GUIDE

Generate a practical guide detailing three new homelab projects suitab

> **What you'll have running:** A working self-hosted setup **Estimated time:** ~2 hours **Difficulty:** INTERMEDIATE **Power usage:** Varies
>
> **Hardware needed:** A dedicated machine (e.g., an old desktop, a mini PC like an Intel NUC or Beelink, or a Raspberry Pi 4/5 with sufficient RAM and storage) running a Linux distribution.

Why This Setup is Worth It

Ever felt like you're just renting your digital life? Cloud services are convenient, sure, but they come with trade-offs: privacy concerns, vendor lock-in, and a recurring bill that slowly creeps up. This guide is your ticket to reclaiming control. We're not just installing software; we're building foundational self-hosted services that enhance your privacy, streamline your digital life, and give you a tangible sense of ownership.

Over a weekend, we'll tackle three core projects:

1. **Vaultwarden (Self-hosted Password Manager):** Say goodbye to trusting third parties with your most sensitive credentials. Vaultwarden, a lightweight Bitwarden server implementation, gives you a secure, private, and fully controlled password manager.
2. **Pi-hole (Network-wide Ad Blocker):** Tired of ads and trackers cluttering your internet experience? Pi-hole acts as a DNS sinkhole, blocking unwanted content for every device on your network, improving performance and privacy.
3. **Jellyfin (Open-source Media Server):** Ditch commercial streaming platforms for your personal media collection. Jellyfin lets you organize, stream, and share your movies, TV shows, and music across all your devices, anywhere.

These projects aren't just about saving a few bucks; they're about learning, empowering yourself, and building a more resilient, private digital infrastructure right in your home.

Prerequisites and Hardware

Before we dive in, let's ensure you have the right foundation. This guide assumes you have a basic understanding of Linux command-line operations and networking concepts.

Hardware Recommendations

You don't need a server rack for these projects. A modest machine will do just fine:

- **Processor:** Any modern multi-core CPU (Intel i3/i5 equivalent or better, or a Raspberry Pi 4/5).
- **RAM:** 4GB minimum, 8GB+ recommended, especially if you plan to expand with more services or use Jellyfin's transcoding features.
- **Storage:** 64GB SSD minimum for the OS and Docker containers. For Jellyfin, you'll need additional storage for your media files (e.g., a large HDD or a NAS share mounted to the server).
- **Network:** A reliable gigabit Ethernet connection is highly recommended for your server. Wi-Fi can work but might introduce latency or instability.

> 💡 Tip: An old laptop makes an excellent homelab server. It has a built-in UPS (battery), keyboard, and screen for initial setup, and is often very power-efficient.

Software Prerequisites

We'll be leveraging Docker and Docker Compose for easy deployment and management.

1. **Operating System:** A fresh installation of a modern Linux distribution. Ubuntu Server LTS (e.g., 22.04 or 24.04) or Debian Stable are excellent choices.
2. **SSH Access:** Ensure you can SSH into your server. This makes remote management much easier.
3. **Docker Engine:** The core containerization platform.

4. **Docker Compose:** A tool for defining and running multi-container Docker applications.

Let's get Docker and Docker Compose installed.

```
# Update your package lists
sudo apt update && sudo apt upgrade -y

# Install necessary packages for Docker
sudo apt install -y ca-certificates curl gnupg lsb-release

# Add Docker's official GPG key
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -
o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Add the Docker repository to Apt sources
echo \
  "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/
docker.gpg] https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Update Apt package index again
sudo apt update

# Install Docker Engine, containerd, and Docker Compose
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin

# Add your user to the docker group to run Docker commands without sudo
sudo usermod -aG docker $USER

# Log out and log back in (or reboot) for group changes to take effect
# Then verify Docker installation
```

Verification: After logging back in, run:

```
docker run hello-world
```

Expected Output:``` Unable to find image 'hello-world:latest' locally latest: Pulling from library/hello-world ... Hello from Docker! This message shows that your installation appears to be working correctly. ...

```
If you see the "Hello from Docker!" message, you're good to go!

## Installation

We'll create a dedicated directory for our homelab projects and then set up
each service using Docker Compose.
```bash
```
```

```
mkdir ~/homelab_services
cd ~/homelab_services
```

Project 1: Self-hosted Password Manager (Vaultwarden)

Vaultwarden is a lightweight, secure, and open-source Bitwarden server implementation. It's fully compatible with official Bitwarden clients.

