

AI-04498 GCP Dockerlab to NUC OnPrem

I'm behind a CGNAT PLDT Cebu internet fiber, can I still run my dockerlab using Tailscale and cloudflare dns gravelcloud.com? My current dockerlab is in GCP 4vcpu, 80GB 16GB RAM and the OpEx is around \$50/month on a 3 year CUD. How do I migrate my GCP dockerlab to my NUC pc on Premise? Possible? Can I travel with my NUC and still have public access to my application? The Postgres and Mariadb db backup and restore, are they covered in your procedure? I have 400mbps in FL, 90mbps in TX and 250mbps in Cebu, will my mobile NUC comparable to GCP speed? Find me a best Nuc in Amazon. Can I use a Dell optiplex micro pc sff, find one in amazon? BKM, Hyper-V or Proxmox? I'm familiar with Hyper-V. Build an Mobile OnPrem dockerlab spec .

Yes, absolutely. You can run your entire Docker setup behind PLDT's CGNAT in Cebu without issues.

Because CGNAT blocks all inbound port forwarding, traditional public IP binding won't work. However, tools like **Tailscale** and **Cloudflare Tunnels (ccloudflared)** establish outbound persistent connections. Since outbound traffic is unblocked on PLDT fiber, your homelab remains fully accessible from anywhere.

How Remote Access Works Behind CGNAT

Depending on who needs access to your apps on `gravelcloud.com` , you have two primary approaches:

1. Private Access (Just for You & Trusted Devices):

- Run **Tailscale** on your NUC.
- In Cloudflare DNS, set wildcard A records (e.g., `*.gravelcloud.com`) pointing to your NUC's Tailscale IP (`100.x.y.z`) with **Proxy status set to DNS Only (Grey Cloud)**.
- Traffic routes securely through your private Tailnet directly to Nginx Proxy Manager on the NUC.

2. Public Web Access (Accessible via standard internet without Tailscale on clients):

- Run **Cloudflare Tunnel (ccloudflared)** in a Docker container on your NUC.

- Map hostnames (e.g., `app.gravelcloud.com`) in Cloudflare Zero Trust panel to route directly to your local reverse proxy or container ports. Zero router ports opened.

Step-by-Step Migration from GCP to Local NUC

Migrating from GCP to an on-premise NUC is straightforward since your stack is already containerized with Docker.

Step 1: Prepare the NUC Host Environment

1. Install a headless Server OS (like **Ubuntu Server 24.04 LTS**).
2. Install **Docker** and the **Docker Compose** plugin.
3. Install **Tailscale** on the NUC host system:

Bash

```
curl -fsSL https://tailscale.com/install.sh | sh
sudo tailscale up
```

4. Record your NUC's Tailscale IPv4 address (`100.x.y.z`) using `tailscale ip -4`.

Step 2: Back Up Data from GCP

On your GCP instance, stop the running containers to guarantee database consistency before archiving:

Bash

```
# Stop containers gracefully
docker compose down

# Archive application configs, persistent bind mounts, and DB volumes
tar -czvf dockerlab_backup.tar.gz /path/to/your/docker-configs /var/lib/docker/volumes
```

Step 3: Transfer Archives to the NUC

Use `scp` or `rsync` over Tailscale (or direct SSH if both are on the same local network):

Bash

```
scp dockerlab_backup.tar.gz user@<NUC-Local-or-Tailscale-IP>:/home/user/
```

Step 4: Restore and Verify on the NUC

1. Extract the backup on your NUC to match your expected directory layout.
2. Update environment variables in your `.env` files (such as new local IP bindings or revised domain URLs if needed).
3. Bring the stack online:
Bash

```
docker compose up -d
```

Step 5: Update Cloudflare DNS / Routing

- **If using Tailscale + Nginx Proxy Manager:** Point your `gravelcloud.com` A/CNAME records in Cloudflare DNS to your NUC's Tailscale IP. Ensure DNS challenge is configured in Nginx Proxy Manager for wildcard Let's Encrypt SSL certificates.
- **If using Cloudflare Tunnels:** Install the `cloudflared` Docker container on the NUC and bind your public hostname routes in the Cloudflare Zero Trust dashboard.

Key Operational Advantages

- **OpEx Savings:** Eliminates the ~\$50/month GCP commit, saving ~\$600/year while giving you dedicated local hardware without cloud egress fees.
- **Security:** Zero open ports on your PLDT home router — your home network remains completely invisible to public port scanners.

Can I travel with my NUC and still have public access to my application? The Postgres and Mariadb db backup and restore, are they covered in your procedure?

Yes to both. You can travel anywhere with your NUC in your backpack and maintain full public access to `gravelcloud.com`, and database backup/restore requires explicit SQL dumps rather than simple file archiving.

Part 1: Traveling with Your NUC

When you travel, your public users won't experience any downtime or DNS issues because of how **Cloudflare Tunnels** (`cloudflared`) and **Tailscale** work:

- **No Port Forwarding or Static IP Required:** Cloudflare Tunnels create outbound encrypted connections (`localhost` $\rightarrow$ Cloudflare Edge). It doesn't matter what local IP address your hotel, Airbnb, or temporary Wi-Fi assigns to your NUC, nor does it matter if you're behind CGNAT, strict firewall rules, or mobile hotspots.
- **Instant Dynamic Routing:** As soon as your NUC boots up and hits any internet connection, `cloudflared` automatically connects back to Cloudflare's nearest edge server. Your domains

under `gravelcloud.com` instantly point to your NUC's new location without needing to change DNS records.

- **Seamless Remote Admin:** Tailscale will also automatically connect in the background, allowing you to SSH into the NUC or access Portainer using its Tailscale IP (`100.x.y.z`) from your phone or laptop.

Part 2: Database Backup & Restore (PostgreSQL & MariaDB)

Copying raw database volume folders directly (like `/var/lib/docker/volumes` or mounted `/data` folders) while containers are running can cause database corruption due to unwritten WAL files or active memory state.

To guarantee clean, consistent migrations, perform **logical SQL dumps** using `docker exec` before transferring data.

1. PostgreSQL (Export & Import)

Export (on GCP):

Run `pg_dumpall` (for all databases, users, and roles) or `pg_dump` (for a specific database):

Bash

```
# Export all databases and roles to a single SQL file
docker exec -t <postgres-container-name> pg_dumpall -U postgres > postgres_all_db.sql
```

Import (on NUC):

Bring up your new PostgreSQL container on the NUC, then restore the dump:

Bash

```
# Pipe the SQL dump directly into the fresh PostgreSQL container
cat postgres_all_db.sql | docker exec -i <postgres-container-name> psql -U postgres
```

2. MariaDB / MySQL (Export & Import)

Export (on GCP):

Use `mariadb-dump` or `mysqldump` to dump all databases:

Bash

```
# Export all MariaDB databases
docker exec <mariadb-container-name> mariadb-dump -u root -p"<ROOT_PASSWORD>" --all-databases > mariadb_all_db.sql
```

Import (on NUC):

Bring up your fresh MariaDB container on the NUC, then load the SQL dump:

Bash

```
# Restore all databases into the new container
cat mariadb_all_db.sql | docker exec -i <mariadb-container-name> mariadb -u root -p"<ROOT_PASSWORD>"
```

Complete Recommended Migration Workflow

1. **Dump Databases:** Generate `postgres_all_db.sql` and `mariadb_all_db.sql` on GCP.
2. **Stop Application Containers:** Run `docker compose stop` for apps using those databases (e.g., InvenTree, OpenProject, Zammad) so no new writes occur.
3. **Archive Application Files:** Compress app persistent bind mounts (uploads, media files, custom configuration files):

Bash

```
tar -czvf app_files.tar.gz /path/to/app/configs /path/to/app/data
```

4. **Transfer:** Use `scp` over Tailscale to copy the `.sql` dumps and `app_files.tar.gz` to your NUC.
5. **Restore:**
 - Extract `app_files.tar.gz` into the same paths on your NUC.
 - Start your PostgreSQL and MariaDB containers on the NUC.
 - Pipe the SQL dumps into the respective containers.
 - Start the rest of your application stack (`docker compose up -d`).

I have 400mbps in FL, 90mbps in TX and 250mbps in Cebu, will my mobile NUC comparable to GCP speed?

Yes — in fact, for overall application response time and database processing, your mobile NUC will likely feel *faster* than your \$50/month GCP instance.

To understand how performance compares, it helps to break down "speed" into **Compute/Storage Processing, Network Throughput**, and **Latency**.

1. Compute & Storage Speed: Huge Win for the NUC

In cloud environments like GCP, 4 vCPUs on a shared or cost-optimized machine (such as an e2 or n2 tier) correspond to vCPUs (hyperthreads), which are throttled alongside shared disk IOPS unless you pay for premium SSD persistent disks.

A modern physical NUC (e.g., Intel Core i5/i7 or modern multi-core x86 host) with a direct PCIe NVMe SSD provides **dedicated physical cores, higher single-core clock speeds, and uncapped local disk I/O**.

- **Database & File Operations:** PostgreSQL/MariaDB queries, file indexing (InvenTree/OpenProject), and container starts on the NUC will execute noticeably faster than on GCP persistent disk storage.

2. Network Throughput: Broadband vs. Datacenter Backbone

GCP offers multi-gigabit pipelines, but your residential connections are constrained by broadband limits:

Location	Typical Download	Critical Factor for Homelab: Upload Speed
Florida	400 Mbps	Often 20–40 Mbps (if cable) or 400 Mbps (if symmetric fiber)
Cebu (PLDT)	250 Mbps	~250 Mbps (PLDT Fiber is typically symmetric)
Texas	90 Mbps	~10–20 Mbps (if asymmetrical) or 90 Mbps (if symmetric)

Why Upload Speed Matters: When your NUC hosts applications, it **uploads** data to you/your clients.

- **Web UI & App Usage:** Standard self-hosted applications (dashboards, document management, ticketing platforms) use negligible bandwidth (~1 to 5 Mbps active burst). Even on a 90 Mbps connection in TX, application response times will be virtually instantaneous.
- **Heavy File Downloads:** If you download multi-gigabyte media or ISO files *from* your NUC while away, your download speed will cap out at whatever location's **upload limit** the NUC is plugged into.

3. Latency & Routing Dynamics

When hosting on a mobile NUC via Cloudflare Tunnels (`c1oudflared`) and Tailscale:

- **Local Access:** When you are sitting in the same room as your NUC (e.g., in Cebu or Florida), accessing it over local LAN / Tailscale gives you **sub-5ms latency**, making local workflows significantly faster than routing out to a GCP data center.
- **Remote Access (Travel):** When you travel, traffic routes through Cloudflare's nearest Edge data center. Latency will primarily depend on physical distance to the NUC:
 - *Accessing a Cebu-based NUC from Asia:* ~15–30ms.
 - *Accessing a Cebu-based NUC from Florida:* ~180–220ms (due to transpacific fiber distance, regardless of whether you use GCP or a NUC).

