AI-04668 Docmost Postgres Backup

I self-hosted Docmost in GCP, how to download all my pages for backup and restore using Postgres method via pgadmin. Create a step by step instructions on for pgAdmin method backup and restore.

Yes. For your **self-hosted Docmost on GCP**, the PostgreSQL/pgAdmin method is a good way to create a real database backup. But there is one important distinction:

> **PostgreSQL backup = your Docmost pages, spaces, users, permissions, page history, etc.**
> **Docmost storage-volume backup = uploaded files/attachments/images.**

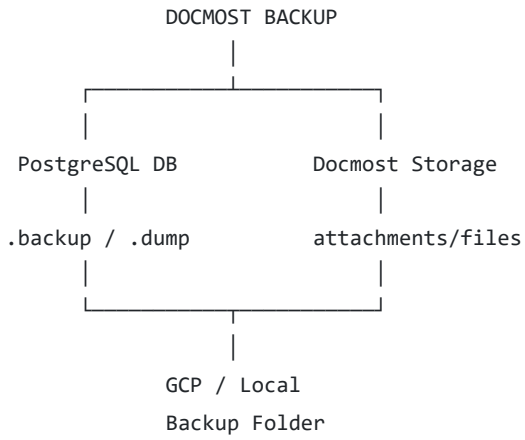
Docmost's current Docker configuration stores PostgreSQL in `db_data` and uploaded files in the Docmost storage volume mounted at `/app/data/storage`. (GitHub)

So for a **complete disaster-recovery backup**, I recommend keeping **both**.

Docmost → PostgreSQL Backup & Restore with pgAdmin

What we are going to accomplish

Your backup will look conceptually like this:



The PostgreSQL portion is the **most important piece for your actual Docmost pages** because Docmost stores page content in PostgreSQL. Uploaded files are stored separately. (GitHub)

PART 1 — Identify your Docmost PostgreSQL database

Before touching pgAdmin, let's establish your actual Docker configuration.

On your GCP VM:

```
cd /path/to/your/docmost
```

Then:

```
docker compose ps
```

You should see something similar to:

```
NAME             IMAGE
docmost          docmost/docmost:latest
docmost-db       postgres:16
docmost-redis    redis:7.2
```

Your names may be different.

Now inspect your compose file:

```
cat docker-compose.yml
```

Look for:

```
db:
  image: postgres:...
  environment:
    POSTGRES_DB: docmost
    POSTGRES_USER: docmost
    POSTGRES_PASSWORD: ...
```

You are interested in:

Item	Example
PostgreSQL database	`docmost`
PostgreSQL user	`docmost`
PostgreSQL server	your PostgreSQL container
Port	`5432`

Also look at:

```
DATABASE_URL:
```

For example:

```
postgresql://docmost:PASSWORD@db:5432/docmost
```

This confirms:

```
Database = docmost
User     = docmost
Host     = db
Port     = 5432
```

Do not change anything yet.

PART 2 — Connect pgAdmin to PostgreSQL

If your pgAdmin is already running in your DockerLab, open:

```
pgAdmin
   ↓
Servers
```

Right-click:

Servers → Register → Server

General tab

Use:

```
Name: Docmost PostgreSQL
```

The name is arbitrary.

Connection tab

If pgAdmin is another container on the same Docker network as Docmost:

```
Host name/address: db
Port:              5432
Maintenance DB:    docmost
Username:           docmost
Password:           your PostgreSQL password
```

The important point is:

If pgAdmin is inside Docker

Use:

```
db
```

NOT:

```
localhost
```

because `localhost` would mean the pgAdmin container itself.

If pgAdmin is running directly on your computer, you'll need PostgreSQL exposed to your computer or an SSH tunnel instead.

PART 3 — Verify the database

After connecting:

```
Servers
 └── Docmost PostgreSQL
      └── Databases
           └── docmost
```

Expand:

```
docmost
 └── Schemas
      └── public
           └── Tables
```

You should see numerous Docmost tables.

This is an excellent sanity check.

