

# Minio S3

## configuration MinIO + service Graph (mode proxy)

Principe :

- MinIO expose S3 ;
- Un service écrit en Python reçoit les requêtes S3 et les traduit en appels Graph vers SharePoint
- Authentification via certificat

## Correspondance S3 <=> SharePoint + gestion DELETE / RENAME

S3 (MinIO)	SharePoint
Bucket	Dossier dans Documents
Objet	Fichier
Prefix	Sous-dossier
DELETE	Suppression
RENAME	DELETE + CREATE
PUT	Upload
GET	Download

S3 ne supporte pas le "rename" natif. MinIO envoie :

- ObjectRemoved
- ObjectCreated
- Le proxy traite automatiquement le renommage.
- Aucun code spécifique requis
- Conforme au comportement S3 officiel

### Exemple :

```
s3://dossier/fichier.pdf -> Documents/dossier/fichier.pdf
```

- utilisation de 2 conteneurs :
  - conteneur 1 : MinIO (S3)
  - conteneur 2 : Un service Graph qui synchronise vers SharePoint
- certificat :
  - certificat .cer
  - clé privée .key au format PKCS#8

## sharepoint-proxy - en Python

Ce service :

- reçoit les events S3 envoyés par MinIO (webhook)
- traduit vers Microsoft Graph
- manipule les fichiers dans SharePoint / Documents

Un webhook est un mécanisme de notification automatique entre applications.

Un webhook est :

- une URL exposée par une application
- qu'une autre application appelle automatiquement (HTTP POST/GET)
- quand un événement précis se produit

Contrairement à une API classique où l'on demande l'information, avec un webhook l'information vient à toi toute seule.

```
MinIO
├─ webhook
│   └─ appelle sharepoint-proxy
│       └─ appelle Microsoft Graph
```

### └─ modifie SharePoint

- Quand un fichier est envoyé à minIO, MinIO déclenche automatiquement un webhook :

```
POST http://sharepoint-proxy:8080/s3event
Content-Type: application/json
```

- Contenu envoyé (payload)

```
{
  "EventName": "s3:ObjectCreated:Put",
  "Key": "dossier/fichierr.pdf"
}
```

## Webhook Python

- Arborescence

```
sharepoint-proxy/
├─ main.py
├─ graph.py
├─ requirements.txt
└─ README.md
```

- requirements.txt

### requirements.txt

```
flask
requests
msal
```

- graph.py - Authentification + appels Graph

### graph.py

```
import os
import msal
import requests

TENANT_ID = os.getenv("TENANT_ID")
CLIENT_ID = os.getenv("CLIENT_ID")
CERT_PATH = os.getenv("CERT_PATH")
KEY_PATH = os.getenv("KEY_PATH")
SITE_PATH = os.getenv("SITE_PATH")

AUTHORITY = f"https://login.microsoftonline.com/{TENANT_ID}"
SCOPE = ["https://graph.microsoft.com/.default"]

def get_token():
    app = msal.ConfidentialClientApplication(
        CLIENT_ID,
        authority=AUTHORITY,
        client_credential={
            "private_key": open(KEY_PATH).read(),
            "public_certificate": open(CERT_PATH).read()
        }
    )
    token = app.acquire_token_for_client(scopes=SCOPE)
    if "access_token" not in token:
        raise RuntimeError(f"Token error: {token}")
    return token["access_token"]

def graph_request(method, url, headers=None, data=None):
    token = get_token()
    h = {
        "Authorization": f"Bearer {token}",
    }
```

```

    "Content-Type": "application/json"
}
if headers:
    h.update(headers)
r = requests.request(method, url, headers=h, data=data)
r.raise_for_status()
return r.json() if r.text else None

```

- main.py - Webhook MinIO → SharePoint

Pour sécuriser le webhook de MinIO vers le Proxy, utilisation d'un secret partagé (token) et vérification dans **main.py** du secret. Voir plus loin pour la configuration de la signature dans le Webhook.

