



Universitatea Tehnica “Gheorghe Asachi” Iasi

Facultatea de Inginerie Electrica, Energetica si Informatica Aplicata

IoT Cloud

Documentația Licenței

Profesor Îndrumător:

PROF.UNIV.DR.ING. Cristian Zet

Student:

Ghervasă Cristian

Iasi – 2025

Cuprins

Capitolul 1. Introducere.....	3
Context și motivație.....	3
Obiectivele lucrării	3
Structura Lucrării.....	4
Capitolul 2. Arhitectura Sistemului Teoretic.....	5
2.1. Hardware	5
2.2. Software.....	19
2.3. Docker	31
Capitolul 3. CI/CD – Arhitectura platformei IoT	33
3.1. CI/CD Docker și Unraid	34
Capitolul 4. Implementarea practică.....	40
4.1. Pregătirea hardware-ului în operarea Unraid.....	40
4.2. Setări Unraid, Paritate, GUI și Magazinul de aplicații	43
4.3. Integrarea protocolului MQTT/RPC în ESP32 și MSP430.....	67
Capitolul 5. Evaluare și concluzii.....	87
5.1. Evaluarea performanței și a securității	87
5.2. Limitări și probleme	89
5.3. Concluzii și direcții viitoare. VMWare ESXi, vSphere, Cloud Director, SDDA	91
Bibliografie.....	93

Capitolul 1. Introducere

Context și motivație

Trăim, fără îndoială, într-o lume modernă, iar importanța viitorului electric este deja o realitate palpabilă. Ca ingineri în domeniul electric, avem responsabilitatea de a familiariza și pe cei din jurul nostru cu aceste concepte, chiar dacă nu e vorba de o înțelegere tehnică aprofundată, ci mai degrabă de evidențierea facilităților pe care tehnologia actuală ni le poate oferi.

Auzim tot mai des termeni precum Smart House, Smart Watch, Smart Phone, Smart Sensors, practic, totul pare să fie „Smart” în zilele noastre. Dar ce înseamnă cu adevărat acest concept de „Smart”?

La bază, toate aceste dispozitive sunt pur și simplu aparate electrice, programate să execute anumite acțiuni la comanda utilizatorului și să permită o configurare personalizată. O altă caracteristică esențială a conceptului „Smart” este posibilitatea de a extinde universul pe care alegem să-l construim. Un exemplu modern este ecosistemul Apple: este mult mai convenabil atunci când, pe lângă un iPhone, ai și un Apple Watch, o tabletă Apple și alte produse ale aceleiași mărci. Trebuie să înțelegem că utilizatorul final nu cumpără doar un produs, ci o experiență, o metodă simplă de a conecta diverse dispozitive și de a le folosi după bunul plac.

Din perspectivă inginerească, proiectarea unor astfel de sisteme este nu doar posibilă, ci și o provocare stimulativă. Acum că știm ce își doresc oamenii, acest lucru fiind experiența utilizatorului, ne rămâne partea cea mai plăcută, aceasta fiind înțelegerea și proiectarea unui produs, punând în aplicare toate cunoștințele fundamentale dobândite în acești patru ani de facultate.

Astfel, se naște întrebarea fundamentală: Ce ne dorim să construim? Ce experiență vom oferi? La ce ne va fi utilă creația noastră?

Obiectivele lucrării

Vastitatea domeniilor tehnice m-a făcut să-mi pun o întrebare fundamentală: cum ar arăta dacă am putea reuni toate aspectele ingineriei într-un singur loc? Așa a luat naștere ideea de IoT Cloud. Această dorință se materializează prin nevoia de a uni diverse discipline. Indiferent dacă suntem utilizatori sau constructori ai acestei tehnologii, avem nevoie ca lucrurile să fie ordonate, ușor de înțeles, accesibile și simplu de utilizat. Ne dorim o experiență plăcută, iar ca ingineri, avem nevoie de toate instrumentele la dispoziție și să gândim experiența finală a utilizatorului. Scopul final al acestei lucrări este de a îmbunătăți calitatea vieții persoanelor mai puțin familiarizate cu tehnologia, de a integra noi arhitecturi și de a deschide posibilități nelimitate de integrare cu tehnologiile viitoare.

IoT Cloud trebuie să fie un spațiu special, accesibil de oriunde, pe care să-l putem controla în totalitate. Aici vom putea instala aplicații, achiziționa și prelucra date, având în vedere necesitatea stocării acestora și prevenirea dezastrelor. De asemenea, vom explora cum se poate lucra în echipă, aplicând modele precum Waterfall sau Agile dar și cum arată implementarea inteligenței artificiale (AI).

Nu am fost niciodată mai tehnologizați decât suntem acum. Putem programa cu ușurință o multitudine de dispozitive și microcontrolere concepute pentru a fi programate, beneficiind de o documentație la fel

de accesibilă. Pe piață există deja dispozitive precum Arduino, ESP32, Raspberry Pi, MSP430, alături de o varietate de senzori, pentru umiditate, temperatură, zgomot, distanță, sunet, prezență, senzori optici sau de presiune. Acestea, în esență, transformă un semnal real într-un semnal electric gata de interpretare.

Sunt multe subiecte, extrem de diverse. Pe parcursul acestei lucrări, urmează să explorăm cum putem contrui propriul nostru ecosistem de la zero, cum îl putem expanda atât pe orizontală (Hardware) cât și pe verticală (Software) și cum îl putem face accesibil și pentru publicul larg.

Structura Lucrării

În această lucrare, conținutul a fost organizat pe cinci capitole principale, astfel încât să urmeze un parcurs de la bază către implementare și evaluare:

1. Capitolul 1 – Introducere

Pornim cu motivarea alegerii temei și definirea clară a obiectivelor. Aici descriem contextul general al IoT-ului.

2. Capitolul 2 – Arhitectura sistemului – componente hardware și software disponibile

În al doilea capitol, definim componentele hardware și software necesare. Partea de hardware acoperă server-ul x86_64, microcontrolerele (ESP32, MSP430), senzorii IoT și protocolele de comunicație fizică și digitală. Partea de software prezintă hypervisori (Unraid, VMware ESXi), strategiile de stocare și de recuperare din erori fatale, soluții de rețelistică (VPN, DDNS, partajare fișiere) și introducerea în containerele Docker.

3. Capitolul 3 – CI/CD: Arhitectura platformei IoT

Aici detaliem cum se aplică principiile de CI/CD în contextul IoT-ului: fluxul de date MQTT → ThingsBoard → PostgreSQL, implementarea serviciilor în container Docker și integrarea uneltelor de management a proiectelor și versiunilor (GitLab, Jenkins, JIRA, Nextcloud etc.). Încheiem cu modul în care un ERP (Odoo) și modulele de inteligență artificială completează ecosistemul.

4. Capitolul 4 – Implementarea practică

În acest capitol descriem partea practică, de la scrierea și bootarea USB-ului cu Unraid, configurarea server-ului și a containerelor, până la adăugarea și vizualizarea datelor de la dispozitivele IoT (crearea dashboard-urilor în ThingsBoard) și integrarea efectivă a microcontrolerelor (codurile sursă pentru ESP32 și MSP430).

5. Capitolul 5 – Evaluare și concluzii

În final, analizăm performanța și securitatea soluției implementate, discutăm limitările întâmpinate pe toate nivelurile (hardware, rețea, software) și propunem direcții de îmbunătățire, inclusiv migrarea spre platforme enterprise (VMware vSphere, Cloud Director) și extinderea infrastructurii cu noi tehnologii.

Capitolul 2. Arhitectura Sistemului Teoretic

2.1. Hardware

2.1.1. Componentele Server x86_64

Pentru ca IoT Cloud să poată susține eficient multiple procese paralele, cea mai bună abordare este utilizarea unui hardware dedicat.

Când vine vorba de performanță, există componente destinate atât uzului personal, cât și celui profesional. În acest context, cea mai bună alegere este să ne orientăm către hardware pentru servere. Chiar dacă aceste componente sunt cu mult mai scumpe decât cele pentru uz personal și pot avea performanțe marginale inferioare în anumite aspecte, ele sunt cu mult superioare pentru scenariile unde scopul principal este integrarea și rularea în paralel a diverse tehnologii. Un server este fără îndoială, alegerea cea mai potrivită pentru un astfel de proiect.

Nu este necesar să optăm pentru cea mai recentă tehnologie, deoarece dorim să menținem costurile sub control. O configurație eficientă și accesibilă ar fi achiziționarea unui workstation pre-construit, dar nepopulat cu componente. Pe piață, există producători de workstation-uri, cum ar fi DELL, Lenovo și HP. Din punct de vedere tehnic, cea mai avantajoasă opțiune este una care permite instalarea a minimum două procesoare. Această configurație ne asigură atât o securitate pe termen lung în fața evoluției tehnologice, cât și o performanță excelentă datorită paralelizării. În ceea ce privește alegerea procesorului, Intel a produs numeroase modele pentru această direcție. Datorită ofertei abundente, procesoarele din seria E5v3 și E5v4, se găsesc la prețuri extrem de avantajoase pe piața second-hand. Specificațiile tehnice variază între 10 și 20 de nucleu, 20 și 40 de fire de execuție, 20 și 45MB L3 Cache, iar prețurile sunt între 100 și 500 Ron pe piața second-hand.

Memoria RAM utilizată în servere este, cel mai adesea, de tip ECC (Error Code Correction). Chiar dacă frecvența RAM-ului ECC nu este cea mai mare, atât placă de bază, cât și procesorul le utilizează în mod Quad Channel (spre deosebire de Dual Channel, comun pe computerele personale), ceea ce le face de patru ori mai rapide. O cantitate minimă recomandată este de 64GB RAM, care se va dovedi extrem de utilă în suportarea multiplelor aplicații.

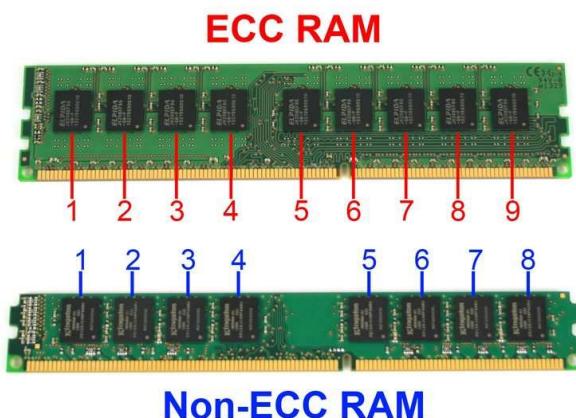


Figura 1. RAM vs ECC RAM

În ceea ce privește mediile de stocare, aspectele sunt puțin mai sensibile. Întrucât sistemul trebuie să fie redundant, iar datele și aplicațiile trebuie să ruleze 24/7 cu un uptime de 99.9%, un singur dispozitiv de stocare nu este suficient. În cazul unei defecțiuni, totul ar fi pierdut. Prin urmare, sunt necesare strategii de tip Disaster Recovery. Pentru o stocare locală eficientă, rapidă și sigură, avem nevoie de mai multe dispozitive de stocare, fie că sunt de tip SSD sau HDD, implementate într-o configurație care să asigure redundanță.

Componentă	Minim	Normal (Home/Small Business)	Recomandat (Prosuser/Mid-size)	Enterprise (Business/Redundant)
CPU	min. 4 nuclee fizice/8 fire de execuție	min. 6 nuclee fizice/12 fire de execuție	min. 8 nuclee fizice/16 fire de execuție	min. 16 nuclee fizice/32 fire de execuție
GPU	Nu este Necessar	Integrată sau dedicată	Dedicată	Dedicată mediu spre profesională
RAM	8GB (Single Channel)	16GB (Dual Channel)	64GB ECC (Dual/Quad Channel)	128GB+ ECC (Quad/Hexa Channel)
Stocare	1x SSD 250GB+ 2x HDD 2TB+	1x SSD NVMe 500GB+; 3x HDD 4TB+	2x SSD NVMe 1TB+ (RAID 1); 4x HDD 8TB+ (RAID 10/ZFS)	4x SSD NVMe 2TB+ (RAID 10); 6x+ HDD (date) 12TB+ (RAID 6/ZFS);
Placa De Rețea	1x Fast/Gigabit Ethernet	2x Gigabit Ethernet	2x Gigabit Ethernet / sau 1x 10 Gigabit Ethernet	4x 1 Gigabit Ethernet 2x 10 Gigabit Ethernet
Capabilitate Docker-e	5-10	15-25	30-50	100+
Capabilitate VM-uri	1-2 VM-uri	2-3 VM-uri	3-5 VM-uri	5-10+ VM-uri (VDI)
Utilizatori	1-3 utilizatori simultan	3-10 utilizatori simultan	10-30+ utilizatori simultan	50+ utilizatori
Scenarii de Utilizare	IoT Cloud pentru testare	IoT Cloud și automatizări	Server IoT cu Inteligență artificială	Infrastructura Virtuală, platforme de dezvoltare CI/CD, SDDA, Disaster Recovery

Tabelul 1. Componentele unui server

CI (Continuous Integration) – Integrare continuă

Integrarea continua face referinta la un proces automatizat prin care modificările de cod sunt integrate frecvent într-un repository (spatiu) comun. Fiecare integrare declanșează automat compilarea și testarea codului, pentru a detecta rapid erorile.

CD (Continuous Delivery) – Livrare continuă

Continuous Delivery: Este o expensie a integrării continue, automatizând și fazele de pregătire a pachetelor software (build, testare, versionare), astfel încât programul rezultat din urma codului să poată fi lansat în producție în orice moment, cu un singur clic.

VM (Virtual Machine) – Mașină virtuală

Mașina virtuală este o unealtă software care poate imita un calculator fizic complet, inclusiv sistemul de operare și aplicații, dar care rulează într-un mediu complet izolat. Acestea pot permite ca mai multe sisteme de operare diferite să ruleze în paralel pe aceeași mașină fizică.

VDI (Virtual Desktop Infrastructure) – Infrastructură de desktop virtual

Infrastructura virtuală a unui desktop este o tehnologie prin care calculatoarele utilizatorilor (sistemul de operare, aplicațiile și datele) rulează pe servere și sunt accesate de pe alte stații de lucru ce sunt interconectate în același rețea de internet locală. Astfel, desktop-ul fizic al utilizatorului este „virtualizat”.

SDDC (Software Defined Data Center) – Centru de date definit prin software

Centrul de date definit prin software este o abordare în care toate elementele infrastructurii unui datacenter (servere, stocare, rețea, securitate) sunt virtualizate și automatizate printr-un strat de management software. Astfel se atinge o flexibilitate maximă și o scalare rapidă a resurselor. În prezent 95% din toate centrele de baze de date sunt definite prin software.

2.1.2. Microcontrolere (ESP32, MSP430)

În arhitectura soluției noastre, ThingsBoard (open source IoT platform) servește drept platformă centrală de telemetrie și suportă, pe lângă MQTT și protocolele HTTP, CoAP și WebSocket pentru schimbul de date cu dispozitivele periferice. Microcontrolerul ales în această lucrare ce va funcționa ca și “nod edge” va fi ESP32 întrucât prezintă principalele caracteristici necesare, acestea fiind conexiunea WI-FI, dispune de conversie analog-digitale dar și de posibilitatea utilizării unor pini drept căi de comunicare (UART).

În cazurile în care un microcontroler nu posedă o interfață de comunicație prin WI-FI integrată, transferul datelor se realizează prin magistrale hardware locale, UART, SPI sau I²C. Aceste interfețe permit comunicarea informațiilor într-un gateway sau într-un dispozitiv intermediar, care, ulterior, le transmite prin rețeaua LAN a serverului către ThingsBoard. Astfel se asigură continuitatea și fiabilitatea fluxului de date de la nivelul senzorilor până la platforma de monitorizare și control.

Microcontroler/ Platformă	CPU (Frecvență x Nuclee)	RAM (KB)	Flash (MB)	Conecțivitate	Consum Energie	Pret Mediu (EUR)
Raspberry Pi 3 Model B+	1.4 GHz × 4	1 GB	microSD	Wi-Fi, BLE	1 A	37
ESP32 Espressif	10-240 MHz x 2	520	0.128-4	Wi-Fi, BLE	80-260 mA	3.5
MSP430 TI	1-24 MHz x 1	0.5-66	0.016- 0.128	Extern	230uA	4

Tabelul 2. Compararea microcontrolerelor

Specificațiile pentru Raspberry Pi 3 Model B+

Procesor

- Broadcom BCM2837B0, quad-core ARM Cortex-A53 la 1.4 GHz, arhitectură 64 biți, rulând un kernel Ubuntu LTS

Memorie

- LPDDR2 SDRAM: 1 GB unificat (pentru CPU și GPU)
- Stocare: slot microSD (fără memorie Flash internă)

Periferice de achiziție și control

- GPIO digital: 40 pini (programabili ca intrări/ieșiri)
- SPI: 2 controlere hardware
- I²C: 2 magistrale
- UART: 1 interfață serială primară + PL011 Bluetooth HCI
- I²S: interfață pentru audio digital
- PWM: 2 canale hardware (până la 1,2 kHz practic)
- CSI/DSI: interfațe pentru cameră (CSI) și display (DSI)

Notă: nu are ADC/DAC integrat, astfel este necesar un convertor extern (ex. MCP3008)

Extensii „industriale”

- Modbus TCP: prin Ethernet + biblioteci Python/C

Conecțivitate rețea & wireless

- Ethernet: Gigabit prin interfață USB 2.0 (teoretic până la ~300 Mbps)
- Wi-Fi dual-band: 802.11 b/g/n/ac la 2,4 GHz și 5 GHz (până la ~150 Mbps)
- Bluetooth 4.2 +
- USB 2.0: 4 porturi host

Consum de energie

- Idle (fără periferice active): ~2,8 W (0,56 A @ 5 V)
- Sarcină medie (CPU și Wi-Fi active): ~4 W (0,8 A @ 5 V)
- Sarcină maximă: ~5 W (1 A @ 5 V)

Specificațiile pentru ESP32

Procesor

- Dual-core Tensilica LX6 la 240 MHz, capabil de task-uri parallele (FreeRTOS)

Memorie

- SRAM intern: 520 kB (util pentru buffer-e, stack-uri și heap)
- PSRAM extern (optional): până la 4 MB, util pentru caching sau stocare temporară

Periferice de achiziție și control

- ADC: 18 canale 12-bit (0–3.3 V), cu sample-and-hold și referință internă
- DAC: 2 canale 8-bit, convertire directă D/A cu filtrare hardware internă
- PWM: 16 canale up to 16-bit, frecvență programabilă (ideal pentru drivere de motoare sau LED dimming)
- Interfețe digitale: multiple SPI, I²C, UART, I²S (pentru microfon digital)

Extensiile industriale

- CAN 2.0b: se adaugă cu MCP2515 + TJA1050; ESP32 comunică pe SPI cu MCP2515 și procesează frame-urile CAN la viteze de până la 1 MB/s, aceasta fiind și viteza maximă a protocolului CAN.
- Modbus: biblioteci FreeModbus RTU (UART) și Modbus TCP (Wi-Fi) – ideale pentru conectarea la PLC-uri, contoare de energie și echipamente SCADA

Conecțivitatea rețea & wireless

- Standard: 802.11 b/g/n, DSSS și OFDM, canale de 20/40 MHz
- Throughput: până la ~72 Mbps în modul 20 MHz, ~150 Mbps în 40 MHz
- Stack TCP/IP: lwIP integrat, suport pentru TLS 1.2/1.3 (mbedtlsTLS) și criptare hardware-accelerată pentru ECC (Curve25519), AES (CBC/GCM)

Specificațiile pentru MSP430FR2355

Procesor

- Nucleu MSP430 16-bit RISC la până la 24 MHz

Memorie

- FRAM: 32 kB
- SRAM: 4KB

Periferice de achiziție și control

- ADC: 12 canale SAR, rezoluție 12 bit, referință internă, 0–VCC
- DAC: 4 canale 12 bit cu filtrare hardware internă
- PWM: până la 6 canale generate de 2 timer-e de 16 bit
- OpAmp: 2 amplificatoare operaționale configurabile
- Comparator analogic, senzor temperatură intern

Conecțivitatea rețea & wireless

- Nu are Wi-Fi și nici BLE.
- – 2× UART/IrDA (eUSCI)
- – 2× SPI
- – 2× I²C

Extensii industriale

- Suport software pentru Modbus RTU (UART) și Modbus TCP
- AES-128/256 și CRC hardware

2.1.3. Senzori pentru IoT

În arhitectura soluției propuse, microcontrolerele mentionate în tabelul anterior pot acționa ca și nuclee de achiziție și preprocesare a semnalelor provenite de la o gamă largă de senzori (temperatură, umiditate, presiune, lumină, acceleratie, pH, gaz, nivel lichid etc.). Fiecare senzor furnizează o valoare brută de ADC, pe care microcontrolerul o convertește (analog-digital), compensează și filtrează, obținând astfel informații gata de transmitere.

Datele procesate de microcontroler sunt ambalate în mesaje conforme cu protocolele suportate de ThingsBoard (MQTT, HTTP, CoAP, WebSocket) și transmise prin rețeaua LAN către serverul țintă.

Clasa Senzorilor	Parametrii	Interfață	Gamă	Precizie	Rezoluție	Example
Temperatură	(°C/F)	Analog, I ² C, SPI, 1-Wire	-55...+150 °C	±0.5 °C (DS18B20)±0.25 °C (TMP117)	0.0625 °C...0.0078 °C	DS18B20, LM35, TMP117
Umiditate relativă	%RH	I ² C, SPI, analog	0...100 %RH	±2 %RH (SHT31)±5 %RH (DHT22)	0.01 %RH...0.1 %RH	SHT31, DHT22, HDC1080
Presiune	hPa/bar, Pa	I ² C, SPI, analog	300...1100 hPa0...40 bar	±1 hPa±0.1 %FS	0.18 Pa...1 Pa	BMP280, MPL3115 A2, MS5611
Lumină (lux)	lux	I ² C, analog	0.1...100 000 lux	±3 % (TSL2591)	0.01 lux...1 lux	TSL2561, BH1750, TEMT6000
Accelerometru	Acceleratie (g)	I ² C, SPI	±2 g...±16 g	0.00098 g (ADXL345)	—	ADXL345, MPU6050, LIS3DH
Giroscop	Rotație (°/s)	I ² C, SPI	±250...±2000 °/s	—	0.07 °/s	L3GD20H, MPU6050, MPU9250
Magnetometru (compas)	Câmp magnetic (µT)	I ² C, SPI	±4 gauss...±16 gauss	±1 gauss	0.1 µT	HMC5883 L, IST8310

Clasa Senzorilor	Parametrii	Interfață	Gamă	Precizie	Rezoluție	Example
(IR/Ultrasunete)	Distanță (cm)	Digital, analog, serial	IR: 10...80 cm: 2...400 cm	IR: ±1 cmUS: ±0.3 cm	IR: 1 cmUS: 0.1 cm	Sharp GP2Y0A21YK0F, HC-SR04
Gaz (MQ Series)	Concentrație (ppm)	Analog	10...10 000 ppm	±5 %FS	—	MQ-2, MQ-7, MQ-135
CO ₂	Concentrație CO ₂ (ppm)	UART, I ² C	400...10 000 ppm	±(30 ppm +3 % din măsură)	—	MH-Z19B, SCD30
pH	Nivel pH	Analog	0...14	±0.1 pH	0.01 pH	Atlas Scientific pH, SEN0161
Debit (Flow)	L/min, volum total	Puls digital	1...30 L/min	±5 %	Puls/min	YF-S201, G1/2 Hall effect
Nivel lichid	(cm/%)	Analog, digital (I ² C), hidraulic	0...100 cm	±1 cm	1 cm	US-100, SEN0244
IMU (Accel + Gyro + Mag)	-	I ² C, SPI	Accel: ±2...±16 g Gyro: ±250...±2000 °/s	Accel: 0.001 g Gyro: 0.07 °/s	—	MPU9250
Vibrații (Piezoelectric)	Vibrație (g), frecvență (Hz)	Analog	±0.1...±500 g	±5 %	Depinde de ADC (12–16 bit)	ADXL100 2, piezo + conditioner
Acustic (Microfon)	dB SPL	Analog, I ² S	30...120 dB	±1.5 dB	—	MAX4466, SPH0645L M4H
UV	UV index, intensitate (mW/cm ²)	Analog, I ² C	0...15 UV index	±1 UV index	0.1 UV	VEML6075, GUVA-S12SD
Culoare (RGB)	R/G/B intensity, lux	I ² C	0...65 535 (16-bit)	±5 %	1 unitate	TCS34725, ISL29125

Clasa Senzorilor	Parametrii	Interfață	Gamă	Precizie	Rezoluție	Example
Gaz Avansați (NO₂/O₃/CO/NH₃)	Concentrație gaz (ppb/ppm)	I ² C, UART	NO ₂ : 0...1000 ppbCO: 0...100 ppm	±(5 %FS +1 ppb)	—	Alphasense A4 NO ₂ , SGX O ₃ 10 ppb
Particule fine (PM1.0/2.5/10)	µg/m ³	UART, I ² C	0...1000 µg/m ³	±10 %	—	PMS5003, SDS011

Tabelul 3. Clase de senzori

2.1.4. Protocole de comunicație hardware pentru ecosisteme embedded și industriale

În acest capitol ne concentrăm pe cele mai răspândite și utile protocole de comunicație pe care le putem folosi în firmware-ul ESP32 (sau oricărui microcontroler) pentru a interconecta dispozitive, a transmite comenzi și date, și a extinde funcționalitățile unor echipamente mai vechi. Vom aborda, în profunzime, următoarele protocole:

1. UART (Universal Asynchronous Receiver/Transmitter)
2. I²C (Inter-Integrated Circuit)
3. SPI (Serial Peripheral Interface)
4. CAN (Controller Area Network) și extensiile sale
5. Modbus (RTU și TCP)
6. VISA/SCPI (Virtual Instrument Software Architecture/Standard Commands for Programmable Instruments)

Pentru fiecare protocol vom parcurge:

- Rolul și motivația, de ce îl folosim și când e preferat.
- Stratul fizic și de legătură, felul în care sunt organizate semnalele.
- Formatul de date.
- Viteze și topologii tipice.

UART – denumit și “Textul pe fir”

UART este protocolul serial fundamental pe microcontrolere, folosit pentru:

- Debug prin port serial
- Comunicare simplă între două plăci (ESP32 ↔ MSP430)
- Posibilitatea de a citi pe osciloscop semnalele fizice, translatate în cod ASCII.

Strat fizic și framing

- Linii: TX, RX (optional RTS/CTS pentru flow control)
- Format: 1 start bit, 5–9 date bits, paritate (optional), 1–2 stop bits
- Viteze comune: 9600, 115200, 230400 bps

Formatul de date

- Mesaj ASCII: fără framing complex, terminat de obicei cu \r (return) sau \n (new line).

Extindere: meniuri și SSH

- Putem implementa un “CLI” (Command Line Interface) peste UART cu comenzi text și meniuri, de exemplu posibilitatea de a cere prin SSH „1:System Info 2:Configure 3:Reboot”.
- SSH direct pe ESP32 (mini-OpenSSH) există proiecte experimentale, dar rareori folosit în producție din cauza resurselor limitate.

I²C – magistrala senzorială

I²C este folosit pentru interfațarea cu:

- Senzori digitali (temperatură, presiune, umiditate)
- EEPROM-uri și ceasuri de timp real
- Expandare I/O, convertor ADC externe

Strat fizic și framing

- Linii: SDA (data), SCL (clock), ambele cu pull-up
- Topologie multi-master, multi-slave
- Adrese pe 7 sau 10 biți. Byte de adresă urmat de read/write bit

SPI – high-speed peer-to-peer

SPI oferă viteză mare (frecvențe de zeci de MHz) și e folosit pentru:

- Memorii flash externe
- Convertor rapida ADC/DAC
- Module radio (LoRa, nRF24)

Strat fizic

- Linii: SCLK, MOSI, MISO, CS (chip select) per dispozitiv
- Full-duplex, master/slave

CAN – Controller Area Network

CAN bus este cel mai prezent în următoarele industrii:

- Automotive (ECU-uri)
- Automatică industrială
- Robotică

Strat fizic și framing

- Linii: CAN_H, CAN_L pe twisted pair
- Frame standard: ID (11 biți) sau extins (29 biți), DLC, date (0–8 octeți), CRC, ACK
- Viteze: până la 1 Mbps (CAN-FD extinde până la 8 Mbps pentru data phase)

Extensii moderne

- CAN-FD: frame-uri variabile, date de până la 64 octeți
- XCP (Universal Measurement and Calibration Protocol) peste CAN/CAN-FD, folosește fișiere .a2l pentru maparea memoriei de ECU
- Securitate CAN: autentificare la nivel de frame, criptare industrială

Modbus – raportare și control industrial

Modbus este un protocol simplu, master-slave, folosit pentru a citi/scrive registre pe:

- PLC-uri industriale
- Contoare de energie
- Controlere de temperatură

Variante

- RTU: peste UART, framing binar, CRC16
- ASCII: framing ASCII
- TCP: pe Ethernet, port 502

VISA/SCPI – instrumente de laborator

VISA (NI-VISA) și SCPI definesc un set standard de comenzi pentru instrumente de măsură (osciloscop, multimeter, sursă de tensiune) ce pot comunica prin:

- RS-232, USB-TMC, GPIB, Ethernet, CAN
- Comenzile standard sunt exprimate în ASCII ca *IDN?, MEAS:VOLT?, CONF:CURR 0.5

Astfel prin unificarea protocolelor UART, I²C, SPI, CAN, Modbus sau VISA/SCPI cu ESP32, putem construi un gateway universal unde:

- Putem controla echipamente mai vechi fără Wi-Fi
- Putem crea meniuri, panouri de configurare și CLI pe UART
- Putem integra rețele industriale robuste (CAN, Modbus) cu infrastructura cloud modernă (MQTT, RPC)
- Putem extinde orice dispozitiv electric cu funcționalități remote, fie el senzor, actuator sau instrument de măsură

În final, orice microcontroler sau dispozitiv devine parte a unei rețele inteligente, pregătită pentru Industria 4.0 și arhitecturi IoT-edge-to-cloud.

