AI-04678 GCP Committed Use Discount CUD vs NUC

Multiple

Calculate/estimate my dockerlab monthly expenses using the normal rate vs 1 year CUD and 3 Year CUD. I have the following GCP Free Tier setup and soon to expire on September 8, 2026.

e2-standard-4 (4vcpu, 16GB RAM and 80GB drive)

us-west-c

Yes. Based on your **GCP DockerLab** configuration:

- **VM:** `e2-standard-4`
- **4 vCPU / 16 GiB RAM**
- **Region:** `us-west1` (Oregon)
- **Disk:** 80 GB
- Assumption: VM runs **24/7**
- Linux OS
- No GPU
- No unusual network egress
- Free Trial credits expire **September 8, 2026**

Google's current pricing shows `e2-standard-4` at about **$0.134/hour** in `us-west1`. Current resource-based CUD pricing is approximately **$0.0844/hr for 1-year** and **$0.0603/hr for 3-year**. (GCloud Compute)

1. Compute cost

Using Google's standard 730-hour billing-month convention:

Pricing	Hourly	Monthly	Annual
Normal / on-demand	$0.1340	**$97.82**	**$1,173.84**
1-Year CUD	$0.0844	**$61.61**	**$739.32**
3-Year CUD	$0.0603	**$44.02**	**$528.24**

So purely on compute, the 1-year CUD saves about **$36/month**, while the 3-year saves about **$54/month** versus on-demand. The published e2-standard-4 pricing independently shows approximately $97.84/month, $61.64/month and $44.03/month. (GCloud Compute)

2. Your 80-GB disk

I'm assuming your 80-GB DockerLab boot disk is **Balanced Persistent Disk**.

Current `us-west1` balanced PD pricing is **$0.000136986/GiB-hour**, which works out to approximately:

80 × $0.000136986 × 730 = $8.00/month. (Google Cloud)

Importantly, **CUD does not discount your Persistent Disk**. The CUD applies to eligible compute resources, not ordinary persistent-disk capacity.

3. External IPv4

If your VM has a normal external IPv4 address, Google currently charges **$0.005/hour**, or approximately **$3.65/month** for a continuously running VM. (Google Cloud)

Your realistic DockerLab monthly bill

Assuming:

e2-standard-4 + 80 GB Balanced PD + one external IPv4 + 24/7 operation

DockerLab component	Normal	1-Year CUD	3-Year CUD
e2-standard-4	$97.82	$61.61	$44.02
80 GB Balanced PD	$8.00	$8.00	$8.00
External IPv4	$3.65	$3.65	$3.65
Estimated total/month	**$109.47**	**$73.26**	**$55.67**
Estimated total/year	**$1,313.64**	**$879.12**	**$668.04**

What the CUD actually saves you

	1-Year CUD	3-Year CUD
Monthly savings vs normal	**$36.21**	**$53.80**
Annual savings	**$434.52**	**$645.60**
Compute savings	**37%**	**55%**
Disk/IP savings	0%	0%

Google confirms that resource-based CUDs can provide up to **37% off for 1-year** and **55% off for 3-year** for eligible non-memory-optimized compute, and commitments are billed monthly even if you underutilize the committed resources. (Google Cloud)

But there's an important catch for you

Your **$300 Free Trial ending September 8** is particularly relevant.

Google states that **projects still in the free-tier/trial period or without payment history don't qualify for resource-based CUDs**. (Google Cloud Documentation)

So I would **not purchase the CUD until the project is fully converted to normal paid billing and Google makes the commitment available**.

Also, a CUD is a **commitment**, not a reservation. It doesn't automatically guarantee VM capacity in a particular zone. (Google Cloud)

My recommendation for your DockerLab

Given what you've told me previously about your DockerLab—Nginx Proxy Manager, Portainer, PostgreSQL, MariaDB, OpenProject, OpenEMR, InvenTree, Zammad, Docmost, WordPress, etc.—this is exactly the kind of workload where a CUD makes sense **if you're going to keep the GCP DockerLab running continuously**.

