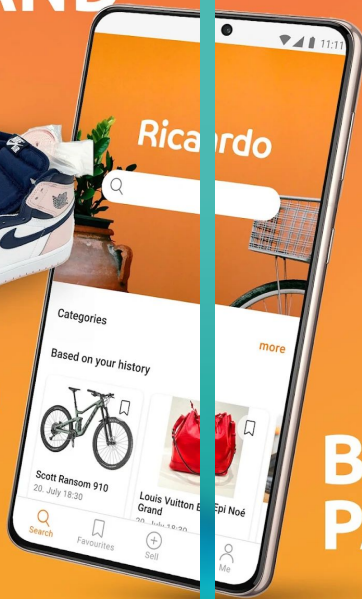


Kafka en prod

[REX] Du concept à l'implémentation qui roule

BUY AND
SELL

all things
second-hand



BECOME
PART

of the most sustainable
community
in Switzerland



4M
members



Edouard Bavoux

Apache Kafka est...

“Plateforme de diffusion de données en continu (streaming)”

“Distribuée”

“Résiliente (Réplication)”

“Proche temps réel”

“N producers vers N consumers”

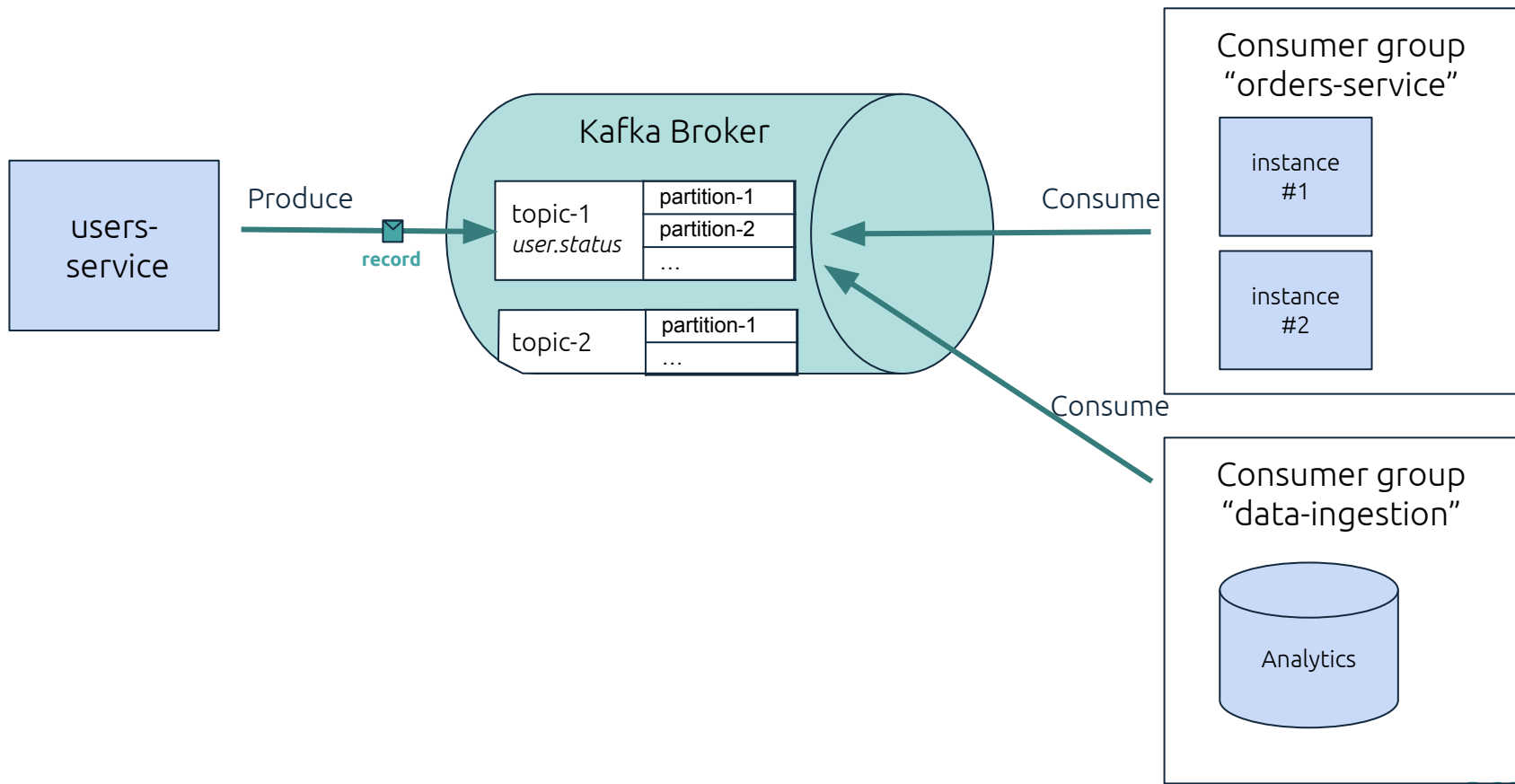
Apache Kafka est...

