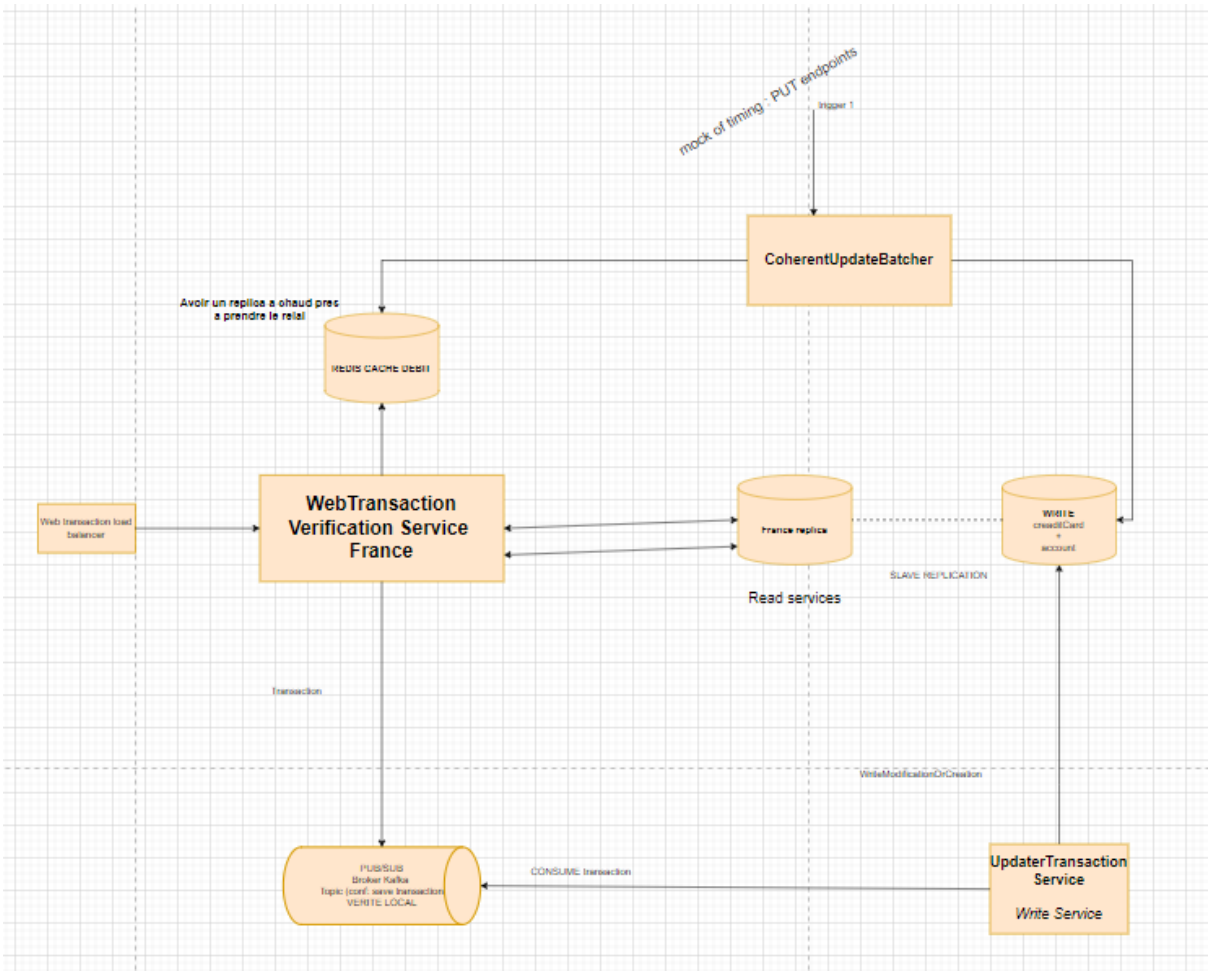
Rapport de suivi d'architecture
AL Néo-Bank - Evolution

Week 47 :

Final state of the services architecture :



Dans l'objectif de la réalisation des points évoqué dans le cadre de l'évolution qui sont :

- Lever l'hypothèse de temporisation entre transactions pour le web (et à terme pour les TPE).
- Algorithme réaliste de vérification des CBs.

Nous avons revu notre architecture et notre manière de peupler la base de données.

Levé de l'hypothèse :

L'hypothèse avait pour principal objectif de nous acheter du temps afin de permettre de procéder au flow de l'update des comptes clients après un achat, que nous avons décidé de rendre asynchrone par soucis de temps.

En effet nos objectif primordiaux été :

- Réaliser une transaction de manière rapide en minimisant le nombre d'interactions avec une base de données sql pendant la période de validation de la transaction.
- Pouvoir traiter plusieurs demandes, au nombre de 2500 en même temps.
- Être fortement disponible, en effet, sans transaction pas d'achat.

Afin de résoudre le problème de cohérence des données, l'hypothèse a été formulée. Nous avons réfléchi à une solution pour résoudre le problème.
En effet, au lieu de directement mettre à jour le compte d'un client après un achat pour retomber sur un montant dans le compte qui est de nouveau cohérent pour le client, nous allons à l'issue d'une transaction envoyer avec la transaction valide un montant, donc rien ne change, mais l'update service va lui faire monter un la valeur du débit du compte client.
Le débit et l'argent consommée par le client.

Les instances de validation de transaction pourront elles dans une BD redis ajouter le débit associé à un client et venir le récupérer pour faire la vérification de la capacité du client à pouvoir effectuer une transaction. En comparant le montant de la transaction avec la somme du solde du compte client (récupérer depuis le slave en lecture) et du débit client (récupérer depuis redis). (Voir ADR_001)

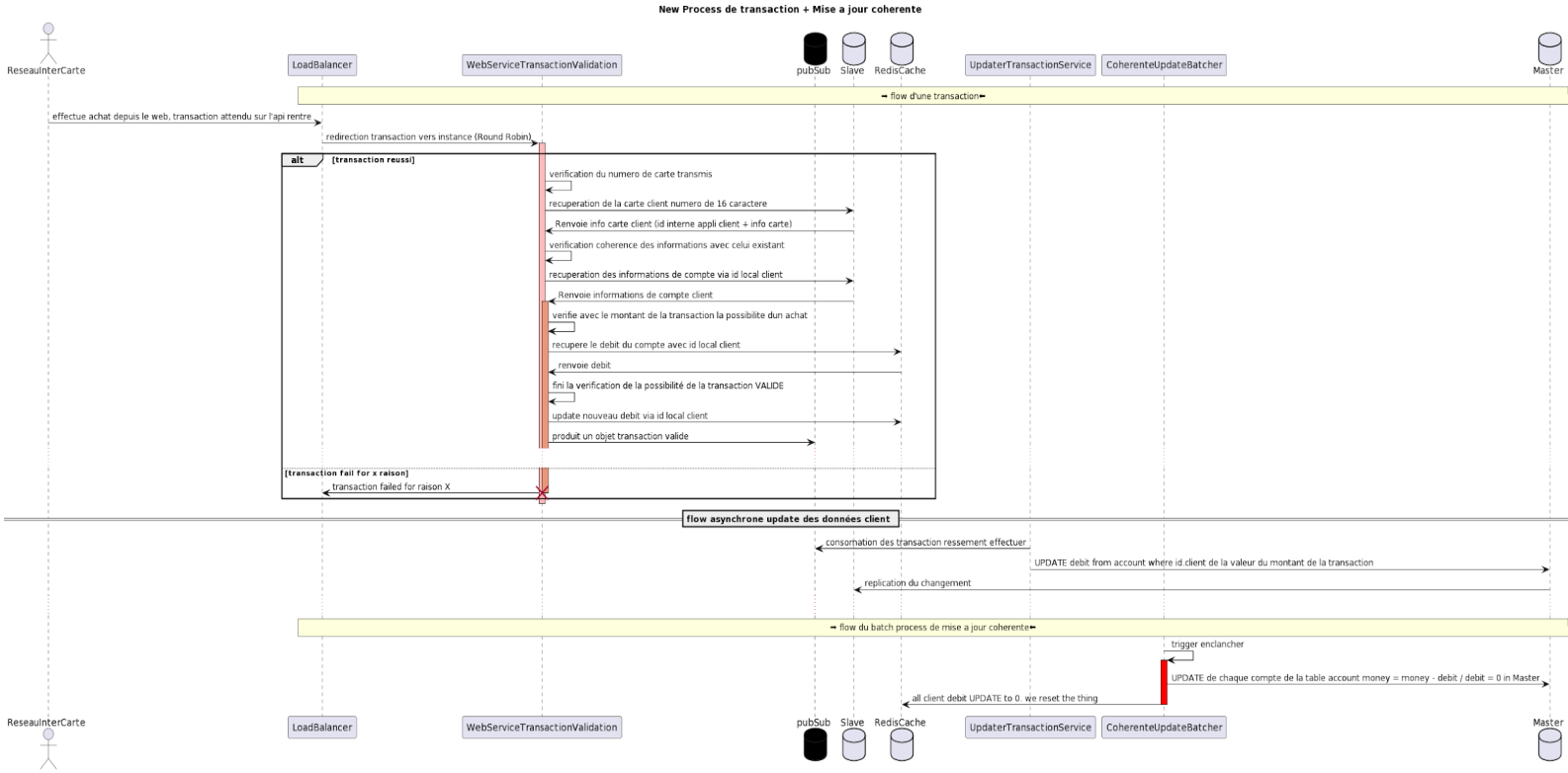
Programmer à un moment T, un service de batch va venir effectuer une "mise à jour cohérente", durant ce laps de temps, le service va venir update les débits à 0 sur Master et Redis, et va mettre à jour les comptes des clients.

REDIS

Redis sera utilisé dans notre cas comme un DB clé-valeur opérant sur la ram, donc étant nettement plus rapide dans la récupération d'un integer. N'impactant ainsi que très peu le temps et les performances.

On pourrait penser qu'utiliser redis comme un cache avec spring pourrait constituer un avantage, mais le caching sous spring implique que la valeur de la clé demandée sera renvoyé sans entrer dans la méthode, or dans notre cas, nous devons toujours a l'issue d'une transaction de nouveau changer la valeur du débit en lui ajoutant le montant de la transaction. Ainsi mettre en cache une valeur que l'on modifie souvent constitue un anti-pattern au cache.

Diagramme de séquence d’une transaction



Remplissage de la DB

Étant liée à notre second problème, le remplissage de la DB pour lancer nos tests nous a obligé à réduire la complexité de nos vérifications. Nous avons changé la manière de faire et maintenant utilisons un CSV pour remplir la DB avec 2500 compte et carte cohérente.