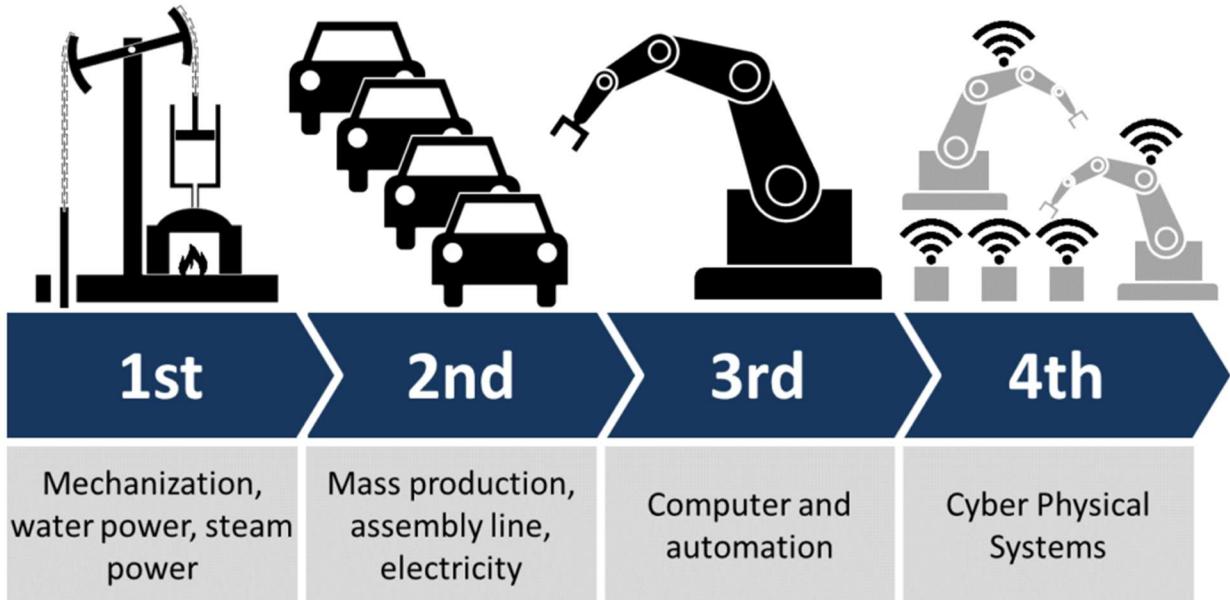


Figura 2. Reprezentarea nivelurilor de industrie

2.1.5. Integrarea claselor de senzori și extinderea capabilităților de achiziție

Până acum am enumerat peste 20 de tipuri de senzori (temperatură, umiditate, presiune, accelerometre, gaz, particule etc.) și protocoalele de comunicație (UART, I²C, SPI, CAN, Modbus, VISA/SCPI) care ne permit să legăm orice dispozitiv la un nod edge ESP32. În acest subcapitol vom explica:

1. **Cum combinăm ADC-urile interne și externe cu multiplexoare** pentru a citi zeci de senzori analogici
2. **Utilizarea magistralelor I²C și SPI** pentru interogarea senzorilor digitali inteligenți
3. **Folosirea CAN și VISA/SCPI** pentru achiziția de la echipamente industriale și de laborator
4. **Arhitectură de sistem**: cum organizăm firmware-ul pentru a gestiona mixul de interfețe și senzori

Multiplexarea canalelor ADC

ESP32 are 18 canale ADC interne, dar liniaritatea și stabilitatea sunt garantate doar pe 6 dintre ele. Dacă vrem să măsurăm zeci de senzori analogici (ex.: MQ-Series, pH, LDR), aflăm două opțiuni:

- **Multiplexor analogic** (ex. CD4051, 8-to-1):
 - Conectăm 8 semnale analogice la intrările MUX, ieșirea comună merge la un pin ADC.
 - Comutăm select lines (S0...S2) prin GPIO, citim ADC, de-multe ori rapid.
- **ADC extern multicanal** (ex. ADS1115 – 4 canale, I²C; MCP3008 – 8 canale, SPI):
 - Folosim I²C sau SPI pentru a citi fiecare canal, cu precizie 12–16 biți.
 - Eliminăm necesitatea multiplexoarelor externe și asigurăm eşantionare sincronă.

Interogarea senzorilor digitali prin I²C și SPI

Majoritatea senzorilor digitali (SHT31, BMP280, TCS34725) oferă interfață I²C sau SPI. Integrarea lor se face astfel:

- **I²C:**
 1. Se inițializează magistrala și rezistențele pull-up (4.7 kΩ)
 2. La o adresă fixă (0x44, 0x76...) se citesc registre de date
 3. Se decodifică MSB/LSB și se aplică formule de compensare
- **SPI:**
 1. Se configerează semnalul de ceas (clock), modul SPI și linia de selecție CS.
 2. În cadrul comunicării master-slave, atunci când CS devine LOW, master-ul trimite comanda de citire
 3. Se recepționează octetii de date și se calculează valorile mărimilor fizice

Achiziție de la echipamente industriale prin CAN și VISA/SCPI

Pe lângă senzorii mici, adesea sunt necesare date de la echipamente profesionale, astfel, un protocol de comunicare extrem de stabil ar următorul:

- **CAN (Controller Area Network)**

Stabil și robust: format din două fire twisted-pair, cu o rezistență de 120 Ohmi la fiecare capăt al liniei de comunicare. Rezistențele se conectează între CAN_High și CAN_Low. Este folosit și azi în mașini sau roboți. Viteza maximă suportată pe CAN-ul clasic 2.0A/B este de 1Mbps.

- **XCP (Universal Measurement and Calibration Protocol)**

Protocolul XCP folosește CAN ca și strat de transport, este standardizat (ASAM) și folosit pentru calibrare, diagnostic și achiziție de date în sisteme embedded. Acesta permite scrierea și citirea în RAM sau Flash în timp real. Un scurt exemplu ar fi calibrarea unui regulator de tip PID în timp ce sistemul se află în funcțiune. Pentru utilizarea acestui protocol este necesar și un fișier A2L. Acesta este un fișier descriptiv unde se regăsesc adresele variabilelor din memorie (simboluri), structura datelor (scalari, conversii) dar și configurații DAQ pentru achiziția de date.

Acest fișier este generat de obicei automat de tool-urile ce pot scrie firmware (de exemplu MATLAB Embedded Coder, Vector).

2.1.6. Protocole software de comunicare (MQTT și RPC)

Ne propunem să dezvoltăm o platformă Cloud IoT complet integrată, care trage la maxim de resursele hardware și software deja discutate. În acest cadru, ESP32-WIFI joacă rolul de intermediar între lumea fizică (senzori, magistrale industriale) și serverul central (containere Docker, broker MQTT, ThingsBoard, baza PostgreSQL).

Astfel platforma devine un mediu în care adăugarea unui nou dispozitiv IoT (ESP32, Raspberry Pi, Desktop) se face cu minim de efort. Toate componentele rulează în containere, unde ThingsBoard este platforma de gestionare și vizualizare. Aceasta prezintă Brokerul MQTT integrat dar și PostgreSQL ce are

ca scop stocarea telemetriei, conturile de acces, baza de date pentru dispozitive și configurațiile aditionale.

Arhitectura Sistemului

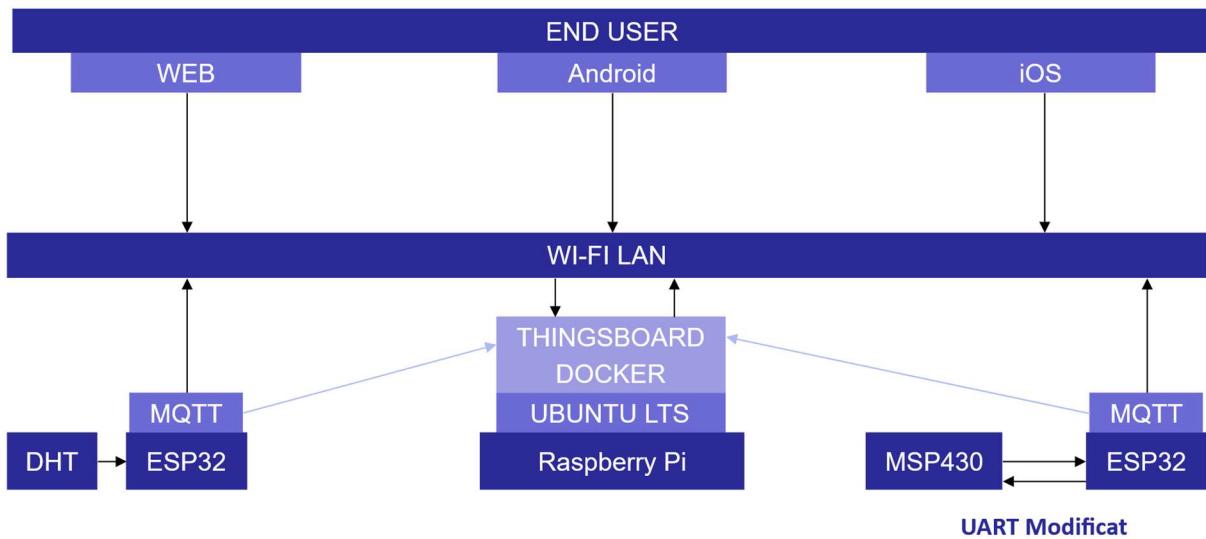


Figura 3. Arhitectura Sistemului

„Man-In-The-Middle” ESP32 și MQTT

ESP32 poate funcționa atât ca “Nod Edge”, ceea ce înseamnă că el efectuează achiziția datelor, prelucrarea și trimiterea cât și posibilitatea de a capta date deja prelucrate de la alte microcontrolere ce nu pot efectua funcția de transmisie. ESP32 poate efectua doar transmiterea lor, devenind astfel omul din mijloc, ce doar ascultă ambele canale de comunicare, server la controler și controler la server.

1. Achiziție

- Preluare de la senzori analogici (DS18B20, SHT31), CAN (E-COM), Modbus (PLC)

2. Filtrare și calcul

- Filtru Kalman simplificat pentru senzorii vibrație
- Compensare termică: ecuație de corecție bazată pe coeficient calibrat
- Generare de statistici (min, max, medie) la fiecare minut

3. Autentificare și securitate

- SSL/TLS over MQTT cu certificate ECDSA stocate în flash pe zona sigură
- Re-reauthenticare automată la expirație

Criteriu	MQTT	HTTP/REST	Tuya Proprietar
Model de comunicatie	Publish/Subscribe	Request/Response	Proprietar SDK/API

Criteriu	MQTT	HTTP/REST	Tuya Proprietar
Eficiență bandă	Format binar compact	Header HTTP masiv	Variabil, criptat
QoS (0/1/2)	Garantat la cerere (QoS 1/2)	Nu există (re-try manual)	Intern, nedeclarat
Suport TLS nativ	Da (TLS 1.2/1.3)	Da	Da
Scalabilitate	Broker dedicat (cluster)	Necesită load balancer	Depinde de furnizor
Complexitate device	MQTT client matur (esp-mqtt)	HTTP client ușor	SDK specific

Tabelul 4. Compararea protocolelor de comunicare

Motivare:

- **Economie de resurse:** ESP32 dedică < 10 % CPU pentru SSL + MQTT, comparativ cu > 30 % pentru HTTPS.
- **Livrare fiabilă:** QoS 1 sau 2 garantează mesajul chiar în rețele ocupate.

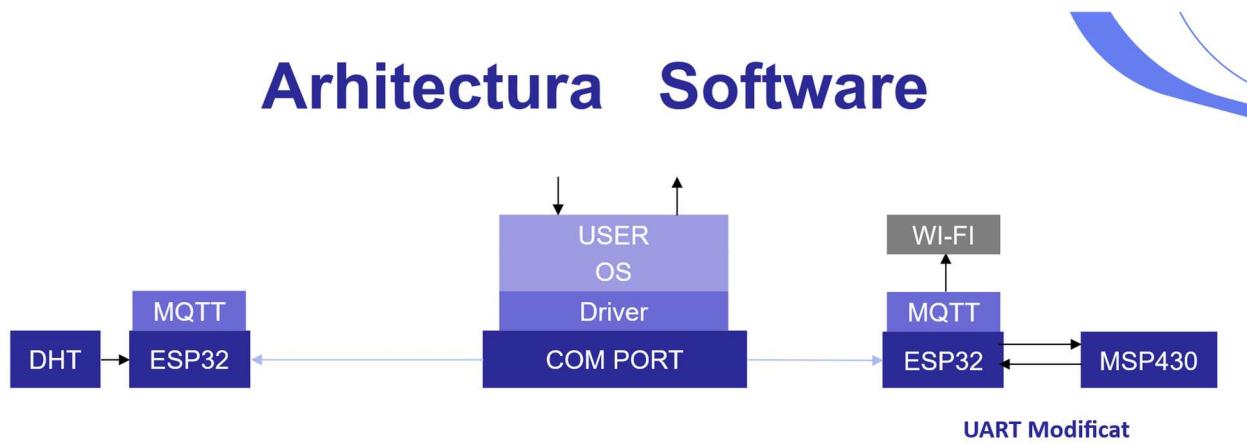


Figura 4. Arhitectura Software

Remote Procedure Call (RPC) Methods – Arhitectură, Mecanisme Detaliate și Capabilități Avansate

Într-un ecosistem distribuit, fie că vorbim de microservicii în cloud, platforme IoT sau aplicații enterprise, RPC (Remote Procedure Call) este mecanismul de bază prin care procese aflate pe dispozitive diferite pot apela funcții unele ale celorlalte la fel de simplu ca și cum ar fi locale. RPC este mecanismul prin care putem:

- Unifica aplicații heterogene

- Ascunde complexitatea rețelei
- Permite evoluția independentă a fiecărui serviciu

Principiile de bază ale RPC

1. Microservicii enterprise: servicii de facturare, user management, notificări, toate apelate prin RPC
2. IoT Edge: ESP32 rulează un client RPC peste MQTT, trimițând comenzi de configurare la distanță

Model	Descriere
Unary	Un singur request urmat de un singur response (ex. JSON-RPC, XML-RPC, gRPC unary)
Server Streaming	Client trimit o cerere iar serverul răspunde cu un flux continuu de mesaje până la încheiere
Client Streaming	Client trimit mai multe mesaje iar serverul răspunde o dată la final
Bidirectional	Flux duplex: client și server pot trimit mesaje independent, concomitent (gRPC bidi-stream)

Tabelul 5. Metode de apelare RPC

2.2. Software

2.2.1. Hypervisor

Alegerea Mediului de Execuție Server-Side

În dezvoltarea și operarea unei infrastructuri robuste pentru aplicații enterprise sau scenariile unui laborator avansat, alegerea mediului software de bază (sistem de operare clasic vs. hypervisor) are un impact major asupra stabilității, scalabilității și costurilor. În această secțiune argumentăm de ce un hypervisor dedicat este, în majoritatea cazurilor, mult mai potrivit decât un sistem de operare clasic și vom compara două soluții populare, Unraid (consumer-grade) și VMware ESXi (enterprise-level), avantaje, dezavantaje și costuri, pentru a susține decizia de a opta, în acest exemplu, pentru Unraid.

Limitările sistemelor de operare tradiționale în medii server

1. Izolare slabă a proceselor

- Un Linux sau Windows Server rulează procese în spațiul același kernel. Un defect sau o scurgere de memorie în aplicație poate afecta întregul server.

2. Mantenanță și compatibilitate

- Actualizările de kernel sau drivere pot cere reporniri frecvente și pot rupe dependențe între aplicații diferite.

3. Resurse partajate

- CPU, memorie și I/O sunt împărțite „la vedere” între procese, fără control fin al alocării stricte.

Toate aceste probleme se amplifică în scenarii în care este necesară rularea simultană a mai multe instanțe de aplicații cu cerințe diferite de sistem, diferite versiuni de librării sau chiar diferite familii de sisteme de operare.

O scurtă comparație între un sistem de operare clasic și un hypervisor poate fi dat de avantajul principal al hypervisor-ului, ce are capacitatea de a rula simultan și izolat sute de instanțe virtuale, fiecare cu propriul kernel și propriile setări, fără risc ca o problemă a uneia să compromită celelalte.

Caracteristică	OS Clasic	Hypervisor (bare-metal)
Izolare	Procese în același kernel	Mașini virtuale complet izolate, cu kernel separat
Mentenanță	Reporiri frecvente	Host stabil, mașini virtuale se pot migra live
Allocare resurse	Partajare implicită	QoS dedicat (CPU, memorie, disk I/O)
Diversitate OS guest	Limitată (containerizare)	Orice sistem de operare suportat de hardware
Scalare & Migrare	Greoaie	Snapshot, live migration, high-availability
Securitate	Slabă, este un all-in-one	Sporită, sisteme izolate.

Tabelul 6. Compararea OS Clasic cu Hypervisor

Containers Vs VMs

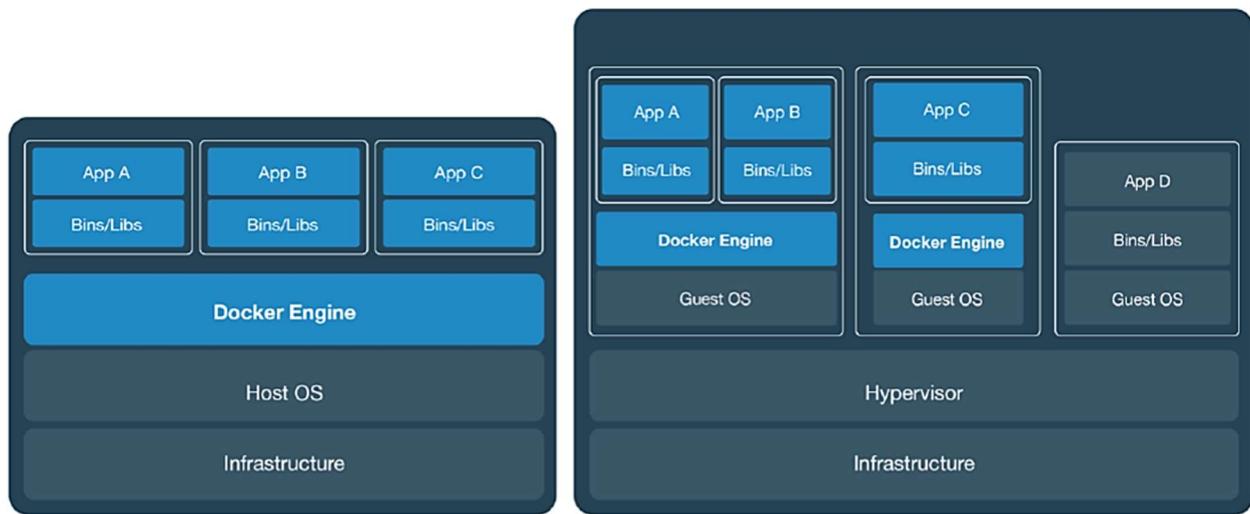


Figura 5. Containere și Mașini virtuale

Exemple de hypervisori server-side

Unraid (Consumer-Grade)

- **Modelul de licențiere:** o singură plată unică (59 USD pentru Basic, 89 USD pentru Plus, 129 USD pentru Pro), fără costuri recurente.
- **Arhitectură:** bazat pe Linux, cu un hypervisor KVM integrat și Docker pentru containere.
- **Puncte forte:**
 - Interfață web foarte prietenoasă pentru administrarea VM-urilor și containerelor.
 - Sisteme de fișiere independente pe fiecare disc, cu paritate opțională.
 - Ideal pentru laboratoare, home-lab sau întreprinderi mici, unde costul total de proprietate contează.
- **Limitări:**
 - Performanță KVM limitată în comparație cu soluții dedicate enterprise.
 - Suport comunitar pentru containere, actualizări mai rare și fără garanție de compatibilitate hardware.

VMware ESXi (Enterprise-Level)

- **Model de licențiere:** costuri inițiale ridicate (începând de la câteva sute de dolari per CPU socket) plus costuri anuale de suport și menenanță.
- **Arhitectură:** hypervisor bare-metal, extrem de optimizat pentru performanță și stabilitate.
- **Puncte forte:**
 - Funcționalități avansate de management (vMotion, DRS, HA, distributed switches).
 - Ecosistem matur de unelte de backup, securitate și orchestrare.
 - Certificări extinse pentru hardware de server, suport garantat.
- **Limitări:**
 - Costuri semnificative de licențiere și menenanță.
 - Complexitate ridicată, necesitând personal specializat.
 - Utilizarea Docker/Kubernetes presupune un VM dedicat.

Motivarea alegării Unraid

1. **Cost-eficiență:**
 - Buget redus pentru licență unică și fără costuri recurente.
2. **Ușurință în utilizare:**

- Interfață web simplificată permite studenților sau echipelor mici să configureze rapid VM-uri și containere.

3. Flexibilitate pentru prototip și testare:

- Schimbări rapide de configurație, snapshot-uri ușoare și mediu hibrid VM+containere.

4. Expunere academică:

- Un medialab bazat pe Unraid permite demonstrarea problemelor de management și orchestrare fără a investi în hardware și suport enterprise.

Concluzie

În contextul licenței, în care scopul este să analizăm și să comparăm soluții de virtualizare pentru servere, hypervisor-ul bare-metal se dovedește a fi alegerea optimă față de un OS clasic, prin:

- Izolare completă a mediilor de execuție
- Control particularizat al resurselor
- Scalabilitate și menenanță redusă

Între soluțiile disponibile, Unraid oferă un echilibru ideal între costuri reduse, simplitate de administrare și suport pentru atât VM-uri cât și containere. Pe de altă parte, VMware ESXi rămâne alegerea preferată pentru organizații cu cerințe de performanță, disponibilitate și suport dedicat.

[Disaster Recovery și strategii de utilizare a spațiului de stocare](#)

RAID-0. Fără redundanță (striping)

Datele sunt împărțite în blocuri și scrise simultan pe 2 sau mai multe discuri, pentru acces paralel. Un singur disc defect duce la pierderea totală a datelor de pe acel disc.

Caracteristică	RAID-0 (striping)
Discuri minime	2
Redundanță	0
Performanță	Foarte mare
Spațiu utilizat	100% capacitate
Risc pierdere date	Foarte ridicat

Tabelul 7. RAID-0

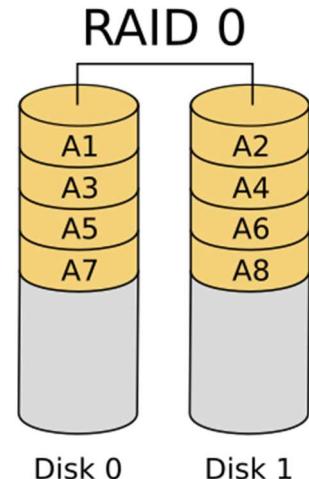


Figura 6. RAID-0

RAID-1. Oglindire (Mirroring)

Datele sunt copiate identic pe două sau mai multe discuri. Dacă un disc se defectează, datele rămân accesibile de pe celălalt

Caracteristică	RAID 1 (mirroring)
Discuri minime	2
Redundanță	1 (copie completă)
Performanță	Citire: mare; Scriere: ca la un disc
Spațiu utilizat	50 % (redundanță 1:1)
Risc pierdere date	Scăzut (suportă 1 disc picat)

Tabelul 8. RAID-1

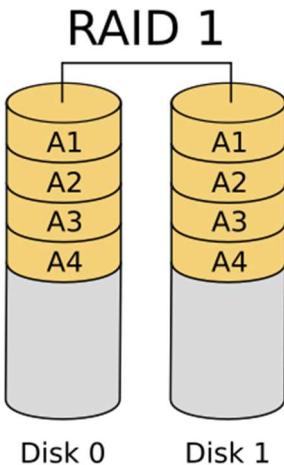


Figura 7. RAID-1

RAID-5. (Striping + Paritate distribuită)

Datele și paritatea (informație pentru reconstruire) sunt dispuse pe 3+ discuri. Un disc poate pica, apoi datele se pot reconstrui.

Caracteristică	RAID 5
Discuri minime	3
Redundanță	1 disc paritate
Performanță	Citire: mare; Scriere: medie (calc. paritate)
Spațiu utilizat	(n-1)/n din capacitate totală
Risc pierdere date	Moderat (până la 1 disc picat)

Tabelul 9. RAID-5

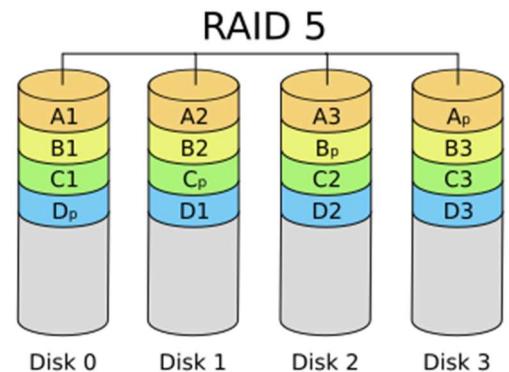


Figura 8. RAID-0

RAID-6. (Striping + Paritate dublă)

Similar RAID 5, dar cu paritate pe două discuri, toleranță pentru două defecțiuni de disc.

Caracteristică	RAID 6
Discuri minime	4
Redundanță	2 discuri paritate
Performanță	Citire: mare; Scriere: lent ($\sim 2 \times$ paritate)
Spațiu utilizat	$(n-2)/n$ din capacitate totală
Risc pierdere date	Scăzut (până la 2 discuri picăte)

Tabelul 10. RAID-6

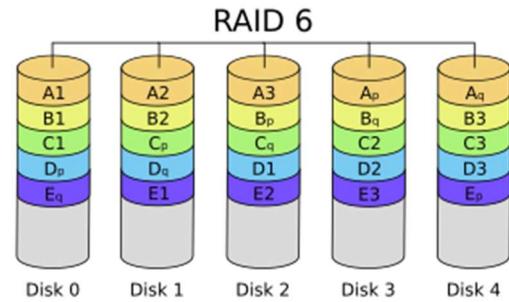


Figura 9. RAID-6

Virtualizare și Mașini Virtuale (VMs)

Virtualizarea este punctul de start a oricărei infrastructuri server moderne, permitând consolidarea, izolarea și managementul eficient al resurselor hardware. În context academic și industrial, înțelegerea tipurilor de virtualizare, a modului în care funcționează hypervisorii și gestionarea VM-urilor este necesară pentru proiectarea unei platforme robuste.

Ce este virtualizarea?

La nivel înalt, virtualizarea abstractizează resursele fizice cum ar fi procesorul, memoria, stocare, rețea, procesarea grafică și creează entități software izolate, numite mașini virtuale, care rulează propriul sistem de operare și aplicații.

- **Hypervisor Type 1 (bare-metal):** Se instalează direct pe hardware (de ex. VMware ESXi, Microsoft Hyper-V, KVM, Xen). Are control direct asupra resurselor și oferă performanță și securitate maxime.
- **Hypervisor Type 2 (hosted):** Rulează deasupra unui OS gazdă (de ex. VMware Workstation, VirtualBox). Mai ușor de configurație pe desktop, dar cu performanță și izolare mai reduse.

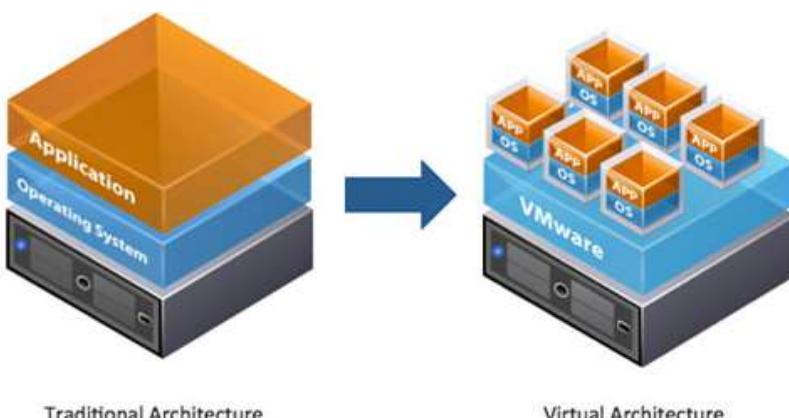


Figura 10. Diferența între arhitectură clasică și arhitectură virtuală

Beneficiile utilizării VM-urilor pe server

1. **Consolidare hardware:** Multe VM-uri pe același server fizic reduc costul pe instanță și spațiul fizic ocupat.
2. **Izolare și securitate:** Fiecare VM are propriul canal de memorie și de stocare, astfel un crash în interiorul unui VM nu afectează celelalte aplicații ce rulează în paralel pe alte VM-uri.
3. **Flexibilitate OS:** Se poate rula simultan Windows Server, Linux, BSD sau alte sisteme de operare pe aceeași gazdă.
4. **Snapshot-uri și rollback:** Există posibilitatea salvării stării unui VM înainte de upgrade, astfel restaurarea se poate face rapid și simplu în cazul în care modificările aduse nu sunt compatibile cu sistemul.
5. **Live migration & High Availability:** Dacă am opta pentru VMWare ESXi, există posibilitatea de a muta o VM între noduri (gazde) fără downtime (vMotion, Live Migration). Se pot configura strategii de failover, pentru a asigura funcționalitatea fără nevoie unui om de a reporni manual o stație de lucru.

Principalele componente ale unei VM

- **Virtual CPU (vCPU):** Se pot aloca cicli CPU extrași din pool-ul fizic. Un hypervisor poate totuși să distribue mai mult CPU decât are fizic, unor mașini virtuale. Acest lucru se numește overcommit sau suprasolicitare.
- **Memorie virtuală:** Blocuri de RAM dedicate fiecărei VM sau partajate prin tehnici de deduplicare (ballooning, transparent page sharing). Memoria RAM poate fi și ea suprasolicitată, altfel spus, alocarea logică unui bloc de memorie mai mare decât există fizic în stația de lucru.
- **Disk virtual (vDisk):** Există fișiere sau volume care apar pentru VM ca disc local. Spațiul acestora poate fi unul fix sau variabil după cerere.
- **Interfețe de rețea virtuală:** Adaptorii (vNIC) conectați la switchuri virtuale, cu VLAN-uri și reguli de securitate la nivel de hypervisor.

Probleme și limitări uzuale

1. **Overcommit și performanță:** Overcommit-ul CPU/memorie poate duce la “noisy neighbor”, o VM consumatoare afectează celelalte VM-uri. Monitorizarea și QoS (reservări, limite) sunt absolut necesare.
2. **I/O contention:** Diskurile virtuale pe același pool pot suferi de I/O head-of-line blocking (FIFO). Soluțiile sunt să construim stocarea în straturi astfel, SSD-uri pe post de memorie cache și HDD-uri pe post de memorie de stocare.
3. **Complexitate rețea virtuală:** Gestiona multiplilor vSwitch-uri, distribuirea VLAN-urilor și asigurarea de securitate (firewalling la hypervisor) pot complica arhitectura.
4. **Backup și recuperare:** Salvarea unui host întreg nu înseamnă salvarea instanțelor de VM, fiind necesară utilizarea unor soluții specifice (snapshot-driven).

Fluxul de lucru tipic pentru gestionarea VM

1. **Provisioning:** Creare unui model (template), definirea vCPU/memoriei, atașarea stocării și a rețelei.
2. **Configurare și personalizare:** Instalarea de drivere paravirtuale (VMware Tools, VirtIO pentru KVM).
3. **Monitorizare și tuning:** Colectare de metrii CPU, memorie, I/O, ajustare QoS și rezervări.
4. **Scalabilitate și Recovery:** Adăugare de noduri de gazdă la cluster este altfel spus atașarea unor noi stații de lucru la cele existente și redistribuirea automată a VM-urilor conform politicilor de resurse și de failover.

Virtualizarea cu VM oferă nivelul maxim de izolare, flexibilitate și control al resurselor pentru un mediu server-side. În capitolele următoare vom analiza detaliat modalitățile de extensie ale stratului de virtualizare.

2.2.2. Networking

Acces securizat la server și comunicații între dispozitive. VPN, DDNS, protocol de fișiere și acces la distanță

Într-o infrastructură server-side modernă, asigurarea accesului securizat, atât pentru utilizatori umani, cât și pentru dispozitive IoT sau microcontrolere, este vitală. În acest capitol vom detalia:

1. **VPN tunneling** (cu accent pe Tailscale, dar și WireGuard, OpenVPN)
2. **DDNS** pentru rezoluția dinamică a IP-ului public
3. **Accesarea la distanță** și de ce evităm accesul direct
4. **Partajarea fișierelor la nivel de aplicație** SMB/CIFS pe Windows, NFS/SMB pe Linux/Mac
5. **Integrări IoT** microcontrolere conectate prin Docker + ThingsBoard

DDNS (Dynamic DNS)

Problema IP-ului dinamic

Majoritatea conexiunilor la internet ce se află la un preț redus au IP-uri publice dinamice. Dacă serverul Unraid rulează pe un astfel de IP, acesta se va schimba periodic, făcând imposibilă conectarea directă.

