Portenta X8: Az okosotthon karmestere

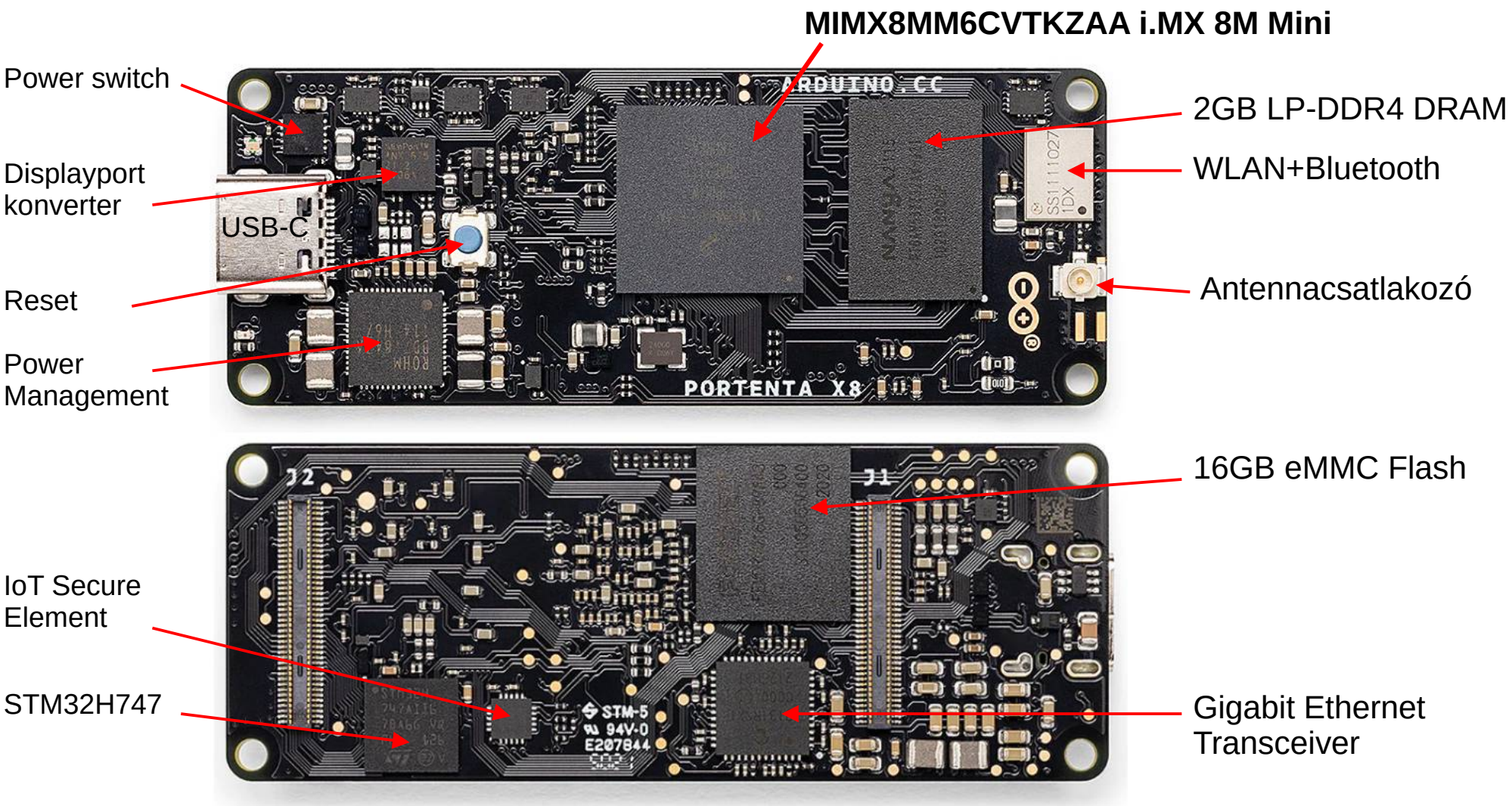
Felhasznált és ajánlott irodalom

- Arduino Llc. - Portenta X8 User Manual
- Tasmota Dokumentáció – https://tasmota.github.io/docs
- Tasmota firmware kiadások – https://github.com/arendst/Tasmota/releases
- Benzino77: TasmoCompiler
- Tasmota Web Installer – https://tasmota.github.io/install
- ESPTool – https://github.com/espressif/esptool
- MQTT alapok – https://www.hivemq.com/mqtt-essentials
- MQTT Explorer – https://mqtt-explorer.com
- MQTT Dash – play.google.com/store/apps/details?id=net.routix.mqttdash

Az Arduino Portenta X8 kártya

- Az **Arduino Portenta X8** egy nagy teljesítményű rendszer-modul (SOM), amelyet ipari IoT-alkalmazások kiszolgálására terveztek.
- A lap az **NXP i.MX 8M Mini** processzort — amely beágyazott Linuxot futtat — ötvözi az **STM32H7** mikrokontrollerrel, hogy kihasználhassa az Arduino könyvtárak és fejlesztési lehetőségek előnyeit is
- A **Portenta X8** funkciói bővíthetők hordozópanellel és fedlapokkal, s ezek referenciaként is szolgálhatnak egyedi rendszerek fejlesztéséhez

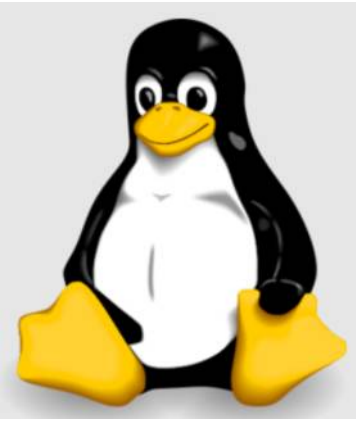
- A **Portenta X8** Wi-Fi és Bluetooth kapcsolattal rendelkezik, emellett - hordozó lap esetén - egy Gigabit Ethernet interfész is rendelkezésre áll ami nagy sebességet és alacsony késleltetést biztosít a legigényesebb alkalmazásokhoz


MIMX8MM6CVTKZAA i.MX 8M Mini
Power switch
Displayport konverter
USB-C
Reset
Power Management
2GB LP-DDR4 DRAM
WLAN+Bluetooth
Antennacsatlakozó
16GB eMMC Flash
IoT Secure Element
STM32H747
Gigabit Ethernet Transceiver
ARDUINO.CC
PORTENTA X8
J2
J1

Műszaki paraméterek

NXP i.MX 8M Mini Processor	4x Arm Cortex-A53 core up to 1.8 GHz 1x Arm Cortex-M4 core up to 400 MHz
STM32H747XI Microcontroller	1x Arm Cortex-M7 core up to 480 MHz, double precision FPU 1x Arm Cortex-M4 core up to 240 MHz with FPU
Onboard memory	2GB Low Power DDR4 DRAM 16GB Foresee eMMC Flash module
USB-C	High Speed USB DisplayPort output Host and Device operation Power Delivery support
WiFi/Bluetooth modul	Wi-Fi 802.11b/g/n 65 Mbps (2.4 GHz) Bluetooth 5.1 BR/EDR/LE
High Density connectors	PCI express, Ethernet interface, 2xUSB HS, 4xUART, 3xI2C, 2xSPI, 4xPWM, 1x SD card, MIPI DSI/CSI, 8xADC, 7xGPIO

Portenta X8: Linux környezet

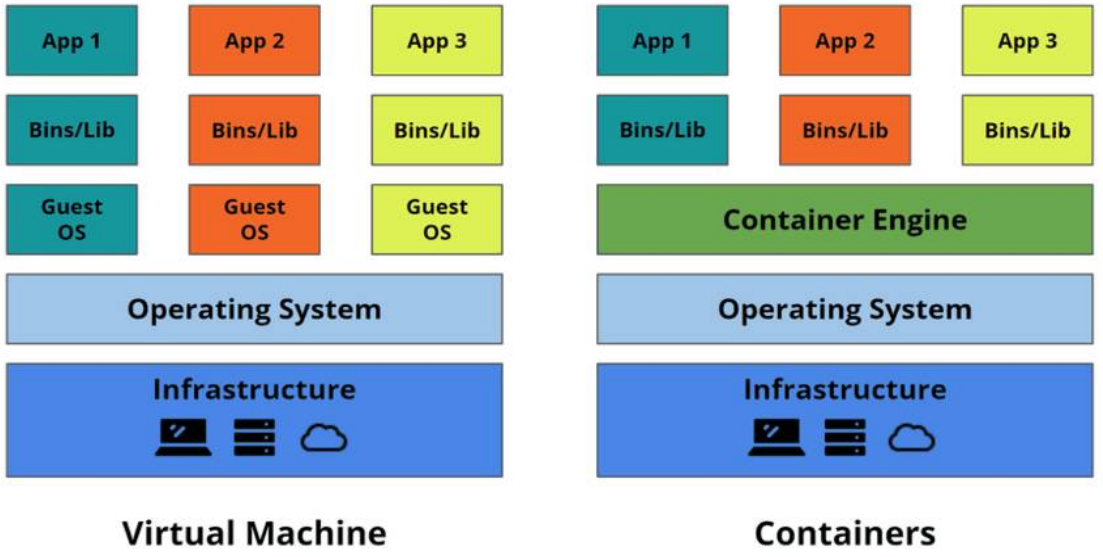


- A **Portenta X8** egy előre elkészített, Yocto-alapú Linuxot futtat, amelyet kifejezetten ehhez a hardverhez optimalizáltak
- A rendszer teljes értékű Linux-környezetet ad: shell, fájlrendszer, csomagkezelés, hálózati szolgáltatások, fejlesztői eszközök
- A Linux-oldal stabil, ipari felhasználásra szánt, és közvetlenül támogatja a Wi-Fi-t, Bluetooth-t és a gigabites Ethernetet
- A fejlesztéshez kész SDK és standard Linux-eszközök használhatók
- A Portenta X8 gyári Linux kiadása konténer-futtatási környezetet tartalmaz ; ezt az Arduino a könnyű telepítés és frissítés miatt választotta
- Az izoláció alkalmazás-szintű: a konténer saját fájlrendszer-nézetet és folyamatkörnyezetet kap, de a kernel közös, így a konténerizálás nem biztonsági határ, inkább stabilitási és karbantarthatósági előny

Konténeres futtatási környezet a Portenta X8-on

- A konténerek elkülönített futtatási környezetet adnak, így az alkalmazások nem zavarják egymást, és ugyanúgy futnak fejlesztői gépen, mint a célhardveren
- A megoldás egyszerűsíti a fejlesztést: minden alkalmazás a saját függőségeivel együtt érkezik, így nincs verzióütközés vagy rendszer-szintű beállítási igény
- Hátrány: bizonyos komponensek (pl. könyvtárak, runtime-ok) többszörösen is megjelenhetnek a rendszerben
- A Portentánál használt konténerek másképp valósítják meg az elkülönítést, mint a virtuális gépek — ezt szemlélteti az alábbi ábra

Az ábra forrása:
Portenta X8 User Manual



Docker konténer-logisztika

- A **Portenta X8** konténer-kezelője a **Docker,** amelyet az alábbi parancsokkal irányíthatunk:
- **docker pull <kép>** – Az „áru” (Image) beraktározása a Docker Hubról.
- **docker images** – A helyi raktárkészlet (letöltött képek) listázása
- **docker run -d <kép>** – Konténer indítása a háttérben
- **docker ps** – A jelenleg mozgásban lévő (futó) konténerek ellenőrzése
- **docker ps -a** – Az összes konténer listázása (a leállítottaké is)
- **docker stop <id>** – Egy futó konténer kíméletes leállítása
- **docker start <id>** – Egy már létező, de leállt konténer újraindítása
- **docker restart <id>** – Gyors újraindítás (ha valami „megakadt”)
- **docker rm <id>** – A már nem használt konténer (a „doboz”) kidobása
- **docker rmi <kép>** – A letöltött kép (a „tervrajz”) végleges törlése
- **docker system prune** – Törli a leállított konténereket, és az árva képeket is
- **docker logs -f <id>** – Élő betekintés a konténer „naplójába” (mi történik bent?)
- **docker exec -it <id> sh** – Belépés a konténer belsejébe „szerelni”

Újratelepítés és beüzemelés

Újratelepítési és beüzemelési útmutató – 1.

Első használat előtt szükség lehet egy „tavaszi nagytakarításra”, ennek lépései:

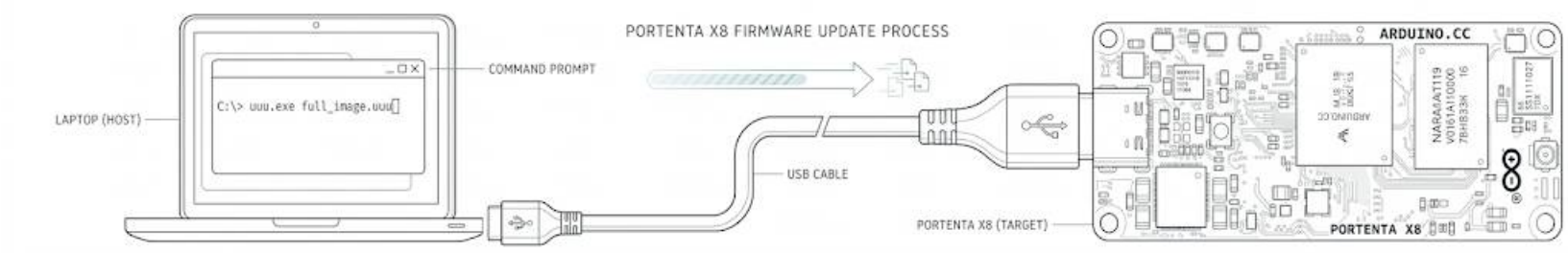
- **1. Előkészületek (PC oldal)**
  - **ADB Tool:** Töltsük le és bontsuk ki az Android Platform Tools csomagot
  - **Firmware:** Töltsük le a legfrissebb Portenta X8 firmware image-et
  - **Kibontás:** A firmware kibontása után a belső *.tar.gz* és *.wic.gz* fájlokat is ki kell bontani a fő mappába. Az **mfgtool-files-portenta-x8** mappában lesz az **uuu.exe**
- **2. A Boot szekciók törlése (Flashing Mode előkészítése)**
  - Csatlakoztassuk a kártyát USB-n, majd nyissunk egy terminált (pl. CMD.exe):
  - **Belépés:** `adb shell` (User: *fio*, Jelszó: *fio* vagy üres)
  - **Root jog megszerzése:** `sudo -s`
  - **Bootloader kényszerítése** (BOOTSEL gomb híján a boot szektorokat kell kitörölni)

```
echo 0 > /sys/block/mmcblk2boot0/force_ro
dd if=/dev/zero of=/dev/mmcblk2boot0 bs=1024 count=4096 && sync
echo 0 > /sys/block/mmcblk2boot1/force_ro
dd if=/dev/zero of=/dev/mmcblk2boot1 bs=1024 count=4096 && sync
exit
```

Újratelepítési és beüzemelési útmutató – 2.

- **Megjegyzés:** A boot szektorok törlésével azt értük el, hogy a következő újraindításkor a kártya nem tud rendesen elindulni, ezért automatikusan a DFU letöltő módba lép és USB-n keresztül várja az új firmware-t. De mielőtt újraindítanánk a kártyát, előtte el kell indítanunk a PC oldali **uuu.exe** letöltő programot, az alábbiak szerint:

- **3. Firmware újraírása az NXP UUU Tool segítségével**
  - Lépjünk be a PC-n egy parancsablakkal az **mfgtool-files-portenta-x8** mappába
  - **Indítsuk el** a figyelő módot: `uuu full_image.uuu`
  - **Hardveres reset:** Húzza ki, majd dugja vissza az USB kábelt. Ha nem indul el (marad a "Waiting for device" üzenet), próbálja meg mégegyszer a ki-be húzást
  - Várjuk meg, amíg a teljes folyamat eléri a **100%** / **Done** állapotot



Újratelepítési és beüzemelési útmutató – 3.

- **4. A gyári konténerek eltávolítása**
  A **Portenta X8** alapértelmezetten futtat néhány gyári **Docker** konténert (például az **Arduino Cloud** regisztrációhoz és a helyi webes felülethez). Ezek végleges leállítása:
- **a) Belépés a kártyára:** `adb shell`
  - **Váltás root jogra:** `sudo -s`
  - A `docker ps` parancs kimenete alapján az alábbi három konténer fut a háttérben, amelyeket a gyári image készítésekor konfiguráltak:
    - **x8-webapp**: Ez szolgálja ki a regisztrációs weboldalt.
    - **x8-provisioning**: Az Arduino Cloud-hoz való kényszerített csatlakozásért felelős
    - **x8-devel**: Egy előre telepített Python fejlesztői shell környezet
- **b) Az indító szervizek letiltása** (hogy a Systemd ne akarja újraindítani a konténereket):

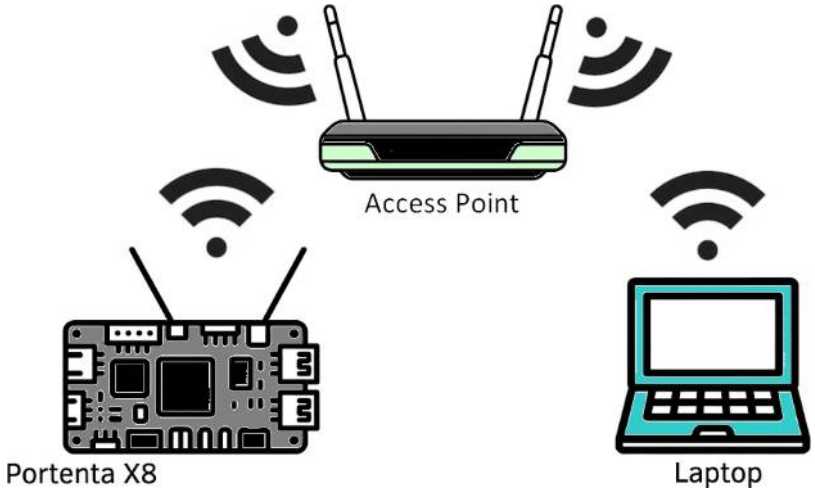
```
systemctl stop compose-apps-early-start.service
systemctl disable compose-apps-early-start.service
systemctl stop compose-apps-early-start-recovery.service
systemctl disable compose-apps-early-start-recovery.service
```

Újratelepítési és beüzemelési útmutató – 4.

- **c) A futó konténerek leállítása és törlése**

  Leállítjuk a három fölösleges konténert (**x8-webapp, x8-provisioning, x8-devel**), majd töröljük az image-eiket is.
  - **Futó konténerek leállítása:** `docker stop x8-webapp x8-provisioning x8-devel`
  - **Konténerek eltávolítása:** `docker rm x8-webapp x8-provisioning x8-devel`
  - **Image-ek és felesleges adatok (volumes) végleges kisöprése:**
    ```
    docker rmi -f x8-webapp x8-provisioning x8-devel
    docker system prune -a --volumes -f
    ```
- **d) A Docker újraindítása**
  - Hogy a rendszer tiszta állapotba kerüljön: `systemctl restart docker`
- **Ellenőrzés:** A `docker ps` parancsnak most már semmit nem szabad mutatnia, és a 80-as port is teljesen szabaddá vált

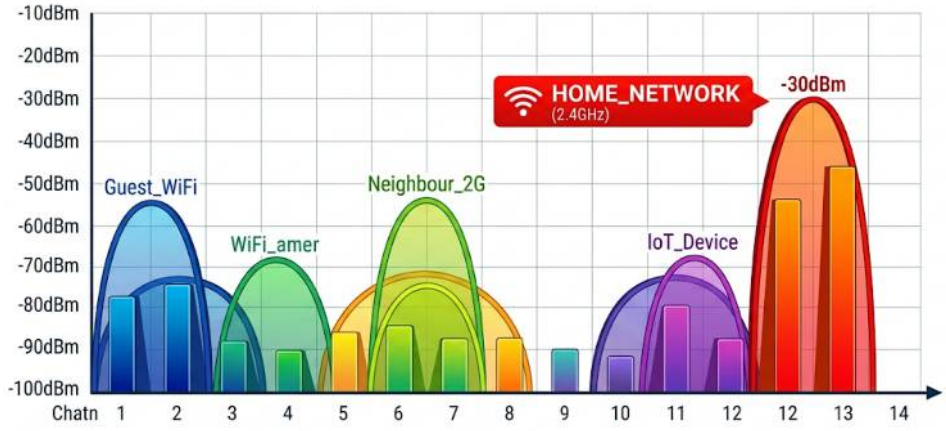
WiFi beállítások



- A **Portenta X8** kártyát két WiFi hálózatra történő automatikus csatlakozásra tanítjuk be: az otthoni hálózatra és egy mobil hotspotra (az utóbbi lehetővé teszi a hordozhatóságot)
- **A WiFi kézi inicializálása:**
  - **A rádió bekapcsolása:** Ha a Wi-Fi szoftveresen le van tiltva, az alábbi paranccsal kell aktiválni az eszközt: `sudo nmcli radio wifi on`
  - **Első kézi csatlakozás:** Az interfész felélesztéséhez és a hálózati profil létrehozásához elengedhetetlen egy egyszeri kézi kapcsolódás minden olyan hálózathoz, amelyekhez automatikus csatlakozni akarunk, prioritásos alapon:
    `sudo nmcli device wifi connect "Hálózat_Neve" password "Jelszó"`
    Ez a lépés automatikusan létrehozza a profilt, kér egy IP-címet a routertől (DHCP), és beállítja, hogy az eszköz minden későbbi indításkor magától csatlakozzon

Regionális korlátozások feloldása

- Az alapértelmezett WiFi régiókód (00) szűkített csatornakészletet használ.
  A HU régiókód beállítása szükséges a 12. és 13. csatornán sugárzó hálózatok észleléséhez



- **Régiókód beállítása:**
  - Ellenőrzés: `iw reg get`
  - Beállítás: `sudo iw reg set HU`
  - Frissítés: `sudo nmcli radio wifi off && sleep 2 && sudo nmcli radio wifi on`
- **A régiókód beállítás rögzítése (hogy reboot után se felejtse el):**
  - Megoldás: A Wi-Fi alrendszerért felelős kernelmodul (cfg80 211) paramétereinek módosítása:

```
echo "options cfg80211 ieee80211_regdom=HU" | sudo tee /etc/modprobe.d/cfg80211.conf
```

- **Felderítés és Kapcsolódás** (hasznos ellenőrző és kézi beavatkozó parancsok)
  - Elérhető hálózatok keresése: `sudo nmcli device wifi rescan`
  - Elérhető hálózatok listázása: `nmcli device wifi list`
  - Kézi csatlakozás: `sudo nmcli device wifi connect "SSID" password "JELSZÓ"`

WiFi hálózati prioritás

- **Prioritások beállítása** *(Nagyobb szám = Erősebb prioritás)*
  Engedélyezzük az automatikus csatlakozást és beállítjuk az alapértelmezett sorrendet
  - **Mobil hotspot:** `sudo nmcli connection modify "CSP-RN7" connection.autoconnect yes connection.autoconnect-priority 100`
  - **Otthoni router:** `sudo nmcli connection modify "HUAWEI-2.4G-9bQR" connection.autoconnect yes connection.autoconnect-priority 10`
- **Megjegyzés:**
  - A prioritás akkor lesz figyelembe véve, amikor a gép hálózatot keres
  - Ha már van kapcsolat, akkor csak a lentebb látható **connection up** paranccsal lehet "átrugdosni" a másik hálózatra
- **Állapotvizsgálat és Kézi vezérlés** (hasznos ellenőrző és kézi beavatkozó parancsok)
  - **Táblázat megnyitása:**
    `nmcli -f NAME, UUID, AUTOCONNECT, AUTOCONNECT-PRIORITY connection show`
  - **Kényszerített váltás:** `sudo nmcli connection up "CSP-RN7"`
  - **IP cím lekérése:** `ip addr show wlan0 | grep inet`

Intelligens névfeloldás (mDNS) és hálózati azonosítás

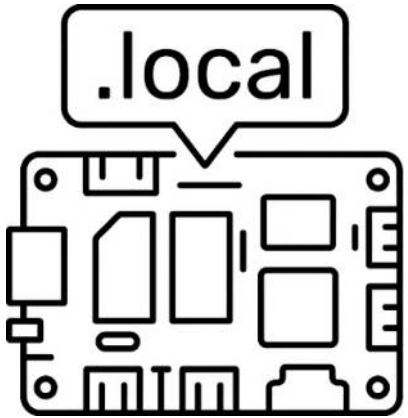
- **Cél:** biztosítani, hogy a **Portenta X8** egyedi névvel (**portenta.local**) azonosítsa magát a hálózaton. Ez lehetővé teszi pl. a **Tasmota** eszközök számára, hogy az aktuális IP-címtől függetlenül, név alapján találják meg az MQTT brókert
- Mielőtt bármit módosítanánk, ellenőrizzük a meglévő statikus névfeloldást:

```
cat /etc/hosts
```

A fájlban szerepelnie kell egy bejegyzésnek, amely a 127.0.1.1 (vagy 127.0.0.1) címet a választott gépnévhez rendeli (pl. 127.0.1.1 portenta)

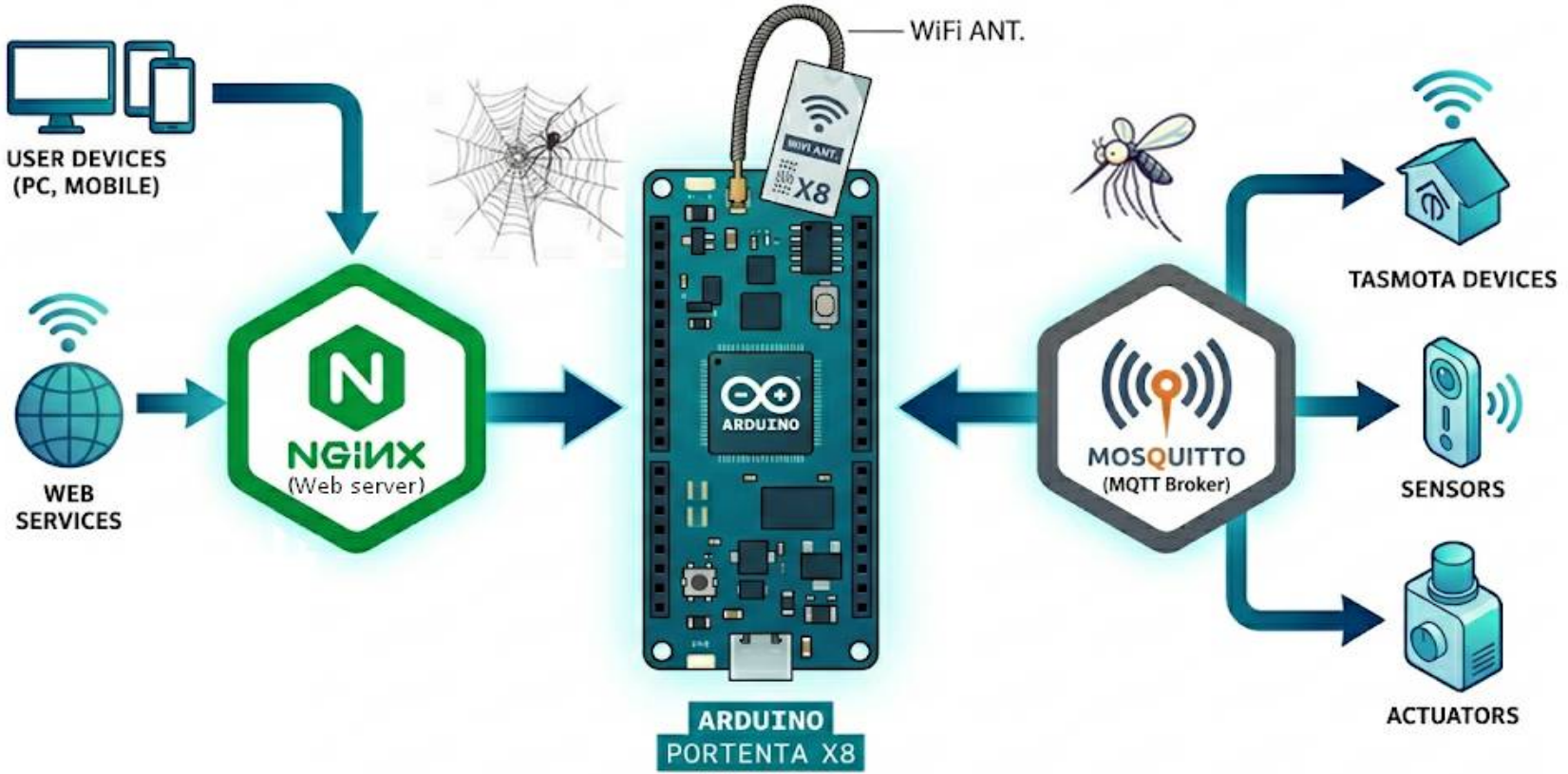
- **A gépnév beállításához** a következő parancsok kellenek:

```
# Gépnév beállítása
sudo hostnamectl set-hostname portenta

# Az /etc/hosts fájl frissítése (hogy önmagát is felismerje)
sudo sed -i 's/127.0.1.1.*/127.0.1.1 portenta/' /etc/hosts

# Az Avahi (mDNS) újraindítása az érvényesítéshez
sudo systemctl restart avahi-daemon
```



IoT szerver telepítés

- Az alábbiakban egy webszerver és egy MQTT szerver telepítését mutatjuk be



Docker adat-root átirányítása (Perzisztencia)

- Az első lépés a **Docker** adatainak átköltöztetése a felejtő (ideiglenes) mappákból a **/home/fio** alá. Ez biztosítja, hogy a **Docker** image-ek ne tűnjenek el reboot során

```
# Mappa létrehozása a külső tárolónak
sudo mkdir -p /home/fio/server/docker-storage

# Daemon konfiguráció felülírása
sudo tee /etc/docker/daemon.json <<EOF
{
  "max-concurrent-downloads": 1,
  "max-concurrent-uploads": 1,
  "data-root": "/home/fio/server/docker-storage"
}
EOF

# Docker újraindítása az új hely használatához
sudo systemctl restart docker
```

- **Eredmény:** A `/home/fio/server/docker-storage` mappába kerül minden letöltött image és konténer-adat, így reboot vagy firmware-frissítés után is megmaradnak

Könyvtárszerkezet és konfigurációs fájlok

- A szerver minden fájlja a **/home/fio/server** mappában helyezkedjen el:

```
# Könyvtárszerkezet létrehozása
mkdir -p /home/fio/server/www                    # Weboldal fájljainak helye
mkdir -p /home/fio/server/mosquitto/config       # Mosquitto konfigurációs fájlok
mkdir -p /home/fio/server/mosquitto/data         # Itt tárolja majd az üzeneteket
mkdir -p /home/fio/server/mosquitto/log          # Naplófájlok helye

# Alapértelmezett konfigurációs fájl létrehozása
cat <<EOF > /home/fio/server/mosquitto/config/mosquitto.conf
listener 1883
allow_anonymous true
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
EOF
```

Tulajdon- és hozzáférési jog beállítása

- A telepíteni kívánt kiszolgálóknak (**nginx, mosquitto**) megfelelő jogosultságokat kell beállítani, hogy a használatra odaadott mappákhoz a szükséges mértékben hozzá tudjanak férni

```
# Specifikus tulajdonos beállítása a Mosquitto konténer számára (UID 1883)
sudo chown -R 1883:1883 /home/fio/server/mosquitto/data
sudo chown -R 1883:1883 /home/fio/server/mosquitto/log
sudo chown 1883:1883 /home/fio/server/mosquitto/config/mosquitto.conf

# Megfelelő hozzáférési szintek beállítása
chmod 755 /home/fio/server/www
chmod 644 /home/fio/server/mosquitto/config/mosquitto.conf
```

- **Megjegyzés:** A webes tartalom helye a gazdagépen a **/home/fio/server/www** mappa. Ezt a **docker compose** kötet-csatolása rendeli hozzá a konténer belső **webroot** könyvtárához, lásd lentebb, a **docker-compose.yaml** állományban

Docker Compose összeállítása

- A **/home/fio/server/docker-compose.yml** fájl vezérli, hogy melyik Docker image-et indítsuk, ezek milyen mappákat lássanak és milyen portokon keresztül érjük el őket
- Fix verziószámokat használtunk az átláthatóság érdekében:
  - Nginx: **nginx:1.25-alpine**
  - Mosquitto: **eclipse-mosquitto:2.0.18**
- Kötések (Volumes):
  - Az Nginx belső HTML mappáját a **/home/fio/server/www** mappára irányítottuk
  - A Mosquitto konfigurációs fájlját és adatmappáit (**config, data, log**) pedig a **/home/fio/server/mosquitto/** mappába csatoltuk be.

docker-compose.yml

```
sudo tee /home/fio/server/docker-compose.yml <<EOF
services:
  web-server:
    image: nginx:1.25-alpine
    container_name: web-server
    restart: always
    ports:
      - "80:80"
    volumes:
      - /home/fio/server/www:/usr/share/nginx/html:ro
  mqtt-broker:
    image: eclipse-mosquitto:2.0.18
    container_name: mqtt-broker
    restart: always
    ports:
      - "1883:1883"
    volumes:
      - /home/fio/server/mosquitto/config/mosquitto.conf:/mosquitto/config/mosquitto.conf
      - /home/fio/server/mosquitto/data:/mosquitto/data
      - /home/fio/server/mosquitto/log:/mosquitto/log
EOF
```

Automatikus indítás (Systemd Service)

- Hogy bootolás után magától elinduljon a szerver, létrehoztunk egy egyedi szolgáltatást, amely garantálja a szerver indulását bootolás után.
  - Fájl: **/etc/systemd/system/my-iot-server.service**
  - Főbb jellemzők:
    - 15 mp várakozás, hogy a hálózat és a Docker daemon stabilizálódjon.
    - Automatikus `docker compose up -d` futtatás a megfelelő munkakönyvtárban
    - Aktiválás: `sudo systemctl enable my-iot-server.service`
- **A saját szervizfájl létrehozása:** lásd a következő oldalon!
- **A szerviz engedélyezése (Enable):** Ez hozza létre a szimbolikus linket a **/etc/systemd/system/multi-user.target.wants/** könyvtárba. Ez mondja meg a Linuxnak, hogy minden bootoláskor indítsa el a fájlt

```
sudo systemctl enable my-iot-server.service
```

- **Szerviz aktiválása:** Ha módosítjuk a fájlt, a systemd nem veszi észre automatikusan

```
sudo systemctl daemon-reload
sudo systemctl enable my-iot-server.service
```

my-iot-server.service

```
sudo tee /etc/systemd/system/my-iot-server.service <<EOF
[Unit]
Description=My Custom IoT Server (Docker Compose)
After=docker.service network-online.target
Requires=docker.service network-online.target
StartLimitIntervalSec=0

[Service]
Type=oneshot
RemainAfterExit=yes
WorkingDirectory=/home/fio/server
ExecStartPre=/usr/bin/sleep 20
ExecStart=/usr/bin/docker compose up -d
ExecStop=/usr/bin/docker compose down
Restart=on-failure
RestartSec=10

[Install]
WantedBy=multi-user.target
EOF
```

- Ensure Docker and Network are fully operational before starting

- Prevent the service from giving up if the Docker daemon stutters

- Using oneshot + RemainAfterExit for background Docker containers
- Wait for potential system background tasks (like Portenta OS updates) to settle
- Start the containers in detached mode
- Clean shutdown when the service is stopped
-Automatic recovery if startup command fails

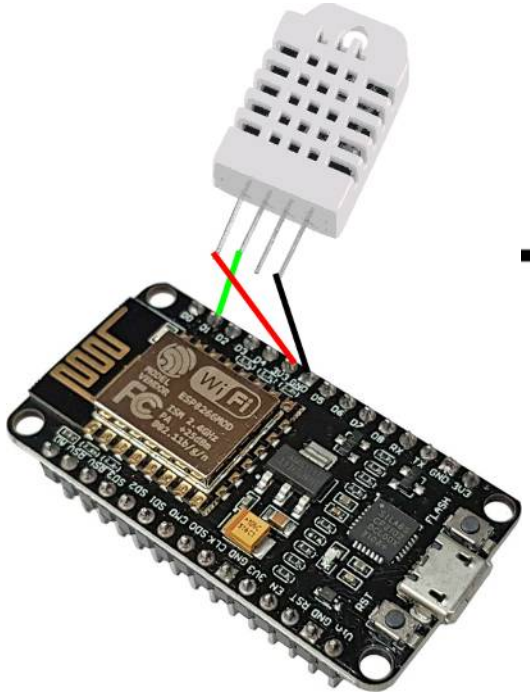
- Start automatically in multi-user runlevel

- Konténerek első elindítása és teszt (az első indítás letölti a Docker image-eket is):

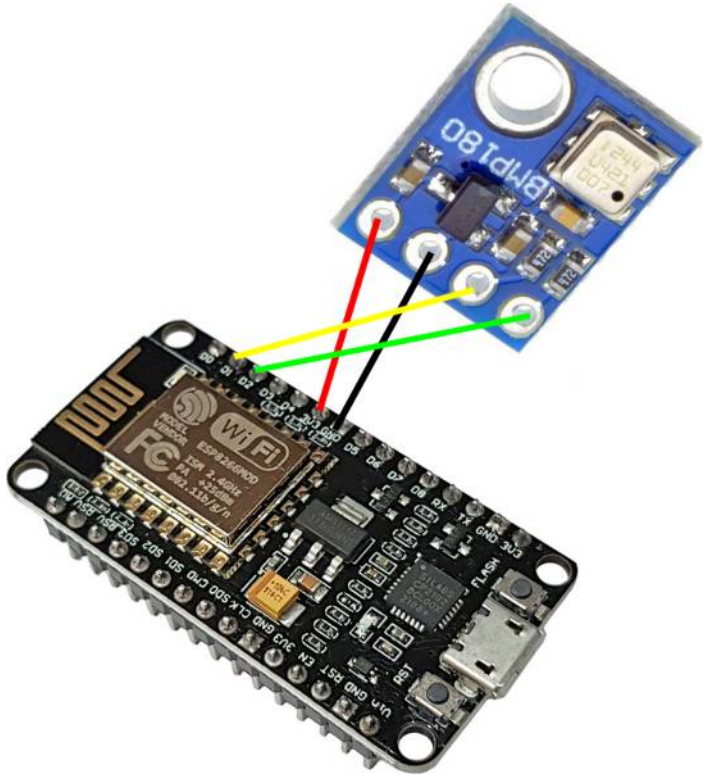
```
cd /home/fio/server
sudo docker compose up -d
```

- Ellenőrzés: `sudo docker ps`

TASMOTA szenzorok konfigurálása

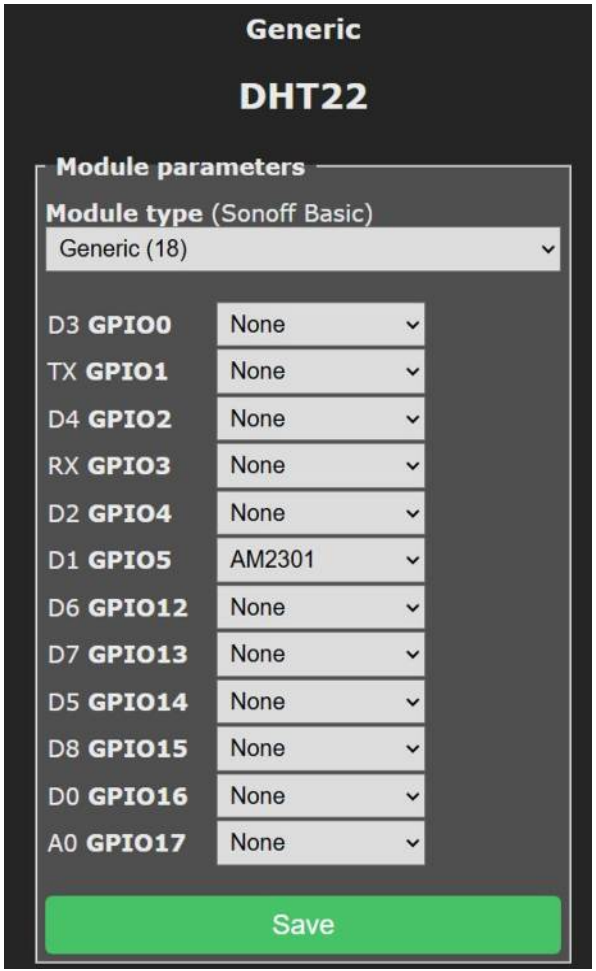


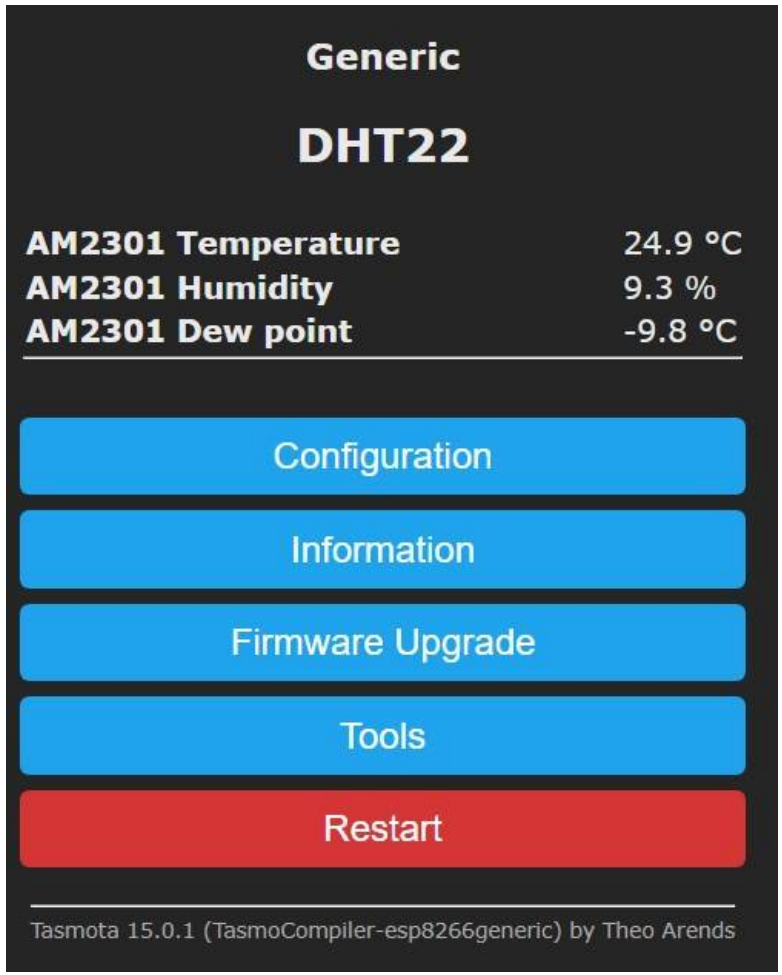
TASMOTA

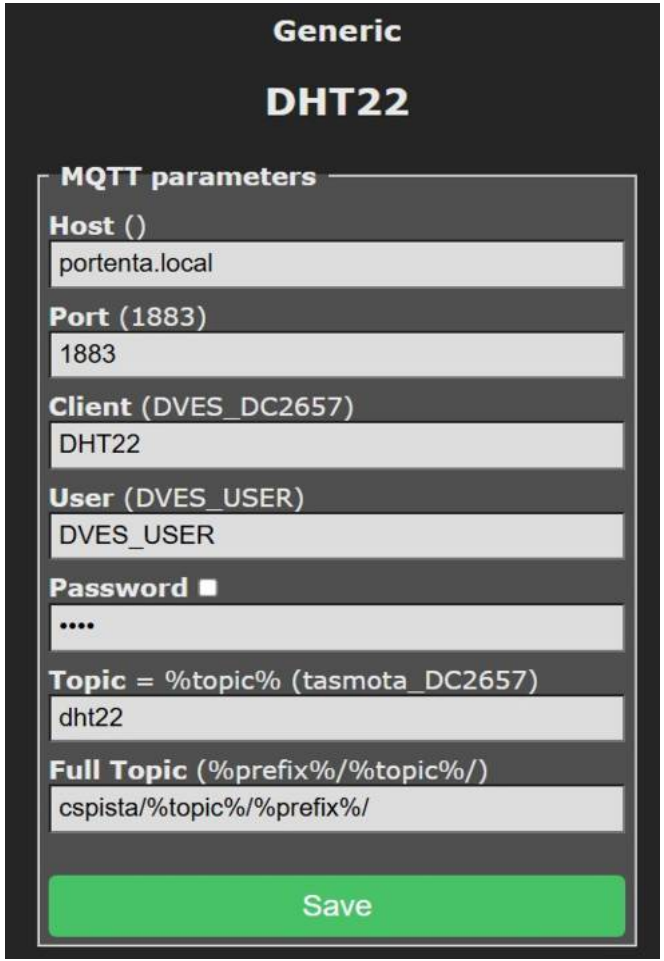


Mivel az **ESP8266**-hoz egyik előre fordított Tasmota-firmware sem felelt meg a céljainknak (DHT22 és BMP180 szenzor + mDNS névfeloldás), saját **tasmota_custom.bin** firmware-t fordítottunk a **TasmoCompilerrel,** és az előadás mellékletében is közzétettük

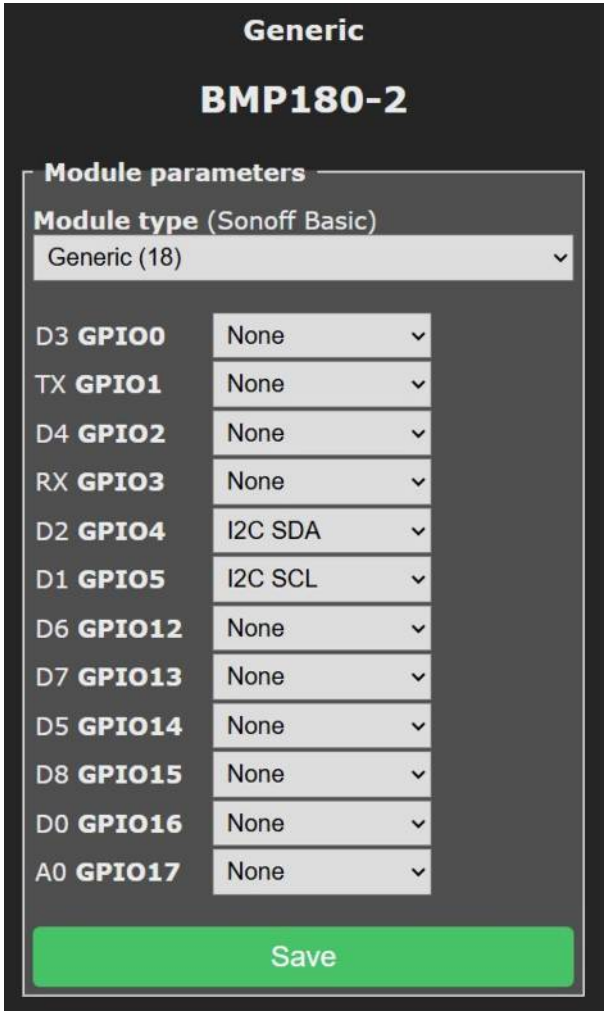
ESP8266 + DHT22 + tasmota_custom.bin

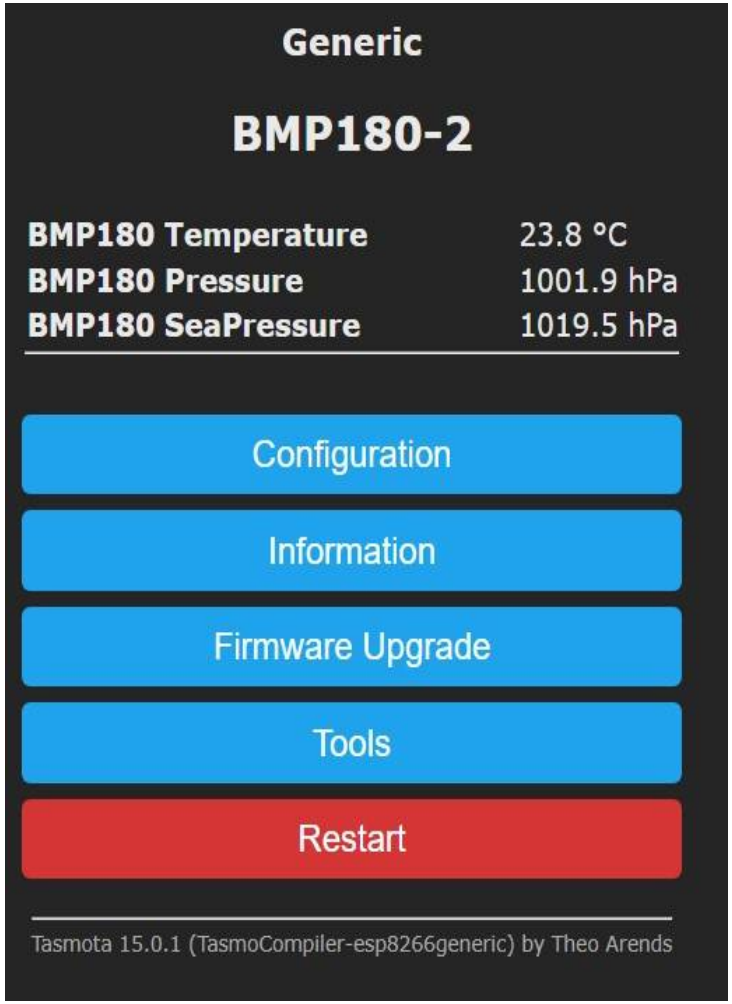


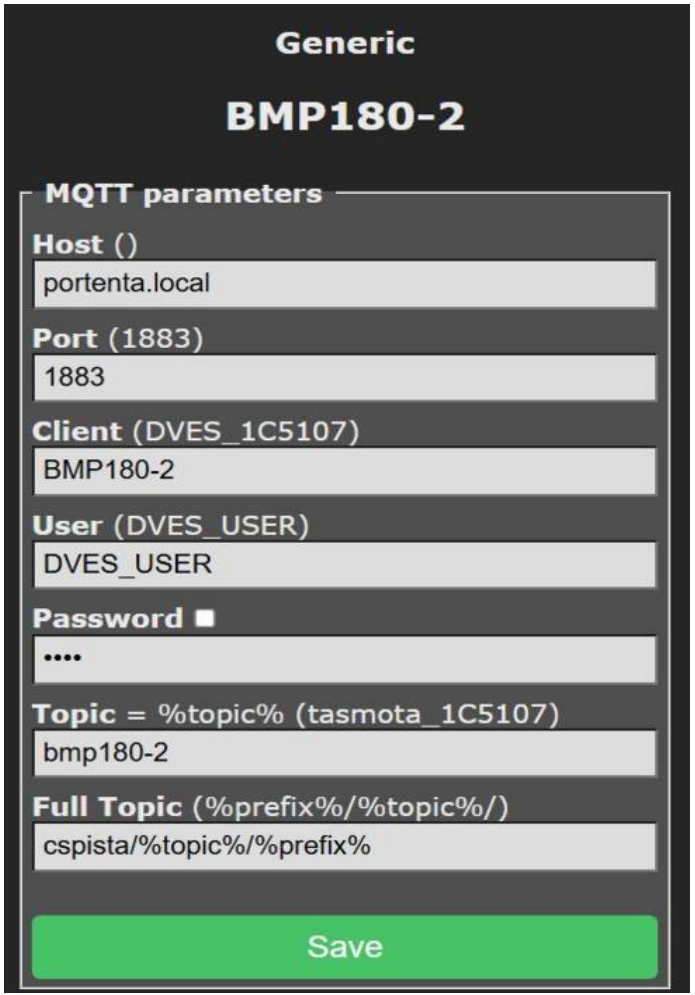




ESP8266 + BMP180 + tasmota_custom.bin







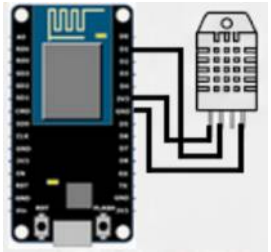
További konfigurálás Console parancsokkal

- A **Tasmota** menürendszere csak a leggyakoribb alapbeállításokat tartalmazza a letisztultság érdekében. Sok olyan speciális paraméter létezik, amelyek neállításához a **Console ablak** az egyetlen út
- A parancs beírása és az Enter lenyomása után a beállítás azonnal életbe lép és megőrződik, így egy esetleges áramszünet vagy újraindítás után is megmarad
- A két szenzorunknál az alábbi parancsokat kell a konzol ablakban kiadnunk (ezt követően **dht22.local** és **bmp180b.local** néven is elérhetjük őket):

ESP8266 + DHT22	ESP8266 + BMP180	A beállítás hatása
`Hostname dht22`	`Hostname bmp180b`	Az mDNS név beállítása
`SetOption55 1`	`SetOption55 1`	mDNS névhirdetés elindítása
`- -`	`Sealevel 139`	Tengerszintfeletti magasság
`Teleperiod 60`	`Teleperiod 60`	Telemetria 60 s gyakorisággal

Tartalék WiFi hálózat konfigurálása

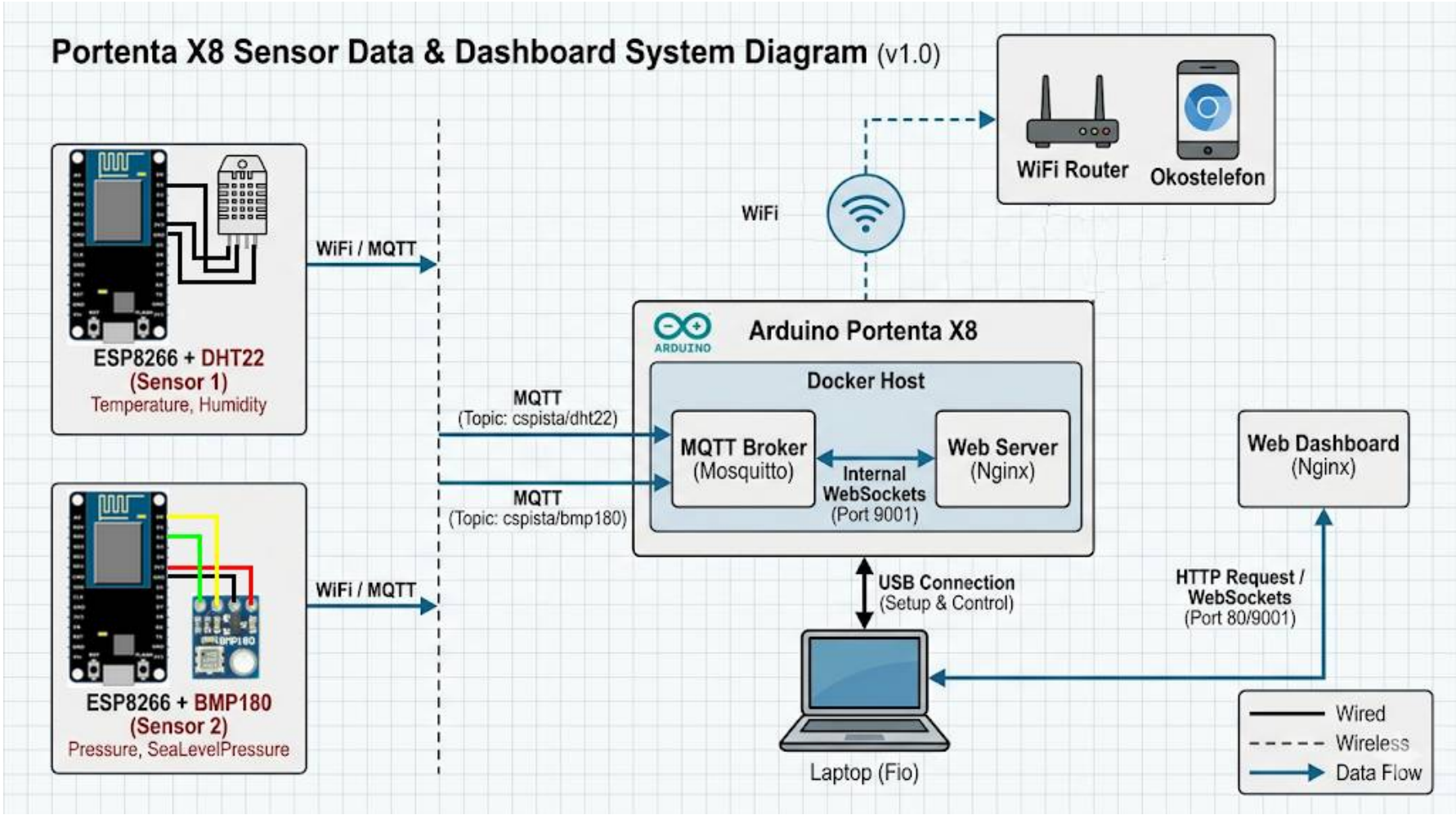
- Ha az otthoni router helyett néha egy mobil hotspotra szeretnénk csatlakozni (pl. bemutatóhoz) akkor kihasználhatjuk, hogy a Tasmota szenzorokon egy tartalék hálózatot is konfigurálhatunk



Mobil hotspot

Otthoni router

Portenta X8 Okosotthon Dashboard



Okosotthon Dashboard – gyorsan, egyszerűen

- **Tasmota** szenzorokkal hőmérsékletet, páratartalmat és légnyomást mérünk egy, vagy több helyen. A **Portenta X8** kártyán fut a helyi **Mosquitto** MQTT bróker és az **Nginx** webszerver
- A legegyszerűbb Dashboard megoldás kliensoldali, JavaScript-alapú, a **Paho MQTT** és a **Chart.js** programkönyvtárakkal. Így nem kell szerveroldali programozással bajlódni: az **Nginx** kiszolgál egy statikus HTML fájlt, ami behúzza a **Paho MQTT** könyvtárat, feliratkozik a Tasmota topikokra, és frissíti a kijelzőt
  A böngésző a WebSockets-en keresztül kommunikál a Mosquitto-val (MQTT)
- Mivel hordozható és zárt (offline) rendszert építünk, nem hivatkozhatunk külső internetes címekre. Minden használni kívánt programkönyvtárat és fájlt előzetesen **le kell tölteni** a webszerver **css/** és **js/** mappáiba

Mosquitto konfiguráció (mosquitto.conf)

- A böngészőből történő adateléréshez engedélyezni kell a **WebSockets** protokollt és a névtelen hozzáférést a `/home/fio/server/mosquitto/config/mosquitto.conf` fájlban

```
persistence true
persistence_location /mosquitto/data/
log_dest stdout

# Normál MQTT (Szenzoroknak, Explorernek)
listener 1883
allow_anonymous true

# WebSockets (A böngészőben futó Dashboardnak)
listener 9001
protocol websockets
allow_anonymous true
```

Ezekkel a sorokkal bővítjük
A konfigurációs fájlt

docker-compose.yml módosítása

```
services:
  web-server:
    image: nginx:1.25-alpine
    container_name: web-server
    restart: always
    ports:
      - "80:80"
    volumes:
      - /home/fio/server/www:/usr/share/nginx/html:ro
  mqtt-broker:
    image: eclipse-mosquitto:2.0.18
    container_name: mqtt-broker
    restart: always
    Ports:
      - "1883:1883" # Normál MQTT
      - "9001:9001" # WebSockets a Dashboardnak
    volumes:
      - /home/fio/server/mosquitto/config/mosquitto.conf:/mosquitto/config/mosquitto.conf
      - /home/fio/server/mosquitto/data:/mosquitto/data
      - /home/fio/server/mosquitto/log:/mosquitto/log
```

Ebben a fájlban csak a 9001-es port megnyitása új fejlemény (a websocket csatorna megnyitásához)

Ezt a sort adjuk hozzá

Fájlok letöltése (wget)

- A Dashboard komponenseit a **/home/fio/server/tasmota** mappában így helyezzük el:
  - **index.html (A fő kód)**
  - **css/ mappa (bootstrap.min.css)**
  - **js/ mappa (mqttws31.min.js, chart.umd.min.js, jquery.min.js)**
- A **/home/fio/server/tasmota** mappában állva a letöltéseket így végezzük:

```
# Az almappák létrehozása
mkdir -p js css

# Bootstrap a kinézethez (opcionális, de a boxok stílusához hasznos)
wget https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css -O css/bootstrap.min.css

# Paho MQTT kliens a kommunikációhoz
wget https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.1/mqttws31.min.js -O js/mqttws31.min.js

# Chart.js UMD verzió a grafikonhoz
wget https://cdn.jsdelivr.net/npm/chart.js@4.4.1/dist/chart.umd.min.js -O js/chart.umd.min.js

# jQuery a DOM manipulációhoz és segédfunkciókhoz
wget https://code.jquery.com/jquery-3.7.1.min.js -O js/jquery.min.js
```

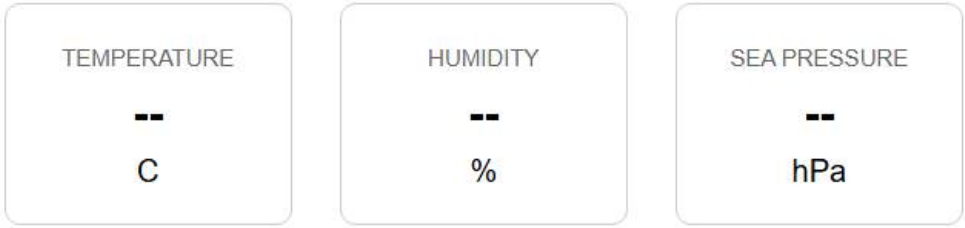
index.html – 5/1.

- A következő oldalakon bemutatott **index.html** valósítja meg a reszponzív kezelőfelületet, ill. az adatok fogadásához és grafikus megjelenítéséhez szükséges kliensoldali logikát

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Portenta Dashboard</title>
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <script src="js/mqttws31.min.js"></script>
    <script src="js/chart.umd.min.js"></script>
    <link rel="icon" href="data:image/svg+xml,<svg xmlns='http://www.w3.org/2000/svg'
          viewBox='0 0 100 100'><circle cx='50' cy='50' r='40' fill='red'/></svg>">
    <style>
        body { font-family: sans-serif; padding: 20px; background-color: #ffffff; color: #000000; }
        .box { border: 1px solid #ccc; padding: 15px; margin: 10px; display: inline-block;
               min-width: 150px; border-radius: 8px; text-align: center; }
        .val { font-size: 24px; font-weight: bold; display: block; }
        .label { font-size: 12px; color: #666; text-transform: uppercase; }
    </style>
</head>
<body>
```

index.html – 5/2.

Az adatkijelző kártyák létrehozása




```
    <div class="box">
        <span class="label">Temperature</span>
        <span id="temp" class="val">--</span> C
    </div>
    <div class="box">
        <span class="label">Humidity</span>
        <span id="hum" class="val">--</span> %
    </div>
    <div class="box">
        <span class="label">Sea Pressure</span>
        <span id="pres" class="val">--</span> hPa
    </div>

    <div style="width: 100%; max-width: 800px; margin-top: 20px;">
        <canvas id="myChart"></canvas>
    </div>
```

```
<script>
    const MQTT_HOST = window.location.hostname;     // Az MQTT szerver ugyanaz, mint a webszerver
    const MQTT_PORT = 9001;
    const CLIENT_ID = "dash_" + Math.random().toString(16).substr(2, 5);
    const client = new Paho.MQTT.Client(MQTT_HOST, MQTT_PORT, CLIENT_ID);
    const ctx = document.getElementById('myChart').getContext('2d');
    const chart = new Chart(ctx, {                  // Egy grafikont is létrehozunk
        type: 'line',
        data: {
            labels: [],
            datasets: [{
                label: 'Temp C',
                data: [],
                borderColor: 'red',
                tension: 0.2,
                fill: false
            }]
        },
        options: {
            animation: false,
            scales: {
                y: { beginAtZero: false }
            }
        }
    });
```

index.html – 5/3.

index.html – 5/4.

```
function onMessage(msg) {
    try {
        const data = JSON.parse(msg.payloadString);          // Az "adatbányászat" (JSON parse)
        const now = new Date().toLocaleTimeString();
        if (data.AM2301) {                                   // Ha DHT22 szenzor mérés történt
            const t = data.AM2301.Temperature;
            const h = data.AM2301.Humidity;
            document.getElementById('temp').innerText = t; // A mérési adatok frissítése
            document.getElementById('hum').innerText = h;
            chart.data.labels.push(now);                     // A grafikon (Chart.js) „etetése”
            chart.data.datasets[0].data.push(t);
            if(chart.data.labels.length > 20) {              // Csak a legutolsó 20 adat látszik
                chart.data.labels.shift();
                chart.data.datasets[0].data.shift();
            }
            chart.update();
        }
        if (data.BMP180) {                                   // Ha BMP180 szenzor mérés történt
            document.getElementById('pres').innerText = data.BMP180.SeaPressure;
        }
    } catch(e) { console.log("Error"); }
}
```

index.html – 5/5.

```
    function connect() {
        client.connect({
            onSuccess: () => {
                client.subscribe("cspista/+/tele/SENSOR");
            },
            onFailure: () => {
                setTimeout(connect, 2000);
            },
            cleanSession: true,
            keepAliveInterval: 30
        });
    }
    client.onMessageArrived = onMessage;
    client.onConnectionLost = () => { setTimeout(connect, 2000); };
    connect();
</script>
</body>
</html>
```

Feliratkozunk a
`cspista/dht22/tele/SENSOR` és a
`cspista/bmp180-2/tele/SENSOR`
témakörökre

A Dashboard „munka közben”