

Les API Push & le streaming d'événements

Amina MARIE

Solutions d'échange inter-applicatif



Ce module présente différentes méthodes d'échange de données entre systèmes (interne ou publics). Il se focalise particulièrement sur l'utilisation d'APIs en présentant leur fonctionnement et les méthodes courantes pour communiquer avec elles (récupérer ou envoyer des données).

- Partie 1 – Exposer ses données via une API REST sécurisée
- Partie 2 – Requêter des données et construire un mini ETL
- Partie 3 – Utiliser des API Push
- Partie 4 – Publier des données avec un POST

Projet fil rouge – API de gestion de personnages de manga



Objectif global : Construire une API complète de gestion de personnages fictifs (type Olive et Tom) avec des fonctionnalités :

- Exposition sécurisée (GET /personnages)
- Scripts de requêtage
- Simulation d'événements (push)
- Envoi de données à l'API (POST /scores)
- Chaque partie du cours alimente et améliore ce projet en y ajoutant une brique fonctionnelle.

Plan de la 3ème partie



Objectifs pédagogiques

- Comprendre le fonctionnement des API Push
- Créer une route FastAPI qui réagit à des événements
- Manipuler des messages entrants, les enregistrer et enrichir
- Découvrir des outils comme Kafka, Pub/Sub, SNS
- Simuler une logique publish/subscribe



Présentation des API Push

Définition

Les **API Push** permettent à un système externe de **pousser** (envoyer) automatiquement des données vers ton application, **sans que tu ne fasses de requête manuelle**.

Exemples : Stripe, GitHub, ou Make envoient des **webhooks** à ton application dès qu'un événement se produit.


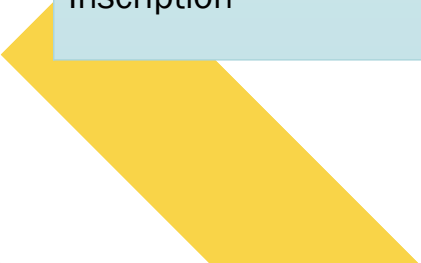
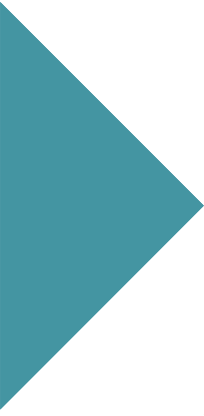
La différence avec une API REST classique

API Pull (REST)	API Push (Webhook)
Tu demandes les données	Tu reçois les données automatiquement
Ex. : GET /paiements	Ex. : POST /webhook/paiement
Nécessite des boucles ou planifications	Réactif, temps réel

Exemples concrets d'usage



Cas	Événement déclencheur	Action
Païement client	Païement validé chez Stripe	Ton app reçoit les infos et crée une facture
Nouvelle livraison	Colis remis au client	Le système envoie un SMS au destinataire
Bug détecté	Incident serveur	Alerte envoyée sur Slack ou Discord
Inscription	Nouvel utilisateur créé	Envoi automatique d'un email de bienvenue



Fonctionnement général

1. Tu t'**abonnes** à un événement (ex. : "paiement reçu")
2. Tu indiques l'**URL à appeler** dès que ça se produit
3. Le système **envoie un message** à ton URL automatiquement



Souscriptions, événements et webhooks

Vocabulaire essentiel

Terme	Signification
Publisher	Le service qui envoie l'événement
Subscriber	Celui qui le reçoit (ton app)
Webhook	URL appelée automatiquement
Event	Action déclenchée (ex. paiement reçu)
Topic (Kafka, Pub/Sub)	Sujet des messages (ex. commandes, produits)

Webhook simple

Contexte : Tu veux être averti quand un nouveau client s'enregistre.

Mise en place :

1. Tu crées une route dans ton API FastAPI :

```
@app.post("/webhook/nouveau_client")
```

```
async def recevoir_client(data: dict):
```

```
    print("Nouveau client reçu :", data)
```

```
    return {"status": "ok"}
```

2. Tu declares cette URL dans un outil externe (ex : Stripe, Make, GitHub)
3. À chaque nouveau client, l'outil appelle **automatiquement** cette route avec un POST



Outils d'API Push / Event Streaming

Apache Kafka

C'est quoi ?

- Plateforme de **messaging distribuée**
- Gère des **flux de données** (streaming) en temps réel
- Repose sur le modèle **publish / subscribe**

Fonctionnement :

- Des producteurs publient des messages dans des **topics**
- Des consommateurs s'y abonnent et les lisent

Apache Kafka

topic = commandes

prod → {id: 23, montant: 80}

→ Kafka stocke

→ consommateur lit et traite

Idéal pour :

- Traiter des événements métier à grande échelle
- Construire des architectures découplées

Google Pub/Sub (GCP)

C'est quoi ?

- Service **managé** de publication/souscription
- Similaire à Kafka, mais **sans maintenance**

Fonctionnement :

- Tu publies un message dans un **topic**
- Pub/Sub le **distribue automatiquement** à tous les abonnés

Idéal pour :

- Applications cloud natives
- Systèmes distribués sur GCP

Amazon SNS (Simple Notification Service)



C'est quoi ?

- Service d'envoi de messages et notifications
- Peut envoyer à : email, SMS, HTTP, autres services AWS

Idéal pour :

- Alerter rapidement plusieurs systèmes (cloud, mobile, humain)
- Gérer des notifications multi-canaux (API, mail, SMS)



Exercices

Objectifs

- Comprendre le principe d'**événement push**
- Créer une route FastAPI réceptrice d'un événement (simulateur de webhook)
- Utiliser un outil ou script pour **simuler un événement entrant**
- Enregistrer les événements dans un fichier
- Explorer la logique **pub/sub** sans Kafka

Exercice 1 - Créer une route webhook dans FastAPI

Créer une route `/webhook/personnage` qui reçoit un événement POST avec un nouveau personnage.

JSON Attendu : `{ "nom": "Naruto", "score": 85 }`

Guide :

- Créer une route `@app.post("/webhook/personnage")`
- Utiliser un `BaseModel` Pydantic pour valider les champs
- Afficher le personnage reçu
- Ajouter un champ `"niveau"` en fonction du score (rappel enrichissement)

Exercice 2 - Simuler l'envoi d'un événement Push

Simuler un appel automatique d'une autre application vers ton API.

Outils proposés :

- curl :
`curl -X POST http://localhost:8000/webhook/personnage \`
`-H "Content-Type: application/json" \`
`-d '{"nom": "Sasuke", "score": 90}'`
- ou **Make** : créer un scénario avec un module HTTP POST JSON

Exercice 3 - Enregistrer chaque événement reçu dans un fichier JSON

Archiver les personnages reçus dans un fichier `webhook_log.json`

Instructions :

- Si le fichier n'existe pas, le créer
- Sinon, ajouter le personnage à la suite des précédents
- Utiliser `with open(...)` et `json.load / dump`


Conseils :

- Penser à ouvrir le fichier en `r+` ou à le lire puis réécrire
- Prévoir un `try/except` pour éviter les erreurs si fichier vide ou absent

Exercice 4 - Notifier un “abonné” (console ou fichier)

Simuler un **abonné** qui reçoit la nouvelle donnée via une “publication”

Exemples d'actions déclenchées :

- Afficher un message dans la console (ex : “ Personnage ajouté avec succès !”)
- Appeler une autre route locale /notifier qui fait autre chose (ex : affiche un badge)
- Enregistrer un log dans notifications.txt

Variante :

- Ajouter une route /subscribe pour activer ou désactiver les notifications

Annexe – Code de démarrage

```
from fastapi import FastAPI
from pydantic import BaseModel






app = FastAPI()

# 👉 Modèle de personnage reçu via un webhook
class Personnage(BaseModel):
    nom: str
    score: int

# 👉 Route à compléter pour recevoir un personnage
@app.post("/webhook/personnage")
async def recevoir_personnage(p: Personnage):
    print(f"Personnage reçu : {p.nom}, score : {p.score}")

    # 💡 Étape 1 : Ajouter un champ "niveau" selon le score
    # 💡 Étape 2 : Enregistrer dans un fichier JSON (à créer s'il n'existe pas)
    # 💡 Étape 3 : Retourner un message de succès
    return {"message": "À compléter"}
```


Annexe – Problèmes fréquents

Problème	Symptôme	Solution
 Erreur 422	"value is not a valid dict"	Vérifie que le corps JSON envoyé correspond bien au schéma Personnage
 Erreur CORS	La requête est bloquée depuis une page HTML	Active allow_origins avec FastAPI.middleware.cors (Partie 1)
 Fichier JSON vide ou corrompu	json.decoder.JSONDecodeError	Supprime ou vide webhook_log.json pour repartir proprement
 Rien ne s'affiche	La route n'est pas appelée	Vérifie l'URL et la méthode (POST). Utilise curl ou Make pour tester.
 Problème d'authentification	Erreur 401	Ajoute un header de sécurité avec une clé ou token (bonus)

Annexe – Ressources

-  **FastAPI – Webhooks (POST, Body, validation)**

→ <https://fastapi.tiangolo.com/tutorial/body/>

-  **Tester une route avec curl (doc FastAPI)**

→ <https://fastapi.tiangolo.com/advanced/testing-simple/>

-  **Manipuler des fichiers JSON en Python**

→ <https://docs.python.org/3/library/json.html>

-  **Créer un webhook dans Make (ex-Integromat)**

→ <https://www.make.com/en/help/tools/webhook>

Merci

