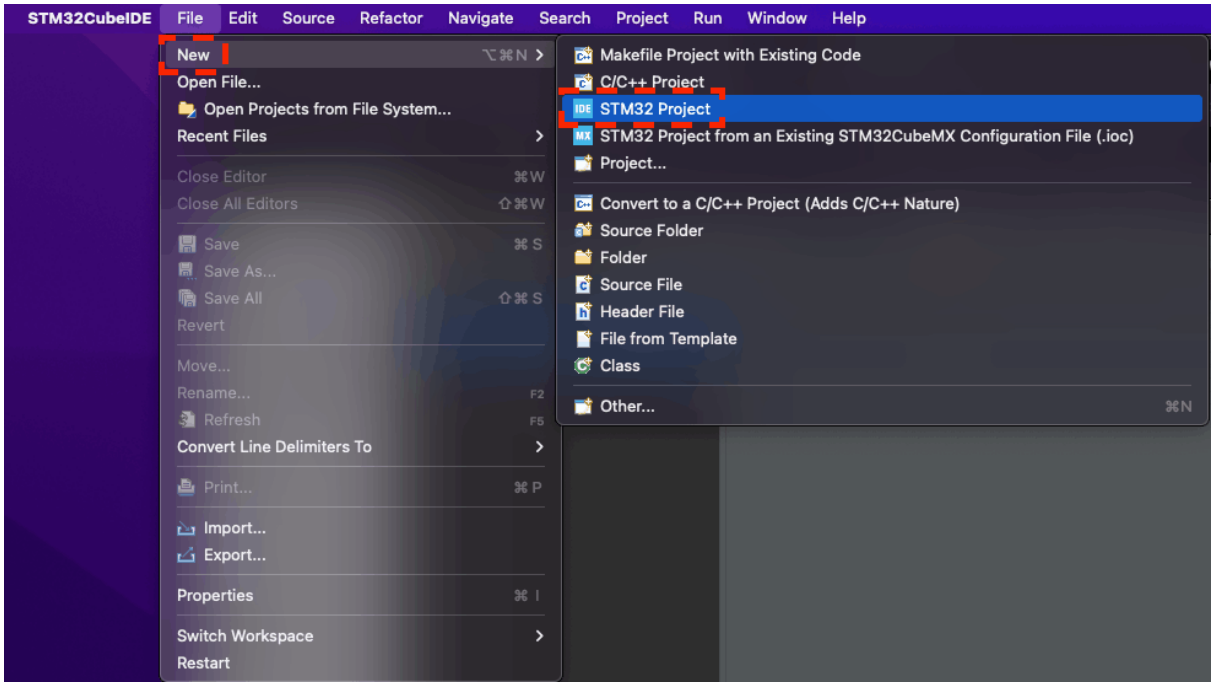
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-22: Download CubeIDE



2. Create a new project by clicking New -> STM32 Project from the File menu.

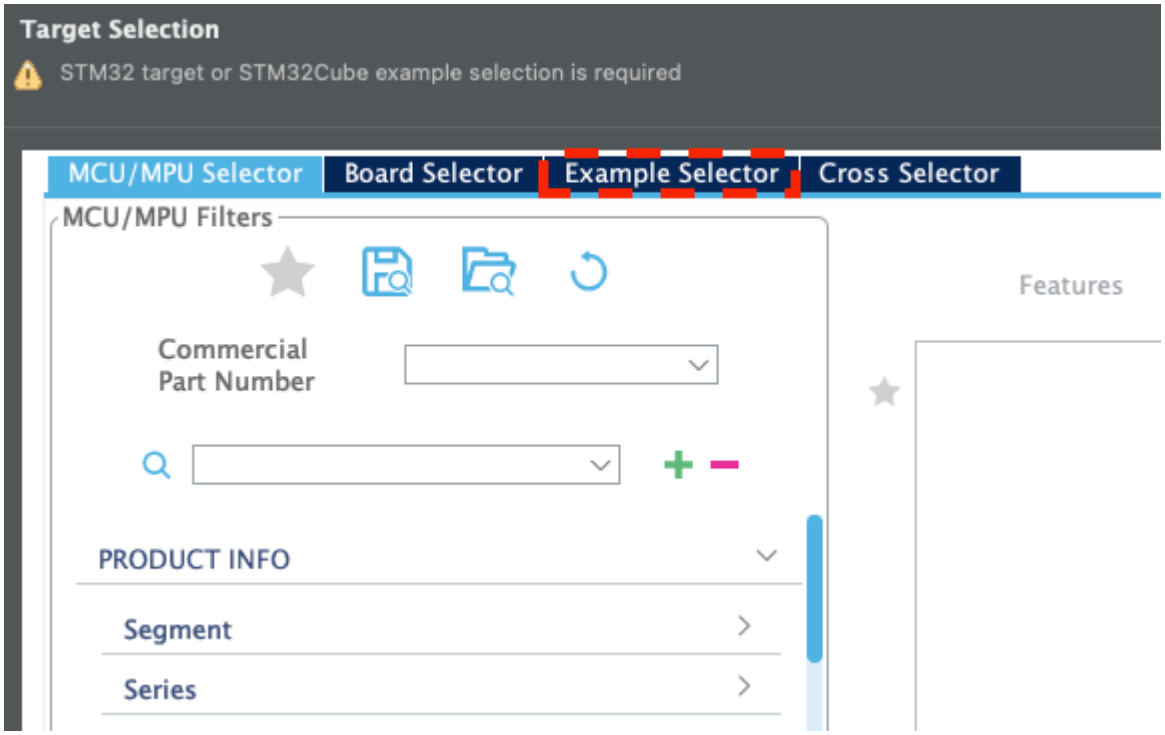
Figure 5-23: New Project



3. When the STM32 Project window appears, click on Example Selector.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

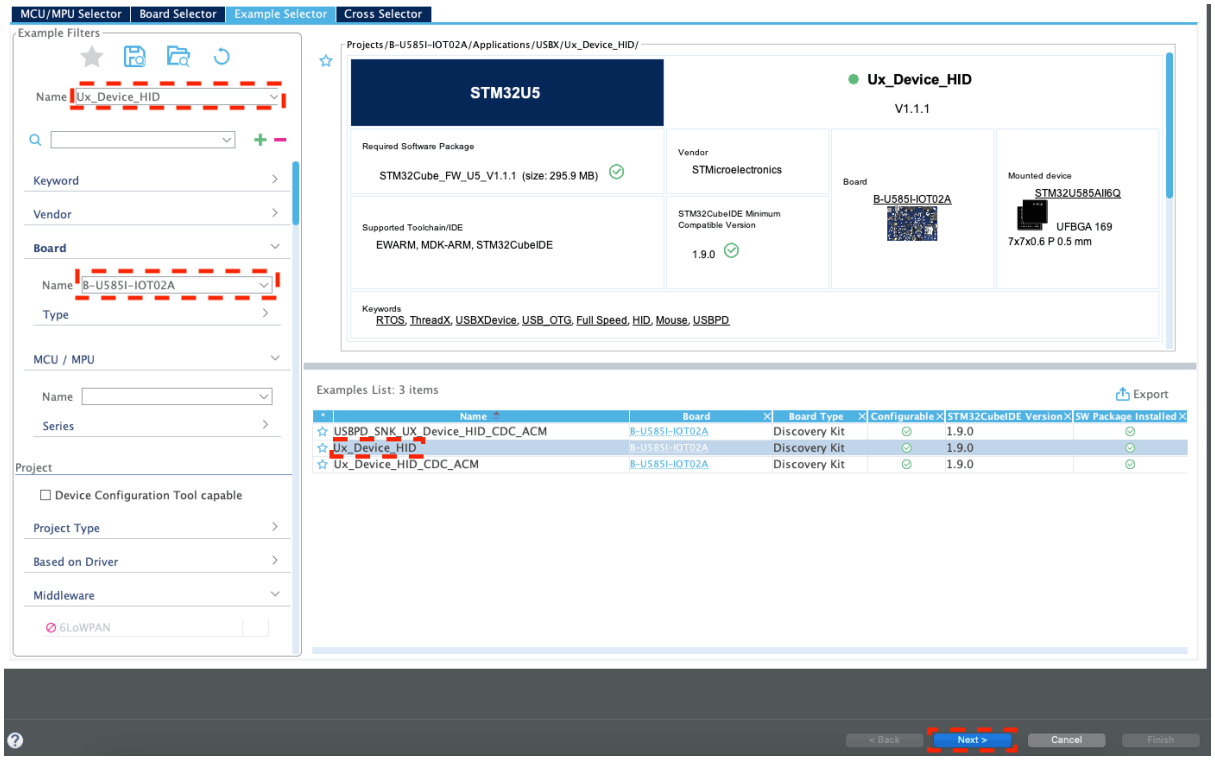
Figure 5-24: Example Selector



4. Choose the example name `Ux_Device_HID` and the board name `B-U585I-IOT02A`, click on the example you want to build, then click Next.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

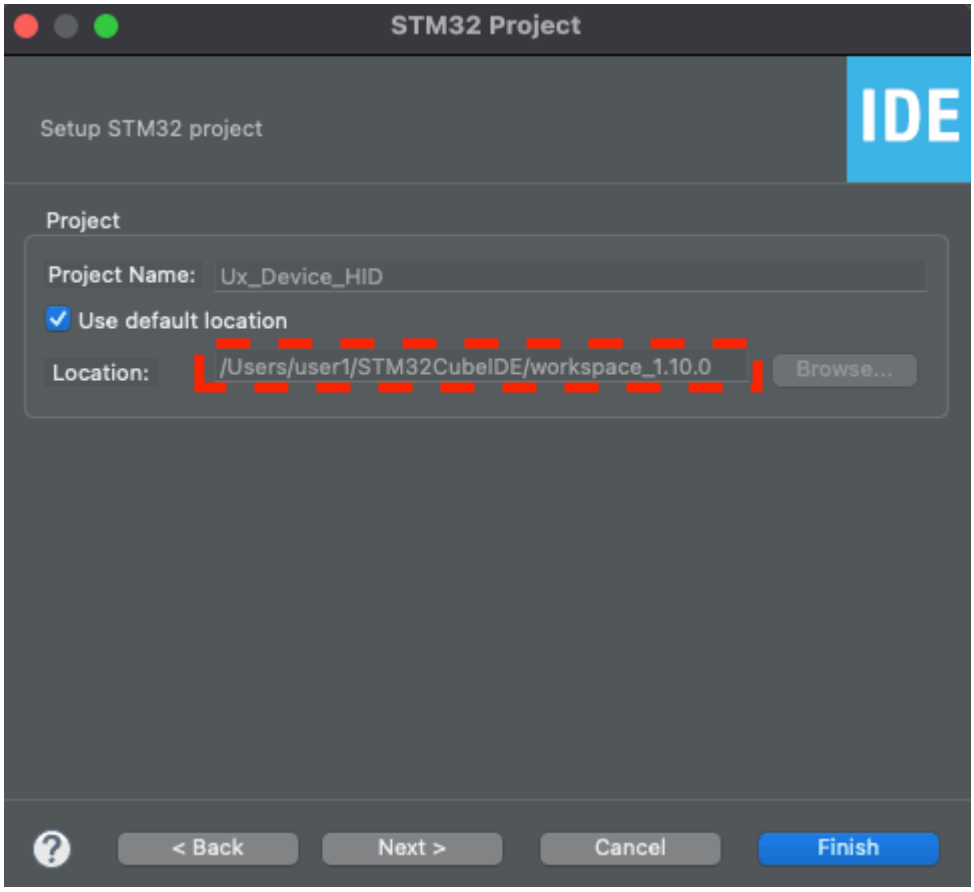
Figure 5-25: Choose Example



5. Make note of your project name and path to your workspace then click Finish.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

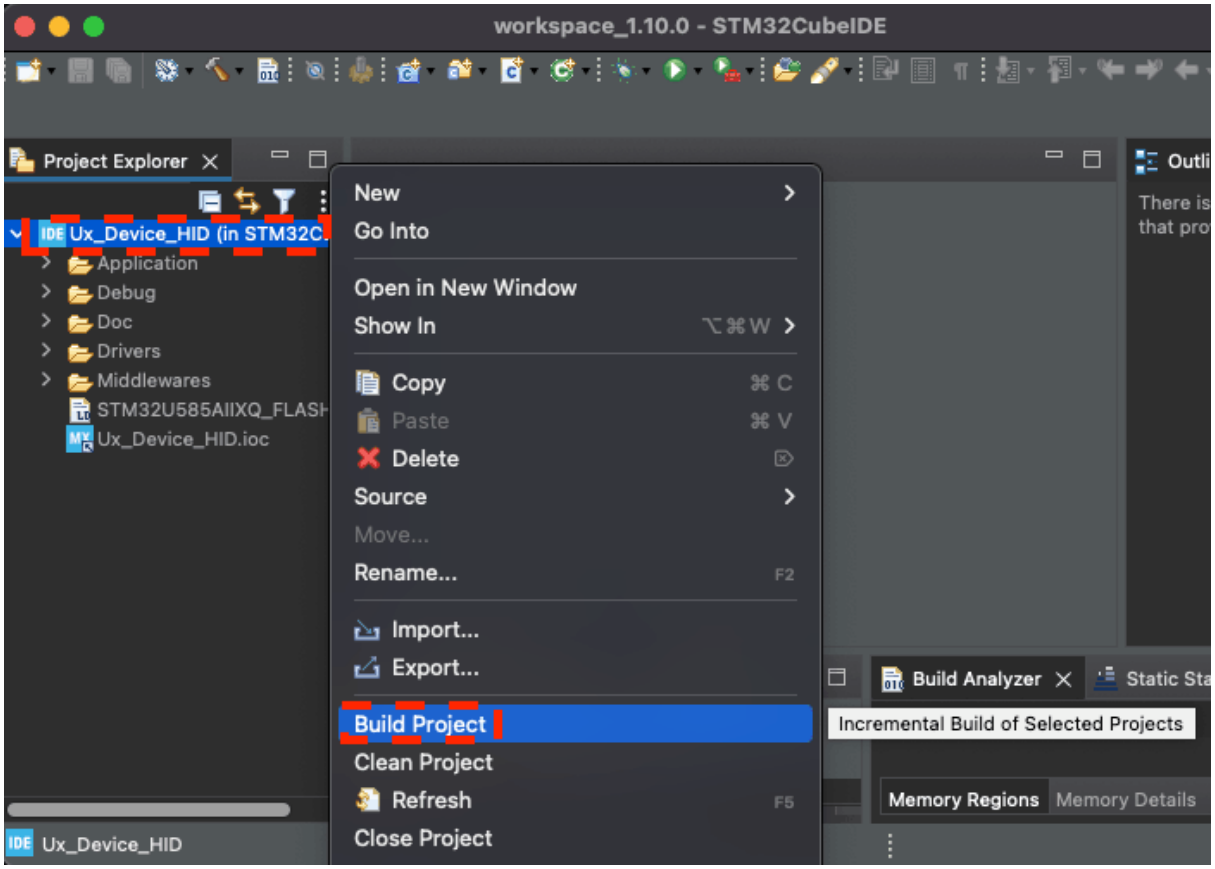
Figure 5-26: Finish Project



6. When the Project Explorer opens, right-click on your project name and select Build Project.

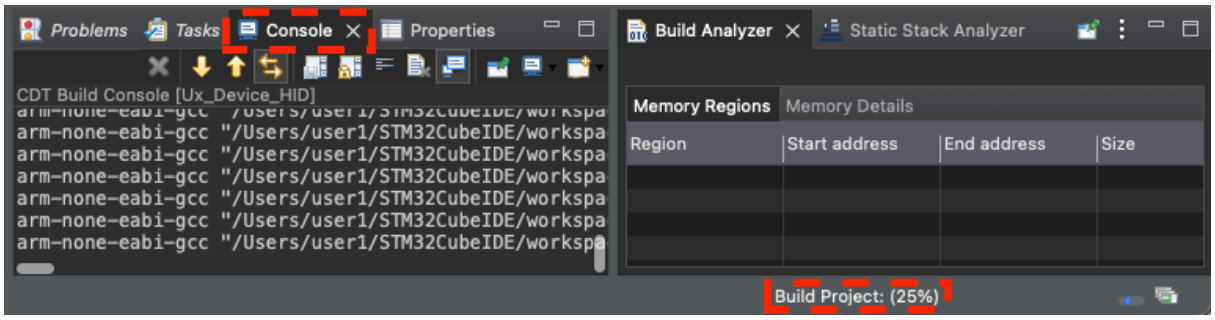
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-27: Build Project



7. While the project is building, you should see a progress message towards the bottom right. Open the Console if you prefer to follow the progress.

