Matter on Ubuntu

Canonical Group Ltd

Aug 26, 2024

CONTENTS

Matter is an open source connectivity standard for smart home. It is a collection of protocols for connecting compatible devices in a secure and reliable way.

This documentation is for building and running Matter devices on Ubuntu. It provides guidance, examples and reference.

Matter on Ubuntu simplifies and streamlines the work of building matter applications, by providing rich and established developer tools. It is intended to serve developers and maintainers of Linux-based Matter applications.

CHAPTER

ONE

IN THIS DOCUMENTATION

1.1 Get started

For your first experience with Matter on Ubuntu, you can use easily-available hardware to explore and understand its basics.

1.1.1 Build your first Matter device with a Raspberry Pi

This tutorial walks you through setting up a Matter Lighting device on a Raspberry Pi. We will use the matter-pi-gpio-commander snap which contains a lighting app built on top of the Matter SDK. The application supports communication over WiFi/Ethernet as well as Thread.

Hardware

In this guide, we use the following hardware:

- A **PC** running Ubuntu 22.04
- A Raspberry **Pi** 4B with Ubuntu Server 22.04 (64-bit) - but it also works on Ubuntu Core 22
- A 10mm 3v LED

Since we use a large 3v LED, we can directly connect it to the GPIO. We connect the LED to GPIO 4 (pin 7) and GND (pin 9). Refer here, for the Raspberry Pi pinout.

Setup

In this section, we'll install and configure the matter-pi-gpio-commander snap.

SSH to the Raspberry Pi and install the snap:

```
sudo snap install matter-pi-gpio-commander
```

> **Tip**
>
> Pre-release versions of this snap are available in different channels.

The application uses a custom-device interface to access the GPIO. The interface should automatically connect upon installation. Let's verify that by looking at the snap's connections:

```
$ sudo snap connections matter-pi-gpio-commander
Interface      Plug                                        Slot                                ␣
↪          Notes
...
custom-device  matter-pi-gpio-commander:custom-gpio      matter-pi-gpio-commander:custom-
↪gpio-dev  -
...
```

Note

On **Ubuntu Core**, the `custom-gpio` interface doesn't auto connect (See issue #67). Connect manually:

```
sudo snap connect matter-pi-gpio-commander:custom-gpio \
                  matter-pi-gpio-commander:custom-gpio-dev
```

Configure the GPIO

Set the GPIO pin/line to 4:

```
sudo snap set matter-pi-gpio-commander gpio=4
```

Apart from the GPIO pin, you may need to configure the GPIO chip. The chip number is set to `0` by default which is suitable for us, since we use a Raspberry Pi 4.

The default works for Raspberry Pi 4 and all older arm64 Pis.
For Raspberry Pi 5, the chip should be set to `4` to use `/dev/gpiochip4`:

```
sudo snap set matter-pi-gpio-commander gpiochip=4
```

Any value other than `0` and `4` gets rejected as they aren't expected on Raspberry Pis nor supported by the `custom-gpio` interface. The check can be disabled by setting `gpiochip-validation=false` option.

Test the GPIO

The application is almost ready to start and join a Matter network. But before doing so, it is better to test it locally to see if we can control the GPIO via the app:

```
$ sudo matter-pi-gpio-commander.test-blink
GPIO: 4
GPIOCHIP: 0
Setting GPIO 4 to Off
Setting GPIO 4 to On
Setting GPIO 4 to Off
^C
```

Take a look at the logs and the LED. If there are no errors and the LED blinks every half second, we are ready to proceed! Use Ctrl+C to stop the application.

CLI flags

The application supports a range of CLI arguments implemented by the Matter SDK.

For a list of supported CLI arguments, execute `matter-pi-gpio-commander.help`.

```
$ matter-pi-gpio-commander.help
Usage: /snap/matter-pi-gpio-commander/x3/bin/lighting-app [opti

GENERAL OPTIONS

  --ble-device <number>
       The device number for CHIPoBLE, without 'hci' prefix, can be found by hciconfig.

  --wifi
       Enable WiFi management via wpa_supplicant.

  --thread
       Enable Thread management via ot-agent.

  ...
```

For example, to override the default passcode:

```
sudo snap set matter-pi-gpio-commander args="--passcode 1234"
```

Multiple CLI arguments can be concatenated with a space. For example:

```
sudo snap set matter-pi-gpio-commander args="--passcode 1234 --ble-device 1"
```

DNS-SD

The application uses DNS-SD to register itself and be discovered over the local network. To allow that, we need to install the dependencies and grant access via a snap interface:

Ubuntu Server / Desktop

Ubuntu Core

```
sudo apt update
sudo apt install avahi-daemon
sudo snap connect matter-pi-gpio-commander:avahi-control :avahi-control
```

The interface connection is between the `matter-pi-gpio-commander` snap and the system.

```
sudo snap install avahi
sudo snap connect matter-pi-gpio-commander:avahi-control avahi:avahi-control
```

On Ubuntu Core, the interface connection is between the `matter-pi-gpio-commander` snap and the `avahi` snap.

Thread

If using Thread instead of WiFi/Ethernet, set `--thread` as a CLI argument:

```
sudo snap set matter-pi-gpio-commander args="--thread"
```

Note that Thread communication requires a Thread Radio Co-Processor (RCP) and an OpenThread Border Router (OTBR) agent enabling that communication over DBus.

To allow communication with the OTBR Snap for Thread management, connect the following interface:

```
sudo snap connect matter-pi-gpio-commander:otbr-dbus-wpan0 \
                  openthread-border-router:dbus-wpan0
```

You may refer to *this guide* for setting up OTBR on Ubuntu.

Bluetooth Low Energy (BLE)

To allow the device to advertise itself over Bluetooth Low Energy:

Ubuntu Server / Desktop

Ubuntu Core

```
sudo apt update
sudo apt install bluez
sudo snap connect matter-pi-gpio-commander:bluez :bluez
```

```
sudo snap install bluez
sudo snap connect matter-pi-gpio-commander:bluez bluez:service
```

Start the application

Now, let's start the application service:

```
sudo snap start matter-pi-gpio-commander
```

You can monitor the logs with:

```
sudo snap logs -n 100 -f matter-pi-gpio-commander
```

Keep it running in a dedicate terminal window. We will commission and control the application in the next section.

Setup Chip Tool

We need a Matter Controller to commission and control the device. We will use Chip Tool which is a CLI Matter controller.

Install the dependencies:

```
sudo apt update
sudo apt install avahi-daemon # for DNS-SD
sudo apt install bluez # for bluetooth
```

Install the chip-tool snap on the PC:

```
sudo snap install chip-tool
```

Commission

WiFi/Ethernet

Thread

Assuming the Pi and PC are connected to the same network, we should be able to commission the device by discovering its IP address via DNS-SD.

To pair:

```
chip-tool pairing onnetwork 110 20202021
```

where:

- `110` is the node id being assigned to this device
- `20202021` is the default setup passcode

Assuming that Chip Tool and the Thread Border Router are on the same network, we should be able to discover the Border Router via DNS-SD.

Pair the Thread device over Bluetooth LE

```
chip-tool pairing ble-thread 110 hex:<active-dataset> 20202021 3840
```

where:

- `110` is the assigned node ID for the app.
- `<active-dataset>` is the Thread network's Active Operational Dataset in hex, taken using the `ot-ctl` command before.
- `20202021` is the PIN code set on the app.
- `3840` is the discriminator ID.

If this doesn't work, it may be because it has taken too long to reach this step and the device has stopped listening to commissioning requests. Try restarting the application with `sudo snap restart matter-pi-gpio-commander`.

Control

There are a few ways to control the device. The `toggle` command is stateless and simplest.

```
chip-tool onoff toggle 110 1
```

1.2 How-to guides

Guides for specific real-world problems and tasks.

1.2.1 How to commission and control Matter devices with Chip Tool

Chip Tool is an open source Matter Controller with a command-line interface (CLI). It is useful for development and testing of Matter devices from a Linux machine.

Chip Tool offers a wide range of capabilities, ranging from device commissioning and control to setup and operational payload generation and parsing.

This document guides you through setting up and configuring Chip Tool using a Snap. This makes it extremely easy to securely run and use the tool on Linux.

Install

First and foremost, make sure to have SnapD installed. It is pre-installed on some distributions such as Ubuntu. Refer to installing SnapD for details.

Install the dependencies:

Ubuntu Server / Desktop

Ubuntu Core

```
sudo apt update
sudo apt install bluez avahi-daemon
```

```
sudo snap install bluez avahi
```

Install the Chip Tool snap:

```
sudo snap install chip-tool
```

> **Tip**
>
> Pre-release versions of Chip Tool are available in different channels.

