Arm Virtual Hardware

Version 5.4

User Guide

Non-Confidential

Copyright © 2022–2023 Arm Limited (or its affiliates).
All rights reserved.

Issue 00
107660_0540_00_en

Arm Virtual Hardware

User Guide

Copyright © 2022–2023 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.
0500-00	10 May 2023	Non-Confidential	Updated for public beta.
0510-00	22 June 2023	Non-Confidential	Release 5.1 beta.
0541-00	14 September 2023	Non-Confidential	Release 5.4.1 beta.

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 © 2022–2023 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 for a Beta product, that is a product under development.

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Contents

1. Overview 8
1.1 Access and Costs 8
1.2 Subscription 9
1.3 Release Notes 11

2. Getting Started 16

3. Supported Platforms 17
3.1 Raspberry Pi 4 17
3.1.1 Quickstart for Raspberry Pi 4 18
3.1.2 Supported Hardware Components for Raspberry Pi 4 21
3.1.3 Stock Firmware for Raspberry Pi 4 22
3.1.4 Custom Firmware for Raspberry Pi 4 23
3.1.5 Storage Files for Raspberry Pi 4 28
3.1.6 Connect to Wi-Fi on Raspberry Pi 4 29
3.1.7 Connect to BLE on Raspberry Pi 4 35
3.1.8 Use vMMIO on Raspberry Pi 4 62
3.1.9 Updating Linux Kernel on Raspberry Pi 4 70
3.1.10 Running Docker on Raspberry Pi 4 78
3.2 i.MX 8M Arm Cortex Complex 81
3.2.1 Quickstart for i.MX 8M Arm Cortex Complex 82
3.2.2 Supported Hardware Components for i.MX 8M 85
3.2.3 Stock Firmware for i.MX 8M 89
3.2.4 Storage Files for i.MX 8M 89
3.2.5 Blink LED on i.MX 8M 90
3.2.6 Play Video Files on i.MX 8M 90
3.3 i.MX 93 Applications Processor 97
3.3.1 Quickstart for i.MX 93 Applications Processor 97
3.3.2 Supported Hardware Components for i.MX 93 101
3.3.3 Stock Firmware for i.MX 93 101
3.4 STM32U5 IoT Discovery Kit 102
3.4.1 Quickstart for STM32U5 IoT Discovery 102
3.4.2 Supported Hardware Components for STM32U5 106

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.4.3 Stock Firmware for STM32U5 109
3.4.4 Storage Files for STM32U5 110
3.4.5 IoT HTTP Web Server on STM32U5 111
3.4.6 TF-M example on STM32U5 117
3.4.7 Detect Audio through the Microphone on STM32U5 Board 122
3.5 Arm Smart Vision Configuration Kit 130
3.6 Arm Fixed Virtual Platforms 131
3.6.1 Quickstart for Arm FVPs 132
3.7 Generic Android 137
3.7.1 Android WiFi 137
3.7.2 Android Debug Bridge 138

4. Device Controls 140
4.1 Console 140
4.2 Action Buttons 140
4.3 Sensors 142
4.3.1 Camera and Microphone 143
4.3.2 Environmental 144
4.3.3 Motion and Position 146
4.3.4 Location 146
4.3.5 Battery 148
4.4 Snapshots 149

5. Device Firmware 154
5.1 Stock Firmware 154
5.2 Linux Firmware Packages 154
5.3 Change Kernel Boot Arguments 156
5.4 Custom Device Tree 157
5.5 MCU Firmware Packages 158
5.6 Upload Firmware in Web UI 159
5.7 Upload Firmware with Python API 163

6. Connect to Virtual Devices 168
6.1 Remote Connections to Devices 168
6.2 Connections between Devices 168
6.3 Quick Connect 169
6.4 Connect to your AVH VPN 176

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.5 Services IP and LAN IP 186
6.6 Connect to Device Console 187
6.7 Connect to Device TCP/UDP Ports 189
6.8 GDB Connection 191
6.9 vMMIO Introduction 192
6.10 Connect USB devices via USB/IP 196

7. Debug 234
7.1 Kernel Debugging Using GDB/LLDB 234
7.2 Debug Accelerator 236
7.3 CoreTrace 240

8. Use AVH via API 243
8.1 Quickstart for AVH REST API 243
8.2 API Token 250
8.3 API Usage Examples 251
8.4 GitHub Actions Example 252

9. Tips and Troubleshooting 258
9.1 Linux console is flooded with messages 258
9.2 Capture input from the Console 258
9.3 VPN keeps disconnecting 259
9.4 Deleting your AVH account 259

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1. Overview

Arm Virtual Hardware (AVH) for third-party boards provides cloud-based functionally accurate virtual models of popular IoT development kits, simulating CPU, peripherals, sensors and board components. It allows developers to execute the same binaries as on the real hardware and so leverage the board SDKs and software code examples. See AVH solutions page for a high-level overview.

This User Guide provides detailed information about AVH and its features, and is structured as follows:

- This chapter (**Overview**) is an entry point into AVH User Guide that outlines Access and Costs, and also includes Release Notes.
- Getting Started chapter references the available step-by-step instructions for getting started with AVH.
- Supported Platforms lists currently available board models, explains key platform-specific capabilities and usage examples.
- Device Controls gives an overview of the various controls available to users on their virtual devices, such as on-board LEDs and Buttons, lifecycle control, console and sensors.
- Device Firmware chapter covers the usage of firmware on AVH, how to package and upload it for execution on devices.
- Connect to Virtual Devices describes how to connect to your virtual devices remotely and how devices can communicate to each other.
- Debug chapter details the AVH capabilities for debugging your programs running on virtual devices.
- Use AVH via API explains how to use Arm Virtual Hardware programatically via API.
- Tips and Troubleshooting chapter contains useful tips and helps addressing common issues.

1.1 Access and Costs

Access to Arm Virtual Hardware (AVH) at avh.arm.com is available for users with a verified Arm account. If you do not have an Arm account you can register for it.

Once registered, you automatically have access to AVH and are enrolled on a 30-day free trial. This trial provides access for up to two active virtual devices at a time and up to 24-hours continuous virtual device uptime. After this trial period, the virtual device is automatically shut down and must be restarted by the user.

Following this trial period or if your usage exceeds the constraints of the trial period, you can activate your subscription. You will be charged based on usage at a rate of $1.50 per device-hour, where a device-hour is one hour of any one instance of Arm Virtual Hardware.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Discounts are available for open-source software development and other non-commercial, academic, or research usage. Volume discounts are also available based on either number of virtual devices or quantity of device hours.

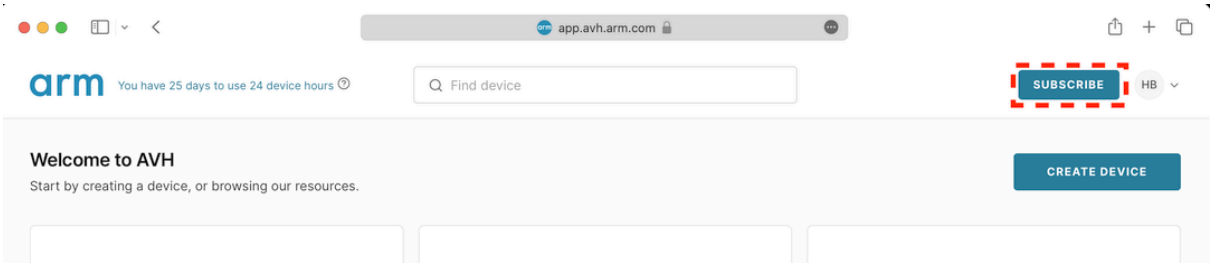
If you have trouble accessing the public beta or have any questions about the beta program, pricing, or discounts, contact us at avhinfo@arm.com.

1.2 Subscription

You can subscribe to AVH service to use your virtual devices beyond the free-trial limitations. After subscription you will be charged only based on the actual active time of your devices as explained in Access and Costs.

1. Log in to your account at https://app.avh.arm.com.
2. Once logged in, click the **SUBSCRIBE** button located on the top right next to your profile menu.

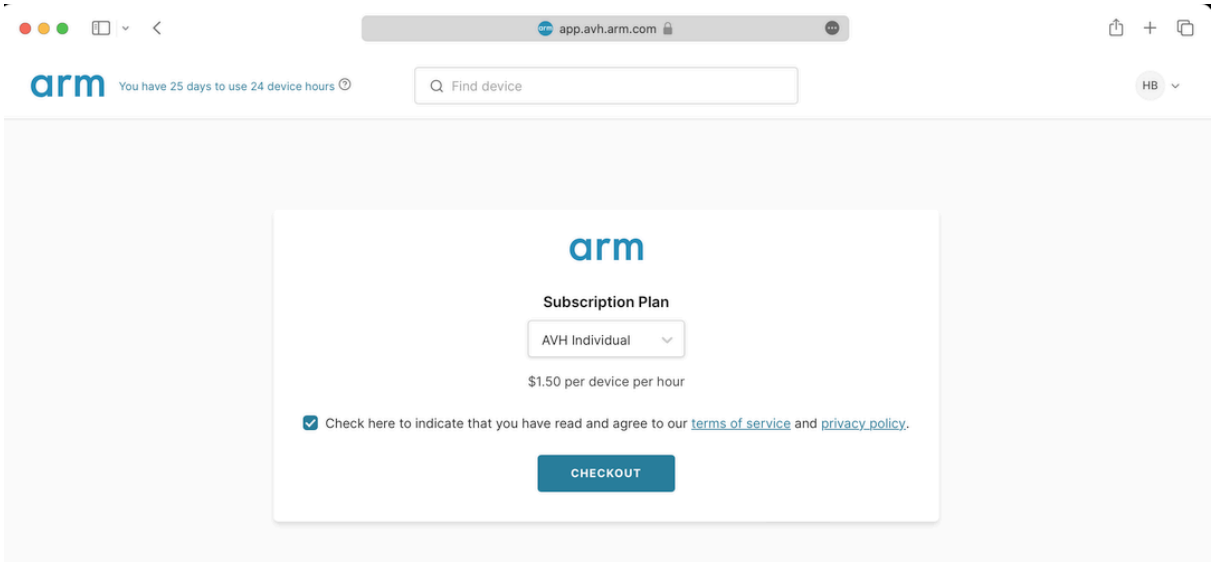
Figure 1-1: Starting subscription



3. Choose your subscription plan. Review the terms of service and privacy policy, and check the confirmation box. Click **CHECKOUT** button to proceed to the payment page.

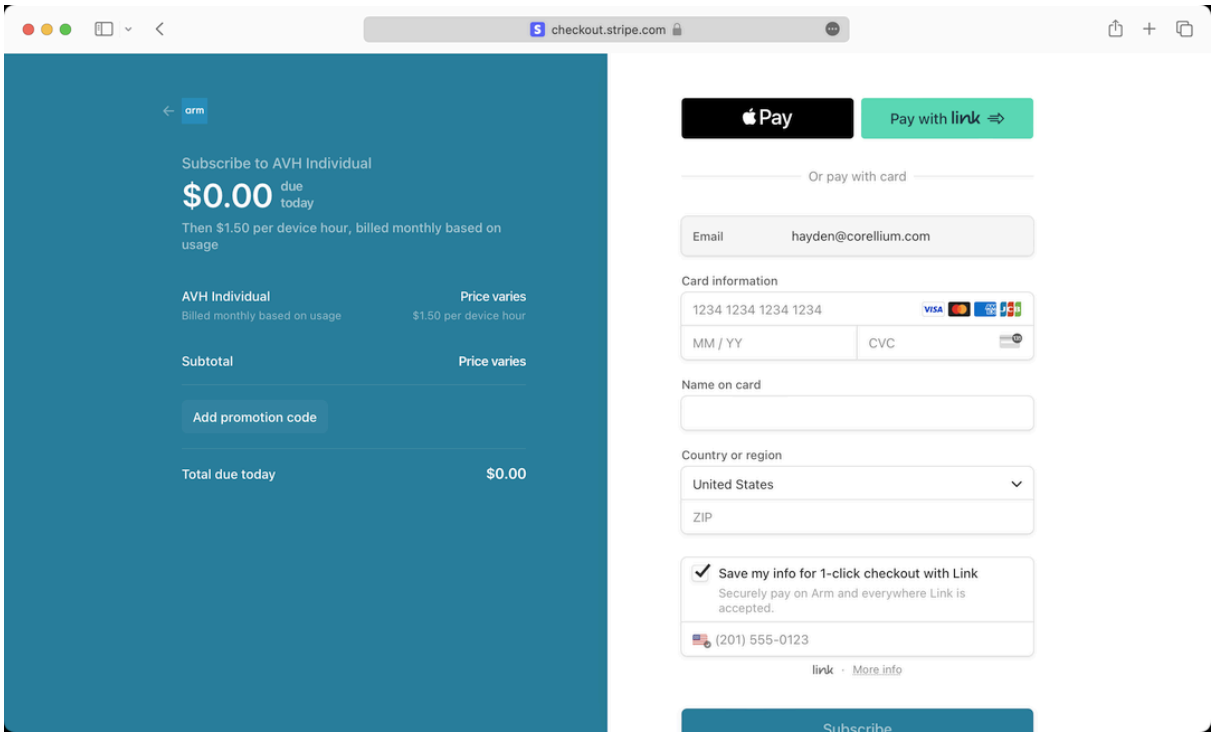
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-2: Choosing subscription plan



4. Once on the payment page, fill out the necessary fields with the appropriate information then click **SUBSCRIBE**.

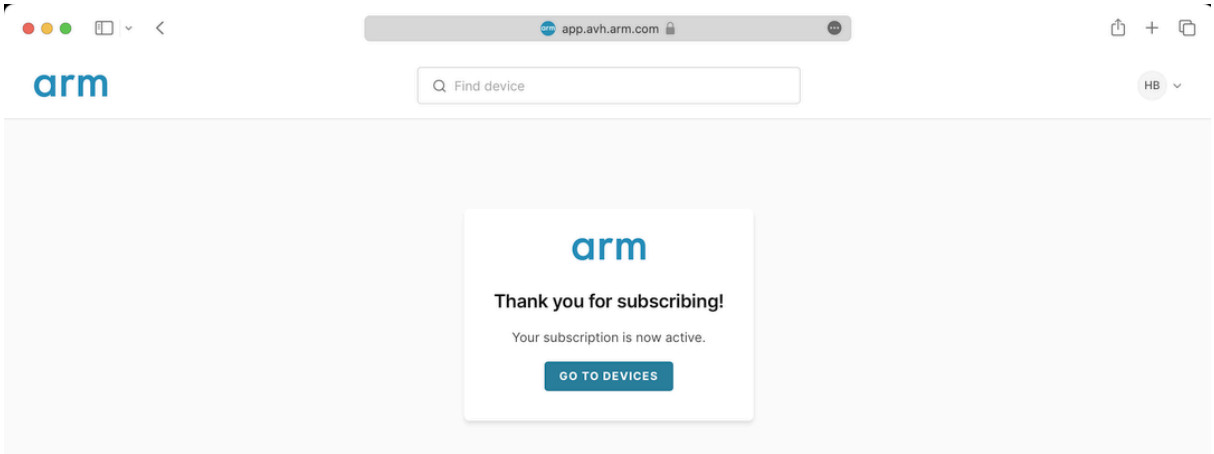
Figure 1-3: Payment page



5. You should see the message: “Thank you for subscribing!” indicating that your subscription was successful.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 1-4: Subscription confirmation



6. Navigate back to your account. The “subscription” button should not be displayed anymore and you can begin creating additional devices. You can find more information about your subscription plan by clicking the drop down menu to the right of your profile icon, then clicking “Admin” then “Account Management”.

1.3 Release Notes

The sections below list the changes introduced in corresponding updates of Arm Virtual Hardware for third-party boards.

Release 5.4.1

September 14, 2023

This minor release contains:

- Bug fixes

Release 5.4.0

September 8, 2023

AVH is proud to announce version 5.4.0:

- Graviton 3 support
- Android fingerprint peripheral support
- Improve the devices page content when no devices exist
- Improve the wording of “OS” to “firmware”
- Various bug fixes

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Release 5.1.0

June 22, 2023

AVH Release 5.1.0 has been shipped and contains the following improvements:

- Team accounts are now available for select customers. Contact Arm for more details.
- Announcing the Arm Smart Vision Configuration Kit, available to create now.
- Migrated all AVH support docs to our Developer Site.
- Announcing the new Debug Accelerator! Find it in the Connect section on your virtual device.
- Various bug fixes and improvements.

Release 5.0.0

May 10, 2023

Arm Virtual Hardware is now in public beta status.

Release 4.3.0

March 31, 2023

The following improvements are introduced with this release:

- Add vMMIO support to the STM32 and i.MX 8M boards
- Add a new button to download the console logs
- Upgrade Android devices to Python 3.11.2 and Frida 16.0.11
- Fix a bug in the STM32 model that could have caused it to enter a bad state when kernel debugging
- Stability improvements and various bug fixes

Release 4.1.4

March 21, 2023

AVH announces the following changes with this release:

- Increased snapshot limit to 10 snapshots per virtual device
- Improvements to the Remote.it integration
- Various UIUX enhancements

Release 4.1.3

February 16, 2023

This minor release contains the following change:

- Updated the Raspberry Pi Model to mitigate a Linux Kernel networking race condition

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Release 4.1.2

February 11, 2023

AVH is excited to announce the following changes with this release:

- NXP i.MX93 board added
- FVP models for the following systems have been added:
  - Corstone-310
  - Corstone-300
  - Cortex-M0
  - Cortex-M0+
  - Cortex-M3
  - Cortex-M4
  - Cortex-M7
  - Cortex-M23
  - Cortex-M33
  - Cortex-M55
  - Cortex-M85
- UI/UX enhancements
- Raspberry Pi input interface improvements
- Remote.it agent is now automatically included in Raspberry Pi OS
- vMMIO documentation updated
- CoreTrace enhanced to support additional boards
- System stability improvements

Release 3.15.0

December 14, 2022

AVH is excited to announce initial Remote.it support, a new Ubuntu Server firmware for the Raspberry Pi 4 board, and improvements to our web interface.

- Various bug fixes
- Frontend enhancements
- Process streamlining improvements

Release 3.14.1

October 28, 2022

AVH introduces new i.MX 8M firmware with AWS IoT Greengrass and STM32U5 board audio file playback simulating microphone input

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Improved Bluetooth support for select models
- Configuration updates
- Enabled kernel debug option
- Firmware package updates
- Various bug fixes

Release 3.14.0

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

Release 3.11.0

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
- Updates to the JavaScript and Python APIs
- Various fixes and improvements to the user interface

Release 3.10.0

May 16, 2022

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

This 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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2. Getting Started

Following step-by-step guides describe how to start with a specific platform or feature on Arm Virtual Hardwre:

- Quickstart for Raspberry Pi 4
- Quickstart for STM32U5 IoT Discovery
- Quickstart for i.MX 8M Arm Cortex Complex
- Quickstart for i.MX 93 Applications Processor
- Quickstart for AVH REST API

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. Supported Platforms

Arm Virtual Hardware (AVH) for third-party boards provides following simulations models:

- Raspberry Pi 4
- i.MX 8M Arm Cortex Complex
- i.MX 93 Applications Processor
- STM32U5 IoT Discovery Kit
- Arm Smart Vision Configuration Kit
- Arm Fixed Virtual Platforms
- Generic Android

Navigate to the platform-specific sections for details about corresponding virtual model and its usage examples.

3.1 Raspberry Pi 4

Arm Virtual Hardware (AVH) provides a board model for the Raspberry Pi 4 Model B. In the AVH documentation this virtual board is referenced as “Raspberry Pi 4” or “Rpi 4” in short form.

This chapter guides through usage specific to the virtual Raspberry Pi 4 board. It explains how to get started, lists the hardware features supported in the model and provides usage examples relevant for it. The chapter is stuctured as follows:

- **Getting started:**
  - Quickstart for Raspberry Pi 4 provides a simple step-by-step guide on setting up a virtual Rpi 4 board.
- **Board model details:**
  - Supported Hardware Components for Raspberry Pi 4 lists the hardware components supported in the virtual Raspberry Pi 4 model.
- **Firmware:**
  - Stock Firmware for Raspberry Pi 4 explains the ready-to-use firmware packages available for virtual Raspberry Pi 4 on AVH.
  - Custom Firmware for Raspberry Pi 4 describes how to package and run a custom Linux firmware on a virtual Raspberry Pi 4.
  - Storage Files for Raspberry Pi 4 provides information about the storage files in AVH firmware packages for Raspberry Pi 4.
- **Usage examples:**
  - Connect to Wi-Fi on Raspberry Pi 4 explains how to connect a virtual Rpi 4 board to a WiFi network available in the AVH environment.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Connect to BLE on Raspberry Pi 4 describes how to use Bluetooth Low Energy (BLE) for communication between a virtual Rpi 4 board and a virtual Android phone.
- Use vMMIO on Raspberry Pi 4 provides an example for using virtual Memory Mapped I/O (vMMIO) for external handler.
- Updating Linux Kernel on Raspberry Pi 4 explains how to update the Linux Kernel and helps fixing potential issues.
- Running Docker on Raspberry Pi 4 demonstrates how to run Docker.

3.1.1 Quickstart for Raspberry Pi 4

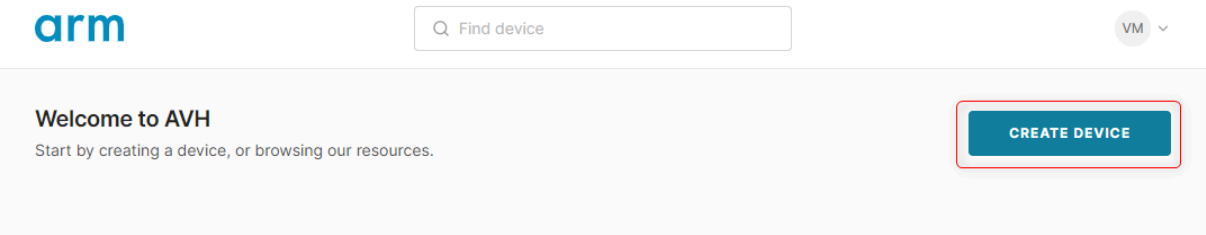
This page guides you through the steps for creating a virtual Raspberry Pi 4 device with a stock firmware `Raspberry Pi OS lite (11.2.0)`, and shows how to access its Linux command line interface.

Create your device

To create your virtual Raspberry Pi 4 device, do the following:

1. Log in to AVH with your Arm account at https://app.avh.arm.com. If you do not have an Arm account you can register here.
2. After login you should land on **Devices** page. There click **CREATE DEVICE**:

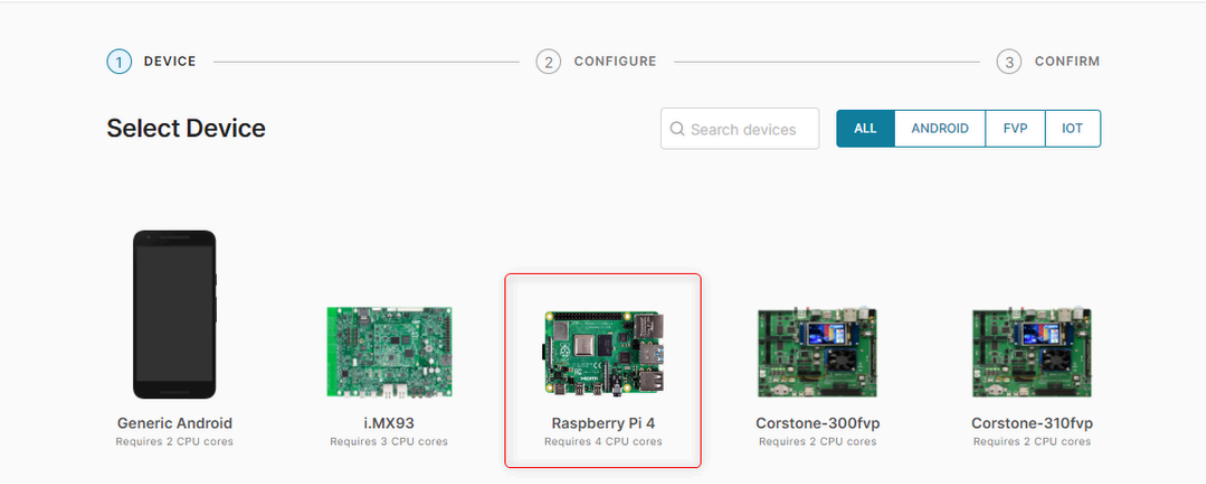
   Figure 3-1: Create device



3. On the **Select Device** view click on the **Raspberry Pi 4** board to proceed with creation of corresponding virtual instance.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-2: Select Raspberry Pi 4

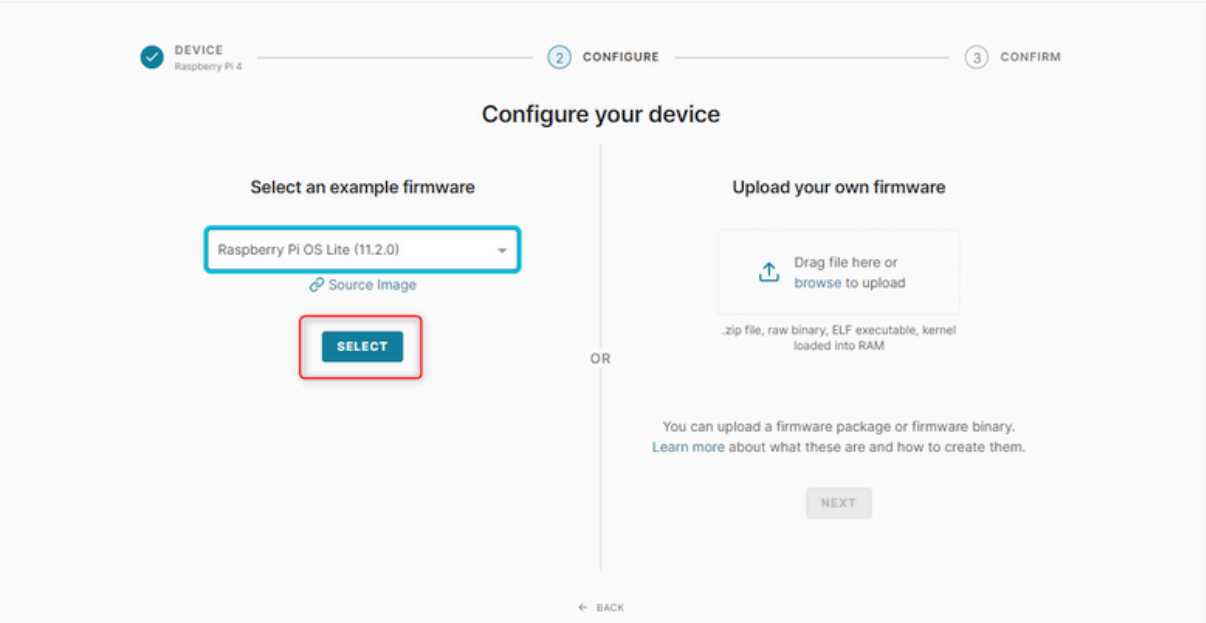


Note

You can narrow down the list of displayed boards by selecting the target category on the top-right side between **| ALL | ANDROID | FVP | IOT**. The Raspberry Pi 4 board is available under IOT and ALL tabs.

4. On the **Configure your device** screen, you will see a dropdown menu **Select an example firmware** with stock firmware options available for the target platform in AVH, see Stock Firmware for Raspberry Pi 4 for more details. For this quickstart example, choose `Raspberry Pi OS lite (11.2.0)` and click **SELECT**.

Figure 3-3: Configure your device



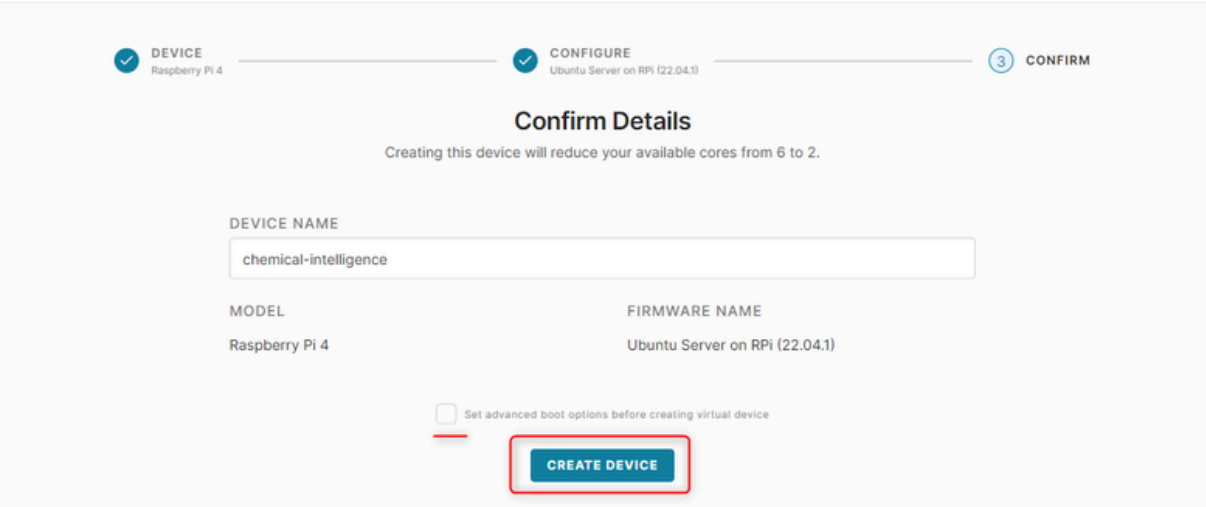
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

It is also possible to upload custom firmware, see Custom Firmware for Raspberry Pi 4 for more information.

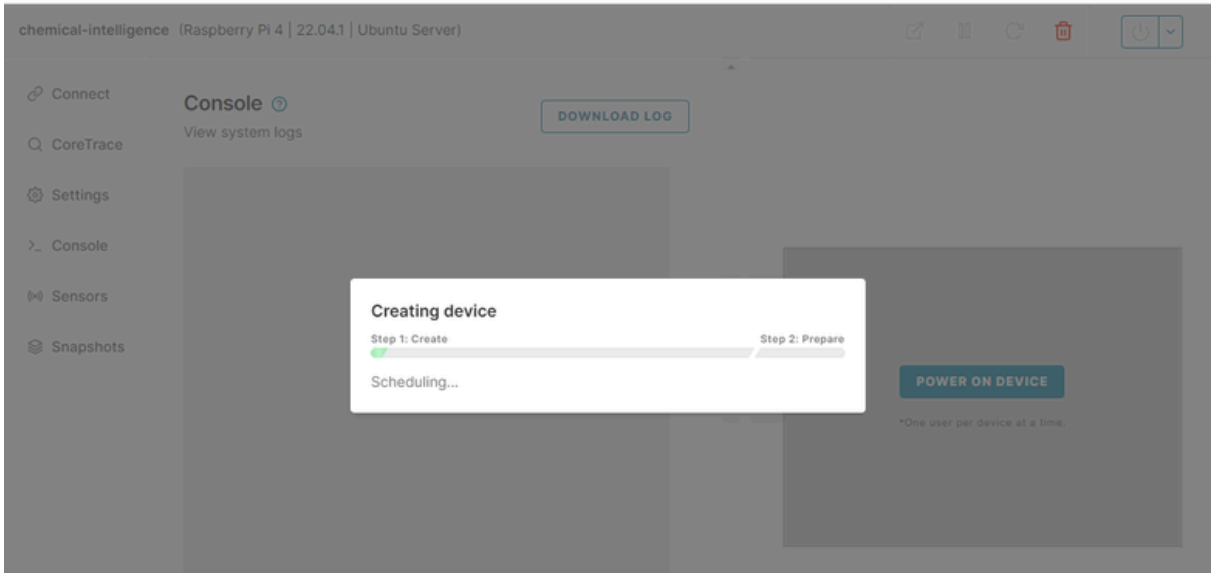
5. On the **Confirm Details** screen, you can update the device name if needed, verify your selections, and then click **CREATE DEVICE**. Keep the advanced boot options unchecked.

Figure 3-4: Confirm details



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

Figure 3-5: Creating device



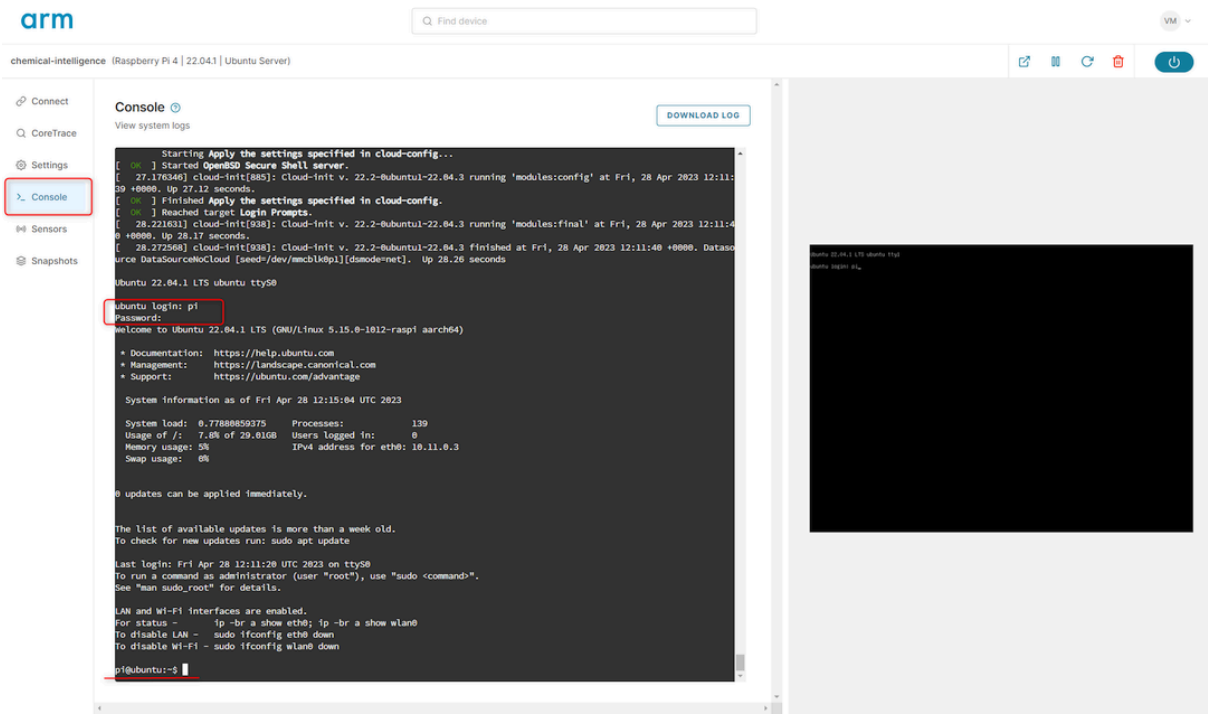
7. Your Raspberry Pi 4 device is now created!

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Log in to console

1. Let's log in to the Linux command line interface on the device. Click **Console** tab in device's menu, if not open by default. Type in the default username: `pi` and password: `raspberry`, then hit the Return/Enter key. You should see the display name change to `pi@rasberrypi`, meaning you are logged in.

   Figure 3-6: Console



   Alternatively, you can also use the graphical console on the left to log in.

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

Next steps

For next steps see Raspberry Pi 4 overview page with references to more usage examples.

3.1.2 Supported Hardware Components for Raspberry Pi 4

The virtualized Raspberry Pi 4 board models the following hardware components:

- **Processor**
  - Broadcom BCM2711B0: 4x Arm Cortex-A72, 64-bit
- **RAM**
  - 1GB, 2GB, or 4GB LPDDR4 SDRAM
- **GPU**

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

  - Broadcom VideoCore VI (partially modeled)
- **Networking**
  - Gigabit Ethernet
  - WiFi (virtual), see Connect to Wi-Fi on Raspberry Pi 4
  - BLE (virtual), see Connect to BLE on Raspberry Pi 4
- **GPIO**
  - 40-pin GPIO header, populated (user can model using vMMIO)
- **Storage**
  - microSD, see Storage Files for Raspberry Pi 4
- **Ports**
  - 2 × micro-HDMI 2.0
  - 2 × USB 2.0
  - 2 × USB 3.0
  - 3.5 mm analog audio-video jack
  - Camera Serial Interface (CSI)

3.1.3 Stock Firmware for Raspberry Pi 4

Arm Virtual Hardware provides several ready-to-use firmware packages (stock firmware) to run on virtual Raspberry Pi 4 devices. These are zip archives prepared according to the AVH Linux firmware package format explained in Linux Firmware Packages. Following stock firmware packages are available:

- **Ubuntu Server on RPi (22.04.1)** firmware runs the distribution of Ubuntu Server 22.0.4 LTS for Raspberry Pi 4. Learn more about this OS at Ubuntu for Raspberry Pi.
- **Raspberry Pi OS Lite (11.2.0)** firmware contains the lightweight variant of Raspberry Pi OS based on Debian 11 (bullseye). See Raspberry Pi OS images and Raspberry Pi OS documentation to learn more.
- **Raspberry Pi OS Desktop (11.2.0)** firmware provides the Desktop variant of Raspberry Pi OS based on Debian 11 (bullseye). See Raspberry Pi OS images and Raspberry Pi OS documentation to learn more.

When creating a new virtual board (see example in Quickstart for Raspberry Pi 4), on the **Configure your device** screen in drop-down menu **Select an example firmware** you can choose the target stock firmware that the virtual board instance shall run.

Alternatively, you can also run Custom Firmware for Raspberry Pi 4.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.1.4 Custom Firmware for Raspberry Pi 4

This page explains how to create, package, and run a custom Linux firmware on a virtual Raspberry Pi 4 board. Ubuntu Server and Ubuntu Desktop images are used as examples as provided directly by Ubuntu. The demonstrated flow can be applied for running other Linux distributions as well.

Firmware Package Contents

For the overall information about firmware packaging see Linux Firmware Packages.

A proper firmware package for a virtual Rapberry Pi 4 board contains following files:

- `Info.plist` - for the meta information in JSON format about firmware version, type, build, unique identifier, and device identifier.
- `nand` - with the preinstalled 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.

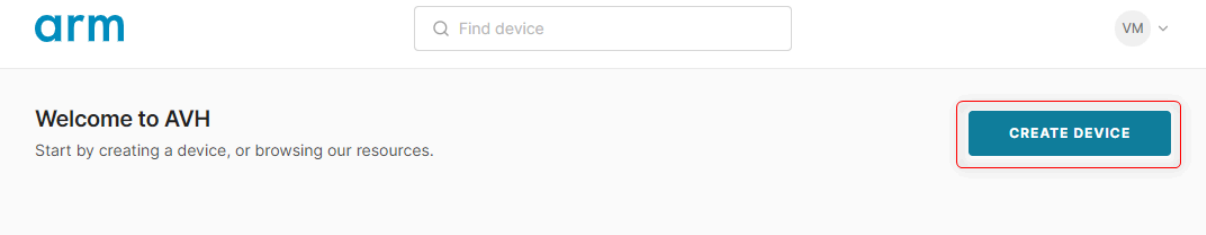
Although specifying the `ramdisk.img` is generally optional, we need to add a reboot command in this case because of how this virtual device handles the first pass.

Install the Package

The steps below are specific the firmware package with Ubuntu Server, but can be followed in the same way to insall Ubuntu Desktop:

1. In your local Linux environment run the shell script for creating the Ubuntu Server image as given in Firmware Packaging Scripts. It shall create firmware package file `rpi4b-ubuntu-server.zip`.
2. On the AVH web interface, click **CREATE DEVICE**.

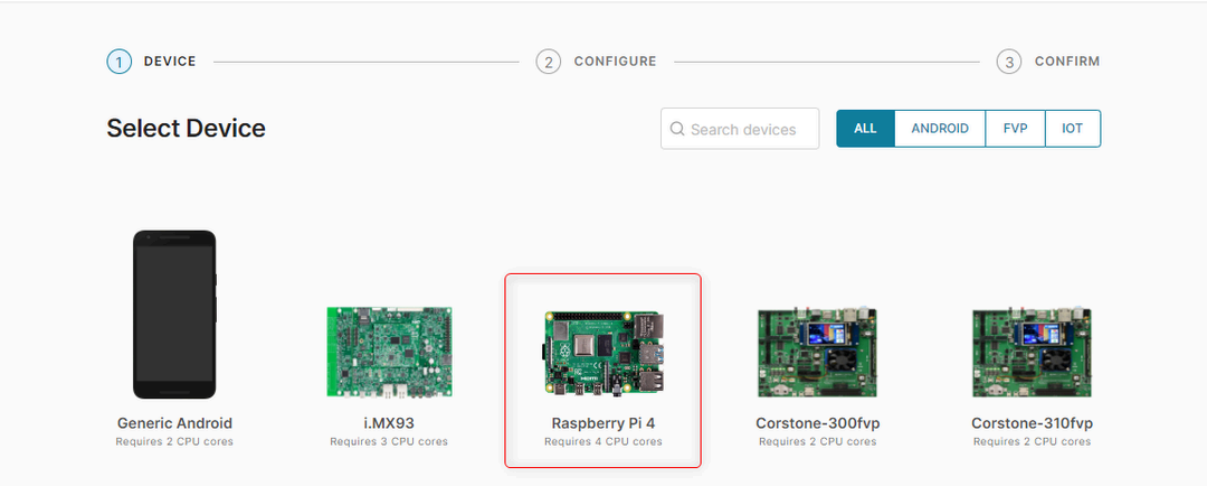
   Figure 3-7: Create device



3. Click on the Raspberry Pi 4 board.

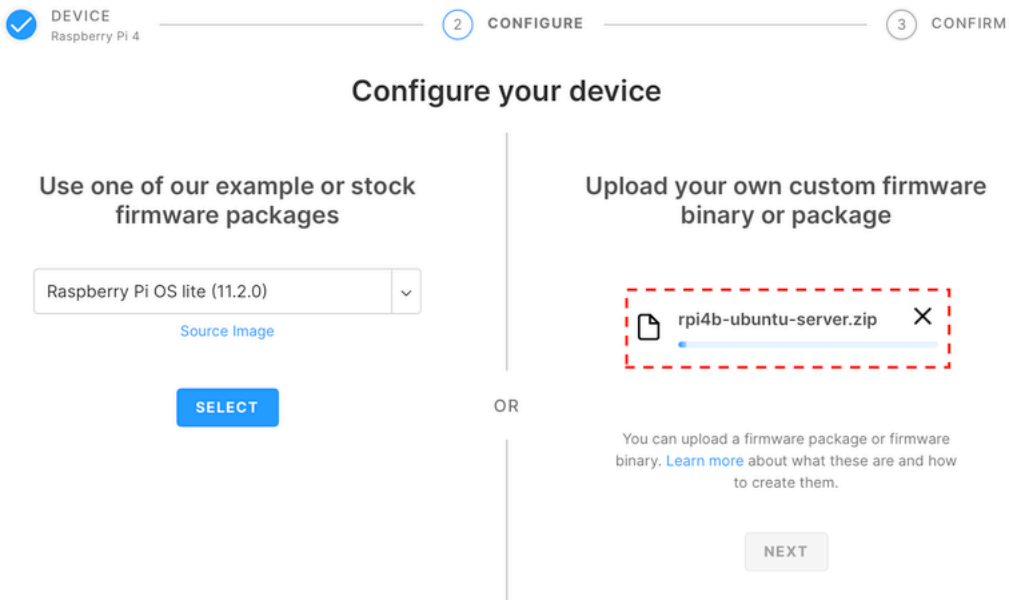
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-8: Select device



4. Upload the custom firmware package.

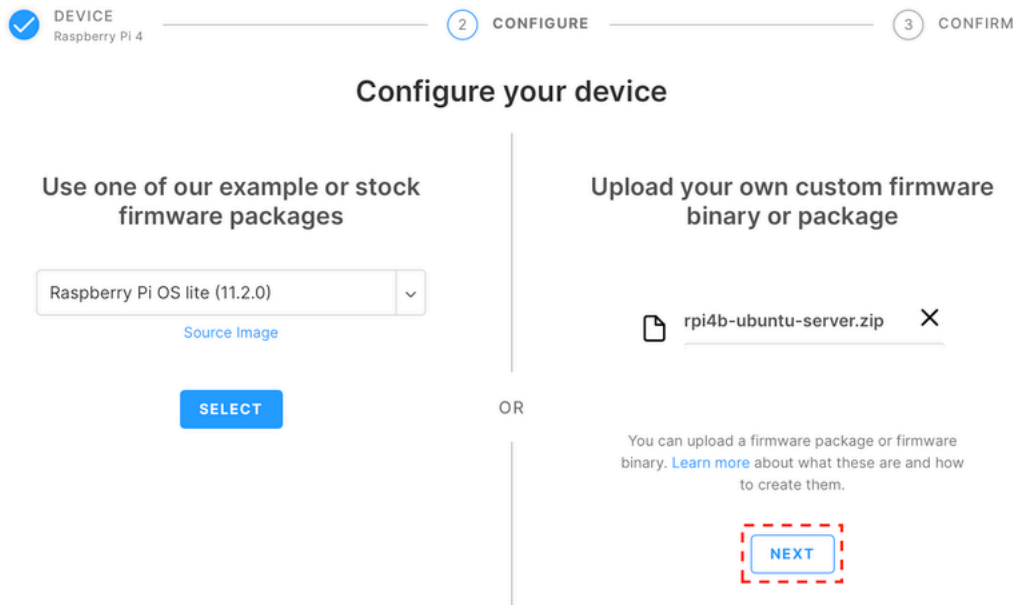
Figure 3-9: Upload the custom firmware package



5. When the process is complete, click **NEXT**.

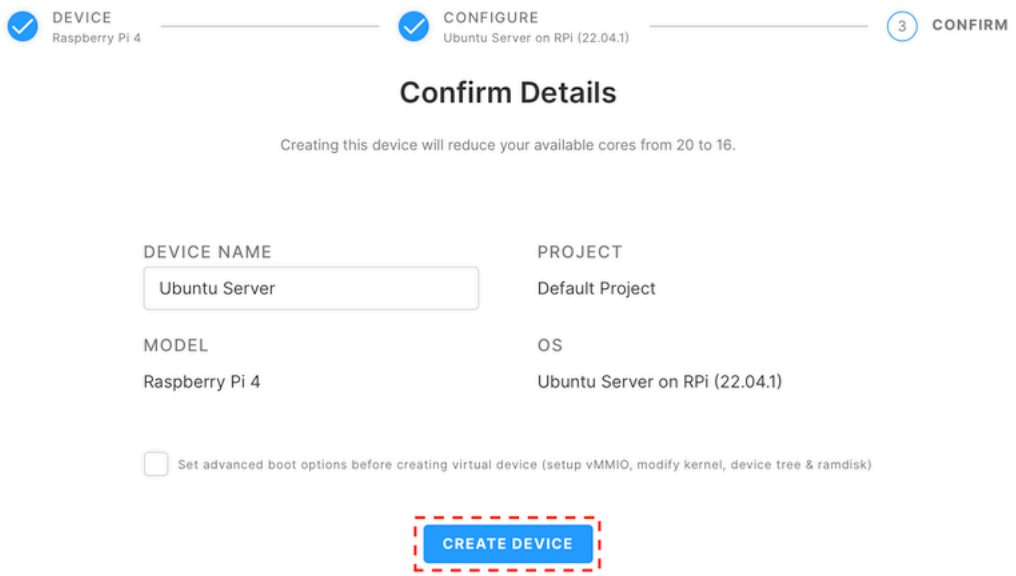
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-10: Click Next



6. Create the device without enabling advanced boot options.

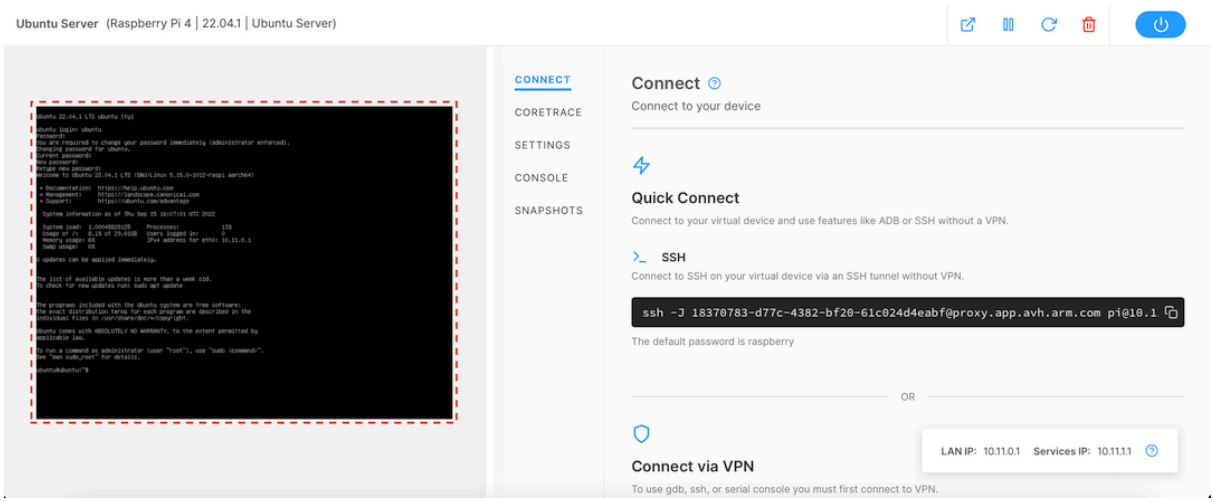
Figure 3-11: Create device



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-12: Login with the default credentials



Note

The SSH commands displayed for the virtual Raspberry Pi 4 instance under the **Connect** tab show information for the stock firmware. With custom package used in this example you need to use `ubuntu` as the username instead of the `pi` shown by default.

Firmware Packaging Scripts

Ubuntu Server image:

The following script creates a custom firmware package `rpi4b-ubuntu-server.zip` in your working directory using the Ubuntu Server 22.04.1 image as provided by Ubuntu.

```
#!/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 "${LO}p1" boot
# Copy the rpi4b devicetree
cp boot/bcm2711-rpi-4-b.dtb devicetree
# Extract the Linux kernel
zcat boot/vmlinuz > kernel
# Extract initrd
lz4 -d boot/initrd.img initrd.cpio
umount boot
rm -r boot/
losetup -d "${LO}"
mkdir ramdisk
cd ramdisk
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
# 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/
```

Ubuntu Desktop image:

The following script creates a custom firmware package `rpi4b-ubuntu-desktop.zip` in your working directory using the Ubuntu Desktop 22.04.1 image as provided by Ubuntu.

```
#!/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
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
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>
<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/
```

3.1.5 Storage Files for Raspberry Pi 4

The following storage files can be used in custom firmware packages for virtual Raspberry Pi 4.

- **nand** file with content for the SD card
  - block_size: 512
  - blocks: 31250000
  - max: 16000000000 bytes

Also see Custom Firmware for Raspberry Pi 4.

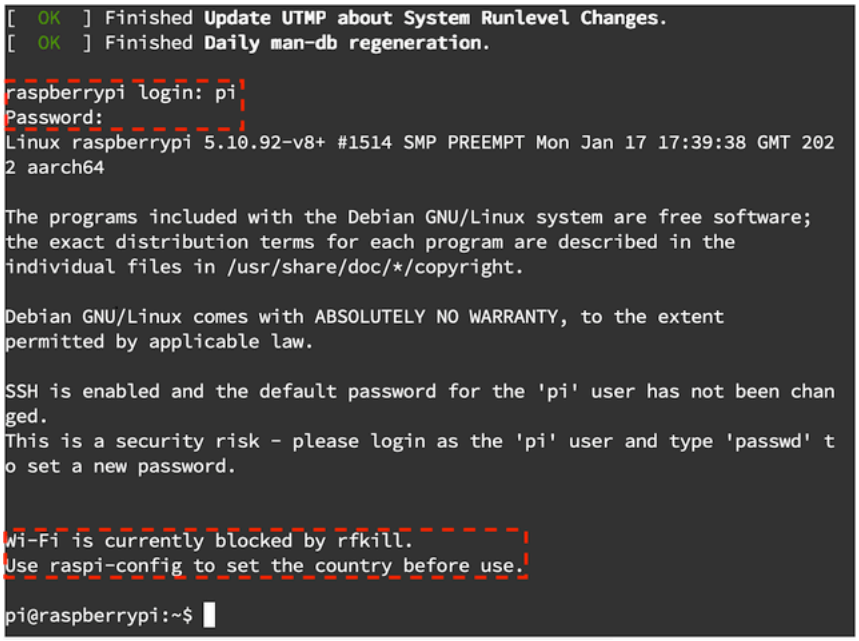
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.1.6 Connect to Wi-Fi on Raspberry Pi 4

The virtual Raspberry Pi 4 board by default connects to the AVH LAN over its Ethernet interface. You can also set up a connection over the emulated Wi-Fi interface, as demonstrated below.

1. Create a Raspberry Pi 4 device (we use the "lite" stock firmware in this example) and log in using the default credentials `(pi/raspberry)`. Quickstart for Raspberry Pi 4 explains this step in more details.

   Figure 3-13: Raspberry Pi login



2. Open the Raspberry Pi configuration tool `raspi-config`.

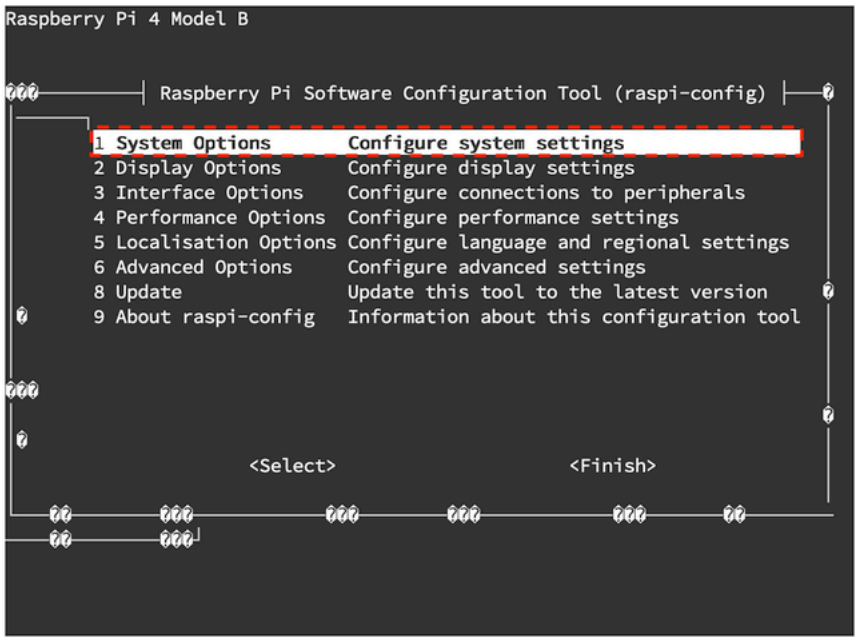
```
sudo raspi-config
```

See raspi-config documentation to learn more about the configuration tool.

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

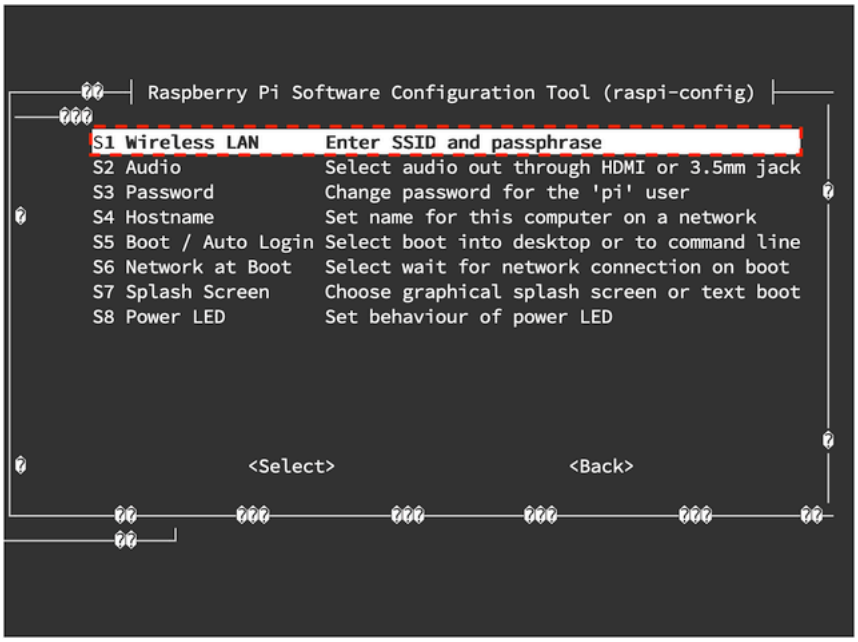
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-14: Configure main screen



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

Figure 3-15: Configure system options



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-16: Set coutry

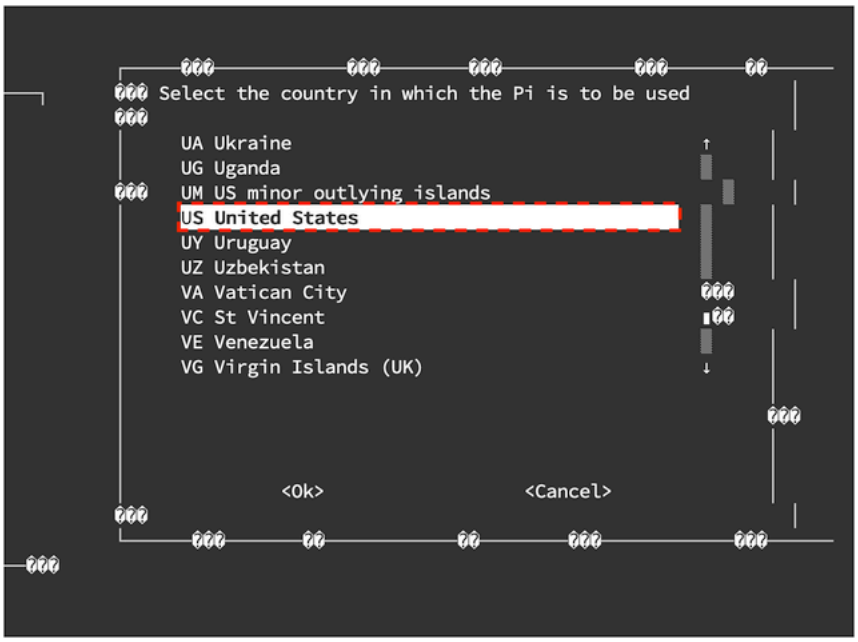
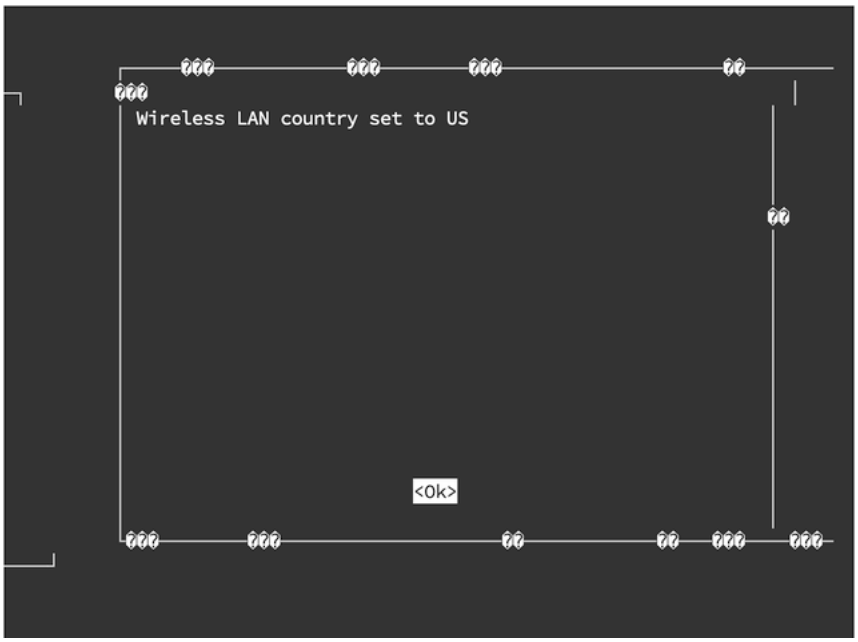


Figure 3-17: Set coutry confirmation



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-18: Set SSID

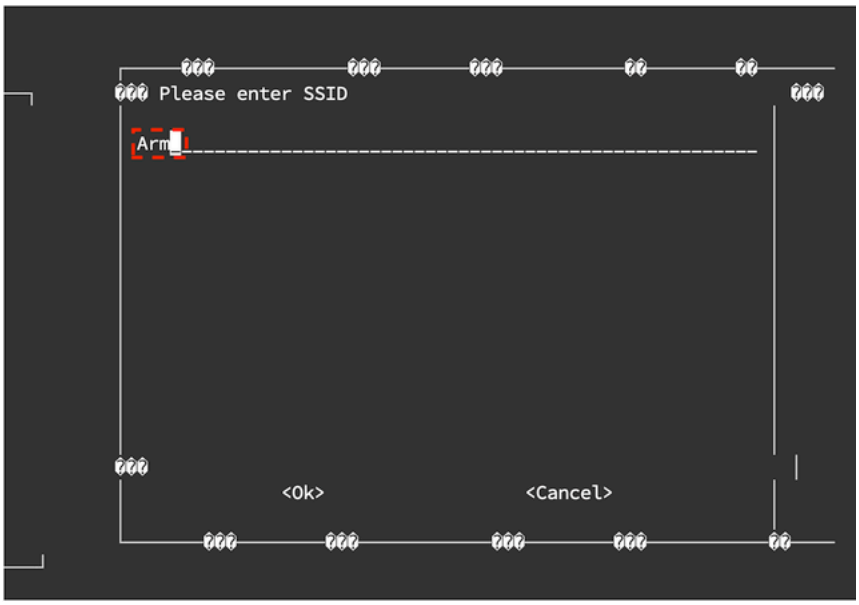
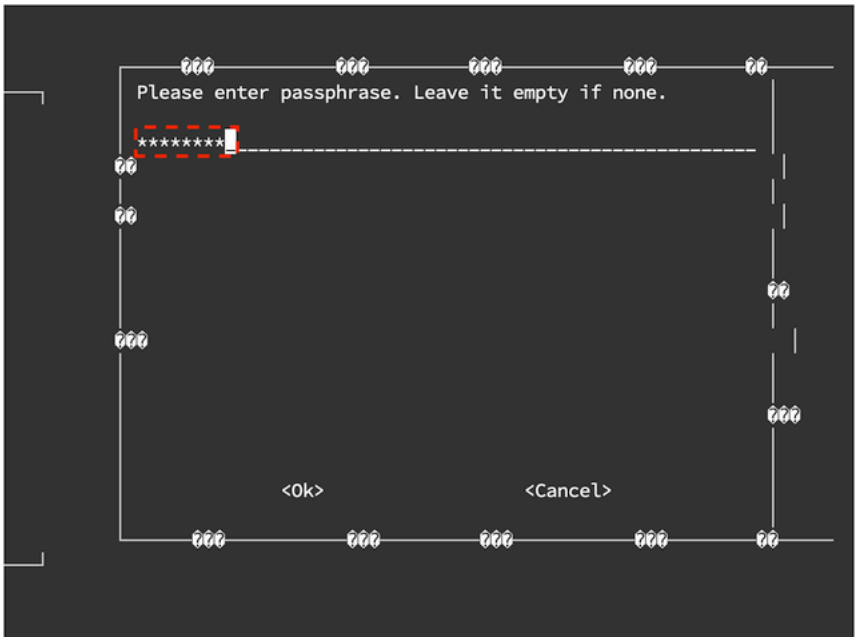


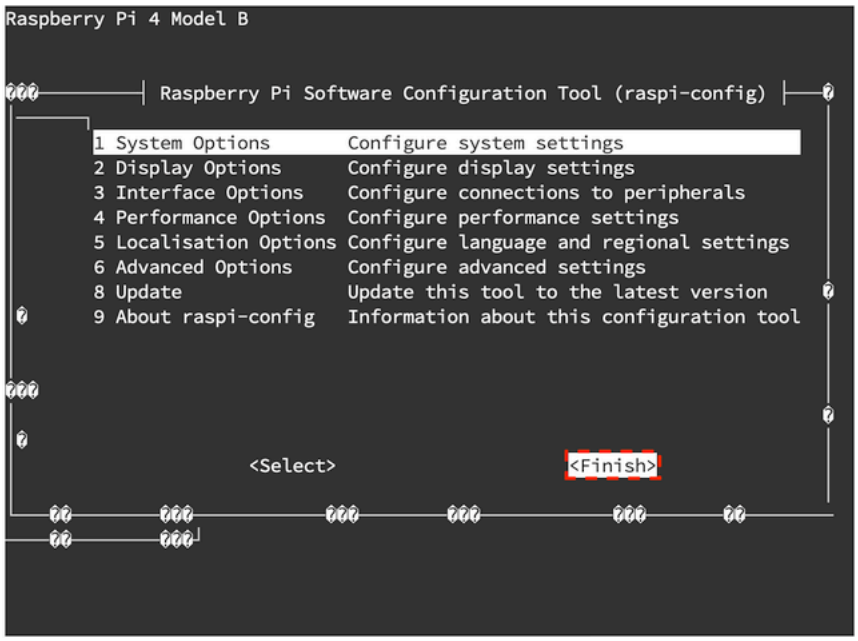
Figure 3-19: Set passphrase



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

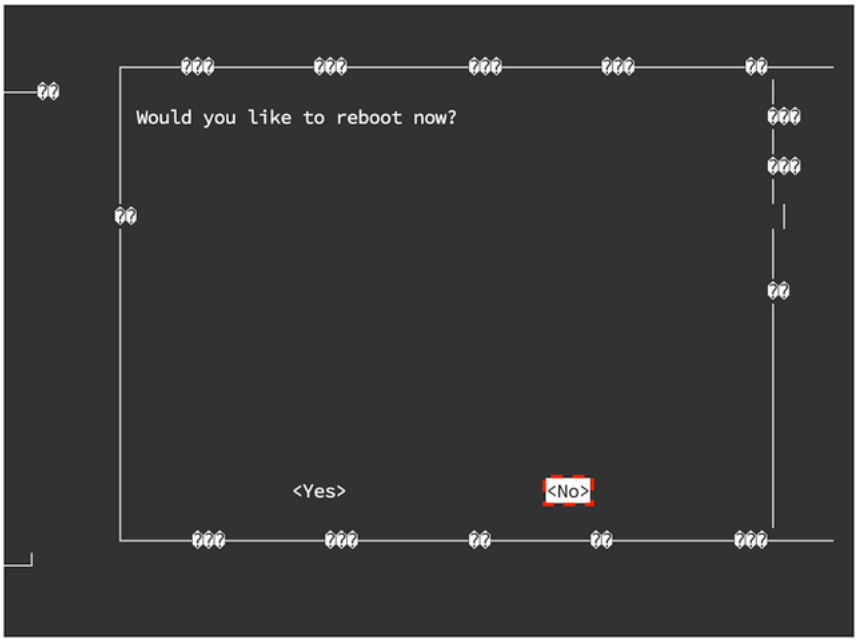
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-20: Choose finish



6. Do not reboot the machine.

Figure 3-21: Decline to reboot

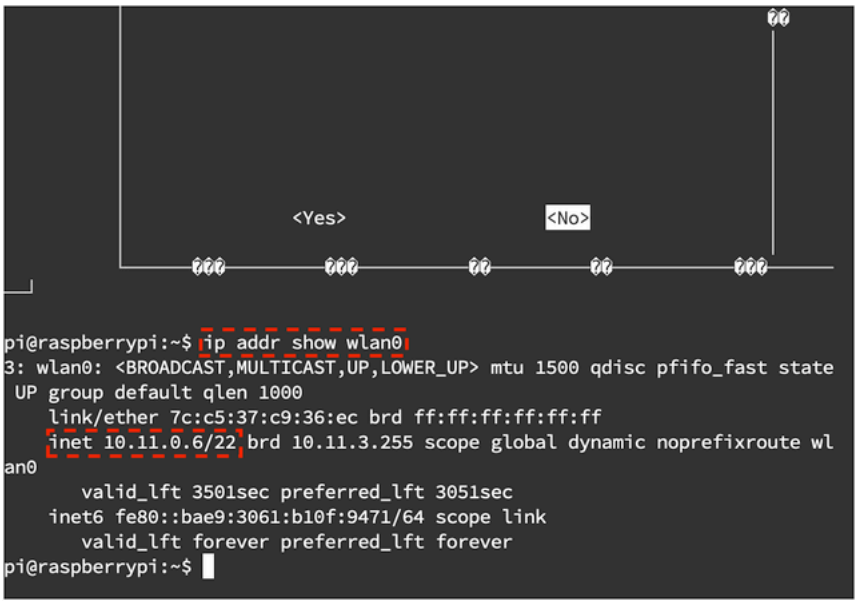


7. Verify that the wlan0 interface shows a valid IPv4 address that does not start with 169. Please note: the IPv6 address for this interface is a local-link auto address and not a real IPv6 address.

```
ip addr show wlan0
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

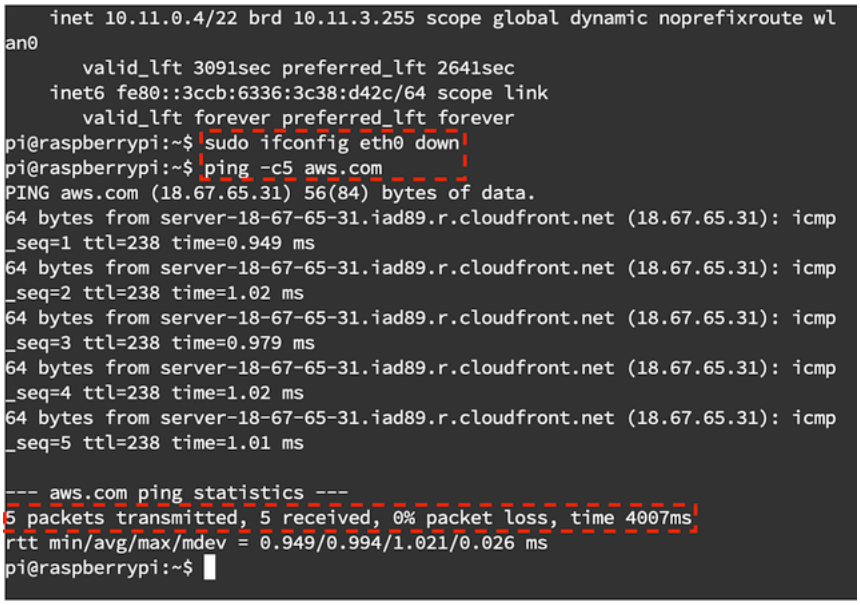
Figure 3-22: Show wlan0



8. 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
```

Figure 3-23: Zero packet loss



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

```
sudo ifconfig eth0 up
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.1.7 Connect to BLE on Raspberry Pi 4

Learn how to use Bluetooth Low Energy (BLE) on a virtual Raspberry Pi 4 and Generic Android 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

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.

Use BLE Between Two Raspberry Pi Devices

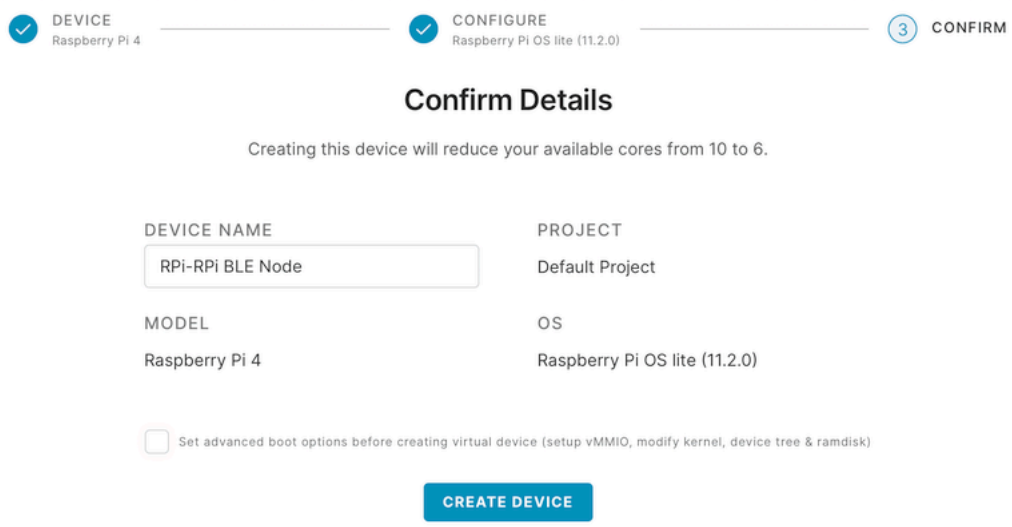
In this example, we will set up a Raspberry Pi 4 BLE Node and a Raspberry Pi 4 BLE Master.

Set Up the Raspberry Pi 4 BLE Node

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

1. Create a Raspberry Pi 4 device. In this example we will use the “lite” stock firmware, and name the device “RPi-RPi BLE Node”. After device is created, log in using the default credentials `(pi/raspberry)`. Quickstart for Raspberry Pi 4 explains this step in more details.

   Figure 3-24: Raspberry Pi node name



2. Open the Bluetooth tool with `bluetoothctl1` console command:

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-25: Raspberry Pi open Bluetooth

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

3. Turn on Bluetooth with `power on` command:

Figure 3-26: Power on Bluetooth

```
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 using comamnd `menu advertise`:

Figure 3-27: Menu advertise

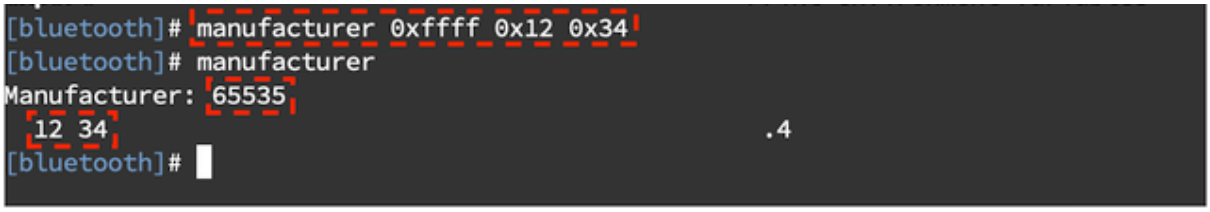
```
[bluetooth]# menu advertise
Menu advertise:
Available commands:
-------------------
uuids [uuid1 uuid2 ...]                          Set/Get advertise uuids
service [uuid] [data=xx xx ...]                  Set/Get advertise service dat
a
manufacturer [id] [data=xx xx ...]               Set/Get advertise 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
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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

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

Figure 3-28: Manufacturer node



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

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

Figure 3-29: Set name node

7. Go back to the main menu using `back` command:

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-30: 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 with `menu gatt` command:

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-31: 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
```

Figure 3-32: Register a new service

```
[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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-33: Register a read-only characteristic

```
[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
```

Figure 3-34: Register a read-write characteristic

```
[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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-35: 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-36: 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
```

Figure 3-37: Advertise node

```
[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.

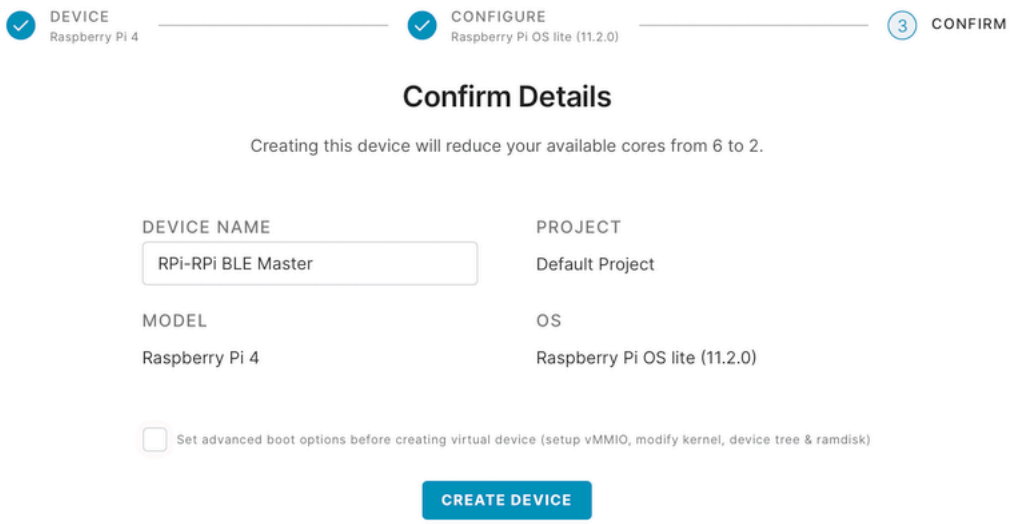
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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".

   Figure 3-38: Create Raspberry Pi 4 instance



2. Log in using the default credentials (`pi/raspberry`) and open the Bluetooth tool using `bluetoothctl1` command:

   Figure 3-39: Raspberry Pi Login

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

3. Turn on Bluetooth with `power on`:

   Figure 3-40: Power on Bluetooth

```
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 with `scan on` command.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-41: Scan node

```
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
```

Figure 3-42: 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-43: 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 with `menu gatt` command.
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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-44: 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 with `read` command. You should see the value you set previously, represented in hexidecimal.

Figure 3-45: 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 with `write` command. You should see an error because the characteristic is read-only.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-46: 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-47: 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 with `read` command. You should see the value you set previously, represented in hexidecimal.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-48: 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-49: 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]#
```

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.