Figure 5-28: Build Progress

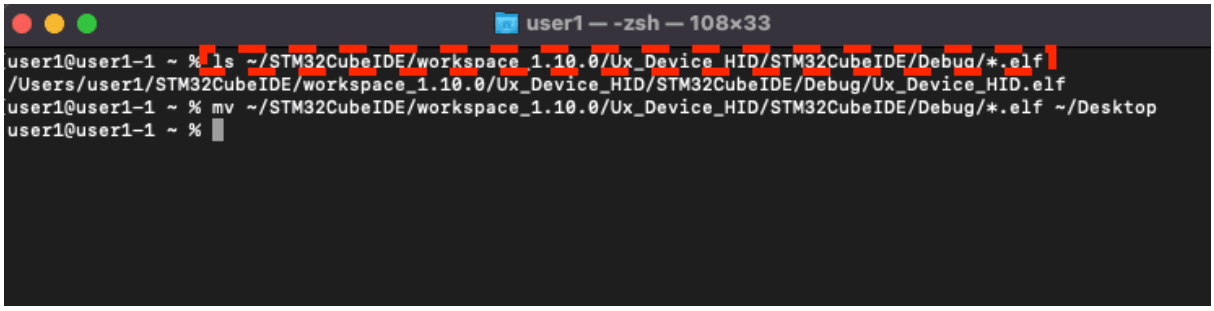


8. The `.elf` firmware file will be created inside your project directory, which is inside `/<path_to_workspace>/<project_name>/STM32CubeIDE/Debug`.

```
ls ~/STM32CubeIDE/workspace_1.10.0/Ux_Device_DFU/STM32CubeIDE/Debug/*.elf
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-29: List Firmware File



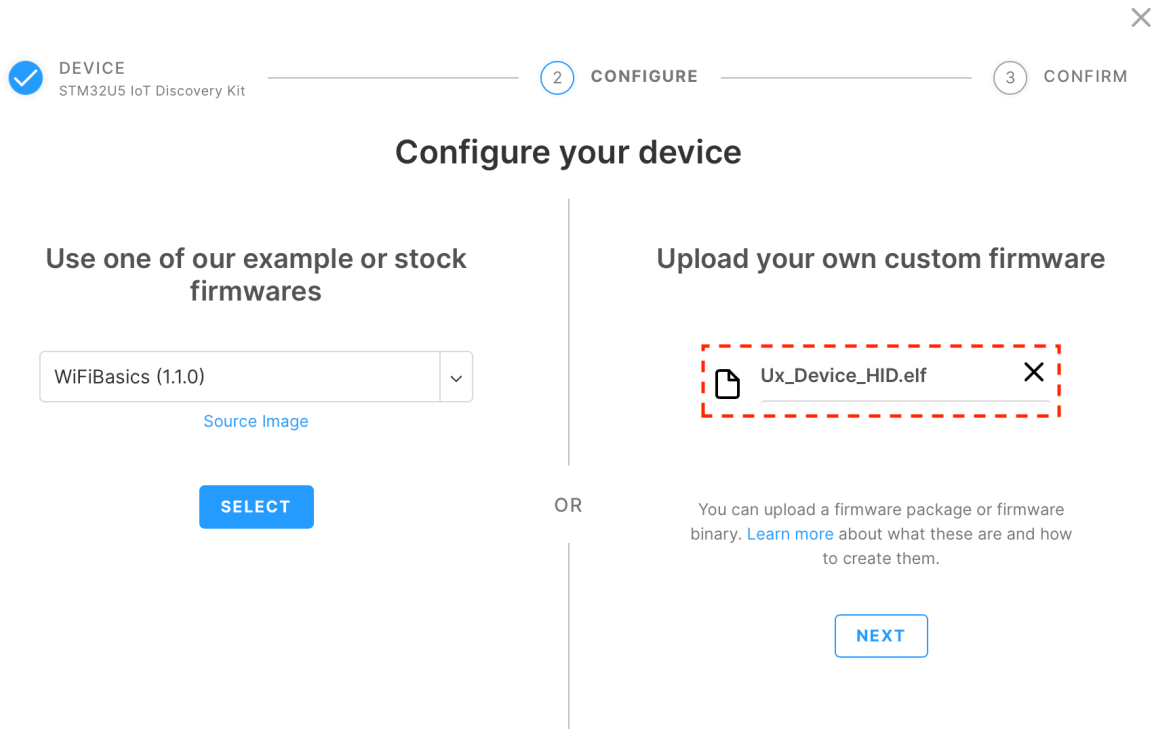
9. The firmware file is ready to be uploaded to your AVH virtual STM32U5 board.

5.2.8 Attach a Virtual USB Peripheral to a Virtual Raspberry Pi 4 Board

To attach a virtual joystick to the Raspberry Pi 4:

1. Create a new virtual STM32U5 board and manually upload the `Ux_Device_HID.elf` custom firmware you created.

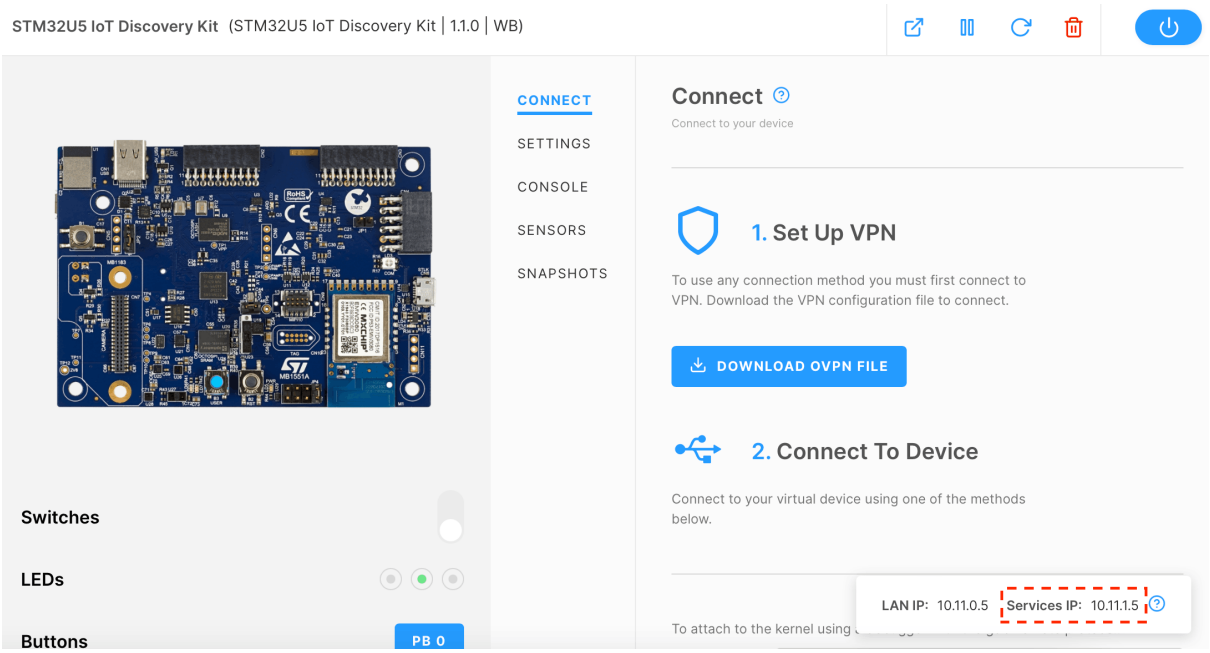
Figure 5-30: Upload joystick firmware



2. Make note of the Services IP address on the Connect tab. In our example, we will use the Services IP `10.11.1.5`.

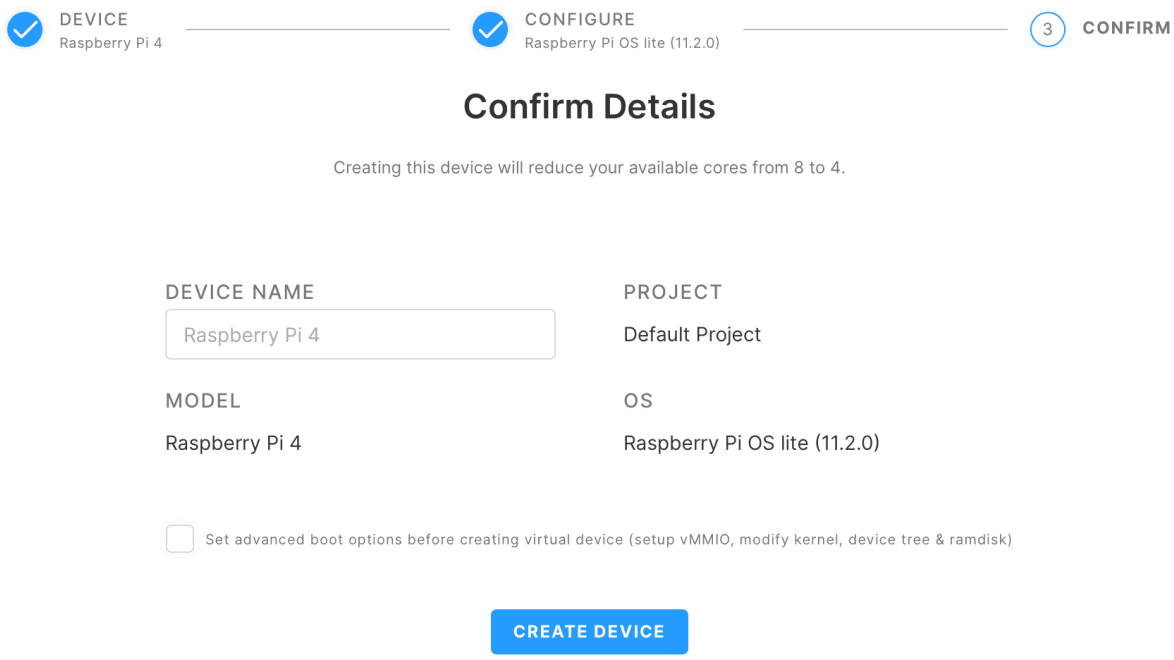
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-31: Joystick Services IP



3. Create a new Raspberry Pi 4 virtual device using any version of the firmware. In our example, we will use the lite version of firmware `11.2.0`.

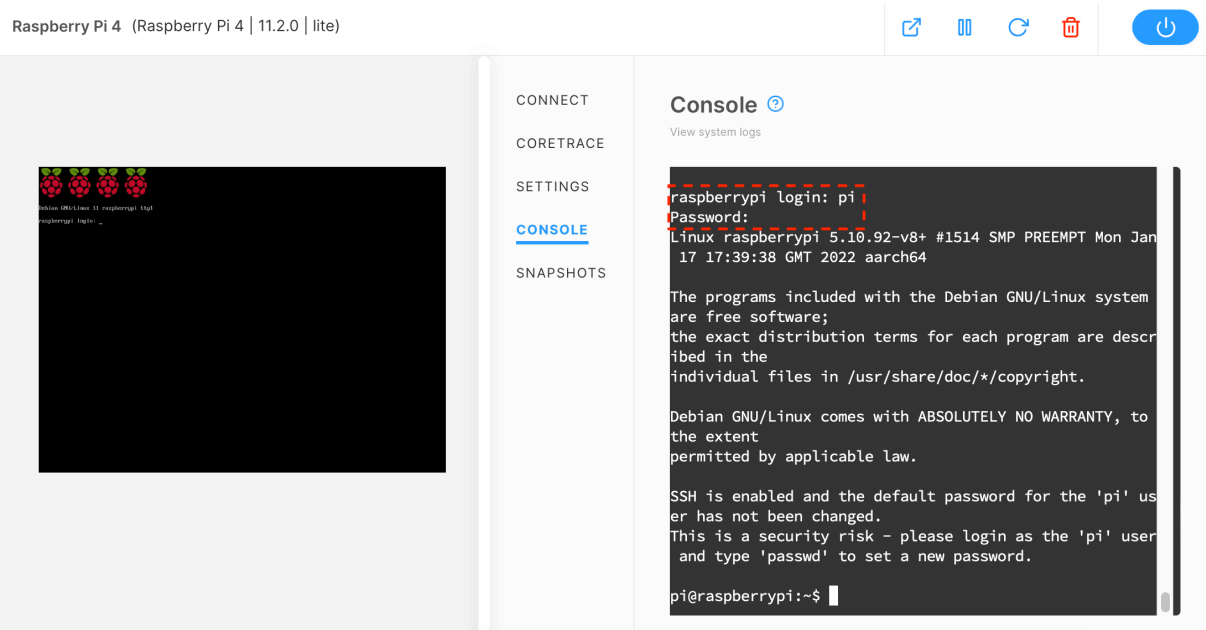
Figure 5-32: Create new Pi



4. Log in as the root user with the username `pi` and password `raspberry`.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-33: Login to Pi as root

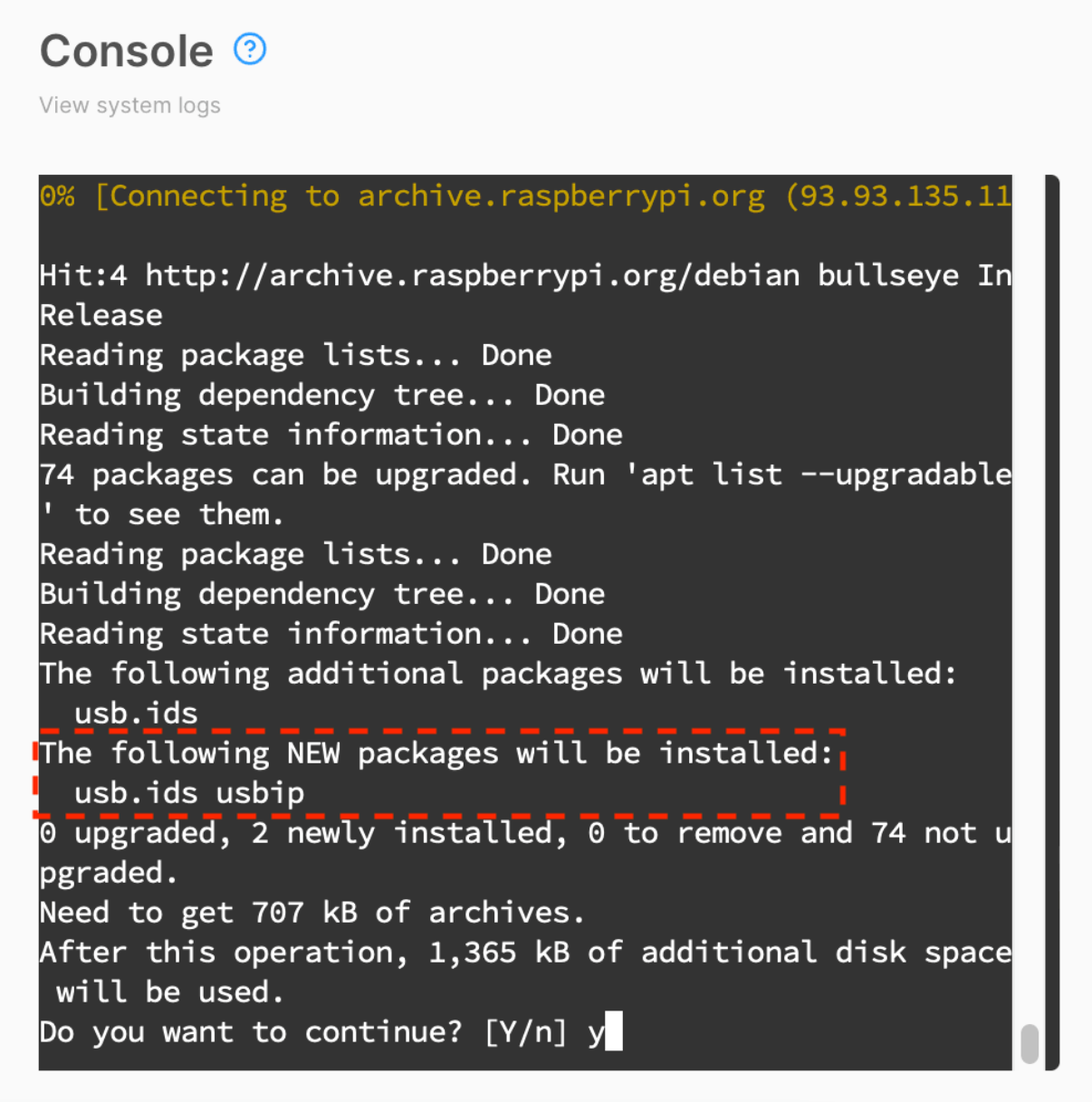


5. Install the USB/IP package.

```
sudo apt update && sudo apt install -y usbip
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-34: Install usbip

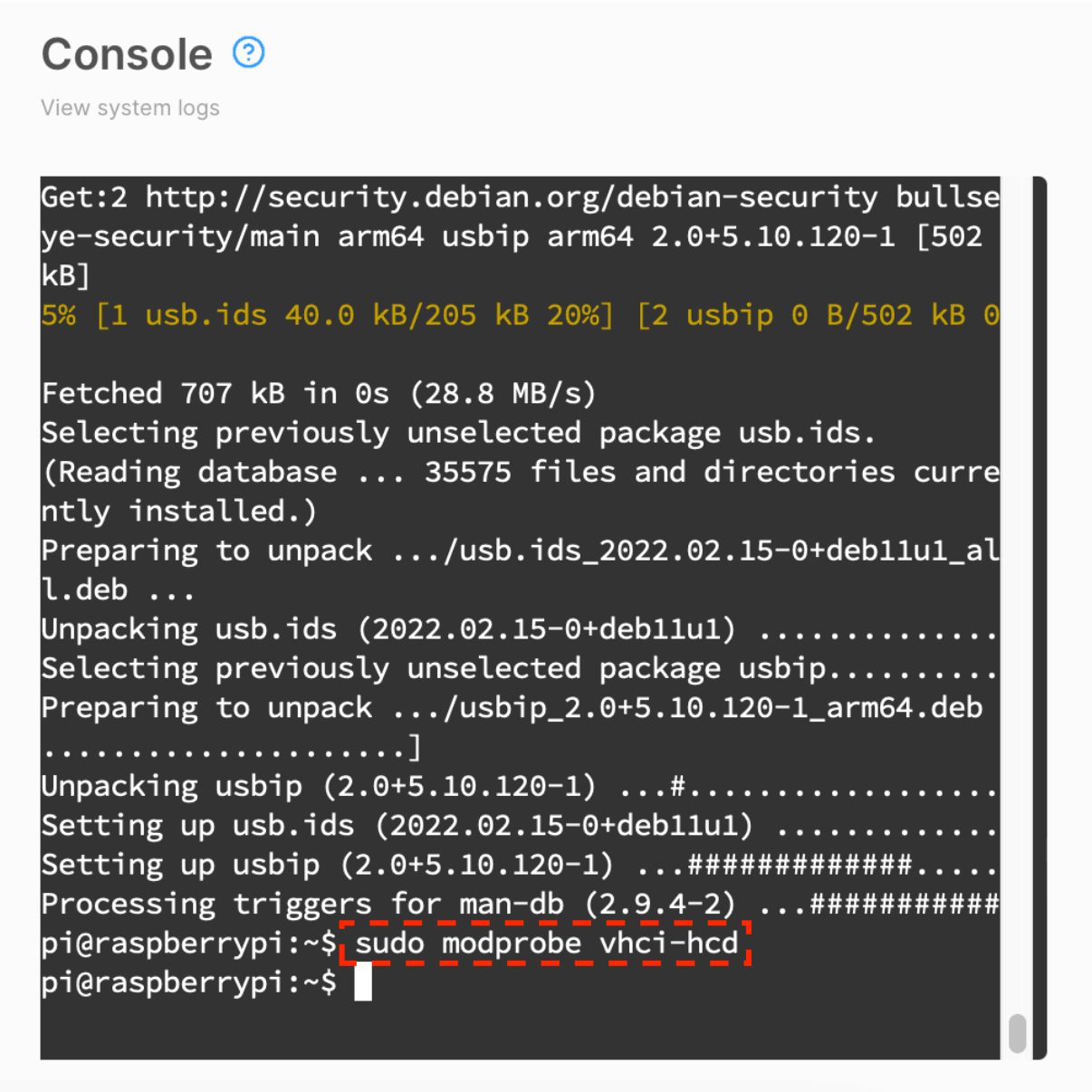


6. Load the Virtual Host Controller Interface kernel module.

```
sudo modprobe vhci-hcd
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-35: Load kernel module

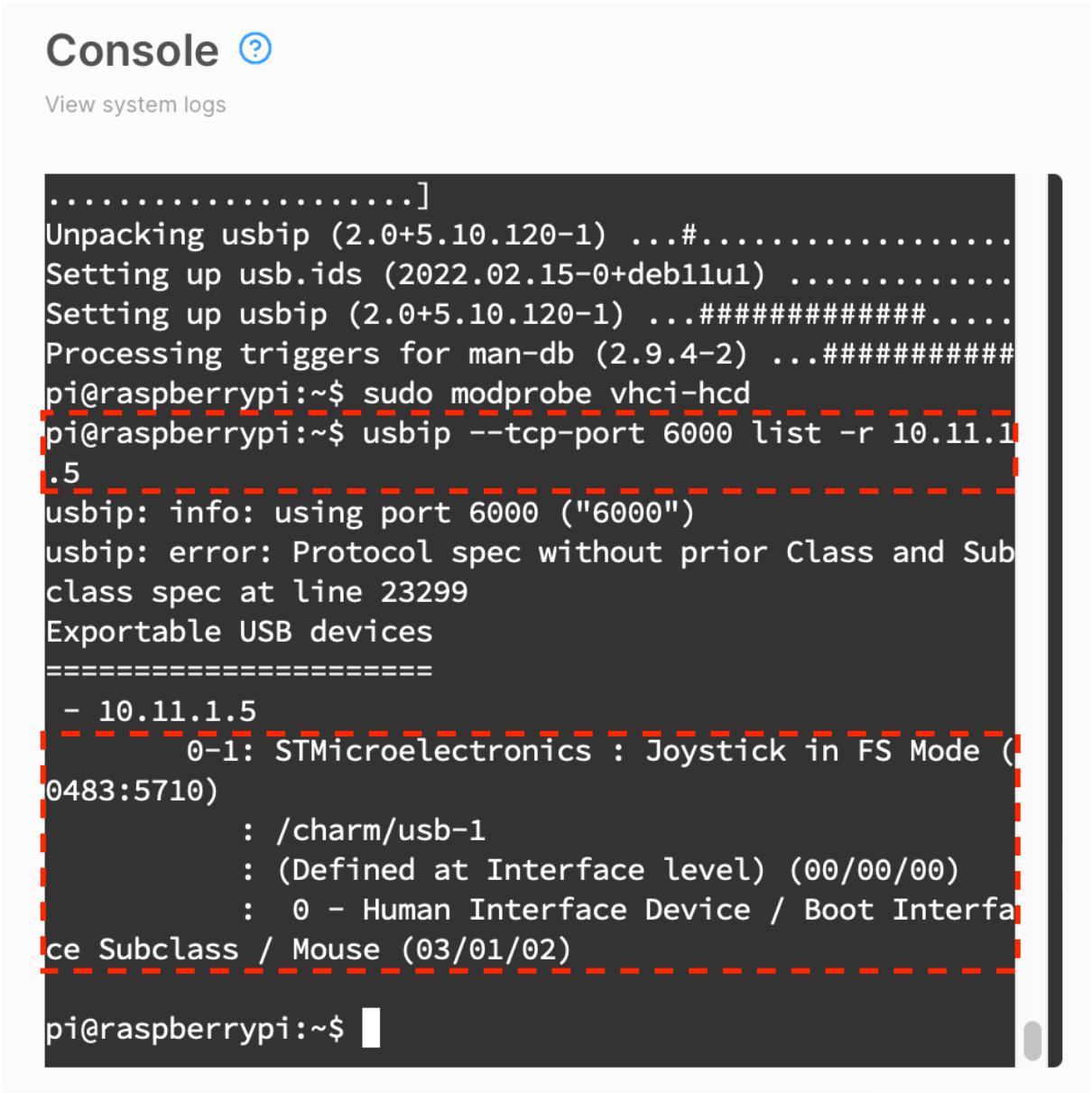


7. List the STM32U5's exportable USB devices using the Services IP and port 6000.