“Open-Source”

“Géré par la fondation Apache”

“Maintenu et développé majoritairement par Confluent”

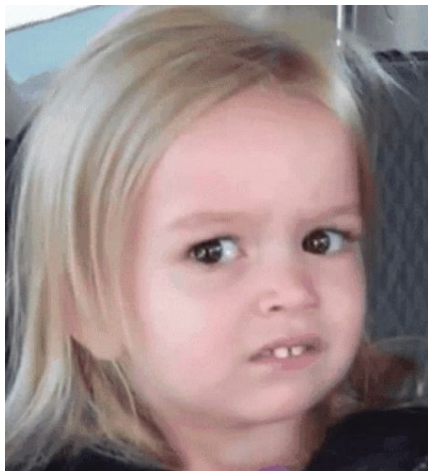
Kafka :



Talk Kafka

1. Un message
2. Publier
3. Consommer

Talk Kafka



1. **Un message**
2. Publier
3. Consommer

Un message : Clé - Valeur - Headers

Choix de la clé :

- Chaîne de caractère
- Optionnelle
- Regroupement logique de messages

Headers :

- Métadonnées, Versioning, Authentification, Tracing...

Un message

	API web-service	Kafka
Format	JSON	?
Principes	REST	?
Documentation	Swagger	?
Génération de code	Openapi-generator	?
Evolutivité	Never return an array,...	?
Conventions	Snake Case for query params,...	?

Un message - JSON

```
"...4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00..."
```

→ JSON Encoding for most topics

Why :

- Byte encoding is more complex to debug
- Performance gain is not worthy for most of our cases...

Un message - Avec un Schéma



```
1  syntax = "proto3";
2  package ch.ricardo.users.schema;
3
4  message User {
5      int32 id = 1;
6      string name = 2;
7      UserDetails user_detail = 3;
8      repeated string roles = 4;
9  }
```



```
1  {
2      "type": "record",
3      "namespace": "ch.ricardo.users.schema",
4      "name": "User",
5      "fields": [
6          {"name": "id", "type": "int"},
7          {"name": "name", "type": "string"},
8          {"name": "user_detail", "type": "ch.ricardo.user.schema.UserDetail"},
9      ],
10     {
11         "name": "roles",
12         "type": {
13             "type": "array",
14             "items": "string"
15         }
16     }
17 ]
18 }
```

Un message - Schéma Evolutif

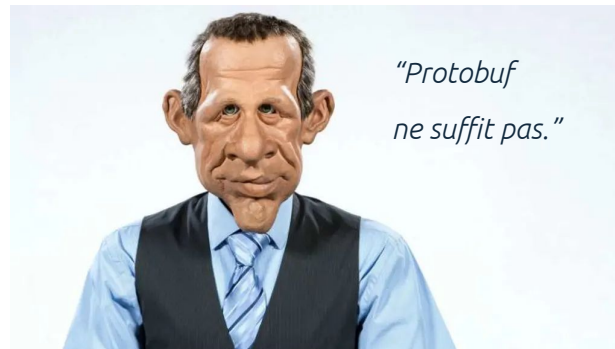
Backward Compatibility :

- Pour rejouer des messages du passé

Forward Compatibility :

- Pour ne pas mettre à jour tous les consumers à chaque évolution

→ *Les deux mon capitaine !*



Un message - Schéma Evolutif

- *Suivre les bonnes pratiques pour éviter les breaking changes*

```
message User {  
  string name = 1;  
  int age = 2;  
}
```



```
message User {  
  reserved 1; // deprecated field [name]  
  int age = 2;  
  string first_name = 3;  
  string last_name = 4;  
}
```

- *Mettre en place le tooling pour s'en assurer*



- *Configurer les consumers pour ignorer les champs inconnus lors de la désérialisation.*

Un message - Documentation et code partagé

- Github repo “proto-schemas”
 - CI/CD includes :
 - Breaking changes detection with Buf
 - Code generation
 - Artifact publication in CI/CD

↳ **We don't use schema registry... for historical reason**

Un message - Company Conventions

- Topic naming
 - `pattern : <project>.<domain>.<event-name>`
 - `example : ricardo.users.status`
- Tips
 - Bloquer la création des topics on-the-fly !
 - `"uers.status"`
 - `"users.statutes"`
 - `"topic"`
 - `"partition"`
 - `"_"`
 - `"replication"`
 - ...

