Architecture microservices et coherence des données : mais on fait comment dans la vraie vie ?

Date : 21 avril 2022
Speaker : Jean-François James (Wordline)
Format : conférence (45mn)

Au cours de cette conférence, le sujet de la **cohérence des données** dans une **architecture microservices** sera abordé pragmatiquement au travers 2 démos.

Jean-François commence par rappeler que le sujet de la cohérence des données n'est pas nouveau et existait déjà en **SOA**. Il donne un exemple de services distribués avec chacun sa base de données.

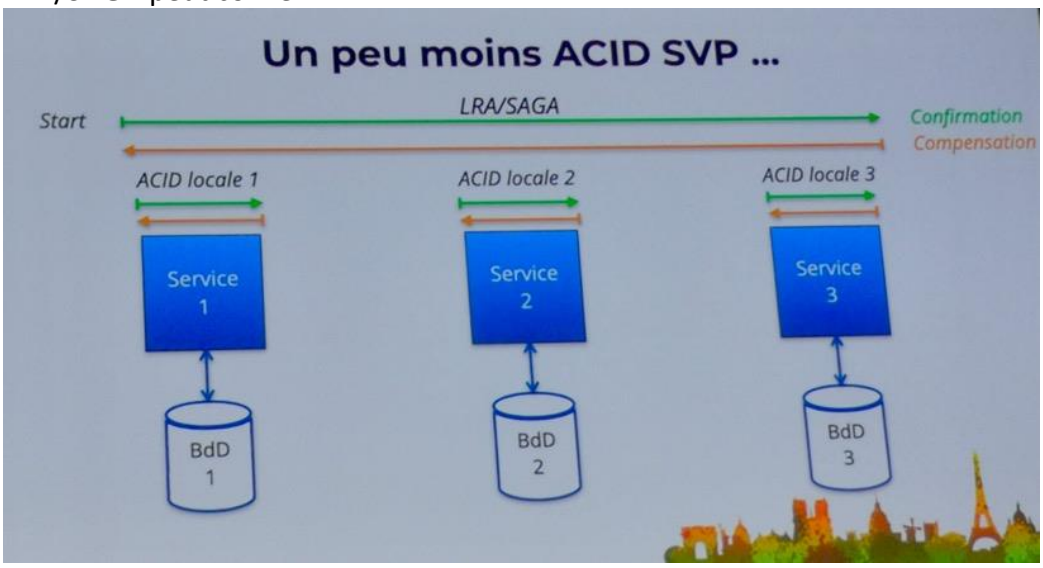
2 grands challenges :

1. Définir et border les services : demande une connaissance fonctionnelle forte
2. Garantir la cohérence d'ensemble des données

Modèle de transaction **ACID** locale.
Lorsque les modules pointent sur la même base, l'utilisation des transactions est suffisante grâce aux propriétés **A**tomicity-**C**onsistency-**i**solation-**D**urability
La plupart du temps, le niveau d'isolation n'est pas positionné à son maximum pour des questions de performance (d'où le i minuscule).

Lorsqu'on découpe ces modules en microservices avec chacun sa base.
Possibilité d'utiliser le **Two Phase Commit** (2PC). Pour Jean-François, ce protocole est verbeux, peu performant et élève le coût de la transaction (qui devient longue). S'ajoute une fragilité certaine liée à la coordination synchrone. Enfin, le modèle 2PC n'est pas supporté par des systèmes nouveaux : Kafka, bases NoSQL ….

L'utilisation d'une couche de coordination légère comme les modèles de conception **LRA**/**SAGA** peut convenir.



Ce modèle nécessite une **action de compensation** à gérer par le développeur (ex : recréditer un compte).