```
usbip --tcp-port 6000 list -r 10.11.1.5
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-36: List USB devices

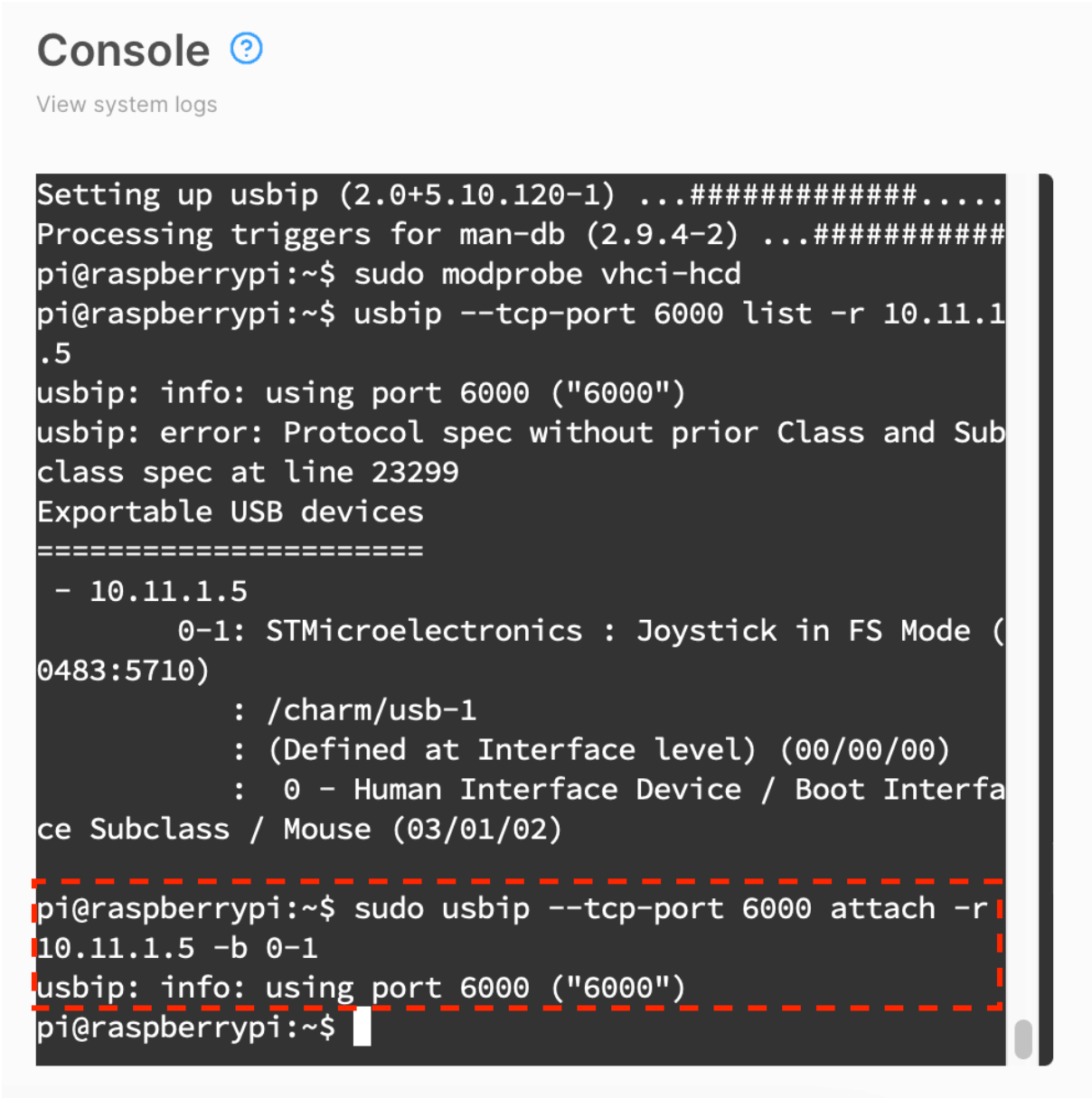


8. Attach to USB bus 0-1, which corresponds to the joystick peripheral. (Don't forget to `sudo` or the command will fail.)

```
sudo usbip --tcp-port 6000 attach -r 10.11.3.3 -b 1-2
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

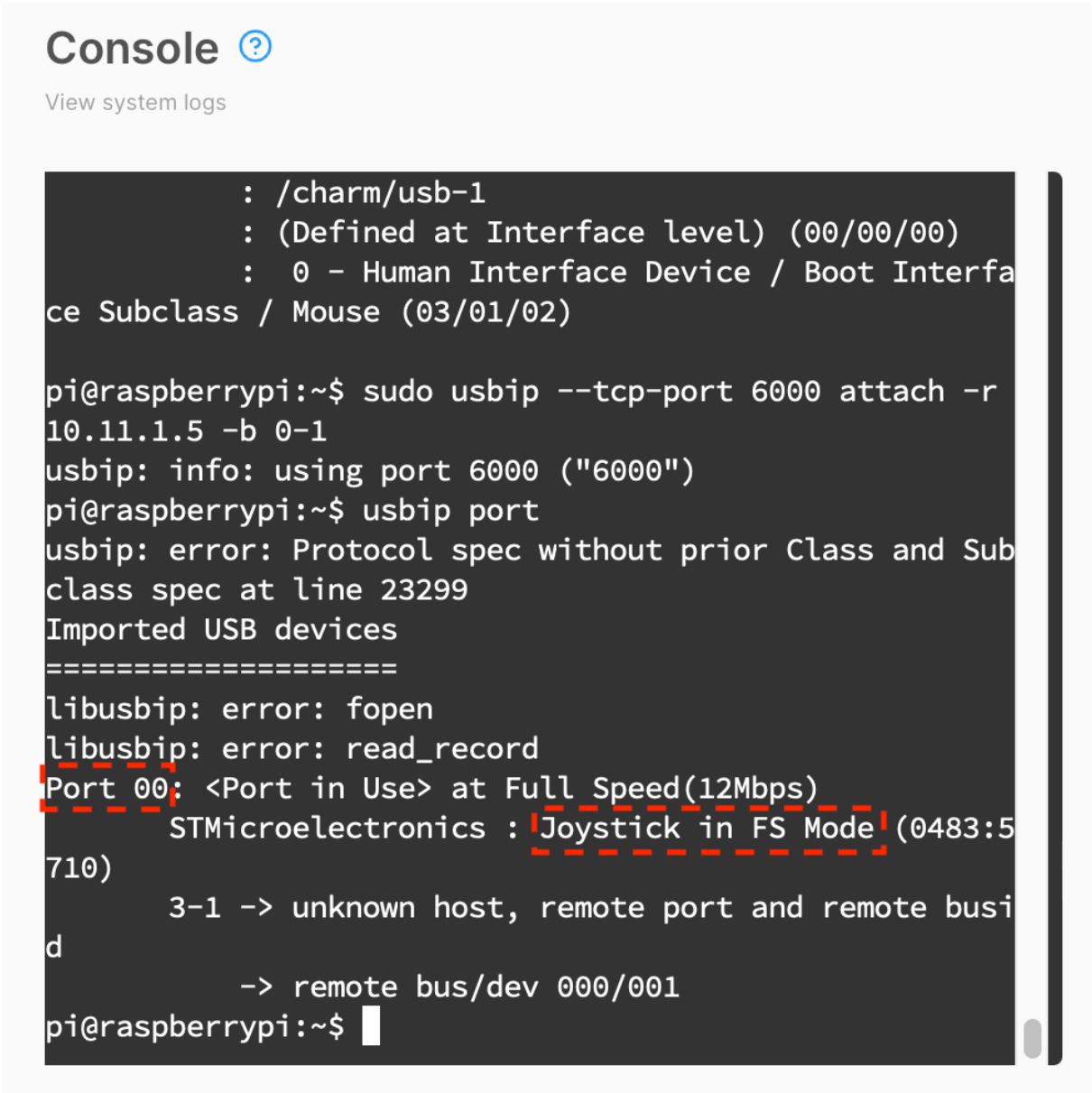
Figure 5-37: Attach to USB bus 0-1



9. The STM32U5's virtual joystick is now attached to your Raspberry Pi 4 virtual board. Confirm the device is attached using `usbip port`. Make note that the device is using port `00`.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-38: Confirm the device is attached

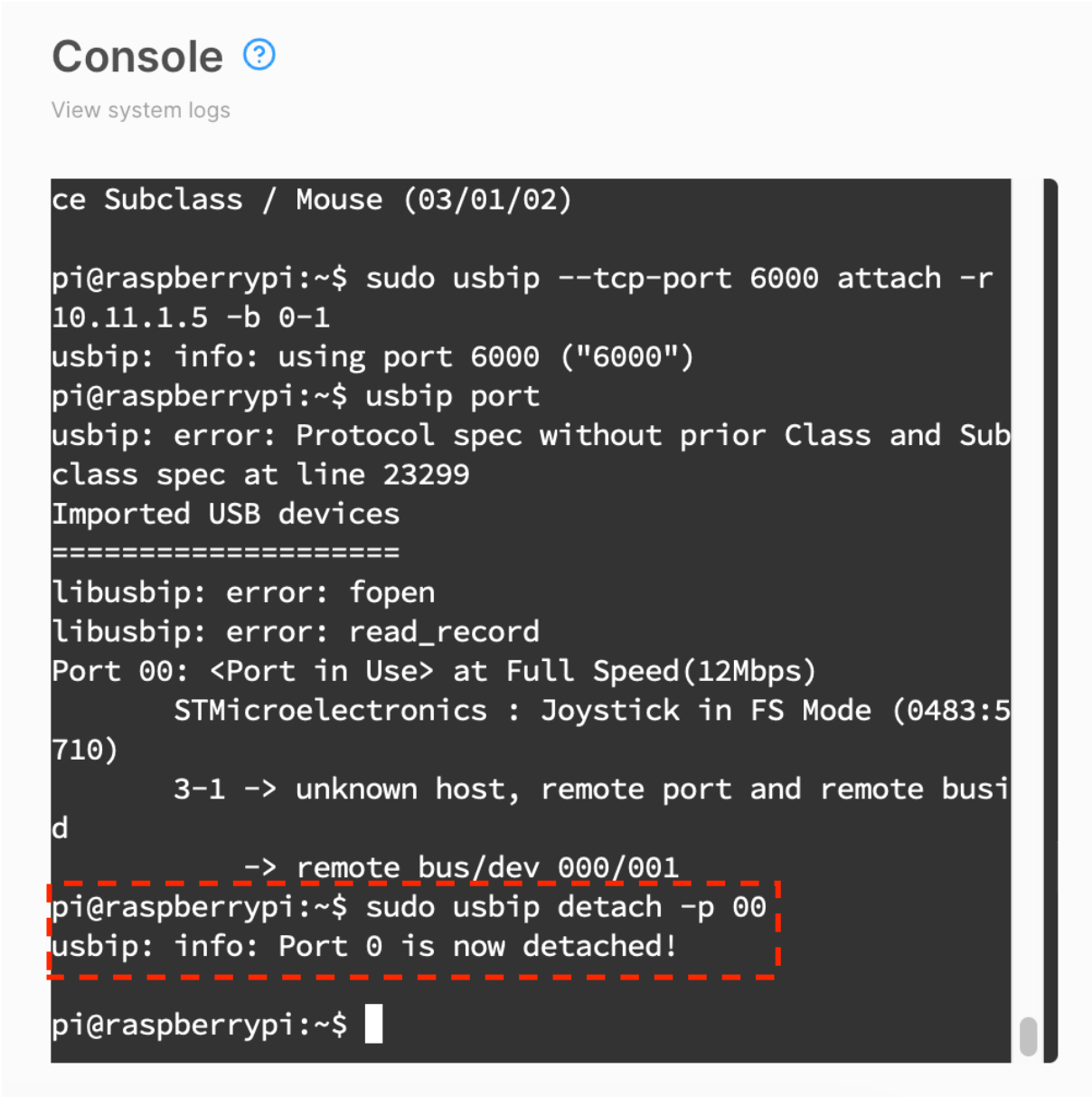


10. When you are finished, detach the peripheral from port `00`.

```
sudo usbip detach -p 00
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-39: Detach the peripheral



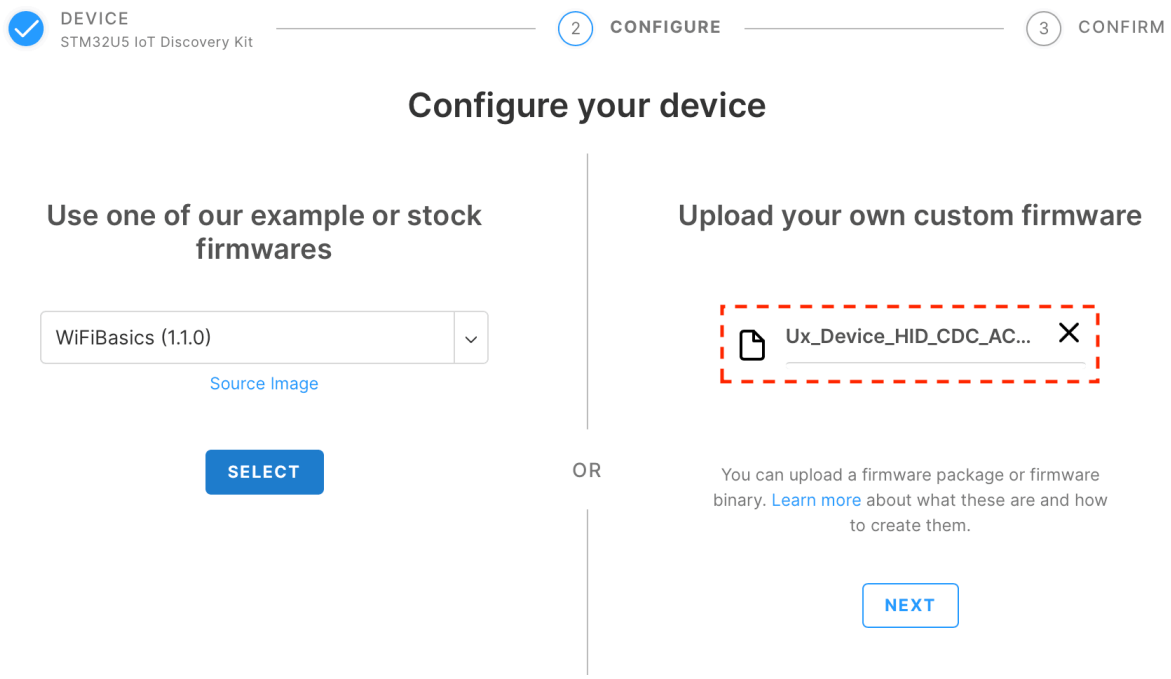
5.2.9 Attach a Virtual USB Peripheral to a Physical Computer Using a VPN Connection

In this example, we will use a laptop running Ubuntu 20.04. We will also use Ux_Device_HID_CDC_ACM.elf, the "Mini LED Display" firmware file we created in the Build USB Firmware Files with the STM32CubeIDE Tool step.

1. Create a new STM32U5 virtual device and manually upload the `Ux_Device_HID_CDC_ACM.elf` firmware you created in the STM32CubeIDE tool.

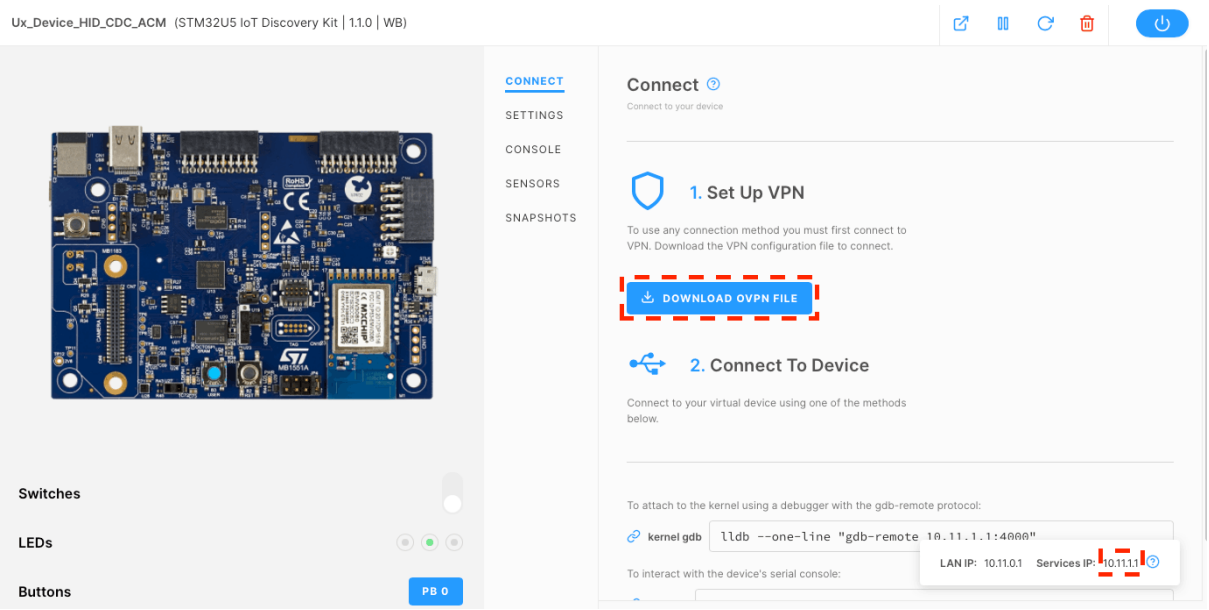
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-40: Upload LED firmware



2. Download the `.ovpn` file and make note of the Services IP address on the Connect tab. In our example, the Services IP is `10.11.1.1`.

