Barnes Home Lab

Smart Home Edge Systems

(smart-home-edge-systems.us)

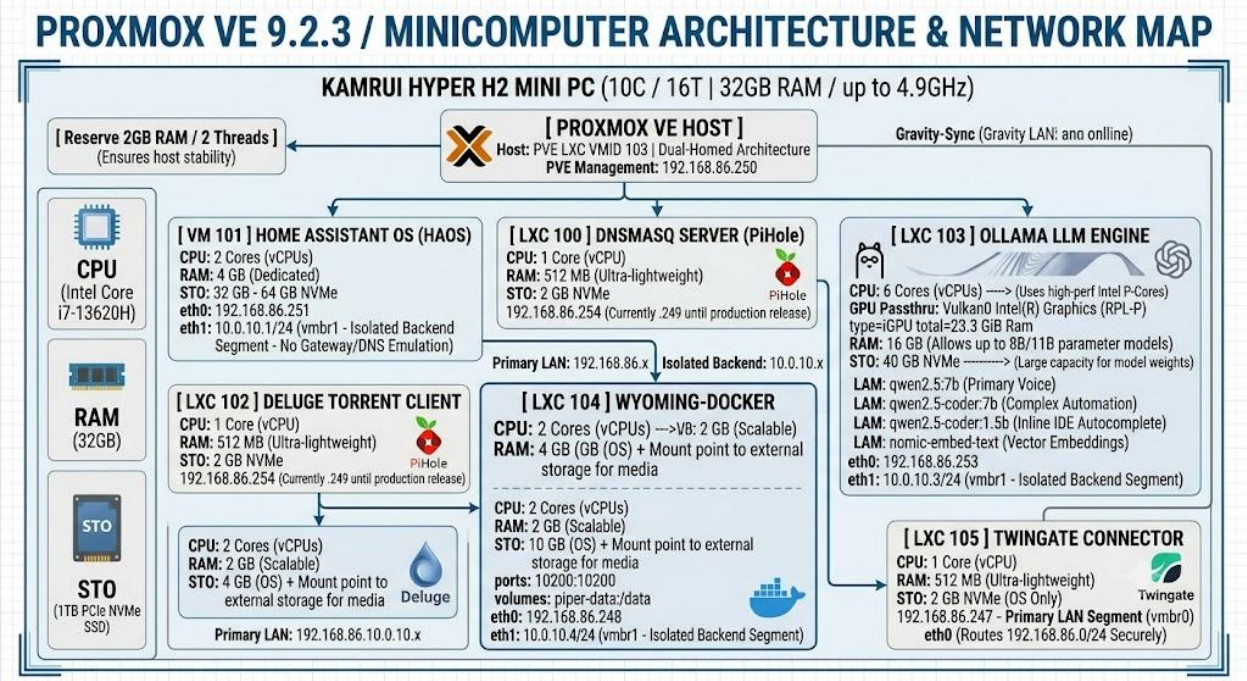


Figure 1: Proxmox VE 9.2.3 / Minicomputer Architecture & Network Diagram

Barnes Home Lab

Table of Contents

Index of Source Code Blocks

Index of Pictures

Introduction

History

Why Build a Home Lab?

A **home lab** is essentially a mini data center set up right in your own living space. It can range from a single, low-power mini PC hidden behind a desk to a dedicated server rack filled with enterprise-grade network gear, switches, and uninterruptible power supplies (UPS).

Figure 2: *A typical structured home lab setup. Source: Digital Spaceport*

While it might sound like overkill to some, there are incredibly practical (and highly addictive) reasons people build them. The motivations generally fall into four main categories:

1. Data Privacy & "De-Googling" (Self-Hosting)

Many people build home labs because they want complete ownership over their data. Instead of trusting third-party cloud corporations with their files, photos, and smart devices, they self-host.

- **Private Cloud Storage:** Running alternatives like Nextcloud to replace Google Drive or Dropbox.
- **Local Smart Home Control:** Using platforms like Home Assistant to manage smart switches, lights, and automated voice clients locally. This ensures your automated routines still work flawlessly even if your internet connection goes down entirely.
- **Media Streaming:** Setting up personal media servers (like Plex or Jellyfin) to stream an owned movie and music collection without relying on changing streaming platform catalogs.

2. A Zero-Risk Sandbox for Career & Learning

For IT professionals, software engineers, or tech enthusiasts, a home lab is an essential playground.

- **Breaking Things Safely:** If you want to learn enterprise network configurations, cluster computing, or test out container environments like Docker, doing it on a production system at work is a massive risk. A home lab lets you experiment, break configurations, and learn how to rebuild them with zero real-world consequences.
- **Hypervisor Mastery:** It provides a space to learn type-1 hypervisors (like Proxmox VE or VMware ESXi) to split one physical machine into dozens of isolated virtual machines (VMs).

3. Network Control & Security

A home lab gives you granular control over what enters and exits your home network.

- **Network-Wide Ad Blocking:** By spinning up a local DNS sinkhole like Pi-hole or AdGuard Home, you can block trackers and advertisements for every device in the house (including smart TVs and mobile apps) before they even load.
- **Network Segmentation:** Setting up virtual local area networks (VLANs) to completely isolate untrusted internet-of-things (IoT) smart devices from your main personal computers and secure backups.

4. Hardware Longevity & The Hobby Element

Finally, there is a massive community built around the sheer joy of tinkering and sustainability.

- **Upcycling E-Waste:** A home lab doesn't have to be expensive. Many people buy cheap, off-lease corporate mini PCs or old desktop towers and give them a second life as a dedicated file server or firewall.

- **The "I Built This" Factor:** There is immense satisfaction in tuning hardware, writing scripts to automate daily digital tasks, and watching a complex system run perfectly on your own terms.

The transition from a simple, plug-and-play smart home to a full-blown home lab is a classic, step-by-step tech journey. It almost always follows the same trajectory: you start out wanting convenience, move toward wanting better customization and privacy, and end up deep in the world of enterprise infrastructure.

Here is exactly how that evolution happens.

Stage 1: The "Basic" Gateway (Cloud-Reliant Convenience)

It usually starts with a single impulse buy—a smart bulb, a plug, or a Wi-Fi camera.

- **The Setup:** You plug the device into the wall, download a proprietary app (like Tuya, SmartLife, or Wyze), connect it to your home Wi-Fi, and link it to a consumer ecosystem like Amazon Alexa or Google Home.
- **The Ceiling:** Everything works, but you quickly notice the limitations:
    - **The "Cloud Dependent" lag:** When you turn a light on, your command travels from your phone, up to a corporate server across the country, and back down to your bulb. If your internet drops, your "smart" house goes dark.
    - **App Fatigue:** You end up with five different apps for five different brands of devices, and getting them to talk to each other in complex ways is frustratingly restricted.

Stage 2: The Logic Leap to Voice Control & Local Hubs

Frustrated by limitations, you look for a centralized "brain" for the house. This is where open-source home automation software like **Home Assistant** enters the picture. You flash it onto a dedicated Raspberry Pi or a spare mini PC.

- **The Setup:** You pull your smart devices off the corporate cloud and bring them onto local control. Devices communicate instantly over your local network using protocols like Zigbee or Z-Wave via a USB coordinator stick.
- **The Voice Evolution:** Next, you want to eliminate cloud-connected smart speakers that listen to your house 24/7.
    - You start exploring private, local voice processing using protocols like **Wyoming**.
- XDA Developers

    - You deploy lightweight, custom hardware endpoints (**voice satellites**) around the house using inexpensive microcontrollers (like ESP32-S3 boards) paired with microphones.

- Reddit
    - When you say a command, the voice satellite streams the audio directly to your local hub. Your hub handles the Speech-to-Text and runs a local Large Language Model (LLM) via an engine like **Ollama** to parse exactly what you mean—completely offline.

Stage 3: The Graduation into a True Home Lab

Once you are running a local automation hub, processing real-time audio streams, and hosting offline AI models, you hit a hardware bottleneck. A tiny Raspberry Pi or a single desktop box can no longer handle the load. You need computational horsepower, stability, and proper data protection.

Welcome to the Home Lab.

- Virtualization & The Hypervisor

Consolidating Hardware

Instead of buying a new mini PC for every single project, you acquire a dedicated server or high-performance machine and install a hypervisor like **Proxmox VE**. This allows you to split that single physical machine into dozens of isolated Virtual Machines (VMs) and Docker containers.

- Network Segregation & Infrastructure

Security and Stability

With dozens of smart devices and voice clients hammering your network, consumer Wi-Fi routers begin to crash. You upgrade to enterprise-grade networking—adding managed switches, discrete Access Points, and a dedicated firewall router. You configure **VLANs** to completely isolate untrusted IoT gear away from your personal data.

- Centralized Storage & Backups

The Personal Cloud

Because your smart home configuration files, local AI models, and personal data are now incredibly valuable to you, you deploy a Network Attached Storage (NAS) array with multiple hard drives in a RAID configuration. If a drive fails, your system keeps running with zero data loss.

The Ultimate Takeaway: You started out just wanting to turn off a living room lamp with your voice. You ended up managing a miniature enterprise data center in your closet—complete with automated backups, network-wide security, containerized applications, and absolute control over your digital footprint.

The Trap

The classic home lab trap is building an incredibly powerful, hyper-customized local network that works flawlessly for you, but leaves your partner, guests, or a house sitter completely stranded if they just want to turn on the TV or adjust the temperature.

To bridge the gap between complex self-hosting (Wyoming voice clients, Pi-hole, Twingate, Deluge, Home Assistant) and everyday human usability, you need a strict design philosophy: **The plumbing should be invisible, and the interface must be intuitive.**

Here is how to structure your lab so it has full programmatic power underneath but feels completely transparent to a normal user.

A Wall-Mounted Panel: Your Usability Anchor

Having a webpage-driven tablet on the wall via Home Assistant is the absolute best starting point for a user-friendly smart home. It works because it provides a fixed, predictable physical location for control—much like a traditional thermostat or light switch.

How it helps:

- **Zero Learning Curve:** A guest doesn't need to download an app, configure a Twingate profile, or authenticate with your network. They just walk up and tap.
- **Immediate Feedback:** Unlike voice assistants (which can mishear) or automations (which happen in the background), a visual panel gives instant confirmation that a command worked.

How to make it better (The "Guest Test"):

The biggest mistake power users make with wall panels is trying to fit their entire Home Assistant dashboard onto one screen. If a normal person has to scroll through processor temperatures, Pi-hole block rates, or Deluge download speeds just to turn off a light, the design has failed.

- **The 3-Second Rule:** The default, ambient screen of your wall panel should only show the essentials:
    - Time/Weather, current indoor temperature, a "Turn Off All Lights" button, and a "Lock Front Door" button.

- **Use Sub-Views or Pop-ups:** Keep your complex entity cards, system monitors, and automation overrides hidden behind a "System Settings" tab protected by a simple PIN or tucked away on a separate dashboard that only you use.
- **Kiosk Mode:** Use the kiosk-mode frontend plugin in Home Assistant to completely hide the top navigation bar and sidebar. This prevents guests from accidentally clicking into the developer tools or editing your layout.

Streamlining the Rest of Your Ecosystem

Beyond the wall panel, you can design the rest of your home lab stack to keep the high-tech elements completely transparent.

1. Voice Assistants (Wyoming & Local Satellites)

When you build custom voice clients using ESP32-S3 hardware and the Wyoming protocol, the goal is to match or beat the seamlessness of commercial smart speakers.

- **The Room-Awareness Trick:** If a guest is standing in the guest bedroom and says "Turn on the light," they shouldn't have to specify "Turn on the guest bedroom overhead light." Ensure your Wyoming satellite entities are strictly assigned to the exact same **Area** in Home Assistant as the lights in that room.
- **Visual/Audio Cues:** Use small LEDs or basic piezoceramic buzzers on your hardware satellites to give immediate physical feedback when the wake word is detected and when a command is finished processing. If the system is silent and slow, a non-technical person will assume it's broken.

2. Network Infrastructure (Pi-hole & Twingate)

Your security and privacy tools should protect the house without breaking the internet for visitors.

- **The "Guest Network" Escape Hatch:** Do not put your guests' personal devices on your main secure VLAN where Pi-hole or restrictive local DNS rules might break their favorite apps or cause formatting issues on websites. Set up an entirely isolated Guest Wi-Fi network that bypasses your aggressive ad-blocking DNS entirely and routes straight out to the internet.
- **Seamless Remote Access with Twingate:** Twingate is fantastic for your remote access because it doesn't require a clunky whole-house VPN. Keep it installed strictly on your personal mobile devices and laptop to access Deluge or Proxmox on the go. Never try to force family members or guests to install a Twingate client just to

control the house remotely—instead, use Home Assistant's secure external URL (via Nabu Casa or a securely reverse-proxied local domain) for their phones.

