CQRS & EVENT SOURCING

Sommaire

- Définitions importantes
  - DDD
  - Pattern d’architecture
- Event Sourcing
  - Definition
  - Un exemple simple
  - Concepts clefs
  - Les deux boucles
- CQRS
  - Avantages / Difficultés
  - Comparaison avec 3 tiers Classique
  - Concepts clefs
- Le lien entre CQRS et Event Sourcing

Quest-ce qu’un pattern d’architecture ?

Domain-Driven Design

Domain-Driven Design (Conception dirigée par le domaine)

Event Sourcing

- Pattern d’architecture.
- Séquences de changement d’état amenant à l’état courant de l’application.

Un exemple : Un compte bancaire.

Ce qui est important ce n’est pas l’état final mais comment on y est parvenu

Pour les analyses/debug : Les évènements qui ont emmené à un bug nous permettent de mieux comprendre ce bug.

Reprise de données : En cas de panne c-a-d revenir dans l'état précédent la panne. (Ce que l'on a vu en base de données sur les pannes

Quel est l'interêt de l'event sourcing ?

Les concepts clés de l’Event Sourcing :

Evénement :

C’est un fait qui s’est produit dans le passé, il ne peut donc pas changé.

Event Store :

- L’event store (ou magasin d’évènements) sert à **stocker l’ensemble des évènements** produits par l’application.

Commande

- une commande est une **intention provoquée par l'extérieur** (utilisateur/système) sur notre application.

Système

- L’utilisation du pattern d’event sourcing nous conduit à utiliser une **logique métier.**
  Qui contient les fonctions de **décision** et **d'évolution.**

Fonction de décision

Lorsque qu’une commande parvient à notre système, celle-ci active la **fonction de décision.**

Qu’est que la fonction de décision ?

- Etat courant + commande reçue liste d’évènement(s).
- Les évènements sont stockés dans **l’event store**.
- évènement -> **fonction d’évolution.**
- Représentation : (State, Command) => List[Event]