Vaultwarden Installation Steps

1. **Create a directory for Vaultwarden:**

```
    mkdir vaultwarden
    cd vaultwarden
```

1. **Create `docker-compose.yml`:**

```
    # ~/homelab_services/vaultwarden/docker-compose.yml
    version: '3.8'

    services:
      vaultwarden:
        image: vaultwarden/server:1.30.5 # Pin to a specific version for
stability
        container_name: vaultwarden
        restart: unless-stopped
        ports:
          - "8000:80" # Host_Port:Container_Port - Access at http://
your_server_ip:8000
        volumes:
          - ./vw-data:/data
        environment:
          # Replace with a strong, unique string for your admin token
          # You can generate one with `openssl rand -base64 32`
          ADMIN_TOKEN: "YOUR_VERY_STRONG_ADMIN_TOKEN_HERE"
          # Disable new user registrations after initial setup for security
          SIGNUPS_ALLOWED: "false"
          # Enable invitations if you want to invite specific users
          # INVITE_ACCEPTED_EMAIL_ADDRESSES:
"user1@example.com,user2@example.com"
          WEBSOCKET_ENABLED: "true" # Required for real-time sync with clients
          # Optionally configure SMTP for email verification, password resets
          # SMTP_HOST: "smtp.your-provider.com"
          # SMTP_PORT: "587"
          # SMTP_FROM: "vaultwarden@yourdomain.com"
          # SMTP_USERNAME: "your_smtp_username"
          # SMTP_PASSWORD: "your_smtp_password"
          # SMTP_SSL: "true"
          # SMTP_TLS: "true"
```

```
> ⚠️ Warning: Change `ADMIN_TOKEN` immediately to a strong, randomly generated
string. If you enable `SIGNUPS_ALLOWED: "true"`, remember to set it back to
```

```
`"false"` after you've created your initial user accounts to prevent
unauthorized sign-ups.
```

1. **Start Vaultwarden:**

```
    docker compose up -d
```

Vaultwarden Verification

Open your web browser and navigate to `<http://YOUR_SERVER_IP:8000>`. You should see the Vaultwarden login page.

Expected Output (browser): The Bitwarden login page, prompting for an email and master password.

Project 2: Network-wide Ad Blocker (Pi-hole)

Pi-hole acts as your network's DNS server, blocking ad and tracker domains before they even reach your devices.

Pi-hole Installation Steps

1. **Go back to the homelab services directory and create a new one for Pi-hole:**

```
    cd ~/homelab_services
    mkdir pihole
    cd pihole
```

1. **Create `docker-compose.yml`:**

```
    # ~/homelab_services/pihole/docker-compose.yml
    version: '3'

    services:
      pihole:
        image: pihole/pihole:2024.05.0 # Pin to a specific version
        container_name: pihole
        hostname: pihole # Set a hostname for the container
        ports:
          - "53:53/tcp"
          - "53:53/udp"
          - "67:67/udp" # Only required if you intend to use Pi-hole's DHCP
server
          - "80:80/tcp" # For the web interface
        environment:
          TZ: "America/New_York" # Set your timezone
          WEBPASSWORD: "YOUR_PIHOLE_WEB_PASSWORD" # Set a strong password for
the web interface
          # DNSMASQ_LISTENING: "all" # Listen on all interfaces
```

```
          # PIHOLE_DNS_: "1.1.1.1;1.0.0.1" # Upstream DNS servers (Cloudflare)
          # ServerIP: "YOUR_SERVER_IP" # Optional, if you have issues with
multiple network interfaces
        volumes:
          - ./etc-pihole/:/etc/pihole/
          - ./etc-dnsmasq.d/:/etc/dnsmasq.d/
        cap_add:
          - NET_ADMIN # Required for Pi-hole to function properly
        restart: unless-stopped
```

```
> ⚠️ Warning: Change `WEBPASSWORD` to a strong, unique password. Replace `TZ`
with your actual timezone (e.g., `Europe/London`, `Asia/Tokyo`).
```