3. The Power of "Fallback" Physical Controls

The ultimate golden rule of home lab design is **Physical Fallback**. If your main server tower dies, your hypervisor crashes, or you are in the middle of a massive software update, a normal human should still be able to live comfortably in the house.

- **Smart Switches Over Smart Bulbs:** Whenever possible, use smart relays (like Shelly modules behind the wall) or smart switches (like Zooz or Inovelli) rather than smart bulbs. This ensures that if someone habitually flips a physical wall switch, the light still turns on and off instantly, and the circuit doesn't cut power to your automation network.
- **Latching/Toggle Emulation:** Configure your smart switches so that a simple tap up/down always controls the load directly via physical hardware wires, while multi-taps (double click, hold) trigger your complex Home Assistant scenes or scripts.

By decoupling your interfaces (clean, simple wall panels and local voice hardware) from your infrastructure (Proxmox, Pi-hole, local LLMs), you get the best of both worlds: a bulletproof playground for your hobby, and a friction-free home for everyone else.

My Recommended Design Architecture

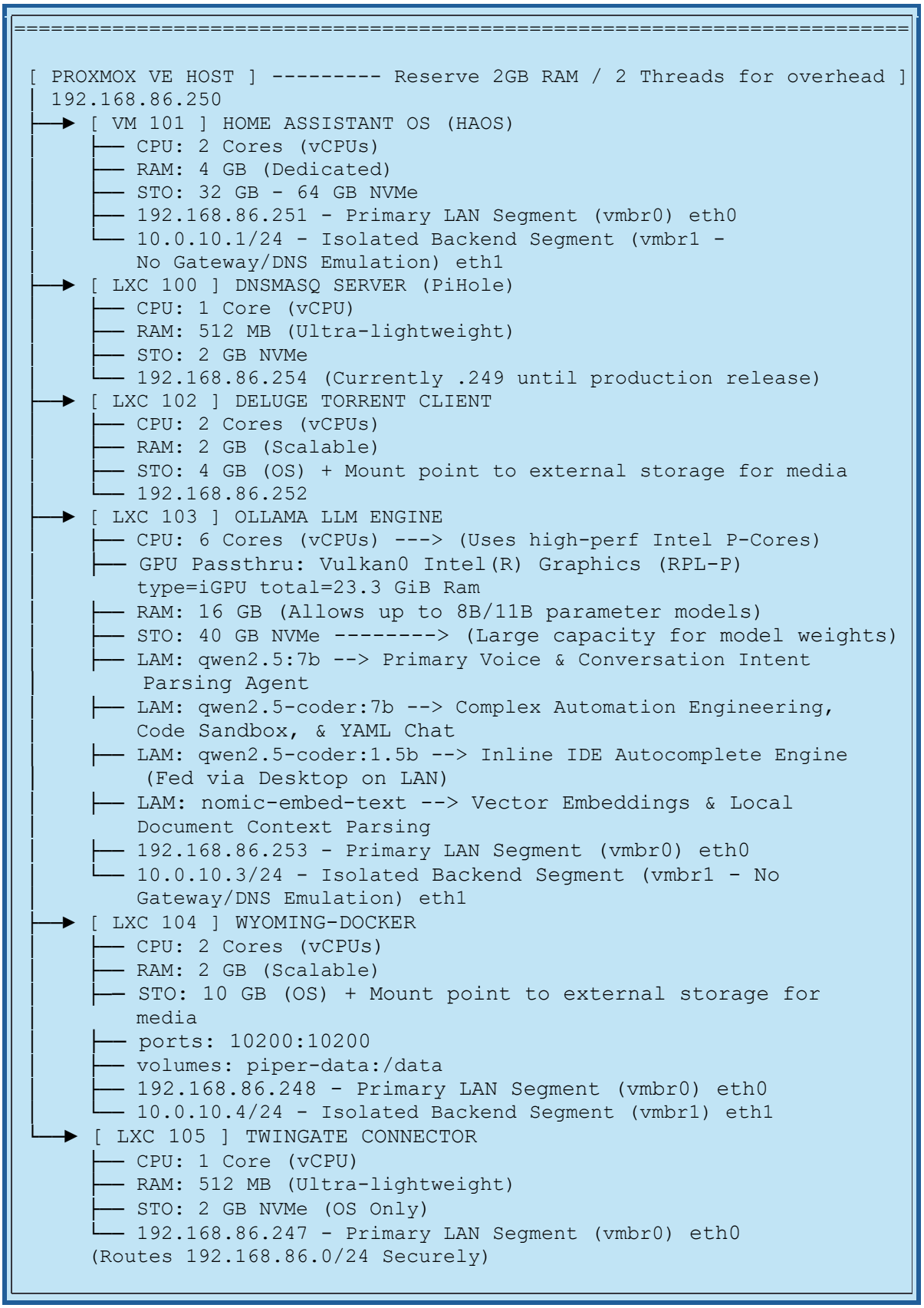
Basic Home Lab Configuration

A Home Lab self-contained in a single mini-computer.

Note: If you want to use this information in an AI window – select this section into your chat window to reference the entire configuration. This can be helpful if you want to ask any AI questions regarding your Home Lab configuration. Just tell your AI client "gemini" to load this config your reference in your conversation. It might be helpful to tell your AI conversation:

> "Please start a clean conversation, flush all buffers and caches and then load this config for reference."

```
=========================================================================
 Barnes Home Lab Configuration
 KAMRUI HYPER H2 MINI PC (10C / 16T | 32GB RAM / up to 4.9GHz)
 Proxmox VE 9.2.3 / Dedicated Ethernet / Intel Core i7-13620H
 Integrated GPU: Intel UHD Graphics (13th Gen) / 1TB PCIe NVMe SSD
```


[PROXMOX VE HOST] --------- Reserve 2GB RAM / 2 Threads for overhead]
192.168.86.250
[VM 101] HOME ASSISTANT OS (HAOS)
CPU: 2 Cores (vCPUs)
RAM: 4 GB (Dedicated)
STO: 32 GB - 64 GB NVMe
192.168.86.251 - Primary LAN Segment (vmbr0) eth0
10.0.10.1/24 - Isolated Backend Segment (vmbr1 - No Gateway/DNS Emulation) eth1
[LXC 100] DNSMASQ SERVER (PiHole)
CPU: 1 Core (vCPU)
RAM: 512 MB (Ultra-lightweight)
STO: 2 GB NVMe
192.168.86.254 (Currently .249 until production release)
[LXC 102] DELUGE TORRENT CLIENT
CPU: 2 Cores (vCPUs)
RAM: 2 GB (Scalable)
STO: 4 GB (OS) + Mount point to external storage for media
192.168.86.252
[LXC 103] OLLAMA LLM ENGINE
CPU: 6 Cores (vCPUs) ---> (Uses high-perf Intel P-Cores)
GPU Passthru: Vulkan0 Intel(R) Graphics (RPL-P) type=iGPU total=23.3 GiB Ram
RAM: 16 GB (Allows up to 8B/11B parameter models)
STO: 40 GB NVMe --------> (Large capacity for model weights)
LAM: qwen2.5:7b --> Primary Voice & Conversation Intent Parsing Agent
LAM: qwen2.5-coder:7b --> Complex Automation Engineering, Code Sandbox, & YAML Chat
LAM: qwen2.5-coder:1.5b --> Inline IDE Autocomplete Engine (Fed via Desktop on LAN)
LAM: nomic-embed-text --> Vector Embeddings & Local Document Context Parsing
192.168.86.253 - Primary LAN Segment (vmbr0) eth0
10.0.10.3/24 - Isolated Backend Segment (vmbr1 - No Gateway/DNS Emulation) eth1
[LXC 104] WYOMING-DOCKER
CPU: 2 Cores (vCPUs)
RAM: 2 GB (Scalable)
STO: 10 GB (OS) + Mount point to external storage for media
ports: 10200:10200
volumes: piper-data:/data
192.168.86.248 - Primary LAN Segment (vmbr0) eth0
10.0.10.4/24 - Isolated Backend Segment (vmbr1) eth1
[LXC 105] TWINGATE CONNECTOR
CPU: 1 Core (vCPU)
RAM: 512 MB (Ultra-lightweight)
STO: 2 GB NVMe (OS Only)
192.168.86.247 - Primary LAN Segment (vmbr0) eth0
(Routes 192.168.86.0/24 Securely)

Code 1: Barnes Home Lab Configuration

Other External Devices on The Local Network

If you want to reference your entire House configuration then this detail is also available to paste in your conversation. Please note that all of this information is also good if you want a quick table that in a simple format describes your Home Lab and its associated home integrations. At some point I will also add a simple table referencing the Home Assistant setup and configuration.

Home Storage Infrastructure

- *synology_nas_01:*
    - host_name: "surveillance"
    - ip_v4: "192.168.86.104"
    - mac_address: "00:11:32:d1:13:c5"
    - hardware: "Synology DS220+"
    - primary_role: "Dedicated Surveillance Station (10x IP Camera Streams)"
    - Surveillance System is configured with 12 PoE Cameras Located throughout the exterior of the home with 2 cameras inside the home (1 on each floor in the open spaces).
- *synology_nas_02:*
    - host_name: "Barnes-NAS-Primary"
    - ip_v4: "192.168.86.93"
    - mac_address: "90:09:d0:0d:cf:69"
    - host_name: "Barnes-NAS-Backup"
    - ip_v4: "192.168.86.105"
    - mac_address: "90:09:d0:0d:cf:6a"
    - hardware: "Synology DS220+"
    - primary_role: "Core Storage & Network Media Server"
    - active_services: [SMB File Shares, Synology Photos, Git Server, Emby, Plex]

smart_home_peripherals:

- *wall_mounted_panels:*
    - hostname: wall_panel
    - quantity: 1

    - primary_role: "Home Assistant Dashboard Terminal"
- *voice_satellites:*
    - quantity: 5
    - hardware_board: "Seeed Studio XIAO ESP32-S3 Sense"
    - onboard_specs: [2.4GHz Wi-Fi, BLE 5.0, OV2640 Camera, Digital Mic, 8MB PSRAM, 8MB Flash]
    - firmware_platform: "ESPHome (Wyoming API Target)"

end_user_clients:

- *mobile_devices:*
    - kathy_s23_ultra: { ip_v4: "192.168.86.8", mac: "92:11:e1:3f:b8:31" }
    - ed_s23_ultra: { ip_v4: "192.168.86.25", mac: "56:59:7c:69:03:6d" }
- *tablets:*
    - kats_ipad_2021: ip_v4: "192.168.86.73", mac: "34:31:8f:82:1f:3a" }
    - Eds-iPad { ip_v4: "192.168.86.21", mac: "60:d9:c7:cd:08:51" }

- *laptops:*
    - eds_laptop_wifi: { ip_v4: "192.168.86.143", mac: "fc:77:74:b7:a0:b6" }
    - laptop_siosk3q0: { ip_v4: "192.168.86.191", mac: "e4:e7:49:51:6a:ac" }
    - laptop_rm11o3dm_wf: { ip_v4: "192.168.86.24", mac: "b0:5c:da:97:97:73" }
    - laptop_rm11o3dm_ln: { ip_v4: "192.168.86.64", mac: "34:31:8f:82:1f:3a" }

Install Home Lab on a Mini-Computer

You may think this will be a complex thing to do but I did it in a week. After fine tuning things you should be able to do this in under a day and have complete running ready for a production rollout of the new Home Lab.

To run Proxmox, Home Assistant, Deluge, PiHole, Twingate Connector and Ollama together, a Proxmox VE (PVE) base layer is the best approach.

Proxmox VE base layer

When running there are some recommended resource allocations that you will be asked for. Use the details presented in Code 1: Barnes Home Lab Configuration when asked to allocate your resources efficiently to avoid crashing the host.

Step-by-Step Deployment Guide

Step 1: Download the latest Proxmox VE ISO from the official website located at https://www.proxmox.com/en/downloads/proxmox-virtual-environment.

1. Flash it to a USB drive using BalenaEtcher or Rufus.
2. Boot your KAMRUI Mini PC into the BIOS and disable Secure Boot. Also enable power on after power reset.
3. Boot from the USB, follow the on-screen installer prompts, and set a static IP address.

Step 2: Setup Putty/ssh for easy communication with your new Home Lab.