I'd rank the choices:

1. 🥇 3-Year CUD — ~$56/month total

Best economics if you are confident GCP remains your DockerLab home for 3+ years.

2. 🥈 1-Year CUD — ~$73/month total

Probably the best balance between savings and flexibility.

3. 🥉 On-demand — ~$109/month total

Best only if you're considering migrating the DockerLab to your NUC/on-premise fairly soon.

And there's an interesting fourth option for you: **move the DockerLab to your planned on-prem NUC and use Cloudflare Tunnel/Tailscale**, eliminating most of the recurring compute expense altogether.

One thing I'd check before you commit: whether your actual 80-GB disk is **Balanced, SSD, or Standard Persistent Disk**, and whether your VM has a **static external IP**. Those two details will let me turn this from an estimate into a much tighter projection of your actual post–September 8 bill.

DockerLab monthly cost by pricing model

Estimated recurring cost for e2-standard-4 + 80 GB balanced persistent disk + one in-use external IPv4, running 24/7.

cost	model
109.47	On-demand
73.26	1-year CUD
55.67	3-year CUD

How to move my entire GCP Dockerlab to my on-prem NUC. Create a clear step by step instruction. Can I copy the entire GCP volume and place it in my 1TB One Drive or 2TB Google Drive. I really don't want to commit to the 3 year CUD. If the answer is no, what are my best options to keep my data. I'm already heavily using Docmost and have live and important data in it.

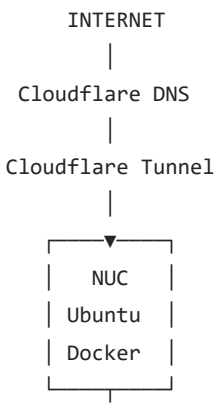
Yes — and **I agree with your instinct not to lock yourself into a 3-year CUD yet**.

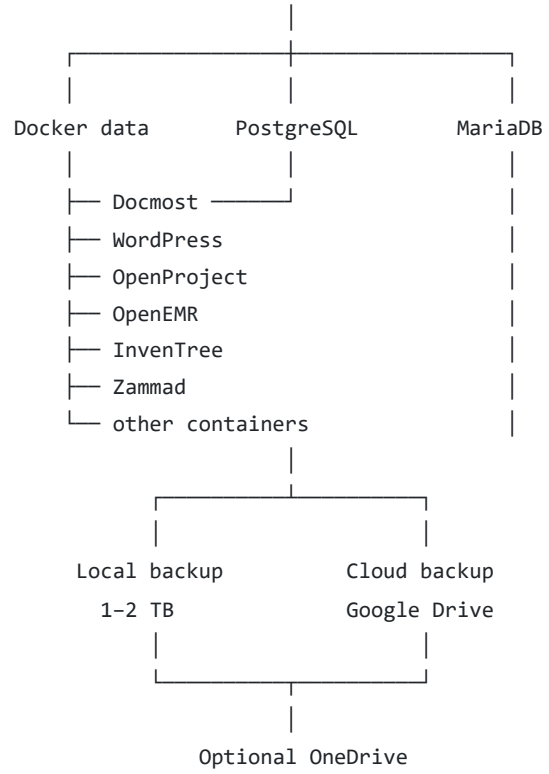
For your situation, I would **not try to "move the GCP volume" directly into OneDrive or Google Drive**. Instead, I would migrate the *DockerLab at the application/data level*, with a separate backup strategy. That gives you a much safer migration and avoids tying your future to GCP.

Google's own documentation supports taking Persistent Disk snapshots while the VM is running, but importantly, a snapshot is a **Google Cloud disk backup**, not a portable DockerLab image that you can simply put into Google Drive or OneDrive and boot on a NUC. (Google Cloud Documentation)

My recommended architecture

I would make your NUC the new primary DockerLab:





The key idea is:

> **Don't make Google Drive your live Docker filesystem. Make it your backup.**

Part 1 — Can you copy the entire GCP volume to Google Drive?

