Évaluation finale, Mini Broker de Messages

Le contexte

Vous avez vu comment `asyncio.Queue` permet de faire communiquer des producteurs et des consommateurs dans un même process. Le problème : si le programme s'arrête, tout disparaît.

Des outils comme Kafka ou RabbitMQ résolvent ça en ajoutant deux choses : la **persistance** (les messages survivent aux redémarrages) et le **réseau** (producteurs et consommateurs sont des programmes séparés qui communiquent via un serveur).

Petit rappel : comment fonctionne Kafka

Kafka est un **broker de messages distribué**. Son rôle est de recevoir des messages, de les stocker de façon durable, puis de permettre à d'autres programmes de les relire quand ils en ont besoin.

Le fonctionnement de base repose sur quelques éléments simples :

- Un **producer** est un programme qui envoie des messages.
- Un **consumer** est un programme qui lit des messages.
- Un **broker** est le serveur Kafka qui reçoit, stocke et sert les messages.
- Un **topic** est une catégorie de messages, comme un canal ou un dossier logique.

Quand un producer envoie un message, il ne parle pas directement au consumer. Il l'envoie au broker, dans un topic. Le broker écrit ce message dans son stockage. Ensuite, un consumer peut demander au broker : "donne-moi les messages de ce topic".

Chaque message dans un topic possède une position numérique appelée **offset**. On peut voir l'offset comme un numéro de ligne dans l'historique des messages :

- offset `0` : premier message
- offset `1` : deuxième message
- offset `2` : troisième message

Grâce à cet offset, un consumer sait où il en est. S'il a déjà lu jusqu'à l'offset `12`, il peut redémarrer plus tard et reprendre à partir de `13` sans relire tout l'historique. C'est une idée centrale dans Kafka : **le broker conserve les messages, et le consumer garde la trace de sa position**.

Dans cette évaluation, vous n'allez pas recréer tout Kafka, mais vous allez reproduire ce principe :

- un programme joue le rôle du **broker**
- un programme joue le rôle du **producer**
- un programme joue le rôle du **consumer**
- et chaque message devra avoir une position permettant de reprendre la lecture après un redémarrage

Votre mission : construire un mini broker de messages en Python.

Ce que le système doit faire

Votre système est composé de trois programmes distincts :

`broker.py`, le serveur central. Il reçoit des connexions, stocke les messages, et les distribue aux consommateurs qui en font la demande.

`producer.py`, un client qui se connecte au broker et envoie des messages vers un topic de son choix.

`consumer.py`, un client qui se connecte au broker, indique un topic et une position de départ, et reçoit tous les messages disponibles depuis cette position.

(Vous êtes libre d'améliorer l'architecture)

Les contraintes

- Les messages doivent survivre à un redémarrage du broker.
- Un consommateur qui redémarre doit pouvoir reprendre là où il s'était arrêté.
- Plusieurs producteurs et plusieurs consommateurs peuvent être connectés en même temps.
- Les topics sont créés automatiquement au premier message reçu.

Ce qui sera évalué

- Le broker tourne sans planter quand plusieurs clients se connectent simultanément.
- Un message envoyé par un producteur est bien reçu par un consommateur qui le demande.
- Après un arrêt et redémarrage du broker, les messages sont toujours là.
- Le code est lisible et les concepts vus en cours sont utilisés à bon escient.

Livrable

Un dossier contenant vos trois fichiers et un court `README` expliquant comment lancer le système et le tester.