1. **Start Pi-hole:**

```
docker compose up -d
```

Pi-hole Verification

Open your web browser and navigate to `<http://YOUR_SERVER_IP/admin>`. You should see the Pi-hole admin login page. Log in with the password you set.

Expected Output (browser): The Pi-hole admin dashboard, showing statistics and configuration options.

Project 3: Open-source Media Server (Jellyfin)

Jellyfin is a free software media system that puts you in control of managing and streaming your media.

Jellyfin Installation Steps

1. **Go back to the homelab services directory and create a new one for Jellyfin:**

```
cd ~/homelab_services
mkdir jellyfin
cd jellyfin
```

1. **Create `docker-compose.yml`:**

```
# ~/homelab_services/jellyfin/docker-compose.yml
version: '3.8'

services:
  jellyfin:
    image: jellyfin/jellyfin:10.8.13 # Pin to a specific version
```

```
        container_name: jellyfin
        restart: unless-stopped
        network_mode: "host" # Recommended for Jellyfin to auto-discover
devices and for transcoding
        # If network_mode: "host" causes issues, use specific ports:
        # ports:
        #   - "8096:8096/tcp" # HTTP traffic
        #   - "8920:8920/tcp" # HTTPS traffic
        #   - "1900:1900/udp" # DLNA
        #   - "7359:7359/udp" # GDM (client discovery)
        volumes:
          - ./config:/config # Configuration files
          - ./cache:/cache   # Transcoding cache
          - /path/to/your/movies:/data/movies:ro # Mount your actual media
paths
          - /path/to/your/tvshows:/data/tvshows:ro # :ro means read-only
          # Add more media folders as needed
        environment:
          # Optional: Specify the user ID and group ID for the container
          # This helps with file permissions if your media is on a network
share
          # PUID: "1000" # Your user's UID (get with `id -u $USER`)
          # PGID: "1000" # Your user's GID (get with `id -g $USER`)
          JELLYFIN_PublishedServerUrl: "http://YOUR_SERVER_IP:8096" # Replace
with your server IP
        # For hardware transcoding (NVIDIA, Intel QuickSync, AMD AMF/VCE)
        # Check Jellyfin documentation for specific device mappings
        # For Intel QuickSync (e.g., on NUCs):
        # devices:
        #   - /dev/dri:/dev/dri
        # For NVIDIA GPUs:
        # runtime: nvidia
        # environment:
        #   NVIDIA_VISIBLE_DEVICES: all
        #   NVIDIA_DRIVER_CAPABILITIES: all
```

```
> 💡 Tip: The `network_mode: "host"` simplifies setup by giving the container
direct access to the host's network interfaces. If you encounter issues or
prefer more isolation, switch to explicit `ports:` mapping.
> ⚠️ Warning: Replace `/path/to/your/movies` and `/path/to/your/tvshows` with
the *actual paths* to your media libraries on your host system. The `:ro`
ensures the container only has read access, preventing accidental deletion.
```

1. **Start Jellyfin:**

```
    docker compose up -d
```

Jellyfin Verification

Open your web browser and navigate to `<http://YOUR_SERVER_IP:8096>`. You should see the Jellyfin setup wizard.

Expected Output (browser): The Jellyfin welcome screen, prompting you to set up your server.

Initial Configuration

Now that our services are running, let's get them configured.

Vaultwarden Initial Setup

1. **Access Web Interface:** Go to `<http://YOUR_SERVER_IP:8000>`.
2. **Create Your First User:** Register a new account with your email and a strong master password. This will be your primary Bitwarden account.
3. **Disable Sign-ups (if not already):** If you initially set `SIGNUPS_ALLOWED: "true"` in your `docker-compose.yml` to create your account, edit the `docker-compose.yml` file to change it to `SIGNUPS_ALLOWED: "false"`.

```
    # ...
        environment:
          ADMIN_TOKEN: "YOUR_VERY_STRONG_ADMIN_TOKEN_HERE"
          SIGNUPS_ALLOWED: "false" # <-- Ensure this is false after initial
user creation
          WEBSOCKET_ENABLED: "true"
    # ...
```

```
Then restart the container:
```

```
    cd ~/homelab_services/vaultwarden
    docker compose down && docker compose up -d
```

1. **Connect Clients:** Download the official Bitwarden client for your browser, desktop, or mobile device. During setup, select "Self-Hosted" and enter your server URL: `<http://YOUR_SERVER_IP:8000>`.

Pi-hole Initial Setup

1. **Access Admin Panel:** Go to `<http://YOUR_SERVER_IP/admin>` and log in with your `WEBPASSWORD`.

2. **Configure Upstream DNS (Optional but recommended):**
    - Navigate to "Settings" > "DNS".
    - You can choose your preferred upstream DNS providers (e.g., Cloudflare, Google, Quad9).
3. **Configure Your Network to Use Pi-hole:** This is the most crucial step.
    - **Option A (Recommended for most home users): Change DNS on your router.**
        - Log in to your home router's administration interface (usually `<http://192.168.1.1>` or `<http://192.168.0.1>`).
        - Look for "WAN Settings," "Internet Settings," "DHCP/DNS," or "Network Settings."
        - Find the DNS server fields and change the Primary DNS to `YOUR_SERVER_IP`.
        - Set the Secondary DNS to something reliable like `1.1.1.1` (Cloudflare) or `8.8.8.8` (Google) as a fallback, or leave it blank if you want all DNS traffic to go through Pi-hole.
        - Save changes and reboot your router.
    - **Option B (Advanced): Enable Pi-hole's DHCP server.**
        - In the Pi-hole admin panel, go to "Settings" > "DHCP".
        - Enable the DHCP server. This will make Pi-hole assign IP addresses and tell devices to use itself for DNS. Make sure your router's DHCP server is disabled if you use this option to avoid conflicts.
    - **Option C (Per-device): Manually configure DNS on individual devices.** This is useful for testing but not practical for a whole network.

Verification: After configuring your network, visit a website known for ads (e.g., a news site). You should see significantly fewer ads. In the Pi-hole dashboard, you'll start seeing queries being processed and ads blocked.

Jellyfin Initial Setup

1. **Access Web Interface:** Go to `<http://YOUR_SERVER_IP:8096>`.

2. **Follow the Setup Wizard:**
    - **Language:** Select your preferred language.
    - **Create Admin User:** Create a username and strong password for your Jellyfin administrator account.
    - **Add Media Libraries:**
        - Click "Add Media Library."
        - Select content type (e.g., "Movies").
        - Click the "+" next to "Folders" and add the paths you mounted in your `docker-compose.yml` (e.g., `/data/movies`, `/data/tvshows`).
        - Choose your preferred language and country for metadata.
        - Click "OK."
    - **Metadata Language & Country:** Set your preferred language and country for movie and TV show information.
    - **Remote Access:** Decide whether to allow remote connections. For now, you can keep it local.
    - **Finish:** Complete the wizard.
3. **Scan Libraries:** Jellyfin will automatically start scanning your media. You can monitor progress from the dashboard.

Verification: After the initial scan, you should see your movies and TV shows populated with posters, descriptions, and other metadata on the Jellyfin dashboard. Try playing a video from a web browser or a Jellyfin client app.

Hardening and Security

Self-hosting comes with the responsibility of security. Here are concrete steps to harden your setup.

1. Firewall Configuration (UFW)

Ensure only necessary ports are open to the outside world. By default, your server should only allow SSH (port 22) from your local network.

```
# Install UFW if not already installed
sudo apt install ufw -y

# Deny all incoming traffic by default
sudo ufw default deny incoming
```

```
sudo ufw default allow outgoing

# Allow SSH from your local network (e.g., 192.168.1.0/24)
sudo ufw allow from 192.168.1.0/24 to any port 22 comment 'Allow SSH from
local network'

# Allow ports for our services (adjust if you used different host ports)
# Vaultwarden (if exposed directly, generally not recommended without reverse
proxy)
# sudo ufw allow from 192.168.1.0/24 to any port 8000 comment 'Allow
Vaultwarden from local network'
# Pi-hole (DNS and Web UI)
sudo ufw allow from 192.168.1.0/24 to any port 53 comment 'Allow Pi-hole DNS
from local network'
sudo ufw allow from 192.168.1.0/24 to any port 80 comment 'Allow Pi-hole Web
UI from local network'
# Jellyfin (if using specific ports, otherwise host network bypasses UFW)
# sudo ufw allow from 192.168.1.0/24 to any port 8096 comment 'Allow Jellyfin
Web UI from local network'

# Enable UFW
sudo ufw enable

# Check UFW status
sudo ufw status verbose
```

Expected Output:``` Status: active Logging: on (low) Default: deny (incoming), allow (outgoing), disabled (routed) New profiles: skip

To Action From

22/tcp ALLOW IN 192.168.1.0/24 53 ALLOW IN 192.168.1.0/24 80/tcp ALLOW IN 192.168.1.0/24

```
> ⚠️ Warning: Be very careful with `ufw enable`. If you haven't allowed SSH,
you might lock yourself out of your server. Always test SSH access from a new
terminal *before* closing your current session.

### 2. Strong Passwords and Credential Rotation

- **All Admin Interfaces:** Use unique, strong passwords for Vaultwarden, Pi-
hole, Jellyfin, and your server's root/user accounts.
- **Vaultwarden `ADMIN_TOKEN`:** Change this token periodically (e.g., every
3-6 months). Regenerate a new token, update `docker-compose.yml`, and restart
the container.
- **SSH Keys:** Disable password-based SSH login and use SSH keys for
authentication.

```bash
    # On your local machine, generate an SSH key if you don't have one
    ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

    # Copy your public key to the server
    ssh-copy-id username@your_server_ip

    # On the server, edit SSH config to disable password authentication
    sudo nano /etc/ssh/sshd_config
```
```

```
    # Find and change/add:
    # PasswordAuthentication no
    # ChallengeResponseAuthentication no
    # UsePAM no # Optional, but good for security if you don't need PAM
    sudo systemctl restart sshd
```

3. Disable Unused Services

Review what's running on your server. If you don't need a desktop environment, web server (other than what's in Docker), or other services, disable them.

```
# List all services
sudo systemctl list-unit-files --type=service --state=enabled
# Disable a service (e.g., apache2 if you installed it by mistake)
sudo systemctl disable apache2
sudo systemctl stop apache2
```

4. Reverse Proxy with SSL (Advanced, but highly recommended for external access)

If you plan to access your services from outside your home network, a reverse proxy (like Nginx Proxy Manager, Caddy, or Traefik) with SSL (Let's Encrypt) is essential. This allows you to expose services on standard ports (80/443) and secure them with HTTPS, rather than exposing individual, often insecure, ports.

> 💡 Tip: Nginx Proxy Manager is a great Docker-based option for beginners, providing a web UI to manage reverse proxies and Let's Encrypt certificates. This is a project for after you've got these three running smoothly.

5. Keep Software Updated

Regularly update your host OS and Docker containers.

Maintenance

Regular maintenance is key to a stable and secure homelab.

1. Updating the Host OS

Update your Linux distribution regularly to get security patches and bug fixes.

```
sudo apt update && sudo apt upgrade -y
sudo apt autoremove -y # Remove unneeded packages
```

2. Updating Docker Containers

We've pinned our container images to specific versions (e.g., `vaultwarden/server:1.30.5`). To update to a newer version:

1. **Check for new versions:** Visit the official Docker Hub pages for `vaultwarden/server`, `pihole/pihole`, and `jellyfin/jellyfin` to find the latest stable tags.
2. **Edit `docker-compose.yml`:** Change the image tag to the new version.

```
    # Example for Vaultwarden
    # image: vaultwarden/server:1.30.5 # Old version
    image: vaultwarden/server:1.31.0 # New version (example)
```

1. **Pull and Recreate:**

```
    cd ~/homelab_services/vaultwarden # Or pihole, or jellyfin
    docker compose pull # Pulls the new image version
    docker compose up -d # Stops the old container, recreates with new image,
starts it
```

```
> 💡 Tip: Consider using a tool like [Watchtower](https://containrrr.dev/
watchtower/) to automate Docker container updates, but be cautious with
critical services like Vaultwarden — sometimes manual review of changes is
better.
```

3. Backups

Your data is paramount. Implement a backup strategy for your container volumes.

- **Vaultwarden:** The `./vw-data` volume contains all your password data.
- **Pi-hole:** The `./etc-pihole` and `./etc-dnsmasq.d` volumes contain your configurations, blocklists, and query logs.
- **Jellyfin:** The `./config` volume contains your server configuration and metadata. Your media files are separate and should be backed up independently.

Manual Backup Example:

```
# Stop the container to ensure data consistency
cd ~/homelab_services/vaultwarden
docker compose stop

# Create a timestamped backup of the data volume
```

```
tar -czvf ~/backups/vaultwarden_data_$(date +%Y%m%d%H%M%S).tar.gz ./vw-data

# Start the container again
docker compose start

# Repeat for Pi-hole and Jellyfin
# For Pi-hole:
cd ~/homelab_services/pihole
docker compose stop
tar -czvf ~/backups/pihole_config_$(date +%Y%m%d%H%M%S).tar.gz ./etc-pihole ./
etc-dnsmasq.d
docker compose start

# For Jellyfin config (media should be backed up separately)
cd ~/homelab_services/jellyfin
docker compose stop
tar -czvf ~/backups/jellyfin_config_$(date +%Y%m%d%H%M%S).tar.gz ./config
docker compose start
```

> 💡 Tip: Store these backups on a separate machine, external drive, or cloud storage. A backup on the same machine as the original data is not a true backup against hardware failure.

Troubleshooting

Here are some common issues and how to fix them.

General Docker Issues

- **`docker compose up -d` fails with port conflict:**

```
    ERROR: for service_name  Cannot start service service_name: driver failed
programming external connectivity on endpoint service_name (...): Error
starting userland proxy: listen tcp 0.0.0.0:80: bind: address already in use
```

```
**Cause:** Another service (or another Docker container) on your host machine
is already using the port that your Docker container is trying to bind to.
**Fix:**
1.  Identify what's using the port: `sudo lsof -i -P -n | grep LISTEN | grep
:80` (replace `80` with the conflicting port).
2.  Stop the conflicting service, or change the `host_port` in your `docker-
compose.yml` (e.g., `8080:80` instead of `80:80`).
3.  Restart your Docker Compose service.
```

- **Container not starting or immediately exiting:**

```
    Error response from daemon: driver failed programming external
connectivity on endpoint ...
```

```
**Cause:** Often a misconfiguration in `docker-compose.yml`, incorrect volume
paths, or missing environment variables.
**Fix:**
1.  Check container logs: `docker compose logs service_name`. This will often
show the exact error message from the application inside the container.
2.  Run the container interactively for debugging: `docker run -it --rm --
entrypoint /bin/bash image_name` (e.g., `vaultwarden/server:latest`). This
lets you explore the container's environment.
3.  Double-check `docker-compose.yml` for typos, correct volume paths, and
required environment variables.
```

Vaultwarden Specific Issues

- **Cannot register new users: Cause:** `SIGNUPS_ALLOWED` is set to `"false"`. **Fix:** Temporarily change `SIGNUPS_ALLOWED: "true"` in `docker-compose.yml`, restart the container (`docker compose down && docker compose up -d`), register your user, then change it back to `"false"` and restart again.
- **Bitwarden clients can't connect to `<http://YOUR_SERVER_IP:8000>`: Cause:** Firewall blocking port 8000, incorrect IP address, or Vaultwarden container not running. **Fix:**
    1. Verify Vaultwarden container is running: `docker ps | grep vaultwarden`.
    2. Check server firewall: `sudo ufw status`. Ensure port 8000 is allowed from your client's IP range.
    3. Double-check the server IP address.

Pi-hole Specific Issues

- **Ads are still showing after Pi-hole setup: Cause:** Devices are not using Pi-hole for DNS, or Pi-hole isn't blocking the specific ad domains. **Fix:**
    1. **Verify DNS configuration:** On a client device, check its DNS server settings. It should point to `YOUR_SERVER_IP`. If not, recheck your router's DNS settings or Pi-hole's DHCP settings.
    2. **Flush client DNS cache:**
        - Windows: `ipconfig /flushdns`
        - macOS: `sudo dscacheutil -flushcache; sudo killall -HUP mDNSResponder`
        - Linux: `sudo systemctl restart systemd-resolved` (if using systemd-resolved)
    3. **Check Pi-hole logs:** In the Pi-hole admin panel, go to "Query Log" to see if queries are reaching Pi-hole and if domains are being blocked. You might need to add more blocklists or manually block specific domains.
- **Internet access is slow or broken after setting up Pi-hole: Cause:** Pi-hole is not running, or its upstream DNS servers are misconfigured/unreachable. **Fix:**
    1. Verify Pi-hole container is running: `docker ps | grep pihole`.
    2. Check Pi-hole's upstream DNS settings (Settings -> DNS). Try changing to reliable public DNS like Cloudflare (`1.1.1.1`, `1.0.0.1`) or Google (`8.8.8.8`, `8.8.4.4`).
    3. Temporarily revert your router's DNS settings to public DNS to confirm Pi-hole is the cause.

Jellyfin Specific Issues

- **Jellyfin can't find media files: Cause:** Incorrect volume mapping in `docker-compose.yml`, or file permissions issues on the host system. **Fix:**
    1. **Check `docker-compose.yml` paths:** Ensure `/path/to/your/movies` on the host side matches the actual location of your media.
    2. **Verify permissions:** The user ID (`PUID`) and group ID (`PGID`) that Jellyfin runs as inside the container need read access to your media files on the host. If you didn't set `PUID`/`PGID`, the container runs as root. Try setting `PUID` and `PGID` to your regular user's ID (`id -u $USER`, `id -g $USER`).

```
        # Example to fix permissions on host
        sudo chown -R your_user:your_group /path/to/your/movies
        sudo chmod -R 755 /path/to/your/movies
```

```
3.  **Rescan libraries:** In the Jellyfin web UI, go to "Dashboard" ->
"Libraries" and click the "Scan Library" icon for the affected library.
```

- **Jellyfin streaming is buffering or slow: Cause:** Network bottleneck, insufficient CPU for transcoding, or media file issues. **Fix:**
    1. **Network:** Ensure your server and client devices have a good network connection (preferably wired Ethernet).
    2. **Transcoding:** If your client device doesn't support the media's original format, Jellyfin will transcode it on the fly, which is CPU-intensive.
        - Check server CPU usage during playback (`htop`).
        - Consider enabling hardware transcoding if your server supports it (see `devices` or `runtime: nvidia` in `docker-compose.yml` comments).
        - Try optimizing your media files to formats widely supported by clients (e.g., H.264/AAC in an MP4 container) to reduce transcoding needs.
    3. **Client Settings:** Check client app settings for streaming quality. Lowering the quality can reduce bandwidth and transcoding requirements.

You've now got a robust foundation for a self-hosted digital life. These three projects are just the beginning. The skills you've gained in setting up Docker Compose, managing containers, and troubleshooting will serve you well as you explore more services and expand your homelab. Remember to keep learning, experimenting, and most importantly, have fun with it!