Technically: yes, as a backup/archive.

Practically: don't use it as your migration method.

Your 80-GB GCP Persistent Disk can be snapshotted. Google supports restoring those snapshots into new Persistent Disks. (Google Cloud Documentation)

But this:

```
GCP Persistent Disk
       ↓
Google Snapshot
       ↓
Google Drive
       ↓
NUC
```

is **not** a clean native migration path.

A GCP snapshot is designed for Google Cloud infrastructure. It isn't equivalent to:

```
dockerlab.img
```

that you can just mount on Ubuntu.

You *could* create an image/export and eventually download it, but you're turning a relatively straightforward Docker migration into a disk-image migration problem.

And there's another important issue:

Your Docker data is more valuable than the VM itself.

Your NUC doesn't need to reproduce:

```
e2-standard-4
80 GB boot disk
GCP networking
GCP VM configuration
```

It needs to reproduce:

```
Docker
Docker Compose
containers
images
volumes
PostgreSQL databases
MariaDB databases
configuration
secrets
SSL
DNS/proxy configuration
application data
```

That's much easier.

Part 2 — The most important thing: protect Docmost first

Because you said:

> "I'm already heavily using Docmost and have live and important data in it."

Do not start by destroying or shutting down the GCP VM.

We're going to build the NUC **beside** the existing DockerLab.

Think of it as:

```
GCP DockerLab
      │
      │  COPY
      ▼
   NUC DockerLab
      │
      │  VERIFY
      ▼
```

```
      NUC works
         │
         │  CUTOVER
         ▼
   NUC becomes primary
         │
         ▼
   GCP becomes temporary backup
         │
         ▼
   GCP eventually deleted
```

This is much safer.

Part 3 — Step 0: Freeze your GCP environment

Before doing anything major:

Do NOT delete the GCP VM.

Keep it running.

You have until September 8, 2026 before your free credits expire, so use that window as your migration insurance.

Your first objective is:

> **Create a complete recoverable backup of Docmost and your other important Docker applications.**

Part 4 — Inventory your current DockerLab

SSH into the GCP VM.

Run:

```
docker ps
```

Then:

```
docker ps -a
```

Then:

```
docker volume ls
```

Then:

```
docker network ls
```

And:

```
docker images
```

Also:

```
df -h
```

This tells us exactly what you're running.

I would also save everything:

```
mkdir -p ~/dockerlab-migration

docker ps -a > ~/dockerlab-migration/docker-containers.txt
docker volume ls > ~/dockerlab-migration/docker-volumes.txt
docker network ls > ~/dockerlab-migration/docker-networks.txt
docker images > ~/dockerlab-migration/docker-images.txt
df -h > ~/dockerlab-migration/disk-usage.txt
```

Part 5 — Find your Docker Compose files

This is extremely important.

Your DockerLab should ideally be reproducible from:

```
docker-compose.yml
.env
configuration files
volumes
```

rather than from an entire VM disk.

Find Compose files:

```
find ~ /opt /srv -name "docker-compose.yml" -o -name "compose.yml" 2>/dev/null
```

Also:

```
find ~ /opt /srv -name ".env" 2>/dev/null
```

You may discover something like:

```
/opt/dockerlab/docker-compose.yml
/opt/dockerlab/.env
```

If that's how you built your DockerLab, **excellent**.

That's essentially your infrastructure blueprint.

Part 6 — Determine where Docmost actually stores its data

This is the most important technical step.

Run:

```
docker inspect docmost
```

Replace `docmost` with your actual container name.

Look for:

```
Mounts
```

You'll probably see something similar to:

```
/var/lib/docker/volumes/docmost_data/_data
```

or:

```
/opt/dockerlab/docmost
```

We need to identify:

Docmost application data

and

Docmost database

Docmost commonly depends on PostgreSQL, so you need to preserve **both** the application's persistent files and its database.