1. To set up passwordless SSH login to a Proxmox VE server using PuTTY, you need to generate a key pair on Windows, add the public key to Proxmox, and configure PuTTY to use the private key.
2. Here is the complete step-by-step process.
    - Generate an SSH Key
        - Open **PuTTYgen** on your Windows computer.
        - Under **Type of key to generate**, select **Ed25519** for modern security and speed.
        - Click the **Generate** button.
        - Move your mouse randomly over the blank area to create entropy for the key generation.
            - **Crucial**: Leave both passphrase fields completely blank to ensure a passwordless login.
        - Click **Save private key** and save it to a secure folder on your PC as a .ppk file Save the location for use below.
        - Keep the PuTTYgen window open; you will need the text string inside the top box labeled *Public key for pasting into OpenSSH authorized_keys file*.
    - Add the Public Key to Proxmox [1]
        - Launch PuTTY and log into your Proxmox server using your standard root username and password.
        - Create the directory for SSH keys if it does not already exist:
            - mkdir -p ~/.ssh
        - Set the appropriate folder permissions:
            - chmod 700 ~/.ssh
        - Open or create the authorization file with a text editor:
            - vi ~/.ssh/authorized_keys

- Go back to your PuTTYgen window, copy the entire string of text from the top box, and paste it into the PuTTY window. It must occupy a single line.
- save the file, and exit the vi editor ("enter :wq! ".
- Secure the authorization file permissions:
    - chmod 600 ~/.ssh/authorized_keys

- Configure PuTTY for Key Authentication
    - Close your current PuTTY session and open a brand new **PuTTY** window.
    - On the **Session** category page (on the left panel), type your Proxmox server's IP address (192.168.86.250) into the **Host Name** field.
    - In the left panel, navigate to **Connection ➔ Data**.
    - In the **Auto-login username** box, type root (or your chosen administrative username).
    - In the left panel, drill down to **Connection ➔ SSH ➔ Auth ➔ Credentials**.
    - Click **Browse** next to *Private key file for authentication* and select the .ppk file you saved above.
    - Navigate back to the **Session** category at the very top of the left panel.
    - Type a name for this connection profile (e.g., Proxmox_Server) in the **Saved Sessions** box and click **Save**.
    - Click **Open**. PuTTY will now log into your Proxmox console automatically using the private key, bypassing the password prompt completely.

Step 3: Use Helper Scripts for Easy Setup of the remaining products.

If you do not think you want that specific product installed, then you may skip that section. Just be aware that when you get to the tools and backup sections below the scripts may change. If you do end up changing the scripts remember that you can comment on the lines by using a "#" at the beginning of the line you do not want executed. That way later you know what to uncomment if you change your mind.

Once Proxmox is installed, open the Proxmox Web UI (https:// 192.168.86.250:8006), select your main node, open the Shell, and use the community-standard Proxmox VE Helper-Scripts listed below to deploy your services instantly.

PiHole (LXC 100)

Run this command to deploy a lightweight Deluge container:

```
  bash -c "$(curl -fsSL https://raw.githubusercontent.com/community-scripts/ProxmoxVE/main/ct/pihole.sh)"
```

Code 2: PiHoleInstallCommand.sh

When you copy and run this command it will start the install script. Always select the advance installation and if prompted select the GUI install method. When prompted enable all ssh keys and enable ssh.

Make the selections in the GUI by using the output presented below:

After you run the above code block you will get an output like this. - Verify that it matches.

```
Choose Advanced, set MAC address to BC:24:11:AA:BB:02, static IP
192.168.1.5/24, and Gateway 192.168.1.1, then finish.
        ⚙️  Using Default Settings on node Promox
        💡  PVE Version 9.2.2 (Kernel: 7.0.2-6-pve)
        🆔  Container ID: 100
        🖥️  Operating System: debian (13)
        📦  Container Type: Unprivileged
        💾  Disk Size: 2 GB
        🧠  CPU Cores: 1
        🛠️  RAM Size: 512 MiB
        🚀  Creating a Pihole LXC using the above default settings
       forwarding unbound
```

Code 3: PiHole Install Parameters and Response

Home Assistant OS (LXC 101 - HAOS VM)

Run this command in the Proxmox host shell to automatically build a fully functional Home Assistant VM: When prompted enable all ssh keys and enable ssh.

```
  bash -c "$(curl -fsSL https://raw.githubusercontent.com/community-scripts/ProxmoxVE/main/vm/haos-vm.sh)"
```

Code 4: HAOSInstallCommand.sh

When you copy and run this command it will start the install script.

Make the selections in the GUI by using the output presented below: When prompted enable all ssh keys and enable ssh.

After you run the above code block you will get an output like this. - Verify that it matches.

```
Please set MAC address to BC:24:11:AA:BB:01, IP Address to
192.168.86.251/24, and gateway to 192.168.86.1
        🆔  Virtual Machine ID: 101
        📦  Machine Type: q35
        💾  Disk Size: 32G
        🏠  Hostname: haos-17.3
        🖥️  CPU Model: KVM64
        🧠  CPU Cores: 2
        🛠️  RAM Size: 4096
        🌉  Bridge: vmbr0
        🔗  MAC Address: 02:93:F8:A8:3A:88
        🏷️  VLAN: Default
        ⚙️  Interface MTU Size: Default
        🌐  Start VM when completed: yes
        🚀  Creating a Homeassistant OS VM using the above default
settings
```

Code 5: HAOS Install Parameters and Response

Deluge (LXC 102):

Run this command to deploy a lightweight Deluge container:

```
bash -c "$(curl -fsSL https://raw.githubusercontent.com/community-
scripts/ProxmoxVE/main/ct/deluge.sh)"
```

Code 6: DelugeInstallCommand.sh

When you copy and run this command it will start the install script. Always select the advance installation and if prompted select the GUI install method.

Make the selections in the GUI by using the output presented below: When prompted enable all ssh keys and enable ssh.

After you run the above code block you will get an output like this. - Verify that it matches.

```
        ⚙️  Using Default Settings on node Promox
        💡  PVE Version 9.2.2 (Kernel: 7.0.2-6-pve)
        🆔  Container ID: 102
        🖥️  Operating System: debian (13)
        📦  Container Type: Unprivileged
        💾  Disk Size: 4 GB
        🧠  CPU Cores: 2
        🛠️  RAM Size: 2048 MiB
```

```
🚀 Creating a Deluge LXC using the above default settings
```

Code 7: Deluge Install Parameters and Response

Ollama AI Engine (LXC 103)

This uses the specialized ollama.sh hardware and software installer path:

```
bash -c "$(curl -fsSL https://raw.githubusercontent.com/community-scripts/ProxmoxVE/main/ct/ollama.sh)"
```

Code 8: OllamaInstallCommand.sh

When you copy and run this command it will start the install script. Always select the advance installation and if prompted select the GUI install method.

Make the selections in the GUI by using the output presented below: When prompted enable all ssh keys and enable ssh.

After you run the above code block you will get an output like this. - Verify that it matches.

```
🧩 Using Advanced Install on node Promox
💡 PVE Version 9.2.2 (Kernel: 7.0.2-6-pve)
🖥️ Operating System: debian
🌟 Version: 13
📦 Container Type: Unprivileged
🆔 Container ID: 103
🏠 Hostname: ollama
💾 Disk Size: 40 GB
🧠 CPU Cores: 6
🛠️ RAM Size: 16384 MiB
🌉 Bridge: vmbr0
📡 IPv4: 192.168.86.249/24
📡 IPv6: none
🗂️ FUSE Support: no
📡 TUN/TAP Support: no
📦 Nesting: Disabled
📦 Keyctl: Enabled
🎮 GPU Passthrough: yes
   Protection: no
💡 Timezone: America/Denver
🔍 Verbose Mode: no
```

Code 9: Ollama Install Parameters and Response

Load the Agents

In the Prolog Web GUI at http://192.168.86.250:8006 and select Datacenter - Promox - 103 (ollama) in the Server View column

In Container 103 (ollama) on node Promox window select the console

It should log you in without using a password.

- If it does not - then log in as root with the password of your Promox server
- Enter "passwd" and set the new password the same as your Promox server's password.
- The next time you log in it should not ask you for the password anymore.

In this window enter the commands listed below:

 # 1. Load the Primary Voice & Conversation Agent

- ollama pull qwen2.5:7b

 # 2. Load the Complex Automation & Code Sandbox Agent

- ollama pull qwen2.5-coder:7b

 # 3. Load the Inline IDE Autocomplete Engine

- ollama pull qwen2.5-coder:1.5b

 # 4. Load the Embedding Model

- ollama pull nomic-embed-text

 # Test the Primary Voice Agent

- echo "Hello, how are you today?" | ollama run qwen2.5:7b

 # Test the Code Sandbox

- echo "Write a simple Python function to add two numbers." | ollama run qwen2.5-coder:7b

 # Test the IDE Autocomplete Engine

- echo "def hello_world():" | ollama run qwen2.5-coder:1.5b

Test the Coding Engine

- curl http://localhost:11434/api/embeddings -d '{

  "model": "nomic-embed-text",

  "prompt": "Testing local document parsing"

}'

How to Connect Ollama to Home Assistant

For Voice: Install the official Ollama Integration in Home Assistant.

Open the Home Assistant Web Page by going to http://homeassistant.local:8123.

- Open your **Home Assistant Dashboard**.
- Navigate to **Settings** > **Voice Assistants**.
- Create an Assist Pipeline, and assign the qwen2.5:7b model to handle your conversational sentences and smart home exposures.

For Coding:

Point your development environment (like VS Code or Cursor) to your server's IP address (http://192.168.86.250:11434) using a local copilot extension to begin drafting automations locally.

It is highly recommended to use external VS Code IDEs instead of the embedded Home Assistant Integration. You can still connect remotely via ssh and do everything without bogging down your Home Assistant Environment.

Wyoming Docker Installation (LXC104 - Docker)

Step 1: Create a Docker LXC in Proxmox

The easiest way to get Docker running in an LXC is to use the community-maintained Proxmox Helper Scripts, which handle setup and dependencies automatically.

Execute the following command inside of a Putty-SSH window connected to the Promox host.

This must be done in a Putty Shell and not via the Promox web GUI because if the GUI gets changed or dies for some reason the script will be backgrounded. If it is waiting for a prompt, it will never exit.

```
bash -c "$(curl -fsSL https://raw.githubusercontent.com/community-scripts/ProxmoxVE/main/ct/docker.sh)"
```

Code 10: DockerInstallCommand.sh

When you copy and run this command it will start the install script. Always select the advance installation and if prompted select the GUI install method.

Make the selections in the GUI by using the output presented below:

Ensure you enable all ssh keys and enable ssh, **Docker Compose** AND install **Portainer** during the prompt, as you will need it to run your Wyoming services later.

After you run the above code block you will get an output like this. - Verify that it matches.

```
Using Advanced Install on node Promox
PVE Version 9.2.2 (Kernel: 7.0.2-6-pve)
Operating System: debian
Version: 13
Container Type: Unprivileged
Container ID: 104
Hostname: wyoming-docker
Disk Size: 10 GB
CPU Cores: 4
RAM Size: 2048 MiB
Bridge: vmbr0
IPv4: 192.168.86.249/24
IPv6: none
FUSE Support: yes
TUN/TAP Support: yes
Nesting: Enabled
Keyctl: Enabled
GPU Passthrough: no
Timezone: America/Denver
Verbose Mode: no
 If you installed Portainer, access it at the following URL:
https://192.168.86.249:9443
```

Code 11: Wyoming Docker Install Parameters and Response

Step 2: Set Up Wyoming Docker Services

Now that Docker is installed in your LXC, you can deploy your chosen Wyoming voice or speech-to-text tools.

1. **SSH** into your Docker LXC IP address do not use the Proxmox Web UI Console.
    - Configure PuTTY for Key Authentication
        - Close your current PuTTY session and open a brand new **PuTTY** window.
        - On the **Session** category page (on the left panel), type your Wyoming-docker IP address (192.168.86.248) into the **Host Name** field.
        - In the left panel, navigate to **Connection ➔ Data**.
        - In the **Auto-login username** box, type root (or your chosen administrative username).

- In the left panel, drill down to **Connection ➔ SSH ➔ Auth ➔ Credentials**.
- Click **Browse** next to *Private key file for authentication* and select the .ppk file you saved above.
- Navigate back to the **Session** category at the very top of the left panel.
- Type a name for this connection profile (e.g., Proxmox_Wyoming) in the **Saved Sessions** box and click **Save**.
- Click **Open**. PuTTY will now log into your Wyoming console automatically using the private key, bypassing the password prompt completely.

