



Continuing Your Homelab

SciaticNerd

DISCLAIMER

“

There are probably other — and likely better — ways to do all of this.

This is just what is working for me.

Experiment. Break things. Learn.

That is the whole point.

”

— SciaticNerd

Last Year: The Origin Story

How the homelab journey began — and what we covered together.

1



Why self-host? If you aren't paying for a product, you **ARE** the product.

2



The Stages:

Machine



Drive



Network



VPN



Virtualization



Platform

3



Setting up accounts:

Email

Tailscale

GitHub

DuckDNS

Let's Encrypt

4



Establishing trust: Wildcard TLS cert, split DNS, reverse proxy via **Cosmos Cloud**.

Where We Left Off

Your lab should look something like this...



● ACTIVE

Cosmos Cloud Running

Installed on Pi or x86 hardware. Your self-hosted control plane is live.

`cosmos-server`



● ACTIVE

Tailscale VPN Active

Secure tunnel into your network from anywhere, on any device.

`tailscale up`



● LIVE

DuckDNS Domain Live

examplelocalcloud pointing to your server IP and resolving correctly.

`*.examplelocalcloud`



● VALID

Wildcard TLS Cert

Let's Encrypt *.examplelocalcloud certificate active & auto-renewing.

`Let's Encrypt`



Domain used throughout: `examplelocalcloud`

Starting Your Homelab — HouSecCon 2025 by SciaticNerd

Missed HouSecCon 2025?

The full talk is available — scan to watch now



Scan to Watch the Full Talk

youtu.be/8nqyA-BmKZA

Today's Agenda

Six steps. One journey. A fully-powered homelab by the end.

🕒 ~40 minutes

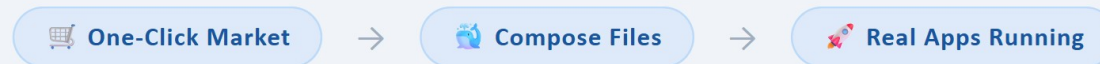
CybrSecCon 2026



“

*You built the lab. You survived the setup.
Now let's actually **DO something** with it.*

”



Today we install **real apps** — from the easy one-click market all the way to building your own **compose files from scratch**.

— CYBRSECCON 2026

Apps at One Click

The easiest way to expand your lab

01



Pre-configured cosmos-compose files ready to deploy.

Each market listing ships with a complete compose definition — no manual YAML writing required.

02



Auto-handles containers, networks, volumes, and reverse-proxy routes.

Cosmos wires up the full stack for you — networking, storage, and routing configured automatically on install.

03



Review sources before installing: repo link, Docker Hub link, raw compose file.

Every market app is transparent — inspect exactly what will run before you commit. Trust, but verify.

04



Set your hostname before clicking Install — DNS must already exist.

Define the subdomain first, e.g. `files.examplelocalcloud`, and confirm DNS resolves before triggering the install.

Step-by-Step: Files App from Cosmos Market

Cosmos Market · One-Click Install Walkthrough

FLOW



LIVE

cosmos-cloud.io
docs/market/

1



Open Cosmos dashboard → click **Market** in left sidebar.

2



Browse or search for **Files** (or **File Manager**) in the market list.

3



Click the app → review repo, Docker Hub link, and raw compose file.

4



Click **Install** → set hostname to **files.examplelocalcloud**.

5



Confirm DNS entry exists, then click **Install** to proceed.

6



Restart Cosmos if prompted to activate the new reverse-proxy route.



That's it. **Your first self-hosted file manager is live.** No terminal required.

Post-Install: What You Get



Files App

`files.examplelocalcloud`

Accessible instantly via Tailscale VPN or directly on your local network. No port-forwarding required.

● Live



SSL via Let's Encrypt

HTTPS · Auto-renewed

Full TLS encryption out of the box. No manual cert setup — Cosmos manages the wildcard certificate for you.

● Secured



Reverse Proxy Auto-Configured

Cosmos · Smart Routing

Cosmos automatically wires the subdomain route to the container. Zero nginx config. Zero manual proxy rules.

● Routed



ServApp in Cosmos Dashboard

Monitor · Restart · Update

Container appears as a ServApp — view CPU & RAM, check live logs, restart or update directly from the Cosmos UI.