You can also open:

Tools → Query Tool

and run:

```
SELECT current_database();
```

You should get:

```
docmost
```

Then:

```
SELECT COUNT(*) FROM pages;
```

If your version has the `pages` table, this gives you an immediate indication that the database contains your Docmost pages.

PART 4 — Create the PostgreSQL backup

This is the important part.

In pgAdmin:

```
Servers
 └── Docmost PostgreSQL
      └── Databases
           └── docmost
```

Right-click:

docmost → Backup...

pgAdmin's Backup function uses PostgreSQL's `pg_dump` utility. It can create several formats, including Custom, Tar, Directory and Plain SQL. (**pgAdmin**)

PART 5 — Configure the backup

You'll see the **Backup** dialog.

General tab

Filename

Choose a location that pgAdmin can actually write to.

For example:

```
/tmp/docmost_2026-08-31.backup
```

or, if pgAdmin has a mounted backup directory:

```
/backup/docmost_2026-08-31.backup
```

Format

Choose:

```
Custom
```

I strongly recommend **Custom** for your primary restore backup.

Why?

Because it is specifically designed to work with `pg_restore`, and you can selectively restore objects if necessary. pgAdmin's documentation also recommends Custom for medium/large databases because it is compressed by default. (**pgAdmin**)

So:

```
Format: Custom
```

PART 6 — Data/Objects tab

For a normal complete Docmost backup, leave the defaults alone.

You want:

```
Pre-data
✓

Data
✓
```

```
Post-data
✓
```

These collectively cover database definitions, actual data, and indexes/triggers/constraints. (pgAdmin)

Do **not** choose:

```
Only data
```

and do **not** choose:

```
Only schema
```

You want the entire database.

PART 7 — Options

For your first backup, I recommend keeping this simple.

You want:

```
Complete database
Custom format
Pre-data + Data + Post-data
```

Don't enable exotic options unless you have a specific reason.

Then click:

Backup

pgAdmin will execute the `pg_dump` operation in the background. You can monitor it through the pgAdmin process/activity interface. (pgAdmin)

PART 8 — IMPORTANT: Where did pgAdmin save the file?

This is probably the part that caused confusion with your earlier Docmost export.

If pgAdmin itself is running in Docker, this:

```
/backup/docmost_2026-08-31.backup
```

means:

> `/backup` **inside the pgAdmin container**

It does **not necessarily mean**:

```
~/Downloads
```

on your Windows computer.

This is a critical Docker concept.

For example:

```
Windows PC
    │
    │
    ▼
pgAdmin container
    │
    └── /backup
          │
          └── docmost_2026-08-31.backup
```

If `/backup` isn't mapped to a host directory, the backup is sitting **inside the container**.

PART 9 — Make the backup downloadable

This is the part I would change in your GCP DockerLab.

Create a host directory:

```
mkdir -p ~/docmost-backups
```

Then configure your pgAdmin container with something like:

```
volumes:
  - ~/docmost-backups:/backup
```

Then recreate pgAdmin.

Now:

```
pgAdmin container
       │
       │ /backup
       ▼
GCP VM
       │
       │ ~/docmost-backups
       ▼
docmost_2026-08-31.backup
```

Now you can download the file from the GCP VM.

For example, from your Windows machine using `gcloud`:

```
gcloud compute scp YOUR_VM_NAME:~/docmost-backups/docmost_2026-08-31.backup .
```

Or use your preferred SCP/SFTP method.

PART 10 — Verify the backup

Do not assume a backup is good merely because pgAdmin said "successful."

First check its size:

```
ls -lh ~/docmost-backups/
```

You should see something like:

```
docmost_2026-08-31.backup    8.7M
```

The actual size depends on your database.

You can also inspect the archive using PostgreSQL tools:

```
pg_restore --list docmost_2026-08-31.backup
```

You should see a large list of database objects.

That gives you much more confidence that you actually created a PostgreSQL archive.

PART 11 — Your first restore test

This is where I strongly recommend being careful.

Do not test your restore against your production Docmost database first.

Instead create a test database.

For example:

```
docmost_restore_test
```

In pgAdmin:

```
Databases
   ↓
Right-click
   ↓
Create
   ↓
Database
```

Name it:

```
docmost_restore_test
```

Then right-click:

```
docmost_restore_test
```

and choose:

Restore...