2. Create a docker-compose.yaml file to manage your Wyoming services. For example, to set up the **Piper Text-to-Speech** and **Whisper Speech-to-Text** services, run:

```
mkdir -p ~/wyoming && cd ~/wyoming
vi docker-compose.yaml
```

Code 12: WyomingDockerCompose.sh

Paste the following configuration, making sure to replace paths and voices with your preferences:

```
services:
  piper:
    image: rhasspy/wyoming-piper
    container_name: wyoming-piper
    restart: unless-stopped
    volumes:
      - ./piper-data:/data
    ports:
      - "10200:10200"
    command: --voice en_US-lessac-medium

  whisper:
    image: rhasspy/wyoming-whisper
    container_name: wyoming-whisper
    restart: unless-stopped
    volumes:
      - ./whisper-data:/data
    ports:
      - "10300:10300"
    command: --model tiny-int8 --language en
```

Code 13: DockerCompose.sh *Code 14: Wyoming-Docker Portainer Stack*

Spin up the containers by typing the following command in you Putty ssh window:

```
docker compose up -d
```

Code 15: DockerCompose.sh

Step 3: Connect in Home Assistant

Once your Docker containers are running on the Proxmox LXC, you can easily link them to Home Assistant:

- Open your **Home Assistant Dashboard**.
- Navigate to **Settings** > **Devices & Services**.
- Click **Add Integration** and search for **Wyoming Protocol**.
- Enter the IP address of your Proxmox Docker LXC (192.168.86.248/24) and the corresponding port for the service (e.g., 10200 for Piper or 10300 for Whisper) integrations and associate the following links.

Portainer/Docker changes

```
Home Assistant Wyoming Integration:
Settings -> Devices & Services -> Add Integration -> Wyoming
   link ports 10200 and 10201.

For Voice: Install the official Ollama Integration in Home Assistant.
  Go to Settings -> Voice Assistants, create an Assist Pipeline,
    and assign the qwen2.5:7b model to handle your conversational
sentences and smart home exposures.

For Coding: Point your development environment (like VS Code or Cursor)
  to your server's IP address (http://<your-server-ip>:11434)
  using a local copilot extension to begin drafting automations locally.

NOTE: I found that running VS Code inside of HAOS is a bad idea.
      I reccommend running the VS Code on a client and point it to HAOS
      You can use a Remote Connection via to see the filesystem.
      Do not mount the root fs - It will hang the server with indexing.
```

Code 16: HAOS-Wyoming Integrations

Twingate Connector (LXC 105)

Installing Twingate on a Proxmox server is straightforward and highly automated, taking only a few minutes to complete.

Before running the script, you must generate the necessary Connector tokens from your Twingate Admin Console.

Phase 1: Generate Twingate Connector Tokens

1. Log in to your **Twingate Admin Console https:// twingate.com**.
2. If you don't have an account you will need to create one.
3. After you are logged in Go to **Network** > **Remote Networks** and select your remote network (or create one).
4. Click to **Add a Connector** and select the **Manual** deployment type.
5. Scroll down and click **Generate Tokens**.
6. Copy the **Access Token** and **Refresh Token** (you will need these below)

```
bash -c "$(curl -fsSL https://raw.githubusercontent.com/community-scripts/ProxmoxVE/main/ct/twingate-connector.sh)"
```

Code 17: TwingateConnectorInstallCommand.sh

Paste the following configuration, making sure to replace paths and voices with your preferences:

```
  💡 Missing jq for script status check. Continuing without status verification.
  🧩 Using Advanced Install on node Promox
  💡 PVE Version 9.2.3 (Kernel: 7.0.6-2-pve)
  🖥️  Operating System: ubuntu
  🌟 Version: 24.04
  📦 Container Type: Unprivileged
  🆔 Container ID: 105
  🏠 Hostname: twingate-connector
  💾 Disk Size: 5 GB
  🧠 CPU Cores: 1
  🛠️  RAM Size: 1024 MiB
  🌉 Bridge: vmbr0
  📡 IPv4: 192.168.86.247/24
  📡 IPv6: none
  🗂️  FUSE Support: no
  📦 Nesting: Enabled
  📦 Keyctl: Enabled
  🎮 GPU Passthrough: no
  💡 Timezone: America/Denver
  🔍 Verbose Mode: no
  🚀 Creating an LXC of Twingate-Connector using the above advanced settings

  ✔️  Updated Container OS
```

```
      Please enter your access token:
        enter your access token from above.

      Please enter your refresh token:
        enter your access token from above.

      Please enter your network name:
        enter your network name from above.
```

Code 18: Twingate Connector Install Parameters

Post Install Scripts

Proxmox Scripts:

Setting Vi as your default editor

To set **vi** as the default editor across all containers in your Barnes Home Lab, you can use a bash loop script run directly from your Proxmox host terminal.

Since vi is natively included in almost every Linux distribution (including minimal Debian, Ubuntu, and Alpine templates), this script will work universally without needing to install extra packages.

The Automation Script

Run this complete script on your Proxmox host terminal:

```
for vmid in $(pct list | awk '{print $1}' | grep -E '^[0-9]+$'); do
    echo "---------------------------------------"
    echo "Updating Container $vmid..."
    # 1. Update root user's profile
    pct exec $vmid -- sh -c 'echo "export EDITOR=\"vi\"" >> /root/.bashrc'
    # 2. Update system-wide environment fallback
    pct exec $vmid -- sh -c 'echo "export EDITOR=\"vi\"" >> /etc/profile'
    echo "Container $vmid updated successfully."
done
echo "---------------------------------------"
echo "All containers configured to use vi as the default editor."
```

Code 19: SetVIasDefault.sh

What This Script Does

- **Finds All LXCs**: Loops through your existing containers (LXC 100, 102, 103, 104, and 105).
- **Updates Root Profiles**: Appends the environment variable to /root/.bashrc for root terminal sessions.

- **Sets a System-Wide Fallback**: Appends the variable to /etc/profile to capture system-wide processes, automation tasks, and alternative shell instances. [1]

How to Apply It to Future Containers

To guarantee that any **new** containers you build in the future also default to vi, run this single command on your Proxmox host to update the global skeleton directory:

```
echo 'export EDITOR="vi"' >> /etc/skel/.bashrc
```

Code 20: ExportVIasDefault.sh

Twingate Connector

To safely disable password-based SSH logins and prevent brute-force attacks on Proxmox, you must **verify your key-based login is working, modify the SSH daemon configuration, and restart the service**.

Phase 1: Verify Your Current Setup

- **Keep your current session open.** Do not close your active SSH window until you verify the new changes work.
- **Test the key login in a new window.** Open a second PuTTY window, load your saved session, and click **Open**.
- **Confirm success.** Ensure you log in automatically without any password prompt before proceeding.

Phase 2: Modify the SSH Configuration File

1. In your active Proxmox terminal, open the SSH configuration file:
    - vi /etc/ssh/sshd_config
2. Press Ctrl + W to search for **PasswordAuthentication**.
3. Remove any # symbol from the beginning of the line to uncomment it.
4. Change the value from yes to no:

    PasswordAuthentication no
5. Press Ctrl + W to search for **ChallengeResponseAuthentication** (or **KbdInteractiveAuthentication** on newer Proxmox versions).
6. Ensure it is also set to no:

    ChallengeResponseAuthentication no

7. Save and exit the file by pressing Ctrl + O, Enter, then Ctrl + X. [1, 2, 3, 4, 5]

Phase 3: Apply and Test the Changes

1. Apply the new settings by restarting the SSH service:
    - systemctl restart sshd
2. **Keep this configuration window open.**
3. Launch a completely new instance of PuTTY.
4. Attempt to log in to your Proxmox server **without** loading your private key.
5. Confirm the server rejects the connection with a No supported authentication methods available error.
6. Open another PuTTY window **with** your private key loaded to confirm you can still access the server.

If both tests pass, you can safely close your original terminal window. Your Proxmox host is now immune to password brute-force attacks.

Bringing it all together

Setting up a Trade Name (Optional)

Reserve your spot now if you think you want to present an image to the external world

Creating a Domain Name

Create an account on Cloudfare.com

I Pointed mine to my github account for login purposes since it is easy to remember.

Step 1: Register Your New Domain

Cloudflare operates as an official domain registrar, selling domain names at wholesale cost without added markups.

1. Log in to your Cloudflare Dashboard.
2. On the left sidebar of your account home page, click **Domain Registration** > **Register Domains**.
3. Type the domain name you want in the search bar and press **Enter**.

4. Check the availability list. Click **Add to cart** next to your preferred extension (e.g., .com, .org, .net).
5. Click **Purchase Now**.
6. Select your registration term length (e.g., 1 year) and fill out your **Contact Information** (required by ICANN for domain ownership logs). Cloudflare provides free WHOIS privacy, so your personal details will stay hidden from the public.
7. Enter your payment details and click **Finalize Checkout**.

Step 2: Access Your New DNS Dashboard

Because you bought the domain directly through Cloudflare, the nameservers are already configured automatically. You do not need to change any registrar settings to start using it.

1. Go back to your main **Cloudflare Dashboard** home page.
2. Click on your newly purchased domain name from your website list.
3. In the left sidebar, navigate to **DNS** > **Records**.
4. You are now inside your active DNS management pane. From here, you can click **Add Record** to point your new domain to your web host, GitHub Pages, or an email provider using the exact setup steps we discussed previously.

Setting up a Public Repository on Github

To share files and host a web page on GitHub, you need to create a **repository**, upload your files, and activate **GitHub Pages**.

Follow this step-by-step process to get everything up and running.

Step 1: Create a Public Repository

GitHub Pages requires a public repository to host web pages for free.

Log in to your GitHub account.

Click the **+** (plus) icon in the top-right corner and select **New repository**.

Configure your repository settings:

- **Repository name**: Type a name (e.g., my-web-site).
- **Public/Private**: Select **Public**.
- **Initialize this repository with**: Check the box for **Add a README file**.
- Click **Create repository**.

Step 2: Upload Your Files

Your main web page must be named exactly index.html (all lowercase) for GitHub to recognize it as the homepage.

Inside your new repository

- click the **Add file** dropdown button and choose **Upload files**.

Drag and drop your website files into the box. Ensure your main page is named index.html. You can also drop any other files (like PDFs, images, or ZIPs) you want to share.

Scroll down to **Commit changes**

- add a short note describing your upload, and click **Commit changes**.

Step 3: Turn on GitHub Pages

Now, tell GitHub to turn those uploaded files into a live website.

- Click the **Settings** tab at the top of your repository.
- In the left-hand sidebar menu, scroll down to the *Code and automation* section and click **Pages**.
- Under **Build and deployment** > **Source**, keep it set to **Deploy from a branch**.
- Under **Branch**, click the dropdown that says *None*, change it to **main** (or *master*), leave the folder as / (root), and click **Save**.
- Wait about 1 to 2 minutes. Refresh the page, and a live link will appear at the top of the Pages section (e.g., https://github.io).

Step 4: Share Your Files and Site

To share the live website: Give people the URL generated in **Step 3**.

To share direct file downloads: Append the exact file name to your website URL. For example, if you uploaded a file named brochure.pdf, your shareable download link is: https://github.io

Pointing it to Git Hub

To point a custom domain managed by **Cloudflare** to a **GitHub Pages** web page, you need to configure specific **DNS records** in Cloudflare and input your domain inside your **GitHub repository settings**.

Follow this complete step-by-step walkthrough to link your domain.

Step 1: Configure DNS Records in Cloudflare

First, you need to map your root domain (e.g., example.com) and your subdomain (e.g., www.example.com) to GitHub's infrastructure.

1. Log in to your Cloudflare Dashboard and select your domain name.
2. In the left-hand sidebar, navigate to **DNS** > **Records**.
3. Delete any existing A or CNAME records that are currently pointing your root (@) or www subdomains to another host.
4. Add four separate **A records** for your root domain using GitHub's official IP addresses:
    - Click **Add Record**, choose Type **A**, set the Name to @, **Proxy status** set to **DNS Only** (grey cloud), and input the IPv4 address 185.199.108.153.
    - Repeat this process to add the remaining three IP addresses:
        - 185.199.109.153
        - 185.199.110.153
        - 185.199.111.153
5. Add a **CNAME record** to handle traffic arriving via www:
    - Click **Add Record**, choose Type **CNAME**, set the Name to www, and set the Target to your default GitHub Pages URL (e.g., yourusername.github.io).
6. **Important Proxy Status Rule**: Ensure the **Proxy status** toggle for all five newly created records is set to **DNS Only** (grey cloud) during the initial setup. This allows GitHub to verify the domain ownership and successfully issue an SSL certificate. You should change this to *Proxied* (orange cloud) later to enforce Cloudflare's security protections.