Once installed, the application should be available as `chip-tool` on your machine.

The snap restricts the app's access to only the necessary resources on the host. This access is managed via snap interface connections.

By default, the snap auto connects the following interfaces:

- network to access the host network
- network-bind to listen on a port (Chip Tool's interactive mode)
- avahi-observe to discover devices over DNS-SD
- bluez to communicate with devices over Bluetooth Low Energy (BLE)

To verify the interface connections, run: `snap connections chip-tool`

Commission

WiFi/Ethernet

Thread

Discover using DNS-SD and pair:

```
chip-tool pairing onnetwork 110 20202021
```

where:

- `110` is the node id being assigned to the device
- `20202021` is the pin code set on the device

To commission a Thread device advertising itself over BLE, you need an active Thread network (formed by a Thread Border Router) and a Bluetooth interface. Chip Tool discovers the Thread Border Router via DNS-SD and communicates with it over WiFi/Ethernet network.

Here, we assume the use of OpenThread implementation of the Thread Border Router.

> **Note**
>
> You may need to enable experimental Bluetooth support to allow BLE advertising and discovery.

1. Obtain the Active Operational Dataset for the existing Thread network:

Snap

Docker

Native

```
sudo openthread-border-router.ot-ctl dataset active -x
```

```
sudo docker exec -it otbr sh -c "sudo ot-ctl dataset active -x"
```

```
sudo ot-ctl dataset active -x
```

The dataset is encoded in hex and contains several values including the network's security key.

2. Discover over Bluetooth Low Energy (BLE) and pair:

```
chip-tool pairing ble-thread 110 hex:0e08...f7f8 20202021 3840
```

where:

- `110` is the node id being assigned to the device
- `0e08...f7f8` is the Thread network credential operational dataset, truncated for readability.
- `20202021` is the pin code set on the device
- `3840` is the discriminator id

> **Note**
>
> It is also possible to commission a Thread device using a manual pairing code, without using Bluetooth Low Energy (BLE).

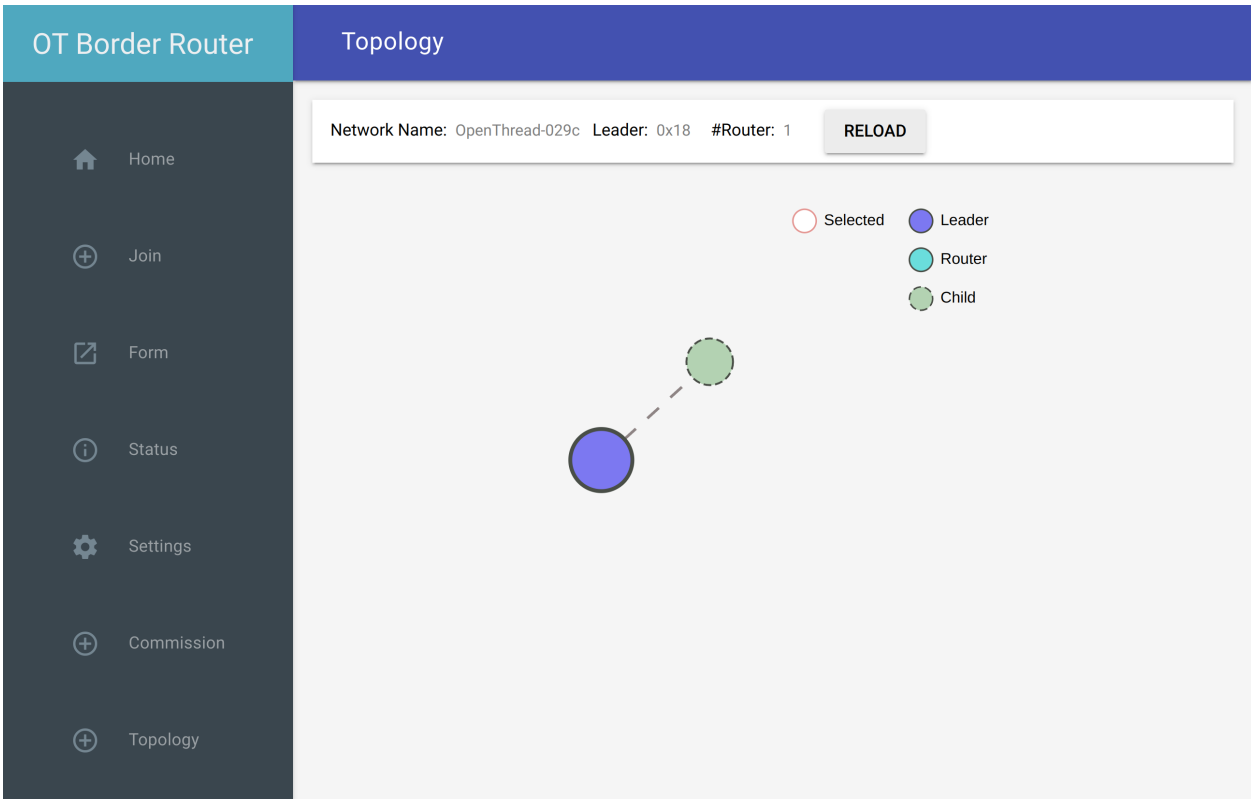
```
chip-tool pairing code-thread 110 hex:0e08...f7f8 34970112332
```

where:

- `34970112332` is the short manual pairing code

Details on how to use this can be found in the Matter documentation.

3. (optional) On the OTBR GUI, under the Topology tab, you can now see the two connected Thread nodes:



Control

Toggle:

```
chip-tool onoff toggle 110 1
```

where:

- `onoff` is the matter cluster name
- `on/off/toggle` is the command name.
- `110` is the node id of the app assigned during the commissioning
- 1 is the endpoint of the configured device

More reading

This documentation covered only some of the common scenarios for commissioning and controlling Matter devices via Chip Tool. The project provides a guide with various usage examples.

However, for a complete list of sub-commands and options, it is best to use the tool's usage instructions using the terminal.

1.2.2 How to set up OpenThread Border Router on Ubuntu

The OpenThread Border Router (OTBR) is an open source Thread Border Router implementation.

A Thread Border Router acts as a gateway between Thread and other IP networks (e.g. WiFi, Ethernet).

In this how to, we will go through the steps to quickly setup OTBR on Ubuntu.

> **Note**
>
> In order to setup OTBR, we need a Radio Co-Processor (RCP) and another IP networking interface such as WiFi or Ethernet.
>
> Moving forward, the assumption is to have the RCP available as a device at `/dev/ttyACM0`.

We use the (unofficial) OTBR Snap because it makes the setup, configuration, and maintenance significantly simpler. Let's get started:

Install the OTBR snap

Install the latest version from the Snap Store:

```
sudo snap install openthread-border-router
```

> **Tip**
>
> Pre-release versions of OpenThread Border Router are available in different channels.

Grant access to resources

Ubuntu Server / Desktop

Ubuntu Core

Install the dependencies:

```
sudo apt update
sudo apt install bluez avahi-daemon
```

Connect the following interfaces:

```
# Allow setting up the firewall
sudo snap connect openthread-border-router:firewall-control
# Allow access to USB Thread Radio Co-Processor (RCP)
```

(continues on next page)

(continued from previous page)

```
sudo snap connect openthread-border-router:raw-usb
# Allow setting up the networking
sudo snap connect openthread-border-router:network-control
# Allow controlling the Bluetooth devices
sudo snap connect openthread-border-router:bluetooth-control

# Allow device discovery over Bluetooth Low Energy
sudo snap connect openthread-border-router:bluez
# Allow DNS-SD registration and discovery
sudo snap connect openthread-border-router:avahi-control
```

Install the dependencies:

```
sudo snap install bluez avahi
```

Connect the following interfaces:

```
# Allow setting up the firewall
sudo snap connect openthread-border-router:firewall-control
# Allow access to USB Thread Radio Co-Processor (RCP)
sudo snap connect openthread-border-router:raw-usb
# Allow setting up the networking
sudo snap connect openthread-border-router:network-control
# Allow controlling the Bluetooth devices
sudo snap connect openthread-border-router:bluetooth-control

# Allow device discovery over Bluetooth Low Energy
sudo snap connect openthread-border-router:bluez bluez:service
# Allow DNS-SD registration and discovery
sudo snap connect openthread-border-router:avahi-control avahi:avahi-control
```

Configure the OTBR snap

The configurations are set via Snap Configuration Options and passed on the services.

First, check the default configurations:

```
$ sudo snap get openthread-border-router
Key        Value
autostart  false
infra-if   wlan0
radio-url  spinel+hdlc+uart:///dev/ttyACM0
thread-if  wpan0
```

Then, override them based on the local setup.

