INFO4057: Software Architecture

TD: Event-Driven Architecture

EDA

1. What's Event?
2. What's Event-Driven Architecture?
3. What are Command and Query?
4. What are the main components of EDA
5. What's the advantage of EDA
6. What's the difference between EDA and PubSub
7. Explain concepts of Orchestration and Choreography. Give an example.
8. Explain the Saga Pattern in a given example.
9. What are Ecent Sourcing and CQRS ?
10. What's the problem with traditional Databases?

Event Streaming

1. What's Event Streaming?
2. What's the difference between Event Sourcing and EDA
3. What's the importance of Logging and Monitoring

Event Practices

Une architecture microservice pilotée par les événements partage un grand nombre des principes des architectures microservices. Toutefois, l'interaction entre les services est axée sur l'utilisation d'événements.

Considerons une application de e-commerce avec 3 service des commandes (**orderService**), un service d'inventaire (**Inventory Service**) et un service de tarification (**Pricing Service**).

Au lieu d'un appel synchrone entre le service de commande et les autres services, les services d'inventaire et de tarification réagissent à l'événement de création de commande du service de commande. Lorsqu'ils consomment l'événement, ils effectuent leur travail de manière asynchrone : le service d'inventaire met à jour le stock et le service de tarification calcule les taxes.

En gérant la communication entre les services à l'aide d'un courtier d'événements, nous découplons davantage les services. Le service de commande ne connaît aucun de ses

abonnés, et les services d'inventaire et de tarification ne connaissent que le contrat d'événement.

Il est également facile d'ajouter de nouveaux consommateurs ; il suffit de les insérer dans la file d'attente des événements. Imaginons que nous devions ajouter un service d'abonnement qui notifie les utilisateurs de la réception de la commande, nous pourrions simplement brancher ce nouveau service dans la file d'attente des messages, et nous n'aurions pas besoin de modifier le service d'ordre. Cela augmente encore la possibilité de modifier chaque composant et de faire évoluer son domaine sans affecter les composants ou les limites non liés.

1 - Proposer une architecture microservice cette même plateforme de commerce électronique avec des microservices spécifiques pour chaque contexte délimité.

2 - Proposer une architecture EDA cette plateforme de commerce électronique avec des microservices pilotés par les événements.

3 - Est'il possible de mélanger des services synchrone et asynchrones ?

4 - Le service de commande et le service inventaire a une dependance synchrone. Ajouter a votre schema la communication des services. Cette dependance a t'elle des avantages ou des incovenients ?

3.1.4 Event-Driven Message Types

La messagerie traditionnelle utilise les messages avec une certaine indifférence ; ils ne sont qu'un moyen de propager des données. Dans les architectures événementielles, cependant, il existe différents types de messages qui ont des objectifs et des significations différents. examinerons les différents **types de messages** et les situations dans lesquelles ils doivent être utilisés.

Les messages sont généralement composés d'un **en-tête et d'un corps**. Nous pouvons utiliser l'en-tête pour transmettre des informations supplémentaires communes à tous les messages, à l'instar des en-têtes HTTP. Un exemple typique est l'identifiant de corrélation, que nous utilisons souvent pour relier plusieurs messages liés à la même entité. Par exemple, si nous créons un produit et que nous modifions ensuite sa marque et sa catégorie, chacune des modifications publiera un événement différent (par exemple, les événements générés correspondants : ***ProductCreated***, **ProductBrandChanged**, **ProductCategoryChanged**) ; les trois événements pourraient avoir un en-tête avec l'identifiant du produit signalant qu'ils sont tous liés au même produit. Bien qu'il n'existe pas

de définition formelle des en-têtes qu'un message doit contenir, nous bénéficions d'un ensemble prédéfini pour l'ensemble du système. L'identifiant de corrélation dont nous avons parlé précédemment est un bon exemple, mais d'autres messages peuvent également être utilisés. comme un identifiant de message qui identifie chaque message de manière unique peut être utile à des fins de débogage. Les informations des en-têtes doivent être choisies avec soin afin d'éviter de faire circuler des informations qui seront associées à la logique du côté du consommateur, comme des drapeaux spécifiques pour des processus spécifiques. Souvent, ce type d'information peut être oublié, ce qui entraîne des erreurs dans le processus de consommation ou même des valeurs par défaut pour les en-têtes ; le chapitre 8 aborde ce sujet plus en détail. Le fait de disposer d'une version ou d'un horodatage de l'événement est également utile pour gérer l'idempotence.