Figure 5-41: STM32U5 VPN and Services IP



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

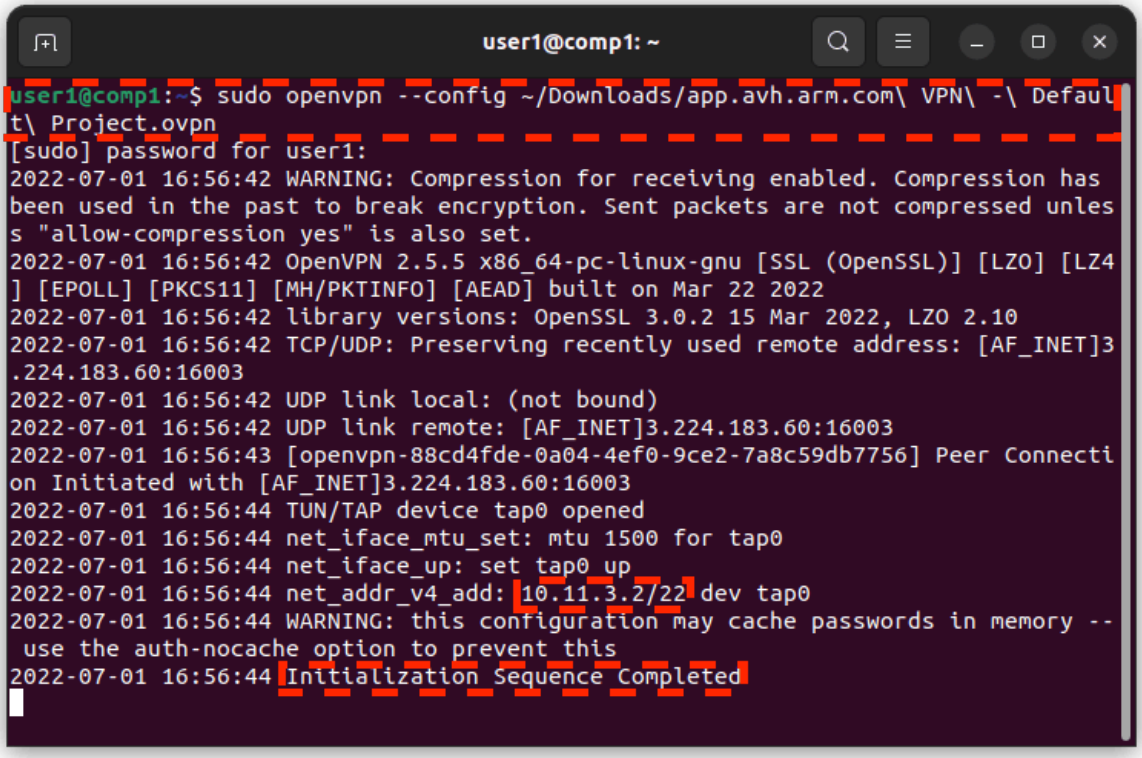
3. On your Linux client machine, install the required packages.

```
sudo apt update && sudo apt install -y linux-tools-$(uname -r) hwdata
```

4. Connect to the AVH VPN using the `.ovpn` file you downloaded.

```
sudo openvpn --config /Downloads/app.avh.arm.com\ VPN\ -\ Default\ Project.ovpn
```

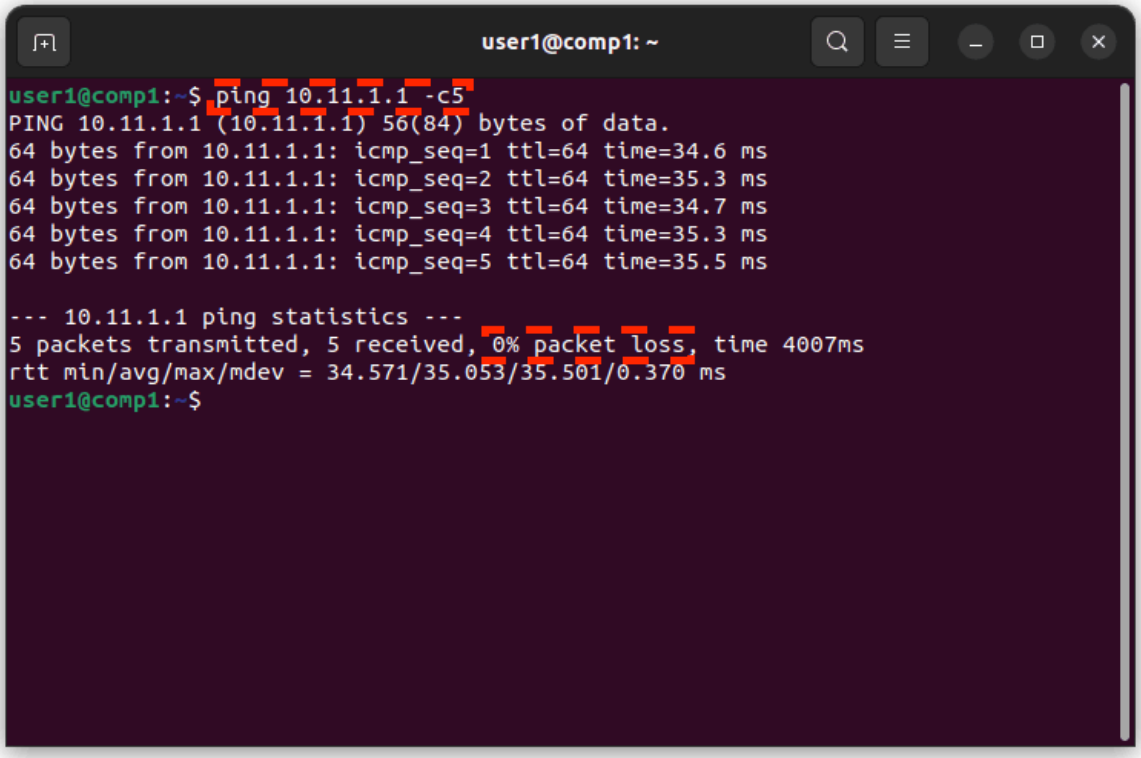
Figure 5-42: OpenVPN



5. Open a second Terminal window. Double-check your VPN connection by pinging the Services IP for your USB/IP host.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-43: Ping Test

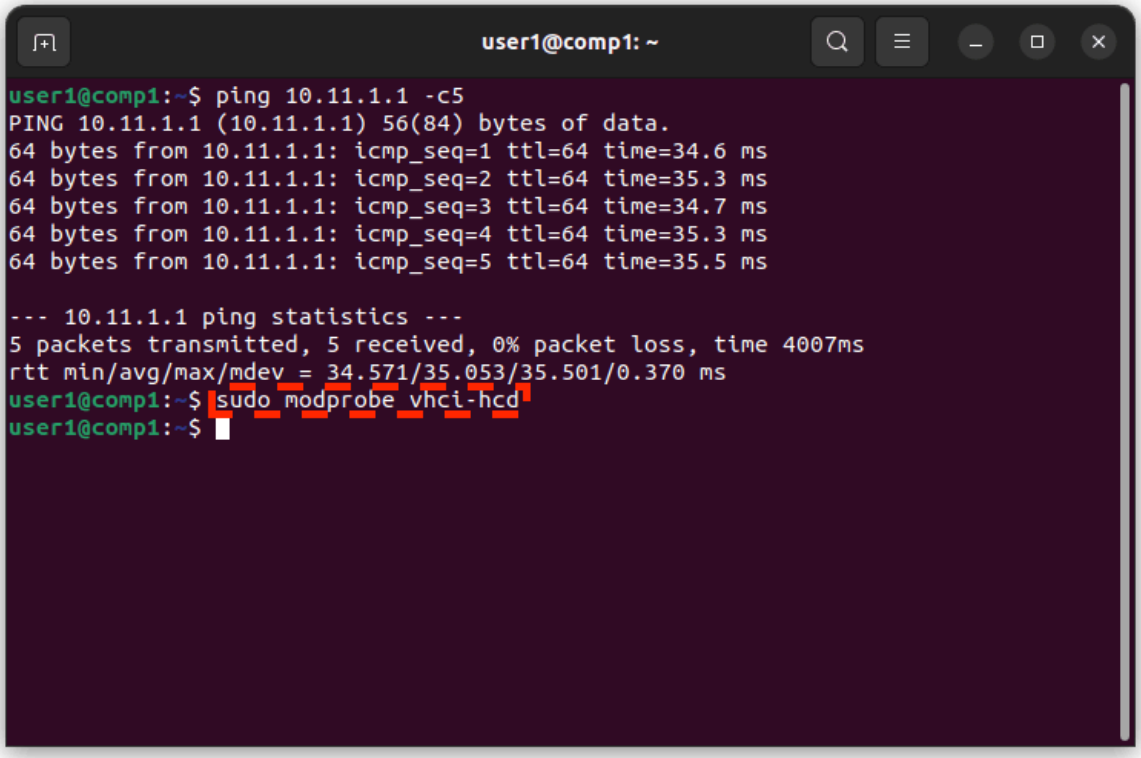


6. Load the Virtual Host Controller Interface kernel module.

```
sudo modprobe vhci-hcd
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-44: Load the kernel module

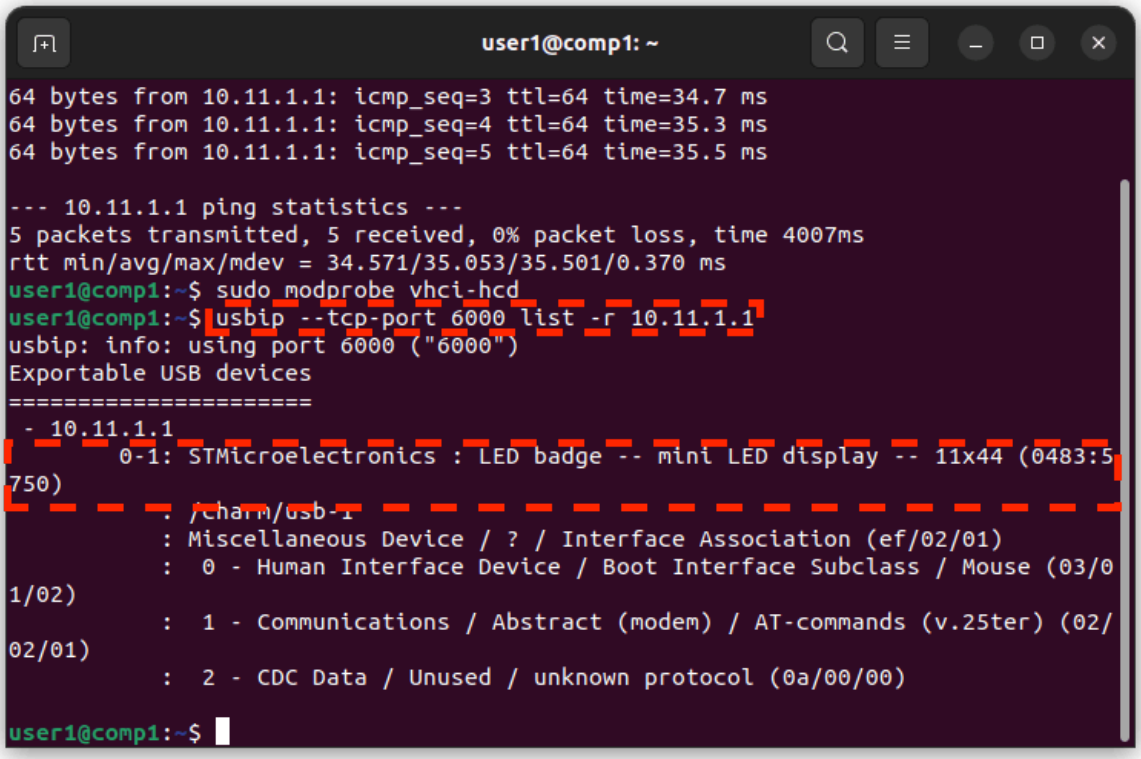


7. List the exportable STM32U5 USB devices.

```
usbip --tcp-port 6000 list -r 10.11.1.1
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-45: List USB Devices



8. Attach to USB bus 0-1, which corresponds to the LED display.

```
sudo usbip --tcp-port 6000 attach -r 10.11.1.1 -b 0-1
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-46: Attach to USB Device

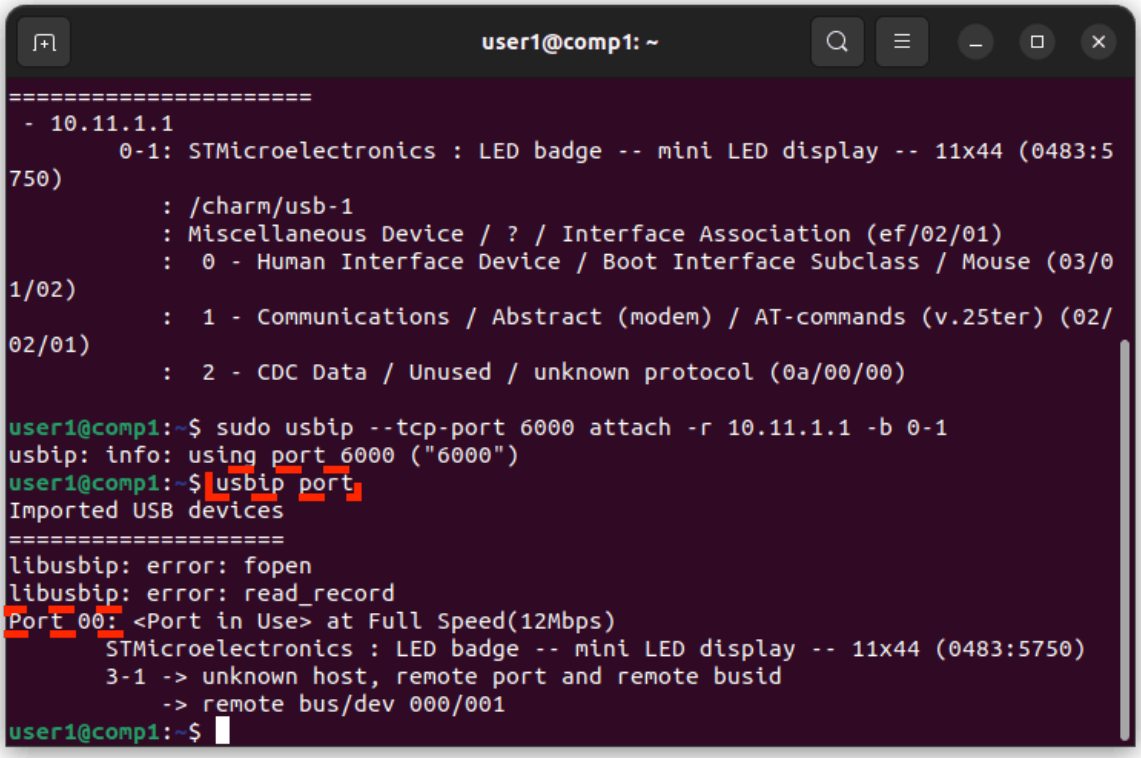
```
user1@comp1: ~
64 bytes from 10.11.1.1: icmp_seq=5 ttl=64 time=34.7 ms

--- 10.11.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 34.660/36.928/40.865/2.114 ms
user1@comp1:~$ sudo modprobe vhci-hcd
user1@comp1:~$ usbip --tcp-port 6000 list -r 10.11.1.1
usbip: info: using port 6000 ("6000")
Exportable USB devices
======================
 - 10.11.1.1
        0-1: STMicroelectronics : LED badge -- mini LED display -- 11x44 (0483:5
750)
           : /charm/usb-1
           : Miscellaneous Device / ? / Interface Association (ef/02/01)
           :  0 - Human Interface Device / Boot Interface Subclass / Mouse (03/0
1/02)
           :  1 - Communications / Abstract (modem) / AT-commands (v.25ter) (02/
02/01)
           :  2 - CDC Data / Unused / unknown protocol (0a/00/00)

user1@comp1:~$ sudo usbip --tcp-port 6000 attach -r 10.11.1.1 -b 0-1
usbip: info: using port 6000 ("6000")
user1@comp1:~$
```

9. List the Imported USB devices to confirm the attach command was successful.

```
usbip port
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-47: Imported USB Device

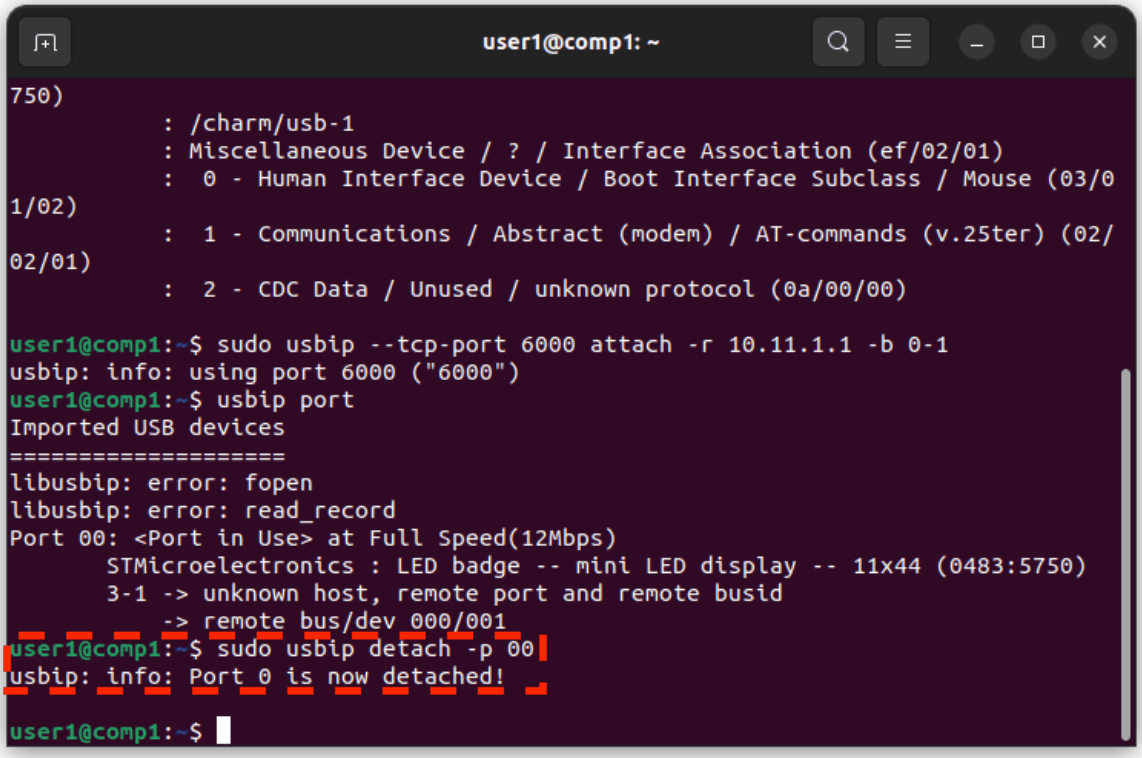


10. When you are finished, detach from port `00`.

```
sudo usbip detach -p 00
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-48: Detach USB Device



5.2.10 Attach a Virtual USB Peripheral to a Physical Computer Using SSH Port Forwarding

In this example, we will use a laptop running Ubuntu 22.04. We will also use `Ux_Device_HID_CDC.elf`, the "Joystick in FS Mode" firmware file we created in the Build USB Firmware Files with the STM32CubeIDE Tool step.

1. Upload the `Ux_Device_HID_CDC.elf` firmware file and start the STM device. Make note of the device's "Services IP".
2. Next, we will find your project's unique identifier. The first option is to use the Get Projects API command. Alternatively, you can do the following.
   a. Boot up a virtual Raspberry Pi 4 board.
   b. Under the CONNECT tab, find to the Quick Connect SSH command
   c. Copy the part of the command showing your project's unique identifier `<project identifier>@proxy.app.avh.arm.com`.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. On your Ubuntu computer, generate a public/private key pair using the Ed25519 signature scheme. In our example, we will use the default file path with no passphrase.