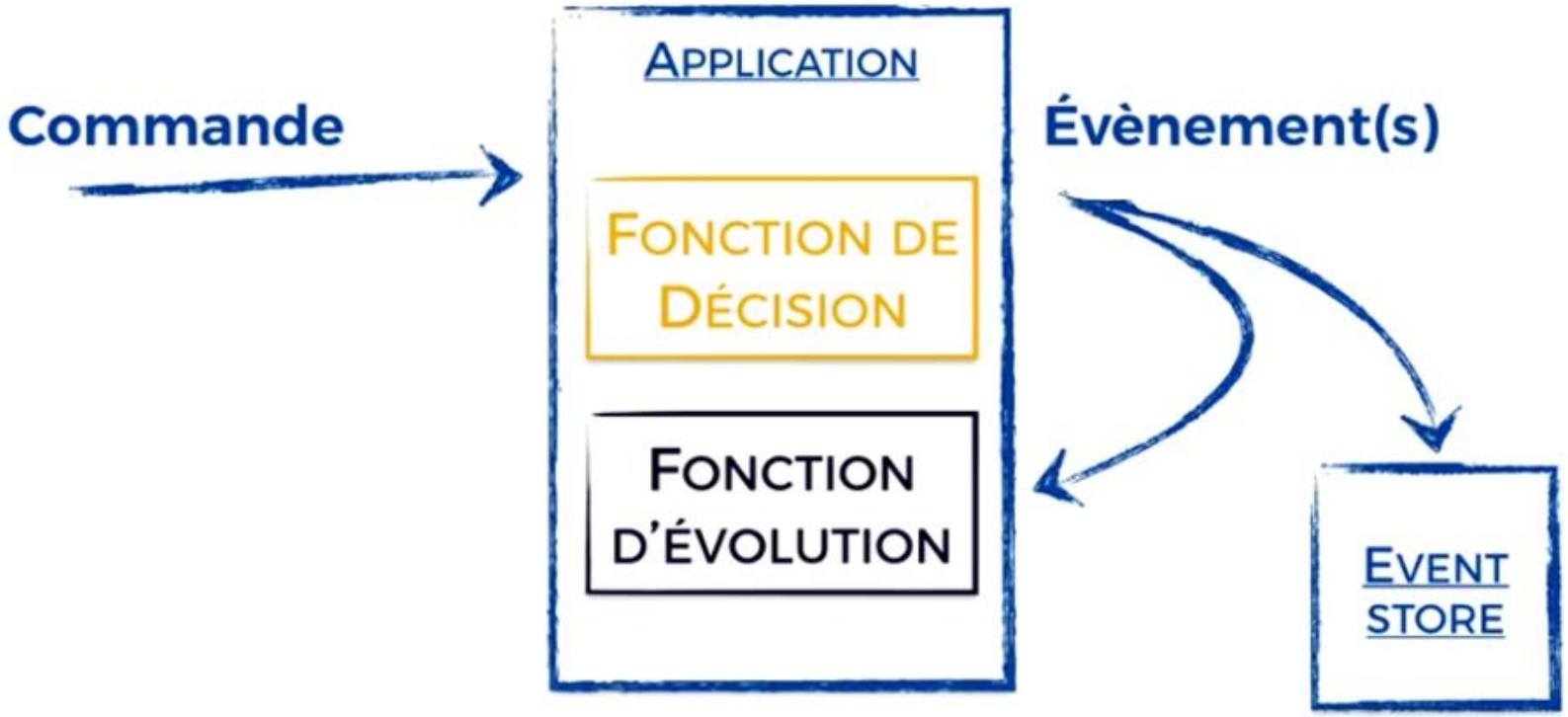
Fonction d’évolution

La fonction d’évolution a pour objectif de faire changer l’état courant de l’application.

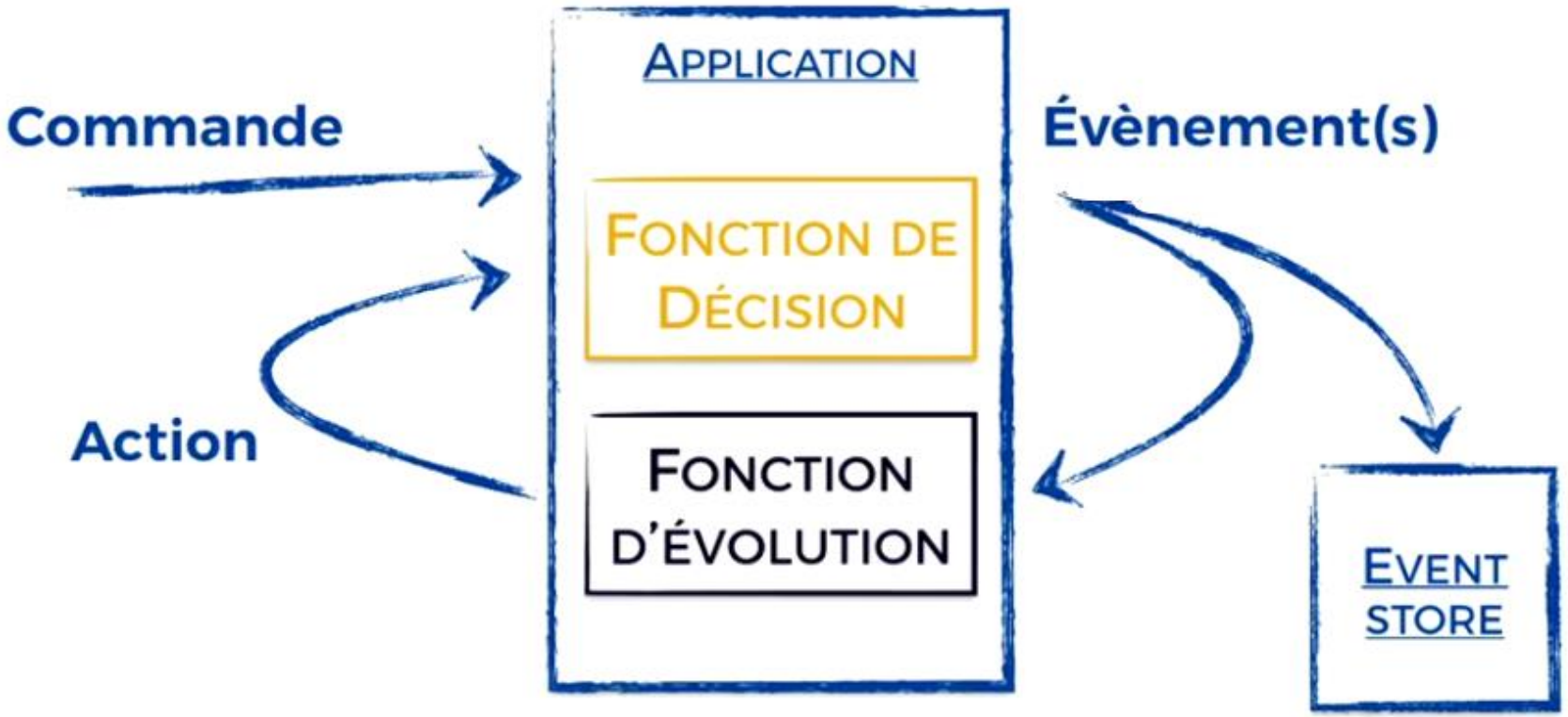
Qu’est que la fonction d’évolution ?

- La fonction d’évolution a pour objectif de faire **muter l’état courant** de l’application.
- état courant + événement -> **nouvel état courant.**
- Représentation : (State, Event) => State

Les deux boucles de l’event Sourcing



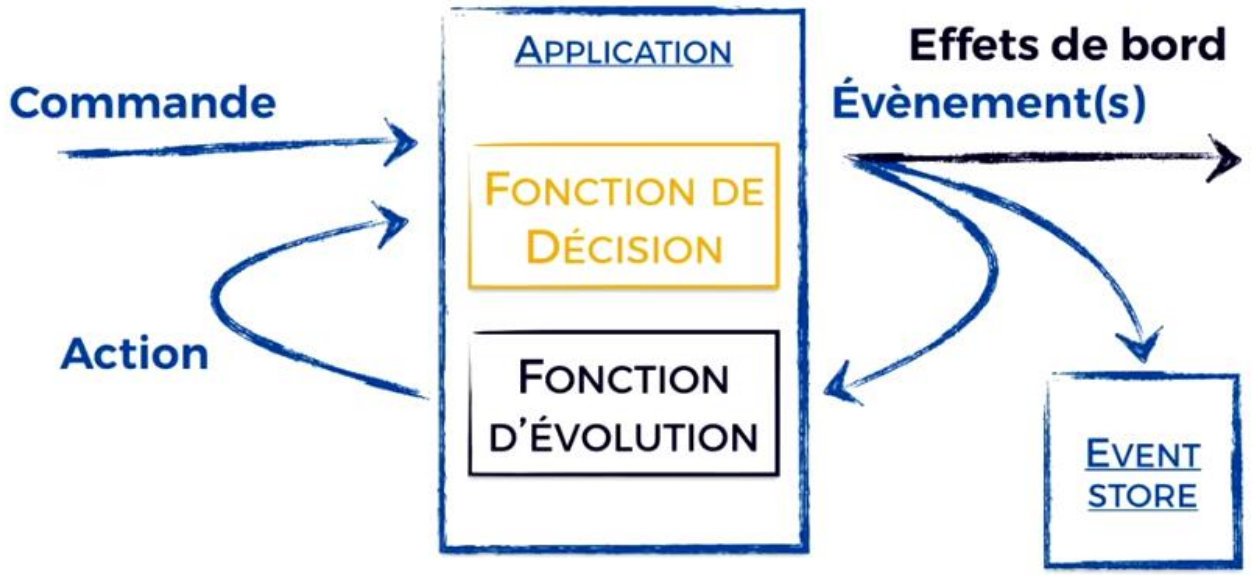
Réception d'une commande (première boucle)



Création d'une commande interne (deuxième boucle)

Qu’est ce que l’effet de bord ?

- Durant l'exécution de ces boucles, on dit que notre application est dans un **état instable,** dès lors qu’elles sont finies, notre système rentre dans un état stable.
- C’est alors qu’on peut parler **d’effet de bord.**



CQRS

- Command Query Responsibility Segregation
- Modèle d’architecture système
- Se base sur la séparation des composants de traitement de métier de l’information (“command”/écriture) et de restitution de l’information (“query”/lecture).

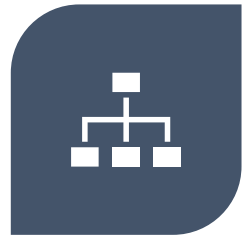
Les avantages :

MISE À L'ÉCHELLE DE MANIÈRE INDÉPENDANTE

SÉCURITÉ

SÉPARATION DES PROBLÈMES

REQUÊTES SIMPLIFIÉES



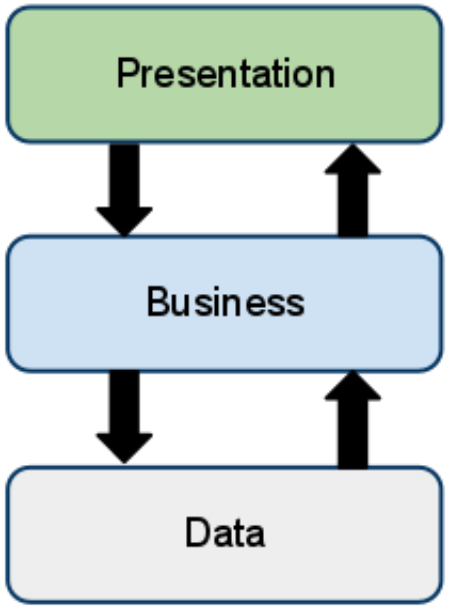
SCHÉMAS DE DONNÉES OPTIMISÉS

Les difficultés :

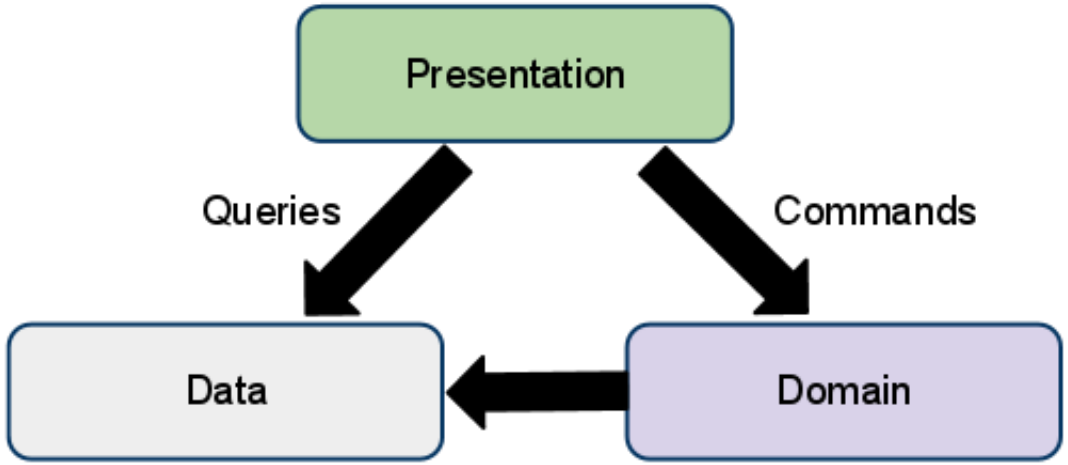
COMPLEXITÉ

MESSAGERIE

COHÉRENCE FINALE



Architecture 3 Tiers



Architecture CQRS

Comparaison entre CQRS et 3 tiers classique :

Les concepts clés de CQRS :

CommandHandler

- C’est un gestionnaire de commandes qui reçoit et retourne un résultat à partir de l’agrégat approprié par exemple un résultat est soit une reussite ou une exception.

EventHandler

- C’est un gestionnaire qui fonctionne de manière asynchrone et gère les entrées reçues dans un événement.

CommandStore

- Il sert à stocker les requêtes de l’application

Le domaine (Domain)

- Cette un zone où toute la connaissance métier de l’application est concentrée. C’est dans cette zone qu’on analyse chaque commande et de décider de leurs suites.

Microservices:

- Le CQRS permet de séparer aussi différentes couches: Interface (API ou front), CommandHandler et l'EventHandler. Elle peut aussi séparé l'EventHandler en plusieurs petites couches, on obtient des microservices spécialisé et optimisé. Le code devient plus simple à maintenir.

Fiabilité des données :

- Toutes les modifications sont automatiquement stockées à deux endroits :
- CommandStore et l’EventStore, cela nous garantit de ne perdre aucune donnée.

Le lien entre DDD, CQRS et EVENT SOURCING

“Basically CQRS + DDD [and Event Sourcing] are just a group of patterns, design principles and approaches that happen to work quite well together”

Rinat Abdullin