The Verdict

For running a self-hosted Docker lab (databases, management tools, web applications), **the NUC will comfortably match or exceed GCP's operational speed** while cutting your OpEx to zero. The only scenario where GCP holds a network advantage is when transferring massive continuous files over multi-gigabit connections.

Find me a best Nuc in Amazon

When selecting a mini PC for a 24/7 self-hosted Docker lab, the best choice depends on whether you prioritize official **Intel/ASUS NUC brand hardware**, **maximum network expansion**, or **budget efficiency**.

For a Docker lab running multiple web platforms and databases (like PostgreSQL and MariaDB), you'll want at least **32GB DDR5 RAM**, **fast NVMe storage**, and reliable **2.5GbE networking**.

Top Recommended Mini PCs on Amazon

1. Official Standard Choice: ASUS NUC 14 Pro

If you want the official successor to Intel's original NUC hardware line, the **ASUS NUC 14 Pro** is the gold standard.



ASUS NUC 14 Pro Mini PC

\$613.94-ASUS

4.3



(19)

The ASUS NUC 14 Pro delivers high single-core performance, dual Thunderbolt 4 ports, and low idle power consumption in a tiny footprint.

- **CPU:** Intel Core Ultra 5 125H or Core Ultra 7 155H
- **RAM / Storage:** Expandable up to 96GB DDR5 with dual M.2 NVMe slots
- **Networking:** 2.5GbE LAN + Wi-Fi 6E
- **Why it fits:** Outstanding stability for 24/7 Docker operation, toolless chassis access, and ultra-low idle power draw (~8–9W).

2. Advanced Homelab Pick: Minisforum MS-01

If you want a homelab-dedicated mini server with enterprise networking built right into the chassis, the **Minisforum MS-01** is widely considered the top homelab mini PC.



Minisforum UM760 Slim Mini PC AMD Ryzen 5 7640HS

\$655.00-MINISFORUM

4.0



(23)

The Minisforum UM760 Slim Mini PC AMD Ryzen 5 7640HS chassis line provides powerful multi-core processing, exceptional expansion, and effective heat management for dense server workloads.

- **CPU:** Intel Core i5-12600H or i9-13900H / AMD Ryzen options
- **Networking:** Dual 10GbE SFP+ ports + Dual 2.5GbE RJ45 ports
- **Storage:** Up to 3x M.2 NVMe slots + U.2 support + PCIe expansion slot
- **Why it fits:** Perfect if you plan to virtualize (e.g., Proxmox) or expand your networking setups down the road.

3. High-Value Alternative: Minisforum UM760 / Beelink SER / Geekom

For cost-effective multi-core processing with high RAM capacity, AMD Ryzen-based mini PCs offer substantial power per dollar.

- **CPU:** AMD Ryzen 5 7640HS or Ryzen 7 7840HS/8845HS
- **RAM / Storage:** Up to 64GB DDR5 and dual M.2 PCIe 4.0 slots
- **Why it fits:** Delivers 8 full physical cores and high multi-threaded performance for Docker databases at lower price points than official NUC kits.

What Specs to Look For on Amazon

1. **RAM:** Search for pre-configured bundles with **32GB DDR5** (or buy a barebone kit and add 2x16GB SO-DIMMs yourself).
2. **Storage:** Look for a **1TB PCIe 4.0 NVMe SSD** to give your PostgreSQL/MariaDB databases headroom.
3. **OS:** Most come pre-installed with Windows 11 Pro, which you can overwrite with **Ubuntu Server 24.04 LTS** or Proxmox VE for running your Docker lab.

If you prefer a direct link or specific bundle recommendations within a set budget (e.g., \$400 vs \$700), let me know and I can filter by your exact budget range!

Can I use a Dell optiplex micro pc sff, find one in amazon

Yes, absolutely. A Dell OptiPlex Micro Form Factor (MFF) or Small Form Factor (SFF) is one of the most popular and reliable choices in the entire homelab community for running Docker containers.

Because Dell manufactures these enterprise PCs in huge volumes for corporate environments, refurbished or renewed models offer incredible value, robust thermal design, and low power draw.

Dell OptiPlex Form Factors: Micro vs. SFF

- **OptiPlex Micro (MFF):** Tiny footprint (roughly the size of a paperback book). Uses ~10–15W at idle and easily fits into a backpack when traveling.
- **OptiPlex Small Form Factor (SFF):** Slightly larger desktop size. Offers additional PCIe expansion slots (great if you want to add extra 2.5GbE/10GbE network cards or extra 3.5" SATA drives).

Recommended Dell OptiPlex Models for Homelabs

1. Excellent Budget / Renewed Pick: Dell OptiPlex 3070 / 7070 Micro

For a cost-effective Docker host running apps like PostgreSQL, MariaDB, and Nginx Proxy Manager, a renewed 9th-generation Intel OptiPlex Micro offers high value.



Dell OptiPlex 3070 Micro Desktop Computer - Intel Quad Core i3 (9th Gen) Windows 11 Pro WiFi + Bluetooth Memory (RAM):32GB 500GB SSD (Solid State

\$299.99·Dell

4.8



(46)

The Dell OptiPlex 3070 Micro Desktop Computer - Intel Quad Core i3 (9th Gen) Windows 11 Pro WiFi + Bluetooth Memory (RAM):32GB 500GB SSD (Solid State is a pre-configured option offering 32GB RAM and an SSD in a tiny, travel-friendly chassis.

- **Processor:** Intel 9th Gen (Core i3 / i5)
- **Memory & Storage:** 32GB RAM with fast SSD boot drive
- **Networking:** Integrated Gigabit Ethernet + Wi-Fi
- **Why it fits:** Extremely low entry cost with ample RAM already installed for your Docker stack.

2. Modern Powerhouse: Dell Pro Micro / OptiPlex 7010 Micro

If you want new, current-generation enterprise hardware with DDR5 memory support and faster multi-core throughput, Dell's newer line provides maximum performance.



Dell Pro Micro QCM1250 Desktop Computer

\$1,418.51 · Dell

4.4



(1.1k+)

The Dell Pro Micro QCM1250 Desktop Computer delivers 20-core processing power, PCIe 4.0 NVMe storage, and high-speed USB-C connectivity for intensive homelab workloads.

- **Processor:** Intel Core Ultra 7 / 13th Gen Core i5/i7
- **Memory & Storage:** Up to 64GB DDR5 RAM, PCIe 4.0 NVMe SSD slots
- **Networking:** 1GbE / 2.5GbE LAN, Wi-Fi 6E, USB-C 3.2
- **Why it fits:** Premium build quality, near-silent operation under load, and exceptional multi-threaded performance for heavy database queries.

Key Things to Check Before Purchasing

1. **RAM:** Ensure it comes with **16GB or 32GB RAM** (or budget ~\$40–\$60 to upgrade SODIMMs yourself).
2. **Storage:** Verify it includes an **NVMe M.2 SSD** rather than an older spinning 2.5" hard drive.
3. **Power Adapter:** If buying renewed/refurbished on Amazon, double-check that the listing includes the external **Dell AC power adapter**.

Would you like help picking between a budget renewed \$200–\$300 model or a current-generation \$600+ model based on your target spend?

From ChatGPT

Building a Mobile On-Prem DockerLab to Replace My GCP Environment

I currently run my production DockerLab on **Google Cloud Platform (GCP)** with the following specifications:

- **4 vCPUs**
- **16 GB RAM**
- **80 GB SSD**
- **Ubuntu Server**
- **Docker & Docker Compose**
- **Nginx Proxy Manager**
- **Portainer**
- **PostgreSQL**
- **MariaDB/MySQL**
- **Docmost**
- **OpenEMR**
- **OpenProject**
- **InvenTree**
- **WordPress**
- Additional self-hosted services

The infrastructure costs approximately **\$50/month** under a **3-year Committed Use Discount (CUD)**.

Objective

I would like to eliminate the monthly cloud infrastructure cost by migrating my entire DockerLab to a **mobile on-premises server**, preferably a compact Intel NUC or similar mini PC that I can take with me while traveling.

Environment

My primary Internet connection is:

- **PLDT Fiber (Cebu)**
- Behind **Carrier-Grade NAT (CGNAT)**

I own the Cloudflare-managed domain:

- gravelcloud.com

I plan to use:

- **Tailscale**
 - **Cloudflare DNS**
 - Possibly **Cloudflare Tunnel**
 - Docker Compose
 - Nginx Proxy Manager
-

Questions

1. Running Behind CGNAT

Can I successfully host my DockerLab behind PLDT's CGNAT using:

- Tailscale
- Cloudflare DNS
- gravelcloud.com

without requiring a public IP address?

Can my applications remain securely accessible from anywhere on the Internet?

2. Migration from GCP

Is it practical to migrate my complete DockerLab from GCP to an on-premises mini PC?

Please provide a detailed migration procedure covering:

- Docker Compose projects
- Volumes
- Environment files
- Reverse proxy configuration
- SSL certificates
- Cloudflare configuration
- Tailscale configuration
- Nginx Proxy Manager
- PostgreSQL

- MariaDB
-

3. Database Backup and Restore

Does the migration procedure include complete backup and restoration of:

- PostgreSQL databases
- MariaDB databases

including:

- users
- permissions
- schemas
- stored procedures
- indexes
- application data

so that every application continues working exactly as before?

4. Mobile DockerLab

If I travel with the mini PC—for example between:

- Cebu
- Texas
- Florida

can I simply plug it into any Internet connection and have all applications become publicly accessible again without changing DNS records or reconfiguring my services?

5. Internet Performance

My available Internet connections are:

- Florida: **400 Mbps**
- Texas: **90 Mbps**
- Cebu: **250 Mbps**

How would a mobile on-prem DockerLab compare with my current GCP server in terms of:

- application responsiveness
- database performance
- remote access latency
- upload/download speeds

- overall user experience

Under what workloads would I notice a significant difference?

6. Hardware Recommendations

Please recommend the best hardware for a mobile DockerLab.

Compare current Amazon options for:

- Intel NUC
- ASUS NUC
- Beelink
- Minisforum
- Dell OptiPlex Micro
- Lenovo ThinkCentre Tiny
- HP EliteDesk Mini

Include recommendations based on:

- CPU performance
- memory capacity
- NVMe storage
- reliability
- power consumption
- noise level
- Linux compatibility
- virtualization performance
- overall value

Please identify your top recommendations available on Amazon.