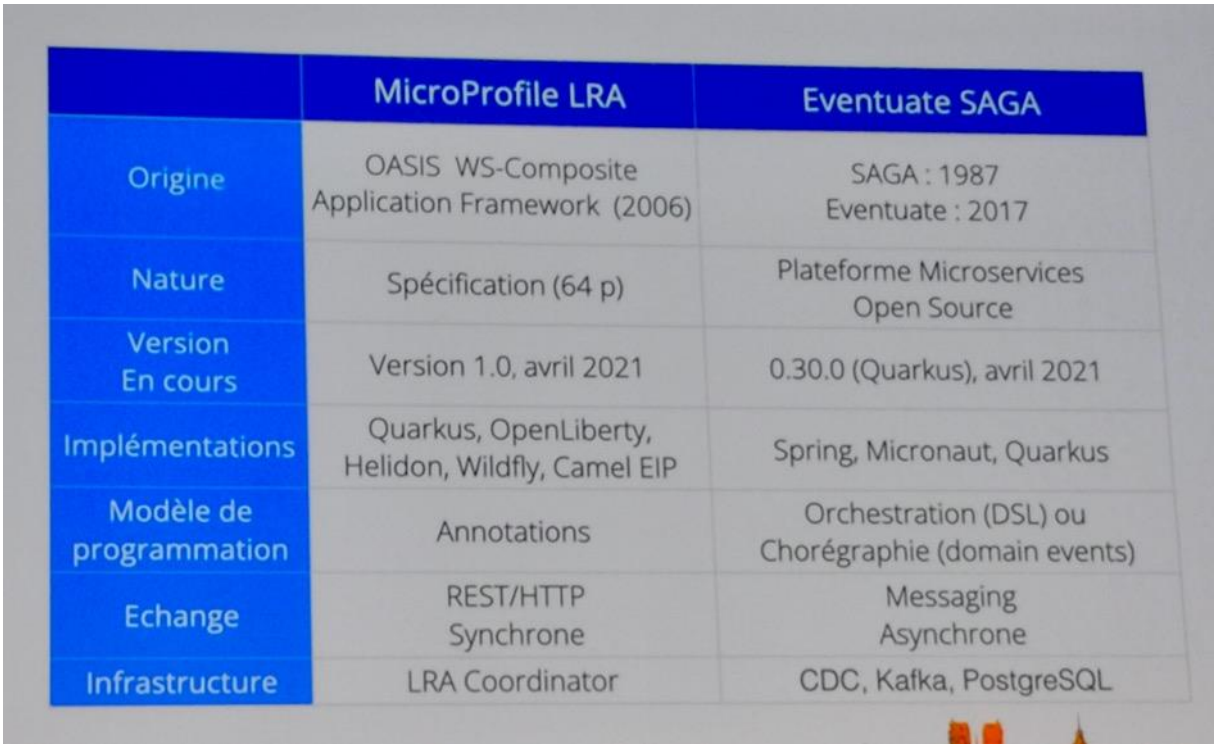
Principes de coordination :

- Des transactions locales
- Une coordination globale légère, faiblement couplée
- Des opérations de compensation explicites
- Perte de l'isolation globale : de ACID à ACD

4 solutions sont identifiées :

1. MicroProfile
2. Eventuate
3. AxonIQ
4. SEATA (Alibaba)

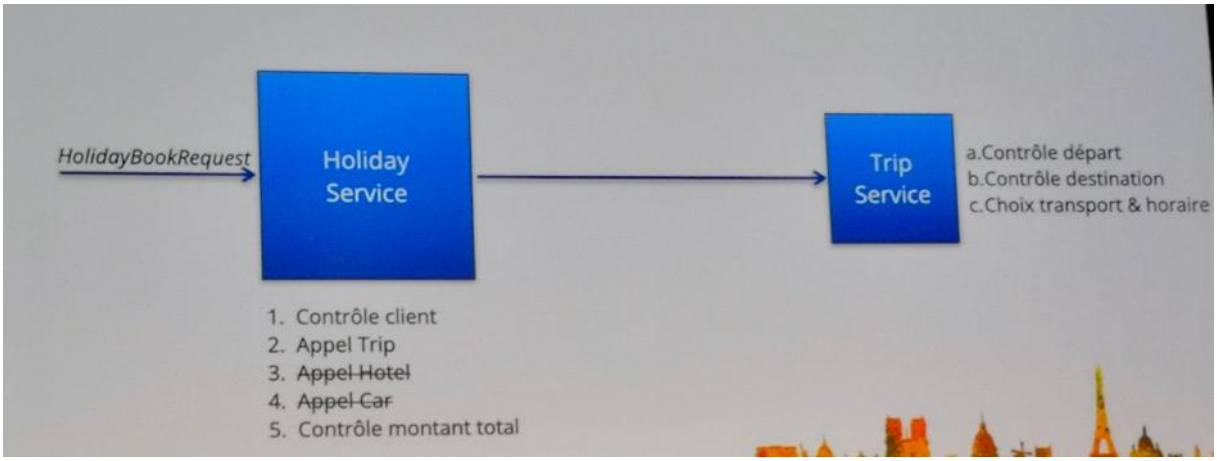
MicroProfile est un ensemble de spécifications.
Jean-François a découvert Eventuate dans le livre Microservices Pattern de Chris Richardson.