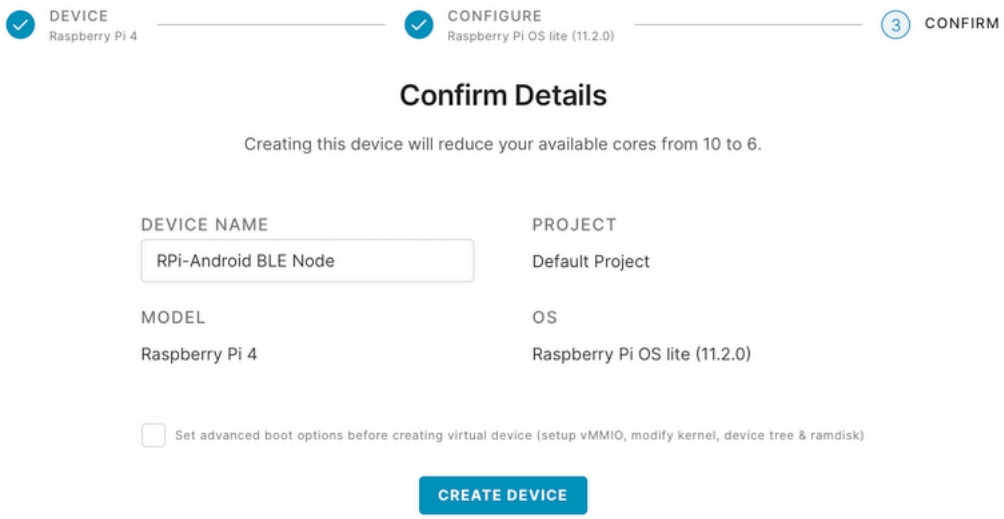
Set Up the Raspberry Pi 4 BLE Node

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-50: 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-51: 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]#
```

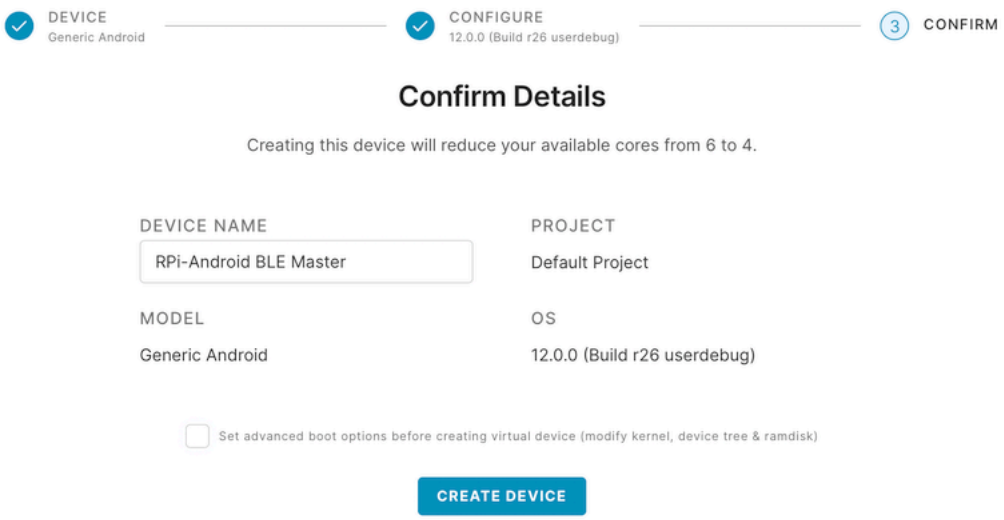
Connect to the Node with the Android BLE Master

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

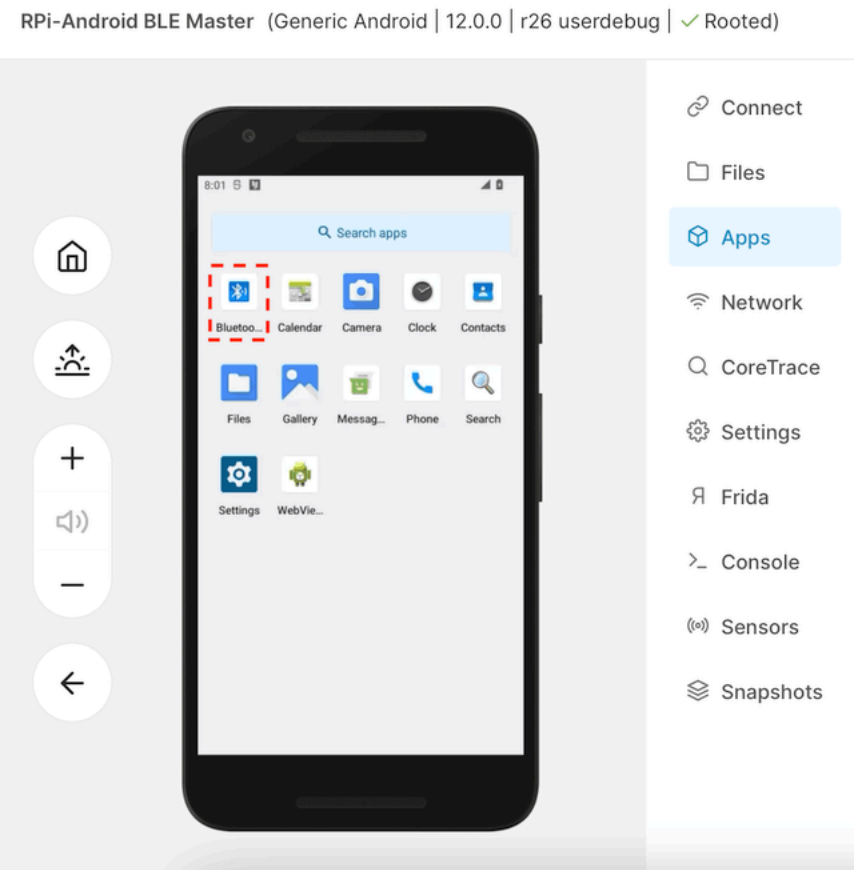
Figure 3-52: Create Raspberry Pi 4 instance



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

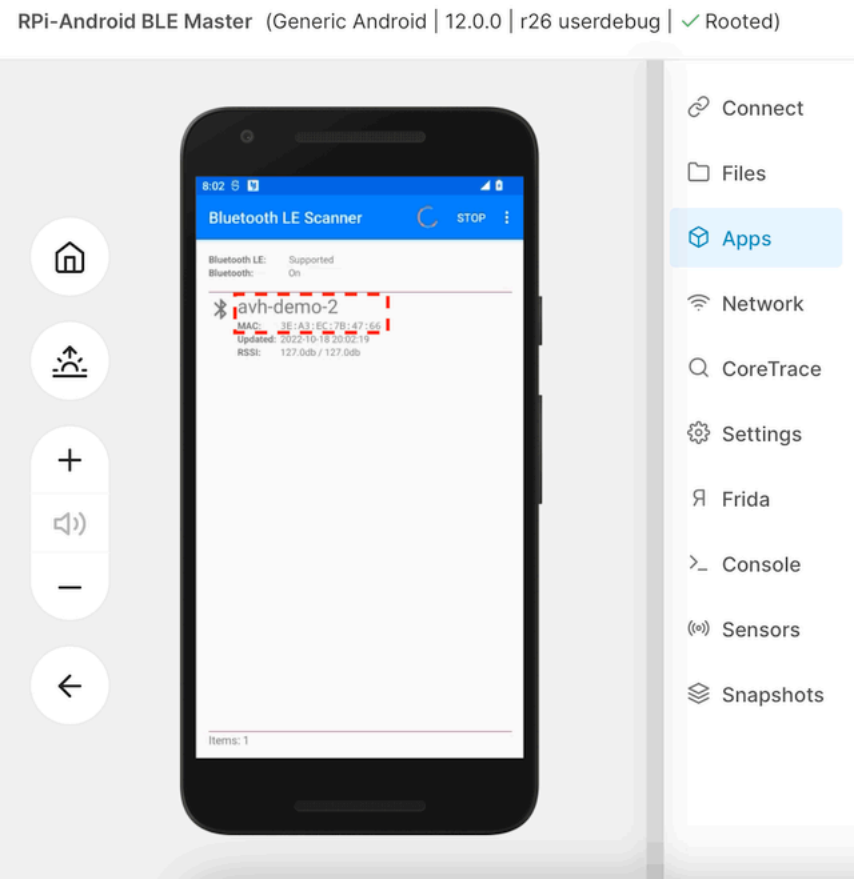
Figure 3-53: Install Bluetooth LE scanner



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

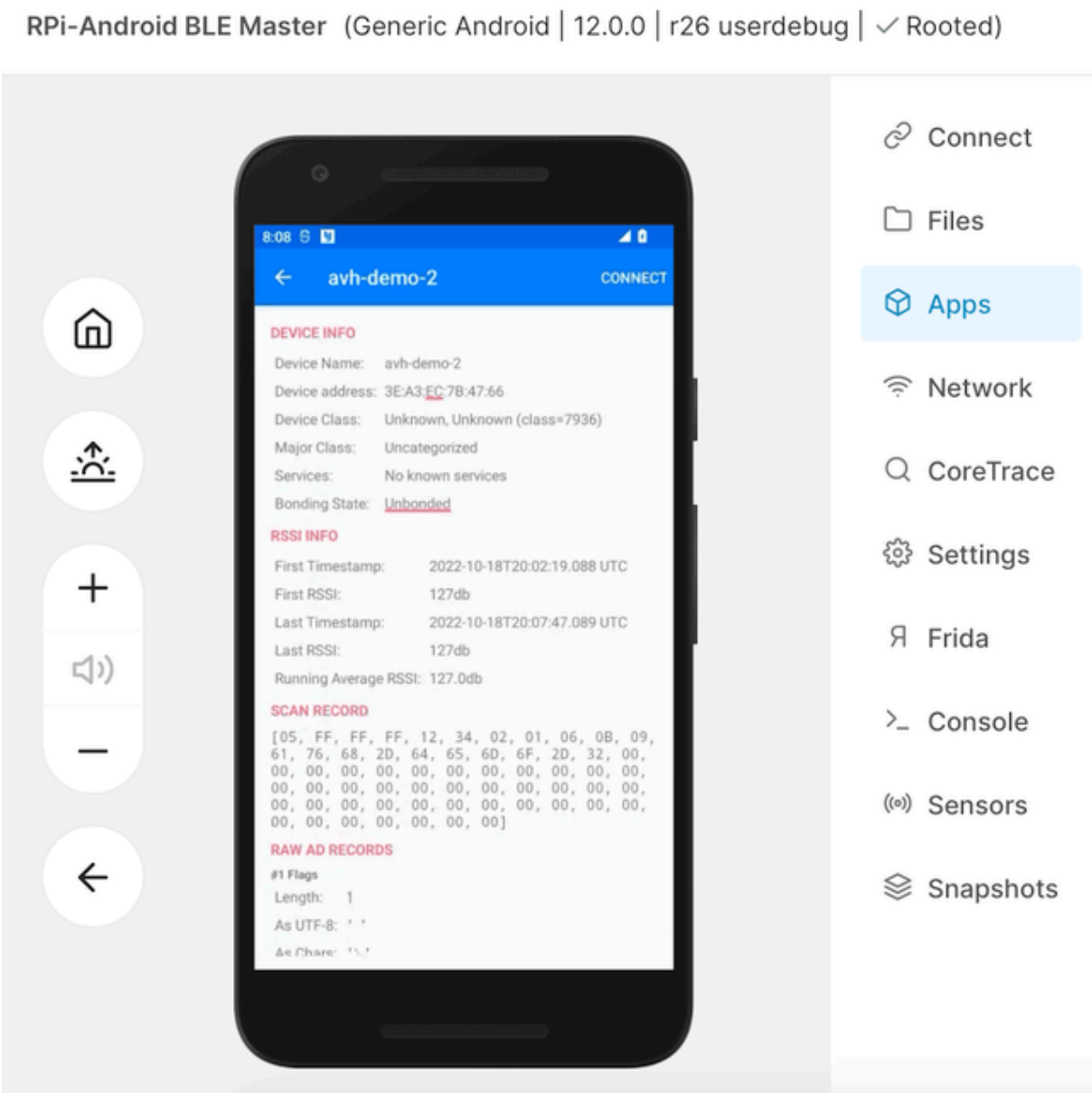
Figure 3-54: Scan for node



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.

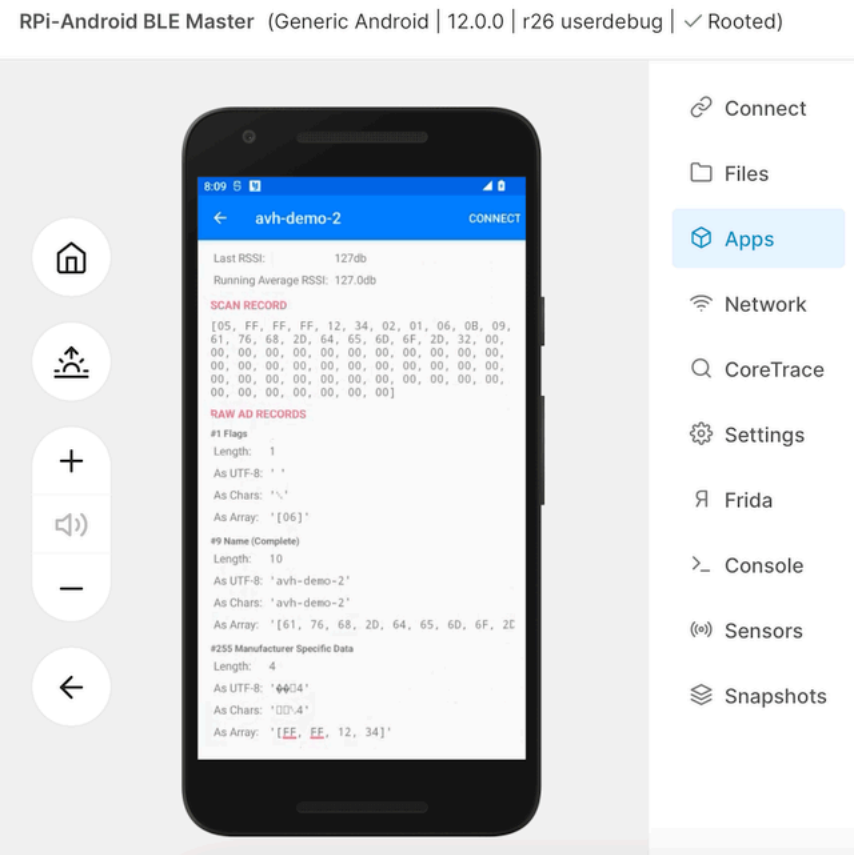
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-55: Additional details view



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

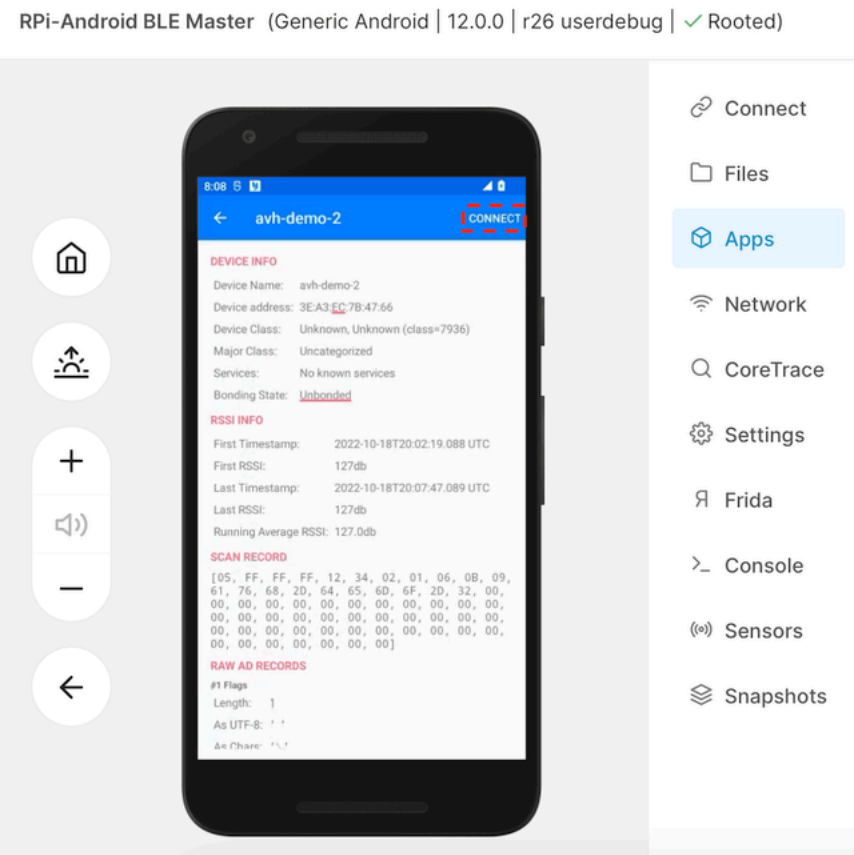
Figure 3-56: Additional details view



5. Click the **CONNECT** button in the Android app.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

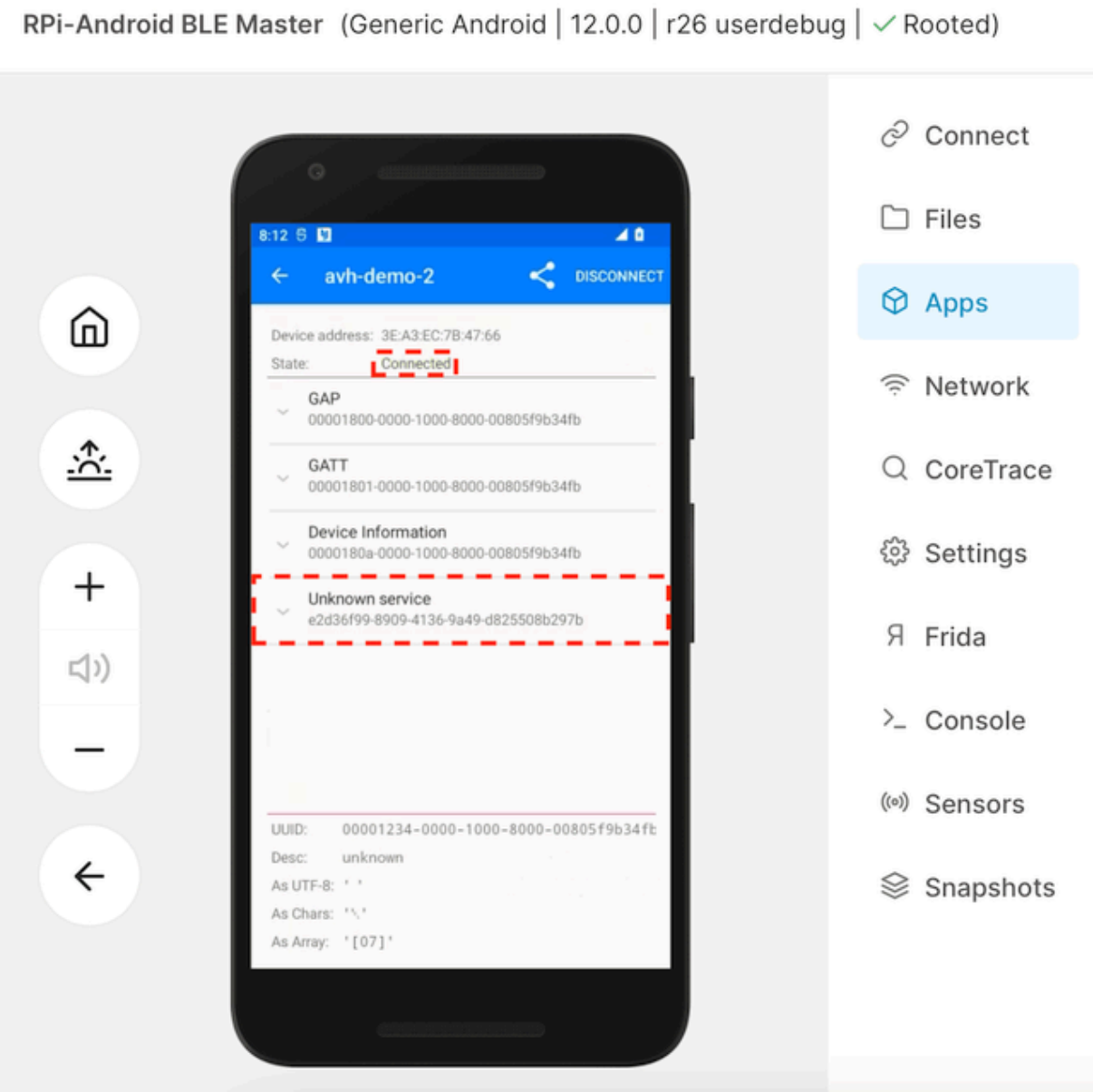
Figure 3-57: Connect to AVH demo



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

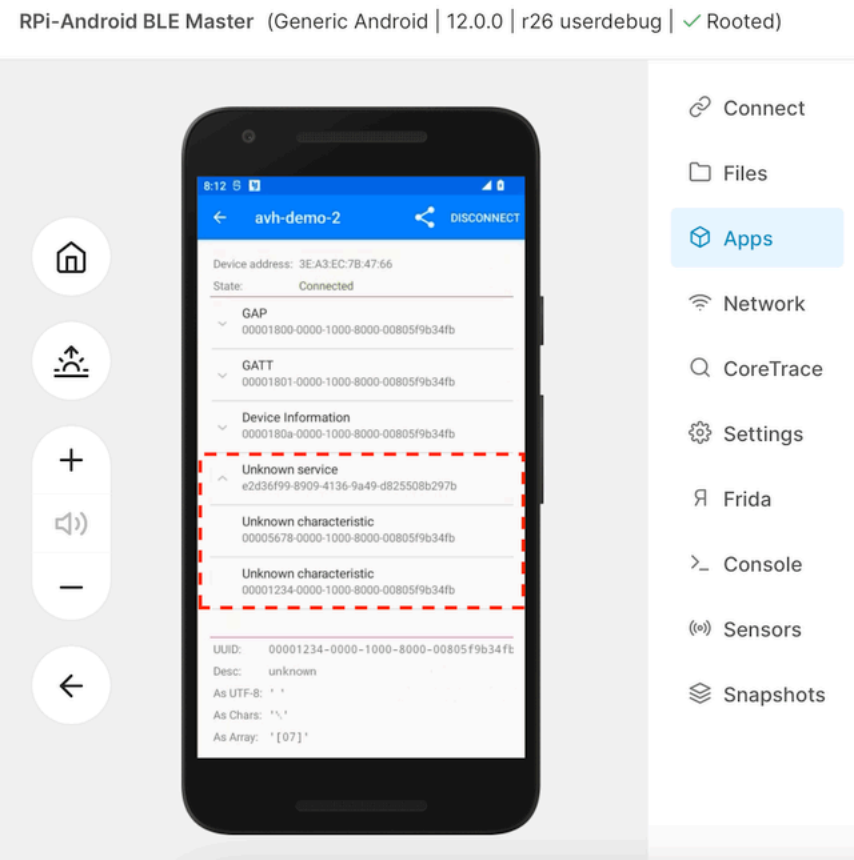
Figure 3-58: Unknown service screen



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

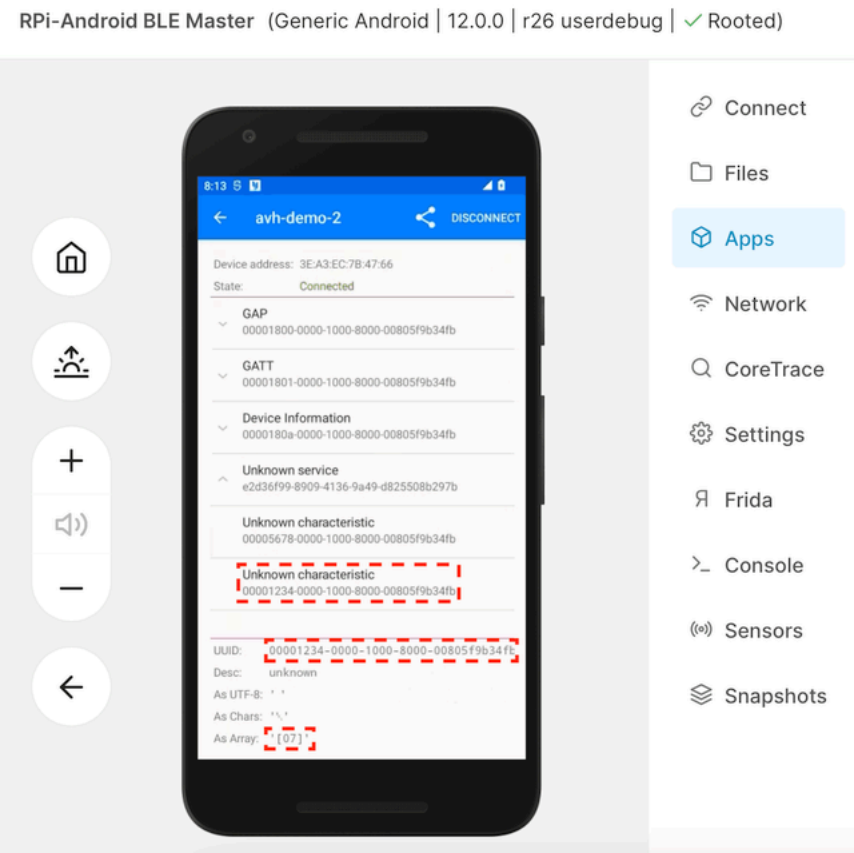
Figure 3-59: Custom characteristics



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

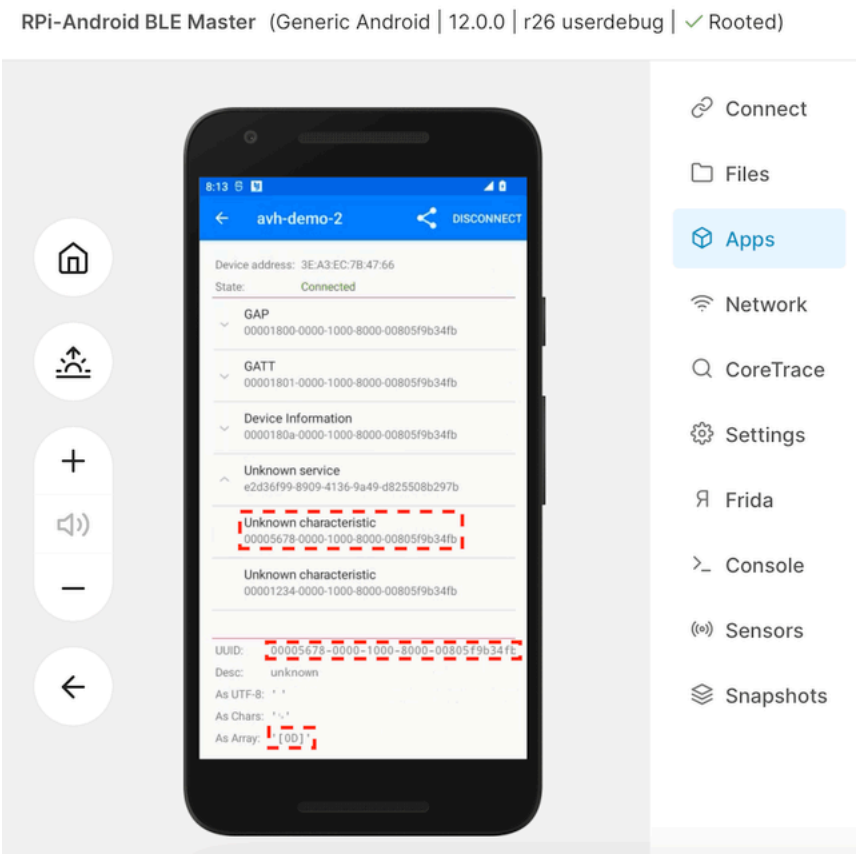
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-60: Custom characteristics values



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-61: Custom characteristics values



3.1.8 Use vMMIO on Raspberry Pi 4

This example exercises virtual Memory Mapped I/O (vMMIO) using the Raspberry Pi 4 model by demonstrating an external (over the network) handler and a local (Raspberry Pi userspace) driver.

For details read the AVH vMMIO Introduction article.

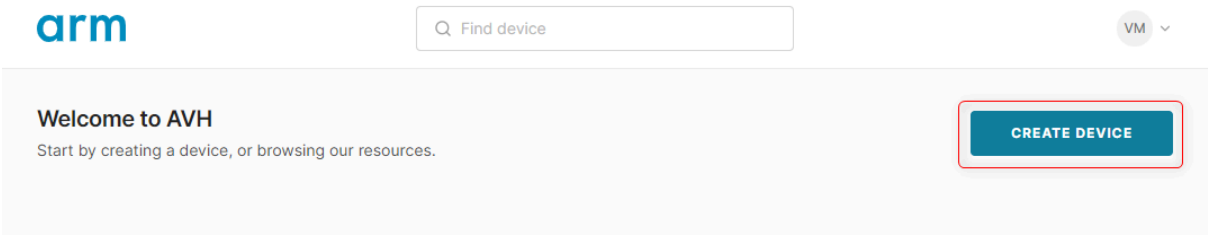
Creating the VM

To create the VM:

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

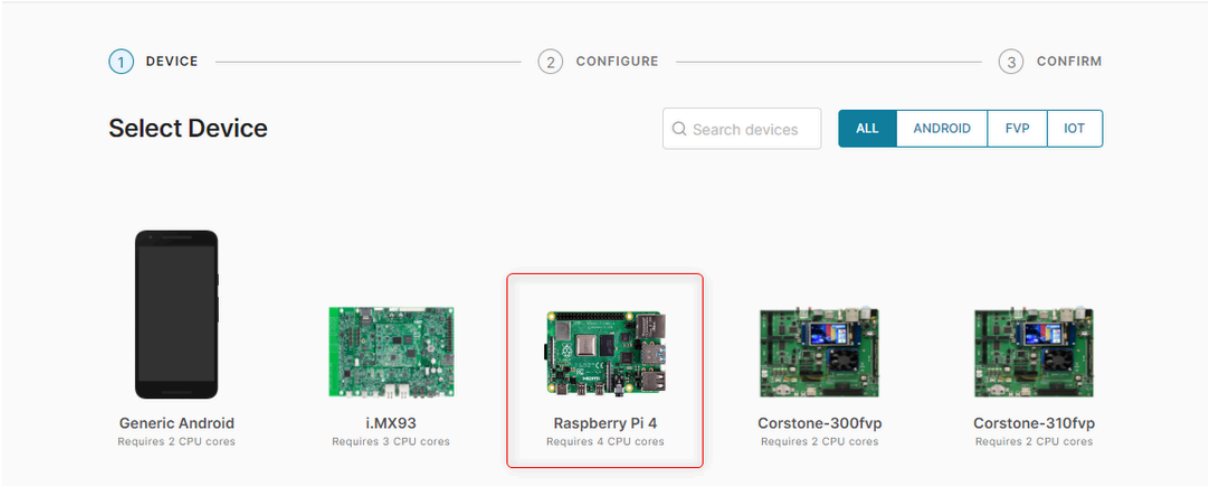
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-62: Create device



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

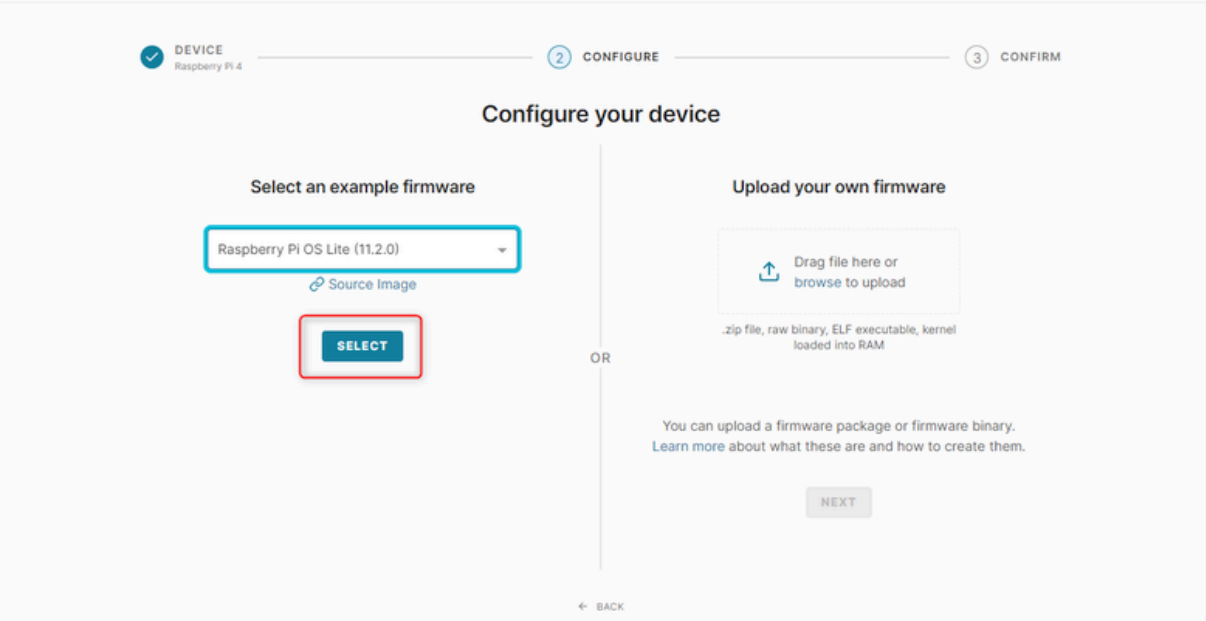
Figure 3-63: Select device



3. Select the `Raspberry Pi OS Lite` firmware if not already selected. Then click **SELECT**.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

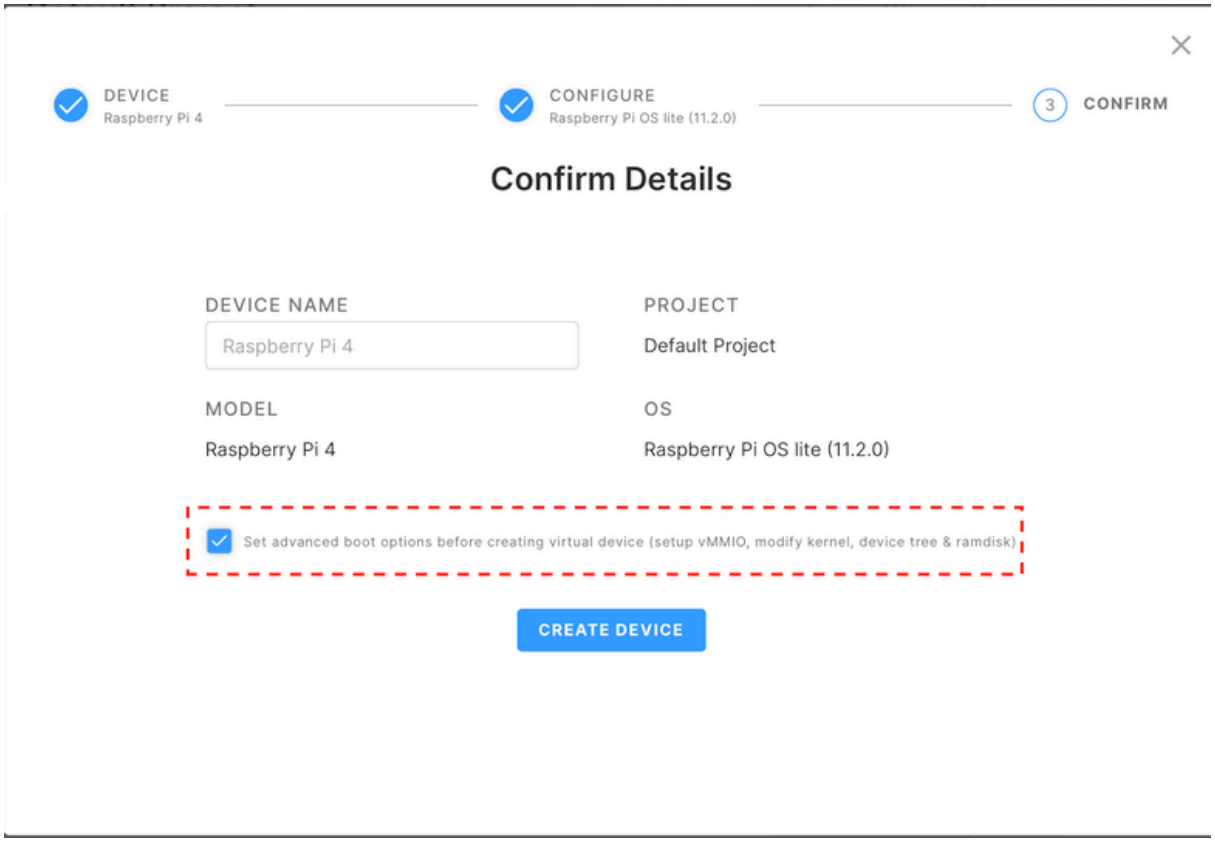
Figure 3-64: Configure your device



4. Select the “Set advanced boot options ...” checkbox and click **CREATE DEVICE**.

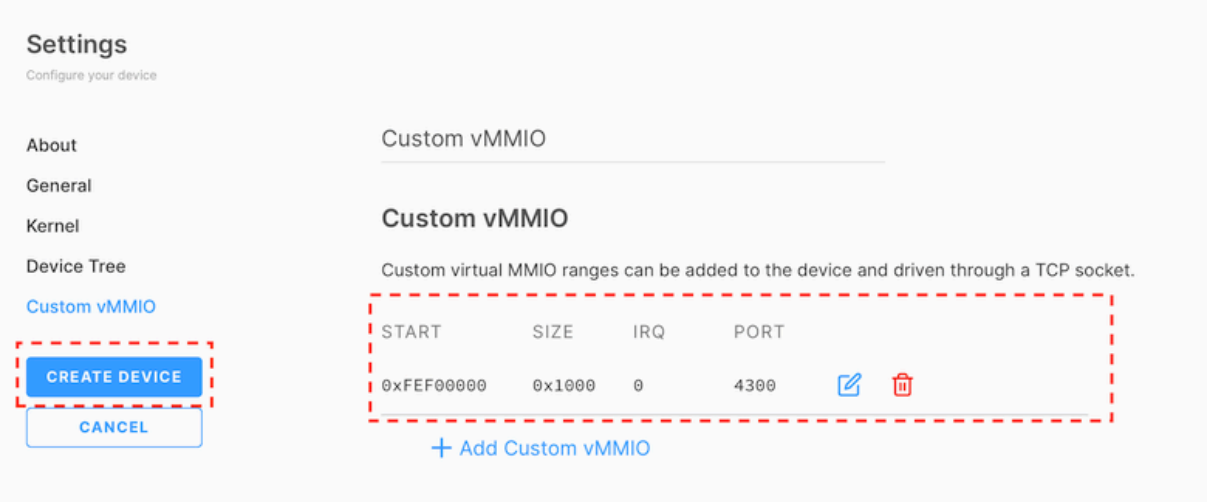
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-65: 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.

Figure 3-66: Custom vMMIO



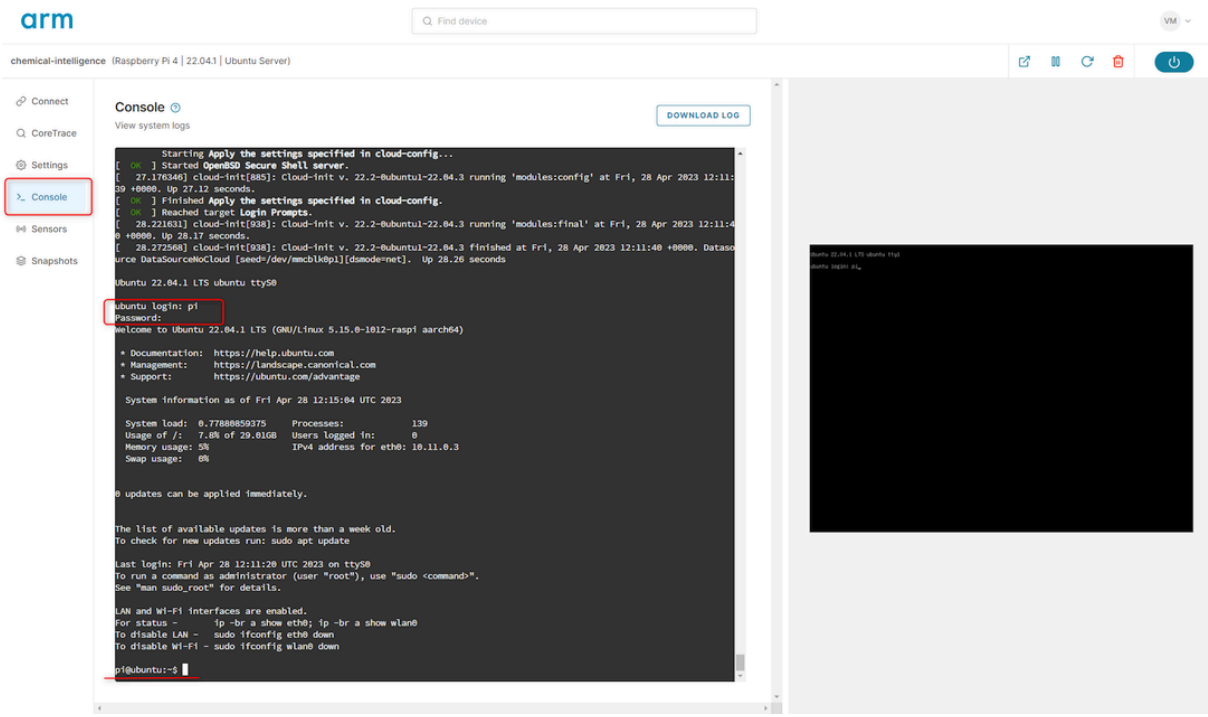
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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 open **Console** tab and log in using the default credentials of `pi/raspberry`.

Figure 3-67: Log in



Writing to MMIO Without a vMMIO Handler Registered

To write to MMIO without a vMMIO handler registered:

1. Download project sources to your Raspberry Pi, extract, and build it with the following commands:

```
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
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2. 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
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

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

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

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

3. Re-run `vmmio-test-inside` within the Raspberry Pi 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.

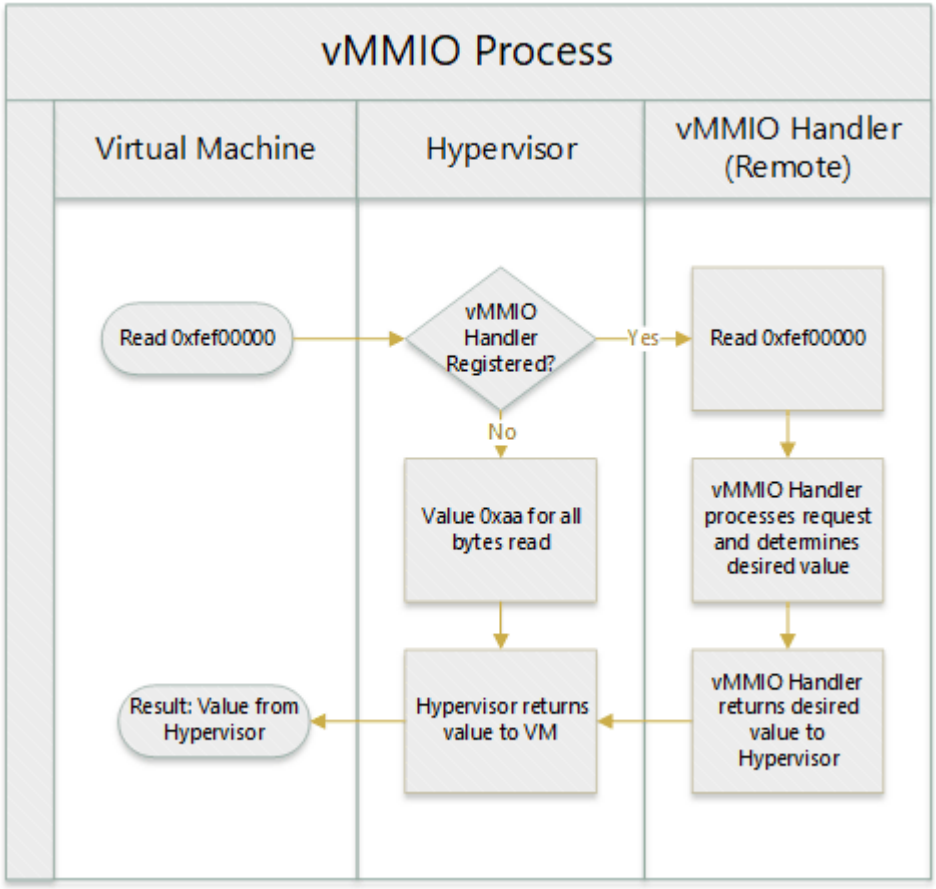
```
$ sudo ./vmmio-test-inside
0: 0x00
1: 0x01
2: 0x02
3: 0x03
...
0: 0x0100
1: 0x0201
2: 0x0302
3: 0x0403
...
0: 0x03020100
1: 0x04030201
2: 0x05040302
3: 0x06050403
...
0: 0x0706050403020100
1: 0x0807060504030201
2: 0x0908070605040302
3: 0x0a09080706050403
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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

Figure 3-68: Flow diagram



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.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.1.9 Updating Linux Kernel on Raspberry Pi 4

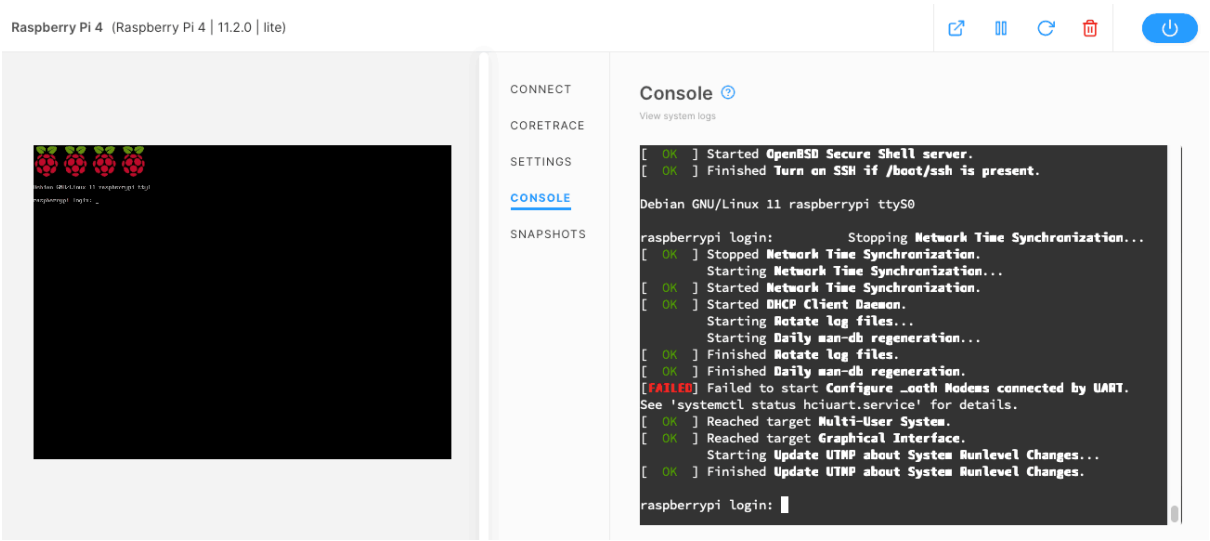
You can update the Linux kernel on your virtual Raspberry Pi 4 board using either the command line or the Desktop firmware's GUI "Welcome to Raspberry Pi". Following sections explain the required steps and potential issues.

- 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

Update Using the Console Command Line

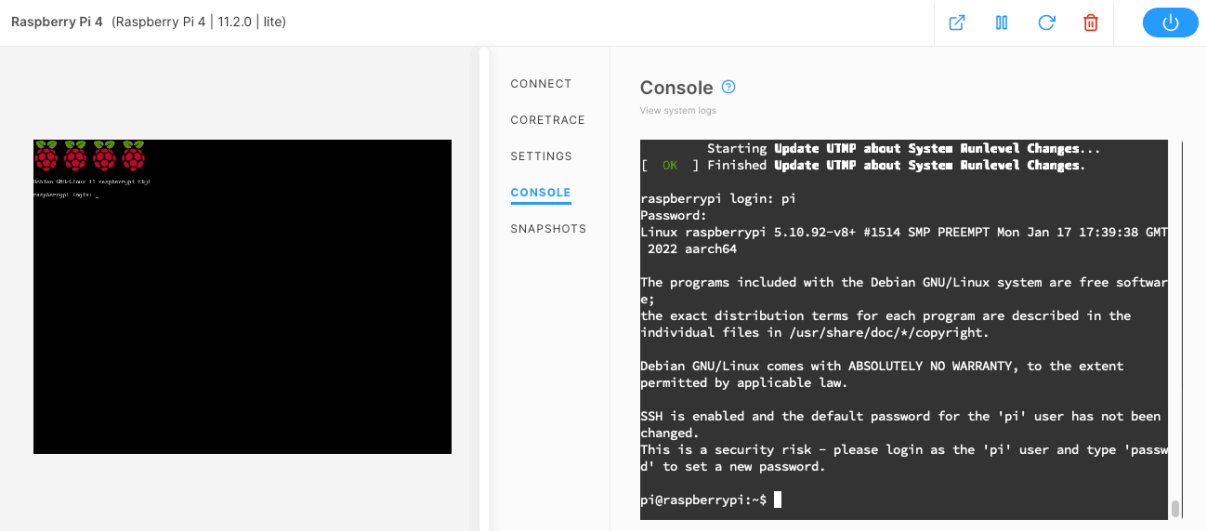
To update the kernel using the console command line:

1. Open the Console tab and press return to show the login prompt.



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2. Log in with the username `pi` and password `raspberry`.



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.

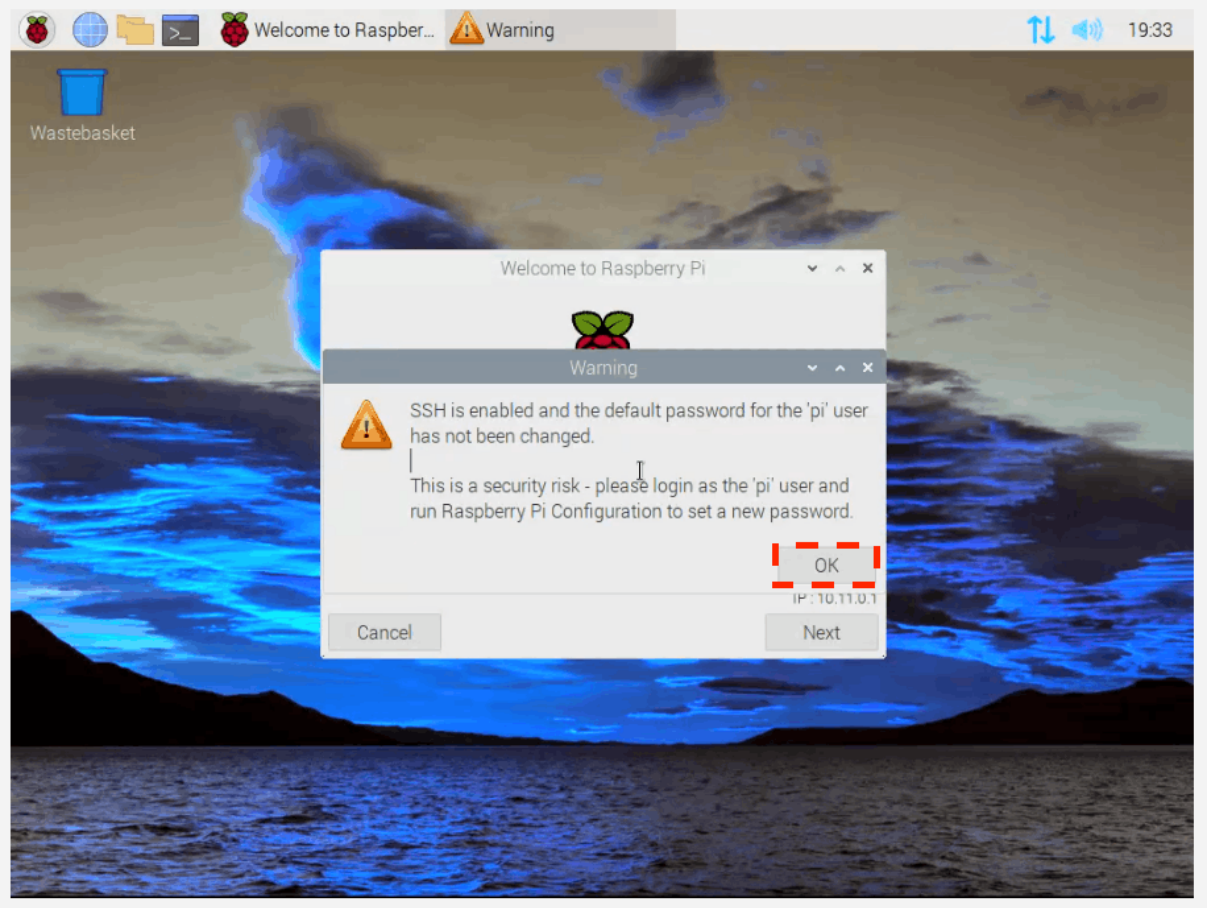
Update Using the Welcome to Raspberry Pi GUI on the Desktop Firmware

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)`.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

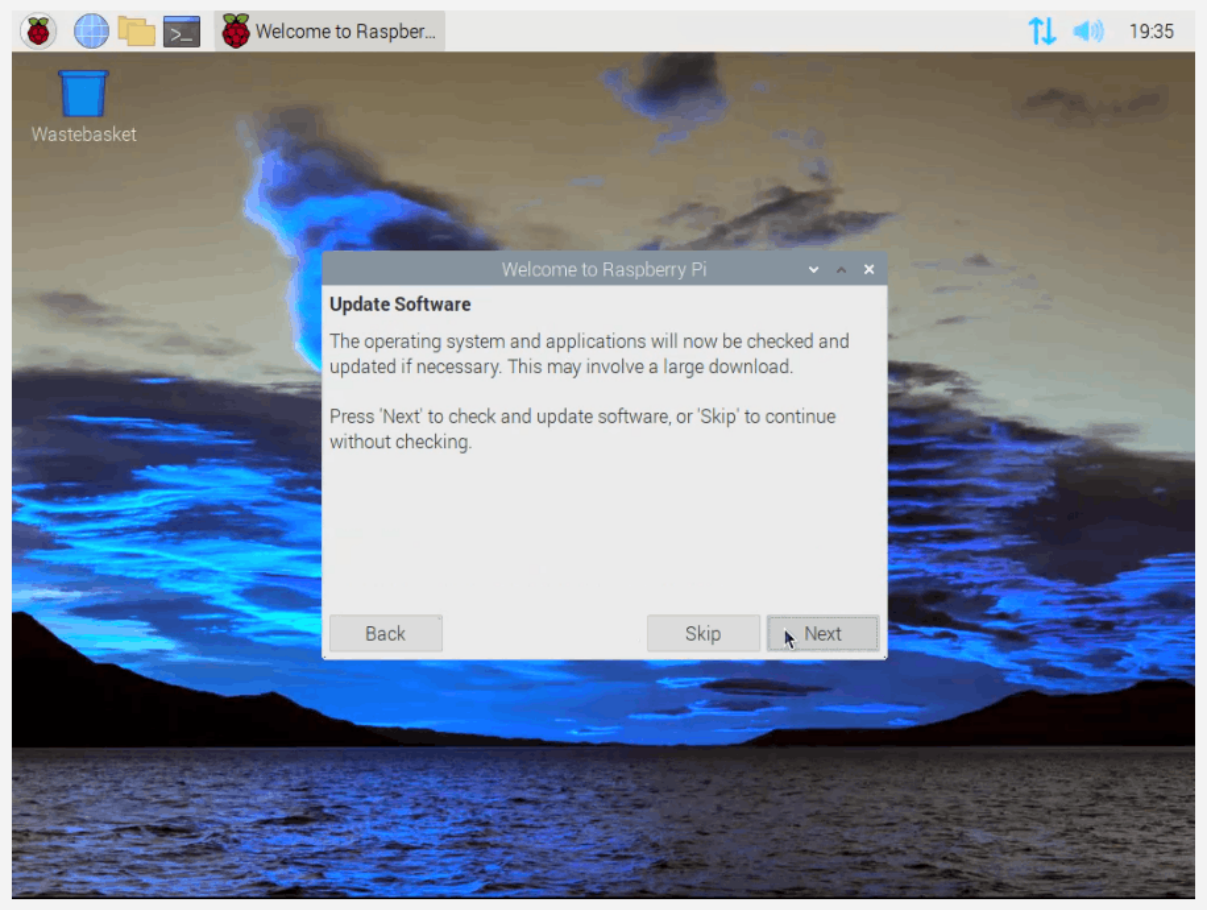
2. Close the warning window about the default SSH password.



3. Click through the Welcome to Raspberry Pi window.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4. The last screen will ask if you want to update. Click **Next**.



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.

Identifying Issues with the Device Kernel

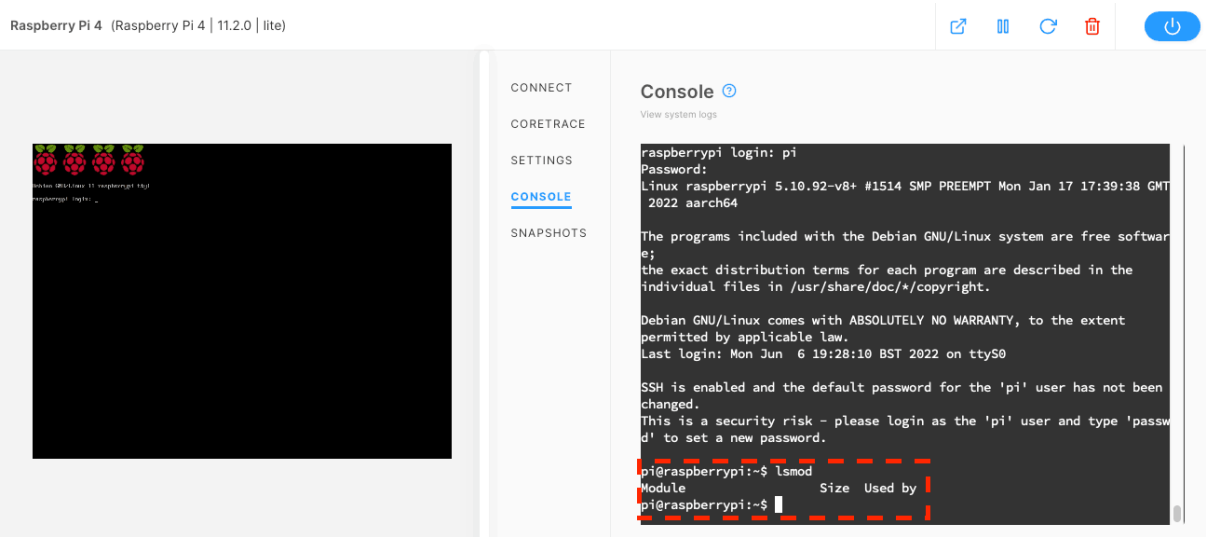
In this section we check if modules are loaded into the kernel and try to install and run Docker.

List the Loaded Kernel Modules

To list the loaded kernel modules:

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1. Run `lsmod` in the console to list the modules that are loaded into the kernel.



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.

Try to Install and Run Docker

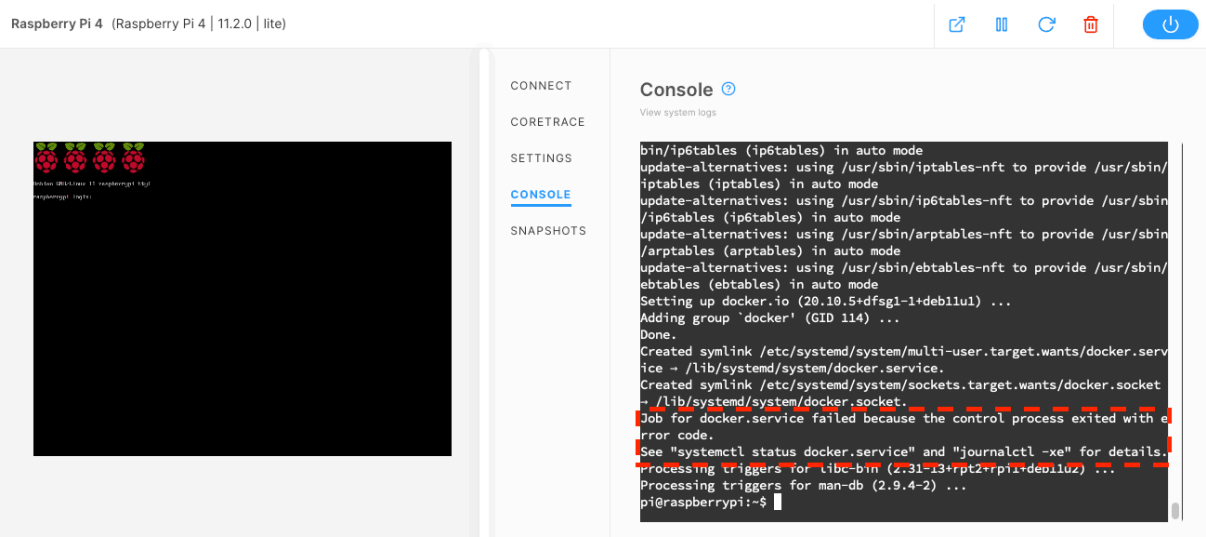
Running docker requires a properly configured kernel. If docker installs but the service fails to run, you may be having an issue with your updated kernel.

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

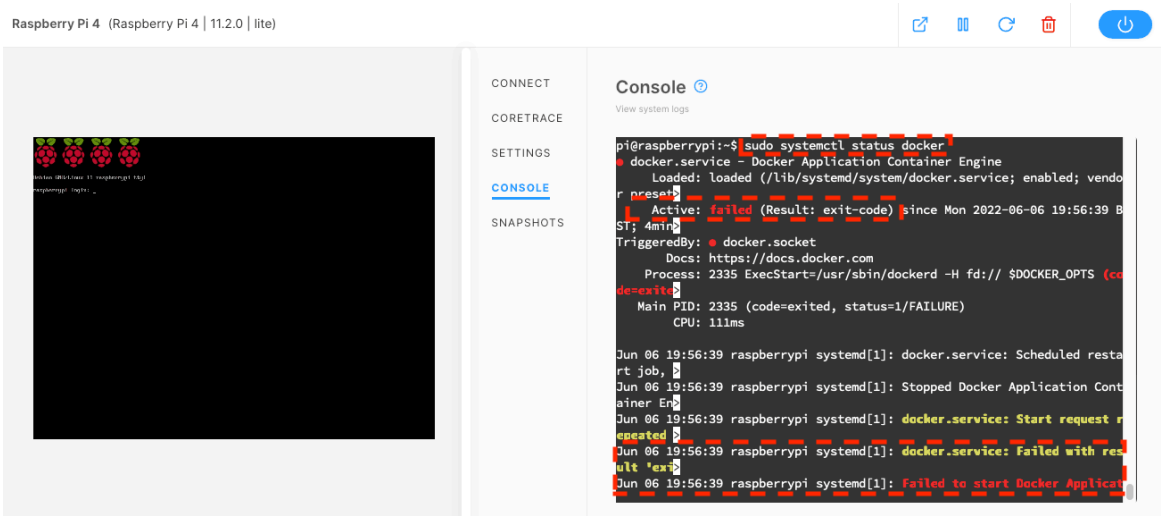
Figure 3-69: Job for docker service failed



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.



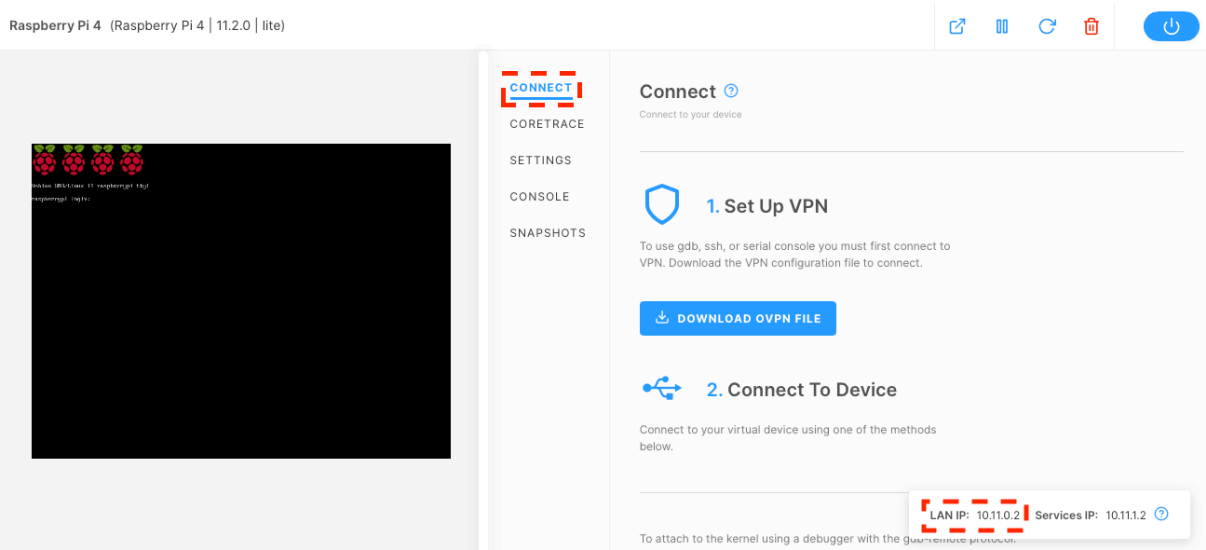
Fixing the Updated Kernel File

To fix the updated kernel file:

1. Connect to your device's VPN using the **Connect** tab.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

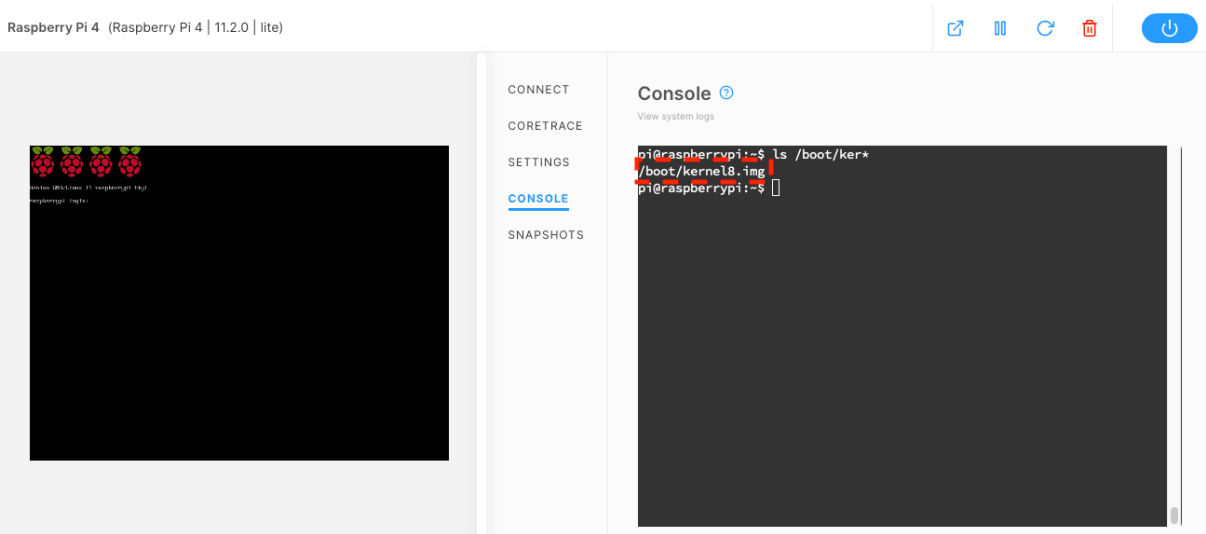
2. Find your device's LAN IP on the bottom-right of the **Connect** tab. In our example, we will use `10.11.0.2`.



3. On your virtual device, locate the path to your kernel file. In our example, we will use `/boot/kernel8.img`.

```
ls /boot/ker*
```

Figure 3-70: 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.

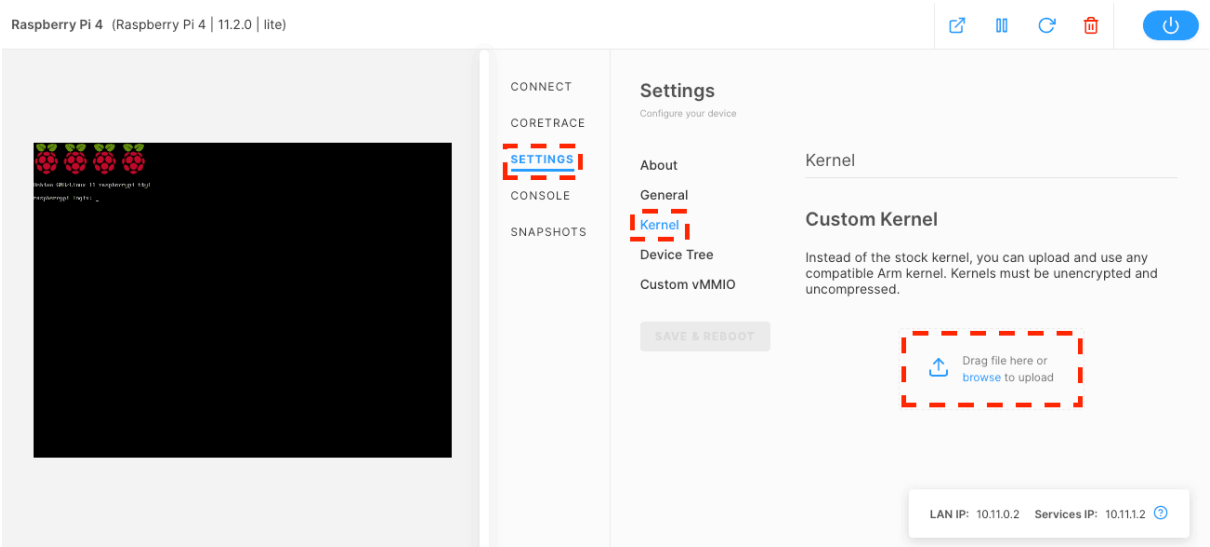
```
scp  pi@10.11.0.2:/boot/kernel8.img ./
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

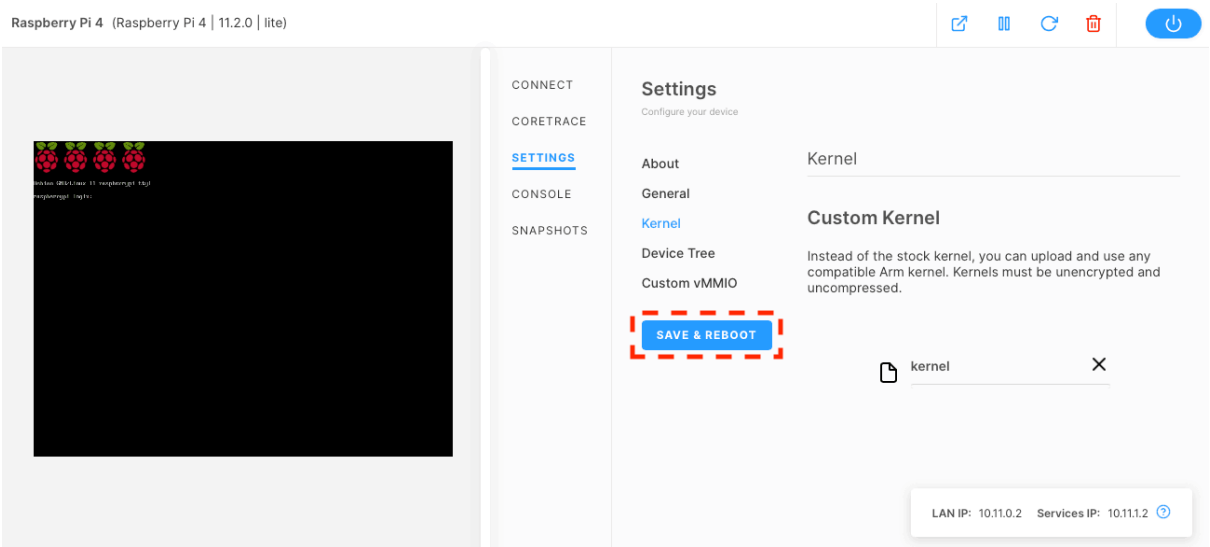
5. Extract the `img` file.

```
gunzip -cd kernel8.img > kernel
```

6. Upload the kernel file in the **Kernel** section of the Settings{.ui} tab.

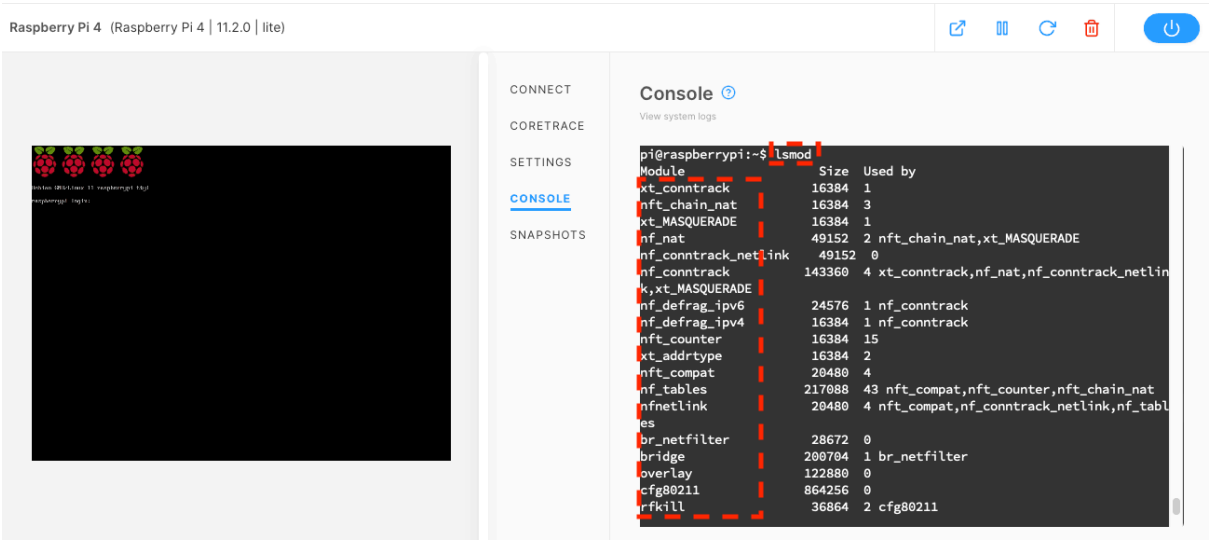


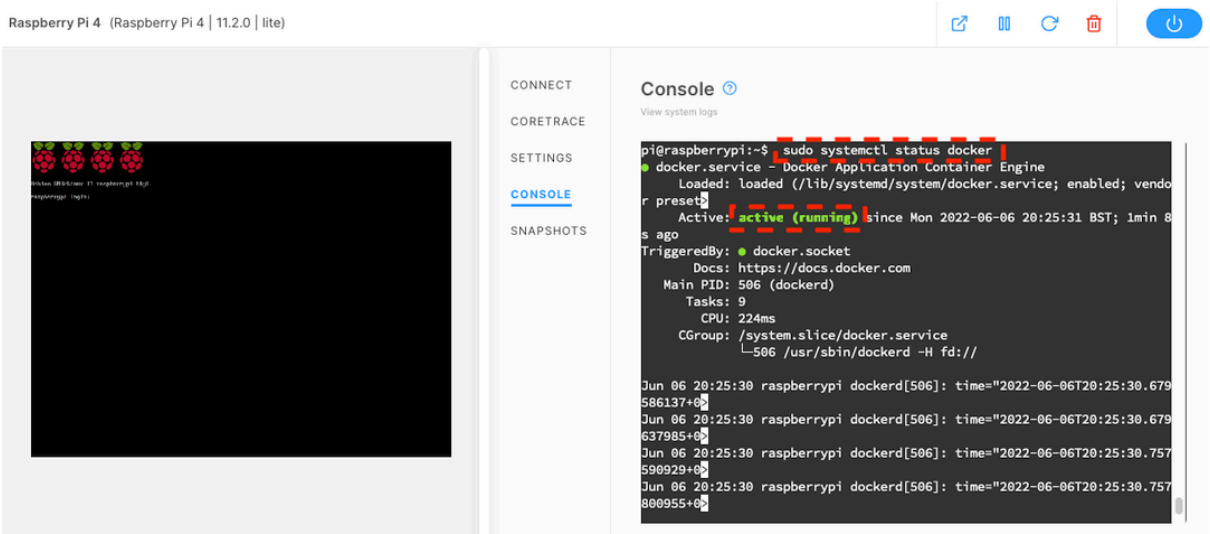
7. Wait for the upload to complete then click SAVE & REBOOT{.ui}.



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8. Confirm the kernel is working properly using `lsmod` or `sudo systemctl status docker`.





9. Clean up your local system.

```
rm kernel8.img kernel
```

10. You're all set!

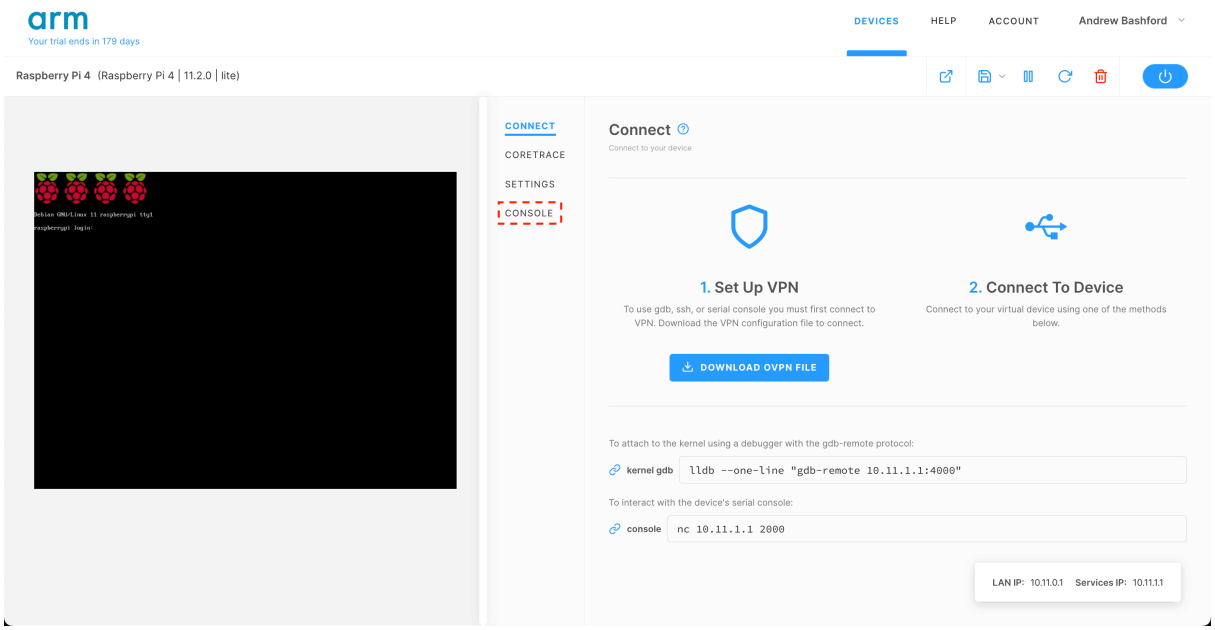
3.1.10 Running Docker on Raspberry Pi 4

This tutorial demonstrates how to run Docker on a virtual Raspberry Pi 4 instance.

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

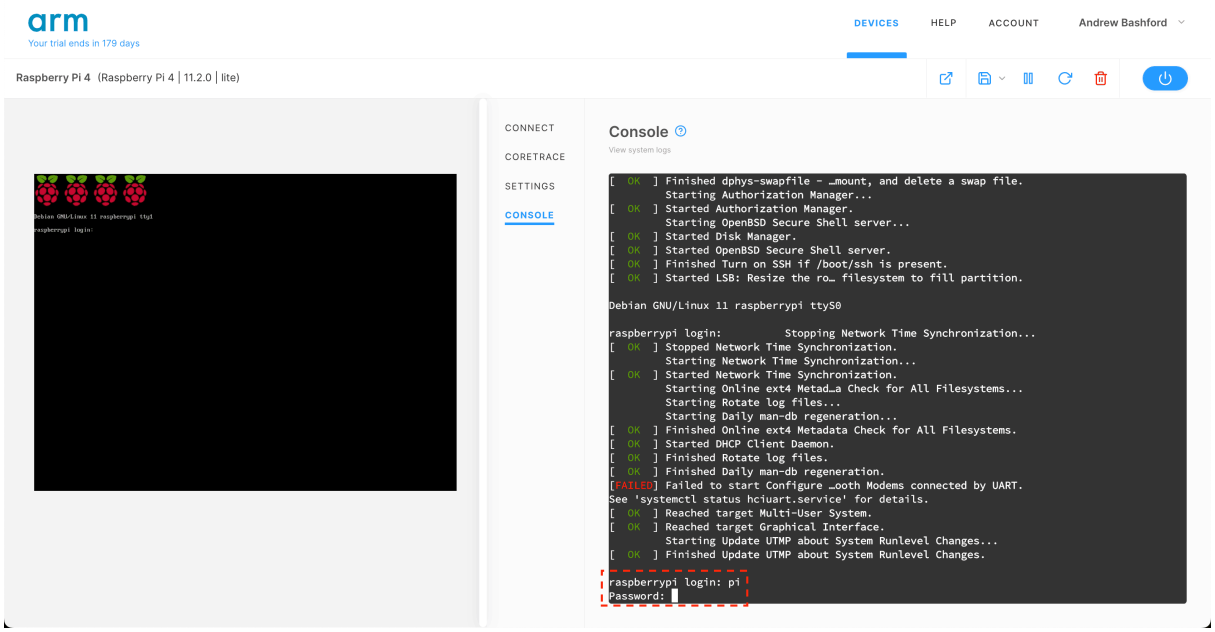
Figure 3-71: Select Console tab



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

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

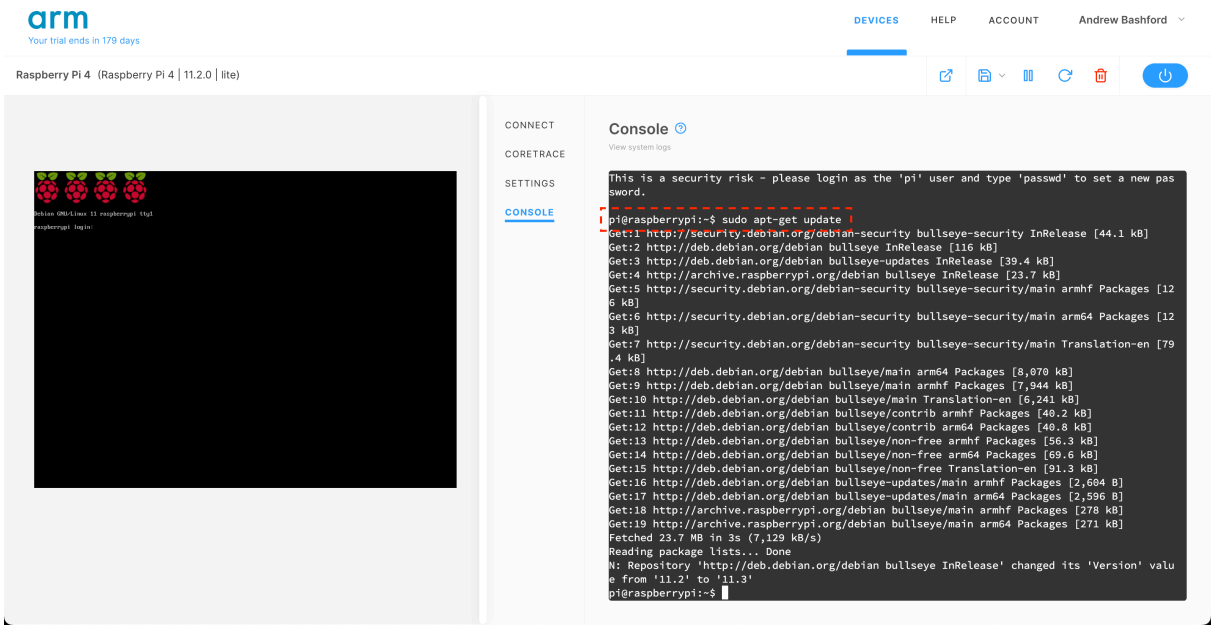
Figure 3-72: Log in



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

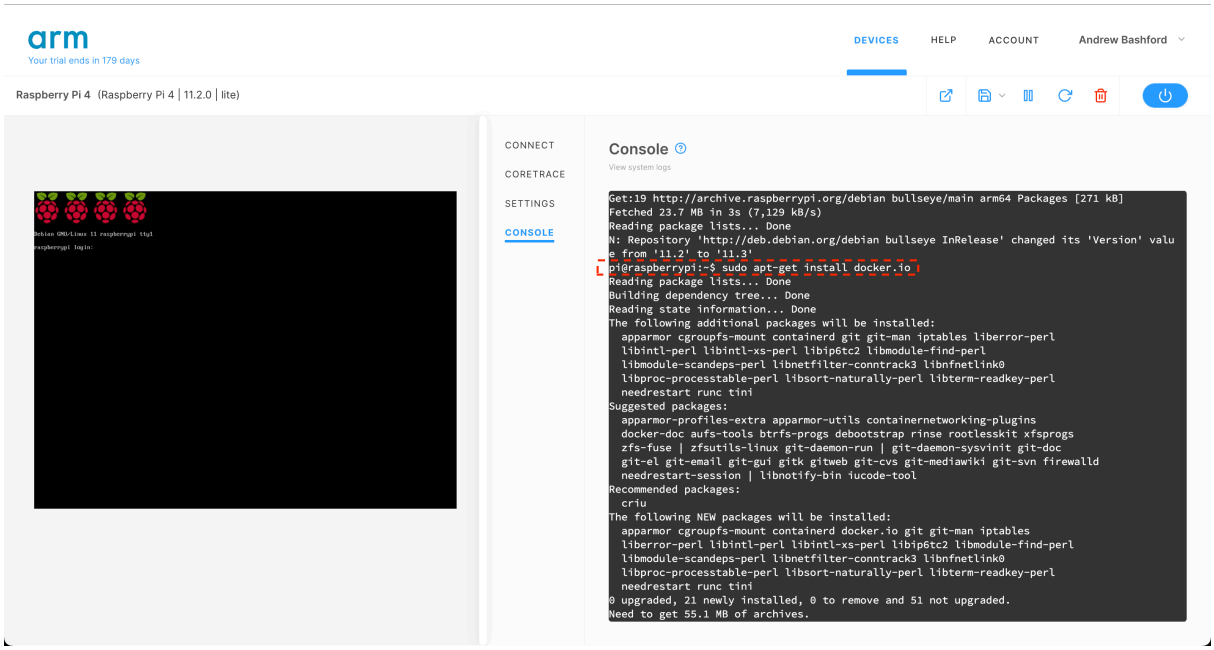
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-73: sudo apt-get update



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

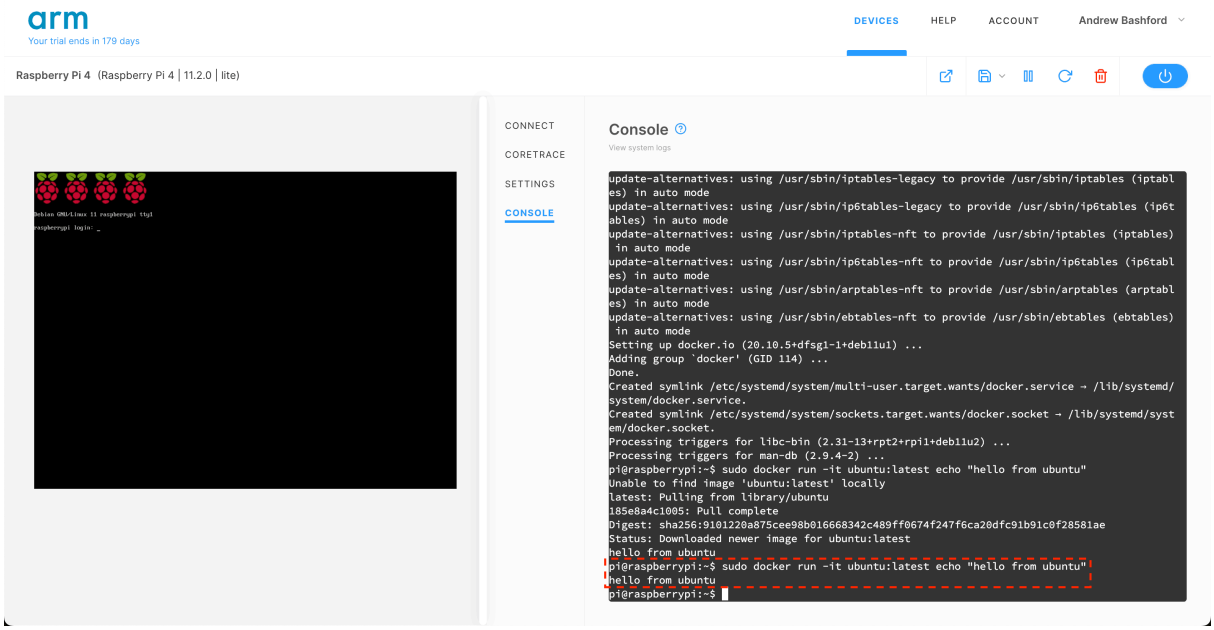
Figure 3-74: sudo apt-get install -y jq docker.io



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-75: Try Docker



3.2 i.MX 8M Arm Cortex Complex

Arm Virtual Hardware (AVH) provides a board model based on NXP Evaluation Kit for the i.MX 8M Plus Applications Processor (8MPLUSLPD4-EVK). In the AVH documentation this virtual platform is referenced as “i.MX 8M Arm Cortex Complex” or in short “i.MX 8M”.

This chapter guides through pages specific to the virtual i.MX 8M Arm Cortex Complex board. It explains how to get started, lists the hardware features supported in the model and provides usage examples relevant for it. The chapter is stuctured as follows:

- **Getting started:**
  - Quickstart for i.MX 8M Arm Cortex Complex provides a simple step-by-step guide on setting up a virtual i.MX 8M board.
- **Board model details:**
  - Supported Hardware Components for i.MX 8M lists the hardware components supported in the virtual i. MX 8M Arm Cortex Complex model.
- **Firmware:**
  - Stock Firmware for i.MX 8M explains the ready-to-use firmware packages available for i.MX 8M Arm Cortex Complex on AVH.
  - Storage Files for i.MX 8M provides information about the storage files in firmware packages on i.MX 8M Arm Cortex Complex.
- **Usage examples:**

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Blink LED on i.MX 8M shows how to control a LED on a virtual i.MX 8M Arm Cortex Complex board.
- Play Video Files on i.MX 8M explains how to play MPEG video from files or web links.

3.2.1 Quickstart for i.MX 8M Arm Cortex Complex

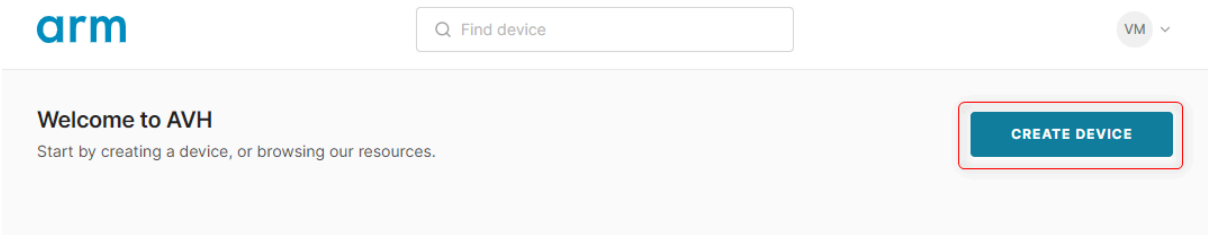
This page guides you through the steps for creating a virtual i.MX 8M Arm Cortex Complex device with a stock firmware `Yocto Linux (2.2.0)`, and shows how to access its Linux command line interface.

Create your device

To create your virtual i.MX 8M Arm Cortex Complex device, do the following:

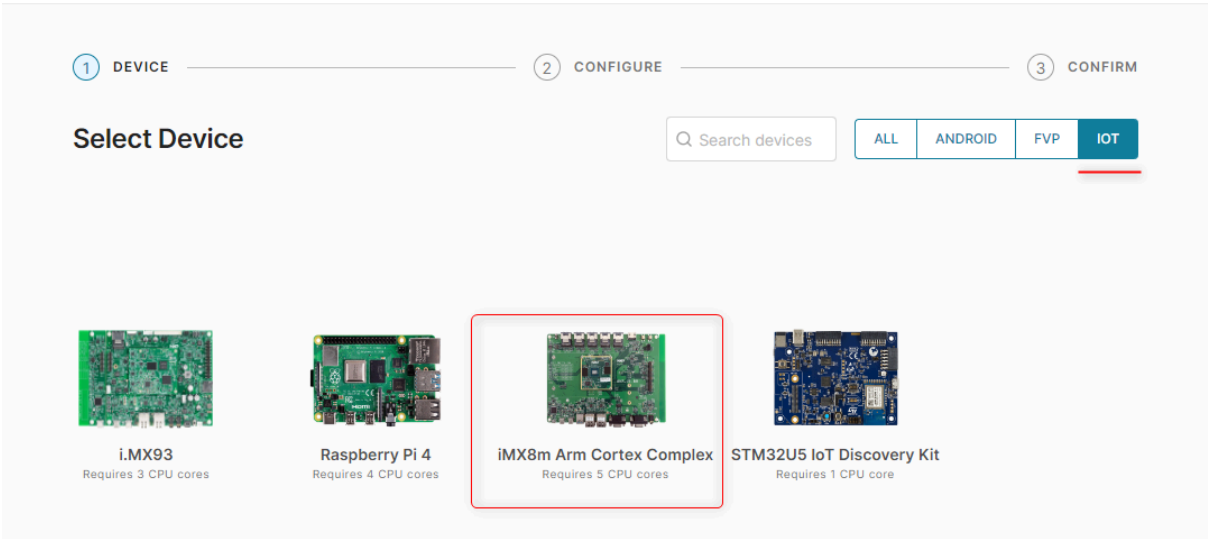
1. Log in to AVH with your Arm account at https://app.avh.arm.com. If you do not have an Arm account you can register here.
2. After login you should land on **Devices** page. There click **CREATE DEVICE**:

Figure 3-76: Start with device creation



3. On the **Select Device** view click on the **i.MX 8M Arm Cortex Complex** board to proceed with creation of corresponding virtual instance.