For example, if the networking interface is `eth0`, change it as follows:

```
snap set openthread-border-router infra-if="eth0"
```

Start OTBR

By default the services are disabled and not started. After everything is configured, we can start and enable the services:

```
sudo snap start --enable openthread-border-router
```

Use the following command to query and follow the logs:

```
snap logs -n 100 -f openthread-border-router
```

> **Note**
>
> To start and enable via a Gadget snap, set *autostart* snap configuration to *true*.

Form a Thread network

Use the CTL tool to initialize the Thread network:

```
sudo openthread-border-router.ot-ctl dataset init new
sudo openthread-border-router.ot-ctl dataset commit active
sudo openthread-border-router.ot-ctl ifconfig up
sudo openthread-border-router.ot-ctl thread start
```

Alternatively, these steps could be performed with the GUI at http://localhost:80. Please refer to the instructions here to configure and form, join, or check the status of a Thread network using the GUI.

Controlling a Thread device

To commission and control a Matter Thread device, you can use Chip Tool; refer to *How to commission and control Matter devices with Chip Tool*.

1.2.3 How to run Matter applications with Thread networking on Ubuntu

This is a tutorial on setting up and running Matter applications that use Thread for networking on Ubuntu. We will scope the tutorial to Matter applications built using the Matter SDK and OpenThread Border Router (OTBR).

Prerequisites

- Two amd64/arm64 machines, with:
  - Ubuntu Server/Desktop 22.04 or newer
  - Thread Radio Co-Processor (RCP)

In this tutorial, we'll use the following:

- Machine A
  - Ubuntu Desktop 23.10 amd64
  - Nordic Semiconductor nRF52840 dongle, using the OpenThread (OT) RCP firmware
- Machine B (Raspberry Pi 4)

- Ubuntu Server 22.04 arm64
- Nordic Semiconductor nRF52840 dongle, using the OT RCP firmware

Machine A will host the Border Router (OTBR) and Matter Controller. Machine B will act as the Matter device and run the Matter application and another instance of OTBR. The second OTBR instance will not act as a Border Router, but rather as an agent which complements the Matter application for Thread networking capabilities.

Note

The API version of OTBR agents running on Machines A and B must match!

In this tutorial, we've used the following:

Component	Upstream Commit/Version	API Version	snap channel
Matter lighting app	connectedhomeip `6b01cb9`	-	-
OTBR snap	ot-br-posix `thread-reference-20230119`	6	latest/edge
OTBR RCP firmware	ot-nrf528xx `00ac6cd`	6	-

1. Set up Border Router on Machine A

Refer to *How to set up OpenThread Border Router on Ubuntu* to set up and configure OTBR.

Then form a Thread network, using the following commands:

```
sudo openthread-border-router.ot-ctl dataset init new
sudo openthread-border-router.ot-ctl dataset commit active
sudo openthread-border-router.ot-ctl ifconfig up
sudo openthread-border-router.ot-ctl thread start
```

These steps could also be performed with the Web GUI, served by default at http://localhost:80. Please refer to the instructions here to form, join, or check the status of a Thread network using the GUI.

The Thread network is now ready for new joiners. Head over to Machine B to setup the Matter application.

2. Run OTBR on Machine B

The OTBR Agent is required for adding Thread networking capabilities to the Matter application. The Matter app communicates with OTBR Agent via the DBus Message Bus.

Similar to Machine A, set up and configure OTBR by following: *How to set up OpenThread Border Router on Ubuntu*.

On Machine B, connecting the `avahi-control` interface isn't required as this OTBR Agent's DNS-SD registration isn't needed.

Note that we do not form a Thread network on Machine B.

3. Run Matter Application on Machine B

The Matter Application can implement any Matter functionality. The requirement for this tutorial is that the application is created using the Matter SDK and runs on Ubuntu.

> **Tip**
>
> Most reference examples from the Matter SDK support Thread networking. For example, the lighting app for Linux can run in Thread mode by setting the `--thread` CLI argument. For more details, refer to its README.

The recommended option here is to use the Pi GPIO Commander application, which helps turn a Raspberry Pi into a Lighting Matter device . The application enables control of a GPIO pin via Matter.

There is a separate tutorial on setting up and running that application. Make sure to follow the Thread-related instructions to set it up and start the application. Then head back here to continue with Thread commissioning and control.

The tutorial for Pi GPIO Commander is available at: *Build your first Matter device with a Raspberry Pi*

4. Control the Matter Application from Machine A

Setup Matter Controller

First, install Chip Tool, a Matter Controller with a command-line interface:

```
sudo snap install chip-tool
```

Chip Tool depends on third-party services for DNS-SD and BLE discovery. If you don't already have them, install Avahi Daemon and BlueZ:

```
sudo apt update
sudo apt install avahi-daemon bluez
```

Pair the device

Get the OTBR operational dataset (OTBR network's credentials), for the network formed in previous sections:

```
sudo openthread-border-router.ot-ctl dataset active -x
```

Now, pair the Thread device over BLE:

```
chip-tool pairing ble-thread 110 hex:<active-dataset> 20202021 3840
```

where:

- `110` is the assigned node ID for the app.
- `<active-dataset>` is the Thread network's Active Operational Dataset in hex, taken using the `ot-ctl` command above.
- `20202021` is the PIN code set on the app.
- `3840` is the discriminator ID.

If this succeeds, skip to the controlling the device.

If it didn't work, it may be because it has taken too long to reach this step and the device has stopped advertising and listening to commissioning requests. Try restarting it on the application on Machine B with `sudo snap restart matter-pi-gpio-commander`.

Control the device

There are a few ways to control the device. The `toggle` command is stateless and the simplest:

```
chip-tool onoff toggle 110 1
```

To turn on and off:

```
chip-tool onoff on 110 1
chip-tool onoff off 110 1
```

1.2.4 How to create an OS Image with OpenThread Border Router

This tutorial walks you through creating an OS image that is pre-loaded with OpenThread Border Router (OTBR). We use Ubuntu Core as the Linux distribution because it is optimized for IoT and is secure by design. We configure the image and bundle the snapped version of OTBR. After the deployment, the snaps will continue to receive updates for the latest security and bug fixes.

Before starting, it is recommended to read the documentation on Ubuntu Core components and get familiar with various useful concepts.

Requirements:

- An amd64 Ubuntu development environment
- An amd64 machine as target for installing the new OS
- A Thread Radio Co-processor (RCP), connected to the target machine.

Used in this tutorial:

- Desktop computer running Ubuntu 23.10
- Intel NUC11TNH with 8GB RAM and 250GB NAND flash storage
- Nordic Semiconductor nRF52840 Dongle, connected to Intel NUC

We need the following tools on the development environment:

- snapcraft to manage keys in the store and build snaps
- yq to validate YAML files and convert them to JSON
- ubuntu-image v2 to build the Ubuntu Core image

Install them using the following commands:

```
sudo snap install snapcraft --classic
sudo snap install yq
sudo snap install ubuntu-image --classic
```

Create a custom gadget

Overriding the snap configurations upon installation is possible with a gadget snap.

The pc gadget is available as a pre-built snap in the store, however, in this chapter, we need to build our own to include custom configurations and interface connections.

We will create our custom gadget by staging the latest stable core22 gadget, and making the necessary modifications.

We need to create two files:

- `snapcraft.yaml`: the definition of our custom Gadget snap. We need this custom gadget in order to ship snap configurations on our image.
- `gadget.yaml`: the volumes layout for the image, list of snap default configuration, and interface connections.

