

**CQRS, DDD, ES...
ATES SOUHAITS !**

MISE EN SITUATION

MISE EN SITUATION

- Retour en l'an 2018.
- Institution financière régionale.
- Systèmes historiques en **Cobol**, application client en WebDev.
- **Pas de base de données** ! Fichiers à plats produits quotidiennement.

MISE EN SITUATION

- Visite des autorités de contrôle bancaire :
 - Système **difficilement auditable** pour ce qui concerne traçabilité des droits et des actions.
 - Demande d'avoir des **listes précises de logs** pour toute action qui touche à de l'argent.
- Problème :
 - Impossible de fournir autre chose que **l'état à l'instant T** du système et les fichiers journaliers sur quelques années précédentes...

MISE EN SITUATION

- Solution : développement d'une application interne pour "digérer" les fichiers produits à plat, et produire une **base de données plus classique**.
- Contraintes : disponibilité, solidité, traçabilité, simplification des processus métier.



La solution trouvée ?



L'EVENT SOURCING



(et les micro-services)

EVENT SOURCING (ES)

MAIS C'EST QUOI UN ÉVÈNEMENT ?

- **Changement d'état** (création d'un objet, ajout d'une donnée dans un tableau...).
- Event Sourcing : l'état d'un système est composé de **la somme de ses changements d'état**.
- Ils sont stockés dans une base spécifique nommée **Event Store**.
- Un évènement est **immutable**, et l'Event-Store est "**append-only**".
- Les objets métier deviennent des **Aggregats** d'évènements.

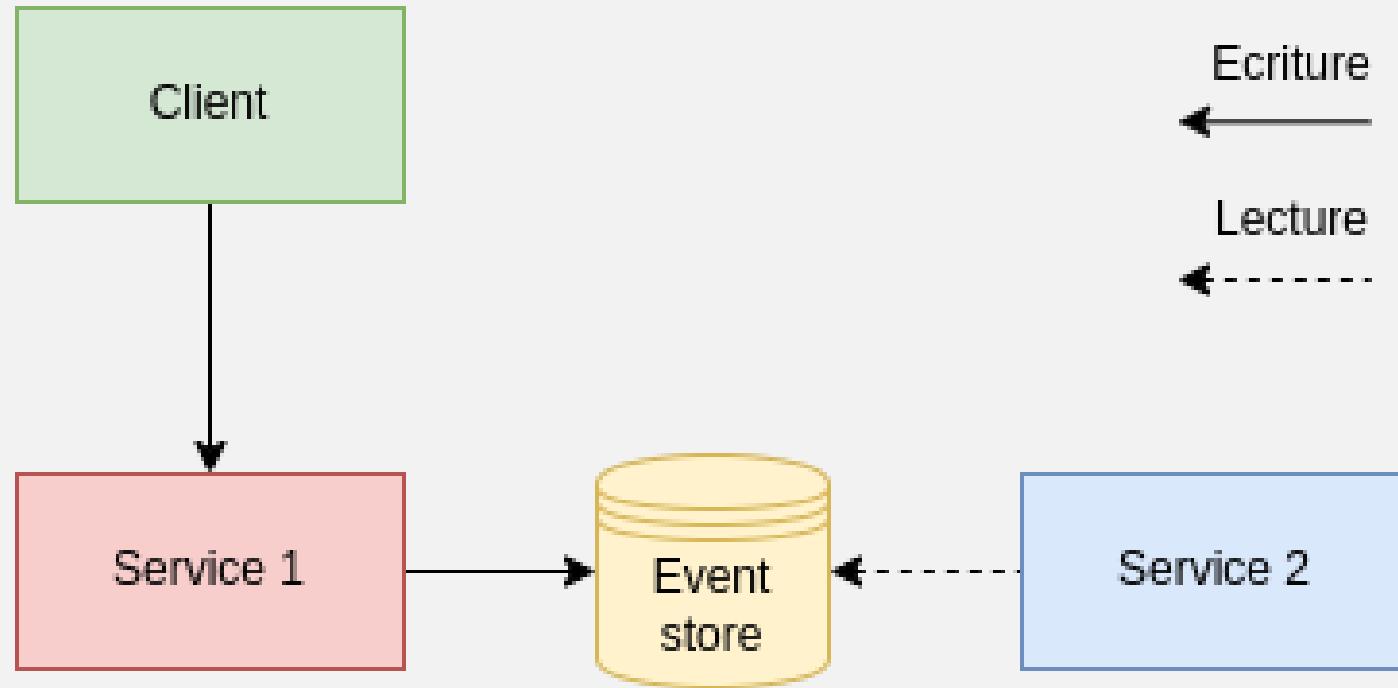


ET CONCRÈTEMENT, ÇA SERT À QUOI ?

