HashDetectMalware

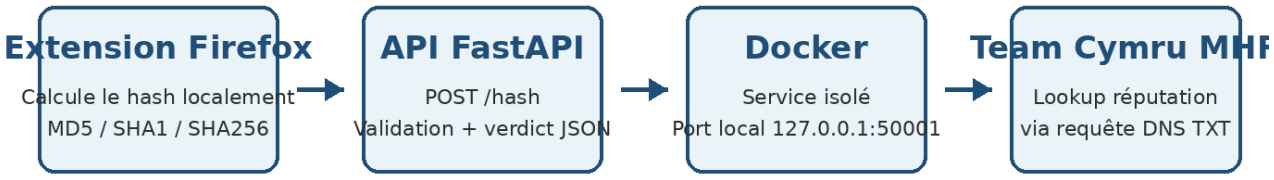
Micro-service HTTP de lookup de réputation par hash

Documentation technique - Sadek Adel

Version 1.0 - Mai 2026

Ce document présente le besoin, le fonctionnement, l architecture, le déploiement Docker et les tests du micro-service HashDetectMalware by Sadek Adel.

Architecture fonctionnelle - HashDetectMalware



Flux : fichier local -> hash uniquement -> requête API -> lookup MHR -> réponse JSON exploitable par

Sommaire

1. Introduction

HashDetectMalware by Sadek Adel est un micro-service HTTP conçu pour recevoir un hash de fichier, par exemple MD5, SHA1 ou SHA256, puis répondre rapidement si ce hash est connu comme malveillant ou non.

Le projet a été pensé pour un cas d usage concret : une extension Mozilla Firefox capable de calculer localement l empreinte d un fichier, puis d interroger une API légère sans envoyer le fichier complet au serveur.

Idée centrale du projet
Au lieu d envoyer un fichier volumineux vers un moteur antivirus distant, on envoie uniquement son empreinte cryptographique. Le service effectue ensuite un lookup de réputation par hash et renvoie un verdict exploitable.

2. Besoin couvert par la solution

Dans un scénario classique de scan antivirus par API, le client doit transmettre le fichier entier au serveur. Cette approche peut fonctionner pour de petits fichiers, mais elle devient rapidement moins adaptée lorsque les fichiers sont lourds, par exemple 800 Mo.

Element	Description
Limiter la bande passante	Seul le hash est envoyé, pas le fichier complet.
Réduire la latence	La requête est légère et la réponse est rapide.
Simplifier le client	L extension n a pas besoin de gérer un upload volumineux.
Améliorer la confidentialité opérationnelle	Le contenu du fichier n est pas transmis au serveur de lookup.
Préparer une intégration navigateur	Le JSON renvoyé est simple à exploiter dans une extension Firefox.

Le projet répond donc à un besoin de performance, de simplicité et d intégration, plutôt qu à un besoin d analyse complète du contenu d un fichier.

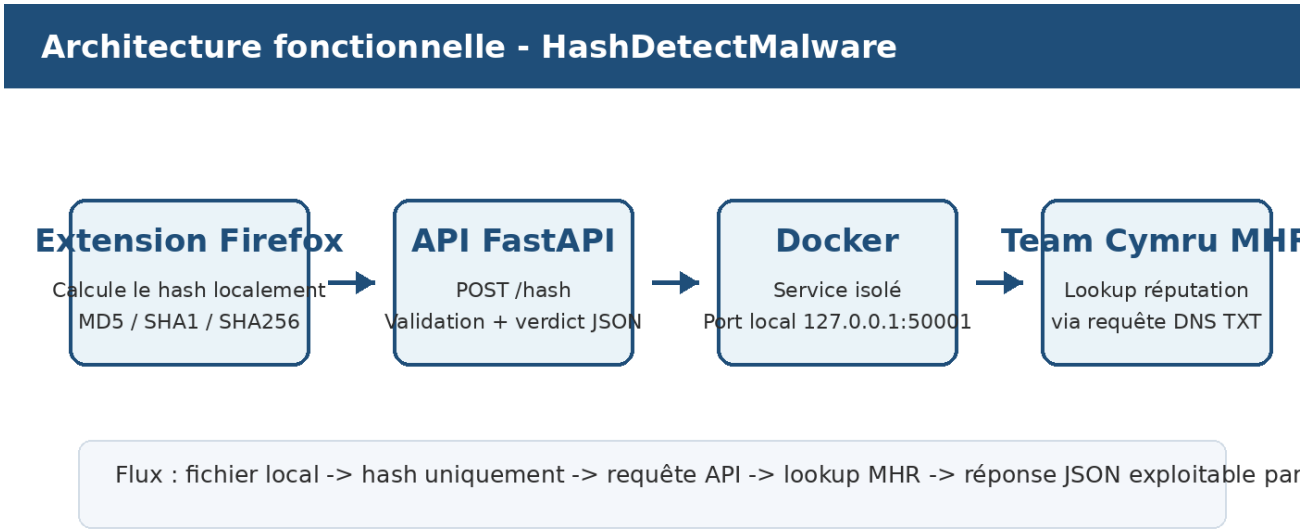
3. Principe général et cas d usage

HashDetectMalware fonctionne comme une interface intermédiaire entre un client léger et une source de réputation par hash. Le client envoie un hash et l algorithme utilisé. Le service valide ces données, interroge Team Cymru Malware Hash Registry, puis renvoie une réponse JSON structurée.

Flux fonctionnel

1. L extension Firefox calcule localement le hash du fichier.
2. Elle envoie une requête HTTP POST vers l endpoint /hash.
3. Le micro-service vérifie le format du hash et l algorithme demandé.
4. Le service construit la requête DNS attendue par Team Cymru MHR.
5. La source de réputation renvoie ou non une correspondance.
6. L API retourne un verdict JSON : malware, clean ou unknown selon le contexte.

4. Architecture fonctionnelle



L architecture est volontairement simple : un micro-service FastAPI est exécuté dans un conteneur Docker, exposé localement sur le port 50001. Un reverse proxy comme Nginx peut ensuite publier l API proprement, par exemple derrière un nom de domaine ou un chemin dédié.

Element	Description
Client	Extension Mozilla Firefox, script Python, interface web ou outil curl.
API	FastAPI avec endpoint /health et endpoint /hash.
Conteneur	Docker avec image Python 3.11 slim.
Source réputation	Team Cymru Malware Hash Registry via DNS TXT.
Réponse	JSON structuré avec found, hash, algo, verdict et source.

5. Description des fichiers du projet

Element	Description
app.py	Code principal de l API FastAPI : validation, lookup DNS, réponse JSON.
dockerfile / Dockerfile	Recette de construction de l image Docker. Idéalement, renommer en Dockerfile ou utiliser docker build -f dockerfile.
docker-compose.yml	Déploiement du service avec port local, restart policy et DNS publics.

6. Fonctionnement de l API

Endpoint de santé

```
GET /health

Réponse attendue :
{
  "status": "ok"
}
```

Endpoint principal

```
POST /hash
Content-Type: application/json

Body :
{
  "hash": "44d88612fea8a8f36de82e1278abb02f",
  "algo": "md5"
}
```

Validation des entrées

L API refuse les entrées qui ne correspondent pas strictement au format attendu. Cela évite d envoyer des requêtes inutiles vers la source de réputation et protège le service contre des entrées incohérentes.

Element	Description
md5	32 caractères hexadécimaux.
sha1	40 caractères hexadécimaux.
sha256	64 caractères hexadécimaux. Le hash est découpé en deux segments DNS de 32 caractères pour la requête MHR.

Illustration du code FastAPI

```
HASH_PATTERN = {
    "md5": 32,
    "sha1": 40,
    "sha256": 64,
}

def is_valid_hash(h: str, algo: str) -> bool:
    if algo not in HASH_PATTERN or len(h) != HASH_PATTERN[algo]:
        return False
    return bool(re.fullmatch(r"[0-9a-fA-F]+", h))
```

Cette partie du code garantit que l algorithme demandé est autorisé et que la longueur du hash correspond bien au format attendu.

```
if algo == "sha256":
    lookup = f"{h[:32]}.{h[32:]}.hash.cymru.com"
else:
    lookup = f"{h}.hash.cymru.com"

answers = DSPYNSER.resolve(lookup, "TXT")
```

Cette logique construit la requête DNS utilisée pour interroger Team Cymru MHR. Le service exploite ensuite la réponse TXT pour extraire la date de dernière observation et le taux de détection.

7. Déploiement Docker et Docker Compose

Le conteneur Docker permet d exécuter l API dans un environnement isolé, reproductible et facile à déployer. Cette approche est adaptée à un VPS, un lab Docker, ou une intégration derrière Nginx.

Dockerfile

```
FROM python:3.11-slim
WORKDIR /app
RUN pip install --no-cache-dir fastapi uvicorn[standard] dnspython
COPY app.py .
EXPOSE 50001
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "50001"]
```

docker-compose.yml

```
services:
  mhr-api:
    build: ./
    container_name: mhr-api
    restart: unless-stopped
    ports:
      - "127.0.0.1:50001:50001"
    dns:
      - 8.8.8.8
      - 1.1.1.1
```

Commandes utiles

```
# Construire l image locale
docker build -t hash_detect_malware_by_sadek_adel:1.0 .

# Construire avec un nom compatible Docker Hub
docker build -t adelsadek95/hash_detect_malware_by_sadek_adel:1.0 .

# Si le fichier s appelle dockerfile en minuscule
docker build -f dockerfile -t adelsadek95/hash_detect_malware_by_sadek_adel:1.0 .

# Envoyer sur Docker Hub
docker push adelsadek95/hash_detect_malware_by_sadek_adel:1.0

# Lancer via Docker Compose
docker compose up -d
```

Point important
Pour pousser une image sur Docker Hub, le nom local doit inclure le namespace du compte : adelsadek95/nom_image:tag. Sinon Docker considère que ce n est pas la même image.

8. Exemples de requêtes et réponses

Les tests suivants montrent les comportements principaux du service : hash malveillant connu, hash EICAR, hash invalide et hash non trouvé.

Exemple 1 - Hash reconnu comme malveillant

```
curl -X POST http://127.0.0.1:50001/hash \
-H "Content-Type: application/json" \
-d '{"hash":"8a62d103168974fba9c61edab336038c","algo":"md5"}'

Réponse simplifiée :
{
  "found": true,
  "hash": "8a62d103168974fba9c61edab336038c",
  "algo": "md5",
  "last_seen": "... UTC",
  "av_hit_rate":  ...,
  "verdict": "malware",
  "source": "team-cymru-mhr"
}
```

Exemple 2 - Hash EICAR reconnu

```
curl -X POST http://127.0.0.1:50001/hash \
-H "Content-Type: application/json" \
-d '{"hash":"44d88612fea8a8f36de82e1278abb02f","algo":"md5"}'

Réponse simplifiée :
{
  "found": true,
  "hash": "44d88612fea8a8f36de82e1278abb02f",
  "algo": "md5",
  "verdict": "malware",
  "source": "team-cymru-mhr"
}
```

Exemple 3 - Hash invalide

```
curl -X POST http://127.0.0.1:50001/hash \
-H "Content-Type: application/json" \
-d '{"hash":"44d88612fea8a8f36de82e127gfgbb02f","algo":"md5"}'

Réponse :
{
  "detail": "Invalid hash or algo. algo must be one of ['md5', 'sha1', 'sha256']"
}
```

Exemple 4 - Hash non trouvé

```
curl -X POST http://127.0.0.1:50001/hash \
-H "Content-Type: application/json" \
-d '{"hash":"d378bffb70923139d6a4f546864aa61c","algo":"md5"}'

Réponse :
{
  "found": false,
  "hash": "d378bffb70923139d6a4f546864aa61c",
  "algo": "md5",
  "verdict": "clean",
  "source": "team-cymru-mhr"
}
```

9. Intégration avec une extension Mozilla Firefox

Le cas d usage principal est l intégration avec une extension Firefox. L extension n envoie pas le binaire au serveur. Elle calcule seulement le hash localement, puis interroge l API.

Element	Description
Côté navigateur	Calcul local du hash du fichier téléchargé ou sélectionné.
Côté API	Réception du hash, validation stricte, lookup MHR.
Côté utilisateur	Affichage d un badge, d une alerte ou d une indication de risque.
Côté sécurité	Aucun upload du fichier complet n est nécessaire.

Cette architecture est particulièrement pertinente pour une extension légère, car elle réduit la complexité réseau tout en donnant un verdict rapide sur les hashes connus.

10. Intérêt technique, limites et évolutions possibles

Intérêt technique

- Pas d upload de gros fichiers.
- Réponse JSON simple et exploitable.
- Micro-service isolé dans Docker.
- Facile à placer derrière Nginx ou un reverse proxy.
- Compatible avec une extension, un script, une interface web ou un pipeline de sécurité.

Limites actuelles

- Un hash inconnu ne prouve pas que le fichier est sain.
- Le service ne réalise pas d analyse dynamique ni de scan complet du contenu.
- Le verdict dépend de la source de réputation interrogée au moment de la requête.
- Une variante modifiée d un malware peut avoir un hash différent et ne pas être détectée par lookup simple.

Evolutions possibles

- Ajouter un cache local pour éviter les requêtes répétées.
- Ajouter plusieurs sources de réputation et agréger les résultats.
- Ajouter une authentification API key pour limiter l usage public.
- Ajouter une page web de test simple pour coller un hash et lire le verdict.
- Ajouter des logs structurés et une supervision via Docker logs, Prometheus ou Grafana.

11. Disponibilité GitHub et Docker Hub

Le projet est pensé pour être versionné sur GitHub et publié sous forme d image sur Docker Hub. Cela permet de partager le code, de documenter l évolution du service et de déployer rapidement l API sur une autre machine Docker.

Element	Description
GitHub	Dépôt du code source : app.py, dockerfile / Dockerfile, docker-compose.yml, README et documentation.
Docker Hub	Image Docker publiable sous le namespace adelsadek95/hash_detect_malware_by_sadek_adel:1.0.
Déploiement	L image peut être tirée puis lancée via Docker Compose ou docker run.

```
# Exemple de récupération depuis Docker Hub
docker pull adelsadek95/hash_detect_malware_by_sadek_adel:1.0

# Exemple de lancement direct
docker run -d --name mhr-api \
  --restart unless-stopped \
  -p 127.0.0.1:50001:50001 \
  adelsadek95/hash_detect_malware_by_sadek_adel:1.0
```

12. Conclusion

HashDetectMalware by Sadek Adel est une solution légère, claire et adaptée à un besoin précis : vérifier rapidement la réputation d un hash sans envoyer le fichier complet au serveur.

Le projet montre une logique propre de micro-service : une API simple, une validation stricte, une intégration Docker, une source de réputation externe et des réponses JSON facilement exploitables par une extension Mozilla Firefox ou un autre client.

Synthèse finale
Le service ne remplace pas un antivirus complet, mais il répond très bien à un besoin de lookup rapide par hash. Il constitue une base solide pour construire une extension navigateur ou un outil de pré-vérification de fichiers avant analyse plus poussée.