MONGODB

# Change Streams

Traiter les données MongoDB comme un flux d'événements temps réel

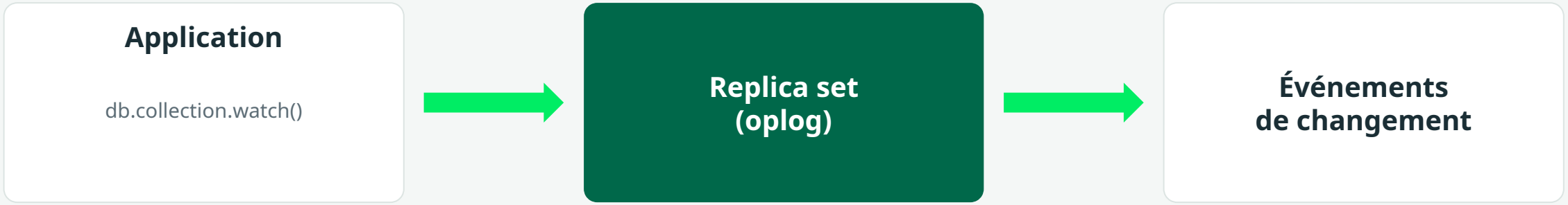
COURS MONGODB

# Sommaire

CONCEPT

# Qu'est-ce qu'un change stream ?

Une API MongoDB qui permet à une application de s'abonner en temps réel aux modifications d'une collection, d'une base ou d'un cluster entier — sans interroger la base en boucle (polling).




*Le flux s'appuie sur l'oplog du replica set : chaque écriture y est déjà journalisée.*

POURQUOI

# Cas d'usage typiques

### Synchronisation temps réel

Répercuter les écritures vers un moteur de recherche, un cache, ou un autre système.

### Invalidation de cache

Purger ou rafraîchir une entrée de cache dès qu'un document change.

### Audit et traçabilité

Enregistrer un journal des modifications sans toucher au code métier.

### ETL / pipelines temps réel

Alimenter Kafka, un data warehouse ou un pipeline analytique en continu.

AVANT DE COMMENCER

# Prérequis techniques

| | |
|---|---|
| **MongoDB 3.6+** | Les change streams existent depuis la version 3.6 du serveur. |
| **Replica set ou cluster shardé** | Impossible sur une instance mongod autonome (standalone) — il faut l'oplog du replica set. |
| **Moteur de stockage WiredTiger** | Moteur de stockage par défaut depuis MongoDB 3.2, requis pour les change streams. |
| **Droits de lecture appropriés** | L'utilisateur doit avoir le privilège changeStream sur la ressource ciblée. |

EN PRATIQUE

# Ouvrir un change stream

```
const changeStream = db.collection('commandes').watch();

changeStream.on('change', (event) => {
  console.log(event.operationType, event.documentKey);
});
```

Écoute continue, événement par événement — le curseur reste ouvert et bloquant jusqu'au prochain changement.

```
// équivalent au niveau base ou cluster entier
db.watch()               // toute la base courante
mongoClient.watch()      // tout le cluster
```

STRUCTURE

# Anatomie d'un événement

### _id

Le resume token — identifie précisément la position dans le flux.

### operationType

Type d'opération : insert, update, replace, delete, drop, rename…

### ns

Namespace concerné : base et collection (db + coll).

### documentKey

Clé (généralement _id) du document modifié.

### fullDocument

État complet du document après modification (selon les options).

### clusterTime

Horodatage logique de l'opération dans le cluster.

OPERATIONTYPE

# Les opérations suivies

| | | | |
|---|---|---|---|
| `insert` | `update` | `replace` | `delete` |
| **drop** | **rename** | **dropDatabase** | **invalidate** |

*invalidate ferme le flux (ex. suppression de la collection écoutée) — il faut alors ouvrir un nouveau curseur.*

AGGREGATION PIPELINE

# Filtrer le flux avec $match

watch() accepte un pipeline d'agrégation : on ne reçoit que les événements pertinents, sans les filtrer côté application.

```
const pipeline = [
  { $match: {
      operationType: { $in: ['insert', 'update'] },
      'fullDocument.statut': 'urgent'
  } }
];

db.collection('commandes').watch(pipeline, {
  fullDocument: 'updateLookup'
});
```

RÉSILIENCE

# Reprendre un flux interrompu

Chaque événement porte un resume token (_id). En cas de coupure — redémarrage, panne réseau — on relance le flux exactement là où il s'est arrêté, sans perte ni doublon.

1

### Stocker le token

Sauvegarder event._id après chaque événement traité (base, fichier, cache).

2

### Flux interrompu

Panne, redéploiement, ou fermeture volontaire du curseur.

3

### Reprendre avec resumeAfter

watch(pipeline, { resumeAfter: token }) relance à partir du dernier point connu.

À RETENIR

# Bonnes pratiques et limites

## Fiabilité

- L'oplog a une taille finie : au-delà de sa fenêtre de rétention, un token trop ancien devient inutilisable.
- Toujours gérer l'événement invalidate et prévoir une reconnexion propre.
- Sur un cluster shardé, les événements sont ordonnés globalement par clusterTime.

## Performance

- fullDocument: 'updateLookup' relit le document — coût supplémentaire à mesurer.
- Un seul flux par processus consommateur : prévoir la scalabilité horizontale en amont.
- Surveiller le lag entre l'écriture et la réception de l'événement en production.