Soluția DDNS

Un serviciu DDNS (de ex. No-IP, DynDNS, DuckDNS) leagă un nume de domeniu (ex: myhome.ddns.net) de IP-ul actual al serverului, astfel, un client DDNS rulează pe server, actualizând înregistrarea de fiecare dată când IP-ul se schimbă.

Cum integrăm DDNS cu VPN

- La conectarea prin VPN, accesăm direct myhome.ddns.net, ori prin IP-ul VPN gateway-ului. A doua variantă este de fapt un IP creat de serviciul de VPN la care se află numele de domeniu myhome.ddns.net.

- Clienții se conectează prin Tailscale/WireGuard/OpenVPN folosind DDNS, apoi pot accesa resurse interne prin IP-urile private din VPN.

VPN Tunneling

Ce este VPN tunneling?

VPN (Virtual Private Network) creează un tunel criptat între client (laptop, telefon, microcontroler) și serverul sau rețea privată. Tot traficul trece prin acest tunel, protejându-l de spionaj (snooping) sau modificare de către terți.

De ce folosim VPN?

- **Criptare end-to-end:** datele sunt criptate și decriptate la fiecare accesare Server-Client sau Client-Server.
- **Izolare:** serverul și resursele nu sunt expuse direct pe internet.
- **Controlul accesului:** doar device-urile autorizate în VPN pot vedea sau folosi resursele interne.

Soluții populare de VPN

Serviciu	Tehnologie	Model	Ușurință instalare	Performanță	Observații
Tailscale	WireGuard	SaaS P2P	Foarte simplu	Foarte bun	Operațional în câteva minute; CRL
WireGuard	WireGuard	Open-source	Simplu	Excelent	Kernel module, puțin cod, auditabil
OpenVPN	SSL/TLS	Open-source	Mai complex	Bun	Multă flexibilitate, compatibilitate

Tabelul 10. Servicii VPN

Acces la distanță vs. VPN

Anydesk, TeamVienwer, RustDesk

Toate cele trei aplicații au ca scop accesarea directă a unui sistem de operare. Totuși, acestea ar facilita un acces direct, necriptat dar și salvarea anumitor date în serverele companiilor ce dețin aceste programe. Cea mai bună variantă este RustDesk, deoarece este singura care este complet gratuită fără limitări. În plus, putem găzdui propriul nostru server de acces RustDesk, astfel, traficul generat către server nu este interceptat de terți.

De ce evităm accesul expus direct?

Reprezintă un punct de acces ce poate fi vulnerabil sau exploarat, lipsește criptarea end-to-end (unele folosesc TLS, dar tot salvează chei pe servere terțe) dar și control este limitat, întrucât nu putem accesa direct serverul. Instalarea unui astfel de program de acces la distanță presupune instalarea lui într-o mașină virtuală, ceea ce înseamnă că depindem de acea mașină virtuală dar și de stabilitatea sistemului

de operare din interiorul acesteia. În plus, deși putem accesa serverul și porturile din interiorul rețelei LAN în care acesta se află, în continuare este necesară cunoașterea datelor de logare.

Modelul propus. Access la distanță din interiorul VPN

Porturile accesului la distanță rămân închise către internet public. Acestea se deschid doar în tunelul VPN. În continuare, VPN-ul se va conecta la myhome.ddns.net, astfel avem un tunel direct la rețea, iar acum se poate lansa TeamViewer/AnyDesk/RustDesk. Dintre beneficii, avem zero expunere directă pe internet dar și administrarea accesului se face cu credentiale și chei VPN (mult mai greu de spart). Dezavantajul acestei metode relativ invazive este reprezentat de faptul că nu am folosi o placă de rețea dedicată pentru management, ci am folosi placă de rețea dedicată pentru fluxul de date, îngreunând traficul extern. Totuși reprezintă o metodă de backup la conectarea cu stația de lucru.

Integrarea microcontrolerelor și Docker ThingsBoard

Microcontrolere în VPN

- Se instalează un client WireGuard/Tailscale pe un gateway (ex. Raspberry Pi/ESP32) ce se va conecta prin DDNS la server, astfel conectând indirect microcontrolere pe rețea internă.
- Dispozitivele IoT pot păstra conexiunea către ThingsBoard intr-un canal MQTT criptat peste VPN.
- Principalul scop este de a putea atașa un dispozitiv conectat la internet într-o clădire A și achiziția informațiilor de către server, ce se află în clădirea B, fără o conexiune fizică a internetului între aceste două clădiri. Acest lucru este valabil și pentru achiziția datelor atunci când serverul și dispozitivul de transmisie se află în țări diferite.

ThingsBoard în Docker

Thingsboard reprezintă o platformă open-source pentru management IoT. Altfel spus, acesta are ca scop principal colectarea telemetriei, vizualizarea unui dashboard, și setarea regulilor de alertă.

ThingsBoard stochează în baza de date și expune API aplicațiilor interne (pe port 8080), astfel utilizatorii accesează Dashboard-ul doar prin VPN; autentificare pe HTTPs intern.

Microcontroler → ThingsBoard (MQTT) → Server

Laptop → VPN → Server → Thingsboard → Microcontroller

În același timp, prin folosirea unui VPN pentru conectarea la server a unui microcontroler, serviciul de VPN devine un nod de rețea. Lucrurile ar arăta astfel.

Microcontroler → VPN → ThingsBoard (MQTT) → Server

Laptop → VPN → Server → Thingsboard → VPN → Microcontroller

În final, prin înțelegerea și utilizarea tehnologiilor menționate, până în acest punct teoretic ceea ce a fost construit arată astfel:

- VPN Tunneling (Tailscale/WireGuard/OpenVPN) pentru accesul exclusiv securizat, astfel nimic nu este expus direct pe internet, minimizând suprafața de atac.

- Accesul administrativ se realizează doar prin tunel VPN (autentificat și criptat)
- Accesul la aplicații (fișiere, API-uri, MQTT) se face pe porturi deschise doar în VPN
- Microcontrolerele pot comunica fără conexiune fizică directă la server.
- Utilizăm Protocol de fișiere (SMB/NFS/AFP) expuse peste VPN dar și local
- Putem folosi mașini virtuale și diverse containere Docker
- Putem accesa ThingsBoard într-un mediu securizat
- Setări adiționale în Firewall
- Configurare RAID0/1/5/6.

Aceasta este cea mai sigură metodă de a asigura comunicări și administrare pentru un server Unraid hibrid (VM+Docker) cu zeci de senzori și microcontrolere interconectate.

[Porturi de rețea](#)

Ce este un port de rețea?

Un port de rețea este un «număr» (de la 1 la 65535) care indică o aplicație sau un serviciu pe care serverul îl pune la dispoziție. Altfel spus, IP-ul reprezintă o adresă locală, iar port-ul este adresa unei aplicații ce se află în interiorul IP-ului. Prințipiu de funcționare este similar cu virtualizarea unui sistem de operare. Portul reprezintă doar un nume ce spune ce aplicație se regăsește la capătul acestuia. Este foarte util în contextul containerelor, întrucât la port 22 avem SSH (Secure Shell), la port 80 avem interfața web locală a stației de lucru, la port 1200 poate fi NextCloud, iar Thingsboard se poate afla la port 8060. Avem astfel o delimitare clară a aplicațiilor când acestea sunt accesate prin intermediul conexiunii la rețea.

Un exemplu mai tehnic poate fi dat de următoarea descriere. Serverul se află la ip-ul local 192.168.0.50, dar în interiorul acestuia, rulează o mașină virtuală, la ip-ul 192.168.0.51. Noi ca și utilizatori, suntem conectați la adresa 192.168.0.101. Dacă dorim acces SSH la unul dintre aceste două entități din rețea, putem specifica clar la ce adresă dorim acest acces. În Windows, se poate folosi comanda “<ssh user@192.168.0.50/51>” care automat va deschide portul 22 spre aceste rețele. Dacă ar fi să presupunem faptul că accesăm direct o anumită rețea, acest lucru poate fi făcut prin accesarea 192.168.0.50:22 ori 192.168.0.51:22, delimitând clar la care instanță dorim să ne conectăm.

Partajarea fișierelor la nivel de aplicație

Într-o rețea locală sau accesată printr-un VPN, fișierele stocate pe server pot fi accesate în mod direct prin atașarea unui folder setat ca și partajat, fie public, privat sau cu acces securizat, ceea ce necesită un nume de utilizator și o parolă. Cu atât mai mult, drepturile de scriere și citire pot fi modificate pentru fiecare utilizator în parte și pentru fiecare folder partajat în parte. Această partajare este cel mai adesea un loc în care mai multe persoane pot accesa același fișier, instantaneu, dar și partajarea de noi fișiere în acest nou spațiu destinat colaborării. Protocolele folosite pentru această accesare directă a mediului de stocare diferă de la un sistem de operare la altul.

Windows → SMB/CIFS (protocolul SAMBA)

- **Porturi:** TCP 445 (SMB direct), TCP 139 (NBT-Session), UDP 137/138 (NetBIOS)
- **Caracteristici:**
 - Autentificare Kerberos/NTLM

- Permisii ACL bogate pe fișiere și directoare
- Integrare în Active Directory

Linux → NFS și SMB

- **NFS (Network File System):** port 2049 TCP/UDP
 - Ușor de configurat în Linux; suportă permisiuni POSIX
- **SMB prin Samba:** porturi identice cu Windows
 - Resursele sunt accesibile atât pe Windows cât și de pe Linux

macOS → AFP/SMB/NFS

- **AFP (Apple Filing Protocol):** port 548 TCP
- **SMB:** port 445 (versiunea modernă recomandată de Apple)
- **NFS:** port 2049

[Firewall](#)

Ce este un firewall?

Un firewall este un „gardian” software (sau hardware) care decide ce trafic de rețea intră șiiese din server. El inspectează pachetele pe baza unor reguli (port, adresă sursă/destinație, protocol).

Ce face mai exact un firewall?

Blochează atacurile, dacă de exemplu un atacator încearcă să acceseze serviciile serverului pe porturi nepermise, firewall-ul respinge pachetele.

Izolează segmente de rețea, astfel se poate permite doar traficul de management dintr-un VPN securizat, iar traficul public doar pe porturile necesare.

Filtrare fragmentată permite a se defini reguli precise. Un exemplu ar fi „permite SSH port 22 de la IP-ul 192.168.1.100” sau „blochează tot trafic pe portul 445 din exterior”.

[Dual-NIC \(Placă de rețea dublă\)](#)

Ce este o placă de rețea?

O placă de rețea (NIC, Network Interface Card) este componenta hardware care leagă serverul la rețea. O configurație Dual-NIC înseamnă pur și simplu că avem **două** plăci de rețea (sau două porturi fizice pe aceeași placă) conectate la switch-uri distincte sau în bonding.

Ce face această placă de rețea?

Plăcile de rețea asigură conexiunea la internet și conexiunea locală dintre server și diverse dispozitive conectate bineîntele, local.

Alte roluri se regăsesc și în redundanță (failover), astfel dacă un cablu sau o placă se defectează, traficul se mută automat pe o placă de rezervă, asigurând accesibilitatea serverului.

Traficul generat de mașinile virtuale dar și a accesării de management poate fi împărțit. În mod normal, o interfață este dedicată raficului de management (interfață web, SSH) iar o altă interfață raficului de date (partajări SMB/NFS, VM-to-VM), evitând suprasolicitarea unei singure legături.

Dacă switch-ul suportă LACP, se poate construi un pod între cele două interfețe, devenind o singură interfață virtuală în care raficul circulă simultan pe ambele cabluri, crescând viteza totală.

2.3. Docker

2.3.1. Introducere

Pentru a înțelege de ce Docker a revoluționat modul în care livrăm și rulăm aplicații, haideți să descompunem pe îndelete ce face, cum funcționează "la nivel de schemă bloc" și cum îl folosim în mediul Unraid.

Docker permite "împachetarea" unei aplicații (cod, librării, fișiere de configurare) într-un container care conține tot ce îi trebuie pentru a rula identic oriunde, astfel se asigură:

- Izolarea, întrucât fiecare container are propriul său spațiu de stocare și propriile procese, separate de restul serverului.
- Portabilitatea deoarece același container care rulează pe laptopul tău rulează la fel și pe Unraid, pe cloud sau pe alt server Linux.
- Rapiditate deoarece lansarea unui container este aproape instantanee, spre deosebire de pornirea unei mașini virtuale, deoarece nu pornești un întreg sistem de operare, ci doar procesele aplicației.

2.3.2. Cum funcționează Docker?

Docker se bazează pe două concepte-cheie:

1. Engine-ul Docker, instanță care rulează pe server (daemon) și care se ocupă de:
 - Crearea, pornirea și gestionarea containerelor
 - Construirea imaginilor pe baza fișierelor Dockerfile
 - Comunicarea cu registrul de imagini (Docker Hub, registry privat)
2. Clientul Docker, interfață cu care se interacționează (comenzi CLI sau GUI pe Unraid), ce trimite instrucțiuni daemon-ului. Altfel spus, este partea de interfațare dintre om și mașină.

Layerele imaginilor Docker

Orice imagine Docker este compusă din mai multe straturi (layers) suprapuse

- Layer 0: imaginea de bază (de ex. debian:bullseye sau python:3.9)
- Layer 1: adăugarea de fișiere, librării și pachete specificate în Dockerfile
- Layer 2, 3 ... N: fiecare instrucțiune RUN, COPY sau ADD din Dockerfile creează un layer nou

Avantajele straturilor:

- Refolosirea resurselor întrucât dacă două imagini au același layer de bază, acel layer se descarcă o singură dată și se partajează între cele două sesiuni.
- Cache: la rebuild, Docker verifică care instrucțiuni nu s-au schimbat și reutilizează layer-ul, accelerând procesul.
- Prin utilizarea containerelor separăm partea de backend cu cea de frontend.
- Se pot rula și testa versiuni diferite de aplicații fără a afecta sistemul de operare principal.
- Utilizarea Docker presupune în mare parte și CI/CD, ce reprezintă pipeline-uri care construiesc automat imagini, rulează teste și le pun în producție printr simpla apelare a comenzi “*docker pull && docker run*”.
- Din punct de vedere al migrației și a creării copiilor de rezervă, toate fișierele pot fi puse într-un singur container, iar acest container poate fi mutat pe alt server, asigurând funcționalitatea 1 la 1.

[Docker pe Unraid – App Store, comunitate și oficial](#)

Unraid include un App Store integrat (numit și Community Applications) ce permite acces la sute de template-uri Docker, adăugate atât de echipa oficială Unraid, cât și de utilizatori din comunitatea globală. Astfel, aceste aplicații se împart în două categorii.

1. Aplicații oficiale, reprezentate de template-uri verificate de Lime Technology (autorii Unraid), de exemplu Plex, Nextcloud sau Home Assistant.
2. Aplicații de comunitate, ce sunt template-uri create de pasionați care adaugă soluții variate (de la servere de jocuri la instrumente de dezvoltare).

Docker este motorul care pune “roțile în mișcare” în cadrul infrastructurii Unraid, permitând instalarea rapidă și izolată a oricărei aplicații disponibile în App Store-ul Unraid, fie ea oficială sau creată de comunitate. Layer-urile imaginilor asigură eficiență la stocare și rapiditate în rebuild-uri, iar GUI-ul Unraid ne scutește de task-uri de lină de comandă, făcând întregul proces accesibil oricui.

Capitolul 3. CI/CD – Arhitectura platformei IoT

CI/CD este un acronim pentru Continuous Integration (Integrare Continuă) și Continuous Delivery/Deployment (Livrare/Implementare Continuă).

- Continuous Integration (CI) se referă la faptul că orice modificare făcută de un dezvoltator în codul sursă este integrată automat într-o ramură principală (de obicei „main”). De îndată ce un „commit” (un update al codului sursă) este trimis în repository (serverul ce face versionare și istoric al schimbărilor), un server CI preia automat codul, îl compilează, rulează teste unitare și raportează imediat eventualele erori, asigurând astfel o structură similară a unui SiL (Software in the Loop). Scopul este să identifice rapid conflictele și defectiunile, înainte ca ele să ajungă să afecteze tot proiectul.
- Continuous Delivery (CD) se referă la faptul că, odată ce codul trece cu succes toate etapele de testare din CI, el este pregătit automat pentru a fi livrat într-un mediu de producție (sau de testare automată), de obicei sub forma unui pachet sau container gata de instalare. Diferența față de deploymentul manual este că tot procesul tehnic (construcția, testarea, ambalarea) este automatizat, iar echipa poate lansa oricând o versiune stabilă cu un singur click. Astfel, lucrurile merg un pas mai departe, dacă toate testele și verificările din pipeline-ul CD trec cu succes, codul este automat „deploy-at” direct în producție, fără niciun pas manual de aprobare.

Legătura cu Docker

1. Izolare și consistență

Docker oferă „containere” care închid într-un mediu controlat toate dependențele (biblioteci, runtime, configurații). Într-un pipeline CI/CD, folosirea Docker înseamnă că mediul de build, test și producție este întotdeauna identic, eliminând cuvintele frecvente „la mine a mers”.

2. Construire rapidă și versionare

Pașii de construire a unei imagini Docker (definite într-un Dockerfile) pot fi integrați direct în pipeline-ul CI. De fiecare dată când se schimbă codul, serverul CI construiește o nouă imagine Docker, o testează și, dacă totul e în regulă, o publică într-un registru (DockerHub).

3. Dezvoltare și scalare

În CD, livrarea containerelor Docker în mediul de producție sau de test este simplă, serviciile automatizate de Kubernetes sau Docker Swarm pot prelua noile imagini și pot gestiona automat scalarea, rularea simultană a mai multor instanțe inclusiv balansarea încărcării și rollback-ul în caz de eșec la nivel de mașină virtuală.

4. Pipeline unificat

Un exemplu de lanț de execuție este:

- Developer face commit în Git.
- Server CI preia codul, rulează docker build și apoi docker run pentru testele automate.

- Dacă testele sunt în regulă, imaginea este etichetată cu un număr de versiune și copiată în regisztr.
- Un job CD o preia din registry și o deploy-e în mediul de staging sau direct în producție.

3.1. CI/CD Docker și Unraid

În acest capitol vom explora în profunzime platforma open-source ThingsBoard, modul în care este proiectată intern, cum funcționează, cine a creat-o și cum o putem rula în containere Docker. Vom detalia fluxul de date de la plăcile IoT care comunică prin Wi-Fi și protocolul MQTT până la persistarea în PostgreSQL și expunerea datelor. Vom acoperi, de asemenea, aspecte concrete de configurare: adrese, porturi, volume Docker, configurații PostgreSQL și strategii de scalare.

ThingsBoard este o platformă IoT open-source lansată în 2016 de compania ThingsBoard Inc., fondată de Andrey Shishkin și Alexandr Shishkin. Scopul inițial a fost de a oferi o soluție ușor de instalat și extins pentru colectarea, vizualizarea și procesarea telemetriei de la mii de dispozitive IoT. De la prima versiune, comunitatea a contribuit cu plugin-uri, drivere pentru protocoale suplimentare și integrare în Docker și Kubernetes.

ThingsBoard se bazează pe o arhitectură modulară, orientată pe microservicii, dar în varianta open-source toate componentele se pot rula într-un singur proces Spring Boot sau, pentru înaltă disponibilitate, în mai multe instanțe. Componentele cheie sunt:

1. Serviciile Centrale

- Serviciul principal care rulează logica platformei: rule engine, device provisioning, gestionează conexiunile MQTT și WebSocket. Realizat în Java folosind Spring Boot de la VMWare.

2. Motorul de procesare

- Motorul de procesare a evenimentelor orchestrează un flux definit grafic, transformă, filtrează sau trimită alerte și comenzi către dispozitive. Fiecare nod din flow poate apela scripturi JavaScript sau trimită cereri către MQTT endpoint-uri externe. Aici se pot executa și RPC (Remote Procedure Call)

3. Adaptoare de transport

Aceste adaptoare fac referință la metoda de accesare a serverului către microcontroler cât și reciproc.

- MQTT (port implicit 1883 non-TLS, 8883 TLS)
- CoAP (port 5683)
- HTTP REST (port 8080)

4. Serviciul API (Interfața de Programare a Aplicațiilor)

- Expune interfața REST și WebSocket pentru dashboard-uri, aplicații mobile și CLI.
 - REST API HTTP: 8080 (sau 80/443 cu un proxy invers)

- WebSocket: 9090 (pentru actualizări live pe dashboard)

5. Baza de date

- **PostgreSQL**: stochează configurațiile (device profiles, rule chains, users), telemetria, atașamente (alarms, audit logs). Schema include tabele pentru tb_device, tb_asset, device_timeseries, alarm, rule_node, rule_chain.
- **TimescaleDB** (optional): extensie pentru stocare de time-series în PostgreSQL, optimizând interogările pe volume mari.

6. UI (Dashboard)

- Aplicația front-end scrisă în Angular, servită de API Service, unde utilizatorii construiesc widget-uri grafice, creează alarme și definesc permisiuni.

3.1.1. Fluxul de date MQTT → ThingsBoard → PostgreSQL

1. Dispozitiv IoT

- Placă ESP32 (sau orice microcontroler Wi-Fi) rulează un client MQTT care:
 - Se conectează la mqtt://<IP_Server>:1883
 - Publică telemetrie pe topic-uri precum v1/devices/me/telemetry
 - Ascultă pentru comenzi pe v1/devices/me/attributes sau v1/devices/me/rpc/request/+

2. Transport MQTT

- Serviciile Centrale ascultă portul 1883 pentru:
 - Autentificare: token device sau username/password
 - Mesaje recepționate sunt transformate în evenimente interne și trimise spre Motorul de procesare

3. Motorul de procesare

- Filtru: ignoră datele atipice (e.g. temperatură > 100 °C)
- Conversie: convertește unități, adaugă timestamp UTC
- Scriere în baza de date: scrie în tabela device_timeseries din PostgreSQL
- Generare de alarmă: dacă umiditatea > 70 %
- Downlink: trimit un mesaj RPC înapoi pe MQTT

3.1.2. Implementare în Docker

Imaginea oficială

ThingsBoard furnizează o imagine Docker oficială (thingsboard/tb-postgres) care include:

- Java Runtime Environment (OpenJDK 11)
- ThingsBoard Application (jar Spring Boot)
- Driver PostgreSQL

PostgreSQL

- PostgreSQL rulează pe rețeaua internă tb-net, oferind port 5432 doar lucrătorilor interni.
- ThingsBoard foloseste porturile:
 - Port 1883 MQTT
 - Port 8080 pentru API HTTP/REST
 - Port 9090 pentru WebSocket

În orice companie serioasă, nevoia de instrumente bine puse la punct pentru gestionarea proiectelor, colaborare, stocare de fișiere, monitorizare și automatizare devine importantă. În trecut aceste soluții rămâneau „în nor” (SaaS, Software as a Service), cu licențe scumpe și dependență de furnizori externi. Astăzi, odată cu maturizarea containerizării, în special Docker, ne putem construi propriul „nor privat”, cu cost zero de licențiere (pentru variantele open-source) și flexibilitatea de a adăuga, elimina sau scala orice serviciu în câteva secunde. Astfel, este esențial să subliniem de ce toate aplicațiile rulează în containere.

1. Izolare

- Fiecare serviciu (JIRA, Nextcloud, GitLab, Thingsboard) are propriul mediu: librării, variabile, versiuni de limbi. Nu mai există conflict pe același Python sau Java instalat la nivel de sistem.

2. Rapiditate în deploy și rollback

- descarcăm imaginea prin “pull”, execuăm în terminal docker run și serviciul e live în câteva secunde.

3. Portabilitate

- Imaginea care rulează în laboratorul de test se comportă identic pe serverul de producție sau pe un laptop cu Linux.

4. Scalare pe verticală

- Docker Compose sau Kubernetes permite rularea a trei instanțe de GitLab Runner ori replicarea cu un singur fișier YAML (aceste fișiere stând la baza configurației oricărui container).

5. Costuri reduse

- Pentru variantele open-source nu se plătesc licențe.

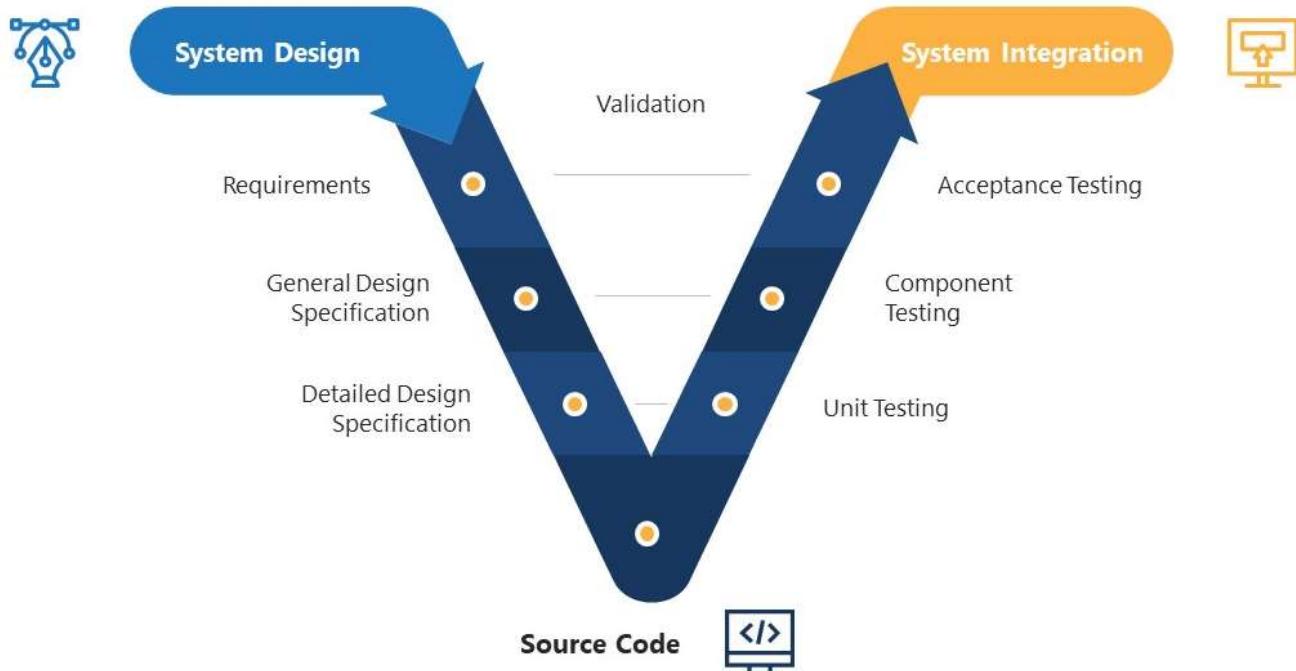


Figura 11. V-Cycle

JIRA (Atlassian Jira Software)

- Descriere: JIRA este, de fapt, standardul industrial pentru urmărirea de cerințe, sarcini și probleme.
- Model licențiere: Este contra cost în funcție de numărul de utilizatori.
- Dacă avem licență Enterprise, putem automatiza backup-uri și upgrade-uri.

OpenProject

- Fiind Open-source, cu licențiere GPLv3, OpenProject este complet gratuit atât timp cât nu este necesar să folosim adiții din spectrul Enterprise. Interfața este similară cu JIRA, și există plugin-uri comunitare.
- Considerații: baze de date PostgreSQL externalizată (volum Docker sau serviciu separat), e-mail server (SMTP) configurat pentru notificările de task-uri.

Redmine

- Redmine reprezintă o alternativă la Open Project, fiind Open-source, cu licențiere GPL, este complet gratuită, deși ceva mai veche, are o comunitate de utilizatori cu experiență și numeroase plugin-uri.
- Caracteristici: tracker de issue, wiki, SCM integration (Git, SVN), notificări.

[Colaborare cu fișiere](#)

[Nextcloud](#)

- Nextcloud reprezintă o variantă înlocuitoare pentru Google Drive sau One Drive cu precădere în mediul enterprise, este open-source și gratuit. Oferă stocare de fișiere, sincronizare, calendare, contacte, video-conferință, colaborare pe documente (prin OnlyOffice/Collabora).
- Control complet asupra datelor (GDPR friendly)

[GitLab Community Edition \(CE\)](#)

- În mod normal, o alternativă plătită și costisitare este dată de BitBuket ce este gândită în a fi utilizată împreună cu JIRA, GitLab este server Git, ce permite istoricul modificărilor, evaluarea codului scris, wiki, și integrare Kubernetes. Ca și licențiere GitLab CE e gratuit.

[Jenkins](#)

- Jenkins reprezintă un server de CI/CD foarte modular, cu sute de plugin-uri. Cel mai mare avantaj este dat de faptul că acesta prezintă libertate totală de scripting (Groovy), astfel există posibilitatea de integrare cu orice altă aplicație deja existentă. Se folosește cu precădere pentru partea de automatizare oricărui tip de software.

[Colaborare și comunicare internă](#)

[Mattermost / Rocket.Chat](#)

- Acestea reprezintă clone open-source ale Slack, cu canale de chat criptate end-to-end și autentificare prin doi pași. Avantajul este că tot ceea ce se discută în echipă, va rămâne cu siguranță doar în acea locație, fără intervenția unei companii terțe.

[Wiki și documentație:](#)

Unul dintre cele mai utile containere este Wiki.js, unde putem să avem propria noastră wikipedia. Este utilă atunci când un proiect presupune competențe multidisciplinare.

- **Wiki.js** – wiki modern pe Node.js, MariaDB/PostgreSQL

[ERP](#)

[Odoo](#)

- **Odoo** este un ERP (Enterprise Resource Planning) modular ce prezintă diverse unelte necesare unei companii ce dorește să economisească și să păstreze întreaga evidență a cheltuielilor și a situației firmei local, în propriul Cloud IoT.

Implementarea tuturor acestor containere oferă o flexibilitate imensă întrucât fiecare serviciu rulează independent, poate fi scalat sau replicat. Costul este aproape minim, majoritatea produselor fiind OpenSource. De altfel, un lucru foarte important este controlul deplin al datelor GDPR dar și propriul nostru ecosistem, facilitând de la management de proiect (JIRA, OpenProject) și colaborare (Nextcloud, Mattermost), până la CI/CD (GitLab, Jenkins), monitorizare (Graphana) și ERP (Odoo).

Inteligentă Artificială

Ollama

Ollama este un proiect open-source, ce îi permite utilizatorului rularea de modele mari de limbaj (LLM) pe serverul remote, oferind o interfață similară cu cea de la OpenAI. Este necesară o placă video pentru procesarea datelor și producerea rezultatelor.



Run [DeepSeek-R1](#), [Qwen 3](#), [Llama 3.3](#), [Qwen 2.5-VL](#),
[Gemma 3](#), and other models, locally.