Name	Type	Content	Proxy status	TTL
smart-home-edg...	A	185.199.109.153	DNS only	Auto
smart-home-edg...	A	185.199.108.153	DNS only	Auto
smart-home-edg...	A	185.199.110.153	DNS only	Auto
smart-home-edg...	A	185.199.111.153	DNS only	Auto
www.smart-hom...	CNAME	barnes-dev.github.io	DNS only	Auto

Figure 3: Cloudflare DNS Configuration

Step 2: Configure Your Custom Domain in GitHub

Now, inform GitHub that your repository should accept traffic originating from your custom domain.

1. Log in to GitHub and open the repository hosting your web page.
2. Click on the **Settings** tab located at the top of your repository navigation bar.
3. Scroll down the left sidebar menu and click on **Pages**.
4. Under the **Custom domain** section, type your root domain (e.g., example.com) into the text box and click **Save**.
5. GitHub will instantly create a file named CNAME in the root of your main repository branch.
6. Wait a few minutes (1-15) for the **DNS check** on the GitHub page to clear and the Enforce HTTPS box to become available.
    - The "Enforce HTTPS" checkbox on GitHub is grayed out for one specific reason: **GitHub hasn't finished provisioning your SSL/TLS security certificate yet.**
    - When you change your DNS settings (like fixing that CNAME record), GitHub has to talk to Let's Encrypt to issue a secure certificate for your custom domain. Until that process completes, it won't let you check that box.
7. Once it successfully clears, check the box labeled **Enforce HTTPS** to secure your live site.

Update Twingate and PiHole

Work in Progress

Bill of Materials

Work in Progress

Table on how much does all this costs - Include Bill of Materials, best location to buy

Server, Accounts and Licenses

Work in Progress

External Accounts needed for complete Setup

Name	WebPage	Type Is this something that is for administrative, discounts, development, warranty based	Fee

- Homeassistant
- Google
- Email address
- Cloudfare
- Github
- Adblock
- Twingate
- Deluge

Resource Budget Summary

To prevent resource exhaustion, your configurations should sum up to just under your maximum physical hardware limits:

- Total CPU Cores Allocated: 11 / 16 Threads (Safe zone. Proxmox can share CPU cores dynamically across containers without performance degradation).
- Total RAM Allocated: 24.5 GB / 32 GB (Leaves a comfortable 7.5 GB cushion for the Proxmox file system cache and host overhead).

Critical Optimization Tips

- CPU Pinning: Do not pin specific cores. Let the Proxmox scheduler pass requests directly to the Intel Thread Director. This ensures Ollama dynamically hits the high-performance P-cores when crunching data.
- Ollama Storage: Place the Ollama storage container on the local NVMe drive. LLMs read massive files into RAM during startup, and network drives will cause slow load times.

The Ultimate Server Setup: Ollama + Quad Qwen Models

Instead of trying to find one compromise model, you should run a central AI backend server (like a Mini PC, a home server with an Nvidia GPU, or a robust NAS) running Ollama. You will load four distinct, highly specialized models from the same family:

Key Enhancements Added:

- Inbound Client Routing: Explicitly maps how your physical XIAO ESP32-S3 Voice Satellites route through your Wyoming-Docker (VMID 104) translation layer before striking the Ollama engine. Middle Execution Layer: Integrates your Home Assistant OS (VMID 101) as the core "Action & Response Layer" bridging the gap between voice intent and physical execution (like updating your iPads or firing smart home APIs).
- Exact Model Alignment: Swapped out abstract placeholders for the actual models running on your hardware (qwen2.5:7b for base/voice tasks and qwen2.5-coder:7b / 1.5b for your automation and coding workflows).

For Voice: Qwen 2.5 Instruct (3B and 7B)

1. Why it wins for Voice:
    a. The Home Assistant Core Team's native AI Agent feature relies heavily on "tool calling"—the ability of the AI to recognize that when you say "Turn off the living room lights," it shouldn't just reply, it should execute code.
    b. Qwen 2.5 is arguably the best open-source model under 10B parameters for precise tool calling.
    c. The Performance Factor: Using the 3B or 7B variant keeps your Time-To-First-Token (TTFT) practically instant. Your voice commands will execute without annoying multi-second delays.

For Code Generation: Qwen 2.5 Coder (7B and 14B)

2. Why it wins for Code: Qwen 2.5 Coder beats out nearly every other local model of its size on coding benchmarks like HumanEval. It natively understands YAML spacing, entity syntax, Home Assistant Blueprints, and Jinja2 templating.
    a. How to use it: You can tie this server backend directly into code extensions like Continue.dev inside VS Code, or open-source terminal agents like OpenCode. It reads your configuration directory and safely spits out complex home automation scripts.

Performance Analysis

🧠 Memory Analysis (RAM)

- Total Capacity: 32 GB
- Total Maximum Allocation: 27 GB (includes your 2GB host safety margin)
- True Available Buffer: 5 GB completely unallocated

Why it works: Your most massive workload is the Ollama engine taking up 16GB. This safely fits your 7B parameter models (qwen2.5:7b), which require roughly 4.8GB to 8GB of vRAM/RAM workspace. Keeping 5GB entirely free prevents your host node from swapping to disk or crashing under heavy background automation processing.

⚡ Processor Analysis (CPU)

- Total Capacity: 10 Cores / 16 Threads
- Total Provisioned vCPUs: 14 vCPUs
- Overprovisioning Factor: 0.88x (Under-provisioned)

Why it works: In Proxmox virtualization, an overprovisioning metric under 1.5x to 2.0x is considered incredibly healthy. Because containers like your DNS server (PiHole) and Twingate sleep at nearly 0% CPU usage when idle, your 6 allocated performance cores are ready to burst to full speed the exact millisecond your local voice assistant triggers an Ollama text intent parser generation request. [1]

🛠️ Hardware Optimization Verdict

- Intel i7 CPU Cores
    - *Current Allocation:* 14 / 16 Virtual Threads
    - *Health Status:* 🟢 Excellent

    - *Upgrade Required?:* No — Dynamic bursting is unhindered.
- System RAM Block
    - *Current Allocation:* 27GB / 32GB Total
    - *Health Status:* 🟢 Balanced
    - *Upgrade Required?:* No — Safe buffer remains for 7B models.
- PCIe NVMe Storage
    - *Current Allocation:* ~100GB / 1TB Total
    - *Health Status:* 🟢 Massive
    - *Upgrade Required?:* No — Plenty of room for future LLM weights.

Your server is not going to starve for resources, and here is the mathematical reason why.

In Proxmox, there is a big difference between how Virtual Machines (VMs) and Linux Containers (LXCs) handle CPU allocation:

VMs (like your Home Assistant 101) "reserve" power: They act more like independent hardware slices.

LXCs (like Pi-hole, Deluge, Ollama, and Wyoming) "share" power: When you assign 4 cores to an LXC, you aren't permanently locking those 4 cores away from the host. You are simply setting a speed limit. It means “this container is allowed to use up to 4 cores worth of processing power if it needs it, but when it’s idling, the host can give those cores to anyone else.”

The Reality of Your Cluster's Idle State

- Right now, your smart home stack operates mostly in an idle or low-utilization state:
- Pi-hole (1 core limit): Uses less than 1% of a single core just translating text addresses.
- Deluge (1 core limit): Only spikes when actively verifying a massive file download.
- Ollama (6 core limit): Sits at 0% CPU utilization until you actively send a chat prompt to your local AI.
- Wyoming (4 core limit): Sits at 0% CPU utilization until you physically say a wake word to a Wio-S3 satellite.

Because all of these services won't hit 100% processing capacity at the exact same fraction of a second, your 16 physical host cores will easily distribute the weight. Even if

you talk to your voice assistant while your local AI is thinking, Proxmox will seamlessly shift the execution threads around across your physical silicon.

Your system is perfectly balanced. Once that script finishes compiling and gives you your command prompt back, let me know and we will get the containers configured!

Backup Routines

Gravity Sync

To synchronize your primary Proxmox Pi-hole (LXC 100) and your backup Raspberry Pi Pi-hole (the Pi running at 192.168.86.253), you will use the official Gravity-Sync utility.

Since Gravity Sync 4.0+, the utility operates as an Active-Peer relationship. This means the software is installed on both machines, automatically monitoring changes on either end and keeping your blocklists, local DNS, and DHCP leases in perfect sync.

Before starting, ensure that SSH is enabled and working on both your Proxmox Pi-hole container and your backup Raspberry Pi.

Step 1: Run the Gravity Sync Installer on BOTH Nodes

You must execute the core installation command on both your Proxmox Pi-hole LXC terminal and your backup Raspberry Pi terminal.

Run this exact command on both systems:

```
curl -sSL https://raw.githubusercontent.com/vmstan/gs-install/main/gs-install.sh | bash
```

Code 21: GravitySyncInstaller.sh

(The script automatically verifies system prerequisites and sets up passwordless sudo privileges so that background synchronization scripts do not get stuck demanding a root password).

Step 2: Configure the Peer Link (Run on the Backup Pi)

Once the software setup completes, the installer will automatically launch the setup configuration wizard. If it does not, manually launch it on your Backup Raspberry Pi:

```
curl -sSL https://raw.githubusercontent.com/vmstan/gs-install/main/gs-install.sh | bash
```

Code 22: GravitySyncConfiguration.sh

The configuration utility will guide you through a visual text wizard. Fill out the prompts with your network parameters:

- Remote Pi-hole IP Address: Enter your Proxmox LXC 100 IP (192.168.86.254 or your staging .249 address).
- Remote SSH User: Enter the system administrator username of your Proxmox container (e.g., root).
- Password authentication: The wizard will prompt you to enter the remote password once. It will then automatically generate and exchange secure SSH keys (gravity-sync.rsa) so that it never needs your password again.

Step 3: Run the Initial Database Alignment

To ensure your custom configurations match perfectly before turning on automation, execute a manual pull down from the master node to the backup node.

Run this command on your Backup Raspberry Pi:

- gravity-sync pull

(This tells Gravity Sync to download your current primary blocklists, local custom DNS names, and rules to the Raspberry Pi).

To confirm the sync engine reads both nodes as structurally identical, run:

- gravity-sync compare

The output should report a perfect match across your databases.

Step 4: Automate the Synchronization

To prevent having to sync the nodes manually, tell Gravity Sync to handle changes in the background.

Run this final command on your Backup Raspberry Pi:

- gravity-sync auto

This tool will generate an optimal background system cron schedule. By default, it schedules an automated comparison check every 5 minutes. If you add a new custom local DNS record inside your Proxmox Pi-hole, the background automation engine will spot it and push it over to your Raspberry Pi seamlessly.

Diagnostics

Run these diagnostics directly on your Backup Raspberry Pi terminal:

1. View the Live Gravity-Sync Tail Log

Gravity-Sync includes a built-in log viewer that streams sync events, SSH verification routines, and database comparisons to your terminal in real-time. Run this command to inspect the logs:

- gravity-sync logs

What to look for: Look for entries marked [INFO] or [SYNC] showing database comparison passes returning clean results with zero transfer flags.

To Exit: Press CTRL + C on your keyboard to close the live view stream.

2. Inspect the Past 30 Sync Operations

If you want to view a history of recent automated runs without waiting for the next 5-minute interval, you can read the raw application log file using tail:

- tail -n 30 /etc/gravity-sync/gravity-sync.log

Each entry will show a timestamp tracking when the script woke up, connected to your Proxmox Pi-hole, and completed its database comparison.

3. Verify the Automated Cron Job Configuration

To confirm that the Gravity-Sync automation engine successfully registered its execution pattern with your Linux operating system task scheduler, print out your system's current crontab rules:

- crontab -l

Expected Output:

You should see a line near the bottom of the list that looks like this:

- text*/5 * * * * /usr/local/bin/gravity-sync auto >/dev/null 2>&1

The */5 syntax indicates that the system is instructed to execute the sync verification script exactly every 5 minutes, 24 hours a day.

4. Test an Automated Run InstantlyTo force an explicit background automation run right now to verify that permissions and directories are clear, execute:

- gravity-sync auto

If the command finishes and drops you back to a clean command prompt without throwing a permission error or an SSH timeout warning, your background task automation is working perfectly.

Post Installation Processing

HAOS Internal Network Connectivity

These are scripts to ensure the network is defined correctly

You must run these commands inside the main Proxmox host shell (the top-level hypervisor layer), not inside the individual containers or virtual machines.