Figure 3-77: Select i.MX 8M Arm Cortex Complex



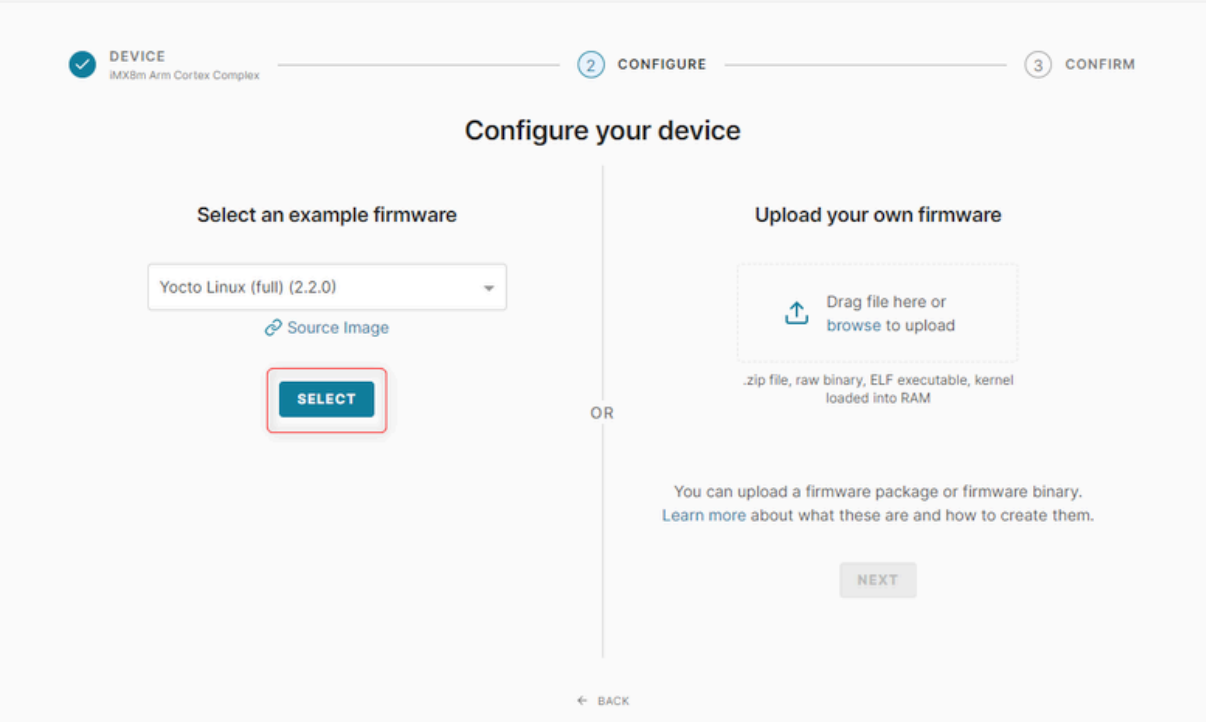
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

You can narrow down the list of displayed boards by selecting the category tab on the top-right side between **| ALL | ANDROID | FVP | IOT**. The i.MX 8M Arm Cortex Complex can be found under IOT and ALL tabs.

4. On the **Configure your device** screen, you will see a dropdown menu **Select an example firmware** with stock firmware options available for the target platform in AVH, see Stock Firmware for i.MX 8M for more details. For this quickstart example we will keep the default selection `Yocto Linux (2.2.0)`. Click **SELECT**.

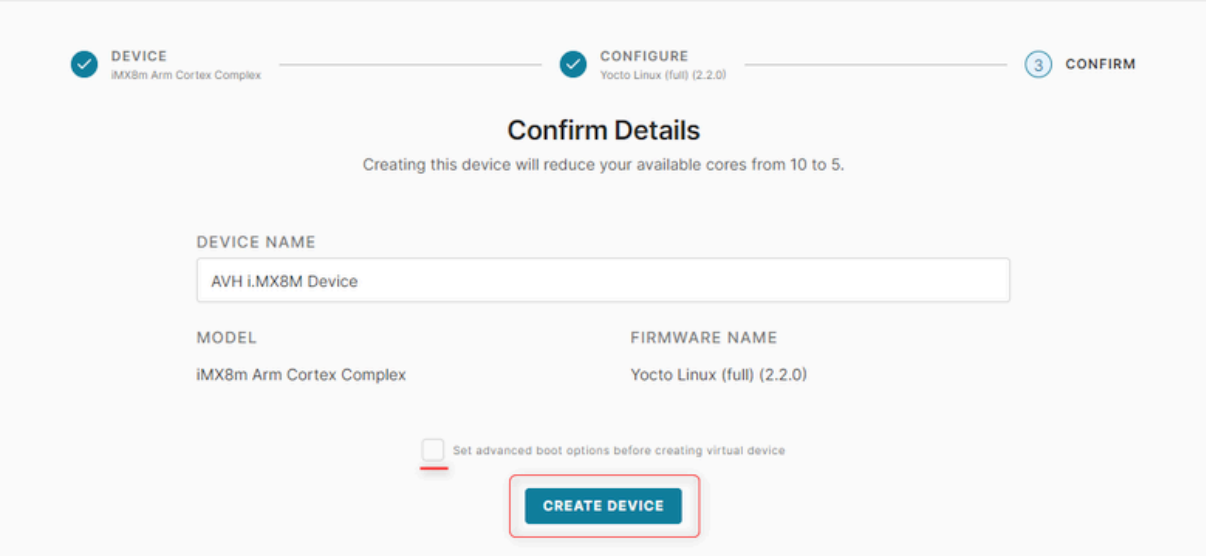
Figure 3-78: Select stock firmware Yocto Linux (2.2.0)



5. On the **Confirm Details** screen you can specify the device name and verify your selections. Keep the advanced boot options unchecked and then click **CREATE DEVICE**.

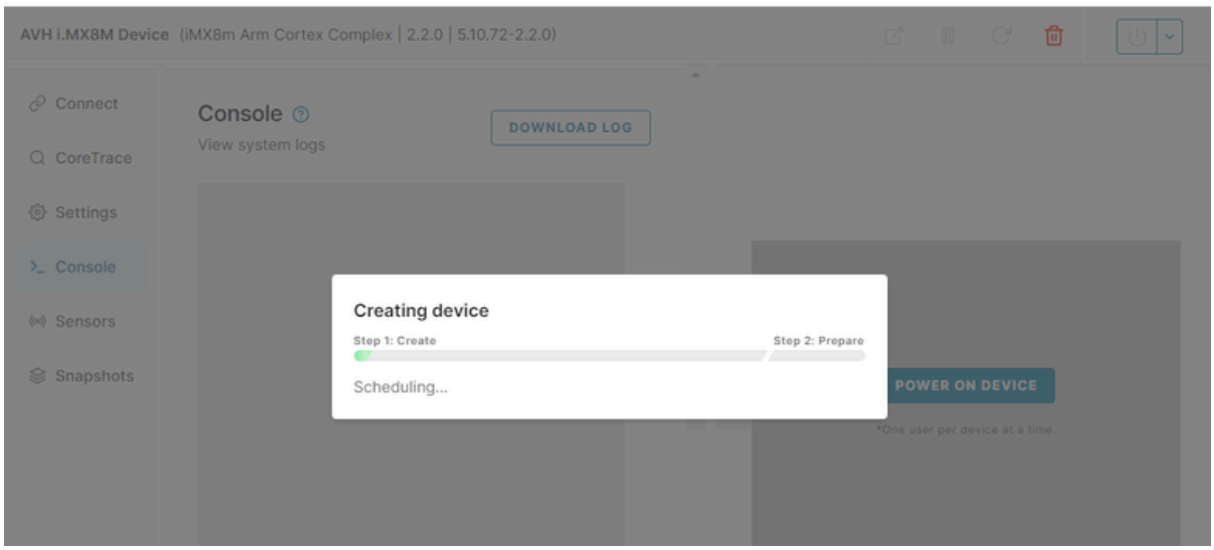
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-79: Confirmation screen



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

Figure 3-80: Creating device



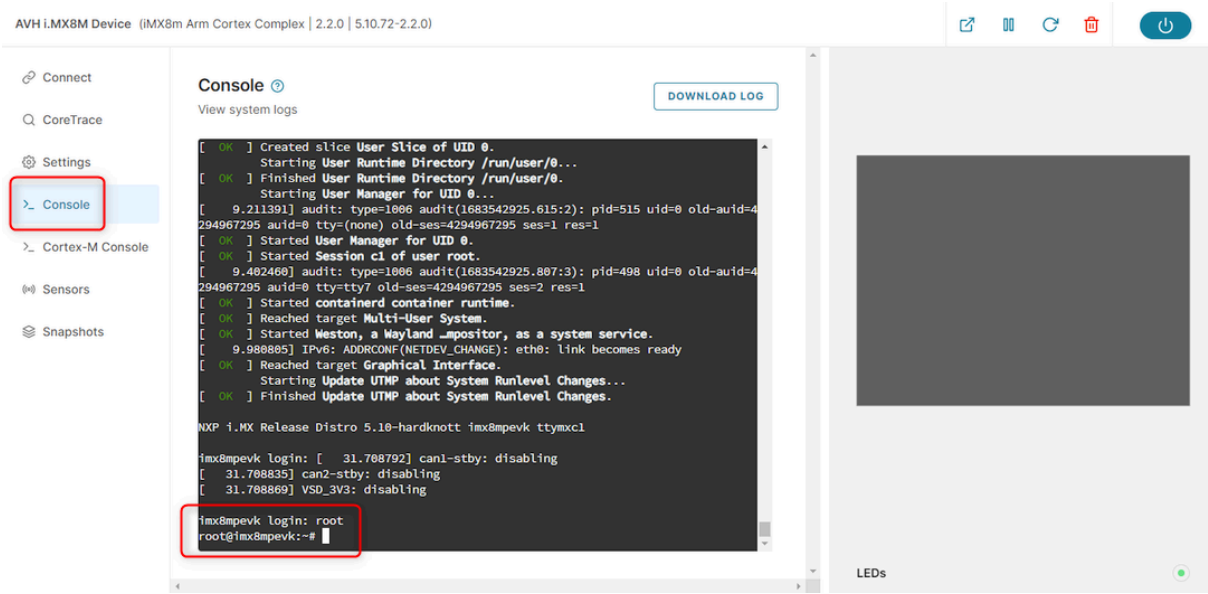
7. Your virtual i.MX 8M Arm Cortex Complex board is now ready to go!

Log in to console

1. Let's log in to the Linux command line interface on the device. Click **Console** tab in device's menu, if not open by default. Type in the default username `root` then hit the Return/Enter key. You should see the display name change to `root@imx8mpevk`, meaning you are logged in.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-81: i.MX 8M Arm Cortex Complex



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

Next steps

For next steps see i.MX 8M Arm Cortex Complex overview page with references to more usage examples.

3.2.2 Supported Hardware Components for i.MX 8M

This page lists the hardware components available on i.MX 8M Arm Complex device.

- High-Level Summary
- Detailed List of Components
  - On-chip resources
  - Off-chip resources

High-Level Summary

The virtual i.MX 8M Arm Cortex Complex board models the following components:

- **Main CPU Platform**
  - 4x Arm Cortex-A53 (TrustZone not implemented)
- **Low Power, Security CPU**
  - Arm Cortex-M7
- **External Memory**
  - 3 SD/eMMC

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- **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

Detailed List of Components

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)

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

  - 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.
  - 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.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

  - 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)
    - 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)

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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
- PLL frequency computation is required for SAI3 operation.

3.2.3 Stock Firmware for i.MX 8M

Arm Virtual Hardware provides a ready-to-use firmware package (stock firmware) to run on virtual i.MX 8M Arm Cortex Complex devices. This is a zip archive prepared according to the AVH Linux firmware package format explained in Linux Firmware Packages. Following stock firmware package is available:

- **Yocto Linux (full) 2.2.0)** firmware package runs i.MX Linux 5.10.72-2.2.0 for i.MX 8M Plus EVK board on its virtualized instance. See Embedded Linux for i.MX Applications Processors for more information and documentation of i.MX Linux.

When creating a new virtual board (see example in Quickstart for i.MX 8M Arm Cortex Complex), on the **Configure your device** screen in the drop-down menu **Select an example firmware** you can choose the target stock firmware that the virtual board instance shall start with.

3.2.4 Storage Files for i.MX 8M

This document outlines the storage files information for virtual i.MX 8M Arm Cortex Complex devices.

- **nand** file for the content of eMMC NAND
  - block_size: 512
  - blocks: 31250000
  - max: 16000000000 bytes
- **nand_flash0** file for the content of raw NAND (chip select 0)
  - block_size: 2048
  - blocks: 67584
  - max: 138412032 bytes
  - Note: includes 4M for ECC
- **nand_flash1** file for the content of raw NAND (chip select 1)
  - block_size: 2048

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

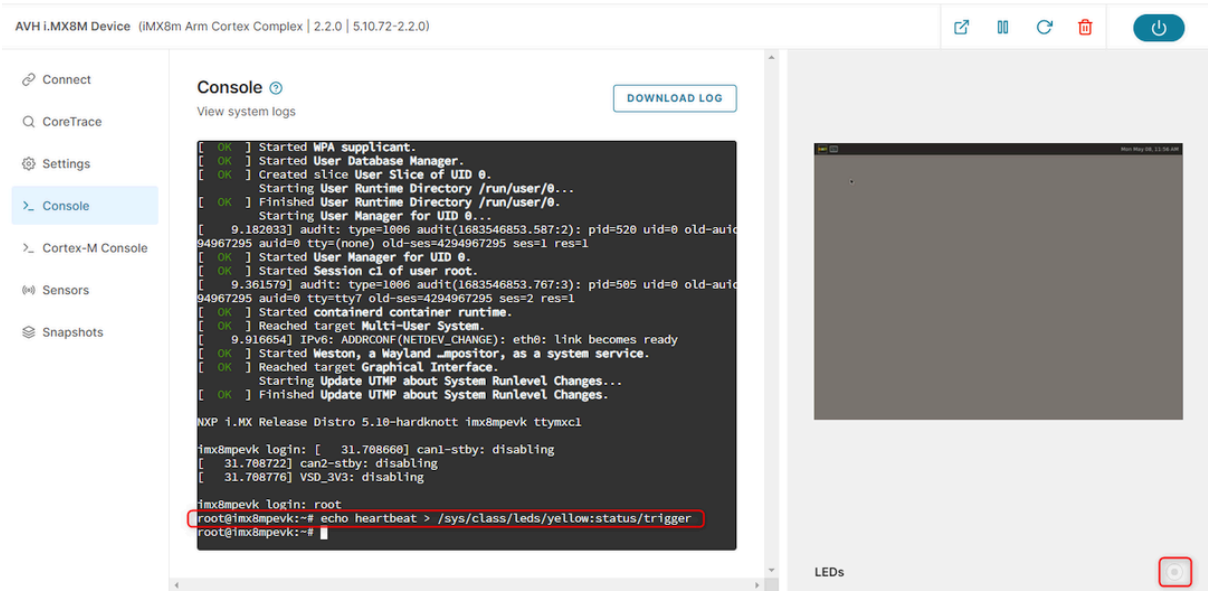
- blocks: 67584
- max: 138412032 bytes
- Note: includes 4M for ECC

3.2.5 Blink LED on i.MX 8M

To blink the LED on your virtual i.MX 8M Arm Cortex Complex device, do the following:

1. Create a new virtual i.MX8M device as explained in the Quickstart for i.MX 8M Arm Cortex Complex tutorial, or go to an already existing device. Ensure that the Linux console is logged in. Note the LED representation at the right-bottom of the screen. In the default state it is on and solid.
2. To blink the LED type `echo heartbeat > /sys/class/leds/yellow:status/trigger` into the console. The LED will now be blinking.

Figure 3-82: LED blinking



3.2.6 Play Video Files on i.MX 8M

You can upload `.mp4` video files to your virtual i.MX 8M Arm Cortex Complex board or provide a URL to play videos in a loop. This page contains the following sections that explain the process:

- Open the GStreamer Multimedia Application
- Play Video Using the Web Interface
  - Use Your Webcam
  - Upload a File - Web Interface

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

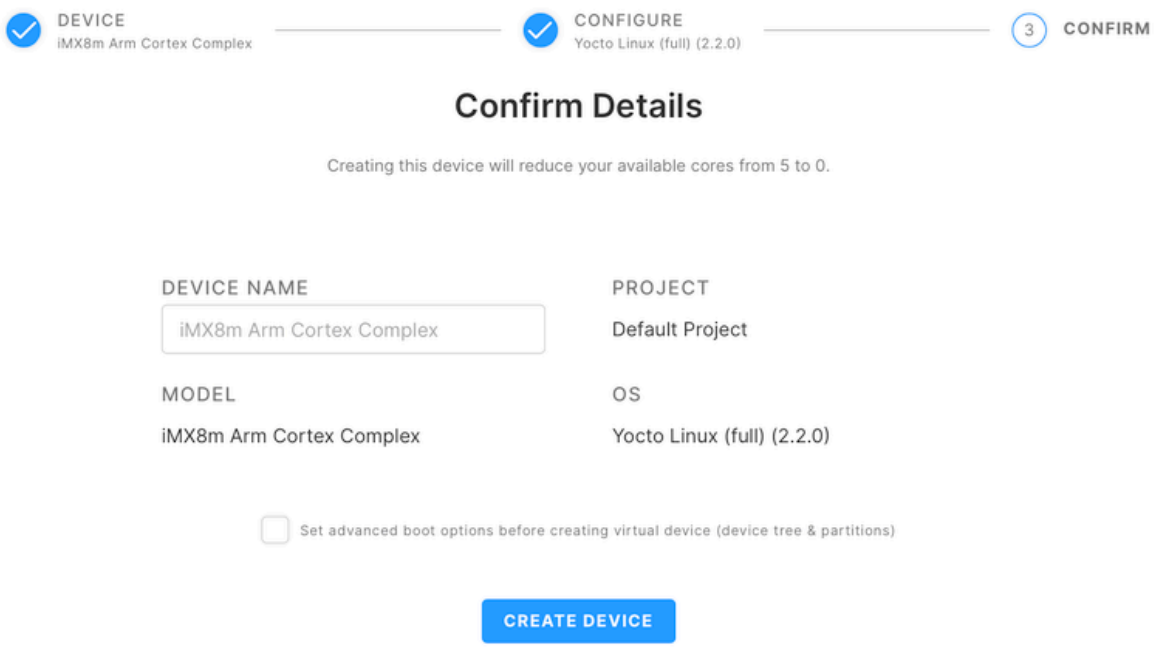
- Link to a URL - Web Interface
- Play Videos Using API and POST Requests
  - Upload a File - API and POST Requests
  - Link to a URL - API and POST Requests

Open the GStreamer Multimedia Application

To open GStreamer:

1. Create a virtual i.MX 8M Arm Cortex Complex board running Yocto Linux OS version 2.2.0.

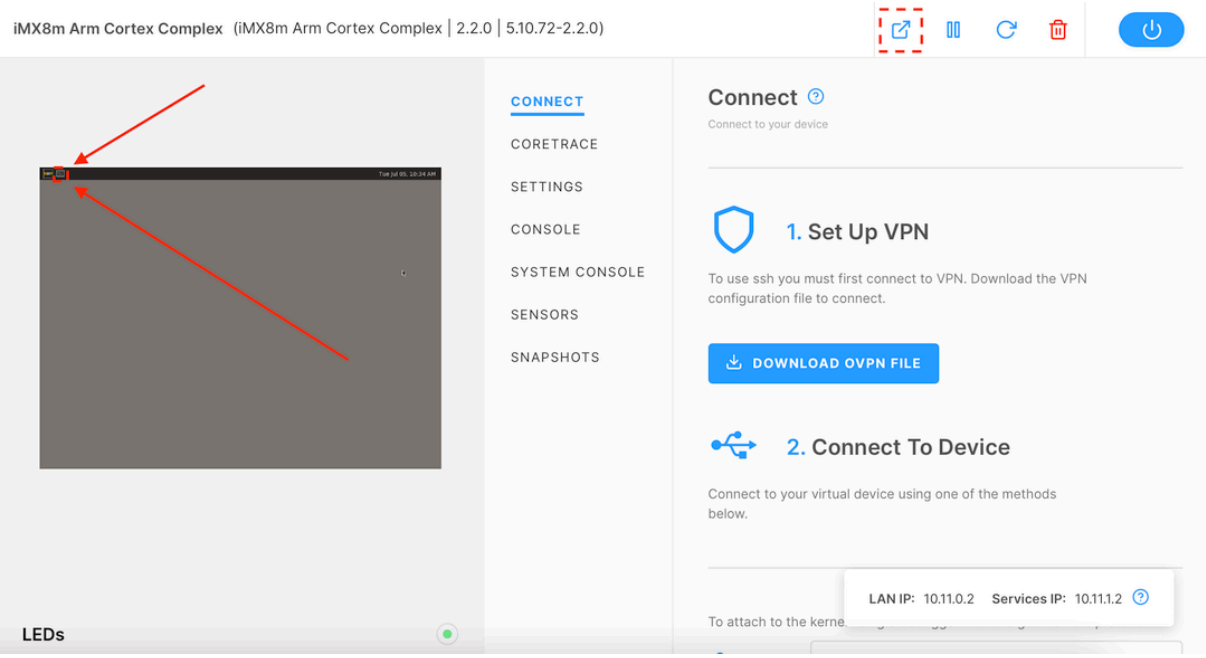
   Figure 3-83: Create iMX8m Board



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.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

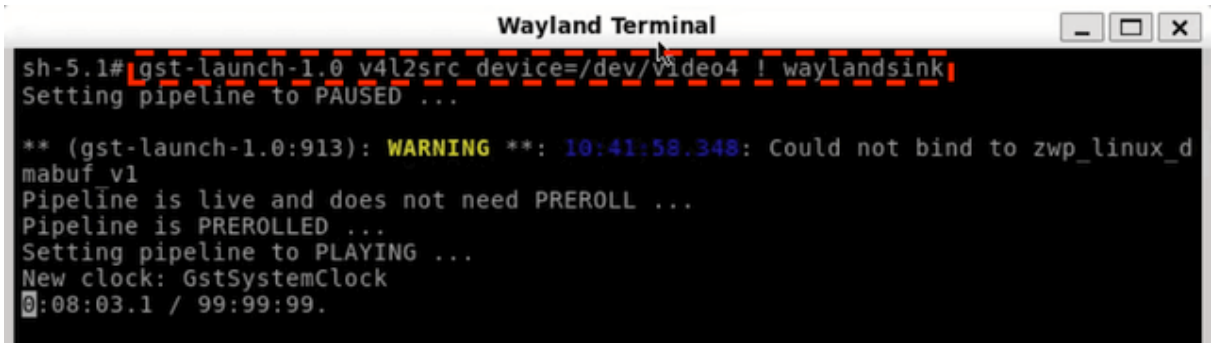
Figure 3-84: Open Terminal



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

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

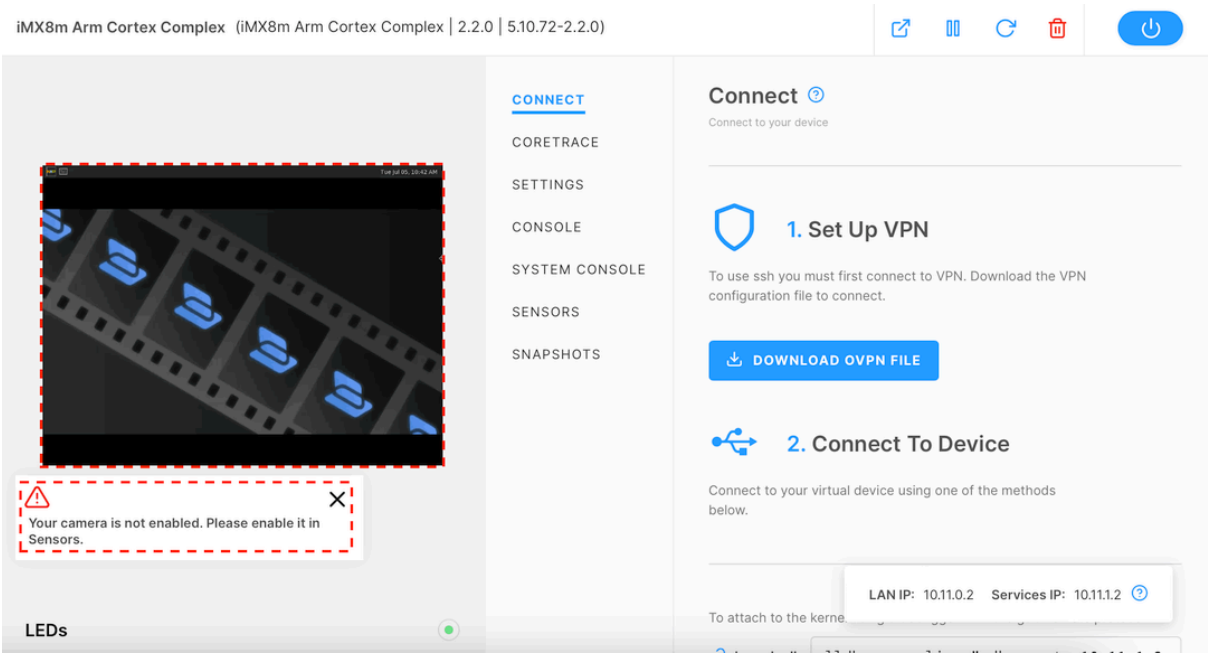
Figure 3-85: Terminal Command



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

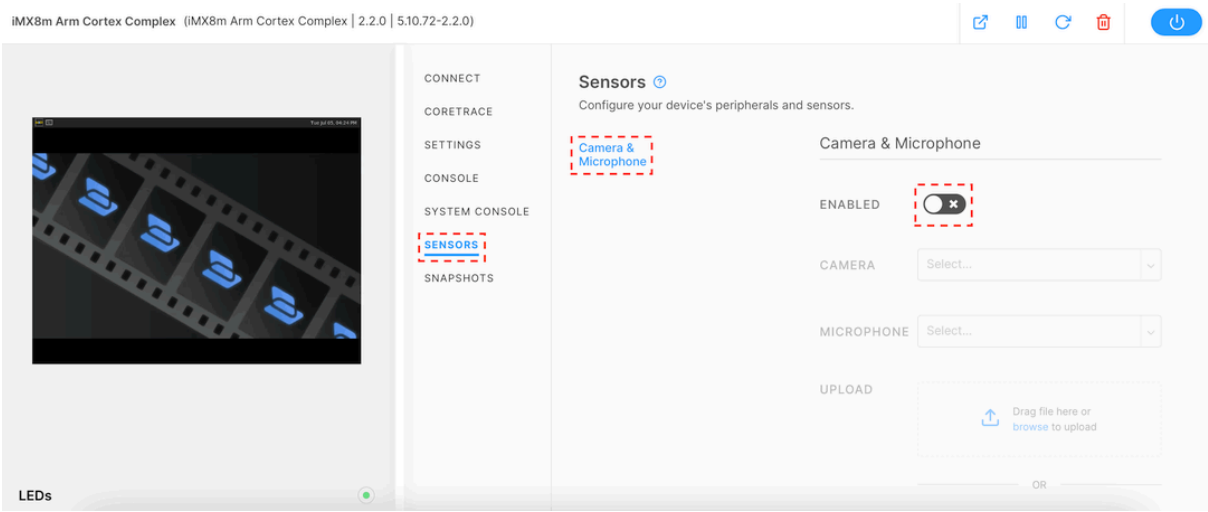
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-86: GStreamer Opened



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

Figure 3-87: Enable the sensors



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.

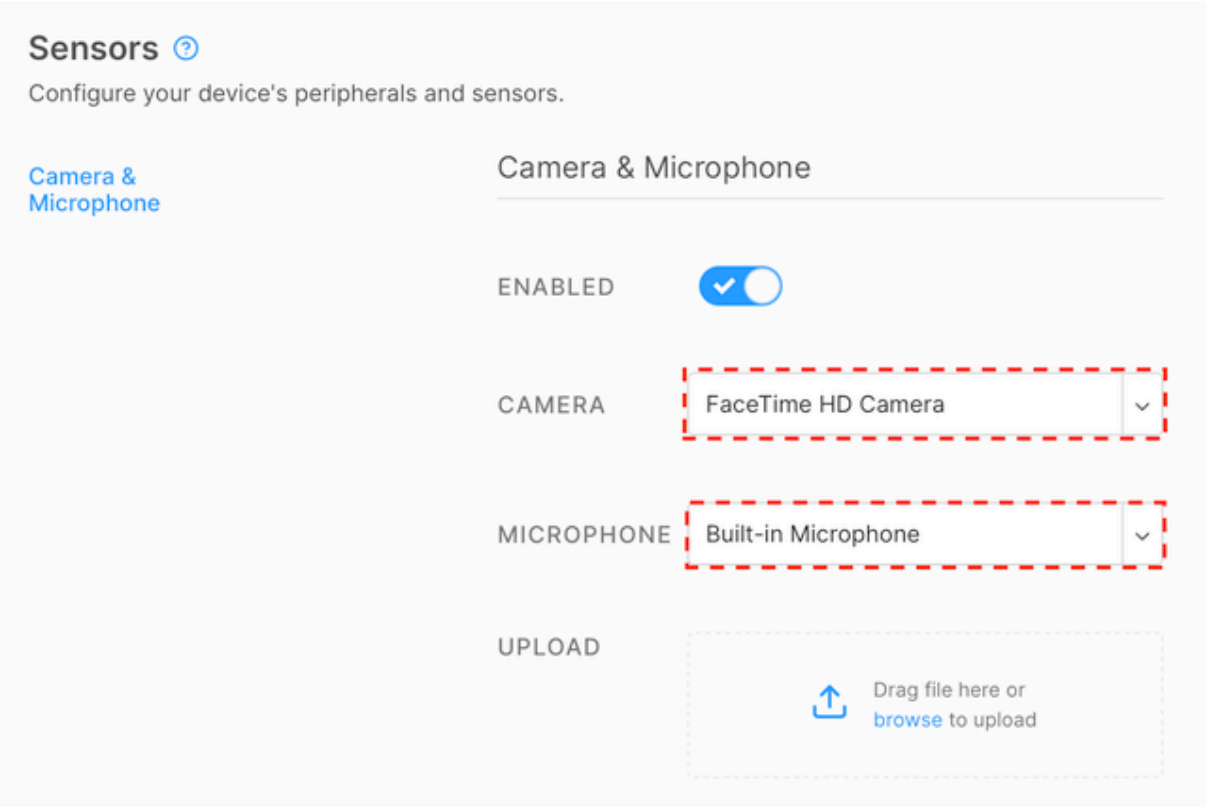
Use Your Webcam

To use your webcam:

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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.

Figure 3-88: Confirm sensors



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

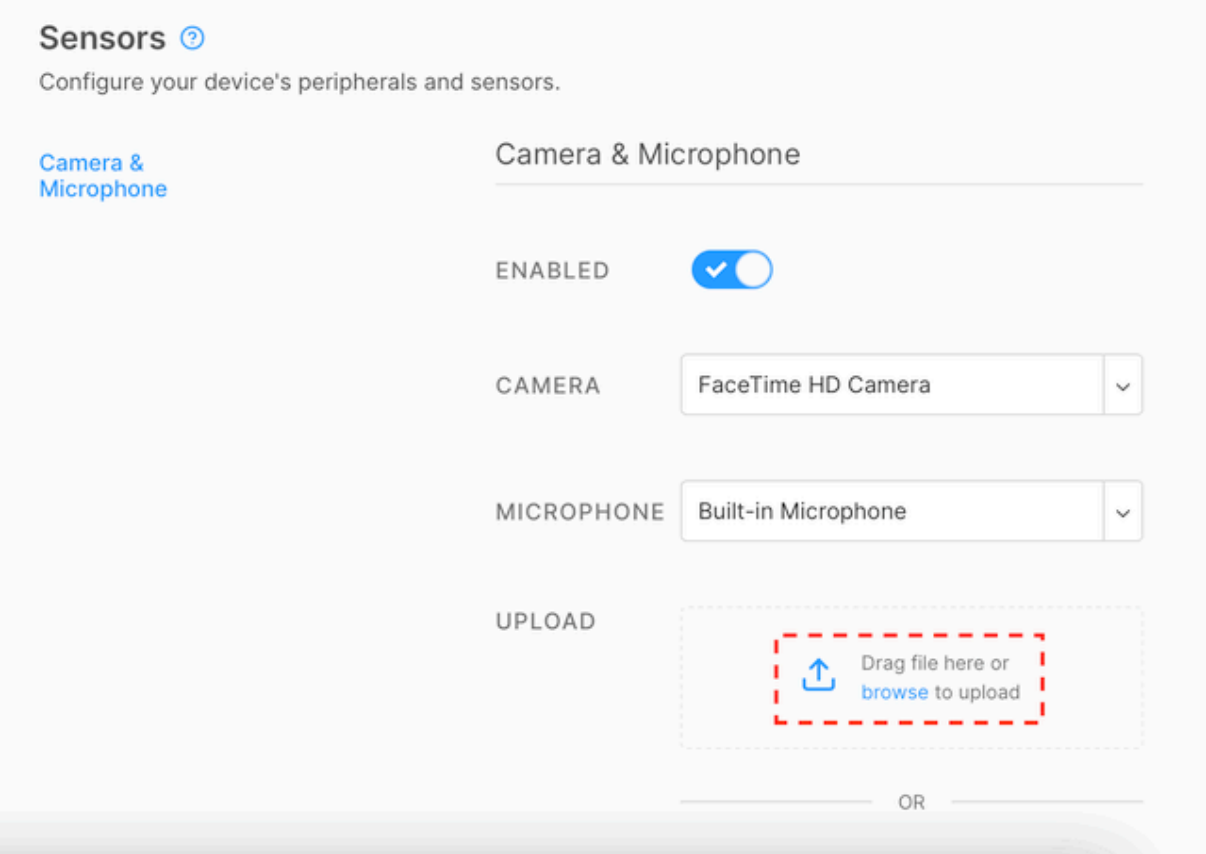
Upload a File - Web Interface

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-89: Upload the



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

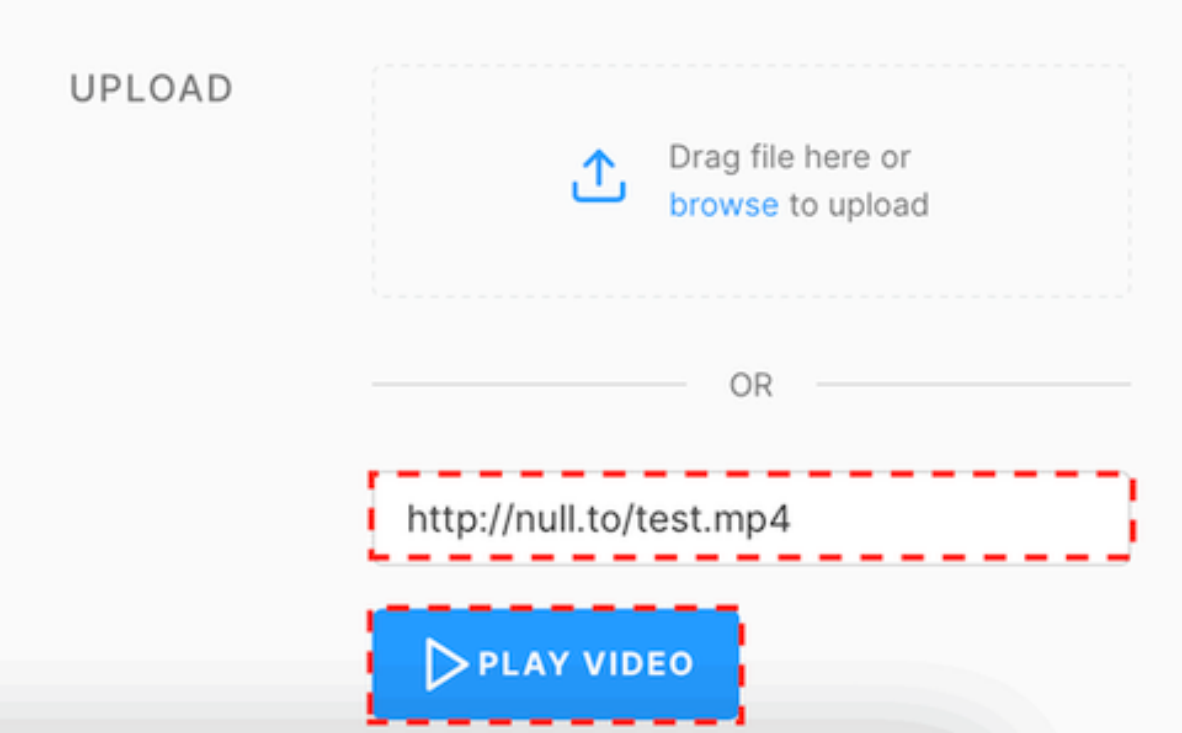
Link to a URL - Web Interface

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-90: Enter a URL



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

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`).

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.

```
{
  instanceId,
  imageId
}
```

5. Verify that you can see the video playing.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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.

3.3 i.MX 93 Applications Processor

Arm Virtual Hardware (AVH) provides a board model based on NXP i.MX 93 Evaluation Kit (i.MX93EVK). In the AVH documentation this virtual platform is referenced as "i.MX 93 Applications Processor" or in short "i.MX 93".

This chapter guides through pages specific to the i.MX 93 Applications Processor board. It explains how to get started, lists the hardware features supported in the model and provides usage examples relevant for it. The chapter is stuctured as follows:

- **Getting started:**
  - Quickstart for i.MX 93 Applications Processor provides a simple step-by-step guide on setting up a virtual i.MX 93 board.
- **Board model details:**
  - Supported Hardware Components for i.MX 93 lists the hardware components supported in the AVH i.MX 93 model.
- **Firmware:**
  - Stock Firmware for i.MX 93 explains the ready-to-use firmware packages available for virtual i.MX 93 Applications Processor.

3.3.1 Quickstart for i.MX 93 Applications Processor

This page guides you through the steps for creating an [Arm Virtual Hardware i.MX 93] with a stock firmware, and shows how to access its Linux command line interface.

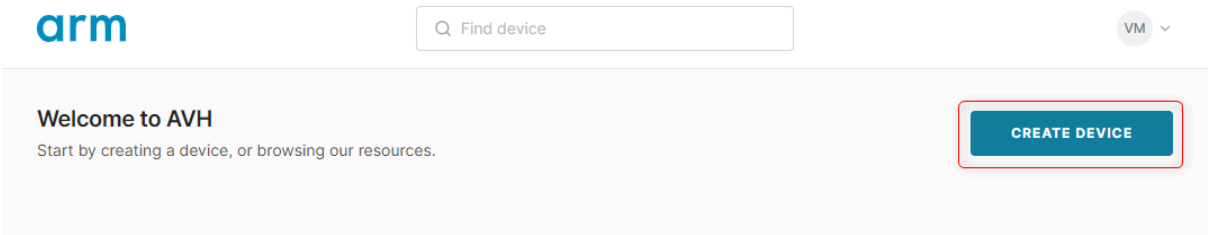
Create your device

To create your Arm Virtual Hardware i.MX 93 instance, do the following:

1. Log in to AVH with your Arm account at https://app.avh.arm.com. If you do not have an Arm account you can register here.
2. After login you should land on **Devices** page. There click **CREATE DEVICE**:

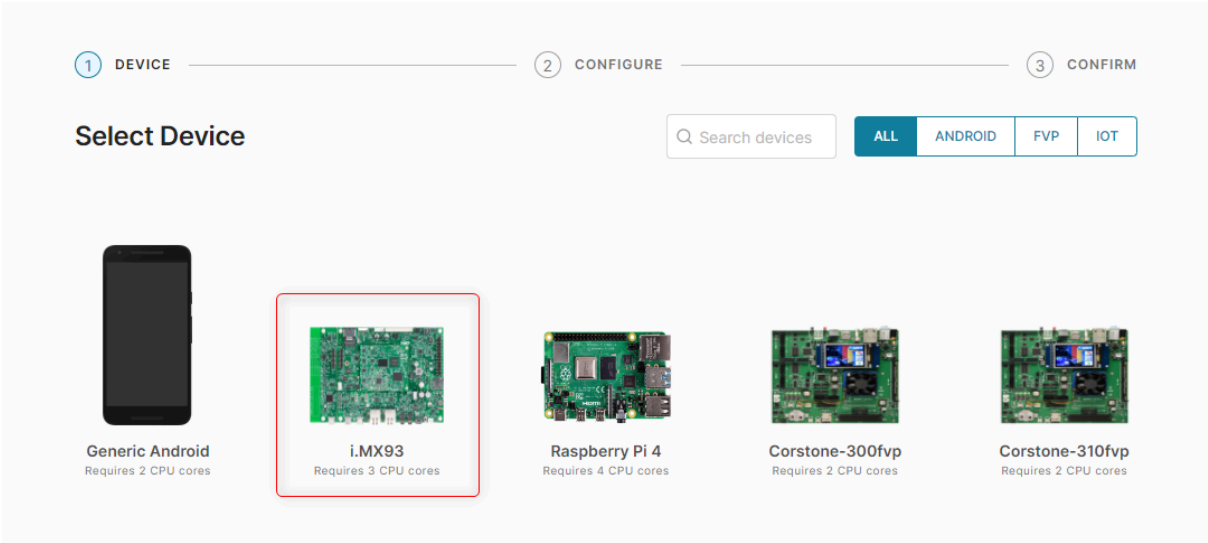
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-91: AVH Devices page



3. On the **Select Device** view click on the **i.MX93** board to proceed with creation of corresponding virtual instance.

Figure 3-92: Select i.MX93 device



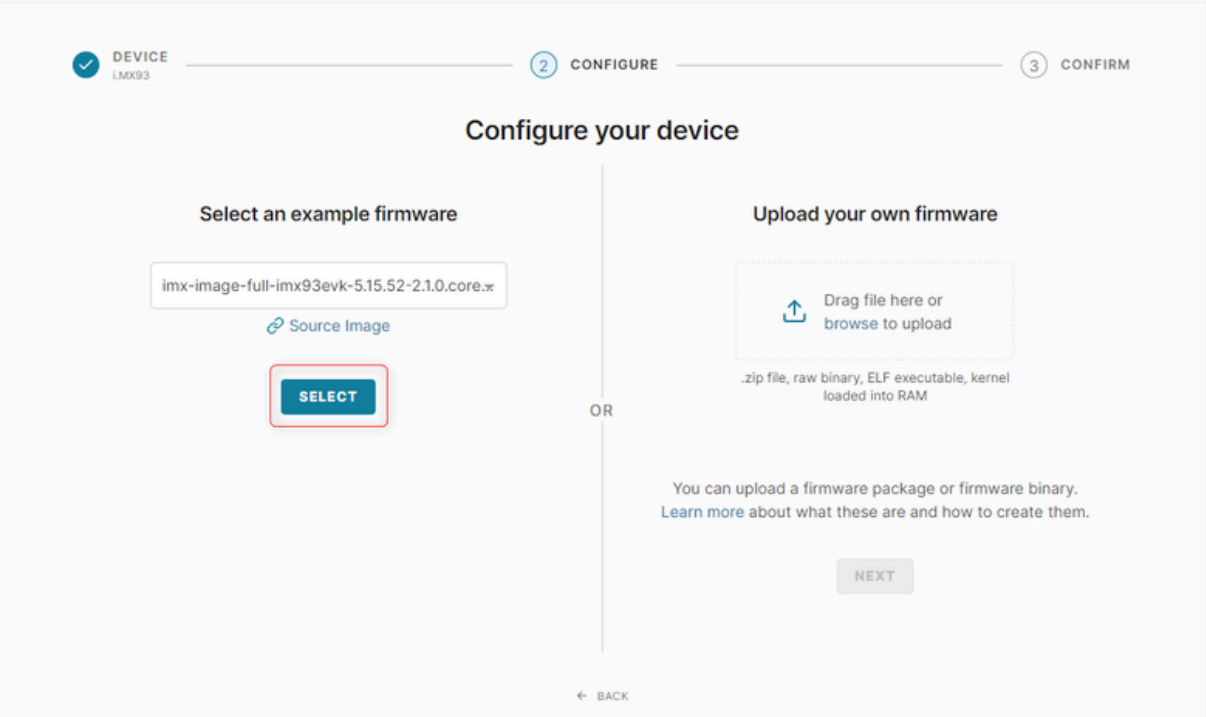
Note

You can narrow down the list of displayed boards by selecting the target category on the top-right side between **| ALL | ANDROID | FVP | IOT**. The i.MX 93 device can be found under IOT and ALL tabs.

4. On the **Configure your device** screen, you will see a dropdown menu **Select an example firmware** with stock firmware options available for the target platform in AVH. For this quickstart example we will keep the default selection. Click **SELECT**.

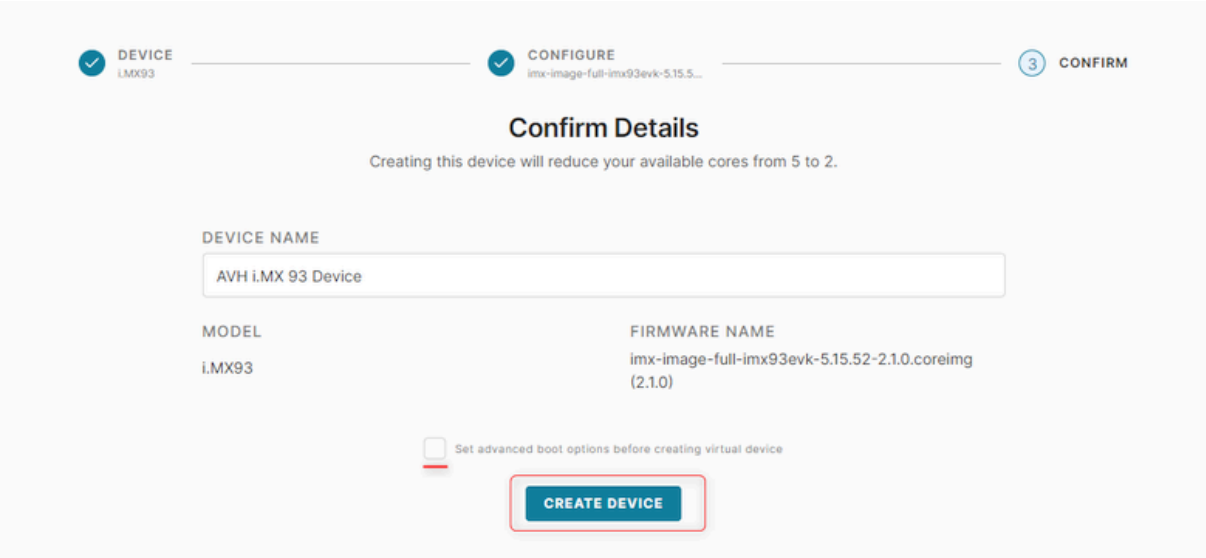
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-93: Default firmware selection for AVH i.MX93



5. On the **Confirm Details** screen you can specify the device name and verify your selections. Keep the advanced boot options unchecked and then click **CREATE DEVICE**.

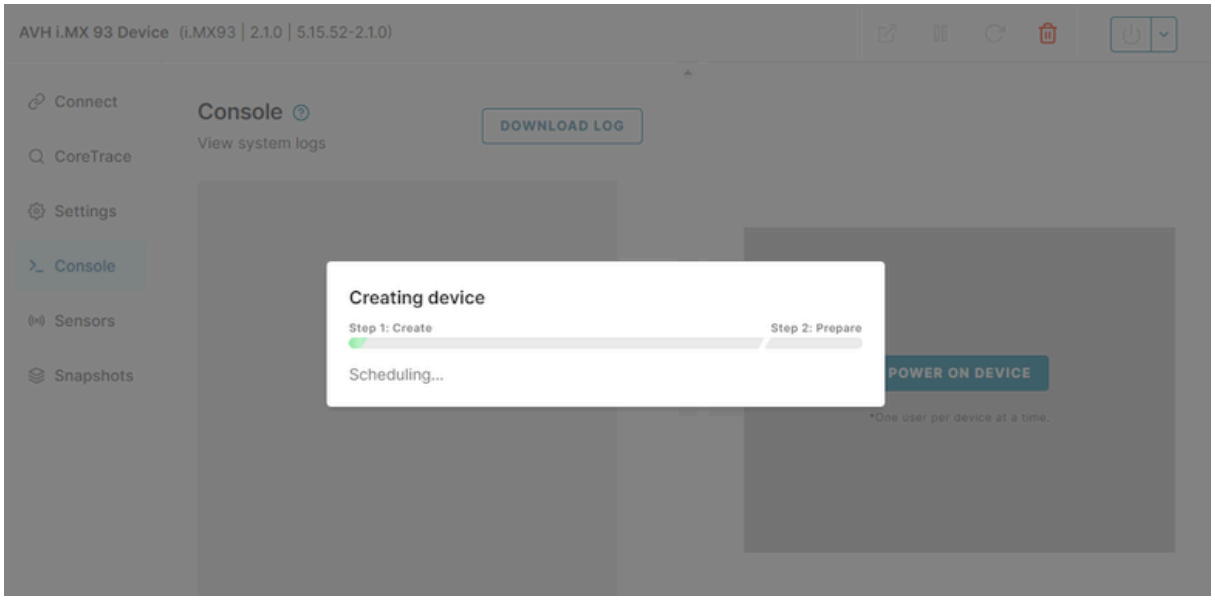
Figure 3-94: Confirmation screen



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-95: Creating device

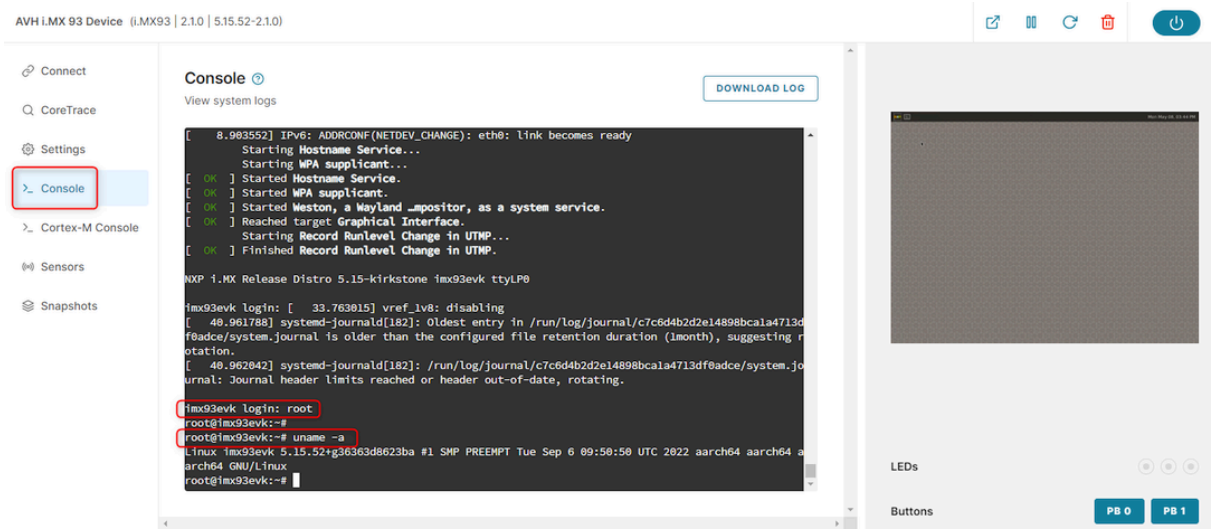


7. Your virtual i.MX 93 Applications Processor board is now ready to go! You shall see a Linux console, display, three output LEDs, and two input buttons.

Log in to console

1. Let's log in to the Linux command line interface on the device. Click **Console** tab in device's menu, if not open by default. Type in the default username: `root` then hit the Return/Enter key. You should see the display name change to `root@imx93evk`, meaning you are logged in. You can verify the system details with `uname -a` command.

Figure 3-96: AVH iMX93



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.3.2 Supported Hardware Components for i.MX 93

The virtual i.MX 93 Applications Processor board supports all blocks except the following:

- EdgeLock
- FlexIO
- WiFi
- Bluetooth
- OTFAD

The following blocks can be accessed from software but writes to them will not alter system behavior:

- SRC - System Reset Controller
- GPC - General Power Controller
- MCM - Miscellaneous Control Module
- AXBS - Crossbar Switch
- DDRC - DDR Controller

3.3.3 Stock Firmware for i.MX 93

Arm Virtual Hardware provides several ready-to-use firmware packages (stock firmware) to run on virtual iMX8m Arm Cortex Complex devices. These are zip archives prepared according to the AVH Linux firmware package format explained in Linux Firmware Packages. Following stock firmware packages are available:

- **imx-image-full-imx93evk-5.15.52-2.1.0** firmware package runs Linux 5.15.52-2.1.0 for i.MX 93 EVK on its virtualized instance. See Embedded Linux for i.MX Applications Processors for more information and documentation of i.MX Linux.

When creating a new virtual board (see example in Quickstart for i.MX 93 Applications Processor), on the **Configure your device** screen in the drop-down menu **Select an example firmware** you can choose the target stock firmware that the virtual board instance shall start with.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.4 STM32U5 IoT Discovery Kit

Arm Virtual Hardware (AVH) provides a board model based on the B-U585I-IOT02A Discovery kit for the STM32U5 series from STMicroelectronics. Throughout the AVH documentation this virtual platform is referenced as "STM32U5 IoT Discovery" or "STM32U5" in short form.

This chapter guides through usage specific to the virtual STM32U5 IoT Discovery Kit. It explains how to get started, lists the hardware features supported in the model and provides usage examples relevant for it. The chapter is stuctured as follows:

- **Getting started:**
  - Quickstart for STM32U5 IoT Discovery provides a simple step-by-step guide on setting up a virtual STM32U5 IoT Discovery board with a default example.
- **Board model details:**
  - Supported Hardware Components for STM32U5 lists the hardware components supported in the virtual STM32U5.
- **Firmware:**
  - Stock Firmware for STM32U5 explains the ready-to-use firmware packages available for virtual STM32U5 IoT Discovery Kit on AVH.
  - Storage Files for STM32U5 provides information about the storage files in AVH firmware packages for STM32U5 IoT Discovery.
- **Usage examples:**
  - IoT HTTP Web Server on STM32U5 example runs a web-server on the STM32U5 device that displays on-board sensor measurements and can be accessed from a web browser remotely.
  - TF-M example on STM32U5 shows usage of Trusted Firmware for Cortex-M (TF-M) for secure operations.
  - Detect Audio through the Microphone on STM32U5 Board describes how to detect audio level using the virtual STM32U5 board.

3.4.1 Quickstart for STM32U5 IoT Discovery

This page guides you through the steps for creating and using a virtual STM32U5 IoT Discovery Kit.

We will use stock firmware `WiFiBasics (1.1.0)` that demonstrates basic WiFi communication features available on this board. The source code for this firmware example is available in the STM32Cube Firmware Package for U5 and can be obtained as "Nx_Network_Basics" example from the STM32Cube.

The example itself works also on a real hardware, and can be used as follows:

- On power up the application performs a network scan function and connects to the default network. The serial console displays the help command list.
- End-user can enter any of the proposed commands:

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- ping : ping a remote host and display time to perform ping request
- echo : send 10 x [1000..2000] bytes to an echo server (armMBED@ one) and check that return data are identical to the one sent
- scan : list available WiFi access points
- http : download a file from an HTTP server, http (default url is `http://public.st.free.fr/500MO.bin`)

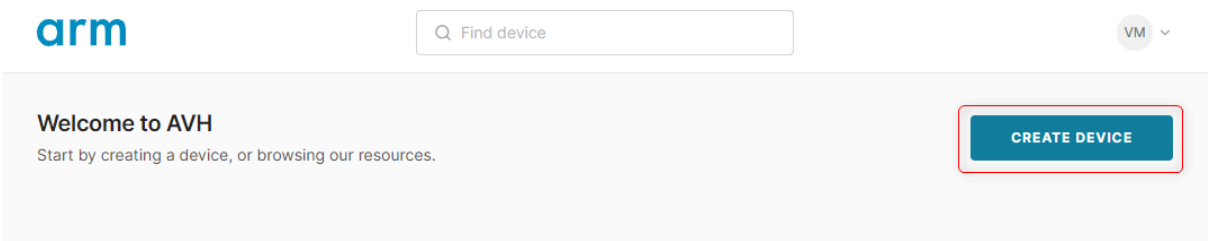
Follow the steps below to execute this example on a virtual STM32U5 IoT Discovery Kit.

Creating your device

To create your device, do the following:

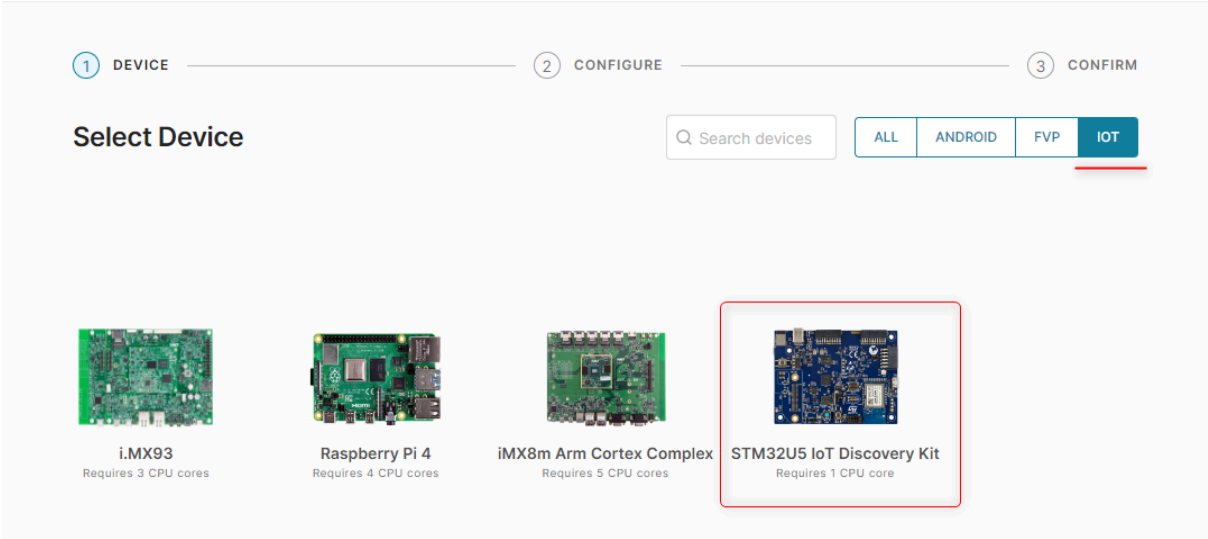
1. Log in to AVH with your Arm account at https://app.avh.arm.com. If you do not have an Arm account you can register here.
2. After login you should land on **Devices** page. There click **CREATE DEVICE**:

Figure 3-97: Start with device creation



3. Click on the **STM32U5 IoT Discovery Kit** to proceed with creation of corresponding virtual instance.

Figure 3-98: Select STM32U5 IoT Discovery Kit



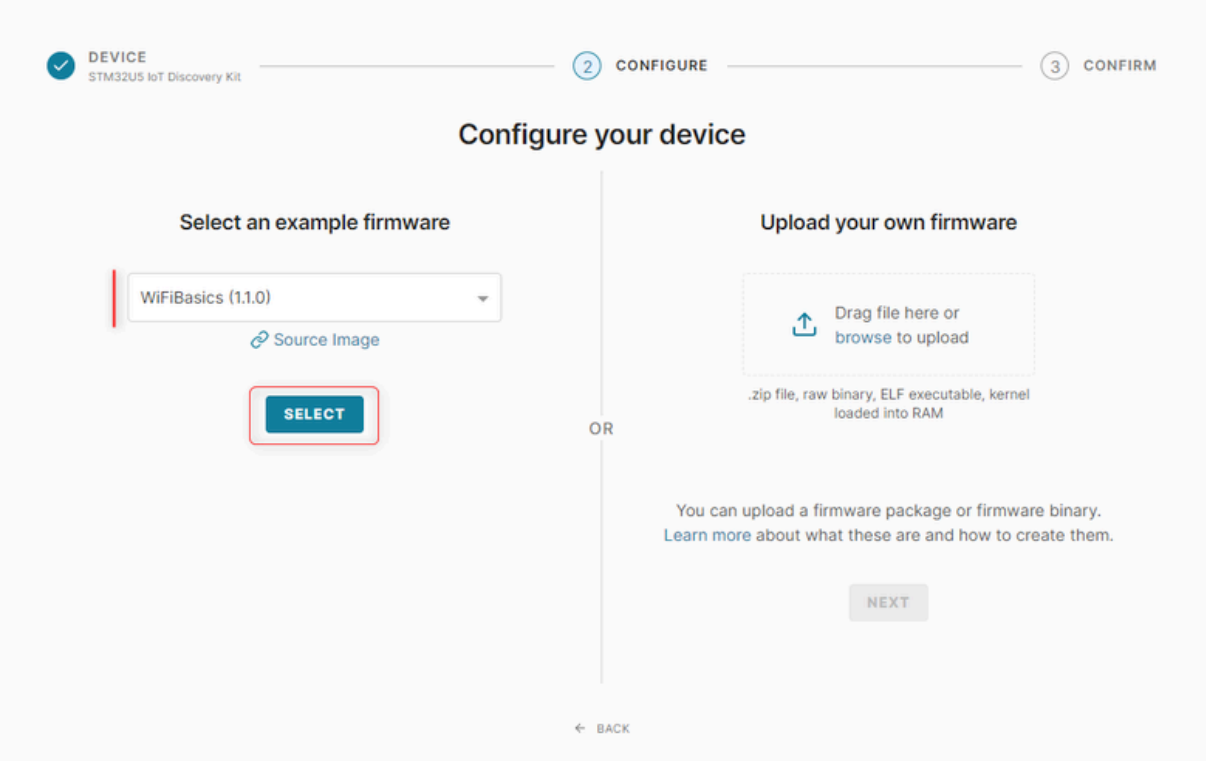
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

You can narrow down the list of displayed boards by selecting the category tab on the top-right side between **| ALL | ANDROID | FVP | IOT**. The STM32U5 IoT Discovery Kit can be found under IOT and ALL tabs.

4. On the page **Configure Your Device**, you will see a dropdown with stock firmware available for the target platform, see Stock Firmware for STM32U5 for details. For this example, choose `WiFiBasics (1.1.0)` and click **SELECT**.

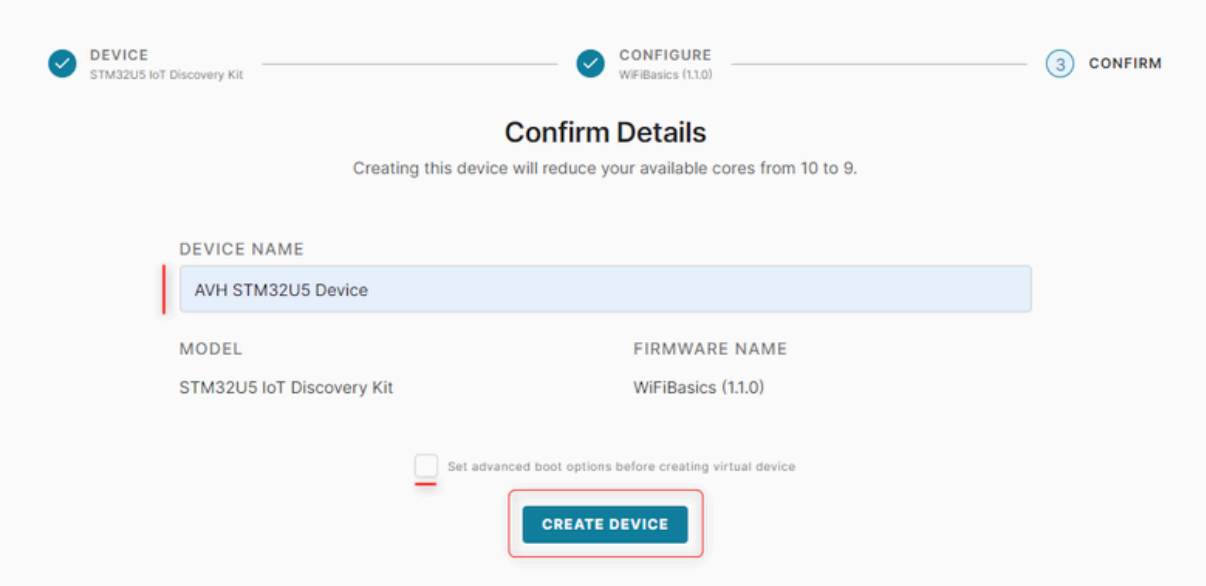
Figure 3-99: Select stock firmware example WiFiBasics (1.1.0)



5. On the **Confirm Details** screen you can specify the device name and verify your selections. Keep the advanced boot options unchecked and then click **CREATE DEVICE**.

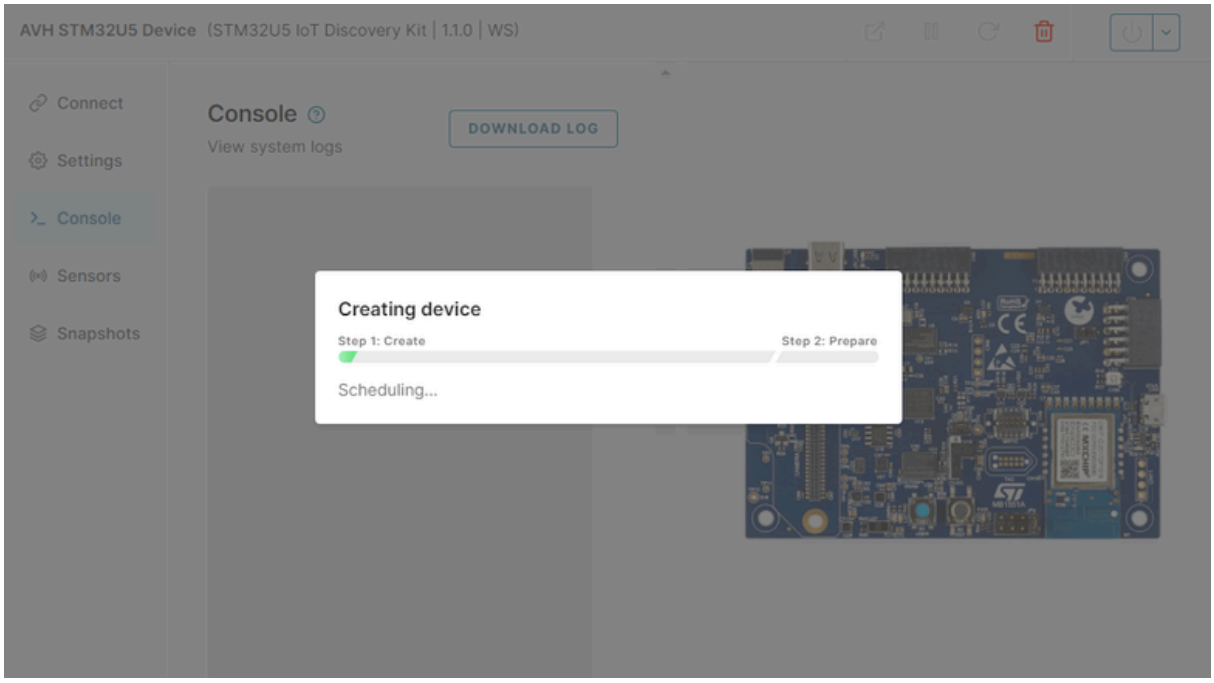
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-100: Confirmation screen



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

Figure 3-101: Creating devices



7. Your STM32U5 IoT Discovery Kit is now ready to go! You shall see its serial console, board picture, and underneath it one switch, three output LEDs, one input button.

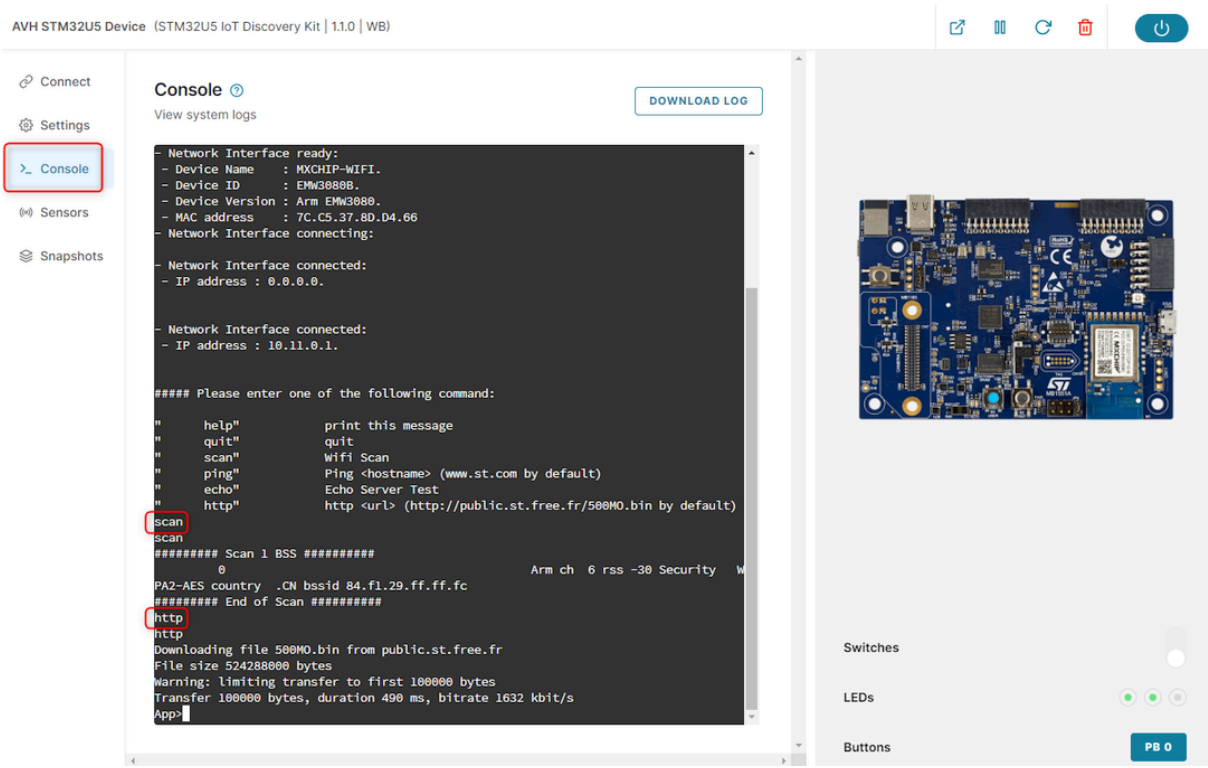
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Running the example

The program scans for available WiFi networks and connects to the default one ("Arm" in this case). There is a console menu to test various functions.

1. Click **Console** tab in device's menu, if it is not open by default. You shall see a console menu with available commands. For example, use `scan` command to see the available WiFi network "Arm", or use `http` to download a binary file from the ST website.

Figure 3-102: Type SSID



Next steps

For next steps see STM32U5 IoT Discovery Kit overview page with references to more usage examples.

3.4.2 Supported Hardware Components for STM32U5

Table of Contents

- High-Level Summary
- Detailed List of Components

High-Level Summary

The virtualized STM325U5 board models the following:

- 802.11 b/g/n compliant Wi-Fi

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- 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

Detailed List of Components

The detailed list of components is as follows:

- CPU
  - ICACHE
  - DCACHE
  - RAMCFG
  - 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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

    - 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
    - OCTOSPI
    - FDCAN
    - UCPD
    - GPIO
- Storage
    - FLASH
    - SDMMC
    - FSMC
    - DLYB
- Security
    - GTZC
    - TAMP
    - RNG
    - AES / SAES
    - HASH
    - PKA
- Audio/Video/Touch
    - MDF

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

  - 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

3.4.3 Stock Firmware for STM32U5

AVH provides several ready-to-use firmware packages (stock firmware) to run on virtual STM32U5 IoT Discovery devices. These are zip archives prepared according to the format explained in MCU Firmware Packages.

When creating a new virtual board (as shown in Quickstart for STM32U5 IoT Discovery), in the **Configure your device** window you have a corresponding drop-down menu where you can choose the target stock firmware that the virtual board instance shall run. Following stock firmware packages are available:

- **WiFiBasics (1.1.0)** firmware demonstrates basic WiFi connectivity via the on-board MXCHIP EMW3080 module. This stock firmware is used in the Quickstart for STM32U5 IoT Discovery that describes it in more details.
- **IOT_HTTP_WebServer (1.1.0)** firmware corresponds to the standard demonstration program for B-U585I-IOT02A hardware board and runs on the device a web-server with LED controls and readings from its environmental on-board sensors. See IoT HTTP Web Server on STM32U5 for mode details.
- **STM32U5-TFM (1.1.0)** firmware uses the security capabilities of the STM32U5 device and runs an example with Trusted Firmware for Cortex-M (TF-M). Page TF-M example on STM32U5 provides the details.
- **STM32U5-ToF (1.1.0)** is a simple ranging application for the proximity sensor.

Alternatively, you can also run custom firmware with own programs as explained in MCU Firmware Packages, or try out our referenced custom examples such as Detect Audio through the Microphone on STM32U5 Board.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.4.4 Storage Files for STM32U5

Following storage files can be used in the firmware packages for virtual STM32U5 IoT Discovery Kit devices. Also see MCU Firmware Packages.

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

Note

For more information about the flashoptions see 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
  - block_size: 512
  - blocks: 1
  - max: 512 bytes
- **sdcard** - SD card
  - block_size: 512
  - blocks: 1953125
  - max: 1000000000 bytes
- **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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- max: 33280 bytes

Note

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

3.4.5 IoT HTTP Web Server on STM32U5

The stock firmware `IOT_HTTP_WebServer (1.1.0)` corresponds to the B-U585I-IOT02A demonstration program at st.com/resource/en/compiled_demos/b-u585i-iot02a_demo.zip.

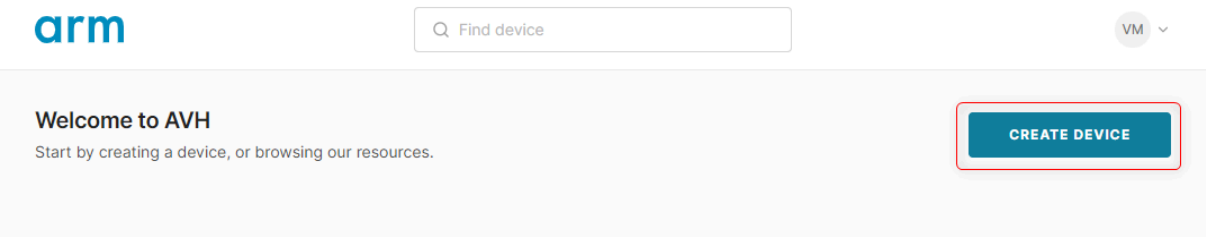
The example runs a web-server on the device displaying the readings from its environmental on-board sensors. The device uses its on-board WiFi module to connect to the specified WLAN SSID, and then its web-server can be accessed remotely from a web-browser. The example source code is available in the STM32Cube Firmware Package for U5 and can be obtained as "IOT_HTTP_WebServer" example from the STM32Cube.

Creating your device

To create your device, do the following:

1. Log in to AVH with your Arm account at https://app.avh.arm.com. If you do not have an Arm account you can register for it.
2. After login you should land on **Devices** page. There click **CREATE DEVICE**:

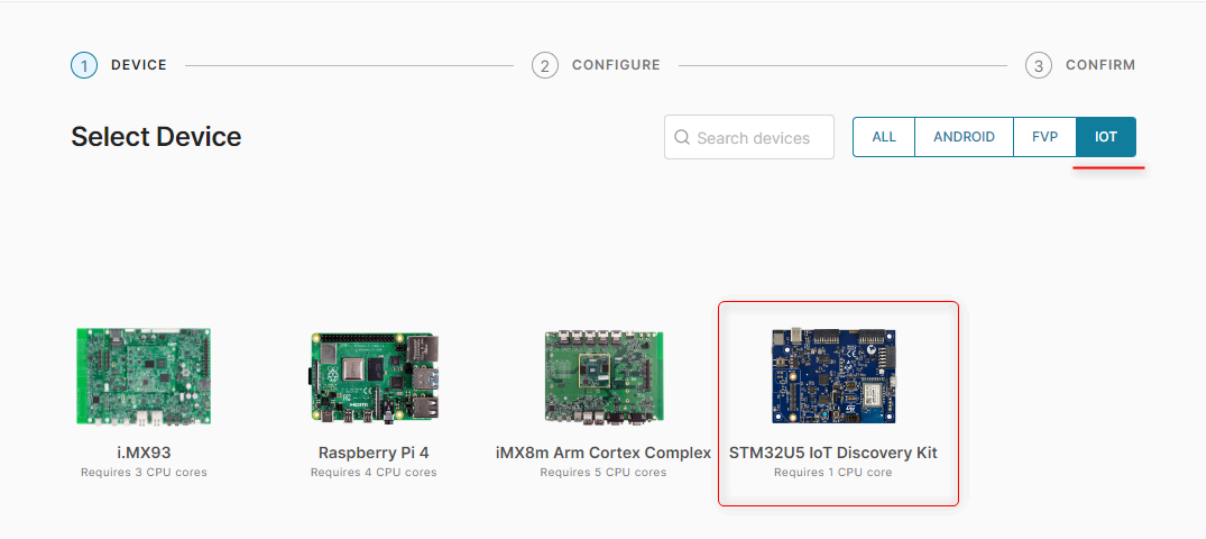
   Figure 3-103: Start with device creation



3. Click on the **STM32U5 IoT Discovery Kit** to proceed with creation of corresponding virtual instance.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-104: Select STM32U5 IoT Discovery Kit



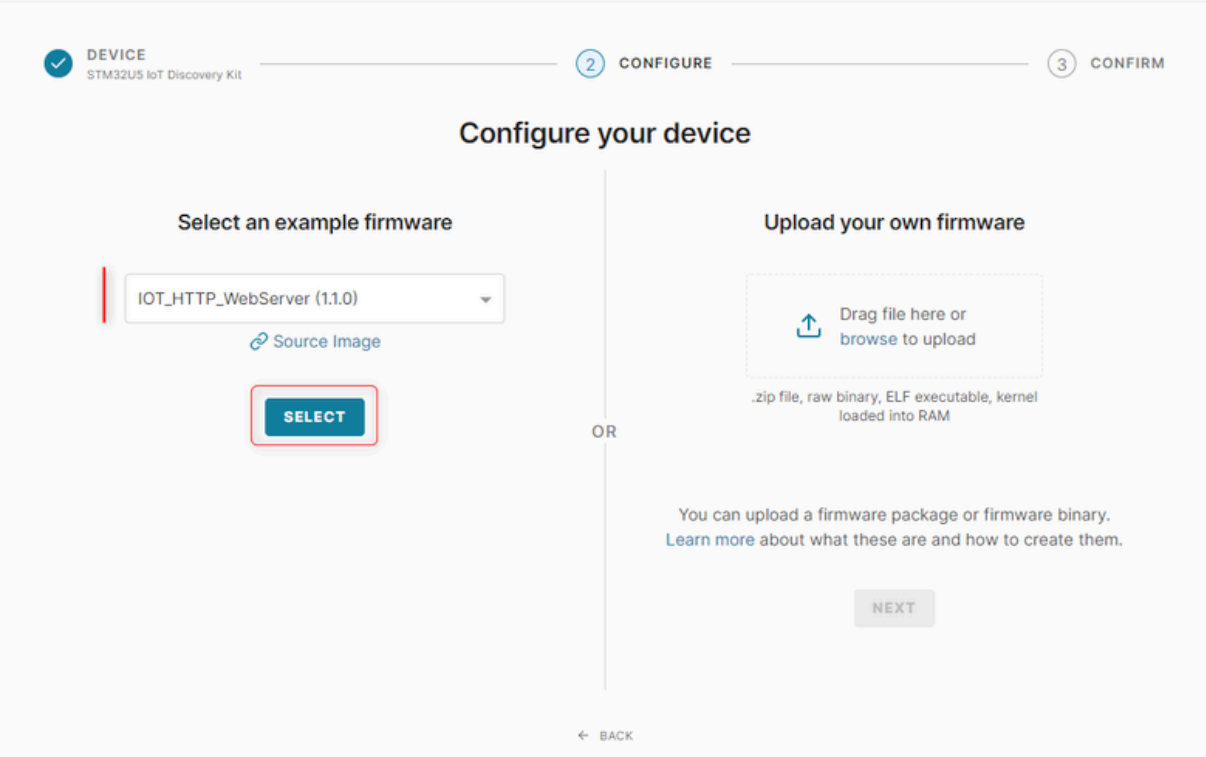
Note

You can narrow down the list of displayed boards by selecting the category tab on the top-right side between **| ALL | ANDROID | FVP | IOT**. The STM32U5 IoT Discovery Kit can be found under IOT and ALL tabs.

4. On the page **Configure Your Device**, you will see a dropdown with stock firmware available for the target platform, see Stock Firmware for STM32U5 for details. For this example, choose `IOT_HTTP_WebServer (1.1.0)` and click **SELECT**.

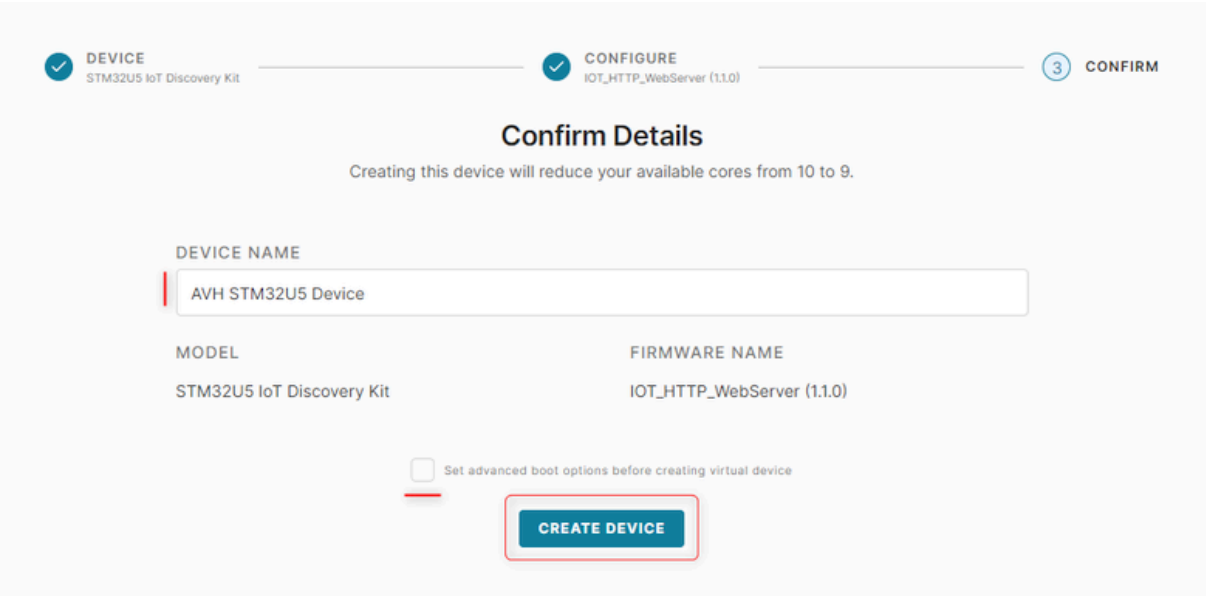
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-105: Select stock firmware example IOT_HTTP_WebServer (1.1.0)



5. In the confirmation screen you can specify the device name and verify your selections. Keep the advanced boot options unchecked and click **CREATE DEVICE**.

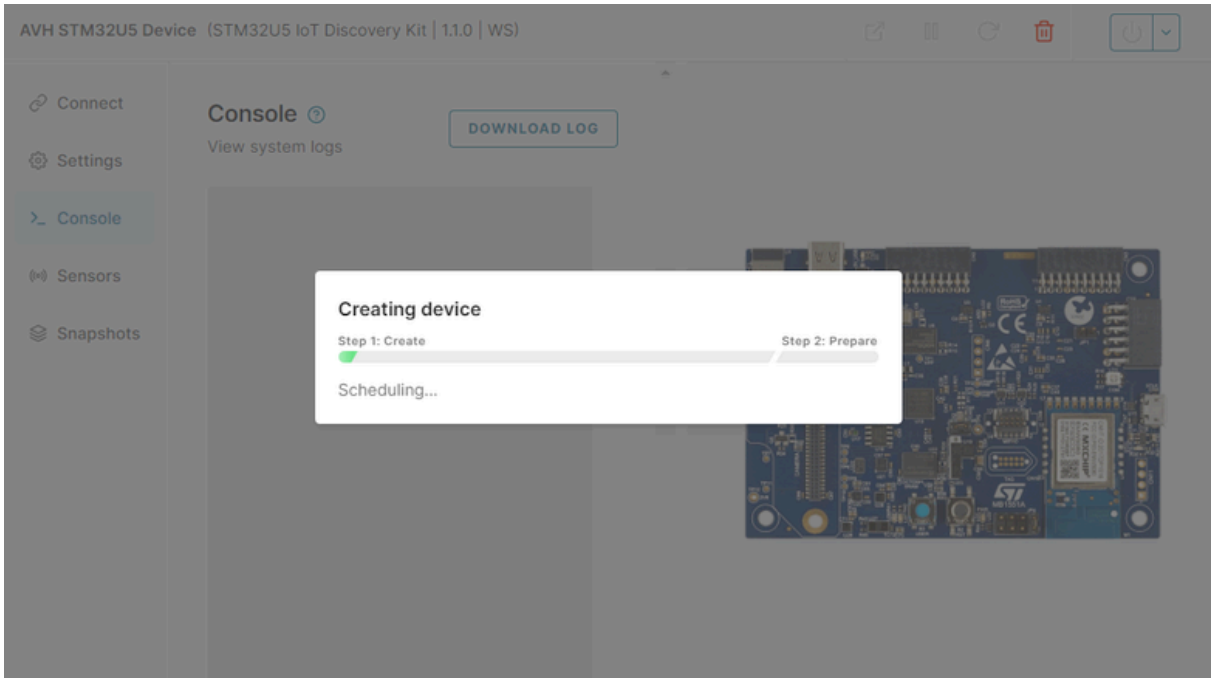
Figure 3-106: Confirmation screen



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-107: Creating devices



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

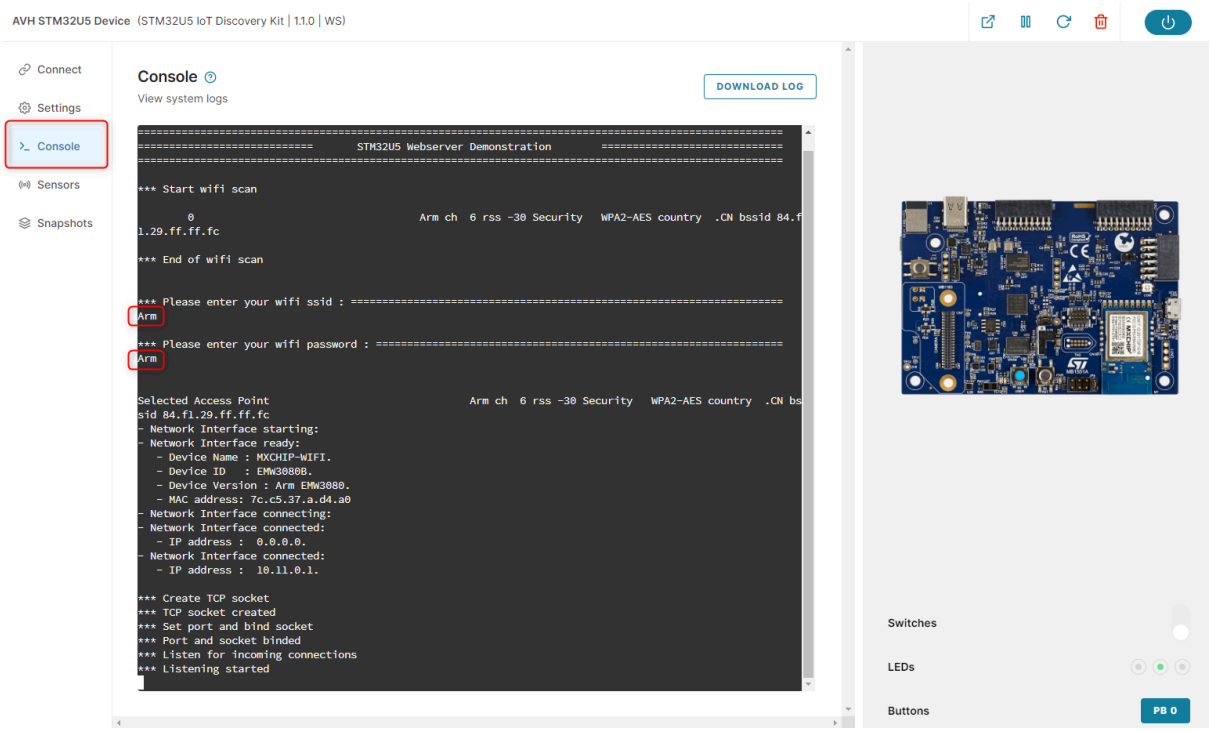
Running the example

To run the example, we need to join the board to a virtual WiFi network first:

1. Click **Console** tab in the device's menu, if it is not open by default. Type SSID from the list (`Arm` in this case) and anything as a password (`Arm` in this case).