● Managed



ServApps: Cosmos's Take on Docker Containers

🕒 ServApp

>_ How You Might Think of It

- Just a Docker container running on my machine
- Managed with `docker ps` and `docker stop` commands
- Logs via `docker logs`

```
$ - bash
$ docker ps
CONTAINER ID   IMAGE     STATUS
a1b2c3d4e5f6   myapp    Up 2 hours
$ docker stop a1b2
a1b2c3d4e5f6
$ docker logs myapp
[info] server started on port 80...
```



What Cosmos Calls It (ServApp)

- Container + reverse-proxy route + URL management — all unified
- **Start / Stop / Restart / Recreate / Kill** from the web UI — no CLI needed
- CPU & RAM monitoring built-in — live resource graphs per container
- Live log viewer and in-browser terminal access
- Environment variable editor — no need to touch compose files for tweaks
- Auto-update checker polls every **6 hours** — one-click updates from the dashboard

🌐 Reverse Proxy

📊 Monitoring

📄 Live Logs

⚙️ Env Editor

🔄 Auto-Update

Know Your File Paths

This matters more than you think when editing compose files



`/var/lib/cosmos`

Cosmos config home. Edit `cosmos-config.json` here for rescue situations.



`/opt/cosmos/appdata`

Recommended location for app bind mounts. Create a subfolder per app here — e.g., `/opt/cosmos/appdata/bookstack`



`docker logs cosmos-server`

Check Cosmos container logs for any issue. First stop for debugging unexpected behavior.



ServApp → Docker tab

View and edit volumes, env vars, labels per container — all without touching the CLI.



When in doubt — find the config file anywhere on the system:

```
sudo find / -name "cosmos-config.json" 2>/dev/null
```

Source: cosmos-cloud.io/docs/issues-and-troubleshooting/

File Paths & Config

Key Paths at a Glance

cosmos-cloud.io



COSMOS CONFIG

`/var/lib/cosmos`

Edit `cosmos-config.json` here. This is your rescue location when the UI is unreachable.



APP DATA

BEST PRACTICE

`/opt/cosmos/appdata`

Create a subfolder per app — e.g.,
`/opt/cosmos/appdata/bookstack` — and use it as your bind mount target.



CHECK LOGS

`docker logs cosmos-server`

First step for **any issue**. Add `--tail 50` or `-f` to follow live output in real time.



PRO TIP

Always define **EXPLICIT bind paths** in compose files — **never** rely on unnamed Docker volumes for app data. Unnamed volumes disappear on `docker rm`.

```
services:
```

```
  my-app:
```

```
    image: lscr.io/linuxserver/my-app:latest
```

```
    container_name: my-app
```

```
    ports:
```

```
      - "8080:80"
```

```
    volumes:
```

```
      - /opt/cosmos/appd
```

```
    restart: unless-stop
```

```
# docker-compose.yml
```

```
version: "3.8"
```

```
networks:
```

```
  default:
```

```
    driver: bridge
```

“

The market is a great starting point. But eventually, you'll want an app that isn't listed — and that's where the real homelab skills begin.

Going outside the market means writing (or adapting) your own `compose` files. It's not as scary as it sounds — especially with the right tools.

✓ You've got this — we'll walk through it together

”

```
environment:
```

```
  - PUID=1000
```

```
  - PGID=1000
```

```
  - TZ=America/Chicago
```

```
docker: run \
```

```
--name my-app \
```

```
-p 8080:80 \
```

```
my-image:latest
```

Why I Use Portainer (Even Though Docker Has a CLI)

Honest confession: raw Docker CLI still trips me up sometimes.



Web UI to manage all containers in one place.

No memorizing flags — start, stop, inspect, recreate from your browser.



Deploy and manage docker-compose stacks from a browser.

Paste YAML, upload a file, or pull from a GitHub repo — all via the Stacks UI.



Inspect logs, exec into containers, manage volumes — all graphically.

Live log tailing, terminal access, and volume browsing without touching the CLI.



Connect to Docker Hub, GHCR, or private registries.

Pull and manage images from any registry — authenticated or public.



Pull from the OFFICIAL Portainer image only.

