# Arm Virtual Hardware Custom Firmware

Version 1.0

## Knowledge Base Article

**Non-Confidential**



**Issue 01**

107683_0100_01_en

# Arm Virtual Hardware Custom Firmware

**Knowledge Base Article**



## Release information

### Document history

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

## Proprietary Notice

## Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

## Product Status

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

## Feedback

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

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

## Inclusive language commitment

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

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

# Contents

# 1. Firmware

Information about firmware on AVH:

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

## 1.1 Understanding Firmware on AVH

Learn how firmwares work on AVH.

### 1.1.1 Overview

AVH supports two types of firmware images.

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

### 1.1.2 Zip package

The format of the zip package looks like this -

Required files:

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

### 1.1.3 Storage

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

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

### 1.1.4 What does the Info.plist look like?

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

Example:

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

### 1.1.5 Formats of raw firmware

There are three supported formats for executable firmware:

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

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

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

### 1.1.6 Sample Firmware Packaging Script for Raspberry Pi

Sample firmware packaging script for Raspberry Pi:

```
#!/bin/bash
```

```
set -e
[ -d pi ] || mkdir pi
cd pi
# Grab the raspberry pi firmware
[ -f 2022-01-28-raspios-bullseye-arm64.zip ] || wget https://
downloads.raspberrypi.org/raspios_arm64/images/raspios_arm64-2022-01-28/2022-01-28-
raspios-bullseye-arm64.zip
rm -rf {nand,devicetree,kernel,Info.plist,boot,rootfs}
unzip 2022-01-28-raspios-bullseye-arm64.zip
mv 2022-01-28-raspios-bullseye-arm64.img nand

# Mount the firmware image and extract the kernel and device tree
LO="$(losetup -f)"
mkdir boot
losetup -P "${LO}" nand
mount "${LO}p1" boot
# Enable ssh
touch boot/ssh
cp boot/bcm2711-rpi-4-b.dtb devicetree
zcat boot/kernel8.img > kernel
umount boot
rm -rf boot
mkdir rootfs
mount "${LO}p2" rootfs
# Don't run dhcpcd on docker interfaces
echo 'denyinterfaces veth*' >> rootfs/etc/dhcpcd.conf
umount rootfs
rm -rf rootfs
losetup -d "${LO}"

# create the Info plist that describes the model image
cat << EOF > Info.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Type</key>
    <string>iot</string>
    <key>UniqueIdentifier</key>
    <string>Raspberry Pi OS Desktop</string>
    <key>DeviceIdentifier</key>
    <string>rpi4b</string>
    <key>Version</key>
    <string>11.2.0</string>
    <key>Build</key>
    <string>desktop</string>
</dict>
</plist>
EOF

# zip image and its ready for use
zip -r ../rpi4b-11.2-desktop.zip Info.plist nand devicetree kernel ramdisk.img
```

## 1.2 Package Ubuntu Server Firmware for AVH

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

AVH supports running custom Linux firmware packages.

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

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

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

### 1.2.1 Firmware Package Contents

A proper Ubuntu Server firmware package file contains the following:

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

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

### 1.2.2 Firmware Packaging Script

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

```
#!/bin/bash
set -e
mkdir rpi_ubuntu_server_firmware
cd rpi_ubuntu_server_firmware
# Download and extract the "Ubuntu 22.04.1 Server for RPi" image
wget https://cdimage.ubuntu.com/releases/22.04.1/release/ubuntu-22.04.1-
preinstalled-server-arm64+raspi.img.xz
xz -dv ubuntu-22.04.1-preinstalled-server-arm64+raspi.img.xz
mv ubuntu-22.04.1-preinstalled-server-arm64+raspi.img nand
# Attach the image file
LO="$(losetup -f)"
losetup -P "${LO}" nand
# Mount partition 1 to directory boot
mkdir boot
mount "${LO}p1" boot
# Copy the rpi4b devicetree
cp boot/bcm2711-rpi-4-b.dtb devicetree
# Extract the Linux kernel
zcat boot/vmlinuz > kernel
# Extract initrd
lz4 -d boot/initrd.img initrd.cpio
umount boot
rm -r boot/
losetup -d "${LO}"
mkdir ramdisk
cd ramdisk
# Extract initrd
cat ../initrd.cpio | cpio -idm
rm ../initrd.cpio
# Add reboot conditional to init before matching string
sed -i '/Move virtual filesystems over to the real filesystem/i \
if /bin/grep init_resize /proc/cmdline; then\
/bin/reboot -f\
```

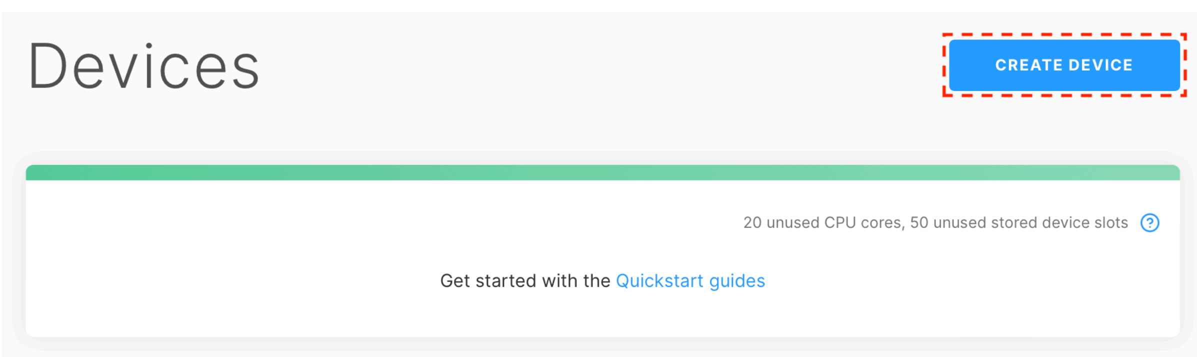
```
fi\
' init
# Create ramdisk.img from directory
find . | cpio -o -H newc -R root:root | lz4 -l > ../ramdisk.img
cd ..
rm -r ramdisk/
# Create the Info.plist file
cat << EOF > Info.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Type</key>
<string>iot</string>
<key>UniqueIdentifier</key>
<string>Ubuntu Server on RPi</string>
<key>DeviceIdentifier</key>
<string>rpi4b</string>
<key>Version</key>
<string>22.04.1</string>
<key>Build</key>
<string>Ubuntu Server</string>
</dict>
</plist>
EOF
zip -rm ../rpi4b-ubuntu-server.zip Info.plist nand devicetree kernel ramdisk.img
cd ..
rm -r rpi_ubuntu_server_firmware/
```