Un message - Company Conventions

- Envelope standard

```
message UserEnvelope {  
    string event_uuid = 1;  
    int64 occurred_at = 2;  
    shared.EventType event_type = 3;  
    User payload = 4;  
    User prev_payload = 5;  
}
```


Un message

	API web-service	Kafka
Format	JSON	JSON
Principes	REST	Kafka best practices
Documentation	Swagger	Protobuf
Génération de code	Openapi-generator	Buf
Evolutivité	Never return an array,...	Buf
Conventions	Snake Case for query params,...	Envelope Standard

Talk Kafka



1. Un message
2. **Publier**
3. Consommer

Publier un message

Nos itérations



#1. Client Kafka dans l'application métier

#2. Kafka Connect sur la table métier

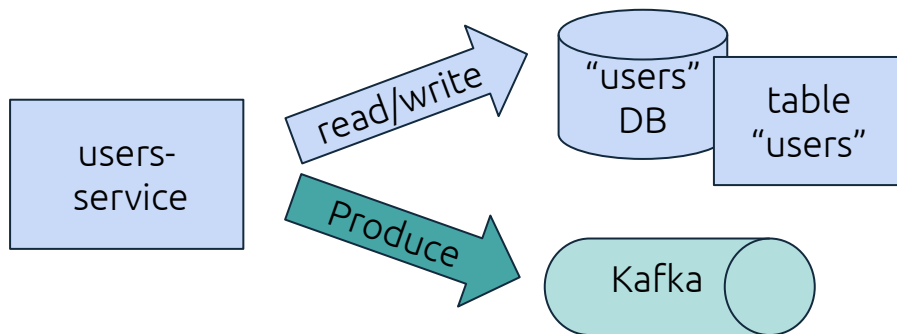
#3. Transactional Outbox Pattern

#4. Transactional Outbox Pattern avec Kafka Connect

#5. Transactional Outbox Pattern avec Kafka Connect et CDC

Publier un message: itération #1

→ Client Kafka dans l'application métier



- Dual Write ! Perte de la garantie de publication des évènements
- Dépendance forte sur la disponibilité de Kafka

Publier un message

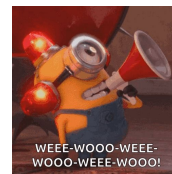
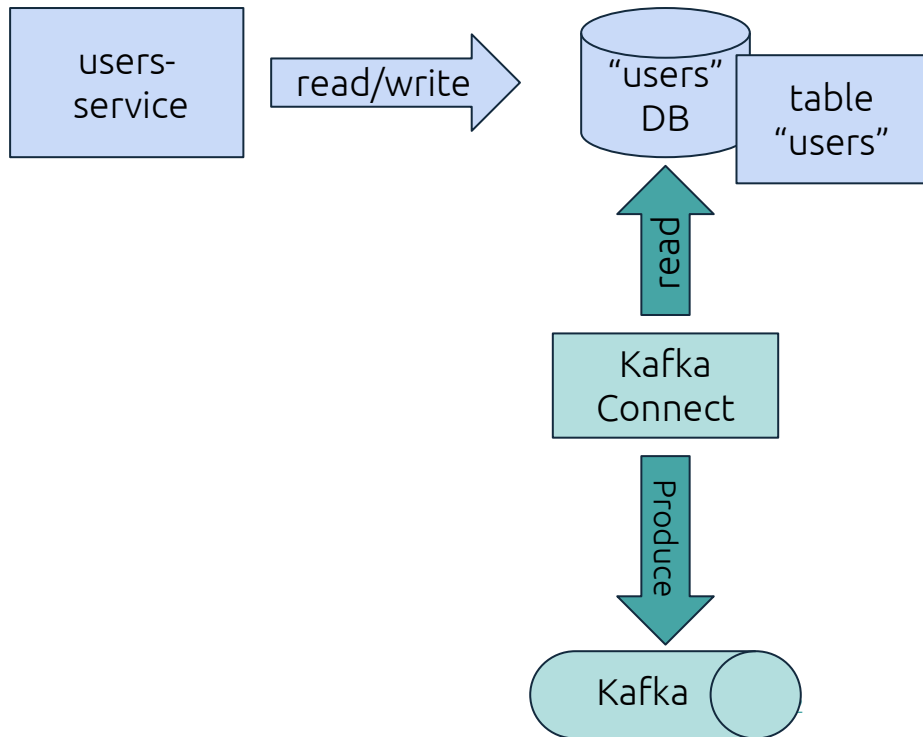
Nos itérations