7. Intel NUC vs Dell OptiPlex Micro

Would a refurbished Dell OptiPlex Micro Small Form Factor be a practical alternative to an Intel NUC?

Please compare:

- price
- reliability
- upgradeability
- cooling
- power efficiency

- portability
 - expected lifespan
 - suitability as a 24/7 DockerLab server
-

8. Hypervisor Selection

I am already familiar with Microsoft Hyper-V.

Should I continue using:

- Hyper-V
- Proxmox VE
- KVM
- Bare-metal Ubuntu with Docker

Please compare each option for:

- Docker workloads
 - backups
 - snapshots
 - disaster recovery
 - networking
 - storage
 - learning curve
 - resource overhead
 - long-term maintainability
-

9. Build a Mobile On-Prem DockerLab

Please design a complete production-grade Mobile On-Prem DockerLab specification, including:

- Recommended hardware
- CPU
- RAM
- NVMe storage layout
- Backup drives
- UPS
- Networking
- Virtualization platform
- Operating system
- Docker architecture

- Security hardening
- Monitoring
- Backup strategy
- Disaster recovery plan
- Cloudflare integration
- Tailscale integration
- CI/CD workflow
- GitHub integration
- PostgreSQL and MariaDB backup strategy
- Performance expectations
- Estimated power consumption
- Total hardware cost
- Migration roadmap from GCP

The design should prioritize reliability, portability, ease of maintenance, and low operating costs while supporting my existing Docker-based self-hosted applications.

Based on your current DockerLab and your long-term goal of mastering self-hosted infrastructure while reducing cloud operating expenses, I think your idea is not only practical, but is a common architecture used by many experienced homelab and small-business administrators.

Executive Summary

Yes, this migration is entirely feasible.

A mobile DockerLab can:

- Replace your current GCP VM
- Operate behind PLDT CGNAT
- Be reachable worldwide
- Travel between Cebu, Texas, and Florida
- Cost only a few dollars per month in electricity
- Eliminate the approximately **\$50/month** GCP VM cost
- Provide equal or better performance for most workloads

I would build it as follows:

Internet
|
|

Cloudflare DNS
|
Cloudflare Tunnel
|
Ubuntu [Server](#)
|
Docker
|
Nginx Proxy Manager
|
Portainer
|

PostgreSQL
MariaDB
Docmost
OpenEMR
OpenProject
WordPress
Inventree
Future AI Apps

Tailscale
|
Remote Administration

1. Running Behind PLDT CGNAT

The answer is **yes**.

CGNAT prevents inbound port forwarding, but modern overlay networks eliminate that requirement.

Public websites

Internet
|
Cloudflare DNS
|
Cloudflare Tunnel
|

Your DockerLab

No public IP required.

No port forwarding.

No DDNS.

No static IP.

Private administration

Laptop



Tailscale VPN



DockerLab

SSH

Portainer

pgAdmin

phpMyAdmin

Ubuntu

Everything works over the encrypted Tailscale network.

Recommended separation

Public:

- WordPress
- Docmost
- OpenEMR (if required)
- OpenProject

Private:

- Portainer
 - pgAdmin
 - phpMyAdmin
 - SSH
 - Monitoring
 - Grafana
 - Prometheus
-

2. Migration from GCP

Fortunately Docker Compose makes migration straightforward.

Step 1

Freeze applications.

```
docker compose down
```

Step 2

Backup databases

PostgreSQL

```
pg_dumpall
```

or individual databases

```
pg_dump
```

MariaDB

```
mysqldump --all-databases
```

Step 3

Copy Docker volumes

```
docker volume ls
```

```
docker run --rm \  
-v volume:/source \  
-v $(pwd):/backup \  
busybox \  
tar czf /backup/volume.tar.gz /source
```

Repeat for every volume.

Step 4

Copy configuration

```
docker-compose.yml
```

```
.env
```

```
nginx
```

```
Portainer data
```

```
SSL certificates
```

```
scripts
```

```
cron jobs
```

Step 5

Clone GitHub

Since your DockerLab is already stored in GitHub:

```
git clone
```

Step 6

Restore

```
docker compose up -d
```

Restore databases.

Restore volumes.

Done.

3. Database Migration

Yes.

Everything migrates.

Including:

- ✓ users
- ✓ permissions
- ✓ schemas
- ✓ indexes
- ✓ foreign keys
- ✓ triggers
- ✓ stored procedures
- ✓ sequences

✓ extensions

✓ application data

Applications should not notice the move if connection strings remain unchanged.

4. Traveling With Your Server

This is one of the biggest advantages of Cloudflare Tunnel and Tailscale.

Imagine:

Today

Cebu

↓

Tomorrow

Texas

↓

Next week

Florida

↓

Hotel

↓

Friend's house

↓

Office

Nothing changes.

The DockerLab reconnects outbound.

Cloudflare Tunnel re-establishes automatically.

Tailscale reconnects automatically.

DNS records stay exactly the same.

Your users continue accessing:

`doc.gravelcloud.com`

`wiki.gravelcloud.com`

`openemr.gravelcloud.com`

without any changes.

5. Performance Compared with GCP

CPU

Modern mobile CPUs outperform many older cloud VM cores.

Example:

Intel Core Ultra

Ryzen 7 8845HS

often deliver higher single-thread performance than a standard 4-vCPU cloud instance.

Storage

Local NVMe

`5,000–7,000 MB/s`

Typical cloud persistent SSD

500–1,200 MB/s

Database performance generally improves.

Memory

64 GB RAM allows:

larger PostgreSQL shared buffers

larger MariaDB buffer pools

filesystem cache

more containers

Latency

Local users

Faster.

Remote users

Dependent on your home Internet upload speed.

Upload Speed Matters

Suppose:

400 Mbps fiber

↓

Your DockerLab

↓

Internet

Everyone downloading files is limited by your home's upload capacity.

For business apps, this is rarely an issue.

For streaming or large backups, cloud infrastructure has an advantage.

6. Hardware Recommendations

For a portable server, I would prioritize:

 Rank	System ASUS NUC (Core Ultra)	Comments Best overall balance of performance, power efficiency,
 2	Minisforum MS series	Excellent CPU options and expandability
 3	Lenovo ThinkCentre Tiny	Enterprise reliability, especially refurbished units
4	Dell OptiPlex Micro	Outstanding value on the refurbished market
5	Beelink SER series	Very good value for home labs

Target specifications

- CPU: Intel Core Ultra 7 or AMD Ryzen 7 8845HS (or newer equivalent)
- RAM: 64 GB DDR5
- Storage:
 - 2 TB NVMe (primary)
 - Optional second 2 TB NVMe for backups or ZFS mirror
- Network: 2.5 GbE if available
- Wi-Fi: Optional (Ethernet preferred)
- USB4/Thunderbolt: Nice to have for fast external backups

7. Dell OptiPlex Micro vs NUC

Price	Excellent (especially refurbished)	Higher

Feature	Dell OptiPlex Micro	ASUS NUC
Cooling	Better sustained cooling	Compact but warmer under heavy load
Upgradeability	Easier serviceability	More limited
Power	Very low	Extremely low
Portability	Good	Excellent
Noise	Quiet	Very quiet
Linux support	Excellent	Excellent
24/7 Operation	Excellent	Excellent

Recommendation

If budget matters:

Refurbished Dell OptiPlex Micro is an outstanding choice.

If maximum portability matters:

ASUS NUC wins.

8. Hypervisor Selection

Bare-metal Ubuntu + Docker	Lowest overhead, simplest, fastest	No VM snapshots
Platform	Pros	Cons
KVM	Flexible and powerful	More manual setup
Hyper-V	Familiar if you already know it	Best suited to Windows-centric environments

For your workload, my recommendation is:

1. **Bare-metal Ubuntu Server + Docker** if all workloads are containerized.
2. **Proxmox VE** if you expect to run virtual machines alongside Docker, experiment with networking, or learn enterprise virtualization.

9. Production Mobile DockerLab Design

Hardware

- ASUS NUC or Dell OptiPlex Micro
- 64 GB RAM
- 2 TB NVMe SSD
- Optional second 2 TB SSD
- External 4 TB USB SSD for backups
- Small pure sine-wave UPS (around 600–900 VA)

Software Stack

Ubuntu **Server** LTS

|

Docker Engine

|

Docker Compose

|

Nginx Proxy Manager

Cloudflare Tunnel

Tailscale

Portainer

PostgreSQL

MariaDB

Docmost

OpenEMR

WordPress

OpenProject

Inventree

Prometheus

Grafana

Loki

Uptime Kuma

Watchtower (optional)

Security

- UFW firewall
- SSH keys only
- Fail2ban
- Automatic security updates
- Container image scanning
- Secrets managed outside Git
- Encrypted backups

Backup Strategy

- **Databases:** Nightly `pg_dump` and `mysqldump`
- **Volumes:** Daily compressed archives or filesystem snapshots
- **Configuration:** Git repository (excluding secrets)
- **Off-site:** Encrypted copies to cloud storage or another trusted location
- **Recovery testing:** Perform periodic full restore tests to verify backups

Monitoring

- Uptime Kuma
- Prometheus
- Grafana
- Node Exporter
- Docker metrics
- SMART disk monitoring

Estimated Power

Typical idle:

- 12–20 W

Moderate load:

- 25–40 W

Heavy load:

- 45–65 W

That is dramatically lower than a traditional rack server.

Migration Roadmap

Purchase hardware	1 day
Install Ubuntu Server	30 minutes
Phase	Duration
Configure Cloudflare Tunnel	30 minutes
Configure Tailscale	15 minutes
Restore databases	30–60 minutes
Restore volumes	1–2 hours (depends on data size)
Validate applications	1–2 hours
Update monitoring and backups	1 hour
Decommission GCP after successful testing	Same day or after a validation period

Final Recommendation

Given your current stack (Ubuntu, Docker Compose, Nginx Proxy Manager, Portainer, PostgreSQL, MariaDB, Docmost, OpenEMR, OpenProject, InvenTree, and WordPress), a **mobile on-prem DockerLab** is a strong fit. It aligns with your goal of deepening your DevOps skills, reducing recurring infrastructure costs, and maintaining full control over your environment.

If I were designing this from scratch today, I would choose:

- **Platform:** Bare-metal Ubuntu Server LTS
- **Hardware:** ASUS NUC (Core Ultra) or a refurbished Dell OptiPlex Micro with 64 GB RAM and a 2 TB NVMe SSD
- **Remote access:** Tailscale for administration
- **Public access:** Cloudflare Tunnel with your gravelcloud.com domain
- **Container management:** Docker Compose with Portainer
- **Backups:** Automated database dumps, volume backups, Git-managed configuration, and encrypted off-site copies
- **Monitoring:** Prometheus, Grafana, and Uptime Kuma

This architecture should comfortably support your current applications while leaving room to add additional self-hosted services and AI-related workloads in the future.