Le corps du message contient toutes les informations que nous voulons publier. Les messages peuvent être des **commandes (commands)**, des **événements (events**) ou des documents (**documents**). Les requêtes sont également un concept courant, et typiquement, elles ne sont pas un message mais un concept courant dans les architectures pilotées par les événements.

- **Commandes (Commands)**

Les commandes sont des ordres d'effectuer une action donnée. Nous devons les nommer avec un verbe à la forme impérative, par exemple, **CreateOrderCommand**. Une commande est une demande adressée à un service donné pour qu'il effectue une action et peut donc être rejetée ou échouer à une validation. Nous pouvons modifier des agrégats en envoyant une commande pour effectuer une action donnée dans cet agrégat et souvent refléter l'action d'un utilisateur. En règle générale, les commandes n'affectent qu'un seul service et un domaine spécifique ; elles ne sont généralement pas publiées à plusieurs abonnés, mais à un seul. Bien que les commandes soient souvent des messages, une commande peut également être une requête HTTP si le service reçoit des modifications par l'intermédiaire d'une API et non d'un courtier de messages.

- **Les événements** (**Events**)

Ils notifient un changement dans un domaine ou un agrégat donné. Ils sont nommés au participe passé et informent que quelque chose s'est produit, par exemple, **OrderCreatedEvent**. Ils sont des faits et, contrairement aux commandes, ne sont pas susceptibles d'être rejetés ; ils sont quelque chose qui s'est déjà produit.Les événements constituent le bloc le plus courant des architectures pilotées par les événements et sont utilisés pour transmettre des informations et signaler des changements pertinents dans tous les composants de l'architecture. Ils sont souvent publiés à l'intention de plusieurs consommateurs et peuvent accueillir de nouveaux consommateurs à l'avenir, contrairement aux commandes qui sont liées à un seul système.

- Les Documents (**Documents**)

Les documents ressemblent beaucoup aux événements ; le service les publie lorsqu'un agrégat ou une entité donnée change, mais ils contiennent toutes les informations de l'entité, contrairement aux événements qui ne contiennent généralement que les informations liées au changement à l'origine de l'événement. Bien qu'ils soient souvent déclenchés par des

changements dans l'agrégat, ils ne donnent souvent pas d'informations sur le changement qui a déclenché le document, à moins que nous ne l'ajoutions spécifiquement au document. Si quelqu'un a modifié l'adresse d'une commande, l'événement généré pourrait être OrderAddressChanged et contenir les informations relatives à l'adresse de la nouvelle commande. Le même changement pourrait déclencher un document, par exemple OrderDocument, qui contiendrait toutes les informations relatives à la commande ; chaque destinataire devrait l'interpréter de la manière qui lui convient.

- **Requêtes (Queries)**

Les requêtes sont des demandes d'informations adressées à un système donné pour obtenir les données du service. En règle générale, il ne s'agit pas de messages et les requêtes sont souvent synchrones, comme les requêtes HTTP. Elles ne modifient pas non plus l'état ; elles demandent simplement des informations.

[Scenario]

Pour votre plateforme de e-commerce ci-dessous considerons l'interaction suivante: Un utilisateur qui passe une commande interroge le service d'inventaire sur le stock du produit, puis déclenche une **commande CreateOrderCommand** en soumettant une commande qui sera consommée par le service de commande. Une fois que le service a créé l'entité, il publie un **événement CreateOrderEvent** et un **document OrderDocument** qui sont consommés par le service d'inventaire et le service de notification, respectivement.

4 - Proposer une architecture EDA qui presente l'interaction avec entre les commades , events, documents et les queries.

5 - What are Commont EDA Messaging Patterns? Explain each Pattern.

[Scenario]

Sur votre plateforme de commerce de commerce l'electronique vous ajoutes le service de gestion des produits (Product Service).

6 - En appliquant le CQRS au niveau architectural, Proposer une architecture qui presente des avoir des services distincts pour les lectures et les écritures pour le Service de gestion desss produits (Product Service) .