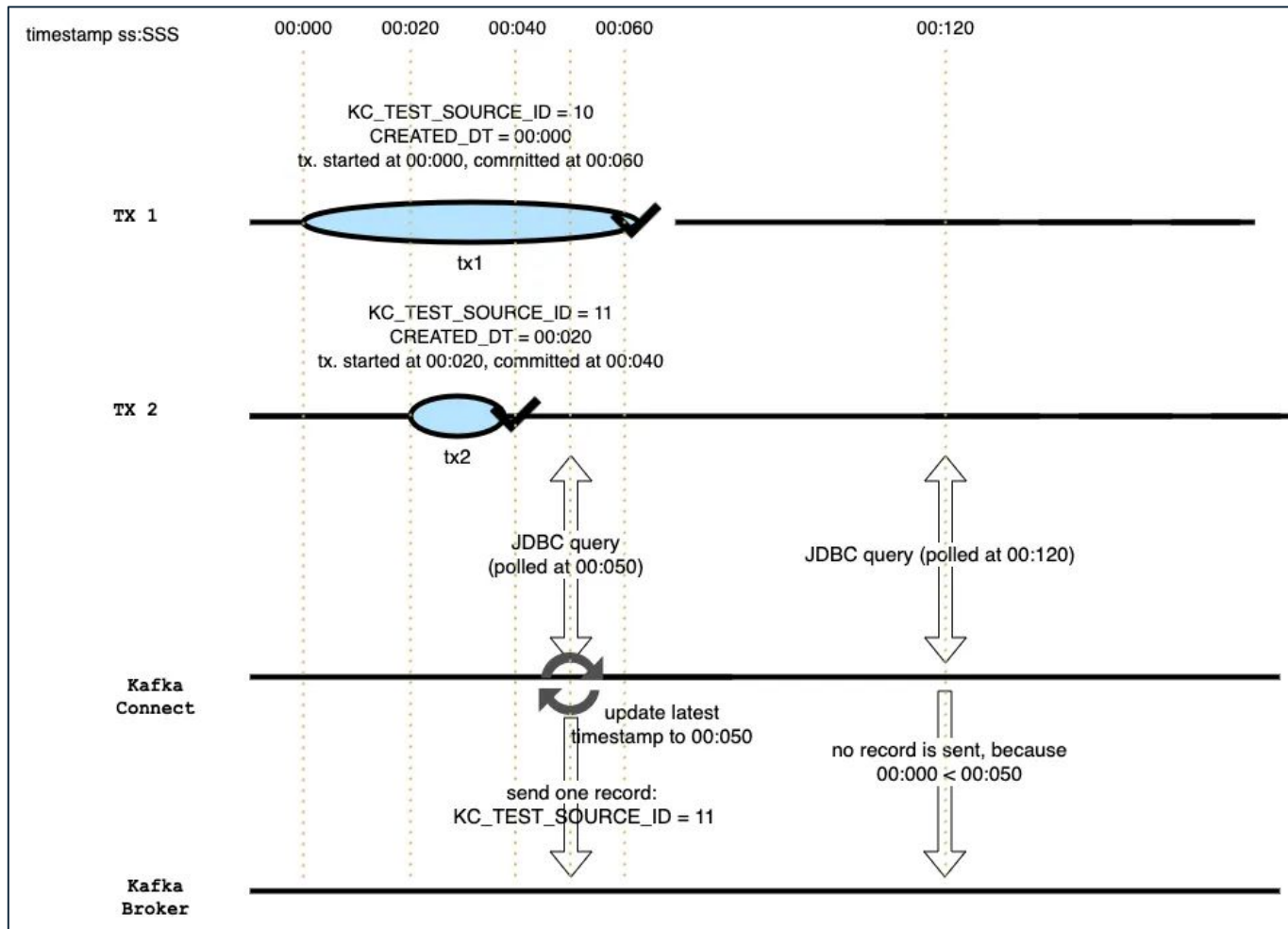
1. Client Kafka dans l'application métier
2. Kafka Connect sur la table métier
3. Transactional Outbox Pattern
4. Transactional Outbox Pattern avec Kafka Connect
5. Transactional Outbox Pattern avec Kafka Connect et CDC

Publier un message: itération #2

→ Kafka Connect sur la table métier




- Pb de perf , et Lock
- Couplage fort entre l'entité métier et le message
- Perte d'événements possible (Race conditions)



source : [Event sourcing with Kafka Connect: inconsistency pitfalls & solutions \(Medium Article\)](#)

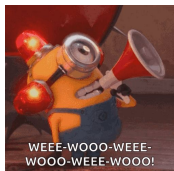
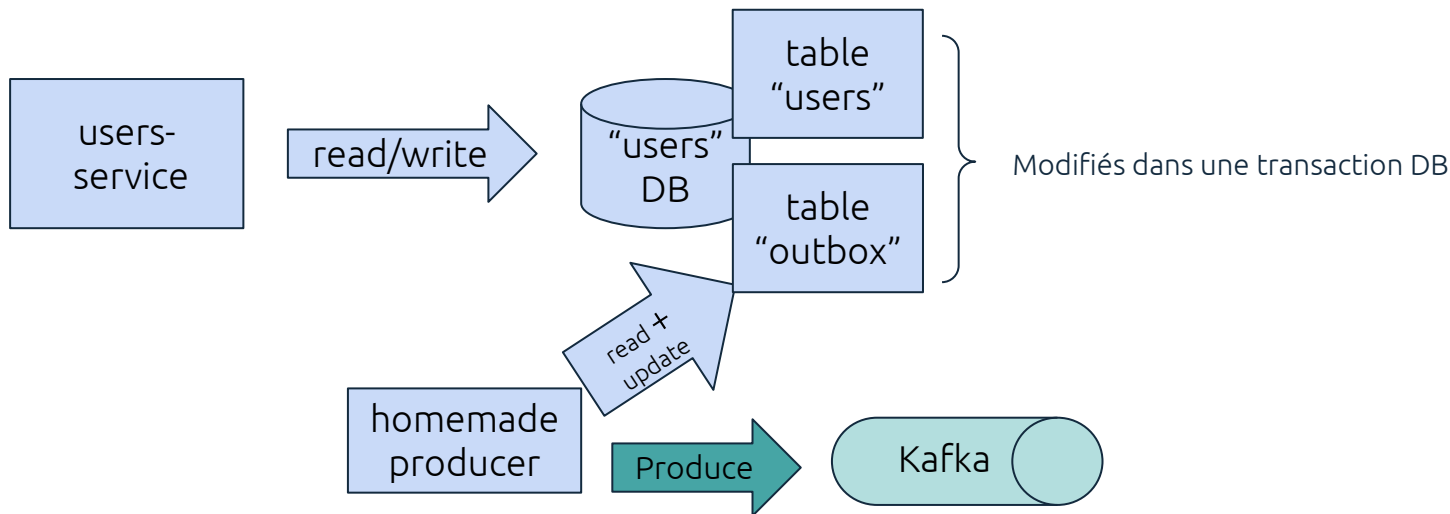
Publier un message