Use `portainer/portainer-ce:lts` on Docker Hub — avoid unofficial forks.



PORTAINER

Container management
made human.

CE

Community Edition · Free

:lts

Always use the LTS tag



Installing Portainer CE on Linux

From the official Portainer documentation

:8000

:9443 HTTPS

1

Create the data volume

```
docker volume create portainer_data
```

2

Run the container (all one command)

```
docker run -d
  -p 8000:8000
  -p 9443:9443
  --name portainer
  --restart=always
  -v /var/run/docker.sock:/var/run/docker.sock
  -v portainer_data:/data
  portainer/portainer-ce:latest
```

3

Verify it started

```
docker ps
```

 — look for `portainer` in the list with status `Up`

4

Open browser & complete setup

`https://[your-server-ip]:9443` → complete first-time setup

Always use the official image



`portainer/portainer-ce:latest`

Pull from official Docker Hub only — avoid unofficial forks.

PORT REFERENCE

`:8000`

Edge Agent tunnel

`:9443`

Web UI (HTTPS) ← use this

VOLUME NOTE

`portainer_data` is a Docker-managed volume storing Portainer's own config — not your app data. For your apps, always use explicit bind mounts.



Docker Socket Mount

`/var/run/docker.sock` gives Portainer control over your Docker engine. Required for full management capability.

docs.portainer.io/start/install-ce/server/docker/linux

Portainer Stacks: Your Compose File Interface

› Navigate to **Stacks** → **Add Stack** to choose your deployment method



OPTION 1

Web Editor

Paste your `docker-compose` YAML directly into the browser. Features syntax highlighting and invisible character detection to catch hard-to-spot errors.



OPTION 2

Upload

Use a `.yaml` file directly from your computer. Ideal when you keep compose files version-controlled locally and want a quick, one-off deploy.



OPTION 3

Git Repository

Auto-deploy from a GitHub repo with GitOps-style updates. Portainer polls the branch and redeploy on commit — true infrastructure-as-code.



OPTION 4

Custom Templates

Save and reuse your own compose configurations as named templates. Build your library once — deploy any app in seconds from the Templates menu.

- All four options support **environment variables** and **.env file** loading — keep secrets out of your compose files.

docs.portainer.io

CybrSecCon 2026 · SciaticNerd.com

Container Architecture Matters

Know your platform before you pull an image — the wrong arch means a container that simply won't start.



Intel / AMD

x86-64 / amd64

- Most homelab PCs and mini PCs
NUC, Beelink, and similar x86 machines
- Cloud VPS providers
DigitalOcean, Linode, Vultr — all amd64 by default
- Look for these tags in Docker Hub:

amd64

x86-64

vs



ARM64

arm64v8 / aarch64

- Raspberry Pi 4 and Pi 5
The classic homelab ARM device — fully arm64 capable
- Apple Silicon Macs running Linux VMs
M1 / M2 / M3 chips — ARM architecture under the hood
- Look for these tags in Docker Hub:

arm64

arm64v8



⚠ Wrong architecture = container won't start.

Always verify the image architecture before pulling — check Docker Hub tags or run `docker inspect` on the image manifest.

QUICK CHECK

`uname -m`

run on your server

How to Find (and Evaluate) a Container Image



START HERE

Search **Docker Hub** (hub.docker.com) first — largest catalog of container images available.



TRUST LEVEL

Prefer images marked **Official** or **Verified Publisher** — the most trusted sources on the registry.



ARCHITECTURE

Look for **multi-arch support** — one image tag that works on both **amd64** and **arm64**.



FRESHNESS CHECK

Check the **last push date** — a stale image (6+ months with no updates) may be unmaintained or insecure.



COMMUNITY GOLD STANDARD

linuxserver.io images (`lscr.io`) are the community benchmark — active maintenance, always multi-arch, and thoroughly documented. When in doubt, check here first.



Weigh sustainability: active GitHub repo? Recent issues addressed? This is **YOUR data** — choose wisely.