Create the `snapcraft.yaml` file with the following content:

```
name: otbr-gadget
type: gadget
base: core22
version: test
summary: OpenThread Border Router Gadget
description: Custom gadget to configure the OpenThread Border Router snap

architectures:
  - build-on: [amd64]

grade: stable
confinement: strict

parts:
  gadget:
    plugin: nil
    stage-snaps:
      - pc/22
```

Then, create the `gadget.yaml` file :

```
# Default and unchanged volume definitions taken from
# https://github.com/snapcore/pc-gadget/blob/22/gadget/gadget-amd64.yaml
volumes:
  pc:
    schema: gpt
    # bootloader configuration is shipped and managed by snapd
    bootloader: grub
    structure:
      - name: mbr
        type: mbr
        size: 440
        update:
          edition: 1
        content:
          - image: mbr.img
        # This one should be removed in core24
        # or if we find a way to allow updates without keeping
        # all partitions
```

(continues on next page)

(continued from previous page)

```
      - name: BIOS Boot
        type: 21686148-6449-6E6F-744E-656564454649
        size: 1M
        offset: 1M
        update:
          edition: 2
      - name: ubuntu-seed
        role: system-seed
        filesystem: vfat
        # UEFI will boot the ESP partition by default first
        type: C12A7328-F81F-11D2-BA4B-00A0C93EC93B
        size: 1200M
        update:
          edition: 2
        content:
          - source: grubx64.efi
            target: EFI/boot/grubx64.efi
          - source: shim.efi.signed
            target: EFI/boot/bootx64.efi
      - name: ubuntu-boot
        role: system-boot
        filesystem: ext4
        type: 0FC63DAF-8483-4772-8E79-3D69D8477DE4
        # whats the appropriate size?
        size: 750M
        update:
          edition: 1
        content:
          - source: grubx64.efi
            target: EFI/boot/grubx64.efi
          - source: shim.efi.signed
            target: EFI/boot/bootx64.efi
      - name: ubuntu-save
        role: system-save
        filesystem: ext4
        type: 0FC63DAF-8483-4772-8E79-3D69D8477DE4
        size: 32M
      - name: ubuntu-data
        role: system-data
        filesystem: ext4
        type: 0FC63DAF-8483-4772-8E79-3D69D8477DE4
        size: 1G

# Custom snap configurations
defaults:
  # openthread-border-router
  AmezHbALZOOhReOPtKyluS5TJmySg15e:
      # Set to enable and start services
      autostart: true
      # For QEMU the networking interface should be enp0s2
      # For Intel NUC 11: enp88s0, enp89s0, or wlo1
      infra-if: enp88s0
```

(continues on next page)

(continued from previous page)

```
      thread-if: wpan0
      radio-url: "spinel+hdlc+uart:///dev/ttyACM0"

# Custom interface connections
connections:
  # openthread-border-router -> system
  - plug: AmezHbALZOOhReOPtKyluS5TJmySg15e:firewall-control
  - plug: AmezHbALZOOhReOPtKyluS5TJmySg15e:raw-usb
  - plug: AmezHbALZOOhReOPtKyluS5TJmySg15e:network-control
  - plug: AmezHbALZOOhReOPtKyluS5TJmySg15e:bluetooth-control

  # openthread-border-router -> avahi
  - plug: AmezHbALZOOhReOPtKyluS5TJmySg15e:avahi-control
    slot: dVK2PZeOLKA7vf1WPCap9F8luxTk9Oll:avahi-control

  # openthread-border-router -> bluez
  - plug: AmezHbALZOOhReOPtKyluS5TJmySg15e:bluez
    slot: JmzJi9kQvHUWddZ32PDJpBRXUpGRxvNS:service
```

Build the gadget snap:

```
snapcraft --verbose
```

This results in creating a snap named `otbr-gadget_test_amd64.snap`.

> **Note**
>
> You need to rebuild the snap every time you change the `gadget.yaml` file.

Create the model assertion

The model assertion is a digitally signed document that describes the content of the Ubuntu Core image.

Below is an example model assertion in YAML, describing a `core22` Ubuntu Core image:

```
type: model
series: '16'
model: ubuntu-core-22-amd64
architecture: amd64
base: core22

# Setting grade to dangerous to allow use of an unsigned gadget snap
grade: dangerous

# Since this is a custom model assertion, set the following to your developer ID
authority-id: <developer-id>
brand-id: <developer-id>

# Timestamp should be within your signature's validity period, in RFC3339 format
timestamp: '<timestamp>'
```

(continues on next page)

(continued from previous page)

```
snaps:
  - # This is our custom, dev gadget snap
    # It has no channel and id, because it isn't in the store.
    # We're going to build it locally and pass it to the image builder.
    name: otbr-gadget
    type: gadget
    # default-channel:
    # id:

  - name: pc-kernel
    type: kernel
    default-channel: 22/stable
    id: pYVQrBcKmBa0mZ4CCN7ExT6jH8rY1hza

  - name: snapd
    type: snapd
    default-channel: latest/stable
    id: PMrrV4ml8uWuEUDBT8dSGnKUYbevVhc4

  - name: core22
    type: base
    default-channel: latest/stable
    id: amcUKQILKXHHTlmSa7NMdnXSx02dNeeT

  # Apps
  - name: avahi
    type: app
    default-channel: 22/stable
    id: dVK2PZeOLKA7vf1WPCap9F8luxTk9Oll

  - name: bluez
    type: app
    default-channel: 22/stable
    id: JmzJi9kQvHUWddZ32PDJpBRXUpGRxvNS

  - name: openthread-border-router
    type: app
    default-channel: latest/edge
    id: AmezHbALZOOhReOPtKyluS5TJmySg15e
```

Refer to the model assertion documentation and inline comments for details. Create a `model.yaml` with the above content, replacing `authority-id`, `brand-id`, and `timestamp`.

> **Note**
>
> Unlike the official documentation which uses JSON, we use YAML serialization for the model. This is for consistency with all the other serialization formats in this tutorial. Moreover, it allows us to comment out some parts for testing or add comments to describe the details inline.

To find you developer ID, use the Snapcraft CLI:

```
$ snapcraft whoami
...
developer-id: <developer-id>
```

or get it from the Snapcraft Dashboard.

Follow these instructions to create a developer account, if you don't already have one.

Next, we need to sign the model assertion. Refer to this article for details on how to sign the model assertion. Here are the needed steps:

1) Create and register a key

```
snap login
snap keys
# Continue if you have no existing keys.
# You'll be asked to set a passphrase which is needed before signing
snap create-key otbr-uc-tutorial
snapcraft register-key otbr-uc-tutorial
```

We now have a registered key named `otbr-uc-tutorial` which we'll use later.

2) Sign the model assertion

We sign the model using the `otbr-uc-tutorial` key created and registered earlier.

The `snap sign` command takes JSON as input and produces YAML as output! We use the YQ app to convert our model assertion to JSON before passing it in for signing.

```
yq eval model.yaml -o=json | snap sign -k otbr-uc-tutorial > model.signed.yaml
```

This will produce a signed model named `model.signed.yaml`.

> **Note**
>
> You need to repeat the signing every time you change the input model, because the signature is calculated based on the model.

Build the Ubuntu Core image

We use `ubuntu-image` and set the path to:

- The signed model assertion YAML file.
- The locally built gadget snap.

```
ubuntu-image snap model.signed.yaml --verbose --validation=enforce \
    --snap otbr-gadget_test_amd64.snap
```

This downloads all the snaps specified in the model assertion and builds an image file called `pc.img`.

The image file is now ready to be flashed on a medium to create a bootable drive with the Ubuntu Core installer!

Install the Ubuntu Core image

The installation instructions are device specific. You may refer to Ubuntu Core section in this page. For example:

- Intel NUC - applicable to most computers with a secondary storage

A precondition to continue with some of the instructions is to compress `pc.img`. This speeds up the transfer and makes the input file similar to official images, improving compatibility with the official instructions.

To compress with the lowest compression rate of zero:

```
xz -vk -0 pc.img
```

A higher compression rate significantly increases the processing time and needed resources, with very little gain.

Now, follow the device specific instructions.

Continue to perform the OS initialization steps appearing *by default*.

Once the installation is complete, you will see the interface of the `console-conf` program. It will walk you through the networking and user account setup. You'll need to enter the email address of your Ubuntu account to create an OS user account with your registered username and have your SSH public keys deployed as authorized SSH keys for that user. If you haven't done so in the past, refer to the Creating your developer account documentation to add your SSH keys before doing this setup.

Read about system user assertion to know how the manual account setup looks like and how it can be automated.

Congratulations. The Ubuntu Core installation is complete and the device is ready for use. The OTBR services should be running and functional.

Sanity check

Now, let's verify that everything is in place and functional.

Connect to the machine over SSH:

```
$ ssh <ubuntu-one-username>@<device-ip>
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-91-generic x86_64)

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

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

 * Ubuntu Core:     https://www.ubuntu.com/core
 * Community:       https://forum.snapcraft.io
 * Snaps:           https://snapcraft.io

This Ubuntu Core 22 machine is a tiny, transactional edition of Ubuntu,
designed for appliances, firmware and fixed-function VMs.

If all the software you care about is available as snaps, you are in
the right place. If not, you will be more comfortable with classic
deb-based Ubuntu Server or Desktop, where you can mix snaps with
traditional debs. It's a brave new world here in Ubuntu Core!
```

(continues on next page)

(continued from previous page)

```
Please see 'snap --help' for app installation and updates.
```

List the installed snaps:

```
<user>@ubuntu:~$ snap list
Name                      Version                          Rev    Tracking      ␣
↪Publisher           Notes
avahi                     0.8                              327    22/stable      ondra   ␣
↪           -
bluez                     5.64-4                           356    22/stable     ␣
↪canonical✓          -
core22                    20231123                         1033   latest/stable ␣
↪canonical✓          base
openthread-border-router  thread-reference-20230119+snap   37     latest/edge  canonical-
↪iot-labs  -
otbr-gadget               test                             x1     -              -       ␣
↪           gadget
pc-kernel                 5.15.0-91.101.1                  1540   22/stable     ␣
↪canonical✓          kernel
snapd                     2.60.4                           20290  latest/stable ␣
↪canonical✓          snapd
```

Avahi, BlueZ, and openthread-border-router are installed.