Nos itérations

1. Client Kafka dans l'application métier
2. Kafka Connect sur la table métier
-  3. Transactional Outbox Pattern
4. Transactional Outbox Pattern avec Kafka Connect
5. Transactional Outbox Pattern avec Kafka Connect et CDC

Publier un message: itération #3


→ Transactional Outbox Pattern



- Pb de performance et scaling
 - Une seule instance travaille à la fois pour éviter les doublons.
 - Un select, puis un update
 - Lors de batch updates, ça pouvait bloquer pendant minutes / heures
- Code boilerplate inutile à maintenir, app à déployer... qui dérive.

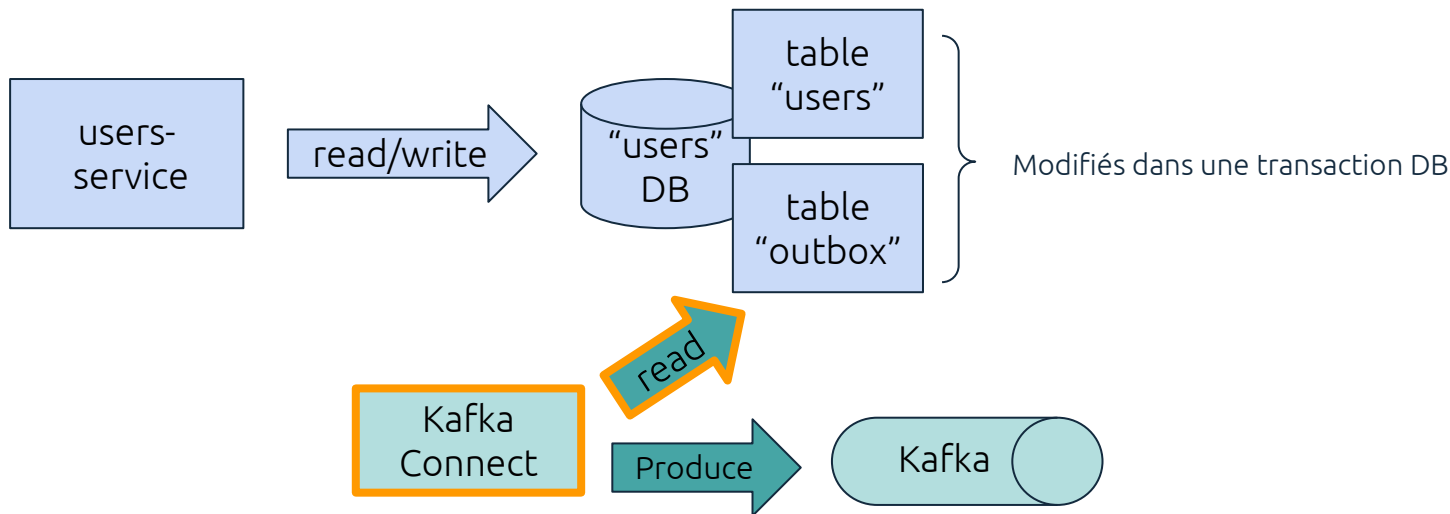
Publier un event

Nos itérations

1. Client Kafka dans l'application métier
2. Kafka Connect sur la table métier
3. Transactional Outbox Pattern
-  4. Transactional Outbox Pattern avec Kafka Connect
5. Transactional Outbox Pattern avec Kafka Connect et CDC

Publier un message: itération #4

→ Transactional Outbox Pattern avec Kafka Connect.