- Where to run them in the Proxmox Web UI:
- Click on your main Proxmox Node (the physical machine name at the very top of your left-hand sidebar tree).
- Click the >_ Shell button in the top right menu bar.
- Paste the commands there.
- Alternatively, if you are using an external terminal client like PuTTY, Terminal, or VS Code, you can SSH directly into your host IP (192.168.86.250) as the root user and execute them there.

Putty SSH is the recommended way since if you accidently click out of the Shell window the script will get backgrounded and may not finish.

NOT Sure what to do with this yet

```
vi /etc/network/interfaces

>>>START OF FILE
# Loopback network interface
auto lo
iface lo inet loopback

# Physical Network Port (Connected to your router)
iface eno1 inet manual

# vmbr0: Primary LAN Segment (Bridged to physical network)
auto vmbr0
iface vmbr0 inet static
        address 192.168.86.250/24
        gateway 192.168.86.1
        bridge-ports eno1
        bridge-stp off
        bridge-fd 0
```

```
        dns-nameservers 192.168.86.254 1.1.1.1
#Comment: Temporary fallback dns setup until production release of LXC
100

# vmbr1: Isolated Backend Segment (No physical ports, No Gateway, No DNS
Emulation)
auto vmbr1
iface vmbr1 inet static
        address 10.0.10.250/24
        bridge-ports none
        bridge-stp off
        bridge-fd 0
#Comment: Completely isolated internal software switch for secure node
inter-communication
>>>END OF FILE

>>> BASH Script Start
# Install the Proxmox network reload dependency if not present
apt-get update && apt-get install -y ifupdown2
# Verify the text configuration syntax is valid
ifquery --check -a
# Apply the network changes instantly
ifreload -a
# Verify both interfaces are up and showing correct IP addresses
ip addr show vmbr0
ip addr show vmbr1
>>> BASH Script End
>>> BASH Script START
# VM 101 - Home Assistant OS
qm set 101 --net0
virtio,bridge=vmbr0,ip=192.168.86.251/24,gw=192.168.86.1
qm set 101 --net1 virtio,bridge=vmbr1,ip=10.0.10.1/24

# LXC 103 - Ollama Engine
pct set 103 --net0
name=eth0,bridge=vmbr0,ip=192.168.86.253/24,gw=192.168.86.1
pct set 103 --net1 name=eth1,bridge=vmbr1,ip=10.0.10.3/24

# LXC 102 - Deluge Client
pct set 102 --net0
name=eth0,bridge=vmbr0,ip=192.168.86.252/24,gw=192.168.86.1

# LXC 100 - Pi-Hole Server (Staging IP assigned)
pct set 100 --net0
name=eth0,bridge=vmbr0,ip=192.168.86.249/24,gw=192.168.86.1

# LXC 104 - Wyoming Docker
pct set 104 --net0
name=eth0,bridge=vmbr0,ip=192.168.86.248/24,gw=192.168.86.1
pct set 104 --net1 name=eth1,bridge=vmbr1,ip=10.0.10.4/24

>>> BASH Script End
```

Code 23: Proxmox Network Setup

Since used the popular Proxmox VE Helper-Scripts (like the popular community scripts by tteck/community-scripts), you can easily customize them before or during deployment.

Here is how to seamlessly inject your exact network architecture, memory targets, and core configurations into those automated environments.

1. The Pre-Provisioning Trick (Easiest Method)

When using the custom scripts, choose the Advanced Settings option when prompted during the interactive console setup. This lets you manually type in your resource limits and network settings:

Set IPv4 Address to your exact layout (e.g., 192.168.86.253/24 for Ollama).

Set Gateway to 192.168.86.1.

Leave the default bridge as vmbr0.

2. Post-Script Configuration Injection

Because helper scripts typically only create the primary interface (eth0 on vmbr0), you can instantly wire up your isolated backend (vmbr1), lock down your resource caps, and configure your pass-through requirements using these post-install snippets.

This is the best way to ensure the entire system is set up correctly.

Run these directly on your Proxmox host terminal right after a script finishes:

1. For Home Assistant OS (VM 101)

Run this because you used the HAOS VM installer to attach your second isolated network adapter:

>>> BASH Script start

```
# Add the second network interface for the isolated 10.0.10.x backend
qm set 101 --net1 virtio,bridge=vmbr1
# Enforce your custom CPU and RAM rules
qm set 101 --cores 2 --memory 4096
```

>>> BASH Script end

2. For Deluge (LXC 102) & Pi-Hole (LXC 100)

Run these to adjust your resource caps to match your exact diagram profile:

>>> BASH Script start

```
# Deluge Limits

pct set 102 --cores 2 --memory 2048

# Pi-Hole Limits (Ensuring it uses the temporary staging IP)

pct set 100 --cores 1 --memory 512 --net0
name=eth0,bridge=vmbr0,ip=192.168.86.249/24,gw=192.168.86.1

pct restart 102

pct restart 100
```

>>> BASH Script end

Step 1: Identify the UUID or Path of your External DriveRun this command in the Proxmox host shell to find your external storage drive:

- lsblk -o NAME,FSTYPE,UUID,SIZE,MOUNTPOINTS

Look for your external drive partition (e.g., sda1).

Copy its UUID (a long string of letters and numbers).

Using the UUID is safer than using /dev/sdb1 because device letters can change when you reboot.

Deluge Drive Mounting

Step 2: Mount the Drive to the Proxmox Host

Create a permanent directory on the host and link the drive to it.

Create the mount directory:

- mkdir -p /mnt/deluge
- chmod -R 777 /mnt/delug

Open the Promox host's filesystem table to make it mount automatically on every reboot:

- vi /etc/fstab

Add this line to the very bottom of the file (replace YOUR-UUID-HERE with your actual UUID, and change ext4 if your drive uses

/dev/sda1 /mnt/deluge ext4 defaults,nofail,noatime,commit=600 0 2

Save and exit (Ctrl+O, Enter, Ctrl+X), then mount it immediately:

- mount -a

Step 3: Link the Storage to Deluge (LXC 102)

Now, use the Proxmox Container Toolkit (pct) to map that host directory directly into your Deluge container as a Mount Point.

Run this command in the Proxmox host shell:

- pct set 102 -mp0 /mnt/deluge,mp=/deluge

What this does:

/mnt/deluge is where the files sit physically on your Proxmox server.

/deluge is the virtual path that will instantly appear inside the Deluge container.

Step 4: Fix Permissions (Crucial)

Because unprivileged LXC containers map user IDs differently than the host, Deluge might get a "Permission Denied" error when trying to write downloads to the external drive. To fix this, grant read/write access to the mount folder from the host shell:

- chmod -R 777 /deluge

This is the directory of the newly mounted deluge drive as viewed on the 102 console

root@deluge:~# #Note the prompt - This says you are running as root on the host deluge

- ls -al /deluge

  total 40

  drwxrwxrwx 7 root root 4096 Jun 10 10:21 .

  drwxr-xr-x 21 root root 4096 Jun 17 10:31 ..

  drwxrwxrwx 8 root root 4096 Jun 17 10:31 config

drwxrwxrwx 2 root root 4096 Jun 10 10:22 download-dir

drwxrwxrwx 2 root root 4096 Mar 16 2024 failed

drwxrwxrwx 2 root root 16384 Oct 5 2023 lost+found

drwxrwxrwx 19 root root 4096 Mar 16 2024 torrent-dir

Once this is done,

create a link in ~/.config by typing the following:

- ln -s deluge /deluge/config

Then open your Deluge Web UI (192.168.86.252:8112), go to Preferences -> Downloads, and change your settings to:

Downloads:

Download to:

/deluge/download-dir

Move completed to:

/deluge/torrent-dir

Copy of .torrent files to:

/deluge/torrent-dir

Network:

Incomming Interface

eth0

Incoming Port

6881

Outgoing Interface

eth0

3. For Ollama LLM Engine (LXC 103)

Run this right after the Ollama LXC script finishes to attach the isolated segment and map your 13th Gen Intel iGPU:

>>> BASH Script start

```
# 1. Attach the isolated network backend interface (eth1)
pct set 103 --net1 name=eth1,bridge=vmbr1,ip=10.0.10.3/24
# 2. Allocate the 6 high-performance vCPUs and 16GB RAM limits
pct set 103 --cores 6 --memory 16384
# 3. Inject the Intel iGPU Passthrough rules into the LXC hardware config
cat <<EOF >> /etc/pve/lxc/103.conf
lxc.cgroup2.devices.allow: c 226:0 rwm
lxc.cgroup2.devices.allow: c 226:128 rwm
lxc.mount.entry: /dev/dri dev/dri none bind,optional,create=dir
lxc.mount.entry: /dev/dri/renderD128 dev/dri/renderD128 none
bind,optional,create=file
EOF
pct restart 103
```

>>> BASH Script end

4. For Wyoming Docker (LXC 104)

Run this to map the isolated network switch so it can talk privately to the Ollama API:

>>> BASH Script start

```
# Attach the isolated network backend interface (eth1)
pct set 104 --net1 name=eth1,bridge=vmbr1,ip=10.0.10.4/24
# Enforce your 2 Cores and 2GB RAM blueprint
pct set 104 --cores 2 --memory 2048
pct restart 104
```

>>> BASH Script end

NOTES

Change my subscription keys at the Promox level

- echo "Enabled: no" >> /etc/apt/sources.list.d/pve-enterprise.sources

- echo "Enabled: no" >> /etc/apt/sources.list.d/ceph.sources
- cat <<EOF > /etc/apt/sources.list.d/pve-enterprise.sources

  Types: deb

  URIs: https://enterprise.proxmox.com/debian/pve

  Suites: trixie

  Components: pve-enterprise

  Signed-By: /usr/share/keyrings/proxmox-archive-keyring.gpg

  Enabled: no

  EOF
- cat <<EOF > /etc/apt/sources.list.d/ceph.sources

  Types: deb

  URIs: https://enterprise.proxmox.com/debian/ceph-squid

  Suites: trixie

  Components: main

  Signed-By: /usr/share/keyrings/proxmox-archive-keyring.gpg

  Enabled: no

  EOF

- vi /etc/apt/sources.list.d/pve-no-subscription.sources

Enter an i in the window "I"hen paste in the following lines

Types: deb

URIs: http://download.proxmox.com/debian/pve

Suites: trixie

Components: pve-no-subscription

Signed-By: /usr/share/keyrings/proxmox-archive-keyring.gpg

Then entera ":!wq" wo save and exit the vi editor

Exectut the following:

- apt dist-upgrade -y

Disaster Recovery:

Text Configuration Backup Script

To secure your precise resource layouts, network definitions, and maps, keep this custom backup tool saved on your host.

Creating the Host Backup Automator

Log into your Proxmox Host Shell.

Create the script file:

- vi /root/backup_all_configs.sh

Paste the following code:

```
#!/bin/bash
# Configuration
BACKUP_DIR="/var/lib/vz/dump/config_backups"
TIMESTAMP=$(date +"%Y%m%d_%H%M%S")
ARCHIVE_NAME="${BACKUP_DIR}/ed_proxmox_configs_${TIMESTAMP}.tar.gz"
TEMP_DIR="/tmp/unified_config_backup"
# Your specific Node IDs
LXCS=(100 102 103 104)
VMS=(101)
echo "================================================="
echo "Starting Configuration Backup for Ed's KAMRUI PC..."
echo "================================================="
# Initialize clean workspace
mkdir -p "$BACKUP_DIR"
mkdir -p "$TEMP_DIR"
echo "[1/3] Copying Proxmox host system parameters..."
cp -r /etc/pve/ "$TEMP_DIR/pve_cluster_virtual_fs/"
cp /etc/network/interfaces "$TEMP_DIR/host_network_interfaces"
cp /etc/passwd /etc/group "$TEMP_DIR/"
echo "-> Host network, storage definitions, and clusters copied successfully."
echo "[2/3] Copying individual Container & VM .conf files..."
for id in "${LXCS[@]}"; do
    if [ -f "/etc/pve/lxc/${id}.conf" ]; then
        echo "  -> Copying LXC ${id}.conf..."
        cp "/etc/pve/lxc/${id}.conf" "$TEMP_DIR/"
    else
        echo "  -> WARNING: LXC ${id}.conf file was not found!"
    fi
```

```
done
for id in "${VMS[@]}"; do
    if [ -f "/etc/pve/qemu-server/${id}.conf" ]; then
        echo "  -> Copying VM ${id}.conf..."
        cp "/etc/pve/qemu-server/${id}.conf" "$TEMP_DIR/"
    else
        echo "  -> WARNING: VM ${id}.conf file was not found!"
    fi
done
echo "[3/3] Packing everything into gzip archive..."
tar -czf "$ARCHIVE_NAME" -C "$TEMP_DIR" .
rm -rf "$TEMP_DIR"

echo "=================================================="
echo "SUCCESS! All text configurations backed up cleanly."
echo "Archive saved to: $ARCHIVE_NAME"
echo "=================================================="
```

Code 24: backup_all_configs.sh

Save and exit (ESC, :wq, Enter).

