# Kafka - Présentation

## 1. Qu’est-ce que Kafka ?

Kafka est une **plateforme de streaming distribuée** pour gérer des **flux de messages** en temps réel dans des systèmes complexes.

## 2. Composants clefs

### Producers

- **Rôle** : Envoie des messages dans des topics.
- **Exemple** : Application de commande envoyant un message à chaque nouvelle commande.

### Consumers

- **Rôle** : Lit les messages des topics.
- **Exemple** : Système de gestion des stocks lisant les messages de commandes.

### Topics

- **Rôle** : Canaux dans lesquels les messages sont envoyés et consommés.
- **Exemple** : Topic `commande-reçue` pour les commandes.

### Messages

- **Rôle** : Données envoyées par un producteur dans un topic.
- **Exemple** : `{ "orderId": "12345", "status": "received", "timestamp": "2025-02-05T12:00:00Z"}`.

## 3. Problèmes résolus par Kafka

### 1. Couplage des systèmes

- **API classiques** : Couplage fort, dépendance immédiate.
- **Kafka** : Découplage grâce au modèle publish-subscribe.

### 2. Scalabilité

- **API classiques** : Goulot d’étranglement avec une forte charge.
- **Kafka** : Scalabilité horizontale, gestion de millions de messages.

### 3. Latence

- **API classiques** : Appels synchrones, latence élevée.
- **Kafka** : Traitement asynchrone, faible latence.

### 4. Perte de données

- **API classiques** : Risque de perte de données.
- **Kafka** : Persistance et réplication des messages pour haute résilience.

### 5. Complexité et dépendances

- **API classiques** : Chaînes d'appels complexes.
- **Kafka** : Communication découplée et indépendante.

### 6. Historique des données

- **API classiques** : Pas d'historique des appels.
- **Kafka** : Stockage configurable des messages, possibilité de rejouer les événements.

## 4. Comparaison Kafka vs API classiques

| Problème | API Classiques | Kafka |
|---|---|---|
| **Couplage** | Fortement couplé | Découplage complet |
| **Scalabilité** | Goulot d'étranglement | Scalabilité horizontale |
| **Latence** | Temps d'attente | Traitement asynchrone |
| **Tolérance aux pannes** | Risque de perte de données | Haute résilience |
| **Complexité** | Chaînes d'appels complexes | Communication découplée |
| **Historique** | Pas d'historique | Conservation configurable |
| **Charge** | Risque de surcharge | Répartition automatique |
| **Traitement parallèle** | Gestion manuelle | Plusieurs consommateurs |

## 5. Schéma

```
graph LR
  subgraph Kafka Cluster
    T1["Topic 1"]
```

```
  T2["Topic 2"]
end

P1["Producer 1"] --> T1
P2["Producer 2"] --> T2

C1["Consumer 1"] --> T1
C2["Consumer 2"] --> T2
```

- **Producers** publient des messages dans des **Topics**.
- **Consumers** lisent les messages de ces **Topics**.
- Kafka est **distribué**, les messages sont répartis sur différents serveurs.

# 6. Conclusion

Kafka permet de gérer de grands flux de données en temps réel, avec une faible latence, une haute résilience et une scalabilité horizontale, tout en résolvant les problèmes liés aux API classiques comme le couplage, la perte de données et la gestion de la charge.