

Creare un progetto Docker Compose su OMV: dalla cartella al container gestito

Tag: docker · Guide · OMV

Ultimo aggiornamento: 2026-09-18

Creare un progetto Docker Compose su OMV: dalla cartella al container gestito

Guida passo-passo per portare un'applicazione esistente (che gira a host, con cron/systemd) in un container gestito dall'app Docker di OMV (openmediavault-compose), la stessa usata per dmnotes, assistente, mailcow.

Esempio concreto usato in tutta la guida: il dashboard DBSoft (busce.it/dashboard/dbsoft) migrato a container su NAS-MAIN il 2026-09-15.

Panoramica dell'architettura OMV

L'app OMV compose gestisce progetti così:

- Il config vive in `conf.service.compose.file`: ogni voce ha `name`, `description`, `body` (il YAML dei services), `env` (variabili opzionali in `.env`), `override` (opzionale `compose.override.yml`).
- Lo stato di deploy (`omv.deploy.compose`) renderizza in `/export/docker_config/<name>/:`
`<name>.yml` — il compose completo (header auto-generato + body) `compose.yml` — symlink su `<name>.yml` `<name>.env` (+ symlink `.env`) e `compose.override.yml` se configurati
- I comandi dell'app: `omv-compose-start/stop/update/build/backup <name>` (bottoni nel WebUI sotto Docker, stessa cosa da CLI).
- I jobs (`conf.service.compose.job`) generano cron in `/etc/cron.d/omv-compose-*` per auto-start al boot, backup, update periodici, ecc.

Regola d'oro: i file renderizzati non si modificano a mano — si cambia il progetto nel WebUI (o il conf) e si rilancia lo stato di deploy.

Passo 1 — Preparare la cartella del progetto

Tutto ciò che gira a host deve essere portabile in un filesystem isolato. Crea una cartella sotto `/export/docker_config/<nome>/` con:

```
/export/docker_config/dashboard/
├── Dockerfile
├── entrypoint.sh
├── <scripti dell'app>          # .py, .sh, ...
└── config/                   # tutto ciò che l'app legge (config, log storici, chiavi se serve)
```

Per il dashboard:

- `status-collector.py` + `dashboard-admin.py` (i due programmi)
- `config/dbsoft.json` (config dell'app), `config/admin.pw`, `config/dbsoft/logs/` (storico log già centralizzato), `config/verify-reports/`

Principio: l'app dentro il container non deve vedere il filesystem della macchina, solo ciò che monta. Perciò i path vanno resi configurabili:

```
# esempio: path da env con default = valore storico (compatibile con il deploy host)
CONFIG_DIR = os.environ.get("DASHBOARD_CONFIG_DIR", "/etc/dashboard")
```

e i comandi local root-aware:

```
SUDO = "" if os.geteuid() == 0 else "sudo -n " # nel container siamo root, senza sudo
```

Il codice resta eseguibile a host invariato: i default non cambiano.

Passo 2 — Dockerfile

Base minimale + solo i pacchetti che servono (nessun `pip install` se l'app usa solo `stdlib`, come il dashboard):

```
FROM python:3.12-slim

RUN apt-get update && apt-get install -y --no-install-recommends \
    openssh-client smartmontools mdadm procs \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /app
COPY status-collector.py dashboard-admin.py entrypoint.sh ./
COPY config/ ./config/

RUN chmod +x /app/entrypoint.sh

EXPOSE 8080
ENTRYPOINT ["/app/entrypoint.sh"]
```

Pacchetti del dashboard: `openssh-client` (SSH verso gli altri host), `smartmontools` (`smartctl`), `mdadm` (RAID), `procs` (`uptime`).

Passo 3 — entrypoint.sh (i processi del container)

Un container = un processo principale + eventuali background. Pattern:

```
#!/bin/bash
set -e
export DASHBOARD_CONFIG_DIR=/app/config
export ADMIN_BIND=0.0.0.0
export ADMIN_PORT=8080

python3 /app/dashboard-admin.py &          # pannello admin
ADMIN_PID=$!

while true; do
    python3 /app/status-collector.py dbsoft # job ciclico: 1 ciclo subito
    sleep 300                               # poi ogni 5 minuti (come il vecchio cron)
    if ! kill -0 $ADMIN_PID 2>/dev/null; then
        echo "[entrypoint] admin morto: exit per far riavviare docker"
        exit 1
    fi
done
```

Con `restart`: `unless-stopped` nel compose, se l'entrypoint esce docker riavvia tutto. I job ciclici "tipo cron" vivono qui, non più in cron del host.

