



Migration d'une Architecture **Microservice** vers une Architecture **Event-Driven** **Microservice** avec Kafka



<https://github.com/fouomene/poc-jcdecaux-spring-kafka-websocket>

<https://github.com/fouomene/poc-microservice-edge-spring-cloud>

<https://github.com/fouomene/poc-event-driven-microservice-edge-kafka-spring-cloud>

Daniel FOUOMENE



daniel.fouomene@afrinnov.xyz





Plan



I. Event-Driven Architecture

1. Introduction
2. Event Broker : Apache Kafka
3. Mise en œuvre : POC App JCDecaux Station avec Spring Kafka, Spring Websocket

II. Microservice Architecture

1. Introduction
2. Edge Microservice : Spring Cloud
3. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud

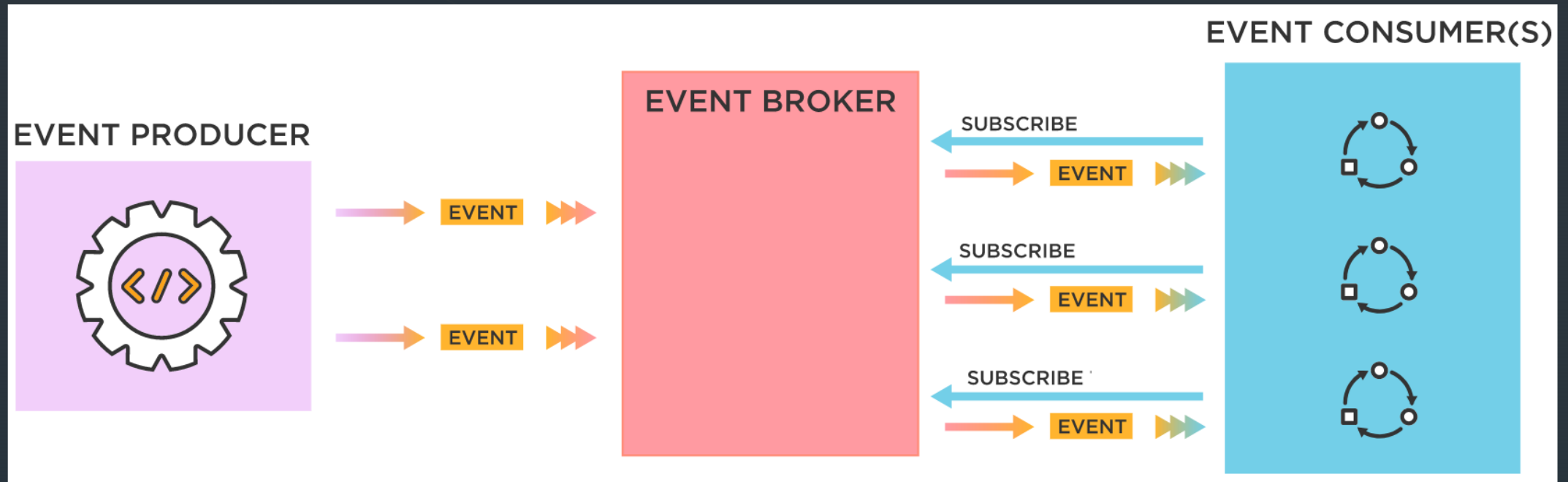
III. Event-Driven Microservice Architecture

1. Introduction
2. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud, Spring Kafka

IV. Références

I. Event-Driven Architecture

1. Introduction



I. Event-Driven Architecture

1. Introduction

- **Event-Driven Architecture** : Architecture Orientée Evénements
 - Un modèle d'architecture logicielle utilisé pour la conception d'applications.
 - La structure centrale de la solution repose
 - sur la **capture, la communication,**
 - le **traitement et la persistance des événements (Events).**
- Souvent appelée communication « **Asynchrone** »
 - Cela signifie que le producteur (**Producer**) et le consommateur (**Consumer**) de l'événement n'ont pas besoin d'attendre l'autre pour passer à la tâche suivante.
 - Lorsqu'un événement a été détecté, il est transmis du producteur d'événements au consommateur via des canaux d'événement ou Bus d'événement (**Event Broker**).

