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
gunicorn
```

Utilisatio de gunicorn (production).

Gunicorn est le serveur d'exécution Python de référence pour les applications Flask en production ; il remplace le serveur de développement Flask, non fiable et non sécurisé.

```
gunicorn -w 2 -b 0.0.0.0:8080 main:app
```

Commentaires :

- -w 2 → 2 workers Python
- -b → bind sur 0.0.0.0:8080
- main:app → ton application Flask

- 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

Gestion de plusieurs dossiers dans une seule équipe SharePoint :

- dossiers automatiquement créés dans Sharepoint
- aucun droit d'écriture manuel côté SharePoint
- traçabilité MinIO → SharePoint

Structure :

```
Documents
├── administration
│   ├── gestion
│   ├── scolaires
│   └── administration
├── bts-sio
│   ├── stages
│   ├── ccf
│   └── livrets
```

```
├── bts-mco
│   ├── stages
│   ├── ccf
│   └── livrets
```

Mapping pédagogique MinIO → SharePoint (préfixe MinIO = dossier SharePoint)

MinIO (bucket pedagogie)	SharePoint (Documents)
administration/gestion/	Documents/administration/gestion/
bts-sio/	Documents/bts-sio/
bts-sio/stages/	Documents/bts-sio/stages/

main.py

```python
from flask import Flask, request, abort
import os
import requests
import subprocess
import time
import json
from datetime import datetime, timezone
from urllib.parse import unquote

from graph import graph_request

# ============================================================
# CONFIGURATION GLOBALE
# ============================================================

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

MINIO_BUCKET = "lycee"
TMP_DIR = "/tmp"

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

# Upload thresholds
MAX_SIMPLE_UPLOAD = 4 * 1024 * 1024      # 4 MB
CHUNK_SIZE = 10 * 1024 * 1024            # 10 MB

# Graph retry / backoff
MAX_RETRIES = 5
INITIAL_BACKOFF = 1.0  # seconds

app = Flask(__name__)

print("=== SHAREPOINT PROXY STARTED ===", flush=True)

# ============================================================
# LOGGING STRUCTURÉ (JSON)
# ============================================================

def log(event, **data):
    print(json.dumps({
        "timestamp": datetime.now(timezone.utc).isoformat(),
        "service": "sharepoint-proxy",
        "event": event,
        **data
    }), flush=True)

# ============================================================
# SÉCURITÉ WEBHOOK MINIO (token statique)
# ============================================================

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")

# =====================================================
# INIT 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"]

log("startup", message="Initializing SharePoint drive")
DRIVE_ID = get_drive_id()
log("startup", drive_id=DRIVE_ID)

# =====================================================
# GRAPH REQUEST RETRY / BACKOFF
# =====================================================

def graph_request_with_retry(method, url, **kwargs):
    backoff = INITIAL_BACKOFF

    for attempt in range(1, MAX_RETRIES + 1):
        try:
            return graph_request(method, url, **kwargs)
        except Exception as e:
            if attempt == MAX_RETRIES:
                log("graph_error", attempt=attempt, error=str(e))
                raise

            log("graph_retry", attempt=attempt, backoff=backoff, error=str(e))
            time.sleep(backoff)
            backoff *= 2

# =====================================================
# MAPPING PÉDAGOGIQUE
# =====================================================

def map_mapping_path(key: str) -> str | None:
    key = key.lstrip("/").split("?")[0]

    ALLOWED_PREFIXES = [
        "administration/",
        "bts-sio/",
        "bts-mco/"
    ]

    for prefix in ALLOWED_PREFIXES:
        if key.startswith(prefix):
            return key

    return None

# =====================================================
# DOWNLOAD DEPUIS MINIO
# =====================================================

def download_from_minio(key: str) -> str:
    local_path = os.path.join(TMP_DIR, os.path.basename(key))

    cmd = [
        "mc",
        "cp",
        f"minio/{MINIO_BUCKET}/{key}",
```

```
        local_path
    ]

    subprocess.check_call(cmd)
    return local_path

# ======================================================
# UPLOAD SHAREPOINT
# ======================================================

def create_upload_session(sp_path: str) -> str:
    res = graph_request_with_retry(
        "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: str, data: bytes):
    graph_request_with_retry(
        "PUT",
        f"{GRAPH_BASE}/drives/{DRIVE_ID}/root:/{sp_path}:/content",
        data=data
    )

def upload_chunked(sp_path: str, file_path: str):
    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)
            if not r.ok:
                raise RuntimeError(f"Chunk upload failed: {r.status_code} {r.text}")

            start += len(chunk)

# ======================================================
# HANDLERS ÉVÉNEMENTS
# ======================================================

def upload_object(key: str):
    sp_path = map_mapping_path(key)

    if not sp_path:
        log("ignored", reason="non_pedagogical", key=key)
        return

    log("upload_start", key=sp_path)

    file_path = download_from_minio(key)
    size = os.path.getsize(file_path)

    try:
        if size <= MAX_SIMPLE_UPLOAD:
            log("upload_mode", mode="simple", size=size)
            with open(file_path, "rb") as f:
```

```
                upload_simple(sp_path, f.read())
        else:
            log("upload_mode", mode="chunked", size=size)
            upload_chunked(sp_path, file_path)

        log("upload_success", key=sp_path, size=size)

    finally:
        os.remove(file_path)

def delete_object(key: str):
    sp_path = map_mapping_path(key)
    if not sp_path:
        return

    log("delete", key=sp_path)

    graph_request_with_retry(
        "DELETE",
        f"{GRAPH_BASE}/drives/{DRIVE_ID}/root:/{sp_path}"
    )

# =====================================================
# ENDPOINT WEBHOOK MINIO
# =====================================================

@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 = unquote(record["s3"]["object"]["key"]).split("?")[0]

    log("event_received", s3_event=event_name, key=key)

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

    return "", 204

# =====================================================
# ENDPOINT HEALTH
# =====================================================

@app.route("/health", methods=["GET"])
def health():
    return {
        "status": "ok",
        "service": "sharepoint-proxy"
    }, 200
```

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
    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
```

- configurer automatiquement l'alias minio dans le conteneur sharepoint-proxy
  - céer un script d'initialisation entrypoint.sh à la racine du conteneur sharepoint-proxy (même niveau que Dockerfile et main.py) :

```
touch entrypoint.sh
chmod +x entrypoint.sh
```

- Contenu de entrypoint.sh : configure l'alias minio, utilise des variables d'environnement et démarre ensuite Gunicorn

```
#!/bin/sh
set -e

echo "Initialising mc alias for MinIO..."

mc alias set minio \
  "http://minio:9000" \
  "$MINIO_ROOT_USER" \
```

```
  "$MINIO_ROOT_PASSWORD" \
  --api S3v4

echo "mc alias 'minio' configured"

# Lancer Gunicorn (process principal)
exec "$@"
```

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

Dockerfile

```
FROM python:3.12-slim

WORKDIR /app

# ---- Dépendances système utiles ----
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
       ca-certificates \
       curl \
    && rm -rf /var/lib/apt/lists/*

# ---- Installer MinIO Client (mc) ----
RUN curl -sSL https://dl.min.io/client/mc/release/linux-amd64/mc \
    -o /usr/local/bin/mc \
    && chmod +x /usr/local/bin/mc

# ---- Dépendances Python ----
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# ---- Code applicatif ----
COPY . .

ENV PYTHONUNBUFFERED=1

EXPOSE 8080

COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
CMD ["gunicorn", "-w", "2", "-b", "0.0.0.0:8080", "main:app"]
```

- construire l'image

```
docker compose build sharepoint-proxy --no-cache
```

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 excéture 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'éxécute le conteneur.
>
> Si le proxy est sur un serveur distinct et non dans Docker, il faut alors utiliser le nom DNS du server 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_webhool 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`.
```

- créer le bucket

```
mc mb minio/lycee

=> résultat attendu
Bucket created successfully `minio/lycee`.
```

- vérification

```
mc ls minio
```

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

> Filtrage côté MinIO (RECOMMANDÉ) : le proxy ne reçoit QUE ce qui est attendu

```
mc event add minio/lycee arn:minio:sqs::sharepoint:webhook \
  --event put,delete \
  --prefix administration/ \
  --prefix bts-sio/ \
  --prefix bts-mco/



=> doit renvoyer
Successfully added arn:minio:sqs::sharepoint:webhook
```

Commentaires :

- minio/lycee → bucket concerné
- sharepoint → webhook qui a été configuré
- put → upload / création
- delete → suppression

- Vérifier que c'est bien pris en compte

```
mc event list minio/lycee

=> doit renvoyer
arn:minio:sqs::sharepoint:webhook   s3:ObjectCreated:*,s3:ObjectRemoved:*   Filter:
```

- supprimer l'event sharepoint du bucket test

```
mc event remove minio/lycee --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 apparait le fichier Documents/fichier.pdf
- Suppression

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

- Fichier supprimé dans SharePoint