Check the running snap services:

```
<user>@ubuntu:~$ snap services
Service                             Startup  Current   Notes
avahi.daemon                        enabled  active    -
bluez.bluez                         enabled  active    -
openthread-border-router.otbr-agent  enabled  active    -
openthread-border-router.otbr-setup  enabled  inactive  -
openthread-border-router.otbr-web    enabled  active    -
```

Avahi and BlueZ's services are enabled and active.
The OTBR agent and web server are enabled and active.
The OTBR setup oneshot service is enabled, but inactive. It is enabled because it needs to run on every boot to setup the firewall and network. It is inactive because it has completed its work and exited.

Check the snap connections:

```
<user>@ubuntu:~$ snap connections openthread-border-router
Interface          Plug                                        Slot                    ␣
↪           Notes
avahi-control      openthread-border-router:avahi-control      avahi:avahi-control     ␣
↪           gadget
bluetooth-control  openthread-border-router:bluetooth-control  :bluetooth-control      ␣
↪           gadget
bluez              openthread-border-router:bluez              bluez:service           ␣
↪           gadget
dbus               -                                           openthread-border-
↪router:dbus-wpan0  -
firewall-control   openthread-border-router:firewall-control   :firewall-control       ␣
```

(continues on next page)

(continued from previous page)

```
↪           gadget
network             openthread-border-router:network            :network                ␣
↪           -
network-bind        openthread-border-router:network-bind       :network-bind           ␣
↪           -
network-control     openthread-border-router:network-control    :network-control        ␣
↪           gadget
raw-usb             openthread-border-router:raw-usb            :raw-usb                ␣
↪           gadget
```

The connections with `gadget` in the Note match those defined as `connections` in our gadget.

Finally, check the snap configurations:

```
<user>@ubuntu:~$ snap get openthread-border-router
Key         Value
autostart   true
infra-if    enp88s0
radio-url   spinel+hdlc+uart:///dev/ttyACM0
thread-if   wpan0
```

The values are according to the `defaults` set in our gadget.

You may further continue by checking the logs, for example with `snap logs -n 100 -f openthread-border-router`.

1.2.5 How to install Home Assistant on Ubuntu Core

This guide walks you through installing Home Assistant (HASS) on Ubuntu Core. Home Assistant is an open source home automation solution, designed with a rich ecosystem of integrations for connecting smart devices. We use Ubuntu Core as the OS, to guarantee a secure and up to date foundation for what runs at the center of your smart home.

Note

The instructions should work on the following architectures:

- ARM64 / AArch64
- AMD64 / x86_64

The guide has been tested on Raspberry Pi 4.

Install Ubuntu Core

Refer to the official documentation for installing Ubuntu Core.

Set up the system

SSH to the machine. If you installed a pre-built Ubuntu Core image, it comes with Console Conf which has guided you to deploy the public keys from your Ubuntu SSO account. In this case, you should use your Ubuntu username to connect: `ssh <user>@<ip>`.

Take a look at what is installed:

```
$ snap list
Name       Version         Rev    Tracking       Publisher   Notes
core22     20230703        821    latest/stable  canonical✓  base
pi         22-2            132    22/stable      canonical✓  gadget
pi-kernel  5.15.0-1048.51  778    22/stable      canonical✓  kernel
snapd      2.61.2          21185  latest/stable  canonical✓  snapd
```

As you see, everything on an Ubuntu Core system is a snap, including the kernel. At least this is how we start. Later on, we'll also add a Docker container, via a snapped Docker Engine.

Let's prepare the machine for the upcoming work.

Change the default hostname (ubuntu):

```
$ sudo hostnamectl set-hostname pi4
```

Install the Avahi snap. The Avahi daemon is needed for local mDNS broadcasts and mDNS discovery:

```
$ sudo snap install avahi
avahi 0.8 from Ondrej Kubik (ondra) installed
```

Reboot (`sudo reboot`) to make the hostname change effective.

Now, you should now be able to SSH to the machine via it's local domain: `ssh <user>@pi4.local`

Install Home Assistant

We will use the Home Assistant snap to deploy it.

Install the latest stable version:

```
$ sudo snap install home-assistant-snap
home-assistant-snap (2023.12/stable) 2023.12.4 from Giaever.online (giaever-online)␣
↪installed
```

We can follow with: `snap logs -f -n 5 home-assistant-snap`

Verify what resources this snap has access to:

```
$ snap connections home-assistant-snap
Interface               Plug                                  Slot            ␣
↪                Notes
bluez                   home-assistant-snap:bluez             -               ␣
↪                -
content                 -                                     home-assistant-
↪snap:components      -
content                 -                                     home-assistant-
↪snap:configurations  -
```

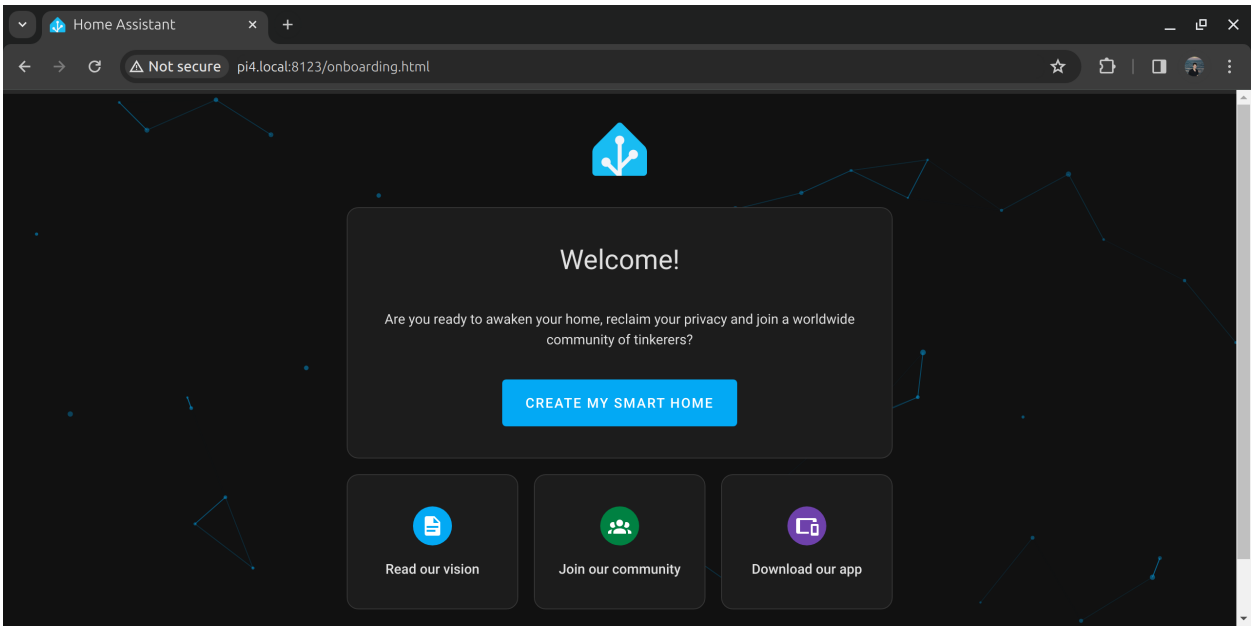
(continues on next page)

(continued from previous page)