	MicroProfile LRA	Eventuate SAGA
Origine	OASIS WS-Composite Application Framework (2006)	SAGA : 1987 Eventuate : 2017
Nature	Spécification (64 p)	Plateforme Microservices Open Source
Version En cours	Version 1.0, avril 2021	0.30.0 (Quarkus), avril 2021
Implémentations	Quarkus, OpenLiberty, Helidon, Wildfly, Camel EIP	Spring, Micronaut, Quarkus
Modèle de programmation	Annotations	Orchestration (DSL) ou Chorégraphie (domain events)
Echange	REST/HTTP Synchrone	Messaging Asynchrone
Infrastructure	LRA Coordinator	CDC, Kafka, PostgreSQL

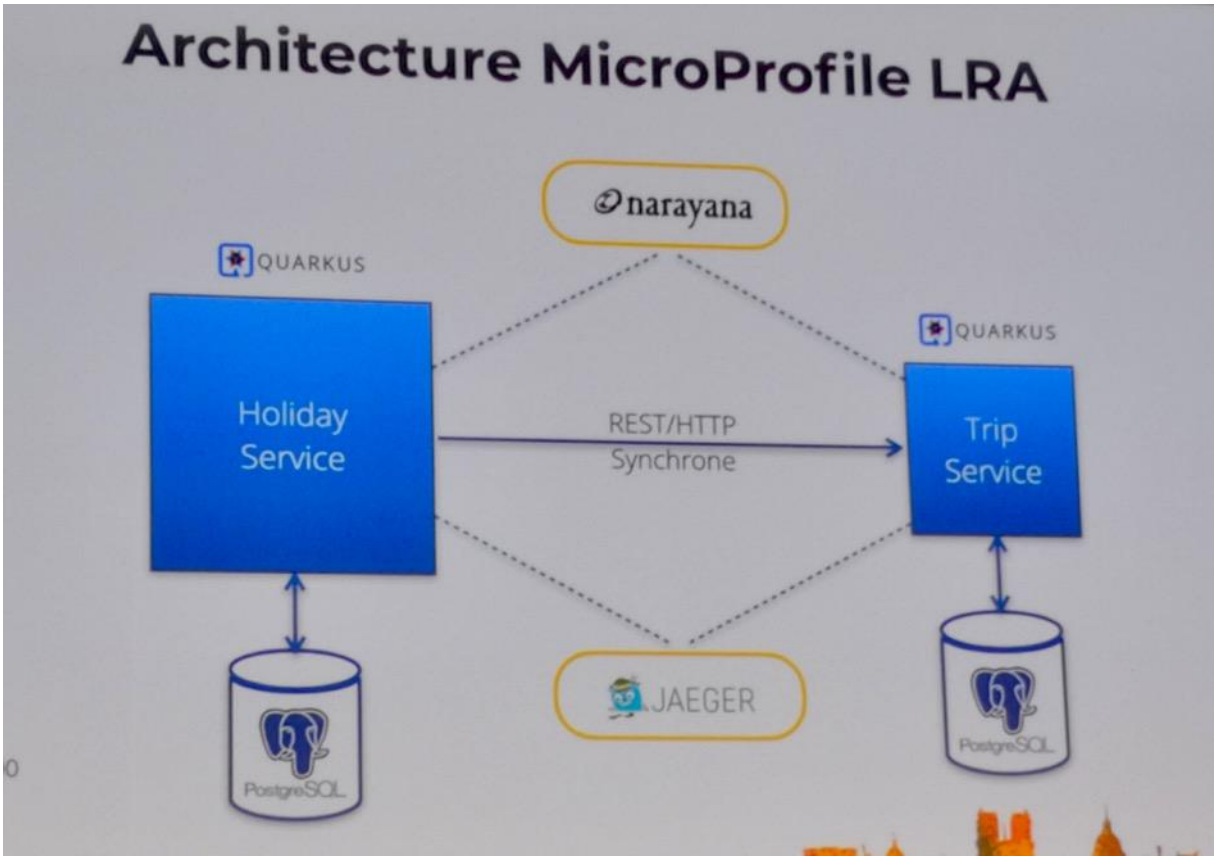
MicroProfile LRA se base sur une spécification OASIS WS-Composite. Nécessite de l'infra : un LRA Coordinator.