### 1.2.3 Install the Package

To install the package:

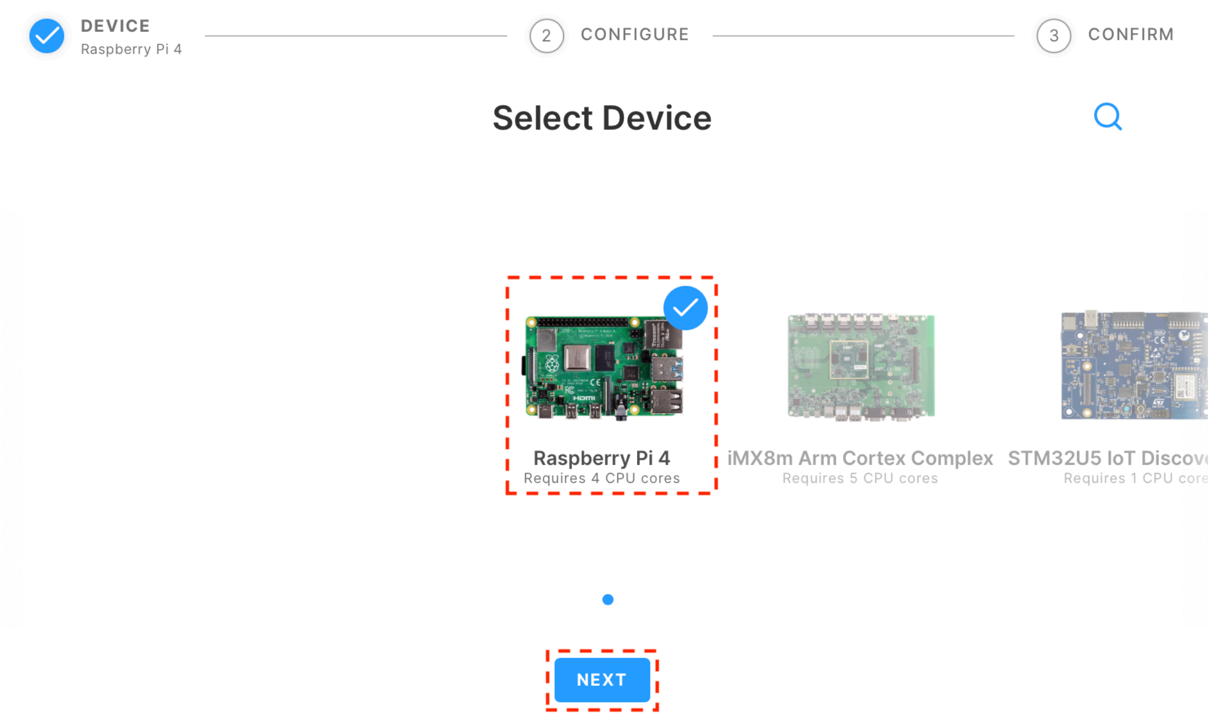
1. Run the above shell script on your local Linux environment to create the firmware package `rpi4b-ubuntu-server.zip`.
2. On the AVH web interface, click CREATE DEVICE.