```
ssh-keygen -t ed25519
```

Add the public key (`cat ~/.ssh/id_ed25519.pub`) to your AVH project's authorized project keys under https://app.avh.arm.com/admin/projects.

1. Install USB/IP and the hardware identification tool.

```
sudo apt update && sudo apt install -y linux-tools-$(uname -r) hwdata
```

2. Load the USB Virtual Host Controller Interface kernel module.

```
sudo modprobe vhci-hcd
```

3. Use ssh to forward local port 6000 to port 6000 of your device's services IP.

```
ssh -M -Ssock -N -f -L 6000:<STM_device_services_IP>:6000
 <your_project_id>@proxy.app.avh.arm.com
```

4. List USBIP devices on localhost port 6000.

```
usbip --tcp-port 6000 list -r localhost
```

5. You should see the virtual STMicroelectronics USB peripheral appear.
6. When you are finished, close the socket.

```
ssh -Ssock -O exit proxy.corellium.com
```

5.3 vMMIO Introduction

Getting started with virtual MMIO (vMMIO):

- Background
- The CoreIO Library API
  - Connecting to a target VM
  - Registering a vMMIO Range
  - Registering a vMMIO Handler
  - Processing Read/Write Events
  - Interrupts
  - Direct Memory Access (DMA)
- Examples

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.3.1 Background

On physical devices, software and hardware interact in a predefined way to yield predefined results. Namely, hardware is driven by software drivers according to its specification. Fundamentally, this interaction distills into a series of address and data bus read and write operations. On AVH virtual devices this behavior is identical, save for the distinction that software interacts with virtual software-defined models of the physical hardware device being virtualized.

AVH implements and extends virtual hardware model capabilities through what is referred to as Virtual Memory Mapped IO (vMMIO). This allows user-supplied code to be executed in response to VM physical memory read or write bus transactions. The full suite of features available also include DMA for arbitrary memory read and write, as well as control of a dedicated virtual IRQ line.

5.3.2 The CoreIO Library API

The CoreIO library allows external programs to act as MMIO responders, send interrupt requests to the VM, and perform DMA.

5.3.3 Connecting to a target VM

To use the library in your C program, you need to connect to a VM:

```
int coreio_connect(const char *target);
      Connect to a VM.
        target          string like "10.10.0.3:23456"
      Returns error flag.
```

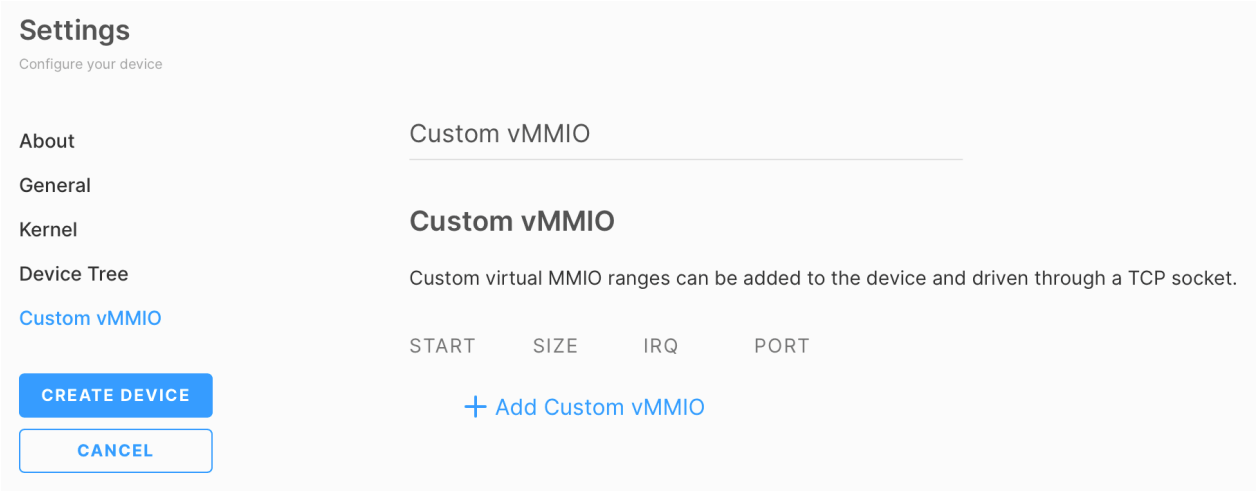
If the result was zero, you can continue using the CoreIO API.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.3.4 Registering a vMMIO Range

The vMMIO ranges must be created either through the web API or through the UI. After VM creation, a vMMIO range may be created by using the relevant tab under `Settings`.

Figure 5-49: Custom vMMIO



`Start` and `Size` represent the physical address space of the virtual MMIO range being created. The `IRQ` is used when vMMIO handlers implement a virtual IRQ line linked to the core interrupt controller. And `port` represents the network port on which AVH will listen for connections.

Note

vMMIO ranges created without any associated client handlers (below) will not trigger bus faults within the VM. The default behavior is to return a `0xaa` in each byte up to the width size of the access (e.g. `0xaaaa` for two byte reads).

5.3.5 Registering a vMMIO Handler

After registering the range, the access handler must also be registered so that AVH knows where to route the access events to.

The API is based around a few concepts. First, there's the vMMIO range. Each range is a 4k-aligned block of address space that a CoreIO client can claim. Ranges are indexed from 0, in the order they were specified during VM creation. Up to 16 vMMIO ranges can be created per VM instance.

To claim a vMMIO range, create and fill out a callback structure:

```
#define COREIO_FLAGS_IFETCH     2
    #define COREIO_FLAGS_PRIV       4
    #define COREIO_FLAGS_SECURE     8
    typedef struct {
        /* MMIO operations. */
        int (*read)(void *priv, uint64_t addr, size_t len, void *buf, unsigned
 flags);
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
        int (*write)(void *priv, uint64_t addr, size_t len, void *buf, unsigned
flags);
        /* MMIO pair-wise operations. If not supplied, read/write are used twice. */
        int (*readp)(void *priv, uint64_t addr, size_t len, void *buf1, void *buf2,
unsigned flags);
        int (*writep)(void *priv, uint64_t addr, size_t len, void *buf1, void *buf2,
unsigned flags);
    } coreio_func_t;
```

The read/write functions are called with:

- `priv` - your unmodified private pointer (`funcp` from register function)
- `addr` - VM physical address of access
- `len` - Byte length of access, for example 1, 2, 4, 8, 16 ...
- `buf` - Data buffer (contains VM write contents on write, else destinations buffer for reads)
- `flags` - Flags describing access mode, see `COREIO_FLAGS_*`

The functions can return 1 to signal failure, which will become a hardware-origin bus error when it hits the VM.

After you have your handlers and `coreio_func_t` filled out, register it:

```
int coreio_register(unsigned rid, const coreio_func_t *func, void *funcp);
      Register a MMIO handler.
        rid             MMIO range ID to register
        func            io handler function table
        funcp           io handler function parameter (passed to callbacks)
      Returns error flag.
```

5.3.6 Processing Read/Write Events

Then you must construct a main loop of your program. The main loop support is intended to work with `select()`. You call `coreio_preparefds` before `select()` to add CoreIO's sockets to select wait list, and `coreio_processfds` after select to process the socket events:

```
int coreio_preparefds(int nfds, fd_set *readfds, fd_set *writefds);
      Prepare fd_sets for select(2).
        nfds            current index of maximum fd in sets + 1
        readfds         readfds to update
        writefds        writefds to update
      Returns new nfds.
```

```
int coreio_processfds(fd_set *readfds, fd_set *writefds);
      Process fd_sets after select(2).
        readfds         readfds to process
        writefds        writefds to process
      Returns error flag (for instance, connection to VM lost).
```

A simple implementation of a main loop, without any other event processing, looks like this:

```
/* ... */
    fd_set readfds, writefds;
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
    int res;
    while(1) {
        FD_ZERO(&writefds);
        FD_ZERO(&readfds);
        int nfds = coreio_preparefds(0, &readfds, &writefds);
        if(nfds < 0) /* error in prepare */
            break;
        select(nfds, &readfds, &writefds, NULL, NULL);
        res = coreio_processfds(&readfds, &writefds);
        if(res) /* error in process */
            break;
    }
    /* ... */
```

This kind of main loop, with a timeout, is provided as a convenience by the library:

```
int coreio_mainloop(long long usec);
      Simple implementation of a main loop.
        usec            time to spend in loop, in microseconds;
                        negative means forever
      Returns error flag.
```

5.3.7 Interrupts

In addition to processing MMIO events, your program can also send interrupts. Note that in the VMs, all interrupt lines are level. The interrupt controller can interpret them as edge, but to trigger an edge interrupt you must send two updates: one from 0 to 1, one from 1 to 0. It is not sufficient to send 1 on each IRQ trigger as this does not create an edge event.

```
int coreio_irq_update(unsigned rid, unsigned iid, unsigned state);
      Send an IRQ update.
        rid             MMIO range ID associated with IRQ
        iid             IRQ index
        state           IRQ line state (1 - active, 0 - inactive)
```

5.3.8 Direct Memory Access (DMA)

There are two ways to perform DMA access, blocking and non-blocking. Blocking access is simpler. You just call a read/write function, and it returns when done. Non-blocking access requires a completion function (and its opaque parameter pointer, which will be passed to it without change). The function will be called at a future time during `coreio_processfds()`.

```
int coreio_dma_read(unsigned mid, unsigned sid, uint64_t addr,
        size_t len,void *buf, unsigned flags,
        void (*cpl)(void *, int), void *cplp);
      Start a DMA read.
        mid             IOMMU ID (0 = direct)
        sid             IOMMU region ID
        addr            target address
        size            size of read to perform
        buf             memory buffer to fill with data
        flags           bus access flags
        cpl             completion function; if NULL, call is blocking
        cplp            completion function parameter
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
      Returns error flag.
```

```
int coreio_dma_write(unsigned mid, unsigned sid, uint64_t addr,
        size_t len, void *buf, unsigned flags,
        void (*cpl)(void *, int), void *cplp);
      Start a DMA write.
        mid             IOMMU ID (0 = direct)
        sid             IOMMU region ID
        addr            target address
        size            size of write to perform
        buf             memory buffer with data
        flags           bus access flags
        cpl             completion function; if NULL, call is blocking
        cplp            completion function parameter
      Returns error flag.
```

On most small VMs, you can pass 0 as IOMMU ID and region ID. This specifies that the access should target VM physical memory directly.

Finally, to cleanly shut down the CoreIO library, use:

```
void coreio_disconnect(void);
```

5.3.9 Examples

The example sources, available as either a tarball or a ZIP file, include the following:

`rpi-test`

- Use an inside and outside VM agent for demonstrating end-to-end vMMIO cycle

`slider-mmio`

- Use Python bindings to send an IRQ to the VM depending on the state of a graphical slider

`vsi-waveio`

- Create a whole virtual device model confronting to Arm's VSI specification that streams a WAV file into the VM

Next steps Raspberry Pi vMMIO Example

5.4 Kernel debugging

This article shows you how to debug a VM kernel.

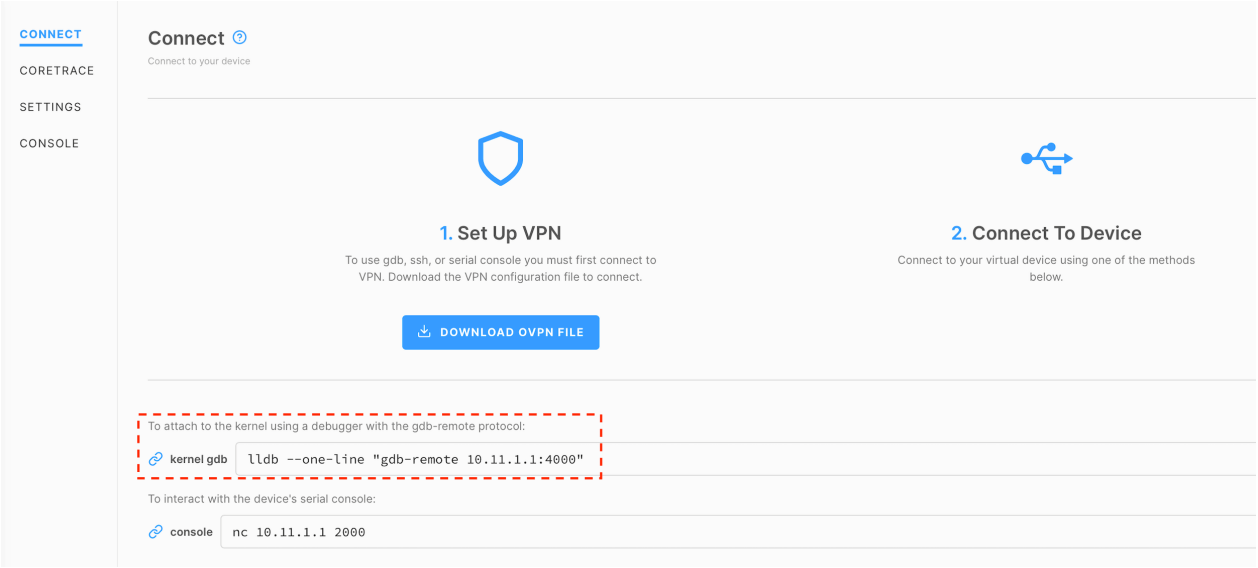
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.4.1 Using GDB/LLDB

AVH implements a GDB server stub to support VM debugging. This allows any debugger that understands this protocol to be used as well, including `lldb`.

After creating a VM, the `Connect` tab provides an `lldb` one-liner that can be used to establish a debugging session with the VM.

Figure 5-50: Connect



While `lldb` is specified in the UI, users are free to use `gdb` as well

```
$ gdb
(gdb) target remote 10.11.1.1:4000
Remote debugging using 10.11.1.1:4000
0xffff80001117094c in ?? ()
(gdb) info reg
x0             0xe0     224
x1             0x11c430 1164336
x2             0x0      0
x3             0x4000000000000000       4611686018427387904
x4             0x0      0
x5             0xffff8000dde56000       -140733765558272
x6             0x80     128
...
(gdb) x/10i $pc
=> 0xffff80001117094c:  hint    #0x1d
   0xffff800011170950:  ret
   0xffff800011170954:  hint    #0x19
   0xffff800011170958:  stp     x29, x30, [sp,#-16]!
   0xffff80001117095c:  mov     x29, sp
   0xffff800011170960:  bl      0xffff800011170940
   0xffff800011170964:  ldp     x29, x30, [sp],#16
   0xffff800011170968:  hint    #0x1d
   0xffff80001117096c:  ret
   0xffff800011170970:  hint    #0x19
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

All the usual debugger features work, including single-step, breakpoints, watchpoints, memory inspection, register inspection, register or memory editing, etc. Consult a guide specific to your debugger.

5.4.2 Monitor Commands

The so-called monitor commands are raw packets interpreted directly by the debug server and not by the client (or marshaled into any known command format). In other words, the monitor is convenient way to expand a debugger's functionality without changing the client code. `gdb` exposes this functionality by prefixing your command with `monitor`. Sending e.g. `monitor help` requests that AVH respond with its list of available commands

```
(gdb) monitor help
Available monitor commands:
  help                  List all commands.
  info                  Show VM information.
  sr                    Access CPU system registers.
  pt                    Dump current pagetable.
  v2p                   Translate a virtual address to physical.
  p2v                   Translate a physical address to virtual.
  vcont-dflt            Apply default vCont action if specific actions are present.
  phys                  Switch to physical access mode (optionally takes base
 address).
  virt                  Switch to virtual access mode.
  bt                    Show iOS-style backtrace.
  thread                Show current thread.
```

To list the contents of the system registers one may use `monitor sr`.

```
(gdb) monitor sr
System registers of CPU 0:
  sctlr_el1        = 0x0000000030d4d91d
  cpacr_el1        = 0x0000000000300000
  zcr_el1          = 0x0000000000000000
  ttbr0_el1        = 0x00000001061c5000
  ttbr1_el1        = 0x0188000041859000
  tcr_el1          = 0x00000032b5503510
...
```

Or list the current configuration of the page tables of the current virtual CPU (vCPU).

```
(gdb) monitor pt
Pagetable for CPU 0:
  0000004000000000-0000004000000fff -> 010a022000 local user pXN uXN memory Owb-rwa
 Iwb-rwa
  0000004000001000-0000004000001fff -> 010a0d9000 local user pXN uXN memory Owb-rwa
 Iwb-rwa
  0000004000002000-0000004000002fff -> 010a023000 local user pXN uXN memory Owb-rwa
 Iwb-rwa
  0000004000008000-0000004000008fff -> 0108871000 local user pXN uXN memory Owb-rwa
 Iwb-rwa
  0000004000009000-0000004000009fff -> 0108741000 local user pXN uXN memory Owb-rwa
 Iwb-rwa
  000000400000a000-000000400000afff -> 010a1db000 local user pXN uXN memory Owb-rwa
 Iwb-rwa
...
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

`lldb` can be used similarly to send monitor commands using a slightly more verbose command structure.