Make the script executable:

- chmod +x /root/backup_all_configs.sh

Execute it at any time to output a fresh, timestamped config snapshot to your host storage directory:

/var/lib/vz/dump/config_backups/

==========

DISASTER RECOVERY: SECONDARY FAILOVER STACK (RASPBERRY PI)

==========

[Raspberry Pi 4 - DNS Failover Node]

- Static Network IP: 192.168.86.253 (Mapped as Secondary DNS in Primary Router)
- Primary Service: Native Pi-hole on Raspberry Pi OS
- Sync Engine: Gravity-Sync automated cron task (Pulls from PVE LXC 100)

[Raspberry Pi 4/5 - Cold Standby Home Assistant Node]

- Storage Target: Dedicated Bootable USB SSD
- Primary Service: Home Assistant OS (Bare-Metal)
- Operational State: Powered Down / Readied State

- Recovery Path: Swap USB Zigbee stick from PVE Host -> Power On -> Restore 3 AM Snapshot

==========

Tools to verify Installation and functionality

How to Quickly Verify Promox Container Configurations:

Sometimes you want to quickly see if there is a problem. In order to do this, you need to look at things from a specific perspective. For example: I may want to know if a specific Promox container can see all my other containers.

You can log into any Promox container via Putty ssh or the Promox web page. In the Promox container select Console and paste in the following code. Or you can open Putty and select the Container you want. In that shell paste in the following code:

```
#!/bin/bash

  declare -A nodes=(

    ["Pi-hole"]="192.168.86.249"
    ["Home Assistant"]="192.168.86.251"
    ["Deluge"]="192.168.86.252"
    ["Ollama Engine"]="192.168.86.253"
    ["Wyoming Docker"]="192.168.86.248"
    ["Surveillance NAS"]="192.168.86.104"
    ["Primary NAS"]="192.168.86.93"
    ["EdsDesktop"]="192.168.86.246"
  )

  echo "=== VERIFYING HOME LAB NETWORK INTERCONNECTS ==="
  for name in "${!nodes[@]}"; do
    ip=${nodes[$name]}
    if ping -c 1 -W 1 "$ip" > /dev/null; then
      echo -e "✅ [ONLINE] $name matches configuration at $ip"
    else
      echo -e "❌ [OFFLINE] $name failed to respond at $ip"
    fi
  done
  # ollama model verification
  curl -s http://192.168.86.253:11434/api/tags | grep -o
'"name":"[^"]*"' | sed 's/"name":"//;s/"//'

echo -e "\n=== VM 101 (HAOS) CONFIG ===" && cat /etc/pve/qemu-
server/101.conf | grep -E "cores|memory|scsi|sata|virtio" && echo -e
"\n=== LXC CONFIGURATIONS ===" && for id in 100 102 103 103 104 105; do
```

```
echo -e "\n--- Container $id ---"; cat /etc/pve/lxc/$id.conf | grep -E
"cores|memory|rootfs|mp"; done
```

Code 25: QuickSystemTest.sh

When you copy and run this simple script in a terminal of your Putty Shell, Proxmox host, or any Linux-based container it automatically verifies that every IP in your config is alive and responsive on your subnet. It will also tell you specific configuration items for each container that you can verify against this output.

After you run the above code block you will get an output like this.

```
Response of script

=== VERIFYING HOME LAB NETWORK INTERCONNECTS ===
✅ [ONLINE]      Primary NAS matches configuration at 192.168.86.93
✅ [ONLINE]      Home Assistant matches configuration at 192.168.86.251
✅ [ONLINE]      Ollama Engine matches configuration at 192.168.86.253
✅ [ONLINE]      Deluge matches configuration at 192.168.86.252
✅ [ONLINE]      Pi-hole matches configuration at 192.168.86.249
✅ [ONLINE]      Surveillance NAS matches configuration at
192.168.86.104
✅ [ONLINE]      Wyoming Docker matches configuration at 192.168.86.248
✅ [ONLINE]      EdsDesktop matches configuration at 192.168.86.246
nomic-embed-text:latest
qwen2.5-coder:1.5b
qwen2.5-coder:7b
qwen2.5:7b

=== VM 101 (HAOS) CONFIG ===
    boot: order=scsi0
    cores: 2
    memory: 5120
    net0: virtio=02:93:F8:A8:3A:88,bridge=vmbr0
    net1: virtio=BC:24:11:3F:40:50,bridge=vmbr1
    scsi0: local-lvm:vm-101-disk-0,discard=on,size=32G,ssd=1
    scsihw: virtio-scsi-pci

    === LXC CONFIGURATIONS ===

    --- Container 100 ---
    cores: 1
    memory: 256
    rootfs: local-lvm:vm-100-disk-0,size=2G

    --- Container 102 ---
    cores: 1
    memory: 2048
    mp0: /mnt/deluge,mp=/deluge
```

```
    rootfs: local-lvm:vm-102-disk-0,size=4G

    --- Container 103 ---
    cores: 6
    memory: 16384
    rootfs: local-lvm:vm-103-disk-0,size=40G

    --- Container 103 ---
    cores: 6
    memory: 16384
    rootfs: local-lvm:vm-103-disk-0,size=40G

    --- Container 104 ---
    cores: 4
    memory: 2048
    rootfs: local-lvm:vm-104-disk-0,size=10G

    --- Container 105 ---
    cores: 1
    memory: 1024
    rootfs: local-lvm:vm-105-disk-0,size=5G
```

Code 26: Quick System Test Response

The check boxes show the hosts are correctly configured and can communicate with each other.

Describing How Voice Interaction Works

The flow of your voice interaction in this setup is designed for efficient processing, leveraging your isolated backend network. Here is the logical sequence of your voice communication:

The Voice Processing Pipeline Capture (XIAO ESP32-S3):

The satellite detects the wake word and streams the raw audio over the LAN (192.168.86.x) to the Wyoming-Docker (VMID 104) container.

Transcription (Wyoming/Whisper):

Inside VMID 104, the Whisper engine converts that audio stream into text.

Intent & Logic (Home Assistant OS):

That text is sent via the Backend Network (10.0.10.x) to Home Assistant OS (VMID 101). Home Assistant analyzes the text to determine the intent (e.g., "turn on the lights").

Processing (Ollama):

If the request requires complex reasoning or a conversational response, Home Assistant sends the prompt to the Ollama LLM Engine (LXC 103) over the Backend Network (10.0.10.x).

Response Generation (Ollama → HAOS):

Ollama processes the request and sends the resulting text response back to Home Assistant over the Backend Network.

Text-to-Speech (Wyoming/Piper):

Home Assistant sends the text to Wyoming-Docker (VMID 104) via the Backend Network. Piper converts the text into an audio file.

Playback (XIAO ESP32-S3):

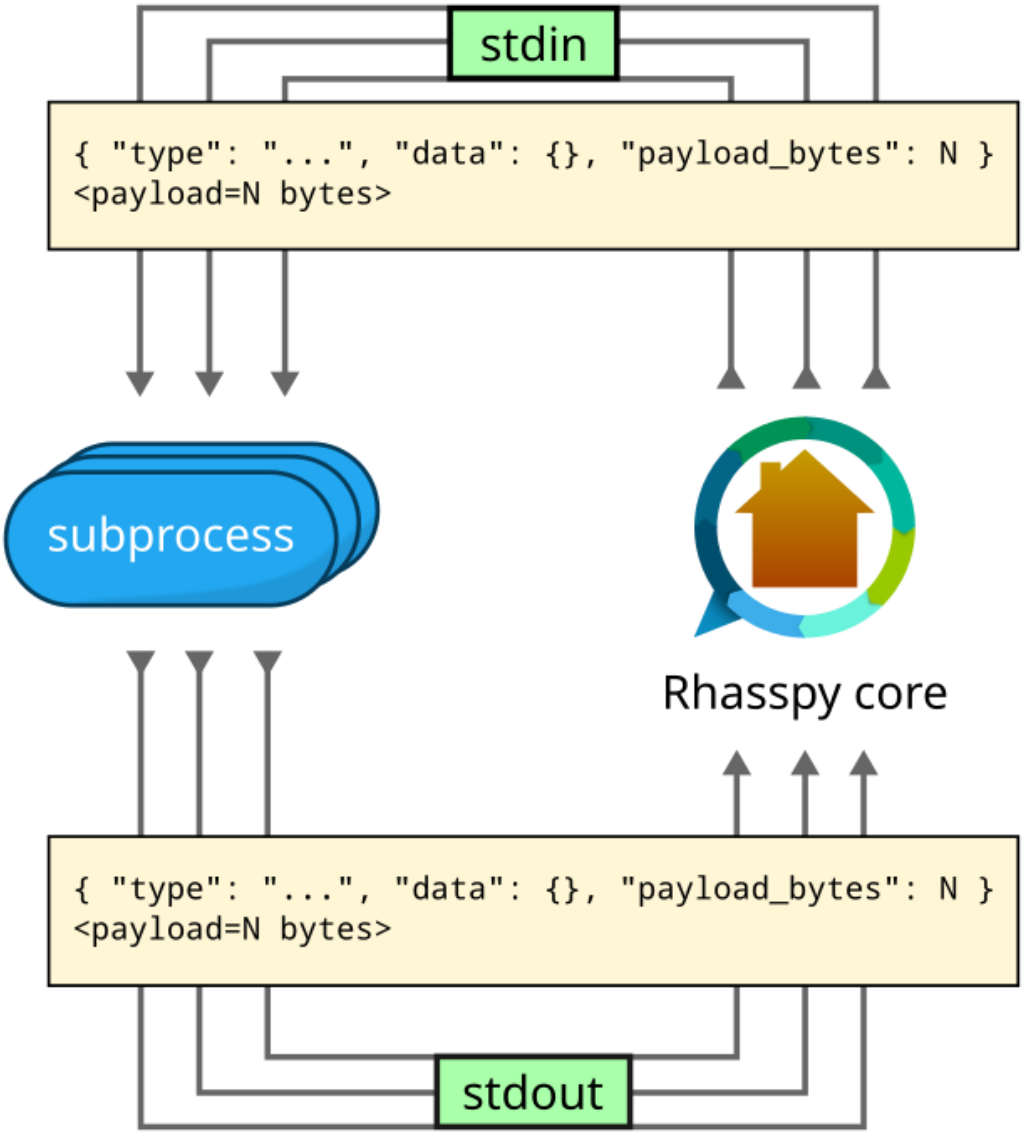
The final audio stream is sent back to the satellite over the LAN for playback.

Summary of Communication Paths

- Stage
- Data Type Network Path
- Satellite ↔ Wyoming
- Raw Audio / Audio Stream
- LAN (192.168.86.x)
- Wyoming ↔ HAOSText / Commands
- Backend Network (10.0.10.x)HAOS ↔ Ollama
- Text Prompts / LLM Response
- Backend Network (10.0.10.x)

Why Wyoming Protocol

- An interprocess event protocol over stdin/stdout for Rhasspy v3.
- (effectively JSONL with an optional binary payload)



Motivation

Rhasspy v2 was built on top of MQTT, and therefore required (1) an MQTT broker and (2) all services to talk over MQTT. Each open source voice program needed a custom service wrapper to talk to Rhasspy.

For v3, a project goal was to minimize the barrier for programs to talk to Rhasspy.

Talking Directly to Programs

Many voice programs have similar command line interfaces. For example, most text to speech programs accept text through standard input and write a WAV file to standard output or a file:

echo “Input text” | text-to-speech > output.wav

A protocol based on standard input/output would be universal across languages, operating systems, etc. However, some voice programs need to consume or produce audio/event streams. For example, a speech to text system may return a result much quicker if it can process audio as it's being recorded.

Event Streams

Standard input/output are byte streams, but they can be easily adapted to event streams that can also carry binary data. This lets us send, for example, chunks of audio to a speech to text program as well as an event to say the stream is finished. All without a broker or a socket!

