

# Plan to Deploy NAS-Cloud (Single Image + Cross-Platform + Auto-Update)

---

## 1) Objectives

- Run NAS-Cloud with **no host dependency installs** (except Docker, or executable runtime).
  - Use **one deployable artifact** for many device types (Pi, Linux server, Windows, macOS).
  - Keep user data safe across restarts/rebuilds.
  - Support controlled updates with rollback.
- 

## 2) Recommended Deployment Models

### Model A (Primary): Single Docker Image (multi-arch)

Best for Raspberry Pi and servers.

- One container image includes:
  - Python backend
  - Built frontend assets (Drive + Photos)
- Runtime data mounted from host volume.
- Image published as multi-architecture manifest:
  - `linux/amd64`
  - `linux/arm64`
  - `linux/arm/v7` (optional, for older Pi)

### Model B (Optional): Standalone Executable App

Best when Docker is unavailable or undesired.

- Package backend + static frontend into native app bundles.
  - Installers per OS.
  - Auto-update via app updater channel (or platform package manager).
- 

## 3) Data Persistence Design (Critical)

Use a dedicated data path **outside image**:

- `/data/nas_cloud.db` (SQLite DB)
- `/data/nas_storage/` (files/uploads)
- `/data/logs/` (optional)

Rules:

- Never store persistent data in image layer.
- Upgrades replace container/app binary, not `/data`.
- Back up `/data` regularly.

---

## 4) Single-Image Docker Strategy

### 4.1 Architecture Choice

Use a **single runtime process** where possible:

- FastAPI serves API + static frontend bundles.

If current architecture requires Node runtime at runtime, use a process supervisor (temporary). Long-term, prefer backend static serving to simplify operations.

### 4.2 Build Pipeline

1. Build frontend artifacts (`dist-drive`, `dist-photos`).
2. Copy artifacts into backend image.
3. Install backend dependencies.
4. Set env defaults:
  - `DATABASE_URL=sqlite:///data/nas_cloud.db`
  - `NAS_STORAGE_PATH=/data/nas_storage`
5. Expose backend port (e.g., `8000`).

### 4.3 Multi-Arch Build & Publish

From CI (GitHub Actions recommended) or local `buildx`:

```
docker buildx create --name nasbuilder --use
docker buildx inspect --bootstrap

docker buildx build ^
  --platform linux/amd64,linux/arm64,linux/arm/v7 ^
  -t ghcr.io/<ORG>/nas-cloud:latest ^
  -t ghcr.io/<ORG>/nas-cloud:1.0.0 ^
  --push .
```

(Windows PowerShell line continuations can use ``` instead of `^` if needed.)

### 4.4 Runtime on Device

```
docker run -d --name nas-cloud ^
  -p 8000:8000 ^
  -v D:\nascloud-data:/data ^
  -e DATABASE_URL=sqlite:///data/nas_cloud.db ^
  -e NAS_STORAGE_PATH=/data/nas_storage ^
  --restart unless-stopped ^
  ghcr.io/<ORG>/nas-cloud:latest
```

On Linux/Pi:

```
docker run -d --name nas-cloud \  
-p 8000:8000 \  
-v /srv/nascloud-data:/data \  
-e DATABASE_URL=sqlite:///data/nas_cloud.db \  
-e NAS_STORAGE_PATH=/data/nas_storage \  
--restart unless-stopped \  
ghcr.io/<ORG>/nas-cloud:latest
```

---

## 5) Auto-Update Strategy (Docker)

Important: Containers should not self-mutate internally.

Use host/orchestrator-driven update flow.

### Option 1: Watchtower (simple)

Watchtower monitors image tags and recreates containers when new images are pushed.

```
services:  
  nas-cloud:  
    image: ghcr.io/<ORG>/nas-cloud:latest  
    container_name: nas-cloud  
    ports:  
      - "8000:8000"  
    volumes:  
      - ./data:/data  
    environment:  
      DATABASE_URL: sqlite:///data/nas_cloud.db  
      NAS_STORAGE_PATH: /data/nas_storage  
    restart: unless-stopped  
  
  watchtower:  
    image: containrrr/watchtower:latest  
    container_name: watchtower  
    volumes:  
      - /var/run/docker.sock:/var/run/docker.sock  
    command: --interval 300 --cleanup  
    restart: unless-stopped
```

### Option 2: Controlled update script (safer for production)

- Pull new image.
- Health-check new container.
- Swap traffic.
- Keep last working tag for rollback.