Do not simply copy a live PostgreSQL directory while PostgreSQL is running.

Instead, make a proper PostgreSQL dump.

Part 7 — Backup PostgreSQL properly

First identify the PostgreSQL container:

```
docker ps | grep -i postgres
```

Then:

```
docker exec -it POSTGRES_CONTAINER \
pg_dumpall -U postgres > ~/dockerlab-migration/postgres-all.sql
```

For a large installation, I prefer individual database dumps as well.

Find databases:

```
docker exec -it POSTGRES_CONTAINER \
psql -U postgres -c "\l"
```

Then dump the relevant database:

```
docker exec POSTGRES_CONTAINER \
pg_dump -U postgres -Fc DATABASE_NAME \
> ~/dockerlab-migration/DATABASE_NAME.dump
```

This gives you a **portable database backup**.

That's much better than trying to transplant PostgreSQL's raw filesystem.

Part 8 — Backup MariaDB/MySQL

Find it:

```
docker ps | grep -Ei 'maria|mysql'
```

Then:

```
docker exec MYSQL_CONTAINER \
mysqldump -u root -p --all-databases \
> ~/dockerlab-migration/mariadb-all.sql
```

Now you have logical database backups.

Part 9 — Backup your Docker volumes

Now we deal with the actual application files.

List them:

```
docker volume ls
```

For each important volume, you can create a compressed archive.

For example:

```
docker run --rm \
  -v docmost_data:/data \
  -v ~/dockerlab-migration:/backup \
  ubuntu \
  tar czf /backup/docmost_data.tar.gz -C /data .
```

Repeat for important volumes.

For example:

```
docmost_data
openproject_data
wordpress_data
inventree_data
zammad_data
```

The exact names depend on your DockerLab.

Part 10 — Your migration package

At the end, you want something resembling:

```
dockerlab-migration/
│
├── docker-compose.yml
├── .env
│
├── postgres-all.sql
├── docmost.dump
├── mariadb-all.sql
│
├── docmost_data.tar.gz
├── openproject_data.tar.gz
├── wordpress_data.tar.gz
├── inventree_data.tar.gz
├── zammad_data.tar.gz
│
├── docker-containers.txt
├── docker-volumes.txt
├── docker-networks.txt
└── docker-images.txt
```

That's your **DockerLab recovery package**.

Part 11 — Now build the NUC

I recommend:

NUC

```
Ubuntu Server 24.04 LTS
Docker Engine
Docker Compose
Cloudflare Tunnel
Tailscale
```

You don't need to duplicate the GCP `e2-standard-4`.

The NUC's hardware becomes your new:

```
4+ cores
16+ GB RAM
1 TB SSD
```

Docker doesn't care that the old server was an E2 instance.

Part 12 — Install Docker

On Ubuntu:

```
sudo apt update
sudo apt upgrade -y
```

Install Docker using Docker's official repository.

Then verify:

```
docker --version
docker compose version
```

Then:

```
sudo systemctl enable docker
sudo systemctl start docker
```

Part 13 — Create your DockerLab directory

I'd use:

```
sudo mkdir -p /opt/dockerlab
sudo chown -R $USER:$USER /opt/dockerlab
```

Then:

```
/opt/dockerlab/
```

becomes your new DockerLab root.

For example:

```
/opt/dockerlab/
├── compose/
├── data/
├── backups/
├── secrets/
└── scripts/
```

Part 14 — Copy your Compose configuration

Copy:

```
docker-compose.yml
.env
```

to:

```
/opt/dockerlab/
```

Then inspect the Compose file carefully.

You may find GCP-specific paths such as:

```
/mnt/...
/home/...
```

or networking assumptions that don't exist on the NUC.

Change those to your new NUC paths.

Part 15 — Restore your volumes

For example:

```
docker volume create docmost_data
```

Then restore:

```
docker run --rm \
-v docmost_data:/data \
-v /opt/dockerlab/backups:/backup \
ubuntu \
tar xzf /backup/docmost_data.tar.gz -C /data
```

Repeat for each application.