PART 12 — Restore the backup

Select your:

```
docmost_2026-08-31.backup
```

Format

Select:

```
Custom
```

pgAdmin's Restore dialog uses `pg_restore` for Custom, Tar and Directory archives. (**pgAdmin**)

PART 13 — Restore options

For your first test restore:

Data Options

Use:

```
Pre-data     Yes
Data         Yes
Post-data    Yes
```

Query Options

For a completely empty test database, you generally don't need:

```
Clean before restore
```

because there is nothing to clean.

You can leave:

```
Include CREATE DATABASE
```

off because you've already created:

```
docmost_restore_test
```

Then:

Restore

PART 14 — Verify the restored database

After the restore completes:

```
docmost_restore_test
    ↓
Schemas
    ↓
public
    ↓
Tables
```

You should see the Docmost tables.

Then:

Tools → Query Tool

Run:

```
SELECT COUNT(*) FROM pages;
```

Compare it to the original:

```
SELECT COUNT(*) FROM pages;
```

You want:

```
Production:       347
Restore test:     347
```

for example.

You can also inspect:

```
SELECT id, title
FROM pages
ORDER BY created_at DESC
LIMIT 20;
```

This gives you a very strong confirmation that the actual Docmost page records were restored.

PART 15 — The critical issue: PostgreSQL is NOT the whole Docmost backup

This is extremely important for your setup.

Docmost's Docker configuration has:

```
docmost:
    volumes:
      - docmost:/app/data/storage
```

and PostgreSQL has:

```
db:
    volumes:
      - db_data:/var/lib/postgresql
```

The current official compose configuration therefore separates:

```
Docmost
├── PostgreSQL
│      └── pages/content/database information
│
├── Docmost storage
│      └── uploaded files/attachments
│
└── Redis
       └── application cache/state
```

(GitHub)

The Docmost project/community discussions specifically note that the page text is in PostgreSQL while uploaded files live in the storage volume. (GitHub)

Therefore:

Your complete backup should be:

Backup #1 — PostgreSQL

```
docmost_2026-08-31.backup
```

Contains the database.

Backup #2 — Docmost storage

For example:

```
docmost-storage-2026-08-31.tar.gz
```

Contains:

```
/app/data/storage
```

including uploaded files/attachments.

PART 16 — Back up the Docmost storage volume

First identify the volume:

```
docker volume ls | grep docmost
```

You may see something like:

```
docmost_docmost
docmost_db_data
docmost_redis_data
```

Find the Docmost storage volume:

```
docker volume inspect docmost_docmost
```

Then you can make an archive.

For example:

```
mkdir -p ~/docmost-backups
```

Then:

```
docker run --rm \
  -v docmost_docmost:/source:ro \
  -v ~/docmost-backups:/backup \
  alpine \
  tar czf /backup/docmost-storage-2026-08-31.tar.gz -C /source .
```

You now have:

```
~/docmost-backups/
│
├── docmost_2026-08-31.backup
│
└── docmost-storage-2026-08-31.tar.gz
```

That is **much closer to a true Docmost disaster-recovery package**.

Community backup procedures for Docmost similarly back up both the PostgreSQL database and the Docmost storage volume. (GitHub)

PART 17 — What about Redis?

For normal Docmost disaster recovery, I would **not make Redis your primary backup target**.

Your critical persistent data is:

```
1. PostgreSQL
2. /app/data/storage
```

Redis is primarily supporting application/cache state in the standard Docker architecture. The official compose configuration gives Redis its own persistent volume, but a database + storage backup is the important foundation for restoring the Docmost instance. (GitHub)

PART 18 — Complete disaster recovery

Imagine your GCP VM completely dies.

You rebuild your DockerLab.

You install:

```
Docker
Docker Compose
Docmost
PostgreSQL
Redis
```

You restore:

```
docmost_2026-08-31.backup
```

into PostgreSQL.

Then restore:

```
docmost-storage-2026-08-31.tar.gz
```

into:

```
/app/data/storage
```

Then start Docmost.

Conceptually:

```
NEW GCP VM
     │
     ├── Docker
     │
     ├── PostgreSQL
     │       ▲
     │       │
     │       │ docmost_2026-08-31.backup
     │       │
     │
     ├── Docmost
     │       ▲
     │       │
     │       │ docmost-storage-2026-08-31.tar.gz
     │       │
     │       └── /app/data/storage
     │
     └── Redis
```

PART 19 — One additional thing you absolutely should preserve

Your `.env` / Docker Compose configuration contains important information such as:

```
APP_SECRET
DATABASE_URL
POSTGRES_PASSWORD
REDIS_URL
APP_URL
```

Keep the Docmost `APP_SECRET` from the original installation.

Don't casually generate a new one during disaster recovery.

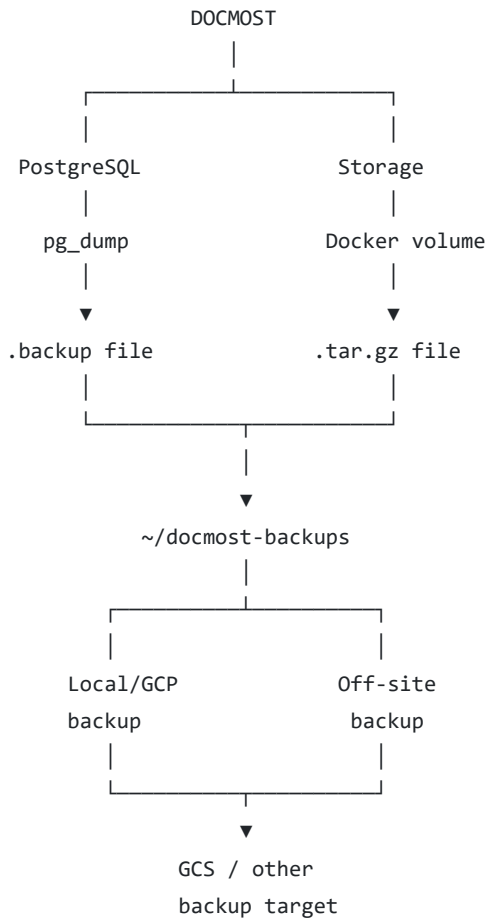
I would therefore make your backup directory:

```
docmost-backups/
│
├── 2026-08-31/
│   │
│   ├── docmost_2026-08-31.backup
│   ├── docmost-storage-2026-08-31.tar.gz
│   ├── docker-compose.yml
│   └── .env
```

But **encrypt the backup**, particularly `.env`, because it contains credentials/secrets.

PART 20 — My recommended backup architecture for your GCP DockerLab

Given your setup, I would ultimately use this:



And I would retain **multiple backup generations**, rather than having only:

```
latest.backup
```

For example:

```
docmost-backups/
│
├── 2026-08-31/
├── 2026-08-24/
├── 2026-08-17/
├── 2026-08-10/
└── 2026-08-03/
```

That protects you from discovering that yesterday's backup was corrupted **after** today's data is lost.

The most important takeaway

For your Docmost installation, think of backup as:

Component	What it protects	Backup method

PostgreSQL	Pages, spaces, users, permissions, history, etc.	**pgAdmin → Backup → Custom**
Docmost storage	Uploaded files/attachments	Docker volume → `.tar.gz`
`.env` **/ compose**	Configuration/secrets	Secure copy
Redis	Supporting application state	Usually not primary DR target

And **Custom PostgreSQL format (** `.backup` **) is the format I recommend for your pgAdmin backup**, because it can be restored with pgAdmin's Restore function/ `pg_restore` and gives you more flexibility than a raw SQL dump. (pgAdmin)

pgAdmin Backup & Restore documentation
Docmost Docker Compose configuration

One thing I would do next

Since your Docmost is already running in **your GCP DockerLab**, I can walk you through the **exact pgAdmin setup using your actual Docker containers/volumes**—including **where to put the backup file so you can actually download it to Windows**, which was the problem you encountered with the Docmost export. We can do it step-by-step without changing your production Docmost installation.