**Figure 1-1: Create device**




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

**Figure 1-2: Select device**




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

**Figure 1-3: Upload the custom firmware package**

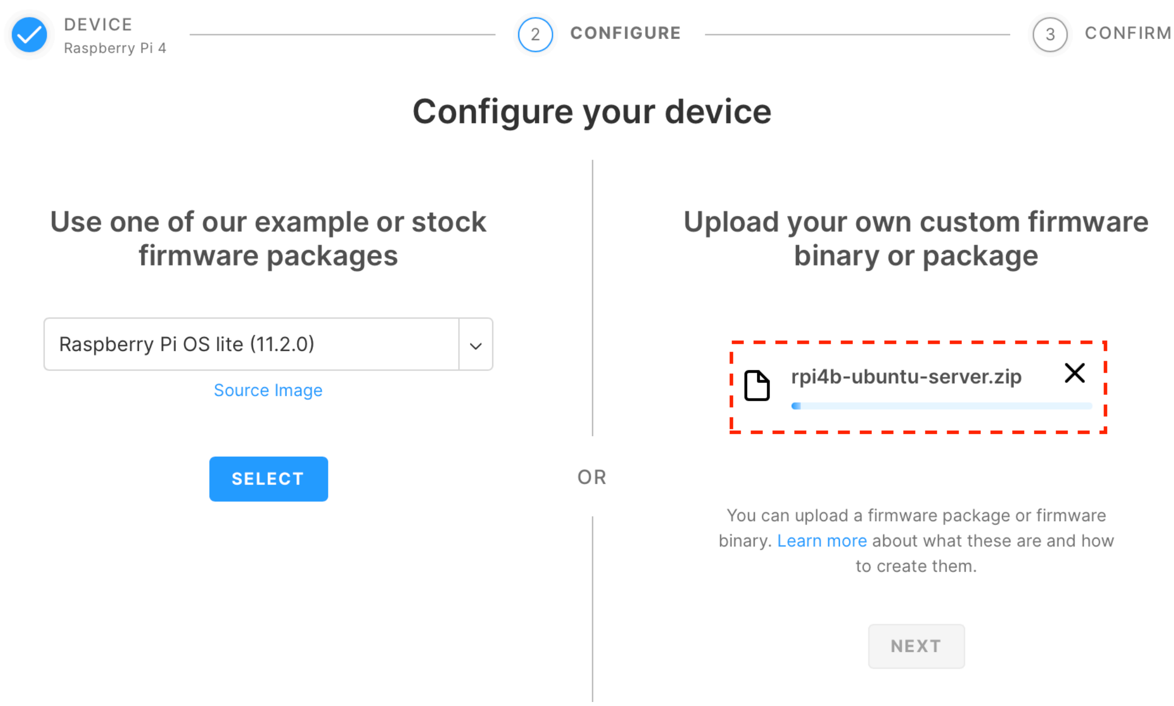
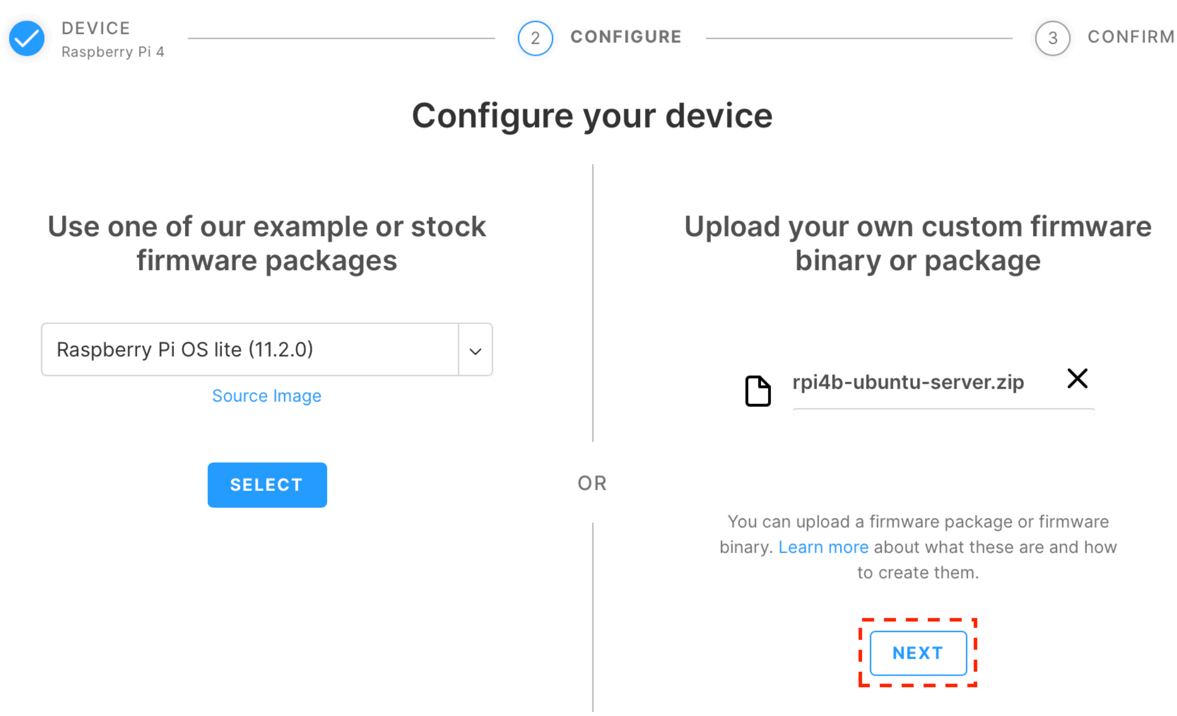
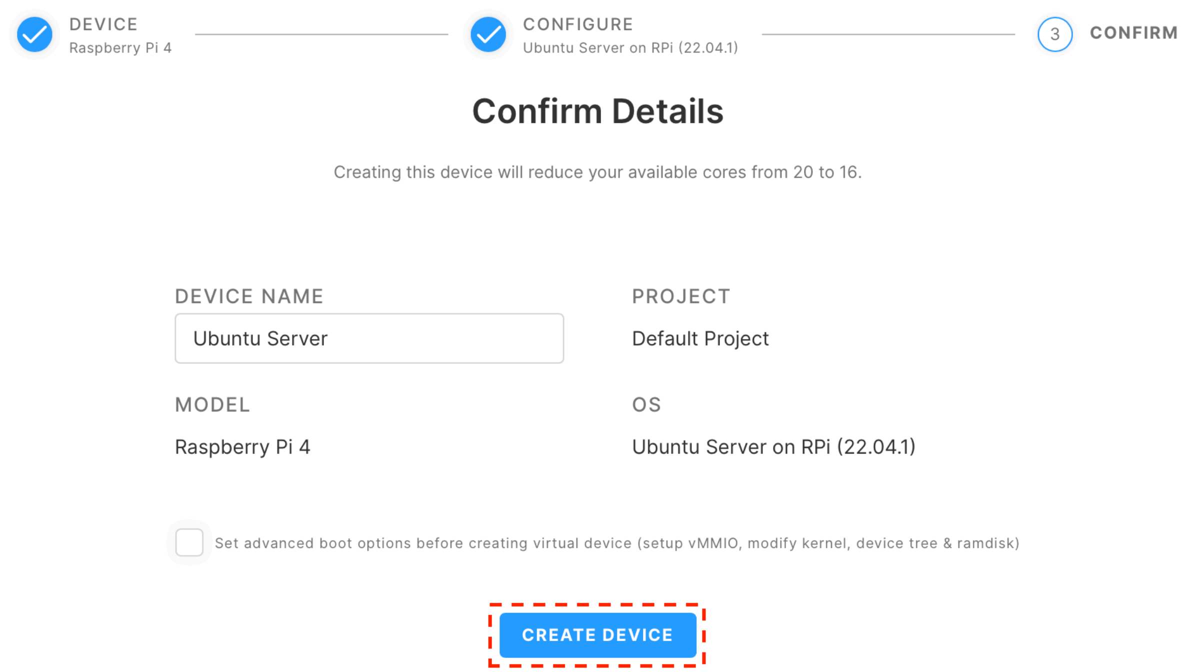