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-108: Type SSID



Note

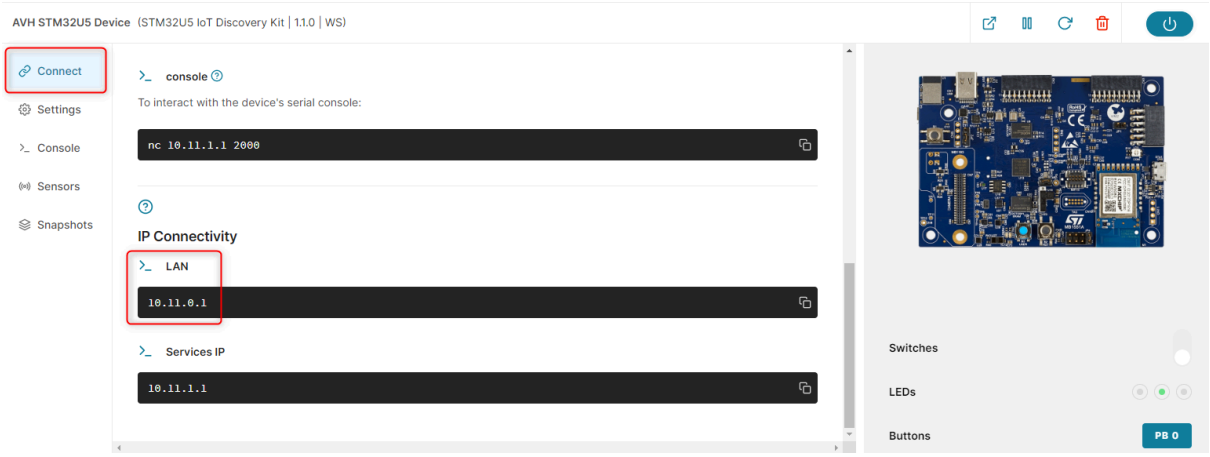
The SSID and password are case-sensitive.

Now the STM32U5 IoT Discovery Kit virtual device is running a sensor web-server.

2. To access the web-server on the virtual board we need to join its LAN. To do this from a local machine follow the steps in [Connecting to the VPN] and ensure the VPN connection is established.
3. In the **Connect** tab find the LAN IP address for the virtual board.

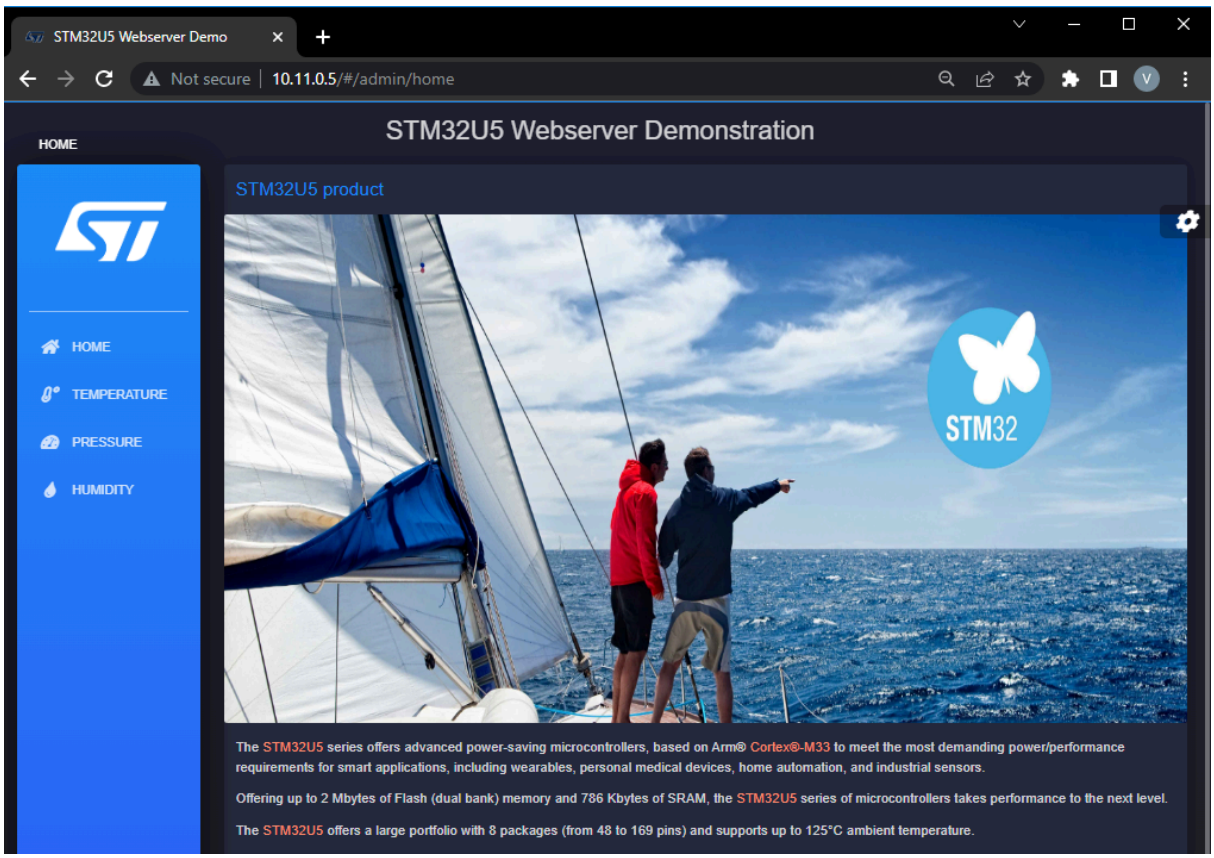
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-109: Type SSID



4. Enter the LAN IP in a web-browser on your local machine that is connected to the AVH VPN, and observe the ST web page as below:

Figure 3-110: Web-server default page

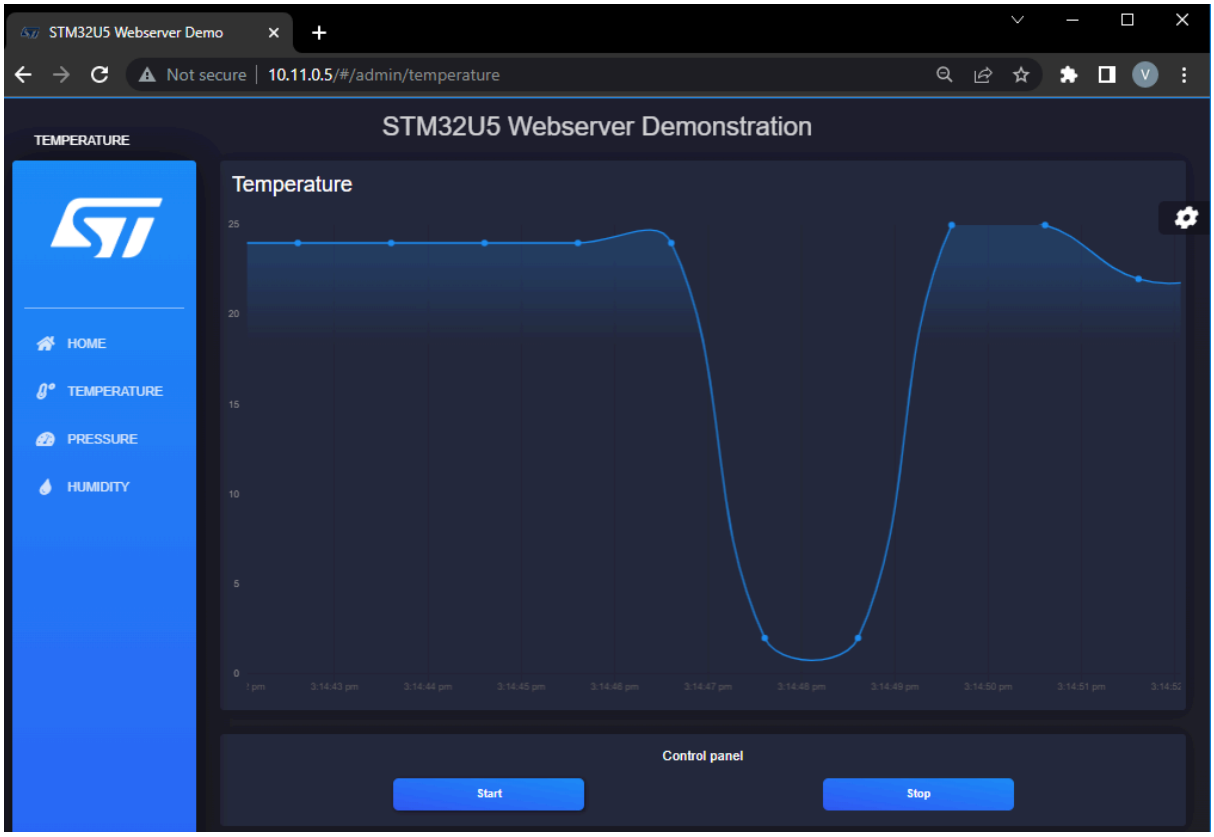


5. On the web-server go to Temperature menu and click Start button to capture the temperature sensor reading. Then on your AVH device page go to **Sensors** tab - [Environmental][.ui] and

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

change the value for the temperature sensor from default 25 degrees. On the web-server page you can see the changes captured.

Figure 3-111: Capturing temperature changes



Next steps

For next steps see STM32U5 IoT Discovery Kit overview page with references to more usage examples.

3.4.6 TF-M example on STM32U5

The stock firmware `STM32U5-TFM (1.1.0)` demonstrates use of Trusted Firmware for Cortex-M (TF-M) on a virtual STM32U5 IoT Discovery Kit for separating secure and non-secure programs.

For this example the **firmware** 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
```

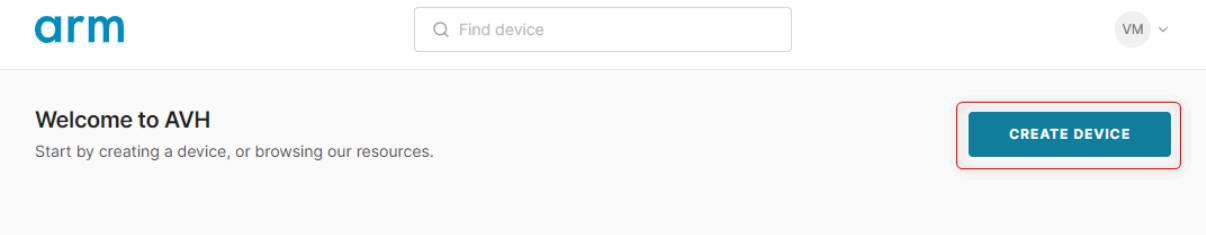
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Creating your device

To create your device:

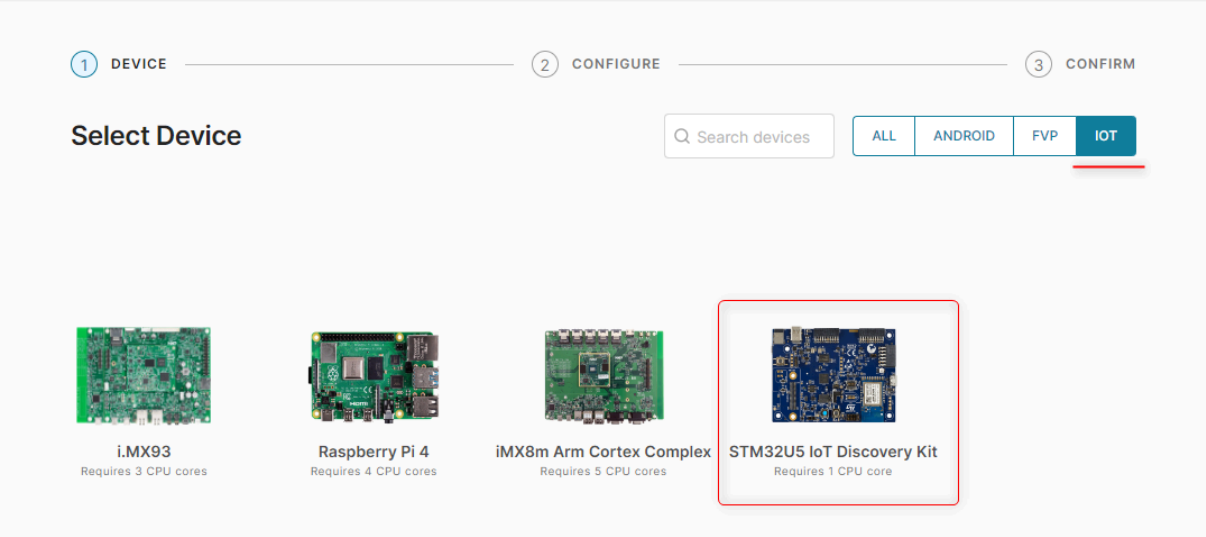
1. Log in to AVH with your Arm account at https://app.avh.arm.com.
2. When logged in you land on the AVH Devices page. Click **CREATE DEVICE**.

   Figure 3-112: Click Create Device



3. Click on the **STM32U5 IoT Discovery Kit** to proceed with its creation.

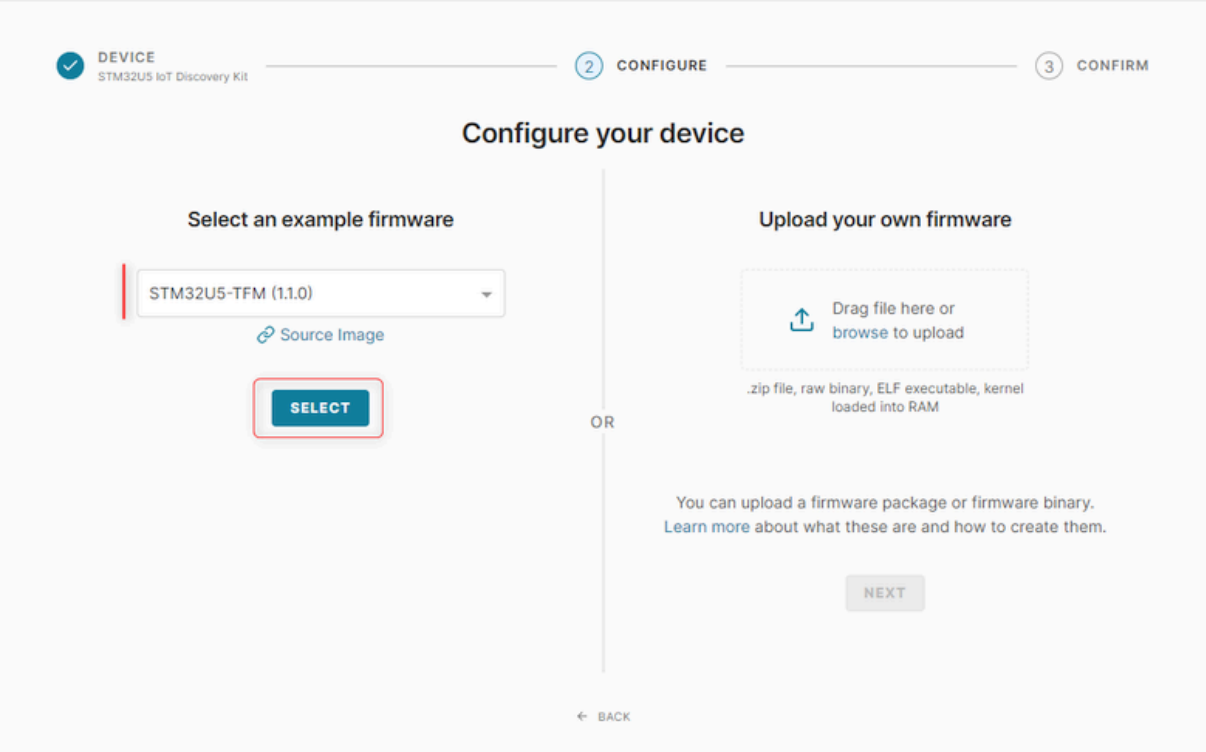
   Figure 3-113: Select device



4. On the page **Configure Your Device**, you will see a dropdown with stock firmware available for the target platform, see Stock Firmware for STM32U5 for details. For this example, choose `STM32U5-TFM (1.1.0)` in the stock firmware section and click **SELECT**.

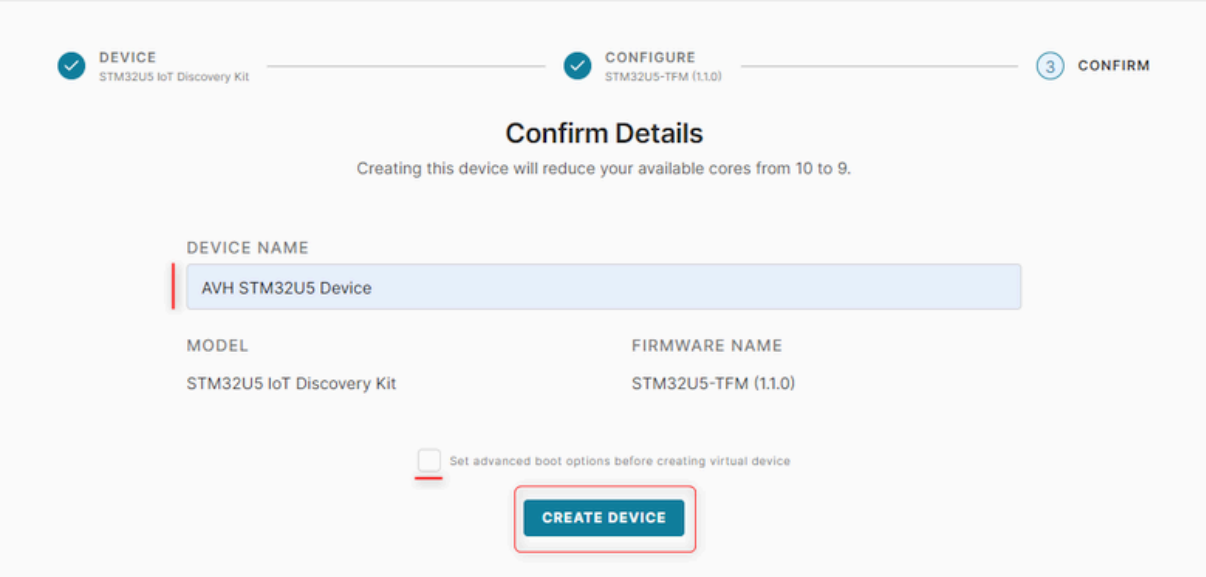
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-114: Configure your device



5. In the confirmation screen you can specify the device name and verify your selections. Keep the advanced boot options unchecked and click **CREATE DEVICE**.

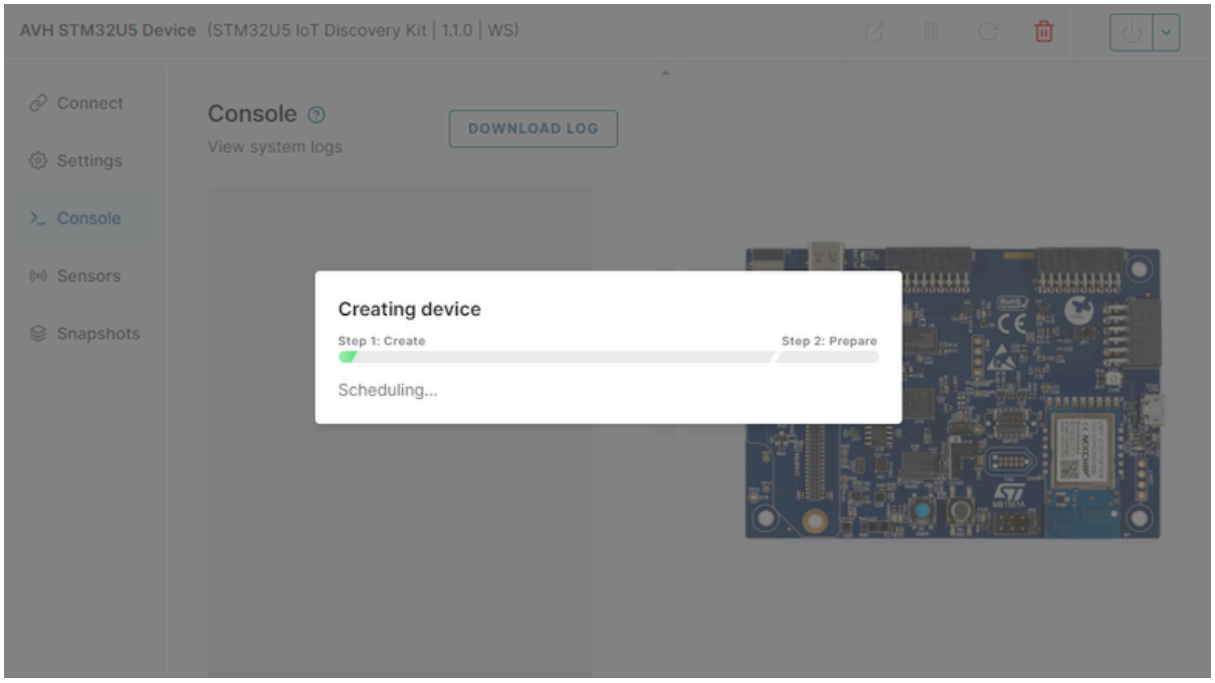
Figure 3-115: Confirm details



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-116: Creating device



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

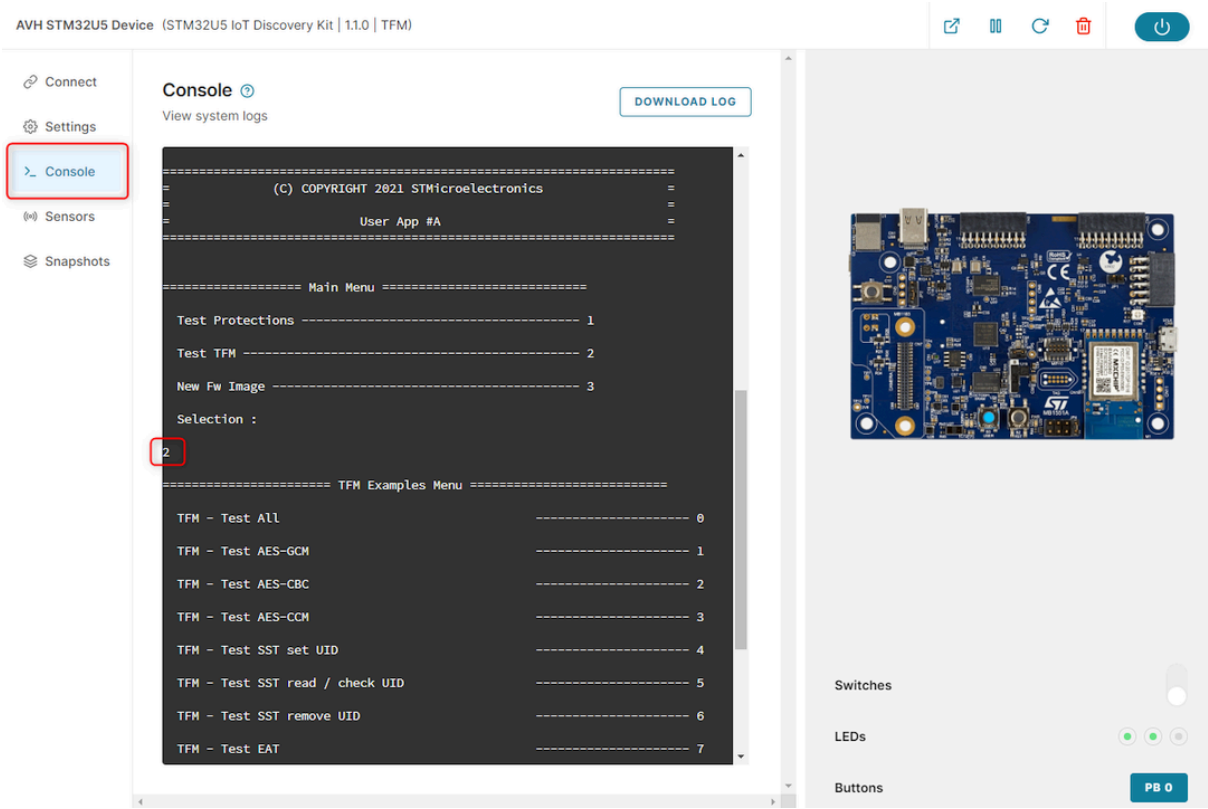
Running the example

To run the example:

1. Click **Console** tab in the device's menu, if it is not open by default. You shal see a console menu to run various tests using Trusted Firmware. For example type 2 (Test TFM) followed by return.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-117: Test TFM



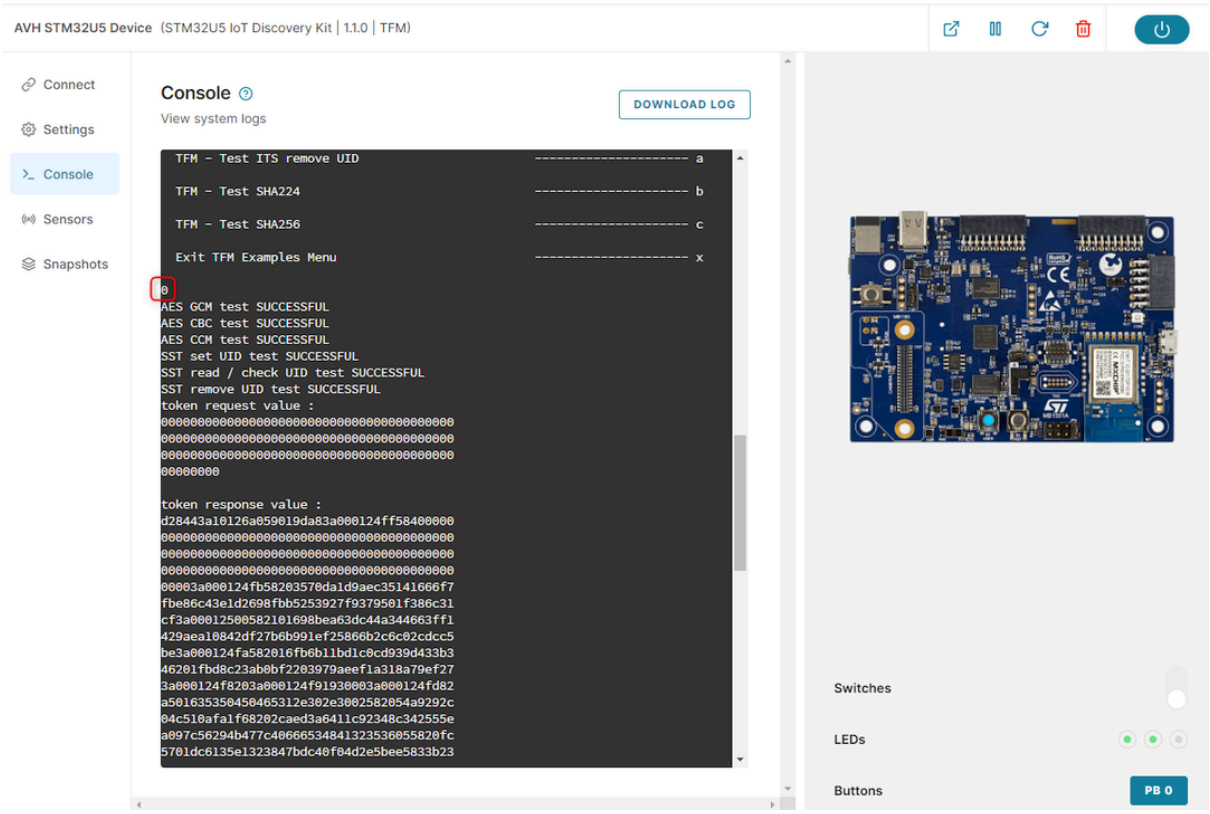
Note

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.

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-118: TFM - Test All



3. Scroll through the Console to observe the results.

3.4.7 Detect Audio through the Microphone on 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
- Boot the Custom Firmware and Detect Audio

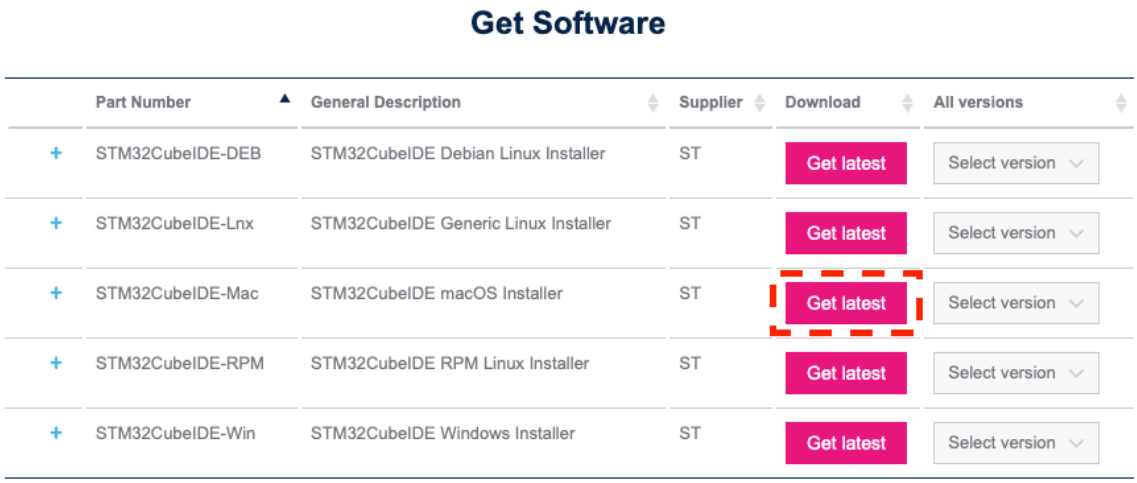
Build the ADF_AudioSoundDetector Firmware File

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.

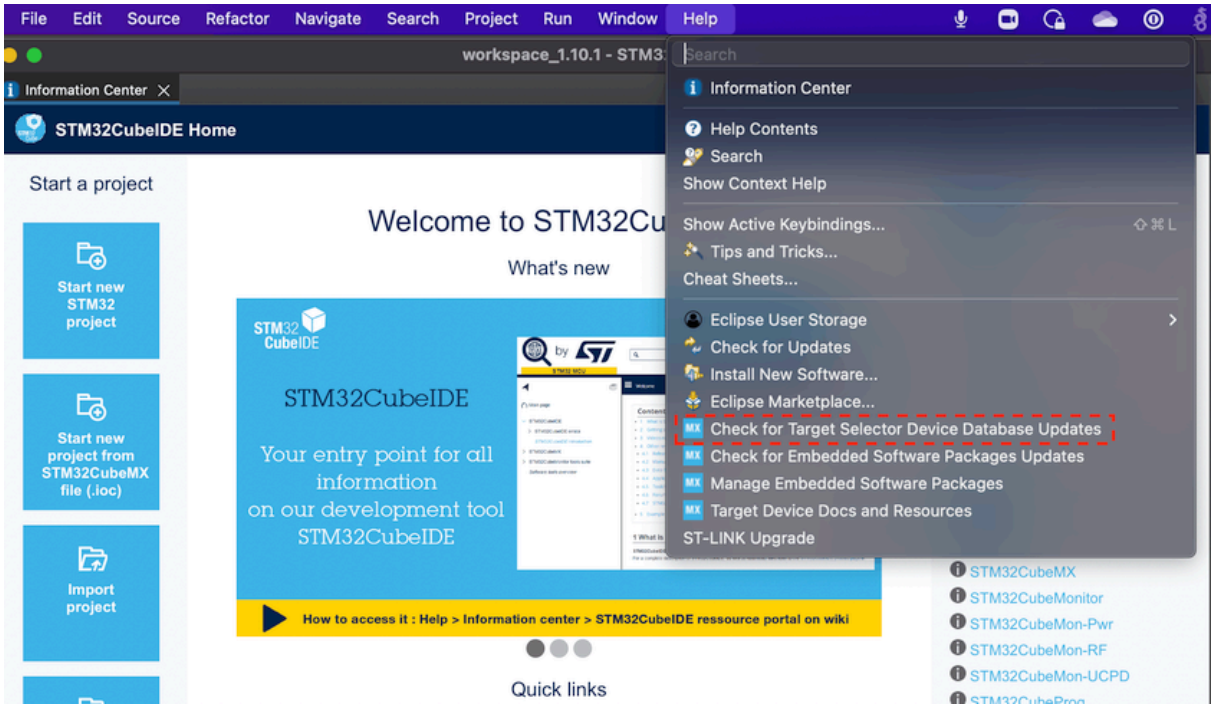
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-119: Download CubeIDE



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

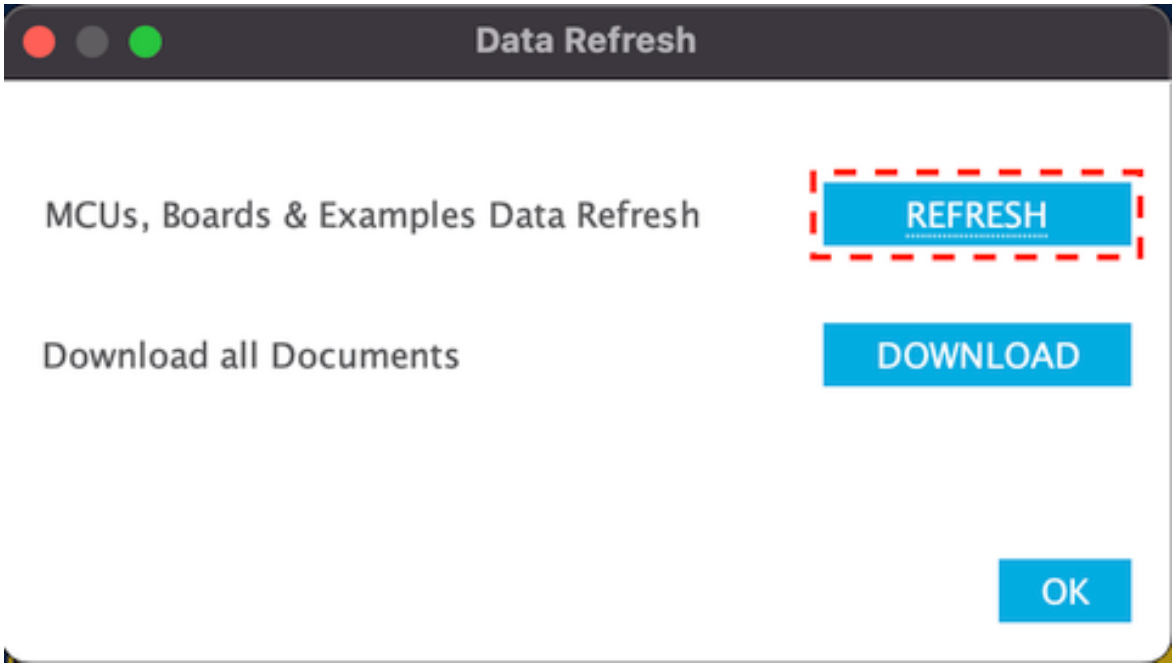
Figure 3-120: Check for updates



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

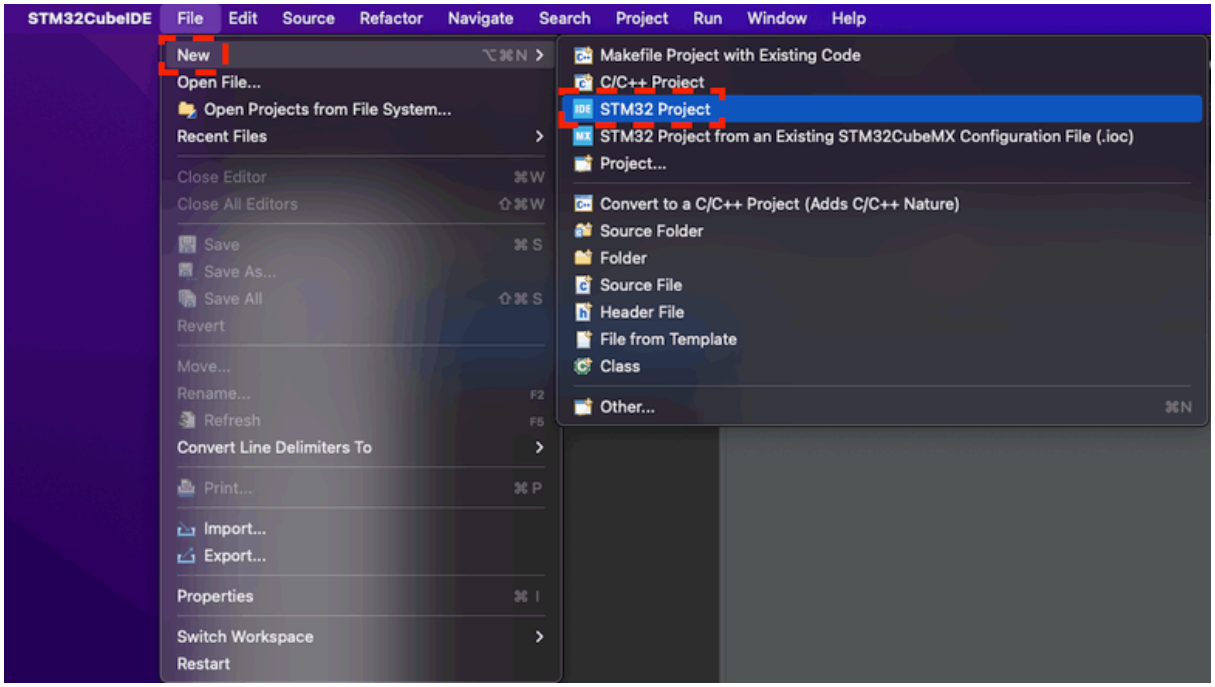
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-121: Refresh button



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

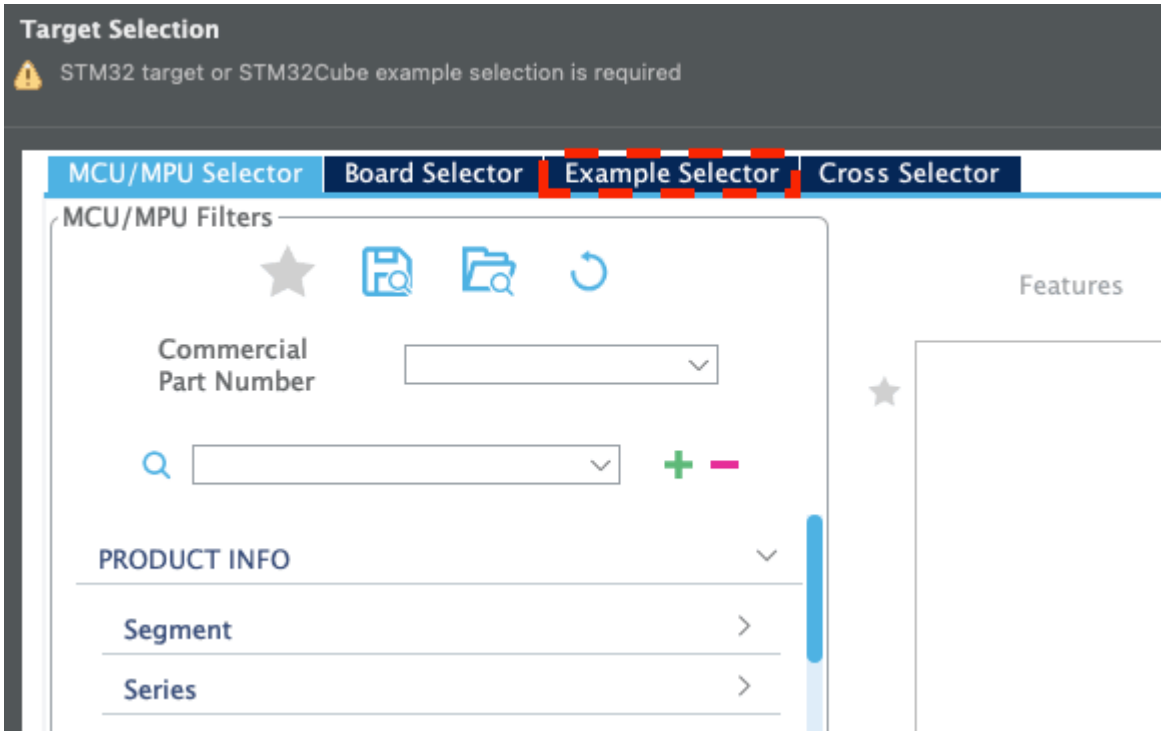
Figure 3-122: New project screen



5. Click on Example Selector.

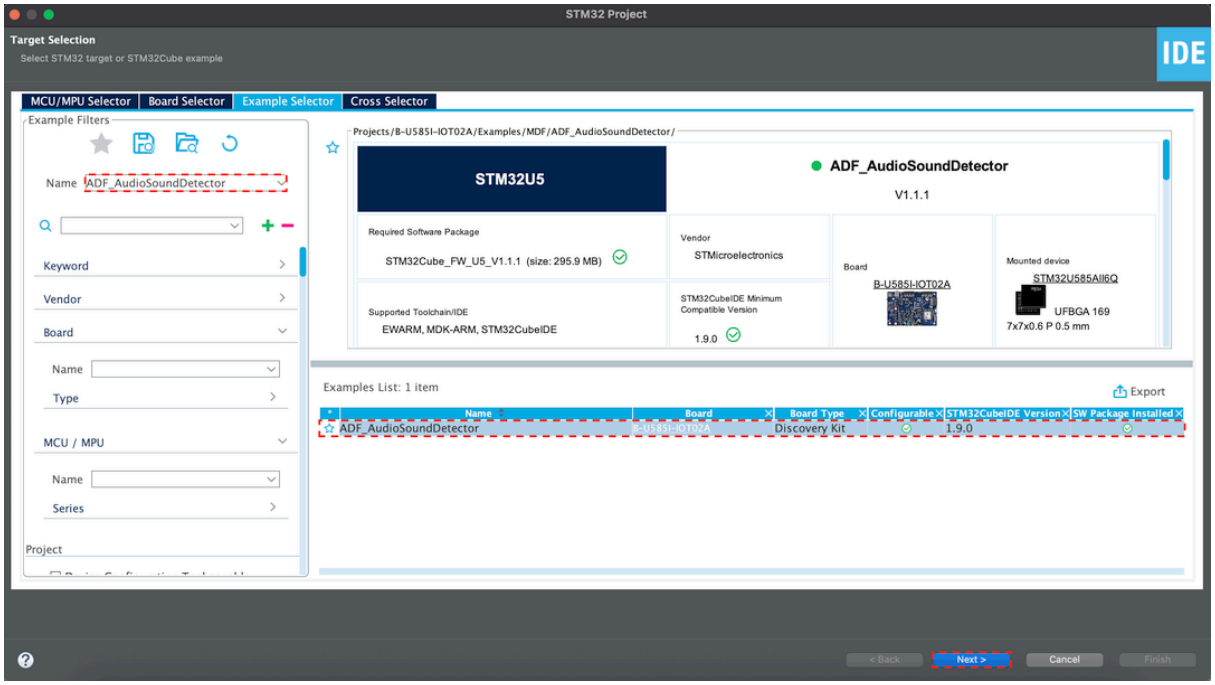
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-123: Target selection



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

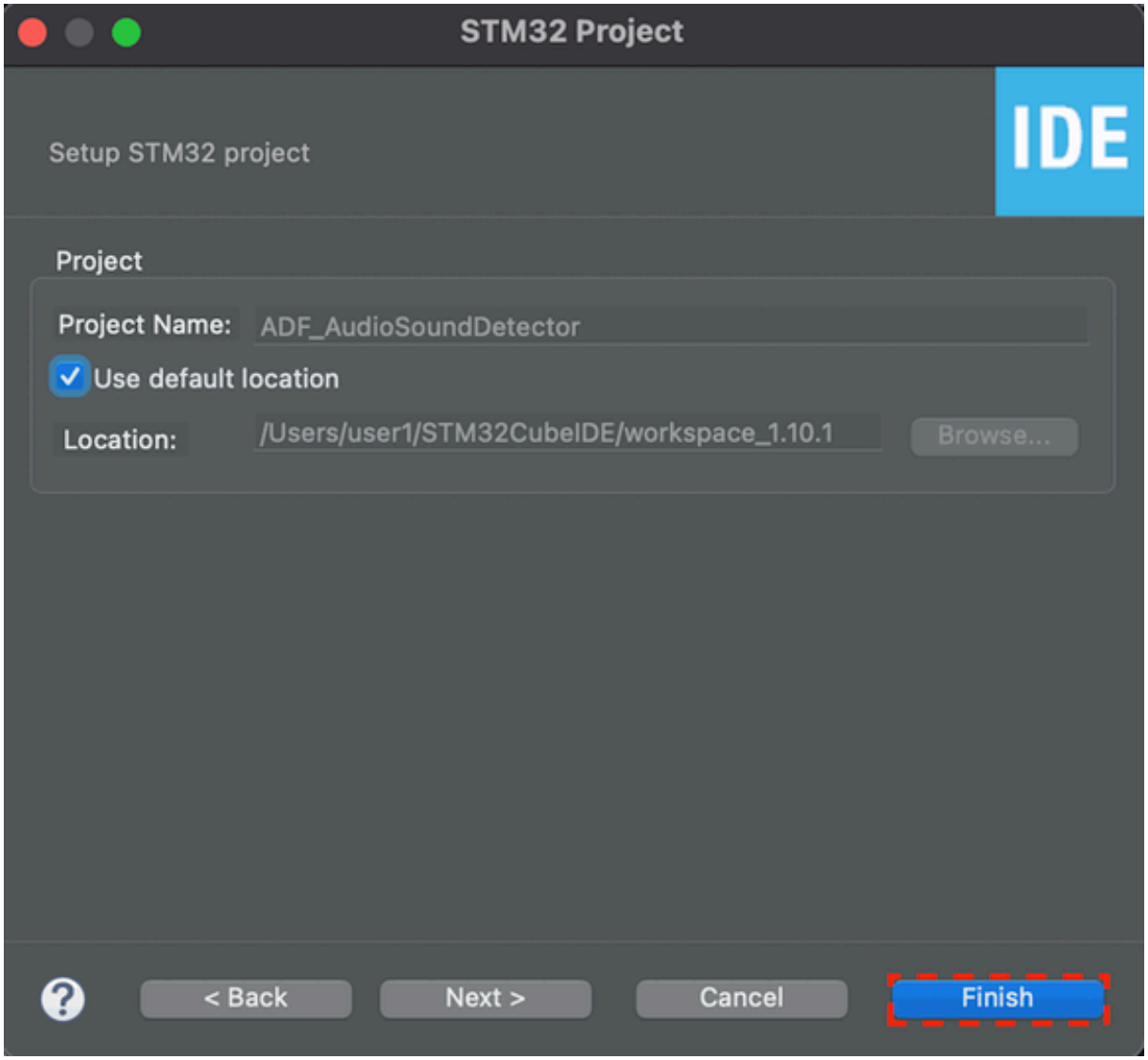
Figure 3-124: ADF_AudioSoundDetector example



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7. Click Finish to create the project.

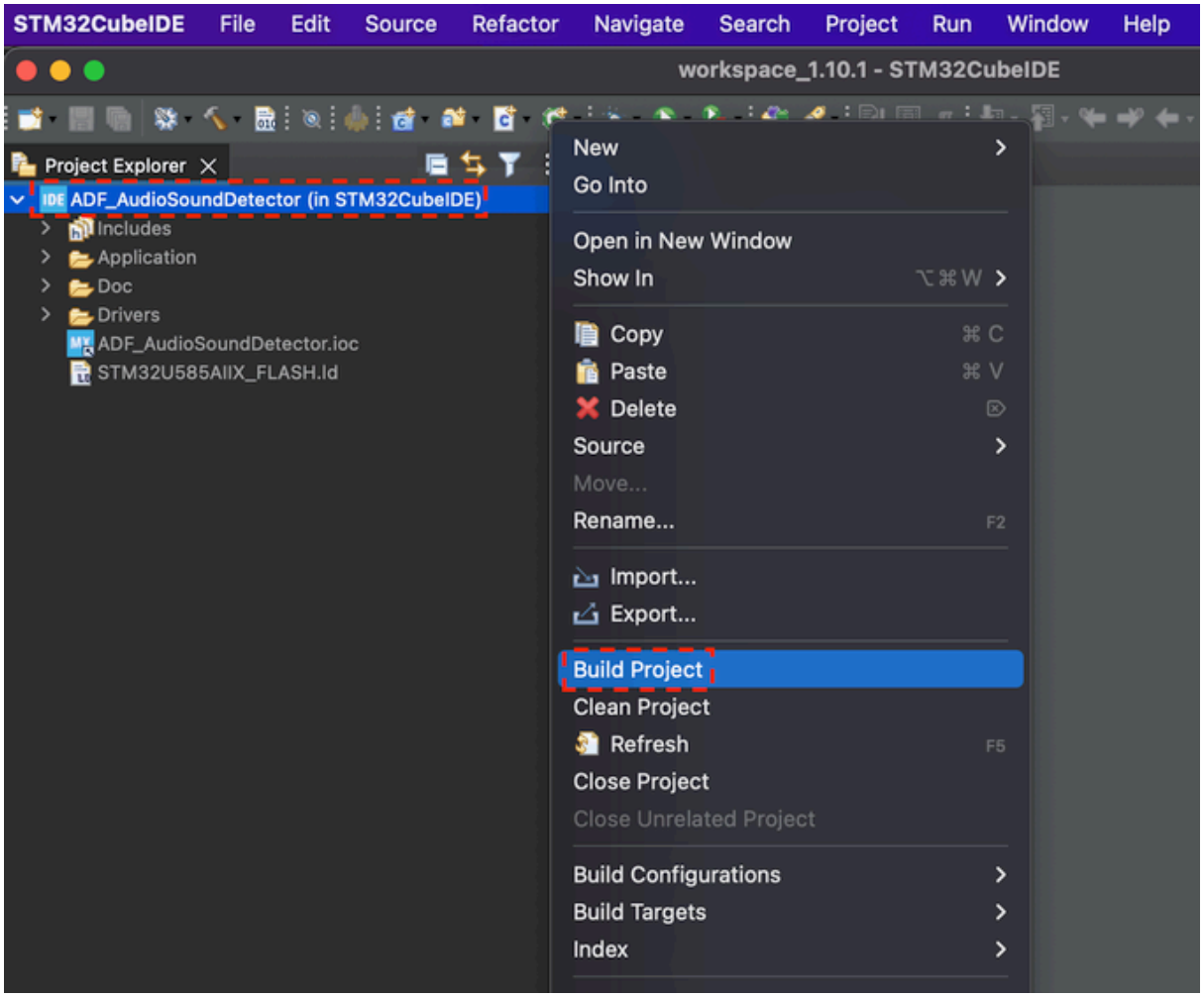
Figure 3-125: Finish project setup



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

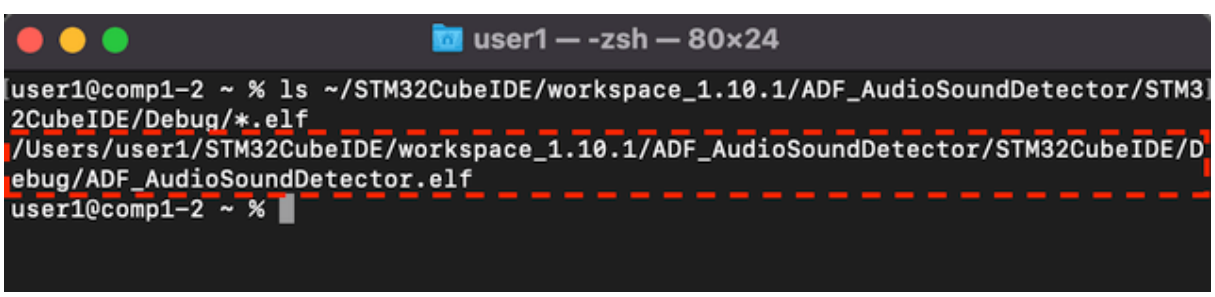
Figure 3-126: Build project



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 3-127: .elf File



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

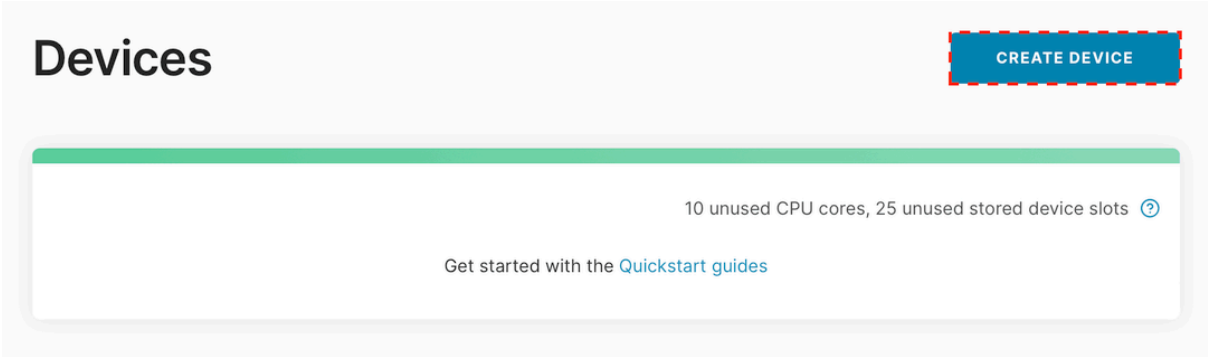
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Boot the Custom Firmware and Detect Audio

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

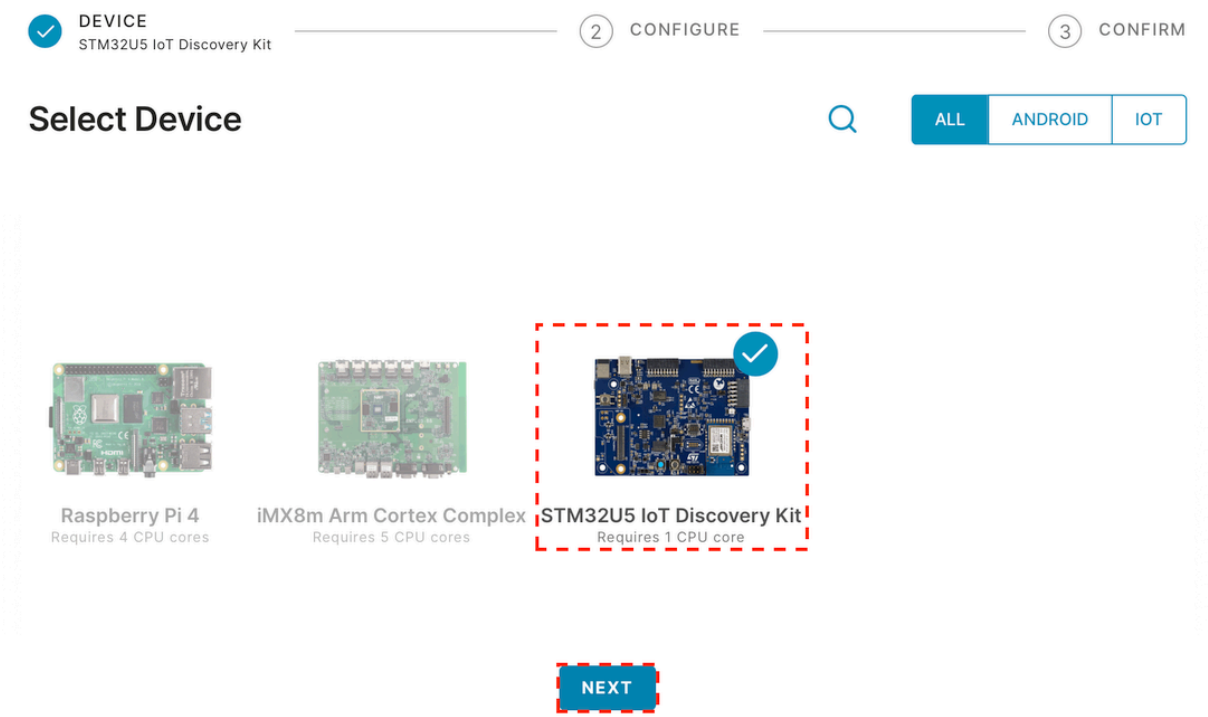
1. Log in to the AVH platform and click **CREATE DEVICE**.

   Figure 3-128: Create device



2. Click on the STM32U5 device.

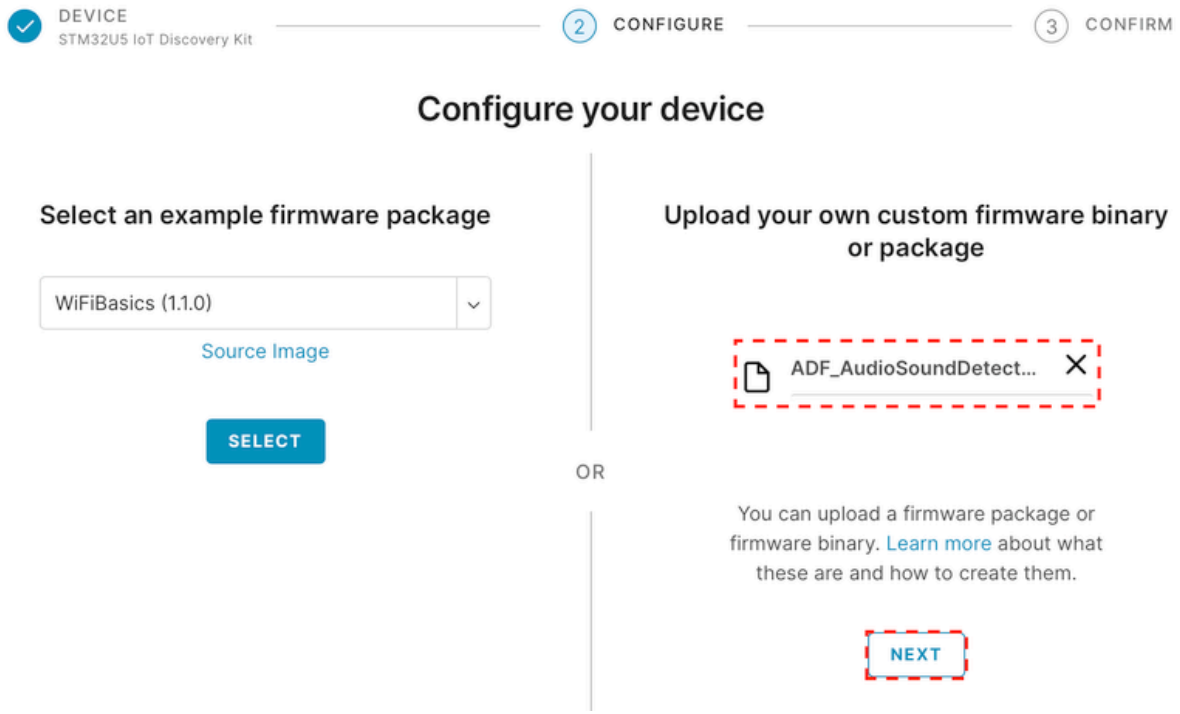
   Figure 3-129: Select device



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

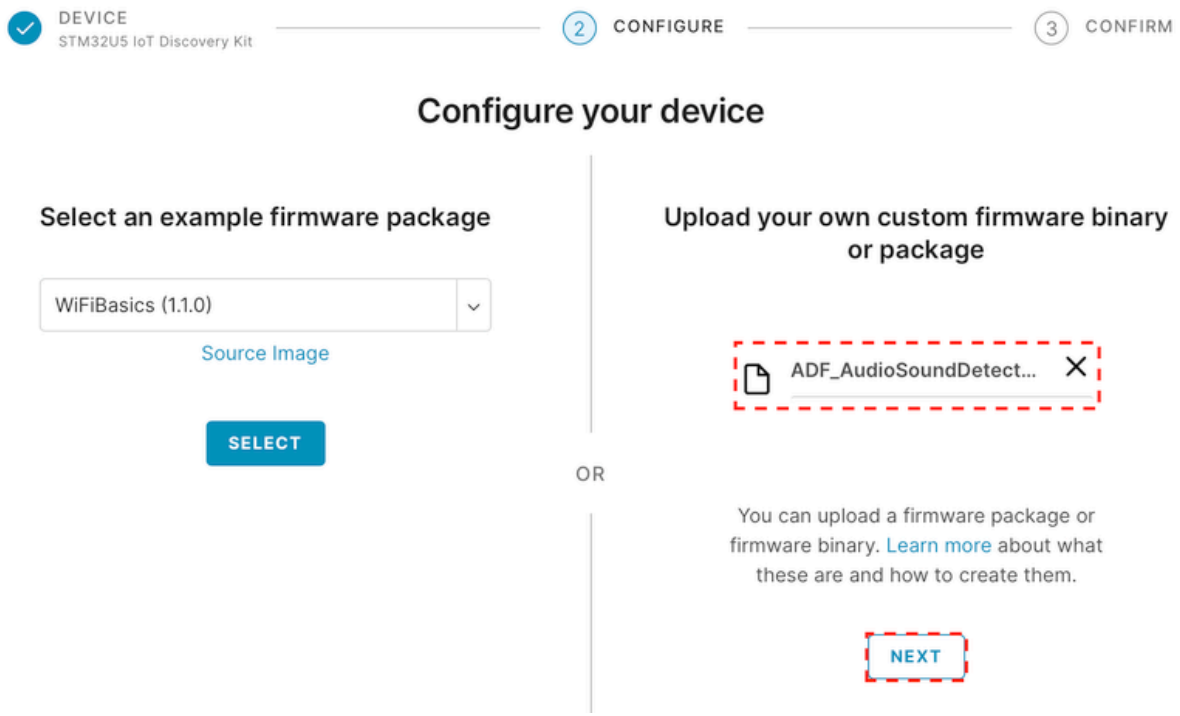
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-130: Upload firmware



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

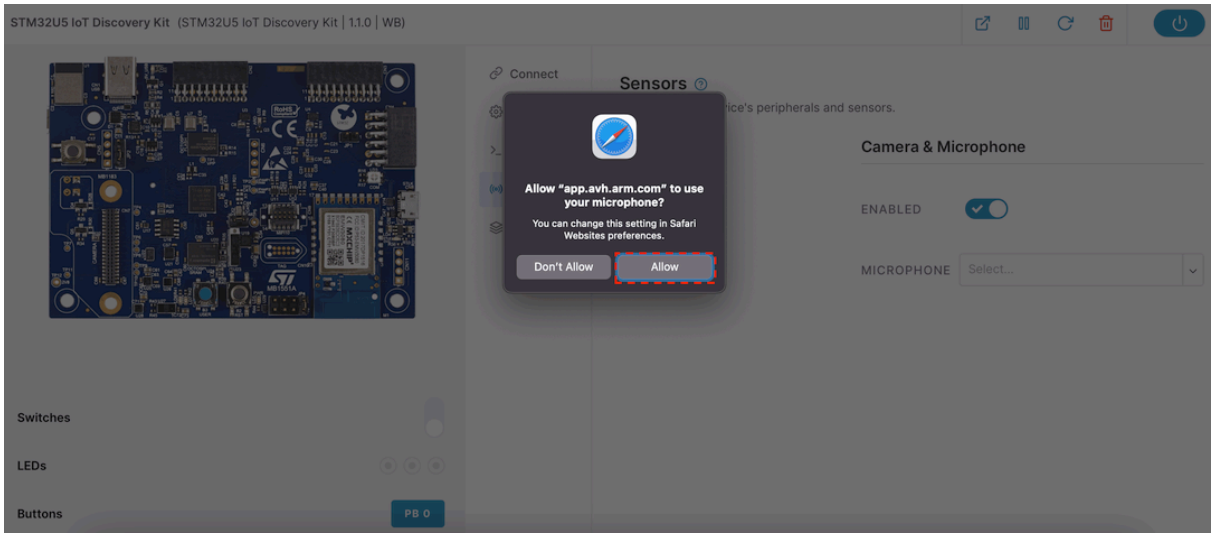
Figure 3-131: Enable microphone



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

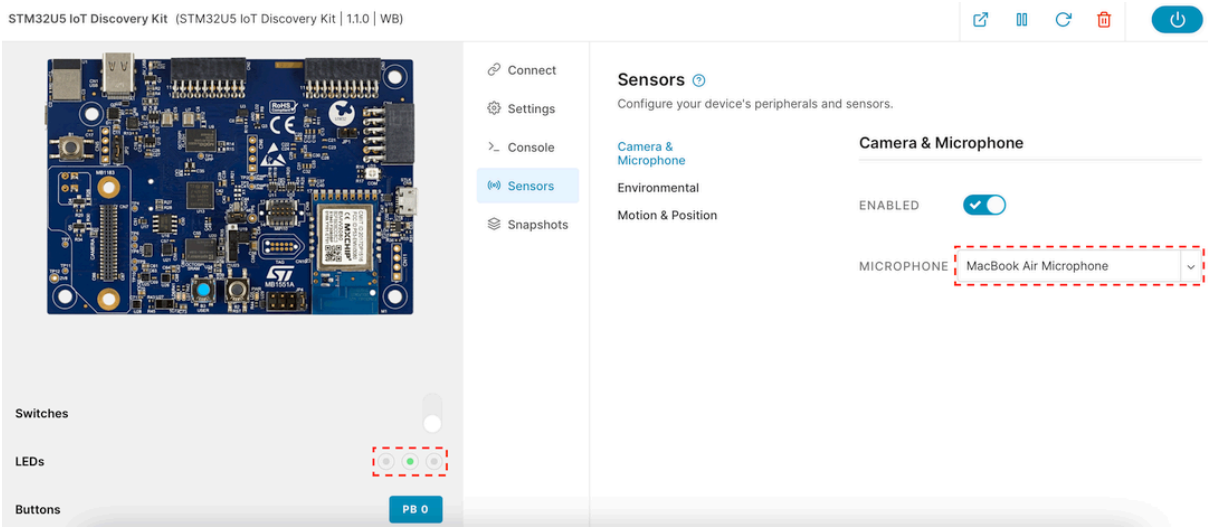
5. Confirm on your browser that you would like to share the microphone.

Figure 3-132: 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 3-133: Audio detection



3.5 Arm Smart Vision Configuration Kit

Arm Virtual Hardware (AVH) provides a model for the Arm Smart Vision Configuration Kit subsystem.

This page will be extended with more details when additional information about the Kit becomes publically available.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Stock Firmware for Arm Smart Vision Configuration Kit

A ready-to-use firmware package (stock firmware) is available to run on a virtual Arm Smart Vision Configuration Kit. This is a zip archive prepared according to the AVH Linux firmware package format explained in Linux Firmware Packages. Following stock firmware package is available:

- **5.10.78- yocto (1.0.0)** firmware package.

When creating a new virtual board, on the **Configure your device** screen in the drop-down menu **Select an example firmware** you can choose the target stock firmware that the virtual board instance shall start with.

3.6 Arm Fixed Virtual Platforms

In addition to Corellium models such as Raspberry Pi 4 and i.MX 8M Arm Cortex Complex, Arm Virtual Hardware (AVH) at avh.arm.com also provides support for Arm Fixed Virtual Platforms (FVPs), that can be accessed and controlled using the same AVH APIs.

The AVH FVPs simulate Arm Cortex-M CPUs and Subsystems. The device names have “fvp” at the end, and also have own FVP category tab. Following FVP variants are available in AVH:

- Corstone-300
- Corstone-310
- Cortex-M0
- Cortex-M0+
- Cortex-M3
- Cortex-M4
- Cortex-M7
- Cortex-M23
- Cortex-M33
- Cortex-M55
- Cortex-M85

This chapter guides through usage specific to the FVP targets.

- **Getting started:**
  - Quickstart for Arm FVPs provides a simple step-by-step guide on setting up an instance with an Arm FVP target.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.6.1 Quickstart for Arm FVPs

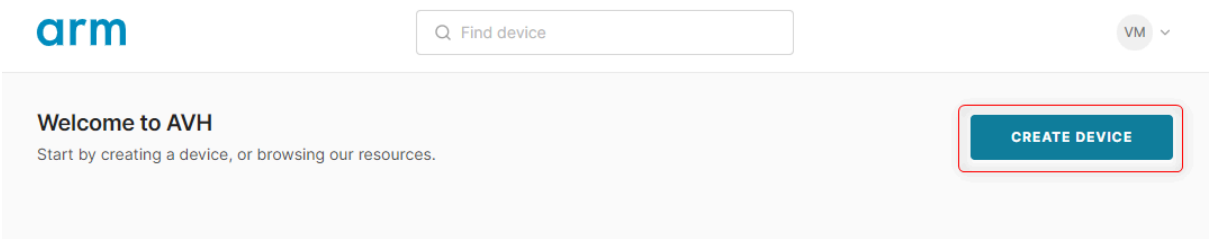
This page guides you through the steps for creating and using an instance with Arm Fixed Virtual Platforms.

Create your FVP device

To create your device, do the following:

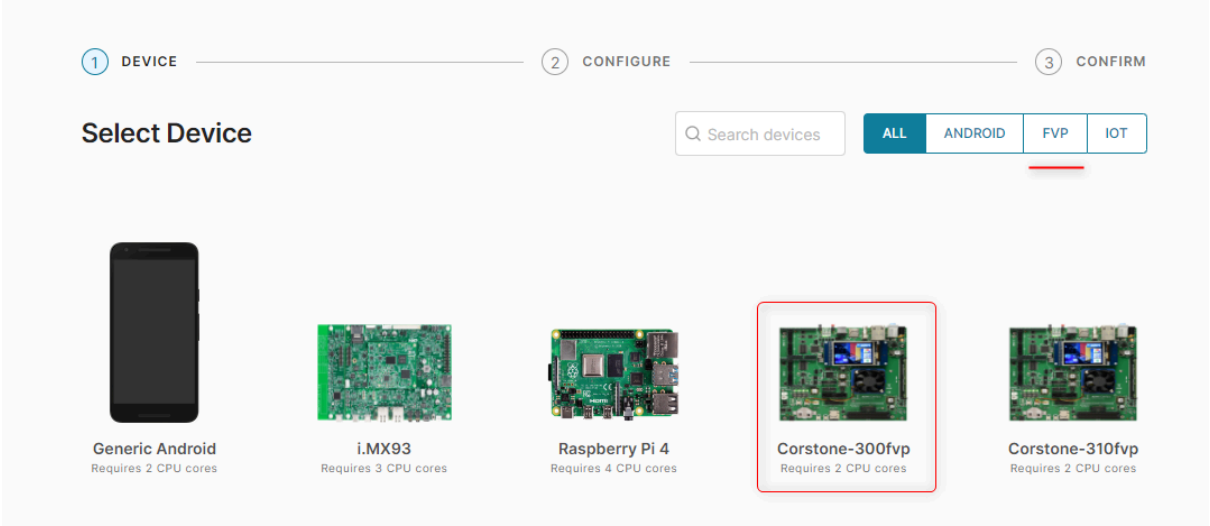
1. Log in to AVH with your Arm account at https://app.avh.arm.com. If you do not have an Arm account you can register here.
2. After login you should land on **Devices** page. There click **CREATE DEVICE**:

   Figure 3-134: Start with device creation



3. On the **Select Device** view click on any device with the name ending with "fvp". In our example we choose **Corstone-300fvp**.

   Figure 3-135: Select Corstone-300 FVP



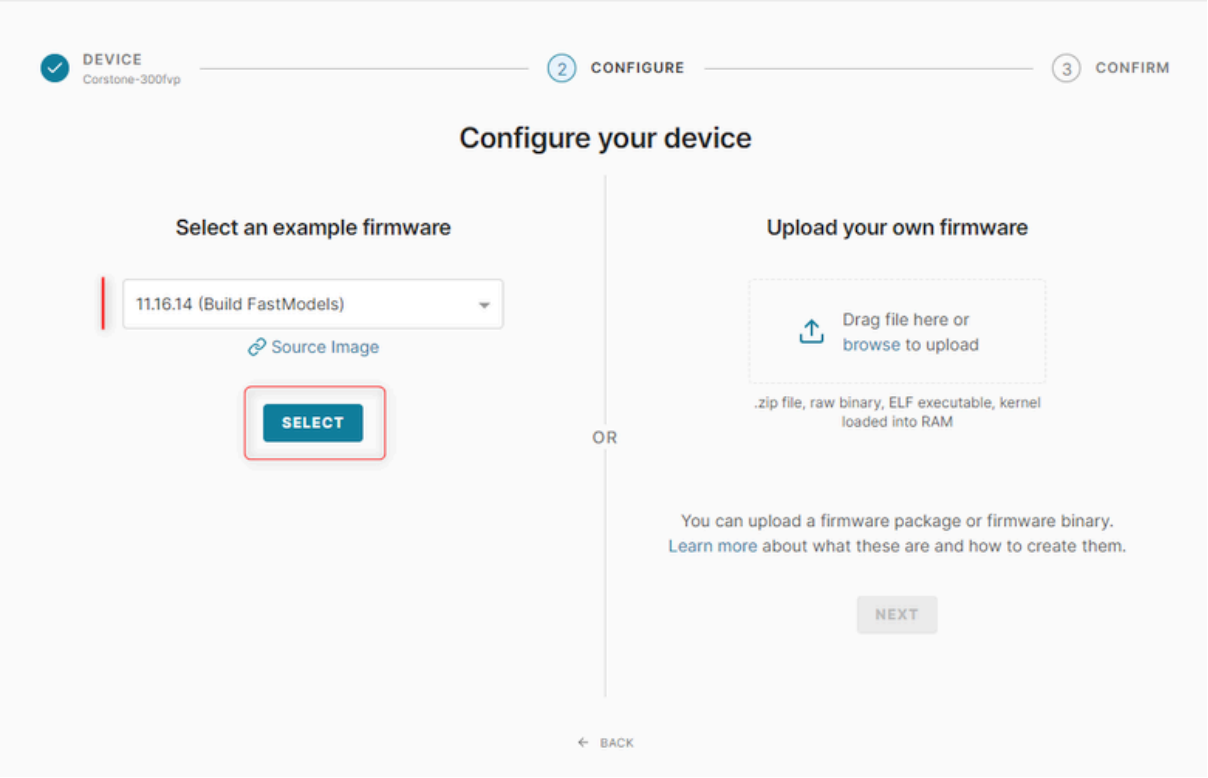
Note

You can narrow down the list of displayed boards by selecting the category tab on the top-right side between **| ALL | ANDROID | FVP | IOT**. The FVP models can be found under FVP and ALL tabs.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4. On the page **Configure your device**, choose the default firmware and click **SELECT**. Do not use this screen to upload your FVP application. FVP targets are started on a virtual machine with Linux OS that is used as the stock firmware image. Uploading new firmware will overwrite that image and corrupt the simulation.

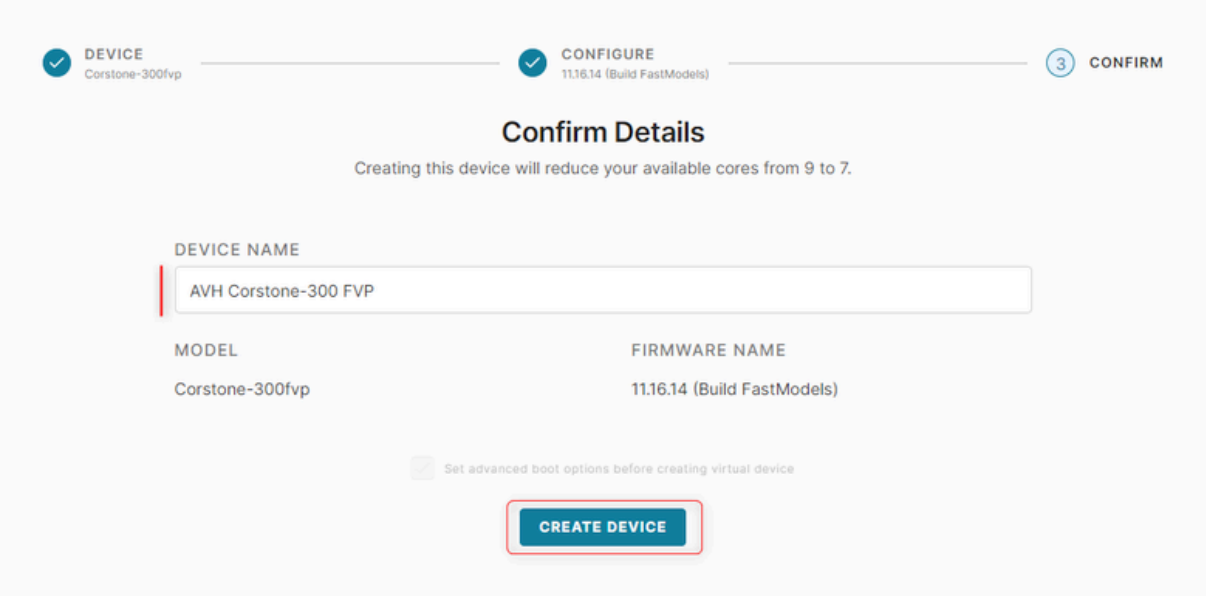
Figure 3-136: Keep default stock firmware



5. On the **Confirm Details** screen you can specify the device name and verify your selections. Click **CREATE DEVICE**.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-137: Confirmation screen



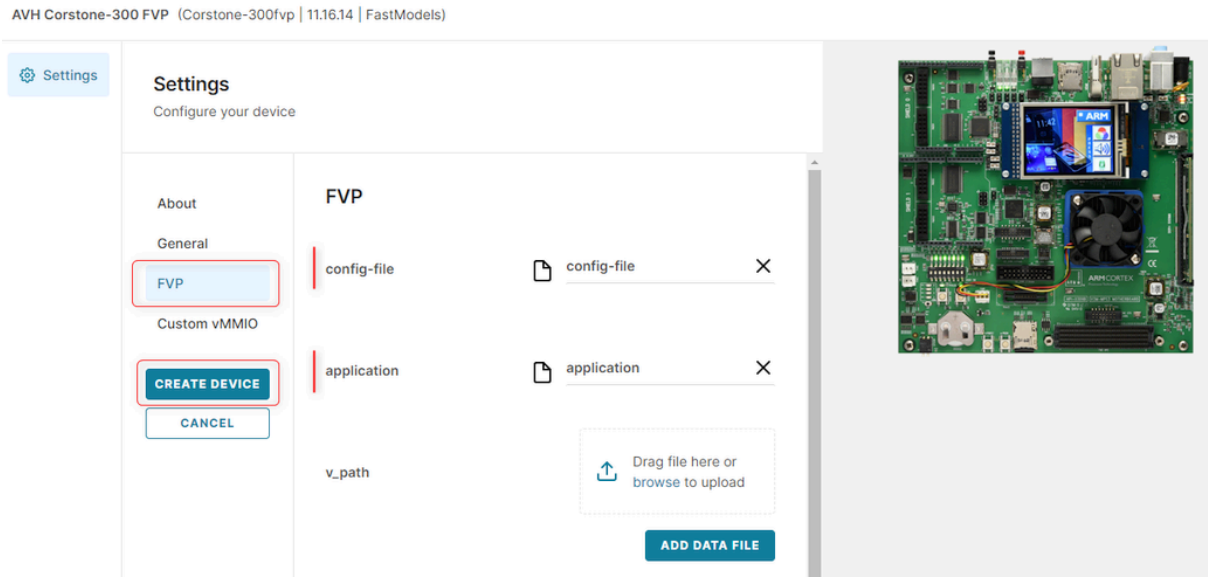
6. In **Settings** navigate to the **FVP** subsection.
7. Upload your application binary in `axf` format for the target FVP. We will use a simple Blinky example from the Corstone-300 CMSIS-Pack.

   You can use this same window to upload a configuration file (`config-file`) or upload the content used for Virtual Interfaces (`v_path`). For more information on using these options, see the AVH FVP GitHub page.

   Confirm that the files have been uploaded successfully and click **CREATE DEVICE**.

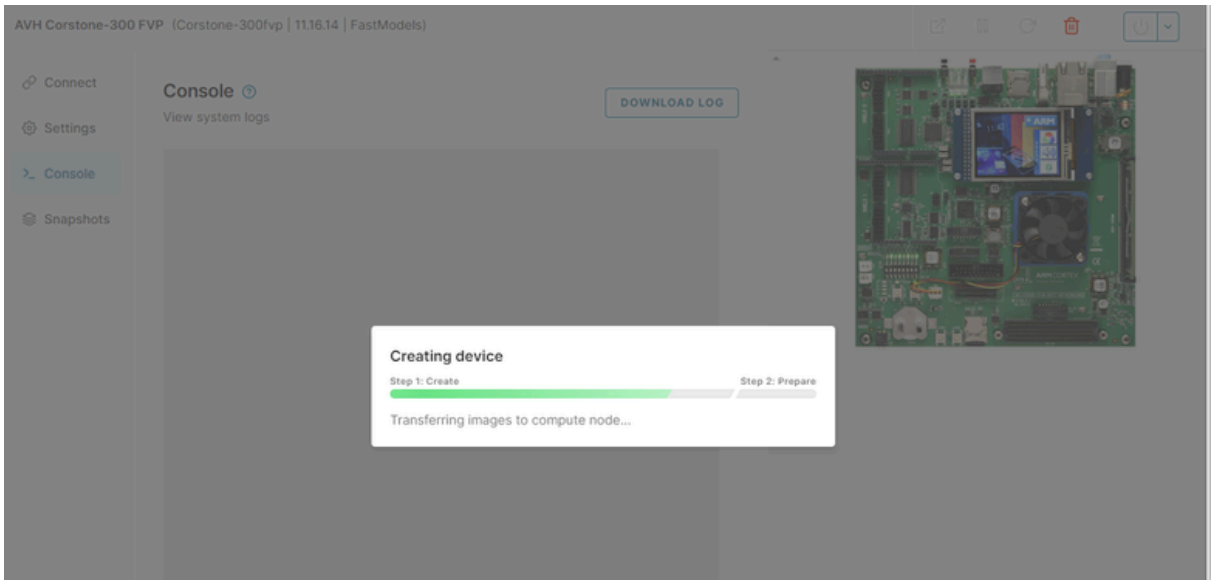
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-138: Start with device creation



8. Wait a few minutes for the virtual device to become ready.

Figure 3-139: Creating device



9. Your FVP device is now ready to go!

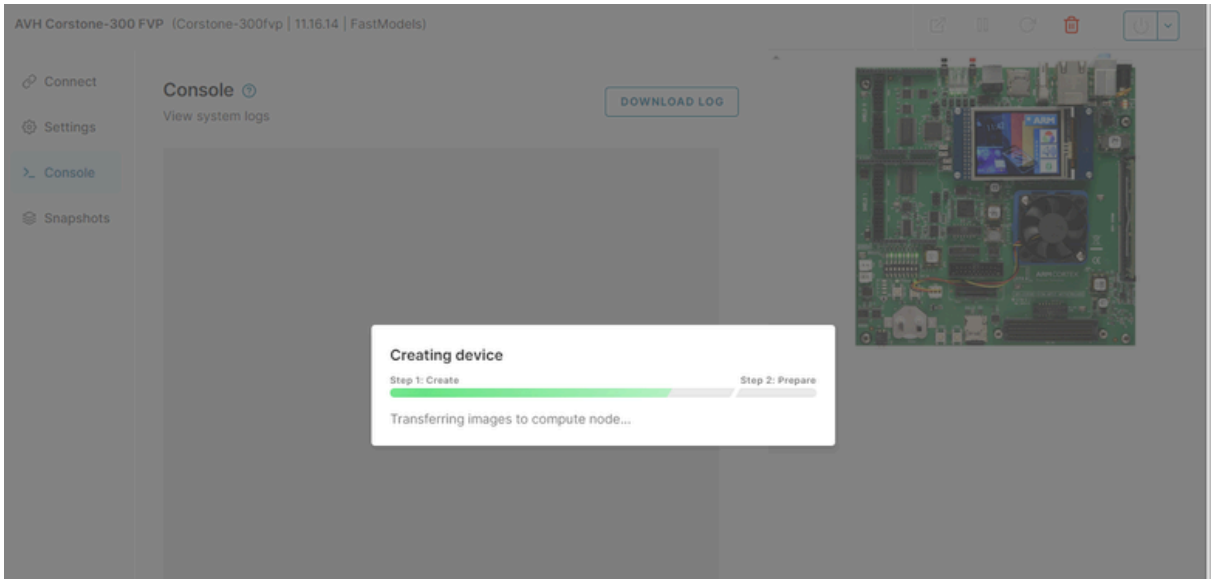
Running the example

The application uploaded in the previous step gets executed on the target FVP.

1. Click **Console** tab in the device's menu, if it is not open by default. You shall see a serial console output of the FVP.

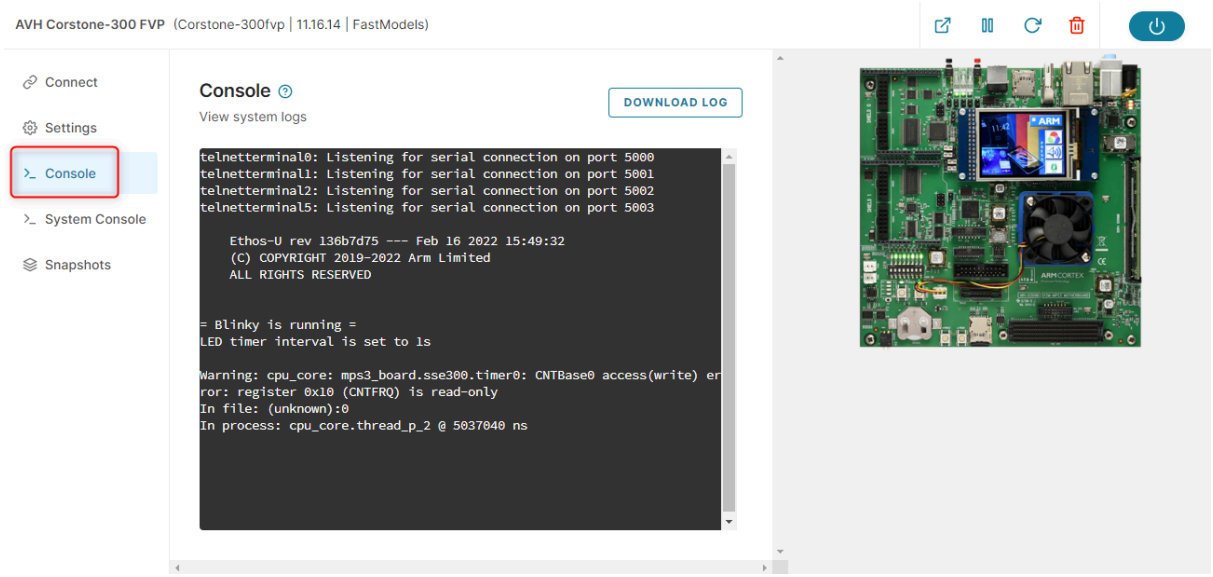
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-140: FVP Console view



2. Click **System Console** tab in device's menu. This gives you access to the host Linux OS where FVP is executed. Log in with the username `ubuntu` and passkey `password`. You can execute the FVP from command line, as well as upload new firmware binaries and download log files directly from this environment.

Figure 3-141: FVP Console view



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.7 Generic Android

Arm Virtual Hardware provides a model of a generic smartphone device with Android OS. It can assist you in building and testing your IoT applications on other virtual platforms. Following sections explain some of the capabilities available on a generic Android device:

- Android WiFi
- Android Debug Bridge

Also refer to section Connect to BLE on Raspberry Pi 4 that shows how to connect a virtual Raspberry Pi 4 to a Generic Android device using virtual Bluetooth Low Energy (BLE) interface.

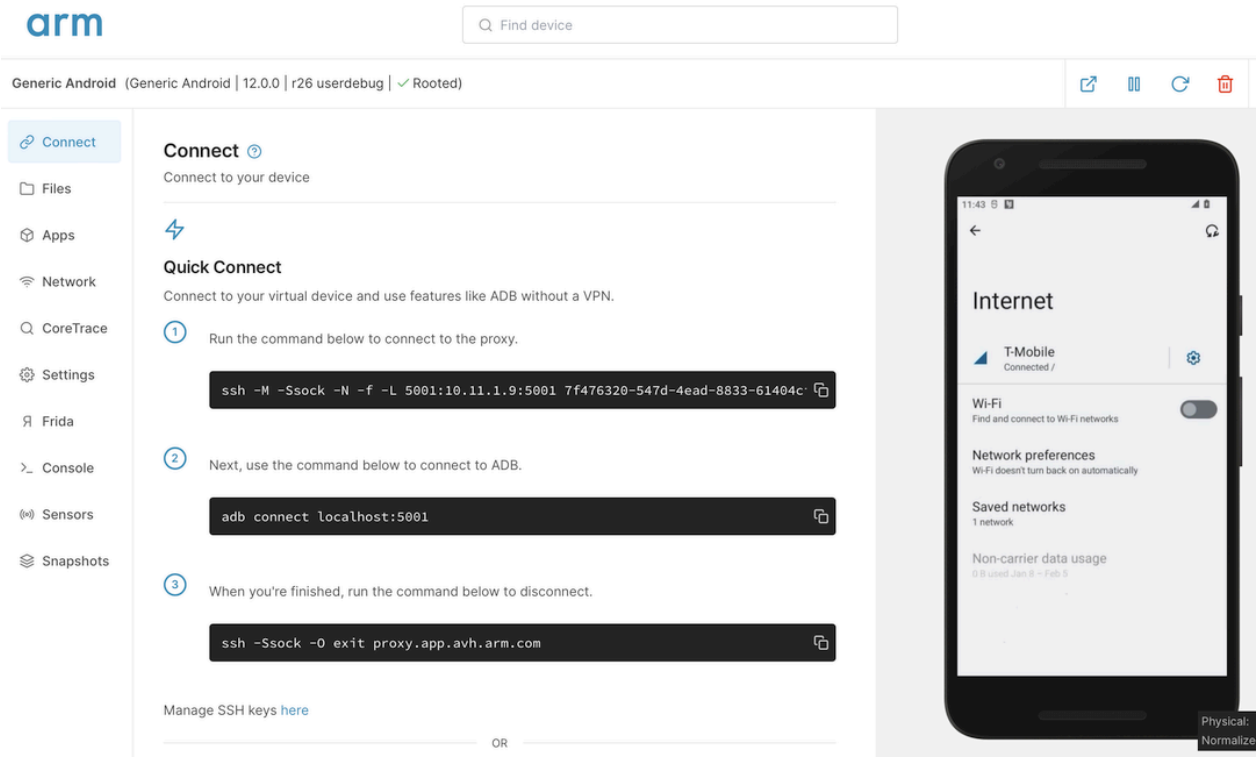
3.7.1 Android WiFi

Virtual Generic Android devices automatically connect to the AVH WiFi network upon creation. However, if you deliberately disconnect a device and need to reconnect, you can easily do so.

On an Android device, you can disconnect and reconnect to the network by toggling the Mobile Data option in Settings -> Network and Internet -> Internet.

Although Android devices include virtual cellular hardware, but they don't include virtual Wi-Fi hardware. This means that you cannot connect virtual Android devices to Wi-Fi.

Figure 3-142: Android WiFi interface



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.7.2 Android Debug Bridge

You can control your virtual Generic Android devices using the Android Debug Bridge (ADB) command-line tool. ADB allows greater control to install and debug applications.

You can read more about ADB on the Android Developer Studio.

Install Android Debug Bridge

The Android Debug Bridge tool is available as part of the `android-platform-tools` SDK.

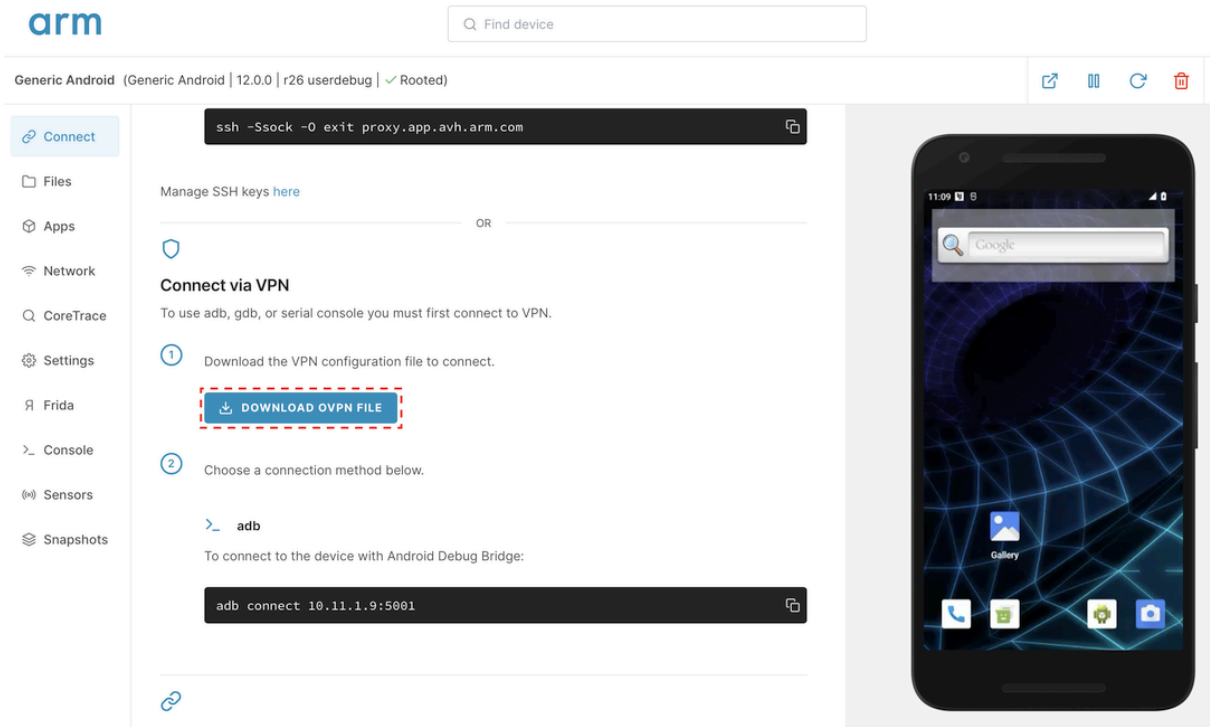
On macOS, install ADB in the Terminal using Brew:

```
brew install android-platform-tools
```

Initiate a Connection to Your Virtual Android Device

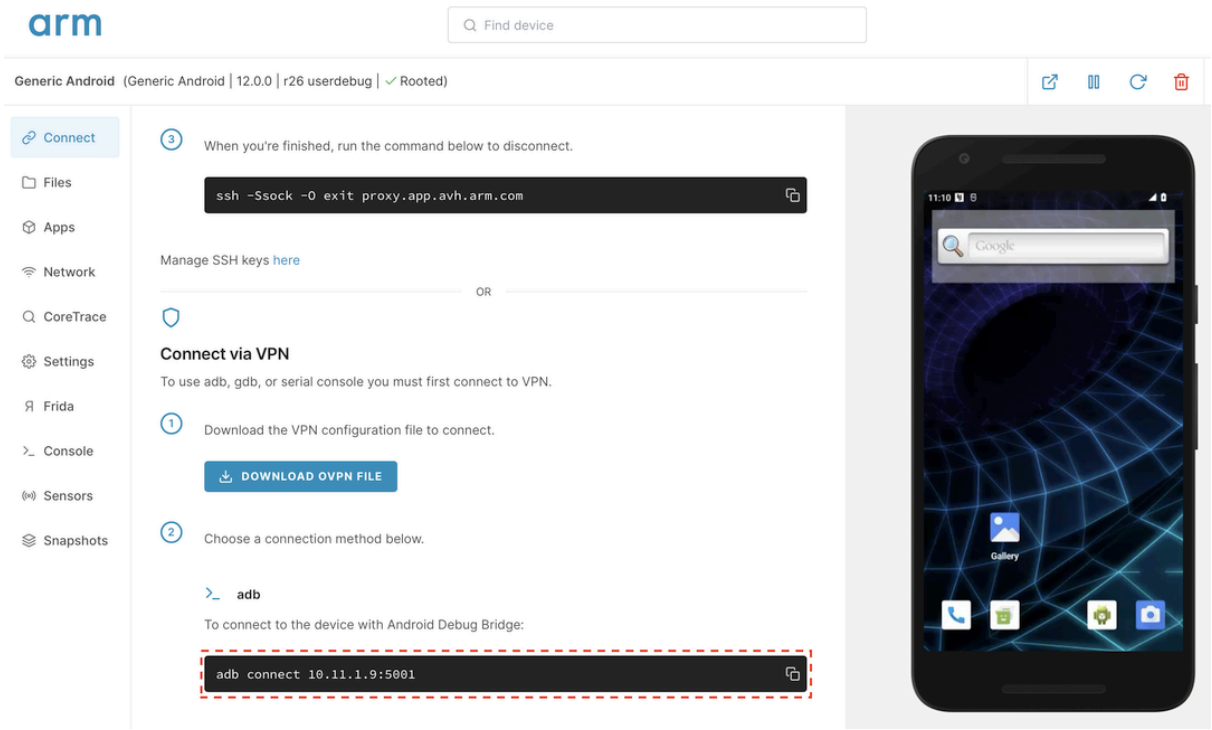
You can connect to your Android device with or without a VPN, below shows the VPN connection method.

1. Create and boot up a virtual Android device.
2. Download the .ovpn file and initiate a VPN connection.

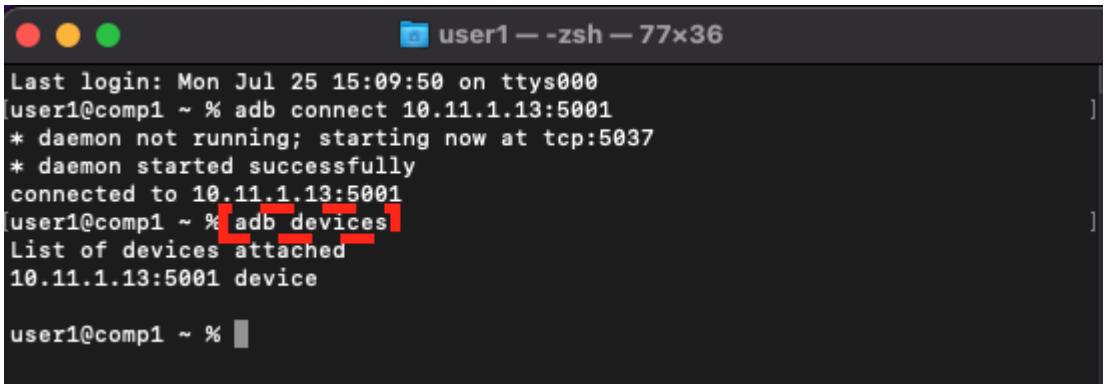


Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. Copy the `adb connect ...` command and run it in Terminal or Powershell.



4. Verify your connection by running adb devices.



Once connected, you can use adb to transfer files, run remote shell commands, install and uninstall apps, view device logs, and more.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4. Device Controls

This chapter explains various controls available for your virtual devices.

- Action Buttons
- Console
- Sensors
  - Environmental
  - Motion and Position
  - Camera and Microphone
  - Location
  - Battery
- Snapshots

4.1 Console

The **Console** tab provides convenient web access to the console of your virtual device. You can use the Console to quickly see logs and run commands without need to establish remote connections. Use the **DOWNLOAD LOG** button to save the current console output as a text file.

Unless different is specified, for devices running the Linux OS it corresponds to the Linux console, and for devices with bare-metal applications it corresponds to the serial console.

To learn how to access the console remotely see page Connect to Device Console.

4.2 Action Buttons

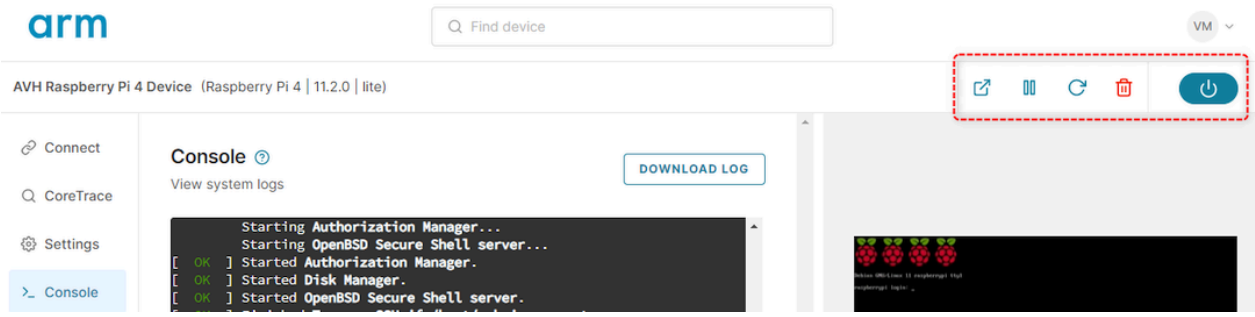
This tutorial will demonstrate how to use your device's action buttons.

- Power On and Off
- Pause
- Restart
- Delete

Action buttons can be found at the top right when you are in a Device.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-1: Power and action buttons



Power On and Off

Powering the virtual device off is like powering off a physical device. All applications are shut down, and information about the current state of the device is not preserved. When a device is in the Off state, it becomes Stored, and it no longer occupies CPU cores. When you are not using the devices it is best practice to turn them off.

Figure 4-2: Power On / Off button

Pause

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

Figure 4-3: Pause button

Restart

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

Figure 4-4: Restart button

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Delete

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

Figure 4-5: Delete button

4.3 Sensors

You can test devices in real world conditions by configuring a wide variety of sensors and controls:

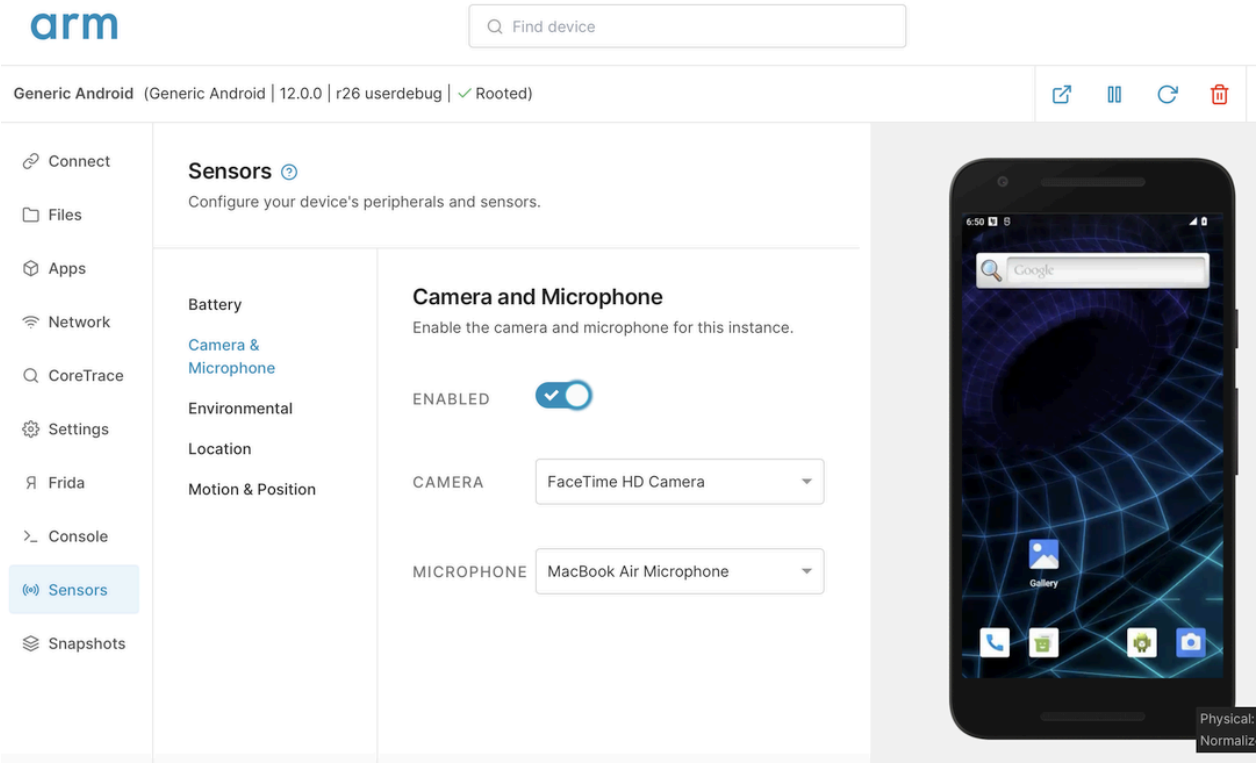
- Environmental tab provides settings for the environmental sensors such as temperature, light, humidity and barometric pressure.
- Motion and Position allows you to simulate your device in motion as it moves around the virtual world.
- Camera and Microphone provides configuration options for connecting hardware camera and microphone to your virtual device
- Location
- Battery

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.3.1 Camera and Microphone

The first thing you will need to do is enable your camera. Start by going to tab **Sensors** > **Camera & Microphone** > **Enabled** > **toggle ON** (toggle should turn blue).

Figure 4-6: Enable camera and microphone on Generic Android



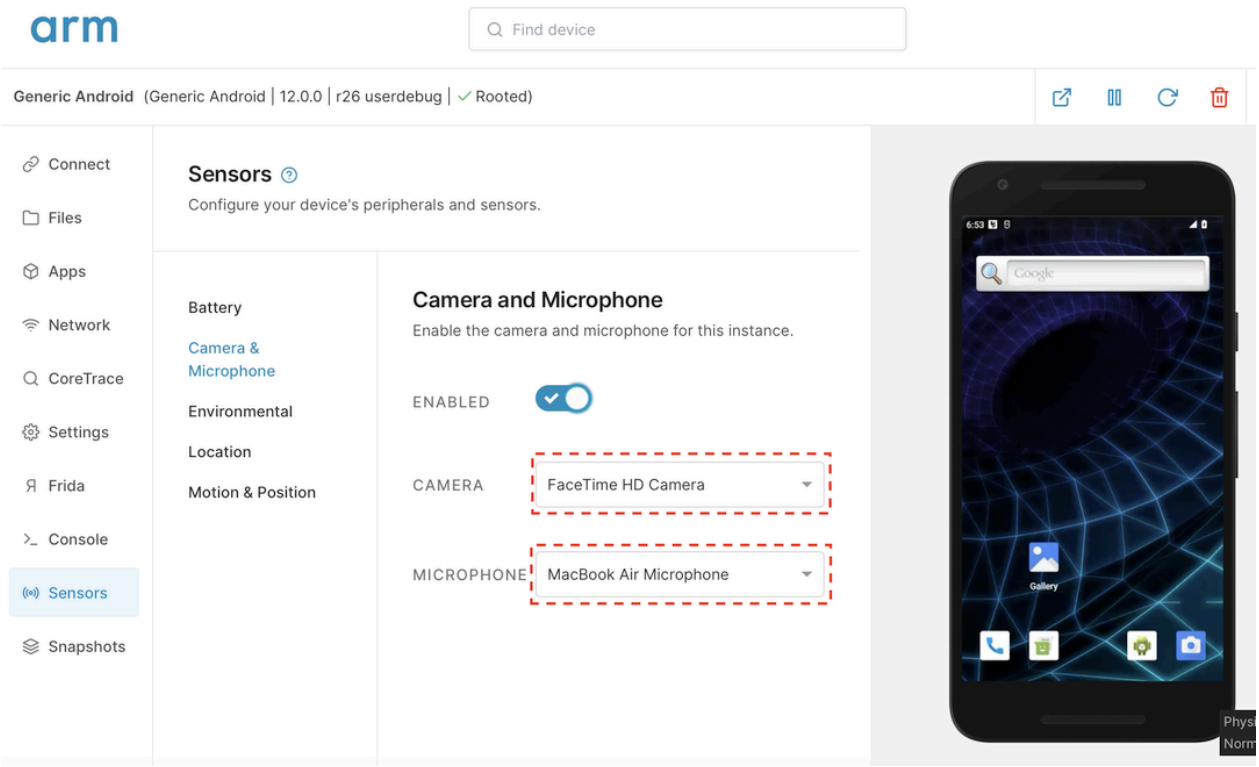
Your web browser may ask for permission to use your Camera and your Microphone. Click Allow for both.

Camera & Microphone Selection

You can choose which camera or microphone you would like to use for your Virtual Machine

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-7: Selecting camera and microphone on Generic Android



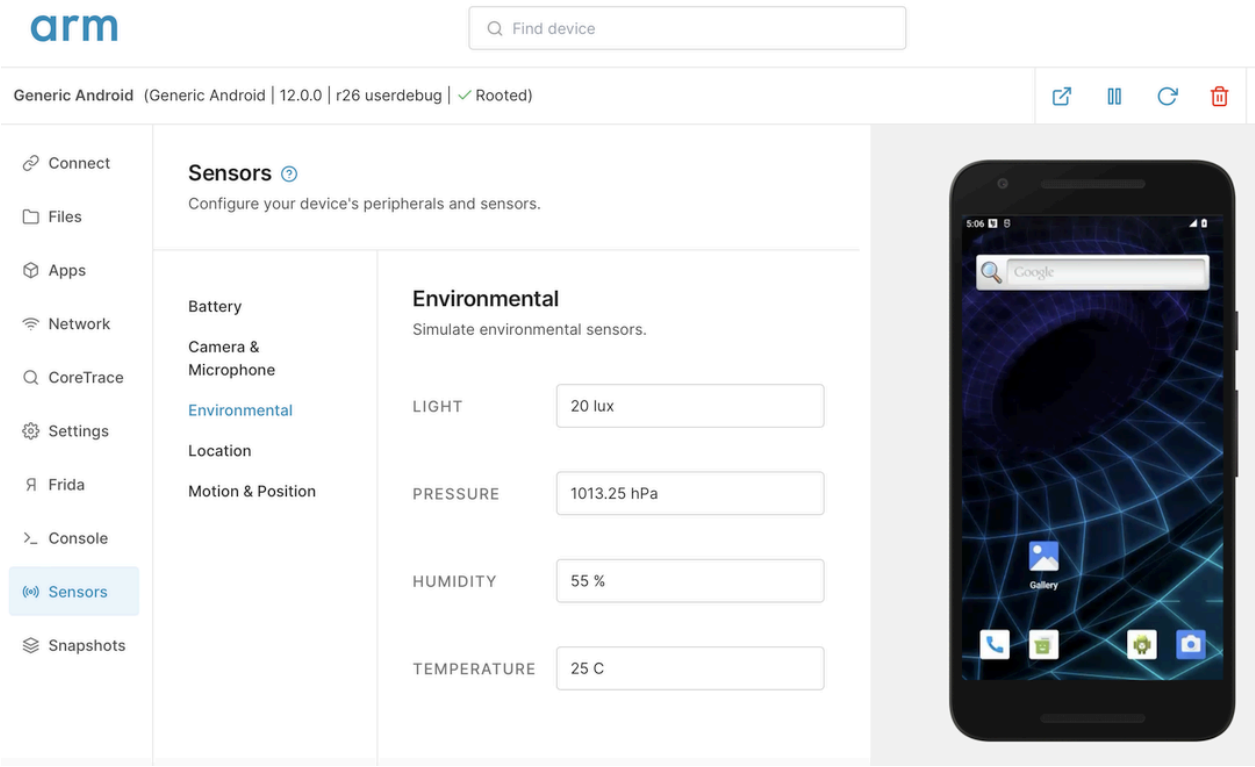
4.3.2 Environmental

Virtual devices may contain environmental sensors such as light, humidity, barometric pressure, and temperature. See Supported Platforms for descriptions of hardware components available on the target virtual model.

If corresponding sensor is present, you can adjust its values by going to tab **Sensors** > **Environmental**. Below is an example of environmental sensors on a Generic Android device.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-8: Environmental sensors on Android device



Light

Configure the ambient light level (in lux)

Our light feature is measured by Lux or lx. One unit of measurement is called a Lux, which describes how much light falls on a certain area. This is different from a unit of lumens, which tells you the total amount of light emitted by a light source.

1 LUX = 1 Lumen / m2

Pressure

This option will let you make changes to the altitude data, in regard to Ambient air pressure. Configure the atmospheric pressure (in hectopascals)

Humidity

This option will adjust the ambient relative humidity in per cent (% RH).

Temperature

This option will adjust the device temperature in degrees Celsius.

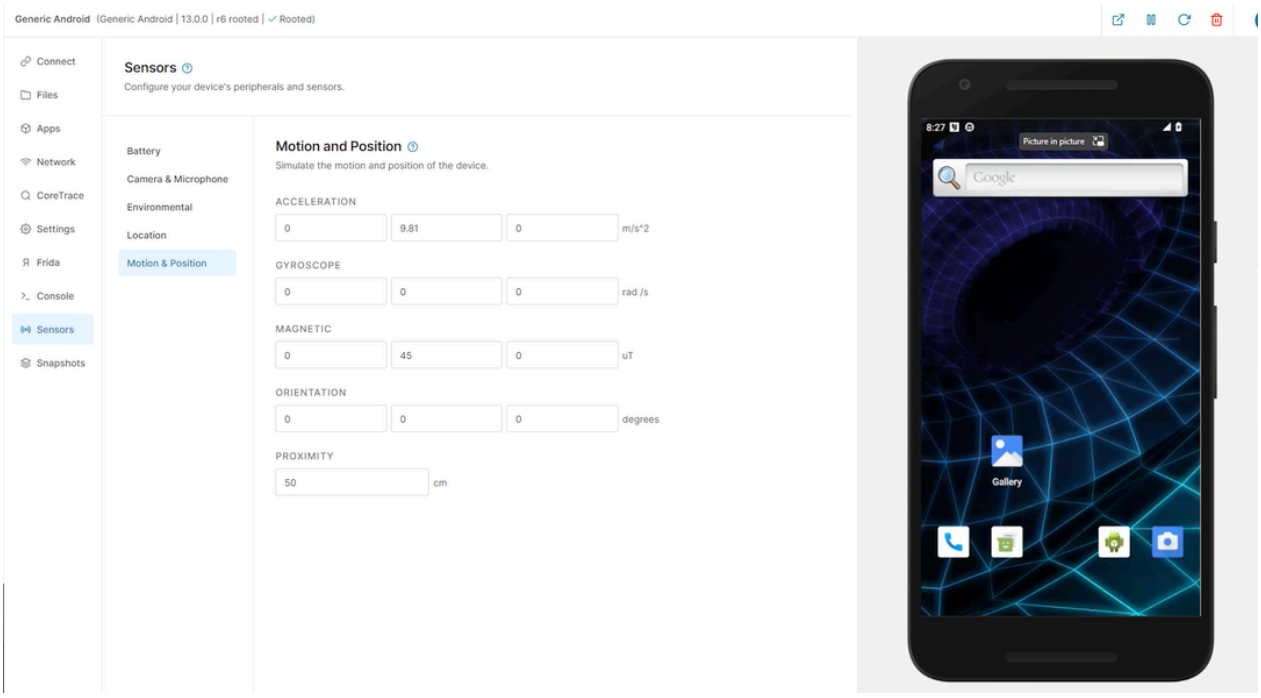
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.3.3 Motion and Position

On your virtual device you can simulate the device motion by configuring values for the following sensors:

- Accelerometer
- Gyroscope
- Magnetic sensor
- Device orientation
- Proximity

Figure 4-9: Motion and Position settings on Android



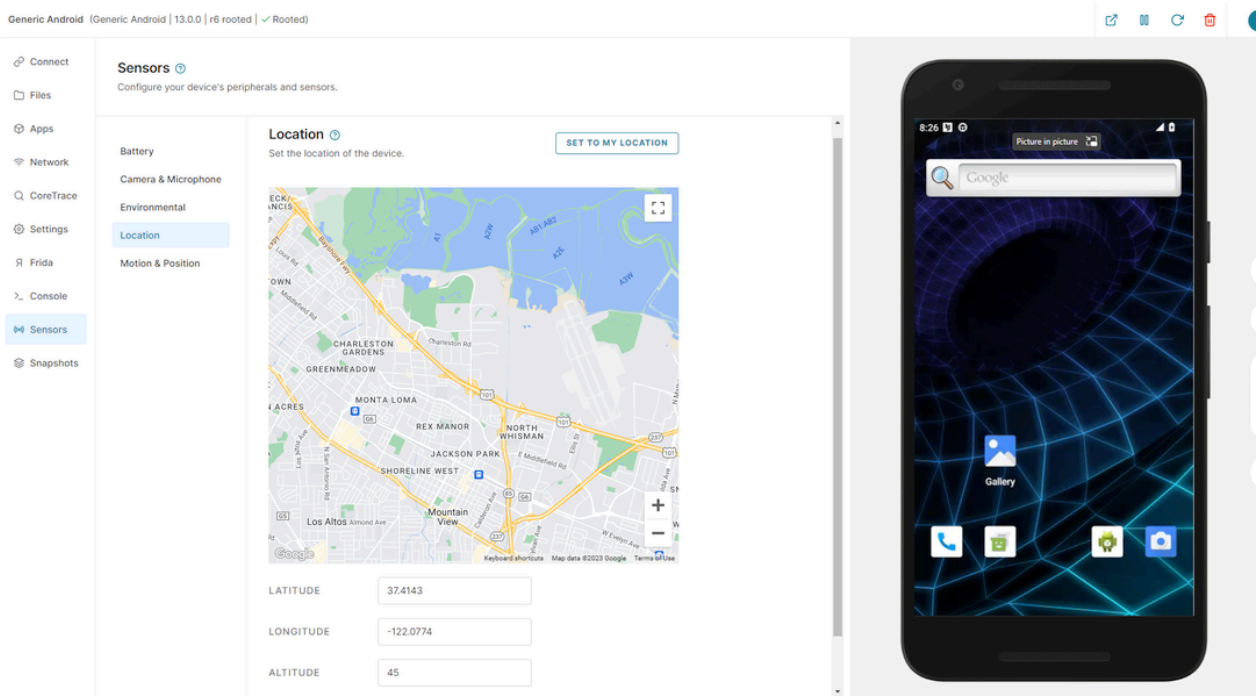
4.3.4 Location

You can specify the geographical location for your virtual Generic Android device.

On your Android device go to **Sensors** > **Location**. This screen will allow you to set the Latitude, Longitude, and Altitude using the provided entry boxes. You can also navigate to a latitude/longitude of your choice by moving the map to the location of interest.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-10: Location settings on Android



Note that location values are provided to the Android system as a GPS hardware change. This means you can read them utilizing the direct permissions android.permission.ACCESS_FINE_LOCATION and android.permission.ACCESS_COARSE_LOCATION.

Directly using location data

Ensure that the application you are trying to test has direct access to the GPS by requesting the correct permissions you would like, either android.permission.ACCESS_FINE_LOCATION or android.permission.ACCESS_COARSE_LOCATION.

A good Android application to test the changes for this is SatSat, which is open-sourced and can be found on F-droid or the GitHub page for the source code. You can also reference a JavaScript example for requesting and accepting the GPS data inside a WebKit component can be tested here and find the source code on the GitHub page.

Using a FusedLocationProvider

The Google FusedLocationProvider API enables functionality such as “Current Location.” Without the FusedLocationProvider enabled, features such as centering on your current location in maps apps will not behave as expected.

To use the FusedLocationProvider, you will first need to install OpenGapps, which will install Google Play Services. Once that’s installed, you will also need to install Google Maps via the Google Play Store or by side-loading. Next, you will need to run Google Maps at least once to activate the feature.

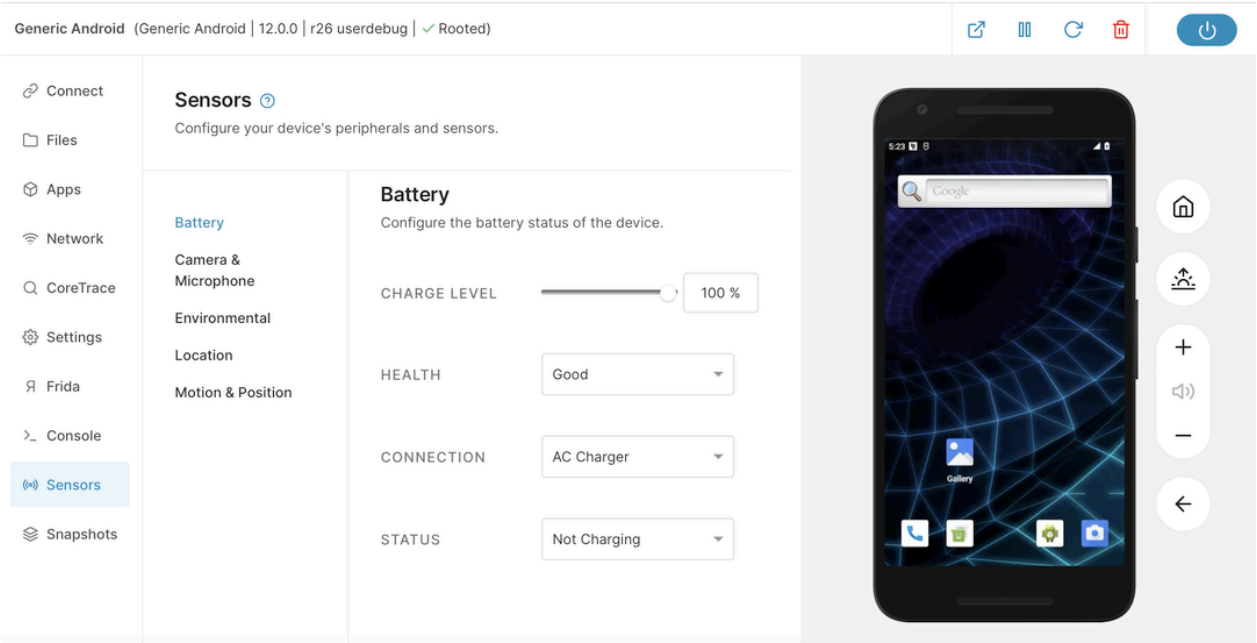
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Google Services handles the FusedLocationProvider API interaction. When an application is subscribed to get location updates, Google Services relies on its services to continually receive periodic locations to update the FusedLocation. This is why you must install and run Google Maps at least once to activate this feature.

4.3.5 Battery

On virtual smartphone devices, such as Generic Android you can find our Battery feature by navigating to tab **Sensors** > **Battery**. It enables you to configure such parameters as battery level, health, charge connection and status and see how your mobile applications will perform on a device running on a low battery. Please take a look at the screenshot below for reference.

Figure 4-11: Battery settings on Android



Charge Level

You can use special Charge Level option as a standardized indicator of electrical force, or voltage, at which the device is set.

Note

Your device may eventually power OFF if you have the charge level set to a low level.

Health

You can adjust the state of your battery's health.

- Good: The battery performance is normal.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- Failure: The battery can power ON the device, but the performance is very poor.
- Dead: The device is not receiving any electrical energy and is powered OFF.
- Overheat: The battery is working extra hard and is likely to rise in temperature.
- Overvolt: The battery has been overcharged which can obstruct the performance of the device.

Connection

You can set whether the device is connected to a charger or not. - AC Charger: the device is connected to a power adapter. - None: the device is not connected to a power adapter.

Status

You can specify the charging status for your Android device as follows: - Charging: the device is receiving and storing electrical energy. - Discharging: the battery is losing voltage or energy. - Not Charging: the device is not receiving and is storing electrical energy.

4.4 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.

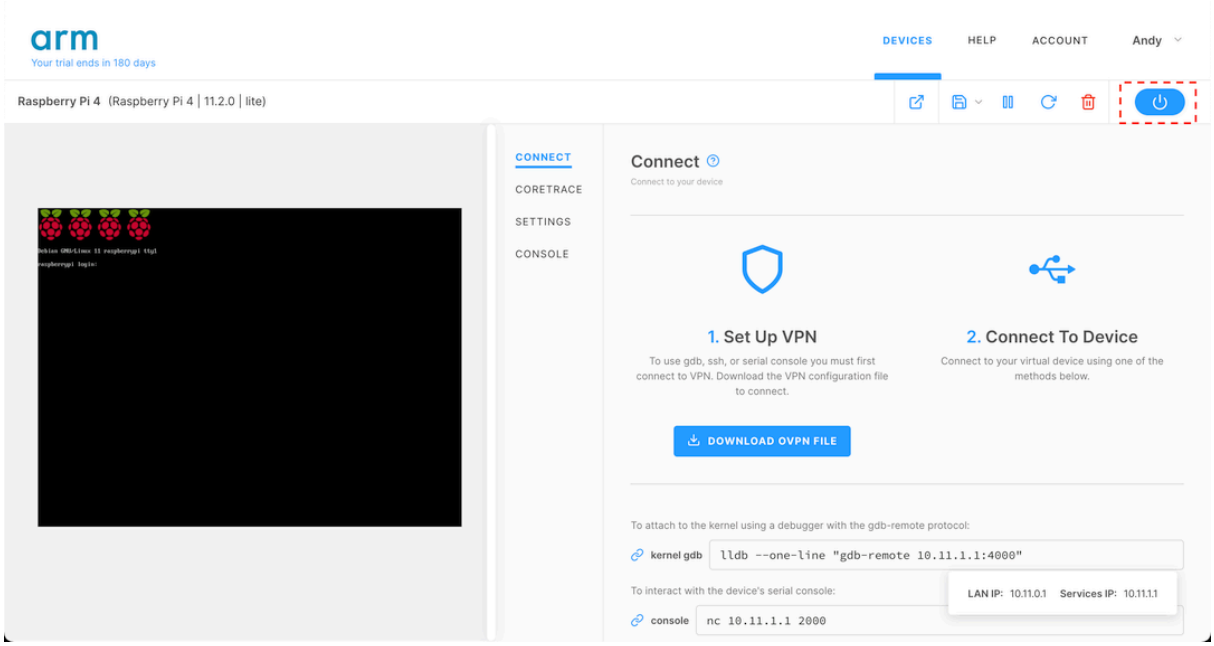
Taking and restoring snapshots

This tutorial will demonstrate Snapshots by creating and restoring them on a virtual Raspberry Pi 4 device.

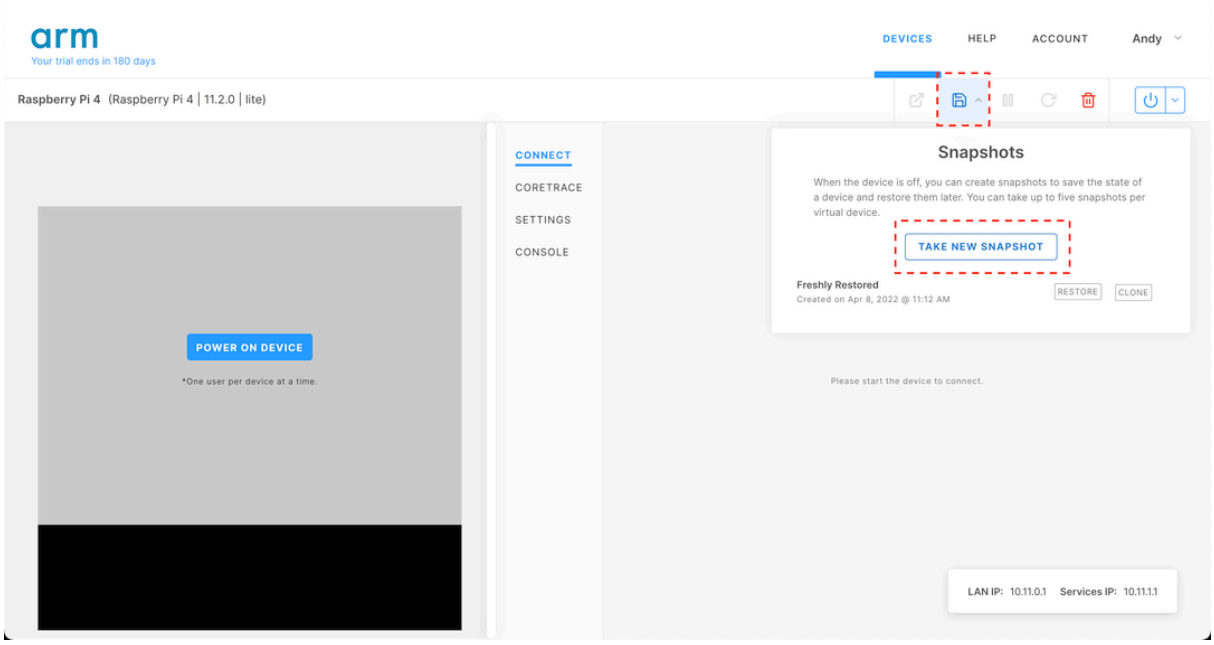
1. Find the device you created in the Quickstart for Raspberry Pi 4 tutorial.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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.

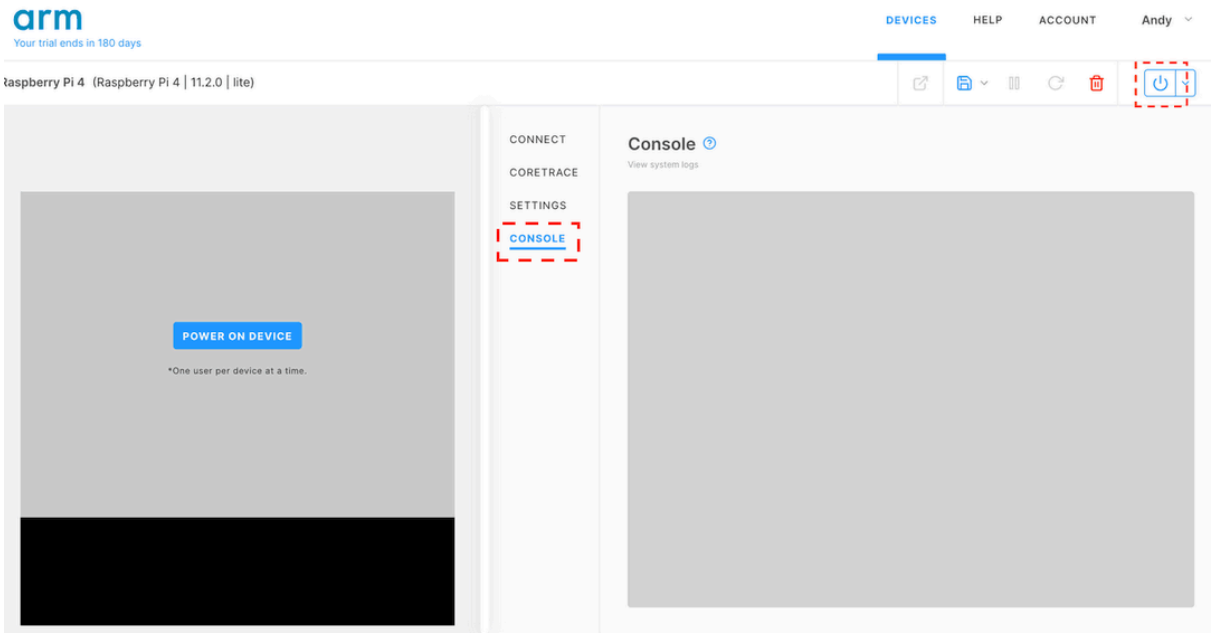


3. Click the **Snapshot** icon, then click **TAKE NEW SNAPSHOT**. You will see **Snapshot # 1** appear below the **Freshly Restored** snapshot.

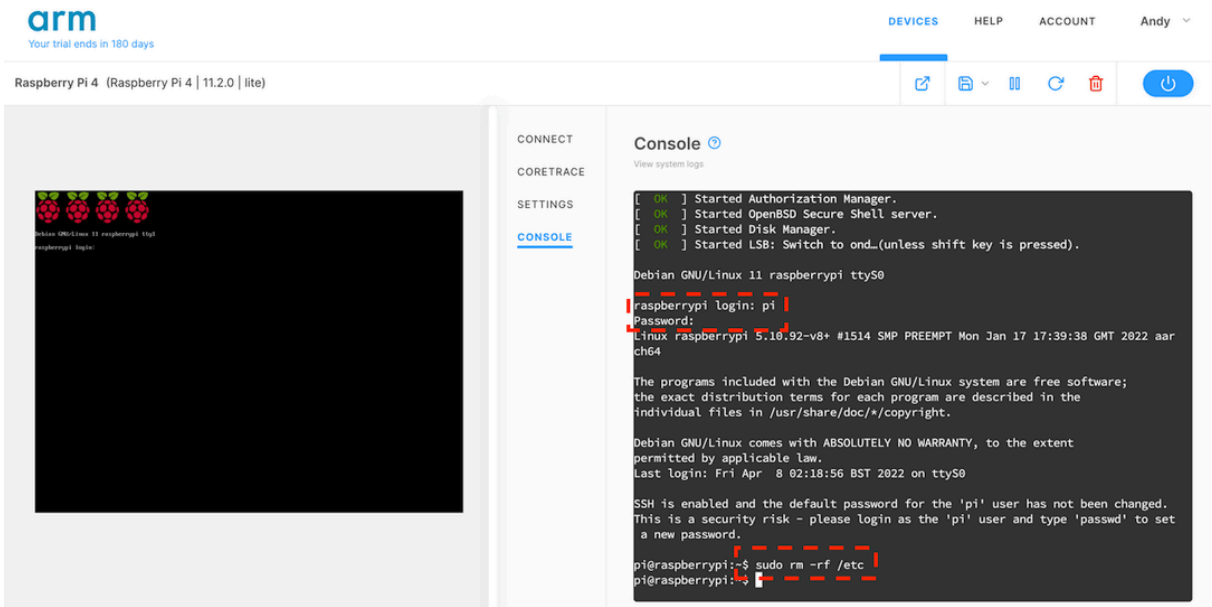


Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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

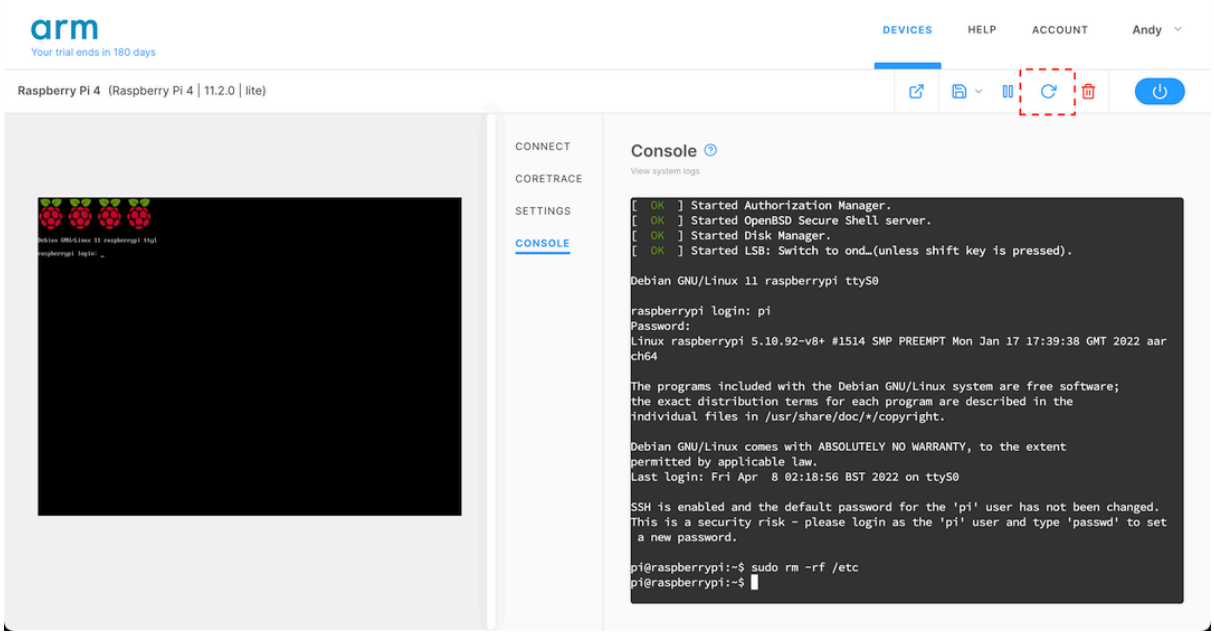


5. Login with username: `pi` and password:`raspberry`, type `sudo rm -rf /etc`, then hit `Return`.

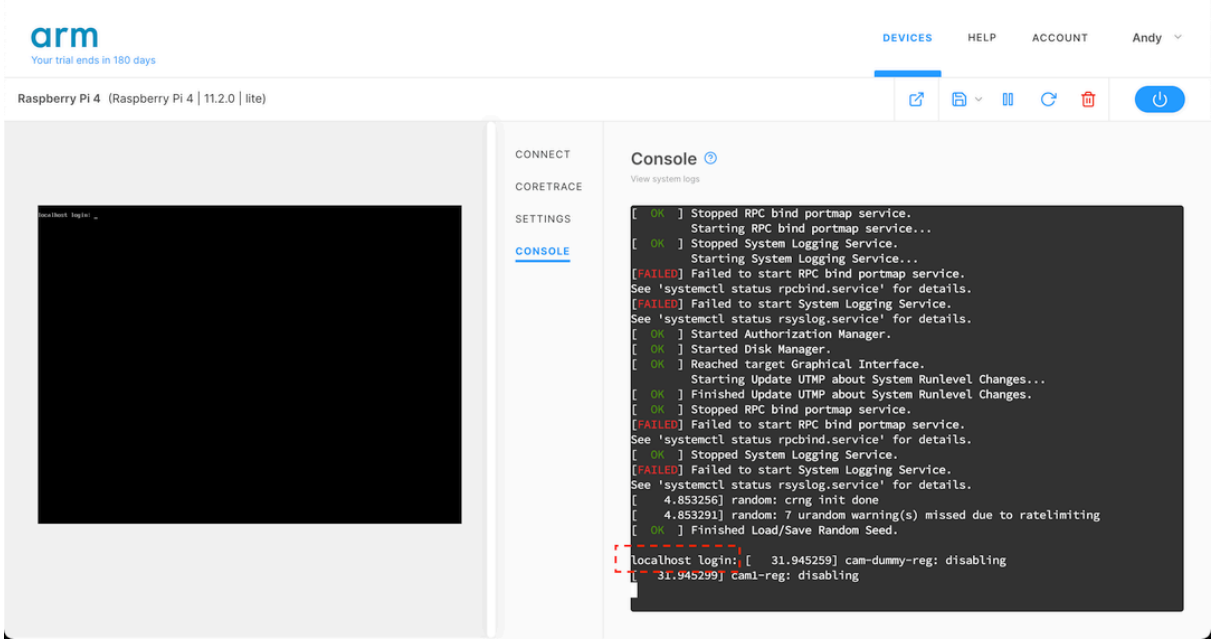


Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6. Click **Restart** on the top-right.

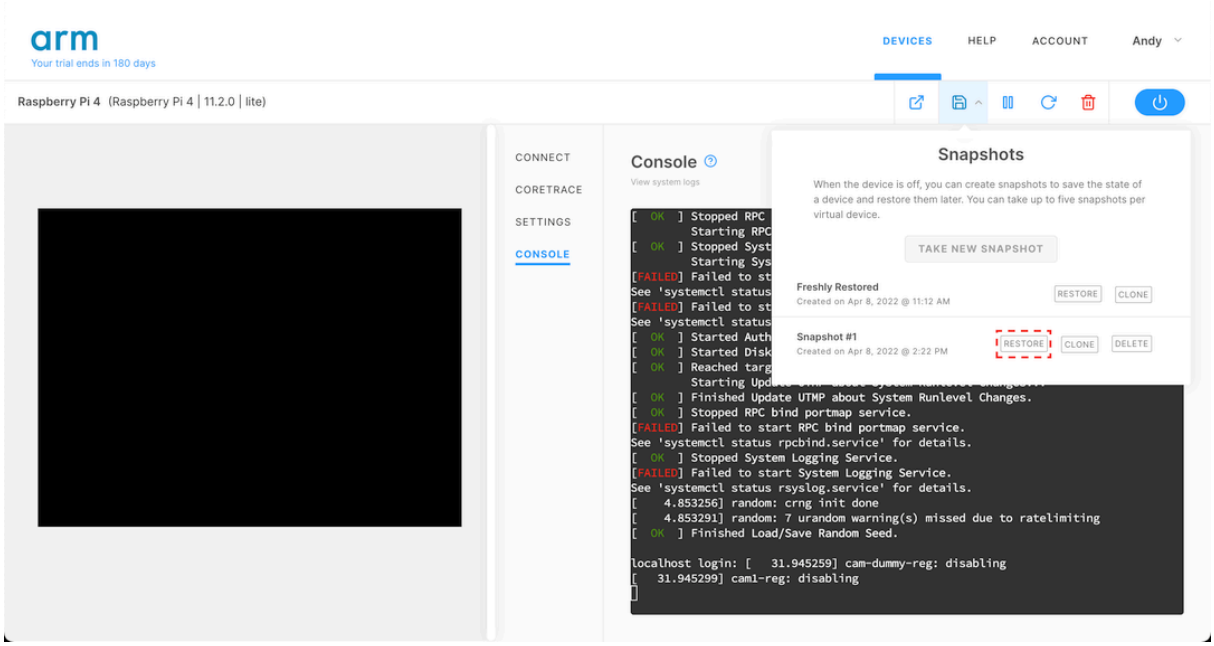


7. The **Console** should display `localhost login:`. This will validate the device is broken, as it is unable to log in normally.

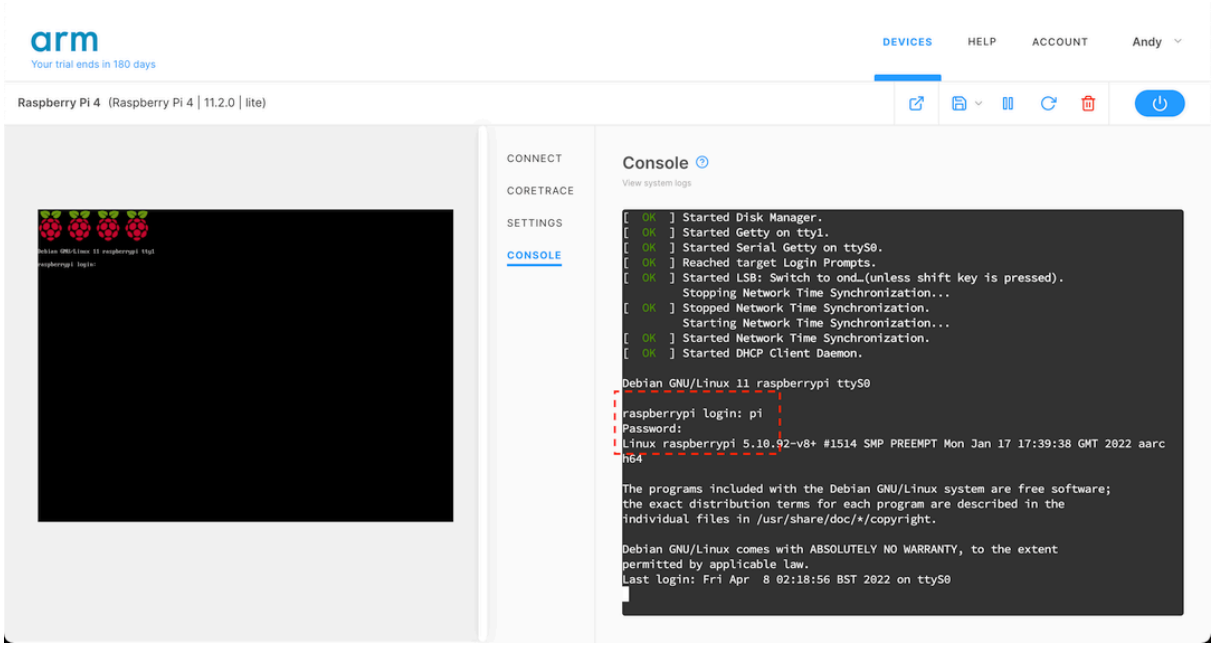


Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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.



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



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5. Device Firmware

See following pages for information about firmware for devices in Arm Virtual Hardware:

- Stock Firmware
- Linux Firmware Packages
  - Change Kernel Boot Arguments
  - Custom Device Tree
- MCU Firmware Packages
- Upload Firmware in Web UI
- Upload Firmware with Python API

5.1 Stock Firmware

To quickly get started with the virtual devices Arm Virtual Hardware provides stock firmware examples - ready-to-use program images stored on the AVH server.

Stock firmware demonstrates key features of the platforms such as OS boot, interaction with console, reading sensors and connectivity. Device-specific sections in Supported Platforms explain the stock firmware available for the virtual devices.

You can choose the stock firmware when creating a new virtual device. On the **Configure your device** screen in drop-down menu **Select an example firmware** you can select the target stock firmware that the virtual board instance shall run. You can also download corresponding stock firmware package by clicking on the **Source image** link.

Stock firmware is packaged in the same format as explained in Linux Firmware Packages and MCU Firmware Packages, and can be used as a reference for your custom firmware.

5.2 Linux Firmware Packages

For virtual devices capable to run Linux (with Arm Cortex-A cores) you can create and run your own firmware packages with Linux distributions. Such firmware package is a zip archive containing multiple file as explained in this section.

Linux firmware zip package content

The firmware package with Linux support for Arm Virtual Hardware is a zip archive that shall contain following files:

- `Info.plist` file with meta information about the firmware package in JSON format. See details in Info.plist file for Linux firmware.
- `kernel` file with a Linux kernel in the `Image` format. See Kernel image file for more.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- `devicetree` file with flattened Device Tree Blob (DTB). See Device Tree file for additional information
- Other files may be required for the boot process on the specific platform. See Storage for details.

An example of Linux firmware package is explained in Custom Firmware for Raspberry Pi 4.

Info.plist file for Linux firmware

An `Info.plist` file in the firmware package specifies the meta information for the firmware: version, type, build, unique identifier and device identifier, and optionally boot arguments. The file can be also provided under name `Info.json` with the same format.

Info.plist file 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>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>
```

Bootargs:

Boot Arguments are specific configuration flags passed to Linux Kernel during the boot process. Every kernel has certain arguments it accepts and the values of these flags can be changed.

See Linux kernel command-line parameters at www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html. Vendor devices may offer other undocumented command-line parameters.

The default `bootargs` are provided by the model, but can be overriden in the `Info.plist` or `Info.json` file. The `normal` key is used for every regular boot. The `restore` key, if present, is used for a “firstboot” prior to the device being declared ready. It is expected that the device will reboot itself to indicate that this phase is complete. It is used, for example, on the Raspberry Pi 4 to expand the root `fs` as shown below:

```
<key>bootargs</key>
<dict>
    <key>normal</key>
    <string>earlycon=uart8250,mmio32,0xfe215040 console=ttyS0,115200n8 rw rootwait
 root=/dev/mmcblk0p2 coherent_pool=1M 8250.nr_uarts=1 cma=64M</string>
    <key>restore</key>
    <string>earlycon=uart8250,mmio32,0xfe215040 console=ttyS0,115200n8 console=tty0
 rw rootwait root=/dev/mmcblk0p2 coherent_pool=1M 8250.nr_uarts=1 cma=64M init=/usr/
lib/raspi-config/init_resize.sh</string>
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
</dict>
```

Kernel image file

A `kernel` file in the Linux firmware package provides the uncompressed Linux kernel image in `Image` format.

Device Tree file

A `devicetree` file in the firmware package is a binary flattened Device Tree Blob (DTB) that represents the hardware used by bootloaders 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. For more information on device trees, see http://elinux.org/Device_Tree.

The `devicetree` file shall be present in the firmware package. You can also change the device tree after the virtual device is created without changing the full firmware or re-creating the device instance, see Custom Device Tree for more information.

Storage

In hardware boards non-volatile memory (NVM) (such as flash memory) is used to provide a OS image and program code for the boot process. In AVH devices this memory is enabled with storage files that need to be provided as part of the firmware pacakge. For the proper names refer to the storage files page for a target device listed in Supported Platforms.

For example, virtual Raspberry Pi 4 uses storage file `nand` for the content of its microSD flash memory, as also referenced in Storage Files for Raspberry Pi 4.

5.3 Change Kernel Boot Arguments

Boot arguments are specific configuration flags passed to the kernel of operating system (OS) during the boot process. Their configuration is available in Arm Virtual Hardware for virtual devices capable to run Linux OS (based on Arm Cortex-A processors), for example Raspberry Pi 4 and i.MX 8M Arm Cortex Complex devices.

Default kernel boot arguments are provided by the virtual model, or as part of the firmware file `Info.plist` as explained in [Linux Firwmare Packages]. But it is also possible to modify the boot arguments for existing device instances.

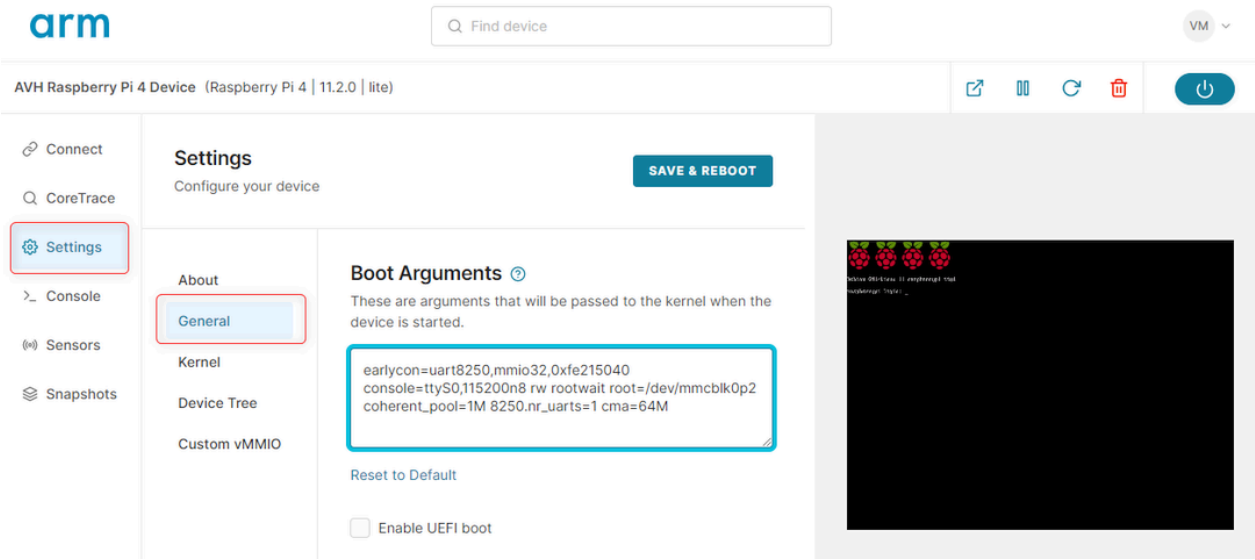
Note

In general, it may be not safe to remove or change the default arguments if you aren't sure what you're doing, since the device and UI features may no longer work correctly.

To customize boot arguments on your virtual device, simply head to the **Settings** tab, open the **General** subtab, modify the appropriate text field, then click **SAVE & REBOOT**.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-1: Kernel Boot Arguments



5.4 Custom Device Tree

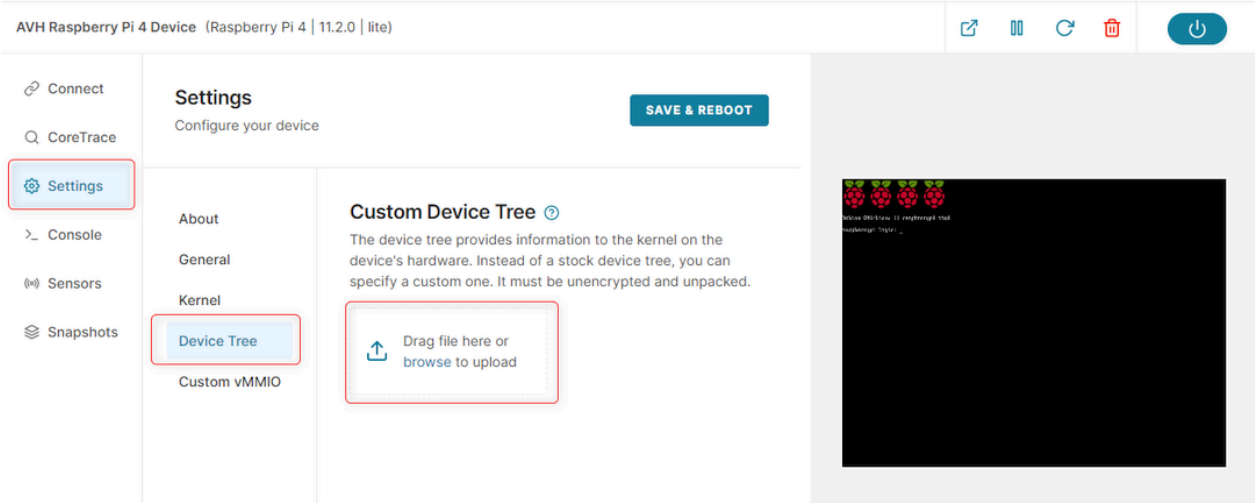
Device Tree is a representation of hardware that is used by bootloaders and provided to the kernel of the operating system. Device Trees are supported in Arm Virtual Hardware for virtual devices capable to run Linux OS (based on Arm Cortex-A processors), for example Raspberry Pi 4 and i.MX 8M Arm Cortex Complex devices.

Default Device Tree is provided as part of device firmware as explained in [Linux Firwmare Packages] and is used for device creation. It is also possible to customize the Device Tree on existing device instances. The Device Tree file must be in a binary flattened Device Tree Blob (DTB format), unencrypted and unpacked.

To upload your custom Device Tree file to your virtual device, simply head to the **Settings** tab, open the **Device Tree** subtab and drag-and-drop your device tree file to upload it. Then click **SAVE & REBOOT**.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-2: Custom device tree



5.5 MCU Firmware Packages

For virtual microcontroller-based devices (MCUs) powered by Arm Cortex-M cores you can mostly use the same firmware files that also run on the real hardware.

There are two approaches to provide the MCU firmware for virtual devices:

1. As a complete package, explained in MCU firmware zip package content.
2. As a standalone firmware file, explained in Firmware file formats.

MCU firmware zip package content

The MCU firmware package is a zip archive that allows you to fully utilize the complexity of the target virtual device and all its memory partitions. It contains following files:

- `Info.plist` file with meta information about the firmware package in JSON format. See more in Info.plist file for MCU firmware.
- `firmware` file with your program. Various formats are supported as explained in Firmware file formats.
- Other files that may be needed for the boot process on the specific platform. See Storage for details.

Info.plist file for MCU firmware

An `Info.plist` file in the firmware package specifies the meta information for the firmware: version, type, build, unique identifier and device identifier. The file can be also provided under name `Info.json` with the same format.

Info.plist file example:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
<!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>1.1.1</string>
    <key>Build</key>
    <string>WB</string>
</dict>
</plist>
```

Firmware file formats

A firmware file contains your program code for the target MCU. It can be provided in one of the following formats:

- **ELF executable** - 32-bit ELF program file containing all information needed to load the program, also supporting 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** - containing several binary files and a text file with name `load.txt` that specifies their load locations in device memory. This format is required when you need to use several binary files. An example load file could look like this:

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

  This `load.txt` would take the content of `bl2.bin` file from the same ZIP archive and load it to memory address `0x00000000` on the virtual device, and the content of `tfm_s_ns_signed.bin` file at address `0x01000000`.

You can use firwmare file as standalone, or also as part of a more complex firmware package as mentioned in MCU firmware zip package content above.

Storage

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

5.6 Upload Firmware in Web UI

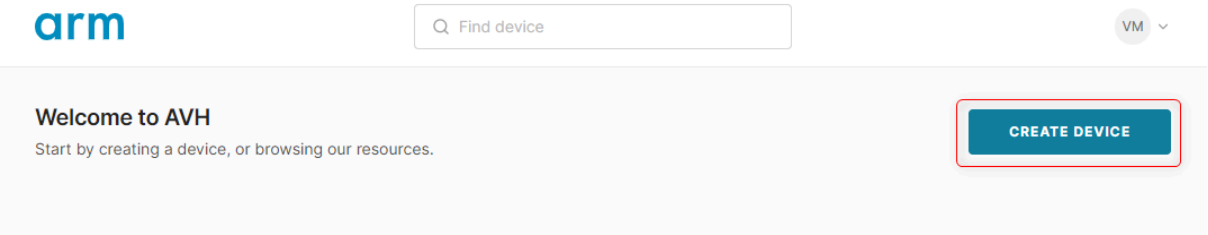
This tutorial demonstrates how to upload custom firmware to a new virtual device. It uses Raspberry Pi 4 as the target platform, but the approach is similar for other devices.

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/.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

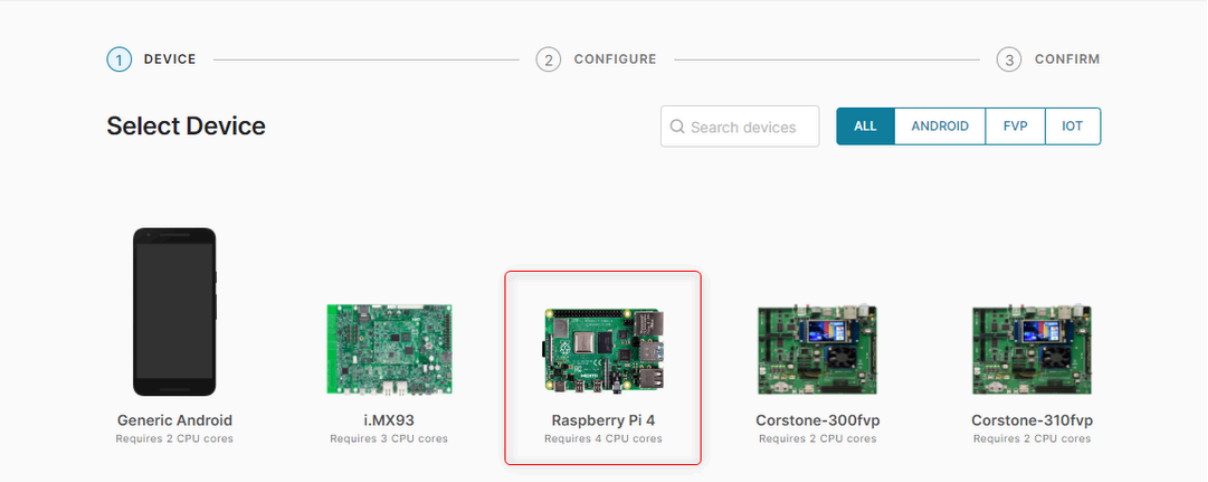
2. When logged in you land on the AVH Devices page. Click **CREATE DEVICE**.

Figure 5-3: Create device



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

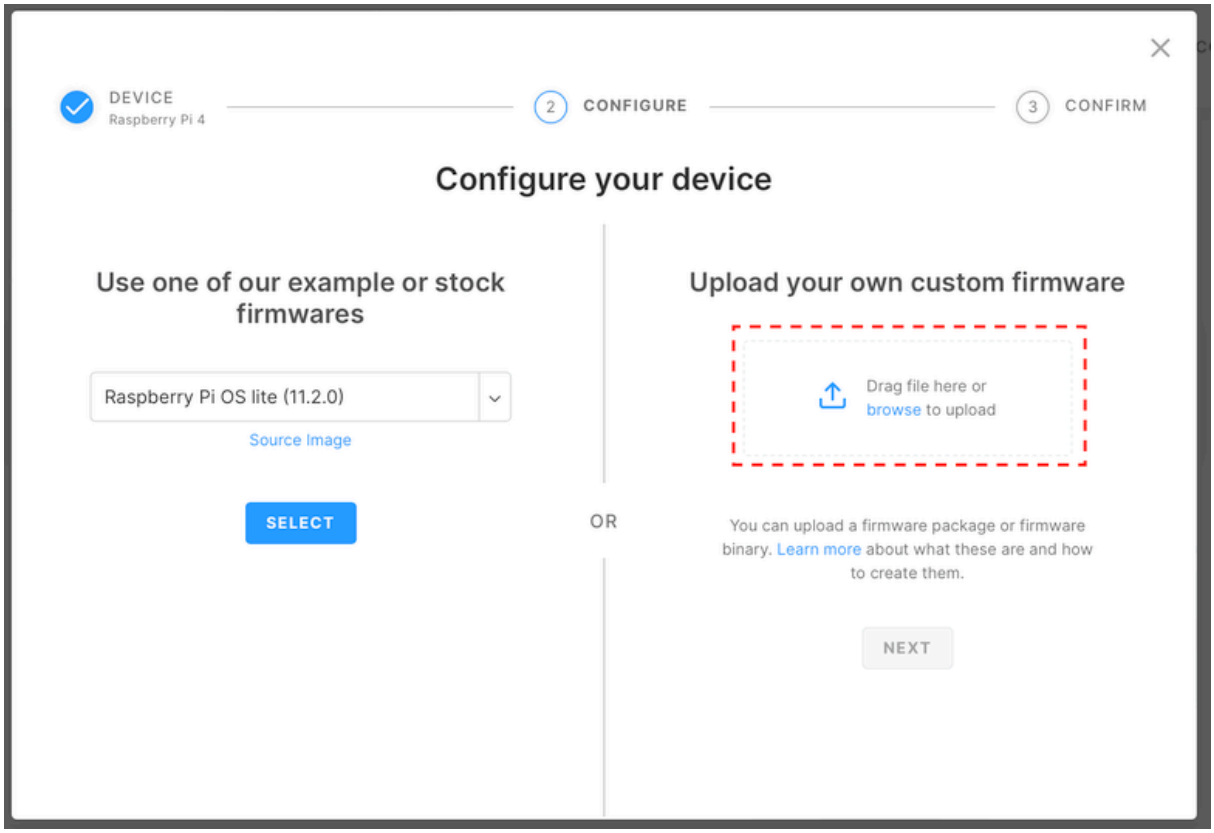
Figure 5-4: Select device



4. In this next step, you'll see a list of pre-configured examples/stock firmware or you can upload your own custom firmware package. 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

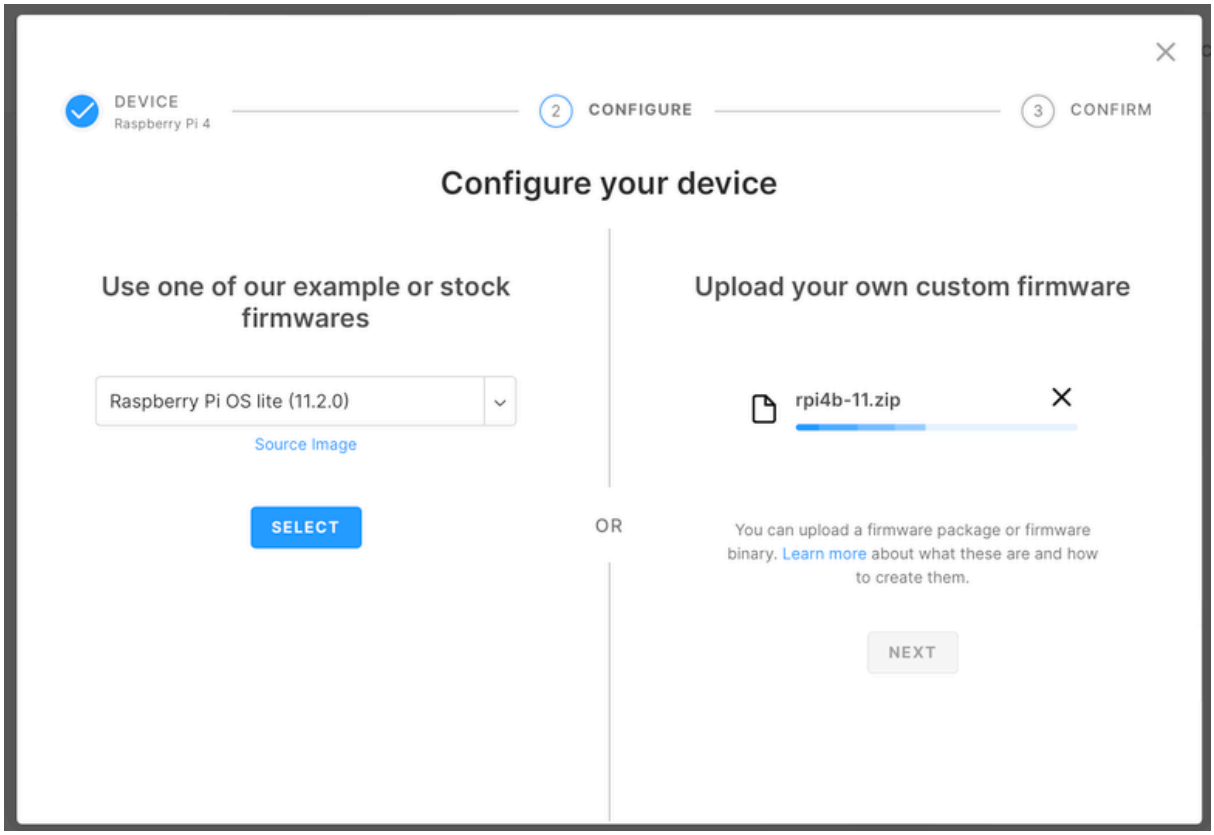
Figure 5-5: Configure your device



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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

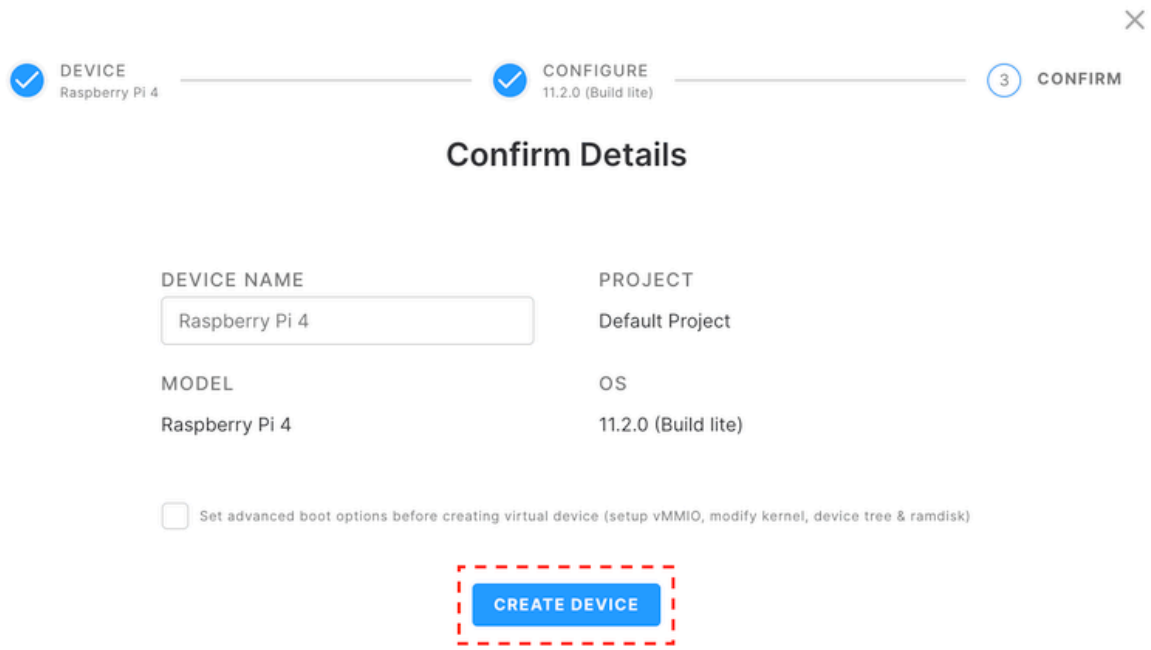
Figure 5-6: Upload firmware



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-7: Confirm details



7. The device will then be created with custom firmware.

5.7 Upload Firmware with Python API

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

Page 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 board's console via a WebSocket using AVH Asynchronous Python API.

Overview of the Python Script

The script performs the following tasks:

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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.
Uploading the custom firmware package ...
Uploaded image cafcb47f-8e48-4ca8-8f00-df07793671.
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 ...
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.

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
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
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:
        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 + '.')

    # 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:
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
      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)
      #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)
```

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.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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'
```

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
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6. Connect to Virtual Devices

Arm Virtual Hardware environment is featured with multiple connectivity options that allow remote connections as well as communication between devices.

- Remote Connections to Devices gives an overview on how to connect to your devices from remote machines for monitoring and control and debug.
- Connections between Devices describes how virtual devices can communicate to each other using LAN/WLAN and Bluetooth connections.

6.1 Remote Connections to Devices

You can connect to your active virtual devices remotely for monitoring, control and debug purposes, but also to extend their functionality, for example by running virtual memory mapped I/O (vMMIO) peripherals outside of AVH environment.

The remote connection is performed in two steps:

- Step 1: you need to establish a secure connection with your AVH environment. This can be done in following ways:
  - Use Quick Connect feature to establish an SSH tunnel.
  - Or Connect to your AVH VPN using a VPN client.
- Step 2: you can start communicating with the interfaces available on individual virtual devices and exposed over Services IP and LAN IP addresses. Following connections are possible:
  - Connect to Device Console for remote input/output communication over the serial console in the same way as it would work locally on the hardware.
  - Connect to Device TCP/UDP Ports available on the device. For example you can establish a Secure Shell (SSH) connection to your devices with Linux, or access a web-page on the HTTP server hosted by the virtual board.
  - Use GDB Connection for debugging the firmware on virtual devices with GNU Project Debugger (GDB).
  - Extend device peripherals with virtual memory-mapped I/Os as explained in vMMIO Introduction.

6.2 Connections between Devices

Your virtual devices in AVH environment have various options to communicate to each other over such interfaces as LAN / WLAN, Bluetooth.

LAN / WLAN

Your virtual devices are automatically configured to be on the same Local Area Network (LAN) and can communicate to each other using the LAN IP addresses. See Services IP and LAN IP.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Arm Virtual Hardware environment provides a virtual WiFi network that can be accessed by WiFi-capable devices by using the following parameters:

- SSID: `Arm`
- Password: `Arm`
- Security: WPA2

The `Arm` WiFi network can be used by devices that have their on-board WiFi interfaces modelled (for example STM32U5 IoT Discovery Kit), as well as by devices with a virtual WiFi interface (for example Raspberry Pi 4), that do not have WiFi present on real hardware by default.

After joining the WiFi network, the virtual device receives the LAN IP address as stated on the device page.

Bluetooth

Arm Virtual Hardware environment provides a virtual Bluetooth Low Energy (BLE) network that can be used for communication by devices with BLE interface. See Connect to BLE on Raspberry Pi 4 for a detailed usage example.

6.3 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 Quick Connect feature allows you to connect to your virtual devices remotely without requiring a VPN connection. It 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.

Note About Your Network Firewall Configuration

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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`).

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.

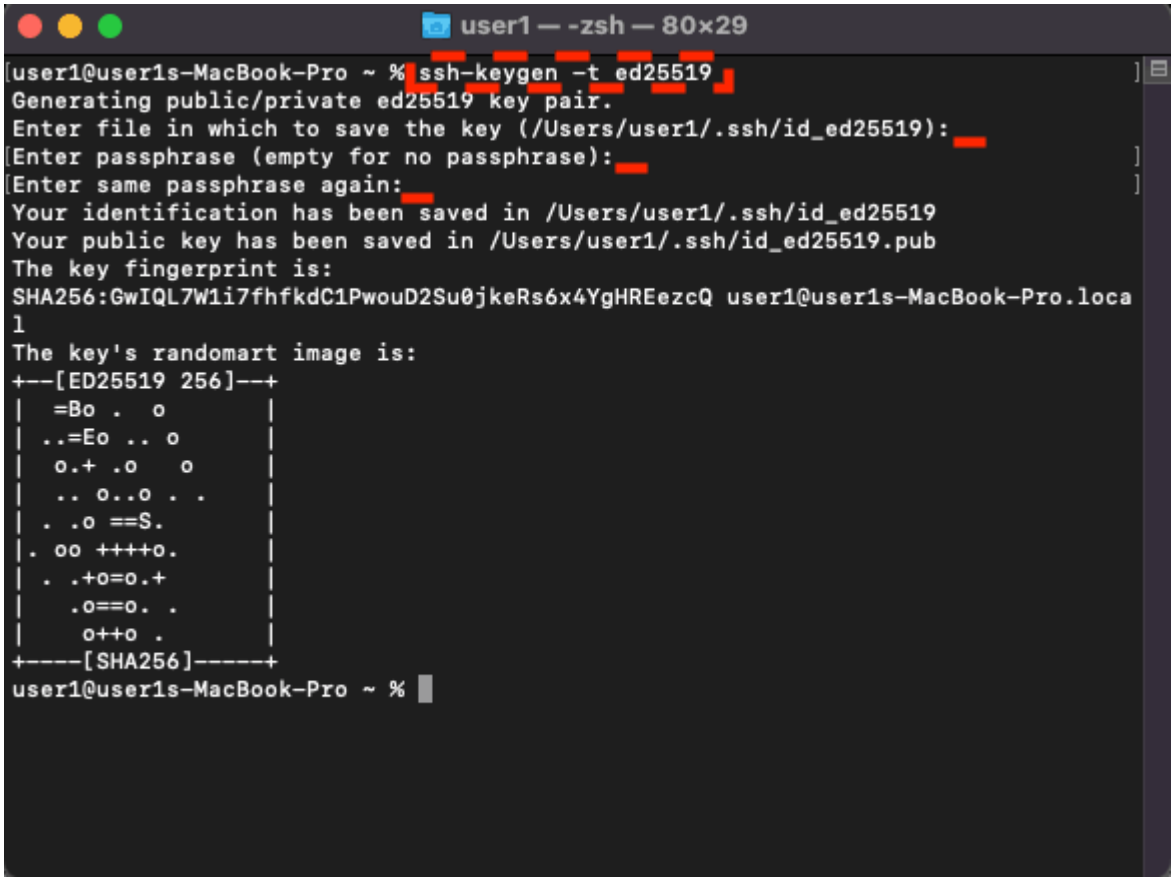
Generate a Key Pair with Terminal

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.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

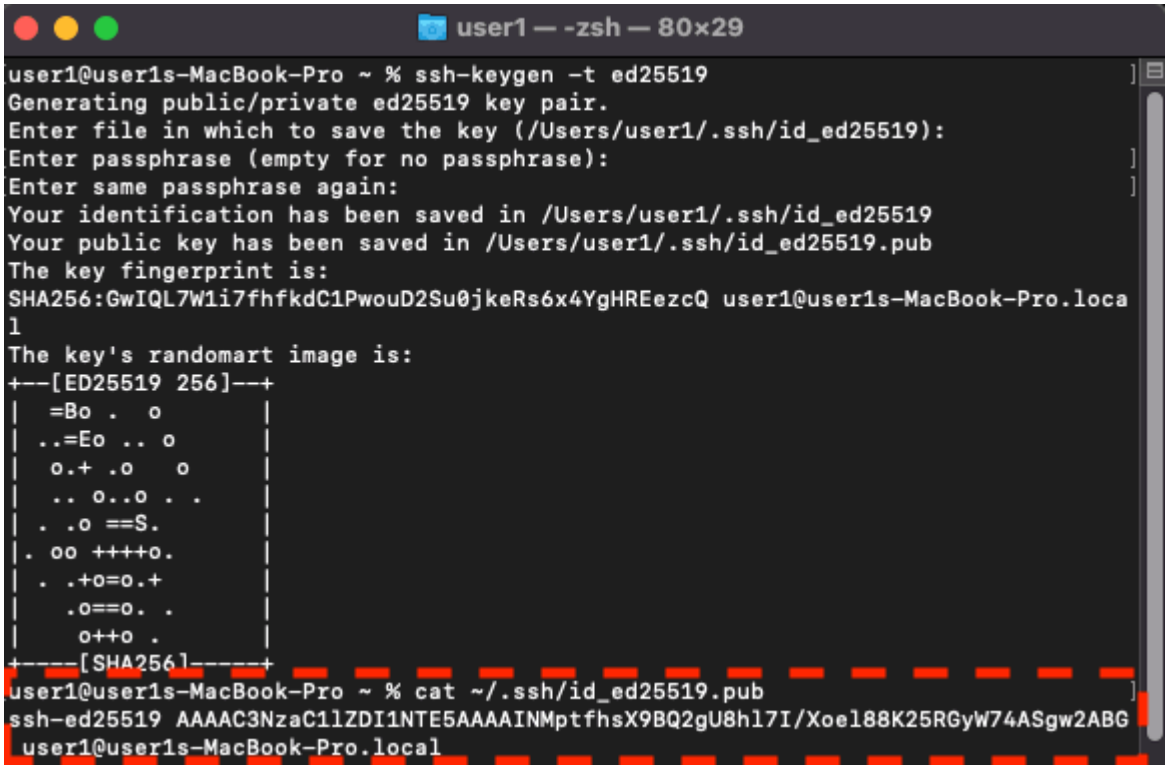
Figure 6-1: ssh-keygen output



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-2: 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
```

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 © 2022–2023 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
```

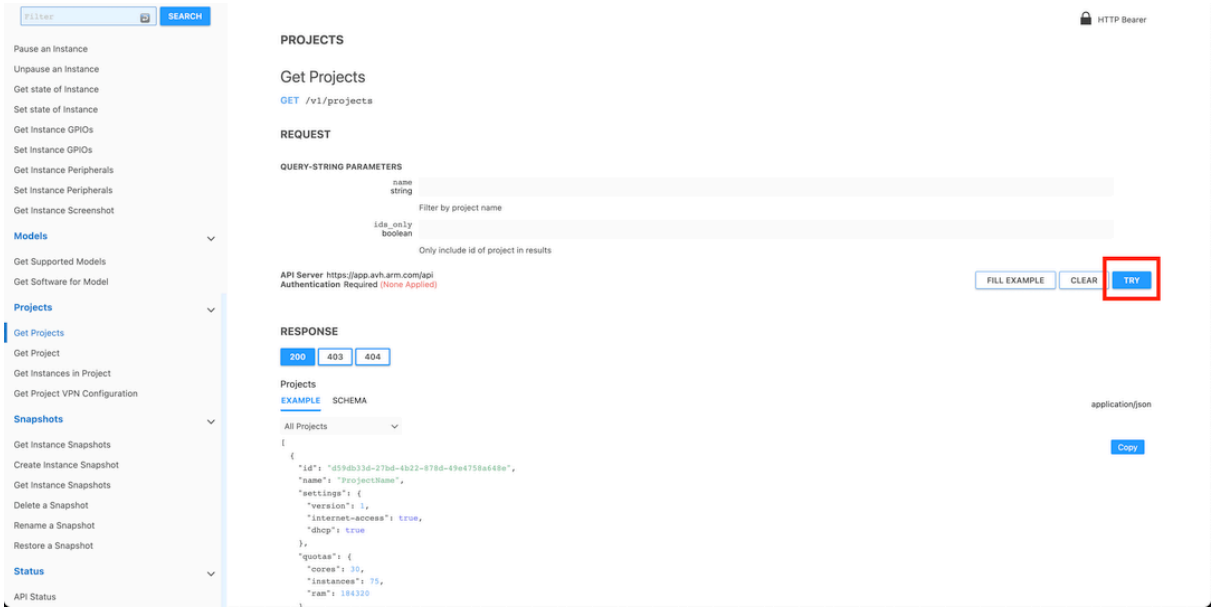
Step 2: Add Your SSH Public Key to Your Project

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 AVH REST API for guidance.
2. Go to the Get Projects in the API and click "Try".

Figure 6-3: 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".

Step 3: Create Your Device

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Note

You cannot add SSH keys to an existing device.

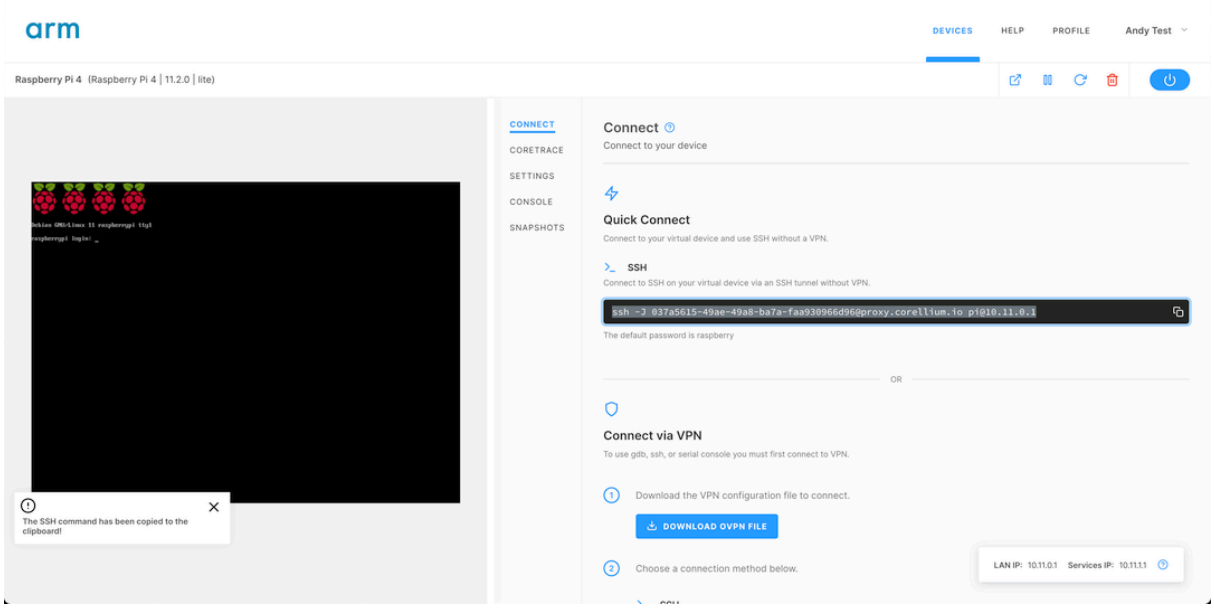
1. Create a new device and wait for it to boot. In this instance, we will create a Raspberry Pi, as explained in Quickstart for Raspberry Pi 4.

Step 4: Use Quick Connect

To use Quick Connect, do the following:

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

Figure 6-4: Quick Connect SSH command



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

Figure 6-5: Paste and run the command

```
Last login: Thu Jun 30 15:21:42 on ttys003
abashford@Andrews-iMac ~ % ssh -J 037a5615-49ae-49a8-ba7a-faa930966d96@proxy.corellium.io pi@10.11.0.1
You are connecting as 037a5615-49ae-49a8-ba7a-faa930966d96 from 101.182.221.124...
The authenticity of host '10.11.0.1 (<no hostip for proxy command>)' can't be established.
ED25519 key fingerprint is SHA256:/hVjAjoDAa1x40fm9vNmVdHCsw0UX1ewuL/hJN8mgak.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

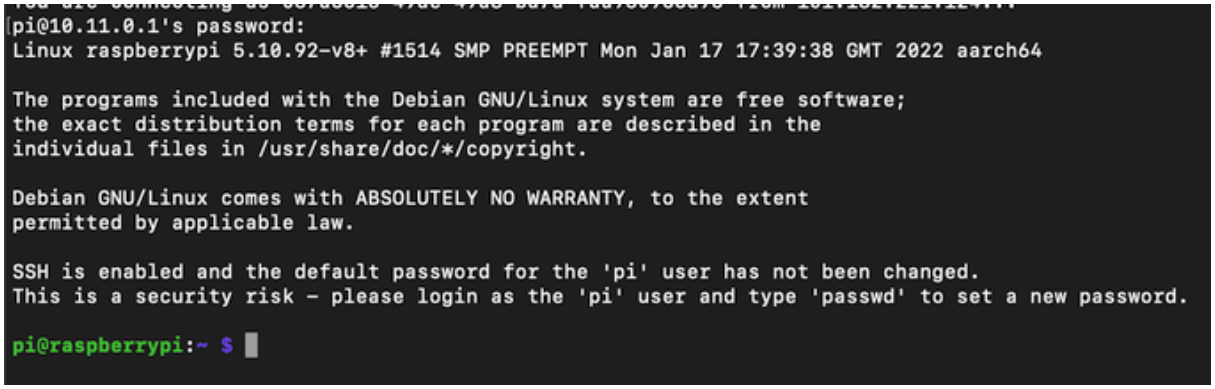
Note

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.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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, as shown above.
4. If there is a password required you can enter it now.