[Download ↓](#)

Available for macOS, Linux,
and Windows

Figura 12. Modele de Inteligență Artificială

Capitolul 4. Implementarea practică

4.1. Pregătirea hardware-ului în operarea Unraid

Primul pas în bootarea hypervisorului Unraid, presupune scrierea acestui sistem de operare pe un drive USB, ce va boota în modul live. Din punct de vedere al hardware-ului, alegerea unei configurații stabile, compatibile și suficient de performante este importantă pentru funcționarea optimă.

Un prim criteriu de selecție este procesorul. Unraid rulează doar pe arhitecturi x86_64, deci orice procesor compatibil pe 64 de biți este acceptat, dar performanța generală va depinde în mare măsură de numărul de nuclee și de suportul pentru virtualizare (ex. VT-x, AMD-V).

- Recomandare minimă:
 - Procesor quad-core x86_64
 - Suport pentru virtualizare hardware (VT-x / AMD-V)
 - Minim 8 GB RAM pentru funcționare stabilă (preferabil 16 GB+ dacă se folosesc VM-uri)

În ceea ce privește stocarea, Unraid necesită cel puțin două unități: una pentru stocarea efectivă a datelor și alta optională pentru paritate, în scopul protecției împotriva pierderii de date în cazul defectării unui disc. SSD-urile pot fi utilizate ca discuri de cache pentru accelerarea accesului la date și a operațiunilor de scriere. Recomandarea minimă este o configurație RAID5 + Cache.

- Tipuri de discuri utile:
 - HDD-uri pentru stocare principală
 - SSD pentru cache
 - HDD-uri suplimentar pentru paritate

Un aspect diferit al hypervisorului Unraid este faptul că rulează de pe un drive USB. Licența este legată de GUID-ul drive-ului USB, unde GUID înseamnă Globally Unique Identifier. Altfel spus, licența achiziționată este fizic legată de drive-ul pe care îl folosim. Mutarea acestea pe un alt drive USB va anula licențierea primului drive.

- Cerințe pentru drive-ul USB:
 - Minim 4 GB, preferabil între 8 și 16 GB
 - Calitativ

După alegerea sistemului și a mediilor de stocare, următorul pas este conectivitatea. Unraid trebuie să fie conectat la rețea locală prin cablu Ethernet, iar conexiunea Wi-Fi este exclusă din cauza lipsei de stabilitate în termen lung la sarcină. Routerul trebuie să aibă DHCP activ, astfel încât sistemul să primească automat o adresă IP la prima pornire. În caz contrar, se consideră că utilizatorul este experimentat și cunoaște exact clasa de IP-uri locale. În mod obișnuit, DHCP-ul este un serviciu activ pe toate routerele, exceptie fac doar cele ce au fost configurate ca acest serviciu să fie dezactivat. Ideal este să fixăm IP-ul serverului, acesta devenind astfel un IP static în rețea LAN.

- Rețea:

- Conexiune prin cablu Ethernet (minim 1 Gbps)
- DHCP activ în router (optional)
- Posibilă alocare manuală a IP-ului (static/reservat)

În rețele mai avansate sau în medii de lucru cu cerințe ridicate de disponibilitate, scalarea accesibilității se face prin configurarea redundanță de rețea (failover). Dacă sistemul are două plăci de rețea, acestea pot fi conectate la două routere sau la un router cu două porturi WAN, pentru a evita întreruperile de rețea.

- Două cabluri Ethernet conectate la routere diferite sau la un switch configurabil
- Bonding / teaming pentru failover, configurabil ulterior în Unraid

În ceea ce privește porturile de rețea, pentru accesul la interfața web a Unraid trebuie să ne asigurăm că portul 80 este deschis în rețea locală. Dacă se dorește acces securizat prin HTTPS, este necesar și portul 443. În cazul în care se folosesc aplicații ca Plex sau Nextcloud, vor fi necesare porturi suplimentare redirecționate în router (port forwarding).

- Porturi relevante:
 - 80 (HTTP)
 - 443 (HTTPS, optional)
 - Porturi suplimentare în funcție de aplicații: ex. 32400 pentru Plex

4.1.1. Configurarea BIOS-ului și instalarea efectivă a Unraid

După ce hardware-ul este pregătit și drive-ul USB pe care vom pune Unraid este pregătit (vezi mai jos scrierea Unraid pe stick), urmează un pas important, accesarea BIOS-ului (Basic Input Output System) și realizarea setărilor necesare pentru ca sistemul să pornească corect.

BIOS-ul este acel sistem de control de bază al plăcii de bază, care decide ce se întâmplă imediat după ce butonul de pornire a fost apăsat. Pentru ca Unraid să funcționeze optim, trebuie să ne asigurăm că boot-ul este făcut corect de pe drive-ul USB și că opțiunile legate de virtualizare sunt active (dacă se doresc și fi folosite VM-uri).

- Intrarea în BIOS se face, de regulă, apăsând o tastă specifică imediat după pornire (Delete, F1, F2, F10. Depinzând de producător).
- În BIOS, ne asigurăm că pornirea inițializării (Boot) este setată să pornească de pe drive-ul USB, și nu de pe HDD/SSD.
- Este extrem de important să activăm următoarele opțiuni:
 - **USB Boot**, activat
 - **UEFI Boot sau Legacy Boot**, preferabil Legacy (sau compatibilitate BIOS), Unraid funcționează stabil pe ambele setări
 - **Intel VT-x** (sau AMD-V), activat, pentru suport virtualizare
 - **IOMMU/VT-d**, activat, pentru passthrough hardware (optional, dar recomandat)

- **AHCI Mode** pentru controlerele SATA (pentru compatibilitate mai bună)

Dacă sistemul are opțiuni pentru Fast Boot sau Secure Boot, acestea trebuie dezactivate. Secure Boot va bloca pornirea Unraid-ului în majoritatea cazurilor.

4.1.2. Scrierea imaginii Unraid pe stick USB

Hypervisor-ul Unraid nu se instalează pe discuri interne. Acesta rulează complet de pe drive-ul USB, iar datele de configurare (setările, licența etc.) sunt salvate în același loc, pe drive-ul USB. Din acest motiv, este important să scriem corect imaginea de boot.

- Se descarcă imaginea oficială Unraid de pe <https://Unraid.net/download>
- Se folosește Unraid USB Creator (disponibil pentru Windows/macOS/Linux)
- Selectăm:
 - Versiunea de Unraid dorită
 - Drive-ul USB conectat
 - Modul de boot (UEFI sau Legacy – se poate schimba ulterior)
- După confirmare, aplicația va scrie sistemul și va face automat drive-ul bootabil

Este recomandat folosirea unui port USB 2.0 (nu 3.0) pentru drive-ul de boot, acesta oferind stabilitate mai mare la pornire. De altfel, drive-ul trebuie lăsat conectat permanent, în lipsa acestuia, sistemul nu pornește.

4.1.3. Boot-ul inițial în Unraid

Odată drive-ul pregătit și BIOS-ul setat să booteze de pe acesta, putem porni efectiv sistemul. Prima bootare durează puțin mai mult și nu există interfață grafică locală – totul se gestionează ulterior prin browser, din rețea.

- La prima pornire, Unraid va încărca kernelul și va afișa câteva mesaje în consolă (fără GUI).
- Se vor afișa informații de tipul:
 - IP-ul local alocat de router prin DHCP
 - Numele sistemului (de obicei Tower)
- Dacă totul este în regulă, sistemul va aștepta conexiune de rețea prin browser

În caz că nu se obține automat o adresă IP:

- Verifică dacă portul Ethernet este conectat corect
- Verifică setările routerului (DHCP activ)
- Eventual conectează monitor tastatură la server pentru debug

4.1.4. Accesarea interfeței web

După boot, Unraid este accesibil din rețeaua locală. Se deschide un browser și se accesează IP-ul afișat la boot sau, alternativ, hostname-ul local (de obicei <http://tower.local> sau <http://tower>).

- Se recomandă:
 - Deschiderea paginii de pe un laptop conectat în aceeași rețea
 - Salvarea adresei IP locale pentru acces facil ulterior
- Din această interfață vom face:
 - Activarea licenței
 - Alegerea discurilor
 - Configurarea de paritate, cache, partajări
 - Adăugarea aplicațiilor și a serviciilor Docker și VM

4.2. Setări Unraid, Paritate, GUI și Magazinul de aplicații

Interfață grafică în Unraid

Odată ce sistemul a pornit cu succes și ai accesat adresa din browser (de exemplu, <http://tower.local> sau IP-ul local dat de router), intri pentru prima dată în interfață grafică a Unraid. Totul se gestionează de aici, nu mai ai nevoie de tastatură sau monitor conectate la server.

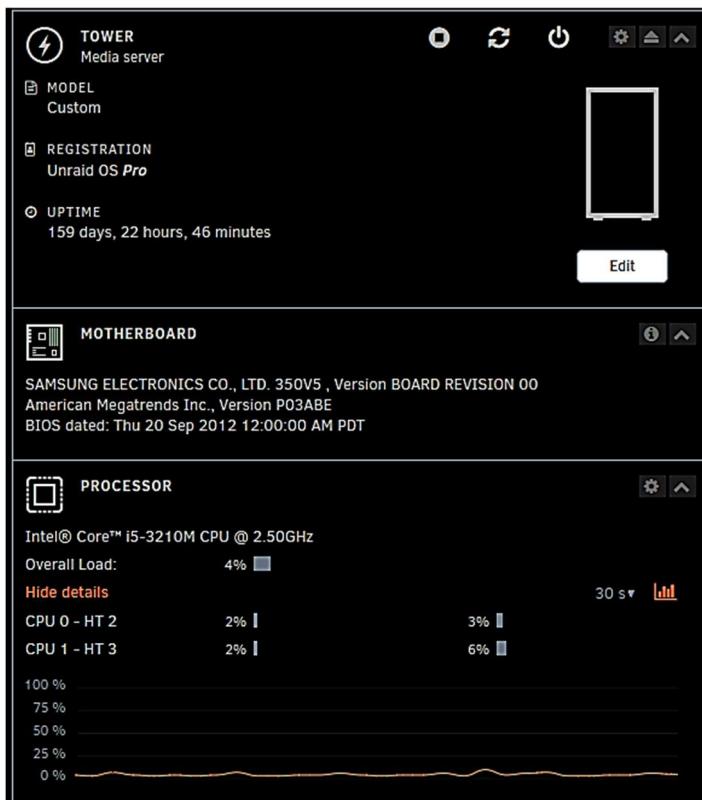


Figura 13. Interfața Unraid

Interfața este relativ minimalistă, dar oferă multe posibilități. În stânga ai meniul principal, iar în paginile centrale apar setările, starea sistemului, discurile, serviciile active etc. La prima accesare, Unraid va cere setarea unei parole de administrator și activarea unei licențe trial (30 zile).

Principalele secțiuni din Unraid sunt:

- **Dashboard** – imagine de ansamblu asupra sistemului
- **Main** – discurile conectate și configurația lor
- **Shares** – folderele partajate în rețea
- **Users** – administrarea conturilor
- **Settings** – toate setările importante
- **Docker** – dacă e activat, găsești containerele aici
- **VMs** – zona dedicată mașinilor virtuale
- **Apps** – App Store-ul comunității, foarte puternic

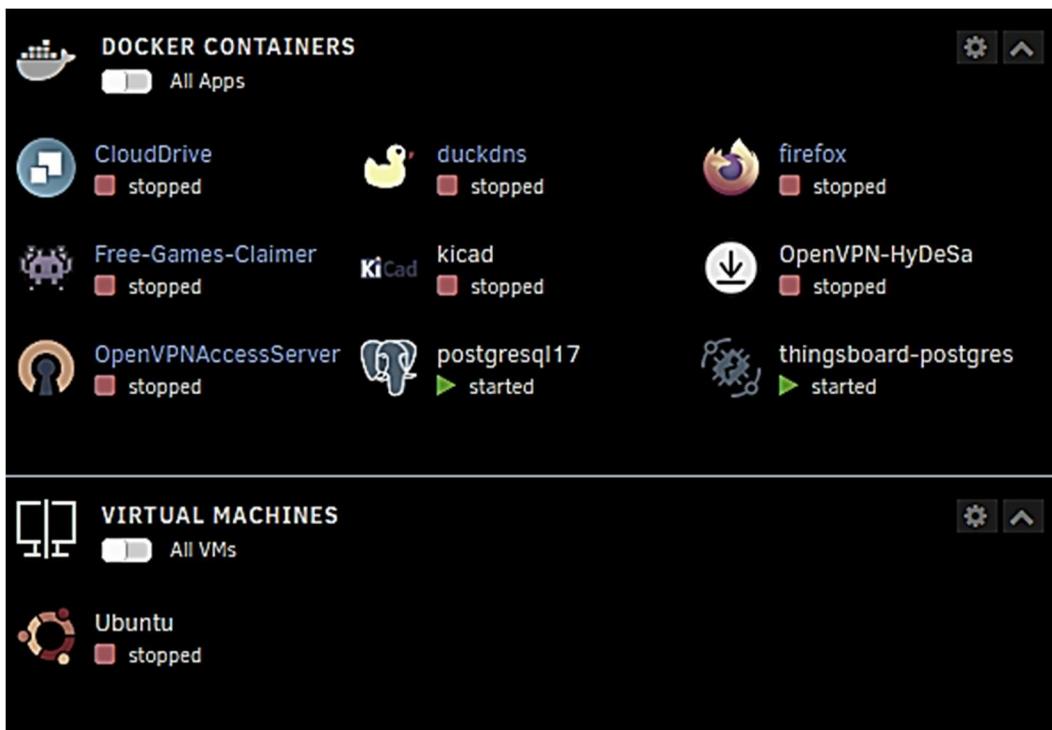


Figura 14. Interfața Unraid

O altă informație utilă pe care interfața o poate oferi este validitatea parității, sătătatea discurilor de stocare, temperatura și procentajul memorie ocupate. Dintre toate aceste detalii, informațiile legate de paritate sunt cele mai importante, întrucât aceasta ne asigură faptul că datele noastre se află în siguranță.

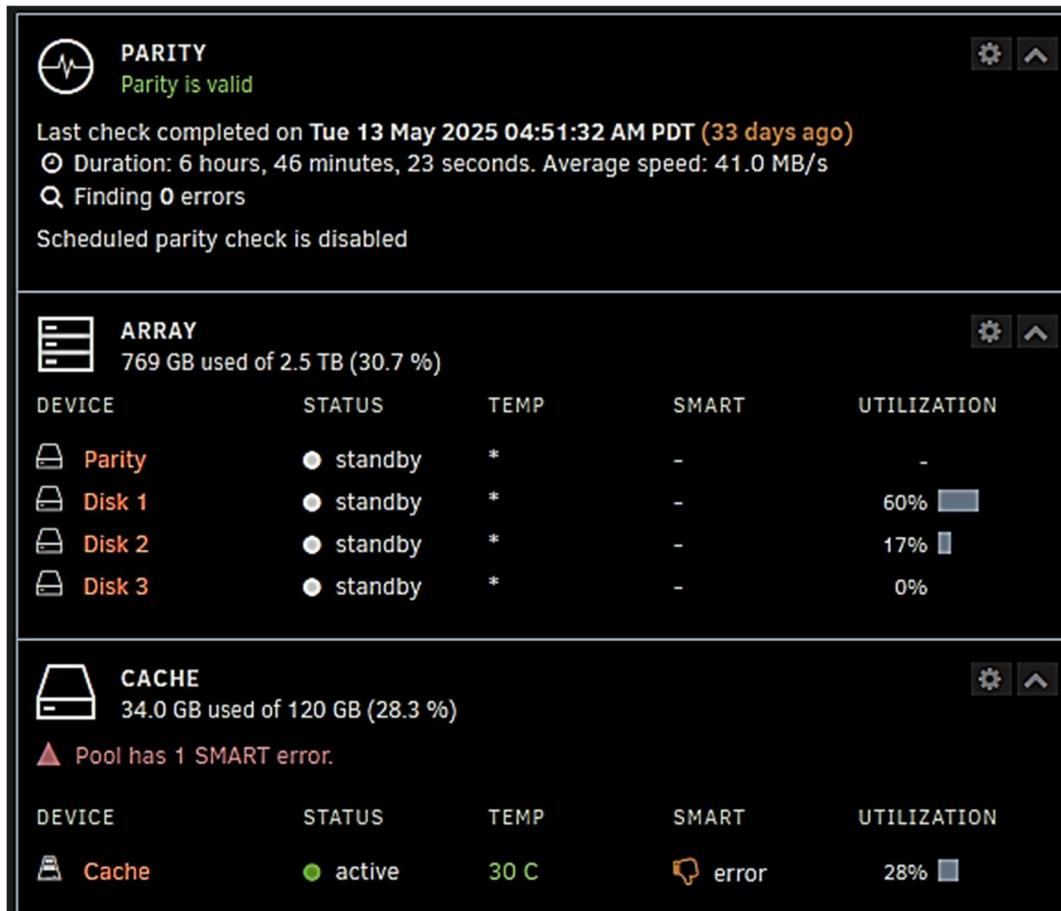


Figura 15. Interfața Unraid

4.2.1. Setarea discului de paritate, protecție anti-catastrofă

Unul dintre motivele pentru care utilizăm Unraid este paritatea. Configurarea dispozitivelor de stocare pentru a funcționa în RAID5 presupune faptul că dispozitivele nu stochează date, ci informații matematice care permit recuperarea datelor în cazul în care un disc din sistem cedează.

- Discul de paritate:
 - Trebuie să fie cel puțin la fel de mare ca cel mai mare disc de date
 - Nu va fi folosit pentru stocare directă – doar protecție
- Poate fi:
 - Un singur disc de paritate – protejează 1 disc
 - Două discuri de paritate – protejează 2 discuri simultan

Discul de paritate trebuie adăugat din tab-ul Main, selectând drive-ul dorit și apăsând "Start Array" pentru inițializare (paritatea se va genera, durează câteva ore).

4.2.2. Ce este cache-ul în Unraid?

Funcționarea și configurarea unui dispozitiv de cache în Unraid

Cache-ul reprezintă un spațiu de stocare intermedian, de regulă un SSD (sau un grup de SSD-uri), utilizat pentru a îmbunătăți performanța sistemului. Acesta interceptează fișierele scrise pe array, păstrându-le

temporar pe suporturi rapide, înainte de a le transfera definitiv pe discurile principale (HDD-uri), mai lente, din array.

Mecanismul de funcționare

- În momentul în care utilizatorul salvează fișiere într-un share configurat cu opțiunea de cache activată, acestea sunt scrise inițial pe dispozitivul SSD.
- Ulterior, sistemul Unraid execută un proces automat denumit „**mover**”, care transferă fișierele de pe SSD către HDD-urile din array.
- Acest proces rulează automat, de obicei pe timpul nopții, însă poate fi configurat să ruleze la o oră prestabilită sau să fie executat manual, în funcție de necesitate.

Configurarea unui SSD ca dispozitiv de cache

1. Accesați tab-ul „**Main**” din interfața Unraid.
2. În secțiunea „**Cache Devices**”, adăugați SSD-ul dorit.
3. Porniți array-ul, sistemul va detecta dispozitivul și îl va forma automat pentru a-l include în pool-ul de cache.
4. Navigați către secțiunea „**Shares**”, selectați un share (de exemplu: „Media”) și modificați opțiunea „**Use cache pool**” după cum urmează:
 - **Yes** – fișierele vor fi scrise mai întâi pe SSD și apoi mutate pe HDD la o oră programată.
 - **Prefer** – fișierele vor rămâne pe SSD atât timp cât există spațiu disponibil.
 - **Only** – fișierele vor fi păstrate exclusiv pe SSD, fără a fi mutate în array.
 - **No** – share-ul nu va utiliza deloc pool-ul de cache.

Avantajele utilizării unui dispozitiv de cache

- **Viteză crescută de scriere:** SSD-urile oferă o rată de transfer semnificativ mai mare comparativ cu HDD-urile, îmbunătățind astfel performanța generală.
- **Reducerea uzurii HDD-urilor:** Fișierele mari nu sunt scrise direct pe discurile mecanice, prelungind durata de viață a acestora.
- **Reducerea zgromotului:** Activitatea de scriere este direcționată către SSD, ceea ce contribuie la un mediu de lucru mai silențios.
- **Performanță sporită pentru aplicații Docker și mașini virtuale (VM):** Aceste aplicații beneficiază de un acces rapid la fișierele stocate pe SSD.

4.2.3. Community App Store

După instalarea și pornirea sistemului Unraid, una dintre cele mai importante funcționalități pe care utilizatorul o poate accesa este magazinul de aplicații, cunoscut sub numele de Community Applications. Acesta nu este activat din start, ci trebuie instalat manual. Practic, această extensie oferă acces rapid și organizat la sute de aplicații și containere Docker, create și întreținute de comunitate, care pot transforma serverul într-un adevărat centru de control personal sau industrial.

Însă, înainte de a putea instala orice container, este necesar să activăm serviciile Docker. Unraid nu pornește automat motorul Docker după prima instalare, deoarece presupune că utilizatorul va decide dacă dorește sau nu să folosească funcționalitatea. Activarea se face simplu din interfață, dar este un pas

esențial. În cazul nostru, vom folosi Docker pentru a instala ThingsBoard, un container open-source specializat în IoT și telemetrie, folosit frecvent în aplicații industriale sau de cercetare.

Pașii principali implicați:

- Se accesează tab-ul **Settings** → **Docker**, și se activează motorul Docker (Enable Docker → Yes).
- Se setează o locație de stocare pentru containerele Docker, de obicei pe SSD (ex: /mnt/cache/system/docker).
- Se salvează setările și se pornește serviciul.
- Ulterior, din tab-ul **Apps**, se poate instala plugin-ul "Community Applications".
- Odată instalat plugin-ul, utilizatorul poate căuta ThingsBoard (sau orice alt container) și îl poate adăuga în sistem.

4.2.4. ThingsBoard în Unraid

Instalarea containerelor și setări generale

Pentru a pregăti mediul ThingsBoard ca și container Docker în Unraid, este nevoie de o instalare în doi pași principali: mai întâi crearea bazei de date (PostgreSQL), iar apoi instalarea aplicației propriu-zise (ThingsBoard). Toți acești pași se desfășoară din interfața grafică, iar odată configurate corect, cele două componente vor colabora fără probleme în mediul local.

Total începe din tab-ul **Apps**, unde putem căuta și instala diverse containere. Aici vom începe prin a căuta containerul PostgreSQL, necesar ThingsBoard pentru stocarea datelor. În cazul de față, s-a optat pentru versiunea 17 a containerului PostgreSQL, disponibilă în magazinul de aplicații. Instalarea este simplă, iar configurarea trebuie realizată cu atenție, respectând anumite reguli de nomenclatură a variabilelor.

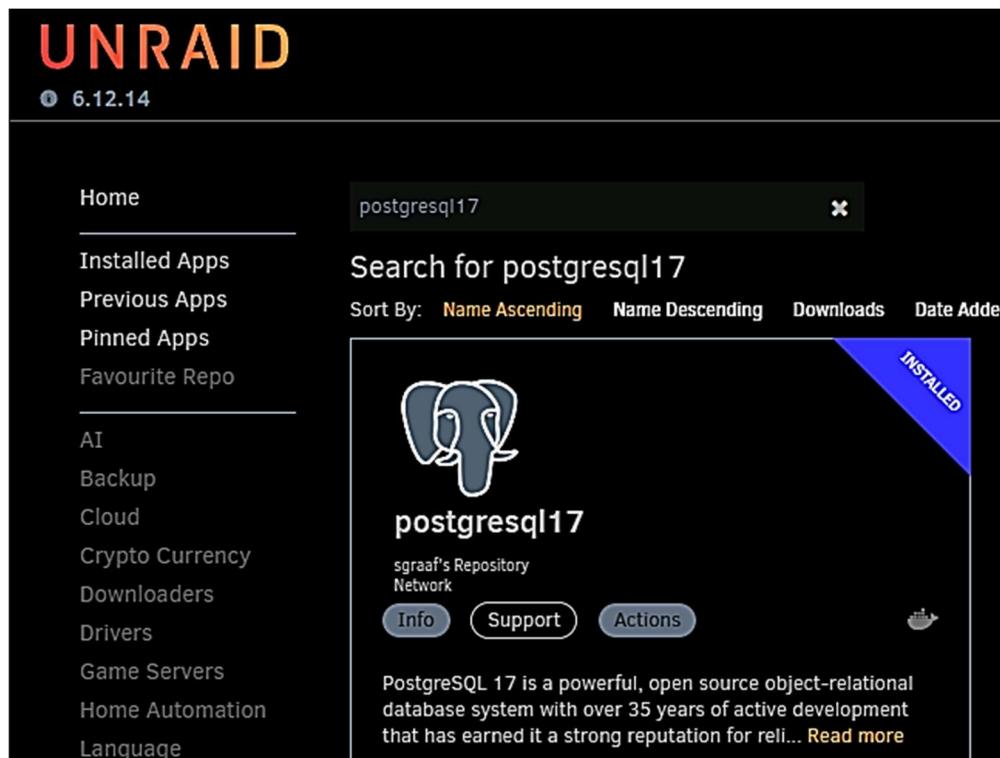


Figura 16. Instalare PostgreSQL

Este important ca, în momentul instalării, toate variabilele de nume utilizator, parolă și bază de date din PostgreSQL să fie setate pe cuvântul „thingsboard”, exact aşa, cu litere mici. Acest lucru este important pentru ca ThingsBoard să poată comunica corect cu baza de date, fără a necesita configurații suplimentare.

- Tab-ul Apps din Unraid este punctul de plecare pentru toate aplicațiile.
- Versiunea PostgreSQL aleasă: 17.
- Variabile recomandate pentru configurație:
 - Username: thingsboard
 - Password: thingsboard
 - Database name: thingsboard

Setting	Value	Description
Repository:	postgres:17	
Network Type:	Bridge	
Console shell command:	Shell	
Privileged:	OFF	
Port: PostgreSQL access port: *	5432	Container Port: 5432 PostgreSQL TCP connection port.
Path: /var/lib/postgresql/data: *	/mnt/user/appdata/thingsboard-postgres	Container Path: /var/lib/postgresql/data PostgreSQL data storage location.
Variable: POSTGRES_PASSWORD: *	thingsboard	Container Variable: POSTGRES_PASSWORD Initial superuser password (required).
Variable: POSTGRES_USER:	thingsboard	Container Variable: POSTGRES_USER Initial superuser name (default: postgres).
Variable: POSTGRES_DB:	thingsboard	

Figura 17. Instalare PostgreSQL

După ce instalarea este completă, revenim la pagina principală a interfeței, în secțiunea Dockers, unde putem vedea toate containerele instalate. Dacă logurile containerului PostgreSQL vor fi deschise, un rezultat pozitiv va indica faptul că serviciul este activ și ascultă pe portul 5432, port care nu trebuie modificat în această etapă, deoarece este cel standard folosit de PostgreSQL.

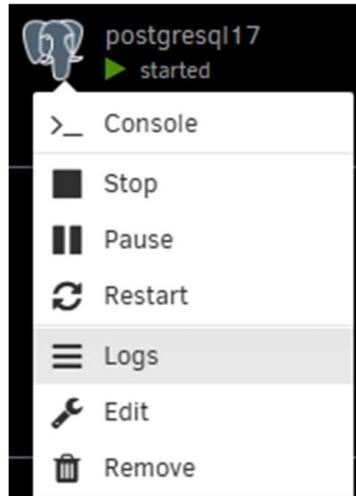


Figura 18. Meniu Docker pentru PostgreSQL

```
server started

/usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*

2025-06-15 06:15:57.616 PDT [49] LOG: received fast shutdown request
waiting for server to shut down....2025-06-15 06:15:57.620 PDT [49] LOG: aborting any active transaction
2025-06-15 06:15:57.623 PDT [49] LOG: background worker "logical replication launcher" (PID 55) exited normally
2025-06-15 06:15:57.626 PDT [50] LOG: shutting down
2025-06-15 06:15:57.628 PDT [50] LOG: checkpoint starting: shutdown immediate
2025-06-15 06:15:57.651 PDT [50] LOG: checkpoint complete: wrote 3 buffers (0.0%); 0 WAL file(s) added, 0 removed, 0 truncated, 0 zeroed
c=0.006 s, total=0.026 s; sync files=2, longest=0.003 s, average=0.003 s; distance=0 kB, estimate=0 kB
2025-06-15 06:15:57.660 PDT [49] LOG: database system is shut down
done
server stopped

PostgreSQL init process complete; ready for start up.
```

Figura 19. Log pentru instalare cu succes

Urmează apoi pasul al doilea, căutarea aplicației ThingsBoard-Postgres din același magazin de aplicații. Este important să alegem exact varianta „Postgres”, deoarece ThingsBoard mai oferă și imagini bazate pe Cassandra sau alte baze de date. Imediat după instalare, putem verifica logurile și vom observa procesul de inițializare. Spre deosebire de PostgreSQL, aici portul poate fi personalizat, este obligatoriu să nu folosim unul ocupat deja. În acest caz, a fost ales portul 9090, fiind liber și ușor de reținut.

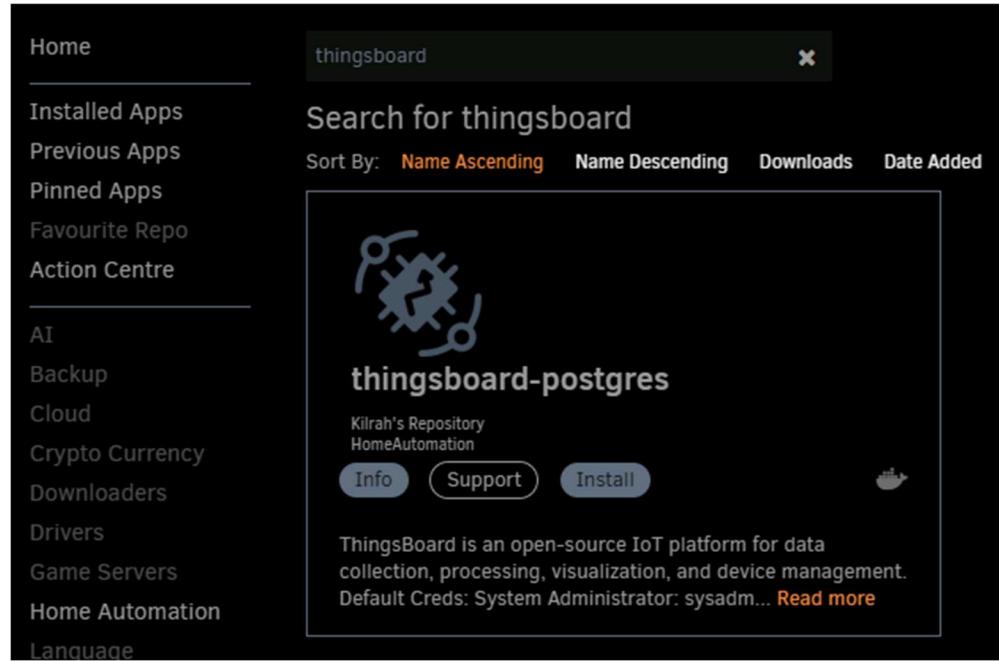


Figura 20. Instalare ThingsBoard

În continuare, dacă instalarea este cu succes, se pot vizualiza mesajele trimise de către server la interfața utilizatorului. Acesta ar trebui să indice faptul că instalarea a avut loc cu succes.

```

Pulling image: thingsboard/tb-postgres:latest
IMAGE ID [1953355631]: Pulling from thingsboard/tb-postgres.
IMAGE ID [6e999acdb790]: Pulling fs layer. Downloading 100% of 27 MB. Verifying Checksum. Download complete. Extracting. Pull complete.
IMAGE ID [db9a34fcba41]: Pulling fs layer. Downloading 100% of 524 KB. Verifying Checksum. Download complete. Extracting. Pull complete.
IMAGE ID [8782fdde02a]: Pulling fs layer. Downloading 100% of 230 MB. Verifying Checksum. Download complete. Extracting. Pull complete.
IMAGE ID [82b18b347f46]: Pulling fs layer. Downloading 100% of 307 MB. Verifying Checksum. Download complete. Extracting. Pull complete.
IMAGE ID [379c71040e7a]: Pulling fs layer. Downloading 100% of 388 MB. Verifying Checksum. Download complete. Extracting. Pull complete.
Status: Downloaded newer image for thingsboard/tb-postgres:latest

TOTAL DATA PULLED: 952 MB

Command execution
docker run
-d
--name='thingsboard-postgres'
--net='bridge'
--pids-limit 2048
-e TZ="America/Los_Angeles"
-e HOST_OS="Unraid"
-e HOST_HOSTNAME="Tower"
-e HOST_CONTAINERNAME="thingsboard-postgres"
-l net.unraid.docker.managed=dockerman
-l net.unraid.docker.webui='http://[IP]:[PORT:9090]'
-l net.unraid.docker.icon='https://github.com/kilrah/unraid-docker-templates/raw/main/icons/thingsboard-postgres.png'
-p '9090:9090/tcp'
-p '1883:1883/tcp'
-p '7070:7070/tcp'
-p '5683-5688:5683-5688/udp'
-v '/mnt/user/appdata/thingsboard-postgres/data':'/data':'rw'
-v '/mnt/user/appdata/thingsboard-postgres/logs':'/var/log/thingsboard':'rw' 'thingsboard/tb-postgres'
219fc1ffeb97d4259910bbe85369825c566e8ff76d0f72f8bf478e007cac75f

The command finished successfully!