- L'Event Store permet de **rejouer l'état d'un système** ou partie d'un système depuis sa création.
- Il est donc possible d'y effectuer des recherches pour **tracer l'historique** d'une donnée ou d'un objet.
- Gros désavantage : si une donnée arrive dans l'Event Store, elle y est **pour toujours** (peut uniquement être annulée par un évènement contraire).



UN PEU D'ARCHITECTURE



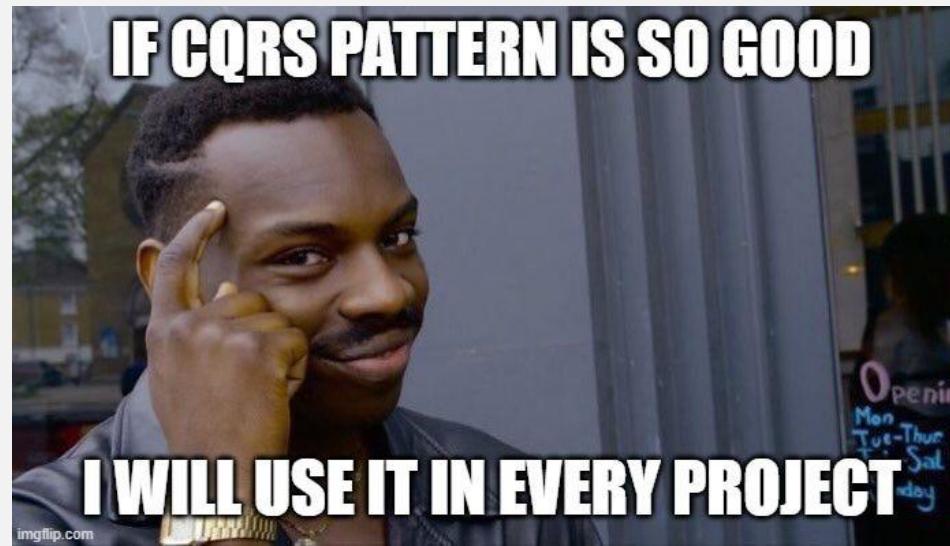
ET NIVEAU PERFORMANCES ?

- Pour un système financier les évènements se chiffrent en millions, voir même milliards !
- Si à chaque requête de données on doit chercher dans des milliards d'évènements, le système sera vite inutilisable.
- Il faut une architecture adaptée pour l'optimisation des requêtes.

COMMAND QUERY RESPONSABILITY SEGREGATION (CQRS)

COMMAND QUERY RESPONSABILITY SEGREGATION

- **Séparation des responsabilités** d'écriture et de lecture dans un système.
- **Chaque fonction peut posséder sa base de données**, et celles-ci sont synchronisées.



IL Y A QUAND MÊME QUELQUES DÉFAUTS...

➤ Avantages :

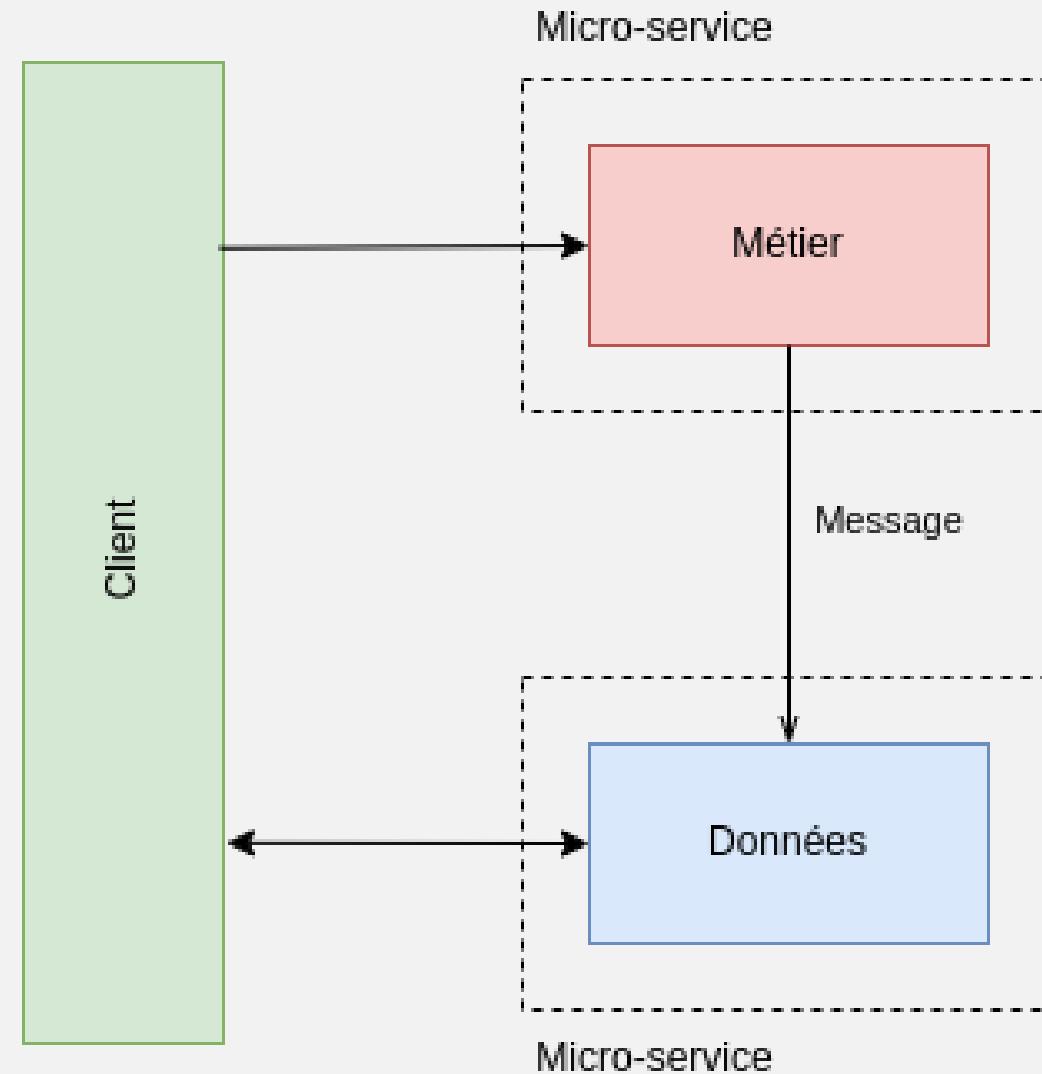
- Mode de fonctionnement asynchrone.
- Faible couplage entre les deux fonctions.

➤ Inconvénients :

- **Eventually consistent**.
- Refactoring plus couteux (deux modèles de données).

➤ Pas adapté à applications simples !

UN PEU D'ARCHITECTURE



DOMAIN DRIVEN DESIGN (DDD)

DOMAIN DRIVEN DESIGN

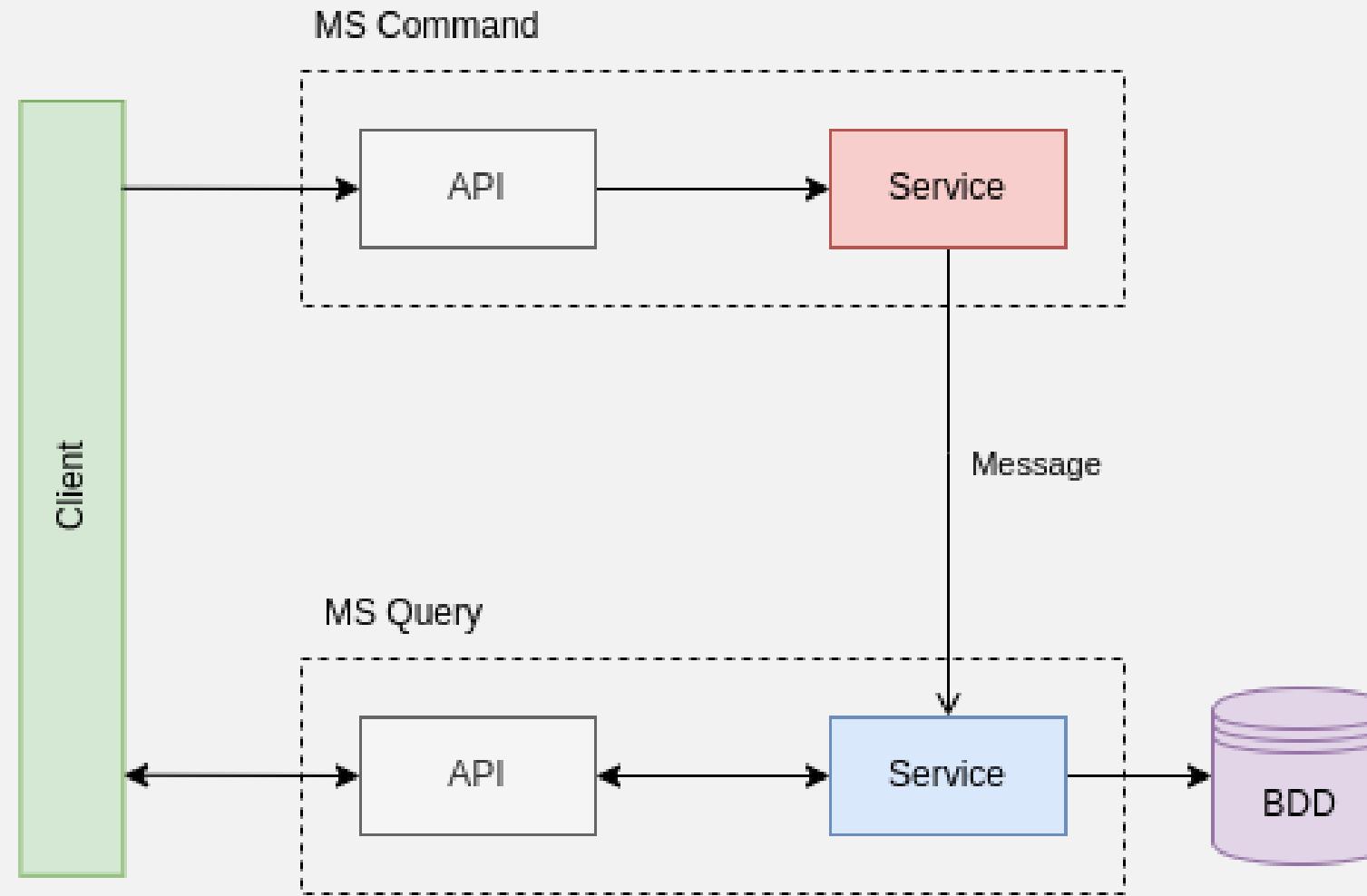
- Ensemble de pratiques pour le couplage de la conception logicielle et du domaine métier.
- Intégration du **langage Métier au sein de l'application**.
- Couplage fort de la couche "Application" et de la couche "Métier"

INTÉRÊT AVEC LES AUTRES PATTERNS

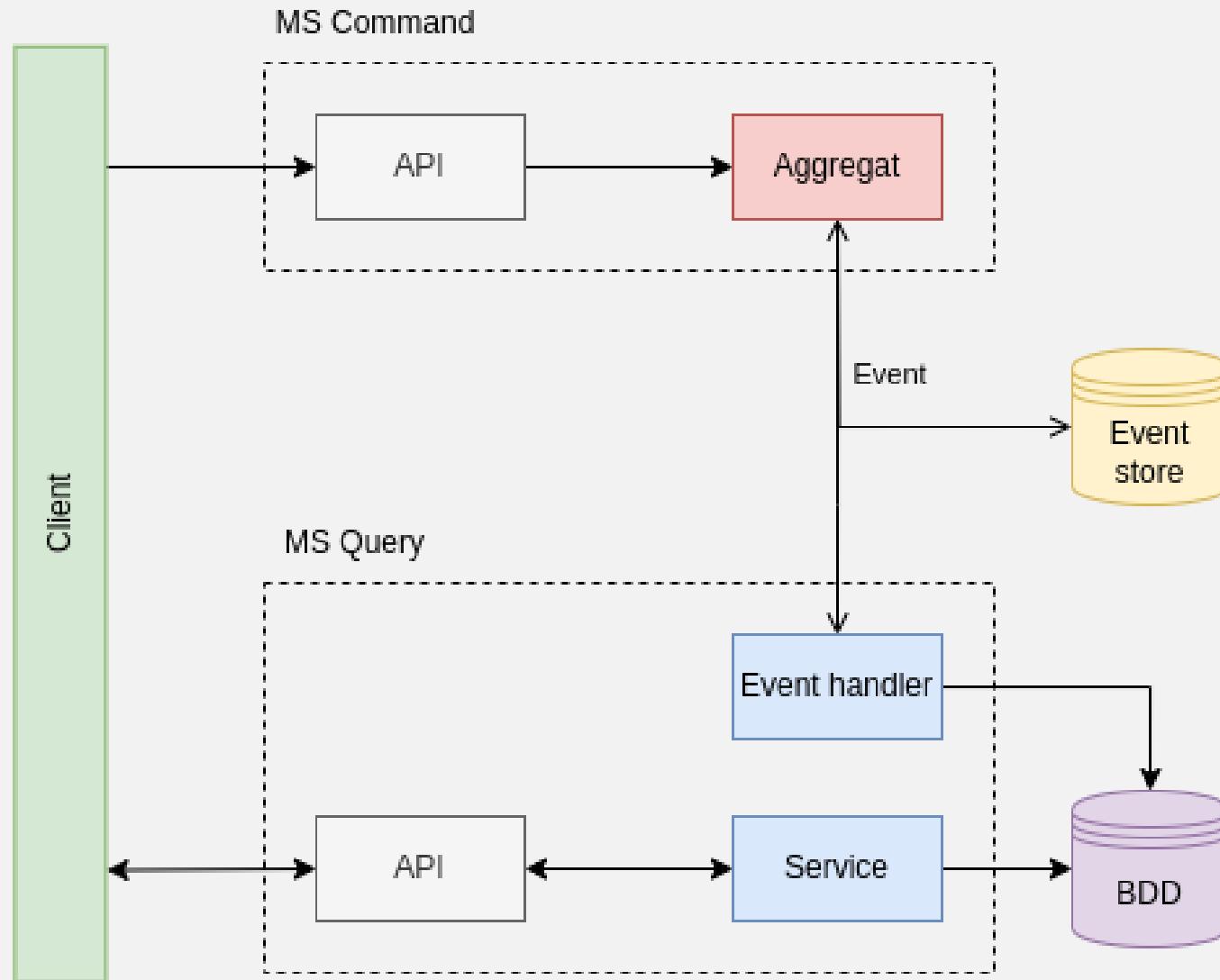
- Nommage des évènements et des Aggregats **transparent et cohérent** avec le métier.
- Exemple : un objet métier "Batiment" constitué des évenements "FondationCoulee", "MursPoses", et "ToitInstalle".
- Même chose pour les **règles de validation des données** avant la création d'un évènement.
- Facilite la conception de manière générale.
- Pas spécialement d'intérêt avec le CQRS, à part le nommage des micro-services.

ET SI ON MIXE TOUT ÇA ?

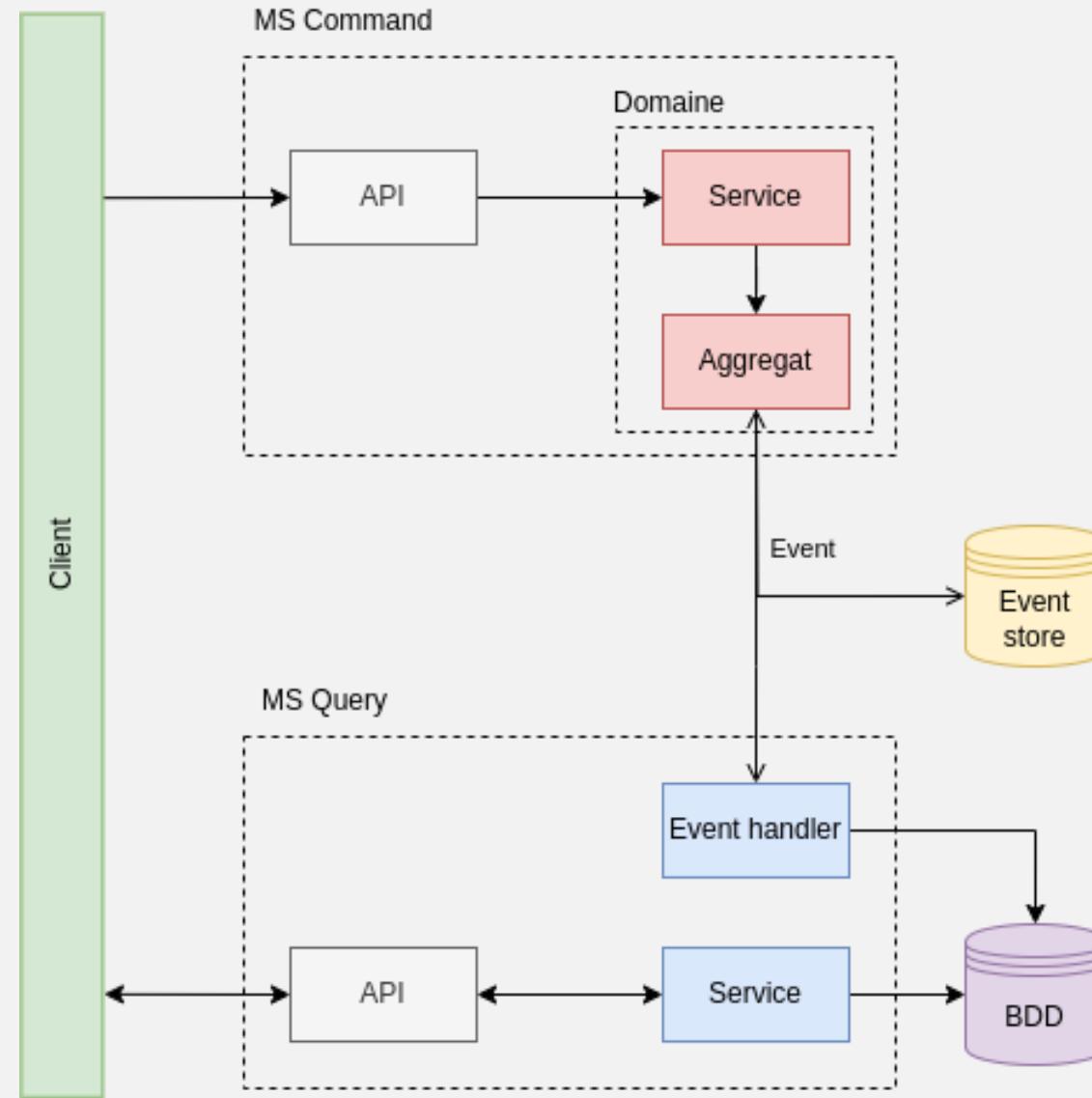
CQRS TOUT SEUL



CQRS + EVENT SOURCING



CQRS + EVENT SOURCING + DDD



DES OUTILS POUR NE PAS AVOIR À TOUT CODER

DES OUTILS POUR NE PAS AVOIR À TOUT CODER

➤ Pour faire de l'**Event Sourcing** :

- Axon Framework
- EventFlow
- Lagom

➤ Pour faire de l'**Event Driven** :

- Apache Kafka
- Spring cloud stream

**UN PETIT POC POUR
TERMINER ?**

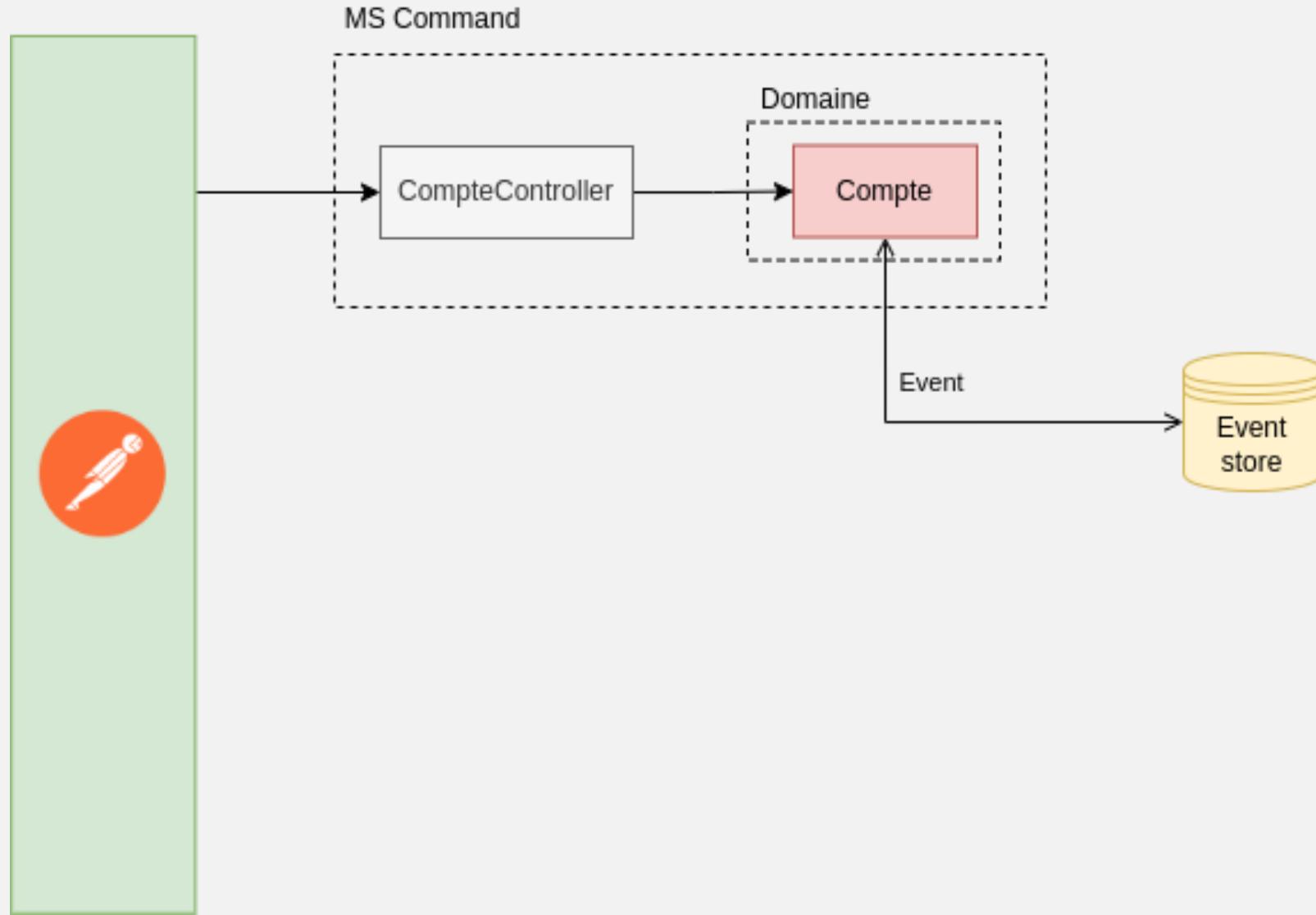
NOTRE POC

- Gestion d'un **compte bancaire** très simplifiée
 - Création d'un compte par POST
 - Mise à jour du solde par POST
- Récupération du solde par GET
- Test de la partie "Event"
 - Constater le défilement des évènements
 - **Restore** de la BDD classique en cas de suppression
 - Tester ce qu'il se passe si on éteint un module ou l'autre

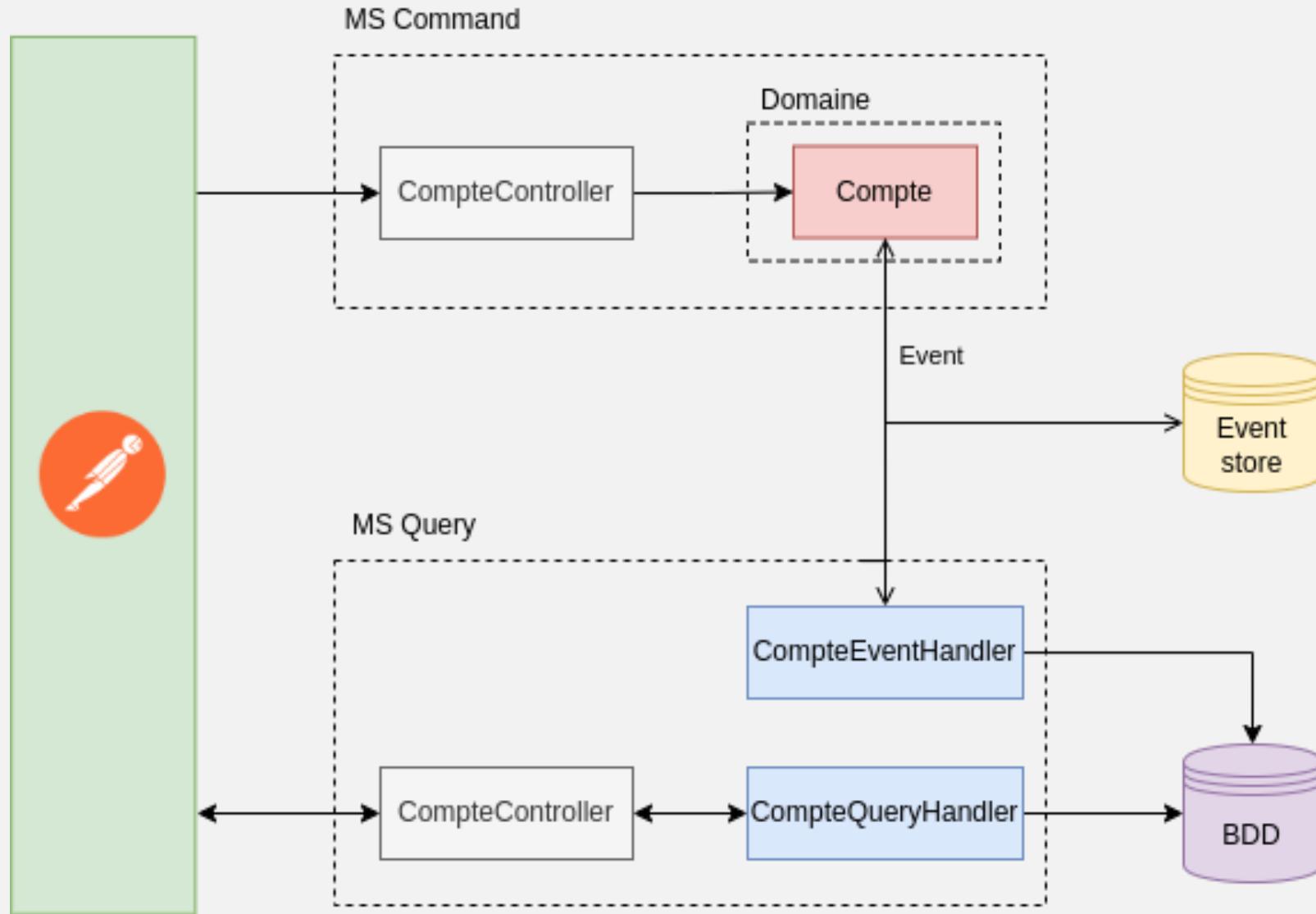
NOTRE POC



NOTRE POC



NOTRE POC



MERCI ! 

CHOSES À AMÉLIORER POUR UNE V2

- Dans le POC : tester avec plus d'events
- Développer la partie infra
 - Sécurisation de l'event store, type de données de l'event store