Each **event** in the Wyoming protocol is:

1. A **single line** of JSON with an object:
    - **MUST** have a type field with an event type name
    - MAY have a data field with an object that contains event-specific data
    - MAY have a payload_length field with a number > 0
2. If payload_length is given, *exactly* that may bytes follows

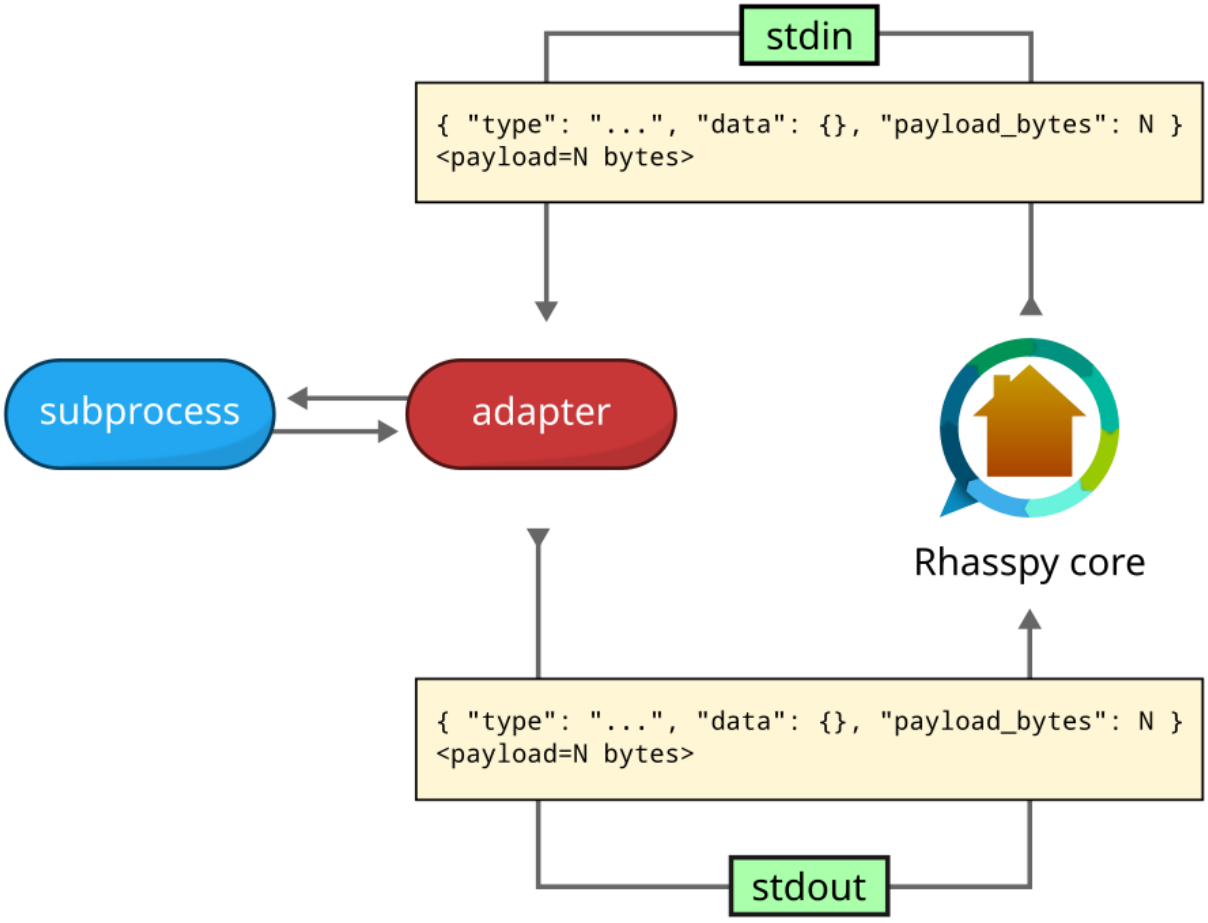
Example:

{ "type": "audio-chunk", "data": { "rate": 16000, "width", "channels": 1 }, "payload_length": 2048 }

<2048 bytes>

Adapter

Using events over standard input/output unfortunately means we cannot talk to most programs directly. Fortunately, small adapters can be written and shared for programs with similar command-line interfaces. The adapter speaks events to Rhasspy, but calls the underlying program according to a common convention like “text in, WAV out”.



Events Types

Voice programs vary significantly in their options, but programs within the same domain have the same minimal requirements to function:

- mic
  - Audio input
  - Outputs fixed-sized chunks of PCM audio from a microphone, socket, etc.
  - Audio chunks may contain timestamps
- wake
  - Wake word detection
  - Inputs fixed-sized chunks of PCM audio
  - Outputs name of detected model, timestamp of audio chunk
- asr

    - Speech to text
    - Inputs fixed-sized chunks of PCM audio
    - Inputs an event indicating the end of the audio stream (or voice command)
    - Outputs a transcription
- vad
    - Voice activity detection
    - Inputs fixed-sized chunks of PCM audio
    - Outputs events indicating the beginning and end of a voice command
- intent
    - Intent recognition
    - Inputs text
    - Outputs an intent with a name and entities (slots)
- handle
    - Intent/text handling
    - Does something with an intent or directly with a transcription
    - Outputs a text response
- tts
    - Text to speech
    - Inputs text
    - Outputs one or more fixed-sized chunks of PCM audio
- snd
    - Audio output
    - Inputs fixed-sized chunks of PCM audio
    - Plays audio through a sound system

The following event types are currently defined:

The Hardware - What it takes

Specification.

KAMRUI Hyper H2 Mini PC Windows 11 Pro, Intel Core 14450HX Mini-Computer (10C/16T, 4.8GHz), 16GB RAM 512GB PCIe4.0 SSD, Triple 4K(USB-C/HDMI/DP), WiFi 6 BT5.2 Micro Desktop PC for Business Office

Type	Product
Brand:	KAMRUI
Operating System:	Windows 11 Pro
CPU Model:	Core i5
CPU Speed:	4.8 GHz
Cache Size:	20 MB
Graphics Card Description:	Integrated
Graphics Coprocessor	UHD Graphics
Memory Storage Capacity:	16 GB

Specific Uses For Product Multimedia, Server, Personal Computer, Business Computer, Office PC, HTPC, Online Learning, Work from Home, Education, Photo Editing, Web browsing

Personal computer design type

14TH GEN INTEL i5-14450HX HX-SERIES DESKTOP-GRADE PROCESSOR

Ditch bulky desktops without sacrificing performance. Intel's HX-series chip outperforms low-power U/H/P-series processors by a wide margin, bringing desktop-grade speed to this compact mini-PC. Powered by 10C/16T Intel Core i5-14450HX (6P+4E architecture, 4.8GHz max turbo, 55W TDP), it effortlessly handles heavy multitasking, content creation, 3D rendering, code compiling and professional development, ideal for power users and business teams.

16GB DUAL-CHANNEL DDR4 + 512GB PCIE 4.0 SSD, MAX 64GB+4TB EXPANDABILITY

Eliminate lag and storage shortages permanently. The Hyper H2 features 16GB 3200MHz dual-channel DDR4 memory (upgradeable to 64GB via dual slots) and a 512GB high-speed PCIe 4.0 NVMe SSD, delivering 40% faster boot and app loading than PCIe 3.0 solutions. Dual M.2 slots support up to 4TB total storage, easily meeting demands for AI models, homelab servers, virtual machines and enterprise data management. Balancing speed, capacity, and flexibility, it supports zero-footprint PC

strategies for modern businesses while consuming far less power than traditional desktop towers—aligning with eco-friendly office goals.

WINDOWS 11 PRO, FULL LINUX COMPATIBILITY & TRIPLE 4K@60HZ OUTPUT

Maximize efficiency with seamless system support and multi-screen productivity. Pre-installed with Windows 11 Pro (Copilot-ready), it optimizes AI-powered office automation and data analysis for business use. It fully supports Ubuntu, Linux and hypervisor environments, perfect for developers and homelab enthusiasts. Driven by 14th Gen Intel UHD Graphics, it drives 3 simultaneous 4K@60Hz displays, boosting multitasking efficiency by 2-3x for financial monitoring, design and data work.

FULL CONNECTIVITY, ULTRA COMPACT CHASSIS & VESA MOUNT SUPPORT

No extra adapters needed, all ports you need in one palm-sized device. Measuring **only 5.04 x 5.04 x 1.63 inches, the Hyper H2 saves 90% of desk space, and can be VESA-mounted behind monitors for a clutter-free workspace. It includes full-function USB-C, 6x USB ports, Gigabit Ethernet, HDMI 2.0, DP 1.4 and an audio jack. With Wi-Fi 6 (3x faster than Wi-Fi 5) and Bluetooth 5.2, it ensures stable, lag-free connections for streaming and conferencing.**

PREMIUM HX-CLASS COOLING & 24/7 STABLE & Enterprise-Grade Reliability

Built for non-stop long-term performance. The Hyper H2 features an advanced cooling **system with dual silent centrifugal fans, dual copper heat pipes, dual fin modules and optimized dual airflow design. It keeps the processor cool under 72h+ high-load workloads, maintaining ≥95% multi-core performance for 24/7 operation, ideal for Proxmox nodes, homelab servers and enterprise deployment. FCC/CE/RoHS certified, it cuts energy use by 75% vs traditional desktops, backed by a 1-year warranty and 24/7 Amazon technical support.**

Brand Assurance & B2B Compatibility

KAMRUI is a leading innovator and manufacturer of high-performance mini PCs, delivering reliable computing solutions for professionals, developers, and businesses worldwide. The Hyper H2 Mini PC is tailored for B2B bulk procurement and corporate IT upgrades, combining eco-friendliness and practicality. With Amazon's responsive customer support and 24/7 technical assistance, you can purchase with confidence—knowing any issues will be resolved promptly, ensuring a seamless experience for your team or customers.

Operating System Windows 11 Pro

Specific Uses For Product Multimedia, Server, Personal Computer, Business Computer, Office PC, HTPC, Online Learning, Work For Home, Education, Photo Editing, Web browsing

Video Output Interfaces:

HDMI 2.0+USB3.2 Gen2 Type-C+USB3.2 Gen2 Type-A Port×2+USB3.2 Gen1 Type-A Port×4+DP1.4+RJ45 Gigabit Ethernet Port +DC IN ×1+3.5mm Audio Jack x1+Power Button x1

Connectivity Technology:

1 full-function USB-C port (10Gbps + DP + PD), Ethernet, LAN, USB, USB 3.2 Gen1 ports (5Gbps)×4, USB 3.2 Gen2 ports (10Gbps)×2, Wi-Fi, Wi-Fi 6, Bluetooth 5.2

Personal Mini Computer Design - Need to update

ADVANCED TECHNICAL SPECIFICATIONS (i5-14450HX 16GB+512GB)	
Type	Mini PC
Color	Sliver
Hard Disk Description	SSD
Hardware Interface	DisplayPort, HDMI, USB 3.2 Gen 2, USB 3.2 Gen 2×2, USB Type C
Power Consumption	55 watts
Hard Disk Interface	PCIE x 4
Style Name	14450HX /16+512GB
Cooling Method	Air
Compatible Devices	Headphones, Keyboard, Monitor, Mouse, Printer
Video Output	HDMI 2.0, DisplayPort 1.4b, USB-C
Memory	
Cache Memory Installed Size	20 MB
Memory Storage Capacity	16 GB
RAM Memory Installed	16 GB
RAM Memory Technology	DDR4
Ram Memory Maximum Size	64 GB
Memory Speed	3200 mt_s
RAM Type	DDR4 SDRAM
Memory Clock Speed	4.8 GHz
Pre-installed RAM	16 GB DDR4-3200MHz Dual-Channel
Max Supported RAM	64GB
Display	

Display Resolution Maximum	**3840x2160 pixels_per_inch**
Aspect Ratio	**16:9**
Resolution	**3840x1080**
Native Resolution	**3840x1080**
Connectivity	
Wireless Compability	**802.11ax, Bluetooth**
Wireless Technology	**Bluetooth, Wi-Fi**
Graphics	
Graphics Description	**Integrated**
Integrated Graphics	**Intel UHD Graphics 730**
Graphics Card Ram	**16 GB**
Graphics Ram Type	**DDR4 SDRAM**
Graphics Card Interface	**Integrated**
Processor	
Processor Series	**Core i5**
Processor	**Intel Core i5-14450HX, 10 Cores (6P+4E), 16 Threads**
Max CPU Turbo Frequency	**4.8GHz**
CPU TDP	**55W**
CPU Model Speed Maximum	**4.8 GHz**
Ports	
Total Usb Ports	**6**
Total Number of HDMI Ports	**1**
Number of Component Outputs	**3**
Input Devices	
Human-Interface Input	**Buttons, Keyboard, Mouse**