I. Event-Driven Architecture

1. Introduction

- **Faiblement couplée**

- Les producteurs d'événements ignorent quels consommateurs d'événements écoutent un événement et que chaque événement ignore les conséquences de son apparition.

- **Très scalable**

- Facilité à multiplier indépendamment les instances des producteurs et consommateurs, en fonction de leur charge de travail.

- **L'exposition des événements en temps réel**

- Facilité d'abonner de nouveaux consommateurs aux événements produits sans impact sur les services en place, en particulier le service producteur de l'événement.

- **Une Approche et non un langage**

- Les applications orientées événements peuvent être créées à l'aide de n'importe quel langage de programmation.

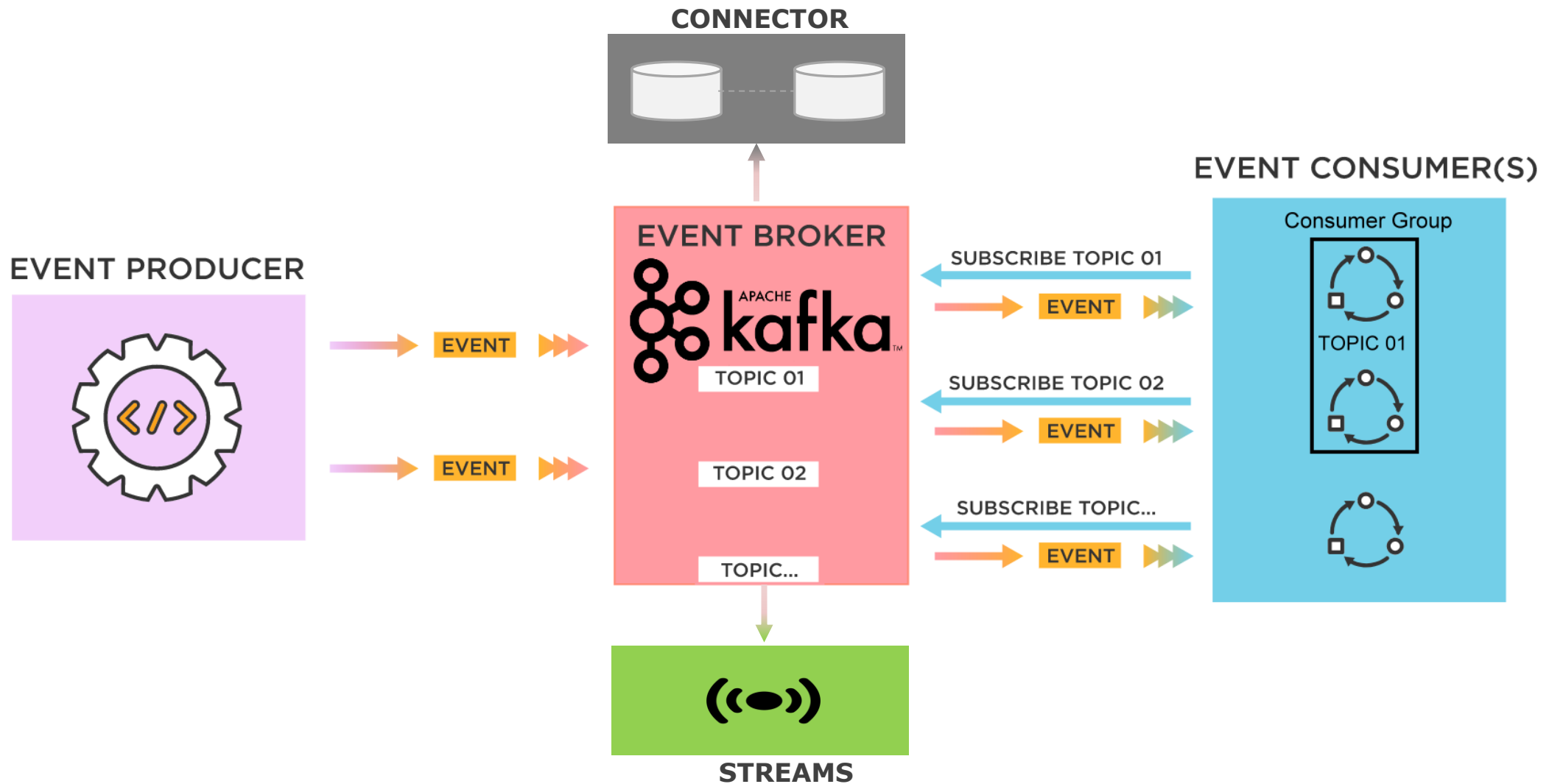
I. Event-Driven Architecture

1. Introduction

- Les challenges principaux à relever dans la mise en place d'une architecture event-driven sont les suivants :
 - **La cohérence entre les services n'est pas immédiate, on parle de cohérence à terme.**
 - Il faudra intégrer cet aspect dans la conception des traitements et veiller à ce **que le délai de synchronisation ne soit pas trop important**, en s'assurant par exemple que la consommation des événements soit plus rapide que leur production.
 - **S'assurer que tous les événements soient acheminés et traités,**
 - sans quoi deux services pourraient se trouver définitivement désynchronisés pour certaines entités
 - **Être vigilant sur le paramétrage du bus d'événements ainsi que des applicatifs producteurs et consommateurs d'événements.**

1. Event-Driven Architecture

2. Event Broker : Apache Kafka



I. Event-Driven Architecture

2. Event Broker : Apache Kafka

- Kafka comme bus d'événement (**Event Broker**)
- Une plateforme distribuée de diffusion (**Streaming**) de données en continu, capable de publier, stocker, traiter et souscrire à des flux d'enregistrement en temps réel.
- Initialement développée par LinkedIn comme système interne pour la gestion de ses **1 400 milliards de messages quotidiens**, c'est aujourd'hui une solution Open Source Apache.



1. Event-Driven Architecture

2. Event Broker : Apache Kafka

- Kafka permet :
 - Garantit le découplage entre producteurs de données et consommateurs
 - Supporte des consommateurs multiples
 - Permet une forte scalabilité horizontale via une Architecture distribuée
 - Met en œuvre la persistance des données
 - Les événements qui sont publiés dans Kafka ne sont pas supprimés dès leur consommation, comme dans les solutions orientées messaging (ex : RabbitMQ).



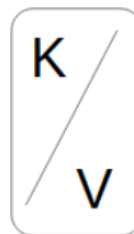
I. Event-Driven Architecture

2. Event Broker : Apache Kafka

Événement

Un **événement** dans Kafka est principalement composé :