Gemini

Building a Mobile On-Prem DockerLab to Replace My GCP Environment

I currently run my production DockerLab on **Google Cloud Platform (GCP)** with the following specifications:

- **4 vCPUs**
- **16 GB RAM**
- **80 GB SSD**
- **Ubuntu Server**
- **Docker & Docker Compose**
- **Nginx Proxy Manager**
- **Portainer**
- **PostgreSQL**
- **MariaDB/MySQL**
- **Docmost**
- **OpenEMR**
- **OpenProject**
- **InvenTree**
- **WordPress**
- Additional self-hosted services

The infrastructure costs approximately **\$50/month** under a **3-year Committed Use Discount (CUD)**.

Objective

I would like to eliminate the monthly cloud infrastructure cost by migrating my entire DockerLab to a **mobile on-premises server**, preferably a compact Intel NUC or similar mini PC that I can take with me while traveling.

Environment

My primary Internet connection is:

- **PLDT Fiber (Cebu)**
- Behind **Carrier-Grade NAT (CGNAT)**

I own the Cloudflare-managed domain:

- **gravelcloud.com**

I plan to use:

- **Tailscale**
 - **Cloudflare DNS**
 - Possibly **Cloudflare Tunnel**
 - Docker Compose
 - Nginx Proxy Manager
-

Questions

1. Running Behind CGNAT

Can I successfully host my DockerLab behind PLDT's CGNAT using:

- Tailscale
- Cloudflare DNS
- gravelcloud.com

without requiring a public IP address?

Can my applications remain securely accessible from anywhere on the Internet?

2. Migration from GCP

Is it practical to migrate my complete DockerLab from GCP to an on-premises mini PC?

Please provide a detailed migration procedure covering:

- Docker Compose projects
 - Volumes
 - Environment files
 - Reverse proxy configuration
 - SSL certificates
 - Cloudflare configuration
 - Tailscale configuration
 - Nginx Proxy Manager
 - PostgreSQL
 - MariaDB
-

3. Database Backup and Restore

Does the migration procedure include complete backup and restoration of:

- PostgreSQL databases
- MariaDB databases

including:

- users
- permissions
- schemas
- stored procedures
- indexes
- application data

so that every application continues working exactly as before?

4. Mobile DockerLab

If I travel with the mini PC—for example between:

- Cebu
- Texas
- Florida

can I simply plug it into any Internet connection and have all applications become publicly accessible again without changing DNS records or reconfiguring my services?

5. Internet Performance

My available Internet connections are:

- Florida: **400 Mbps**
- Texas: **90 Mbps**
- Cebu: **250 Mbps**

How would a mobile on-prem DockerLab compare with my current GCP server in terms of:

- application responsiveness
- database performance
- remote access latency
- upload/download speeds
- overall user experience

Under what workloads would I notice a significant difference?

6. Hardware Recommendations

Please recommend the best hardware for a mobile DockerLab.

Compare current Amazon options for:

- Intel NUC
- ASUS NUC
- Beelink
- Minisforum
- Dell OptiPlex Micro
- Lenovo ThinkCentre Tiny
- HP EliteDesk Mini

Include recommendations based on:

- CPU performance
- memory capacity
- NVMe storage
- reliability
- power consumption
- noise level
- Linux compatibility
- virtualization performance
- overall value

Please identify your top recommendations available on Amazon.

7. Intel NUC vs Dell OptiPlex Micro

Would a refurbished Dell OptiPlex Micro Small Form Factor be a practical alternative to an Intel NUC?

Please compare:

- price
 - reliability
 - upgradeability
 - cooling
 - power efficiency
 - portability
 - expected lifespan
 - suitability as a 24/7 DockerLab server
-

8. Hypervisor Selection

I am already familiar with Microsoft Hyper-V.

Should I continue using:

- Hyper-V
- Proxmox VE
- KVM
- Bare-metal Ubuntu with Docker

Please compare each option for:

- Docker workloads
 - backups
 - snapshots
 - disaster recovery
 - networking
 - storage
 - learning curve
 - resource overhead
 - long-term maintainability
-

9. Build a Mobile On-Prem DockerLab

Please design a complete production-grade Mobile On-Prem DockerLab specification, including:

- Recommended hardware
- CPU
- RAM
- NVMe storage layout
- Backup drives
- UPS
- Networking
- Virtualization platform
- Operating system
- Docker architecture
- Security hardening
- Monitoring
- Backup strategy
- Disaster recovery plan
- Cloudflare integration
- Tailscale integration

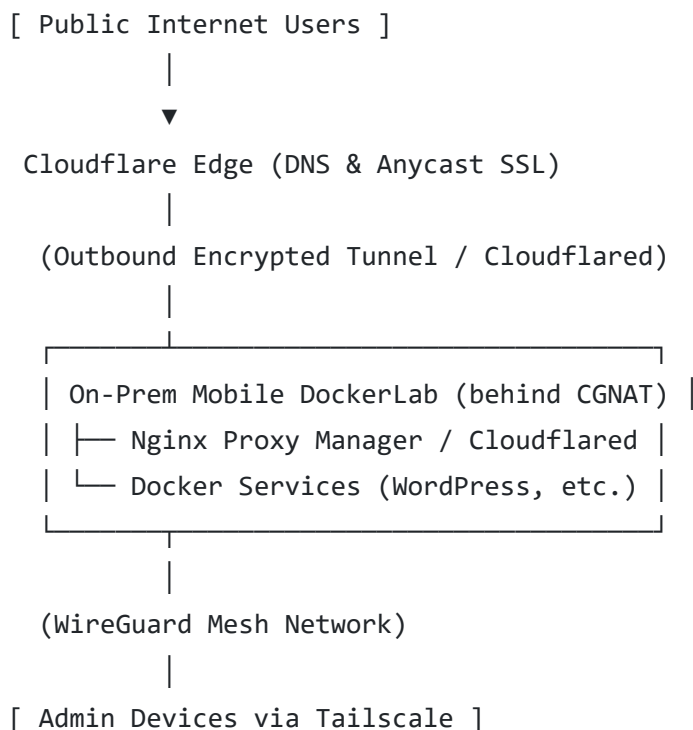
- CI/CD workflow
- GitHub integration
- PostgreSQL and MariaDB backup strategy
- Performance expectations
- Estimated power consumption
- Total hardware cost
- Migration roadmap from GCP

The design should prioritize reliability, portability, ease of maintenance, and low operating costs while supporting my existing Docker-based self-hosted applications.

Migrating your GCP setup (4 vCPUs, 16 GB RAM, 80 GB SSD) to a mobile, on-premises mini PC will give you a **3x–10x boost in raw CPU/database performance**, eliminate your \$50/month (\$600/year) cloud bill, and make your stack completely portable across Cebu, Texas, and Florida.

1. Running Behind CGNAT

Yes, you can run your entire DockerLab behind PLDT's CGNAT using Cloudflare Tunnel, Cloudflare DNS, and Tailscale—with no public IP required.



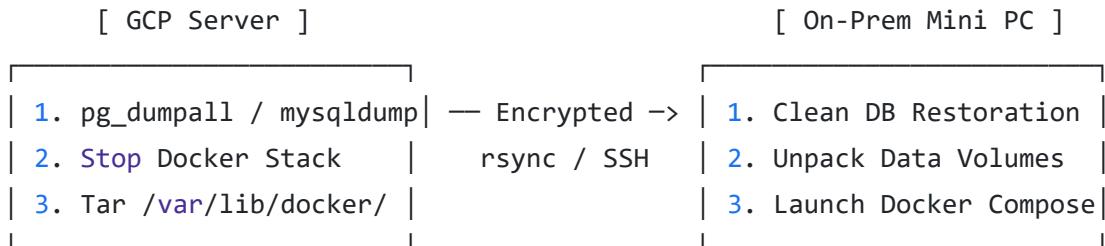
- **Public Web Access (WordPress, Docmost, InvenTree, etc.):** Run the `cloudflared` daemon in Docker. It initiates outbound HTTPS/QUIC connections to Cloudflare's nearest edge nodes. Cloudflare terminates public TLS for `*.gravelcloud.com` and proxies traffic into your local

containers over the encrypted tunnel. This completely bypasses PLDT's CGNAT and incoming port blocks (Ports 80/443).

- **Private Admin Access (Portainer, NPM UI, Database ports, SSH):** Keep these off the public internet entirely. Use **Tailscale** on your mini PC and client devices (laptop, phone, tablet). Tailscale uses WireGuard to establish encrypted, direct peer-to-peer mesh connections across NAT boundaries without opening open firewall ports.
- **Security:** Public-facing web traffic benefits from Cloudflare WAF, DDoS protection, and bot mitigation. Admin services remain invisible to public port scanners.

2 & 3. Migration Procedure & Database Backup/Restore

Migrating from GCP to an on-prem mini PC is straightforward. Because Docker encapsulates your runtime, you only need to sync persistent volume data, configuration files, and database dumps.



Step 1: Export Databases (Full Schema, Data, Users & Permissions)

Run these commands directly on the GCP host to create complete logical backups before stopping services.

PostgreSQL (Full Dump)

`pg_dumpall` captures all databases, schemas, tables, indexes, stored procedures/functions, roles, and global permissions:

Bash

```
docker exec -t postgres-container-name pg_dumpall -U postgres > /tmp/pg_all_databases.sql
```

MariaDB / MySQL (Full Dump)

`mysqldump` with `--all-databases` includes all tables, schemas, stored routines, triggers, user accounts, and privilege tables (`mysql` database):

Bash

```
docker exec -t mariadb-container-name mysqldump -u root -p'YOUR_ROOT_PASSWORD' \  
  --all-databases \  
  --routines \  
  --triggers \  
  --events \  
  --single-transaction > /tmp/mariadb_all_databases.sql
```

Step 2: Backup Files, Volumes, and Configurations

1. Stop Application Containers:

Bash

```
cd /path/to/docker-compose-folder  
docker compose down
```

2. Archive Compose Projects & Persistent Volumes:

Bash

```
tar -czvf /tmp/dockerlab_compose.tar.gz /path/to/docker-compose-folder  
tar -czvf /tmp/docker_volumes.tar.gz /var/lib/docker/volumes
```

Step 3: Transfer Files to On-Prem Mini PC

From your new mini PC, pull the backup archives over Tailscale using `rsync` :

Bash

```
rsync -avzP user@gcp-tailscale-ip:/tmp/*.sql /opt/dockerlab/backups/  
rsync -avzP user@gcp-tailscale-ip:/tmp/*.tar.gz /opt/dockerlab/backups/
```

Step 4: Restore on On-Prem Host

1. Unpack Compose Directories & Volumes:

Bash

```
tar -xzf /opt/dockerlab/backups/dockerlab_compose.tar.gz -C /opt/dockerlab/  
tar -xzf /opt/dockerlab/backups/docker_volumes.tar.gz -C /var/lib/docker/volumes/
```

2. Start Database Containers Only:

Bash

```
cd /opt/dockerlab  
docker compose up -d postgres mariadb
```

3. Restore PostgreSQL Data:

Bash

```
cat /opt/dockerlab/backups/pg_all_databases.sql | docker exec -i postgres-container-  
name psql -U postgres
```

4. Restore MariaDB Data:

Bash

```
cat /opt/dockerlab/backups/mariadb_all_databases.sql | docker exec -i mariadb-  
container-name mysql -u root -p'YOUR_ROOT_PASSWORD'  
# Flush privileges to ensure users & grants load cleanly  
docker exec -it mariadb-container-name mysql -u root -p'YOUR_ROOT_PASSWORD' -e "FLUSH  
PRIVILEGES;"
```

5. Start Remaining Services:

Bash

```
docker compose up -d
```

4. Mobile DockerLab (Traveling Between Locations)

Yes. The mini PC will be 100% plug-and-play across Cebu, Texas, and Florida without requiring any DNS or service reconfigurations.