```

Figura 21. Log Instalare ThingsBoard

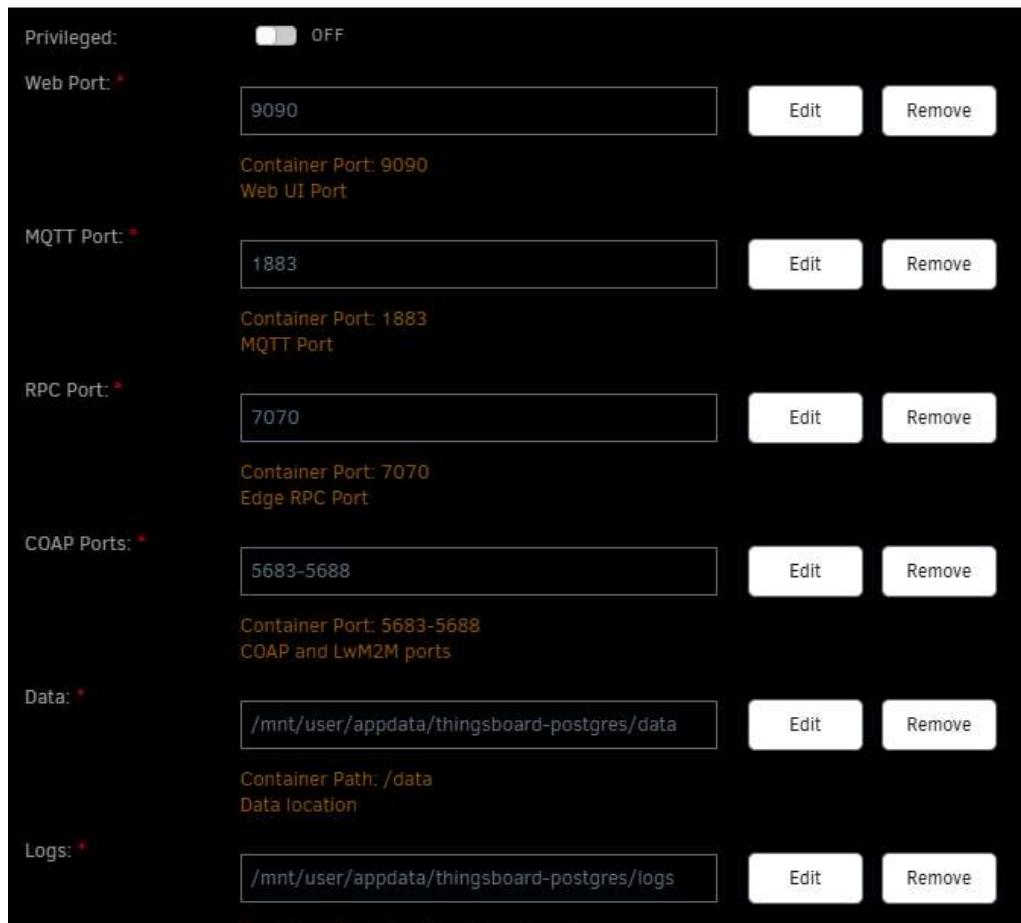


Figura 22. Configurație ThingsBoard

Există însă o particularitate la ThingsBoard. Imediat după ce containerul este pornit prima dată, el se va opri, din cauza permisiunilor implicate ale sistemului de a accesa fișierele. Nu este o eroare, este de fapt o etapă necesară. Pentru a o rezolva, vom deschide terminalul Unraid și vom executa o comandă care schimbă drepturile pe directorul aplicației. Pașii necesari sunt următorii:

- Containerul se rulează, acesta se va opri din cauza permisiunilor.
- Se deschide terminalul Unraid și rulează comanda:

„chown -R 799:799 /mnt/user/appdata/thingsboard-postgres”

- Se repornește containerul. După câteva minute va fi gata de utilizare.

```
root@Tower: ~ | /bin/bash --login (Tower) - Google Chrome
⚠ Not secure 192.168.1.205/webterminal/ttyd/
root@Tower:~# chown -R 799:799 /mnt/user/appdata/thingsboard-postgres
root@Tower:~#
```

Figura 23. Terminal Unraid

```

=====
:: ThingsBoard ::      (v4.0.1)
=====

Starting ThingsBoard Installation...
Installing DataBase schema for entities...
Installing SQL DataBase schema part: schema-entities.sql
Installing SQL DataBase schema indexes part: schema-entities-idx.sql
Installing SQL DataBase schema PostgreSQL specific indexes part: schema-entities-idx-psql-addon.sql
Installing SQL DataBase schema views and functions: schema-views-and-functions.sql
Successfully executed query: DROP VIEW IF EXISTS device_info_view CASCADE;
Successfully executed query: CREATE OR REPLACE VIEW device_info_view AS SELECT * FROM device_info_active_attribute_view;
Installing DataBase schema for timeseries...
Installing SQL DataBase schema part: schema-ts-psql.sql
Successfully executed query: CREATE TABLE IF NOT EXISTS ts_kv_indefinite PARTITION OF ts_kv DEFAULT;
Loading system data...
Creating JWT admin settings...
Loading system widgets
Loading system SCADA symbols
Creating default notification configs for system admin
Creating default notification configs for all tenants
[...]

```

Figura 24. Log instalare ThingsBoard

Dacă totul a decurs corect, logurile ThingsBoard vor afișa procesele de inițializare, crearea tabelelor, importul de date și pornirea serverului HTTP. La final, accesăm ThingsBoard direct din browser, folosind IP-ul serverului local, urmat de portul specificat – în cazul nostru: **192.168.1.205:9090**.

- **System Administrator:** sysadmin@thingsboard.org / sysadmin
- **Tenant Administrator:** tenant@thingsboard.org / tenant
- **Customer User:** customer@thingsboard.org / customer

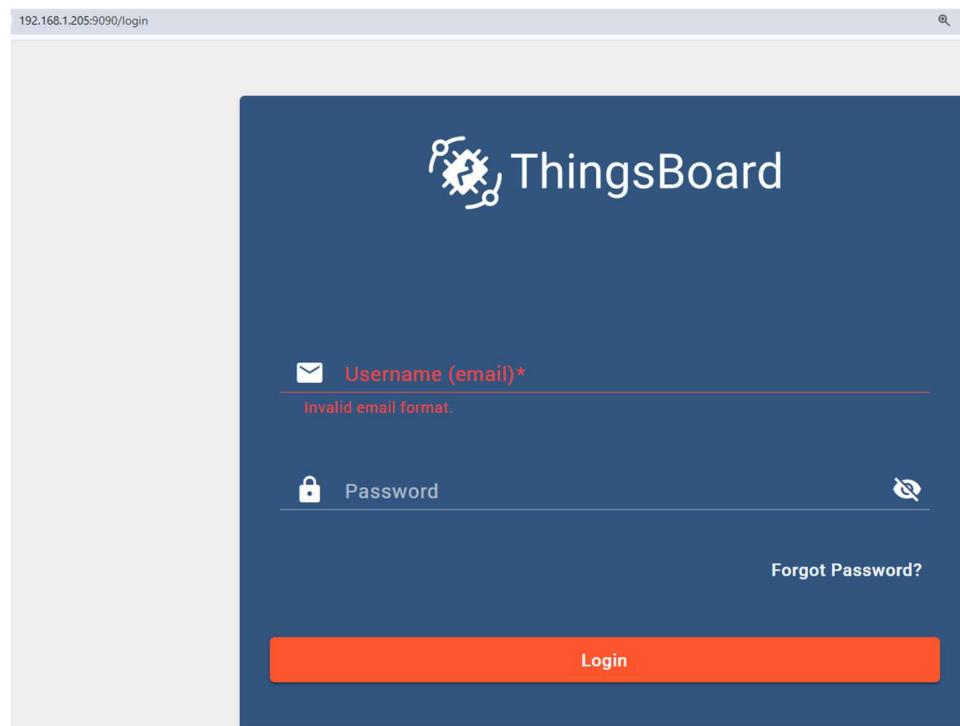


Figura 25. Pagină de logare

Pagina principală ThingsBoard se va încărca, oferindu-ne interfața de autentificare. Datele implicate de autentificare sunt deja stabilite și pot fi schimbate ulterior din interfață:

The screenshot shows the ThingsBoard main dashboard with the following components:

- Left sidebar:** A dark sidebar with a navigation menu including Home, Alarms, Dashboards, Entities, Profiles, Customers, Rule chains, Edge management, Advanced features, Resources, Notification center, Mobile center, API usage, Settings, and Security.
- Top header:** Shows the logo, "Home", the user name "tenant@thingsboard.org Tenant administrator", and a settings icon.
- Dashboard cards:**
 - Devices:** Inactive: 9, Active: 0, Total: 9.
 - Alarms:** Critical: 0, Assigned to me: 0, Total: 0.
 - Activity:** History - last 30 days (0 May 17 to 0 Jun 13).
 - Documentation:** Getting started, Rule engine, API, Device profiles.
 - Usage:** Entities (Devices: 9 / 00, Assets: 0 / 00, Users: 5 / 00, Dashboards: 4 / 00, Customers: 3 / 00), API calls.
- Get started:** A section with numbered steps:
 - Create device
 - Connect device
 - Create dashboard
 - Configure alarm rules
 - Create alarm
 - Create customer and assign dashboard
 It also includes links to "How to create Device" and "Devices".
- Connect mobile app:** Includes QR codes for App Store and Google Play, and download links.

Figura 26. Pagina principală ThingsBoard

Astfel, ThingsBoard este acum complet funcțional, rulează izolat ca un container în Unraid și este gata pentru a fi utilizat în rețea, fie local, fie accesat de dispozitive IoT care trimit date prin MQTT, HTTP sau CoAP. Interfața este accesibilă din LAN și poate fi extinsă și pentru acces extern, dacă se configurează în router redirecționarea porturilor sau se folosește un reverse proxy.

Utilizarea Thingsboard

Adăugarea unui dispozitiv

Odată ce platforma ThingsBoard este instalată și funcțională în containerul nostru din Unraid, următorul pas firesc este adăugarea unui dispozitiv, fie real, fie de test, pentru a simula sau colecta date. Acest lucru este important în contextul unui proiect IoT, unde fiecare senzor sau microcontroler (precum un ESP32 ce devine client) are nevoie de o identificare și o autentificare proprie.

Pentru a realiza acest lucru, accesăm interfața grafică a ThingsBoard și, de pe prima pagină, selectăm opțiunea *Entities*, aflată în meniul lateral. Acest tab este dedicat gestionării entităților din platformă: dispozitive, gateway-uri, clienți, utilizatori etc. Din lista de opțiuni, alegem *Devices*, adică dispozitivele propriu-zise ce urmează să fie înregistrate în sistem.

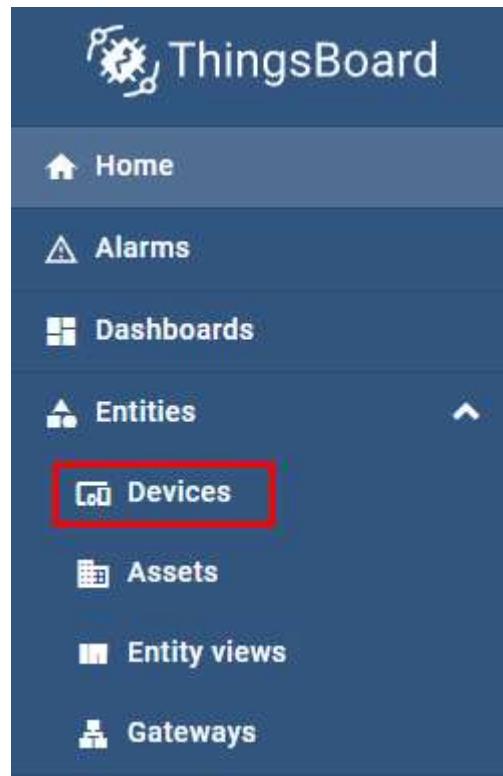


Figura 27. Meniu Principal

În momentul în care se deschide noua pagină, interfața este momentan goală, nu există niciun dispozitiv adăugat. În colțul din dreapta sus, lângă bara de căutare, vom observa un mic simbol „plus” (+). Acesta este butonul care ne permite să adăugăm manual un nou dispozitiv în ThingsBoard.

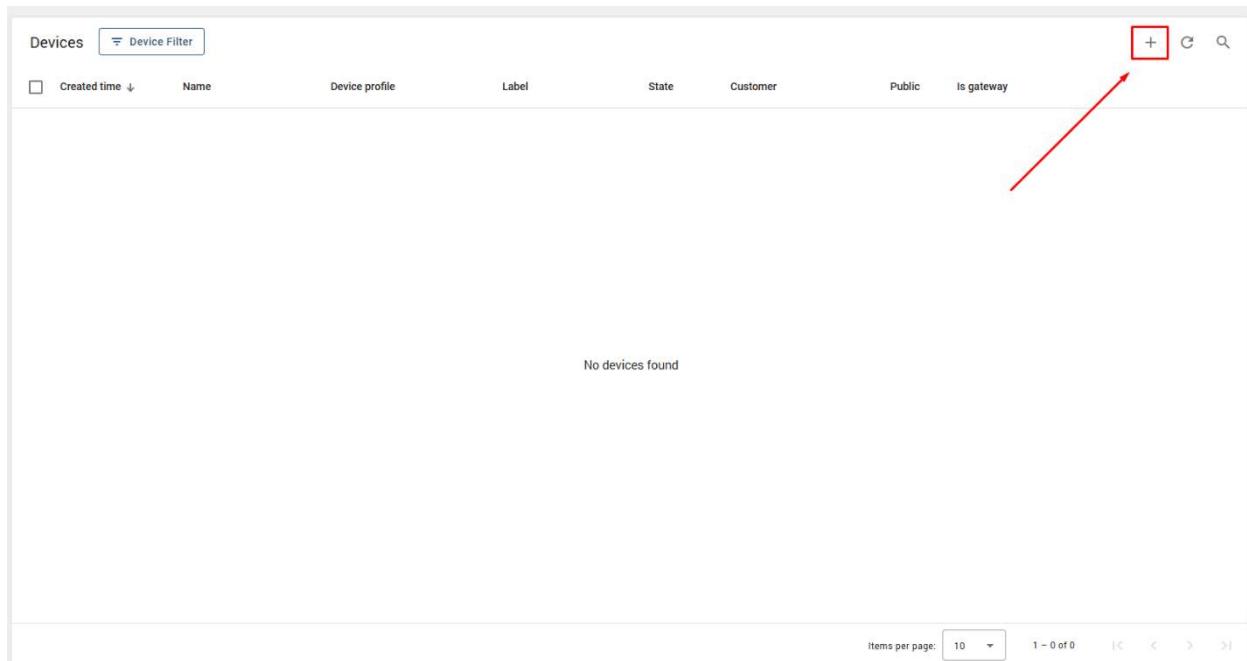


Figura 28. Adăugare Dispozitive

După ce facem click pe acel simbol, selectăm opțiunea *Add new device*. Se va deschide o fereastră de configurare care ne oferă o serie de câmpuri optionale și obligatorii pentru identificarea dispozitivului.

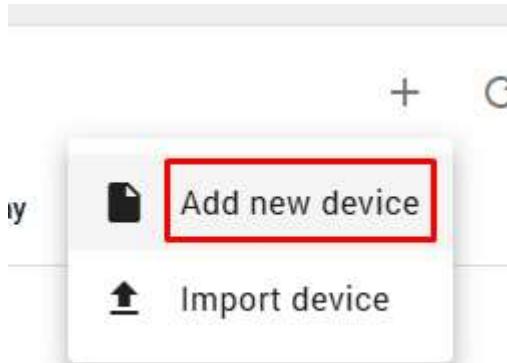


Figura 29. Adăugare Dispozitive

În acest meniu de configurare, putem completa următoarele:

- Numele dispozitivului – câmp esențial.
- Etichete (tags) – utile pentru organizare ulterioară.
- Atribuirea la un anumit profil de client, dacă există.
- Marcare ca gateway principal, dacă este cazul.
- Atribuirea la un utilizator/client specific.
- Adăugarea unei descrieri detaliate pentru claritate.

Pentru exemplul nostru, am completat doar numele, pe care l-am setat ca ESP32-Test, și am apăsat butonul Add. Restul opțiunilor sunt rezervate scenariilor avansate sau configurațiilor bazate pe şablonane (template-uri) pe care le putem crea și salva separat.

- Tab accesat: Entities > Devices
- Creare dispozitiv: click pe (+), apoi Add new device
- Câmp completat: Name = ESP32-Test
- Opțiuni avansate disponibile: client profile, gateway, descriere

Add new device

1 Device details 2 Credentials
Optional

Name*
ESP32-Test

Label

Device profile*
default

Is gateway

Assign to customer

Description

Figura 30. Adăugare Dispozitive

Imediat după confirmare, ThingsBoard ne notifică faptul că dispozitivul a fost creat cu succes și că este pregătit pentru testare. La acest moment, putem verifica conectivitatea chiar și fără un dispozitiv fizic, folosind comenzi în terminal sau Command Prompt pentru a simula trimiterea unor date. Este un pas optional, util pentru validarea funcționalității de bază a platformei.

```
root@Tower:~ | /bin/bash --login (Tower) - Google Chrome
Not secure 192.168.1.205/webterminal/ttyd

root@Tower:~# docker run --rm -it thingsboard/mosquitto-clients mosquitto_pub -d -q 1 -h 192.168.1.205 -p 1883 -t v1/devices/me/telemetry -m "temperature:25"
"7pZUxQjhck15MHLf1Tx" -m "temperature:25"
Unable to find image 'thingsboard/mosquitto-clients:latest' locally
latest: Pulling from thingsboard/mosquitto-clients
29291e31a76a: Pull complete
1530d01e76fd: Pull complete
ab8c273c1250: Pull complete
Digest: sha256:7fd749de80ad411ef9f6d010e9ce4b0cb85c863c63748b7e3827ba577d810f58
Status: Downloaded newer image for thingsboard/mosquitto-clients:latest
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q1, r0, m1, 'v1/devices/me/telemetry', ... (16 bytes))
Client (null) received PUBACK (Mid: 1, RC:0)
Client (null) sending DISCONNECT
root@Tower:~#
```

Figura 31. Terminal Unraid

ESP32-Test

Device details

Details Attributes Latest telemetry Calculated fields Alarms

Telemetry

Last update time	Key ↑	Value
2025-06-15 17:53:04	temperature	25

Figura 32. Telemetrie ThingsBoard

Device created. Let's check connectivity!

HTTP MQTT CoAP

Use the following instructions for sending telemetry on behalf of the device using shell

Windows macOS Linux

Install necessary client tools
Starting Windows 10 b17063, cURL is available by default

Execute the following command

```
curl -v -X POST http://192.168.1.205:8080/api/v1/ptcsEBdEOj85DnH32m
```

State Inactive

Latest telemetry

Time	Key	Value
No latest telemetry		

Do not show again Close

Figura 33. Meniu MQTT în ThingsBoard

În continuare, închidem notificarea cu Close și selectăm noul dispozitiv din listă (în cazul nostru: ESP32-Test). În partea dreaptă a interfeței ThingsBoard se va deschide un tab nou, unde avem acces la detalii, setări și chei de acces.

	Created time ↓	Name	Device profile
<input type="checkbox"/>	2025-06-15 17:23:46	ESP32-Test	default

Figura 34. Dispozitive în ThingsBoard

Unul dintre cele mai importante elemente din acest tab este Access Token-ul, adică „cheia” de autentificare pe care dispozitivul o va folosi pentru a se conecta și comunica cu platforma. Vom face click pe butonul Copy Access Token, iar ThingsBoard ne va informa că tokenul a fost copiat cu succes în clipboard.

Această cheie trebuie adăugată ulterior în codul sursă al microcontrolerului (în cazul nostru, un ESP32). Fără acest token, ThingsBoard nu va permite conexiunea dispozitivului, iar datele trimise de acesta vor fi ignorate.

- Click pe dispozitivul ESP32-Test
- Se deschide tab-ul de detalii
- Apăsăm pe Copy Access Token
- Tokenul este folosit în codul ESP32 pentru conectare și comunicare

The screenshot shows the 'ESP32-Test' device details page. At the top, there's a navigation bar with tabs: Details, Attributes, Latest telemetry, Calculated fields, Alarms, Events, Relations, Audit logs, Version control, and a settings icon. Below the navigation bar, there are several buttons: Open details page, Make device public, Assign to customer, Manage credentials, Check connectivity, Delete device, Copy device Id, and the highlighted Copy access token. The main content area contains fields for Name (ESP32-Test), Device profile (default), Label, Assigned firmware, and Assigned software.

Figura 35. Dispozitive în ThingsBoard

Device access token has been copied to clipboard [Close](#)

Figura 36. Confirmarea copierii codului de acces

Pe scurt, tokenul este cheia unică de acces, folosită de dispozitivul ESP32 pentru a trimite date către server, dar și pentru a primi comenzi de tip RPC (Remote Procedure Call). Acesta este mecanismul prin care platforma ThingsBoard controlează sau monitorizează dispozitivele în timp real, într-un mod securizat. Următorul pas este să adăugăm această cheie în codul nostru care urmează a fi scris pe ESP32.

```
9
10 #include <Arduino_MQTT_Client.h>
11 #include <Server_Side_RPC.h>
12 #include <ThingsBoard.h>
13
14 #define ENCRYPTED false
15 #define MAX_ADC_VALUES 1000 // Max expected number of values
16 #define JSON_OVERHEAD 32 // Rough buffer overhead
17 StaticJsonDocument<JSON_ARRAY_SIZE(MAX_ADC_VALUES) + JSON_OVERHEAD> doc;
18
19 constexpr char WIFI_SSID[] = "IoT Server";
20 constexpr char WIFI_PASSWORD[] = "Floridemar3";
21 constexpr char THINGSBOARD_SERVER[] = "192.168.1.205";
22 constexpr uint16_t THINGSBOARD_PORT = ENCRYPTED ? 8883U : 1883U;
23 constexpr char TOKEN[] = "FU2288y6XZqCTvjpYfN"; -----^
24 constexpr uint16_t MAX_MESSAGE_SEND_SIZE = 256U;
25 constexpr uint16_t MAX_MESSAGE_RECEIVE_SIZE = 256U;
26 constexpr uint32_t SERIAL_DEBUG_BAUD = 115200U;
27
28 constexpr char RPC_JSON_METHOD[] = "example_json";
29 constexpr char RPC_TEMPERATURE_METHOD[] = "example_set_temperature";
30 constexpr char RPC_SWITCH_METHOD[] = "example_set_switch";
31 constexpr char RPC_TEMPERATURE_KEY[] = "temp";
32 constexpr char RPC_SWITCH_KEY[] = "switch";
33 constexpr uint8_t MAX_RPC_SUBSCRIPTIONS = 3U;
34 constexpr uint8_t MAX_RPC_RESPONSE = 5U;
35
36 #if ENCRYPTED
```

Figura 37. Plasarea codului de acces în codul sursă

Utilizarea Dashboard-ului

Crearea și personalizarea unui Dashboard în ThingsBoard

După ce am adăugat cu succes un dispozitiv (precum ESP32-Test) în platforma ThingsBoard, următorul pas este reprezentarea vizuală a datelor primite. Pentru aceasta, vom crea un Dashboard, un panou interactiv ce permite afișarea grafică a parametrilor înregistrări de dispozitivele noastre, precum și controlul acestora în timp real prin comenzi RPC. Pentru început, din meniul principal al ThingsBoard, vom accesa secțiunea Dashboards. Aici vom observa că platforma vine preconfigurată cu câteva dashboard-uri de test sau exemplu. Totuși, pentru scopurile noastre, vom crea unul nou, complet personalizat.



Figura 38. Meniu Thingsboard

În colțul din dreapta sus al paginii, lângă iconițele de refresh și căutare, vom regăsi simbolul clasic „plus” (+). Apăsăm pe acesta și alegem opțiunea Create new dashboard.

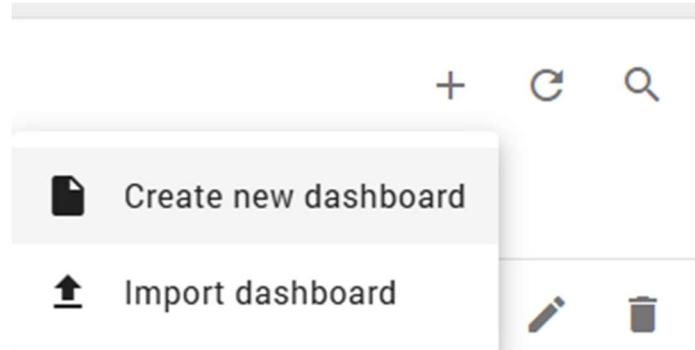


Figura 39. Creare Dashboard

Se va deschide o fereastră unde putem seta următoarele:

- **Titlul** dashboard-ului – un nume sugestiv, care să reflecte conținutul său.
- **Descrierea** – opțională, dar utilă pentru claritate.
- **Asocierea cu un client** – în cazul în care platforma este utilizată în mod multi-user, putem asigna dashboard-ul unui utilizator sau client specific.
- **Personalizarea vizuală** – ThingsBoard permite adăugarea de imagini sau logo-uri proprii pentru un aspect profesional.

După completarea acestor informații, apăsăm butonul Add. Se va încărca o pagină complet nouă, momentan goală, care reprezintă dashboard-ul nostru neconfigurat.

A screenshot of the 'Add dashboard' configuration dialog. The title bar says 'Add dashboard'. The main area has several sections:

- Title***: A text input field containing 'TestESP'.
- Description**: A text input field that is currently empty.
- Assigned customers**: A section that is currently empty.
- Mobile application settings**: A section with a toggle switch labeled 'Hide dashboard in mobile application' which is turned off (grayed out).
- Dashboard order in mobile application**: A section that is currently empty.
- Dashboard image**: A section with three buttons:
 - 'No image selected'
 - 'Browse from gallery'
 - 'Set link'

At the bottom right are 'Cancel' and 'Add' buttons.

Figura 40. Creare Dashboard

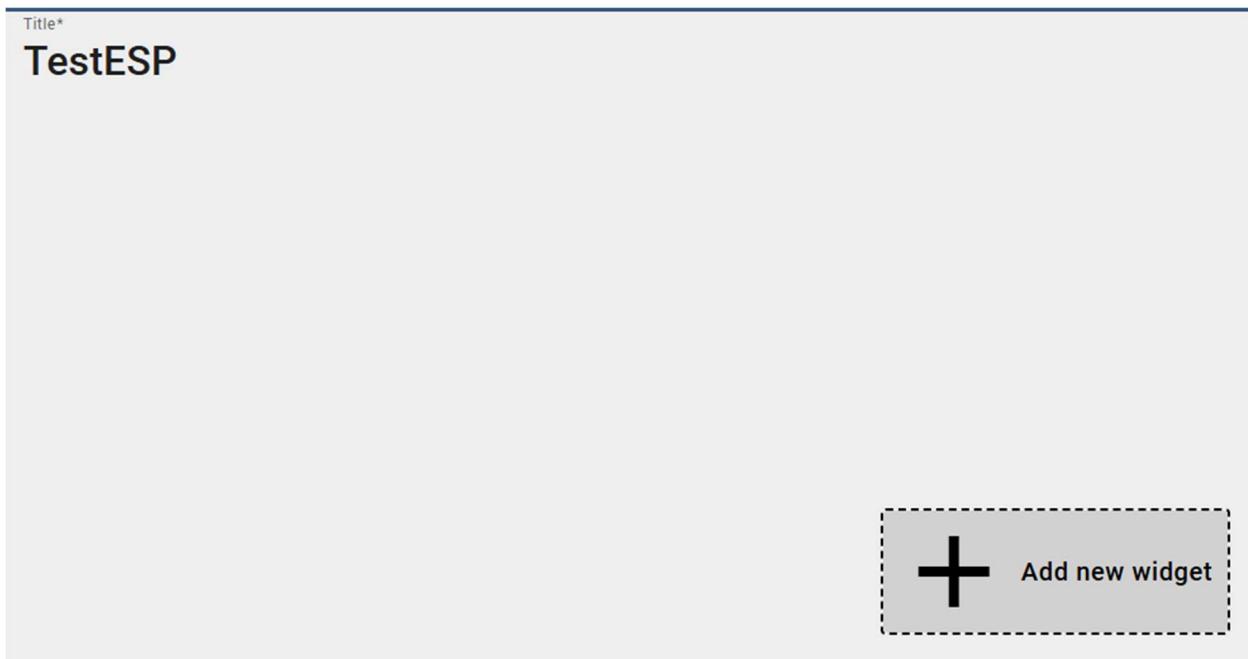


Figura 41. Creare Dashboard

Adăugarea primului widget – reprezentarea grafică a datelor

Pentru a începe popularea dashboard-ului cu informații, vom apăsa în centrul paginii pe butonul Add new widget. Această acțiune va deschide în partea dreaptă o nouă interfață, un tab lateral, unde avem acces la o vastă colecție de biblioteci de widget-uri.