Passo 4 — Creare il progetto in OMV

Due vie:

A. WebUI (consigliata): OMV → Docker → Aggiungi, inserisci nome e il body del compose (sotto). OMV crea la voce in conf e lo stato di deploy.

B. Da CLI/root (senza password OMV, come fatto per il dashboard): script Python con `openmediavault.config.Database` — serve solo `OMV_CONFIG_FILE` (`/etc/openmediavault/config.xml`, da `/etc/default/openmediavault`) e il trucco `OMV_CONFIGOBJECT_NEW_UUID` per la creazione:

```
import os, uuid as u
os.environ["OMV_CONFIG_FILE"] = "/etc/openmediavault/config.xml"
import openmediavault
openmediavault.getenv = lambda k, d=None, return_type="str": os.environ.get(k, d)
from openmediavault.config import Database, Object

new = str(u.uuid4())
os.environ["OMV_CONFIGOBJECT_NEW_UUID"] = new # serve che db.set() lo veda come "nuovo"
f = Object("conf.service.compose.file")
f.set("uuid", new)
f.set("name", "dashboard")
f.set("description", "Dashboard DBSoft: collector stato host + pannello amministrazione")
f.set("body", "")
services:
    dashboard:
        ...
"".strip())
f.set("showenv", False); f.set("env", "")
f.set("showoverride", False); f.set("override", "")
Database().set(f)
```

Il body va così com'è nel file renderizzato (il template lo stampa verbum tergo dopo l'header auto-generato), quindi deve contenere `services:` e tutto il resto del `compose`.

Passo 5 — Rilanciare lo stato di deploy

```
omv-salt deploy run compose
```

Dopo: in `/export/docker_config/dashboard/` compaiono `dashboard.yml`, `compose.yml` (symlink), `dashboard.env` (+ `.env`), `compose.override.yml`. Verificare subito che il YAML renderizzato sia quello atteso (specie `healthcheck` e `quote`).

Passo 6 — Primo avvio

L'app usa questo pattern di comandi (stesso ordine dei bottoni WebUI):

```
cd /export/docker_config/dashboard
docker compose -f dashboard.yml -f compose.override.yml \
    --env-file /export/docker_config/global.env --env-file dashboard.env up -d
```

- `omv-compose-build dashboard` → `docker compose build` (immagine locale)
- `omv-compose-update dashboard` → pull + ricreazione dei container running
- al primo avvio `up -d` (crea il container con le label dell'app:
`com.docker.compose.project.config_files` → `dashboard.yml`)

Nota: se l'immagine è solo di build locale (es. `dashboard:1.0.0`, non presente in alcun registry), `docker compose pull` fallisce: per gli update usarla solo dopo una build, oppure `up -d --build`.

Passo 7 — Auto-avvio al boot

Aggiungere un job (WebUI: Docker → job, oppure da conf come al Passo 4):

```
j = Object("conf.service.compose.job")
j.set("uuid", str(u.uuid4()))
j.set("enable", True)
j.set("filter", "dashboard") # solo questo progetto
j.set("filestart", True) # start al boot
j.set("execution", "reboot") # @reboot (o schedulazioni: minute/hour/...)
```

```
# gli altri flag (backup/update/prune/filestop/...) = False
Database().set(j)
```

poi `omv-salt deploy run compose` → genera `/etc/cron.d/omv-compose-start` con `@reboot root omv-compose-start-multi -f 'dashboard' ....`

Passo 8 — Rendere il servizio raggiungibile

Dipende dal caso:

- Serve sulla LAN/Internet: rete esterna del reverse proxy (es. `web_proxy` di `dmnotes`, `buscesite_default` per il busce) + Caddy: `handle_path /dmnotes/* { reverse_proxy dmnotes:8000 }`
- Solo da host (es. pannello admin del dashboard): pubblica sulla loopback del host: `ports: ["127.0.0.1:8080:8080"]`.
- Serve file statici (il caso dashboard): nessuna route Caddy — il container scrive i file in una dir già servita dal `file_server`: `volumes: - /export/docker_config/buscesite/site/dashboard:/site # rw` con `site_root: /site` nella config dell'app.

Attenzione ai nomi host: dentro il container non esistono le voci di `/etc/hosts` del NAS. Se l'app usa nomi (es. `PC-DESKTOP`), metterli in config con IP espliciti, altrimenti la risoluzione fallisce.

Passo 9 — Healthcheck e verifica

Sempre uno healthcheck (i bottoni OMV e il dashboard lo usano):

```
healthcheck:
  test:
    - CMD
    - python3
    - -c
    - "import urllib.request; urllib.request.urlopen('http://127.0.0.1:8080/', timeout=5).read()"
  interval: 30s
  timeout: 7s
  retries: 3
  start_period: 15s
```

Verifiche post-deploy (checklist):

```
docker ps --filter name=dashboard          # Up (healthy)
docker inspect dashboard --format '{{index .Config.Labels "com.docker.compose.project.config_files"}}'
curl -sk https://busce.it/dashboard/dbsoft/ # pagina pubblica 200
curl -s http://127.0.0.1:8080/              # admin (solo host)
docker logs dashboard --tail 20
```

e testare davvero i comandi dell'app: `omv-compose-stop dashboard` → `omv-compose-start dashboard`.

Passo 10 — Gestione quotidiana

Azione	WebUI	CLI
start/stop	bottoni su Docker → dashboard	<code>omv-compose-start/stop dashboard</code>
update (pull+recreate)	bottone Update	<code>omv-compose-update dashboard</code>
build immagine	bottone Build	<code>omv-compose-build dashboard</code>

Azione	WebUI	CLI
cambiare config/volumi/porte	modificare il progetto nel WebUI	modificare conf.service.compose.file + omv-salt deploy run compose + omv-compose-update dashboard
log	Docker → dashboard → log	docker logs dashboard

Il compose finale del dashboard (body del progetto OMV)

```

services:
  dashboard:
    build:
      context: /export/docker_config/dashboard
    image: dashboard:1.0.0
    container_name: dashboard
    restart: unless-stopped
    environment:
      - TZ=Europe/Rome
    volumes:
      - /export/docker_config/dashboard/config:/app/config
      - /export/docker_config/buscbsite/site/dashboard:/site
      - /var/log/omv-backup.log:/var/log/omv-backup.log:ro
      - /var/log/NAS-BACKUP-orchestrator:/var/log/NAS-BACKUP-orchestrator:ro
      - /var/lib/NAS-BACKUP-orchestrator:/var/lib/NAS-BACKUP-orchestrator:ro
      - /export/DRD_Backup:/export/DRD_Backup:ro
      - /root/.ssh:/root/.ssh
    networks:
      - buscbsite_default
    ports:
      - "127.0.0.1:8080:8080"
    healthcheck:
      test:
        - CMD
        - python3
        - -c
        - "import urllib.request; urllib.request.urlopen(\"http://127.0.0.1:8080/\", timeout=5).read()"
      interval: 30s
      timeout: 7s
      retries: 3
      start_period: 15s

networks:
  buscbsite_default:
    external: true

```

Errori tipiche (viste sul campo)

- 1 Healthcheck con URL malformato: attenzione alle quote nel body del compose (YAML scalar con quote dentro). Verificare sempre la riga renderizzata in <name>.yaml e che docker inspect mostri health OK.
- 2 is_new di Database.set: senza OMV_CONFIGOBJECT_NEW_UUID impostato e letto, la creazione di un oggetto conf viene trattata come update e fallisce.
- 3 SSH self-connect: se l'app deve SSH su se stessa (es. host locale diventato root@IP nel container) la key di root va in authorized_keys (il vecchio collector usava type: local e non ne aveva bisogno).
- 4 Nomi host non risolti nel container → usare IP espliciti in config.

- 5 `/etc/hosts` del host non esiste nel container → idem, IP espliciti.
- 6 File renderizzati editati a mano → persi al prossimo deploy; passare sempre dal WebUI/conf.
- 7 `omv-compose-start` non crea i container mancanti (fa solo `docker compose start`): il primo avvio richiede `up -d/update`.
- 8 Pull di immagini solo locali fallisce negli update: fare prima build o usare `up -d --build`.

Pulizia del vecchio deployment host

Dopo il cutover, verificare che non restino riferimenti e rimuovere:

- 1 disabilitare prima (per sicurezza): `systemctl disable <svc>`, `cron` → `.disabled`
- 2 testare il container (pagina, log, cicli)
- 3 archiviare in un tarball di sicurezza (es. `/root/backup-<nome>-host-YYYYMMDD.tgz`)
- 4 rimuovere: unit `systemd` (+ `daemon-reload`), `cron`, config dir (es. `/etc/<app>`), script in `/usr/local/sbin/` (e `.bak`, `__pycache__`), log dedicati (es. `/var/log/<app>/`), file compose manuali (`docker-compose.yml` → rimosso, al suo posto il `<name>.yml` dell'app)
- 5 verifica finale: `grep -r` dei path in `/etc/systemd`, `/etc/cron*`, `/usr/local/sbin` → nessun riferimento; container healthy; servizi 200.