Figure 6-6: Enter the password



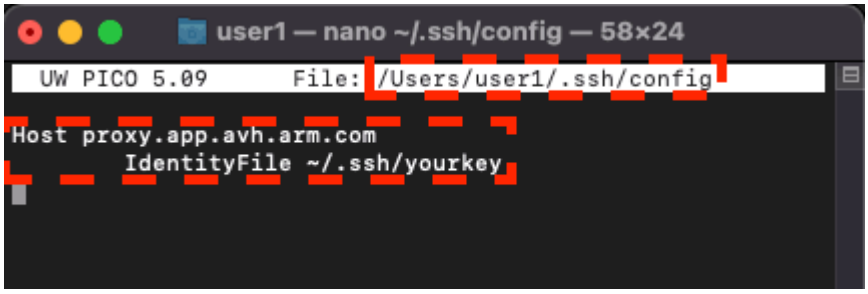
5. Now you are connected to AVH network without the need for a VPN.

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.)

Figure 6-7: Specify the IndentityFile



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.4 Connect to your AVH 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

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.

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

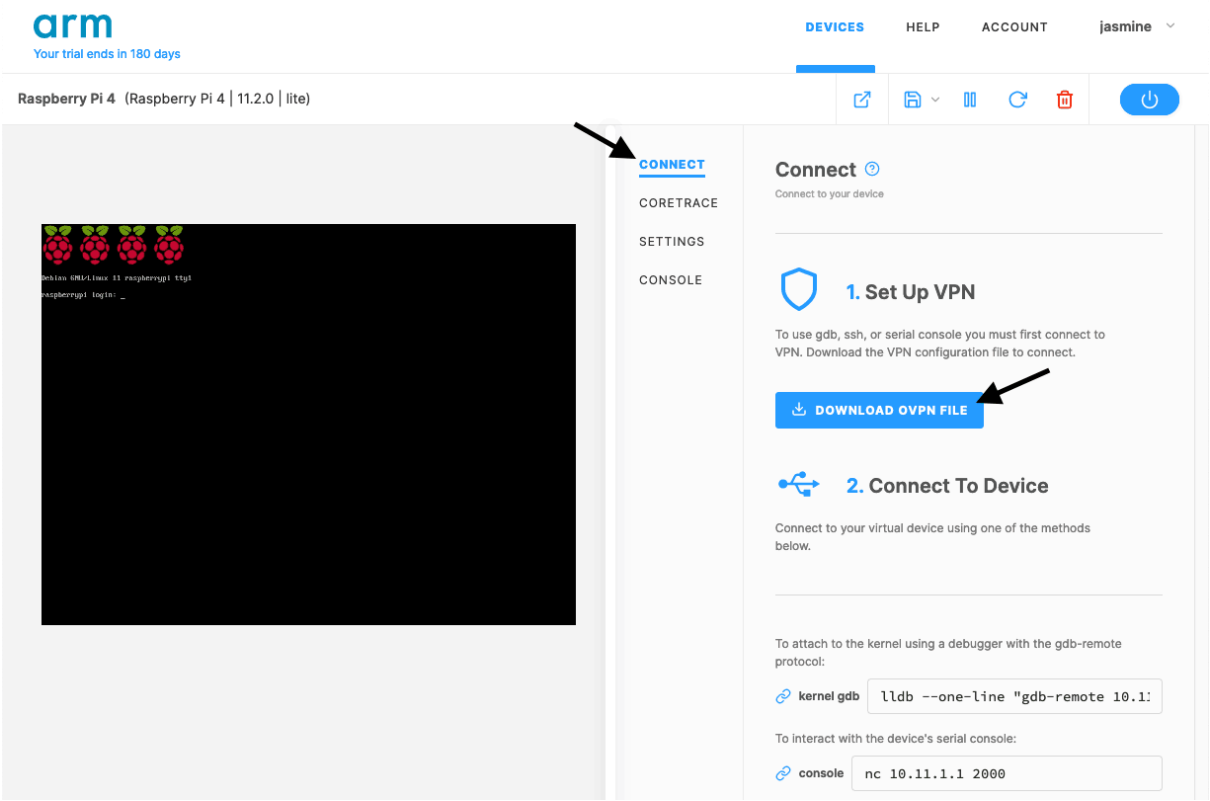
- Run a vmmio peripheral over TCP/IP on your local computer to feed stimuli data through a peripheral running locally on your computer into the Virtual Device for application and algorithm validation.

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, navigate to the **Connect** tab and then in the **Connect via VPN** area click **DOWNLOAD OVPN FILE**.

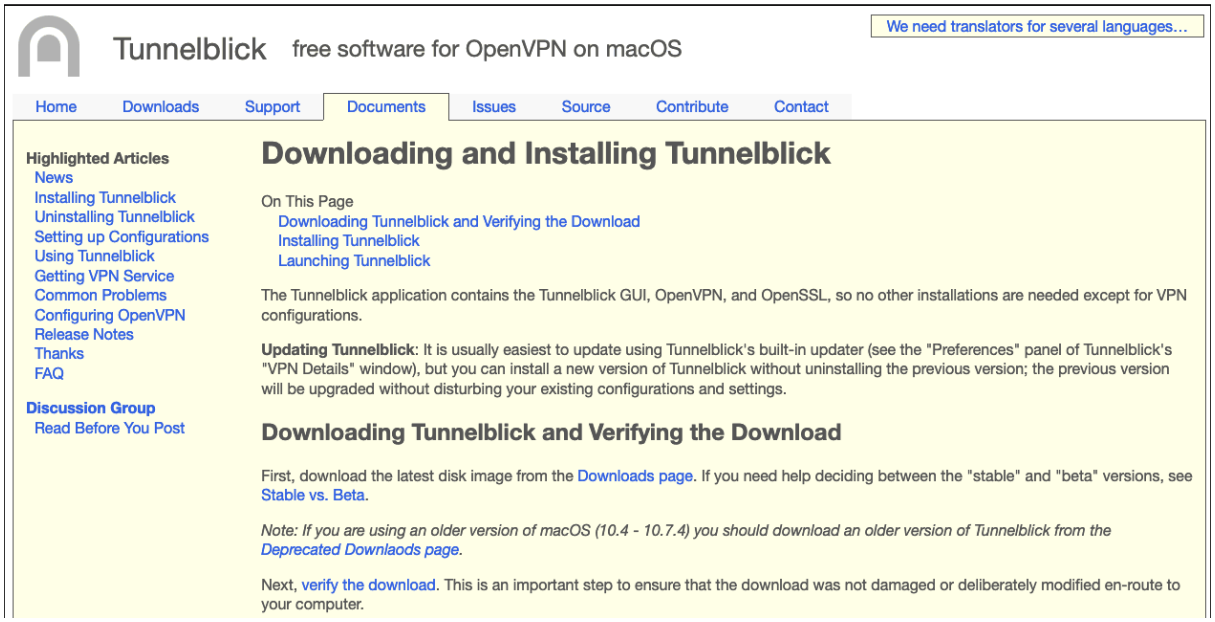
   Figure 6-8: DOWNLOAD OVPN FILE



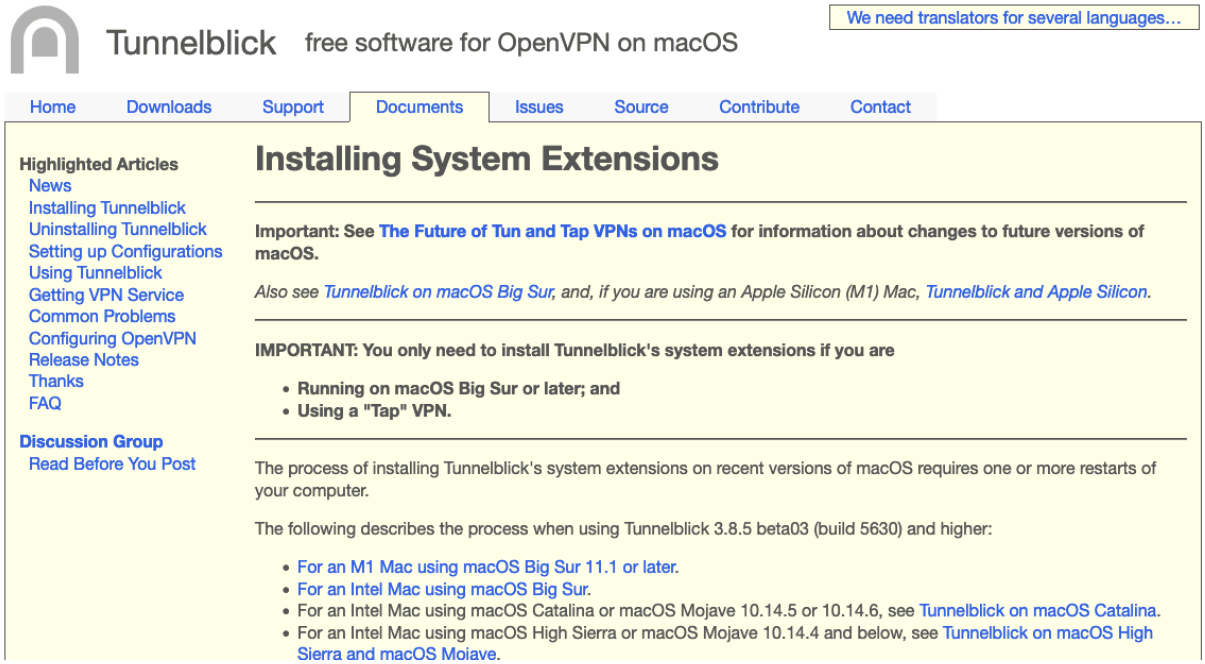
3. Download the appropriate version of Tunnelblick for your Machine from this URL
   - If you are on a new version of MacOS you need to use a patched version of Tunnelblick as MacOS no longer allows kext loading. That version is available from https://github.com/sbingner/Tunnelblick/releases/download/v3.8.8beta03%2Biouserethernet1/Tunnelblick.dmg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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-9: Downloading and installing Tunnelblick



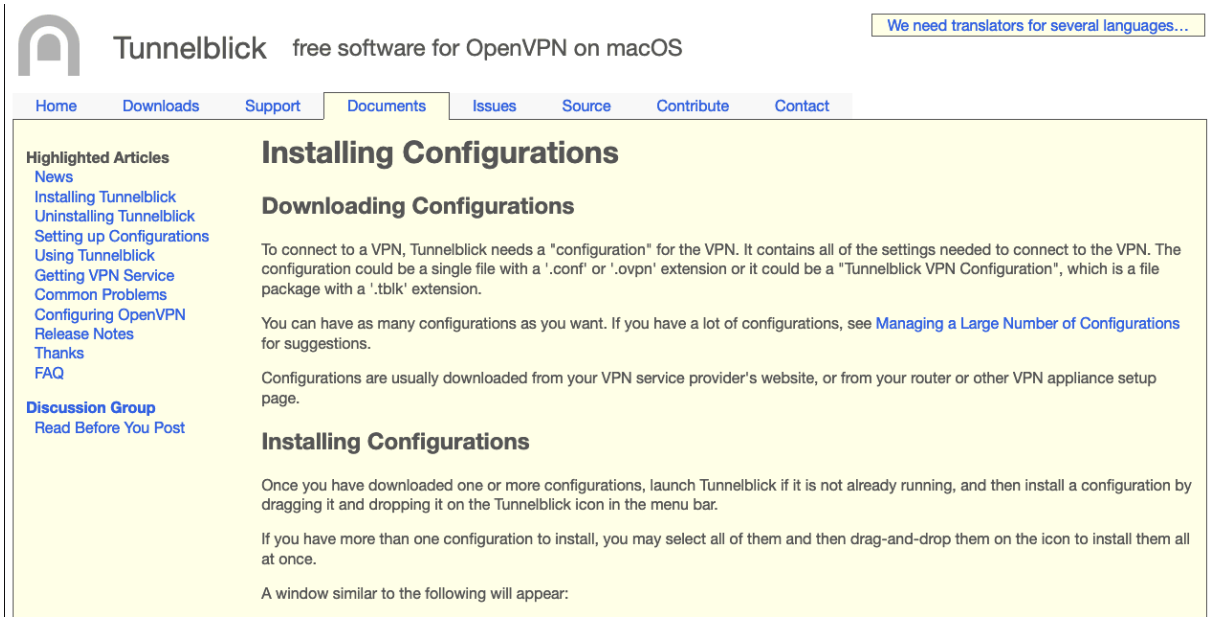
5. Navigate to this URL to install the system extensions for Tunnelblick.



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

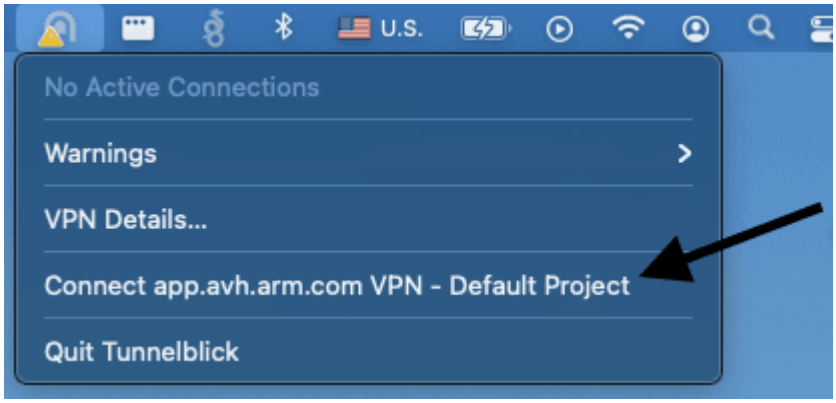
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-10: Installing Configurations



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

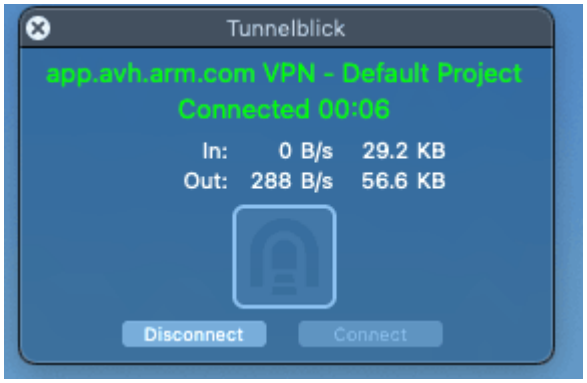
Figure 6-11: Connect



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-12: Connected message

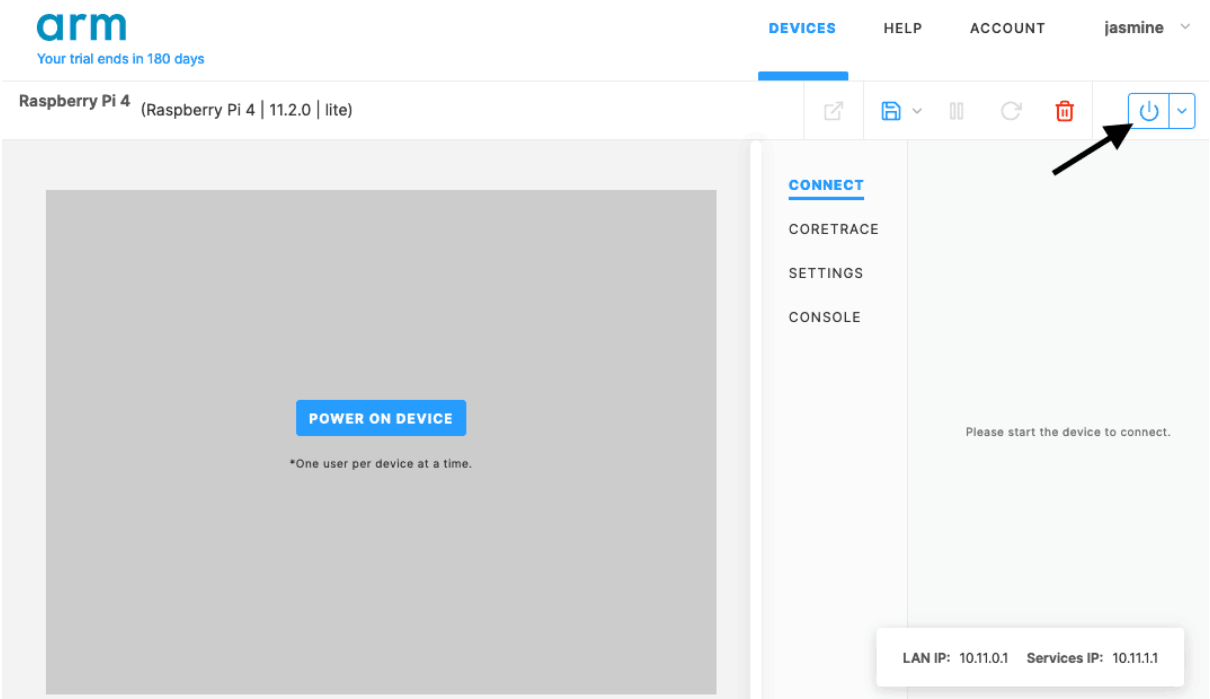


How to Connect with Windows OS

To connect with Windows OS, do the following:

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

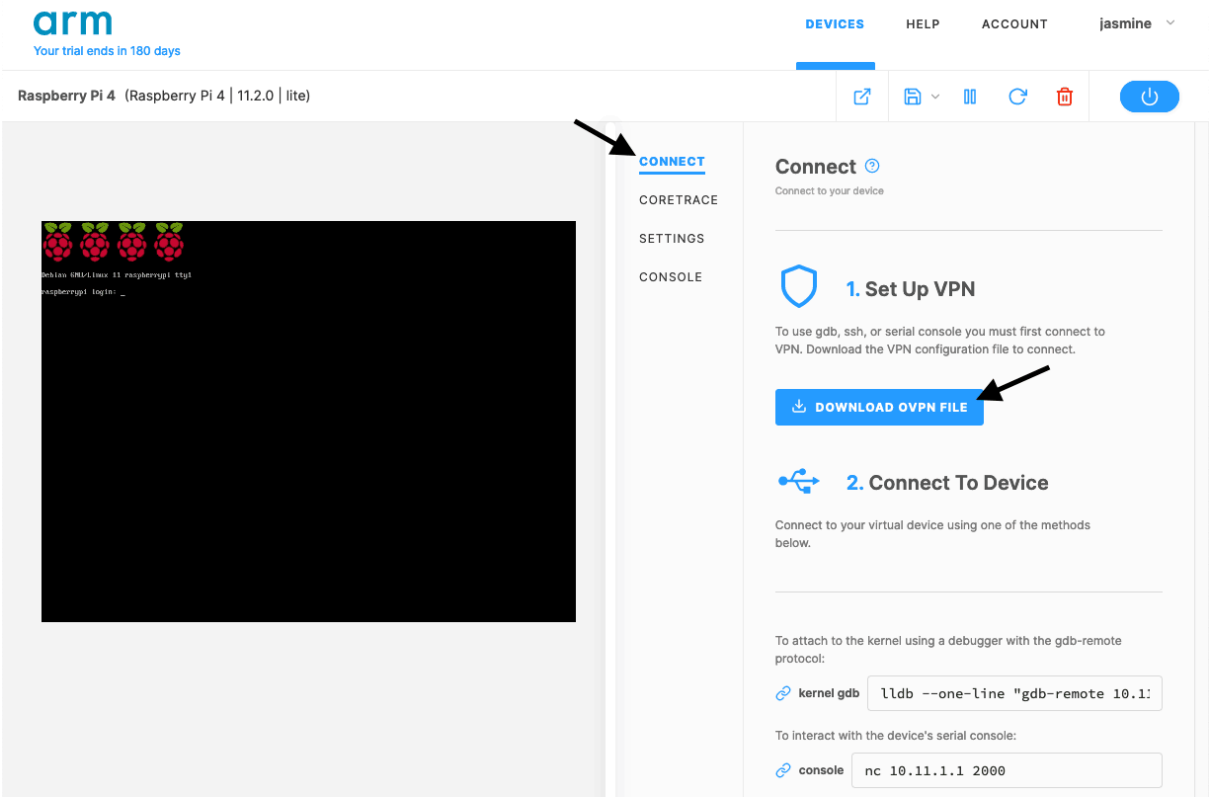
Figure 6-13: Turn on the device



2. Navigate to the **Connect** tab and then in the **Connect via VPN** area click **DOWNLOAD OVPN FILE**.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

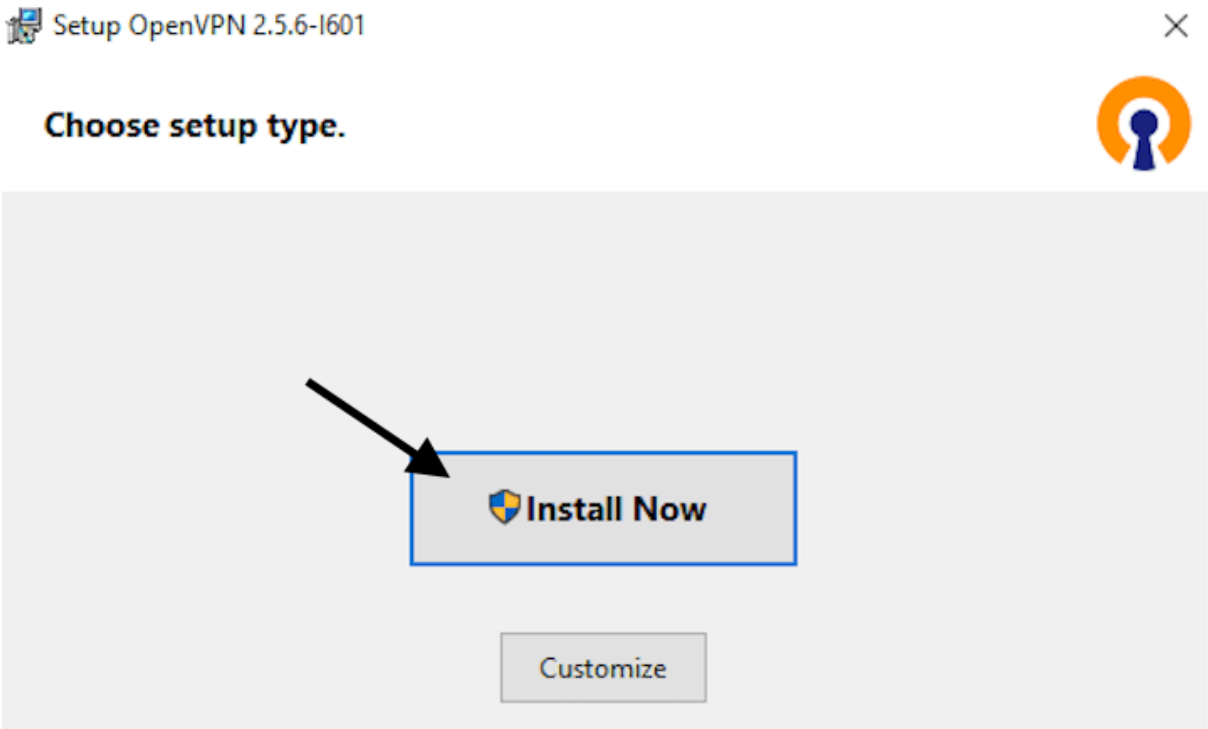
Figure 6-14: DOWNLOAD OVPN FILE



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

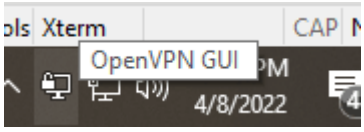
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-15: Install now



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

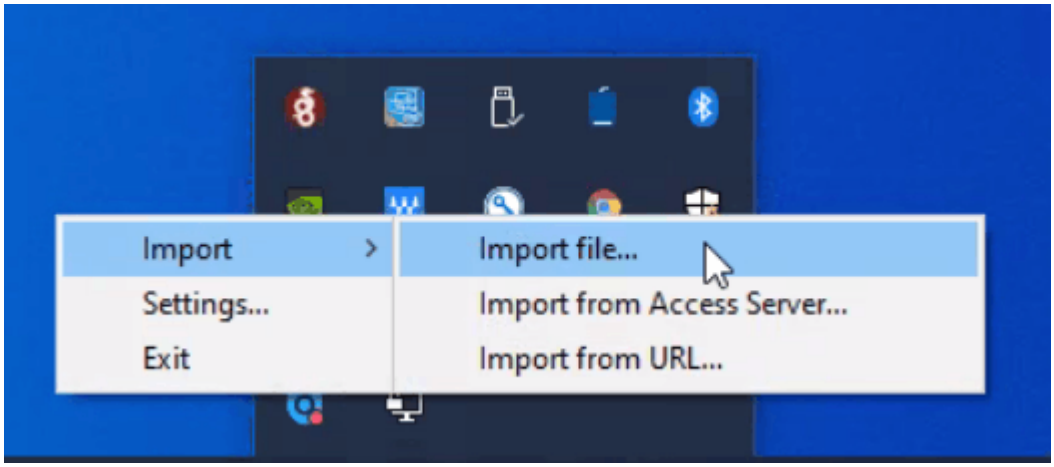
Figure 6-16: OpenVPN GUI



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

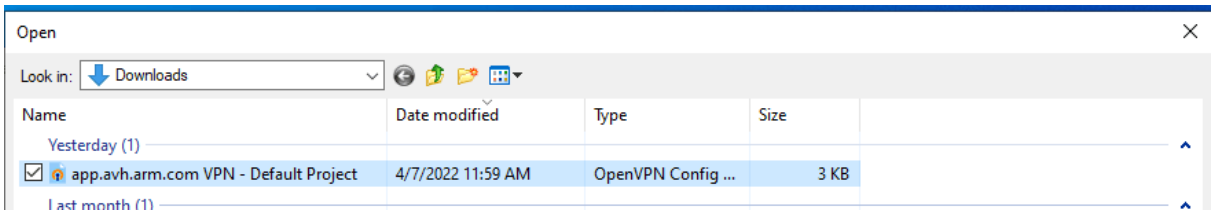
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-17: Imort file



7. Then, select the downloaded OVPN file.

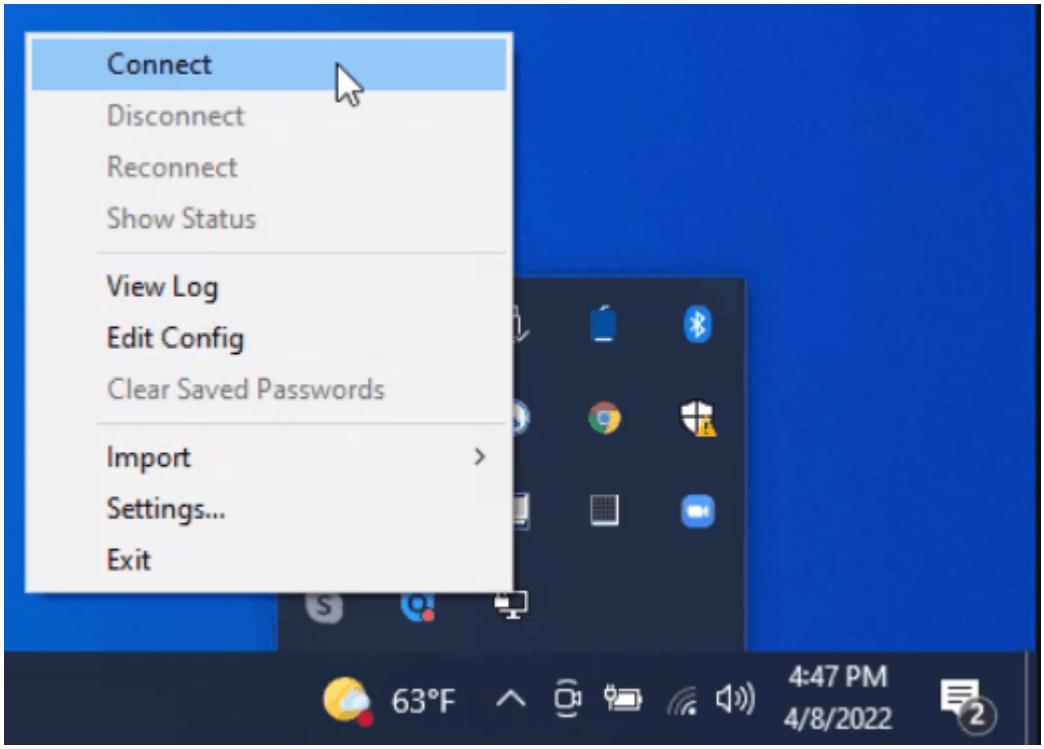
Figure 6-18: Select the OVPN file



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

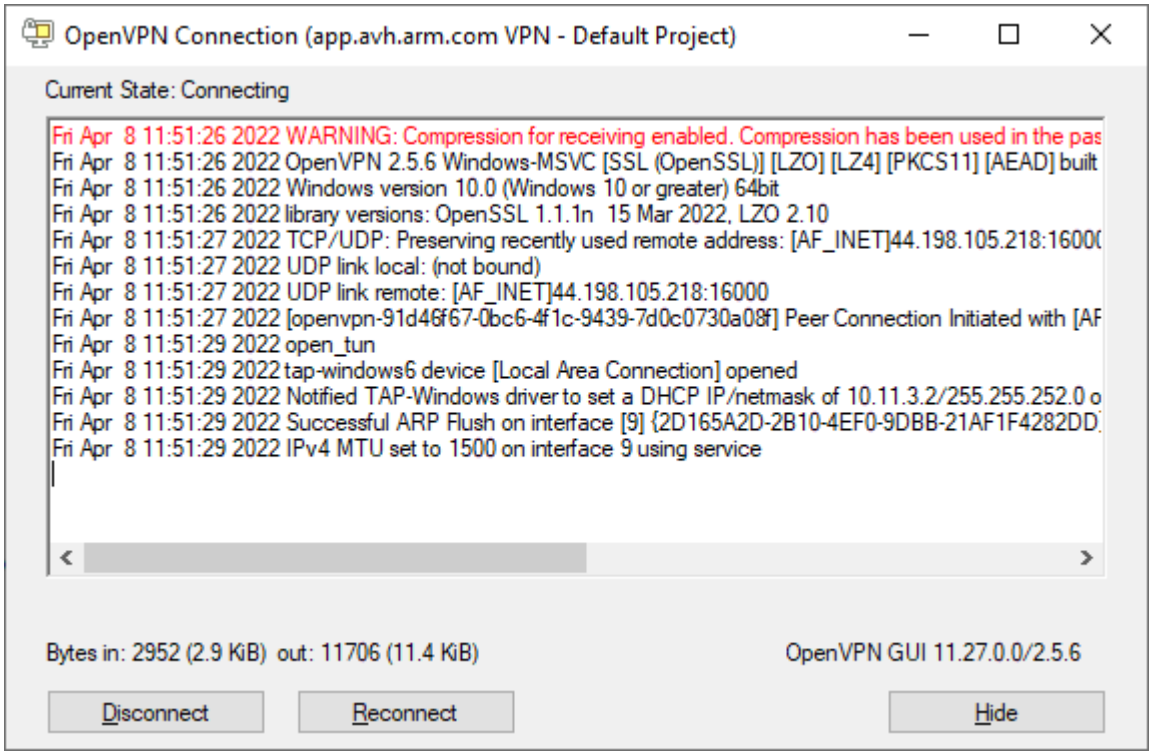
Figure 6-19: Connect



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-20: 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.

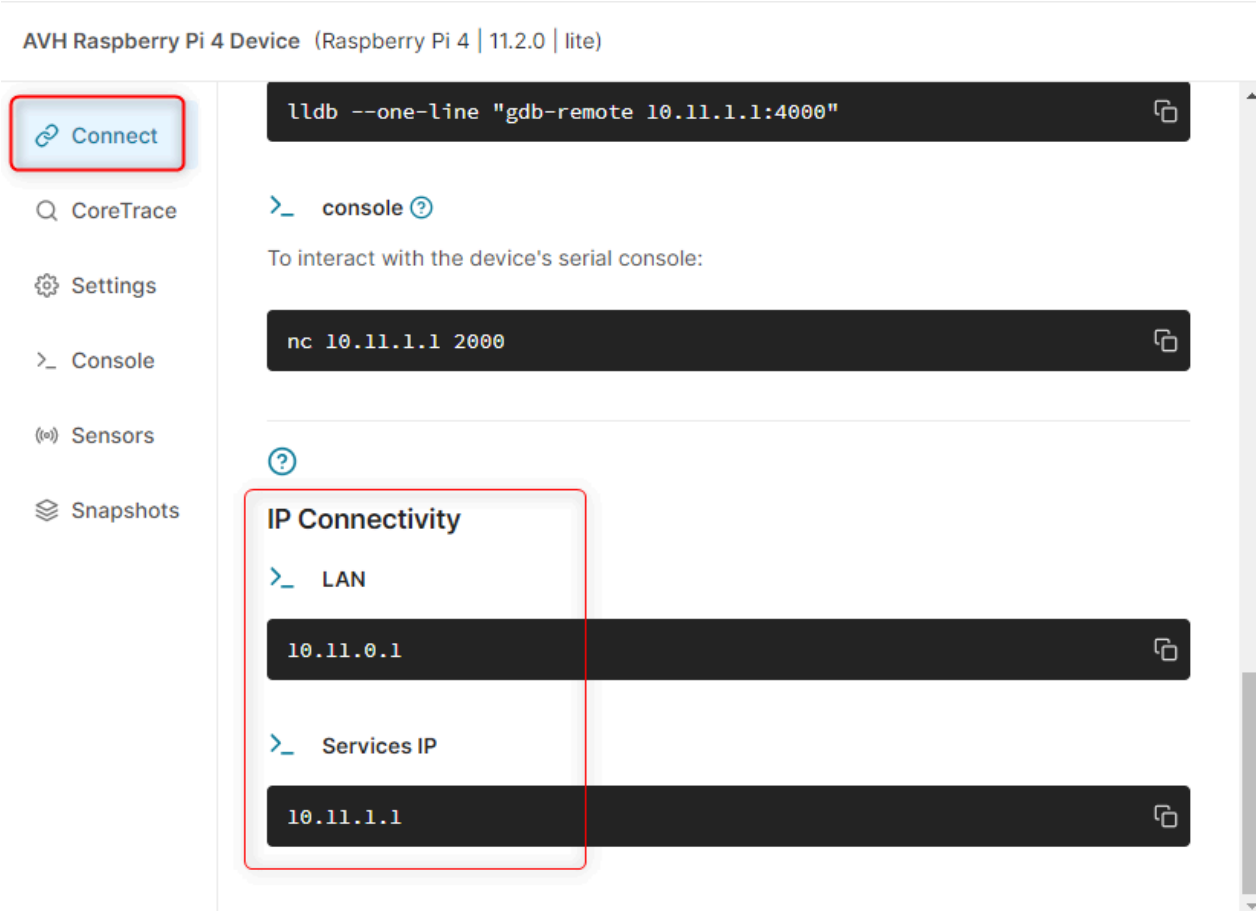
Next try connecting to your local machine with SSH with the Connect via SSH tutorial.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.5 Services IP and LAN IP

Service IP and LAN IP can be found on the **Connect** tab of a device.

Figure 6-21: LAN IP and Services IP



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.

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.

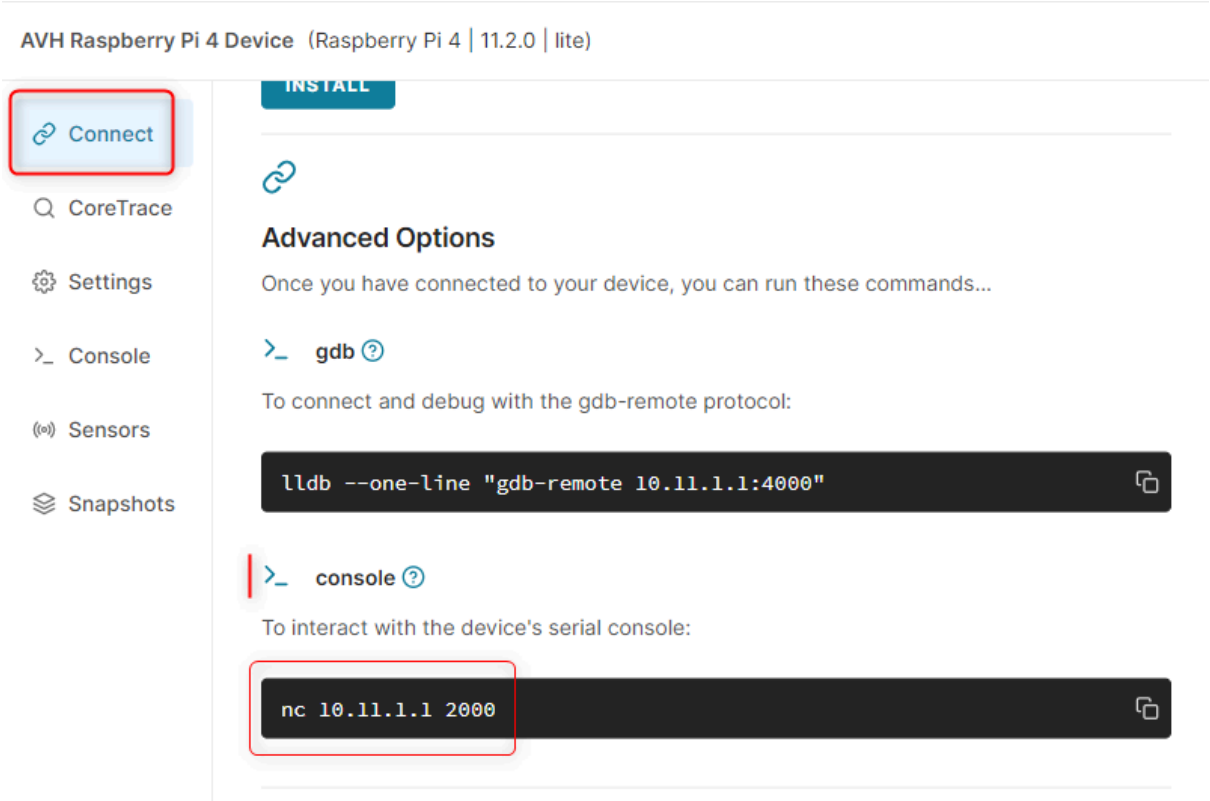
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.6 Connect to Device Console

You can connect to device serial console remotely from your local console program, such as Terminal. This can be convenient for enabling advanced console features, such as logging and search.

1. If you haven't created a virtual device yet, create one by following the instructions in the Getting Started tutorials.
2. From your computer machine establish a secure connection to the AVH service, either by using Quick Connect feature, or use VPN as explained in [Connect to the VPN].
3. On your device page, click on **Connect** tab and copy the `console command: nc <device IP> <port>` in the example below it is `nc 10.11.1.1 2000`

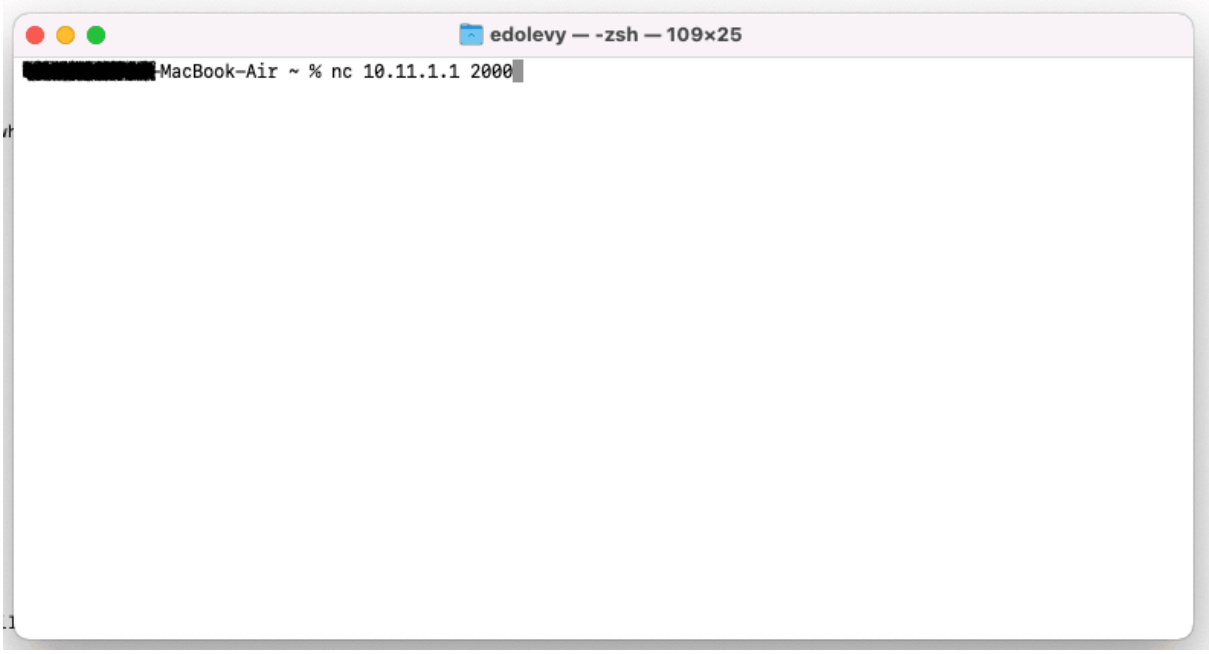
   Figure 6-22: Connect



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

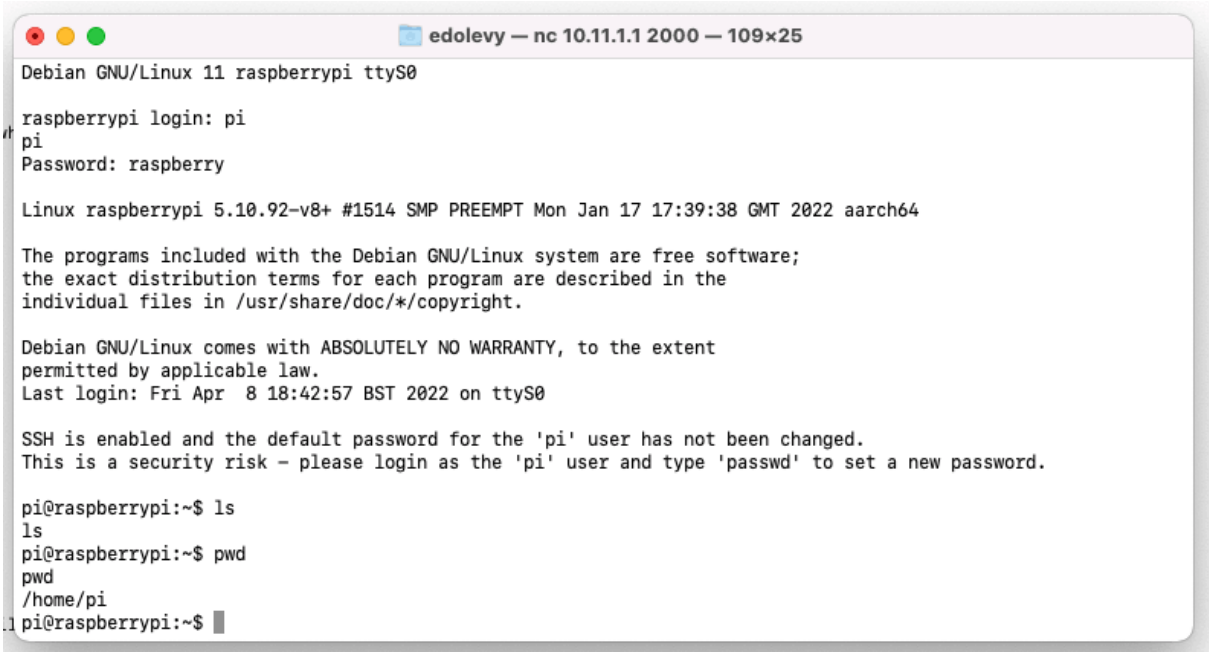
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-23: Paste the command



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

Figure 6-24: Console



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.7 Connect to Device TCP/UDP Ports

You can connect to TCP/IP ports that are exposed on the device by its firmware. For that the LAN IP address of the target device shall be used.

Such connections are possible remotely from an external machine and in such case a secure connection to AVH is required as explained in Remote Connections to Devices. But you can also connect from other virtual devices that are always located in the same LAN, as described in Connections between Devices.

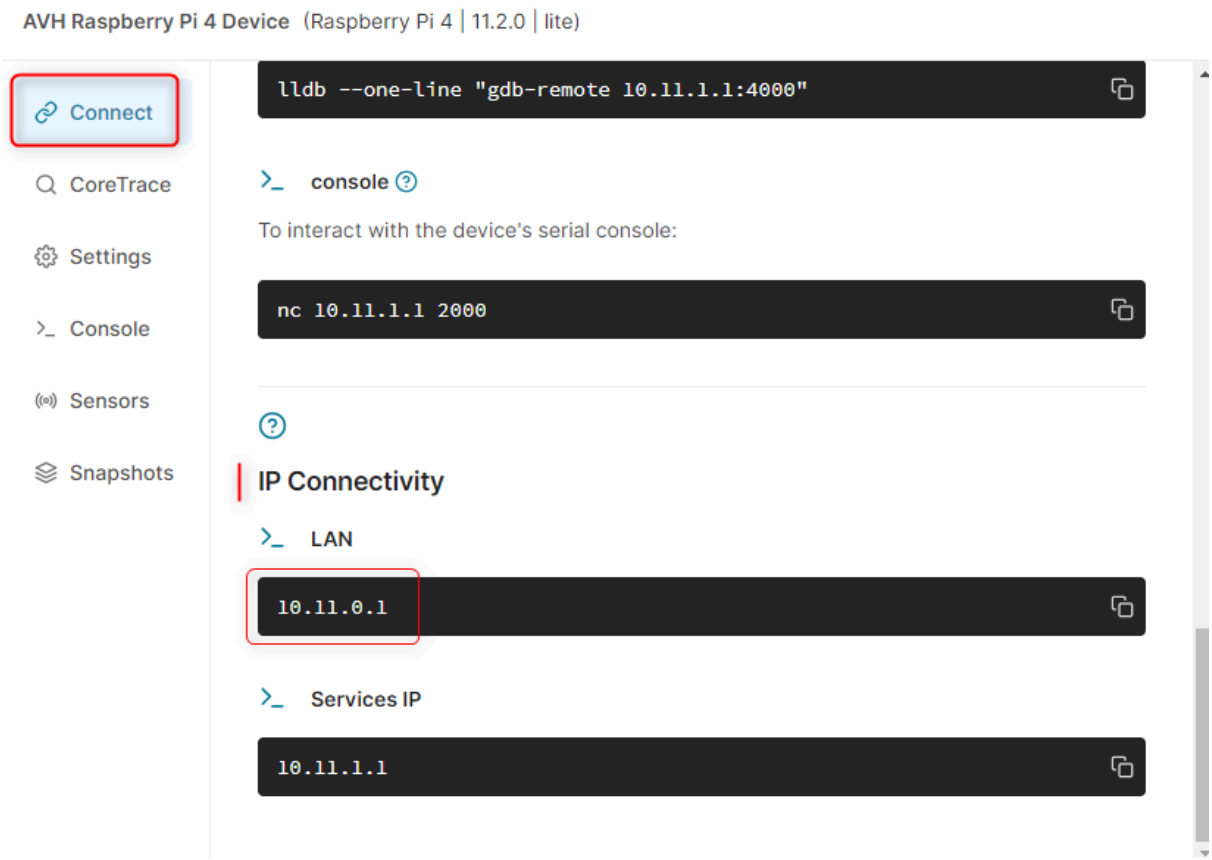
For a virtual device running Linux a simple example could be to Connect via SSH to access remotely Linux console on the device. For a microcontroller device an example could be to Connect to a Web Server hosted on the device. Connect USB devices via USB/IP shows a more complex use case for sharing USB peripherals over IP.

Connect via 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

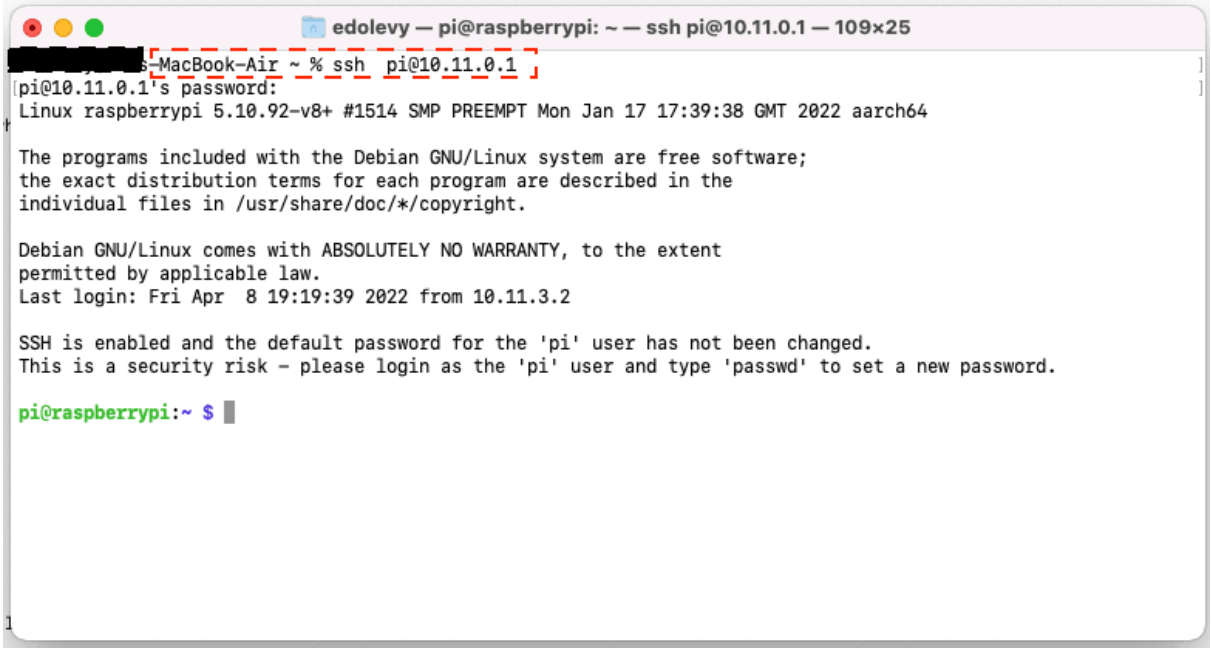
Figure 6-25: LAN IP address



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`

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-26: Connect to SSH



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

Connect to a Web Server

Another example of remote communication with TCP/IP ports of a virtual device could be access to a web server hosted on the device. IoT HTTP Web Server on STM32U5 demonstrates such an example.

6.8 GDB Connection

A GDB server stub is available on virtual devices enabling you to debug your program using any GDB-capable debug client, for instance lldb and IDA debugger.

The GDB stub for a particular virtual device is accessible via the Services IP address of the device and port `4000` (see Services IP and LAN IP). The **Connect** tab on your device's AVH page contains **gdb** section with an lldb command that can be used to establish a debugging session. For example:

```
lldb --one-line "gdb-remote 10.11.1.2:4000"
```

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.

See Kernel Debugging Using GDB/LLDB for an example demonstrating kernel debug on your virtual machine.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.9 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

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.

The CoreIO Library API

The CoreIO library allows external programs to act as MMIO responders, send interrupt requests to the VM, and perform DMA.

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.
```

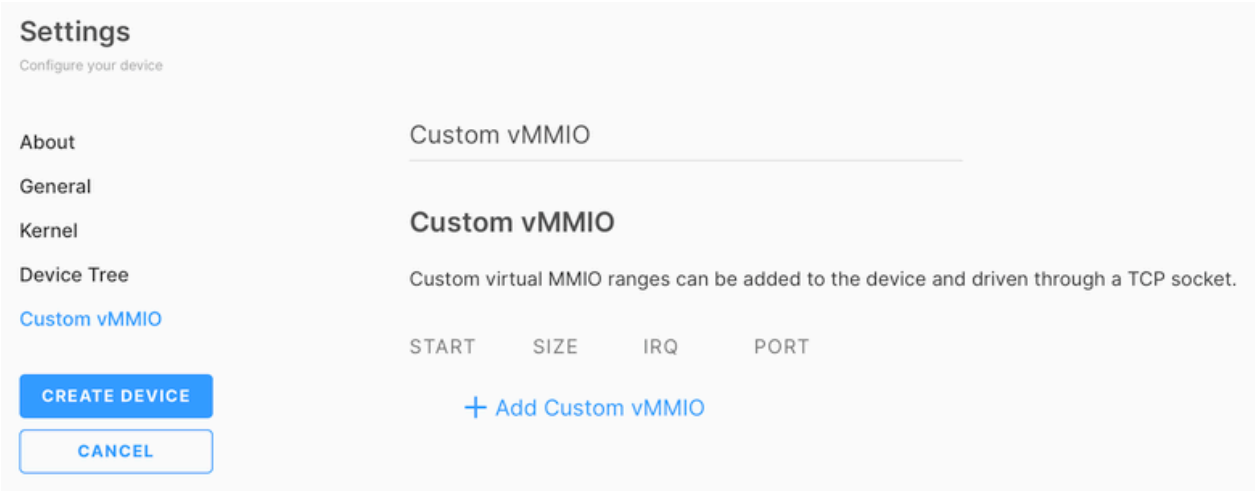
If the result was zero, you can continue using the CoreIO API.

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`.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-27: 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).

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:

```
//add defines here
    typedef struct {
        /* MMIO operations. */
        int (*read)(void *priv, uint64_t addr, size_t len, void *buf, unsigned
 flags);
        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;
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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.
```

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;
    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 */
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
            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.
```

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)
```

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
      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
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
      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);
```

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

[vMMIO Example on Raspberry Pi 4]

6.10 Connect USB devices via USB/IP

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

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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.

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.

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.

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.

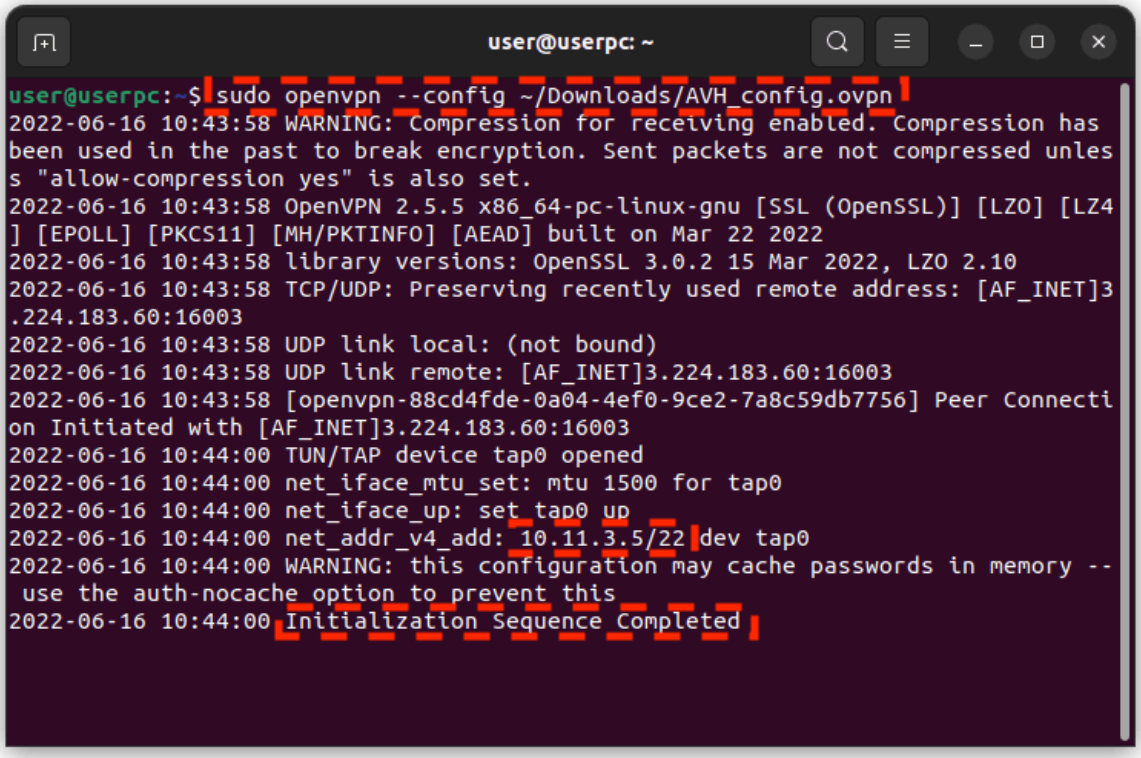
```
sudo apt update && sudo apt install -y linux-tools-$(uname -r) hwdata ubuntu-restricted-extras
```

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-28: Connect to AVH VPN

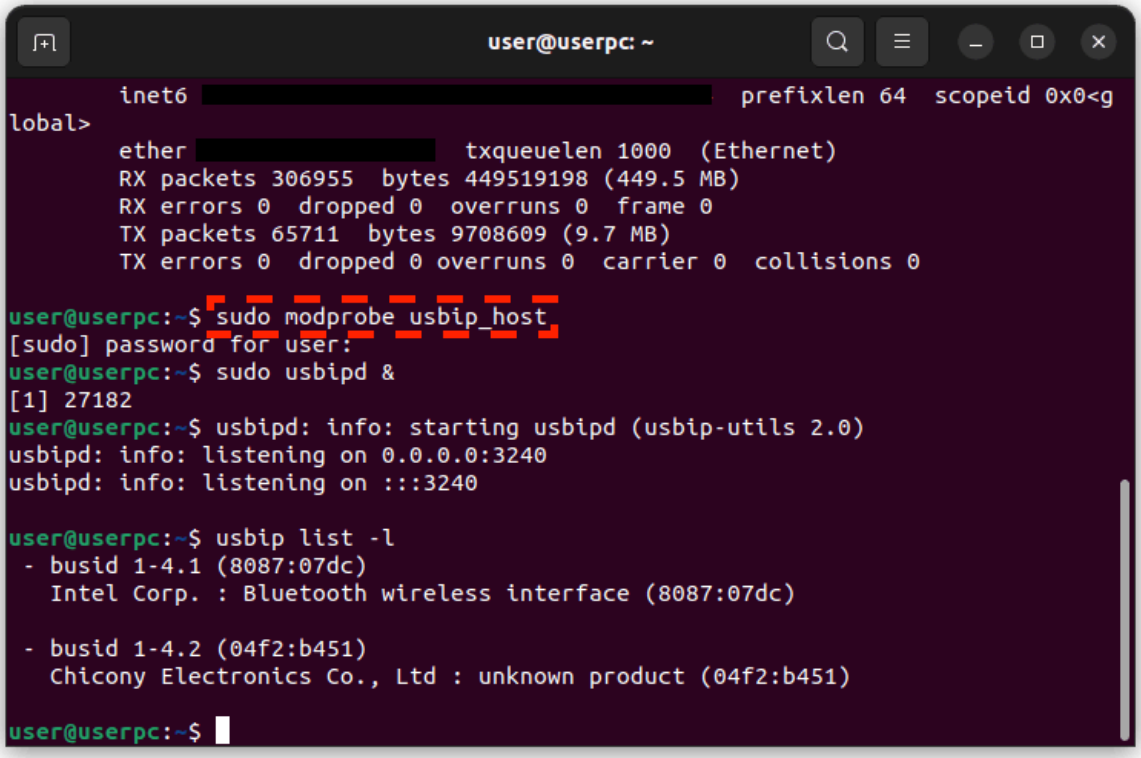


3. Open a new Terminal window and load the USB/IP Host kernel module.

```
sudo modprobe usbip_host
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-29: Load the USBIP host kernel package

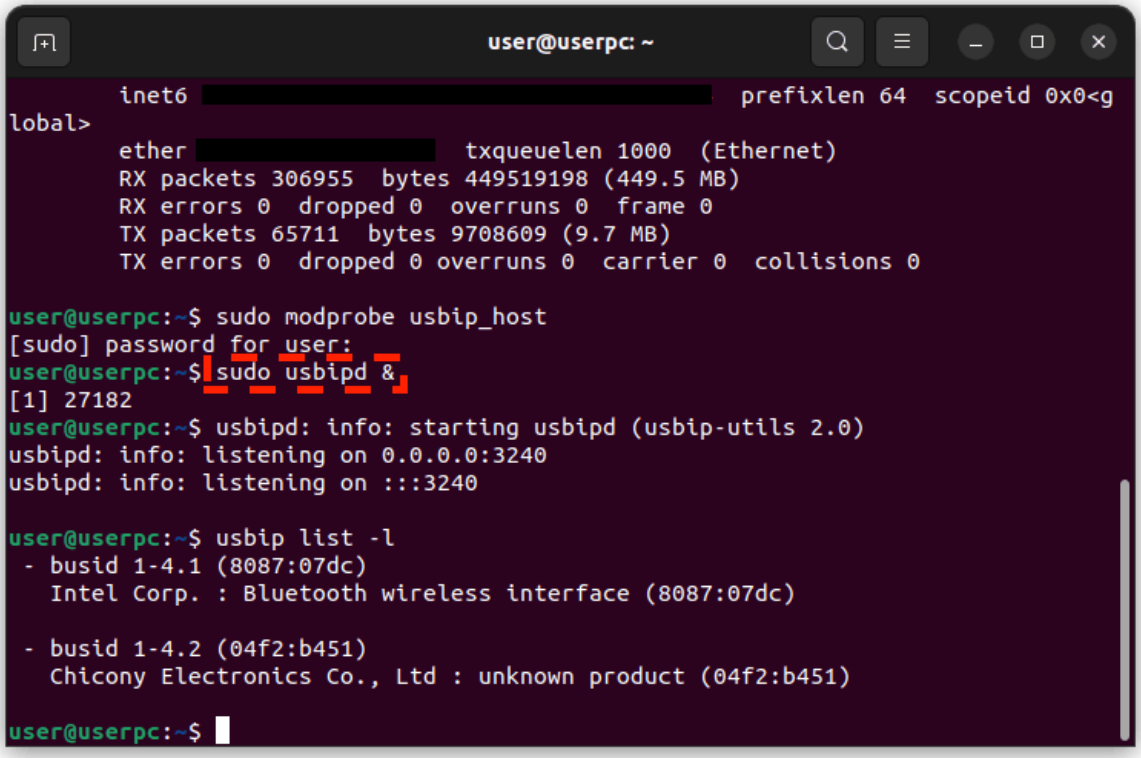


4. Start the USB/IP daemon in the background.

```
sudo usbipd &
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-30: Run the USBIP daemon



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

```
usbip list -l
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

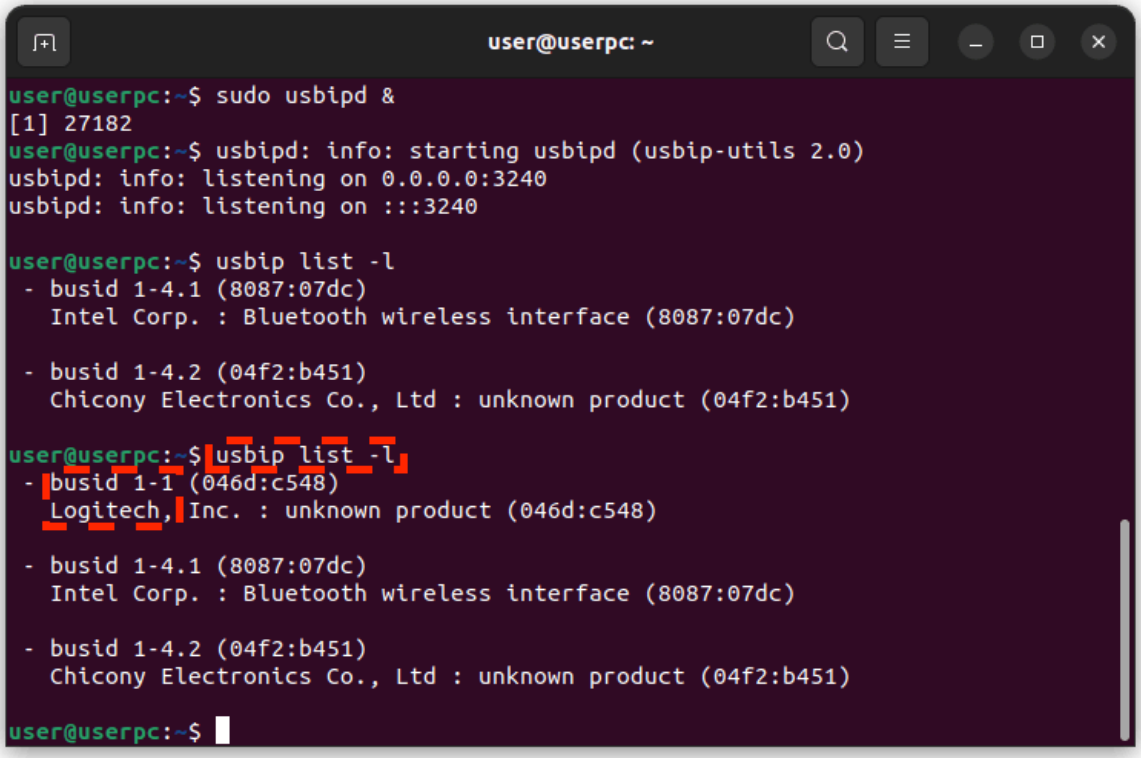
Figure 6-31: USBIP list without device

```
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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-32: USBIP list with device

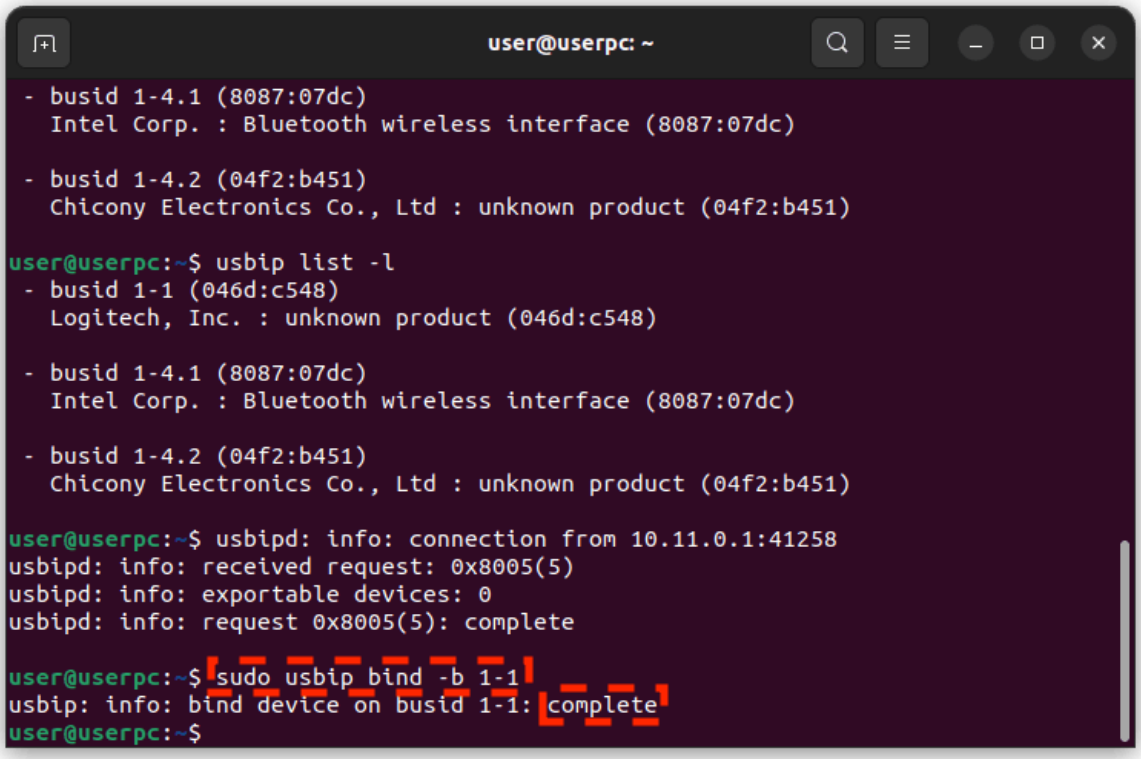


7. Bind the appropriate USB bus, `1-1` in our case. (This will cause the device to stop functioning on your host computer, so make sure to have a backup in the case of a mouse.)

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-33: USBIP bind



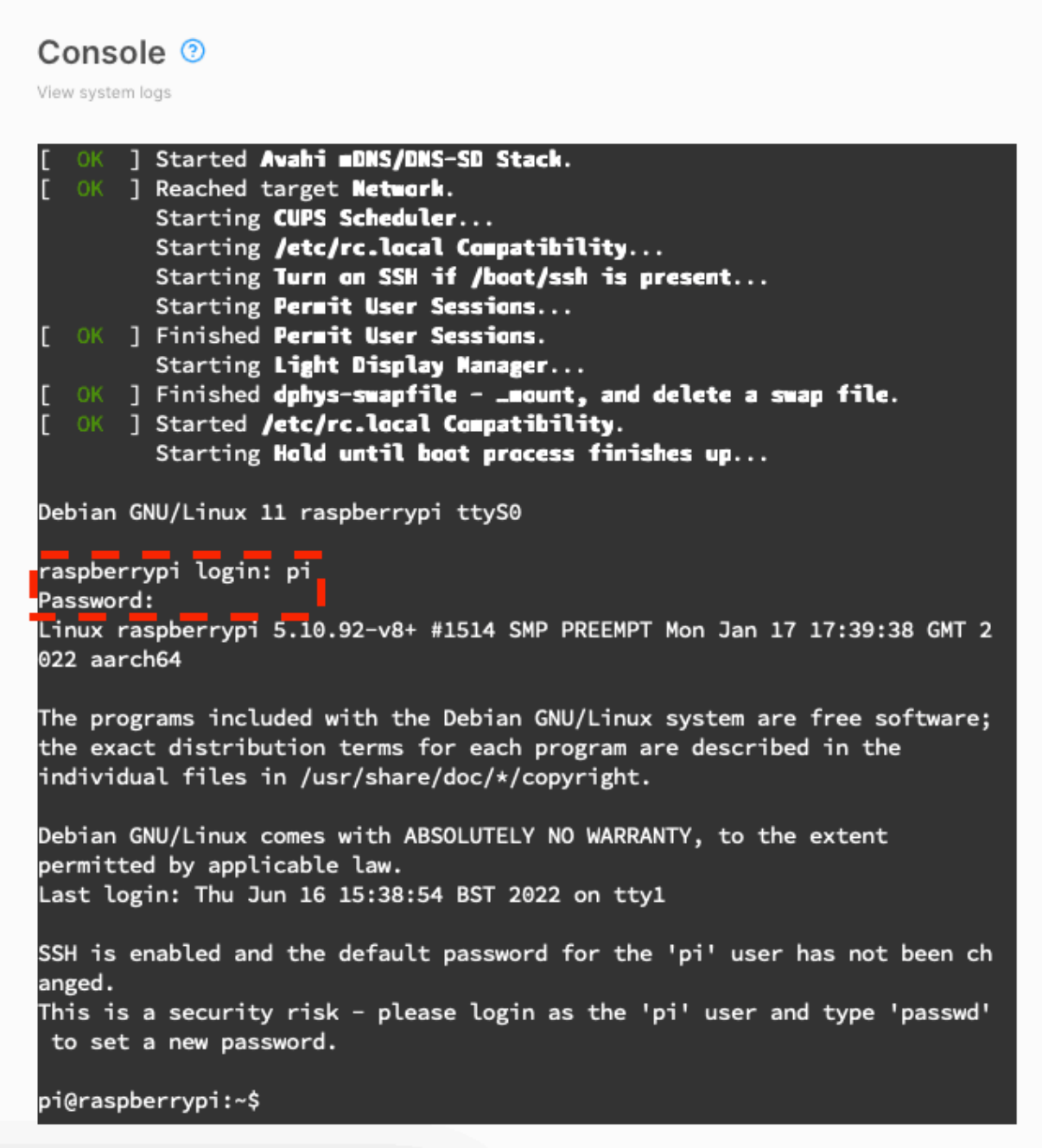
Attach a Physical Device to the Virtual Board Client

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-34: Log in to RPi 4 Desktop



2. Install the USB/IP package.

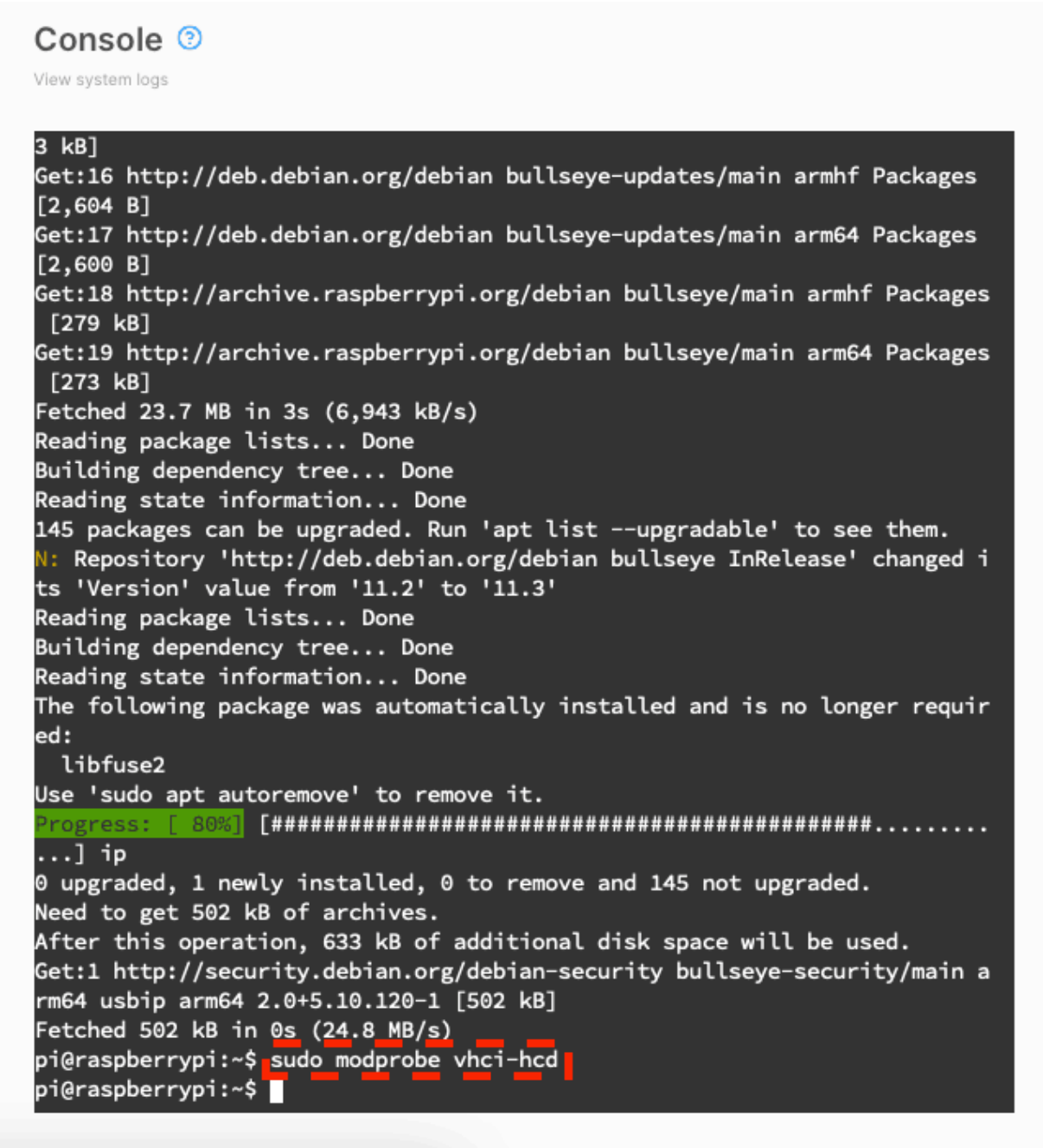
```
sudo apt update && sudo install usbip
```

3. Load the Virtual Host Controller Interface kernel module.

```
sudo modprobe vhci-hcd
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-35: Load the VHCI kernel module

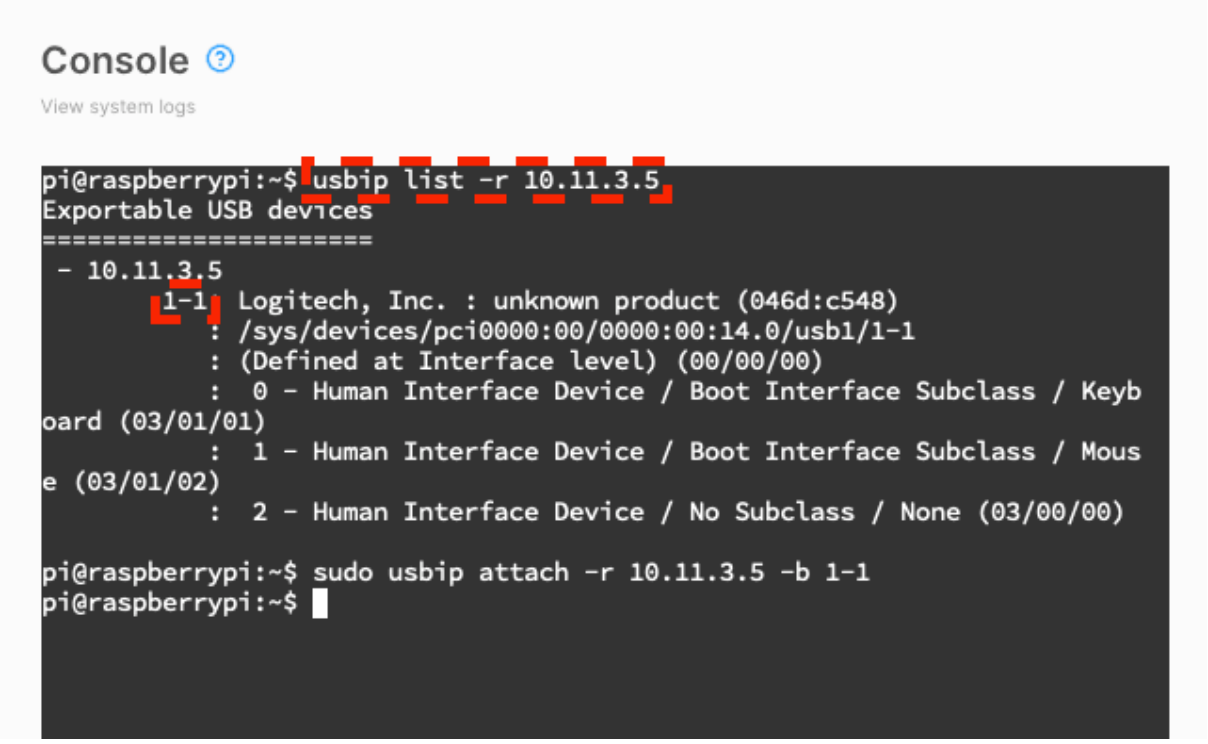


4. Using the VPN IP address from openvpn, list the exportable USB devices on the host. Look for the same bus number from the previous section.