Part 16 — Restore PostgreSQL

Start only PostgreSQL first.

Don't start everything.

For example:

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

Wait:

```
docker logs -f postgres
```

When PostgreSQL is ready, restore the database.

For a dump:

```
cat docmost.dump | \
docker exec -i postgres \
pg_restore -U postgres -d docmost --clean --if-exists
```

The exact command depends on your existing database/user/container configuration.

This is why I strongly recommend we inspect your current Compose configuration before doing the actual migration.

Part 17 — Bring up Docmost by itself

Once PostgreSQL is restored:

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

Then:

```
docker logs -f docmost
```

Test Docmost locally:

```
http://NUC-IP
```

or whatever port you've configured.

DO NOT switch DNS yet.

You want:

```
Internet
   |
   ▼
GCP Docmost
```

still functioning while:

```
Local LAN
   |
   ▼
NUC Docmost
```

is being tested.

Part 18 — Test Docmost thoroughly

This is the part I would take seriously.

Don't just verify:

> "The login page appears."

Test:

- Login
- Spaces
- Pages
- Attachments
- Images
- Search
- Editing
- Page history
- Users
- Permissions
- Links
- Embedded content
- File uploads
- Downloads

Especially verify the **attachments**.

Your database can be perfect while your attachment volume is missing.

Part 19 — Then migrate the other applications

Do them one at a time.

I'd use this order:

Tier 1 — critical

```
Docmost
PostgreSQL
MariaDB
```

Tier 2

```
WordPress
OpenProject
OpenEMR
```

Tier 3

```
InvenTree
Zammad
phpMyAdmin
pgAdmin
```

Tier 4

```
Portainer
Nginx Proxy Manager
other utilities
```

Part 20 — Move Nginx Proxy Manager carefully

Your current GCP architecture has:

```
Internet
   ↓
NPM
   ↓
Docker containers
```

On the NUC, I would eventually use:

```
Internet
   ↓
Cloudflare
   ↓
Cloudflare Tunnel
   ↓
NUC
   ↓
NPM
   ↓
Docker containers
```

This is particularly attractive for you because you previously had the **PLDT CGNAT problem**.

You don't need a publicly reachable IPv4 address.

Part 21 — Cloudflare Tunnel becomes your GCP replacement

You already own:

```
gravelcloud.com
```

So your architecture can become:

```
docmost.gravelcloud.com
        │
        ▼
    Cloudflare
        │
        │ encrypted tunnel
        ▼
      NUC
        │
        ▼
     Docmost
```

Same for:

```
openproject.gravelcloud.com
wordpress.gravelcloud.com
inventree.gravelcloud.com
zammad.gravelcloud.com
```

You can keep your public URLs while moving the underlying server.

Part 22 — Tailscale gives you another access path

I'd also install Tailscale on the NUC.

Then:

```
Laptop
   │
   │ Tailscale
   ▼
NUC
```

becomes your private administrative path.

This is excellent for:

- SSH
- Portainer
- admin interfaces
- maintenance
- troubleshooting

while Cloudflare Tunnel handles your public applications.

Part 23 — What about Google Drive?

YES — use your 2 TB Google Drive.

But use it as:

> **Backup storage, not live Docker storage.**

For example:

```
NUC
 │
 ├── Docker
 ├── Databases
 └── Application data
       │
       ▼
   Backup script
       │
       ▼
  Google Drive
```

You could use `rclone` to synchronize encrypted backups to Google Drive.

I particularly like:

```
NUC
 ↓
restic
 ↓
encrypted repository
```

```
 ↓
Google Drive
```

rather than simply:

```
NUC
 ↓
Google Drive folder
```

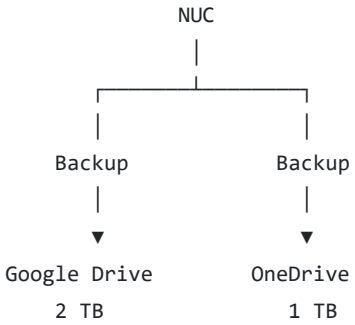
because **restic gives you versioned backups, deduplication and encryption**.

