Use ESPHome from the command line

By jean-claude.feltes@education.lu

ESPHome is an addon for HomeAssistant that allows programming controllers so that they can easily be integrated into HA. Why use it from the command line? Because it is so much faster, if run not on the Raspi running HA, but on another Linux computer.

1. Install Docker

```
sudo apt-get install docker.io
```

User must be in docker group

```
sudo adduser jcf docker
groups jcf
jcf : jcf adm dialout cdrom sudo dip plugdev lpadmin sambashare docker
```

and user must be in the plugdev group (for using USB to program controllers)

Test docker:

```
$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:aa0cc8055b82dc2509bed2e19b275c8f463506616377219d9642221ab53cf9fe
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

Use docker as non-root user

If you want to run docker as non-root user then you need to add it to the docker group.

1. Create the docker group if it does not exist

   ```
   $ sudo groupadd docker
   ```

2. Add your user to the docker group.

```
$ sudo usermod -aG docker $USER
```

3. Log in to the new `docker` group (to avoid having to log out / log in again; but if not enough, try to reboot):

```
$ newgrp docker
```

The newgrp command is used to change the current group ID during a login session. If the optional - flag is given, the user's environment will be reinitialized as though the user had logged in, otherwise the current environment, including current working directory, remains unchanged.

4. Check if docker can be run without root

```
$ docker run hello-world
```

Another test that runs an Ubuntu bash:

```
jcf@jclab:~$ docker run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
677076032cca: Pull complete
Digest: sha256:9a0bdde4188b896a372804be2384015e90e3f84906b750c1a53539b585fbbe7f
Status: Downloaded newer image for ubuntu:latest
root@6f231217afa9:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
root@6f231217afa9:/# exit
exit
jcf@jclab:~$
```

2. Install ESPHome on Docker

When the test above works, esphome can be installed to docker:

```
jcf@jclab:~$ docker pull esphome/esphome
Using default tag: latest
latest: Pulling from esphome/esphome
e9995326b091: Pull complete
...
1539b854fa9a: Pull complete
Digest: sha256:85713a8acb56af661ccdc0dbdb5779cb974c71dd89d8a126567d0dccef5ef1ce
Status: Downloaded newer image for esphome/esphome:latest
docker.io/esphome/esphome:latest
```

3. ESP32Cam

The module must be connected to the computer via an USB-serial interface:

https://www.instructables.com/Getting-Started-With-ESP32-CAM-Streaming-Video-Usi/

Connections:

U0R = Rx0 to Tx on USB adapter
U0T = Tx0 to Rx on USB adapter
5V – 5V
GND – GND
Short IO0 to GND for programming, open the connection after programming

Use the ESPHome wizard to create basic config:

```
docker run --rm -v "${PWD}":/config -it esphome/esphome wizard espcam.yaml
```

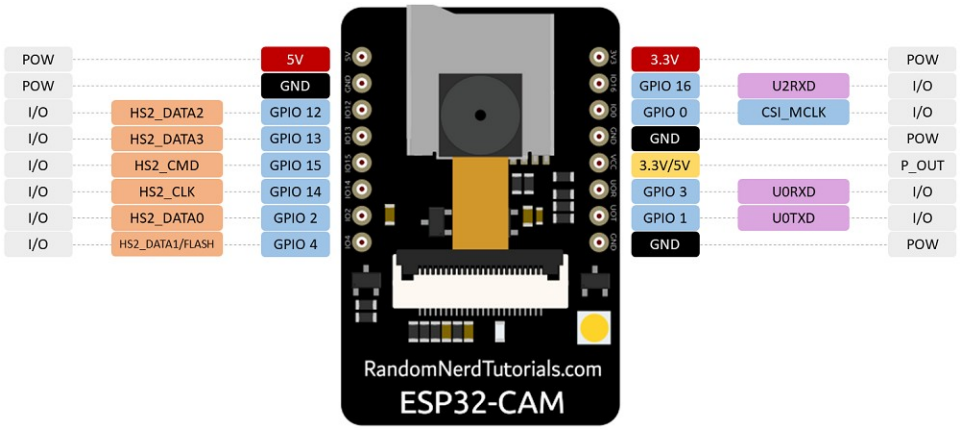
Follow the instructions of the wizard to create configuration for Core, controller, WiFi and OTA (Over the air update)

After all these easy steps, there will be a configuration file espcam.yaml in the current folder.

The file's owner is root, this can be changed to edit the file:

```
sudo chown jcf espcam.yaml
nano espcam.yaml
```

Edit the config file to add the camera:



https://randomnerdtutorials.com/esp32-cam-camera-pin-gpios/

https://sequr.be/blog/2021/03/esp32-cam-and-esphome/

We still have to integrate the code for the camera (at the bottom):

