hash_tool

Documentation technique — Dockerisation

Version 0.12
Février 2026

Périmètre

Ce document couvre la containerisation de hash_tool via Docker : conception de l'image Alpine multi-stage, stratégie de build, entrypoint, orchestration Docker Compose, et déploiement sur les trois environnements cibles (Windows/WSL, NAS Synology, serveur Debian).

Table des matières

1. Contexte et objectifs

1.1 Problème résolu

hash_tool dépend de deux binaires — b3sum et jq — dont les versions et les méthodes d'installation varient selon l'OS. Sur Windows, b3sum n'est pas disponible nativement. Sur un NAS Synology, l'accès à un gestionnaire de paquets est limité. Sur un serveur Debian de production, toute installation manuelle est une dette technique.

La containerisation élimine cette dépendance d'environnement : l'image Docker embarque des versions fixes et vérifiées de tous les outils. L'hôte n'a besoin que de Docker.

Environnement	Problème sans Docker	Solution Docker
Windows / WSL	b3sum non dispo nativement, jq à installer manuellement	docker run — aucune installation
NAS Synology	Gestionnaire de paquets limité, accès SSH requis	Image ARM64 compatible DSM 7.x
Serveur Debian	Versions système potentiellement anciennes	Versions pinned dans le Dockerfile

1.2 Contraintes de conception

- Image la plus légère possible — pas de toolchain Rust dans l'image finale
- Support multi-architecture : amd64 (x86), arm64 (NAS DS923+, serveurs ARM), armv7 (vieux NAS)
- Binaire b3sum officiellement signé et vérifié au build
- Interface identique à l'outil natif — pas de changement de workflow
- Isolation des données par volumes — l'image ne contient aucune donnée utilisateur

2. Architecture de l'image

2.1 Multi-stage build

L'image utilise un build en deux stages pour dissocier les outils de compilation des outils d'exécution.

```
# Stage 1 : fetcher — téléchargement et vérification de b3sum
FROM alpine:3.19 AS fetcher
  → wget b3sum binaire musl depuis GitHub Releases
  → b3sum --check (auto-vérification cryptographique)

# Stage 2 : image finale — runtime uniquement
FROM alpine:3.19
  → apk install bash jq coreutils findutils
  → COPY --from=fetcher /usr/local/bin/b3sum
  → COPY scripts hash_tool
```

Le stage fetcher nécessite wget et ca-certificates, qui ne sont pas copiés dans l'image finale. La toolchain Rust (~700 Mo) n'est jamais présente — b3sum est récupéré sous forme de binaire musl pré-compilé depuis les releases officielles BLAKE3.

> ! **Pourquoi musl ?**
> Alpine Linux utilise musl libc au lieu de glibc. Les binaires musl sont statiquement liés et fonctionnent sans dépendances dynamiques — plus robustes dans un conteneur minimal. Le projet BLAKE3 publie des binaires musl officiels pour amd64, arm64 et armv7.

2.2 Taille finale de l'image

Couche	Taille approx.	Contenu
Alpine 3.19 base	~7 Mo	OS minimal musl libc
bash + jq + coreutils	~5 Mo	Shell, parser JSON, outils POSIX
b3sum binaire musl	~2 Mo	Binaire officiel BLAKE3
Scripts hash_tool	< 100 Ko	runner.sh, integrity.sh, lib/
Total image finale	~14 Mo	—
Image Debian équivalente	~180 Mo	Référence comparative

2.3 Détection d'architecture

La sélection du binaire b3sum adapté à l'architecture se fait automatiquement dans le Dockerfile via uname -m :

```
ARCH="$(uname -m)"
case "$ARCH" in
  x86_64)  B3SUM_ARCH="linux_amd64_musl"   ;;  # PC, serveur standard
  aarch64) B3SUM_ARCH="linux_aarch64_musl"  ;;  # NAS Synology ARM, RPi 4
  armv7l)  B3SUM_ARCH="linux_armv7_musl"    ;;  # Vieux NAS, RPi 2/3
esac
```

Un seul Dockerfile couvre les trois architectures. Le build multi-platform Docker Buildx permet de produire une image manifeste unique pour les trois cibles simultanément.

2.4 Vérification cryptographique du binaire

Le projet BLAKE3 publie pour chaque release un fichier .b3 contenant le hash BLAKE3 du binaire. Le stage fetcher vérifie le binaire téléchargé avant de le copier dans l'image finale :

```
# Télécharger binaire et sa signature
wget -O /usr/local/bin/b3sum "${BASE_URL}/b3sum_${B3SUM_ARCH}"
wget -O /tmp/b3sum.b3        "${BASE_URL}/b3sum_${B3SUM_ARCH}.b3"

# Auto-vérification : b3sum vérifie sa propre signature
chmod +x /usr/local/bin/b3sum
cd /usr/local/bin && b3sum --check /tmp/b3sum.b3

# Si le hash ne correspond pas → build échoue immédiatement
```

! **Chaîne de confiance**

b3sum se vérifie lui-même : le binaire téléchargé calcule son propre hash et le compare au fichier .b3 publié sur la même release GitHub. Si un seul bit a été altéré (attaque MITM, corruption réseau), le build échoue avec une erreur explicite.

3. Entrypoint et interface utilisateur

3.1 Rôle de l'entrypoint

Le script docker/entrypoint.sh est le point d'entrée unique du conteneur. Il dispatche les commandes passées en argument vers les scripts internes appropriés.

Commande	Script appelé	Description
compute <dossier> <base.b3>	src/integrity.sh compute	Calcule les hashes BLAKE3
verify <base.b3> [dossier]	src/integrity.sh verify	Vérifie l'intégrité
compare <old.b3> <new.b3>	src/integrity.sh compare	Compare deux bases
runner [pipeline.json]	runner.sh	Exécute un pipeline JSON
shell / bash	/bin/bash	Shell interactif (debug)
help	—	Affiche l'aide inline
version	—	Affiche les versions des outils

3.2 Option --quiet

L'option `--quiet` passée en premier argument est interceptée par l'entrypoint et transmise à integrity.sh. Elle supprime toute sortie terminal et propage l'exit code, ce qui la rend utilisable dans des pipelines CI/cron.

```
# --quiet doit être le premier argument
docker run --rm [...] hash_tool --quiet verify /bases/hashes.b3 /data

# Exit code 0 = OK, non-nul = FAILED ou ERREUR
# Résultats écrits dans /resultats malgré --quiet
```

3.3 Commande par défaut

Sans argument, le conteneur affiche l'aide (`CMD ["help"]`). Cela évite l'erreur silencieuse d'un conteneur qui démarre et s'arrête sans indication.

4. Volumes et gestion des données

4.1 Convention de montage

L'image définit quatre points de montage conventionnels. Cette convention est stable — les pipelines et scripts peuvent référencer ces chemins sans connaître les chemins réels de l'hôte.

Volume	Usage	Mode recommandé	Notes
/data	Données à hacher	:ro (lecture seule)	Jamais modifié par hash_tool
/bases	Fichiers .b3	Lecture/écriture	Écriture uniquement pour compute
/pipelines	Fichiers pipeline.json	:ro	Monter un fichier spécifique
/resultats	Résultats verify/compare	Lecture/écriture	RESULTATS_DIR=/resultats par défaut

> ¡ **Séparation données / bases**
> Les fichiers .b3 doivent être stockés sur un support DISTINCT des données vérifiées. Si /data et /bases pointent vers le même disque physique, une corruption du disque pourrait affecter les deux — rendant la vérification inopérante.
> Sur VeraCrypt : stocker les .b3 sur C: (stable), jamais sur la partition montée.

4.2 Variable d'environnement RESULTATS_DIR

La variable `RESULTATS_DIR` est définie à `/resultats` dans l'image. Elle peut être surchargée via `-e RESULTATS_DIR=/mon/chemin`. Le champ `"resultats"` dans pipeline.json surcharge cette variable pour un bloc compare spécifique.

```
# Surcharge globale via variable d'environnement
docker run --rm \
  -v /mes/resultats:/custom_res \
  -e RESULTATS_DIR=/custom_res \
  hash_tool verify /bases/hashes.b3

# Surcharge locale via pipeline.json (bloc compare uniquement)
"resultats": "/resultats/compare_jan_vs_fev"
```

5. Docker Compose

5.1 Structure du fichier

Le fichier docker-compose.yml définit trois services spécialisés partageant la même image, et une section x-volumes centralisant les chemins de montage de l'hôte.

Service	Usage	Démarrage
integrity	Commandes ponctuelles compute/verify/compare	docker compose run --rm integrity <cmd>
pipeline	Exécution de runner.sh avec pipeline.json	docker compose run --rm pipeline
cron	Vérification périodique (optionnel)	docker compose --profile cron up -d

5.2 Configuration des chemins

La section x-volumes permet de modifier tous les montages en un seul endroit. C'est le seul fichier à adapter lors d'un déploiement sur un nouvel environnement.

```
# docker-compose.yml — section à adapter
x-volumes:
  data:      &vol-data      /chemin/vers/donnees    # ex: /volume1/data sur
Synology
  bases:     &vol-bases     /chemin/vers/bases       # ex: /srv/bases sur
Debian
  pipelines: &vol-pipelines /chemin/vers/pipelines
  resultats: &vol-resultats /chemin/vers/resultats

# Les ancres YAML (*vol-data, etc.) propagent les chemins
# dans tous les services automatiquement
```

5.3 Profil cron

Le service `cron` est isolé derrière un profil Docker Compose (`profiles: ["cron"]`). Il n'est pas démarré par `docker compose up` par défaut, et nécessite l'option `--profile cron` . Cela évite un service fantôme actif en permanence.

6. Déploiement par environnement

6.1 Windows / WSL

Prérequis : Docker Desktop avec intégration WSL2 activée.

```
# Build depuis WSL
cd /mnt/c/Users/TonNom/Desktop/hash_tool
docker build -t hash_tool .

# Compute d'une partition VeraCrypt montée sur A:
docker run --rm \
  -v /mnt/a/dossier:/data:ro \
  -v /mnt/c/Users/TonNom/bases:/bases \
  -v /mnt/c/Users/TonNom/resultats:/resultats \
  hash_tool compute /data /bases/hashes_$(date +%Y-%m-%d).b3
```

> ¡ **Chemins VeraCrypt dans Docker**
> Les partitions VeraCrypt montées sur des lettres Windows (A:, I:, H:) sont accessibles depuis WSL sous /mnt/a/, /mnt/i/, /mnt/h/. Docker Desktop sur WSL2 hérite de ces montages. Adapter uniquement les chemins -v lors du changement de lettre de partition.

6.2 NAS Synology

Les NAS Synology récents (DS923+, DS720+) utilisent un processeur AMD64. Les modèles plus anciens (DS220j) utilisent ARM64 ou ARMv7. Vérifier avec : uname -m depuis SSH.

```
# Build ARM64 (NAS DS923+, DS720+)
docker build --platform linux/arm64 -t hash_tool:arm64 .

# Depuis SSH Synology ou via Portainer
docker run --rm \
  -v /volume1/data:/data:ro \
  -v /volume1/bases:/bases \
  -v /volume1/resultats:/resultats \
  hash_tool verify /bases/hashes.b3 /data
```

Modèle Synology	CPU	Architecture	Tag image
DS923+	AMD Ryzen R1600	x86_64 (amd64)	hash_tool
DS720+	Intel Celeron J4125	x86_64 (amd64)	hash_tool

Modèle Synology	CPU	Architecture	Tag image
DS220+	Intel Celeron J4025	x86_64 (amd64)	hash_tool
DS220j	Realtek RTD1296	aarch64 (arm64)	hash_tool:arm64
DS218j	Marvell ARMADA 385	armv7	hash_tool:armv7

6.3 Serveur Debian

Sur un serveur Debian, la méthode recommandée est le cron Docker pour une vérification nocturne automatique.

```
# /etc/cron.d/hash-integrity
0 3 * * * root docker run --rm \
  -v /srv/data:/data:ro \
  -v /srv/bases:/bases:ro \
  -v /srv/resultats:/resultats \
  hash_tool --quiet verify /bases/hashes.b3 /data \
  >> /var/log/hash-integrity.log 2>&1 \
  || mail -s "ALERTE intégrité $(hostname)" admin@example.com
```

7. Maintenance et évolution

7.1 Mise à jour de b3sum

La version de b3sum est contrôlée par `ARG B3SUM_VERSION=1.5.4` dans le Dockerfile. Pour mettre à jour, modifier uniquement cette ligne et rebuilder. La vérification cryptographique garantit l'intégrité du nouveau binaire.

```
docker build --build-arg B3SUM_VERSION=1.6.0 -t hash_tool .

# URLs des releases BLAKE3 officielles :
# https://github.com/BLAKE3-team/BLAKE3/releases
# Pattern binaire : b3sum_linux_<arch>_musl
# Pattern signature : b3sum_linux_<arch>_musl.b3
```

7.2 Build hors-ligne

Sur un NAS ou serveur sans accès à GitHub, le binaire b3sum peut être pré-téléchargé et copié directement dans le contexte de build.

```
# Sur une machine avec accès réseau
wget https://github.com/BLAKE3-
team/BLAKE3/releases/download/1.5.4/b3sum_linux_amd64_musl

# Modifier le Dockerfile pour COPY au lieu de wget
COPY b3sum_linux_amd64_musl /usr/local/bin/b3sum
RUN chmod +x /usr/local/bin/b3sum

# La vérification cryptographique reste applicable manuellement
```

7.3 Extension de l'image

Pour ajouter des fonctionnalités à l'image (notifications email, export vers un serveur, etc.), deux approches sont disponibles :

- Ajouter un module dans src/lib/ (ex. notify.sh) — sourcé par integrity.sh
- Ajouter un service dans docker-compose.yml — image étendue FROM hash_tool
- Créer un Dockerfile.extend héritant de l'image de base — versions séparées

Évolution	Approche recommandée	Impact sur l'image
Rapport PDF	Ajouter wkhtmltopdf dans l'image	Image +30 Mo

Évolution	Approche recommandée	Impact sur l'image
Notifications email	Service dédié dans docker-compose.yml	Aucun
Export S3	Script lib/export.sh + awscli dans image	Image +20 Mo
Interface web	Service séparé, hash_tool en dépendance	Aucun

8. Référence rapide

8.1 Commandes essentielles

```
# Build standard
docker build -t hash_tool .

# Build ARM64
docker build --platform linux/arm64 -t hash_tool:arm64 .

# Aide
docker run --rm hash_tool help

# Versions
docker run --rm hash_tool version

# Compute
docker run --rm -v /data:/data:ro -v /bases:/bases \
  hash_tool compute /data /bases/hashes_$(date +%Y-%m-%d).b3

# Verify
docker run --rm -v /data:/data:ro -v /bases:/bases:ro -v /res:/resultats \
  hash_tool verify /bases/hashes.b3 /data

# Compare
docker run --rm -v /bases:/bases:ro -v /res:/resultats \
  hash_tool compare /bases/old.b3 /bases/new.b3

# Pipeline
docker run --rm -v /data:/data:ro -v /bases:/bases \
  -v /res:/resultats -v /pipe/pipeline.json:/pipelines/pipeline.json:ro \
  hash_tool runner

# Debug interactif
docker run --rm -it -v /data:/data hash_tool shell
```

8.2 Fichiers du projet

Fichier	Rôle
Dockerfile	Build multi-stage Alpine + b3sum + jq

Fichier	Rôle
.dockerignore	Exclusions du contexte de build
docker/entrypoint.sh	Dispatcher des commandes Docker
docker-compose.yml	Orchestration 3 services
docs/docker.md	Guide d'utilisation détaillé
src/integrity.sh	Logique métier compute/verify/compare
src/lib/report.sh	Génération rapport HTML
runner.sh	Exécuteur de pipeline JSON
pipelines/pipeline.json	Pipeline de test local
pipelines/pipeline-full.json	Pipeline VeraCrypt multi-disques