Part 24 — What about OneDrive?

Same answer.

Your 1 TB OneDrive is useful as a **second backup destination**.

You could eventually have:



That is actually a very nice setup.

Part 25 — My preferred backup strategy for you

I'd implement a **3-2-1 strategy**.

Copy #1 — live

```
NUC
```

Copy #2 — local backup

```
External 2–4 TB USB HDD/SSD
```

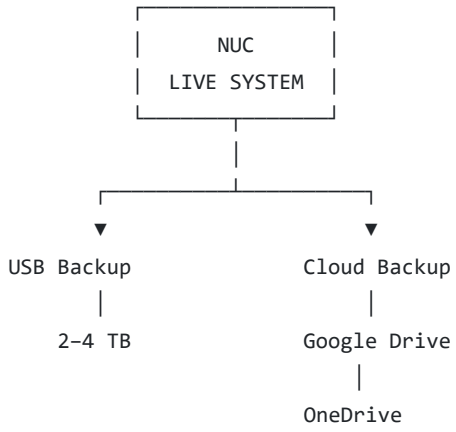
Copy #3 — cloud

```
Google Drive 2 TB
```

And optionally:

```
OneDrive 1 TB
```

So:



That's substantially more resilient than keeping your DockerLab dependent on a single GCP Persistent Disk.

Part 26 — And keep GCP alive temporarily

This is the part I would **not rush**.

After the NUC is working:

```
GCP DockerLab
      │
      │
      ├── Still operational
      │
      ▼
NUC DockerLab
      │
      │
      └── Being tested
```

Then eventually:

```
NUC
 │
 ├── Docmost verified
 ├── OpenProject verified
 ├── WordPress verified
 ├── OpenEMR verified
 ├── databases verified
 ├── Cloudflare verified
 ├── backups verified
 └── restore tested
```

Only then shut down GCP.

Google specifically notes that snapshots can be created while a disk is attached to a running VM, although data written after the snapshot isn't included; Google recommends a final snapshot after stopping the VM when you need the most current state. (**Google Cloud Documentation**)

Part 27 — Your final GCP exit backup

Right before shutting down GCP:

Stop the Docker applications

```
docker compose down
```

Then create your final Google Cloud snapshot of the boot disk.

Google supports creating snapshots from attached Persistent Disks and restoring them later to disks or VMs. (**Google Cloud Documentation**)

I would keep that final GCP snapshot for a while.

So your final safety net becomes:

```
NUC LIVE
   │
   ├── USB backup
   │
   ├── Google Drive
   │
   ├── OneDrive
   │
   └── GCP snapshot
```

That's **excellent redundancy**.

Part 28 — Don't delete GCP immediately

I'd keep the GCP environment for perhaps **30–60 days** after successful migration if the cost is tolerable.

You don't need the VM running.