```
$ lldb --one-line "gdb-remote 10.11.1.1:4000"
(lldb) gdb-remote 10.11.1.1:4000
Process 1 stopped
* thread #1, stop reason = signal SIGINT
    frame #0: 0xffffffe01117094c
Target 0: (No executable module.) stopped.
(lldb) process plugin packet monitor help
Available monitor commands:
  help                  List all commands.
  info                  Show VM information.
  sr                    Access CPU system registers.
...
```

Note

Debugger threads are discrete VM CPUs, and so switching between threads allows users to interact with specific cores.

5.5 Custom device tree

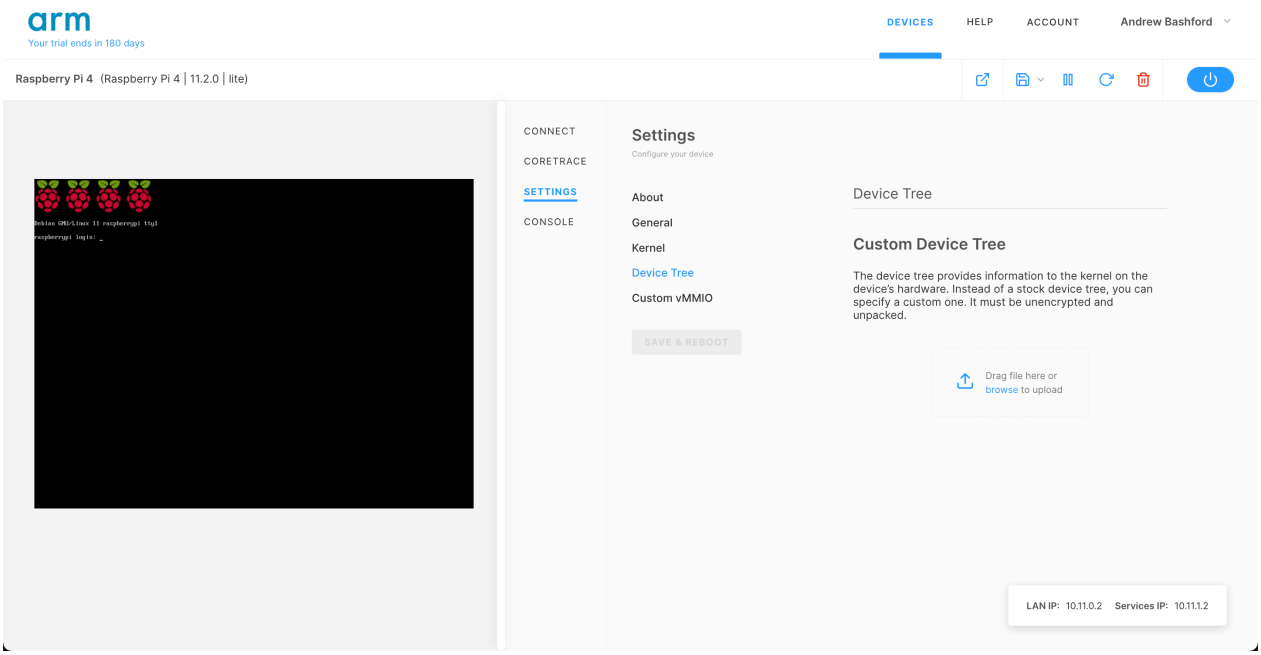
Custom Device Trees are available on Raspberry Pi 4 and the iMX8m Arm Cortex Complex devices.

A device tree is a representation of hardware used by boot loaders provided to the kernel. It is a hierarchical representation of devices by connection, used by the kernel to communicate with the I/O buses and other low-level hardware.

Instead of a stock device tree, you can specify a custom one. It must be unencrypted and unpacked. To upload a custom device tree, simply head to Settings > Device Tree and drag-and-drop your file to upload it.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-51: Custom device tree



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6. Troubleshooting and FAQs

The following articles are available:

- CPU Cores and Device Slot Limits
- Using the console interactively
- FAQ - My VPN keeps disconnecting
- FAQ - What is a core?]
- Deleting your AVH account
- Updating the Raspberry Pi 4 Kernel

6.1 CPU Cores and Device Slot Limits

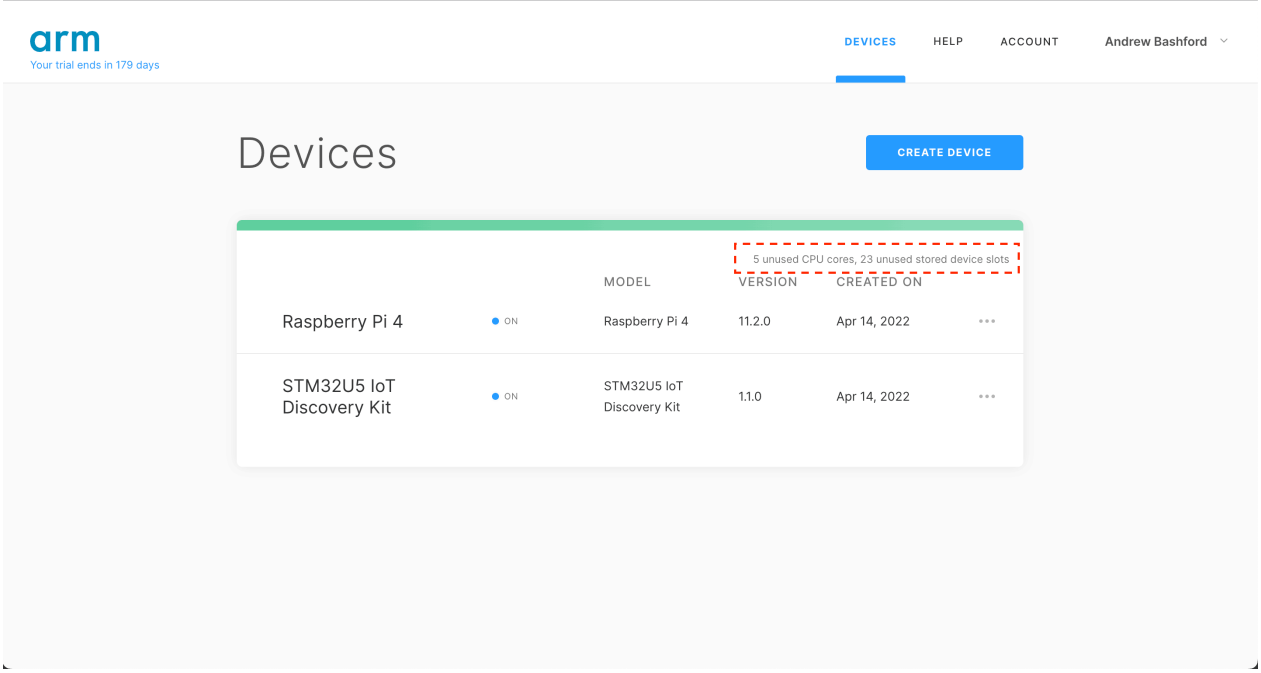
In AVH your account is initially provisioned with 10 CPU Cores and 25 Device Slots.

CPU Cores are the combined maximum number of cores you can have across all of your devices at any one time.

Device Slots are the combined maximum number of devices you can have in your devices list at any one time.

You can find out your current usage on the devices page.

Figure 6-1: CPU usage



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

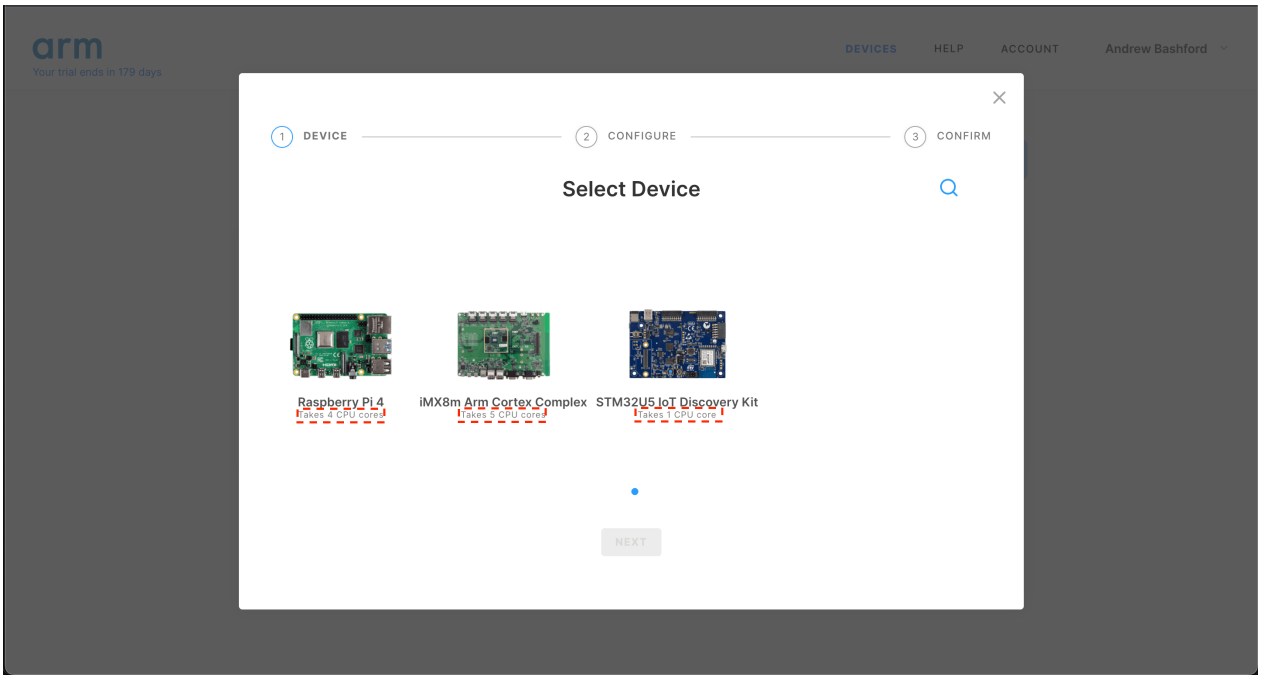
In the image above the user initially had 10 CPU Cores and 25 Device Slots.

The Raspberry Pi 4 device takes 4 CPU Cores and 1 device slot. The STM32U5 IoT Discovery Kit takes 1 CPU Cores and 1 device slot.

If you delete a device you will get the number of CPU Cores it occupied and Device Slot credited back to your account.

You can see how many CPU Cores a device requires before you create it.

Figure 6-2: Select device



The number of cores per device:

- Raspberry Pi 4 - Takes 4 CPU cores
- iMX8m Arm Cortex Complex - Takes 5 CPU cores
- STM32U5 IoT Discovery Kit - Takes 1 CPU core

To request more CPU Cores and Device Slots please click on the chat button in the bottom right of the screen and send us a message with how many more you would like and their intended purpose.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.2 Using the console interactively

The console of devices running Linux 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 behaviour, and run `dmesg -n 7` to log all messages to the console.

6.3 FAQ - My VPN keeps disconnecting

The VPN is only active when at least one device in the project is in the On state. If all devices are off, or if no devices exist in the project, then the VPN will disconnect. It will reconnect when a device in that project turns on.

6.4 FAQ - What is a core?

You are allotted a certain number of CPU cores, and you can run as many virtual devices as that number of CPUs will permit at a time.

These CPU cores correspond to the number of CPU cores available on the Arm servers assigned to that account. Each virtual device requires a certain number of CPU cores in order to run.

For instance, if you have a 12-core account, you can spin up two 6-core devices for your first test run, then you could turn those off for storage and create six 2-core devices for the next test. For every two active CPU cores allotted to your account, you can store up to five devices in an Off state.

The number of cores per device:

- Raspberry Pi 4 - Takes 4 cores
- iMX8m Arm Cortex Complex - Takes 5 cores
- STM32U5 IoT Discovery Kit - Takes 1 core

6.5 Deleting your AVH account

Deleting your AVH account will permanently remove your account from AVH. All data, devices and snapshots will be erased.

Deleting your AVH account does not delete or alter your Arm account that you use to sign into AVH.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Once deleted should you wish to create a new AVH account you can sign up again as long as you have an active Arm Account.

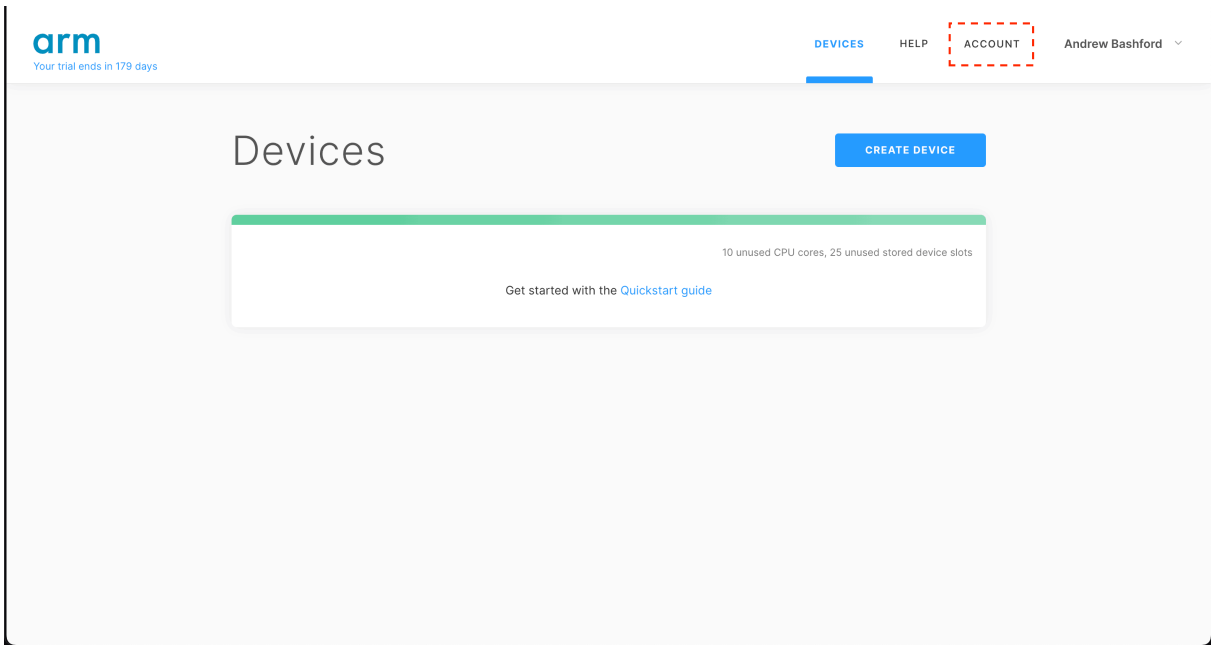
If your Arm account is deleted or suspended your access to AVH will be removed.

6.5.1 Deleting your account

To delete your account:

1. To delete your account go to ACCOUNT in the navbar.

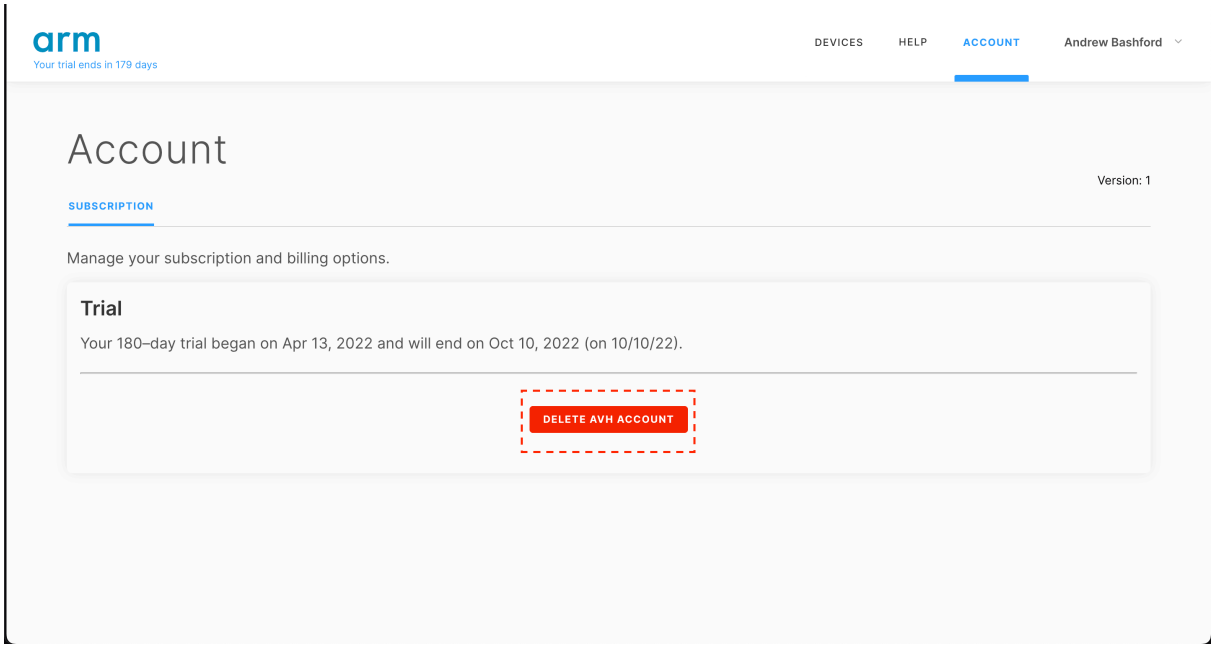
   Figure 6-3: Account



2. Click DELETE AVH ACCOUNT.

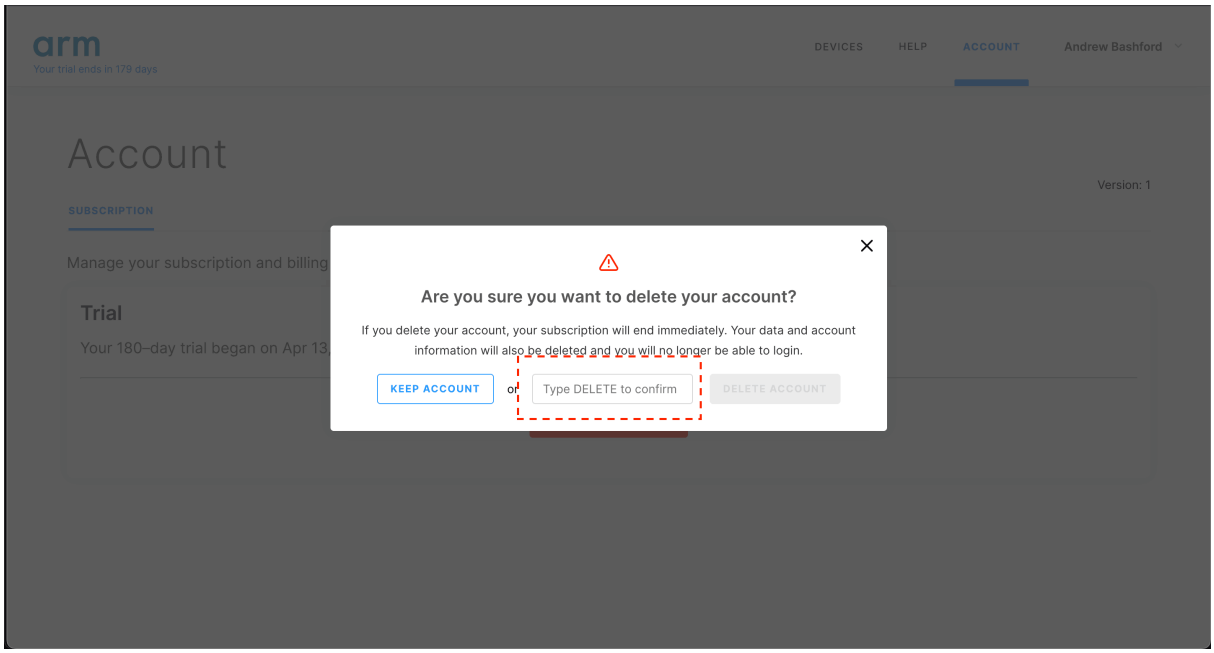
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-4: Delete AVH account



3. Type DELETE into the input, it is case sensitive.

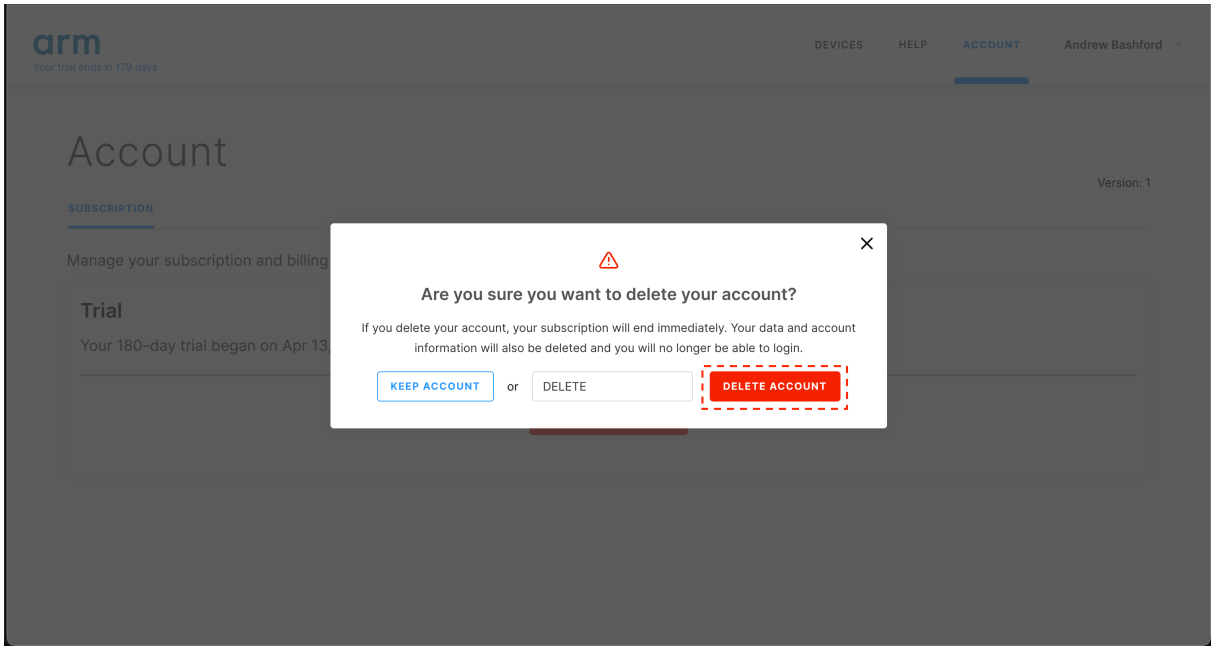
Figure 6-5: Type DELETE



4. Click DELETE ACCOUNT to confirm deleting your account. All of your data will be permanently deleted.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-6: Click DELETE ACCOUNT



6.6 Updating the Raspberry Pi 4 Kernel

Follow these steps to fix your kernel after updating your virtual Raspberry Pi 4 board.

6.6.1 Table of Contents

This section contains the following:

- Updating the Raspberry Pi 4 Kernel
  - Update Using the Console Command Line
  - Update Using the Welcome to Raspberry Pi GUI on the Desktop Firmware
- Identifying Issues with the Device Kernel
  - List the Loaded Kernel Modules
  - Try to Install and Run Docker
- Fixing the Updated Kernel File

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.6.2 Updating the Raspberry Pi 4 Kernel

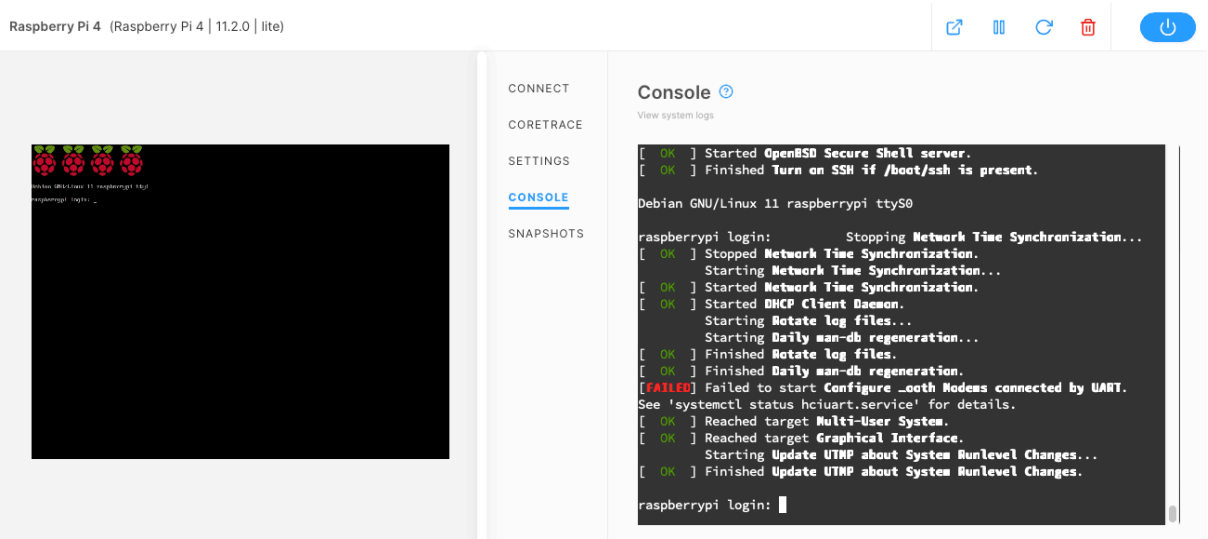
You can update the kernel using either the command line or the Desktop firmware's "Welcome to Raspberry Pi" GUI.

6.6.3 Update Using the Console Command Line

To update the kernel using the console command line:

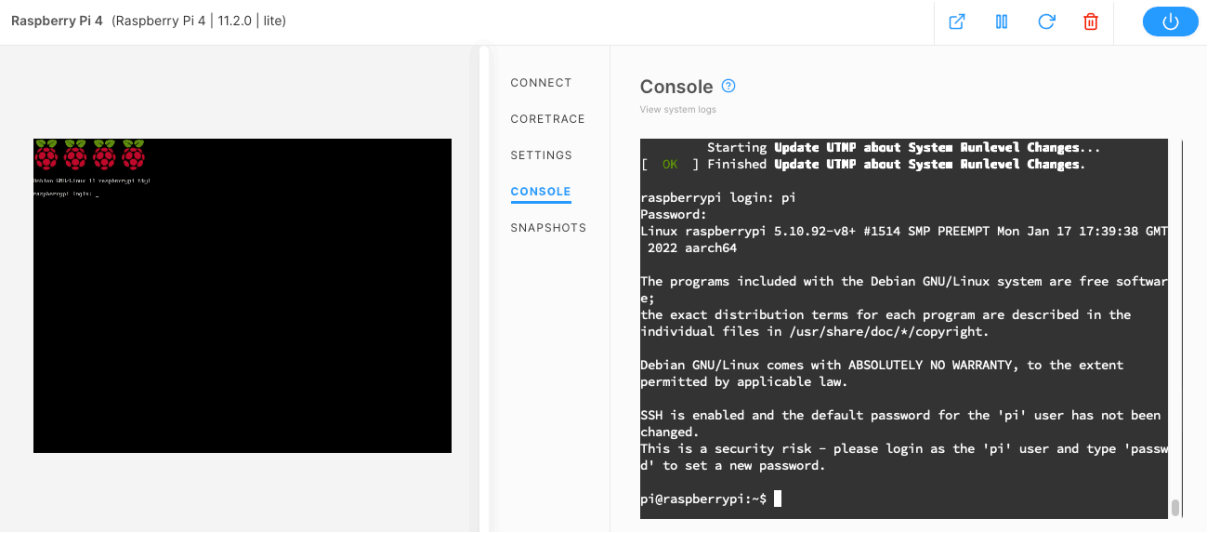
1. Open the Console tab and press return to show the login prompt.

Figure 6-7: Open Console tab



2. Log in with the username `pi` and password `raspberry`.

Figure 6-8: Logged in



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. Update the device using:

```
sudo apt-get update && sudo apt-get upgrade -y
```

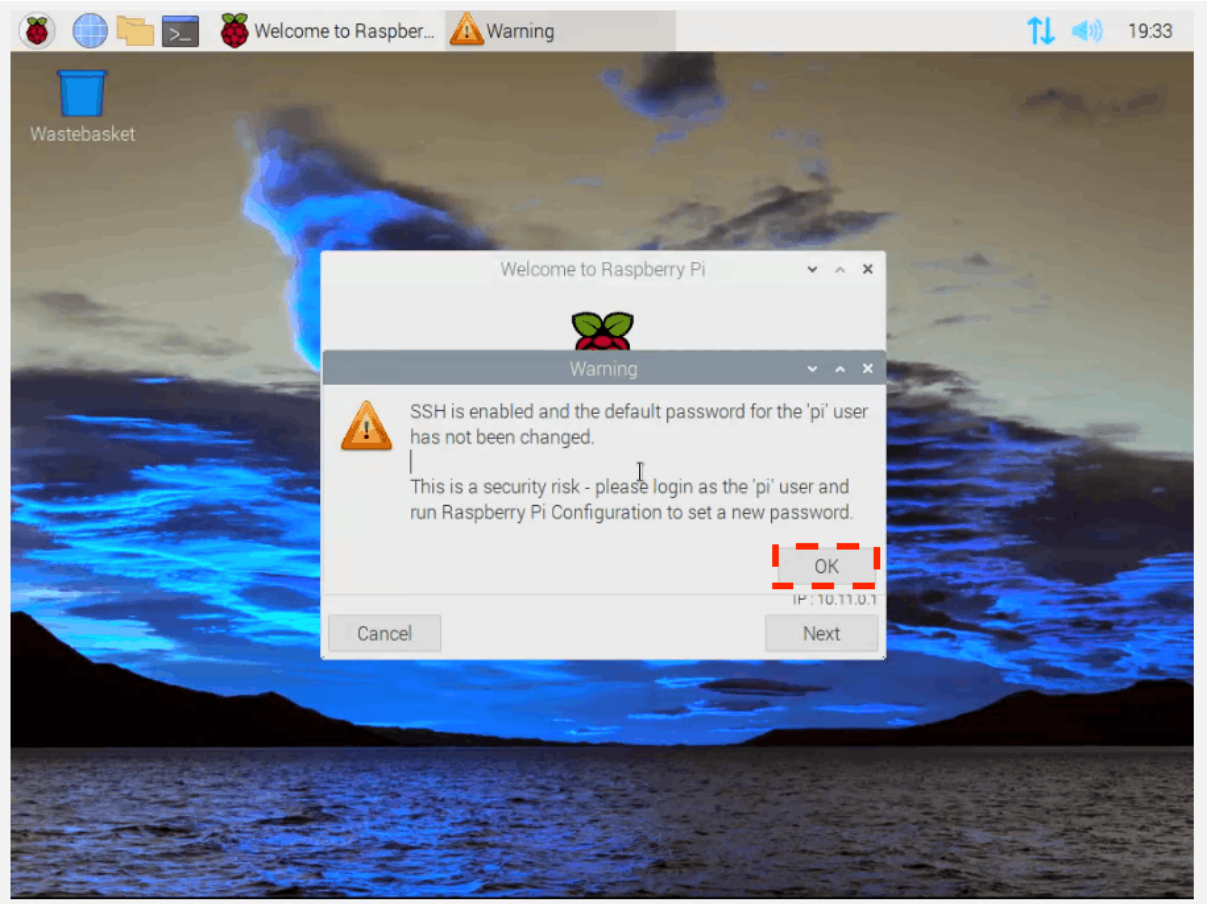
4. When the installation is complete, click the Restart button towards the top-right.
5. Continue to the Identifying Issues with the Device Kernel section to check for issues with your kernel.

6.6.4 Update Using the Welcome to Raspberry Pi GUI on the Desktop Firmware

To update the kernel using the desktop firmware GUI:

1. Create a Raspberry Pi 4 device using the stock firmware `Raspberry Pi OS Desktop (11.2.0)`.
2. Close the warning window about the default SSH password.

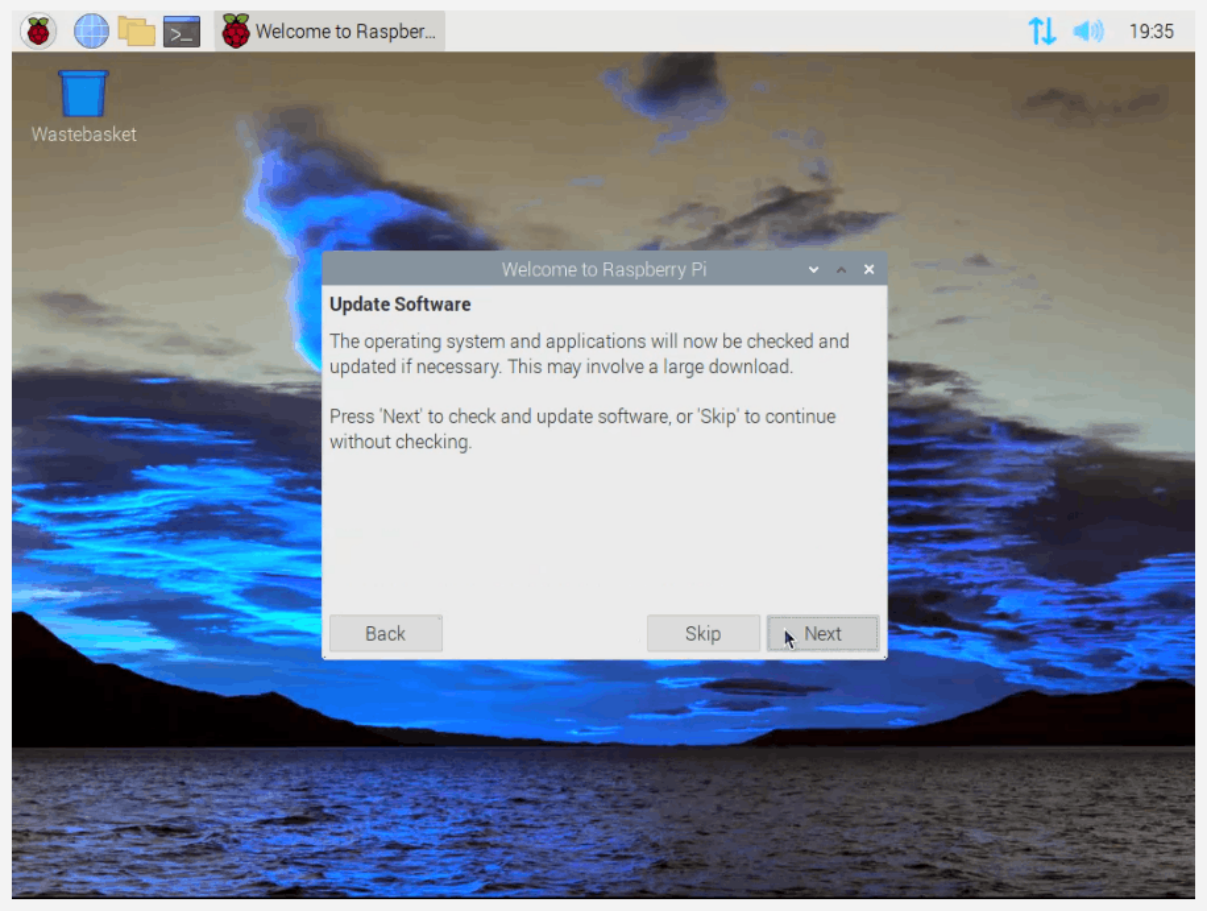
Figure 6-9: Close SSH popup



3. Click through the Welcome to Raspberry Pi window.
4. The last screen will ask if you want to update. Click Next.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-10: Update software



5. Restart the device when the update completes.
6. Continue to the Identifying Issues with the Device Kernel section to check for issues with your kernel.

6.6.5 Identifying Issues with the Device Kernel

In this section we check if modules are loaded into the kernel and try to install and run Docker.

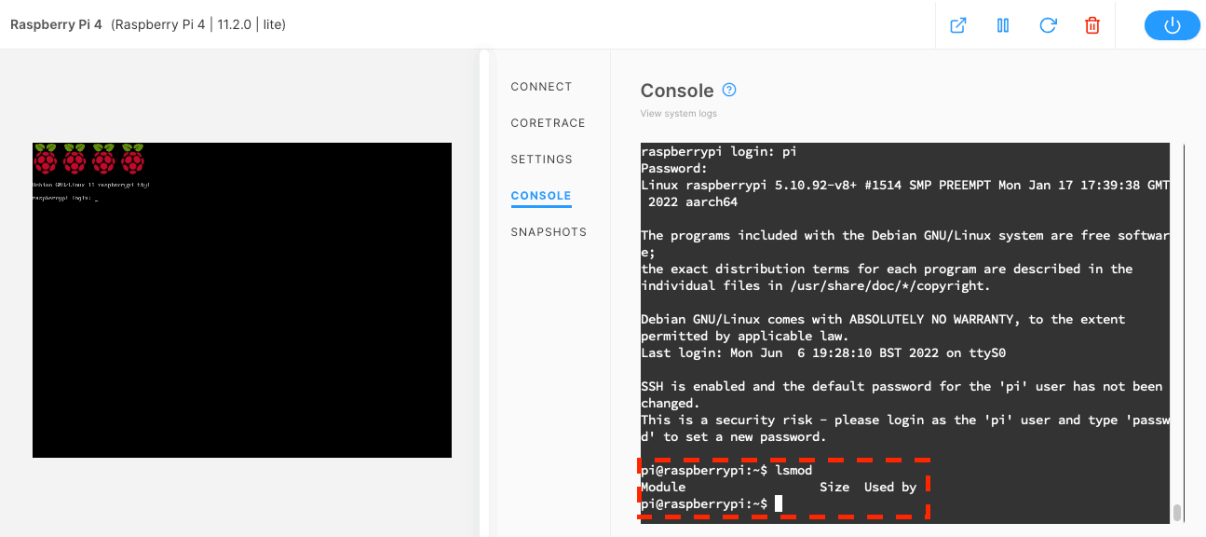
6.6.6 List the Loaded Kernel Modules

To list the loaded kernel modules:

1. Run `lsmod` in the console to list the modules that are loaded into the kernel.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-11: List the loaded kernel modules



2. Review the results.
   - If the list is empty (or nearly empty,) you likely have kernel issues that need to be manually resolved. Continue to the Fixing the Updated Kernel File section to correct your issue.
   - If the list comes back with multiple results, such as `xt_conntrack`, `nft_chain_nat`, `xt_MASQUERADE`, `nf_nat`, `nf_conntrack_netlink`, `nf_defrag_ipv6`, `nf_defrag_ipv4`, `nf_conntrack nft_counter`, `xt_addrtype`, `nft_compat`, `nf_tables`, and `nfnetlink`, the kernel is likely behaving normally.

6.6.7 Try to Install and Run Docker

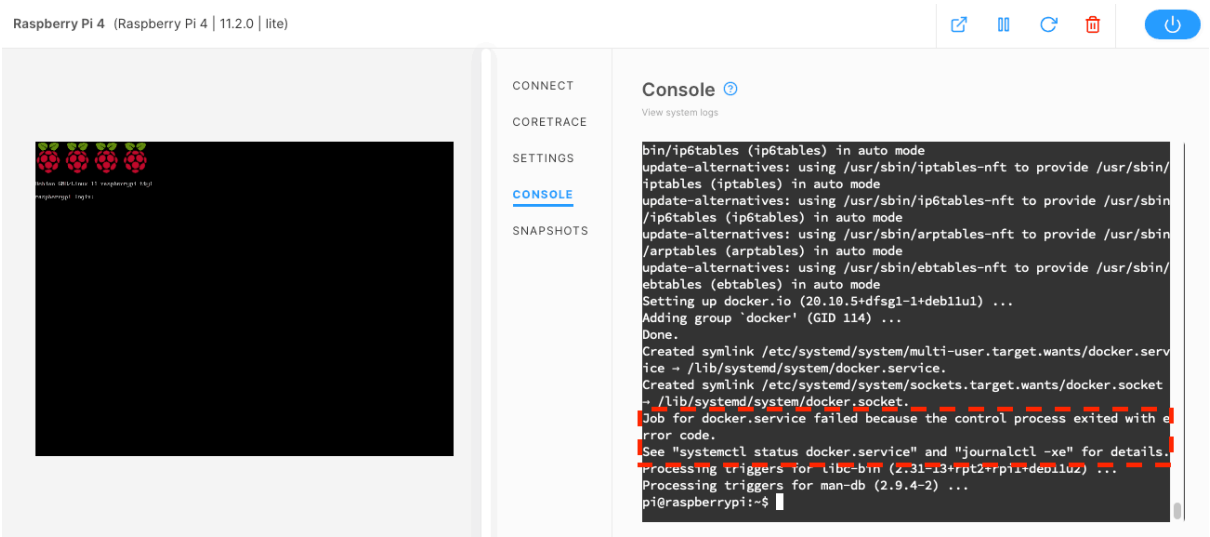
Running docker requires a properly configured kernel. If docker installs but the service fails to run, you may be having an issue with your updated kernel.

1. Install docker and look for a `Job for docker.service failed` message.