The Three Registries You'll Use Most

Container Registries Explained • Where images actually come from

Prefix in compose: `image-name` `lscr.io/...` `ghcr.io/...`



✓ Default Registry

Docker Hub

`hub.docker.com`

Largest registry in existence. The **default** when no prefix is specified in your compose file. Contains official images, verified publishers, and community builds.

```
COMPOSE IMAGE: FIELD
image: nginx:latest
image: portainer/portainer-ce:1ts
# no prefix = Docker Hub
```



☆ Community Gold Standard

linuxserver.io

`lscr.io`

High-quality, **actively maintained**, always **multi-arch** (amd64 + arm64). The community's most trusted image source for self-hosted apps.

```
COMPOSE IMAGE: FIELD
image:
  lscr.io/linuxserver/bookstack:latest
image: lscr.io/linuxserver/mariadb:latest
# prefix: lscr.io/linuxserver/
```



🔗 Tied to GitHub Repos

GitHub Container Registry

`ghcr.io`

Tied **directly to GitHub repos**. Often the **primary release location** for open-source projects — images are published alongside the source code.

```
COMPOSE IMAGE: FIELD
image: ghcr.io/owner/image:tag
image: ghcr.io/linuxserver/bookstack
# prefix: ghcr.io/owner/
```

Pro tip: When you omit a registry prefix in your compose file, Docker assumes `docker.io/library/` — meaning it pulls from Docker Hub. Always use the full prefix for `lscr.io` and `ghcr.io` images.

Choosing the Right Image: Official vs Community

✓ Official / linuxserver.io

✓ RECOMMENDED

👤 Maintained by app developers or trusted community

🛡️ Regular security patches and updates

📦 Multi-arch support built-in (amd64 + arm64)

📁 Predictable, documented image names and tags

📈 Active GitHub with open issues addressed

VS

⚠️ Random Community Builds

⚠️ RISKY

✗ May be abandoned without warning

✗ No security SLA or update cadence

✗ Custom tweaks may break standard configs

✗ Harder to verify the Dockerfile source

✗ You inherit their mistakes



RULE

Always prefer official or linuxserver.io. When in doubt — check the GitHub repo activity before trusting it with your data.

Cosmos Compose: Docker Compose's Smarter Cousin

cosmos-cloud.io/docs/cosmos-compose/



Superset of docker-compose — most compose files work with minor tweaks.



Key addition: routes section — defines reverse-proxy rules right in the compose file.



Supports both YAML and JSON format — use whichever you're comfortable with.



Import a standard docker-compose.yml via Cosmos UI: Create ServApp → Import Docker Compose.



Unsupported docker-compose features are silently ignored — always test after import.

COSMOS-COMPOSE.YML — ROUTES BLOCK EXAMPLE

```
cosmos-compose.yml

# Standard docker-compose section
services:
  bookstack:
    image: lscr.io/linuxserver/bookstack:latest
    container_name: bookstack
    environment:
      - APP_URL=https://bookstack.examplelocalcloud
    ports:
      - "6875:80"
    restart: unless-stopped

# ✦ Cosmos-specific addition below
cosmos-compose:
  routes:
    - name: bookstack
      type: servapp
      serviceName: bookstack
      useHost: true
```

→ The `cosmos-compose` block is what separates it from plain docker-compose — reverse proxy baked right in.

cosmos-cloud.io/docs/cosmos-compose/

Standing on the Shoulders of Giants

Great places to find compose file inspiration



mariushosting.com

Docker & Synology Deep Dives

Excellent Docker and Synology guides with deep knowledge on compose files for many popular apps. Step-by-step, well-structured, and regularly updated.

⚠ Formatting may need tweaks to work with Cosmos Compose — always test after importing.



composerize.com

docker run → Compose YAML

Converts any `docker run` command into a proper, clean Compose YAML file automatically. Paste a command in, get structured YAML out — instantly.

<> Paste command → Convert → Copy
YAML



[awesome-selfhosted](https://github.com/awesome-selfhosted) · [GitHub](#)

The Curated App Catalog

A community-curated list of hundreds of self-hostable apps with links to official sources and documentation. The definitive starting point when exploring what to run next in your lab.

🔖 Browse categories · Find official repos · Link to docs

💖 PS: If these resources helped you, consider supporting the creators! [Marius](#) has a support link on his site — these folks put in real work for the community.