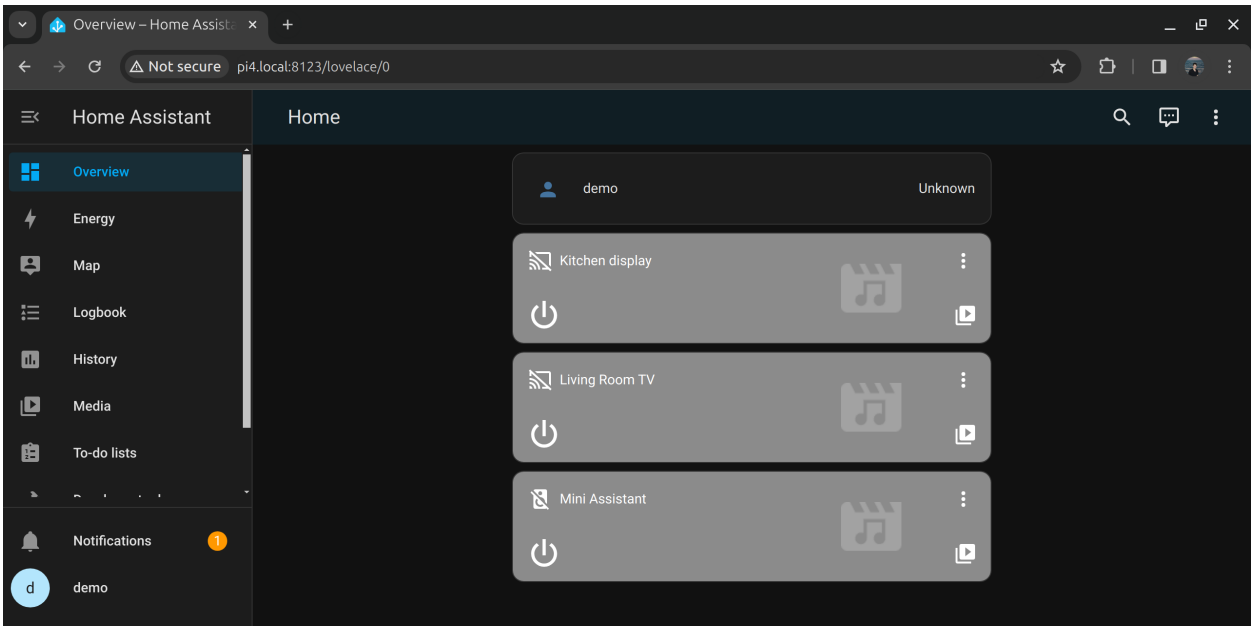
```
content                    -                                        home-assistant-
↪snap:vscs-content     -
content                    home-assistant-snap:bin                  -                 ␣
↪                  -
desktop                    home-assistant-snap:desktop              -                 ␣
↪                  -
hardware-observe           home-assistant-snap:hardware-observe     :hardware-observe ␣
↪                  -
network                    home-assistant-snap:network              :network          ␣
↪                  -
network-bind               home-assistant-snap:network-bind         :network-bind     ␣
↪                  -
network-control            home-assistant-snap:network-control      :network-control  ␣
↪                  -
physical-memory-control    home-assistant-snap:physical-memory-control  -             ␣
↪                  -
raw-usb                    home-assistant-snap:raw-usb              -                 ␣
↪                  -
removable-media            home-assistant-snap:removable-media      -                 ␣
↪                  -
serial-port                home-assistant-snap:serial-port          -                 ␣
↪                  -
```

The essential networking interfaces have been connected, which are sufficient for us. It is possible to remove extra access, or add additional ones.

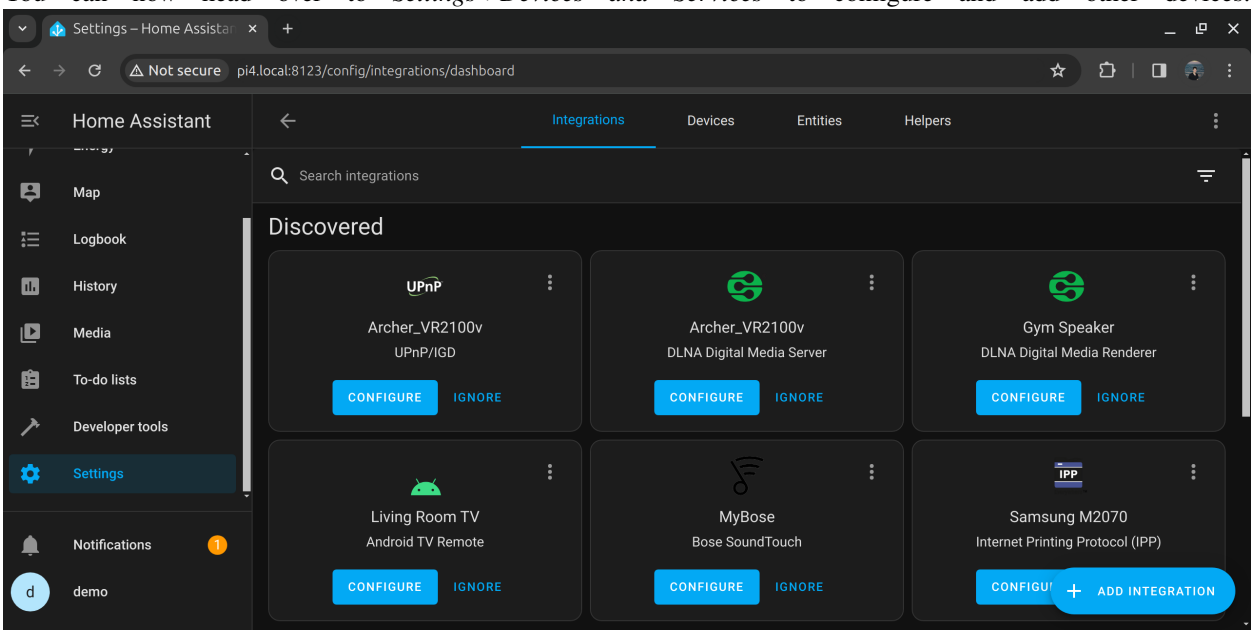
Now open the following address via a web browser to start onboarding Home Assistant: http://pi4.local:8123



Follow the wizard to set up your instance. In the end, you will be redirected to the default dashboard with some possible auto-configured devices. In my case, there are a few Chromecast devices:



You can now head over to *Settings->Devices and Services* to configure and add other devices:



That's really it. You now have a fully functional Home Assistant instance, which stays up to date and secure.

Home Assistant comes with numerous *Integrations* out of the box, enabling you to add your smart home with little efforts. In the next section, we'll walk you through adding Matter integration.

Add Matter Integration

> **Important**
>
> This guides uses a beta version of Python Matter Server from Home Assistant Libs, which is not ready for production.

In order to add Matter integration to Home Assistant, we need to use the Python Matter Server. This component is not available as a snap, so we will deploy it as a Docker container.

Install Docker snap:

```
$ sudo snap install docker
docker 24.0.5 from Canonical✓ installed
```

Run the Docker container for Python Matter Server:

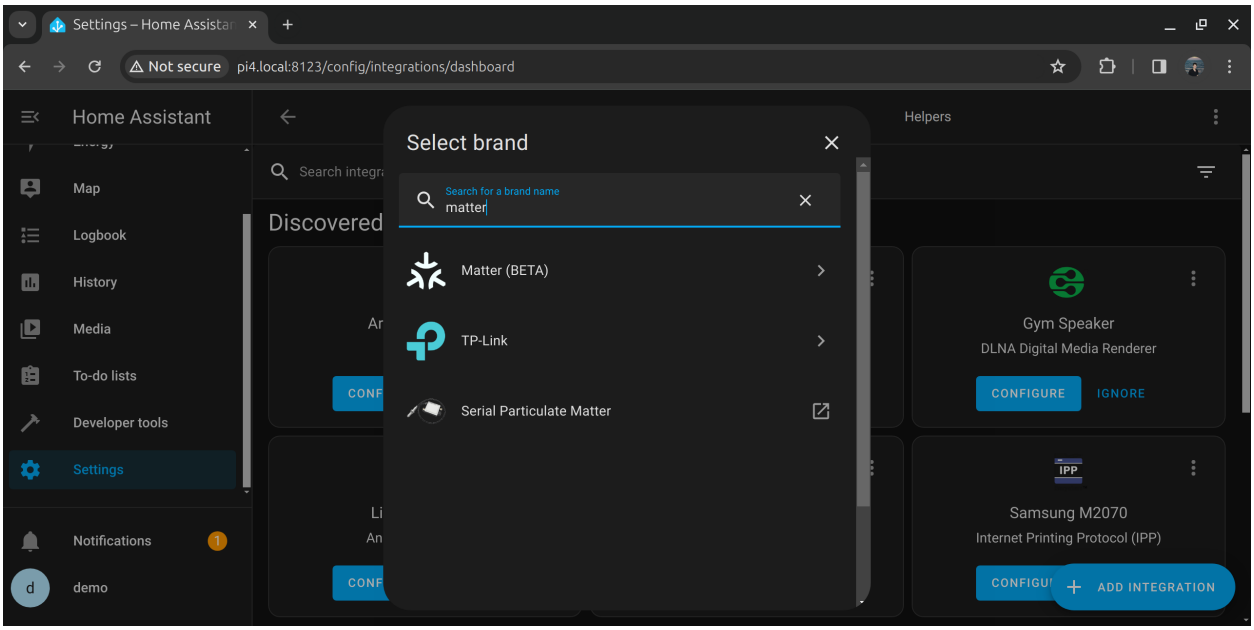
```
$ sudo docker run -d \
  --name matter-server \
  --restart=unless-stopped \
  --security-opt apparmor=unconfined \
  -v $(pwd)/matter-server:/data \
  -v /run/dbus:/run/dbus:ro \
  --network=host \
  ghcr.io/home-assistant-libs/python-matter-server:stable

Unable to find image 'ghcr.io/home-assistant-libs/python-matter-server:stable' locally
stable: Pulling from home-assistant-libs/python-matter-server
abd2c048cba4: Pull complete
861eb9f546f8: Pull complete
f7bb0ec509a9: Pull complete
3ec31f44b517: Pull complete
c4b248828bce: Pull complete
3738fbd089b3: Pull complete
252ff7c1d11a: Pull complete
675008dad2ae: Pull complete
Digest: sha256:aab82f903670b7bf4f72eb24c7d5b3520c854fe272f196e32b354c63f02d8724
Status: Downloaded newer image for ghcr.io/home-assistant-libs/python-matter-
↪server:stable
5753ab4ecbc6f181be2669d4281cd27e0cb4d591d1faa4fa640759ff7547a38a
```

The above command pulled the image (because it didn't exist locally) and then started it in the background.

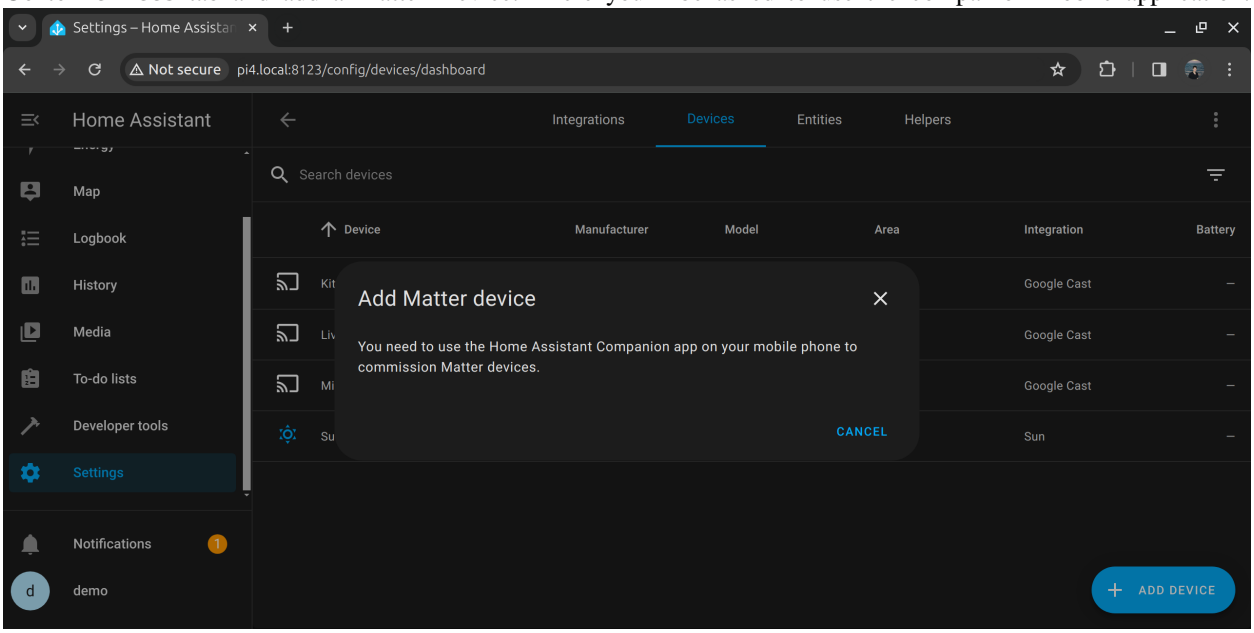
We can follow with: `sudo docker logs -f -n 5 matter-server`

Now, head to the Home Assistant *Settings->Devices & Services* and add the Matter integration:



Leave the server URL as default: `http://localhost:5580/ws`, because we run the server on the same machine as the Home Assistant server.

Go to `Devices` tab and add a Matter Device. Here you'll be asked to use the companion mobile application:



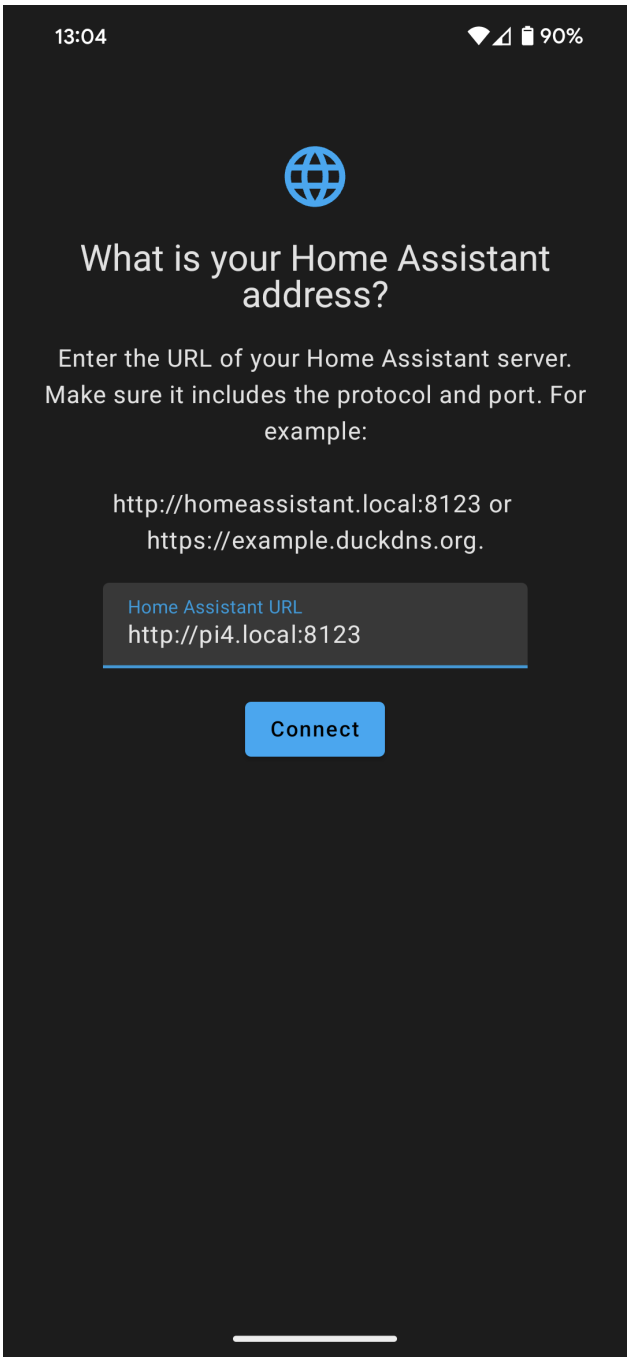
Install the Home Assistant Companion mobile app for iOS or Android. The application is documented here. In the following steps, we'll use the Android application.

The application will usually discover the running Home Assistant instance. But we advise that you configure it manually to use the local domain name. Alternatively, you could set up an IP address on the device and use that instead.

Once you've completed configuring the application, you'll land on the Home Assistant dashboard. Go to *Settings->Devices and services->Devices* and add your Matter device.

We'll use a Matter-compliant Smart Plug, from an unknown manufacturer.

Tip: One of the benefits of Matter standard is that we don't need to worry much about who made the device, because


13:04
90%
What is your Home Assistant address?
Enter the URL of your Home Assistant server. Make sure it includes the protocol and port. For example:
http://homeassistant.local:8123 or https://example.duckdns.org.
Home Assistant URL
http://pi4.local:8123
Connect

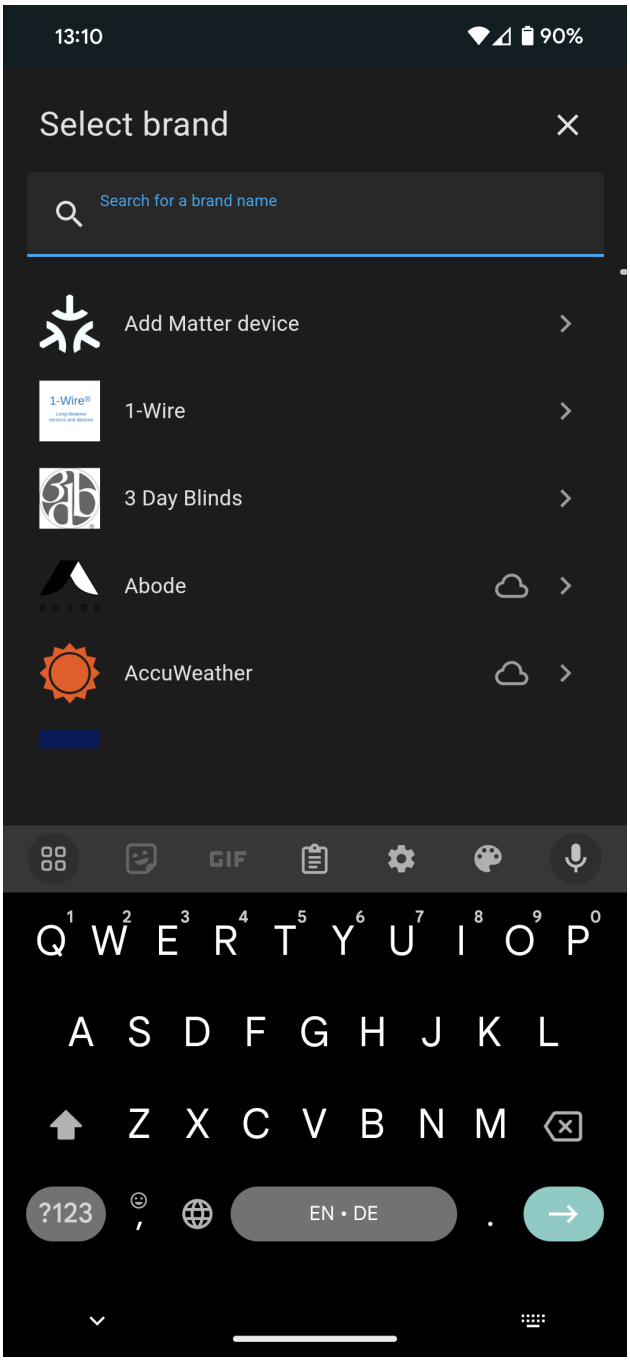
we should be able to use it as long as it is compliant with the standard. It is still essential to ensure the device is secured, possibly by sandboxing it inside the local network (block internet access).

Let's add the device:

You need to scan it's QR code:

This will drive the commissioning, through the following steps:

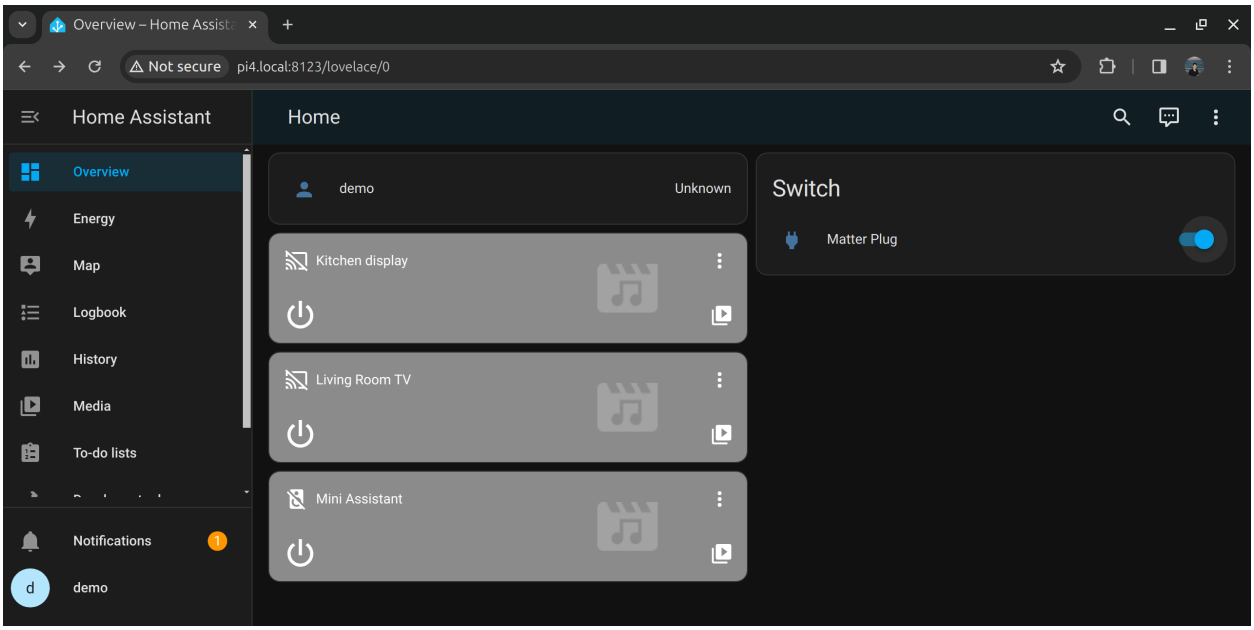
- Connecting to device …
- Checking network connectivity …
- Generating Matter Credentials …


13:10
90%
Select brand
Search for a brand name
Add Matter device
1-Wire
3 Day Blinds
Abode
AccuWeather
GIF
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
?123
EN • DE


13:10
90%
Scan Matter QR code
Make sure it's steady and well-lit
Set up without QR code
Powered by Google

- Connecting device to Home Assistant …
- Device connected

Now, you should be able to control this device via the smart phone app or the web browser:

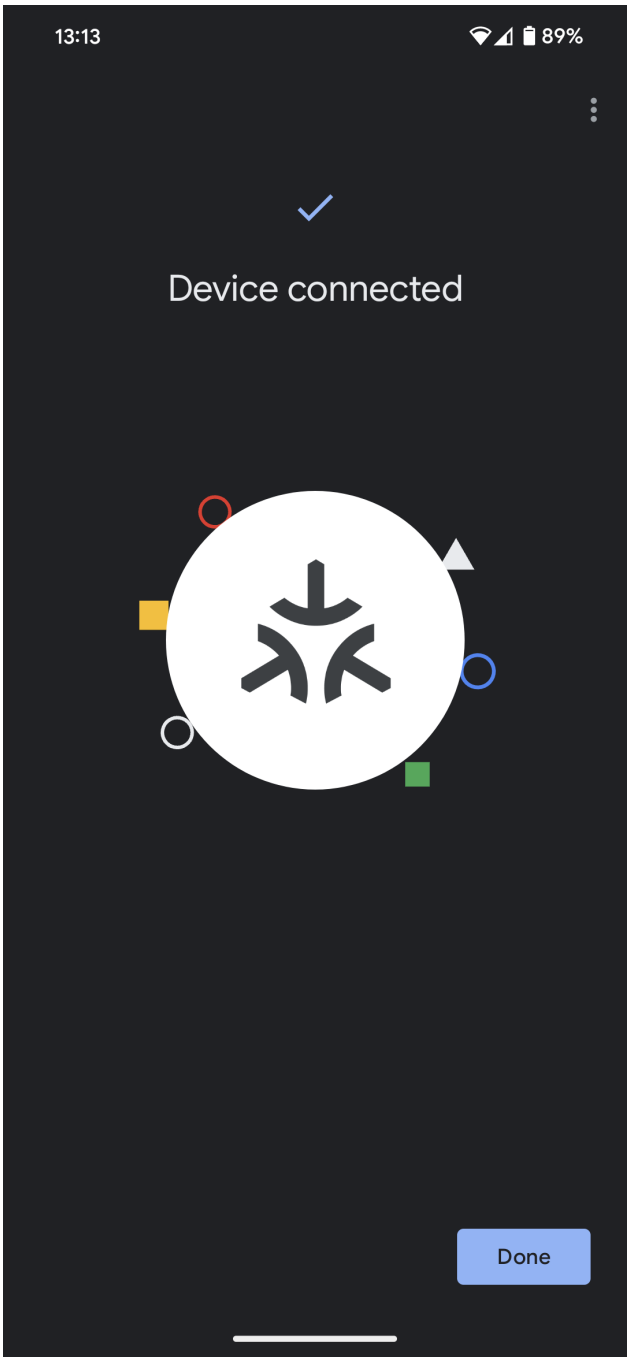


Take it to the next level

The Home Assistant instance can further configured and extended with community driver integrations.

You may refer to the following snaps from the same publisher:

- Home Assistant Community Store - to manage custom integrations and plugins
- Home Assistant Toolbox - to add tools such as cURL
- Home Assistant Configurator - to configure Home Assistant remotely via a web-based text editor based on Ace
- Remote configuration via VSCode Server - to run a VSCode server and allow remote management via VSCode code editor.


13:13
89%
Device connected
Done

CHAPTER

TWO

PROJECT AND COMMUNITY

Matter on Ubuntu references existing open source implementations. The Matter standard and SDK are supported by the Connectivity Standards Alliance (CSA).

Canonical is responsible for Snap packages presented in this documentation:

- Chip Tool
- OpenThread Border Router
- Matter Pi GPIO Commander

Other resources:

- Matter Standard
- Matter SDK