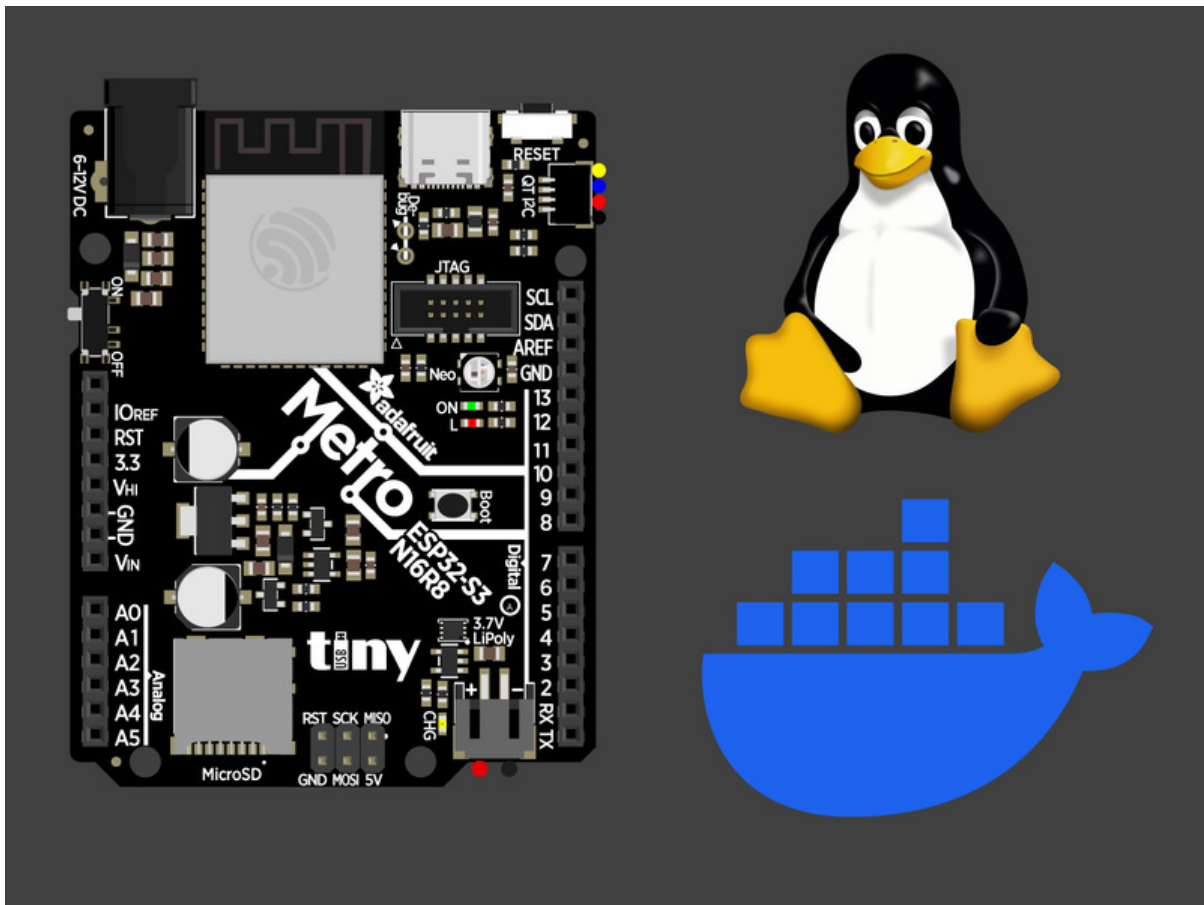




Use Docker to Compile Linux for ESP32-S3

Created by Liz Clark



<https://learn.adafruit.com/docker-esp32-s3-linux>

Last updated on 2024-06-03 03:53:58 PM EDT

Table of Contents

Overview	3
• Parts	
Install Docker	5
Docker Test Image	6
Docker ESP32-S3 Linux Image	9
Upload the Files to the ESP32-S3	13
• esptool	
• ESP-IDF	
• Prep the Metro ESP32-S3	
• Binary Files	
• File System and Kernel	
Boot Linux on the ESP32-S3	16
• WiFi	
• Customize	
Customize with menuconfig	18
• Boot into Linux	
Customize with ESP-IDF menuconfig	22
• Partition Table	
• ESP-IDF menuconfig	

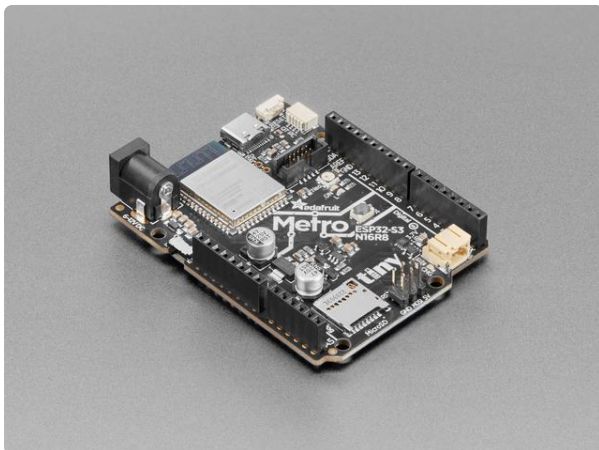
Overview

Have you ever wanted to run Linux on a microcontroller like the ESP32-S3? GitHub user [jcmvbkbc](https://adafru.it/18Uf) (<https://adafru.it/18Uf>) has a [Linux for ESP32-S3 script](https://adafru.it/18UB) (<https://adafru.it/18UB>) that lets you do just that! However, compiling it can be tricky depending on your operating system with all of the toolchain dependencies. Don't worry though because there is a way around all of that: [a Docker container](https://adafru.it/18UC) (<https://adafru.it/18UC>)! A container is a lightweight and fast virtual computer. The best part is it's easy for people to run no matter what operating system they are on. It's also much simpler and smaller than VirtualBox because it has only the bare essentials.

In this guide, you'll setup and install Docker, run a test Dockerfile to create a test container, run the ESP32-S3 Linux Dockerfile, upload the compiled files to the ESP32-S3 and then perform the ultimate skateboard trick: boot Linux on an ESP32-S3!

This version of Linux is barebones, the file system is utilizing [cramfs](https://adafru.it/18UD) (<https://adafru.it/18UD>), so don't expect it to be like booting up a Raspberry Pi with Raspberry Pi OS. You'll access it over serial and you'll only be able to do super basic things but you'll learn a ton along the way.

Parts



[Adafruit Metro ESP32-S3 with 16 MB Flash 8 MB PSRAM](https://www.adafruit.com/product/5500)

What's Metro-shaped and has an ESP32-S3 WiFi module? What has a STEMMA QT connector for I2C devices and a Lipoly charger circuit? What has your favorite Espressif WiFi...

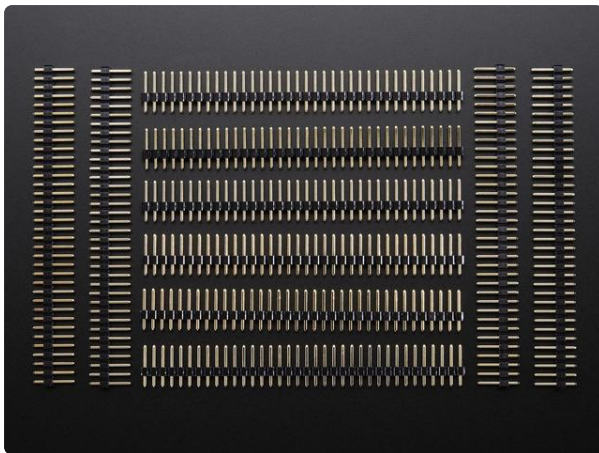
<https://www.adafruit.com/product/5500>



USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi

The cable is easiest way ever to connect to your microcontroller/Raspberry Pi/WiFi router serial console port. Inside the big USB plug is a USB<->Serial conversion chip and at...

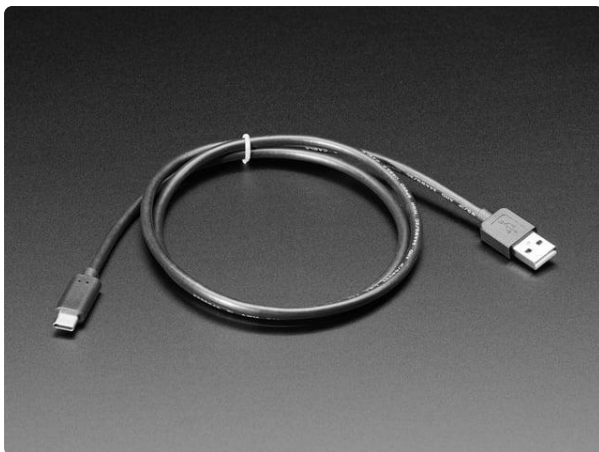
<https://www.adafruit.com/product/954>



Break-away 0.1" 36-pin strip male header - Black - 10 pack

Breakaway header is like the duct tape of electronics. It's great for connecting things together, soldering to perf-boards, fits into any breakout or breadboard, etc. We go through...

<https://www.adafruit.com/product/392>

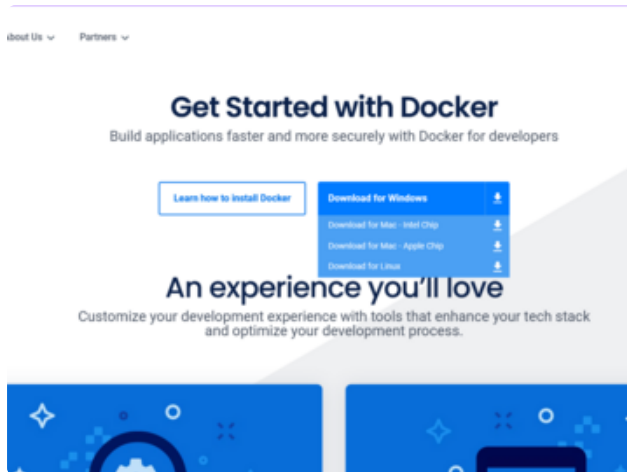


USB Type A to Type C Cable - approx 1 meter / 3 ft long

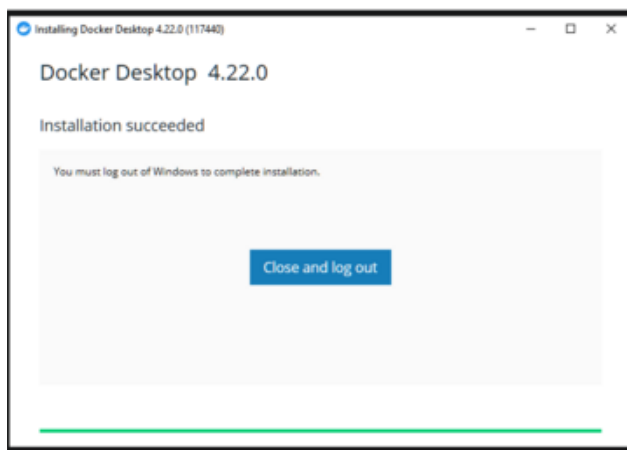
As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

<https://www.adafruit.com/product/4474>

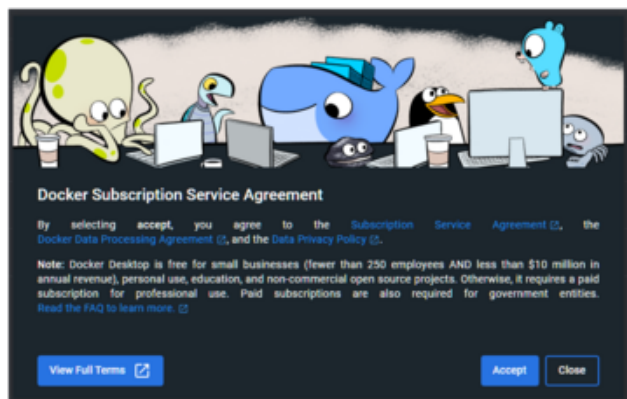
Install Docker



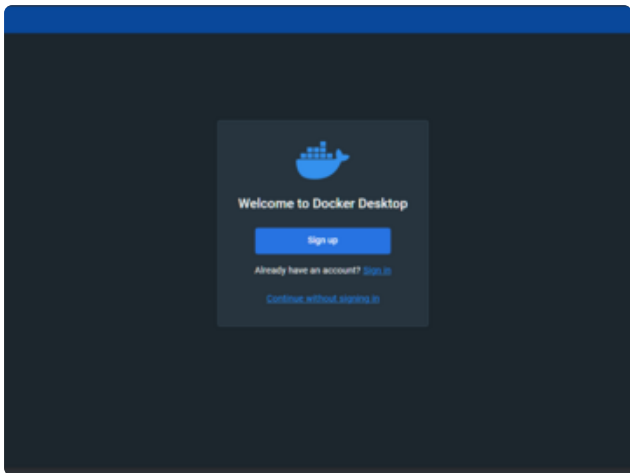
Go to the [Docker website \(https://adafru.it/18UE\)](https://adafru.it/18UE) and download the Docker version for your operating system. Then, launch the installer file that downloads.



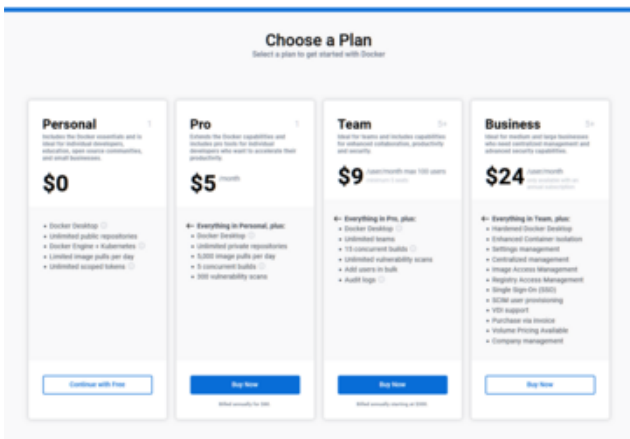
After installation finishes, you may be prompted to logout of your system to finish setting things up.



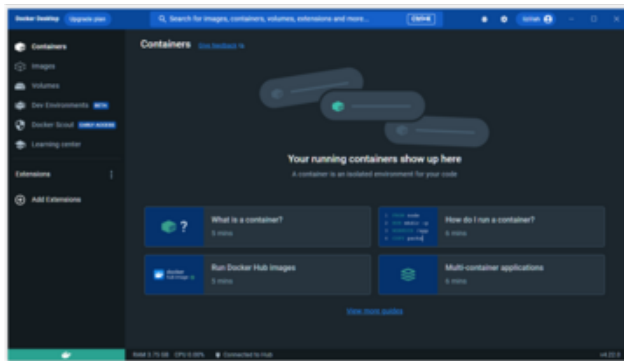
When Docker launches for the first time, you'll be prompted with a service agreement. Click **Accept**.



Then you'll be prompted to sign up for a Docker account. Click the **Sign Up** button to create an account.



After creating an account on the Docker website, you'll be prompted to login for the first time. After logging in, you'll need to select a plan. There is a free plan for Docker (**Personal**) and everything covered in this guide works with the free tier.

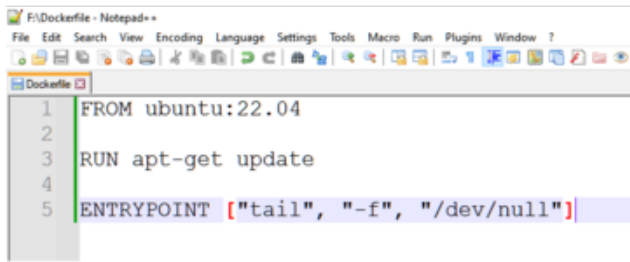


Then you can log into the Docker desktop application, where you can access your containers and images.

Next, you'll create a test Docker image to get used to how Docker works and check your workflow.

Docker Test Image

Before you go diving into compiling the Linux kernel, you can run this quick test Docker image to get familiar with Docker.



```
1 FROM ubuntu:22.04
2
3 RUN apt-get update
4
5 ENTRYPOINT ["tail", "-f", "/dev/null"]
```

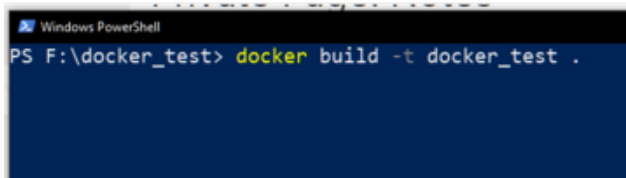
First, you're going to create a Dockerfile file. These files are run to create the Docker images and they use their own syntax. Copy and paste the following text into your preferred text editor:

```
FROM ubuntu:22.04

RUN apt-get update

ENTRYPOINT ["tail", "-f", "/dev/null"]
```

Then, save the file as **Dockerfile** with no file extension. The Dockerfile you just created builds an Ubuntu 22.04 system. Then, it runs `apt-get update` and opens a terminal window for you to interact with the system. If you were to leave out the `ENTRYPOINT` line, the image would simply run `apt-get update` and then close down.



```
PS F:\docker_test> docker build -t docker_test .
```

Next, open up a terminal and navigate to the folder where you just saved your Dockerfile. Run the following command to build the image:

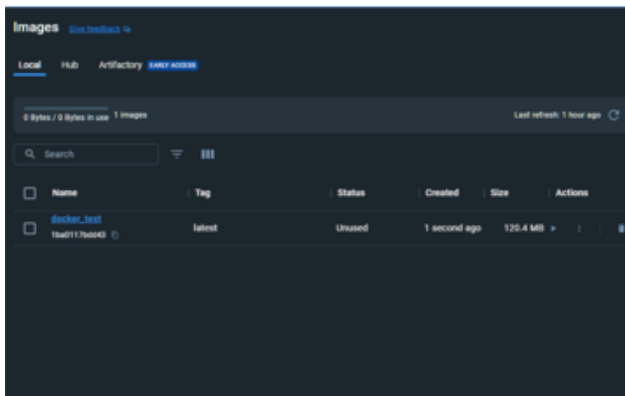
```
docker build -t docker_test .
```

This tells docker to build the Dockerfile as an image and tag it (name it) as **docker_test**.

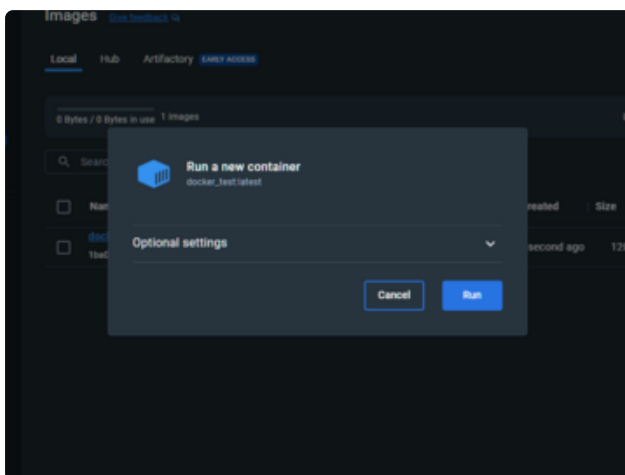
```
PS F:\docker_test> docker build -t docker_test .
[+] Building 12.8s (7/7) FINISHED          docker:default
=> [internal] load .dockerignore          0.1s
=> [internal] transferring context: 28   0.0s
=> [internal] load build definition from Dockerfile 0.1s
=> [internal] transferring dockerfile: 134B 0.0s
=> [internal] load metadata for docker.io/library/ubuntu:22.04 1.6s
=> [auth] library/ubuntu:pull token for registry-1.docker.io 0.0s
=> [1/2] FROM docker.io/library/ubuntu:22.04@sha256:80ced47ffffa3561ef 1.8s
=> resolve docker.io/library/ubuntu:22.04@sha256:80ced47ffffa3561ef 0.0s
=> sha256:80ced47ffffa3561efa981854fcabc04577cd43ce 1.13kB / 1.13kB 0.0s
=> sha256:5008ffff8a1561c3e6deadb487b990100fc26830b 424B / 424B 0.0s
=> sha256:5a81c4b582e4979e75bd8f91343b950bd095abb 2.30kB / 2.30kB 0.0s
=> sha256:3153aa3880826a2235e1ed0169e330e451f 29.53MB / 29.53MB 0.7s
=> extracting sha256:3153aa3880826a2235e1ed0169e330e451f01a8a31 0.8s
=> [2/2] RUN apt-get update 3.0s
=> exporting to image 0.2s
=> exporting layers 0.2s
=> writing image sha256:1ba0117bd0431bab14cbb46211284295194693d242 0.0s
=> naming to docker.io/library/docker_test 0.0s

What's Next?
View summary of image vulnerabilities and recommendations → docker scout quickview
PS F:\docker_test>
```

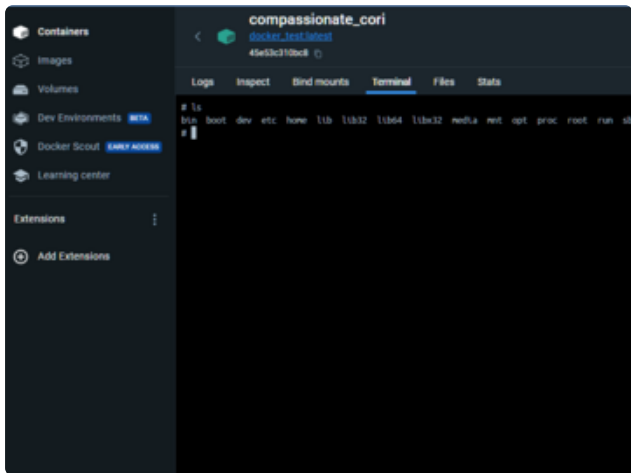
After running the command, you'll see Docker go through a series of build steps and then finish.



Go to your Docker desktop application and click on the **Images** tab. You should see your **docker_test** image that you just built.

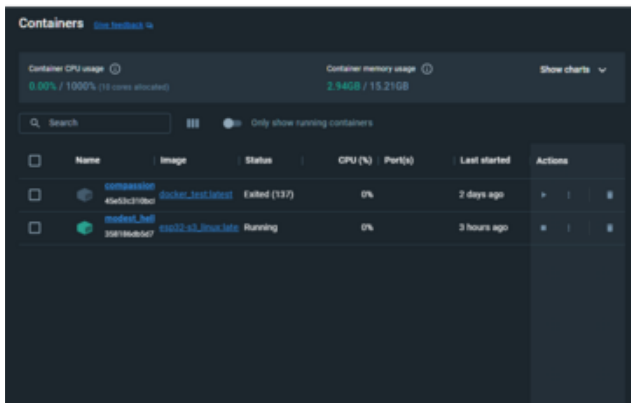


The image is static, similar to creating a USB drive with installation media if you've ever done a fresh install of Windows or Linux on hardware. To deploy the image that you just built, click on the **Play** button next to the image name. A dialog box will open and you'll click **Run**.



This launches your image into a container. The container is where your virtual computer lives and runs. It is given the name of an adjective along with the name of a scientist as a nickname by default.

If you click on the **Terminal** tab in the container, you can interact with your new Ubuntu image.



On the main Containers page, you can start or stop your container from running by clicking the **Play** or **Stop** buttons on the right-hand side of the window.

Now that you've created your first Dockerfile, image and container, you can try using Docker to compile Linux for the ESP32-S3.

Docker ESP32-S3 Linux Image

To begin, you'll save the Dockerfile below to a folder on your local filesystem. Name it as **Dockerfile** with no file extension, just like you did on the [Test Image page \(https://adafruit.it/18UF\)](https://adafruit.it/18UF).

```
# Dockerfile port of https://gist.github.com/jcmvbkbc/
316e6da728021c8ff670a24e674a35e6
# wifi details http://wiki.osll.ru/doku.php/etc:users:jcmvbkbc:linux-
xtensa:esp32s3wifi

# we need python 3.10 not 3.11
FROM ubuntu:22.04

RUN apt-get update
RUN apt-get -y install gperf bison flex texinfo help2man gawk libtool-bin git unzip
ncurses-dev rsync zlib1g zlib1g-dev xz-utils cmake wget bzip2 g++ python3 python3-
dev python3-pip cpio bc virtualenv libusb-1.0 && \
    ln -s /usr/bin/python3 /usr/bin/python

WORKDIR /app
```

```

# install autoconf 2.71
RUN wget https://ftp.gnu.org/gnu/autoconf/autoconf-2.71.tar.xz && \
  tar -xf autoconf-2.71.tar.xz && \
  cd autoconf-2.71 && \
  ./configure --prefix=`pwd`/root && \
  make && \
  make install
ENV PATH="$PATH:/app/autoconf-2.71/root/bin"

# dynconfig
RUN git clone https://github.com/jcmvbkbc/xtensa-dynconfig -b original --depth=1 && \
  git clone https://github.com/jcmvbkbc/config-esp32s3 esp32s3 --depth=1 && \
  make -C xtensa-dynconfig ORIG=1 CONF_DIR=`pwd` esp32s3.so
ENV XTENSA_GNU_CONFIG="/app/xtensa-dynconfig/esp32s3.so"

# ct-ng cannot run as root, we'll just do everything else as a user
RUN useradd -d /app/build -u 3232 esp32 && mkdir build && chown esp32:esp32 build
USER esp32

# toolchain
RUN cd build && \
  git clone https://github.com/jcmvbkbc/crosstool-NG.git -b xtensa-fdpic --depth=1
&& \
  cd crosstool-NG && \
  ./bootstrap && \
  ./configure --enable-local && \
  make && \
  ./ct-ng xtensa-esp32s3-linux-uclibcfdpic && \
  CT_PREFIX=`pwd`/builds ./ct-ng build || echo "Completed" # the complete ct-ng
build fails but we still get what we wanted!
RUN [ -e build/crosstool-NG/builds/xtensa-esp32s3-linux-uclibcfdpic/bin/xtensa-
esp32s3-linux-uclibcfdpic-gcc ] || exit 1

# kernel and rootfs
RUN cd build && \
  git clone https://github.com/jcmvbkbc/buildroot -b xtensa-2023.02-fdpic --depth=1
&& \
  make -C buildroot 0=`pwd`/build-xtensa-2023.02-fdpic-esp32s3
esp32s3wifi_defconfig && \
  buildroot/utils/config --file build-xtensa-2023.02-fdpic-esp32s3/.config --set-
str TOOLCHAIN_EXTERNAL_PATH `pwd`/crosstool-NG/builds/xtensa-esp32s3-linux-
uclibcfdpic && \
  buildroot/utils/config --file build-xtensa-2023.02-fdpic-esp32s3/.config --set-
str TOOLCHAIN_EXTERNAL_PREFIX '$(ARCH)-esp32s3-linux-uclibcfdpic' && \
  buildroot/utils/config --file build-xtensa-2023.02-fdpic-esp32s3/.config --set-
str TOOLCHAIN_EXTERNAL_CUSTOM_PREFIX '$(ARCH)-esp32s3-linux-uclibcfdpic' && \
  make -C buildroot 0=`pwd`/build-xtensa-2023.02-fdpic-esp32s3
RUN [ -f build/build-xtensa-2023.02-fdpic-esp32s3/images/xipImage -a -f build/build-
xtensa-2023.02-fdpic-esp32s3/images/rootfs.cramfs ] || exit 1

# bootloader
ENV IDF_PATH="/app/build/esp-hosted/esp_hosted_ng/esp/esp_driver/esp-idf"
RUN cd build && \
  git clone https://github.com/jcmvbkbc/esp-hosted -b shmem --depth=1 && \
  cd esp-hosted/esp_hosted_ng/esp/esp_driver && cmake . && \
  cd esp-idf && . ./export.sh && \
  cd ../network_adapter && idf.py set-target esp32s3 && \
  cp sdkconfig.defaults.esp32s3 sdkconfig && idf.py build

# move files over
RUN cd build && mkdir release && \
  cp esp-hosted/esp_hosted_ng/esp/esp_driver/network_adapter/build/bootloader/
bootloader.bin release && \
  cp esp-hosted/esp_hosted_ng/esp/esp_driver/network_adapter/build/
partition_table/partition-table.bin release && \
  cp esp-hosted/esp_hosted_ng/esp/esp_driver/network_adapter/build/

```

```

network_adapter.bin release && \
cp build-xtensa-2023.02-fdpic-esp32s3/images/xipImage release && \
cp build-xtensa-2023.02-fdpic-esp32s3/images/rootfs.cramfs release

# keep docker running so we can debug/rebuild :)
USER root
ENTRYPOINT ["tail", "-f", "/dev/null"]

# grab the files with `docker cp CONTAINER_NAME:/app/build/release/\* .`
# now you can burn the files from the 'release' folder with:
# python esptool.py --chip esp32s3 -p /dev/ttyUSB0 -b 921600 --before=default_reset
--after=hard_reset write_flash 0x0 bootloader.bin 0x10000 network_adapter.bin
0x8000 partition-table.bin
# next we can burn in the kernel and filesystems with parttool, which is part of esp-idf
# parttool.py write_partition --partition-name linux --input xipImage
# parttool.py write_partition --partition-name rootfs --input rootfs.cramfs

```

```

1 # Dockerfile port of https://gist.github.com/icvbkbc/316e6da728021c8ff670a2
2 # wifi details http://wiki.osll.ru/doku.php/etc:users:icvbkbc:linux-xtensa:
3
4 # we need python 3.10 not 3.11
5 FROM ubuntu:22.04
6
7 RUN apt-get update
8 RUN apt-get -y install gperf bison flex texinfo help2man gawk libtool-bin g
9 ln -s /usr/bin/python3 /usr/bin/python
10
11 WORKDIR /app
12
13 # install autoconf 2.71
14 RUN wget https://ftp.gnu.org/gnu/autoconf/autoconf-2.71.tar.xz && \
15 tar -xf autoconf-2.71.tar.xz && \
16 cd autoconf-2.71 && \
17 ./configure --prefix='pwd'/root && \
18 make && \
19 make install
20 ENV PATH="$PATH:/app/autoconf-2.71/root/bin"
21
22 # dynconfig
23 RUN git clone https://github.com/icvbkbc/xtensa-dynconfig -b original --dep
24 git clone https://github.com/icvbkbc/config-esp32s3 esp32s3 --depth=1 &
25 make -C xtensa-dynconfig ORIG=1 CONF_DIR='pwd' esp32s3.so
26 ENV XTENSA_GNU_CONFIG="/app/xtensa-dynconfig/esp32s3.so"

```

As you can see, this Dockerfile is a lot more detailed than the test image that you just ran. It installs all of the toolchains and dependencies on an Ubuntu build and then runs the script to compile the files needed to load onto the ESP32-S3.

```

$ F:\esp32-s3_linux> docker build -t esp32-s3_linux .
[+] Building 1614.4s (18/18) FINISHED
[+] [internal] load build definition from Dockerfile 0.1s
[+] [internal] transferring Dockerfile: 4.12s 0.0s
[+] [internal] load dockerignore 0.1s
[+] [internal] transferring context: 1s 0.0s
[+] [internal] load metadata for docker.io/library/ubuntu:22.04 0.4s
[+] [authn] library/ubuntu:pull token for registry-1.docker.io 0.0s
[+] [pull] FROM docker.io/library/ubuntu:22.04@sha256:0b94d7fffa3a1e6021818f6a
[+] [internal] 2/13) RUN apt-get update 0.0s
[+] [3/13] RUN apt-get -y install gperf bison flex texinfo help2man gawk libtool 187.5s
[+] [4/13] WORKDIR /app 0.1s
[+] [5/13] RUN wget https://ftp.gnu.org/gnu/autoconf/autoconf-2.71.tar.xz && \ 1.7s
[+] [6/13] RUN git clone https://github.com/icvbkbc/xtensa-dynconfig -b original 2.4s
[+] [7/13] RUN mkdir -p /app/build && cd /app/build && git clone https://github.com/icvbkbc/esp32s3 0.7s
[+] [8/13] RUN cd /app/build && git clone https://github.com/icvbkbc/cross-toolchain 748.4s
[+] [9/13] RUN cd /app/build && git clone https://github.com/icvbkbc/build-xtensa 0.5s
[+] [10/13] RUN cd /app/build && git clone https://github.com/icvbkbc/build-xtensa 541.8s
[+] [11/13] RUN cd /app/build && git clone https://github.com/icvbkbc/build-xtensa 541.8s
[+] [12/13] RUN cd /app/build && git clone https://github.com/icvbkbc/build-xtensa 541.8s
[+] [13/13] RUN cd /app/build && git clone https://github.com/icvbkbc/build-xtensa 541.8s
[+] [exporting to image] 0.0s
[+] [exporting layers] 0.0s
[+] [writing image sha256:187f6b0a07481b0e78430a73737977021c132212007f6d 0.0s
[+] [pushing to docker.io/library/esp32-s3_linux 0.0s

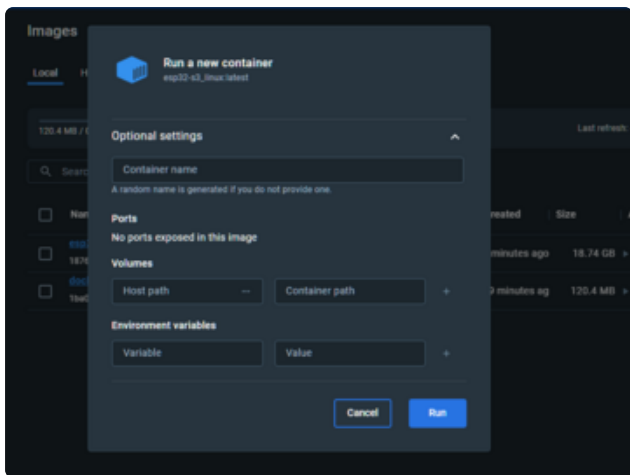
What's Next?
View summary of image vulnerabilities and recommendations + docker scout quickfixes
$ F:\esp32-s3_linux>

```

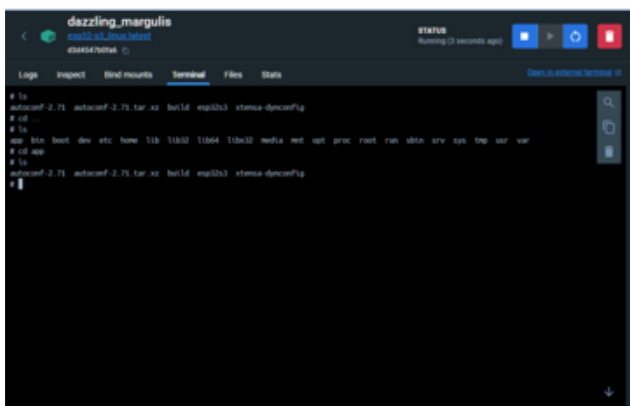
The steps for building are the same though. Open a terminal window, navigate to the folder where you saved the Dockerfile and enter:

```
docker build -t esp32-s3_linux .
```

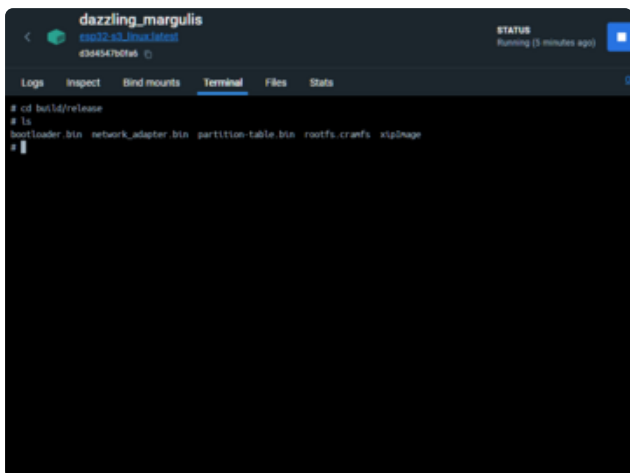
This builds the image and names it esp32-s3_linux.



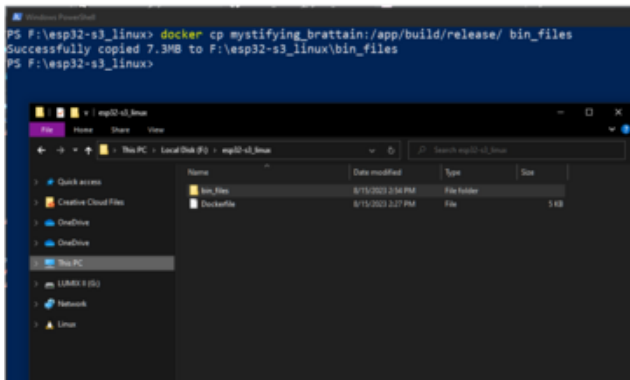
After building, go to the Docker desktop app and look under **Images**. Select your newly created **esp32-s3_linux** image and click the **Play** button. Then click **Run** to start a container with the image.



Once the container opens, go to the **Terminal** tab. You'll be able to interact with the Ubuntu machine that has your compiled files for the ESP32-S3.



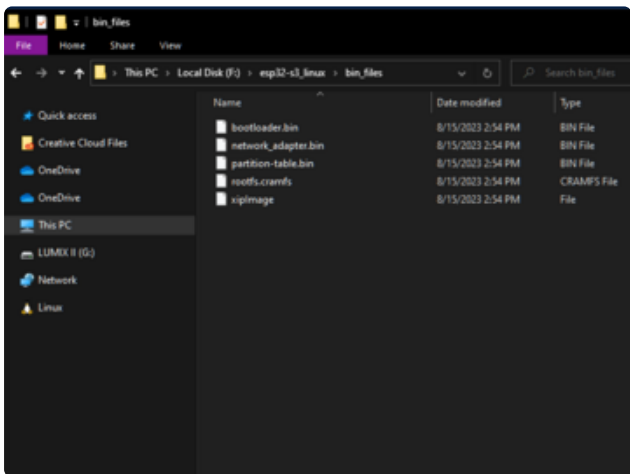
If you change directories to the releases folder (`cd /app/build/releases`) you'll see the kernel, file system and binary files that will be loaded onto your ESP32-S3. To use these files, you'll need to copy them to your local filesystem out of the Docker container.



To do this, go back to a terminal window on your local desktop and navigate to the directory where you saved the Dockerfile. Enter the following command:

```
docker cp CONTAINER-NAME:/app/build/release/ bin_files
```

Replace `CONTAINER-NAME` with the name of your container in the Docker app. This copies the `/release` folder from your container in Docker to a folder called `/bin_files`.



Open the file location on your computer. You should see the five files now stored locally in the folder. This will allow you to use `esptool` and the ESP-IDF to load the files onto your ESP32-S3.

Upload the Files to the ESP32-S3

You will need two utilities from Espressif installed on your computer in order to upload the compiled files onto the ESP32-S3: `esptool` and the ESP-IDF.

`esptool`

You'll use `esptool` to upload the binary files to the ESP32-S3. `esptool` is installed with `pip`. Open a terminal window and enter:

```
pip install esptool
```

For more information on `esptool`, you can check out the [documentation from Espressif \(https://adafru.it/18Va\)](https://adafru.it/18Va).

Espressif Documentation for esptool

<https://adafru.it/18Va>

ESP-IDF

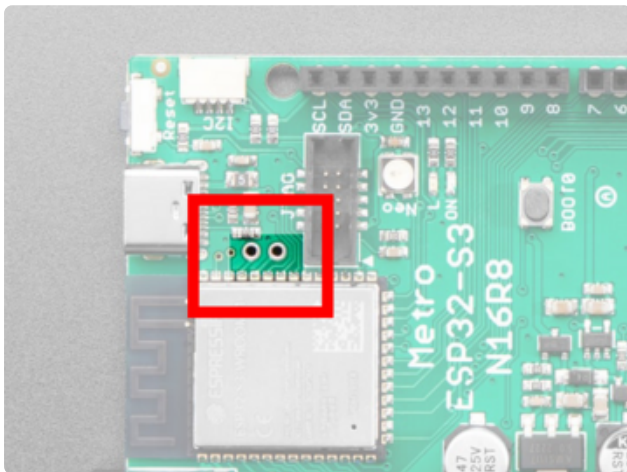
The kernel and filesystem are written to the ESP32-S3 using parttool.py, which is a tool included in the ESP-IDF. You can install the ESP-IDF as a part of your IDE or separately on your local computer. [Espressif has documentation and installers \(https://adafru.it/18Vb\)](https://adafru.it/18Vb) on their site.

ESP-IDF Installers and Installation Documentation

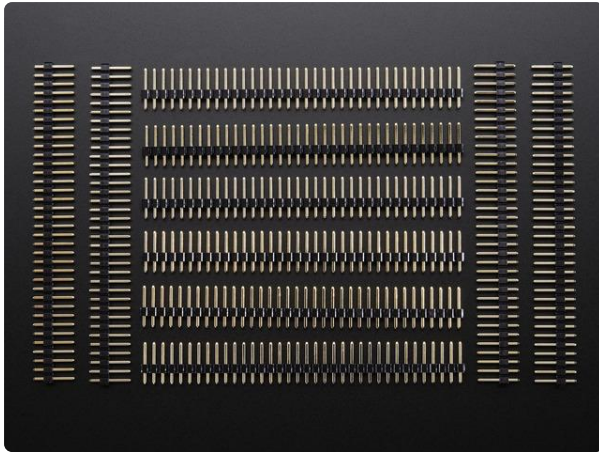
<https://adafru.it/18Vb>

Prep the Metro ESP32-S3

Now you finally get to play with hardware. You'll use the Metro ESP32-S3, which has 16 MB of Flash and 8 MB of PSRAM making it perfect for this skateboard trick.



You'll want to upload the files that you just built over hardware UART. The RX and TX hardware UART pins are accessible next to the ESP32-S3 module on the board. Solder two headers to those pins.



Break-away 0.1" 36-pin strip male header - Black - 10 pack

Breakaway header is like the duct tape of electronics. It's great for connecting things together, soldering to perf-boards, fits into any breakout or breadboard, etc. We go through...

<https://www.adafruit.com/product/392>

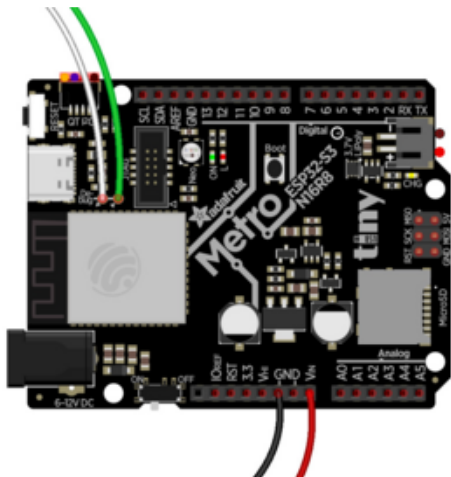


USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi

The cable is easiest way ever to connect to your microcontroller/Raspberry Pi/WiFi router serial console port. Inside the big USB plug is a USB<->Serial conversion chip and at...

<https://www.adafruit.com/product/954>

Attach a USB to serial cable to the Metro ESP32-S3. You should see the board power up once you plug in the USB cable.



- USB RX to ESP32-S3 TX (white wire)
- USB TX to ESP32-S3 RX (green wire)
- USB power to ESP32-S3 VIN (red wire)
- USB ground to ESP32-S3 GND (black wire)

Binary Files

First, you'll upload the three binary files (**bootloader.bin**, **network_adapter.bin** and **partition-table.bin**) to the ESP32-S3 with esptool.

Put the Metro ESP32-S3 into bootloader mode by holding down the boot button, pressing the reset button and then releasing the boot button. Then, open a terminal window and navigate to the directory where you saved the exported files from the Docker container. Enter this command to upload the three files to the ESP32-S3:

```
python esptool.py --chip esp32s3 -p YOUR-PORT-HERE -b 921600 --before=default_reset
--after=hard_reset write_flash 0x0 bootloader.bin 0x10000 network_adapter.bin
0x8000 partition-table.bin
```

Replace **YOUR-PORT-HERE** with the USB port number that the ESP32-S3 is attached to. On Linux or Mac it may look like **/dev/ttyUSB0**. On Windows, it will be COM followed by a number. For example, **COM3**.

File System and Kernel

Next, you'll use the ESP-IDF terminal to upload the file system and kernel file. Open an ESP-IDF terminal window and navigate to the directory where you saved the exported files from the Docker container. First, put the Metro ESP32-S3 into bootloader mode by holding down the boot button, pressing the reset button and then releasing the boot button. Then upload the kernel file (**xipImage**) first with:

```
parttool.py write_partition --partition-name linux --input xipImage
```

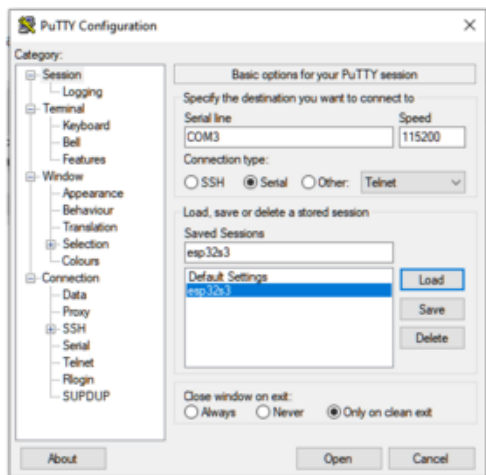
Put the Metro ESP32-S3 into bootloader mode again and then upload the file system file (**rootfs.cramfs**) with:

```
parttool.py write_partition --partition-name rootfs --input rootfs.cramfs
```

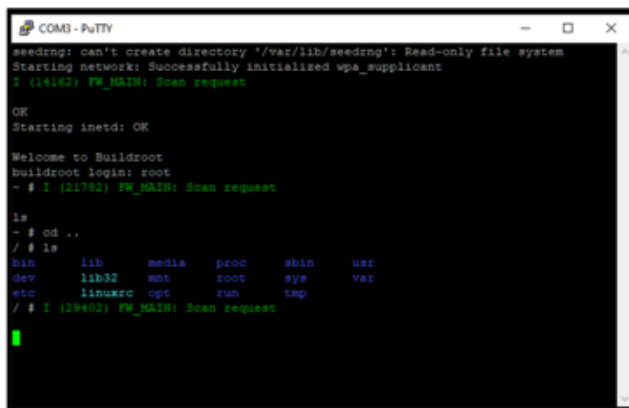
Next you'll boot into Linux on an ESP32-S3!

Boot Linux on the ESP32-S3

You'll be able to log into Linux on the ESP32-S3 by using a serial connection.



Using your preferred serial client, open a serial connection with the hardware UART port on the ESP32-S3 at a 115200 baudrate.



Press the reset button on the ESP32-S3. In your serial monitor, you should see Linux booting up. When its ready, it will prompt you for a login. Enter `root`. This will bring you to the filesystem.

You may see spurious `FW_MAIN` print statements randomly pop up in your serial monitor. This is the WiFi core printing status messages and is normal for this build.

WiFi

To connect to WiFi, enter the following commands:

```
ip link set espsta0 up
cat &gt; /tmp/wpa_supplicant.conf &&&EOF
network={
    ssid="YOUR-SSID-HERE"
    psk="YOUR-SSID-PASSWORD-HERE"
}
EOF
wpa_supplicant -B -i espsta0 -c /tmp/wpa_supplicant.conf &
udhcpc -i espsta0
```

Replace `YOUR-SSID-HERE` and `YOUR-SSID-PASSWORD-HERE` with your network SSID and SSID password. This creates a `wpa_supplicant` configuration file in the `/tmp` folder that will let you connect to a secured network.

```
COM3 - PuTTY
# ping google.com
PING google.com (142.250.81.238): 56 data bytes
64 bytes from 142.250.81.238: seq=0 ttl=116 time=33.517 ms
64 bytes from 142.250.81.238: seq=1 ttl=116 time=61.542 ms
64 bytes from 142.250.81.238: seq=2 ttl=116 time=24.840 ms
(283550) FW_MAIN: Scan request

^C
-- google.com ping statistics --
# packets transmitted, 3 packets received, 25% packet loss
round-trip min/avg/max = 24.840/39.966/61.542 ms
#
```

You can test your network connection by pinging a URL with:

```
ping google.com
```

Press Control + C to stop pinging the URL. You should see packets transmitted.

Customize

Otherwise there isn't a lot more that you can do with this Linux installation. The following page will show you how to run the [Buildroot menuconfig \(https://adafru.it/18Vc\)](https://adafru.it/18Vc) in the Docker container to customize the image and recompile it.

Customize with menuconfig

You've booted Linux on an ESP32-S3 and now you want to add some customization. You can do that with the menuconfig tool in `/buildroot`. This lets you select options to include in your kernel and filesystem. After running menuconfig, you'll rebuild the files and upload them to the ESP32-S3 partition tables so that you can boot into your new Linux image.

First you'll clean up the previously compiled files by running `make clean` in the `/buildroot` directory.

```
cd /app/build/buildroot
make clean
```

Next, you'll run some commands to rebuild the filesystem and kernel. You're essentially re-running commands manually that are performed in the Dockerfile. First, you'll go up one level in the directory with:

```
cd ..
```

Then you'll `clean` the previous compilation run with:

```
make -C buildroot O=`pwd`/build-xtensa-2023.02-fdpic-esp32s3 clean
```

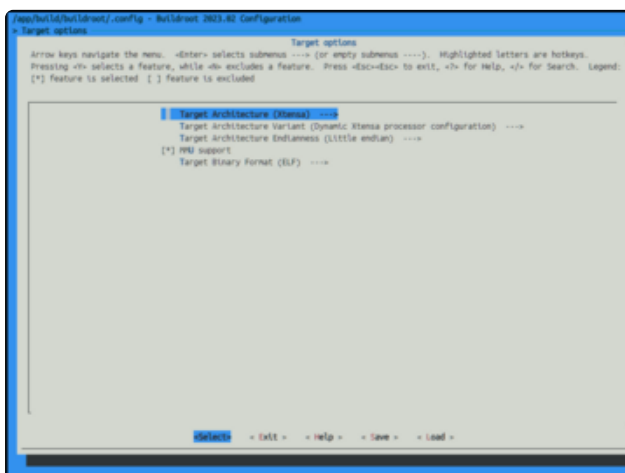
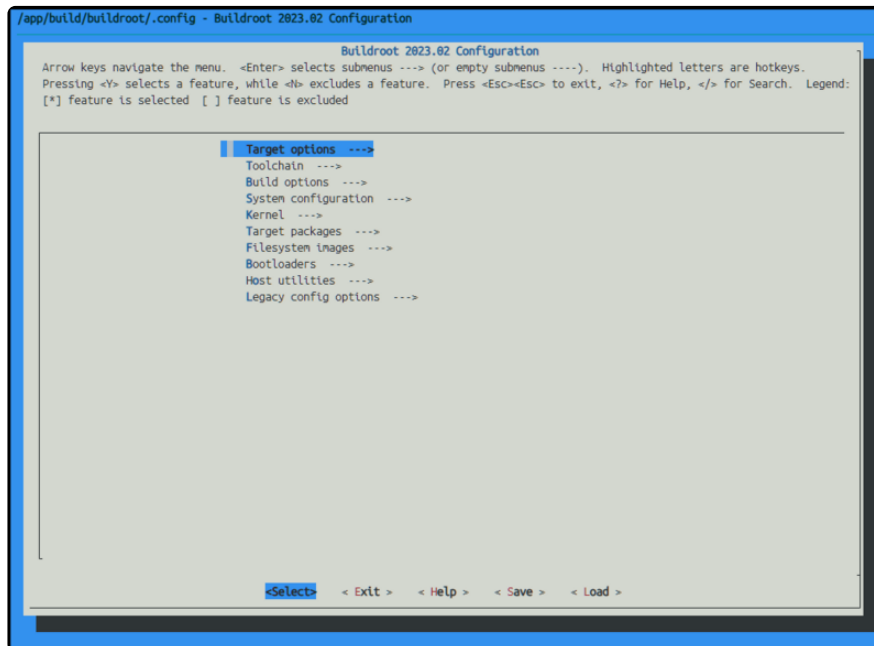
Run the following commands to prep the `.config` file. This file contains all of the build settings.

```
make -C buildroot 0=`pwd`/build-xtensa-2023.02-fdpic-esp32s3 esp32s3wifi_defconfig
buildroot/utlils/config --file build-xtensa-2023.02-fdpic-esp32s3/.config --set-str
TOOLCHAIN_EXTERNAL_PATH `pwd`/crosstool-NG/builds/xtensa-esp32s3-linux-uclibcfdpic
buildroot/utlils/config --file build-xtensa-2023.02-fdpic-esp32s3/.config --set-str
TOOLCHAIN_EXTERNAL_PREFIX '$(ARCH)-esp32s3-linux-uclibcfdpic'
buildroot/utlils/config --file build-xtensa-2023.02-fdpic-esp32s3/.config --set-str
TOOLCHAIN_EXTERNAL_CUSTOM_PREFIX '$(ARCH)-esp32s3-linux-uclibcfdpic'
```

Then you'll enter the `menuconfig` tool with:

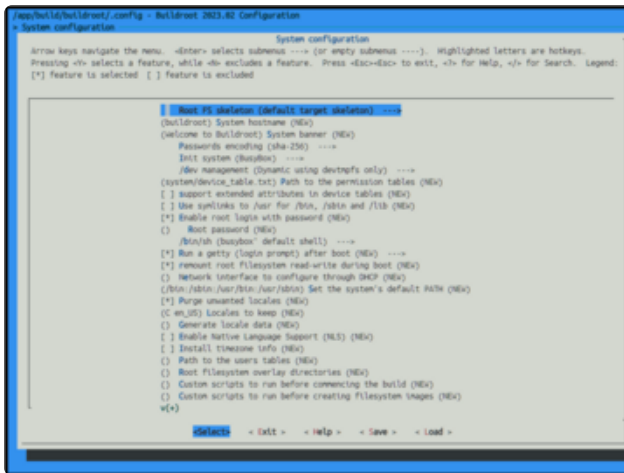
```
make -C buildroot 0=`pwd`/build-xtensa-2023.02-fdpic-esp32s3 menuconfig
```

This opens a GUI tool that you can navigate with the arrow keys on your keyboard.

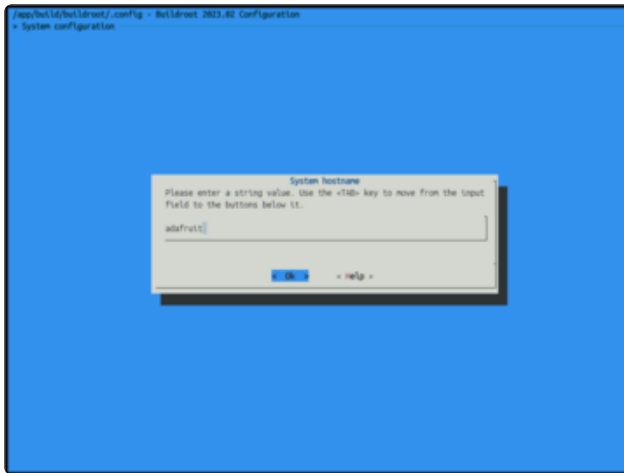


Start by going to **Target options** and making sure that the image is being built for the correct target.

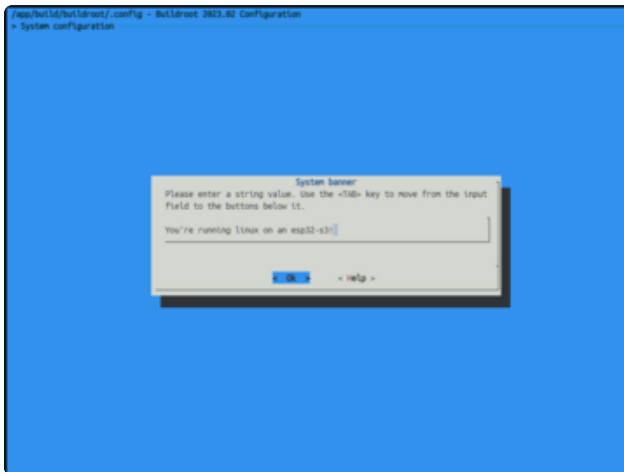
Target Architecture: **Xtensa**
Target Architecture Variant: **Dynamic Xtensa processor configuration**
Target Architecture Endianness: **Little endian**
(*) MMU support
Target Binary Format: **ELF**



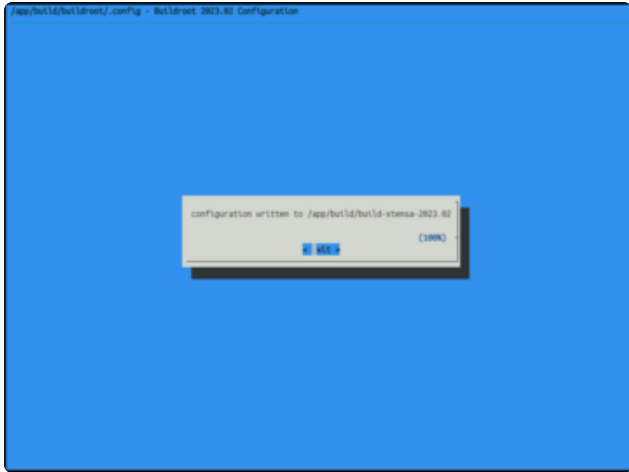
Navigate to the **System configuration** menu. On this page, you'll edit the **hostname** and **welcome banner**.



Select **System hostname** and enter the hostname that you would like for your system to have. Highlight **OK** and hit **enter** to confirm.



Select the **System banner** and enter the custom welcome message you'd like to see when you log into the system. Highlight **OK** and hit **enter** to confirm.



Select **Save** on the main GUI. This will open a dialog box showing that you are saving to the **.config** file. Select **OK** to confirm.

Now you'll use **make** to compile all of the build settings and build the filesystem and kernel files.

```
make -C buildroot 0=`pwd`/build-xtensa-2023.02-fdpic-esp32s3  
[ -f build/build-xtensa-2023.02-fdpic-esp32s3/images/xipImage -a -f build/build-  
xtensa-2023.02-fdpic-esp32s3/images/rootfs.cramfs ] || exit 1
```

You'll move the files to the **/release** folder with:

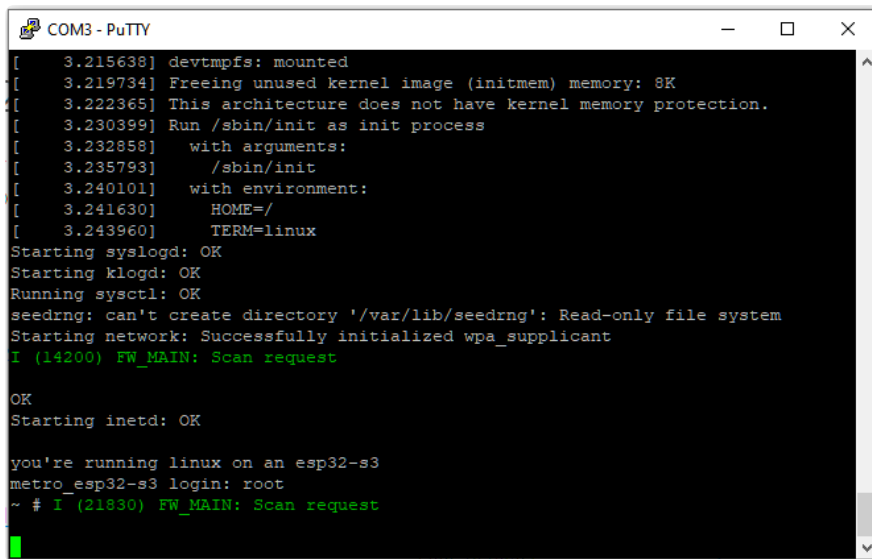
```
cd build  
cp build-xtensa-2023.02-fdpic-esp32s3/images/xipImage release  
cp build-xtensa-2023.02-fdpic-esp32s3/images/rootfs.cramfs release
```

You'll move the files from Docker to your local filesystem by opening a terminal on your local system, navigating to the directory where you want the files to be and entering:

```
docker cp YOUR-CONTAINER-NAME-HERE:/app/build/release bin_files
```

This is the same step that you did earlier with the unmodified files. Next, you'll flash the **xipImage** and **rootfs.cramfs** files using **parttool.py** in the ESP-IDF as described on the [Upload Files page \(https://adafru.it/18Vd\)](https://adafru.it/18Vd).

Boot into Linux



```
COM3 - PuTTY
[ 3.215638] devtmpfs: mounted
[ 3.219734] Freeing unused kernel image (initmem) memory: 8K
[ 3.222365] This architecture does not have kernel memory protection.
[ 3.230399] Run /sbin/init as init process
[ 3.232858]   with arguments:
[ 3.235793]     /sbin/init
[ 3.240101]   with environment:
[ 3.241630]     HOME=/
[ 3.243960]     TERM=linux
Starting syslogd: OK
Starting klogd: OK
Running sysctl: OK
secdng: can't create directory '/var/lib/secdng': Read-only file system
Starting network: Successfully initialized wpa_supplicant
I (14200) FW_MAIN: Scan request

OK
Starting inetd: OK

you're running linux on an esp32-s3
metro_esp32-s3 login: root
~ # I (21830) FW_MAIN: Scan request
```

Log into the ESP32-S3 over serial to see your Linux system in action. If all went well, you should be greeted with your custom welcome message and hostname.

Customize with ESP-IDF menuconfig

On the previous page you customized the kernel and filesystem with buildroot's menuconfig tool. The [ESP-IDF has a similar menuconfig \(https://adafru.it/18Ve\)](https://adafru.it/18Ve) that lets you adjust parameters for the targeted ESP chip, such as flash size and partition table. Just like how you recompiled the file system and kernel, you can recompile the bootloader, partition table and network adapter .bin files using the following steps.

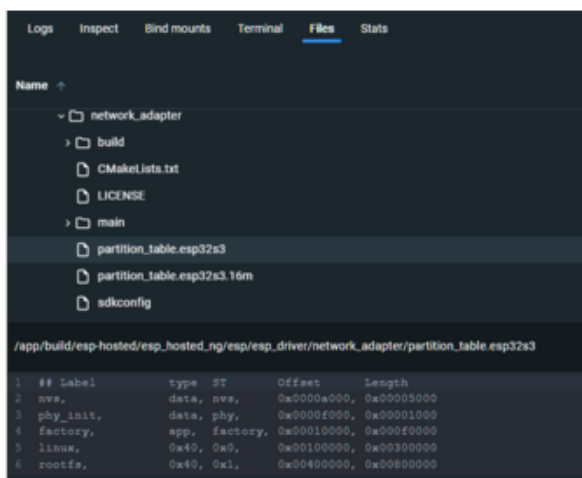
Partition Table

The partition table in an ESP-IDF project determines the file structure on the ESP32 chip. In the Docker container, the partition table file is located in `/app/build/esp-hosted/esp_hosted_ng/esp_driver/network_adapter/partition_table.esp32s3`

The default partition table is setup as follows:

## Label	type	ST	Offset	Length
nvs,	data,	nvs,	0x0000a000,	0x00005000

phy_init,	data,	phy,	0x0000f000,	0x00001000
factory,	app,	factory,	0x00010000,	0x000f0000
linux,	0x40,	0x0,	0x00100000,	0x00300000
rootfs,	0x40,	0x1,	0x00400000,	0x00400000



You can edit the partition table directly in the Docker container by going to the **Files** tab and navigating to the file location. **Right-click** on the partition table file and you can edit it in the text editor that opens at the bottom of the screen.

The changes to the partition table only take effect after recompiling the ESP-IDF project. These steps are outlined next.

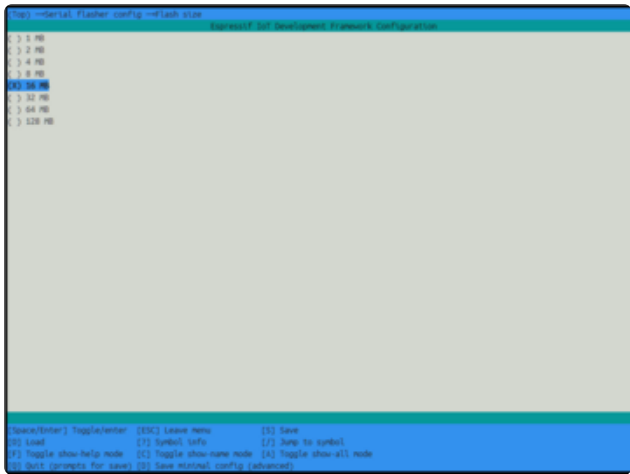
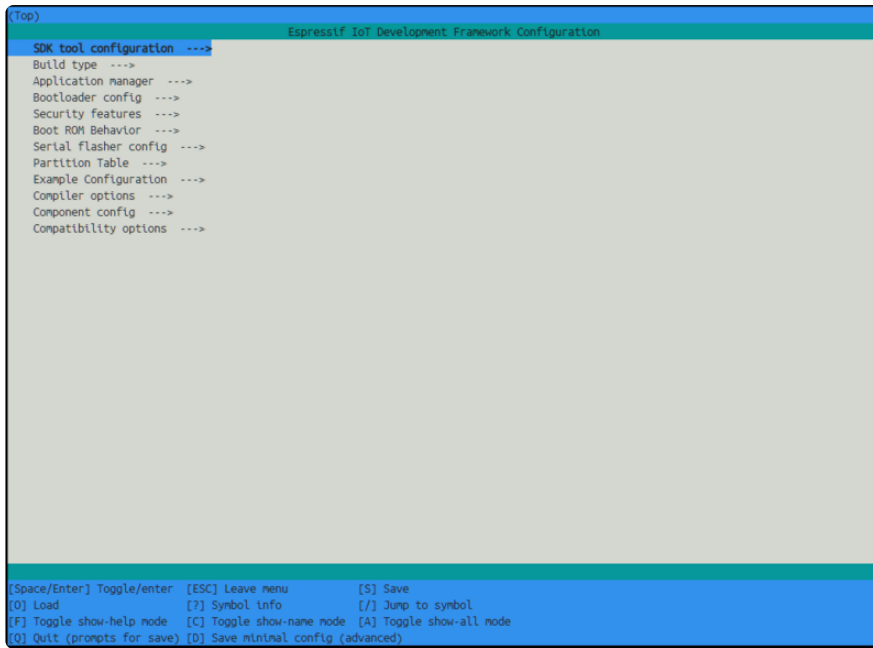
ESP-IDF menuconfig

To recompile the ESP-IDF generated .bin files, you'll run the ESP-IDF steps from the Dockerfile. In the **Terminal** tab in your Docker container, run these commands:

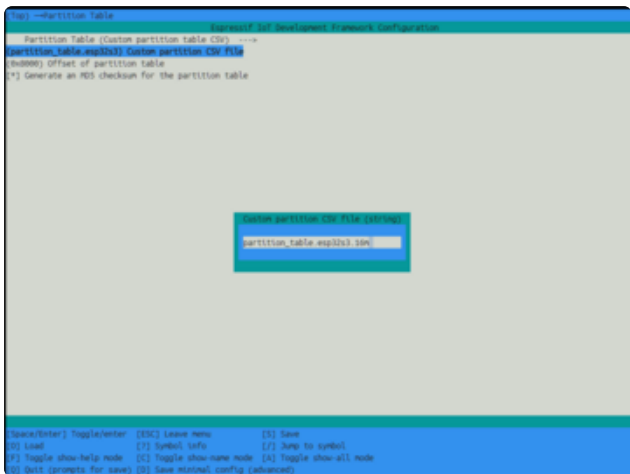
```
cd esp-hosted/esp_hosted_ng/esp/esp_driver &&& cmake .
cd esp-idf &&& . ./export.sh
cd ../network_adapter &&& idf.py set-target esp32s3
cp sdkconfig.defaults.esp32s3 sdkconfig
```

Then enter the **menuconfig** tool with:

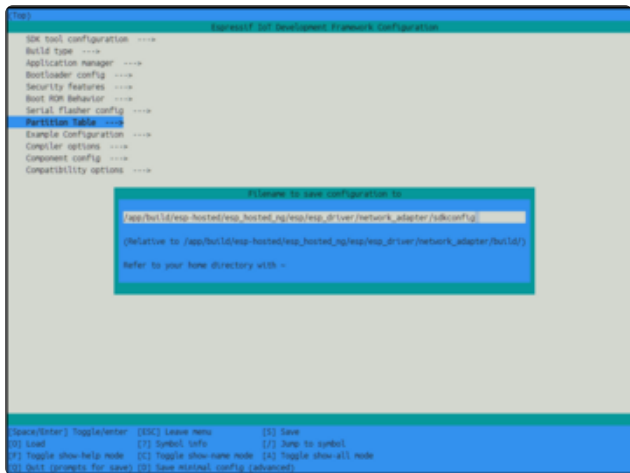
```
idf.py menuconfig
```



You can change the flash size to match your target board by navigating to **Serial flasher config - flash size**. There you can select the flash size on your board. In the case of the Metro ESP32-S3, its **16MB**.



You can also change or confirm your partition table by navigating to the **Partition Table** page.



When you're finished, save your configuration as `sdkconfig` (the default option) and then `exit menuconfig`.

Next you'll build your project with:

```
idf.py build
```

Finally you'll copy your generated `.bin` files to the `/releases` folder in the Docker container:

```
cd /app/build

cp esp-hosted/esp_hosted_ng/esp/esp_driver/network_adapter/build/bootloader/
bootloader.bin release
cp esp-hosted/esp_hosted_ng/esp/esp_driver/network_adapter/build/partition_table/
partition-table.bin release
cp esp-hosted/esp_hosted_ng/esp/esp_driver/network_adapter/build/
network_adapter.bin release
```

And copy the newly generated `.bin` files to your local filesystem, just as you did [earlier in the guide](https://adafru.it/18Vf). (<https://adafru.it/18Vf>)

```
docker cp CONTAINER-NAME:/app/build/release/ bin_files
```

Now you can load your new `.bin` files onto your [ESP32-S3 using esptool](https://adafru.it/18Vd). (<https://adafru.it/18Vd>)