From `docker run` to Cosmos / Portainer in 6 Steps

1 DISCOVER

Find an app on mariushosting.com, Docker Hub, or [awesome-selfhosted](#).



3 CONVERT

Paste into composerize.com → click **Convert** → copy the resulting YAML.



5 ADJUST

Adjust volume paths to match your server layout (e.g., `/opt/cosmos/appdata/appname`).



2 COPY

Copy the `docker run` command from the docs or guide.



4 IMPORT

Paste into **Cosmos** (Create ServApp → Import Docker Compose) *OR* **Portainer** Stacks → Web Editor.



6 DEPLOY & VERIFY

Deploy → check logs → verify the app is accessible at your subdomain.



Fallback tip: If Cosmos Compose import fails, Portainer Stacks is your fallback — paste, adjust, deploy.

GOAL APPLICATION



BookStack

A Beautiful Wiki for Documenting Your Homelab

BookStack is a **free, open-source wiki platform** featuring a WYSIWYG editor and Markdown support. It's organized into **Books, Chapters, and Pages** — perfect for documenting every step of your homelab journey. And unlike cloud docs, *you own every byte of it.*



WYSIWYG + Markdown



SQL Database Backend



Self-Hosted & Private



bookstackapp.com



awesome-selfhosted



lscr.io/linuxserver/bookstack

QUICK FACTS



Free

Open Source (MIT)



Multi-Arch

amd64 + arm64



6875

Internal Port

HIERARCHY

• Book

• Chapter

• Page



Before You Deploy BookStack

docs.linuxserver.io/images/docker-bookstack/

Six things to know before writing a single line of compose YAML.



OFFICIAL IMAGE

`lscr.io/linuxserver/bookstack:latest` (multi-arch: amd64 + arm64)



REQUIRES MARIADB

`lscr.io/linuxserver/mariadb:latest` — runs as a separate container



PORT MAPPING

Internal port `6875` → maps to the web UI · set `APP_URL` to your full domain



APP_URL MUST MATCH EXACTLY

`APP_URL=https://bookstack.examplelocalcloud`



GENERATE APP_KEY FIRST

```
docker run -it --rm --entrypoint /bin/bash
lscr.io/linuxserver/bookstack:latest appkey
```



DEFAULT LOGIN CREDENTIALS

`[email protected]` / `password` — change immediately after first login

⚠ APP_URL mismatch is the #1 cause of a broken BookStack install. The URL must be exact — protocol, subdomain, and all.

[Deploy via Portainer](#) [Stacks or Cosmos](#) [Compose](#)



BookStack docker-compose.yml

lscr.io/linuxserver

▶ SERVICE: BOOKSTACK

docker-compose.yml — bookstack

```
services:
  bookstack:
    image: lscr.io/linuxserver/bookstack:latest
    container_name: bookstack
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=America/Chicago
      - APP_URL=https://bookstack.examplelocalcloud
      - APP_KEY=<your-generated-key>
      - DB_HOST=bookstack-db
      - DB_PORT=3306
      - DB_USERNAME=bookstack
      - DB_PASSWORD=yourpassword
      - DB_DATABASE=bookstackapp
    volumes:
      - /opt/cosmos/appdata/bookstack:/config
    ports:
      - 6875:80
    restart: unless-stopped
```

▶ SERVICE: BOOKSTACK-DB

docker-compose.yml — bookstack-db

```
bookstack-db:
  image: lscr.io/linuxserver/mariadb:latest
  container_name: bookstack-db
  environment:
    - MYSQL_ROOT_PASSWORD=rootpassword
    - MYSQL_DATABASE=bookstackapp
    - MYSQL_USER=bookstack
    - MYSQL_PASSWORD=yourpassword
  volumes:
    - /opt/cosmos/appdata/bookstack-db:/config
  restart: unless-stopped
```

■ Keys / directives

■ Image refs

■ Numeric values

■ String values

■ Paths / URLs

■ Secrets / placeholders



Deploy this in [Portainer Stacks](#) → [Web Editor](#) — or import via [Cosmos Compose](#) → [Create ServApp](#)



DB_PASSWORD **must match** in both services —
change before deploying!

docs.linuxserver.io
docker-bookstack



Bind Mounts vs Unnamed Volumes

Persistent Data: Why It Matters

Unnamed Docker Volume (Risky)

- ⊗ Container data stored in a **Docker-managed location** — not where you expect it.
- ⊗ **Invisible to you** unless you run `docker volume inspect`.
- ⊗ Data is **lost or orphaned** if the container is removed incorrectly.
- ⊗ Nearly **impossible to back up** without extra tooling or manual inspection.

VS

Bind Mount to Host Path (Recommended)

- ✓ Data lives at `/opt/cosmos/appdata/bookstack` — exactly where you put it.
- ✓ **Fully accessible** via terminal or file manager — browse and edit directly.
- ✓ **Survives** container stop, recreate, and updates without losing a byte.
- ✓ Trivially easy to back up with `rsync` or `cp` — no special tools needed.

☆ **ALWAYS** map `/config` (or data dirs) to explicit host paths. This is the *single most important Docker habit* to build.

Keeping Data on a NAS — Why and How

 NAS STORAGE

Sometimes your server disk isn't the right home for your data.



01

USE CASE

Large media libraries, shared documents, and backup targets — where server disk space is precious.



02

SOLUTION

Mount a network share (`SMB3/CIFS`) from your NAS as a directory on the host — it looks like any local folder to Docker.



03

COMPOSE VOLUMES

Reference the NAS mount path in your `volumes:` section exactly like any local path — containers can't tell the difference.



04

CONTAINER REBUILDS? NO PROBLEM

Update your compose, wipe and recreate the container — your data persists safely on the NAS mount, untouched.



05

BEST PRACTICE

Store NAS credentials in `/etc/.nasCredentials` (`chmod 600`) — **never** hardcode them in your compose file.

- Compatible with Synology DSM 7.x and Unraid current release · See next slides for step-by-step mount instructions ·

Connecting to Synology DSM 7.x via SMB3

NAS Integration

Mount your Synology share on the Cosmos host for persistent, accessible container data

1 Synology Control Panel → File Services → SMB → Enable SMB service

Set `Minimum SMB protocol` to `SMB3` for security

2 Create a Shared Folder — e.g., `homelab-data`

Grant your user `read/write` access in Synology permissions panel

3 On Cosmos host — install CIFS utils & create mount point:

```
sudo apt install cifs-utils && sudo mkdir -p /mnt/nas/homelab-data
```

4 Mount the share (test one-time mount):

```
sudo mount -t cifs //[NAS-IP]/homelab-data \
/mnt/nas/homelab-data \
-o username=user,password=pass,vers=3.0
```

5 Create credentials file with `username=` and `password=` lines:

```
# /etc/.nasCredentials
username=youruser
password=yourpass
---
sudo chmod 600 /etc/.nasCredentials
```

6 Add to `/etc/fstab` for auto-mount on reboot:

```
//[NAS-IP]/homelab-data /mnt/nas/homelab-data \
cifs credentials=/etc/.nasCredentials,
vers=3.0,_netdev 0 0
```

⚠ Never store credentials in plain text inside a compose file — always use `/etc/.nasCredentials` with `chmod 600`

UNRAID NAS

Connecting to Unraid via SMB3

Mount an Unraid share on your Cosmos host and wire it directly into your container volumes.

Protocol: SMB2 / SMB3

Package: cifs-utils

Mount root: /mnt/nas/

~5 min setup

1

CREATE SHARE ON UNRAID

Unraid UI → Shares → Create Share. Enable SMB access. Note your `share name` and `[UNRAID-IP]`.

2

SET MINIMUM SMB PROTOCOL

Unraid → Settings → Network Services → Samba → set *Minimum protocol* to `SMB2` or `SMB3` for security.

3

INSTALL CIFS-UTILS & MOUNT ON COSMOS HOST

```
sudo apt install cifs-utils && sudo mkdir -p /mnt/nas/unraid-share
sudo mount -t cifs //[UNRAID-IP]/share-name /mnt/nas/unraid-share \
-o username=unraid-user,password=pass,vers=3.0
```

4

PERSIST VIA /ETC/FSTAB

Same pattern as Synology — add entry to `/etc/fstab` using your Unraid IP, share name, and `credentials=/etc/.nasCredentials,vers=3.0,_netdev`.

5

REFERENCE IN COMPOSE FILE

```
volumes:
  - /mnt/nas/unraid-share/bookstack-uploads:/config/www/files
```

!

Pro Tip: The Unraid community plugin **Recycle Bin** helps prevent accidental deletes on shares — any deleted file goes to the bin instead of being permanently removed. Install it from the **Community Applications** plugin in Unraid.

Wiring NAS Paths into Compose Files

BookStack config paths mapped to NAS

docker-compose.yml — volumes section

```
services:
  bookstack:
    ...
    volumes:
      - /opt/cosmos/appdata/bookstack:/config
        # local config, .env, logs

      - /mnt/nas/homelab-data/bookstack-uploads:
        /config/www/files
        # user uploads → NAS

      - /mnt/nas/homelab-data/bookstack-images:
        /config/www/images
        # images → NAS
```

BOOKSTACK /CONFIG PATH REFERENCE

```
/config/www/.env → .env file (APP_KEY, DB creds)
/config/backups → storage/backups
/config/www/images → storage/uploads/images
/config/www/files → storage/uploads/files (attachments)
```



Mount your NAS BEFORE starting containers — a missing mount = container bind failure on startup.



Keep credentials in `/etc/.nasCredentials` with `chmod 600` — **never in the compose file.**



BookStack /config paths reference — shown in the code panel at left. Keep local config on the host; push uploads & images to NAS for easy access and backup.



PRO TIP

Store only **config**, **.env**, and **logs** locally. Push **uploads** and **images** to NAS — they grow large over time and benefit most from centralized backup.



Deploy via [Portainer Stacks](#) or [Cosmos](#) → [Import Docker Compose](#)

Keeping Your Lab Healthy

Build these habits now — before something breaks.

WEEKLY 🕒

Check Cosmos ServApps for available updates.
Review container logs for recurring errors.
`docker ps` to confirm all containers running.

ON UPDATE ↻

`docker pull [image]:latest` → recreate via Cosmos or Portainer.
Test login and functionality after every update.
Check logs for errors post-update.

📅 MONTHLY

- `docker image prune -a` → removes unused images (free up disk).
- `docker volume prune` → removes orphaned volumes.
- Verify NAS mounts still connected.

🛡️ AFTER ANY CHANGE

- Verify your bind mounts are still correct.
- Confirm URLs still resolve at `examplelocalcloud`.
- Check Let's Encrypt cert renewal status in Cosmos.

• Maintenance Cadence

Consistency is what separates a homelab from a homecrash.

Back It Up Before It's Gone

BACKUP STRATEGY

A backup plan you will actually follow.



WHAT TO BACK UP

`/var/lib/cosmos` (Cosmos config) · `/opt/cosmos/appdata` (all app data) · NAS share contents



SIMPLE TOOL

One command to rule them all:

```
rsync -av --progress /opt/cosmos/appdata/ /mnt/backup/appdata/
```



DATABASE CONTAINERS

Always back up the `/config` volume — it contains the actual DB files for MariaDB and other database containers.



WHERE TO BACK UP

External USB drive · Second NAS · Cloud — **Backblaze B2** or **Wasabi** (affordable S3-compatible)



SUGGESTED CADENCE

Daily — automated rsync for app data · **Weekly** — full config snapshot · **Monthly** — off-site cloud upload



TEST YOUR RESTORES!

A backup you've **never tested** is not a backup. Periodically restore to a test location and verify your data is intact.

Source: cosmos-cloud.io/docs/

3-2-1 Rule: 3 copies · 2 media types · 1 off-site



True Story: The Ghost Incident



*I was running a Ghost blog on my homelab. One day, I decided to clean things up — and accidentally wiped the database volume. The blog content? **Gone**. Had to start completely from scratch.*



1

Know **EXACTLY** what you are deleting before you run any `docker rm` or `volume prune`.

2

Always back up your database volumes separately — they are the most critical and the most often forgotten.

3

This is **exactly what the homelab is for** — live, break, learn, and do better next time.

"Failure is just a lesson you paid for with your afternoon."

Questions?



SciaticNerd.com

@SciaticNerd

THANK YOU FOR ATTENDING CYBRSEC CON 2026