```
# ESP32-CAM
esp32_camera:
  external_clock:
    pin: GPIO0
    frequency: 20MHz
  i2c_pins:
```

```
    sda: GPIO26
    scl: GPIO27
  data_pins: [GPIO5, GPIO18, GPIO19, GPIO21, GPIO36, GPIO39, GPIO34, GPIO35]
  vsync_pin: GPIO25
  href_pin: GPIO23
  pixel_clock_pin: GPIO22
  power_down_pin: GPIO32

  name: mycamera

# Flashlight
output:
  - platform: gpio
    pin: GPIO4
    id: gpio_4

## GPIO_4 is the flash light pin
light:
  - platform: binary
    output: gpio_4
    name: camera light
```

Compile and upload code:

```
docker run --network=host --rm -v "${PWD}":/config --device=/dev//ttyUSB0 -it
esphome/esphome run espcam.yaml
```

network=host allows docker to use the network to install all the components needed, like PlatformIO and the Expressif SDK

After programming the firmware a reset is normally done via the RST line. As we have not connected that, we have to do the reset manually: remove the wire shorting GPIO0 to GND and push the reset button.

Leave the camera connected to see the boot messages, nicely coloured.
You can see if the WiFi is accessed for example, with indication of the signal strength. Well done!

Alternatively you can connect to a serial terminal, set the port to *dev*ttyUSBx, 115200 baud, and see what's going on.

Problems

At the first test my camera started to boot, but got lost in a loop of brownout detection messages.

This seems to be a common problem, as a search on the Internet confirmed. I soldered two capacitors 100nF // 100uF between +3.3V and GND (the shielding of the Sdcard holder is GND and near the 3.3V pin), and I used a very short good USB cable.

After this the camera worked and was easily integrated to HA, as it was already noticed by the system.

4. Another example with a ESP8266 D1mini

In this example, on the D1mini a switch between G6 = GPIO2 and GND is used as binary input and the internal LED (GPIO12) is used as output.

We have a basic config for the ESP32, but we need a new one for the ESP8266.

Create basic config file:

```
docker run --rm -v "${PWD}":/config -it esphome/esphome wizard test.yaml
```

Note that the controller must be pluged in for this step!

Now we have a basic config file test.yaml that we can edit.

Follow the instructions (4 steps)

Edit basic config to create specific config:

```
esphome:
  name: testd1mini

esp8266:
  board: d1_mini

# Enable logging
logger:

# Enable Home Assistant API
api:
  password: !secret api_password

ota:
  password: !secret ota_password

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Testd1Mini Fallback Hotspot"
    password: !secret ap_password

captive_portal:

# switch LED from HA:
switch:
  - platform: gpio
    name: "LED"
    pin:
      number: GPIO2
      inverted: True

# Show state of a switch in HA
```

```
binary_sensor:
  - platform: gpio
    name: "Switch_1"
    pin:
      number: GPIO12
      inverted: True
      mode: INPUT_PULLUP
```

In this example a little trick has been used to modularize code that will be used more than once: the ssid and passwords have been put into a file secrets.yaml

This file contains the lines

```
wifi_ssid: "..."
wifi_password: "..."
ap_password: "..."
api_password: "..."
ota_password: "..."
```

Compile and program the device:

```
docker run --network=host --rm -v "${PWD}":/config --device=/dev//ttyUSB0 -it
esphome/esphome run test2.yaml
```

Observe the boot messages indicating for example the quality of the WiFi network.

Integrate device in HA:

After this, the device should be found by HA and you get a notification about a new device.

To integrate it you must enter the api password defined in the configuration file.

5. Modularising further: includes and substitutions

The aim is to put common config items into separate files, so they can be reused for other devices, and so that the main code gets easyer to overview.

Until now we have the following structure for the D1mini example:

test.yaml reads ssid and passwords from secrets.yaml

We can take out the non specific part

```
# Enable Home Assistant API
api:
  password: !secret api_password

ota:
  password: !secret ota_password

wifi:
```

```
ssid: !secret wifi_ssid
password: !secret wifi_password

# Enable fallback hotspot (captive portal) in case wifi connection fails
ap:
  ssid: "Testd1Mini Fallback Hotspot"
  password: !secret ap_password
```

and put it into a file commonconfig.yaml

This can then be included into the main config file via

```
<<: !include commonconfig.yaml
```

(at the place where the cut out code block was before).

Eventually the include file can be put into a separate folder “inc”. Of course this must be taken into consideration for the include statement:

```
<<: !include inc/commonconfig.yaml
```

There is another trick: names that are used more often can be substituted:

```
substitutions:
  devicename: mini2

esphome:
  name: $devicename

...

<<: !include commonconfig.yaml

...

# switch LED from HA:
switch:
  - platform: gpio
    name: $devicename LED
…

# Show state of a switch in HA
binary_sensor:
  - platform: gpio
    name: $devicename Button
    ...
```

This way, all the elements have the same prefix “mini2”:

6. Using a temperature sensor

We use a DS18B20 from Dallas communicating over the one wire bus.

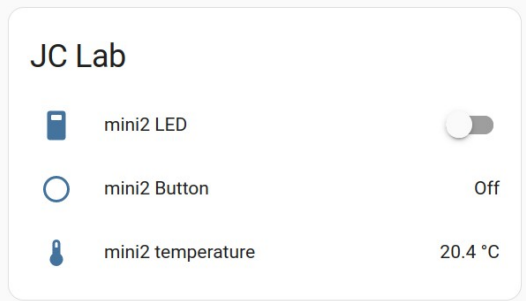
Pinout: 1 = GND, 2 = Data (with 4k7 pullup), 3 = VCC +3.3V

The data pin is connected to GPIO 13 = D7 of the D1 mini.

This short addition to the configuration file is all that is needed:

```
dallas:
  - pin: GPIO13

sensor:
  - platform: dallas
    index: 0
    name: $devicename temperature
```



7. Using multiple DS18B20 sensors

Multiple sensors may be connected to one pin. They are distinguished by their unique ID.

To get the ID information, we can activate debugging in the yaml file:

```
# Enable logging
logger:
  level: DEBUG
```

On boot we get the information via USB serial:

```
[07:46:38][C][dallas.sensor:075]: DallasComponent:
[07:46:38][C][dallas.sensor:076]:   Pin: GPIO13
[07:46:38][C][dallas.sensor:077]:   Update Interval: 60.0s
[07:34:53][D][dallas.sensor:082]:   Found sensors:
[07:34:53][D][dallas.sensor:084]:     0x9c000802c37d0210
[07:34:53][C][dallas.sensor:089]:   Device 'mini2 temperature'
[07:34:53][C][dallas.sensor:089]:     Device Class: 'temperature'
[07:34:53][C][dallas.sensor:089]:     State Class: 'measurement'
[07:34:53][C][dallas.sensor:089]:     Unit of Measurement: '°C'
[07:34:53][C][dallas.sensor:089]:     Accuracy Decimals: 1
[07:34:53][C][dallas.sensor:091]:     Index 0
[07:34:53][C][dallas.sensor:097]:     Address: 0x9c000802c37d0210
[07:34:53][C][dallas.sensor:098]:     Resolution: 12
```

There are other interesting informations like Pin and update interval.

When we add the second sensor, we see its address the same way.

Now we can give a name to these addresses:

```
# measure temperature
dallas:
  - pin: GPIO13

sensor:
  - platform: dallas
    address: 0x9c000802c37d0210
    name: $devicename temp. 1

  - platform: dallas
    address: 0x15012113b0510128
    name: $devicename temp. 2
```

More configuration options can be found here:

https://esphome.io/components/sensor/dallas.html?highlight=ds18b20

For example the standard update interval can be changed (for all sensors):

```
dallas:
  - pin: GPIO13
    update_interval: 1s
```

(This very short interval makes only sense for demonstration effects like warming the sensor with the hand and looking at the temperature)

In the History menu you can select an area / device / entity and display all or several sensor values.

You can set beginning and end date and time.

In our example one sensor was warmed up by hand, the other was left alone.

Watch also the state information about LED and switch.

8. Going on from here …

To see what's possible, look here:

https://esphome.io/index.html#misc-components

Some interesting ones:

- pulse counter:
  https://esphome.io/components/sensor/pulse_counter.html
- non invasive powr meter:
  https://esphome.io/cookbook/power_meter.html
- status led indicating the state of a device:
  https://esphome.io/components/status_led.html
- basic objects for custom components programmed in C++:
  https://esphome.io/custom/custom_component.html
- custom components for I2, SPI, UART:
  https://esphome.io/custom/custom_component.html
  https://esphome.io/custom/spi.html
  https://esphome.io/custom/uart.html

- port expander:
  https://esphome.io/components/pcf8574.html
- ADC:
  https://esphome.io/components/sensor/adc.html
  https://esphome.io/components/sensor/ads1115.html

9. Monitoring the chip voltage

```
sensor:
  - platform: adc
    pin: VCC
    name: "VCC Voltage"
```

This monitors the internal 3.3V supply of the chip (not the voltage on the USB plug!)

10. Literature

Denis Bodor: “Home Assistant: Domotique vite fait, bien fait” in Hackable 46/2023 (in french)