Because outbound connectivity is maintained via Cloudflare Tunnel and Tailscale:

- **Zero DNS Changes:** Cloudflare edge nodes locate your server using outbound persistent tunnels (`cloudflared`). Whether your mini PC is assigned an IP in Cebu (`192.168.1.x`) or Florida (`10.0.0.x`), `cloudflared` automatically registers its active outbound connection to Cloudflare.
- **Zero Router Configuration:** You do not need to log into local routers, set up port forwarding, or worry about local CGNAT setups in temporary locations.
- **Instant Re-connection:** Within 10–30 seconds of plugging in an Ethernet cable or connecting to Wi-Fi, Cloudflare Tunnels and Tailscale re-establish connectivity. All public traffic routed to `*.gravelcloud.com` and private Tailscale traffic reaches your server seamlessly.

5. Internet Performance Comparison

Comparing your cloud instance to an on-premises setup across your location speeds:

Metric	GCP Cloud Instance (e2-standard-4)	On-Prem Mobile Mini PC (Modern x86)
CPU Performance	Shared/vCPU throttled (2 Core / 4 Thread equivalent)	Dedicated 8-Core / 16-Thread physical processor
Database I/O (Disk)	~1,200–3,000 IOPS (Standard SSD limit)	500,000+ IOPS (PCIe 4.0 NVMe SSD)
Local Access Latency	50 ms – 200 ms (depends on cloud region vs location)	< 1 ms (when connected to local LAN)
Remote Access Latency	20 ms – 150 ms globally	15 ms – 250 ms (bounded by local ISP upload bandwidth)
Upload / Download	1 Gbps / 1 Gbps burst	Florida: 400 Mbps Cebu: 250 Mbps Texas: 90 Mbps

Workload Impacts

- **Database & App Responsiveness (Docmost, OpenProject, WordPress): Significantly Faster on On-Prem.** Standard cloud vCPUs and throttled Cloud SSD storage struggle with complex SQL queries and file indexing. Modern mini PC NVMe drives and high single-core clocks make applications snappy.
- **Large File Uploads/Downloads (InvenTree attachments, OpenEMR documents): Fastest in Florida/Cebu, Slower in Texas.** When accessing remote files outside your local network, your upload speed is capped by the location's upload bandwidth (e.g., Texas at 90 Mbps).
- **Multi-User Concurrent Traffic:** If 50+ public users hit WordPress simultaneously, GCP’s enterprise backbone handles peak bandwidth better. For personal, lab, or small team use, the local hardware

advantages far outweigh cloud bandwidth advantages.

6. Hardware Recommendations

When selecting hardware for a portable server, prioritize high single-core CPU speeds, dual M.2 NVMe storage slots (for OS + RAID/Backups), low power draw under load, and strong Linux support.

Mini PC Form Factor Comparison

| Beelink / Minisforum (Ultra Compact, ~0.8L) |

| - High performance Ryzen 7 / Core Ultra |

| - Max portability & dual NVMe slots |

vs.

| Enterprise Micro PCs (OptiPlex, Tiny, Mini) |

| - Ultra-durable chassis & cooling |

| - Excellent 24/7 reliability & value |

Brand Comparison Matrix

Brand / Model Class	Performance	Portability	24/7 Reliability	Price Range	Top Amazon Option
Beelink (SER8 / EQi12)	Excellent	Extreme (< 0.8L)	High	\$400 - \$600	Beelink SER8 (Ryzen 7 8845HS/8745 HS)
Minisforum (UM880 Pro / UM790)	Excellent	Extreme (< 0.8L)	High	\$450 - \$650	Minisforum UM880 Pro (Ryzen 7 8845HS)
ASUS NUC (NUC 14 Pro)	Moderate-High	Extreme (< 0.6L)	Enterprise Grade	\$600 - \$900	ASUS NUC 14 Pro (Core Ultra 5/7)
Dell OptiPlex Micro	Moderate-High	Compact (~1.2L)	Enterprise Grade	\$250 - \$450 (Refurb)	Dell OptiPlex 7090 Micro

7/28/26, 9:07 AMAI-04498 GCP Dockerlab to NUC OnPrem

(7090/7000)					(Intel i7-10700T/11700T)
Lenovo ThinkCentre Tiny (M90q/M70q)	Moderate-High	Compact (~1.2L)	Enterprise Grade	\$250 - \$450 (Refurb)	Lenovo M90q Tiny Gen 2
HP EliteDesk Mini (800 G6/G8)	Moderate-High	Compact (~1.2L)	Enterprise Grade	\$230 - \$400 (Refurb)	HP EliteDesk 800 G6 Mini

Top 2 Amazon Recommendations

1. Top Performance & Value Choice: Beelink SER8 Mini PC
 - Specs:** AMD Ryzen 7 8845HS (8 Cores / 16 Threads), 32 GB DDR5 RAM, 1 TB PCIe 4.0 NVMe SSD, Dual M.2 PCIe 4.0 slots.
 - Why:** Compact aluminum chassis, low power usage (~15W–45W), quiet vapor-chamber cooling, and an extra M.2 slot for adding a second SSD.
2. Top Reliability & Commercial Choice: ASUS NUC 14 Pro
 - Specs:** Intel Core Ultra 5 125H / Ultra 7 155H, up to 96GB DDR5, Dual M.2 slots, 2.5G LAN.
 - Why:** Continuous 24/7 rating, robust Linux kernel driver support, and compact hardware built specifically for commercial deployments.

7. Intel NUC vs. Refurbished Dell OptiPlex Micro

A refurbished **Dell OptiPlex Micro** (or Lenovo Tiny / HP Mini) is an outstanding, cost-effective alternative to a new Intel/ASUS NUC.

Feature	Intel / ASUS NUC	Dell OptiPlex Micro (Refurb)
Average Price	\$500 - \$800	\$250 - \$380
Reliability	High (Consumer/Prosumer)	Enterprise Grade (24/7 Rated)
CPU Generation	Intel Ultra / AMD Ryzen 8000	Intel 10th / 11th Gen
Memory Support	DDR5 (Up to 64GB/96GB)	DDR4 (Up to 64GB)
Upgradeability	2x M.2 NVMe	1x M.2 NVMe + 1x 2.5" SATA
Power Consumption	10W idle / 45W load	12W idle / 65W load
Footprint	~0.5 Liters	~1.2 Liters

Recommendation

- **Choose Refurbished Dell OptiPlex 7090 Micro if:** You want maximum value for money (\$250–\$350), enterprise thermal design, flexible 2.5" SATA + M.2 drive configuration, and proven 24/7 durability.
- **Choose NUC / Beelink SER8 if:** You want the smallest physical footprint for travel, lower overall power consumption, and peak CPU performance for multi-tenant database workloads.

8. Hypervisor & OS Selection

To determine the best OS base for running Docker containers:

Metric	Bare-Metal Ubuntu + Docker	Proxmox VE (PVE)	KVM / QEMU	Microsoft Hyper-V
Resource Overhead	Lowest (~200MB RAM)	Low (~1GB RAM)	Low (~1GB RAM)	Medium-High (~2GB+ RAM)
Docker Workloads	Direct Native Execution	Run inside LXC or VM	Run inside VM	Run inside Ubuntu VM
Backups & Snapshots	File-level / Docker volume	Block-level VM/LXC Snapshots	Manual QCOW2 snapshots	Hyper-V Checkpoints
Disaster Recovery	Fast (Re-run Ansible/Compose)	Instant (Restore VM Image)	Moderate	Moderate
Networking	Simple Linux Bridge / Host	Advanced vSwitch/VLANs	Manual bridge setup	Virtual Switch setup
Learning Curve	Minimal	Low-Moderate	High (CLI based)	Minimal

Verdict

- **Top Choice for Performance & Simplicity: Bare-Metal Ubuntu Server 24.04 LTS with Docker.** It delivers zero hypervisor overhead, full hardware access, efficient power usage, and quick maintenance.
- **Top Choice for System Snapshots & Isolation: Proxmox VE.** If you want full-system rollbacks before performing system updates or testing new applications, install Proxmox VE, create an Ubuntu Server VM for Docker, and schedule automated nightly ZFS/PBS snapshots.

- **Avoid Hyper-V for Linux On-Prem:** Hyper-V incurs higher overhead, lacks native EXT4 performance features, and complicates networking across travel routers.