```
usbip list -r 10.11.3.5
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-36: USBIP list on RPi4 Desktop

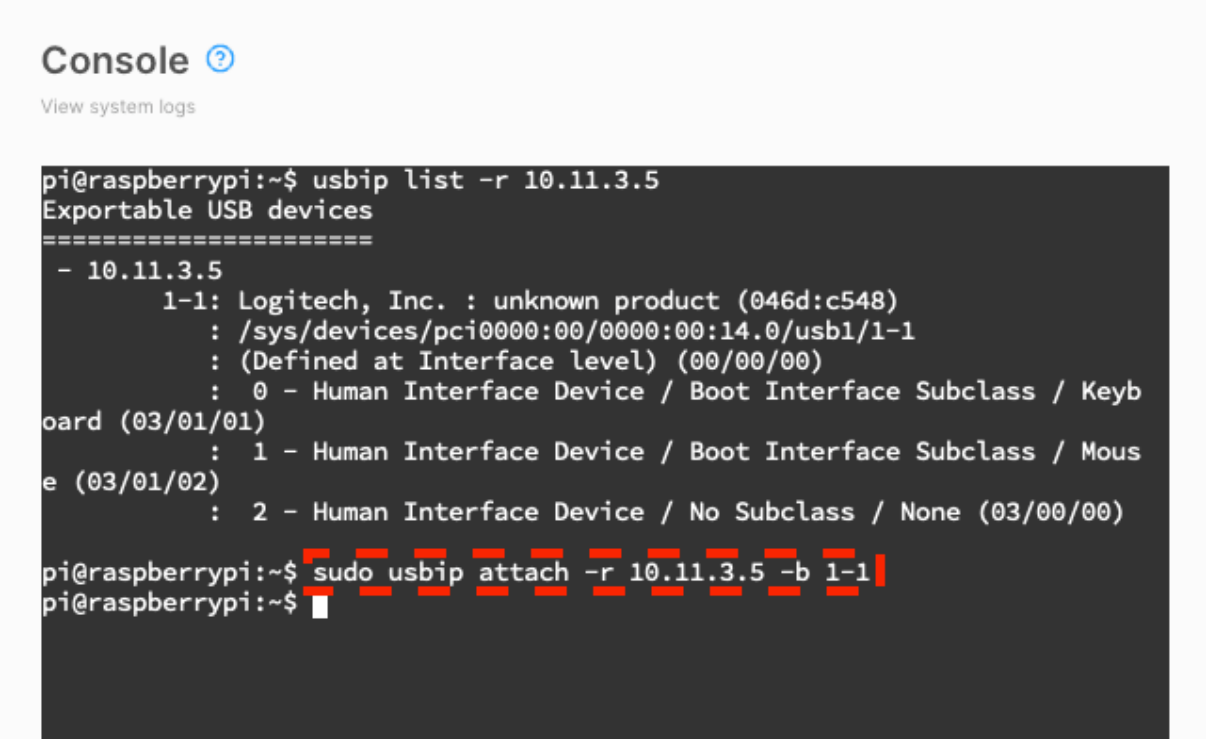


5. Attach to USB bus `1-1`, which corresponds to our mouse.

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-37: USBIP attach on RPi4 Desktop



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

(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.

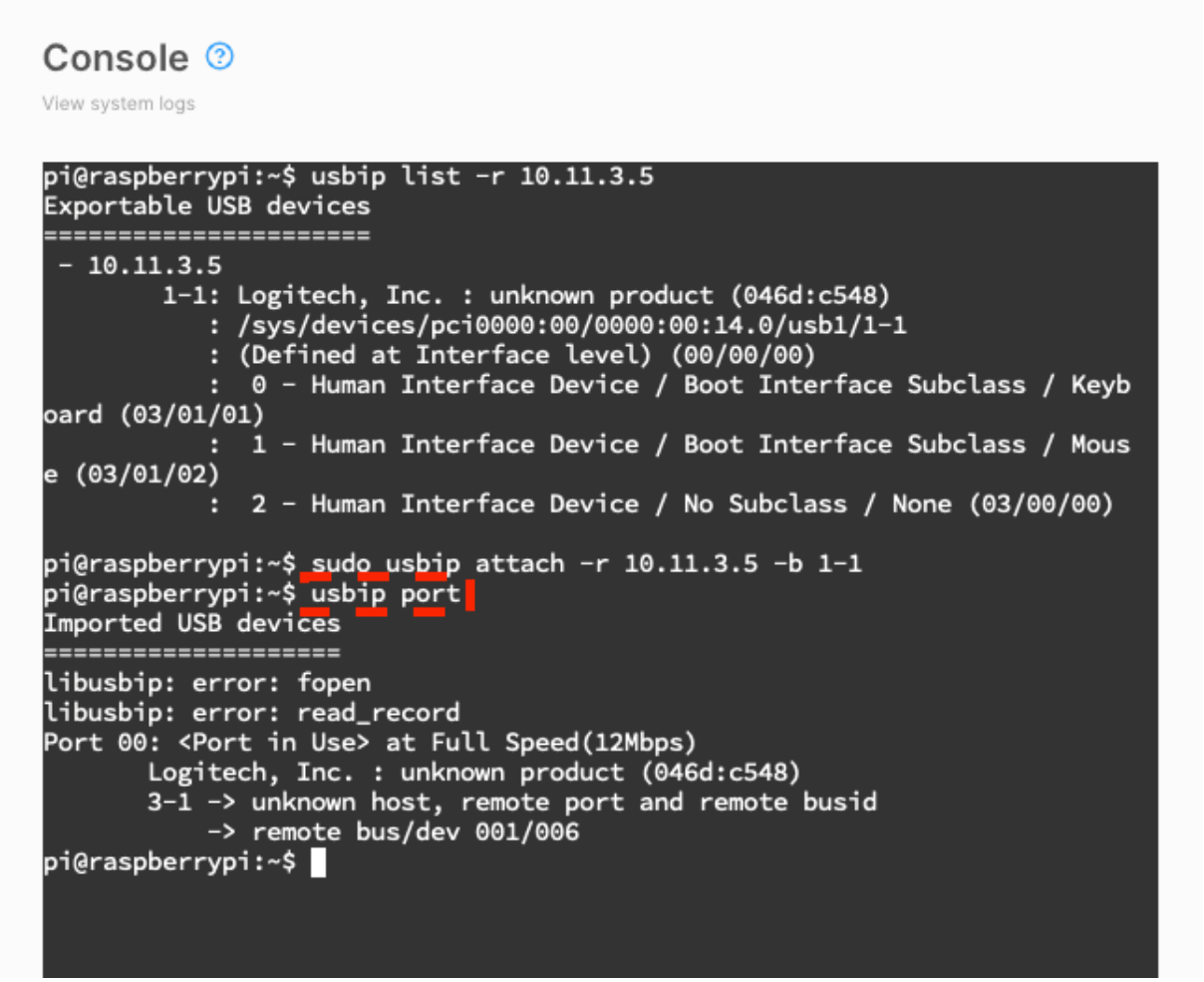
This process is necessary if you want to access the USB resource again on your host device.

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-38: USBIP port on RPi4 Desktop

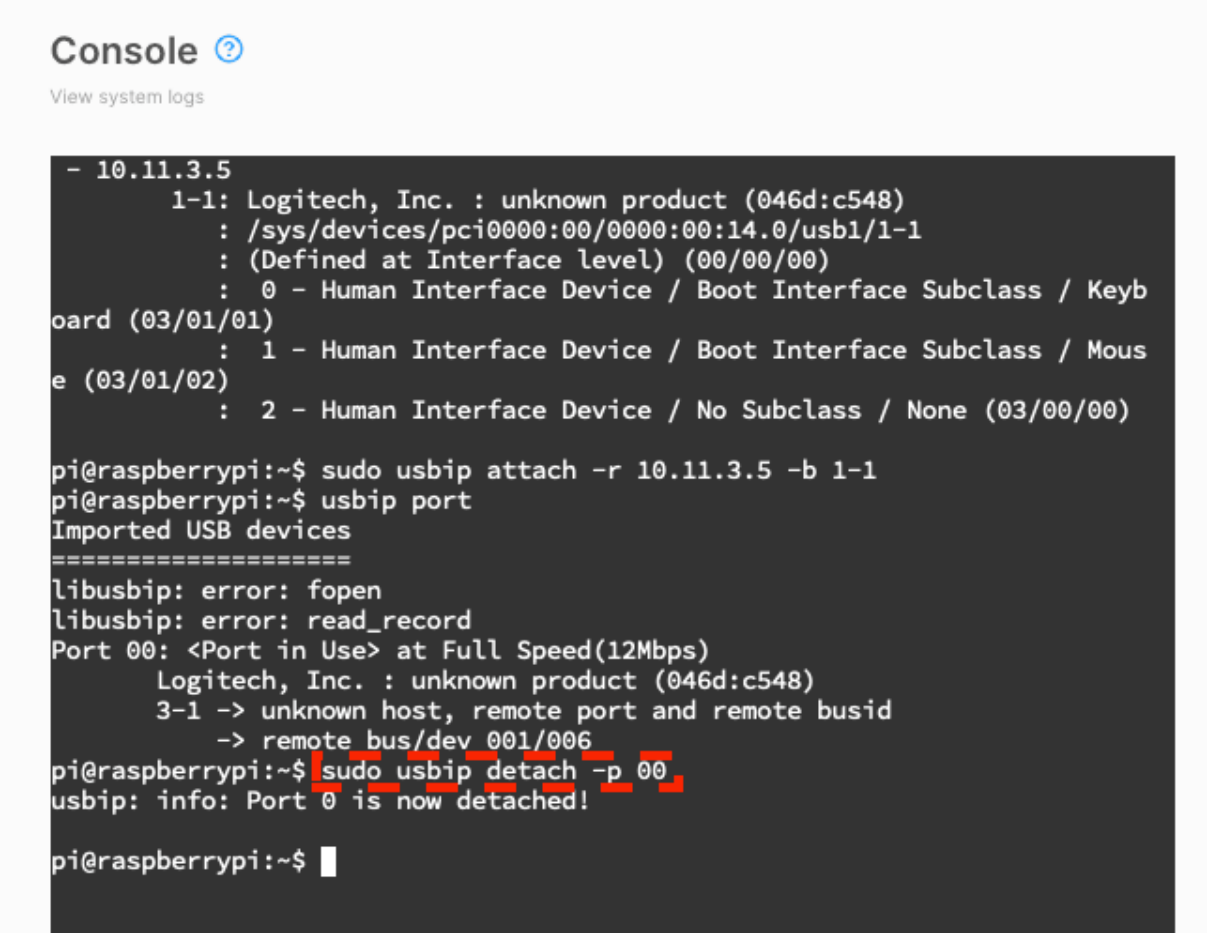


2. Detach port `00`, which corresponds to our mouse. Look for a response showing `detached`.

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-39: USBIP detach on RPi4 Desktop

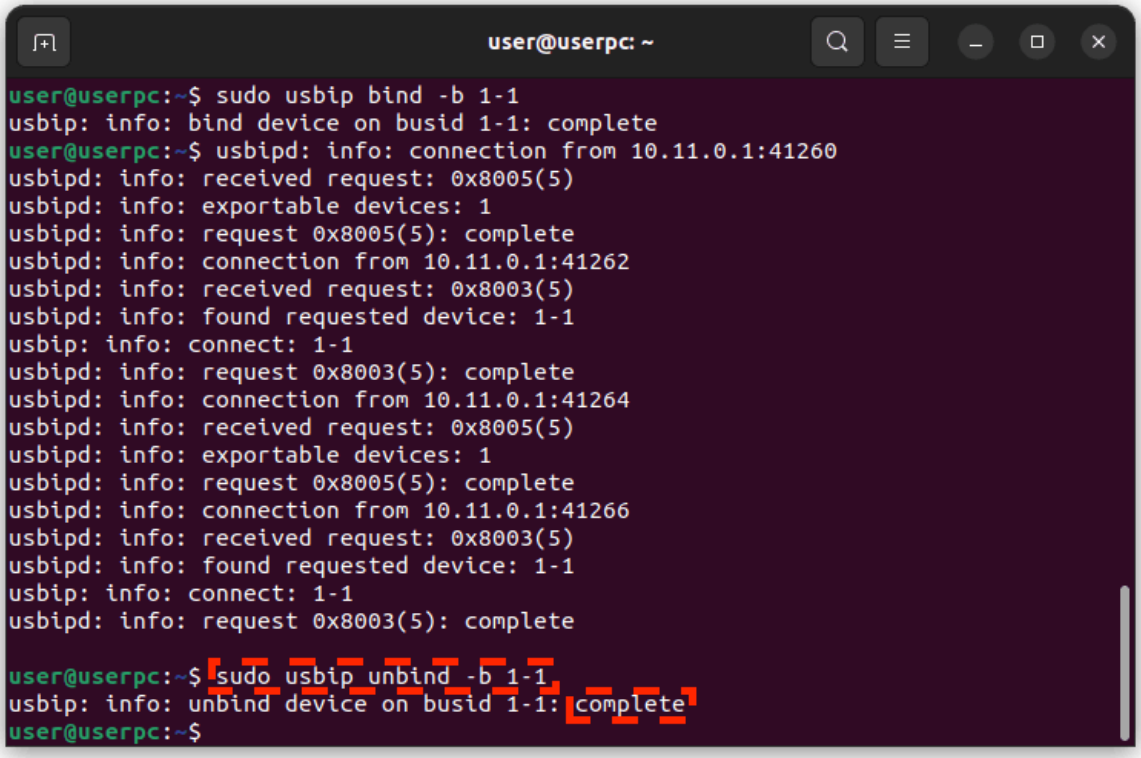


3. Switch over to your USB/IP host, which in our case is the Ubuntu machine, and unbind the appropriate USB bus. Look for a response showing `complete`.

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-40: USBIP unbind



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.

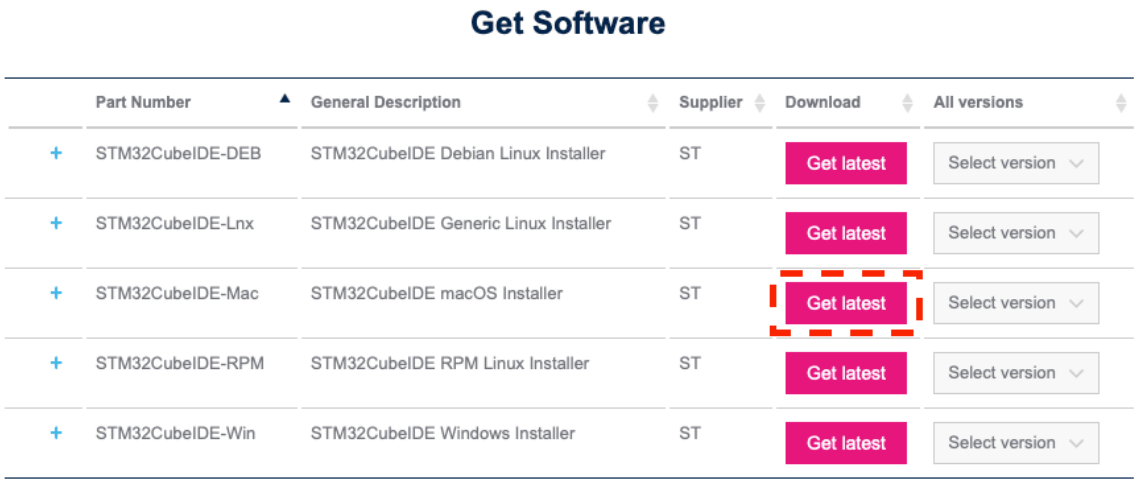
Build USB Firmware Files with the STM32CubeIDE Tool

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.

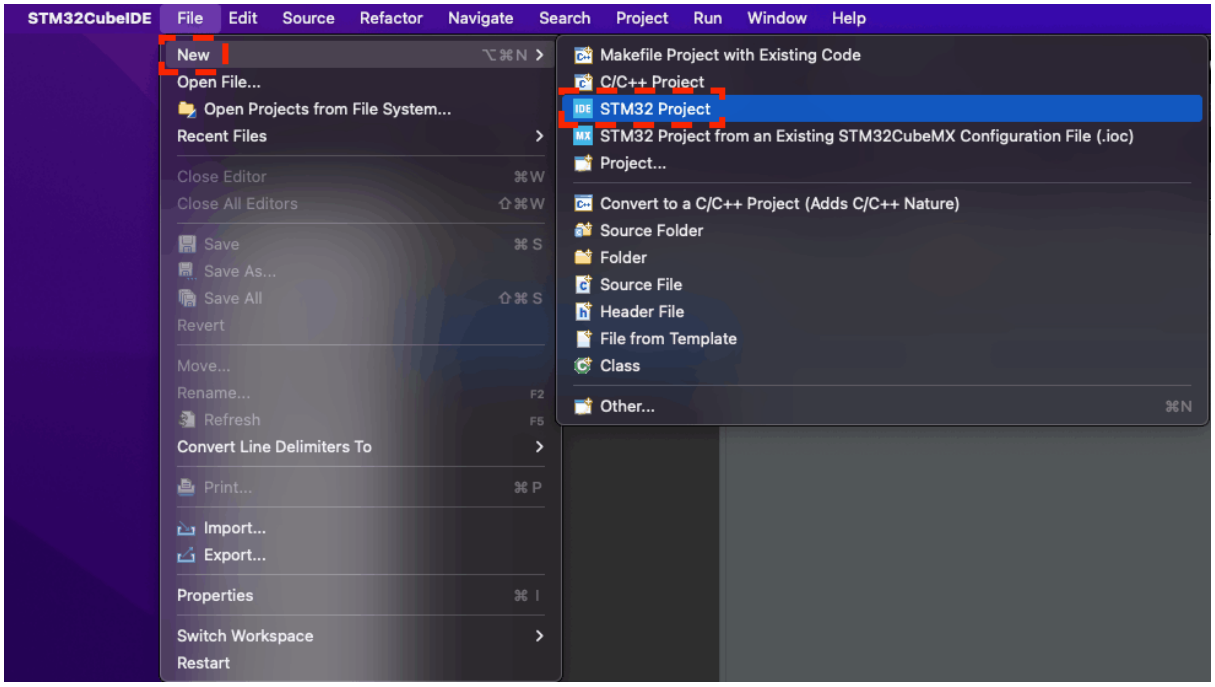
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-41: Download CubeIDE



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

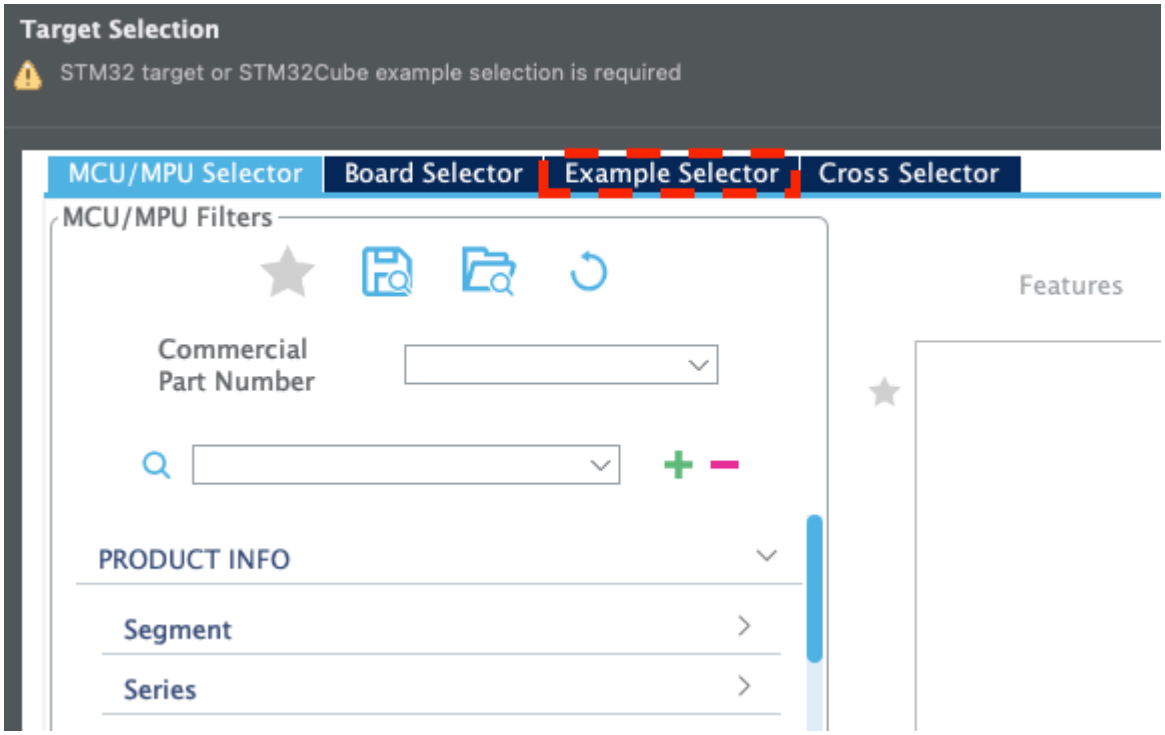
Figure 6-42: New Project



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

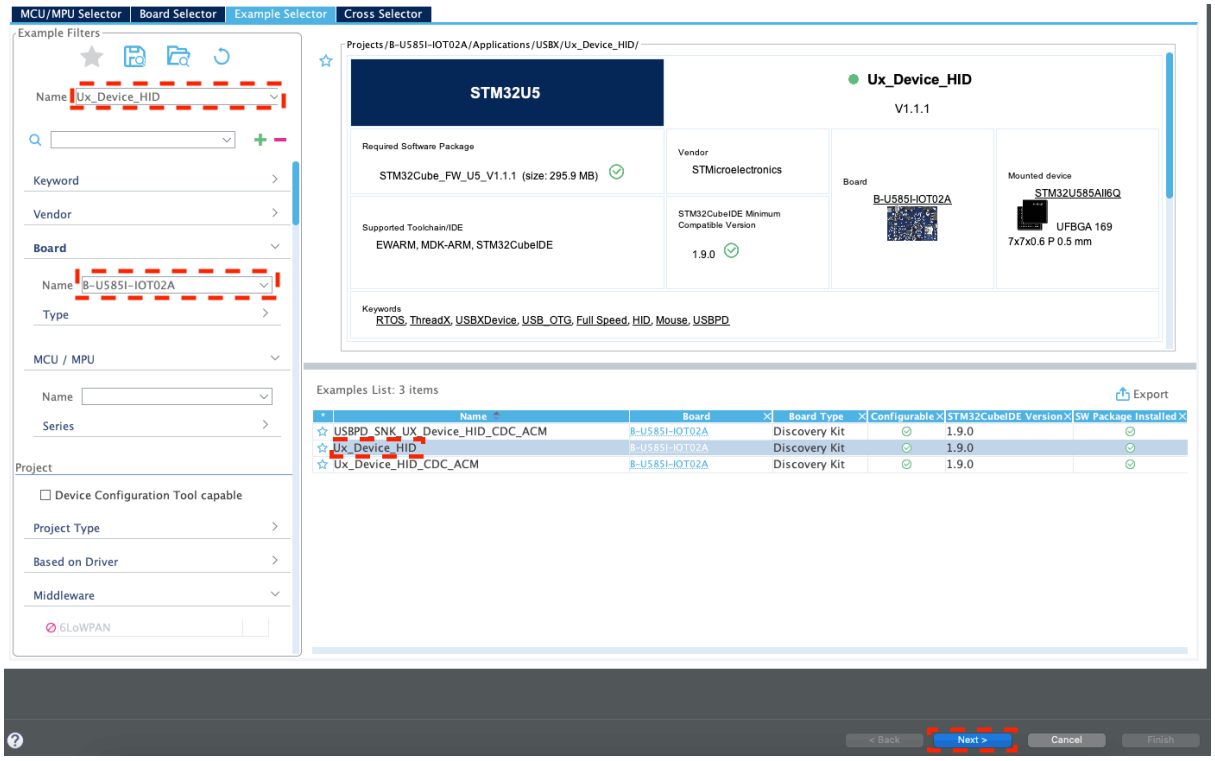
Figure 6-43: Example Selector



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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

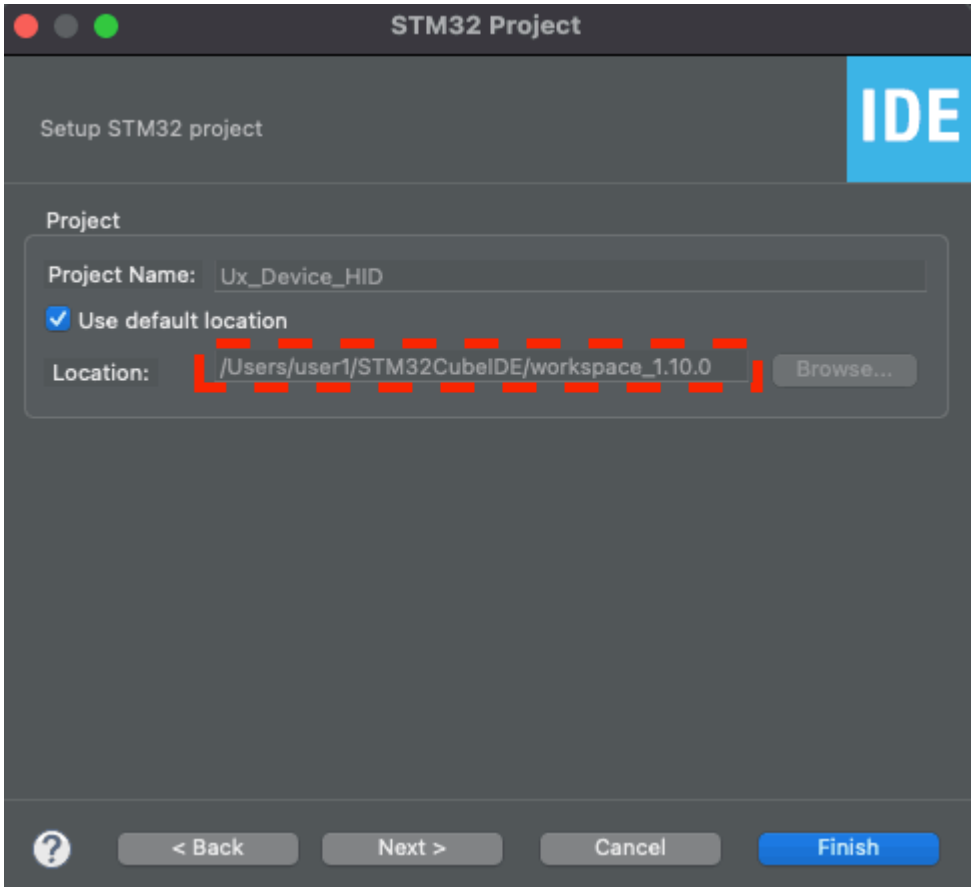
Figure 6-44: Choose Example



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

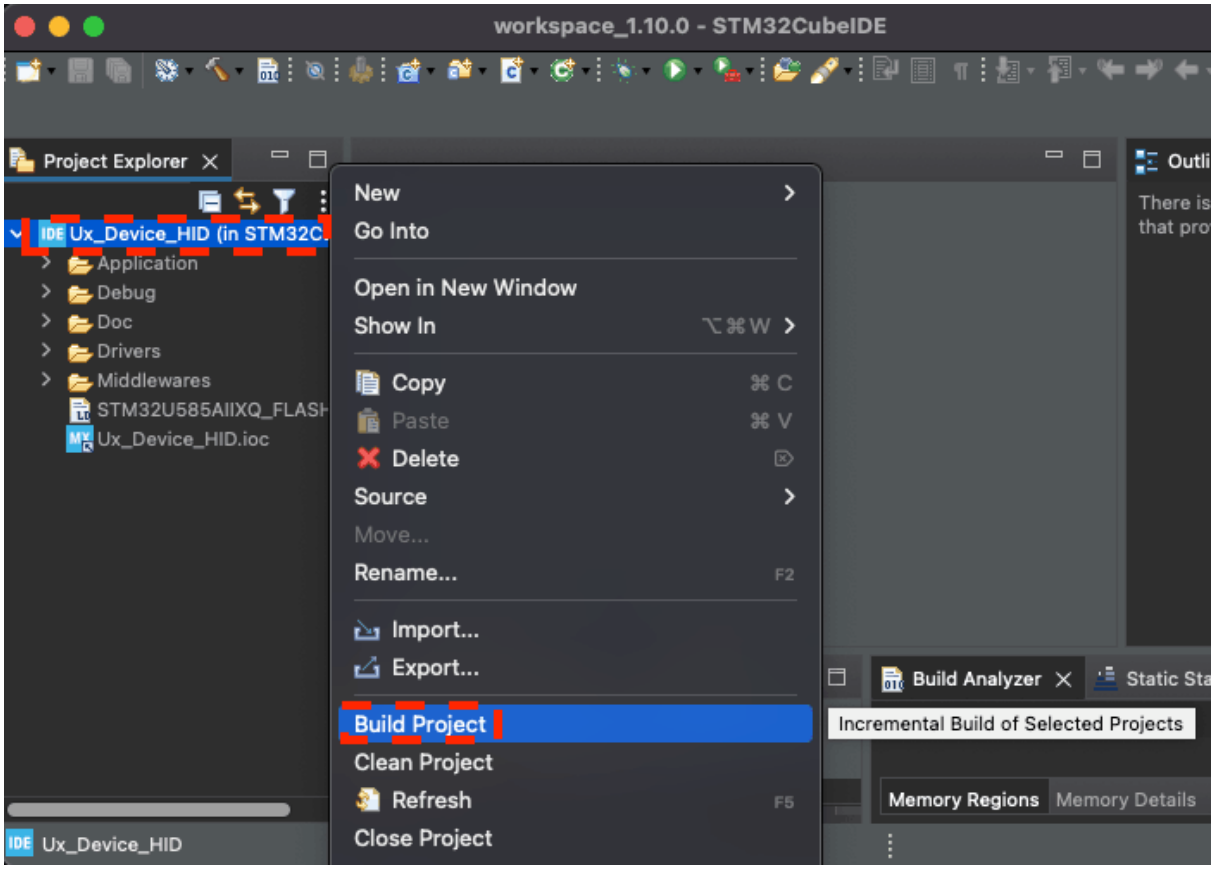
Figure 6-45: Finish Project



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

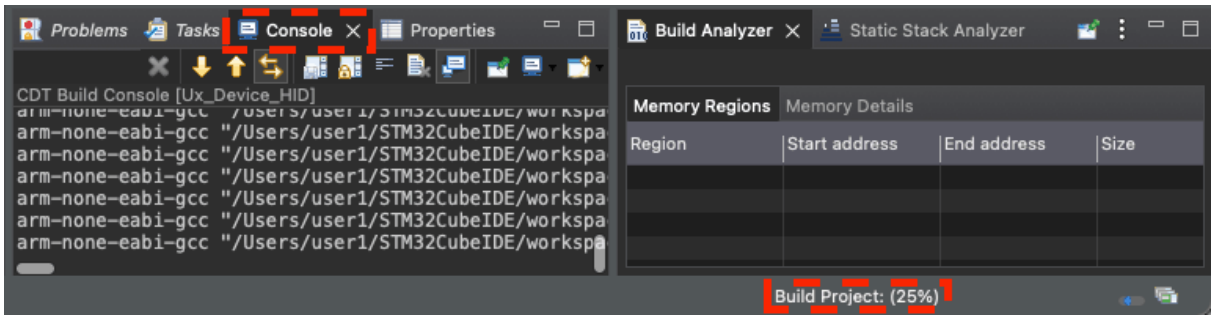
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-46: 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 6-47: Build Progress

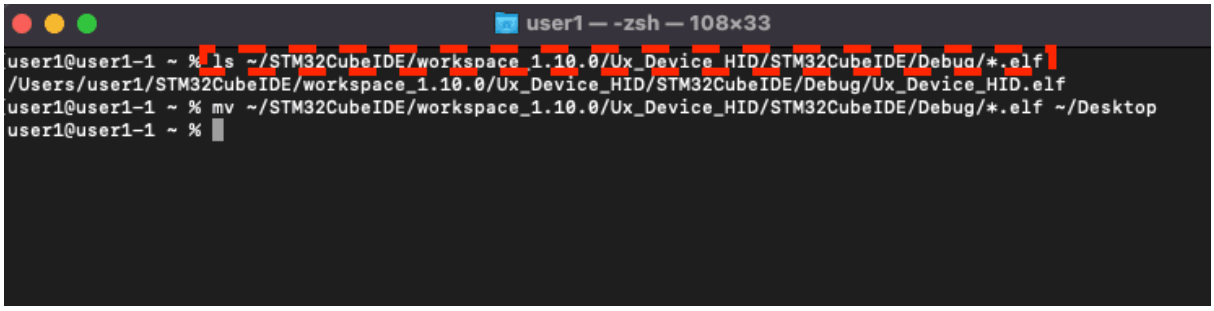


8. The `.elf` firmware file will be created inside your project directory, which is inside `/<path_to_workspace>/<project_name>/STM32CubeIDE/Debug`.

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-48: List Firmware File



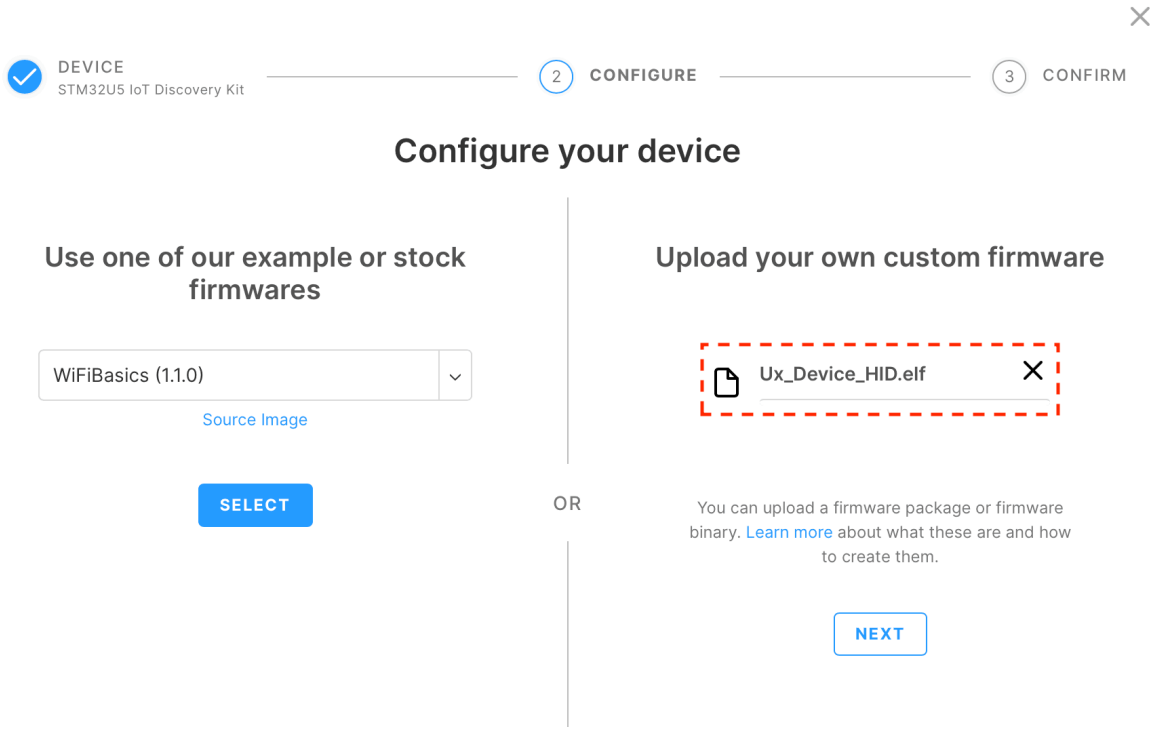
9. The firmware file is ready to be uploaded to your AVH virtual STM32U5 board.

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.

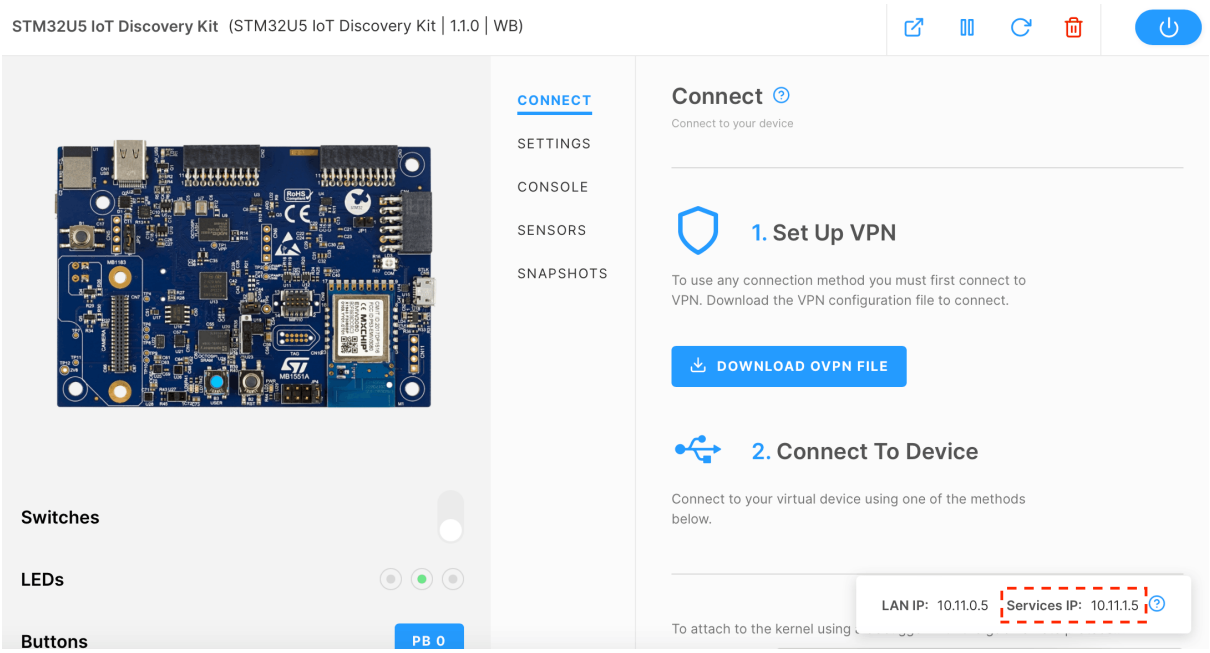
Figure 6-49: Upload joystick firmware



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`.

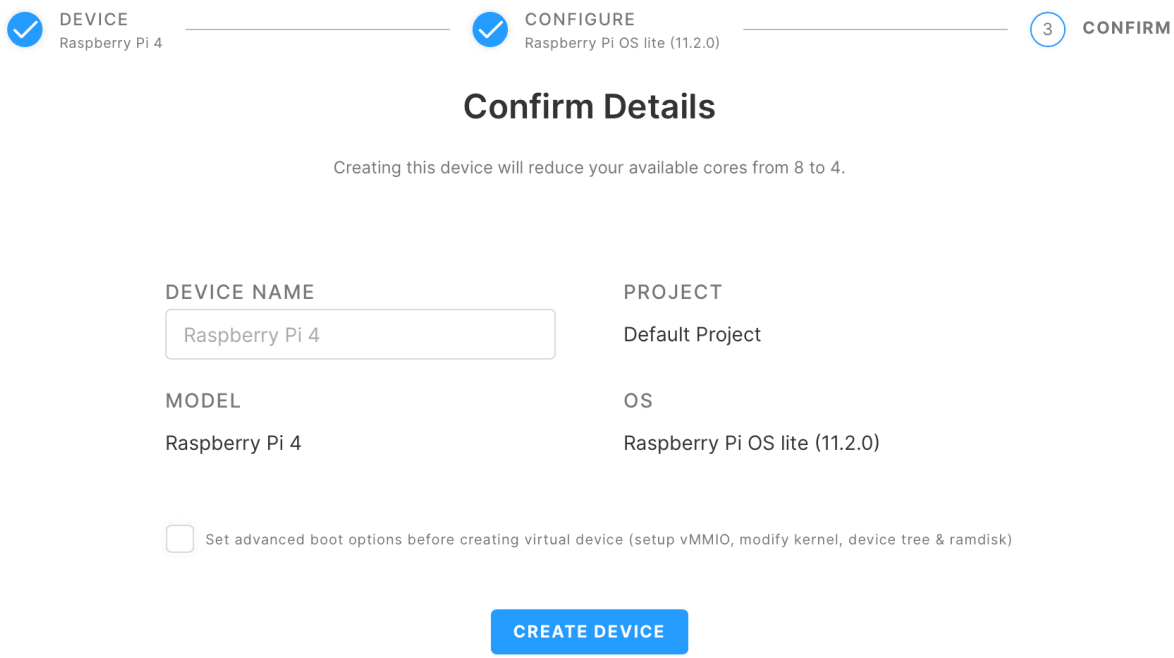
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-50: 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`.

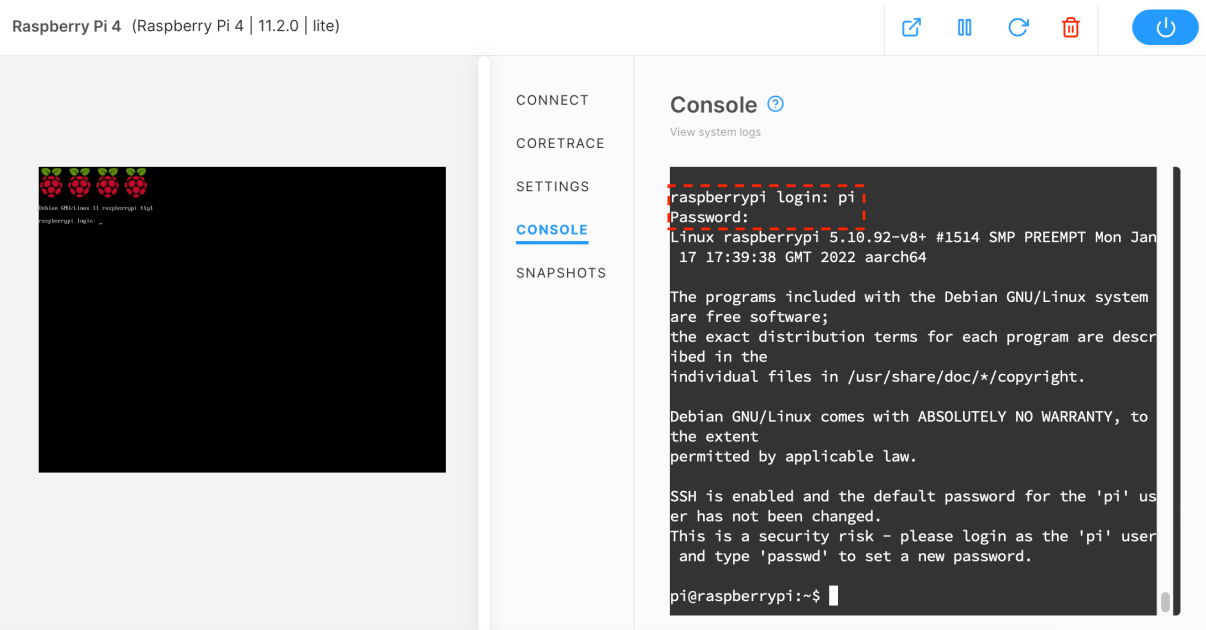
Figure 6-51: Create new Pi



4. Log in as the root user with the username `pi` and password `raspberry`.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-52: Login to Pi as root

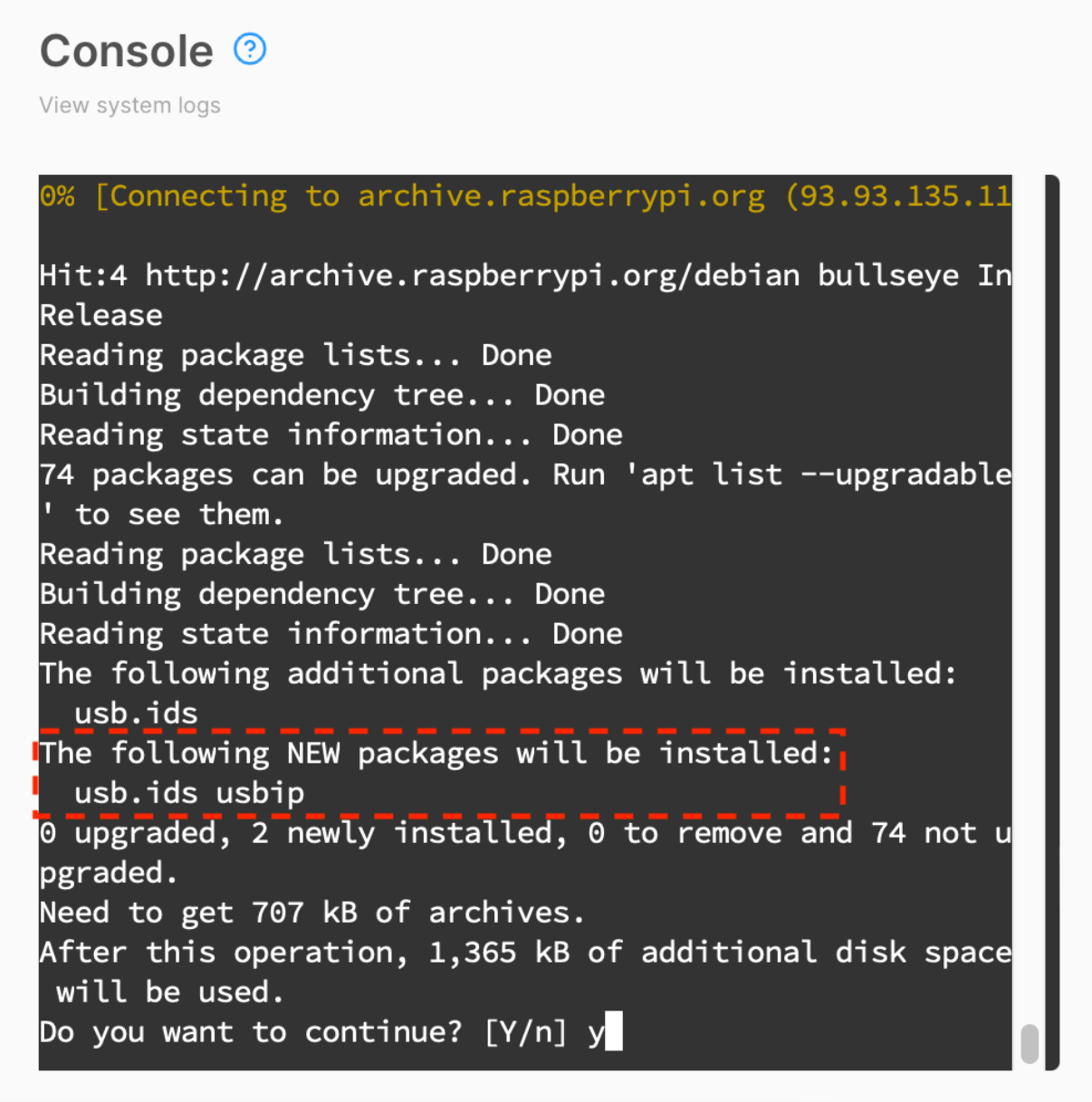


5. Install the USB/IP package.

```
sudo apt update && sudo apt install -y usbip
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-53: Install usbip

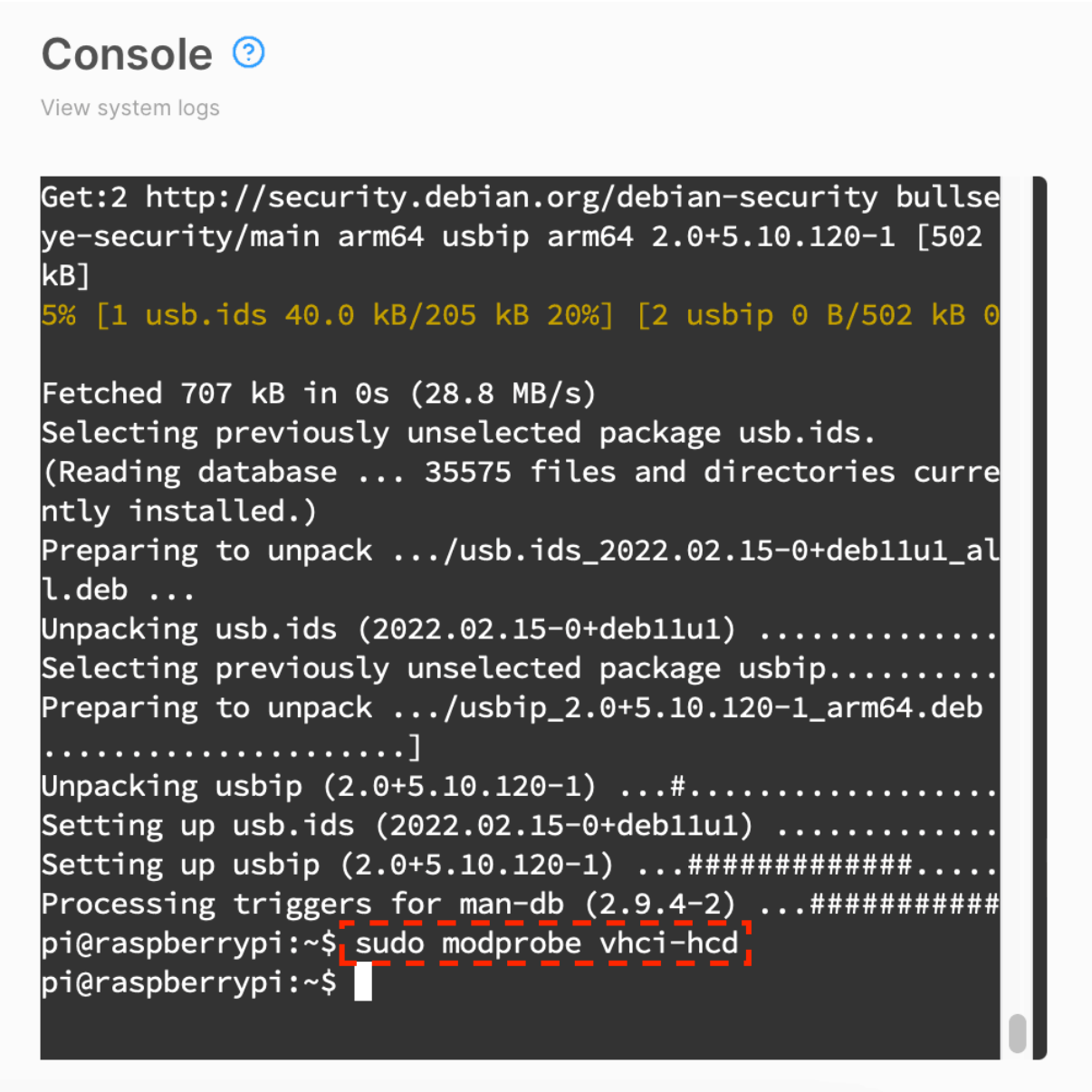


6. Load the Virtual Host Controller Interface kernel module.

```
sudo modprobe vhci-hcd
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-54: Load kernel module

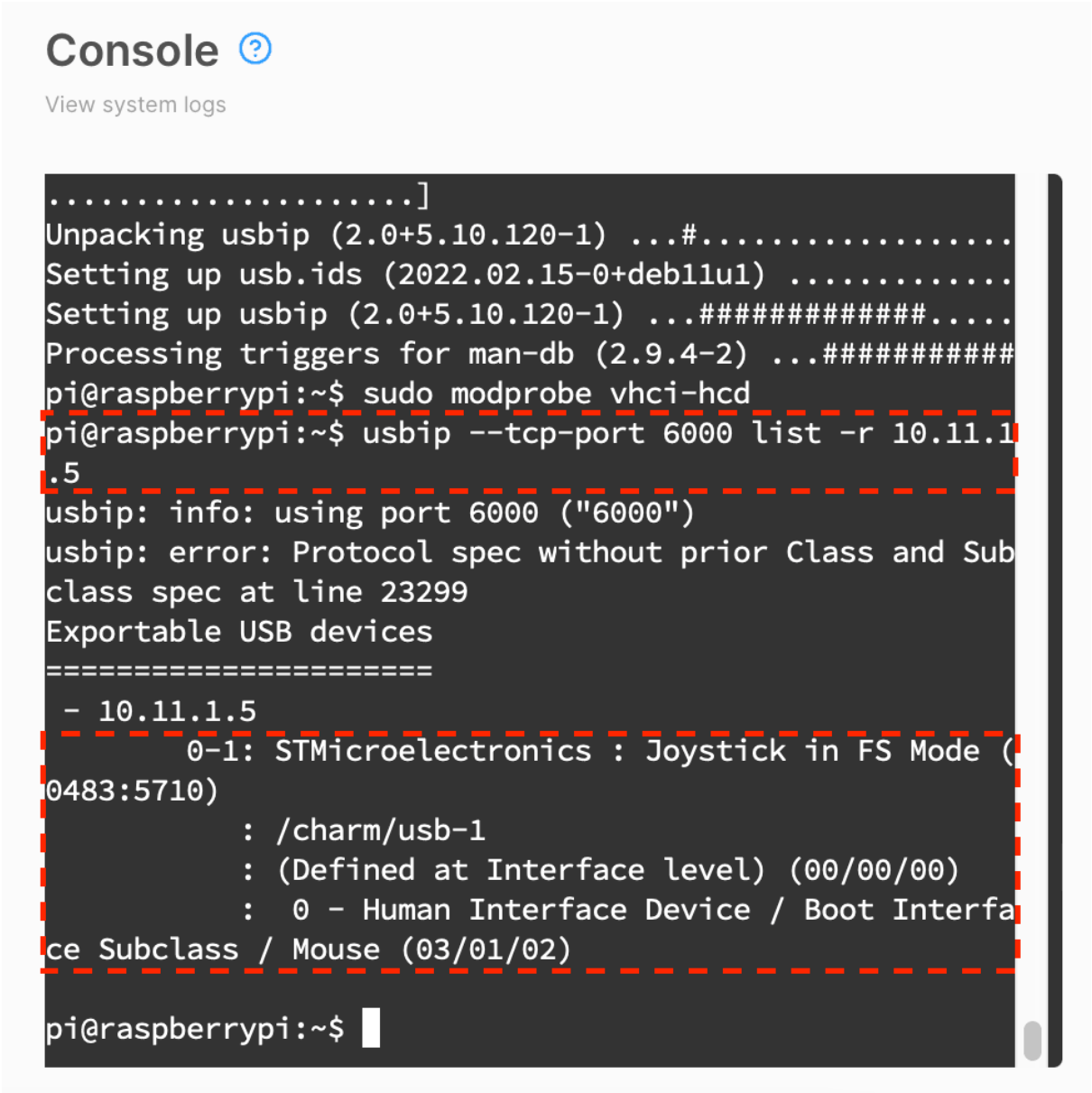


7. List the STM32U5's exportable USB devices using the Services IP and port 6000.

```
usbip --tcp-port 6000 list -r 10.11.1.5
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-55: List USB devices

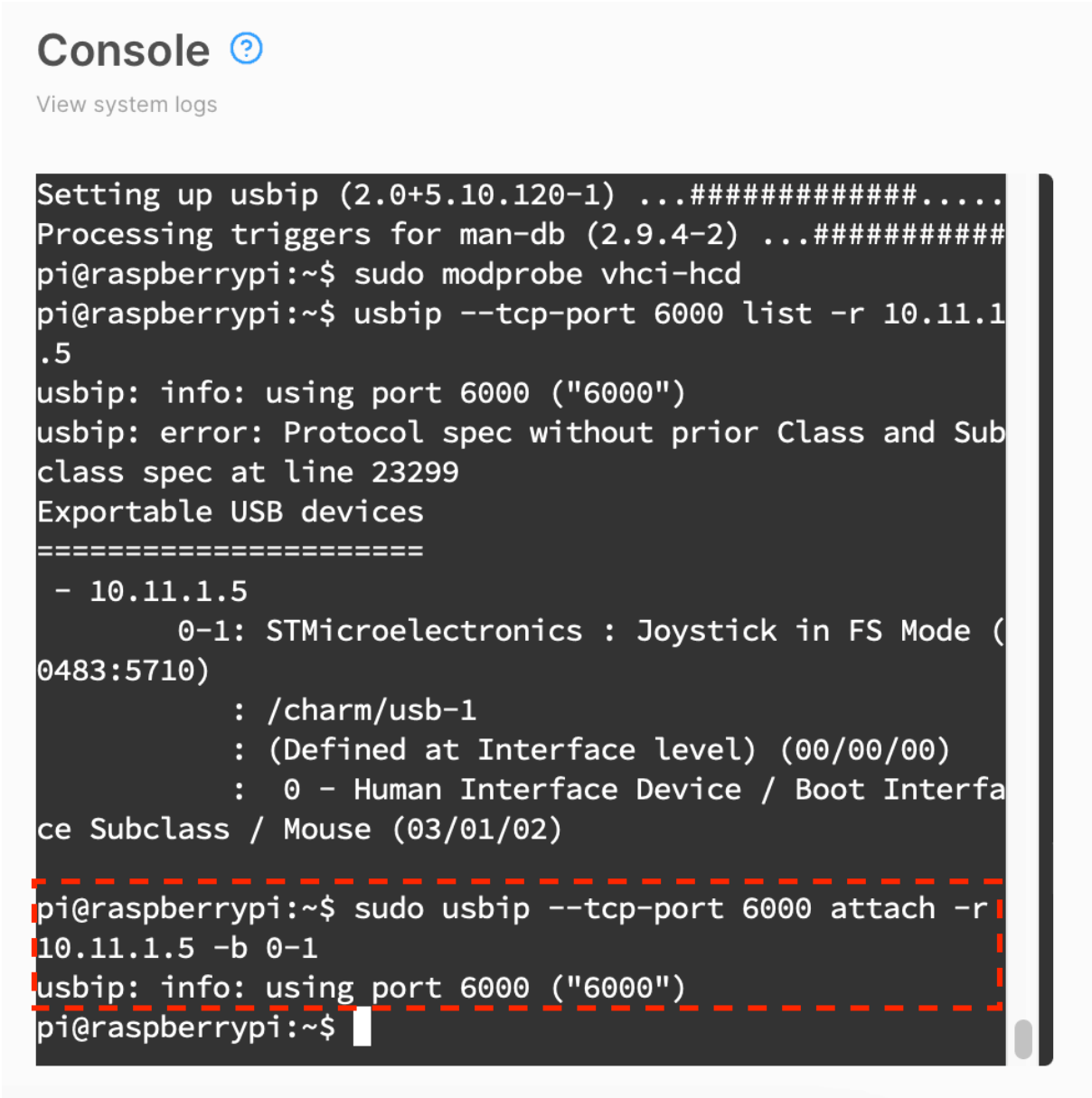


8. Attach to USB bus 0-1, which corresponds to the joystick peripheral. (Don't forget to `sudo` or the command will fail.)

```
sudo usbip --tcp-port 6000 attach -r 10.11.3.3 -b 1-2
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

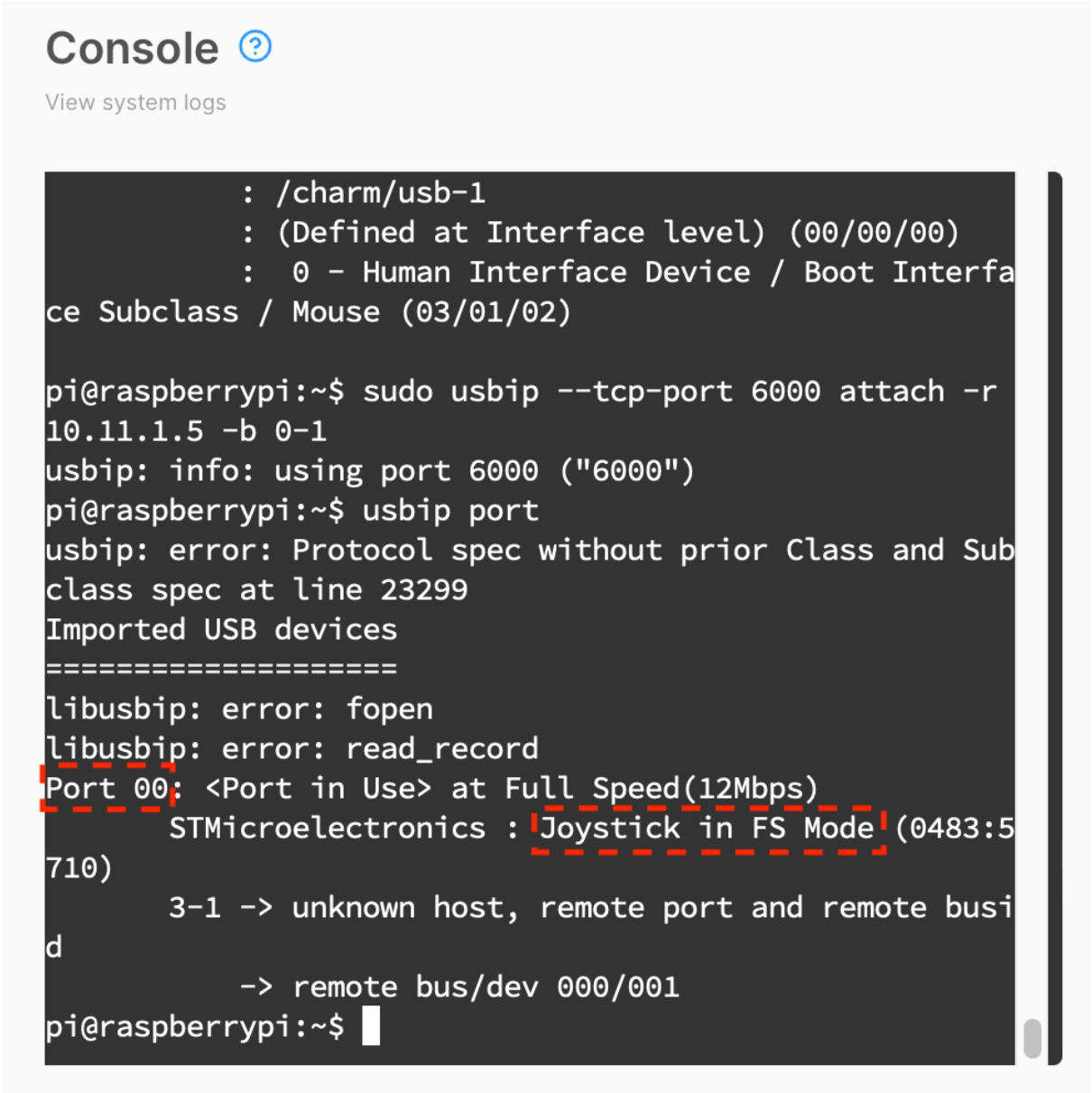
Figure 6-56: Attach to USB bus 0-1



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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-57: Confirm the device is attached

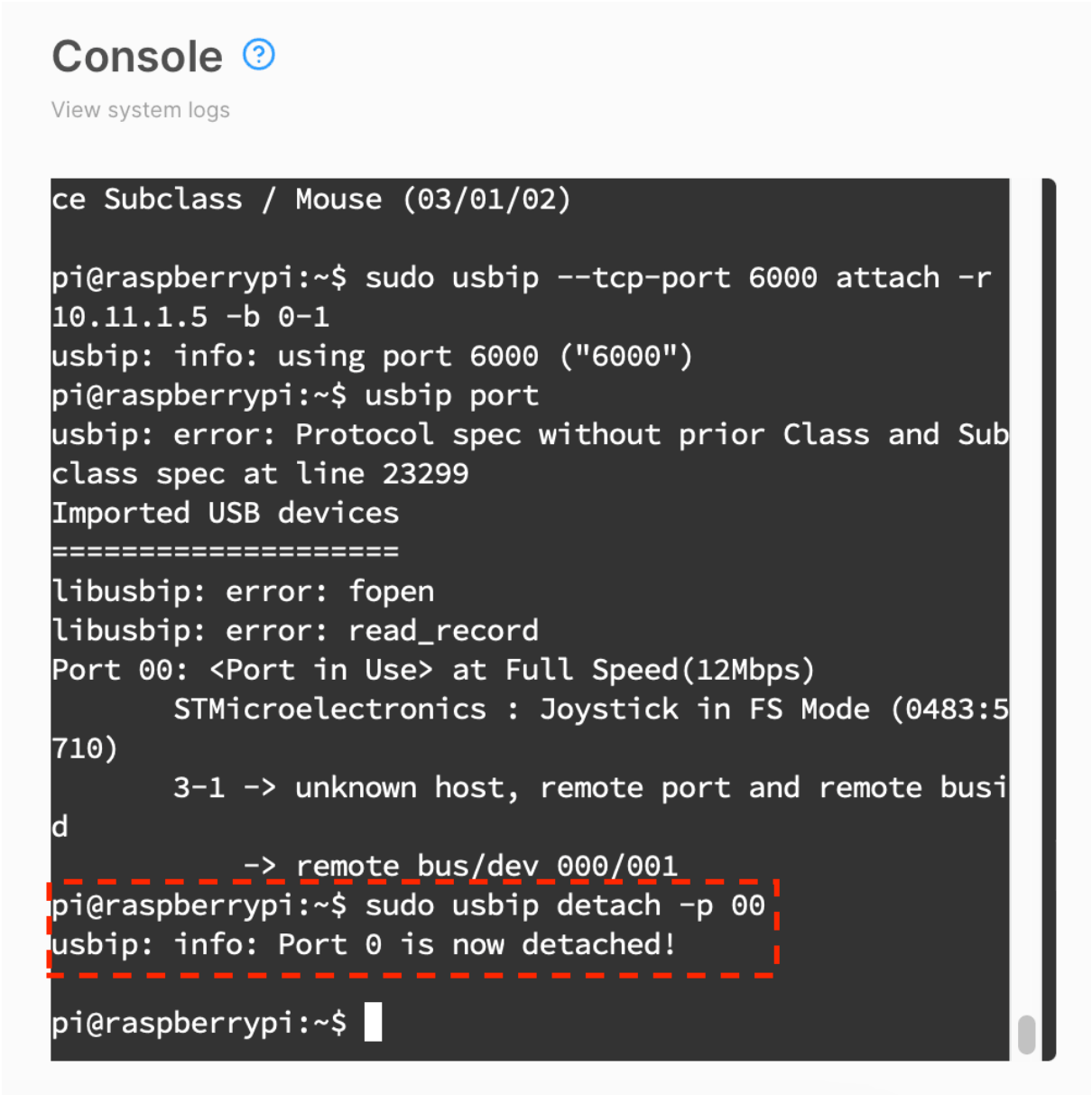


10. When you are finished, detach the peripheral from port `00`.

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-58: Detach the peripheral



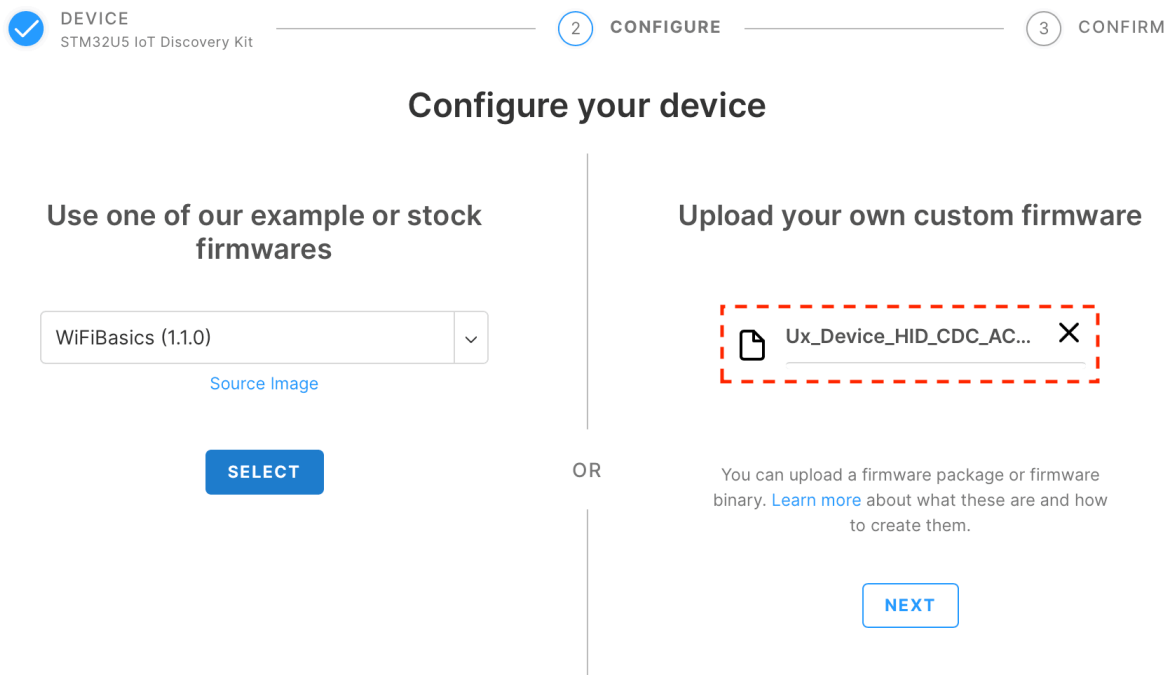
Attach a Virtual USB Peripheral to a Physical Computer Using a VPN Connection

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.

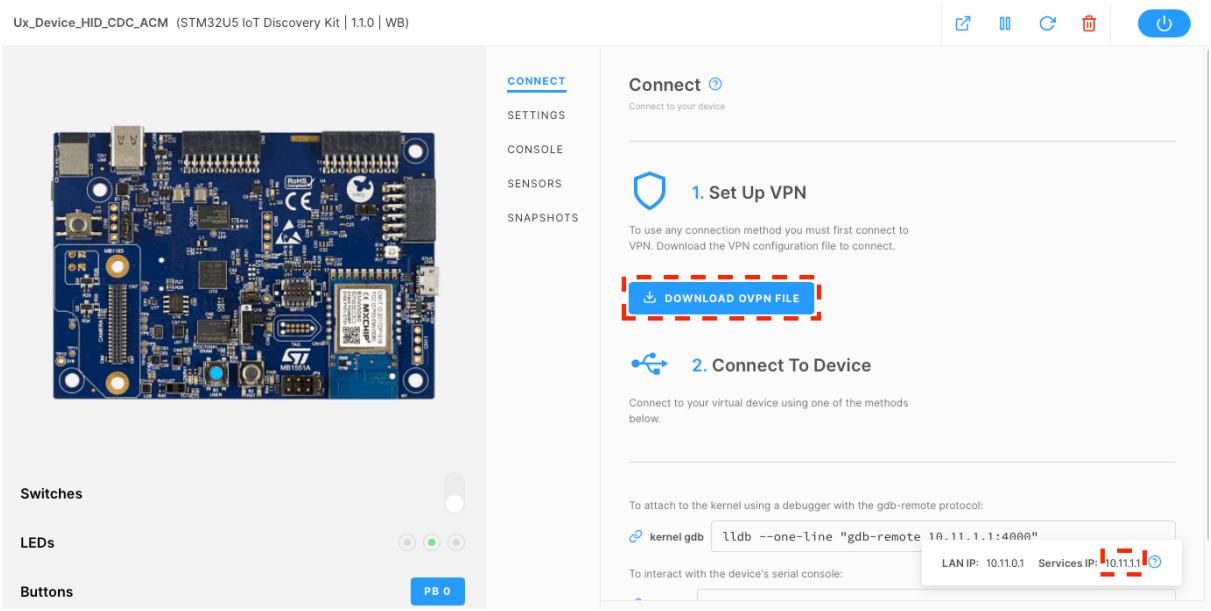
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-59: 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 6-60: STM32U5 VPN and Services IP



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

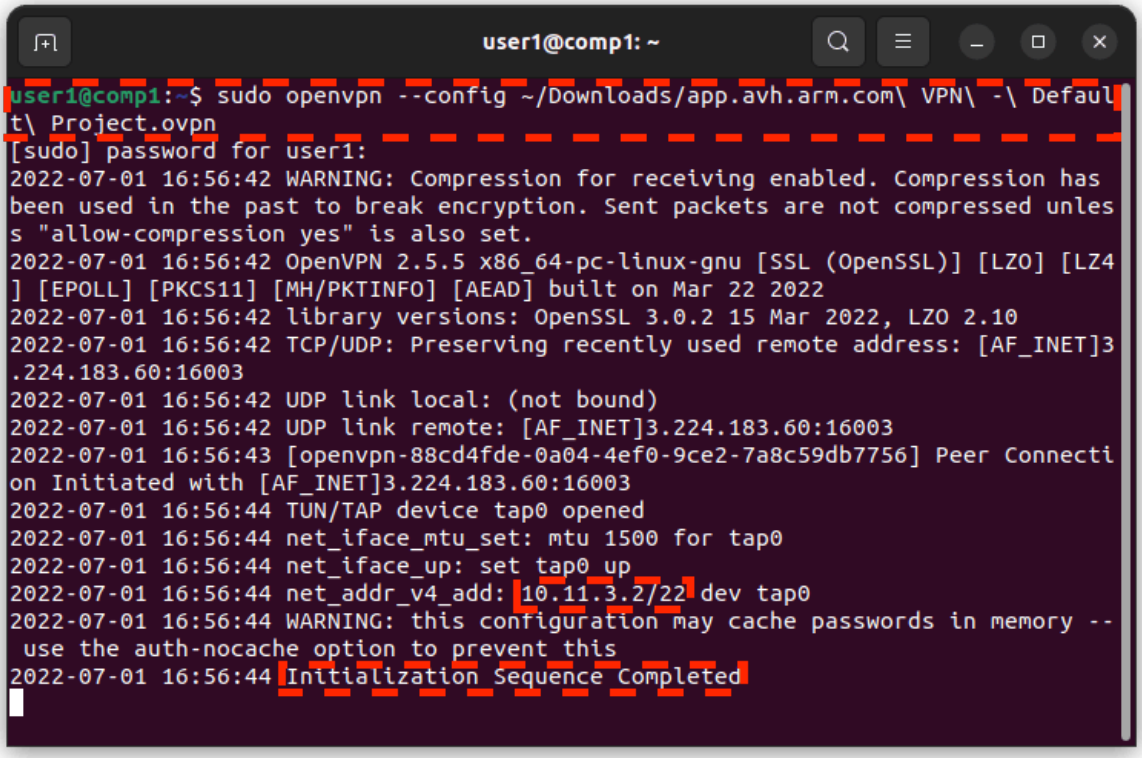
3. On your Linux client machine, install the required packages.

```
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
```

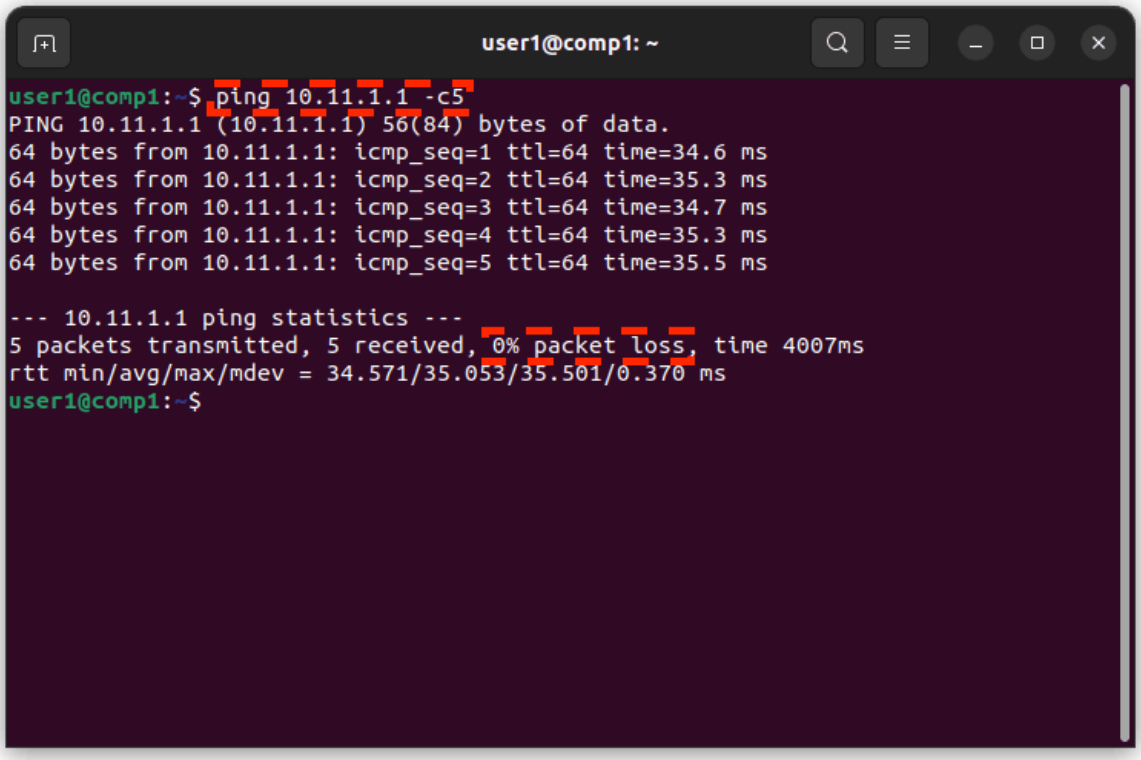
Figure 6-61: OpenVPN



5. Open a second Terminal window. Double-check your VPN connection by pinging the Services IP for your USB/IP host.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-62: Ping Test

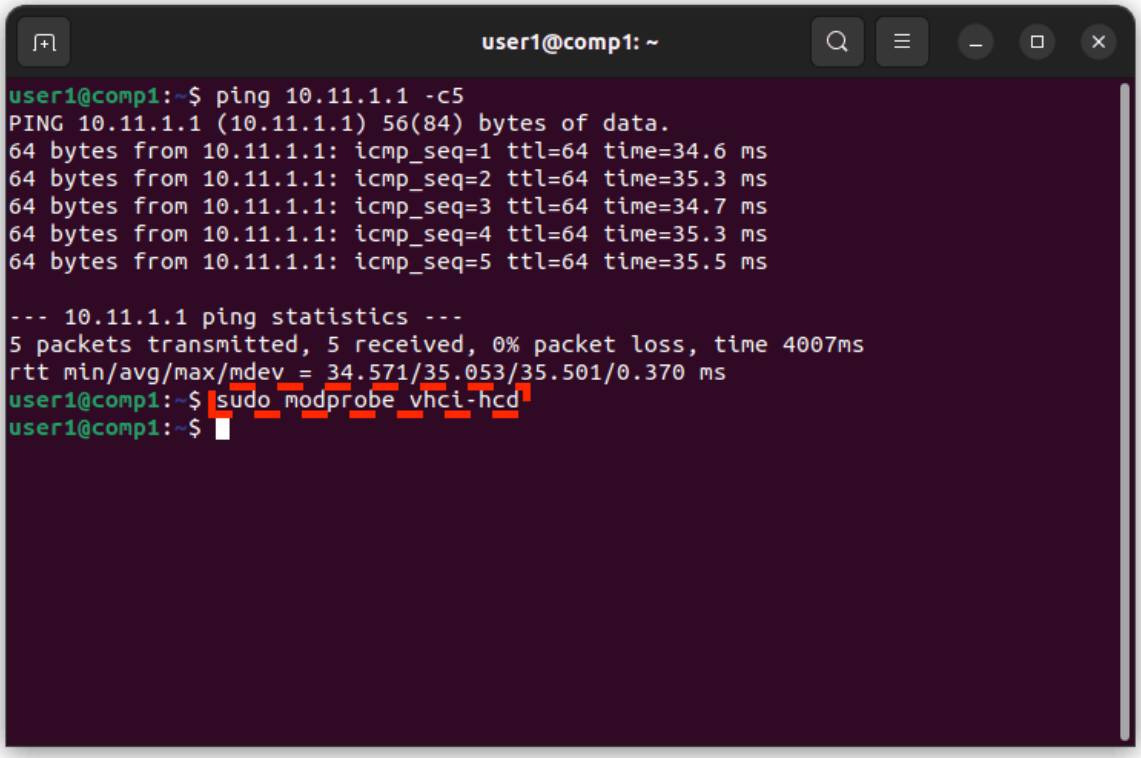


6. Load the Virtual Host Controller Interface kernel module.

```
sudo modprobe vhci-hcd
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-63: Load the kernel module

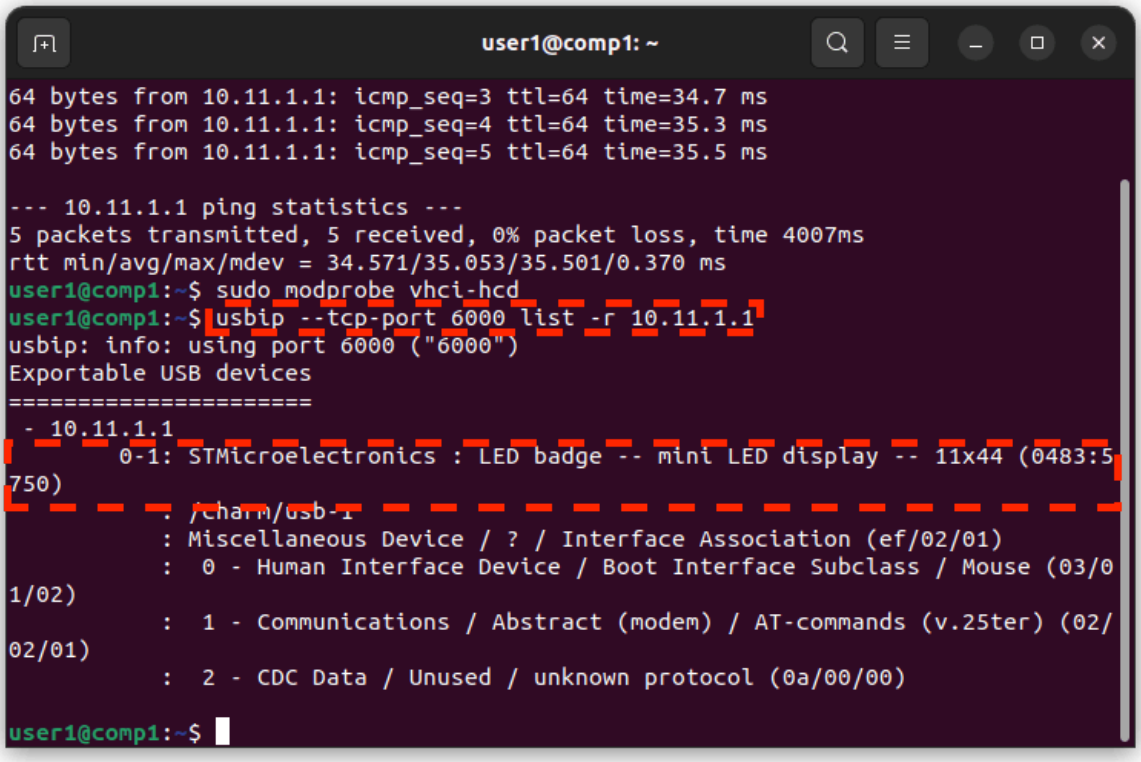


7. List the exportable STM32U5 USB devices.