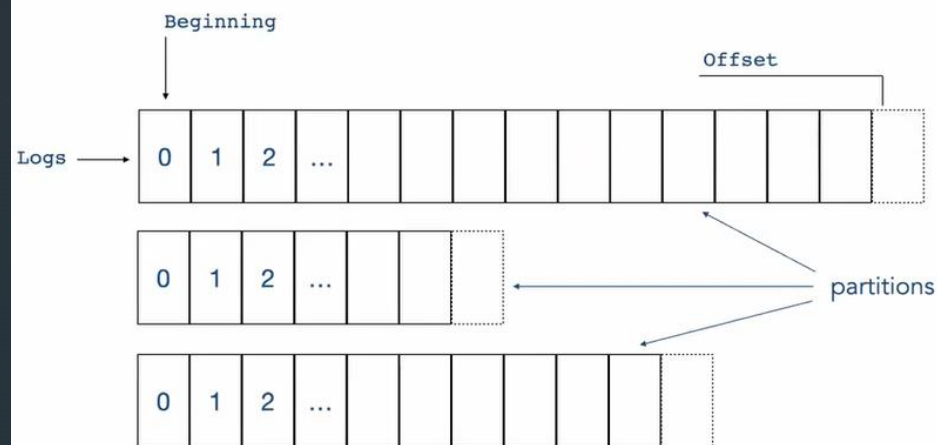
- d'une clé
- d'une valeur



Topics

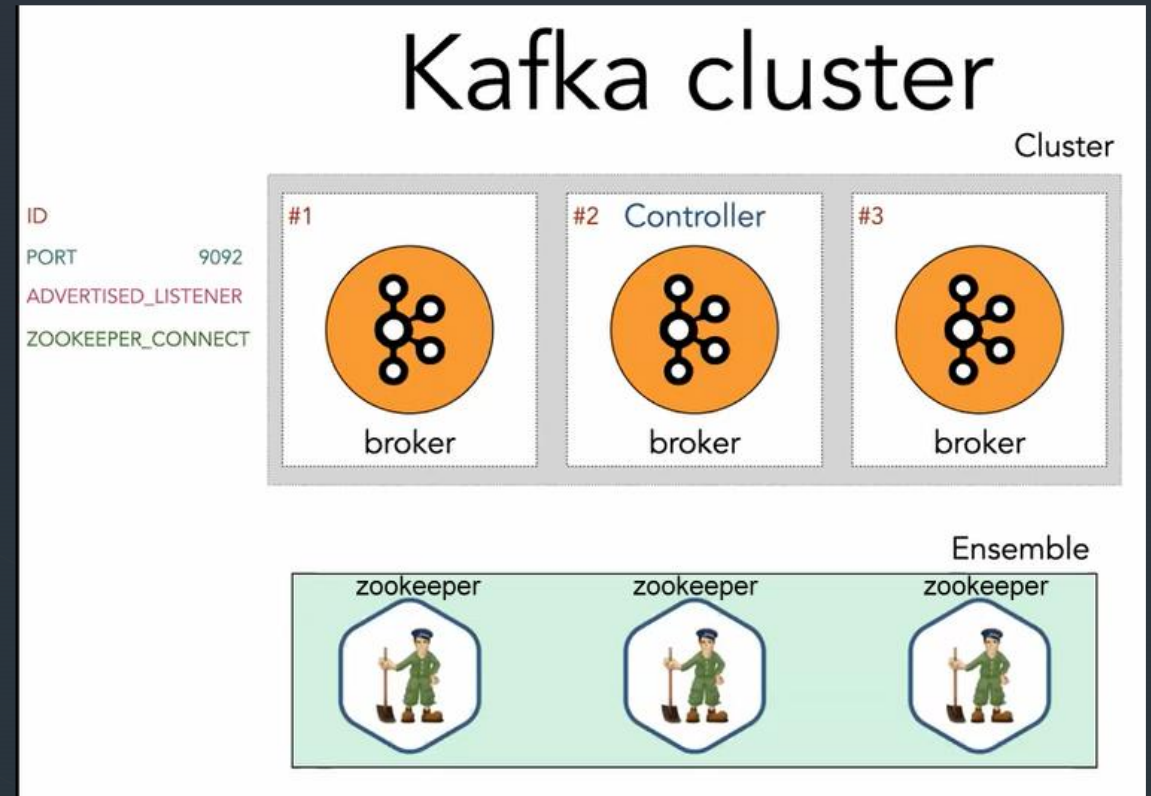
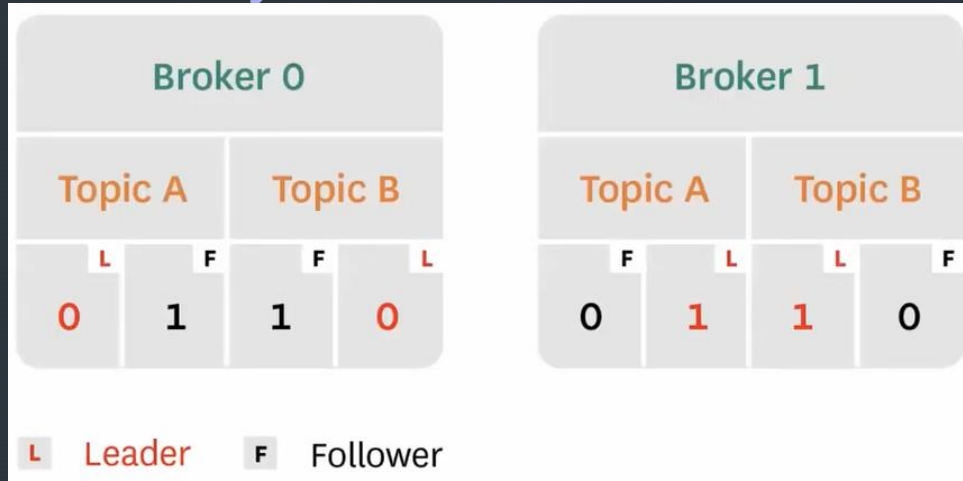


Topics & partitions



1. Event-Driven Architecture

2. Event Broker : Apache Kafka



Config Kafka

“default.replication.factor=2” // nombre de répliqués des partitions dans les brokers

“min.insync.replicas=1” // le nombre minimal de répliqués à maintenir en synchronisation à 2 tout juste après écriture sur la partition leader

config Producer

“request.required.acks=all” pour que le broker qui contient la partition leader n’acquiesce (ack) l’écriture qu’après avoir procédé à la répliqués. Pour que la répliqués soit effective à tout moment et éviter la perte de message.

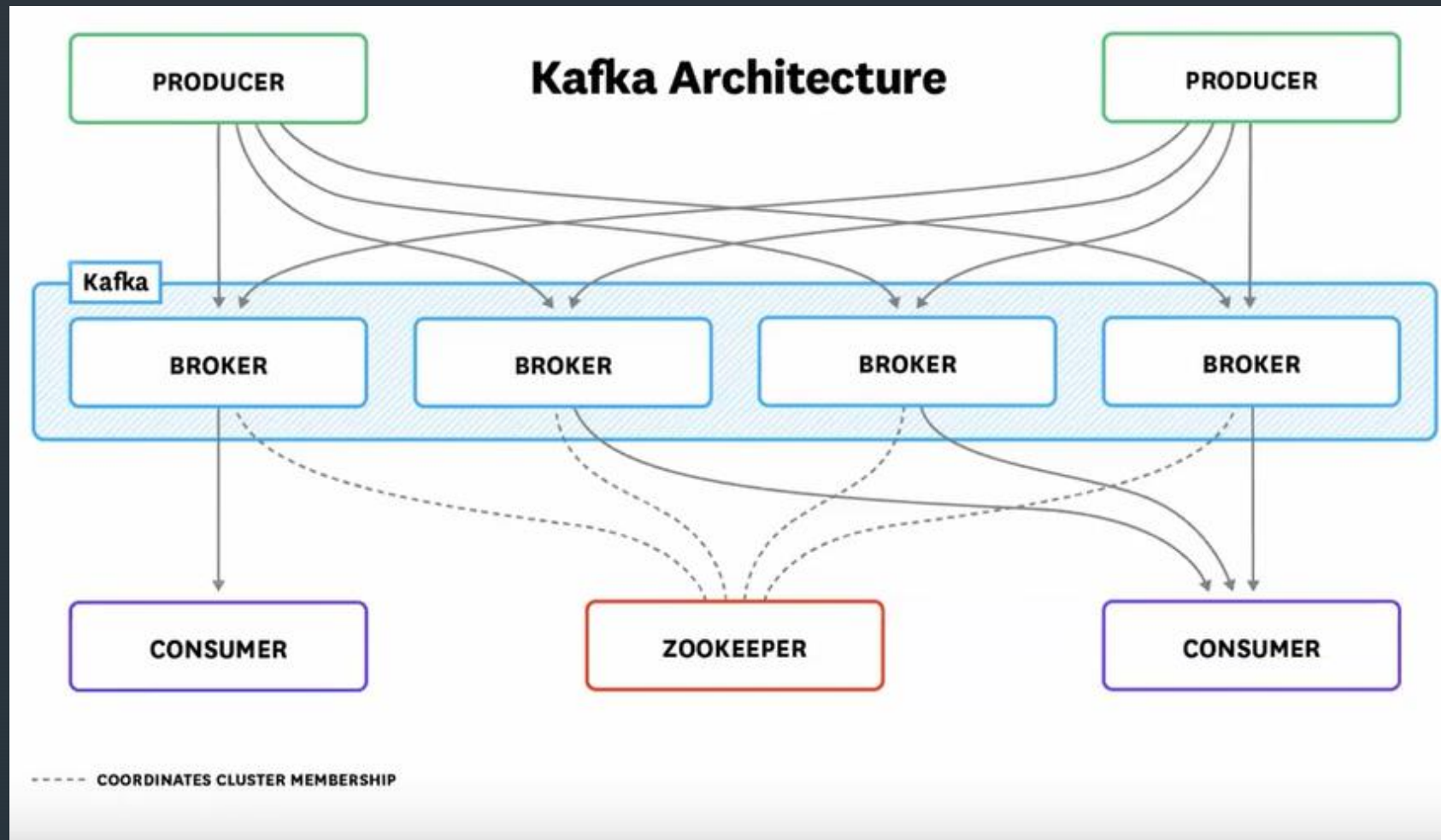
Config Consumer

“enable.auto.commit=false” // désactiver l’autocommit et réaliser un commit explicite en fin de traitement pour garantir que chaque événement qui a fait l’objet d’un commit a bien été pris en compte par le consommateur.



1. Event-Driven Architecture

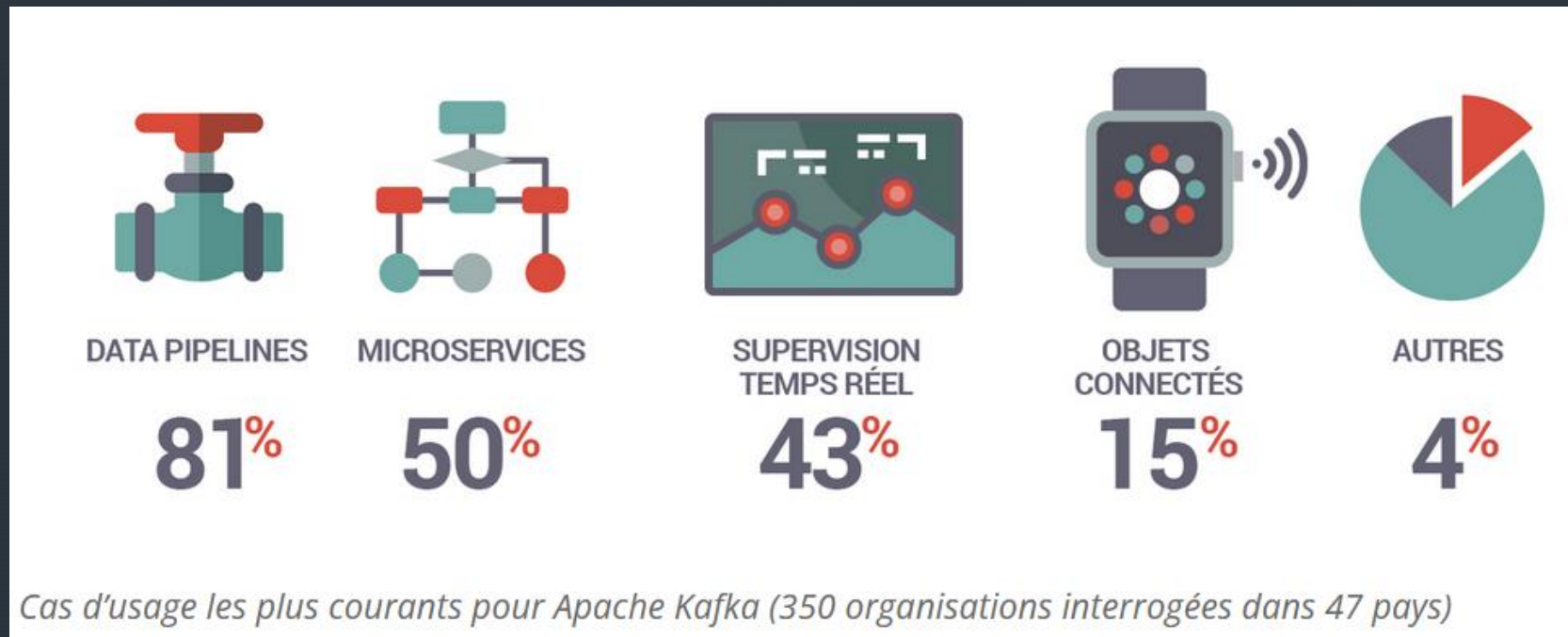
2. Event Broker : Apache Kafka



I. Event-Driven Architecture

2. Event Broker : Apache Kafka

- Data Streaming, le terrain de prédilection de Kafka



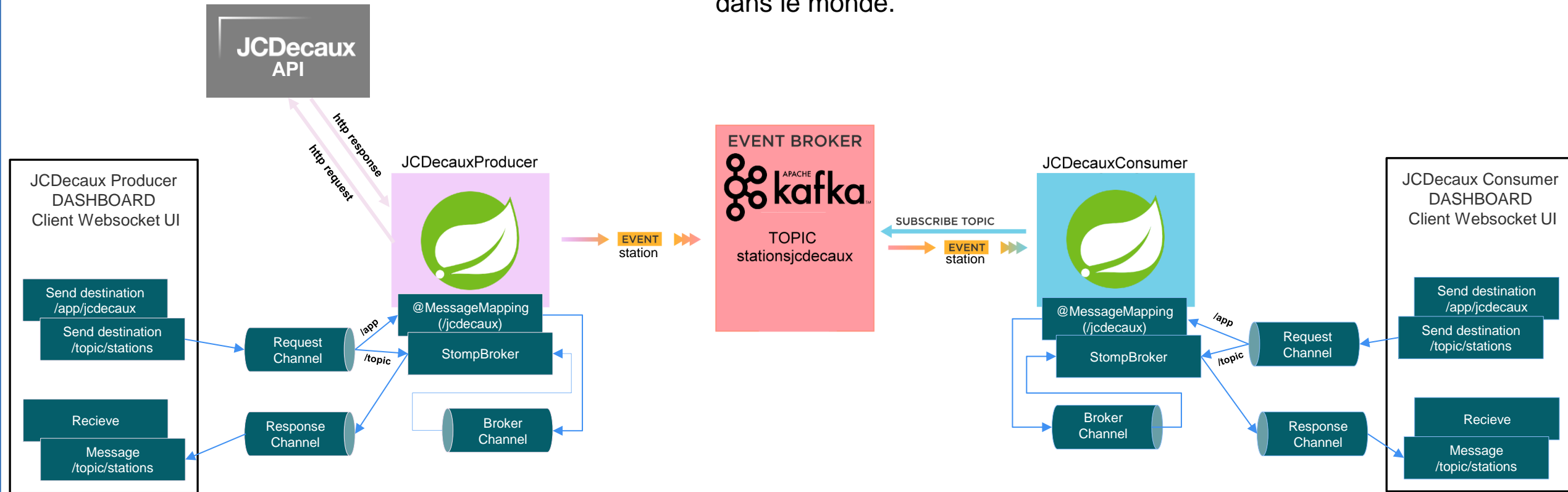
- Dans cette étude(2017), si **50% des interrogés mentionnent « le messaging »** parmi les tâches confiées à Kafka, 66% évoquent le « streaming process ». Parmi les cas d'usages, se détachent les data pipelines (81% de mentions), les Microservices (50%), ou encore la supervision temps réel (43%).

I. Event-Driven Architecture

3. Mise en œuvre : POC App JCDecaux Station avec Spring Kafka, Spring Websocket



Observation en temps réel des locations des vélos à chaque station JCDecaux dans le monde.



I. Event-Driven Architecture

3. Mise en œuvre : POC App JCDecaux Station avec Spring Kafka, Spring Websocket



JCDecaux Producer DASHBOARD x Start Producer JCDecaux x JCDecaux Consumer DASHBOARD x +

localhost:8181/start

Start Producer Station JCDecaux

Interval time to call api milliseconde :