- Toujours un compromis sur le délai de polling (`poll.interval.ms`)
- Et puis la solution ultime n'est pas plus chère

Publier un event

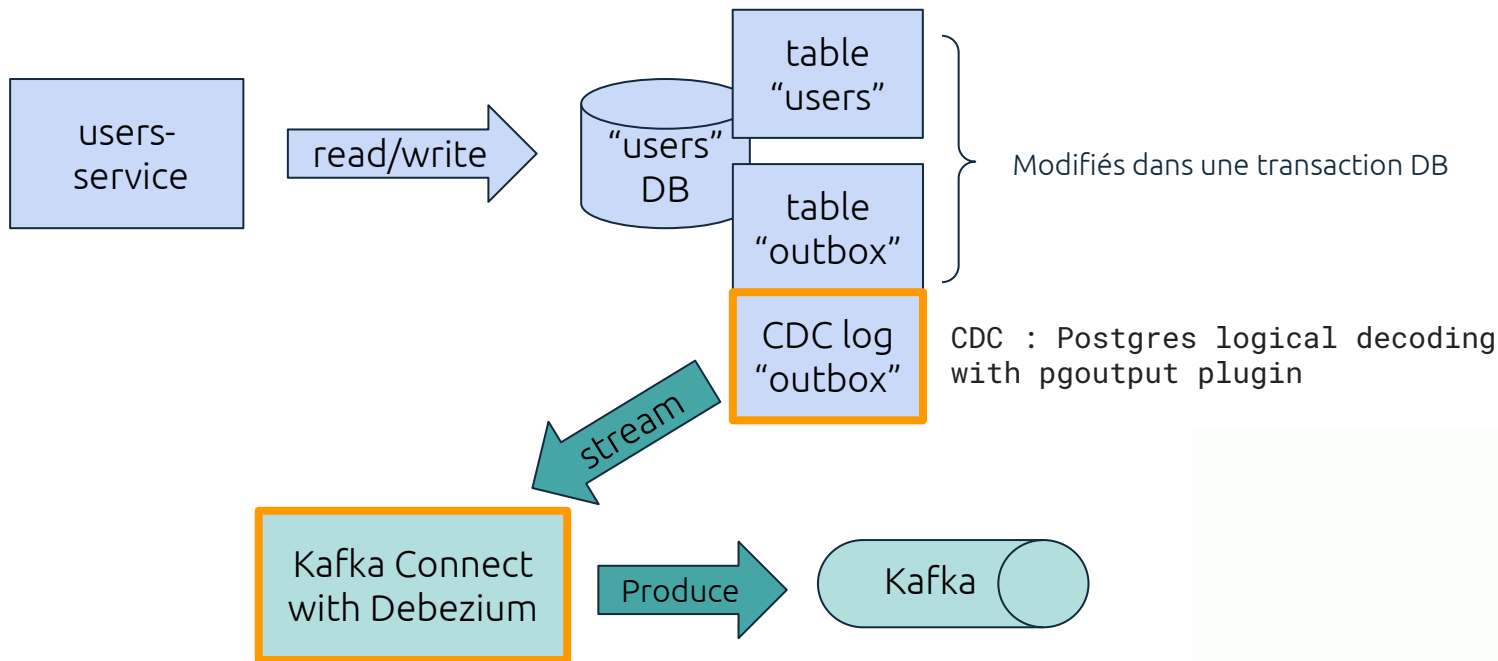
Nos itérations

1. Client Kafka dans l'application métier
2. Kafka Connect sur la table métier
3. Transactional Outbox Pattern
4. Transactional Outbox Pattern avec Kafka Connect
5. Transactional Outbox Pattern avec Kafka Connect et CDC



Publier un message: itération #5

→ Transactional Outbox Pattern avec Kafka Connect et CDC.



Publier des events : Bonus

Un intérêt de la clé du message

- Chronologie

Différentes “Partitioning strategy”

- Chacun son tour (Round robin)
- Hash de la clé, puis modulo le nombre de partitions
 - ... Algo de hash par défaut :
 - Murmur2 pour la librairie producer java de Apache Kafka
 - CRC32 pour Librdkafka (C/C++, used for Python, Go, .NET, C#)

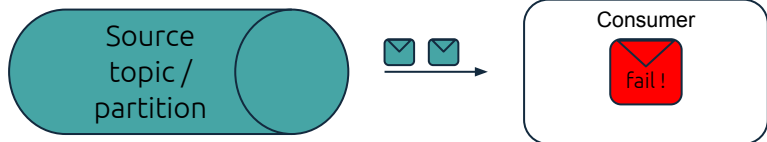
→ On a dû définir une custom implem en Go

Talk Kafka



1. Un message
2. Publier
- 3. Consommer**

Error Handling Pattern



- **Stop everything**
 - Manual intervention required



- **Blocking retry**
 - Every 3 seconds, exponential backoff,...
 - But the following event wait in queue.



- **Skip it**
 - But well... the process won't happen



- **Send to dead-letter topic**
 - But you shouldn't forget them (Set up alarms)
 - Manual intervention required
 - And chronology is lost



- **Send to retry topic**
 - But chronology is lost



- **Retry topic + Redirect topic + in-memory store to keep chronological order...**
 - "Perfect solution"
 - But way too complex