Eventuate SAGA : SAGA a commencé en 1987, Eventuate date de 2017. Beaucoup d'évolutions en 2021. Propose 2 modèles de programmation. Échanges obligatoirement en asynchrones. Nécessite 3 composants d'infrastructure : CDC (fournit), Kafka, base PostgreSQL

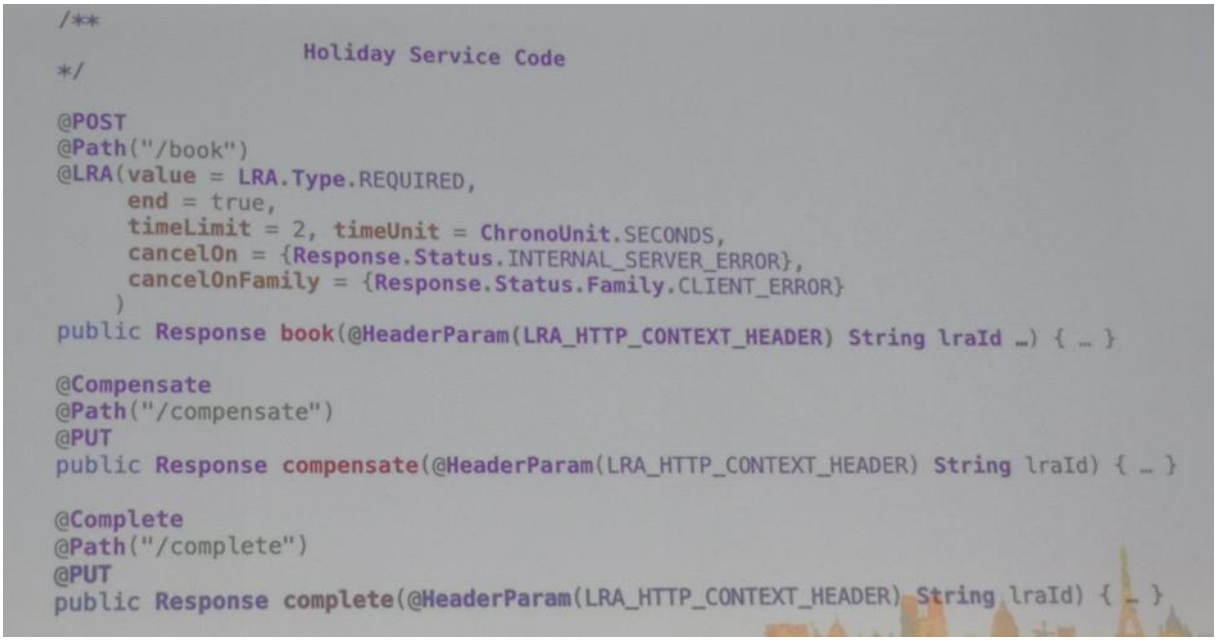


Jean-François présente une démo avec une architecture volontairement allégée : les appels vers les services Hotel et Car ont été volontairement retirés. Plusieurs cas d'erreur.