Gérer les fichiers volumineux :

Microsoft Graph refuse :

- les PUT /content simples
- dès que le fichier dépasse ~4-10 Mo (limite variable)

Pour des fichiers > 250 Mo (vidéos, archives, sauvegardes)

- upload en session + chunks obligatoire

Principe Graph (officiel)

- Créer une upload session
- Envoyer le fichier par blocs (chunks)
- Graph reconstruit le fichier côté SharePoint

main.py

```

from flask import Flask, request, abort
import os, requests

from graph import graph_request

TENANT_ID = os.getenv("TENANT_ID")
SITE_PATH = os.getenv("SITE_PATH")
GRAPH_BASE = "https://graph.microsoft.com/v1.0"

WEBHOOK_SECRET = os.getenv("WEBHOOK_SECRET")
if not WEBHOOK_SECRET:
    raise RuntimeError("WEBHOOK_SECRET not set")

# Détection de la taille du fichier / Upload thresholds
MAX_SIMPLE_UPLOAD = 4 * 1024 * 1024 # 4 MB
CHUNK_SIZE = 10 * 1024 * 1024 # 10 MB

app = Flask(__name__)

# -----
# Sécurité webhook MinIO
# -----
def verify_minio_webhook(req):
    auth = req.headers.get("Authorization")

    if not auth or not auth.startswith("Bearer "):
        abort(401, "Missing Authorization header")

    token = auth[len("Bearer "):].strip()

    if token != WEBHOOK_SECRET:
        abort(403, "Invalid MinIO webhook token")

```

```
# -----
# SharePoint drive
# -----
def get_drive_id():
    site = graph_request(
        "GET",
        f"{GRAPH_BASE}/sites/{SITE_PATH}"
    )
    drive = graph_request(
        "GET",
        f"{GRAPH_BASE}/sites/{site['id']}/drive"
    )
    return drive["id"]

# INITIALISATION AU DÉMARRAGE
print("Initializing SharePoint drive...")
DRIVE_ID = get_drive_id()
print(f"Drive ID initialized: {DRIVE_ID}")

# -----
# Upload helpers
# -----
def create_upload_session(sp_path):
    res = graph_request(
        "POST",
        f"{GRAPH_BASE}/drives/{DRIVE_ID}/root:{sp_path}:createUploadSession",
        json={
            "item": {
                "@microsoft.graph.conflictBehavior": "replace"
            }
        }
    )
    return res["uploadUrl"]

def upload_simple(sp_path, data):
    graph_request(
        "PUT",
        f"{GRAPH_BASE}/drives/{DRIVE_ID}/root:{sp_path}:content",
        data=data
    )

def upload_chunked(sp_path, file_path):
    upload_url = create_upload_session(sp_path)
    size = os.path.getsize(file_path)

    with open(file_path, "rb") as f:
        start = 0
        while start < size:
            chunk = f.read(CHUNK_SIZE)
            end = start + len(chunk) - 1

            headers = {
                "Content-Length": str(len(chunk)),
                "Content-Range": f"bytes {start}-{end}/{size}"
            }

            r = requests.put(upload_url, headers=headers, data=chunk)
            r.raise_for_status()
            start += len(chunk)

# -----
# Event handlers
# -----
def upload_object(key):
    print(f"Uploading {key} to SharePoint", flush=True)
```

```

# TODO : ici télécharger le fichier depuis MinIO vers /tmp
file_path = f"/tmp/{os.path.basename(key)}"

# (temporaire pour tests)
with open(file_path, "wb") as f:
    f.write(b"TEST")

size = os.path.getsize(file_path)

if size <= MAX_SIMPLE_UPLOAD:
    with open(file_path, "rb") as f:
        upload_simple(key, f.read())
else:
    upload_chunked(key, file_path)

os.remove(file_path)

def delete_object(key):
    print(f"Deleting {key} from SharePoint", flush=True)
    graph_request(
        "DELETE",
        f"{GRAPH_BASE}/drives/{DRIVE_ID}/root:{key}"
    )

# -----
# Webhook endpoint
# -----
@app.route("/s3event", methods=["POST"])
def s3_event():
    verify_minio_webhook(request)
    event = request.get_json()
    record = event["Records"][0]
    event_name = record["eventName"]
    key = record["s3"]["object"]["key"]

    if event_name.startswith("s3:ObjectCreated"):
        upload_object(key)
    elif event_name.startswith("s3:ObjectRemoved"):
        delete_object(key)

    return "", 204

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)

```

## Docker compose

[docker-compose.yml](#)

```

services:
  minio:
    image: minio/minio:latest
    container_name: minio
    command: server /data --console-address ":9001"
    env_file:
      - .env
    volumes:
      - minio-data:/data
    ports:
      - "9000:9000"
      - "9001:9001"

  sharepoint-proxy:
    build: ./sharepoint-proxy
    container_name: sharepoint-proxy
    volumes:
      - ./sp-proxy:/app

```

```
- ./certs:/certs:ro
working_dir: /app
command: python main.py
env_file:
- .env
depends_on:
- minio
ports:
- "8080:8080"

volumes:
minio-data:
```

Utiliszer gunicorn en production

- fichier .env

[.env](#)

```
MINIO_ROOT_USER=admin
MINIO_ROOT_PASSWORD=motdepasse

TENANT_ID=<ID du tenant>
CLIENT_ID=<ID de l'application dans Entra ID>
SITE_PATH=tenant.sharepoint.com:/sites/site
CERT_PATH=/certs/minio-sharepoint.pem
KEY_PATH=/certs/minio-sharepoint.key

AUTHORITY=https://login.microsoftonline.com/<ID du tenant>
GRAPH_SCOPE=https://graph.microsoft.com/.default

WEBHOOK_SECRET="super-secret-minio"

DOCUMENT_LIBRARY="Documents"
```

- vérifier que les variables d'environnement ont bien renseignées dans le conteneur

```
docker compose exec sharepoint-proxy env
```

- ajouter un Dockerfile dans le sous-dossier sharepoint-proxy pour inclure le module Flash à l'image Python

[Dockerfile](#)

```
FROM python:3.12-slim

WORKDIR /app

# Installer les dépendances système minimales
RUN apt-get update && apt-get install -y \
    ca-certificates \
    && rm -rf /var/lib/apt/lists/*

# Copier les dépendances Python
COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

# Copier le code
COPY . .

EXPOSE 8080

CMD ["python", "main.py"]
```

- construire l'image

```
docker compose build sharepoint-proxy
```

## Déclarer le webhook avec mc

MinIO n'active pas automatiquement le Webhook : il faut le déclarer avec mc.

### Installer mc

La mc signifie MinIO Client : C'est l'outil en ligne de commande officiel de MinIO.

- installer mc dans la VM Debian qui exécute le Docker Compose de minio

```
curl -L https://dl.min.io/client/mc/release/linux-amd64/mc -o mc
```

- Rendre le binaire exécutable

```
chmod +x mc
```

- Déplacer dans un dossier du PATH

```
mv mc /usr/local/bin
```

- Vérifier l'installation

```
mc --version
```

- Définir l'alias minio

```
mc alias set minio http://localhost:9000 minioadmin minioadmin123  
=> résultat attendu  
Added `minio` successfully
```

- Vérifier les alias

```
mc alias list  
=> résultat attendu  
gcs  
  URL      : https://storage.googleapis.com  
  AccessKey : YOUR-ACCESS-KEY-HERE  
  SecretKey : YOUR-SECRET-KEY-HERE  
  API      : S3v2  
  Path     : dns  
  Src      : /root/.mc/config.json  
  
local  
  URL      : http://localhost:9000  
  AccessKey :  
  SecretKey :  
  API      :  
  Path     : auto  
  Src      : /root/.mc/config.json  
  
minio  
  URL      : http://localhost:9000  
  AccessKey : admin  
  SecretKey : Orpheus87  
  API      : s3v4  
  Path     : auto  
  Src      : /root/.mc/config.json  
  
play  
  URL      : https://play.min.io  
  AccessKey : Q3AM3UQ867SPQQA43P2F  
  SecretKey : zuf+tfteSlsWRu7BJ86wekitnifILbZam1KYY3TG  
  API      : S3v4  
  Path     : auto  
  Src      : /root/.mc/config.json
```

```
s3
URL      : https://s3.amazonaws.com
AccessKey : YOUR-ACCESS-KEY-HERE
SecretKey : YOUR-SECRET-KEY-HERE
API      : S3v4
Path     : dns
Src      : /root/.mc/config.json
```

## Créer le webhook

Dans Docker, il faut utiliser le **nom du service Docker** et pas le nom DNS réseau de la VM où s'exécute le conteneur.

Si le proxy est sur un serveur distinct et non dans Docker, il faut alors utiliser le nom DNS du serveur qui exécute le proxy.

Pour sécuriser le webhook MinIO vers le Proxy, utilisation d'un header statique contenant un token pour empêcher toute requête non légitime d'appeler /s3event.

- Configurer le webhook avec le token :

```
mc admin config set minio notify_webhook:sharepoint \
  endpoint="http://sharepoint-proxy:8080/s3event" \
  auth_token="$WEBHOOK_SECRET"
```

=> résultat attendu :

Successfully applied new settings.

Please restart your server 'mc admin service restart minio'.

- sharepoint = nom logique du webhook
- sharepoint-proxy = nom Docker du conteneur (pas le nom DNS de la VM)
- Redémarrer MinIO

```
mc admin service restart minio
```

- Vérifier la configuration
  - identifier le nom de ton alias MinIO

```
mc alias list
```

=> revoie notamment

```
minio
URL      : http://localhost:9000
```

- afficher TOUTE la configuration MinIO

```
mc admin config get minio
```

- afficher la configuration des webhook

```
mc admin config get minio notify_webhook
```

=> doit renvoyer

```
# MINIO_NOTIFY_WEBHOOK_ENABLE=on
notify_webhook enable=off endpoint= auth_token= queue_limit=0 queue_dir= client_cert= client_key=
notify_webhook:sharepoint endpoint=http://sharepoint-proxy:8080/s3event auth_token= queue_limit=1000
queue_dir=/tmp/notify client_cert= client_key=
```

Commentaires :

- le 1er notify\_webhook est le webhook global et est désactivé.
- Le webhook nommé **sharepoint** est bien configuré avec le endpoint <http://sharepoint-proxy:8080/s3event>.
- afficher la configuration d'un webhook

```
mc admin config get minio notify_webhook:sharepoint
```

=> doit renvoyer

```
notify_webhook:sharepoint endpoint=http://sharepoint-proxy:8080/s3event auth_token= queue_limit=1000
queue_dir=/tmp/notify client_cert= client_key=
```

- Supprimer un webhook existant

```
mc admin config reset minio notify_webhook:sharepoint
=> doit renvoyer
'notify_webhook:sharepoint' is successfully reset.
Please restart your server with `mc admin service restart minio`.
```

- Lier le bucket aux événements afin que MinIO envoie les données

```
mc event add minio/test arn:minio:sqs::sharepoint:webhook --event put,delete
=> doit renvoyer
Successfully added arn:minio:sqs::sharepoint:webhook
```

Commentaires :

- minio/test → bucket concerné
- sp → webhook que tu as configuré
- put → upload / création
- delete → suppression
- Vérifier que c'est bien pris en compte

```
mc event list minio/test
=> doit renvoyer
arn:minio:sqs::sharepoint:webhook s3:ObjectCreated:*,s3:ObjectRemoved:* Filter:
```

- supprimer l'événement sharepoint du bucket test

```
mc event remove minio/test --force
```

## Lancement et test

- lancer le docker compose

```
docker compose up -d
=> on doit voir
Initializing SharePoint drive...
Drive ID initialized: xxxx
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8080
```

- vérifier les logs du conteneur sharepoint-proxy

```
docker compose logs -f sharepoint-proxy
```

## Exemples de test

- Upload

```
mc cp devoir.pdf minio/dossier/fichier.pdf
```

- Vérifier que MinIO envoie réellement le webhook

```
docker compose exec minio curl -v http://sharepoint-proxy:8080/s3event
=> doit renvoyer code HTTP 204 ou 405 (selon ton proxy) : Connexion OK
```

- dans SharePoint apparaît le fichier Documents/fichier.pdf
- Suppression

```
mc rm minio/dossier/fichier.pdf
```

- Fichier supprimé dans SharePoint

## Gérer les fichiers volumineux (Upload Microsoft Graph en chunks > 250 Mo)

Microsoft Graph refuse :

- les PUT /content simples
- dès que le fichier dépasse ~4-10 Mo (limite variable)
- Pour des fichiers > 250 Mo (vidéos, archives, sauvegardes) upload en session + chunks obligatoire

Principe Graph (officiel)

- Créer une upload session
- Envoyer le fichier par blocs (chunks)
- Graph reconstruit le fichier côté SharePoint

### Détection de la taille

```
import os

MAX_SIMPLE_UPLOAD = 4 * 1024 * 1024 # 4 MB

size = os.path.getsize(file_path)

if size <= MAX_SIMPLE_UPLOAD:
    upload_simple(path, file_path)
else:
    upload_chunked(path, file_path)
```

- Création de la session d'upload

```
def create_upload_session(drive_id, sp_path):
    return graph_request(
        "POST",
        f"{GRAPH_BASE}/drives/{drive_id}/root:{sp_path}/createUploadSession",
        json={
            "item": {
                "@microsoft.graph.conflictBehavior": "replace"
            }
        }
    )["uploadUrl"]
```

- Upload par chunks (robuste et repreneable)

```
import requests

def upload_chunked(sp_path, local_file):
    upload_url = create_upload_session(DRIVE_ID, sp_path)

    chunk_size = 10 * 1024 * 1024 # 10 MB
    size = os.path.getsize(local_file)

    with open(local_file, "rb") as f:
        start = 0
        while start < size:
            data = f.read(chunk_size)
            end = start + len(data) - 1

            headers = {
                "Content-Length": str(len(data)),
                "Content-Range": f"bytes {start}-{end}/{size}"
            }

            r = requests.put(upload_url, headers=headers, data=data)
            r.raise_for_status()

            start += chunk_size
```

From:  
[/ - Les cours du BTS SIO](#)

Permanent link:  
[/doku.php/systeme/documenso/minio](#)

Last update: **2026/04/11 16:25**

