Explication du code du Watchdog

Import des modules

```
import os
import re
import time
import pymongo
```

- `os` : ce module fournit une interface pour interagir avec le système d'exploitation sous-jacent, tel que la création de répertoires, la suppression de fichiers, etc.
- `re` : ce module fournit des fonctions pour travailler avec des expressions régulières (regex). Il est utilisé ici pour extraire le nom du service à partir du nom de fichier du log.
- `time` : ce module fournit des fonctions pour travailler avec le temps, telles que l'obtention de l'heure actuelle ou l'attente d'un certain nombre de secondes.
- `pymongo` : ce module fournit une interface pour interagir avec une base de données MongoDB. Il est utilisé ici pour se connecter à la base de données et effectuer des requêtes.

Connection à la base de données

```
# Connexion à la base de données MongoDB
client = pymongo.MongoClient("mongodb://root:example@127.0.0.1:27017/")
db = client["logs_db"]
```

Le code utilise donc le module `pymongo` pour se connecter à une base de données MongoDB, fonctionnant dans un Docker. La fonction `pymongo.MongoClient()` est utilisée pour créer une instance du client MongoDB, qui peut ensuite être utilisée pour interagir avec la base de données via la variable `client`.

L'argument passé à `MongoClient()` est une URL qui spécifie l'emplacement de la base de données, ainsi que les identifiants pour s'y connecter. Ici, l'URL spécifie l'hôte `127.0.0.1` et le port `27017` (le docker étant executé avec le paramètre `--network host`, le docker partage les ports de l'hôte. Nous pouvons donc utiliser l'adresse 127.0.0.1 pour atteindre le docker si il est hébergé sur la même machine que ce script), ainsi que le nom d'utilisateur `root` et le mot de passe `example`.

Une fois le client MongoDB créé, le code utilise la variable `client` pour sélectionner la base de données à utiliser. Dans cet exemple, la base de données sélectionnée est `"logs_db"`. Si la base de données n'existe pas, elle sera créée automatiquement lors de la première insertion de données.

Une fois la connexion établie, le code peut utiliser la variable `db` pour interagir avec la base de données en effectuant des opérations telles que l'insertion de données ou la recherche de données.

```
# Fermeture de la connexion à la base de données
client.close()
```

A la fin d'execution du script, le code doit fermer la connexion à la base de données en appelant la méthode `close()` sur l'objet `client`.

Montage du dossier distant

```
os.system(
    "sshfs -o KexAlgorithms=diffie-hellman-group14-sha1 "
    "-oHostKeyAlgorithms=+ssh-dss msfadmin@127.0.0.1:/var/log/ ./log")
```

La commande `sshfs` est appelée au début du script, elle utilise le protocole SSH pour monter un dossier distant sur le système local. Les options `-o KexAlgorithms=diffie-hellman-group14-sha1` et `-o HostKeyAlgorithms=+ssh-dss` spécifient les algorithmes de chiffrement et d'authentification à utiliser pour la connexion SSH.

Les arguments suivants spécifient les informations de connexion à utiliser pour la connexion SSH. Dans cet exemple, la commande se connecte à l'utilisateur `msfadmin` sur l'hôte `127.0.0.1` , et monte le dossier `/var/log/` sur le dossier local `./log` .

En d'autres termes, le code monte le dossier `/var/log/` du système distant sur le dossier local `./log` . Cela permet au code de lire les fichiers de log à partir du système distant comme s'ils étaient présents sur le système local, ce qui facilite le traitement des fichiers de log.

Parcourir les fichiers de logs

```
while True:
    # Le contenu de la boucle sera exécuté en continu
    for root, dirs, files in os.walk(log_dir):
        for filename in files:
            # Le contenu de la boucle sera exécuté pour chaque
            # fichier de log dans le répertoire log_dir
```

Cette partie de code est composée de deux boucles imbriquées : une boucle infinie `while True` et une boucle `for` qui parcourt les fichiers de log dans le répertoire spécifié par la variable `log_dir` .

La boucle infinie permet de faire tourner le script en continu, en bouclant sans arrêt sur les fichiers de logs, afin de les ajouter en temps réel à la base de données. Cette boucle s'exécute en permanence tant que le programme n'est pas interrompu.

La deuxième boucle `for` utilise la fonction `os.walk()` pour parcourir tous les fichiers dans le répertoire `log_dir` .

La boucle infinie et la boucle `for` sur les fichiers de logs permettent au script de rester actif en permanence et d'ajouter de nouveaux logs à la base de données en temps réel.

Traitement du fichier log

```
# Vérification que le fichier est un fichier de log
if filename.endswith(".log"):
    # Détermination du nom de service
    service_name = re.sub(r'\..*', '', filename)

    # Ouverture du fichier de log
    file_path = os.path.join(root, filename)
    try:
        if filename.endswith(".log"):
            log_file = open(file_path, "r")
        else:
            continue
    except Exception as e:
        continue

    # Boucle sur les lignes du fichier de log
    for line in log_file:
        timestamp = time.time()

        # Rename some services
        if service_name == "error":
            service_name = "apache_error"
        if service_name == "access":
            service_name = "apache_access"
        if service_name == "access":
            service_name = "apache_access"

        # Extraction des informations de la ligne
        # (cette partie dépend des formats de vos fichiers de log)
        log_info = {
            "service": service_name,
            "id": idx_id,
            "timestamp": timestamp,
            "filename": filename,
            "line": line
        }

        # Vérification que la ligne n'a pas déjà été insérée
        if not db[log_collection].find_one({"line": line}):
            # Enregistrement des informations dans la base de données
            db[log_collection].insert_one(log_info)
            idx_id = idx_id + 1

    # Fermeture du fichier de log
    log_file.close()
```

Cette partie de code traite chaque fichier de log du répertoire log_dir.

Le code vérifie si le fichier se termine par ".log" et, si c'est le cas, détermine le nom du service en utilisant la méthode `re.sub()`. La méthode `re.sub()` est utilisée pour remplacer tous les caractères après le premier point dans le nom de fichier par une chaîne vide, ce qui donne le nom du service. Par exemple, si le nom du fichier est `apache_access.log`, le nom du service sera `apache_access`.

Le code ouvre ensuite le fichier de log en utilisant la fonction `open()`. Si le nom du fichier ne se termine pas par ".log", la boucle continue à traiter le fichier suivant. Si une erreur se produit lors de l'ouverture du fichier, la boucle continue également à traiter le fichier suivant.

Ensuite, la boucle lit chaque ligne du fichier de log en utilisant une boucle `for`. Pour chaque ligne lue, le code extrait les informations de la ligne et les stocke dans un dictionnaire appelé `log_info`. Les informations stockées sont le nom du service, un identifiant unique (`idx_id`), le timestamp et le contenu de la ligne de log.

Avant d'insérer les informations dans la base de données, le code vérifie si la ligne de log a déjà été insérée en utilisant la méthode `find_one()`. Si la ligne n'a pas été insérée, le code insère les informations dans la base de données en utilisant la méthode `insert_one()`. Après l'insertion, l'identifiant unique est incrémenté de 1.

Finalement, le fichier de log est fermé à l'aide de la méthode `close()`. La boucle passe ensuite au fichier de log suivant.