```
usbip --tcp-port 6000 list -r 10.11.1.1
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-64: 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-65: Attach to USB Device

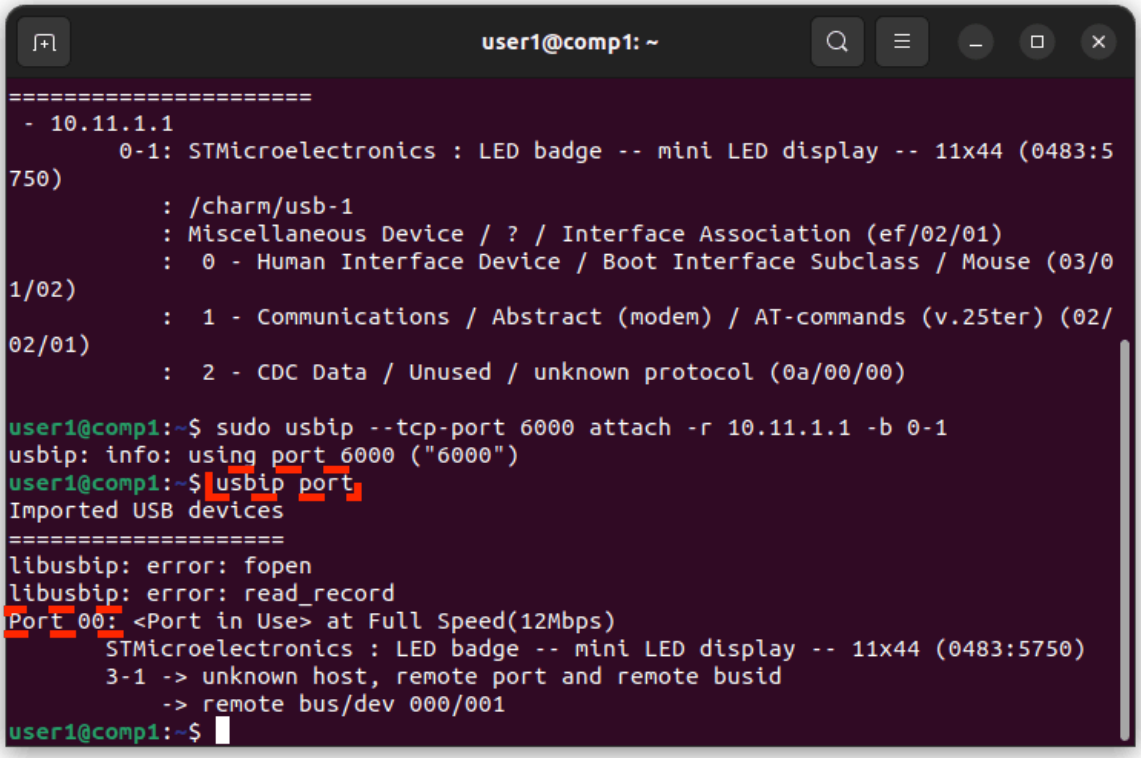
```
user1@comp1: ~
64 bytes from 10.11.1.1: icmp_seq=5 ttl=64 time=34.7 ms

--- 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-66: Imported USB Device

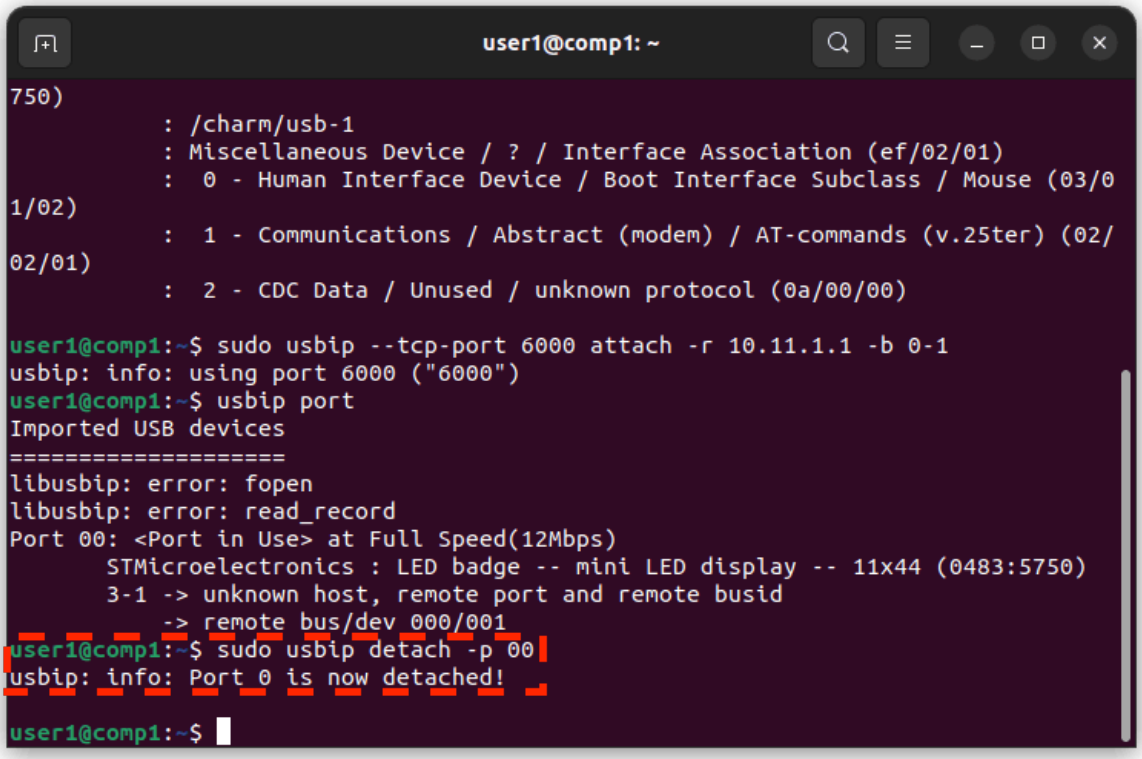


10. When you are finished, detach from port `00`.

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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 6-67: Detach USB Device



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`.
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
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7. Debug

The following articles are available:

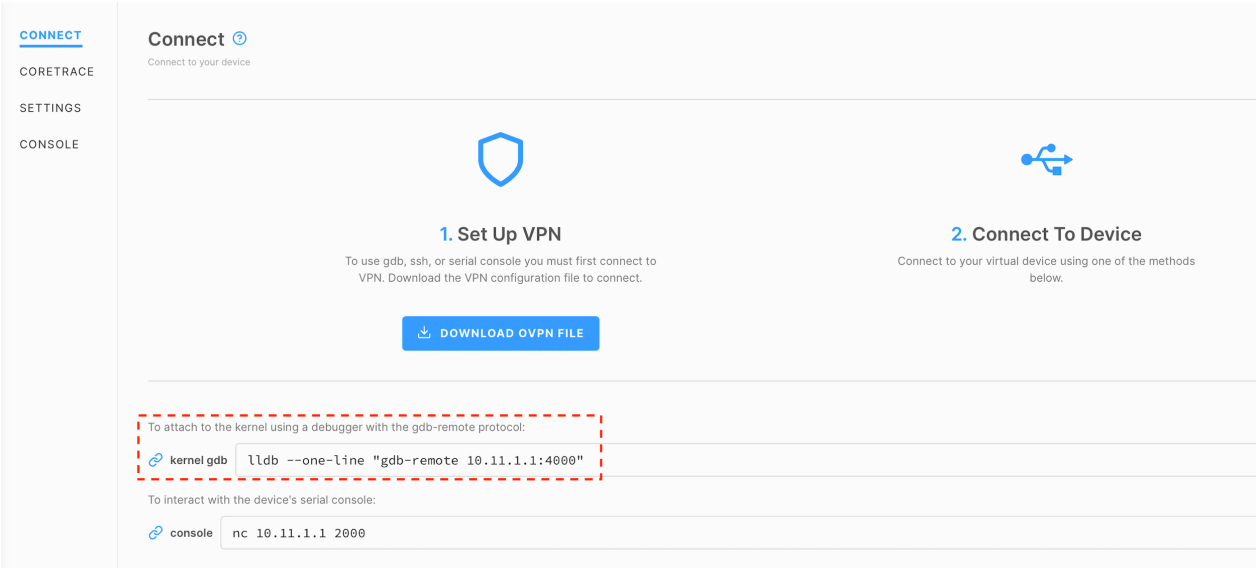
- Kernel Debugging Using GDB/LLDB explains how to use device's GDB Connection for kernel debug.
- CoreTrace describes CoreTrace feature.

7.1 Kernel Debugging Using GDB/LLDB

This article shows you how to debug a kernel on your virtual device using GDB/LLDB protocol. For overall information about debugging virtual devices see [Debug Connections].

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 7-1: 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
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
...
(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
```

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.

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          = 0x5503510
...
```

Or list the current configuration of the page tables of the current virtual CPU (vCPU).

```
(gdb) monitor pt
Pagetable for CPU 0:
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
  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
...
```

`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.

7.2 Debug Accelerator

The AVH Debug Accelerator reduces the effect of network latency. It improves the debugging speed by upto 10x, and makes the debugging experience more natural.

Use Debug Accelerator on macOS and Linux

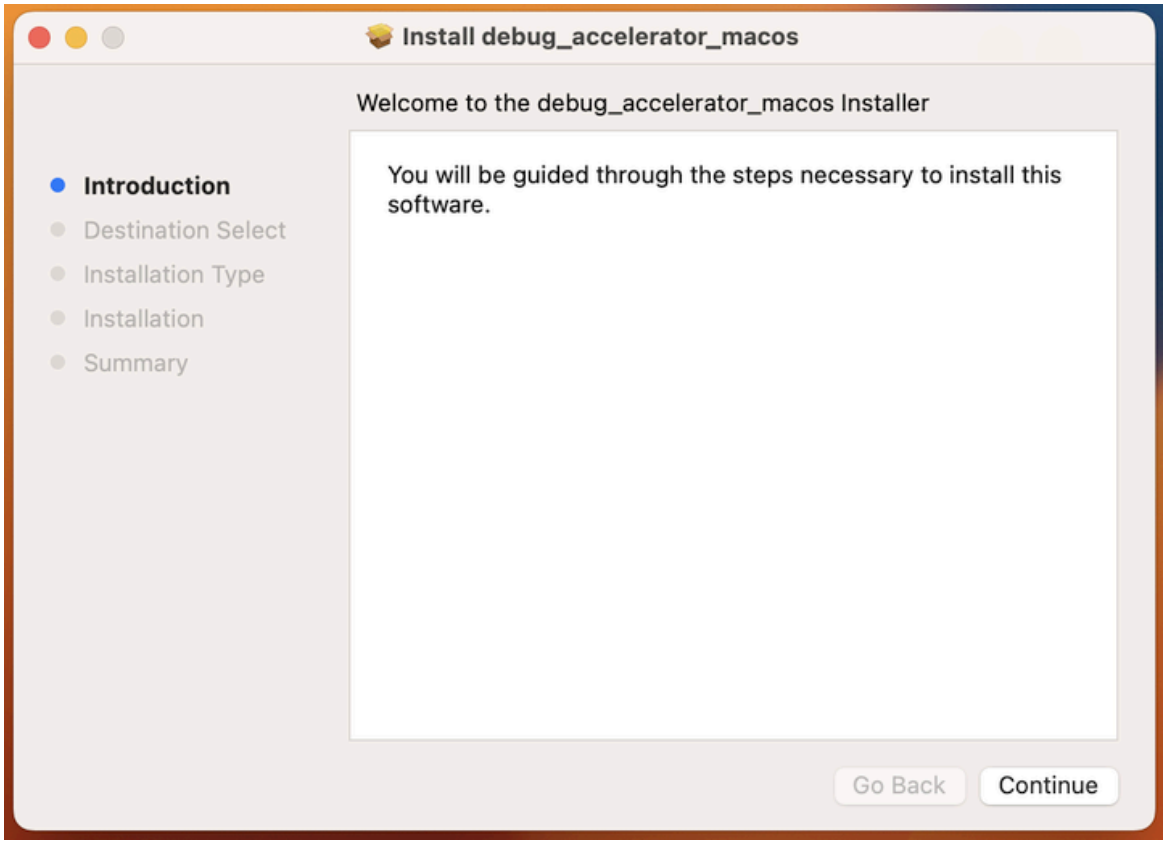
1. Download the Linux binary or the macOS installation package.
   - On Linux, set the Debug Accelerator binary as executable.

     ```
     chmod u+x /path/to/debug_accelerator_linux_x86_64
     ```

   - On MacOS, run the installer for Debug Accelerator for macOS. Click through the prompts and complete the installation. The executable will be installed under `/usr/local/bin/`.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 7-2: macOS installer



2. Connect to the Virtual Device's VPN. Refer to Connect to your AVH VPN for more details
3. Make note of your device's Services IP from the Connect tab. In our example, we use `10.11.1.1`.
4. Run the Debug Accelerator, referencing `<services_ip>:4000` for the remote host and port then `127.0.0.1:4000` for the local host and port. Look for a response: `Listening on '127.0.0.1:4000'`.

```
/path/to/debug_accelerator 10.11.1.1:4000 127.0.0.1:4000
```

5. Slightly modify the gdb command from the AVH device's Connect, referencing the localhost IP. The command examples all reference lldb but gdb can be used as well.

```
lldb --one-line "gdb-remote 127.0.0.1:4000"
```

6. You will notice a significant performance improvement when debugging.
   - Speed when using standard debugging:

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 7-3: Speed of standard debugging

```
    frame #0: 0xffffffc010adaae0
->  0xffffffc010adaae0: nop
    0xffffffc010adaae4: mov    x0, #0x60
    0xffffffc010adaae8: msr    DAIFSet, #0x2
    0xffffffc010adaaec: bl     -0x3fefef9778
Target 0: (No executable module.) stopped.
(lldb) si
Process 1 stopped
* thread #1, stop reason = instruction step into
    frame #0: 0xffffffc010adaae4
->  0xffffffc010adaae4: mov    x0, #0x60
    0xffffffc010adaae8: msr    DAIFSet, #0x2
    0xffffffc010adaaec: bl     -0x3fefef9778
    0xffffffc010adaaf0: mov    x0, #0xe0
Target 0: (No executable module.) stopped.
(lldb)
```

- Speed when using Debug Accelerator:

Figure 7-4: Speed of Debug Accelerator

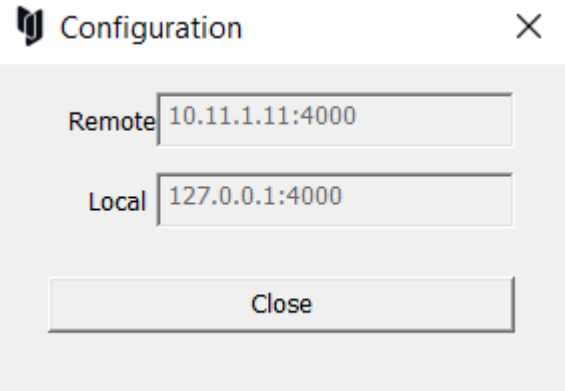
```
->  0xffffffc010adaae0: nop
    0xffffffc010adaae4: mov    x0, #0x60
    0xffffffc010adaae8: msr    DAIFSet, #0x2
    0xffffffc010adaaec: bl     -0x3fefef9778
Target 0: (No executable module.) stopped.
(lldb) si
Process 1 stopped
* thread #1, stop reason = instruction step into
    frame #0: 0xffffffc010adaae4
->  0xffffffc010adaae4: mov    x0, #0x60
    0xffffffc010adaae8: msr    DAIFSet, #0x2
    0xffffffc010adaaec: bl     -0x3fefef9778
    0xffffffc010adaaf0: mov    x0, #0xe0
Target 0: (No executable module.) stopped.
(lldb) fini
```

Use Debug Accelerator on Windows CLI

1. Install LLVM for your system type.
2. Connect to the Virtual Device's VPN. Refer to Connect to your AVH VPN for more details.
3. Make note of your device's Services IP from the Connect tab. In our example, we use `10.11.1.11`.
4. Download and run the installer for Debug Accelerator for Windows. The executable will be installed under `C:\Program Files\Corellium\Debug Accelerator`. Windows MSI Installer
5. Run Debug Accelerator and configure Remote to `<services_ip>:4000` and Local to `127.0.0.1:4000`.

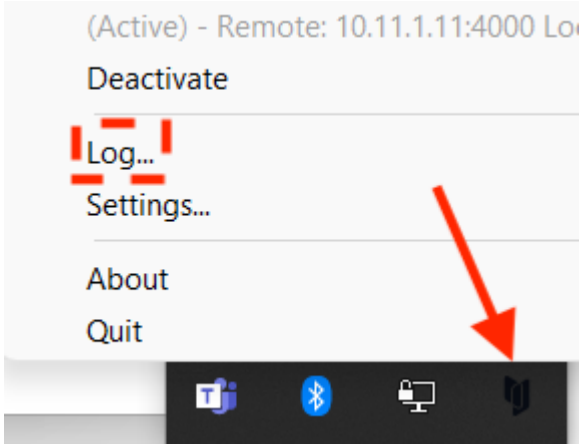
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 7-5: Configure on Windows



6. The program will continue running in the background. Right-click on the systray icon, click Log..., and look for a line that says `Listening on '127.0.0.1:4000'`.

Figure 7-6: Click log on Windows



7. From PowerShell or Windows Command Prompt, modify the lldb command provided for the device through the web interface with your localhost IP and port 4000.

```
lldb --one-line "gdb-remote 127.0.0.1:4000"
```

8. You should notice a significant performance improvement when debugging.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7.3 CoreTrace

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

What is CoreTrace?

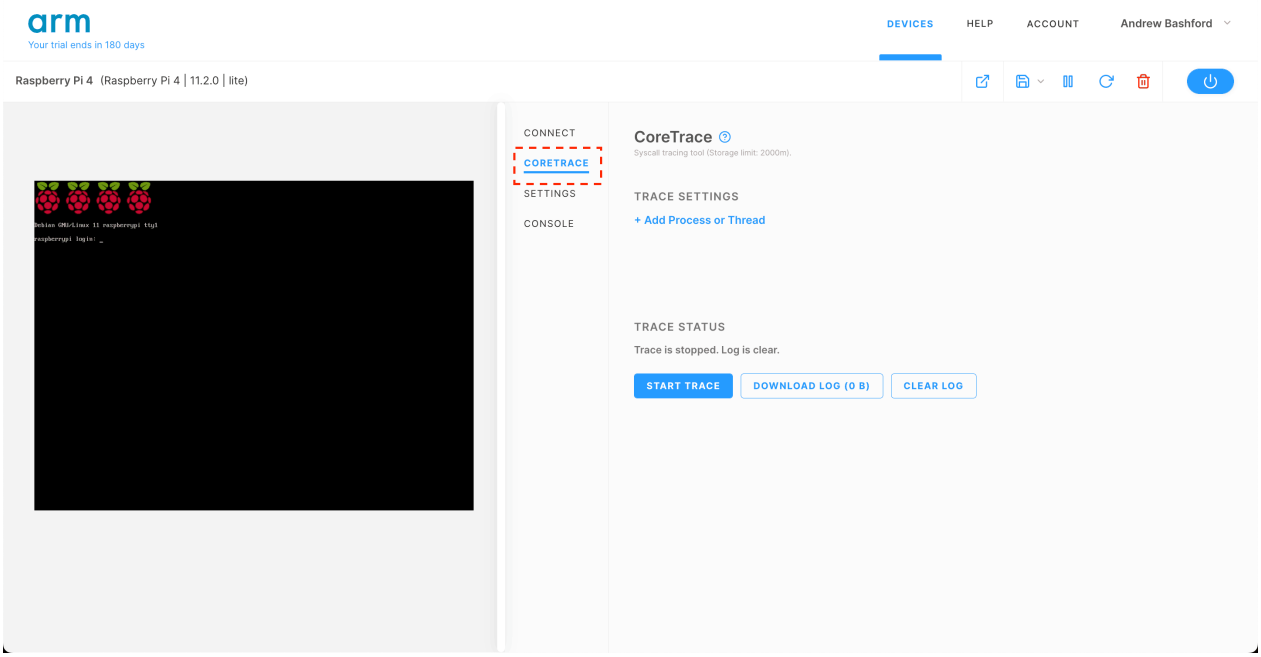
You can trace system calls using either strace, a standard command-line Linux tool, or our proprietary CoreTrace tool. strace is included in our virtual devices, and it is implemented with ptrace. Applications can employ anti-debugging techniques to detect and prevent ptrace-based tracing. However, these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

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 7-7: CoreTrace tab



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:

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

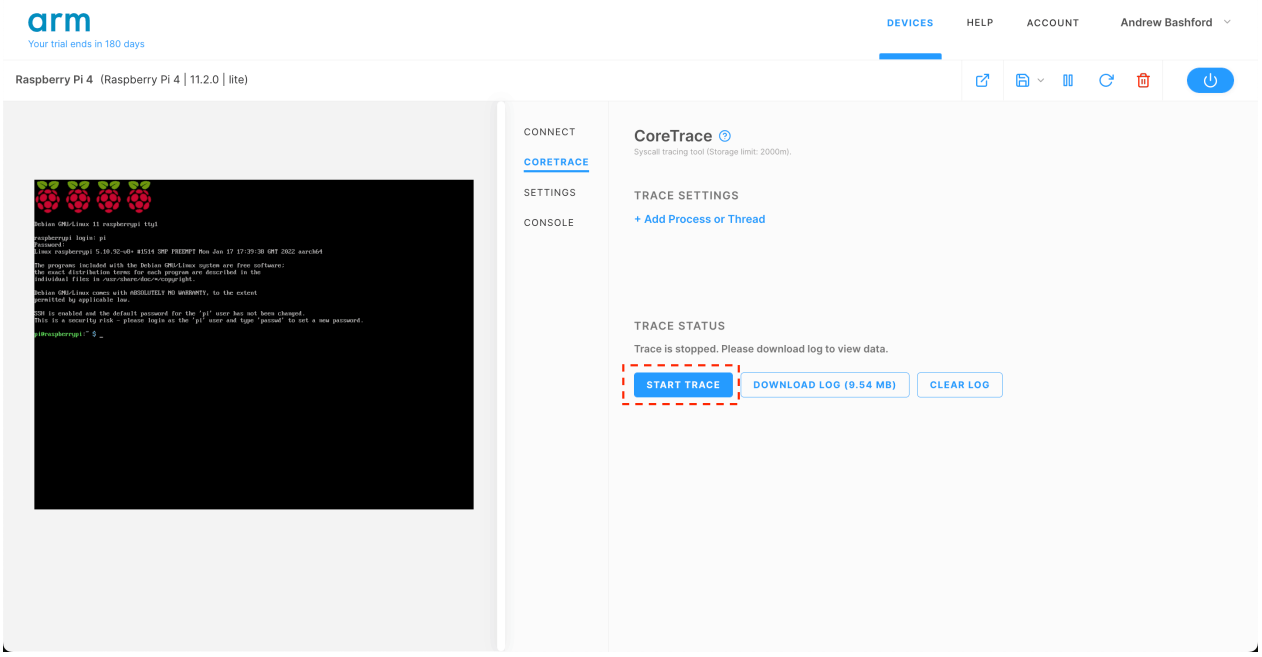
The Processes dialog displays all processes and threads in the system. To examine the threads inside a process, click the “THREADS” button in the process’ row.

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 7-8: Start Trace



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/ @00000070efc834
 read ( fd: 5, buf: 0x6e5f6ec980, count: 4 ) ... @[ 0000006e5f6f9778
 0000006e5f6f9840 0000006e5f6f948c 0000006e5f6f7f90 0000006e5f6f54 0000006e5f6f7434
 00000070efc2088c 00000070efbc0e0c ]
```

or like this:

```
1<1> [00248.864656648] ffffff806401e040-0/337:sensors@2.1-ser.379/
 @00000070efc834 ... read ( result: 4, buf: 0x6e5f6ec980 -> [s"001e"] )
```

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8. Use AVH via API

Arm Virtual Hardware (AVH) exposes most of its functionality via the REST API interface, making it simple to integrate AVH in Continuous Integration (CI) flows for verifying your programs. With programatic access over APIs you can create and manage virtual devices, run custom firmware and obtain execution logs. API Token allows secure authentication to your AVH environment.

The REST API is defined in OpenAPI format, that is used for generating clients in specific programming languages, such as Python, Javascript and C.

- **REST API endpoint** `https://app.avh.arm.com/api` is the access point for the API.
- REST API web interface lists the available REST API functionality and also allows manual use from your browser. Quickstart for AVH REST API shows how to start using it.
- avh-api python client explains the AVH API python package for use in synchronous mode.
- avh-api-async python client explains the AVH API python package for use in asynchronous mode.
- avh-api JavaScript client describes the JavaScript package for using AVH REST API.
- avh-api C client explains how to create and use a C library for accesing AVH REST API.

Following pages in this guide provide additional instructions and examples:

- API Token
- Quickstart for AVH REST API
- API Usage Examples
- GitHub Actions Example

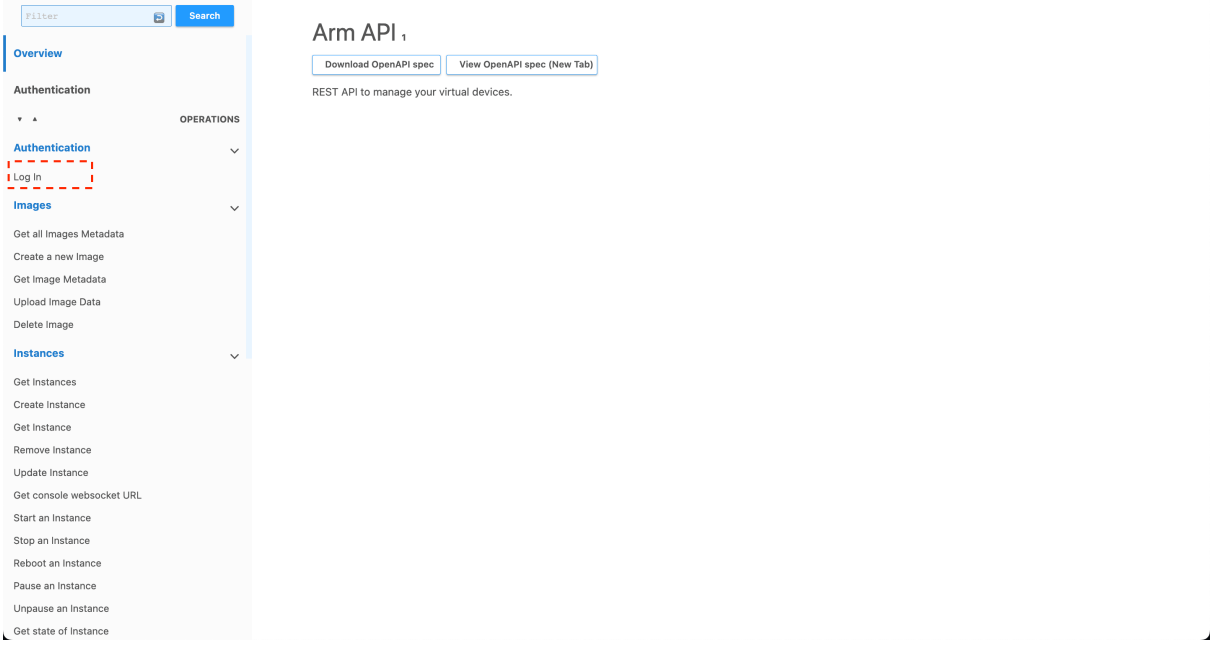
8.1 Quickstart for AVH REST API

This quickstart explains how to start using Arm Virtual Hardware REST API. It guides through the initial steps: logging in, setting the API Token, finding your project ID, and finally creating a device.

1. Log in to AVH with your Arm account at https://app.avh.arm.com. If you do not have an Arm account you can register here.
2. If you don't have an API token, generate it as explained in the [API Authentication Token] article.
3. Next, go to the AVH API Docs https://app.avh.arm.com/api/docs
4. Click on **Log In** on the menu.

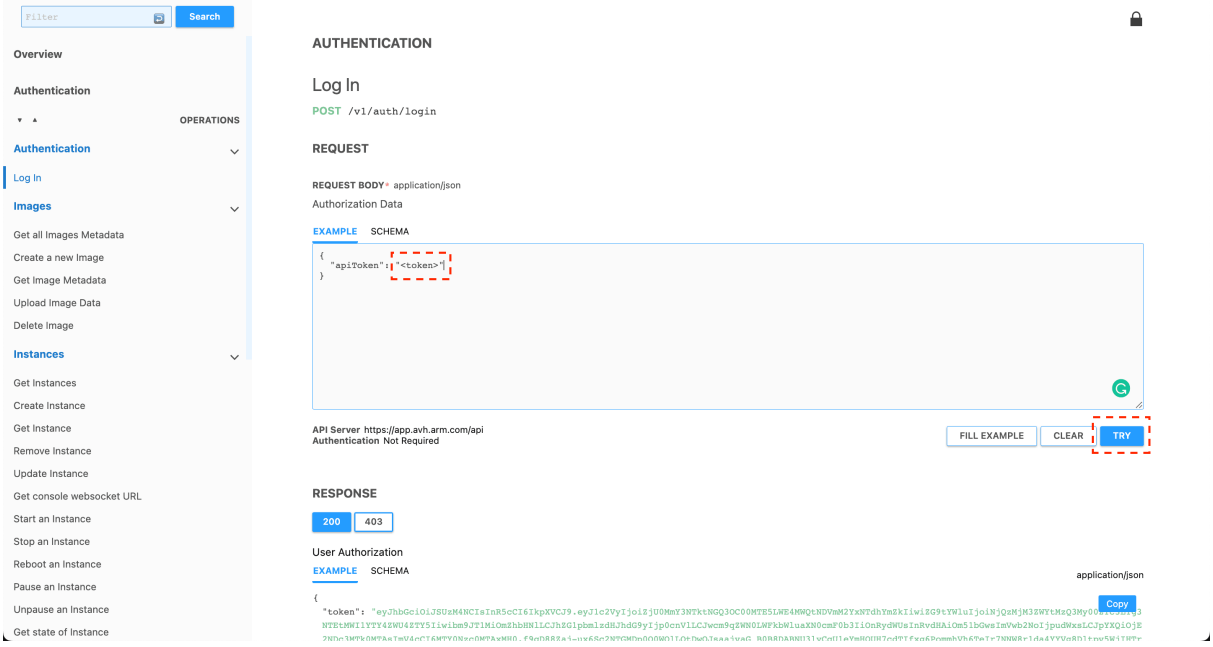
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 8-1: Click “Log In”



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

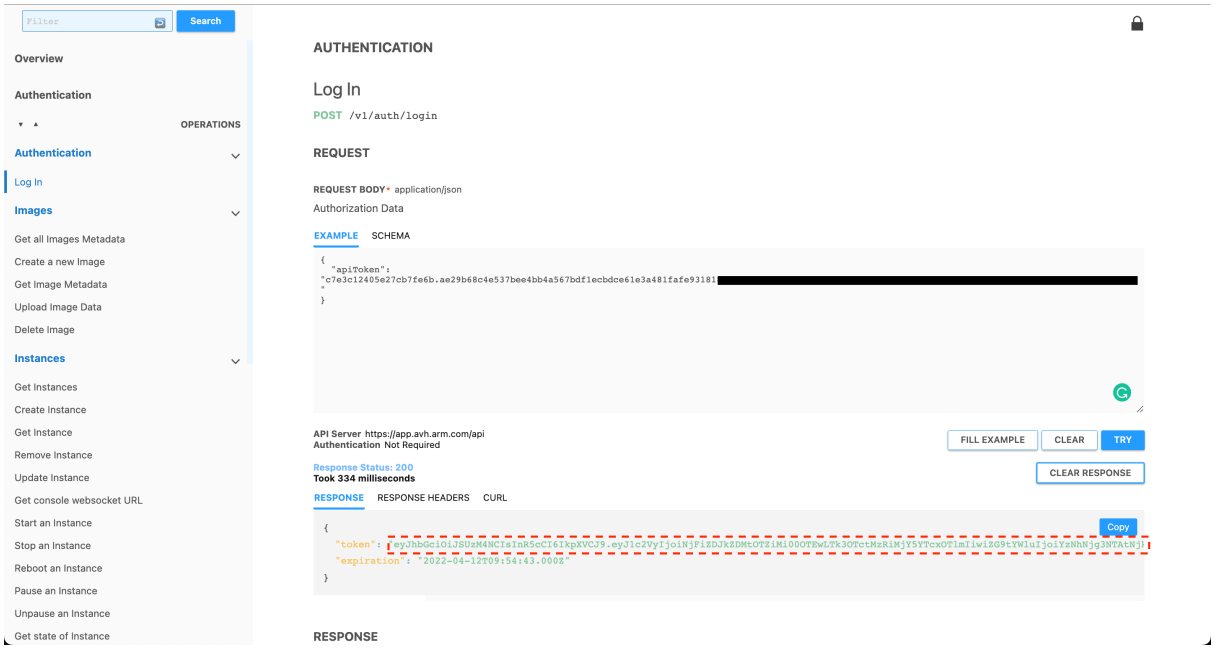
Figure 8-2: Paste the API Token



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.

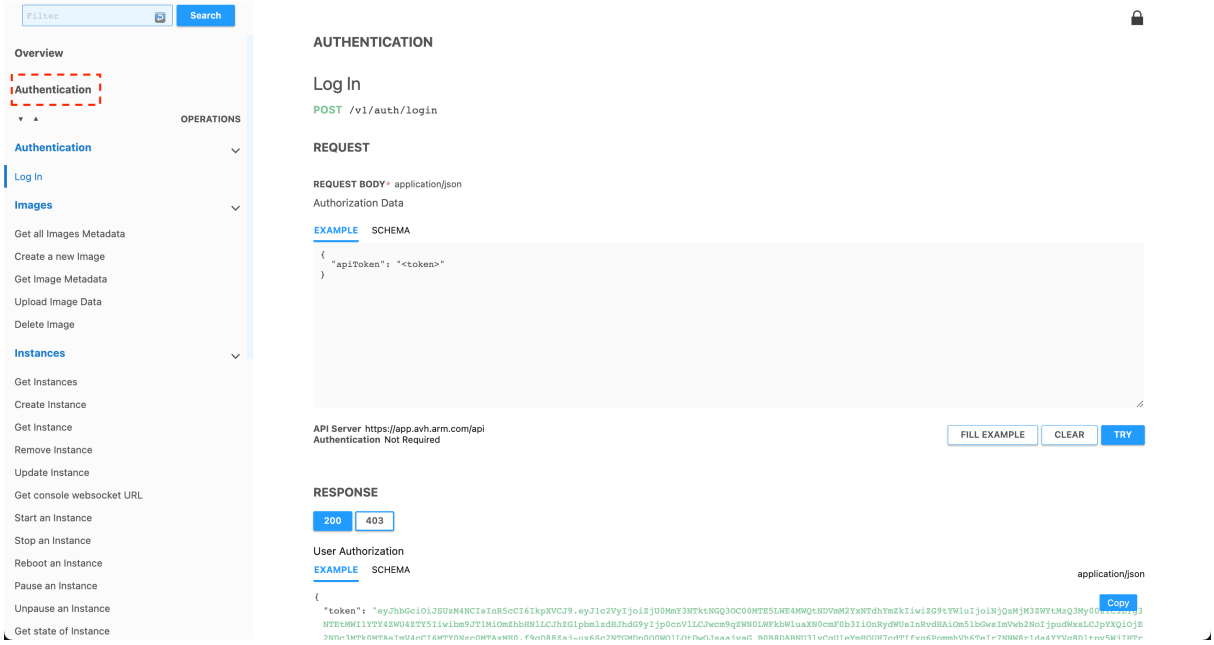
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 8-3: The auth token



7. Click **Authentication** in the menu.

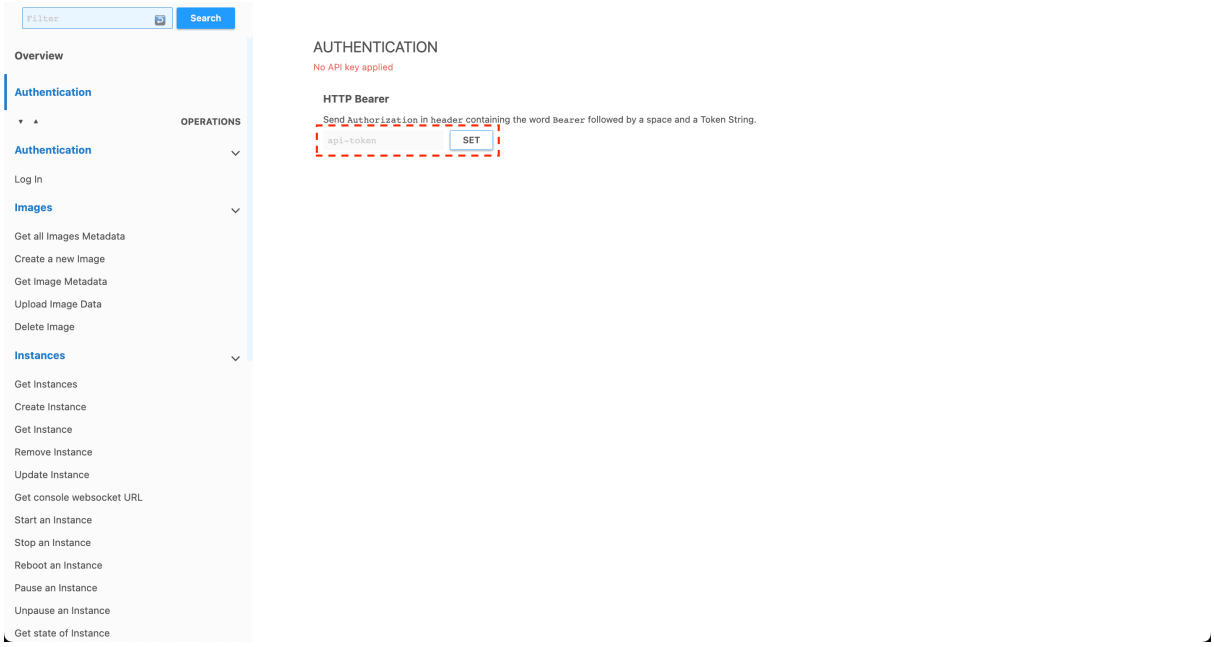
Figure 8-4: Click “Authentication”



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

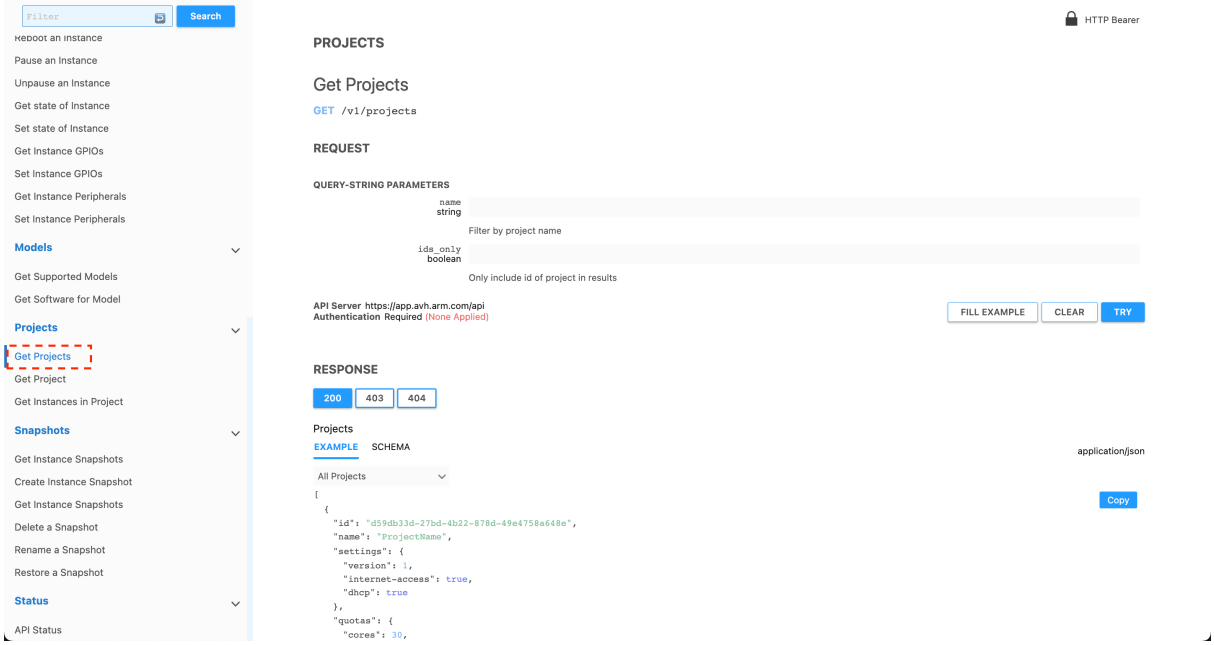
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 8-5: Paste the auth token



9. Click **Get Projects**

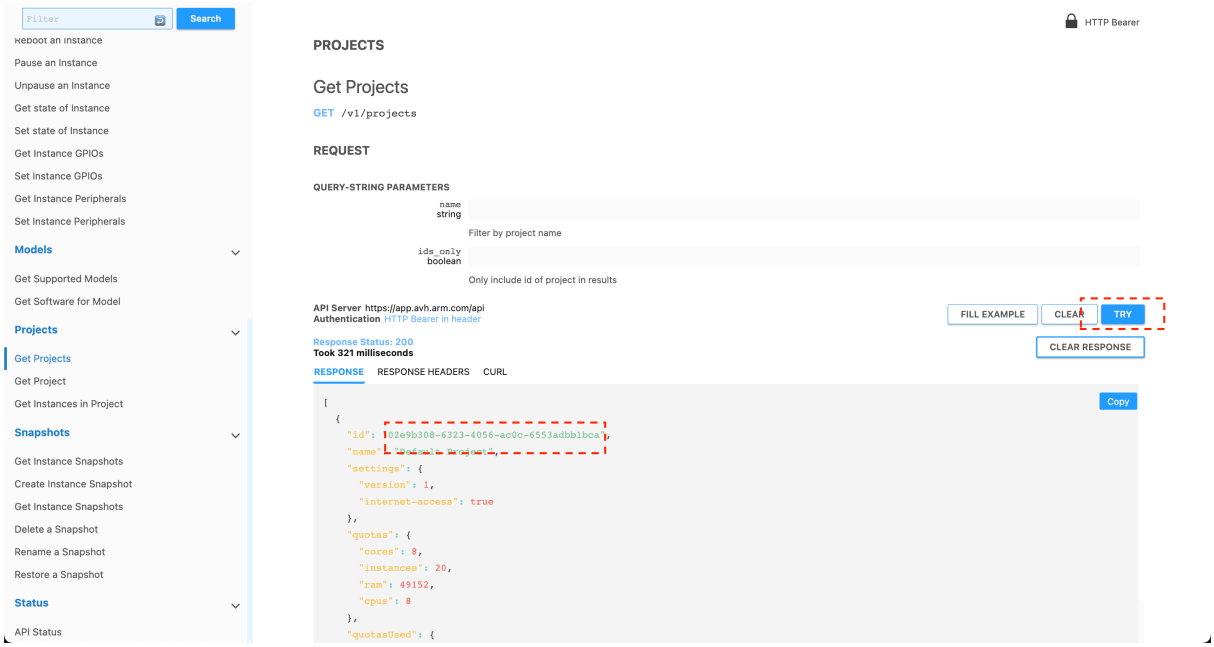
Figure 8-6: Click "Get Projects"



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

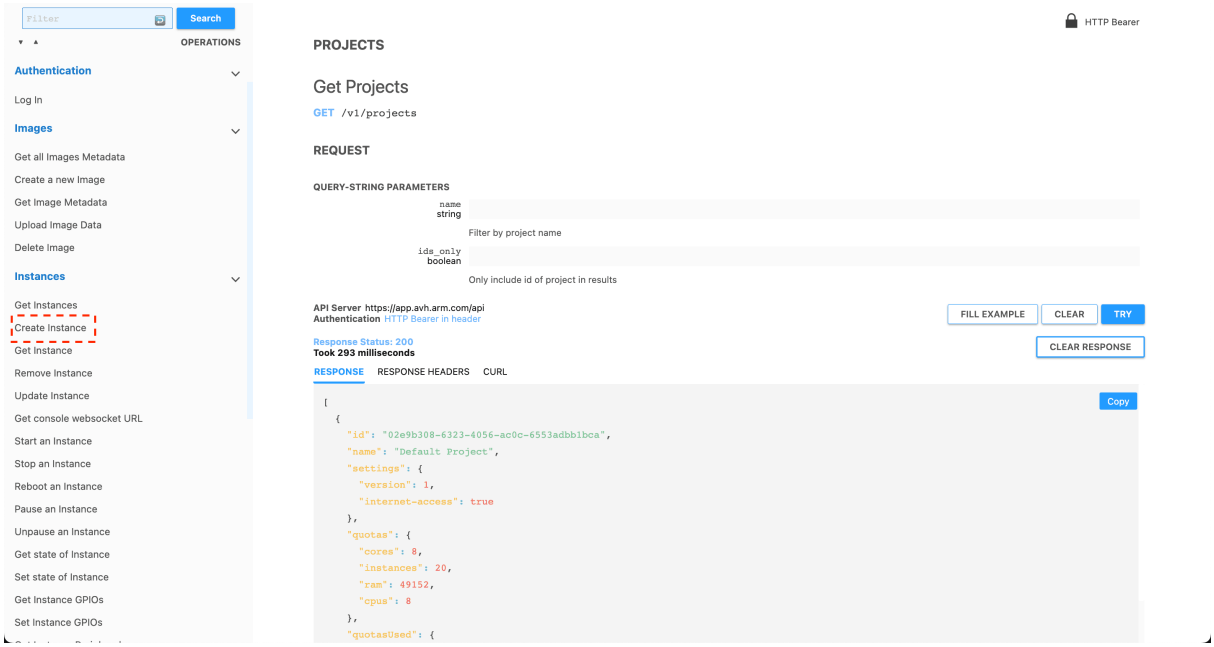
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 8-7: Copy the id from the JSON



11. In the menu click **Create Instance**

Figure 8-8: 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.

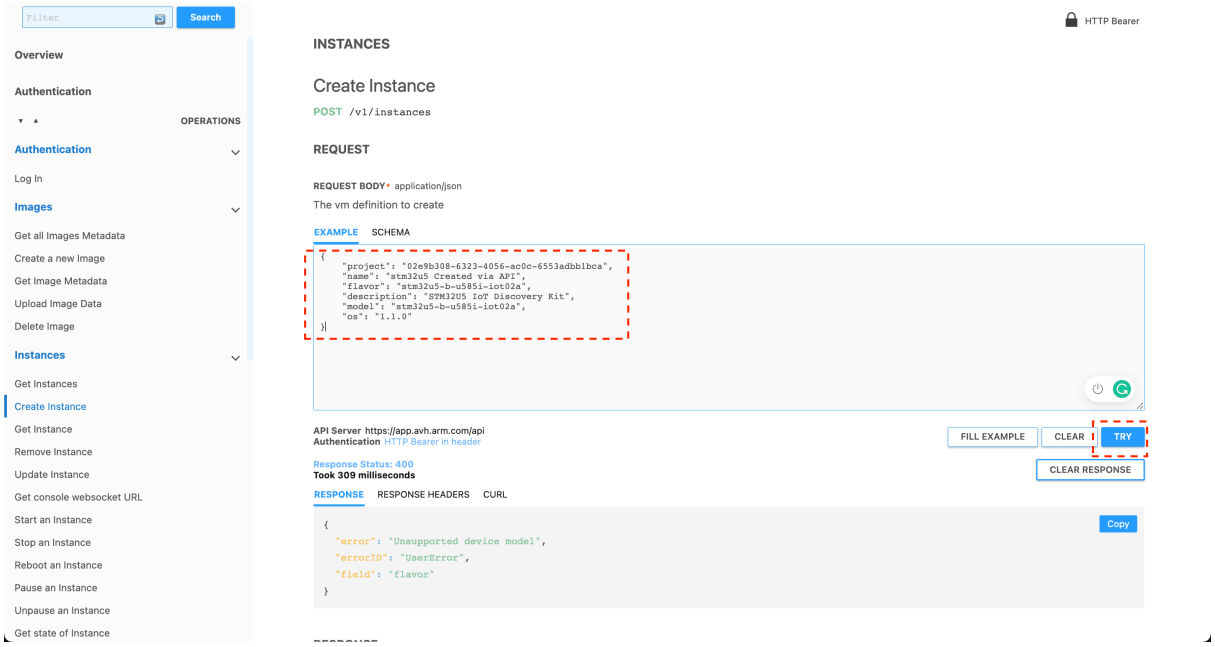
```
{
```

Copyright © 2022–2023 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 8-9: Click "Try"



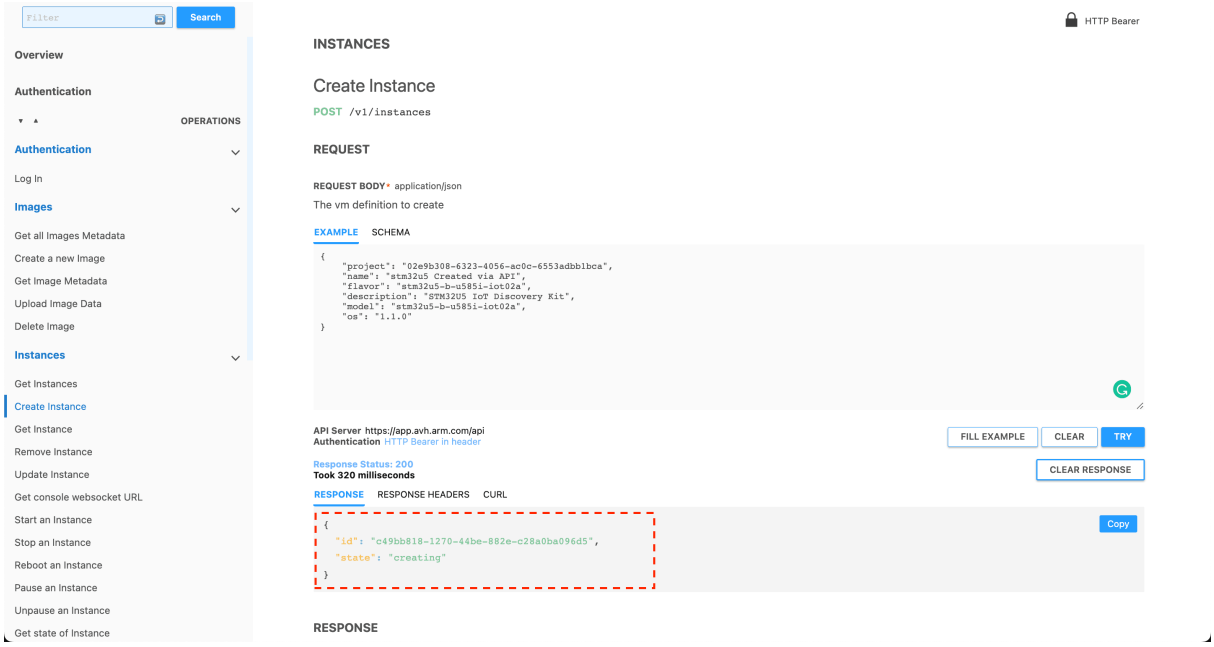
Note

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.

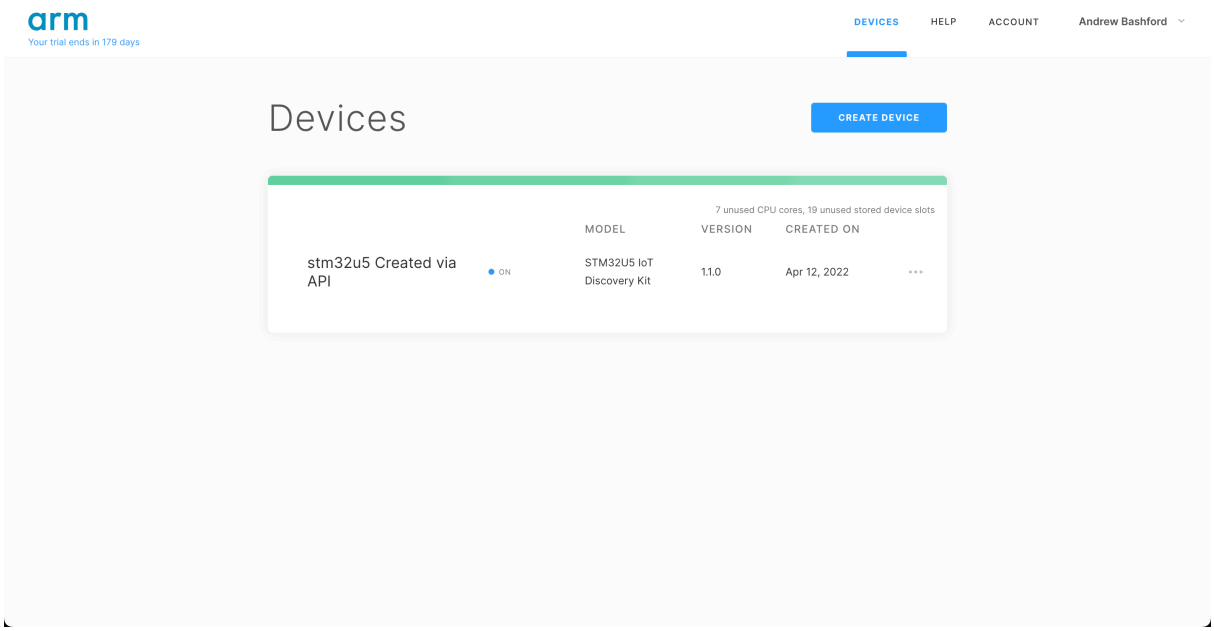
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 8-10: Response showing device status



14. When you go back to your AVH devices list, your shall see the device we have just created in this quickstart guide.

Figure 8-11: AVH devices list



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

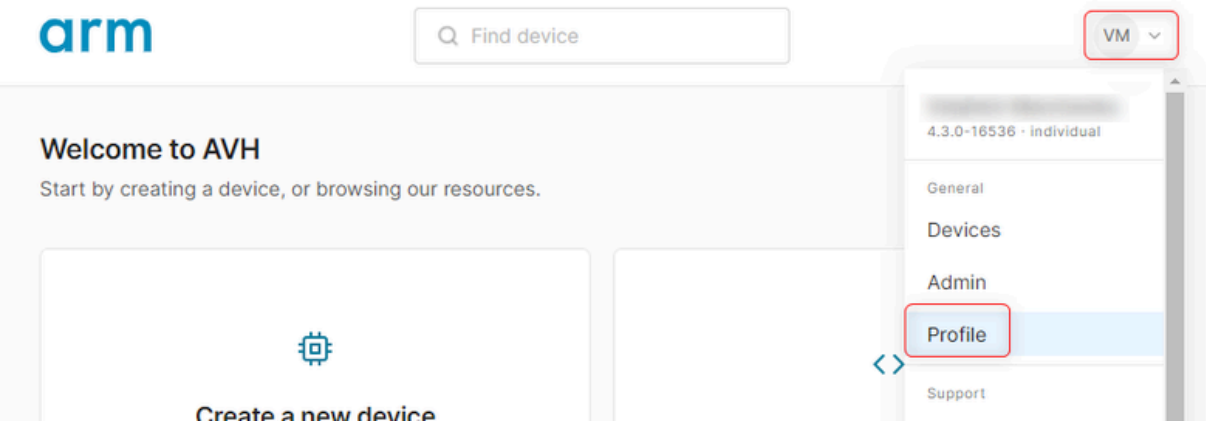
8.2 API Token

API Token allows you to authenticate your user for API access to Arm Virtual Hardware. Make sure to keep it secure. You can only have 1 token per user at a time.

Generate the API token as follows:

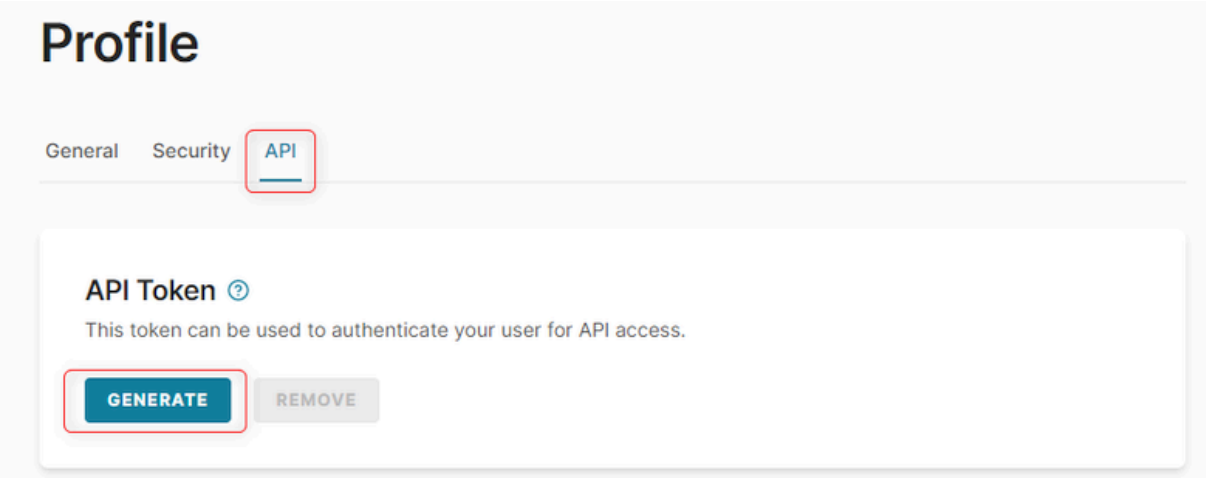
1. Log in to AVH with your Arm account at https://app.avh.arm.com.
2. To retrieve your API Token, click on your name at the top right corner. Then click **Profile** and change to **API** tab.

   Figure 8-12: API token



3. Click **GENERATE** to generate your API Token.

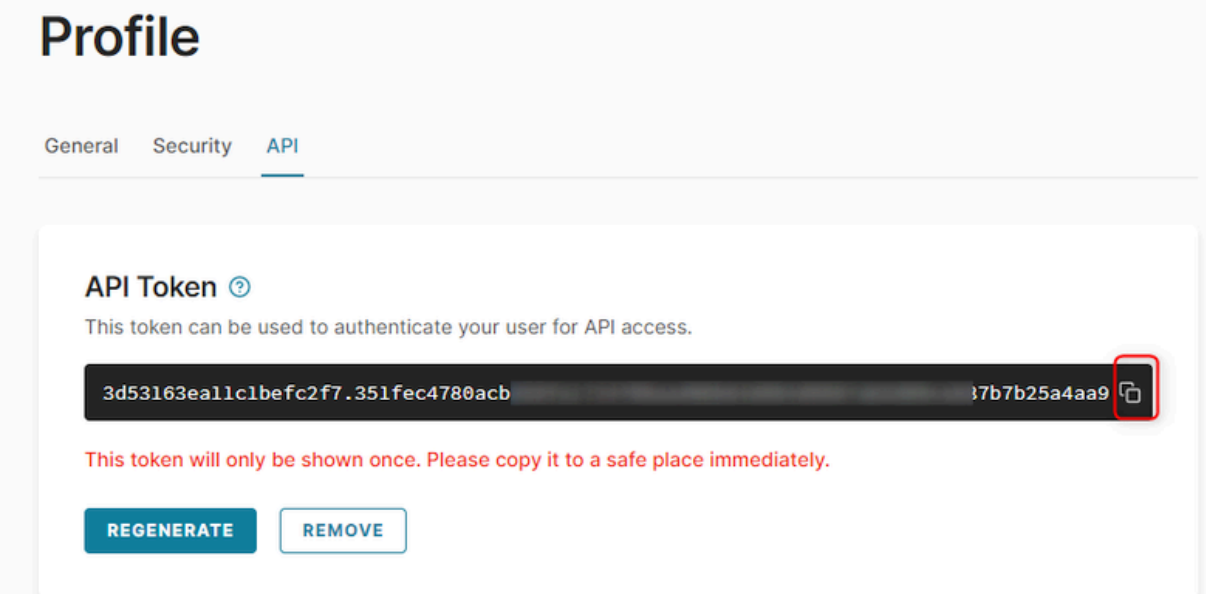
   Figure 8-13: Generate API token



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.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 8-14: Copy API token



8.3 API Usage Examples

github.com/ARM-software/avh-api/ repository provides examples for using AVH API via Python and Javascript clients.

Python Examples

Following python examples are provided in python/examples/ directory of the repo, both for asynchronous (folder async/) and synchronous (folder /sync/) :

- `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

Javascript Examples

The JavaScript examples are provided in javascript/examples/ directory as follows:

- `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`

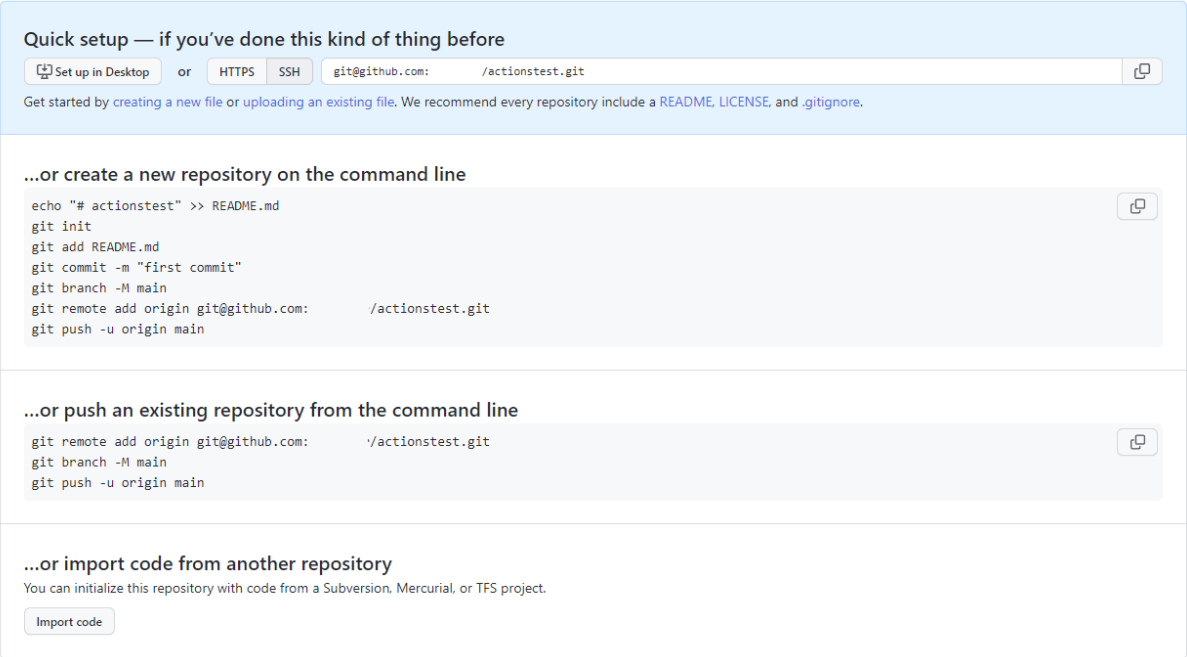
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8.4 GitHub Actions Example

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

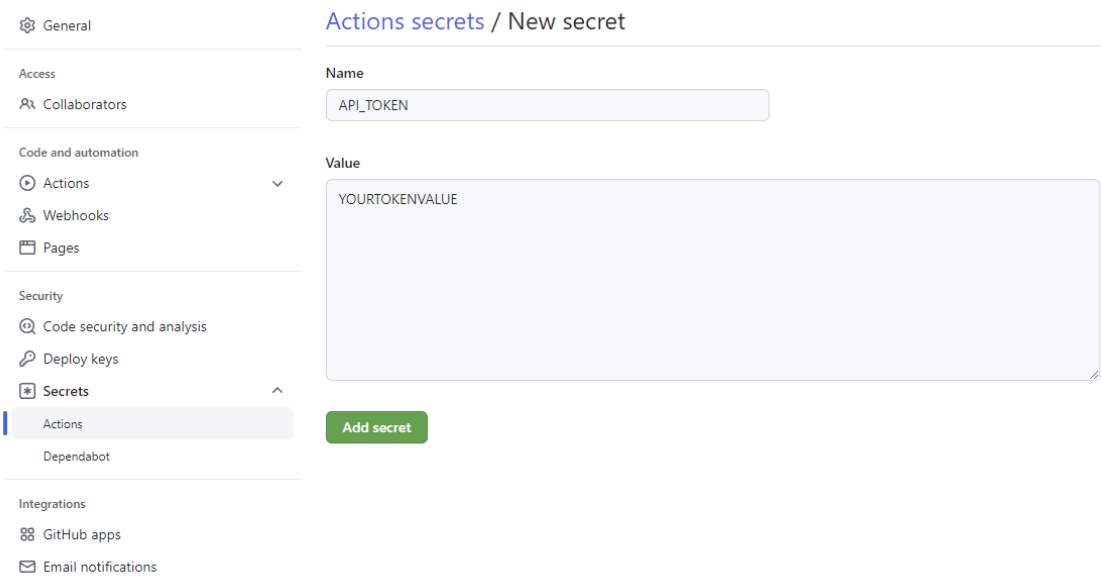
Figure 8-15: Create an empty GitHub repository



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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

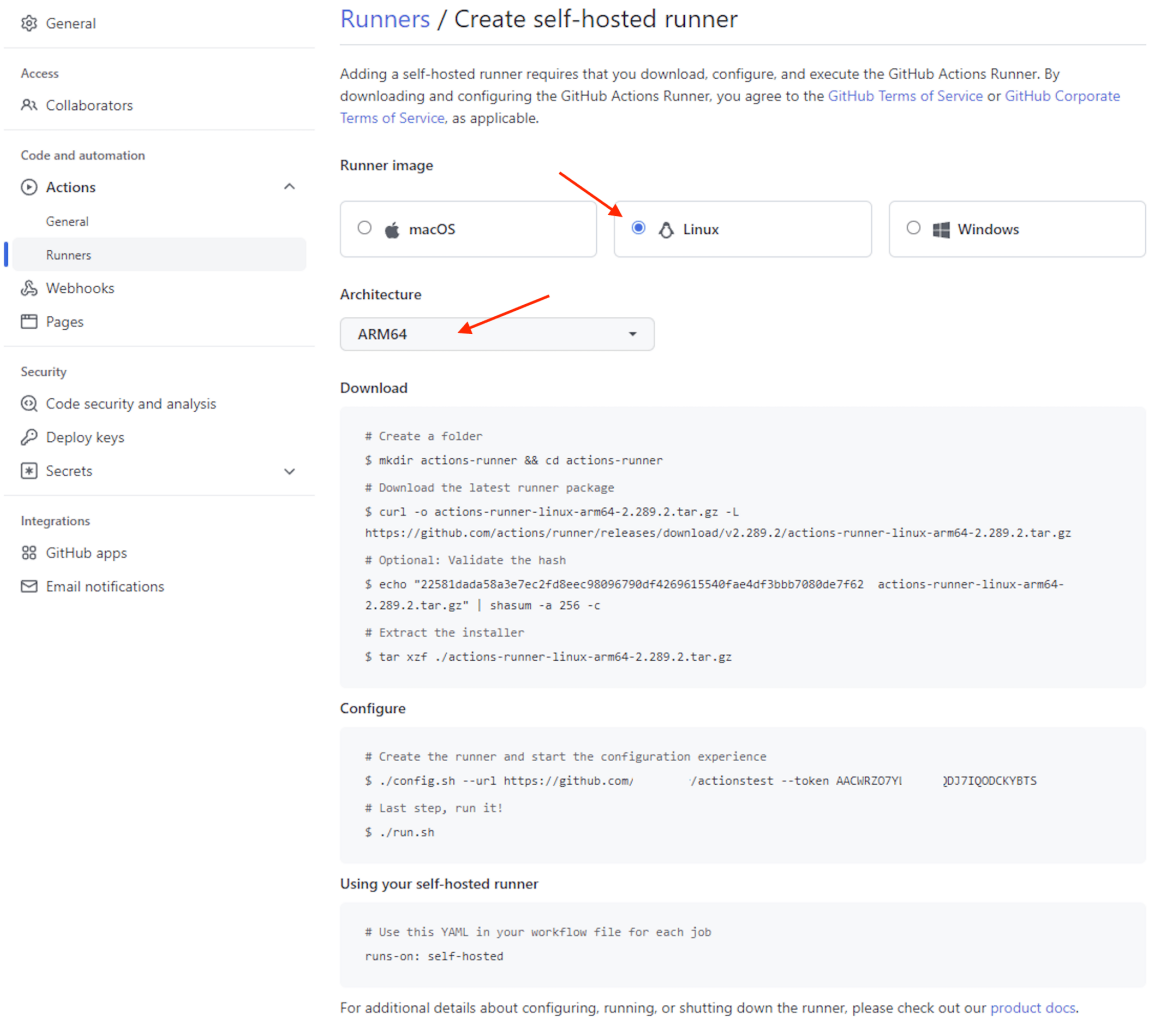
Figure 8-16: API_TOKEN



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.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

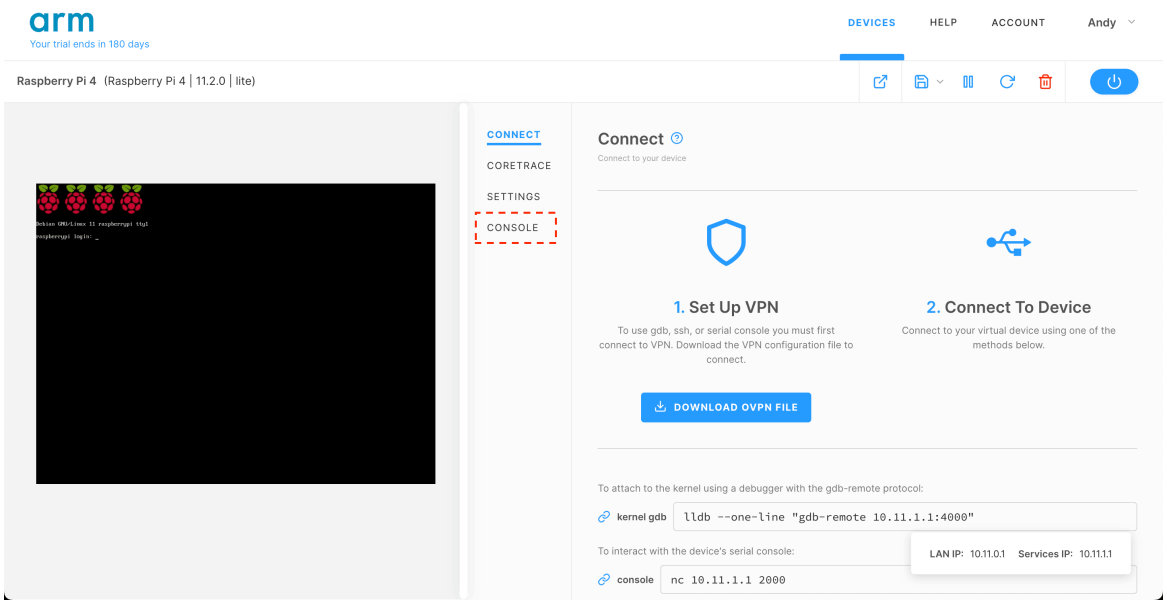
Figure 8-17: Adding self hosted runners



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

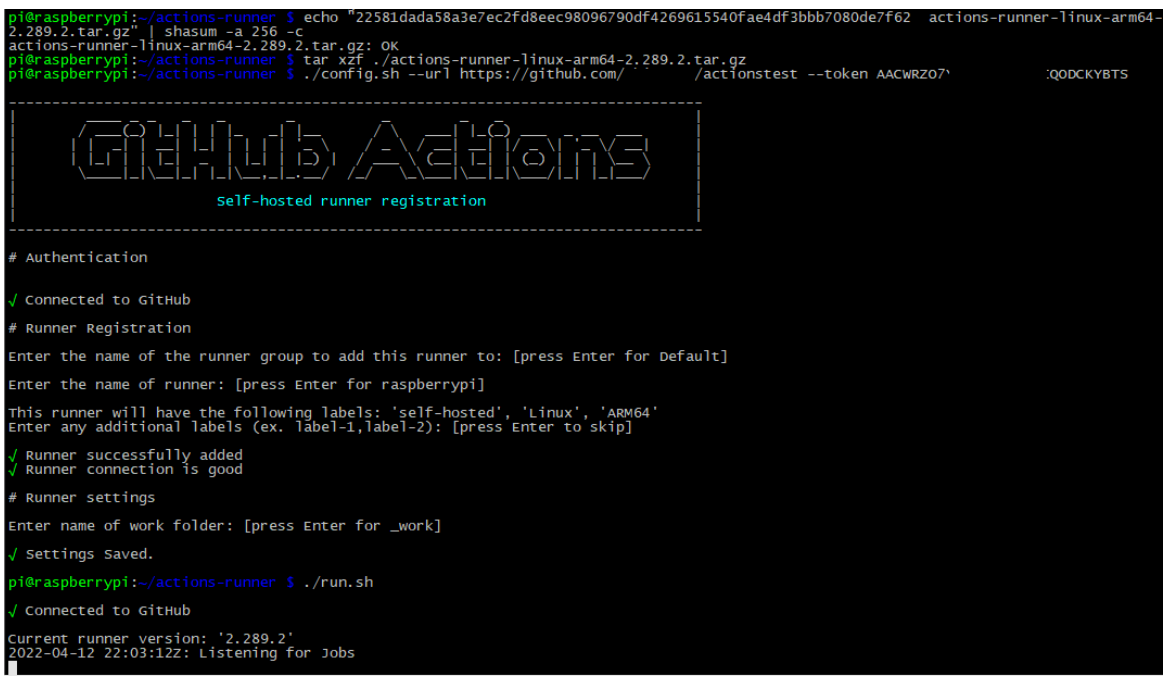
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 8-18: Select Console



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

Figure 8-19: 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 8-20: 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 8-21: 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 8-22: Run GITREMOTE=

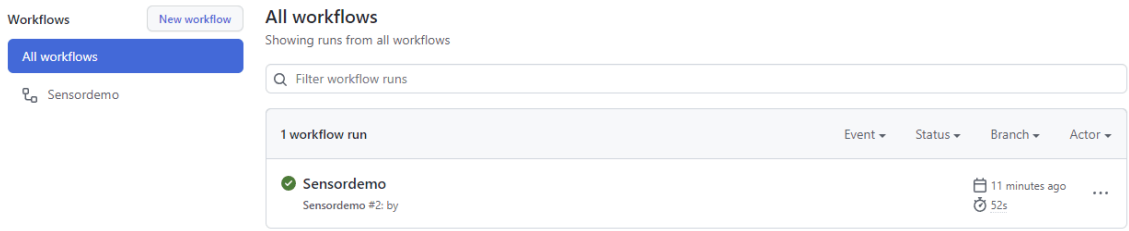
```
pi@raspberrypi:~/avh-api/javascript/sensordemo-docker $ make clean gitrepo GITREMOTE='https://github.com/        /actionstest.git'
rm -rf gitrepo
git archive HEAD --prefix="gitrepo/" | tar -xf -
cp -a ../api gitrepo/api
cd gitrepo; \
  git init; \
  mkdir -p .github/workflows; \
  mv github_workflow.yml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

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

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

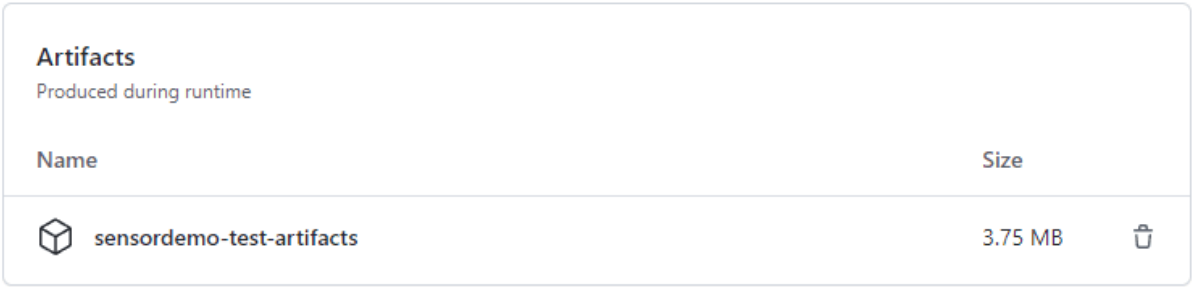
Figure 8-23: Executed action



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 8-24: 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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

9. Tips and Troubleshooting

The following articles are available:

- Linux console is flooded with messages
- Capture input from the Console
- VPN keeps disconnecting
- Deleting your AVH account

9.1 Linux console is flooded with messages

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.

9.2 Capture input from the Console

Learn how to capture character and newline console input.

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);
```

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');
```

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 © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

9.3 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.

9.4 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.

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.

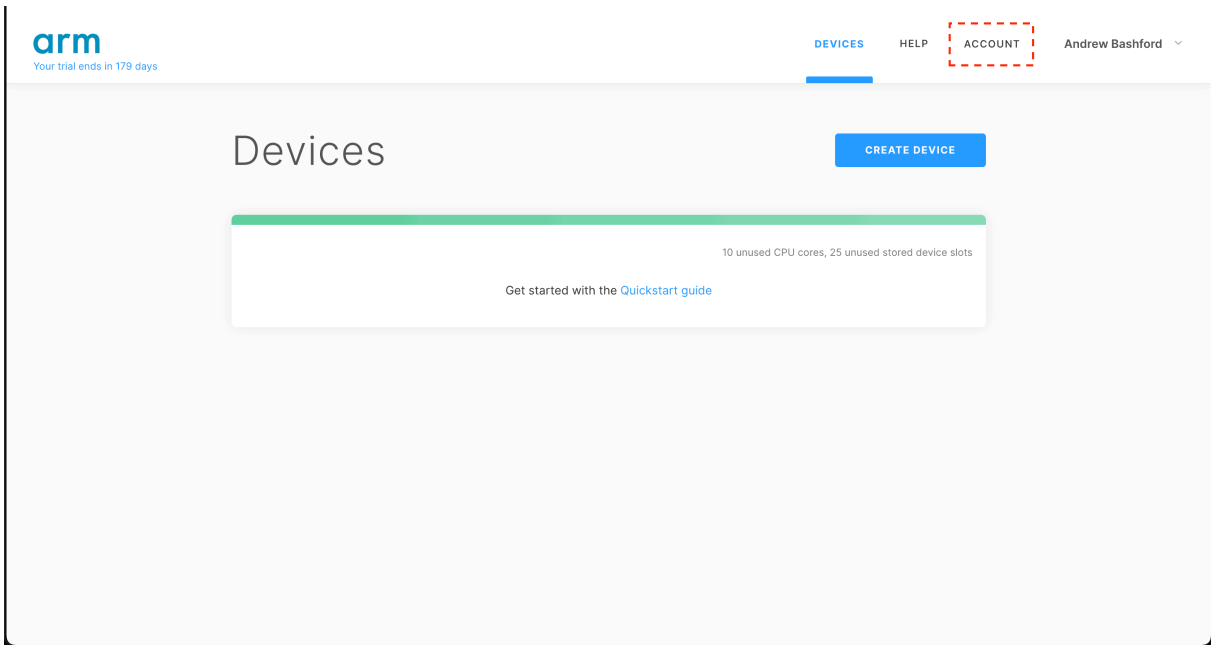
If your Arm account is deleted or suspended your access to AVH will be removed.

Deleting your account

To delete your account:

1. To delete your account go to ACCOUNT in the navbar.

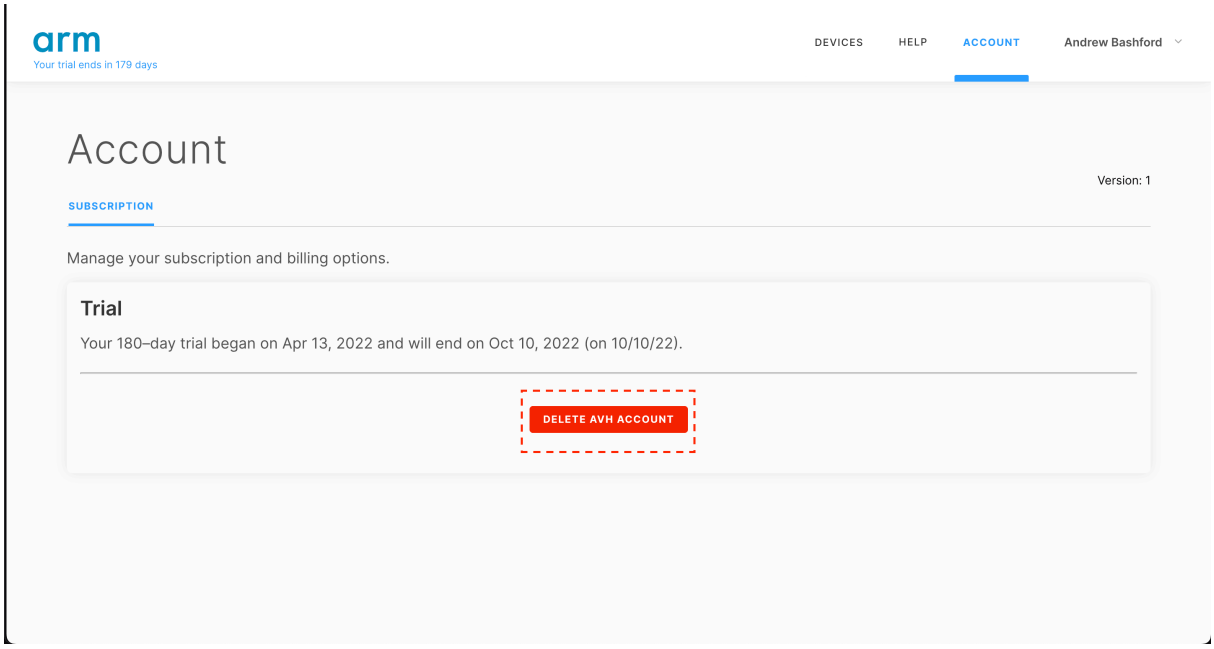
Figure 9-1: Account



2. Click DELETE AVH ACCOUNT.

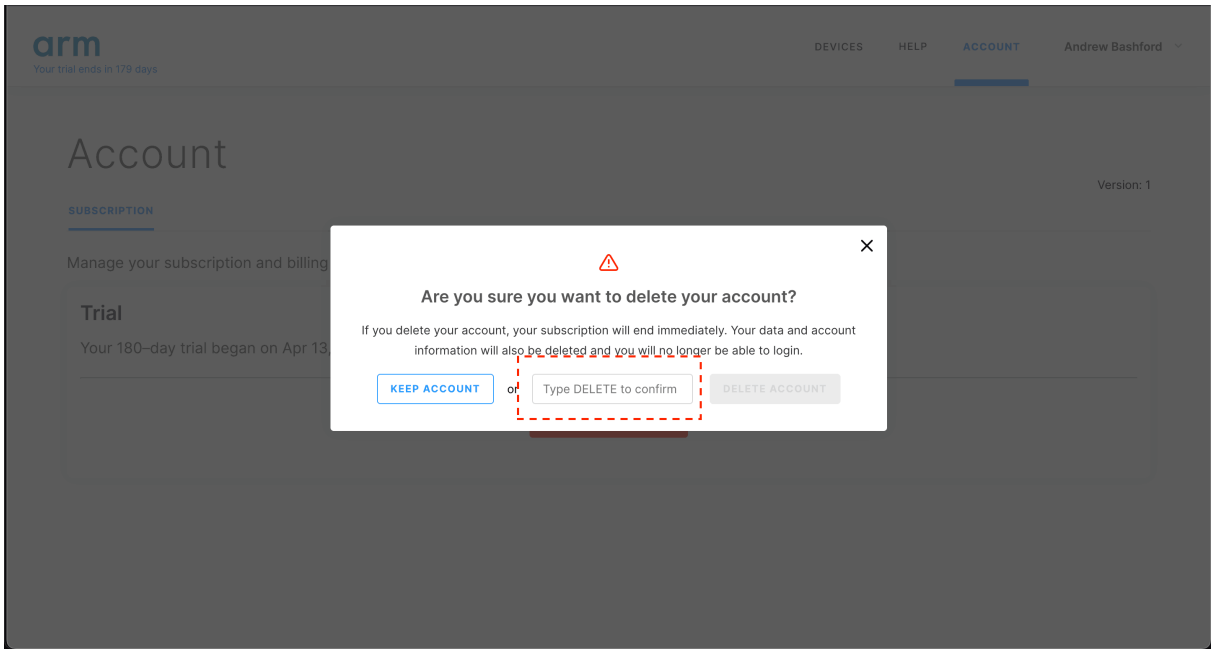
Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 9-2: Delete AVH account



3. Type DELETE into the input, it is case sensitive.

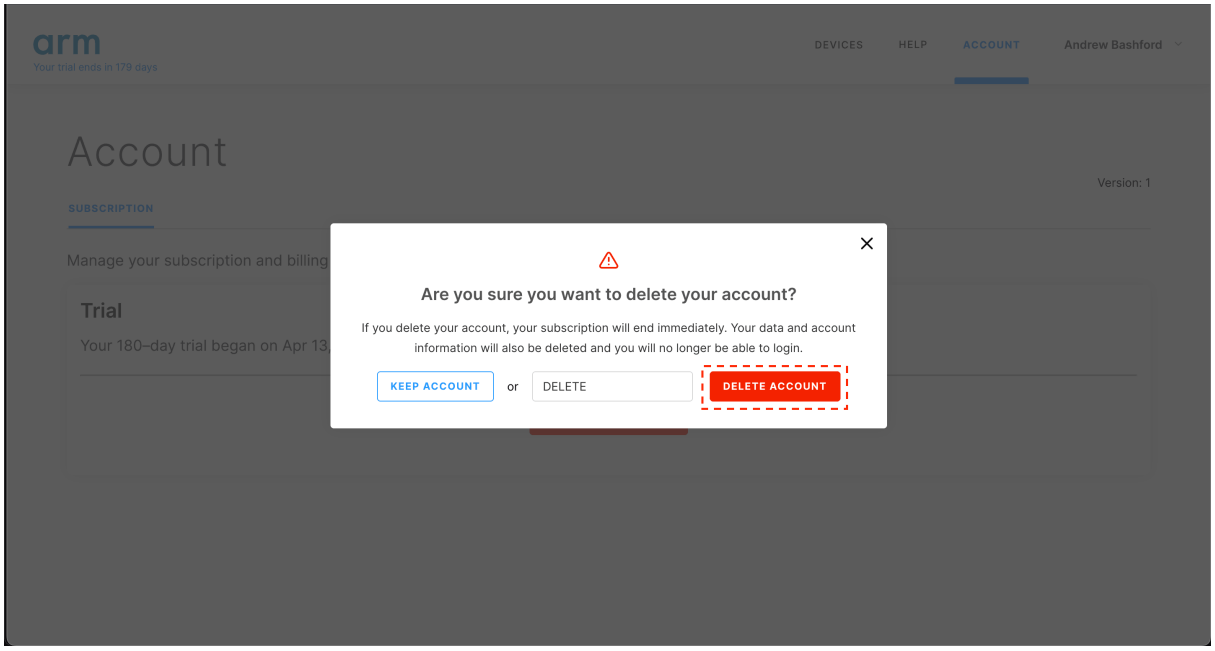
Figure 9-3: Type DELETE



4. Click DELETE ACCOUNT to confirm deleting your account. All of your data will be permanently deleted.

Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 9-4: Click DELETE ACCOUNT



Copyright © 2022–2023 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential