GUIDE

Self-Host Immich on an Old Android Pixel Phone

> **What you'll have running:** You will have a self-hosted Google Photos alternative (Immich) running on an old Android Pixel phone, accessible securely from your other devices. **Estimated time:** ~5 hours **Difficulty:** INTERMEDIATE **Power usage:** ~5W idle (~$5/year)
>
> **Hardware needed:**

- Old Android Pixel phone (Pixel 3 or newer recommended for better performance)
- USB-C charger and cable (ensure continuous power)
- High-speed microSD card (128GB+ recommended) or external USB-C SSD for photo storage
- OTG adapter (if using external USB storage and phone lacks direct USB-C host mode)

Ever looked at that old Pixel phone gathering dust in a drawer and thought, "There has to be a better use for this than a paperweight"? Well, you're in luck. We're going to transform it into a privacy-respecting, self-hosted Google Photos alternative using Immich, all powered by Termux and Docker Compose. This isn't just about saving a few bucks; it's about reclaiming ownership of your memories and learning a ton about self-hosting on unconventional hardware.

Why Use an Android Pixel for Self-Hosting Immich?

Repurposing an old Android Pixel phone as a self-hosted server for Immich is a fantastic homelab project, blending resourcefulness with practical utility. It's a prime example of the "reduce, reuse, recycle" philosophy applied to technology.

Why this matters: In an era where cloud services often come with privacy concerns and recurring costs, self-hosting gives you complete control over your data. Your photos, your server, your rules. Using an old phone keeps hardware costs at zero (if you already have one) and power consumption incredibly low.

Core concept: An Android phone, especially a Pixel, is a surprisingly capable mini-computer. It has an ARM processor, ample RAM (for its original purpose), Wi-Fi, and a built-in battery backup. With Termux, we can essentially run a Linux environment, enabling us to install tools like Docker and host applications like Immich.

Here’s why it's a smart move:

- **Ultra-Low Power Consumption:** Modern phones are designed for efficiency. An old Pixel typically sips power, consuming around 5W at idle. This translates to an annual electricity cost of roughly $5, which is significantly less than a dedicated mini-PC or NAS.
- **Cost-Effective Hardware:** If you have an old Pixel lying around, your hardware cost is effectively zero. Even buying a used Pixel 3 or 4 can be cheaper than a Raspberry Pi 4 or similar single-board computer, especially when considering storage.
- **Integrated Battery Backup:** The phone's internal battery acts as a small UPS. In case of a brief power outage, your Immich server won't immediately shut down, giving you a grace period or preventing abrupt data corruption.
- **Compact Footprint:** Phones are tiny! They can be tucked away discreetly, consuming minimal physical space compared to traditional server hardware.
- **Dedicated Device:** By dedicating an old phone to this task, you create a focused appliance. It's not running other apps, so resources are primarily allocated to Immich.

What changes when it fails or scales: While powerful for its size, an Android phone isn't a powerhouse server. Performance limitations, especially during initial photo indexing or heavy concurrent access, will be noticeable. If the phone fails, you lose your server, but with a good backup strategy, your data remains safe. Scaling means moving to more robust hardware, but for personal use, this setup is surprisingly capable.

Hardware Preparation and Considerations

Before we dive into the software, let's get your Pixel phone ready. Think of this as preparing a dedicated server.

1. **Factory Reset Your Pixel:**
    - This ensures a clean slate, removing any old apps, data, or configurations that could interfere or consume resources.
    - Go to `Settings > System > Reset options > Erase all data (factory reset)`.
    - `> ⚠️ Warning:` This will delete absolutely everything on the phone. Ensure all personal data is backed up elsewhere before proceeding!
2. **Update Android OS:**
    - After the reset, connect to Wi-Fi and update the phone to the latest stable Android version available for your device. This ensures you have the latest security patches and performance improvements.
    - Go to `Settings > System > System update`.

3. **Disable Unnecessary Features and Apps:**
    - **Remove Google Account:** While you might need it initially for the Play Store (though we'll use F-Droid), consider removing it after installing Termux to minimize background activity and data syncing.
    - **Disable Bloatware:** Disable or uninstall any pre-installed apps you don't need (`Settings > Apps > See all apps > [App Name] > Disable`).
    - **Developer Options:** Enable Developer options (`Settings > About phone`, tap 'Build number' 7 times).
        - **Keep screen on while charging:** This is crucial. Go to `Developer options` and toggle `Stay awake`.
        - **Disable animations:** Set `Window animation scale`, `Transition animation scale`, and `Animator duration scale` to `.5x` or `Off` for a snappier feel.
        - **USB debugging:** Enable this if you plan to access the phone via ADB from your computer for advanced troubleshooting, but disable it when not in use for security.
    - **Battery Optimization:** Ensure Termux and any other critical apps are not battery optimized, otherwise Android might kill them in the background. Go to `Settings > Apps > See all apps > Termux > Battery > Unrestricted`.
4. **Continuous Power:**
    - Connect the phone to its USB-C charger. Ensure it's a reliable charger that can provide continuous power. You want the phone to stay at 100% charge and not cycle the battery unnecessarily.
    - `> 💡 Tip:` Consider a smart plug that can monitor power usage or even remotely cycle power if the phone ever becomes unresponsive (though this is a last resort).

5. **Storage for Immich Data:**
    - Immich stores a lot of data: original photos, thumbnails, and a PostgreSQL database. The phone's internal storage might be sufficient for the OS and application files, but not for your photo library.
    - **microSD Card:** If your Pixel has a microSD slot (older models might, newer ones typically don't), this is a simple option. Format it as `Portable Storage` (not `Internal Storage`) so Termux can access it easily.
    - **External USB-C SSD:** This is the recommended choice for modern Pixels and better performance.
        - **Format:** Format the SSD as `exFAT` or `FAT32` on a computer for maximum compatibility with Android. `ext4` might work but could require root or specific kernel modules to mount directly in Termux.
        - **Connection:** Connect the SSD via a USB-C to USB-C cable directly to the phone. If your phone's USB-C port is only for charging or doesn't support host mode for external storage, you might need a USB-C OTG adapter that allows charging and data passthrough.
        - `> 🧠 Important:` Ensure the SSD is externally powered if it draws too much power from the phone, or use a powered USB-C hub. Most modern portable SSDs are efficient enough to run directly off the phone.
6. **Network Configuration:**
    - **Stable Wi-Fi:** Connect the phone to your home Wi-Fi network.
    - **DHCP Reservation:** Log into your home router's administration panel (usually `192.168.1.1` or `192.168.0.1`). Find the DHCP settings and reserve an IP address for your Pixel phone's MAC address. This ensures the phone always gets the same internal IP, making it easier to access.
        - **Where to find it:** Look for sections like `LAN Settings`, `DHCP Server`, `Address Reservation`, or `Static Lease`. You'll need the phone's MAC address (found in `Settings > About phone > Wi-Fi MAC address`).

Setting Up Termux and Essential Tools

Termux is your gateway to a Linux environment on Android. We'll install it and get some basic tools ready.

1. **Install Termux from F-Droid:**
    - **Why F-Droid?** The version of Termux on the Google Play Store is often outdated and no longer maintained. F-Droid provides the latest, actively developed version.
    - **Download F-Droid:** On your Pixel, open Chrome and navigate to `<https://f-droid.org/>`. Download and install the F-Droid client app. You'll need to allow installation from unknown sources.
    - **Install Termux:** Open the F-Droid app, search for "Termux", and install it.
2. **Initial Termux Setup Commands:**
    - Open Termux. You'll see a basic shell prompt.
    - First, update and upgrade all packages:

```
        pkg update -y && pkg upgrade -y
```

```
- Grant storage permissions (this is crucial for accessing your external
storage):
```

```
        termux-setup-storage
```

```
    This will pop up an Android permission request. Grant it.
- Install essential tools: `git` for cloning repositories, `wget` for
downloading files, and `nano` (or `vim`) for editing text files.
```

```
        pkg install git wget nano -y
```

```
- Verify installations:
```

```
        git --version
        # Expected output: git version 2.44.0 (or similar)
```

```
        wget --version
        # Expected output: GNU Wget 1.21.4 built on linux-android... (or
similar)

        nano --version
        # Expected output: GNU nano, version 7.2 (or similar)
```

1. **Mounting External Storage (if not automatically mounted):**
    - Android usually mounts external storage under `/sdcard/` or `/storage/`. After `termux-setup-storage`, Termux creates symlinks.
    - Your external SSD or microSD card should be accessible via `~/storage/external-1` or similar.
    - Check available storage:

```
        ls -l ~/storage
        # Expected output might include:
        # lrwxrwxrwx 1 u0_a... u0_a... 25 2026-05-19 10:00 external-1 -> /
storage/XXXX-XXXX
        # lrwxrwxrwx 1 u0_a... u0_a... 25 2026-05-19 10:00 shared -> /sdcard
```

```
    Note the path to your external storage (e.g., `~/storage/external-1`).
We'll refer to this as `EXTERNAL_STORAGE_PATH` later.
```

Installing Docker and Docker Compose on Android

This is where it gets interesting. Docker relies on Linux kernel features (cgroups, namespaces) that aren't directly exposed or fully implemented in stock Android's kernel for user-space applications. You can't run Docker natively in Termux.

The Solution: `proot-distro`

`proot-distro` allows you to install and run full Linux distributions (like Ubuntu or Debian) within Termux, using `proot` to emulate the necessary environment. We'll install Docker and Docker Compose inside this `proot-distro` environment.

1. **Install `proot-distro`:**

```
    pkg install proot-distro -y
```

```
- Verify installation:
```

```
        proot-distro list
        # Expected output: Lists available distributions like ubuntu, debian,
alpine, etc.
```

1. **Install a Linux Distribution (e.g., Ubuntu):**

```
    proot-distro install ubuntu
```

```
This will download and set up an Ubuntu rootfs. It might take a while
depending on your internet speed.
```

1. **Enter the Ubuntu Environment:**

```
    proot-distro login ubuntu
```

```
Your prompt will change (e.g., `root@localhost:~#`), indicating you're now
inside the Ubuntu chroot-like environment.
```

1. **Install Docker within Ubuntu:**

    - First, update packages inside Ubuntu:

```
        apt update && apt upgrade -y
```

```
- Install necessary packages for Docker:
```

```
        apt install apt-transport-https ca-certificates curl gnupg lsb-
release -y
```

```
- Add Docker's official GPG key:
```

```
        curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --
dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

```
- Add the Docker APT repository:
```

```
        echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/
keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu
\
        $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list
> /dev/null
```

```
- Update APT index again and install Docker Engine:
```

```
        apt update
        apt install docker-ce docker-ce-cli containerd.io -y
```

```
- `> ⚠️ Warning:` Docker daemon (dockerd) will likely *not* start
automatically or successfully in `proot-distro` due to kernel limitations.
This is expected. We will run it manually or use a wrapper.
```

1. **Install Docker Compose within Ubuntu:**
    - Download the latest stable Docker Compose binary. Check Docker Compose releases for the latest version. Replace `v2.26.1` with the current stable version.

```
        curl -L "https://github.com/docker/compose/releases/download/v2.26.1/
docker-compose-linux-aarch64" -o /usr/local/bin/docker-compose
```

```
- Make it executable:
```

```
        chmod +x /usr/local/bin/docker-compose
```

```
- Verify Docker Compose installation:
```

```
        docker-compose version
        # Expected output:
        # Docker Compose version v2.26.1 (or similar)
        # Docker Engine:
        #  Version:           26.1.3 (or similar)
        #  API version:       1.45 (or similar)
        #  Go version:        go1.22.2 (or similar)
```

```
        #  Git commit:        e705afc (or similar)
        #  Built:             Mon May 20 10:14:04 2026 (or similar)
        #  OS/Arch:           linux/arm64 (or similar)
```

1. **Starting Docker Daemon (Crucial Step):** Since `systemd` isn't available in `proot-distro`, you need to start the Docker daemon manually.

```
    dockerd &
```

```
This command will start the Docker daemon in the background. You might see
some warnings or errors about `cgroupfs` or `systemd`, but as long as it
doesn't immediately crash, it should be working enough for `docker-compose`.
- Verify Docker daemon is running:
```

```
        docker ps
        # Expected output: An empty table with "CONTAINER ID", "IMAGE", etc.
if no containers are running, but it means the daemon is active.
```

```
    If you get an error like `Cannot connect to the Docker daemon`, `dockerd`
didn't start correctly. Check `docker logs` (if it's logging somewhere) or try
`dockerd --debug &` for more output.

`> 📌 Key Idea:` Because `proot-distro` is not a full virtual machine,
`dockerd` runs with some limitations. It's often sufficient for single-user
applications like Immich, but don't expect enterprise-grade stability or
performance.
```

Configuring and Deploying Immich

Now that Docker and Docker Compose are ready inside your Ubuntu environment, we can deploy Immich.

1. **Create a Directory for Immich:** Still inside the `proot-distro login ubuntu` environment:

```
    mkdir -p ~/immich
    cd ~/immich
```

1. **Download the Immich `docker-compose.yml`:** Immich provides a template `docker-compose.yml` and `.env` file. It's best to clone their example.

```
    git clone https://github.com/immich-app/immich-docs.git
    cp immich-docs/docs/install/docker-compose/example.docker-compose.yml dock
er-compose.yml
    cp immich-docs/docs/install/docker-compose/example.env .env
    rm -rf immich-docs # Clean up the cloned repo
```

1. **Edit the `.env` File:** This file contains crucial environment variables for Immich.

```
    nano .env
```

```
- **`UPLOAD_LOCATION`:** This is the most important setting. It defines where
Immich stores your original photos. Change it to point to your external
storage.
    - Recall `EXTERNAL_STORAGE_PATH` from Termux setup (e.g., `~/storage/
external-1`). We need to map this path into the Docker container.
    - In your `.env` file, set:
```

```
            UPLOAD_LOCATION=/mnt/immich_data
```

```
        (We'll map `~/storage/external-1` to `/mnt/immich_data` in the
`docker-compose.yml`)
- **`DB_PASSWORD`:** Change this to a strong, unique password.
- **`REDIS_PASSWORD`:** Change this to a strong, unique password.
- **`TYPESENSE_API_KEY`:** Change this to a strong, unique key.
- **`IMMICH_WEB_URL`:** For now, you can leave this as `<http://
localhost:2283`>, or set it to your phone's IP address (e.g., `<http://
192.168.1.100:2283`>). This will be updated for Tailscale later.
- Save and exit (`Ctrl+X`, `Y`, `Enter`).
```

1. **Edit the `docker-compose.yml` File:** We need to add the volume mapping for your external storage.

```
    nano docker-compose.yml
```

```
- Locate the `volumes` section for the `immich-server` and `immich-
microservices` services.
- Add a new volume mapping that links your Termux external storage path to the
`UPLOAD_LOCATION` you defined in `.env`.
- **Important:** The path `~/storage/external-1` is relative to the *Termux*
home directory. When inside `proot-distro`, Termux's home directory is
typically mounted at `/root` or `/home/user`. To access Termux's storage from
inside `proot-distro`, the easiest way is to use `/data/data/com.termux/files/
```

```
home/storage/external-1` (assuming Termux is installed as `com.termux`). If
you used `termux-setup-storage` and it created `~/storage/external-1`, then
within the `proot-distro` environment, this path is usually accessible
directly from `/data/data/com.termux/files/home/storage/external-1`.

Here's a complete `docker-compose.yml` example. **Remember to replace `/data/
data/com.termux/files/home/storage/external-1` with your actual external
storage path.**
```

```
    version: "3.8"

    services:
      immich-server:
        container_name: immich_server
        image: ghcr.io/immich-app/immich-server:release
        command: ["start.sh", "immich"]
        volumes:
          - ${UPLOAD_LOCATION}:/usr/src/app/upload
          - /etc/localtime:/etc/localtime:ro
          # Mount Termux's external storage into the container
          - /data/data/com.termux/files/home/storage/external-1:/mnt/
immich_data # <--- ADJUST THIS PATH
        env_file:
          - .env
        ports:
          - 2283:3001
        depends_on:
          - immich-redis
          - immich-database
          - immich-microservices
        restart: always

      immich-microservices:
        container_name: immich_microservices
        image: ghcr.io/immich-app/immich-microservices:release
        command: ["start.sh", "microservices"]
        volumes:
          - ${UPLOAD_LOCATION}:/usr/src/app/upload
          - /etc/localtime:/etc/localtime:ro
          # Mount Termux's external storage into the container
          - /data/data/com.termux/files/home/storage/external-1:/mnt/
immich_data # <--- ADJUST THIS PATH
        env_file:
          - .env
        depends_on:
          - immich-redis
          - immich-database
        restart: always

      immich-machine-learning:
        container_name: immich_machine_learning
        image: ghcr.io/immich-app/immich-machine-learning:release
        volumes:
          - ${UPLOAD_LOCATION}:/usr/src/app/upload
          - /etc/localtime:/etc/localtime:ro
          # Mount Termux's external storage into the container
          - /data/data/com.termux/files/home/storage/external-1:/mnt/
immich_data # <--- ADJUST THIS PATH
        env_file:
          - .env
```

```
        restart: always

      immich-web:
        container_name: immich_web
        image: ghcr.io/immich-app/immich-web:release
        env_file:
          - .env
        ports:
          - 2283:80
        depends_on:
          - immich-server
        restart: always

      immich-redis:
        container_name: immich_redis
        image: redis:6.2-
alpine@sha256:757697415414002621014e7a0305a415a770381622345517a944208a32947116
        env_file:
          - .env
        restart: always

      immich-database:
        container_name: immich_database
        image: postgres:14-
alpine@sha256:224c66004b752254e20b08064972e90e722a4501a351187d722d57d07d188219
        env_file:
          - .env
        environment:
          POSTGRES_USER: ${DB_USERNAME}
          POSTGRES_PASSWORD: ${DB_PASSWORD}
          POSTGRES_DB: ${DB_DATABASE_NAME}
        volumes:
          - immich_pgdata:/var/lib/postgresql/data
        restart: always

      immich-typesense:
        container_name: immich_typesense
        image: typesense/
typesense:0.25.1@sha256:9199990e181c00290535a0921477123992167d4fdf8b39c7198a28
7930999554
        env_file:
          - .env
        volumes:
          - immich_tsdata:/data
        restart: always

    volumes:
      immich_pgdata:
        name: immich_pgdata
      immich_tsdata:
        name: immich_tsdata
```

```
Save and exit (`Ctrl+X`, `Y`, `Enter`).
```

1. **Deploy Immich with Docker Compose:** Ensure `dockerd` is running (if you exited the session, you'll need to `proot-distro login ubuntu` and then `dockerd &` again).

```
    docker-compose up -d
```

```
This command will download all necessary Docker images and start the Immich
containers. This will take a significant amount of time, especially
downloading images over Wi-Fi on a phone's CPU.
```

1. **Verify Immich Containers:**

```
    docker ps
    # Expected output: All Immich containers (server, microservices, machine-
learning, web, redis, database, typesense) should be listed with "Up" status.
```

```
If any container is not "Up", check its logs:
```

```
    docker logs immich_server # or immich_microservices, etc.
```

Initial Data Import and Sync

With Immich running, it's time to get your photos in!

1. **Access Immich Web UI:**
    - Open a web browser on your phone (if you dare, performance might be poor) or, more practically, on another device on your home network.
    - Navigate to `<http://[YOUR_PIXEL_IP_ADDRESS]:2283>`. Replace `[YOUR_PIXEL_IP_ADDRESS]` with the static IP you reserved for your phone.
    - You should see the Immich login/signup page. Create your first admin user account.

2. **Upload Existing Photos:**
    - **Web Uploader:** For a small number of photos, you can use the web interface's upload feature.
    - **CLI Uploader:** Immich provides a CLI tool (`immich-go`) for bulk uploads, which is more robust.
        - You'd typically run this from your computer, pointing it to your Immich server.
        - Download `immich-go` from the Immich GitHub releases for your computer's OS.
        - Example command from your computer:

```
            ./immich-go upload --server=http://[YOUR_PIXEL_IP_ADDRESS]:2283 --
api-key=[YOUR_IMMICH_API_KEY] /path/to/your/photos
```

```
        You can generate an API key in the Immich web UI under `Settings >
User Settings > API Keys`.
- **Directly Place Files:** If you've placed your photos directly onto the
external storage at `/data/data/com.termux/files/home/storage/external-1/` (or
whatever path you mapped to `/usr/src/app/upload` in `docker-compose.yml`),
Immich won't automatically detect them. You'll need to use the "Scan Library"
feature in the Immich web UI (`Settings > Admin > Library`) to import them.
```

1. **Initial Indexing:**
    - `> ⚠️ Warning:` The initial indexing and thumbnail generation process will be very CPU and I/O intensive. Your Pixel phone will likely get warm, and the process will take a long time for large libraries (hours, possibly days, for tens of thousands of photos). Be patient.
    - Monitor progress in the Immich web UI.

Securing Remote Access with Tailscale

Accessing Immich from outside your home network securely without opening ports on your router is where Tailscale shines.

1. **Tailscale Account:**
    - If you don't have one, sign up for a free Tailscale account at `<https://tailscale.com/>`.

2. **Install Tailscale on the Android Pixel Phone:**
    - **From F-Droid/Play Store (Recommended):** The easiest way is to install the official Tailscale app from F-Droid or the Google Play Store on your Pixel phone.
        - Open the app, log in with your Tailscale account, and enable the VPN. This will add your phone to your Tailnet.
    - **Via Termux (Advanced, for CLI access):** If you prefer to manage Tailscale entirely from the command line within your `proot-distro` environment:
        - Log into your Ubuntu environment: `proot-distro login ubuntu`
        - Install Tailscale:

```
            curl -fsSL https://pkgs.tailscale.com/stable/ubuntu/
jammy.noarmor.gpg | apt-key add -
            curl -fsSL https://pkgs.tailscale.com/stable/ubuntu/
jammy.tailscale-key.gpg | apt-key add -
            echo "deb https://pkgs.tailscale.com/stable/ubuntu jammy main" | t
ee /etc/apt/sources.list.d/tailscale.list
            apt update
            apt install tailscale -y
```

```
        (Note: `jammy` is for Ubuntu 22.04, adjust if you installed a
different `proot-distro` version).
    - Start Tailscale and authenticate:
```

```
            tailscaled & # Start the daemon in background
            tailscale up
```

```
        This will output an authentication URL. Copy it, open it in a browser
on another device, and log in with your Tailscale account. This links your
Pixel to your Tailnet.
    - `> 💡 Tip:` Using the Android app is generally easier for a server like
this, as it handles the VPN connection more robustly in the background.
```

1. **Install Tailscale on Client Devices:**
    - Install the Tailscale client on your computer, laptop, and other mobile devices (iPhone, iPad, etc.) and log them into the same Tailscale account.

2. **Access Immich via Tailscale:**
    - Once all devices are connected to your Tailnet, your Pixel phone will have a Tailscale IP address (e.g., `100.x.y.z`).
    - On any device connected to your Tailnet, open a browser and navigate to `<http://[YOUR_PIXEL_TAILSCALE_IP_ADDRESS]:2283>`.
    - This provides secure, encrypted access to your Immich server from anywhere, without needing port forwarding on your router.
    - `> 📌 Key Idea:` Tailscale creates a secure mesh network, making your devices act as if they are all on the same local network, regardless of their physical location.

Setting Up Immich Client Apps

Immich offers official mobile apps that provide a Google Photos-like experience, including automatic backup.

1. **Download Immich Mobile App:**
    - Download the Immich mobile app from the Google Play Store (for Android) or Apple App Store (for iOS) on your daily driver phone(s).
2. **Configure App to Connect to Your Server:**
    - Open the Immich app.
    - You'll be prompted for the server address. Enter your Pixel phone's **Tailscale IP address and port**: `<http://[YOUR_PIXEL_TAILSCALE_IP_ADDRESS]:2283>`.
    - Log in with your Immich username and password.
    - **Enable Automatic Backup:** In the app settings, configure automatic photo and video backup to your Immich server. This is the core functionality you're after!

Data Management, Backup, and External Storage

A self-hosted photo library is only as good as its backup strategy. Data loss is a real risk, so plan accordingly.

1. **Understanding Immich's Data Structure:**
    - Immich stores original assets, generated thumbnails, and metadata.
    - The `UPLOAD_LOCATION` (mapped to `/mnt/immich_data` in our Docker setup) holds your actual photos and videos.
    - The PostgreSQL database (volume `immich_pgdata`) stores all metadata, user information, albums, and facial recognition data.
    - The Typesense data (volume `immich_tsdata`) stores search indexes.
2. **Accessing External Storage from Termux/Proot:**
    - As established, your external storage is mounted somewhere like `/data/data/com.termux/files/home/storage/external-1` from within your `proot-distro` environment.
    - This is where your actual photo files reside.
3. **Backup Strategy (Crucial!):**
    - **Regular Backups:** You need to regularly back up the `UPLOAD_LOCATION` (your photos) and the PostgreSQL database.
    - **Method 1: `rsync` to another device:**
        - From your main computer (connected via Tailscale), you can `rsync` the photos directly from the Pixel's external storage.
        - First, enable SSH in your `proot-distro` Ubuntu environment:

```
            proot-distro login ubuntu
            apt install openssh-server -y
            mkdir -p ~/.ssh # Create .ssh if it doesn't exist
            # Edit sshd_config to allow password login for simplicity, or set
up SSH keys
            nano /etc/ssh/sshd_config
            # Change PasswordAuthentication yes (and PermitRootLogin yes if
logging in as root)
            # Restart SSH: service ssh start
            # Set a password for root: passwd root
```

```
        `> ⚠️ Warning:` Allowing `PermitRootLogin yes` and
`PasswordAuthentication yes` is less secure. For a production environment,
always use SSH keys. For a homelab, it might be acceptable if access is
```

```
restricted by Tailscale.
    - From your computer (also on Tailscale):
```

```
            rsync -avz --delete root@[YOUR_PIXEL_TAILSCALE_IP_ADDRESS]:/data/
data/com.termux/files/home/storage/external-1/ /path/to/your/backup/location/
immich_photos/
```

```
- **Method 2: Database Dump:**
    - Inside `proot-distro login ubuntu`, you can dump the PostgreSQL
database:
```

```
            cd ~/immich # Go to your immich directory
            docker-compose exec immich-database pg_dumpall -U immich > immich_
db_backup.sql
```

```
    - Then `rsync` this `immich_db_backup.sql` file to your backup location.
- **Automation:** Use `cron` inside your `proot-distro` environment to
schedule these backups.
```

```
        crontab -e
```

```
    Add lines like:
```

```
        0 3 * * * cd /root/immich && docker-compose exec immich-database
pg_dumpall -U immich > immich_db_backup.sql && rsync -avz
immich_db_backup.sql /data/data/com.termux/files/home/storage/backup_target/
immich_db_backup.sql
```

```
    (Adjust `/data/data/com.termux/files/home/storage/backup_target/` to a
path on your external storage that is *also* backed up off-device).
```

1. **External Storage Health:**
    - Regularly check the health and free space of your external SSD/microSD card.
    - From Termux: `df -h /data/data/com.termux/files/home/storage/external-1`

Long-Term Reliability and Maintenance

Keeping your Pixel Immich server running smoothly requires periodic attention.

1. **Keep Android OS Updated:**
    - Periodically check for and install Android system updates. These often include security patches and performance improvements that can benefit your server.
2. **Update Termux Packages:**
    - Regularly update Termux and its installed packages.
    - From Termux:

```
pkg update -y && pkg upgrade -y
```

1. **Update `proot-distro` Environment:**
    - Log into your Ubuntu environment and update its packages.
    - From Termux:

```
proot-distro login ubuntu
apt update && apt upgrade -y
exit # Exit Ubuntu environment
```

1. **Update Docker and Docker Compose:**
    - Follow the same installation steps as before to get the latest versions of `docker-ce` and `docker-compose` within your `proot-distro` environment.
2. **Update Immich:**
    - Immich is under active development, so updates are frequent.
    - From within your `proot-distro` Ubuntu environment, navigate to your Immich directory:

```
cd ~/immich
docker-compose pull # Downloads the latest Immich container images
docker-compose up -d # Recreates containers with new images
docker image prune -f # Clean up old images
```

```
- `> 🧠 Important:` Always check the [official Immich release notes](https://immich.app/docs/changelog) before updating for any breaking changes or specific migration steps.
```

1. **Monitoring:**
    - **Phone Temperature:** Keep an eye on the phone's temperature, especially during heavy indexing. Ensure it has good airflow.
    - **Resource Usage:** You can use `htop` (install `apt install htop` in Ubuntu) within your `proot-distro` environment to monitor CPU and RAM usage.
    - **Log Files:** Periodically check Docker container logs for errors: `docker logs immich_server`.
2. **Dealing with Android Killing Termux:**
    - Android's aggressive battery management can sometimes kill background apps, including Termux.
    - Ensure Termux is set to "Unrestricted" battery usage (`Settings > Apps > Termux > Battery`).
    - Consider using a Termux widget or a small script to restart `proot-distro` and `dockerd` if it gets killed.
    - `> ⚠️ What can go wrong:` If Termux is killed, your Immich server goes down. You'll need to manually restart `proot-distro login ubuntu`, `dockerd &`, and `docker-compose up -d` in your Immich directory.

Security Hardening and Best Practices

Running a server on an Android phone introduces unique security considerations.

1. **Keep Android OS Updated:**
    - The most fundamental security measure. Android security patches address vulnerabilities that could affect the entire system, including Termux.
2. **Strong Passwords for Immich:**
    - Use long, complex, and unique passwords for your Immich admin account and for the `DB_PASSWORD`, `REDIS_PASSWORD`, and `TYPESENSE_API_KEY` in your `.env` file.

3. **Tailscale for Remote Access:**
    - **No Open Ports:** By using Tailscale, you avoid opening any ports on your home router to the internet. This is a massive security win, as it eliminates the most common attack vector for home servers.
    - **Access Control:** Tailscale allows you to control which devices in your Tailnet can access which services.
4. **Disable Unused Services:**
    - **ADB/Developer Options:** When not actively debugging, disable `USB debugging` in Developer Options.
    - **SSH (if enabled):** If you set up SSH for `rsync`, ensure it's configured with SSH keys rather than passwords, and only allow access from trusted Tailscale IPs.
5. **Physical Security of the Phone:**
    - The phone itself is your server. Keep it in a secure location where it won't be easily stolen, damaged, or tampered with.
6. **Regular Backups:**
    - As detailed above, regular, off-device backups are your last line of defense against data loss due to hardware failure, corruption, or security incidents.
7. **Review Immich Permissions:**
    - In the Immich web UI, manage user roles and permissions carefully. Don't grant admin access to everyone.

Troubleshooting Common Issues and Performance Limitations

You're running a full server stack on a phone; expect some quirks!

1. **Docker Daemon Not Starting (`dockerd &` fails):**
    - **Cause:** `proot-distro` environment might lack necessary kernel capabilities or `cgroupfs` is not properly emulated.
    - **Fix:** Ensure you are inside the `proot-distro` environment (`proot-distro login ubuntu`). Sometimes, simply trying `dockerd &` again works. Check `dmesg` (if available) or `journalctl` (unlikely in `proot-distro`) for more specific errors. Often, the warnings are harmless, and Docker still functions. If it consistently fails, your specific Android kernel/Termux setup might not be compatible.

2. **Immich Containers Not Starting or Crashing:**
    - **Cause:** Incorrect `docker-compose.yml` or `.env` configuration, resource limitations, or database issues.
    - **Fix:**
        - Check `docker logs [container_name]` (e.g., `docker logs immich_server`). Look for error messages.
        - Ensure `DB_PASSWORD`, `REDIS_PASSWORD`, `TYPESENSE_API_KEY` in `.env` are correctly set and match the `docker-compose.yml` (they should if you used the example).
        - Verify the `UPLOAD_LOCATION` path in `.env` and the volume mapping in `docker-compose.yml` are correct and accessible by Termux.
        - **Example Error:**

```
            immich_server | Error: connect ECONNREFUSED 172.18.0.3:5432
            immich_server |     at TCPConnectWrap.afterConnect [as oncomplete]
(node:net:1247:16) {
            immich_server |   errno: -111,
            immich_server |   code: 'ECONNREFUSED',
            immich_server |   syscall: 'connect',
            immich_server |   address: '172.18.0.3',
            immich_server |   port: 5432
            immich_server | }
```

```
        **Cause:** `immich-database` (PostgreSQL) container isn't running or
isn't accessible.
        **Fix:** Check `docker ps` to ensure `immich_database` is `Up`. If
not, check its logs (`docker logs immich_database`). It might be failing due
```

```
to a corrupted `immich_pgdata` volume (try `docker volume rm immich_pgdata`
*only if you don't care about existing data* and restart compose).
```

1. **Slow Performance / Freezing during Indexing:**
    - **Cause:** CPU, RAM, or I/O bottlenecks. Pixel phones have limited resources compared to dedicated servers. The machine learning component is particularly demanding.
    - **Fix:**
        - **Patience:** Initial indexing simply takes a long time.
        - **Disable ML (Temporarily):** If performance is critical, you can comment out the `immich-machine-learning` service in `docker-compose.yml` for the initial import, then re-enable it later. This will speed up initial import but delay facial recognition, object detection, etc.
        - **External SSD:** Ensure you're using a fast external SSD over a microSD card for better I/O.
        - **Cooling:** Ensure the phone has adequate airflow to prevent thermal throttling.
2. **Termux Process Killed by Android:**
    - **Cause:** Android's aggressive memory management or battery optimizations.
    - **Fix:**
        - Ensure Termux is set to "Unrestricted" battery usage (`Settings > Apps > Termux > Battery`).
        - Consider installing a "Keep Alive" app from F-Droid (e.g., `Wake Lock`) to prevent the phone from deep sleeping too aggressively, though this can impact battery health over time.
        - If it still happens, you'll need to manually restart `proot-distro login ubuntu`, `dockerd &`, and `docker-compose up -d` in your Immich directory.

3. **External Storage Not Accessible in Termux/Docker:**
    - **Cause:** Incorrect `termux-setup-storage` permissions, external drive not mounted by Android, or incorrect volume mapping in `docker-compose.yml`.
    - **Fix:**
        - Run `termux-setup-storage` again and grant permissions.
        - Verify the external drive is visible and accessible in Android's file manager.
        - Double-check the exact path in your `docker-compose.yml` volume mapping. The path within `proot-distro` to Termux's storage is critical.

This setup is a fantastic way to breathe new life into an old device, learn about Docker, and take control of your photo library. It's a true homelab triumph!