Démo 1 - Utilisation de la solution OSS Narayana



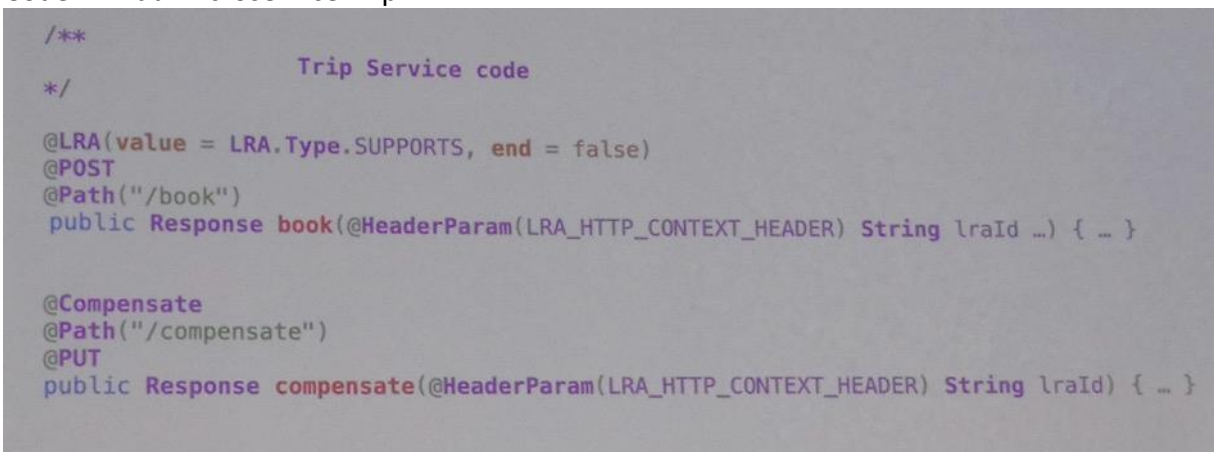
Annotation @LRA positionnée en amont du service métier.
Le développeur dot obligatoirement implémenter une méthode de compensation afin d'ajouter le code de compensation en cas d'erreur.
L'en-tête HTTP ***lraId*** est générée par le coordinateur : cela permet un couplage faible entre les différents microservices (une chaine passée par en-tête http).



```
/**
                Holiday Service Code
*/

@POST
@Path("/book")
@LRA(value = LRA.Type.REQUIRED,
     end = true,
     timeLimit = 2, timeUnit = ChronoUnit.SECONDS,
     cancelOn = {Response.Status.INTERNAL_SERVER_ERROR},
     cancelOnFamily = {Response.Status.Family.CLIENT_ERROR}
    )
public Response book(@HeaderParam(LRA_HTTP_CONTEXT_HEADER) String lraId …) { … }

@Compensate
@Path("/compensate")
@PUT
public Response compensate(@HeaderParam(LRA_HTTP_CONTEXT_HEADER) String lraId) { … }

@Complete
@Path("/complete")
@PUT
public Response complete(@HeaderParam(LRA_HTTP_CONTEXT_HEADER) String lraId) { … }
```

Le modèle LRA parait simple dans le cas facile sans erreur.
Dans le cas d'une compensation, la méthode ***compensate*** peut être appeler de manière concurrente à la méthode ***@LRA*** exécutée sur le service distant. Nécessité de blinder le code pour gérer cet aspect. L'appel du *compensate* peut avoir lieu avant la fin du traitement distant. On persiste donc le *lraId* rejeté en base afin de pouvoir faire un check

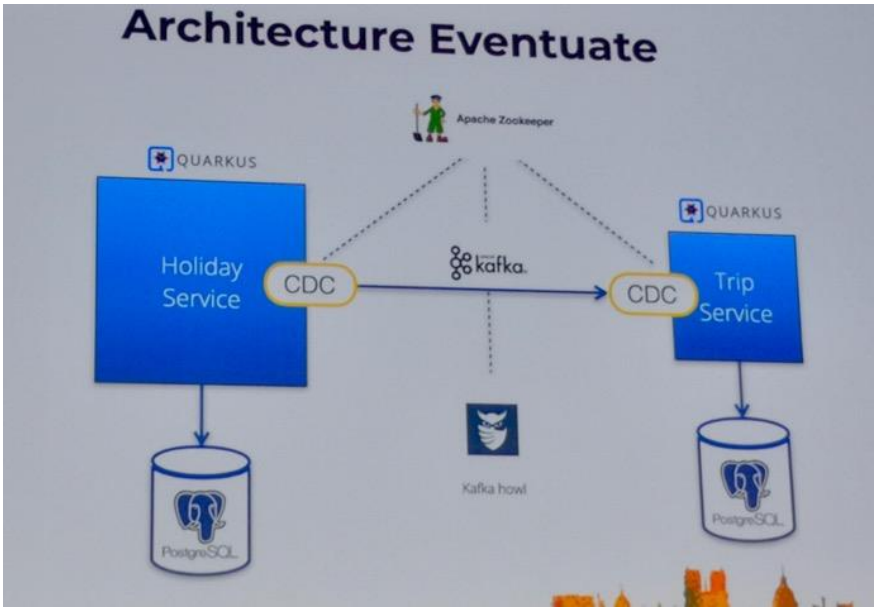
Code LRA du microservice Trip :