**Figure 1-4: Click Next**



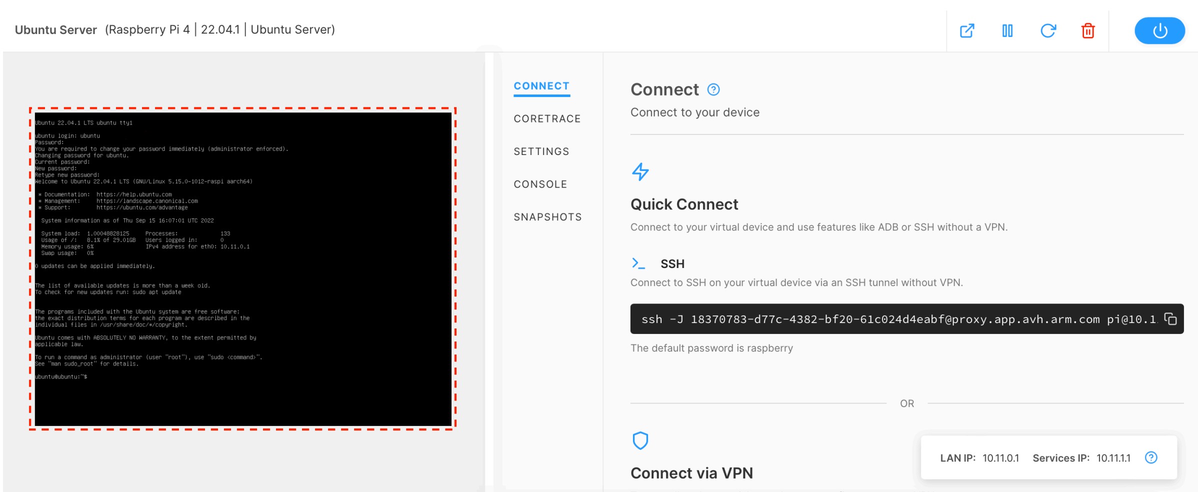

5. Create the device without enabling advanced boot options.

**Figure 1-5: Create device**




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

**Figure 1-6: Login with the default credentials**




Note

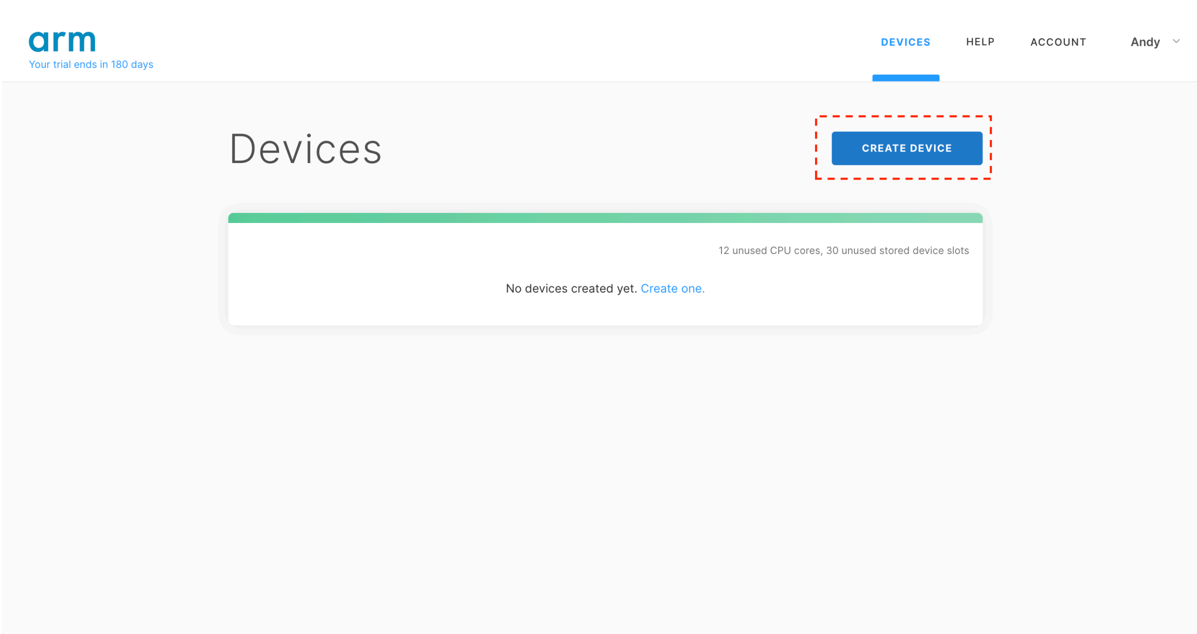
Since we are using a custom configuration, you will need to adjust the SSH commands under the CONNECT tab to use `ubuntu` as the username instead of the default (`pi`).

## 1.3 Upload custom firmware using the UI

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

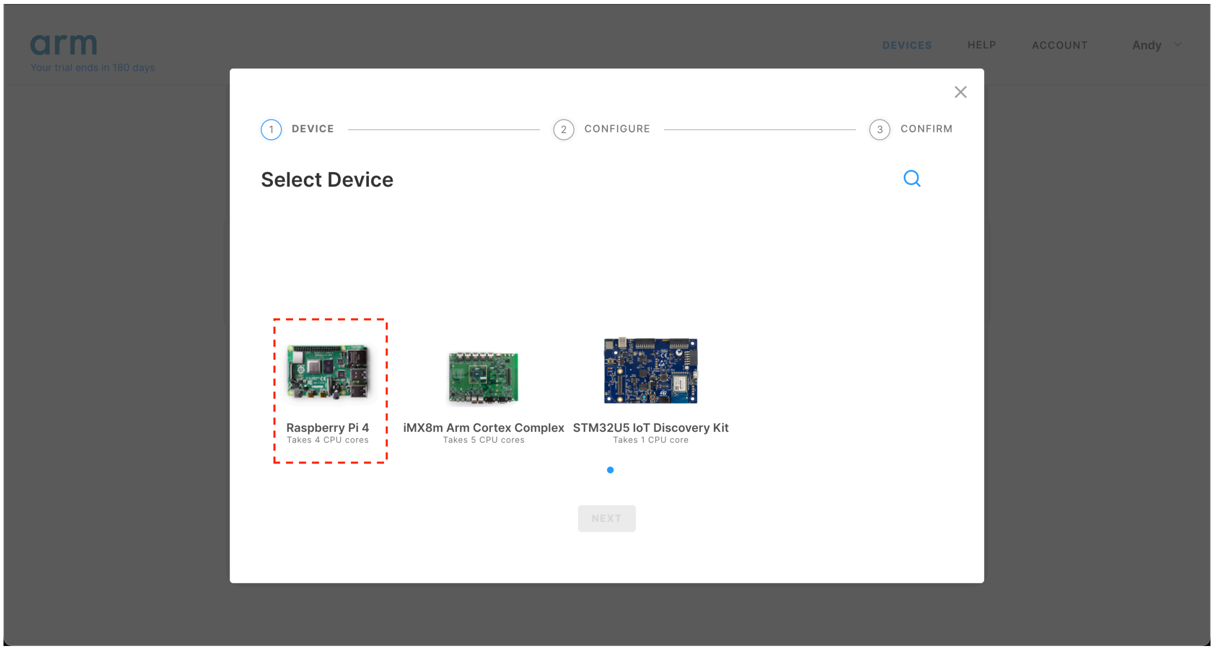
1. Log in to AVH with your Arm user account. Log in to https://avh.arm.com/ If you do not have or need to create an Arm account register at https://developer.arm.com/
2. When logged in you land on the AVH Devices page. Click CREATE DEVICE.