Acestea sunt organizate tematic, pentru a facilita alegerea în funcție de scop:

- **Charts** (grafice) – pentru reprezentarea valorilor numerice în timp.
- **Cards** – pentru afișare simplă, cu pictograme.
- **Alarms** – pentru notificări de tip alertă.
- **Tables** – tabele dinamice cu valori.
- **Maps** – pentru urmărire geografică a dispozitivelor.
- **Analog Gauges** – indicatoare circulare (tip ceas).
- **Buttons și Control widgets** – utile pentru trimitera de comenzi (ex: aprindere LED).
- **Status indicators** – semnalizări ON/OFF.
- **SCADA symbols** – simboluri industriale.
- **Navigation și menus** – pentru organizarea mai complexă.
- **Air Quality, Digital Displays**, etc.
- **Cel mai important – GPIO Widgets**, folosite pentru control de tip RPC (comenzi la distanță către ESP32 sau alte dispozitive).

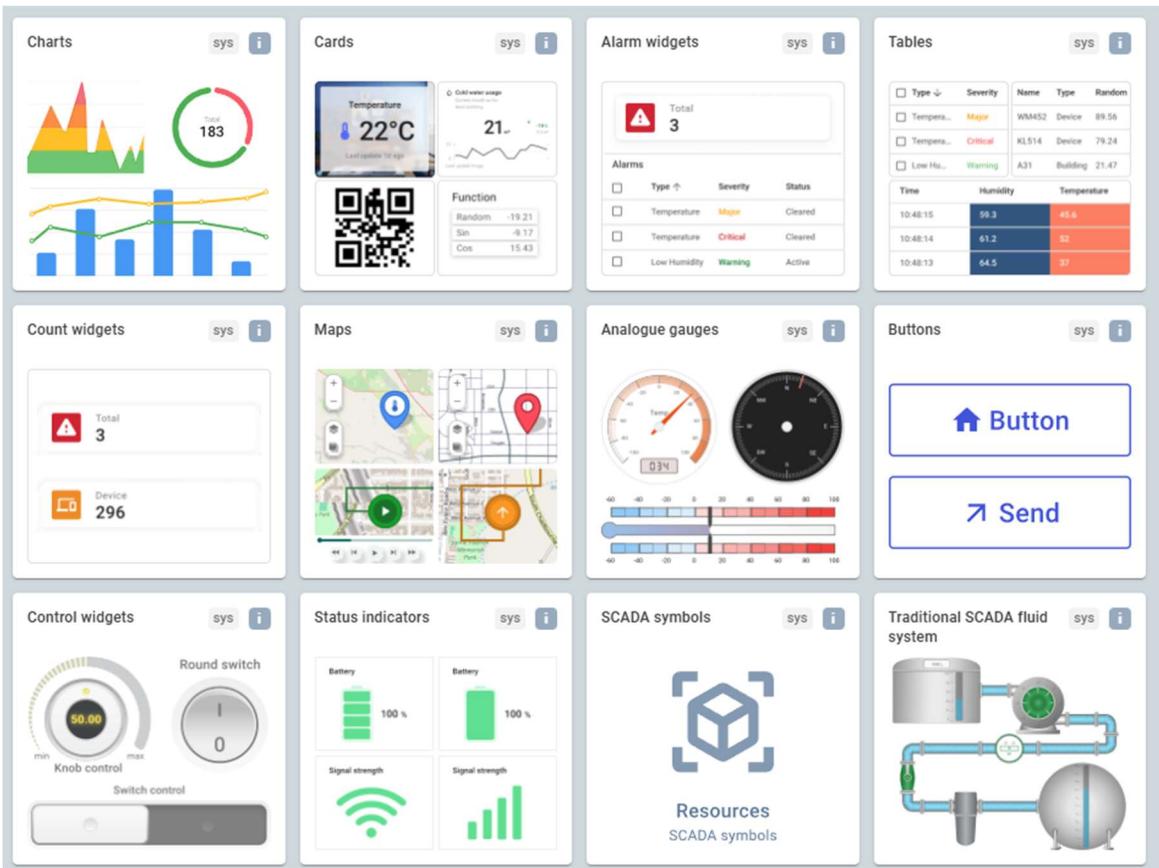


Figura 42. Creare Dashboard - grafice

Pentru demonstrație, vom selecta ceva simplu: din categoria Charts, alegem widget-ul Line Chart, adică un grafic de tip linie, unde valorile datelor achiziționate vor fi reprezentate în domeniul timp, ideal pentru a vizualiza variația unui parametru precum temperatura. Facem click pe imaginea corespunzătoare, iar widget-ul este adăugat instant pe dashboard-ul nostru.

Configurarea sursei de date pentru widget

Următorul pas este personalizarea graficului. Ne vom poziționa cu cursorul în partea din dreapta sus a graficului adăugat, iar acolo va apărea un mic creion (simbol de editare). Facem click pe acesta pentru a deschide meniul de configurare.

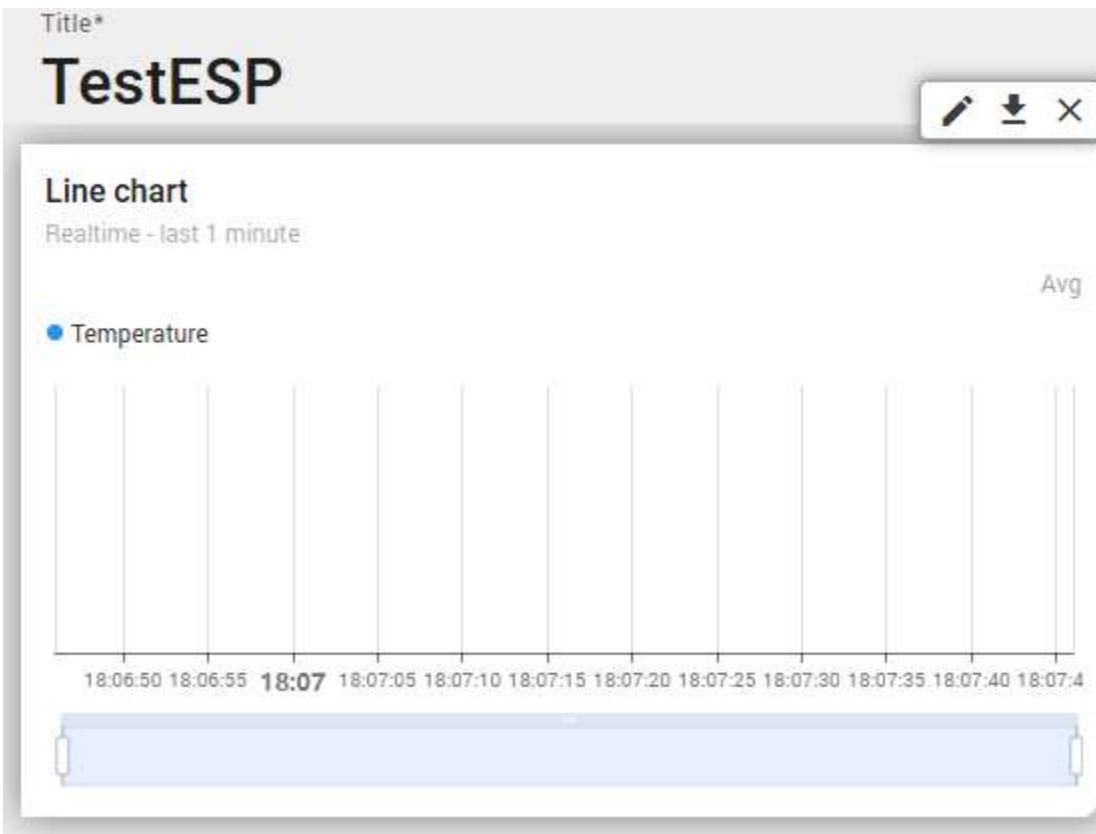


Figura 43. Creare Dashboard

Aici se poate selecta sursa de date (Data source). Se vor regăsi toate dispozitivele înregistrate anterior în tabul Entities, Devices. Se va selecta dispozitivul (ESP32-Test) din listă.

Din același meniu de editare se poate regla intervalul de timp pentru afișarea datelor, culoarea graficului și numărul de puncte utilizate în reprezentare, oferind astfel posibilitatea de adaptare după preferințe.

Totuși, pe măsură ce se mărește numărul de puncte, se va observa apariția unei probleme ce este legată de întârzierea în a afișa valorile, deoarece întârzierea conexiunii prin Wi-Fi crește proporțional cu distanța față de router. În testele realizate, s-a optat pentru o frecvență de eșantionare de 100 Hz, folosind un semnal sinusoidal, condiții sub care sistemul s-a comportat în parametri optimi.

Acest lucru poate fi evitat dacă se folosește un mediu de comunicare care este pe fir, și nu tehnologie wireless.

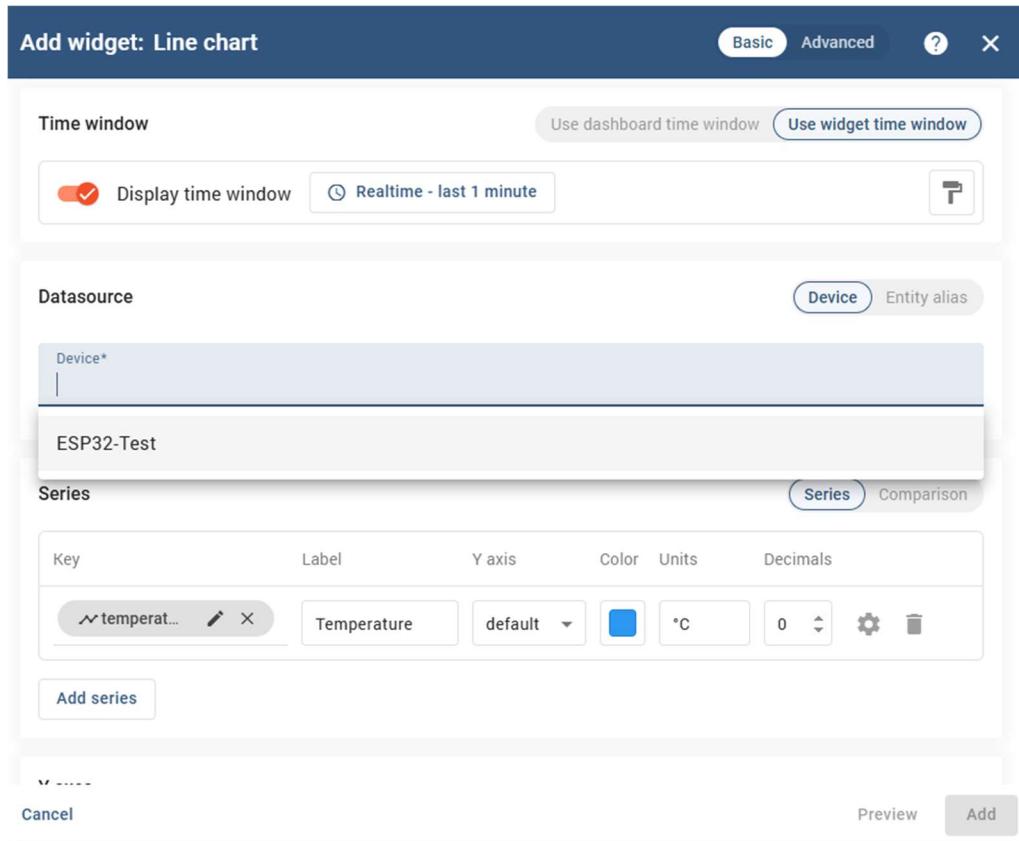


Figura 44. Editare Dashboard

După ce am selectat dispozitivul, vom apăsa din nou pe creionul de lângă pictograma de temperatură, ceea ce va deschide o nouă fereastră de editare pentru parametrul afișat. La finalizarea acestor setări, apăsăm pe Save pentru a salva modificările și a reveni la dashboard.

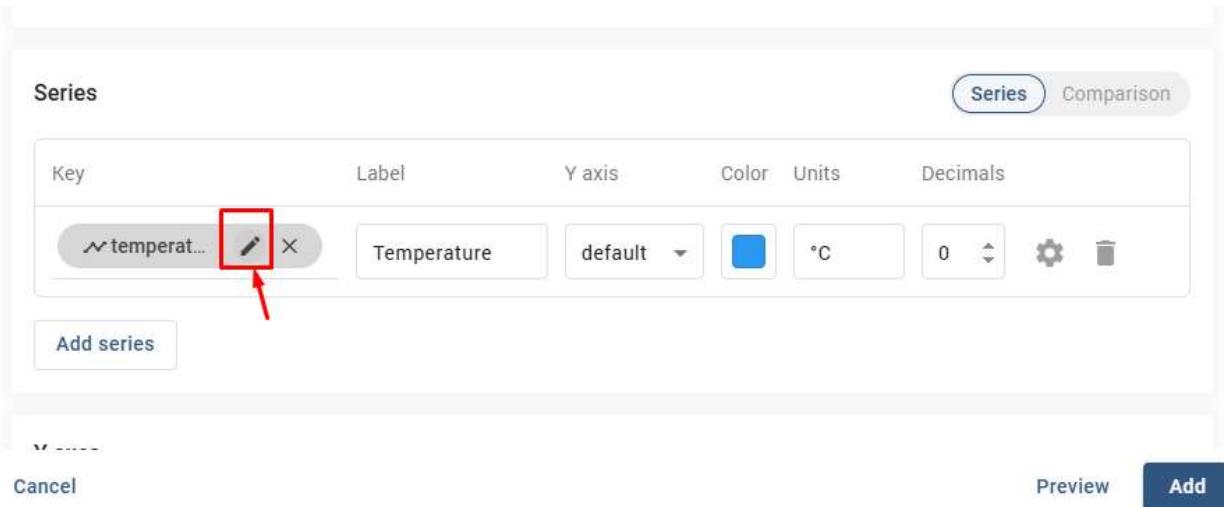


Figura 45. Editare Dashboard

În acest meniu sunt prezente următoarele câmpuri:

- **Key** – reprezintă exact denumirea variabilei transmise de ESP32 prin MQTT (de exemplu: temperature, humidity, voltage, etc.). Această denumire trebuie să fie identică cu cea declarată în codul sursă al ESP32.
 - **Label** – este eticheta vizibilă în grafic. Poate fi personalizată pentru claritate (ex: „Temperatură ambientală”).
 - **Unitate de măsură** – putem introduce °C, %, V etc.
 - **Număr de zecimale** – câte cifre să fie afișate după virgulă.
 - **Culoare linie și stil grafic** – pentru personalizare vizuală.
- Dacă se va folosi un alt parametru, se poate șterge cheia temperature și alege una dintre cele disponibile deja, transmise de dispozitiv.

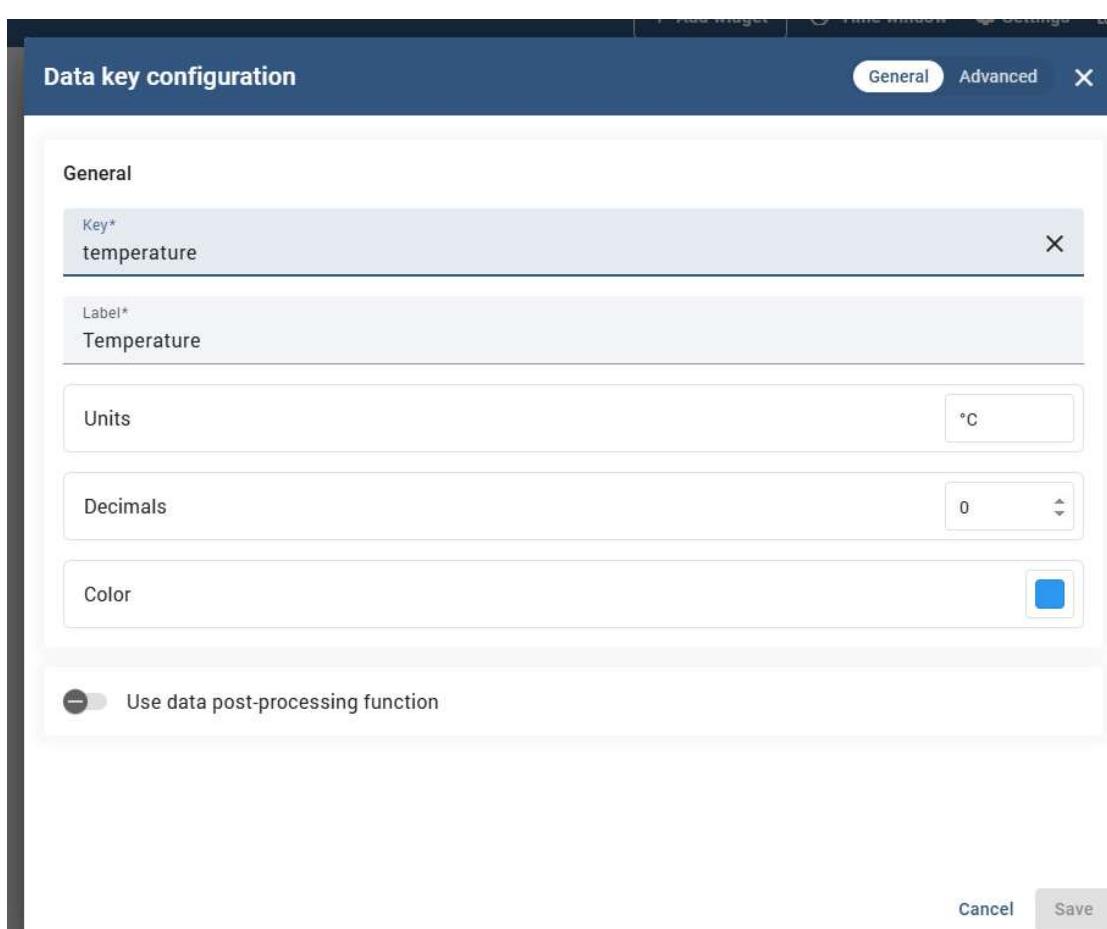


Figura 46. Editare Dashboard

Testarea comportamentului graficului

Pentru a valida funcționalitatea graficului, putem simula transmiterea unor valori MQTT direct din terminalul UNRAID, folosind comenzi corespunzătoare pentru a trimite date către serverul ThingsBoard.

Această simulare ne permite să observăm în timp real cum se actualizează graficul și dacă datele sunt interpretate corect. Vom atașa și o imagine de ecran (screenshot) cu comenzi utilizate pentru trimitera datelor.

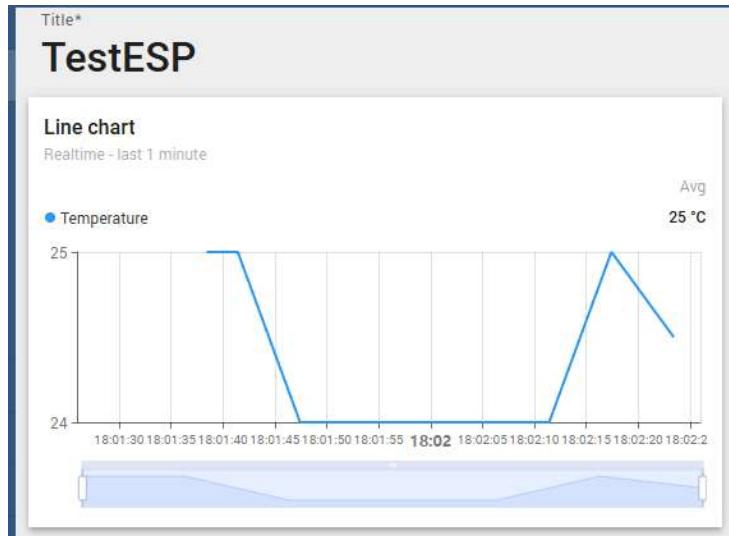


Figura 47. Testare Dashboard

```
root@Tower: ~ | /bin/bash --login (Tower) - Google Chrome
Not secure 192.168.1.205/webterminal/ttyd

x0jhck15MIHLf1Tx" -m "{temperature:25"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q1, r0, m1, 'v1/devices/me/telemetry', ... (16 bytes))
Client (null) received PUBACK (Mid: 1, RC:0)
Client (null) sending DISCONNECT
^[[Aroot@Tower:~#
root@Tower:~# docker run --rm -it thingsboard/mosquitto-clients mosquitto_pub -d -q 1 -h 192.168.1.205 -p 1883 -t v1/devices/me/telemetry -u "7pZU
x0jhck15MIHLf1Tx" -m "{temperature:24"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q1, r0, m1, 'v1/devices/me/telemetry', ... (16 bytes))
Client (null) received PUBACK (Mid: 1, RC:0)
Client (null) sending DISCONNECT
root@Tower:~# docker run --rm -it thingsboard/mosquitto-clients mosquitto_pub -d -q 1 -h 192.168.1.205 -p 1883 -t v1/devices/me/telemetry -u "7pZU
x0jhck15MIHLf1Tx" -m "{temperature:24"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q1, r0, m1, 'v1/devices/me/telemetry', ... (16 bytes))
Client (null) received PUBACK (Mid: 1, RC:0)
Client (null) sending DISCONNECT
root@Tower:~# docker run --rm -it thingsboard/mosquitto-clients mosquitto_pub -d -q 1 -h 192.168.1.205 -p 1883 -t v1/devices/me/telemetry -u "7pZU
x0jhck15MIHLf1Tx" -m "{temperature:24"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q1, r0, m1, 'v1/devices/me/telemetry', ... (16 bytes))
Client (null) received PUBACK (Mid: 1, RC:0)
Client (null) sending DISCONNECT
root@Tower:~# docker run --rm -it thingsboard/mosquitto-clients mosquitto_pub -d -q 1 -h 192.168.1.205 -p 1883 -t v1/devices/me/telemetry -u "7pZU
x0jhck15MIHLf1Tx" -m "{temperature:25"
Client (null) sending CONNECT
Client (null) received CONNACK (0)
```

Figura 48. Testare Dashboard



Figura 49. Meniu Salvare

După verificare, ultimul și cel mai important pas este apăsarea butonului Save, aflat în partea de sus a dashboard-ului.

Atenție: Dacă se părăsește pagina fără a se salvează, tot progresul de configurare va fi pierdut.

4.3. Integrarea protocolului MQTT/RPC în ESP32 și MSP430

4.3.1. Interconectarea cu MSP430 și firmware

Cele două plăcuțe, ESP32 și MSP430 “poartă o conversație” prin UART la 115200 kbps, astfel încât informația circulă rapid și sigur între cele două.



Figura 50. Conexiunea între microcontrolere.

Pașii de execuție sunt următorii:

1. Setarea formei de undă

Din interfața grafică ThingsBoard, utilizatorul poate schimba poziția unui slider cu valori cuprinse între 0 și 4. Fiecare cifră înseamnă o formă de undă pentru DAC.

- **0** = DAC opriți
- **1** = undă sinusoidală
- **2** = undă dreptunghiulară
- **3** = undă de tip dint de fierastrău
- **4** = funcție tangentă

Când slider-ul este apăsat, ThingsBoard trimite un apel RPC la ESP32, iar acesta pregătește un mesaj tip „DAC:X” (unde X va fi o cifră de la 0 la 4).

Arhitectura Software

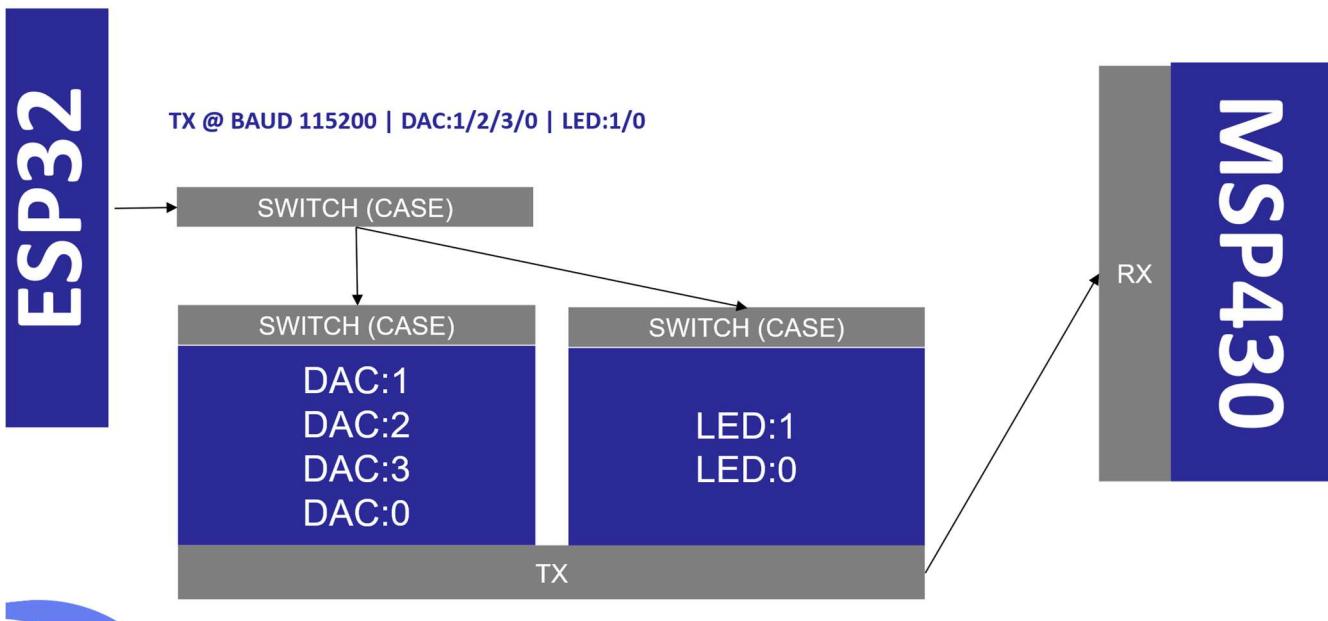


Figura 51. Arhitectura Software. Protocolul de comunicare.

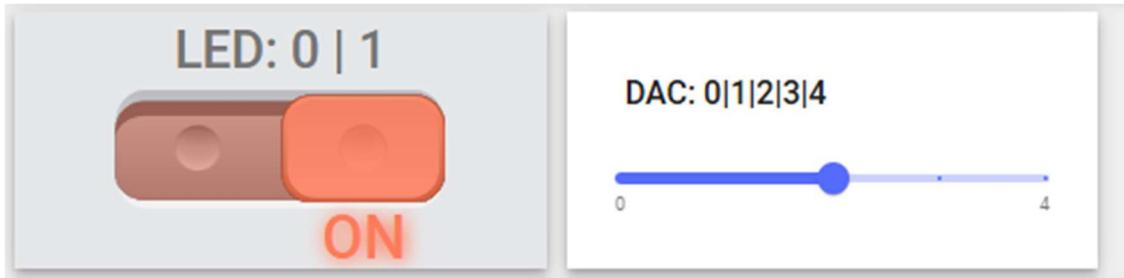


Figura 52. Meniu de control web. Thingsboard.

2. Transmiterea către MSP430

ESP32 primește comanda RPC și trimitе exact textul DAC:X pe linia UART. Până în acest moment, MSP430 s-a aflat în starea de sleep așteptând îintreruperi. MSP430 “se trezește” din modul sleep datorită îintreruperii generate de primirea de date pe UART.

3. Prelucrarea comenzi pe MSP430

În rutina de îintreruperi, MSP430 va lua primele caractere (în acest caz DAC) și intra într-un switch-case:

- Dacă e “DAC”, înseamnă comanda DAC
- Dacă e “LLED”, atunci este o comandă pentru LED-uri. Pentru controlul LED-ului se va trimite comanda LED:0 sau LED:1.
După recunoașterea tipului, switch-case-ul citește ultima cifră (0–4) ceea ce va selecta rapid ramura de cod potrivită, care generează semnalul cerut de utilizator.

4. Achiziția

ADC

Odată semnalul trimis la DAC, MSP430 începe achiziția valorilor de la convertorul analog-digital. Circuitul constă în conectarea fizică a unui fir de la DAC-ul microcontrolerului la ADC-ul acestuia. Această achiziție se face la fiecare 3 secunde și fiecare citire va fi stocată într-un buffer.

5. Împachetarea și trimiterea datelor

După trecerea celor 3 secunde, MSP430 va trimite înapoi pe UART valorile achiziționate de acesta. Mesajul trimis de MSP430 va fi de forma ADC[@v1,@v2,@v3,...];

- „ADC” anunță microcontrolerul ESP32 faptul că urmează valori de ADC
- „[” deschide lista de valori și anunță faptul că va urma un vector
- virgulele separă fiecare măsurare
- „]” închide vectorul
- „;” marchează sfârșitul mesajului

Pentru a nu întâmpina probleme legate de atingerea maximă a memoriei microcontrolerului ESP32 cu un text prea lung, codul de pe acesta este gândit să trimită spre server fiecare valoare imediat ce este urmată de virgula de separare. Când acesta întâmpină caracterul „]” va știi că urmează semnul de final al seriei, iar caracterul „;” confirmă faptul că transmisia s-a încheiat cu succes.

Prin acest flux clar, cele două microcontrolere pot comunica deși sunt sisteme diferite, pot gestiona cererile și apelurile unei alteia și mențin întârzierea la un nivel minim, asigurând o funcționare stabilă în timp a sistemului.

Acest exemplu dovedește faptul că infrastructura propusă poate fi scalată și parametrizată. Această interconectare a două microcontrolere diferite este un lucru des întâlnit în industria electronica întrucât doar așa se poate atinge un raport cost/performanță optim. Mai mult decât atât, există anumite microcontrolere care sunt specializate să execute anumite funcții, cum sunt de exemplu microcontrolerle în timp real, ce pot executa calcule matematice rapid pentru că au hardware-ul necesar pentru a calcula matematic, în timp ce alte microcontrolere prezintă componente hardware ce le permit comunicarea cu rețelele fără fir.

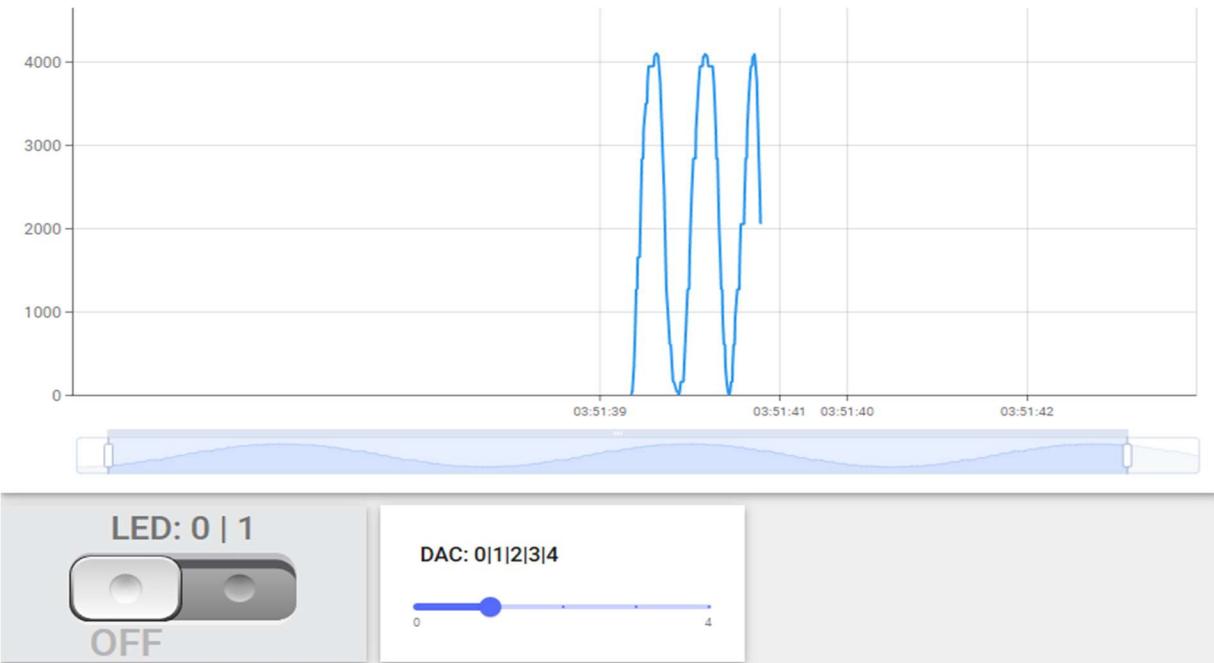


Figura 53. Testare GUI. Citire semnal sinusoidal MSP430 în ESP32

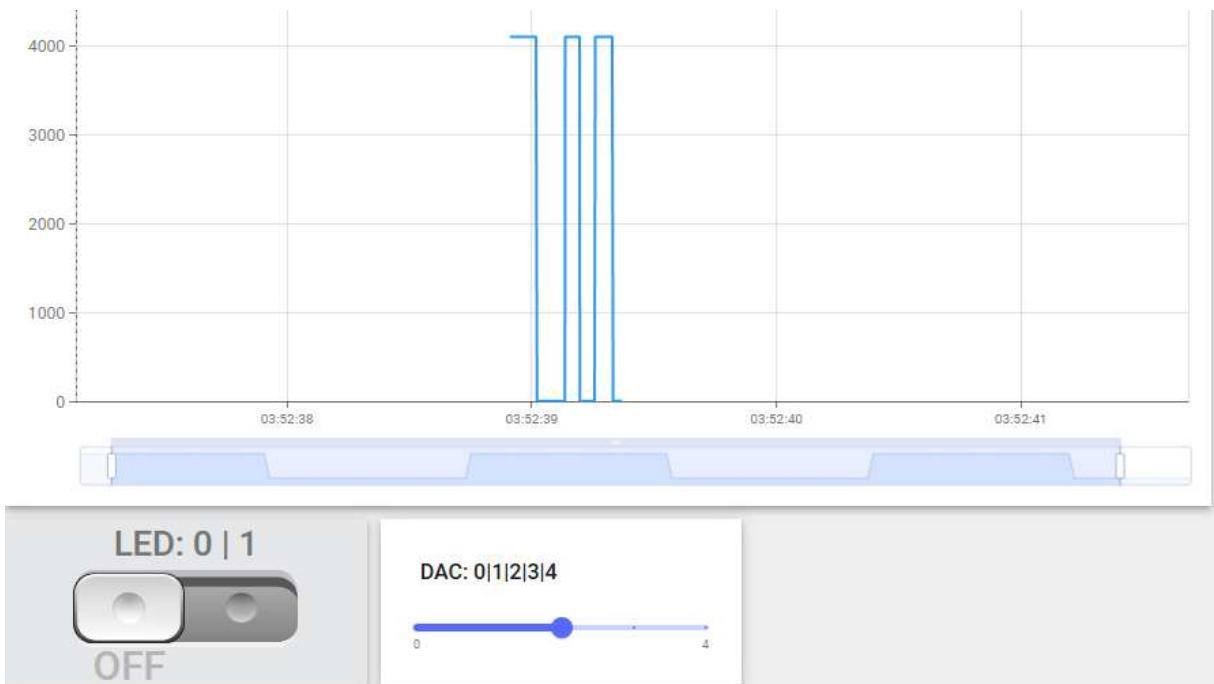


Figura 54. Testare GUI. Citire semnal dreptunghiular MSP430 în ESP32

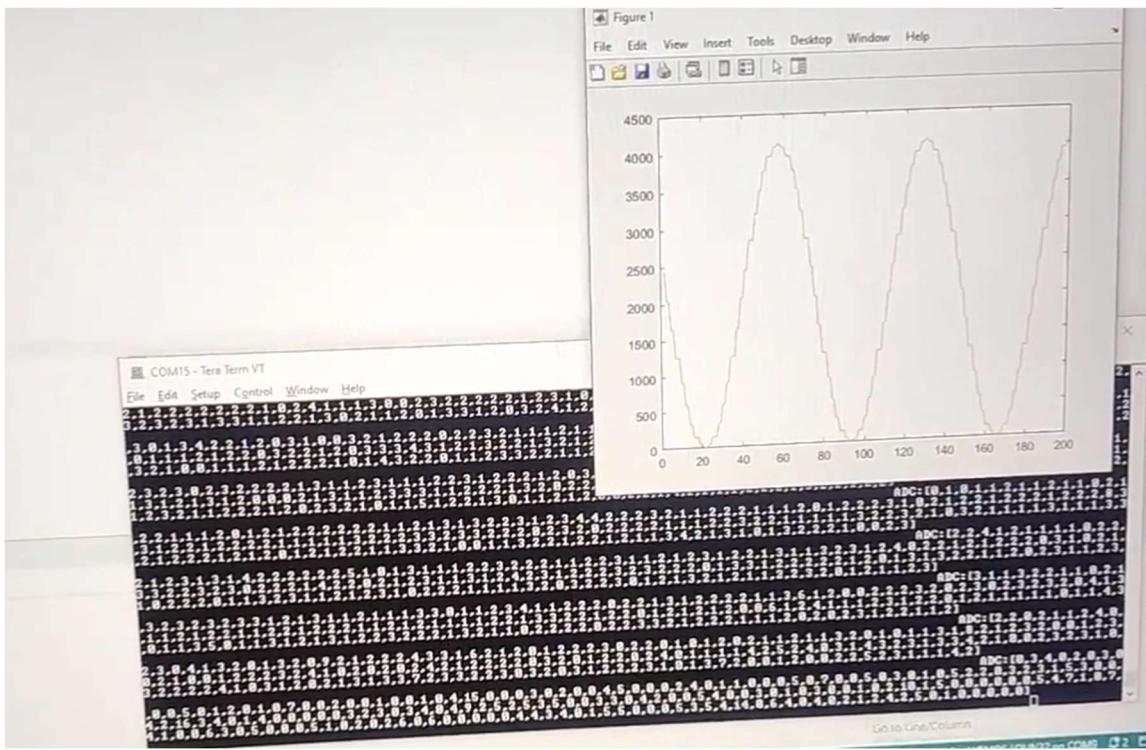


Figura 55. Ascultarea transmisiei UART. Validarea vectorului prin MATLAB.

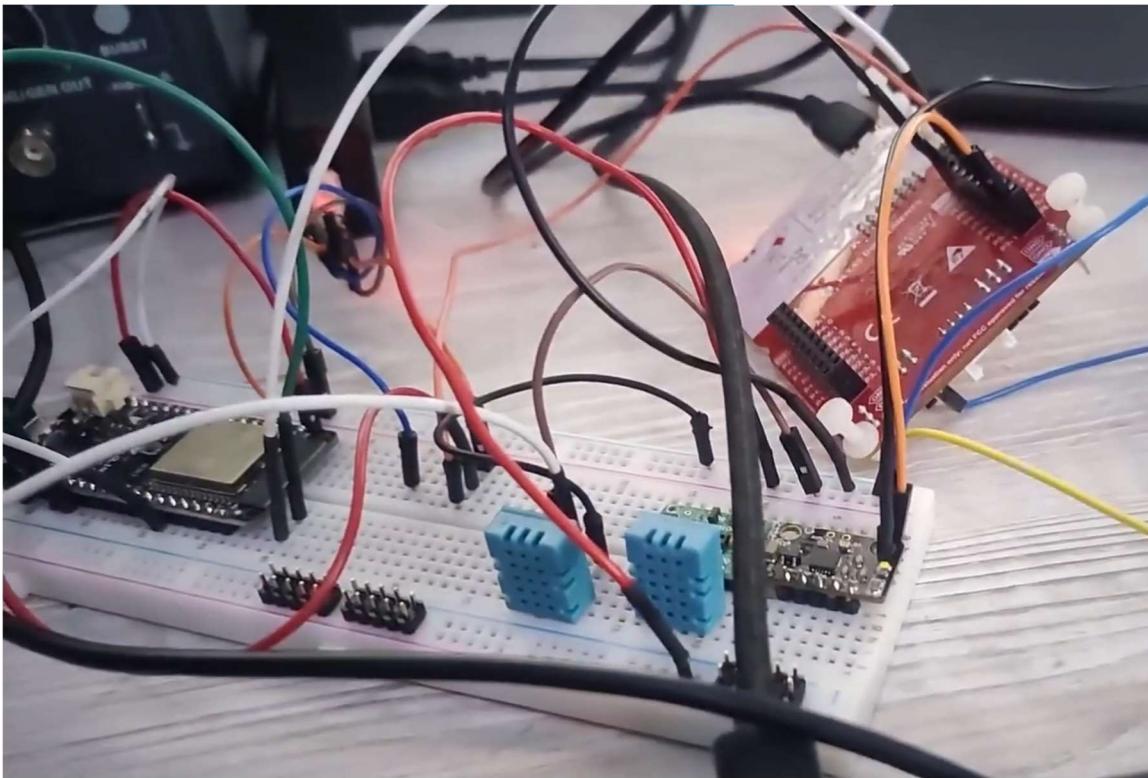


Figura 56. ESP32 și MSP430. Comunicare UART.

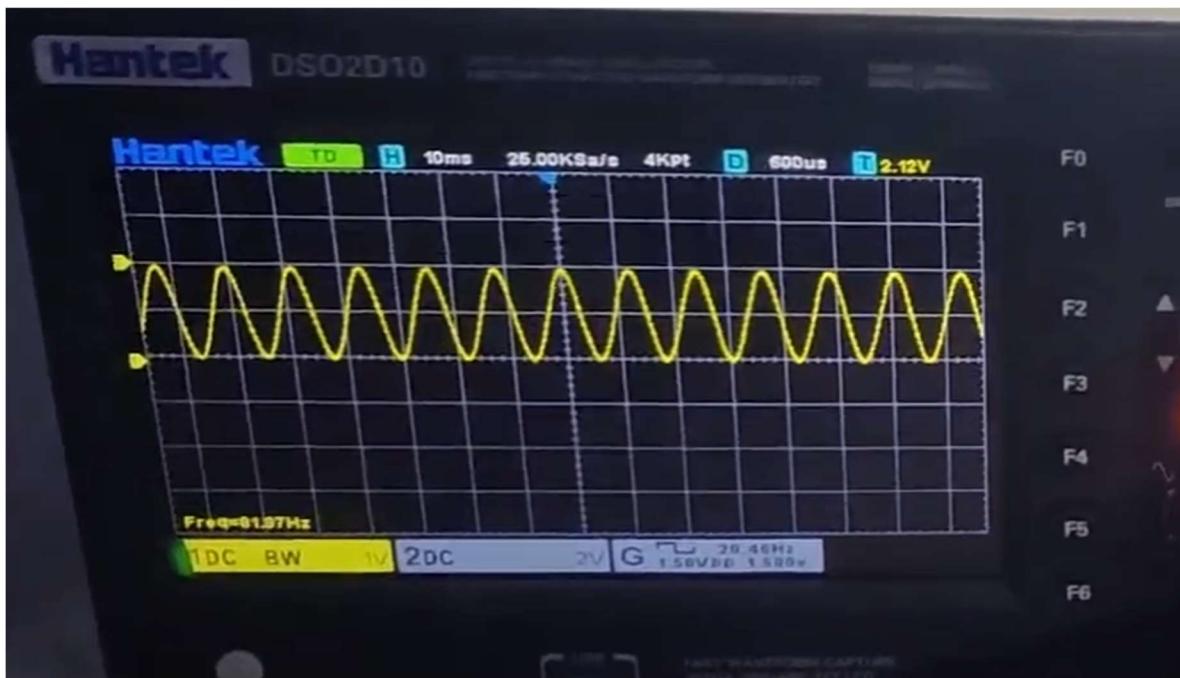


Figura 57. Validarea undei sinusoidale cu osciloscopul.

Codul pentru microcontrolerul ESP32 a fost scris în Arduino IDE, unde mediul de dezvoltare se ocupă automat de compilarea (build) și de încărcarea în memorie (flash) acestuia pe placa ESP32 prin interfața USB-to-UART integrată.

Câteva funcții importante din codul microcontrolerului ESP32 folosite în realizarea proiectului sunt acestea:

InitWiFi()

Realizează tot ceea ce ține de conectarea la rețeaua Wi-Fi. Inițiază modulul radio, pornește „begin” cu SSID și parolă, și așteaptă până când WiFi.status() == WL_CONNECTED. Fără o conexiune stabilă aici, nici MQTT (ThingsBoard) nu poate funcționa.

processTemperatureChange(...)

Este callback-ul RPC care răspunde la comanda de schimbare a temperaturii (example_set_temperature). Extragă valoarea "temp" din JSON-ul primit, o trimită pe UART în format DAC:<valoare> și pregătește un răspuns JSON cu "status":"done". Cu alte cuvinte, traduce comanda de la server într-un semnal DAC al ESP32-ului.

sendUARTDataStreamed()

Monitorizează continuu buffer-ul UART de la MSP430, detectând vectorul de date care începe cu ADC:[, îl citește valoare cu valoare (până la]), și pentru fiecare citire apelează tb.sendTelemetryData("adc_value", val). Astfel, fluxul ADC ajunge direct în ThingsBoard ca telemetrie, fără blocaje.

În cazul MSP430, programul a fost dezvoltat în Code Composer Studio de la Texas Instruments. IDE-ul realizează pașii de compilare și linking ai surselor C/C++, apoi încarcă firmware-ul pe MSP430 folosind interfața JTAG sau bootloader-ul UART, în funcție de configurația hardware.

Funcții importante din codul microcontrolerului MSP430 ce închid bucla creată de către utilizator atunci când cere ca o anumită comandă să se execute ar fi:

Process_Command(const char *cmd)

- Primește un sir de caractere (de ex. „LED:1” sau „DAC:3”), acesta este citit cu strncmp() și atoi(), apoi apelează fie LED(val) pentru a comanda LED-ul, fie DAC(val) pentru a încărca forma de undă corespunzătoare în buffer-ul WaveAux.
- Practic, traduce comenziile text primite pe UART în acțiuni hardware.

Transmit_ADC_Buffer()

- Este apelată din rutina TimerB0 când adc_ready este setat. Parcurge adc_buffer[], convertește fiecare valoare numerică în text și o trimită secvențial pe UART sub formatul de „ADC:[v0,v1,...,vN]\n”.
- Asigură împachetarea și trimiterea completă a seriei de măsurători ADC către ESP32 (sau orice alt receptor UART).

ADC_ISR() (întreruperea ADC)

- Se declanșează după fiecare conversie de la ADC. Salvează în adc_buffer[adc_index++] valoarea citită (ADCMEM0), reîncepe conversia (dacă buffer-ul nu este plin) iar atunci când buffer-ul atinge ADC_BUFFER_SIZE, marchează adc_ready = 1 și resetează adc_index.
- Gestioneză achiziția continuă a datelor analogice cu un minim de întârziere.

4.3.2. Codul sursă pentru ESP32, RX si TX pe UART. Comandă MSP430

Codul pentru microcontrolerul ESP32 începe prin inițializarea plăcii, după care se configurează conexiunea Wi-Fi și comunicarea serială. Dacă conexiunea Wi-Fi nu este stabilă până în acest punct, se încearcă reinicializarea până când aceasta are succes. Pasul următor presupune trecerea într-o buclă separată în care se verifică conexiunea cu serverul ThingsBoard, dacă aceasta nu reușește, se încearcă reconectarea. Conexiunea cu serverul este verificată periodic pentru buna funcționare. Datele primite pe UART sunt convertite într-un fișier JSON și apoi trimise la server. Se verifică subșcrierea RPC la server iar dacă aceasta este eșuată, se încearcă reinregistrarea. La primirea unui apel RPC, ESP32 transmite pe UART comanda solicitată de utilizator.

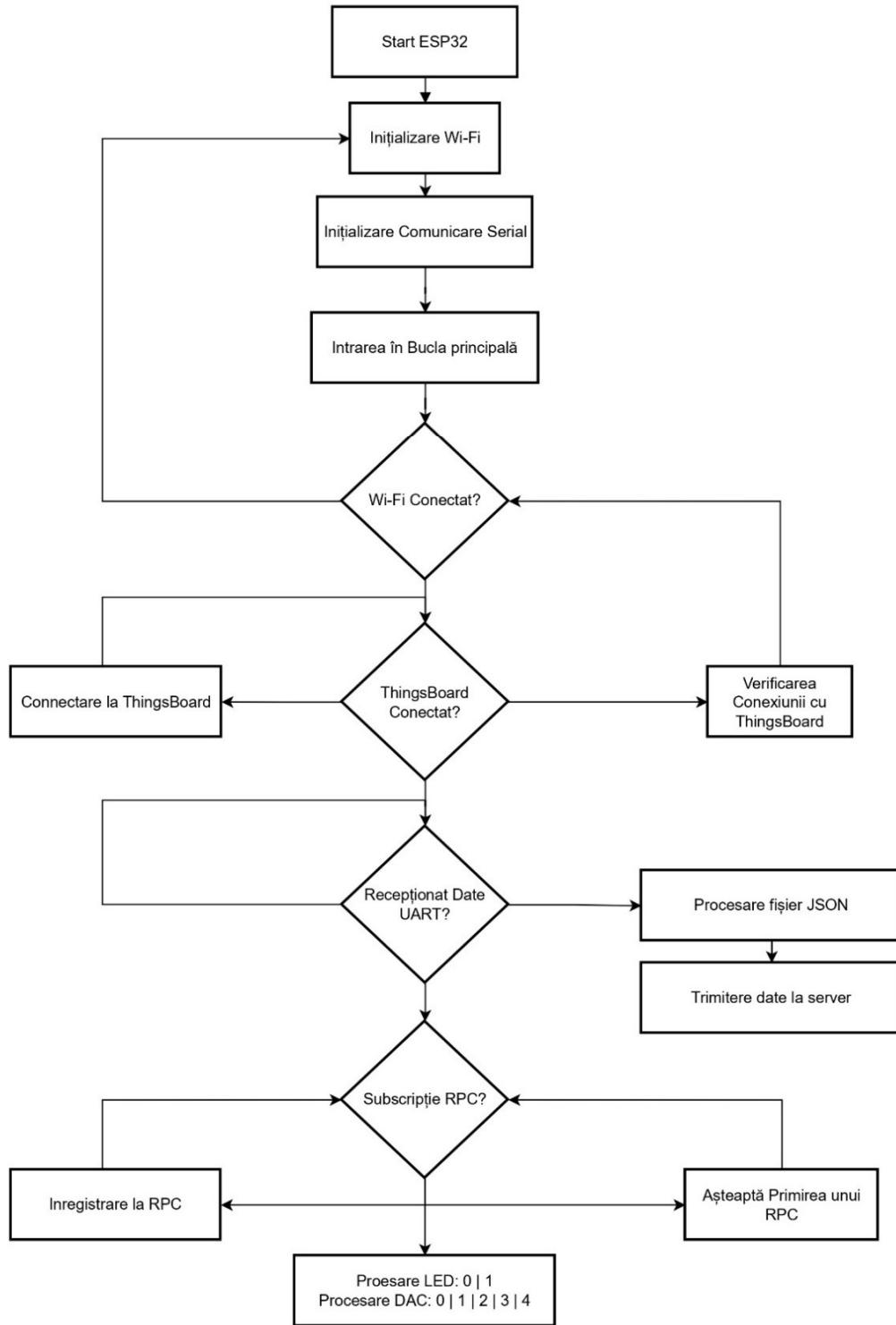


Figura 58. Flux operațional ESP32 pentru date UART și comenzi RPC

```

#ifndef ESP8266
#include <ESP8266WiFi.h>
#else
#ifndef ESP32
#include <WiFi.h>
#include <WiFiClientSecure.h>
#endif // ESP32
#endif // ESP8266

#include <Arduino_MQTT_Client.h>
#include <Server_Side_RPC.h>
#include <ThingsBoard.h>

#define ENCRYPTED false
#define MAX_ADC_VALUES 1000 // Max expected number of values
#define JSON_OVERHEAD 32 // Rough buffer overhead
StaticJsonDocument<JSON_ARRAY_SIZE(MAX_ADC_VALUES) + JSON_OVERHEAD> doc;

constexpr char WIFI_SSID[] = "IoT Server";
constexpr char WIFI_PASSWORD[] = "Floridemar3";
constexpr char THINGSBOARD_SERVER[] = "192.168.139.32";
constexpr uint16_t THINGSBOARD_PORT = ENCRYPTED ? 8883U : 1883U;
constexpr char TOKEN[] = "FU2288y6XZZqCTvjpYfN";
constexpr uint16_t MAX_MESSAGE_SEND_SIZE = 256U;
constexpr uint16_t MAX_MESSAGE_RECEIVE_SIZE = 256U;
constexpr uint32_t SERIAL_DEBUG_BAUD = 115200U;

constexpr char RPC_JSON_METHOD[] = "example_json";
constexpr char RPC_TEMPERATURE_METHOD[] = "example_set_temperature";
constexpr char RPC_SWITCH_METHOD[] = "example_set_switch";
constexpr char RPC_TEMPERATURE_KEY[] = "temp";
constexpr char RPC_SWITCH_KEY[] = "switch";
constexpr uint8_t MAX_RPC_SUBSCRIPTIONS = 3U;
constexpr uint8_t MAX_RPC_RESPONSE = 5U;

#if ENCRYPTED
WiFiClientSecure espClient;
#else
WiFiClient espClient;
#endif

Arduino_MQTT_Client mqttClient(espClient);
Server_Side_RPC<MAX_RPC_SUBSCRIPTIONS, MAX_RPC_RESPONSE> rpc;
const std::array<IAPI_Implementation*, 1U> apis = { &rpc };
ThingsBoard tb(mqttClient, MAX_MESSAGE_RECEIVE_SIZE, MAX_MESSAGE_SEND_SIZE,
Default_Max_Stack_Size, apis);

bool subscribed = false;

```

```

// Initialize HardwareSerial on UART2 (GPIO16=RX, 17=TX)
HardwareSerial mySerial(2);

void InitWiFi() {
    Serial.println("Connecting to AP ...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("Connected to AP");
}

bool reconnect() {
    if (WiFi.status() != WL_CONNECTED) InitWiFi();
    return true;
}

void processGetJson(const JsonVariantConst &data, JsonDocument &response) {
    Serial.println("Received JSON RPC");
    StaticJsonDocument<64> innerDoc;
    innerDoc["info"] = "ok";
    response["json_data"] = innerDoc;
}

void processTemperatureChange(const JsonVariantConst &data, JsonDocument &response) {
    float temperature = data["temp"];
    Serial.printf("Set temperature: %.2f\n", temperature);
    mySerial.printf("DAC:%d\n", (int)temperature);
    response["status"] = "done";
}

void processSwitchChange(const JsonVariantConst &data, JsonDocument &response) {
    bool state = data["switch"];
    Serial.printf("Set LED: %d\n", state);
    mySerial.printf("LED:%d\n", state);
    response["status"] = "done";
}

bool adcStreamActive = false;
String adcToken = "";

void sendUARTDataStreamed() {
    while (mySerial.available()) {
        char ch = mySerial.read();

```

```

// Detect start of ADC stream
if (!adcStreamActive) {
    adcToken += ch;
    if (adcToken.endsWith("ADC:["))
        adcStreamActive = true;
    adcToken = ""; // clear to start collecting values
} else if (adcToken.length() > 5) {
    adcToken.remove(0, 1); // shift buffer window
}
continue;
}

// If active, collect token until comma or closing bracket
if (ch == ',' || ch == ']') {
    adcToken.trim();
    if (adcToken.length() > 0) {
        int val = adcToken.toInt();

        // Send value using sendTelemetryData with a generic key like
"adc_value"
        tb.sendTelemetryData("adc_value", val);
        Serial.println("→ adc_value: " + String(val));
    }
    adcToken = ""; // reset token

    if (ch == ']') {
        adcStreamActive = false; // end of ADC stream
    }
} else {
    adcToken += ch; // accumulate digits
}
}

void setup() {
    Serial.begin(SERIAL_DEBUG_BAUD);
    delay(1000);
    InitWiFi();
    mySerial.begin(115200, SERIAL_8N1, 16, 17);
}

void loop() {
    if (!reconnect()) return;

    if (!tb.connected()) {

```

```
Serial.printf("Connecting to TB: (%s) token (%s)\n",
THINGSBOARD_SERVER, TOKEN);
if (!tb.connect(THINGSBOARD_SERVER, TOKEN, THINGSBOARD_PORT)) {
    Serial.println("TB connect failed");
    return;
}
}

if (!subscribed) {
    Serial.println("Subscribing RPC...");
    const std::array<RPC_Callback, MAX_RPC_SUBSCRIPTIONS> callbacks = {
        RPC_Callback{"example_json", processGetJson},
        RPC_Callback{"example_set_temperature", processTemperatureChange},
        RPC_Callback{"example_set_switch", processSwitchChange}
    };
    if (!rpc.RPC_Subscribe(callbacks.cbegin(), callbacks.cend())) {
        Serial.println("RPC subscribe failed");
        return;
    }
    subscribed = true;
}
sendUARTDataStreamed();
tb.loop();

// if (mySerial.available()) {
//     String line = mySerial.readStringUntil('\n');
//     line.trim();
//     parseADCAndSend(line);
//}
}
```

4.3.3. Codul sursă pentru MSP430, RX si TX pe UART. Comandă ESP32

În cazul microcontrolerului MSP430, primul pas este inițializarea acestuia și a modulelor sale. După ce această inițializare este efectuată cu succes, microcontrolerul așteaptă întreruperi într-o rutină de receptie UART. Dacă niciun mesaj nu este primit, ascultarea continuă, în caz contrar, se verifică dacă mesajul ce a generat întreruperea trebuie procesat în bucla de selecție a DAC-ului sau în cea de control al LED-ului. Dacă este selectată o formă de undă pentru DAC, aceasta va fi și unda generată și citită de ADC, iar valorile achiziționate sunt trimise pe UART, pentru a fi preluate de ESP32.

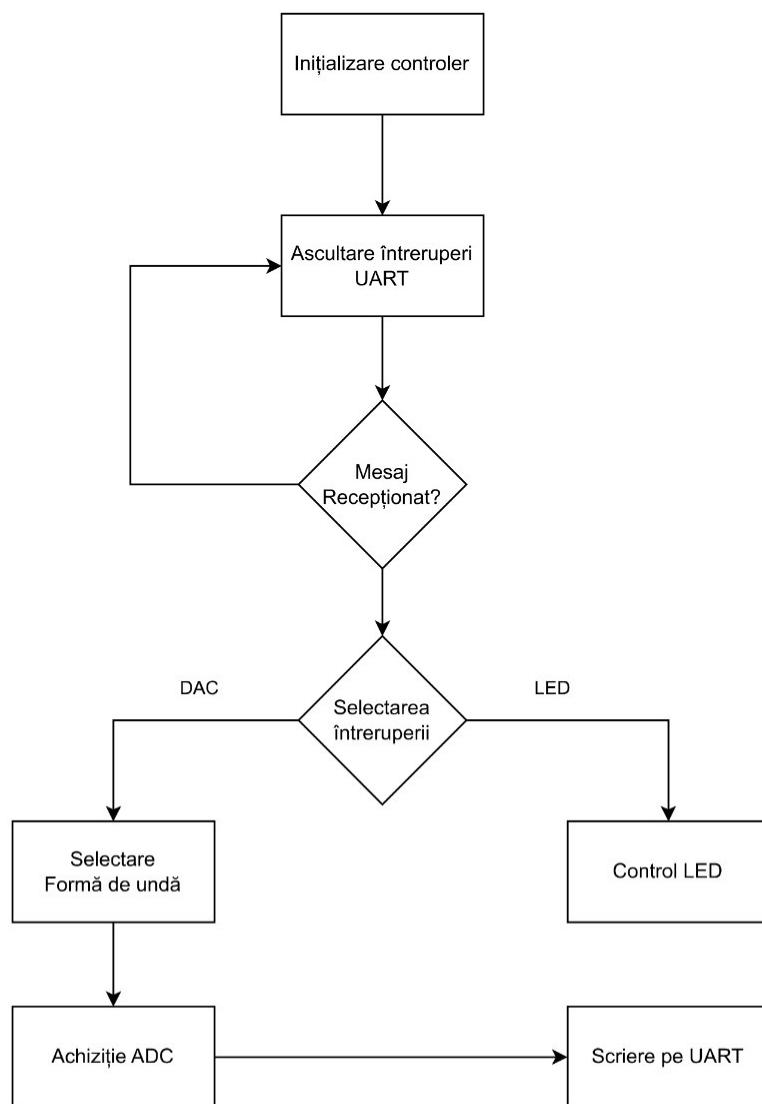


Figura 59. Flux operațional MSP430 pentru recepție și scriere UART

```

// EXTENDED VERSION: UART RECEIVE, DAC/LED CONTROL, ADC, UART
#include <msp430.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define CALADC_15V_30C  *((unsigned int *)0x1A1A)
#define CALADC_15V_85C  *((unsigned int *)0x1A1C)
#define ADC_BUFFER_SIZE 200
volatile unsigned short int adc_buffer[ADC_BUFFER_SIZE];
volatile unsigned int adc_index = 0;
volatile unsigned char adc_ready = 0;

volatile unsigned short int sine[] = {
    2048, 2447, 2831, 3185, 3495, 3750,
3939,
    4056, 4095, 4056, 3939, 3750, 3495,
3185,
    2831, 2447, 2048, 1649, 1265, 911,
601,
    346, 157, 40, 1, 40, 157,
346,
    601, 911, 1265, 1649 };

volatile unsigned short int square[] = {
    0, 0, 0, 0, 0,
0, 0,
    0, 4095, 4095, 4095, 4095,
4095, 4095,
    4095, 4095, 4095, 4095, 4095,
4095, 4095,
    4095, 4095, 4095, 0, 0,
0, 0,
    0, 0, 0, 0 };

volatile unsigned short int sawtooth[] = {
    0, 129, 259, 388, 518, 647, 776,
    906, 1035, 1164, 1294, 1423, 1553, 1682,
    1811, 1941, 2070, 2199, 2329, 2458, 2587,
    2717, 2846, 2976, 3105, 3234, 3364, 3493,
    3622, 3752, 3881, 4011 };

volatile unsigned short int tan[] = {
    1086, 1191, 1299, 1411, 1529, 1658, 1801,
    1964, 2156, 2390, 2687, 3088, 3669, 4613,
    6465, 11950, 32767, -9778, -4293, -2441, -1497,

```

```

-916, -515, -218,    16,   208,   371,   514,
 643,  761,  873,  981 };

volatile unsigned short int WaveAux[32];
volatile unsigned int i, i_1, i_2,j,k;

void Init_GPIO();
void Init_UART();
void Init_TimerB0();
void Init_ADC();
void Transmit_Temperature(float tempC);
void LED(unsigned char state);
void DAC(unsigned char type);
void Process_Command(const char *cmd);

volatile float temp;
volatile float IntDegC;
char uart_buffer[20];
volatile unsigned int uart_index = 0;
//volatile int i;

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;
    Init_GPIO();
    Init_UART();
    Init_TimerB0();
    Init_ADC();
    Init_Values();
    Init_DAC();

    PM5CTL0 &= ~LOCKLPM5;
    __bis_SR_register(GIE);

    while (1)
    {
        __bis_SR_register(LPM0_bits | GIE);
        __no_operation();
    }
}

void Init_DAC()
{
// Configure reference module
    PMMCTL0_H = PMMPW_H;                      // Unlock PMM registers
    PMMCTL2 = INTREFEN | REFVSEL_1;             // Enable 2V internal ref
    while (!(PMMCTL2 & REFGENDRDY));           // Wait for ref to settle
}

```

```

// Configure SAC2
SAC2DAC = DACSREF_1 | DACLSEL_3 | DACIE; // 2V ref, DAC triggered by
TB2.2
SAC2DAT = square[i_2];
SAC2DAC |= DACEN; // Enable DAC

SAC2OA = NMUXEN | PMUXEN | PSEL_1 | NSEL_1; // OA input settings
SAC2OA |= OAPM_0; // High-speed mode
SAC2PGA = MSEL_1; // Buffer mode
SAC2OA |= SACEN | OAEN; // Enable SAC and OA

// Configure TimerB2 to generate DAC triggers
TB2CCR0 = 200 - 1;
TB2CCTL2 = OUTMOD_4; // Toggle mode (reset/set
also OK)
TB2CCR2 = 100; // Match at middle
TB2CTL = TBSSEL__SMCLK | MC_1 | TBCLR; // SMCLK, Up mode, Clear
}

void Init_Values()
{
    i = 0;
    i_1 = 0; // initial phase square 1
    i_2 = 0; // starting at 0 index
}

void Init_GPIO()
{
    P1DIR |= BIT0;
    P1OUT &= ~BIT0;
    P1SEL0 |= BIT6 | BIT7; // UART pins

    // Configure P3.1 for SAC2 OA output
    P3SEL0 |= BIT1;
    P3SEL1 |= BIT1;

    //P1.0 Analog input ADC
    P1SEL0 |= BIT0;
    P1SEL1 |= BIT0;

}

void Init_UART()
{
    UCA0CTLW0 = UCSWRST;
    UCA0CTLW0 |= UCSSEL__SMCLK;
    UCA0BR0 = 8;
}

```

```

UCA0MCTLW = 0xD600;
UCA0BR1 = 0;
UCA0CTLW0 &= ~UCSWRST;
UCA0IE |= UCRXIE; // Enable RX interrupt
}

void Init_TimerB0()
{
    TB0CCTL0 = CCIE;
    TB0CCR0 = 32768;
    TB0CTL = TBSSEL__ACLK | MC__UP | TBCLR;
}

void Init_ADC()
{
    ADCCTL0 |= ADCSHT_8 | ADCON;
    ADCCTL1 |= ADCSHP;
    ADCCTL2 &= ~ADCRES;
    ADCCTL2 |= ADCRES_2;
    //ADCMCTL0 |= ADCSREF_1 | ADCINCH_12; //internal Temp sensor
    ADCMCTL0 |= ADCSREF_1 | ADCINCH_0; // A0 = P1.0

    ADCIE |= ADCIE0;
    PMMCTL0_H = PMMPW_H;
    PMMCTL2 |= INTREFEN | TSENSOREN;
    __delay_cycles(400);
}

void Transmit_ADC_Buffer()
{
    while (!(UCA0IFG & UCTXIFG));
    UCA0TXBUF = 'A';
    while (!(UCA0IFG & UCTXIFG));
    UCA0TXBUF = 'D';
    while (!(UCA0IFG & UCTXIFG));
    UCA0TXBUF = 'C';
    while (!(UCA0IFG & UCTXIFG));
    UCA0TXBUF = ':';
    while (!(UCA0IFG & UCTXIFG));
    UCA0TXBUF = '[';

    for ( j = 0; j < ADC_BUFFER_SIZE; j++) {
        char value_str[6];
        int len = snprintf(value_str, sizeof(value_str), "%u",
adc_buffer[j]);
        for ( k= 0; k < len; k++) {
            while (!(UCA0IFG & UCTXIFG));
            UCA0TXBUF = value_str[k];
        }
    }
}

```

```

        }
        if (j < ADC_BUFFER_SIZE - 1) {
            while (!(UCA0IFG & UCTXIFG));
            UCA0TXBUF = ',';
        }
    }
    while (!(UCA0IFG & UCTXIFG));
    UCA0TXBUF = ']';
    while (!(UCA0IFG & UCTXIFG));
    UCA0TXBUF = '\n';
}

void Transmit_Temperature(float tempC)
{
    char buffer[10];
    int len = snprintf(buffer, sizeof(buffer), "% .2f\n", tempC);
    for (i = 0; i < len; i++)
    {
        while (!(UCA0IFG & UCTXIFG));
        UCA0TXBUF = buffer[i];
    }
}

void LED(unsigned char state)
{
    if (state)
        P1OUT |= BIT0;
    else
        P1OUT &= ~BIT0;
}

void DAC(unsigned char type)
{
    // Placeholder: Replace with real DAC setup depending on your
    implementation
    switch (type)
    {
        case 0:
            for (i = 0; i < 32; i++)
                WaveAux[i] = 0;
            break;
        case 1:
            for (i = 0; i < 32; i++)
                WaveAux[i] = sine[i];
            break;
        case 2:
            for (i = 0; i < 32; i++)
                WaveAux[i] = square[i];
    }
}

```

```

        break;
    case 3:
        for (i = 0; i < 32; i++)
            WaveAux[i] = sawtooth[i];
        break;
    case 4:
        for (i = 0; i < 32; i++)
            WaveAux[i] = tan[i];
        break;
    default:
        break;
    }
}

void Process_Command(const char *cmd)
{
    if (strncmp(cmd, "LED:", 4) == 0)
    {
        int val = atoi(cmd + 4);
        LED(val);
    }
    else if (strncmp(cmd, "DAC:", 4) == 0)
    {
        int val = atoi(cmd + 4);
        DAC(val);
    }
}

#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    if (UCA0IFG & UCRXIFG)
    {
        char rx = UCA0RXBUF;
        if (rx == '\n' || rx == '\r')
        {
            uart_buffer[uart_index] = '\0';
            Process_Command(uart_buffer);
            uart_index = 0;
        }
        else if (uart_index < sizeof(uart_buffer) - 1)
        {
            uart_buffer[uart_index++] = rx;
        }
    }
}

#pragma vector=TIMER0_B0_VECTOR

```

```

__interrupt void Timer_B0_ISR(void)
{
    static unsigned int tick = 0;
    if (adc_ready) {
        Transmit_ADC_Buffer();
        adc_ready = 0;
    } else if (tick == 0) {
        ADCCTL0 |= ADCENC | ADCSC;
    }
    tick++;
    if (tick >= 10) tick = 0;
}

#pragma vector=ADC_VECTOR
__interrupt void ADC_ISR(void)
{
    switch (__even_in_range(ADCIV, ADCIV_ADCIFG)) {
        case ADCIV_NONE: break;
        case ADCIV_ADCIFG:
            if (adc_index < ADC_BUFFER_SIZE) {
                adc_buffer[adc_index++] = ADCMEM0;
                if (adc_index >= ADC_BUFFER_SIZE) {
                    adc_ready = 1;
                    adc_index = 0;
                } else {
                    ADCCTL0 |= ADCENC | ADCSC; // Restart ADC
                }
            }
            break;
        default: break;
    }
}

// ISR for SAC2 DAC
#pragma vector=SAC0_SAC2_VECTOR
__interrupt void SAC2_ISR(void)
{
    switch (__even_in_range(SAC2IV, SACIV_4))
    {
        case SACIV_0: break;
        case SACIV_2: break;
        case SACIV_4:
            i_2++;
            if (i_2 >= sizeof(WaveAux)/sizeof(WaveAux[0]))
                i_2 = 0;
            SAC2DAT = WaveAux[i_2];
            break;
        default: break;}}

```

Capitolul 5. Evaluare și concluzii

5.1. Evaluarea performanței și a securității

În contextul acestei lucrări, securitatea trebuie abordată pe mai multe niveluri, de la senzor și microcontroler, până la infrastructura virtuală și procesele CI/CD. Câteva puncte importante ce pot reprezenta vulnerabilități ale sistemului pot fi:

Securitatea la nivel hardware și firmware

Protejarea senzorilor și a microcontrolerelor (ESP32, MSP430) reprezintă securitatea locală, atunci când un posibil atacator are acces fizic la microcontrolere dar și la server. Pentru asigurarea securității firmware-ului și a comunicațiilor pe microcontrolerul ESP32, propunem implementarea următoarelor măsuri:

1. Secure Boot

Secure Boot garantează că orice cod executat pe dispozitiv este semnat criptografic de o entitate autorizată și nu a fost alterat. În momentul pornirii, bootloader-ul ESP32 verifică semnatura digitală a firmware-ului înainte de a permite încărcarea acestuia, blocând astfel orice încercare de rulare a unui cod neautorizat sau modificat.

2. Criptarea memoriei flash

Pentru a preveni extragerea și analiza neautorizată a codului sursă de pe dispozitiv, întreg conținutul memoriei flash va fi criptat. Această criptare hardware-accelerată împiedică atacatorii să recupereze imaginea firmware-ului, chiar dacă obțin fizic acces la cip.

3. Izolarea și autentificarea canalelor de comunicație

Fiecare magistrală hardware (I^2C , SPI, CAN) va fi segregată în domenii de acces izolate, iar dispozitivele conectate vor fi obligate să se autentifice înainte de a transmite comenzi. În practică:

- Pe magistrala I^2C și SPI, se folosește un protocol de handshake cu challenge-response pentru fiecare operațiune de citire/scriere.
- Pe magistrala CAN, pachetele de date sunt semnate și validate la nivel de cadru, prevenind injectarea de mesaje malițioase.

Principiul „Least Privilege” în arhitectura firmware-ului

Fiecare componentă a firmware-ului (de exemplu, modulul de achiziție ADC, cel de RPC sau cel de MQTT) este proiectată să execute doar acele acțiuni pentru care are în mod strict nevoie de permisiuni. Astfel, accesul la resursele de sistem — registre hardware, segmente de memorie sau canale de comunicație — este restricționat la nivel de modul, reducând semnificativ suprafața de atac și impactul în cazul unui exploat.

Protocole de comunicație și asigurarea integrității datelor

Pentru schimbul de mesaje și comenzi între dispozitiv și server, se impun următoarele măsuri de securitate:

- **MQTT**

- Autentificare robustă pe broker, folosind token-uri unice sau certificate digitale, în combinație cu liste de control (ACL) pentru definirea permisunilor pe topic-uri.
- Criptare end-to-end a canalului de comunicație prin TLS/SSL, pentru a împiedica interceptarea sau manipularea pachetelor.
- Mecanisme anti-Man-in-the-Middle (MITM), precum verificarea certificatelor și pinning-ul cheilor, esențiale atunci când ESP32 funcționează drept gateway.

- **RPC și VISA/SCPI**

- Fiecare comandă transmisă către echipamentele industriale trebuie semnată criptografic, iar dispozitivul receptor verifică semnatura înainte de executare.
- Răspunsurile sunt, de asemenea, însotite de semnături digitale care atestă proveniența și integritatea datelor primite.

Securitatea virtualizării și a hypervisor-ului

Pentru a asigura integritatea și izolația mediului virtual, sunt necesare setări de sistem la nivelul hypervisor-ului și al rețelelor interne. În primul rând, securizarea Unraid implică dezactivarea tuturor serviciilor neesențiale, aplicarea la zi a update-urilor de securitate și testarea configurației pentru a elimina orice punct de atac. Politica de networking actuală nu permite segregarea traficului. Practic deși definim segmente de rețea dedicate pentru management, stocare și datele de producție, acest strat de microsegmentare nu ne permite să aplicăm politici de control ale fluxurilor de trafic la nivel de VM. Rezolvarea acestei probleme se află în virtualizarea rețelei de internet prin VMWare NSX.

Retenția și protecția datelor

Protejarea informațiilor sensibile stocate pe infrastructura virtuală presupune, în primul rând, utilizarea criptării la repaus („at rest”) pe volumele RAID. Serviciu de paritate din Unraid va cripta automat blocurile de date, împiedicând astfel accesul neautorizat la nivel fizic. Totuși, dacă sistemul suferă prea multe eșecuri la nivel de mediu de stocare, datele sunt imposibil de recuperat. Avarierea sistemului presupune imposibilitatea accesării serverului, neexistând soluții de backup la nivel de sistem. Acest lucru poate fi rezolvat prin implementarea hypervisor-ului VMWare ESXi și utilizarea serviciului vSphere.

Securitatea în procesul CI/CD

Adoptarea principiilor „Infrastructure as Code” și a containerizării introduce riscuri noi, care trebuie gestionate. Fiecare imagine Docker este supusă unei scanări automate pentru vulnerabilități CVE înainte de a fi publicată în registru, astfel imaginile aprobată sunt semnate digital („signed images”) și pot intra în pipeline-ul Jenkins sau GitLab CI numai după validarea semnăturii și a rezultatului scanărilor. Mediile

de dezvoltare, testare și producție sunt clar separate. Totuși accesul la chei, token-uri, sau certificate nu este centralizat și controlat de un vault (VMware vSphere Secrets).

Accesul la distanță și VPN

Pentru a evita vulnerabilitățile generate de expunerea serviciilor în Internet, toate conexiunile administrative și de date ale microcontrolerelor vor fi realizate exclusiv prin tuneluri VPN. Serviciile de DDNS vor rezolva schimbările de adresă IP, dar vor fi utilizate doar intern, în cadrul rețelei virtuale protejate. Accesul va fi reglementat de reguli stricte de firewall, aplicate atât la nivelul fiecărei interfețe de rețea (NIC).

5.2. Limitări și probleme

5.2.1. Limitări și probleme legate de senzori

În practică, deși ESP32 oferă o gamă largă de interfețe (GPIO, ADC, I²C, SPI, UART) și un procesor dual-core de 240 MHz, încercarea de a conecta simultan zeci de senzori diferenți scoate rapid la iveală câteva limite fundamentale.

- **Resurse fizice și de conversie**

Numărul de pini GPIO și canalele ADC interne nu sunt suficiente pentru a citi direct toate semnalele analogice și digitale, astfel sunt necesare expansiuni de tip I/O-expander sau ADC extern multiplexat.

- **Încărcarea magistralelor comunicaționale**

Prea multe dispozitive pe aceeași magistrală I²C sau SPI, plus UART-uri multiple, provoacă degradări de semnal, time-out-uri și un management complicat al selectării de slave.

- **Interferențe electromagnetice și consum energetic**

Transmiterea Wi-Fi/BLE generează vârfuri de curent și zgomot RF care afectează liniile analogice la rândul lor din cauza interferențelor electromagnetice.

- **Putere de procesare și memorie**

Task-urile de citire, filtrare și compensare concură cu cele de rețea și criptare, memoria RAM poate fi rapid ocupată dacă nu se externalizează buffering-ul sau stocarea.

- **Complexitate software și mențenanță**

Scrierea unui protocol de comunicare între un dispozitiv ce suportă doar comunicare pe fir cu un dispozitiv capabil să trimită datele la server reprezintă un strat în plus de complexitate. La adăugarea unei noi funcționalități, este necesar să actualizăm codul microcontrolerul.

Pentru a realiza un sistem stabil cu o multitudine de senzori, este recomandată fie distribuirea sarcinilor pe mai multe ESP32 (sau microcontrolere slave), fie adaugarea de extensii hardware (expansoare I²C/SPI, ADC externe), algoritmi de filtrare eficienți, gestionare a modurilor de sleep și programarea execuțiilor în funcție de prioritate. Aceste măsuri mențin sistemul în limite funcționale și scalabile.

5.2.2. Limitări și probleme legate de protoalele de comunicare ale ESP32

Prin combinarea multiplexoarelor, ADC-urilor externe, a magistralelor I²C/SPI și a protoalelor CAN, Modbus, VISA/SCPI, ESP32 devine un gateway universal.

- Poate citi zeci de senzori analogici și digitali
- Poate interoga echipamente industriale vechi sau de laborator
- Poate traduce și centraliza datele în cloud sau în sisteme SCADA moderne

Desigur, orice arhitectură embedded care își propune să interconecteze și să controleze dispozitive multiple prin protoale variate va întâmpina anumite limitări tehnice și funcționale. Spre exemplu, ESP32, deși este un microcontroler foarte versatil, are un număr limitat de canale ADC interne, iar cele disponibile pot suferi de instabilitate și zgromot, mai ales în prezența comunicațiilor Wi-Fi active. O soluție practică în acest sens este utilizarea multiplexoarelor analogice (precum CD4051) sau a convertizoarelor ADC externe de înaltă precizie (ADS1115, MCP3008), care oferă izolare, rezoluție crescută și o mai bună linearitate.

În cazul UART-ului, pot apărea probleme legate de sincronizare, pierderea de date la baud rate-uri mari sau parsing incorrect al mesajelor, mai ales dacă acestea nu sunt terminate clar sau dacă nu se folosesc delimitatori standard precum \r\n. Aceste probleme se pot atenua prin implementarea de buffere circulare, rutine ISR optimizate și, ideal, control hardware al fluxului de date (RTS/CTS). În mod similar, I²C, deși simplu și eficient pentru comunicații între circuite integrate, se confruntă frecvent cu conflicte de adresa (deoarece există doar 127 adrese disponibile) și probleme de integritate a semnalului pe trasee mai lungi. Acestea se rezolvă prin utilizarea de multiplexoare I²C (ex. TCA9548A), trasee scurte, și pull-up-uri corect dimensionate (tipic 4.7kΩ).

SPI-ul, cu avantajul vitezelor ridicate, implică la rândul său complexitate în managementul pinilor de selecție (CS), iar dacă tranzacțiile nu sunt delimitate corect sau dacă semnalele se degradează la frecvențe mari, pot apărea erori de comunicație. Aici este importantă folosirea funcțiilor SPI.beginTransaction() și SPI.endTransaction(), reducerea frecvenței în caz de instabilitate, și izolarea liniilor cu rezistențe de terminare.

Protocolul CAN, deși extrem de stabil și utilizat industrial, presupune o implementare mai elaborată: e nevoie de un transceiver hardware dedicat (precum MCP2551 sau SN65HVD), iar la nivel software trebuie gestionate filtre și priorități pe ID-uri. Pentru protoale mai avansate, cum ar fi XCP peste CAN sau CAN-FD, este esențială existența unor fișiere de descriere precum A2L, care definesc toate obiectele accesibile în memorie – fără de care comunicarea rămâne opacă. Lipsa criptării native în CAN clasic este o limitare serioasă, motiv pentru care în aplicațiile moderne se preferă extensii mai sigure (ex: CAN-FD cu XCP securizat).

Modbus, utilizat frecvent în dispozitive industriale mai vechi, poate ridica probleme la nivel de timing și verificare CRC, mai ales în varianta RTU. E nevoie de o implementare robustă, cu time-out-uri clare, retry logic și, în cazul RS-485, management corect al pinului DE/RE pentru activarea transmiterii. VISA și SCPI, utilizate pentru interfațarea cu echipamente de laborator, pot fi lente pe RS232 și sunt adesea implementate diferit între producători. Aici este critică înțelegerea profundă a manualului de comenzi

SCPI și a documentației tehnice. În plus, trebuie adăugate delay-uri corecte între comenzi și validată fiecare răspuns pentru a evita blocarea.

Pe lângă toate aceste limitări specifice de protocol, la nivel general, firmware-ul poate deveni repede complex și dificil de întreținut dacă nu se aplică o structurare clară. Concurența între task-uri, accesul simultan la magistrale sau ISR-uri prea lungi pot duce la instabilitate. Aceste probleme se atenueză prin folosirea unui RTOS (precum FreeRTOS), unde fiecare sarcină (comunicare, procesare, transmisie) rulează separat și sincronizat prin cozi, semafoare și priorități. DMA-ul trebuie utilizat oriunde este posibil, iar rutina de intrerupere (ISR) trebuie păstrată cât mai scurtă.

Prin abordarea sistematică a acestor limitări și adoptarea soluțiilor moderne și scalabile, putem realiza un sistem embedded complet, în care un nod central precum ESP32 este capabil să achiziționeze date, să controleze echipamente, și să expună o interfață de control și monitorizare, fie local, fie remote – cu aplicabilitate imediată în domeniul industrial, educațional sau de cercetare.

5.2.3. Limitări și probleme legate de software

Deși arhitectura completă pe care am descris-o, pornind de la hypervisor/bare-metal, VM-uri, rețea și firewall, prin containerizare Docker, Disaster Recovery, tunel VPN și până la platforma IoT și celelalte servicii containerizate, a fost creat un mediu extrem de flexibil și scalabil, în practică există întotdeauna constrângeri.

Limitări fizice

Alocarea de vCPU și RAM peste capacitatea fizică poate duce la „noisy neighbor” (o VM zgomotoasă consumă CPU/memorie peste măsură și încetinește celelalte procese).

Lipsa priorității poate genera situații în care VM-urile critice nu au resurse atunci când au nevoie.

Persistența datelor

Volumele Docker trebuie gestionate separat; pierderea unui container nu înseamnă pierderea datelor, dar configurațiile incorecte pot duce la coruperea volumelor.

Managementul imaginilor și versiuni

Imaginile Docker pot crește rapid în dimensiune și număr (build-uri repetitive, tag-uri multiple), ocupând spațiu și facând push/pull lente.

VPN Tunneling

Dacă VPN gateway-ul (Tailscale exit node, WireGuard pe Raspberry Pi) cade, toți administratorii rămân fără acces. Este bine să existe alternative (mirroring) pe aceste servicii mereu. Aceasta reprezintă în teorie un singur punct ce poate cauza defecțiuni (single point of failure) și poate duce la inoperabilitatea totală a sistemului dacă nu există soluții de backup.

5.3. Concluzii și direcții viitoare. VMWare ESXi, vSphere, Cloud Director, SDDA

Ca direcție de viitor, propunem mutarea întregii infrastructuri virtuale pe VMware ESXi, platforma enterprise a VMware. În acest model, Unraid ar funcționa ca o mașină virtuală în ESXi, iar toate

celealte sisteme de operare pe care le rulăm astăzi în Unraid vor fi migrate într-un mediu complet gestionat de ESXi. Alegerea acestei soluții se bazează pe patru avantaje majore:

Clusterizare și replicare distribuită

Prin vSphere, putem reuni mai multe servere fizice într-un singur cluster, unde datele sunt replicate în timp real între noduri, asemănător unui RAID la nivel de rețea. Într-un cluster tipic de trei noduri:

- Un nod este activ și rulează efectiv sarcinile de lucru.
- Al doilea nod servește drept standby și menține o copie sincronă a datelor.
- Al treilea nod, Witness, monitorizează starea întregului cluster și previne situația de „split-brain” (două instanțe care cred că sunt active simultan).

Această arhitectură asigură un uptime de 99,99%, deoarece, în cazul în care nodul activ întâmpină probleme hardware sau de rețea, unul dintre celealte noduri preia instantaneu rolul de gazdă.

Rețea definită software (VMware NSX)

În loc să folosim switch-uri fizice și echipamente de rutare tradiționale, NSX ne permite să creăm rețele virtuale complete la nivel de hipervizor. Putem:

- Defini VLAN-uri și trunk-uri direct în software, fără a mai schimba fire și porturi.
- Configura politici de securitate distribuită cu reguli de firewall integrate în fiecare mașină virtuală.
- Minimizăm traficul fizic între servere și reducem latența prin rutare și filtrare internă, în interiorul acelorași servere.

Portal self-service și orchestrare (Cloud Director)

VMware Cloud Director oferă un interfață web prin care utilizatorii finali sau echipele de proiect pot:

- Selecta rapid câte CPU virtuale, memorie RAM, spațiu de stocare și bandă de rețea doresc.
- Monitoriza în timp real consumul de resurse și costurile asociate.
- Programează automat backup-uri, snapshot-uri și politici de reciclare a resurselor.

Acest nivel de control transformă infrastructura într-un Software-Defined Data Center (SDDC), unde componentele fizice (compute, storage, networking) devin servicii configurabile dintr-un singur loc.

Scalabilitate și automatizare

- Adăugarea unui nou server ESXi în cluster se face în câțiva pași simpli, iar vSphere extinde automat pool-ul de resurse compute și storage.
- Cu vSphere Auto Deploy și host profiles, putem provisona și configura zeci de servere identice, fără intervenție manuală.
- În caz de incident, mecanismele de failover (HA) și migrare live (vMotion, Storage vMotion) asigură continuitatea serviciilor, fără downtime vizibil pentru utilizatori.

Bibliografie

- Boettiger, C. (2015). An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1), 71-79.
- Branzila, M., Cișlariu, S. A., Vornicu, S., & Brinzila, C. (2023, October). Sensory system based on intelligent sensors for implementation of a modern weather station. In *2023 International Conference on Electromechanical and Energy Systems (SIELMEN)* (pp. 1-4). IEEE.
- Hrițcan, D. F., & Balan, D. (2024, September). Exposing IoT Platforms Securely and Anonymously Behind CGNAT. In *2024 23rd RoEduNet Conference: Networking in Education and Research (RoEduNet)* (pp. 1-4). IEEE.
- Haletky, E. (2011). *VMware ESX and ESXi in the Enterprise: Planning Deployment of Virtualization Servers*. Pearson Education.
- Abhilash, G. B. (2016). *Disaster recovery using VMware vSphere replication and vCenter site recovery manager*. Packt Publishing Ltd.
- Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., & Patterson, D. A. (1994). RAID: High-performance, reliable secondary storage. *ACM Computing Surveys (CSUR)*, 26(2), 145-185.
- Babiuch, M., Foltýnek, P., & Smutný, P. (2019, May). Using the ESP32 microcontroller for data processing. In *2019 20th International Carpathian Control Conference (ICCC)* (pp. 1-6). IEEE
- MSP430FR2355
- Texas Instruments. (2016). *MSP430FR2355, MSP430FR2352 mixed-signal microcontrollers data sheet* (Rev. I) [PDF]. Texas Instruments. www.ti.com/lit/ds/symlink/msp430fr2355.pdf
- ESP32
- Espressif Systems. (2017). *ESP32 Series Datasheet* (v2.0) [PDF]. Espressif Systems. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- ThingsBoard. (2024). *ThingsBoard IoT Platform Documentation* (v 3.x) [Software documentation]. <https://thingsboard.io/docs/>
- Docker, Inc. (2024). *Docker Documentation* (v 24.0) [Software documentation]. <https://docs.docker.com/>
- Nextcloud GmbH. (2024). *Nextcloud Administrator Manual* (v 27) [Software documentation]. <https://docs.nextcloud.com/>
- Lime Technology, Inc. (2024). *Unraid Server OS User Guide* (v 6.12) [Software documentation]. <https://unraid.net/>
- Atlassian. (2024). *Jira Software Documentation* (v 9.x) [Software documentation]. <https://support.atlassian.com/jira-software/>

- Jenkins Project. (2024). *Jenkins User Documentation* (v 2.414) [Software documentation].
<https://www.jenkins.io/doc/>
- Raspberry Pi Foundation. (2016). *Raspberry Pi 3 Model B Hardware Documentation* (Rev. 1.2) [Hardware documentation].
<https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md>
- Arduino. (2024). *Arduino IDE* (v 2.2) [Software]. <https://www.arduino.cc/en/software>
- TeraTerm Project. (2016). *Tera Term User's Manual* (v 4.104) [Software manual].
<https://ttssh2.osdn.jp/manual/>
- Odoo S.A. (2024). *Odoo 17.0 Documentation* [Software documentation].
<https://www.odoo.com/documentation/17.0/>
- Wikipedia contributors. (2025). *Enterprise resource planning* [Encyclopedia entry].
https://en.wikipedia.org/wiki/Enterprise_resource_planning
- Tailscale Inc. (2024). *Tailscale Documentation* (v 1.46) [Software documentation].
<https://tailscale.com/kb/>
- OpenVPN Inc. (2024). *OpenVPN Access Server Administrator Manual* (v 2.11) [Software documentation]. <https://openvpn.net/vpn-server-resources/>
- WireGuard LLC. (2024). *WireGuard Documentation* (v 1.0) [Software documentation].
<https://www.wireguard.com/#documentation>
- Harding, S. (2019, 10 martie). *What Is ECC Memory in RAM? A Basic Definition* [Articol online]. Tom's Hardware. <https://www.tomshardware.com/reviews/ecc-memory-ram-glossary-definition,6013.html> tomshardware.com
- collabnix. (n.d.). *Difference between VM vs Docker* [Pagină web]. DockerLabs.
<https://dockerlabs.collabnix.com/beginners/difference-docker-vm> dockerlabs.collabnix.com
- Renaix. (n.d.). *How Does Industry 4.0 Differ From The Previous Generation?* [Postare pe blog]. Renaix. <https://www.renaix.com/industry-4-0-the-fourth-industrial-revolution/> renaix.com
- Ollama Inc. (2025). *Ollama* [Site web]. <https://www.ollama.com>
- PostgreSQL Global Development Group. (2024). *PostgreSQL 16.0 Documentation* [Software documentation]. <https://www.postgresql.org/docs/>
- Mastering VMware. (n.d.). *What is Virtualization?* <https://masteringvmware.com/what-is-virtualization>
- The Samba Team. (2024). *Samba Documentation* [Software documentation].
<https://www.samba.org/samba/docs/>
- The Linux NFS Project. (2024). *NFS Overview and HowTos* [Software documentation].
https://wiki.linux-nfs.org/wiki/index.php/Main_Page

SlideModel. (n.d.). *V Cycle PowerPoint Diagram [PowerPoint template]*.
<https://slidemodel.com/templates/v-cycle-powerpoint-diagram/>

OpenProject GmbH. (2024). *OpenProject Documentation [Software documentation]*.
<https://www.openproject.org/docs/>

Redmine.org. (2024). *Redmine User Guide [Software documentation]*. <https://www.redmine.org/guide>

GitLab Inc. (2024). *GitLab Documentation (v 17) [Software documentation]*. <https://docs.gitlab.com/>

Mattermost Inc. (2024). *Mattermost Documentation [Software documentation]*.
<https://docs.mattermost.com/>

Requarks. (2024). *Wiki.js Documentation (v 2.5) [Software documentation]*. <https://docs.requarks.io/>

Rocket.Chat Technologies Corp. (2024). *Rocket.Chat Documentation [Software documentation]*.
<https://docs.rocket.chat/>

VMware Inc. (2023). *What is a Software-Defined Data Center (SDDC)? [Web page]*.
<https://www.vmware.com/topics/glossary/content/software-defined-data-center>

Cisco Systems, Inc. (2023). *What is a Firewall? [Web page]*.
<https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html>

Cloudflare, Inc. (n.d.). *What is a Port Number? [Web page]*.
<https://www.cloudflare.com/learning/network-layer/what-is-a-port-number/>

Cisco Systems, Inc. (n.d.). *Internet Protocol Overview (IP) [Web page]*.
<https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/blogs/2020/what-is-ip.html>

Texas Instruments. (2024). *Code Composer Studio Integrated Development Environment (IDE) [Software documentation]*. <https://www.ti.com/tool/CCSTUDIO>