## 6) Rollback Plan

Tag every release (do not rely only on **latest**).

Rollback command:

```
docker stop nas-cloud
docker rm nas-cloud
docker run -d --name nas-cloud \
  -p 8000:8000 \
  -v /srv/nascloud-data:/data \
  --restart unless-stopped \
  ghcr.io/<ORG>/nas-cloud:<previous-version>
```

Data remains intact because **/data** is external.

---

## 7) Security & Reliability Baseline

- Run container as non-root user where possible.
  - Pin dependency versions.
  - Add health endpoint (**/health**).
  - Configure log rotation.
  - Use HTTPS via reverse proxy (Caddy/Nginx/Traefik) on LAN/WAN.
  - Back up **/data** daily and before each upgrade.
- 

## 8) CI/CD Release Flow (Recommended)

On every tagged release:

1. Run tests.
  2. Build multi-arch image.
  3. Push to GHCR/Docker Hub.
  4. Create release notes/changelog.
  5. Optional: trigger staged rollout (not immediate auto-update in all devices).
- 

## 9) Standalone Executable Plan (No Docker)

### 9.1 When to choose this

Use for environments where Docker cannot be installed.

### 9.2 Packaging approach

- Backend: package Python app with **PylInstaller** (or Nuitka).
- Frontend: pre-build static assets and include in package.
- App starts local web server and opens browser (or use desktop shell like Tauri/Electron).

## 9.3 Cross-platform build matrix

Build on each target OS (preferred):

- Windows: `.exe` + installer (Inno Setup/MSIX)
- Linux: AppImage or `.deb/.rpm`
- macOS: `.app` + notarized `.dmg`

## 9.4 Runtime data locations (per OS)

Use user data directories, not install directory:

- Windows: `%ProgramData%\NASCloud\` or `%AppData%\NASCloud\`
- Linux: `/var/lib/nascloud/` (service mode) or `~/.local/share/nascloud/`
- macOS: `~/Library/Application Support/NASCloud/`

Store DB/files there exactly as:

- `nas_cloud.db`
- `nas_storage/`

## 9.5 Auto-update for executable apps

Two patterns:

1. **Built-in app updater framework**
  - Tauri/Electron updater channels.
2. **External updater service**
  - App checks signed manifest, downloads next package, replaces app on restart.

Requirements:

- Signed binaries
- HTTPS update endpoint
- Versioned releases
- Rollback cache of prior version

## 9.6 Service mode (headless)

For “always-on” behavior:

- Windows: install as Windows Service
- Linux: `systemd` service unit
- macOS: LaunchDaemon/LaunchAgent

---

## 10) Recommended Path for This Project

1. Implement/keep **single Docker image** as primary distribution.
2. Publish **multi-arch tags** (`amd64`, `arm64`, optional `arm/v7`).
3. Use host volume `/data` for DB + files.

4. Use Watchtower only if fully unattended updates are required.
  5. Add controlled rollout + rollback for production.
  6. Build standalone installer later for non-Docker users.
- 

## 11) Deployment Checklist

- ☐ Multi-arch image builds successfully
  - ☐ `/data` mount verified
  - ☐ DB survives restart/redeploy
  - ☐ Upload files survive restart/redeploy
  - ☐ Health checks pass
  - ☐ Backup/restore tested
  - ☐ Update tested
  - ☐ Rollback tested
  - ☐ TLS/reverse proxy configured
  - ☐ Release notes published
- 

## 12) Minimum Commands Quick Reference

Build and run:

```
docker build -t nas-cloud:local .
docker run -d --name nas-cloud -p 8000:8000 -v ./data:/data --restart unless-
stopped nas-cloud:local
```

Update:

```
docker pull ghcr.io/<ORG>/nas-cloud:latest
docker stop nas-cloud && docker rm nas-cloud
docker run -d --name nas-cloud -p 8000:8000 -v ./data:/data --restart unless-
stopped ghcr.io/<ORG>/nas-cloud:latest
```

Rollback:

```
docker run -d --name nas-cloud -p 8000:8000 -v ./data:/data --restart unless-
stopped ghcr.io/<ORG>/nas-cloud:<old-tag>
```