Architecture du code et déploiements

Nos itérations



#1. Une app

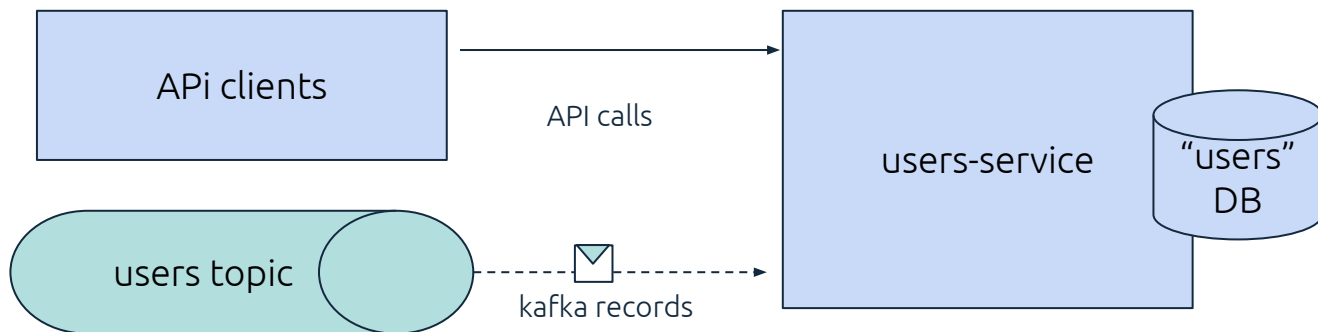
#2. Deux apps

#3. Mono-repo, deux déploiements

3. Consommer : Architecture micro-service

#1 : Une app

- Un repo Github
- Un déploiement k8s (avec auto-scaling)
- Pb :
 - Autoscaling lié au trafic sur l'API cause des rebalance sans arrêts
 - Complexité du code
 - Dépendance des pannes
 - Ressources non optimisées



Architecture du code et déploiements

Nos itérations

#1. Une app



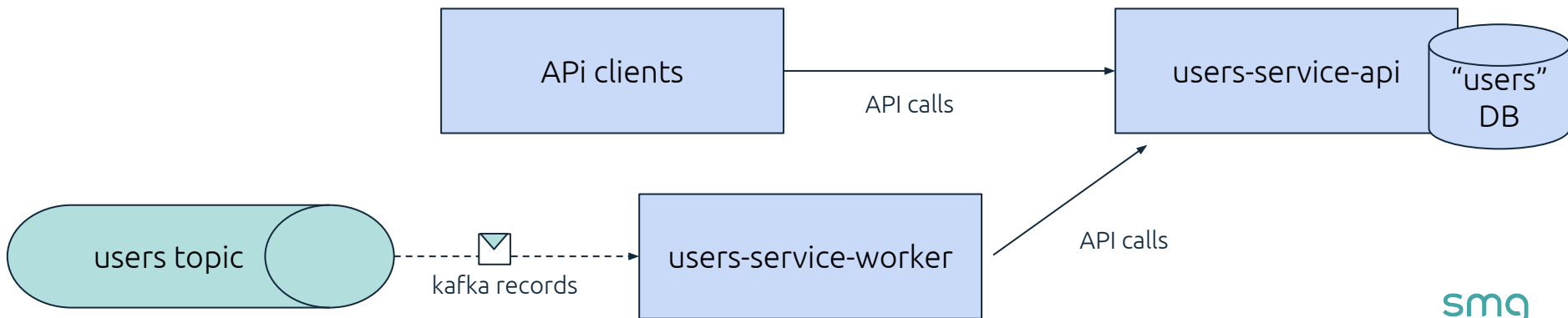
#2. Deux apps

#3. Mono-repo, deux déploiements

3. Consommer : Architecture micro-service

#2 : Deux Apps

- Deux repos Github
 - Deux déploiement k8s (avec auto-scalings indépendants, sur métriques dédiées)
-
- Pb :
 - duplication de code business
 - complexité accrue: le worker a besoin d'api différentes.



Architecture du code et déploiements

Nos itérations

#1. Une app

#2. Deux apps



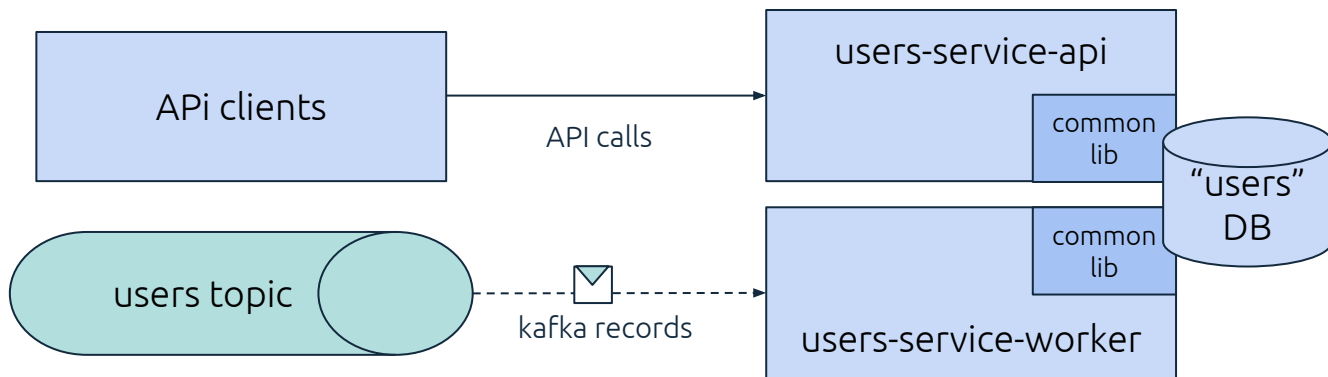
#3. Mono-repo, deux déploiements

3. Consommer : Architecture micro-service

#3 : Mono-repo, 2 déploiements

- Un seul repo Github
 - librairie common (business logic, repository, clients to external API, ...)
 - code api
 - code worker
- Deux déploiement k8s (avec auto-scalings indépendants, sur métriques dédiées)

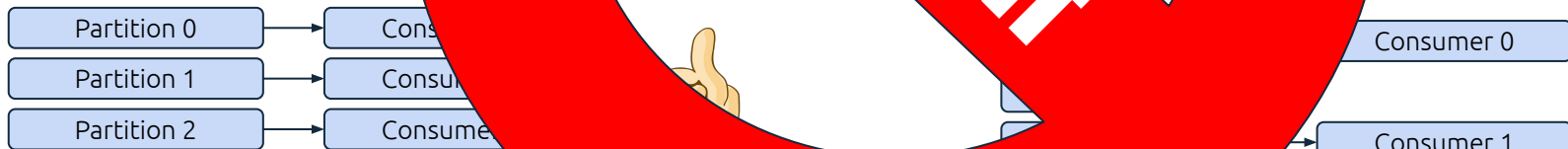
```
▼ incidents-api ~/sources/incidents-api
> api
▼ common
> worker
```



Consommer : Scalabilité du worker

Si la clé est bien choisie, la charge de processeur est équilibrée.

Donc dans l'idéal : Nombre de partitions = Nombre de consommateurs.



- ❖ 12 partitions par exemple : Multiplication du nombre de partitions.
- ❖ Logique des scaling non linéaire...
- ❖ Idéalement basée sur l'accélération du lag.
- ❖ Chaque scale up/down déclenche un rebalancing.

Bonus : Top #1 Failure...

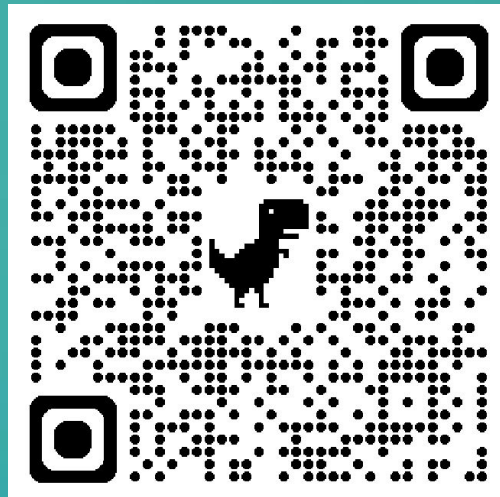


- ❖ Renvoyer 3 mois d'historique de mails à tous nos utilisateurs...

Conclusion : Si je devais démarrer aujourd'hui

01	Un message	<ul style="list-style-type: none">- Protobuf / Avro- Schema registry- JSON par défaut- Conventions : Métadonnées, nommage
02	Publier	<ul style="list-style-type: none">- Outbox pattern- CDC- Kafka Connect / Debezium
03	Consommer	<ul style="list-style-type: none">- Mono-repository- Pas d'auto-scaling sur le consumer- 3 patterns : Skip / Blocking retry / Retry topic

Merçi



Feedbacks

Merçi Etienne / Erwan !



smg

Annexes

Sources

<https://aws.amazon.com/fr/what-is/pub-sub-messaging/>

<https://danielfoo.medium.com/event-driven-architecture-vs-restful-architecture-99d4a98222cc>

<https://www.journaldunet.com/big-data/1439692-confluent-quelle-est-la-societe-derriere-apache-kafka/>

<https://medium.com/adidoescode/event-sourcing-with-kafka-connect-inconsistency-pitfalls-solutions-11a771eb697>

<https://www.journaldunet.com/big-data/1142059-comment-fonctionne-la-fondation-apache/#confirmation>

<https://www.redhat.com/fr/topics/integration/what-is-apache-kafka>

<https://debezium.io/documentation/reference/stable/connectors/postgresql.html>

<https://medium.com/@kiranprabhu/kafka-topic-naming-conventions-best-practices-6b6b332769a3>

<https://mikemybytes.com/2022/07/11/json-kafka-and-the-need-for-schema/>

<https://support.confluent.io/hc/en-us/articles/360042745491-Introducing-Confluent-Platform-5-5>

<https://medium.com/slalom-technology/introduction-to-schema-registry-in-kafka-915ccf06b902>

<https://protobuf.dev/programming-guides/proto3/#json>

<https://docs.confluent.io/platform/current/schema-registry/fundamentals/schema-evolution.html#compatibility-types>

<https://www.confluent.io/blog/standardized-hashing-across-java-and-non-java-producers/>

<https://docs.confluent.io/platform/current/clients/producer.html>

<https://www.confluent.io/blog/error-handling-patterns-in-kafka/>