9. Comprehensive Mobile On-Prem DockerLab Architecture Specification

ON-PREM MOBILE DOCKERLAB
HARDWARE: Beelink SER8 / Dell OptiPlex Micro (8C/16T, 64GB DDR4/5, 2TB NVMe)
POWER: Mini 60W DC UPS / Power Bank with Pass-Through
OPERATING SYSTEM: Ubuntu Server 24.04 LTS (Bare-Metal)
SECURITY: UFW (Default Deny), Fail2ban, SSH Key Only, Docker Bench
CONTAINER LAYER (Docker Compose & Portainer)
└─ Ingress: Nginx Proxy Manager + Cloudflared (Cloudflare Tunnels)
└─ VPN Access: Tailscale Container / Host Daemon
└─ Web Services: WordPress, Docmost, InvenTree, OpenEMR, OpenProject
└─ Databases: PostgreSQL 16 + MariaDB 10.11
└─ Monitoring & Backups: Prometheus, Grafana, Dozzle, Restic / Offsite Sync

Complete System Blueprint

1. Recommended Hardware Configuration

- **System:** Beelink SER8 (AMD Ryzen 7 8845HS, 8C/16T) or Dell OptiPlex 7090 Micro (i7-10700T).
- **RAM:** 64 GB DDR4/DDR5 SODIMM (allows ample head-room for MySQL, Postgres, and Docker caches).
- **Storage Layout:**
 - **Drive 1 (Primary):** 1 TB PCIe 4.0 NVMe SSD (OS, Docker Containers, Active DBs).
 - **Drive 2 (Internal/External Backup):** 2 TB PCIe NVMe SSD or Samsung T7 Shield USB Drive (Local automated backups).
- **Portable UPS:** TalentCell 12V Uninterruptible Power Supply (UPS) or Anker Power Bank with AC/DC pass-through (~\$40–\$60). Keeps the server online during short power flickers when traveling.
- **Travel Router (Optional but Recommended):** GLiNet GL-MT3000 (Beryl AX) Pocket Travel Router (\$80). Automatically routes the server's Ethernet connection through Wi-Fi hot spots or hotel portals.

2. Virtualization & Operating System

- **OS:** Ubuntu Server 24.04 LTS (Bare-Metal install).
- **Filesystem:** EXT4 or ZFS (if using Proxmox for snapshot capabilities).

3. Security Hardening

- **Firewall (UFW):**

Bash

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow in on tailscale0
sudo ufw enable
```

- **SSH Hardening:** Disable password authentication, allow SSH key logins only, and change the default SSH port.
- **Fail2ban:** Monitor SSH and system logs to block suspicious login attempts automatically.

4. Cloudflare & Tailscale Integration

- **Public Services (ccloudflared):** Configure Cloudflare Tunnel in Docker using ccloudflared . Map public hostnames directly to Nginx Proxy Manager or target service ports:
 - app.gravelcloud.com \$ \rightarrow\$ http://nginx-proxy-manager:80
- **Private Admin Services (Tailscale):** Authenticate Tailscale on the host. Access Portainer (http://100.x.y.z:9000), NPM Admin (http://100.x.y.z:81), and PostgreSQL/MariaDB ports (5432 , 3306) securely over Tailscale IPs.

5. Database Backup Strategy

- **Automated Daily Backups:** Set up offsite-db-backup container or a cron job using pg_dumpall and mysqldump .
- **Pruning & Retention:** Retain 7 daily backups, 4 weekly backups, and 12 monthly backups.
- **Offsite Replication:** Encrypt backups via **Restic** or **BorgBackup** and sync them to backblaze B2, AWS S3, or Google Cloud Storage (\$1–\$2/month storage cost).

6. CI/CD & GitHub Workflow

- Run a **Watchtower** container or **Portainer Webhooks** linked to GitHub Actions.
- When code changes are pushed to your application repositories, GitHub Actions builds and pushes the image to GitHub Packages (GHCR), then calls a secure Portainer webhook over Tailscale to re-deploy containers automatically.

7. Performance, Power, and Cost Summary

FINANCIAL BREAKDOWN & PAYBACK PERIOD	
GCP Cloud Cost:	\$50 / month (\$600 / year)
On-Prem Hardware Cost:	~\$450 - \$650 (One-time investment)
On-Prem Power Cost:	~\$1.50 / month (~15W average)
NET SAVINGS:	~\$580 / year
ROI / PAYBACK PERIOD:	~9 to 12 Months

- **Estimated Power Draw:** 12W–18W idle, 45W maximum load.
- **Monthly Power Cost:** ~\$1.50/month (based on average \$0.15/kWh rates).
- **Total Initial Hardware Cost:** \$450 – \$650 (Hardware + RAM/SSD upgrade + Mini UPS).
- **Break-Even Point:** 9 to 12 months, after which your infrastructure runs virtually free.

10. Step-by-Step Migration Roadmap

Phase 1: Procurement & Local Setup

- └─ Acquire Mini PC, RAM/SSD, Portable UPS
- └─ Install Ubuntu Server 24.04 LTS, Docker, & Tailscale

Phase 2: Network & Tunnel Configuration

- └─ Authenticate Tailscale on Mini PC
- └─ Provision Cloudflare Tunnels (cloudflared) for gravelcloud.com

Phase 3: Data Migration & Testing

- └─ Export DB Dumps (pg_dumpall / mysqldump) & Docker Volumes from GCP
- └─ Transfer & Restore on Mini PC over Tailscale
- └─ Validate All Services via Local Tailscale Access

Phase 4: Cutover & GCP Decommissioning

- └─ Switch Cloudflare DNS / Tunnels to Point to On-Prem Hardware
- └─ Verify Public Connectivity, Stop GCP Instance, & Cancel Subscription

Please generate a complete Docker Compose file and shell backup script using Restic to automatically backup PostgreSQL, MariaDB, and Docker volumes to S3 offsite storage.

Python

```
import os

# Create directory structure for code / scripts
os.makedirs("backup_solution", exist_ok=True)

# 1. Generate docker-compose.yml
docker_compose_content = """version: '3.8'

services:
  # Restic Backup Service (Runs on schedule via cron / entrypoint)
  restic-backup:
    image: restic/restic:latest
    container_name: restic_backup_service
    restart: unless-stopped
    hostname: mobile-dockerlab
    environment:
      - RESTIC_REPOSITORY=${RESTIC_REPOSITORY}
      - RESTIC_PASSWORD=${RESTIC_PASSWORD}
      - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
      - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_HOST=${POSTGRES_HOST:-postgres}
      - MARIADB_ROOT_PASSWORD=${MARIADB_ROOT_PASSWORD}
      - MARIADB_HOST=${MARIADB_HOST:-mariadb}
      - CRON_SCHEDULE=${CRON_SCHEDULE:-0 2 * * *}
      - KEEP_DAILY=${KEEP_DAILY:-7}
      - KEEP_WEEKLY=${KEEP_WEEKLY:-4}
      - KEEP_MONTHLY=${KEEP_MONTHLY:-12}
    volumes:
      - /var/lib/docker/volumes:/docker_volumes:ro
      - /opt/dockerlab/app_data:/app_data:ro
      - ./backup_script.sh:/scripts/backup_script.sh:ro
      - ./dumps:/tmp/dumps
      - /var/run/docker.sock:/var/run/docker.sock:ro
```

```

networks:
  - lab_network
  entrypoint: ["/bin/sh", "-c", "apk add --no-commission --no-cache mariadb-client
postgresql-client curl tzdata docker-cli && chmod +x /scripts/backup_script.sh && crond -f
-l 2"]

```

```

networks:
  lab_network:
    external: true
"""

```

2. Generate backup_script.sh

```

backup_script_content = """#!/bin/sh
set -e

```

```

# =====
# Mobile On-Prem DockerLab Automated Restic S3 Backup Script
# =====

```

```

TIMESTAMP=$(date +"%Y%m%d_%H%M%S")
DUMP_DIR="/tmp/dumps"
LOG_PREFIX="[BACKUP ${TIMESTAMP}]"

```

```

echo "${LOG_PREFIX} Starting backup process..."
mkdir -p "${DUMP_DIR}"

```

```

# --- Step 0: Ensure Restic Repository is Initialized ---
echo "${LOG_PREFIX} Checking Restic repository initialization..."
if ! restic cat config > /dev/null 2>&1; then
    echo "${LOG_PREFIX} Repository not initialized. Initializing now..."
    restic init
else
    echo "${LOG_PREFIX} Repository initialized successfully."
fi

```

```

# --- Step 1: PostgreSQL Database Dump ---
echo "${LOG_PREFIX} Dumping PostgreSQL databases..."
PG_DUMP_FILE="${DUMP_DIR}/postgres_all_${TIMESTAMP}.sql.gz"
export PGPASSWORD="${POSTGRES_PASSWORD}"

```

```

if pg_dumpall -h "${POSTGRES_HOST}" -U "${POSTGRES_USER}" | gzip > "${PG_DUMP_FILE}"; then
    echo "${LOG_PREFIX} PostgreSQL dump succeeded: ${PG_DUMP_FILE}"

```

```
else
    echo "${LOG_PREFIX} ERROR: PostgreSQL dump failed!" exit 1
fi

# --- Step 2: MariaDB / MySQL Database Dump ---
echo "${LOG_PREFIX} Dumping MariaDB databases..."
MARIADB_DUMP_FILE="${DUMP_DIR}/mariadb_all_${TIMESTAMP}.sql.gz"

if mysqldump -h "${MARIADB_HOST}" -u root -p"${MARIADB_ROOT_PASSWORD}" \
    --all-databases \
    --routines \
    --triggers \
    --events \
    --single-transaction | gzip > "${MARIADB_DUMP_FILE}"; then
    echo "${LOG_PREFIX} MariaDB dump succeeded: ${MARIADB_DUMP_FILE}"
else
    echo "${LOG_PREFIX} ERROR: MariaDB dump failed!" exit 1
fi

# --- Step 3: Run Restic Backup ---
echo "${LOG_PREFIX} Performing Restic snapshot to S3 storage..."
restic backup \
    --tag "dockerlab-daily" \
    --tag "hostname:mobile-dockerlab" \
    "${DUMP_DIR}" \
    /docker_volumes \
    /app_data

echo "${LOG_PREFIX} Restic snapshot complete."

# --- Step 4: Cleanup Temporary Database Dumps ---
echo "${LOG_PREFIX} Cleaning up local temporary dump files..."
rm -rf "${DUMP_DIR}"/.*

# --- Step 5: Apply Retention Policy (Prune Old Snapshots) ---
echo "${LOG_PREFIX} Applying retention policy (Keep Daily: ${KEEP_DAILY}, Weekly:
${KEEP_WEEKLY}, Monthly: ${KEEP_MONTHLY})..."
restic forget \
    --keep-daily "${KEEP_DAILY}" \
    --keep-weekly "${KEEP_WEEKLY}" \
    --keep-monthly "${KEEP_MONTHLY}" \
    --prune
```

```
# --- Step 6: Verify Repository Integrity ---
echo "${LOG_PREFIX} Running restic check to ensure backup integrity..."
restic check --read-data-subset=1%

echo "${LOG_PREFIX} Backup and maintenance completed successfully at $(date)!"
""

# 3. Generate env template file (.env.example)
env_example_content = """"# Restic S3 Backup Configuration
RESTIC_REPOSITORY=s3:https://s3.us-west-004.backblazeb2.com/your-bucket-name
RESTIC_PASSWORD=YourSuperSecureResticRepositoryPasswordHere

# AWS / S3 Credentials (Works for AWS S3, Backblaze B2, Cloudflare R2, MinIO, Wasabi)
AWS_ACCESS_KEY_ID=your_access_key_id
AWS_SECRET_ACCESS_KEY=your_secret_access_key

# Database Credentials
POSTGRES_USER=postgres
POSTGRES_PASSWORD=your_postgres_password
POSTGRES_HOST=postgres

MARIADB_ROOT_PASSWORD=your_mariadb_root_password
MARIADB_HOST=mariadb

# Cron & Retention Schedule
CRON_SCHEDULE=0 2 * * *
KEEP_DAILY=7
KEEP_WEEKLY=4
KEEP_MONTHLY=12
""""

# 4. Generate README / Restore Guide
restore_guide_content = """"# Mobile On-Prem DockerLab Backup & Disaster Recovery Guide

## Overview

This automated backup solution uses Restic and S3-compatible storage (Backblaze B2, AWS S3, Cloudflare R2, Wasabi, or MinIO) to back up:

1. Complete PostgreSQL logical database dumps (`pg_dumpall`).
2. Complete MariaDB logical database dumps (`mysqldump` with all routines/triggers/grants).
3. All persistent Docker volumes (`/var/lib/docker/volumes`).
```

4. Custom application data directories (`/opt/dockerlab/app\_data`).

1. Quick Setup Instructions

1. ****Copy Environment File:****

```
```bash
cp .env.example .env
```

#### 2. **Edit .env with your actual credentials:**

- Set `RESTIC_REPOSITORY` to your S3 bucket endpoint (e.g., `s3:https://s3.us-west-004.backblazeb2.com/my-bucket` ).
- Set strong `RESTIC_PASSWORD` (Store this safely! Without it, backups cannot be restored).
- Fill in your S3 credentials ( `AWS_ACCESS_KEY_ID` / `AWS_SECRET_ACCESS_KEY` ).
- Fill in database credentials matching your running PostgreSQL and MariaDB containers.

#### 3. **Ensure Network Connectivity:**

Make sure your database containers are attached to the Docker network specified in `docker-compose.yml` ( `lab_network` ). If using a different network, run:

Bash

```
docker network create lab_network
```

#### 4. **Launch Backup Service:**

Bash

```
docker compose up -d
```

#### 5. **Run Manual Test Backup:**

Bash

```
docker exec -it restic_backup_service /scripts/backup_script.sh
```

## 2. Restoring Data in Case of Disaster / Migration

## A. List Available Snapshots

Bash

```
docker exec -it restic_backup_service restic snapshots
```

## B. Restore Database Dumps & Volumes from a Snapshot

To restore a specific snapshot (replace <SNAPSHOT\_ID> with the actual ID):

Bash

```
1. Restore snapshot contents to a restore folder
```

```
docker exec -it restic_backup_service restic restore <SNAPSHOT_ID> --target /tmp/restore
```

```
2. Restore PostgreSQL Database
```

```
docker exec -i postgres-container psql -U postgres <
/tmp/restore/tmp/dumps/postgres_all_XXXXXXXX.sql
```

```
3. Restore MariaDB Database
```

```
docker exec -i mariadb-container mysql -u root -p'YOUR_ROOT_PASSWORD' <
/tmp/restore/tmp/dumps/mariadb_all_XXXXXXXX.sql
```

```
docker exec -i mariadb-container mysql -u root -p'YOUR_ROOT_PASSWORD' -e "FLUSH
PRIVILEGES;"
```

```
4. Restore Docker Volumes / Data
```

```
cp -r /tmp/restore/docker_volumes/* /var/lib/docker/volumes/
```

## C. Mount Restic Backup as a Local Folder (Interactive File Recovery)

You can browse all past snapshots like regular folders:

Bash

```
docker exec -it restic_backup_service restic mount /mnt/restic
```

.....

## Write files

```
with open("backup_solution/docker-compose.yml", "w") as f:
 f.write(docker_compose_content)

with open("backup_solution/backup_script.sh", "w") as f:
 f.write(backup_script_content)

with open("backup_solution/env.example", "w") as f:
 f.write(env_example_content)

with open("backup_solution/README.md", "w") as f:
 f.write(restore_guide_content)
```

## Save a bundled zip archive

```
import zipfile

with zipfile.ZipFile("dockerlab_restic_s3_backup.zip", "w") as zipf:
 zipf.write("backup_solution/docker-compose.yml", arcname="docker-compose.yml")
 zipf.write("backup_solution/backup_script.sh", arcname="backup_script.sh")
 zipf.write("backup_solution/env.example", arcname=".env.example")
 zipf.write("backup_solution/README.md", arcname="README.md")

print("Backup solution generated successfully!")
```

```
```text?code_stdout&code_event_index=1
Backup solution generated successfully!
```

Your automated backup solution using **Restic** and **S3-compatible cloud storage** is ready.



dockerlab_restic_s3_backup

ZIP

Open

File Structure Overview

1. `docker-compose.yml` : Restic backup service runner configured with automated cron scheduling and host volume mounts.
2. `backup_script.sh` : Production-grade Shell script executing sequential PostgreSQL (`pg_dumpall`), MariaDB (`mysqldump`), Docker Volume snapshots, retention pruning (`7 daily / 4 weekly / 12 monthly`), and data integrity checks.
3. `.env.example` : Environment template for S3 credentials, DB passwords, and retention policies.
4. `README.md` : Step-by-step installation and disaster recovery / restoration guide.

1. `docker-compose.yml`



backup_solution/docker-compose

YML

Open

YAML

```
version: '3.8'

services:
  restic-backup:
    image: restic/restic:latest
    container_name: restic_backup_service
    restart: unless-stopped
    hostname: mobile-dockerlab
    environment:
      - RESTIC_REPOSITORY=${RESTIC_REPOSITORY}
      - RESTIC_PASSWORD=${RESTIC_PASSWORD}
      - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
      - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_HOST=${POSTGRES_HOST:-postgres}
      - MARIADB_ROOT_PASSWORD=${MARIADB_ROOT_PASSWORD}
      - MARIADB_HOST=${MARIADB_HOST:-mariadb}
      - CRON_SCHEDULE=${CRON_SCHEDULE:-0 2 * * *}
```

```

- KEEP_DAILY=${KEEP_DAILY:-7}
- KEEP_WEEKLY=${KEEP_WEEKLY:-4}
- KEEP_MONTHLY=${KEEP_MONTHLY:-12}
volumes:
- /var/lib/docker/volumes:/docker_volumes:ro
- /opt/dockerlab/app_data:/app_data:ro
- ./backup_script.sh:/scripts/backup_script.sh:ro
- ./dumps:/tmp/dumps
- /var/run/docker.sock:/var/run/docker.sock:ro
networks:
- lab_network
entrypoint: ["/bin/sh", "-c", "apk add --no-cache mariadb-client postgresql-client
curl tzdata docker-cli && chmod +x /scripts/backup_script.sh && crond -f -l 2"]

networks:
lab_network:
external: true

```

2. Backup Script (backup_script.sh)



backup_solution/backup_script

SH

Open

Bash

```

#!/bin/sh
set -e

# =====
# Mobile On-Prem DockerLab Automated Restic S3 Backup Script
# =====

TIMESTAMP=$(date +%Y%m%d_%H%M%S)
DUMP_DIR="/tmp/dumps"
LOG_PREFIX="[BACKUP ${TIMESTAMP}]"

echo "${LOG_PREFIX} Starting backup process..."

```

```
mkdir -p "${DUMP_DIR}"

# --- Step 0: Ensure Restic Repository is Initialized ---
if ! restic cat config > /dev/null 2>&1; then
    echo "${LOG_PREFIX} Repository not initialized. Initializing now..."
    restic init
else
    echo "${LOG_PREFIX} Repository initialized successfully."
fi

# --- Step 1: PostgreSQL Database Dump ---
echo "${LOG_PREFIX} Dumping PostgreSQL databases..."
PG_DUMP_FILE="${DUMP_DIR}/postgres_all_${TIMESTAMP}.sql.gz"
export PGPASSWORD="${POSTGRES_PASSWORD}"

if pg_dumpall -h "${POSTGRES_HOST}" -U "${POSTGRES_USER}" | gzip > "${PG_DUMP_FILE}"; then
    echo "${LOG_PREFIX} PostgreSQL dump succeeded: ${PG_DUMP_FILE}"
else
    echo "${LOG_PREFIX} ERROR: PostgreSQL dump failed!"; exit 1
fi

# --- Step 2: MariaDB / MySQL Database Dump ---
echo "${LOG_PREFIX} Dumping MariaDB databases..."
MARIADB_DUMP_FILE="${DUMP_DIR}/mariadb_all_${TIMESTAMP}.sql.gz"

if mysqldump -h "${MARIADB_HOST}" -u root -p"${MARIADB_ROOT_PASSWORD}" \
    --all-databases \
    --routines \
    --triggers \
    --events \
    --single-transaction | gzip > "${MARIADB_DUMP_FILE}"; then
    echo "${LOG_PREFIX} MariaDB dump succeeded: ${MARIADB_DUMP_FILE}"
else
    echo "${LOG_PREFIX} ERROR: MariaDB dump failed!"; exit 1
fi

# --- Step 3: Run Restic Backup ---
echo "${LOG_PREFIX} Performing Restic snapshot to S3 storage..."
restic backup \
    --tag "dockerlab-daily" \
    --tag "hostname:mobile-dockerlab" \
    "${DUMP_DIR}" \
```

```
/docker_volumes \  
/app_data  
  
echo "${LOG_PREFIX} Restic snapshot complete."  
  
# --- Step 4: Cleanup Temporary Database Dumps ---  
echo "${LOG_PREFIX} Cleaning up local temporary dump files..."  
rm -rf "${DUMP_DIR}"/*  
  
# --- Step 5: Apply Retention Policy (Prune Old Snapshots) ---  
echo "${LOG_PREFIX} Applying retention policy (Keep Daily: ${KEEP_DAILY}, Weekly:  
${KEEP_WEEKLY}, Monthly: ${KEEP_MONTHLY})..."  
restic forget \  
    --keep-daily "${KEEP_DAILY}" \  
    --keep-weekly "${KEEP_WEEKLY}" \  
    --keep-monthly "${KEEP_MONTHLY}" \  
    --prune  
  
# --- Step 6: Verify Repository Integrity ---  
echo "${LOG_PREFIX} Running restic check to ensure backup integrity..."  
restic check --read-data-subset=1%  
  
echo "${LOG_PREFIX} Backup and maintenance completed successfully at $(date)!"
```

3. Environment Template (.env.example)

Code snippet

```
# Restic S3 Backup Configuration  
# S3 Compatibility: Backblaze B2, AWS S3, Cloudflare R2, MinIO, Wasabi  
RESTIC_REPOSITORY=s3:https://s3.us-west-004.backblazeb2.com/your-bucket-name  
RESTIC_PASSWORD=YourSuperSecureResticRepositoryPasswordHere  
  
# S3 Access Credentials  
AWS_ACCESS_KEY_ID=your_access_key_id  
AWS_SECRET_ACCESS_KEY=your_secret_access_key  
  
# Database Host & Authentication  
POSTGRES_USER=postgres  
POSTGRES_PASSWORD=your_postgres_password
```

```
POSTGRES_HOST=postgres
```

```
MARIADB_ROOT_PASSWORD=your_mariadb_root_password
```

```
MARIADB_HOST=mariadb
```

```
# Cron & Retention Schedule (Default: Daily at 2:00 AM)
```

```
CRON_SCHEDULE=0 2 * * *
```

```
KEEP_DAILY=7
```

```
KEEP_WEEKLY=4
```

```
KEEP_MONTHLY=12
```

4. Disaster Recovery Instructions (README.md)



backup_solution/README

MD

Open

A. List Available Backups

To list all available deduplicated snapshots stored in S3:

Bash

```
docker exec -it restic_backup_service restic snapshots
```

B. Restore Snapshot Data

Replace <SNAPSHOT_ID> with your snapshot ID from the previous command:

Bash

```
# 1. Restore files from S3 snapshot to temporary restore folder
```

```
docker exec -it restic_backup_service restic restore <SNAPSHOT_ID> --target /tmp/restore
```

```
# 2. Restore PostgreSQL (Database, Users, Roles, Functions, Schemas)
```

```
docker exec -i postgres-container psql -U postgres <  
/tmp/restore/tmp/dumps/postgres_all_XXXXXXXXX.sql
```

```
# 3. Restore MariaDB / MySQL (Databases, Users, Grants, Triggers)
```

```
docker exec -i mariadb-container mysql -u root -p 'YOUR_ROOT_PASSWORD' <
/tmp/restore/tmp/dumps/mariadb_all_XXXXXXXXX.sql
docker exec -i mariadb-container mysql -u root -p 'YOUR_ROOT_PASSWORD' -e "FLUSH
PRIVILEGES;"
```

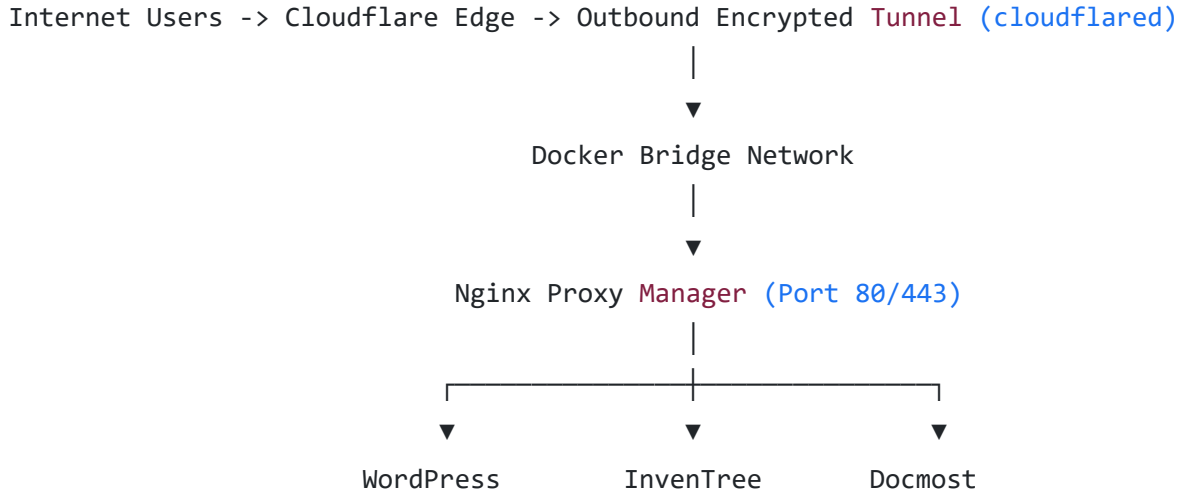
4. Restore Persistent Volumes

```
cp -a /tmp/restore/docker_volumes/* /var/lib/docker/volumes/
```

Can you provide a step-by-step guide to installing and configuring cloudflared via Docker Compose alongside Nginx Proxy Manager for my gravelcloud.com domain?

Exposing your Docker lab through a combined **Cloudflare Tunnel** (`cloudflared`) and **Nginx Proxy Manager (NPM)** architecture allows Cloudflare to handle inbound public web traffic without opening any firewall ports on your home internet, while NPM acts as your local routing manager.

Architecture Overview



- **Cloudflare Tunnel (cloudflared)**: Opens a secure outbound-only connection to Cloudflare.
- **Nginx Proxy Manager**: Receives traffic passed from the tunnel container, inspects the requested host header (`*.gravelcloud.com`), applies rules, and forwards requests to local Docker containers.

Step 1: Create a Dedicated Docker Bridge Network

2. Create docker-compose.yml

YAML

```
version: '3.8'

services:
  # Cloudflare Tunnel Container
  cloudflared:
    image: cloudflare/cloudflared:latest
    container_name: cloudflared
    restart: unless-stopped
    command: tunnel --no-autoupdate run
    environment:
      - TUNNEL_TOKEN=${TUNNEL_TOKEN}
    networks:
      - proxy_network

  # Nginx Proxy Manager Core Service
  nginx-proxy-manager:
    image: 'jc21/nginx-proxy-manager:latest'
    container_name: nginx-proxy-manager
    restart: unless-stopped
    ports:
      - '80:80'      # Public HTTP (handled via tunnel)
      - '81:81'      # NPM Web Admin Console (access via Tailscale/Local LAN)
      - '443:443'    # Public HTTPS
    environment:
      DB_MYSQL_HOST: "npm-db"
      DB_MYSQL_PORT: 3306
      DB_MYSQL_USER: "npm"
      DB_MYSQL_PASSWORD: "${MYSQL_PASSWORD}"
      DB_MYSQL_NAME: "npm"
    volumes:
      - ./npm_data:/data
      - ./npm_letsencrypt:/etc/letsencrypt
    depends_on:
      - npm-db
    networks:
      - proxy_network
```

Database for Nginx Proxy Manager

```

npm-db:
  image: 'mariadb:10.11'
  container_name: npm-db
  restart: unless-stopped
  environment:
    MYSQL_ROOT_PASSWORD: "${MYSQL_ROOT_PASSWORD}"
    MYSQL_DATABASE: 'npm'
    MYSQL_USER: 'npm'
    MYSQL_PASSWORD: "${MYSQL_PASSWORD}"
  volumes:
    - ./npm_db_data:/var/lib/mysql
  networks:
    - proxy_network

networks:
  proxy_network:
    external: true

```

3. Start the Proxy Stack

Bash

```
docker compose up -d
```

Step 4: Route Cloudflare Tunnel Traffic to NPM

1. Return to the **Cloudflare Zero Trust** dashboard where you created your tunnel.
2. In the **Public Hostnames** section, configure two public routes:

Route 1: Apex Domain (gravelcloud.com)

- **Subdomain:** *(Leave blank)*
- **Domain:** gravelcloud.com
- **Path:** *(Leave blank)*
- **Type:** HTTP
- **URL:** nginx-proxy-manager:80

Route 2: Wildcard Subdomains (*.gravelcloud.com)

- **Subdomain:** *
- **Domain:** gravelcloud.com

- **Path:** *(Leave blank)*
- **Type:** HTTP
- **URL:** nginx-proxy-manager:80

How this works: Cloudflare automatically creates CNAME DNS records pointing `gravelcloud.com` and `*.gravelcloud.com` to your encrypted tunnel ID. All web traffic hits Cloudflare, enters your local `cloudflared` container, and gets passed to NPM at port 80 via Docker's internal DNS (`nginx-proxy-manager`).

Step 5: Configure Nginx Proxy Manager

1. Open your browser and go to your NPM admin panel: `http://<YOUR_SERVER_IP>:81` (or via your Tailscale IP).
2. Log in with the default credentials:
 - **Username:** `admin@example.com`
 - **Password:** `changeme`
3. Promptly change your email and administrator password when requested.

Step 6: Connect Applications to NPM

When deploying self-hosted applications (e.g., WordPress, InvenTree, Docmost), attach their containers to the external `proxy_network` in their respective `docker-compose.yml` files.

Example WordPress Setup (`docker-compose.yml`):

YAML

```
version: '3.8'

services:
  wordpress:
    image: wordpress:latest
    container_name: wordpress-app
    restart: unless-stopped
    # NOTICE: Do NOT expose host ports (e.g., ports: - "8080:80").
    # NPM accesses the container directly through the shared proxy_network.
    environment:
      WORDPRESS_DB_HOST: wordpress-db
      WORDPRESS_DB_PASSWORD: wordpress_password
    networks:
      - proxy_network
      - internal_wp_net
```

```
wordpress-db:
  image: mariadb:10.11
  container_name: wordpress-db
  restart: unless-stopped
  environment:
    MYSQL_ROOT_PASSWORD: wordpress_password
    MYSQL_DATABASE: wordpress
  networks:
    - internal_wp_net

networks:
  proxy_network:
    external: true
  internal_wp_net:
    driver: bridge
```

Exposing WordPress in NPM Admin UI:

1. In NPM, navigate to **Hosts** $\rightarrow$ **Proxy Hosts** $\rightarrow$ **Add Proxy Host**.
2. **Domain Names:** wp.gravelcloud.com
3. **Scheme:** http
4. **Forward Hostname / IP:** wordpress-app *(Use the exact Docker container name)*
5. **Forward Port:** 80
6. Check **Block Common Exploits** and **Websockets Support** if required by the application.
7. Click **Save**.

Step 7: (Optional) Set Real User IP Headers in NPM

Because traffic passes through Cloudflare and `cloudflared`, NPM sees all incoming connections as originating from the `cloudflared` container's internal Docker IP.

To ensure logs in applications like WordPress or Docmost reflect true client IP addresses:

1. Create a custom configuration file on your host at `./npm_data/nginx/custom/http_top.conf` :
Nginx

```
# Restore Real Client IPs behind Cloudflare Tunnel
set_real_ip_from 172.16.0.0/12; # Standard Docker network range
set_real_ip_from 10.0.0.0/8;
set_real_ip_from 192.168.0.0/16;
```

```
real_ip_header CF-Connecting-IP;
```

2. Restart Nginx Proxy Manager:

Bash

```
docker compose restart nginx-proxy-manager
```

Your setup is now fully operational behind CGNAT with end-to-end routing.