```
sudo apt-get update && sudo apt-get install -y docker.io
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-12: Job for docker service failed

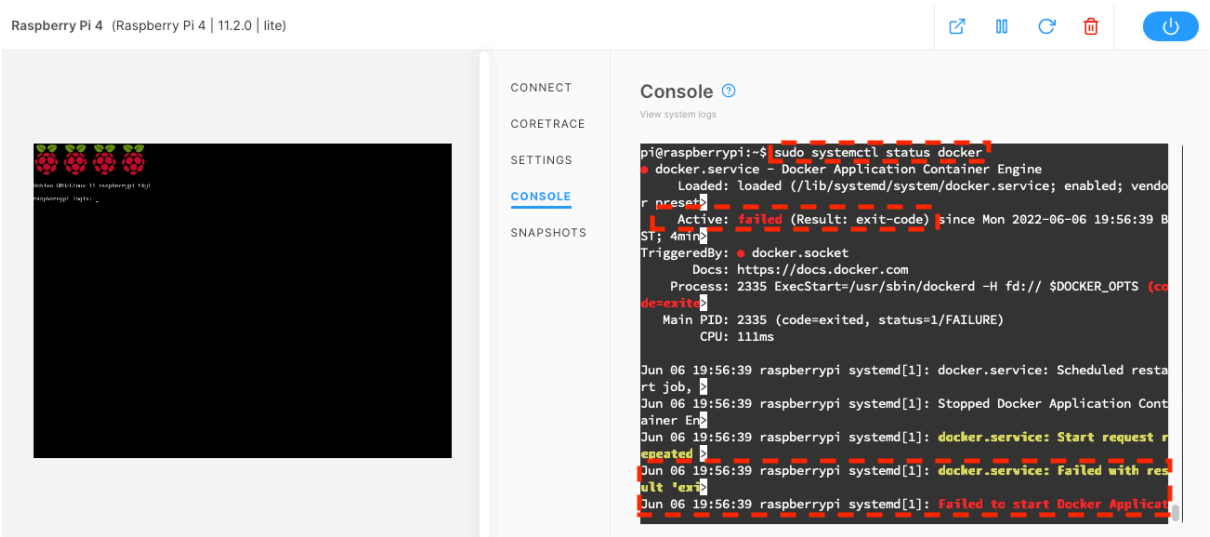


2. Check if the service for docker is running.

```
sudo systemctl status docker
```

- If you see a `failed` state, continue to the Fixing the Updated Kernel File section to correct your issue.

Figure 6-13: Docker failed



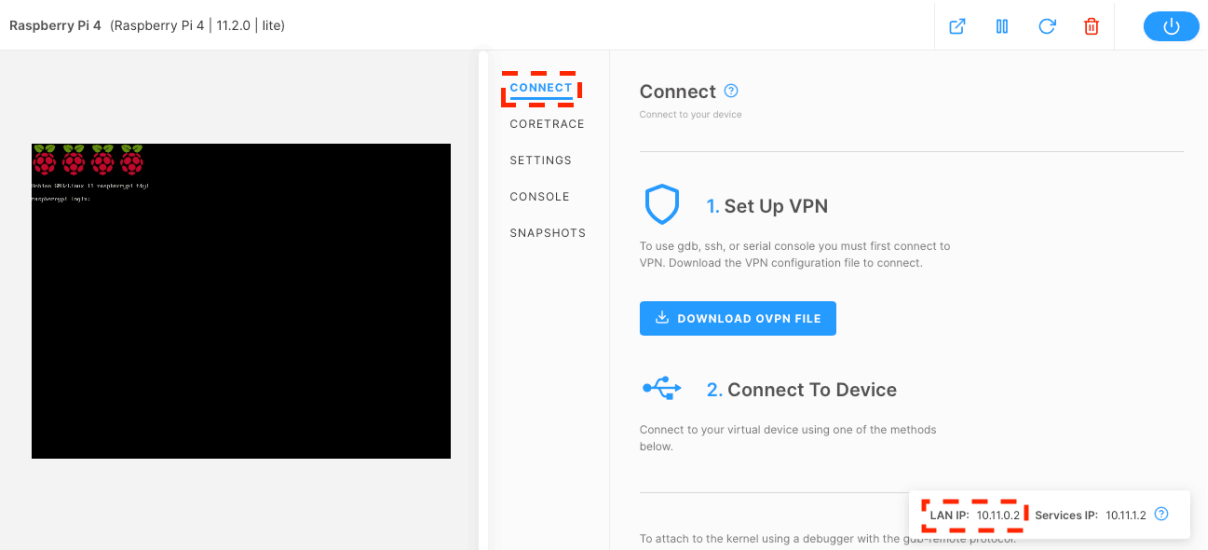
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.6.8 Fixing the Updated Kernel File

To fix the updated kernel file:

1. Connect to your device's VPN using the Connect tab.
2. Find your device's LAN IP on the bottom-right of the Connect tab. In our example, we will use `10.11.0.2`.

Figure 6-14: Find your LAP IP

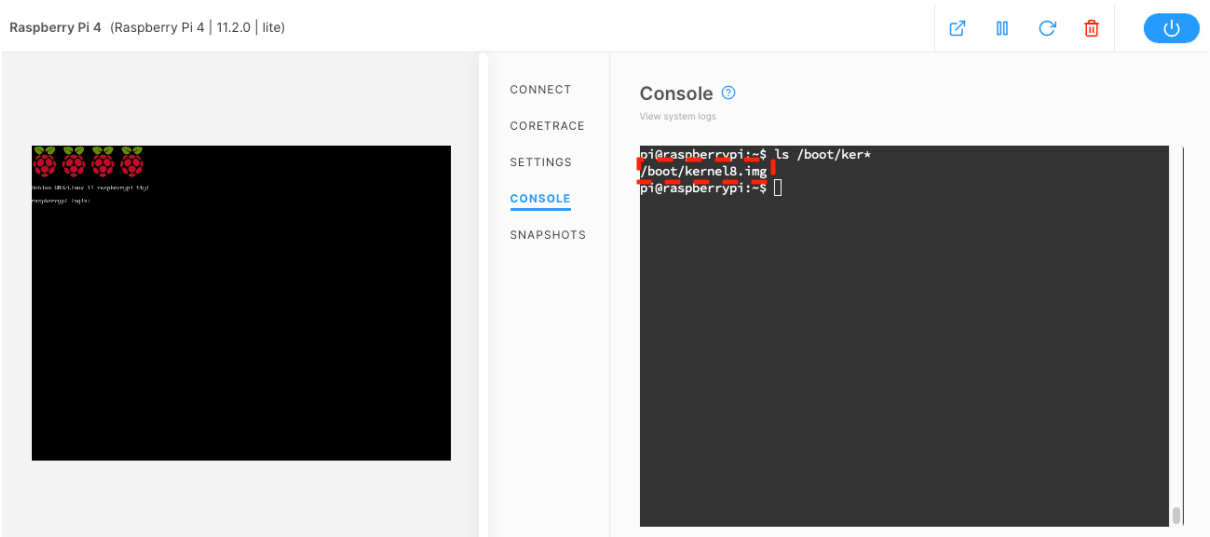


3. On your virtual device, locate the path to your kernel file. In our example, we will use `/boot/kernel8.img`.

```
ls /boot/ker*
```

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-15: Locate the kernel img file



4. From your computer's terminal, download the VM's kernel using scp and the LAN IP and kernel path we just found.

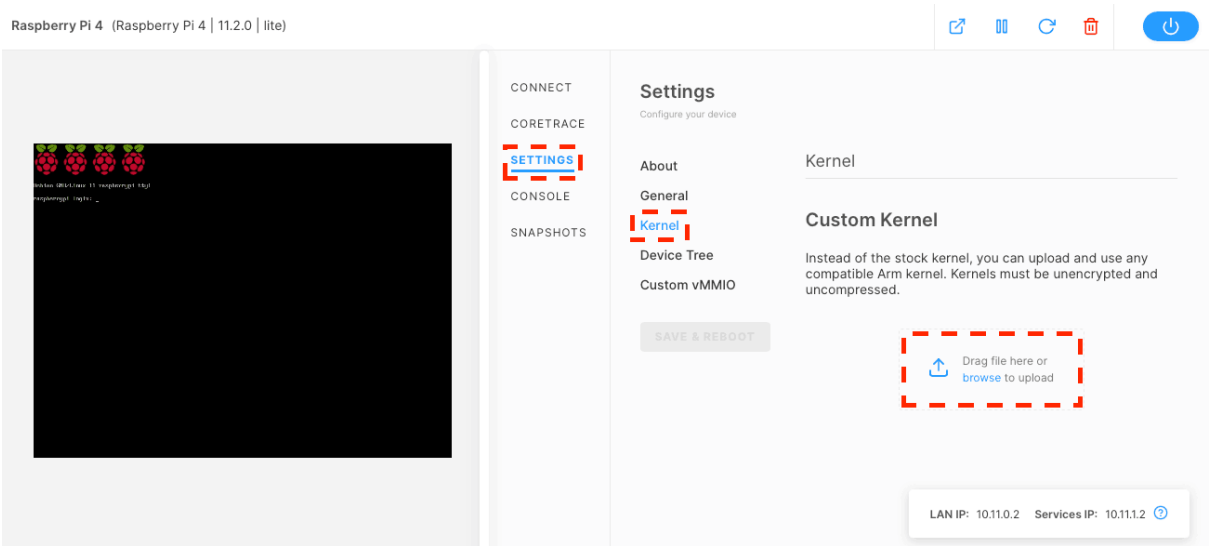
```
scp  pi@10.11.0.2:/boot/kernel8.img ./
```

5. Extract the `img` file.

```
gunzip -cd kernel8.img > kernel
```

6. Upload the kernel file in the Kernel section of the Settings tab.

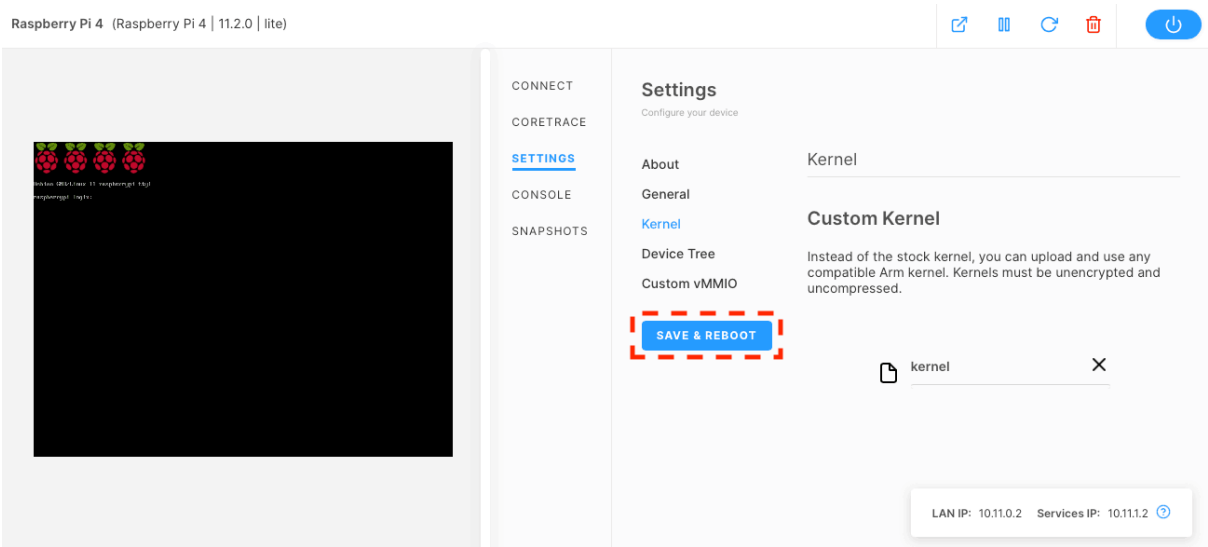
Figure 6-16: Upload new kernel



7. Wait for the upload to complete then click SAVE & REBOOT.

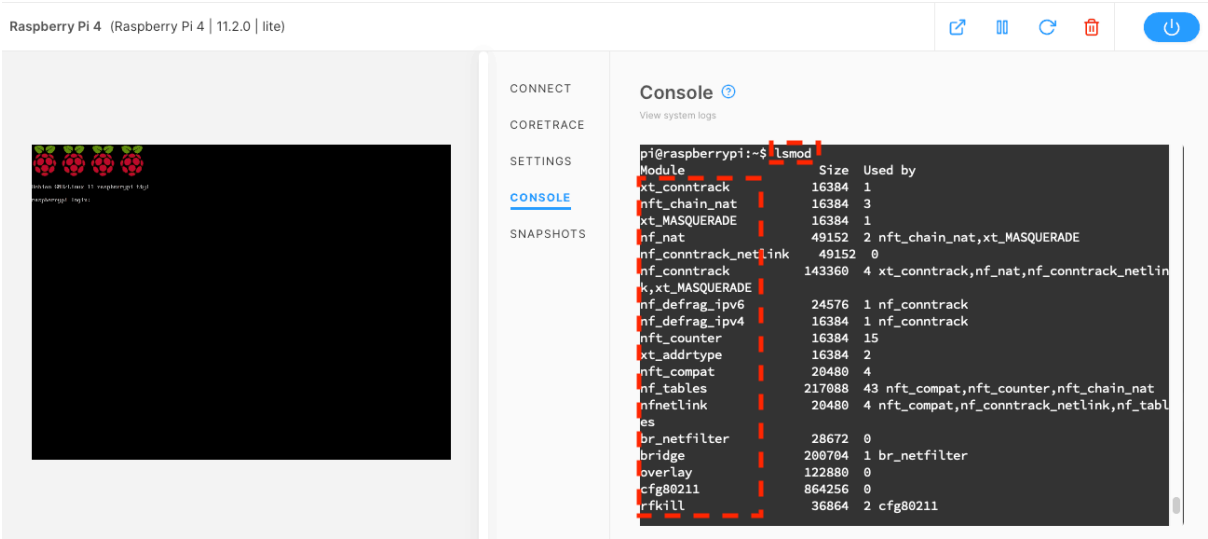
Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-17: Save & reboot



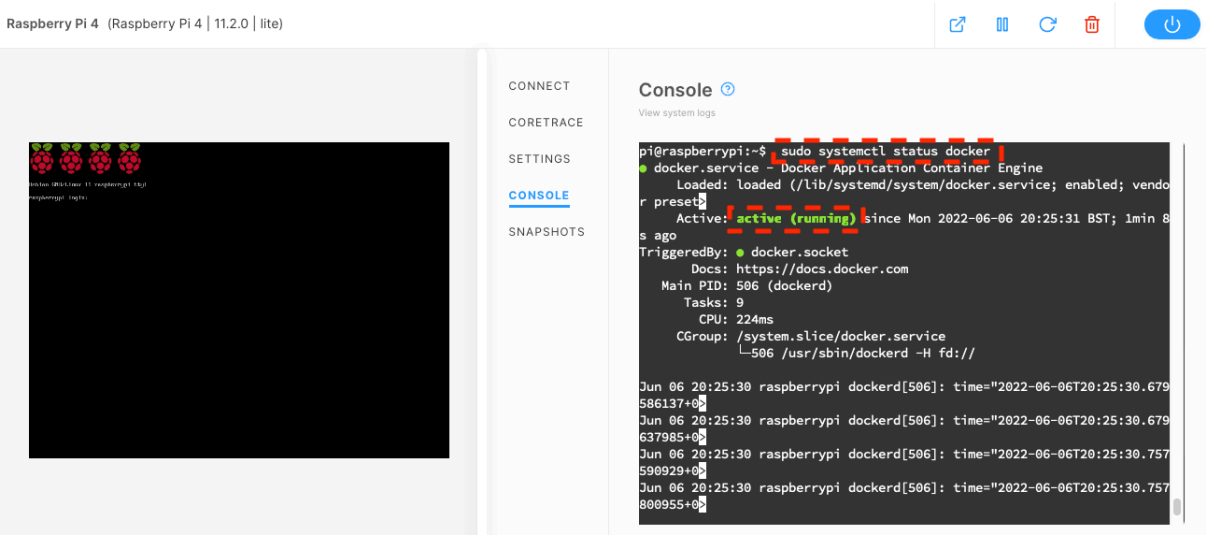
8. Confirm the kernel is working properly using `lsmod` or `sudo systemctl status docker`.

Figure 6-18: Modules are loaded



Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-19: Docker running



9. Clean up your local system.

```
rm kernel8.img kernel
```

10. You're all set!

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7. Updates

The following articles are available:

- Release 3.14 and 3.14.1
- Release 3.11
- Release 3.10

7.1 Release 3.10

May 16, 2022

Our latest release adds website improvements including an updated login process, new device sensors and LEDs, and a user profile page.

- Responsive design enhancements to the front end
- Added modal reporting for user during error states
- Enhanced stability for VMs during deployment
- Redesigned landing page and SSO login sequence
- New line buffered echo feature for STM32U5 devices
- New switches, LEDs, and buttons for STM32U5 devices
- New LED functionality for iMX8m devices
- Improved behavior for devices without video screens
- Added user profile page

7.2 Release 3.11

July 7, 2022

This release adds Quick Connect SSH tunneling, camera and microphone support, custom kernels for i.M X8M devices, and increased Raspberry Pi 4 storage.

- New Quick Connect feature enables SSH without a VPN connection
- Add camera and microphone support for i.M X8M devices
- Add microphone support for STM23U5 devices
- Add custom kernel support for i.M X8M devices
- Add video file playback support through the camera for i.M X8M devices
- Increased disk partition size for Raspberry Pi 4 devices
- Fixed support for the `shutdown` command on Raspberry Pi 4 devices

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Updates to the JavaScript and Python APIs
- Various fixes and improvements to the user interface

7.3 Release 3.14 and 3.14.1

October 14, 2022

This release adds Quick Connect SSH tunneling, camera and microphone support, custom kernels for i.M X8M devices, and increased Raspberry Pi 4 storage.

- New Quick Connect feature enables SSH without a VPN connection
- Add camera and microphone support for i.M X8M devices
- Add microphone support for STM23U5 devices
- Add custom kernel support for i.M X8M devices
- Add video file playback support through the camera for i.M X8M devices
- Increased disk partition size for Raspberry Pi 4 devices
- Fixed support for the `shutdown` command on Raspberry Pi 4 devices
- Updates to the JavaScript and Python APIs
- Various fixes and improvements to the user interface

October 28, 2022

AVH introduces new i.MX 8M firmware with AWS IoT Greengrass and STM32U5 board audio file playback simulating microphone input

- Improved Bluetooth support for select models
- Configuration updates
- Enabled kernel debug option
- Firmware package updates
- Various bug fixes

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential