MONGODB ATLAS

Functions & Triggers

Exécuter du code serverless en réaction aux événements de vos données

COURS MONGODB

Sommaire

CONCEPT

Qu'est-ce qu'une Atlas Function ?

Une fonction JavaScript serverless, hébergée et exécutée par Atlas — aucun serveur à gérer. Elle s'appelle directement, via un endpoint HTTPS, ou automatiquement par un trigger.

Appel direct

Depuis le SDK client ou une autre function, via context.functions.execute().

Endpoint HTTPS

Exposée comme webhook, appelable depuis n'importe quel système externe.

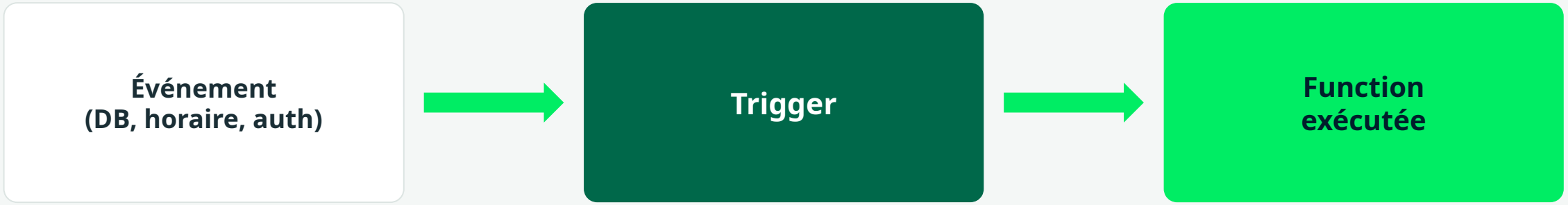
Déclenchée par un trigger

Invoquée automatiquement en réaction à un événement surveillé.

CONCEPT

Qu'est-ce qu'un trigger ?

Un trigger surveille un événement — une écriture en base, une heure planifiée, une connexion utilisateur — et invoque automatiquement une function en réponse, sans code de supervision côté application.



Sous le capot, les database triggers s'appuient sur les change streams vus précédemment.

TROIS FAMILLES

Les types de triggers

Database

Réagit aux écritures sur une collection : insert, update, delete, replace.

Scheduled

S'exécute selon une planification — expression cron ou intervalle simple.

Authentication

Se déclenche sur les événements d'authentification : création de compte, connexion, suppression.

EN PRATIQUE

Écrire une Atlas Function

```
exports = async function(changeEvent) {
  const doc = changeEvent.fullDocument;

  const notifications = context.services
    .get("mongodb-atlas")
    .db("shop")
    .collection("notifications");

  await notifications.insertOne({
    message: `Nouvelle commande : ${doc._id}`,
    createdAt: new Date(),
  });
};
```

context.services accède aux ressources Atlas ; context.values et context.secrets gèrent la configuration sensible.

CONFIGURATION

Créer un database trigger

```
{
 "name":"onNewOrder",
 "type":"DATABASE",
 "config":{
  "service_name":"mongodb-atlas",
  "database":"shop",
  "collection":"orders",
  "operation_types":["INSERT"],
  "full_document":true
 },
 "function_name":"notifyNewOrder"
}
```

full_document: true équivaut à l'option updateLookup vue sur les change streams.

CONFIGURATION

Créer un scheduled trigger

```
{
 "name":"dailyReport",
 "type":"SCHEDULED",
 "config":{
  "schedule_type":"CRON",
  "schedule":"0 6 * * *"
 },
 "function_name":"generateDailyReport"
}
```

Alternative : schedule_type "BASIC" avec un intervalle simple (toutes les X minutes/heures) pour les cas les moins précis.

POURQUOI

Cas d'usage typiques

Notifications et alertes

Envoyer un email, un SMS ou un message Slack dès qu'un événement métier survient.

Agrégations planifiées

Recalculer chaque nuit des statistiques ou des rapports sans infrastructure batch dédiée.

Validation et enrichissement

Compléter ou corriger un document automatiquement après son insertion.

Intégrations tierces

Appeler une API externe (paiement, CRM, webhook) en réaction à un changement.

À RETENIR

Bonnes pratiques et limites

Fiabilité

- Rendre les functions idempotentes : un trigger peut réémettre un événement après une erreur.
- Toujours gérer les exceptions dans la function — une erreur non attrapée interrompt le traitement de l'événement.
- Un trigger peut prendre du retard si le volume d'écritures dépasse sa capacité de traitement.

Sécurité et performance

- Timeout d'exécution limité (par défaut 90 secondes) — déporter les traitements longs.
- Stocker les secrets (clés API, identifiants) dans context.values / context.secrets, jamais en dur.
- Limiter les droits du trigger aux règles strictement nécessaires sur la collection surveillée.