**Figure 1-7: Create device**



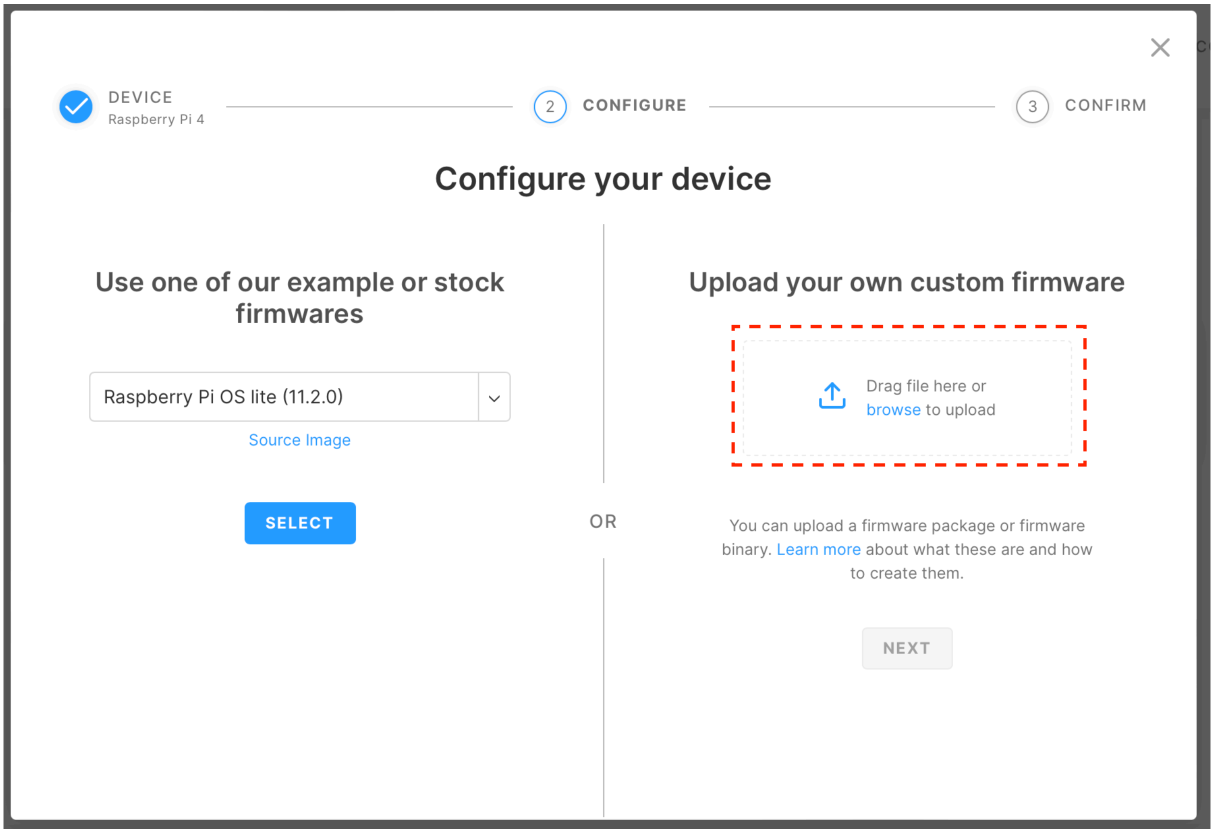

3. Click on the Raspberry Pi 4 to select it. Then click NEXT.

**Figure 1-8: Select device**



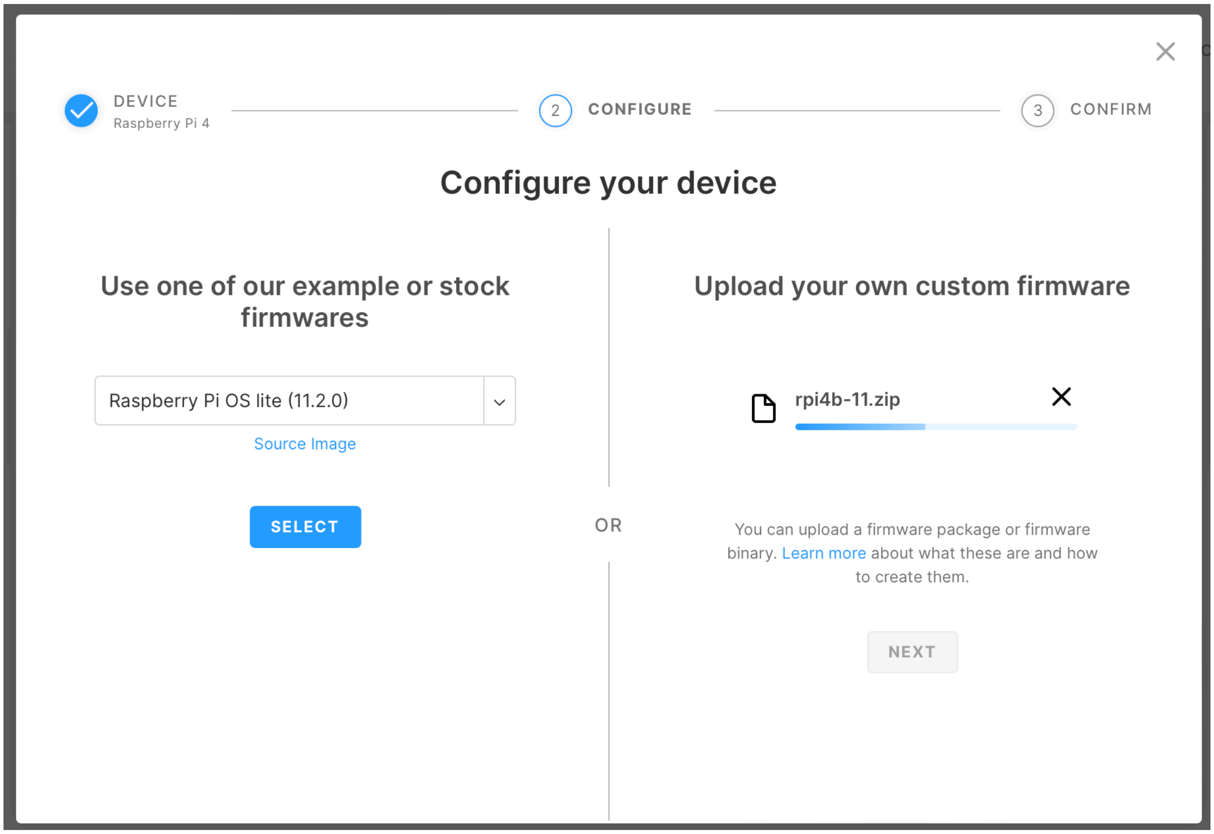

4. In this next step, you'll see a list of pre-configured examples/stock firmware or you can upload your own custom firmware. Either drag&drop your custom firmware into the Drag file here box or use the browse option, navigate and select the desired custom firmware.

**Figure 1-9: Configure your device**



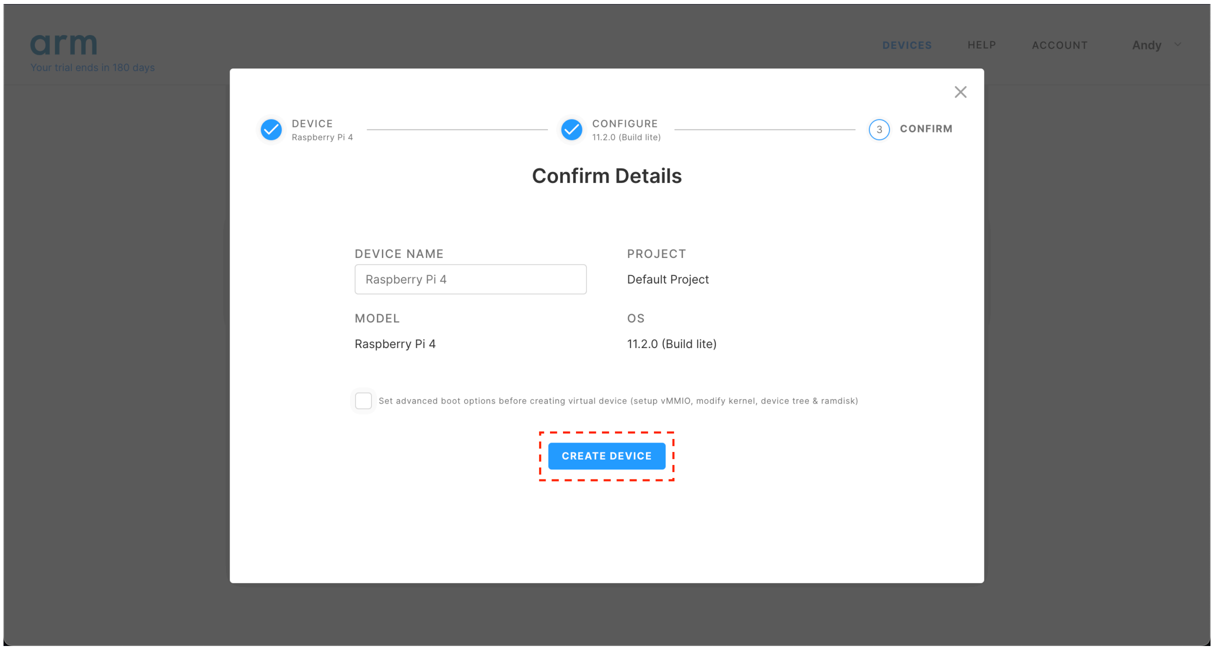

5. You will see the progress of your custom firmware uploading. This might take a while, depending on the size of custom firmware you're uploading. Once the upload is complete, click NEXT.

**Figure 1-10: Configure your device**




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

**Figure 1-11: Confirm details**




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

# 1.4 Package Ubuntu Desktop Firmware for AVH

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

AVH supports running custom Linux firmware packages.

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

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

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

## 1.4.1 Firmware Package Contents

A proper Ubuntu Desktop firmware package file contains the following:

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

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

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

### 1.4.2 Firmware Packaging Script

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

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

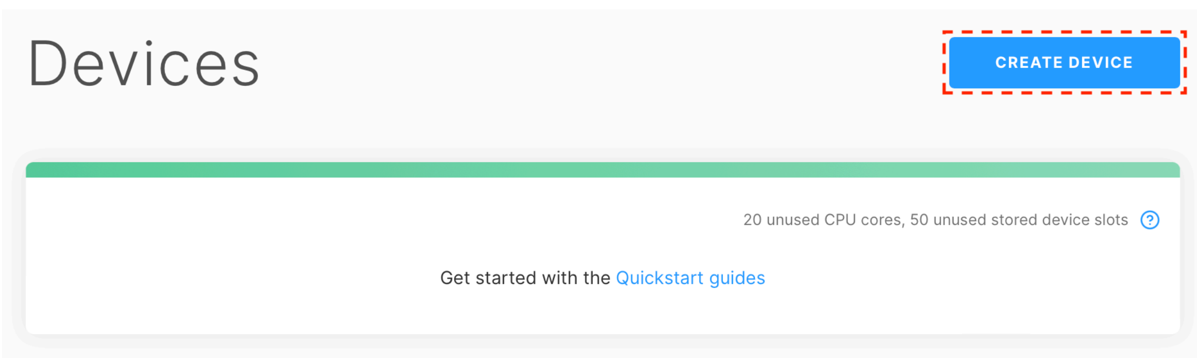
```
<key>Type</key>
<string>iot</string>
<key>UniqueIdentifier</key>
<string>Ubuntu Desktop on RPi</string>
<key>DeviceIdentifier</key>
<string>rpi4b</string>
<key>Version</key>
<string>22.04.1</string>
<key>Build</key>
<string>Ubuntu Desktop</string>
</dict>
</plist>
EOF
zip -rm ../rpi4b-ubuntu-desktop.zip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

### 1.4.3 Install the Package

To install the package:

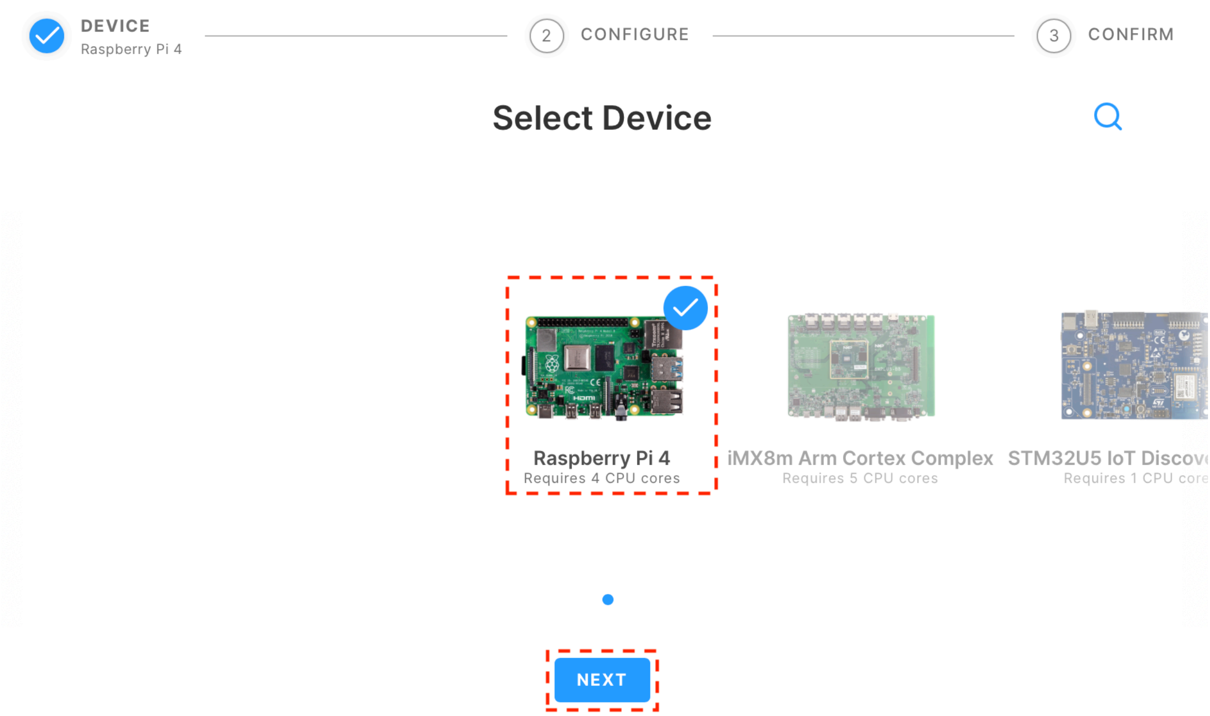
1. Run the above shell script on your local Linux environment to create the firmware package `rpi4b-ubuntu-desktop.zip`.
2. On the AVH web interface, click CREATE DEVICE.

   **Figure 1-12: Create device**

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

**Figure 1-13: Select device**




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

**Figure 1-14: Upload the custom firmware package**

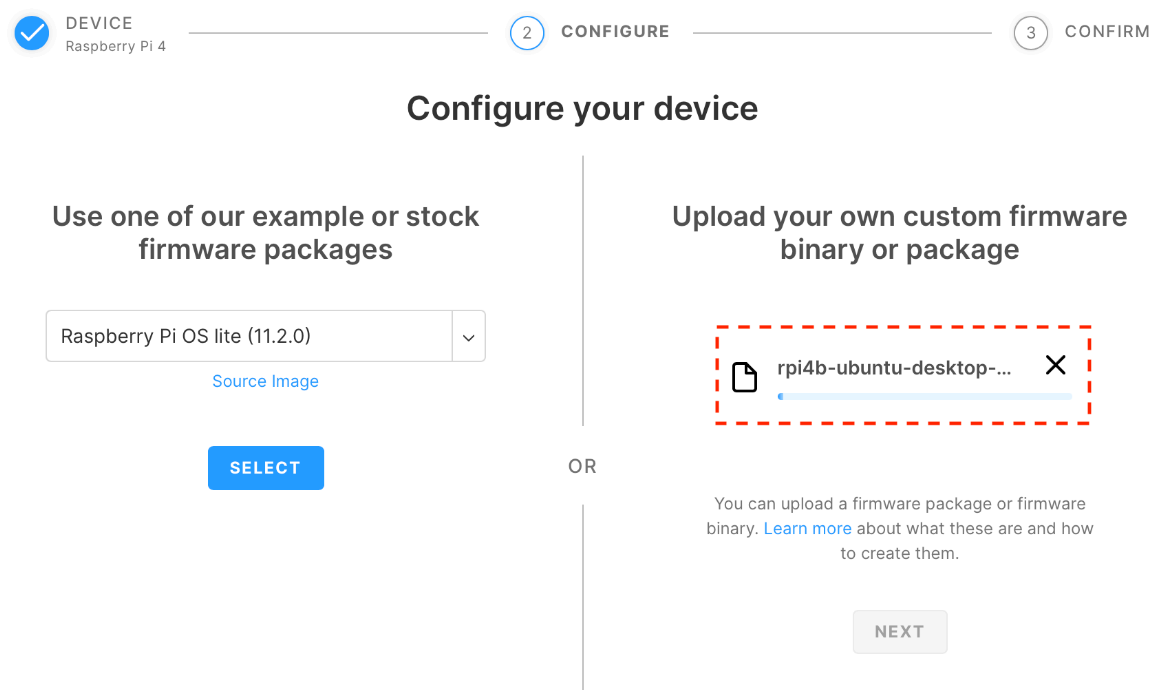
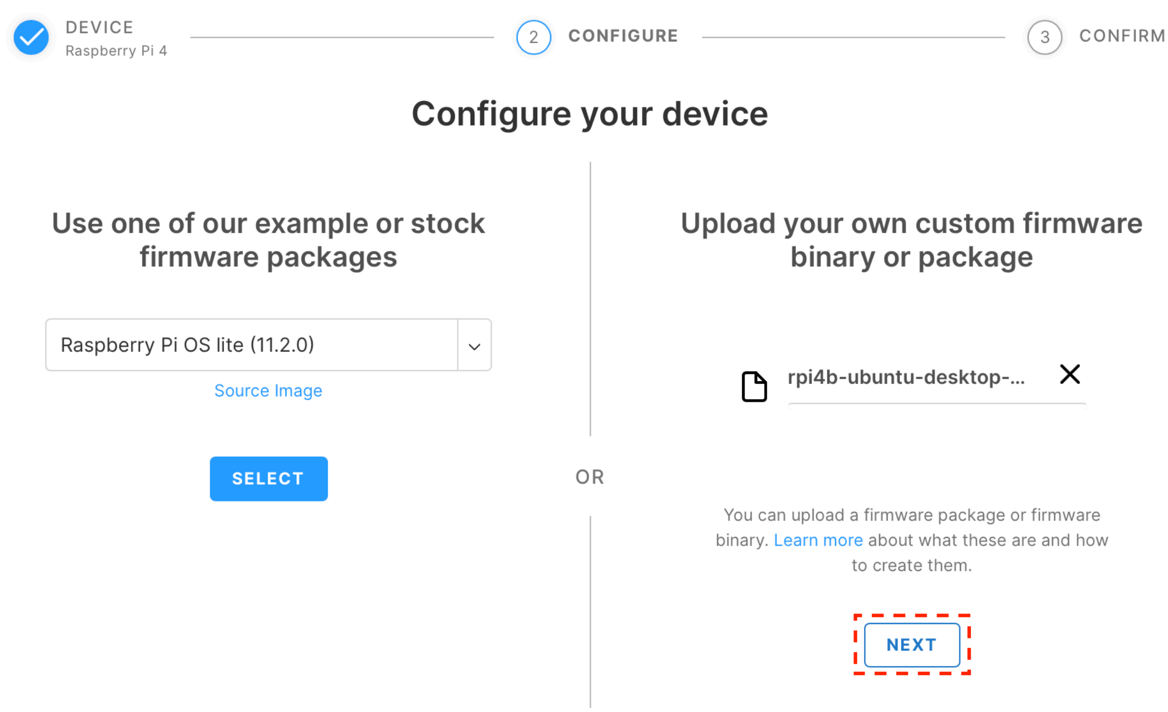
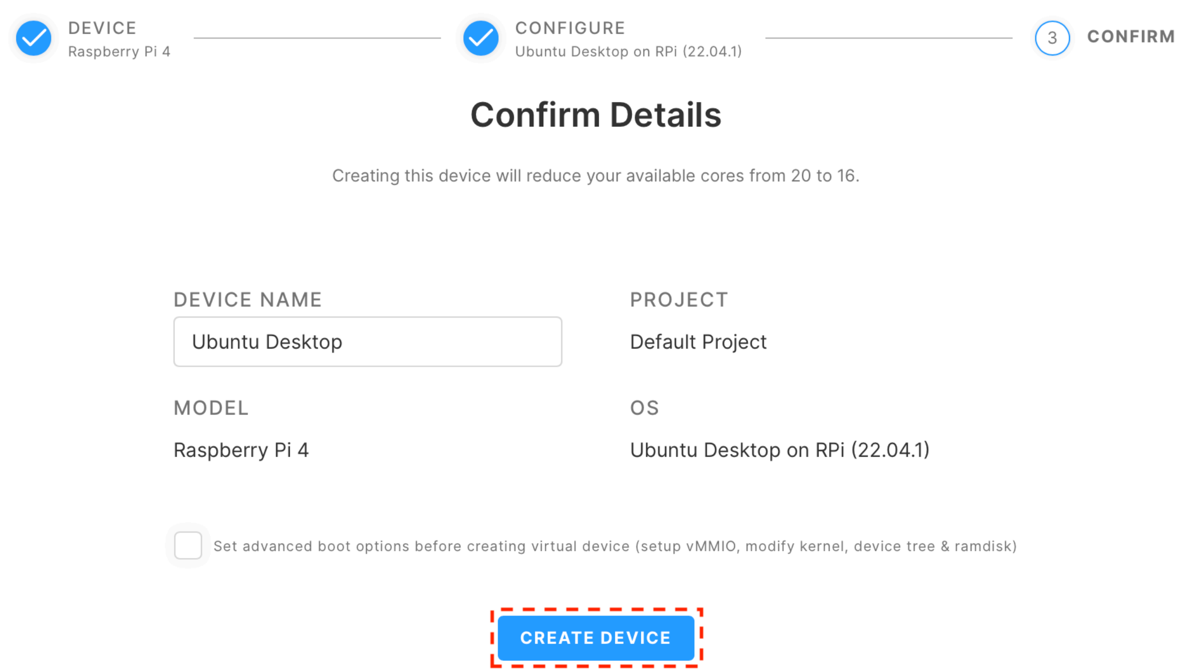



**Figure 1-15: Click Next**



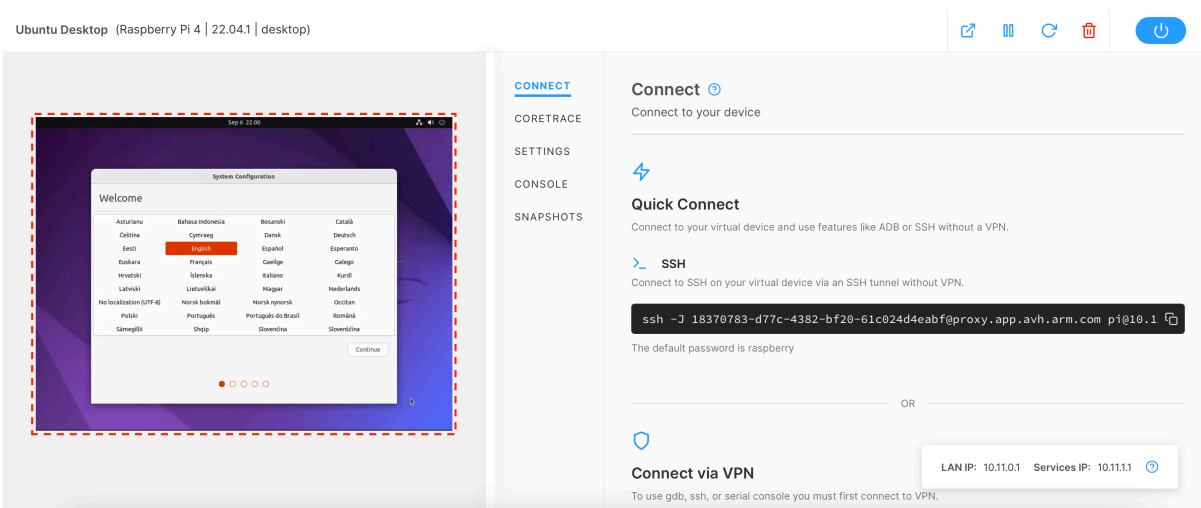

5. Create the device without enabling advanced boot options.

**Figure 1-16: Create device**




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

**Figure 1-17: Login with the default credentials**




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

# 2. Storage Files

Information about storage files for different platforms:

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

## 2.1 Storage Files for Raspberry Pi 4

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

### 2.1.1 Required files

The following files are required:

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

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

## 2.2 Storage Files for iMX8m Arm Cortex Complex

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

### 2.2.1 Required files

The following files are required:

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

- max: 138412032 bytes

---

Note

Includes 4M for ECC

---

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

---

Note

Includes 4M for ECC

---

Find out more at the NXP website.

## 2.3 Storage Files for STM32U5 IoT Discovery Kit

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

### 2.3.1 Required files

The following files are required:

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

Note

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

- flash-otp - OTP memory in Flash block
    - block_size: 512
    - blocks: 1
    - max: 512 bytes
- sdcard - SD card
    - block_size: 512
    - blocks: 1953125
    - max: 1000000000 bytes

### 2.3.2 Optional files

The following files are optional:

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

Note

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

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

Note

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