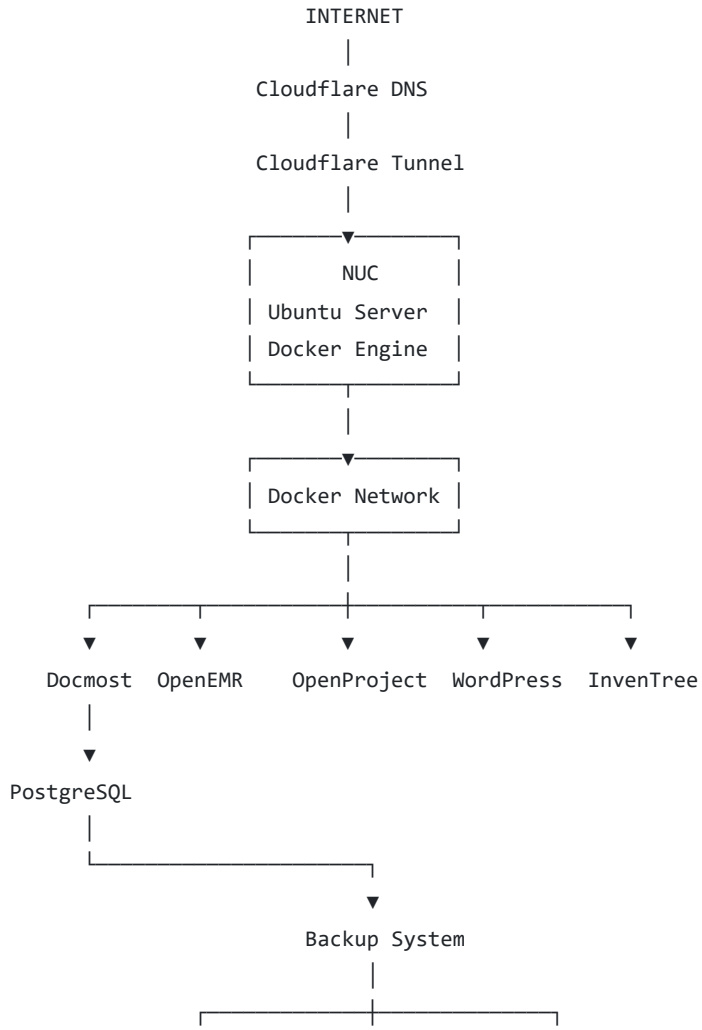
You could:

```
Stop VM
   ↓
Keep snapshot
   ↓
Keep important backups
   ↓
Test NUC
   ↓
Wait
   ↓
Delete GCP resources
```

The snapshot itself may incur storage charges, but that's dramatically different from paying for an always-running `e2-standard-4`.

The architecture I recommend specifically for you

Given your existing DockerLab, I would ultimately target:



```
              ▼           ▼           ▼
          USB HDD    Google Drive    OneDrive
```

And:

```
Tailscale
     │
     ├── SSH
     ├── Portainer
     ├── Admin
     └── Maintenance
```

One thing I would NOT do

I would **not** do this:

```
GCP 80GB disk
      ↓
copy entire disk
      ↓
Google Drive
      ↓
download to NUC
      ↓
hope Docker works
```

It can be made to work in various ways, but it gives you a **VM migration problem** when what you actually have is a **Docker application migration problem**.

Instead:

```
GCP
 │
 ├── Compose files
 ├── Config
 ├── Secrets
 ├── Docker volumes
 ├── PostgreSQL dumps
 ├── MariaDB dumps
 └── application backups
            ↓
          NUC
```

That is cleaner, portable, and much easier to recover.

And there's a particularly good reason for doing this now

Your September 8 deadline is actually useful.

Don't buy the 3-year CUD.

Instead, use the next few days to establish:

> **NUC = primary DockerLab**

while:

> **GCP = temporary migration safety net**

Then:

> **Google Drive/OneDrive = off-site backup**

That gives you freedom to decide later whether you even need GCP.

I would do this in 5 phases:

Phase	Objective	GCP
1	Inventory + backup	**RUNNING**
2	Build NUC + restore Docmost	**RUNNING**
3	Migrate remaining Docker apps	**RUNNING**
4	Cloudflare/Tailscale + backup verification	**RUNNING**
5	Final snapshot → stop/delete VM	**DONE**

Most importantly: because Docmost contains live, important data, I would make **Docmost the first application we migrate and the first one we prove can be restored**. Once that works, the rest of your DockerLab becomes considerably less intimidating.

If you want to proceed, the next thing I'd have you do is **not change anything on GCP yet**. Give me the output of:

```
docker ps
docker volume ls
docker network ls
docker compose ls
df -h
```

and, if you know where it is, your **Docker Compose file**. From that, I can map your *actual* DockerLab into a **specific GCP → NUC migration plan**, including exactly which volumes/databases need backup, what order to migrate them, and the restore commands for **Docmost specifically**.