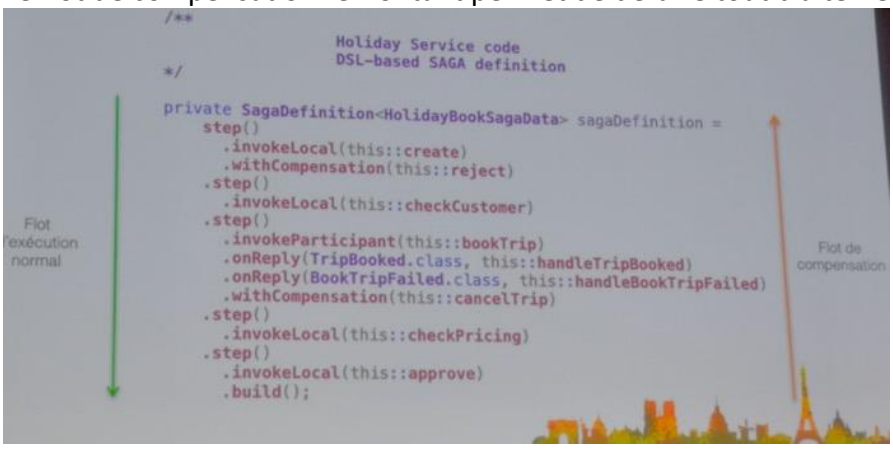
```
/**
                Trip Service code
*/

@LRA(value = LRA.Type.SUPPORTS, end = false)
@POST
@Path("/book")
public Response book(@HeaderParam(LRA_HTTP_CONTEXT_HEADER) String lraId …) { … }


@Compensate
@Path("/compensate")
@PUT
public Response compensate(@HeaderParam(LRA_HTTP_CONTEXT_HEADER) String lraId) { … }
```

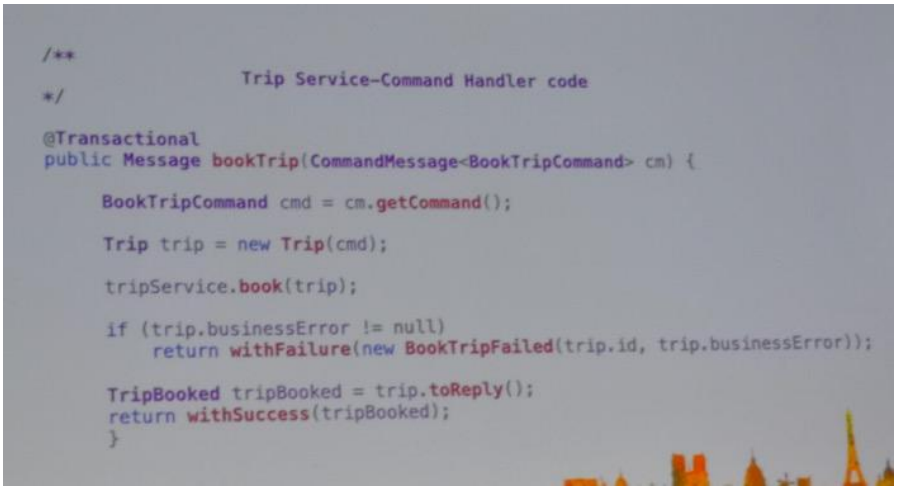
Démo 2- Utilisation de la solution SAGA Eventuate

Eventuale requière une architecture asynchrone.
Zookeeper surveille Kafka et le Change Data Capture (CDC). Il permet de gérer la redondance des CDC en production.

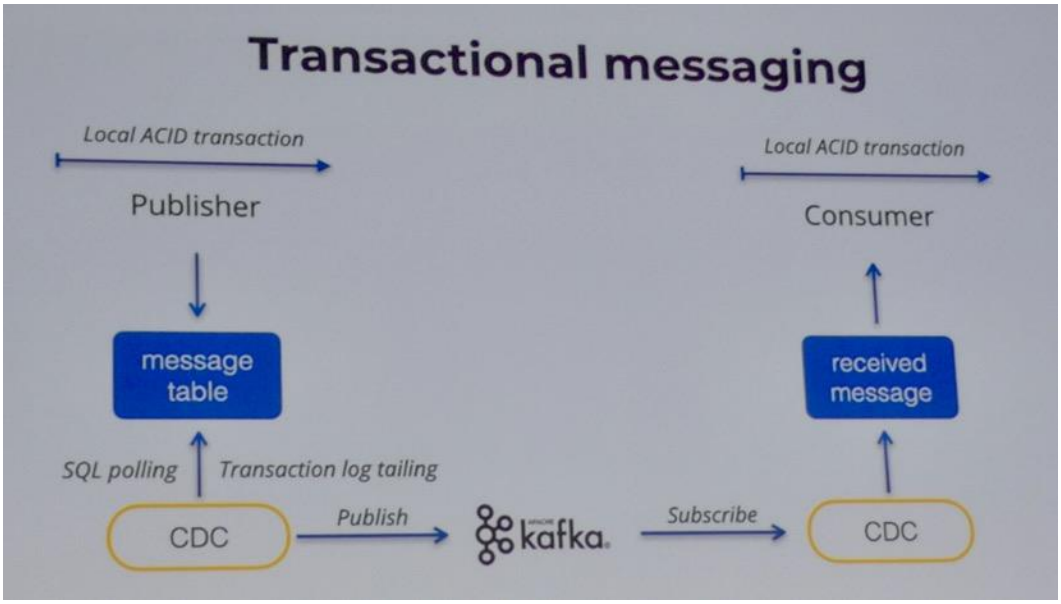


Le flot de compensation remontant permet de défaire tout traitement :



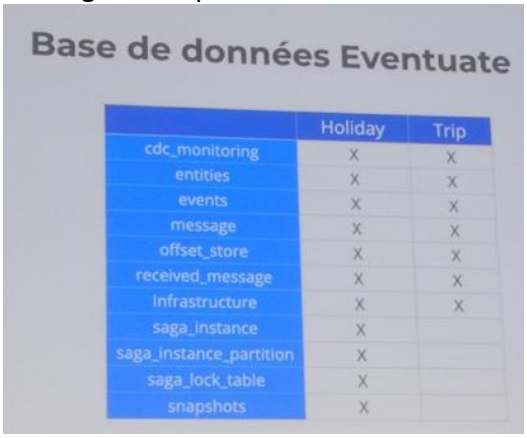


Dans le code métier, l'appel aux méthodes ***withFailure*** et ***withSuccess*** sont requises pour alimenter le résultat de l'appel dans Kafka.



Fonctionnant sur MySQL et PostgreSQL, le CDC scrute le log de transactions du publisher et l'envoie au consumer.
L'intérêt vient de conserver l'usage des transactions ACID.

Les sagas sont persistées dans des tables dédiées à Eventuate.



Les CDC sont développés en Spring Boot et exposent des healthchecks via les actuators.

Lors de la démo, Jean-François utilise l'UI Kowl pour parcourir les topics Kafka.

Conclusion

Narayana composant clé de MicroProfile LRA doit être particulièremet surveillé en production.
Les 2 démos ont le même nombre de ligne de code : 1600 LOC.
Eventuate demande beaucoup de configuration et un peu partout. Cout d'entrée très élevé.

Dans son livre, Chris Richardson prône l'asynchrone, ce qui donne à Eventuate.

	MicroProfile LRA	Eventuate
Composants clés SPOF potentiels	Narayana	CDC et Kafka
Courbe d'apprentissage	*****	***
Simplicité de codage	****	***
Simplicité de configuration	*****	**
Couplage fonctionnel	****	***
Couplage technique	***	****
Observabilité	**	****
Généricité	***	*****

Pour aller plus loin

2 démos sur GitHub :

- https://github.com/jefrajames/lra-demo
- https://github.com/jefrajames/saga-demo

LRA : le blog NArayana, les exemples Quarkus, Open Liberty, Helidon

Eventuate : le site eventuate.io et les exemples