Number of api JCDecaux call(s) :

JCDecaux Producer DASHBOARD x Start Producer JCDecaux x JCDecaux Consumer DASHBOARD x +

localhost:8181

WebSocket Dashboard: Votre Remarque :

JCDecaux Producer DASHBOARD : Consommation en temps réel des stations JCDecaux dans le monde.
[Parametage et Demarrage du Producteur](#)

Démarrage du Producteur JCDecaux avec les parametres : SleepTime=10000 NumberOfCall=10

Nbre de stations JCDecaux récupérer et envoyer au Consumer = 2527

Nbre de stations JCDecaux récupérer et envoyer au Consumer = 2527

Nbre de stations JCDecaux récupérer et envoyer au Consumer = 2527

Nbre de stations JCDecaux récupérer et envoyer au Consumer = 2527

Nbre de stations JCDecaux récupérer et envoyer au Consumer = 2527

Nbre de stations JCDecaux récupérer et envoyer au Consumer = 2527

Nbre de stations JCDecaux récupérer et envoyer au Consumer = 2527

Nbre de stations JCDecaux récupérer et envoyer au Consumer = 2527

Nbre de stations JCDecaux récupérer et envoyer au Consumer = 2527

Arrêt du Producteur JCDecaux ...

JCDecaux Producer DASHBOARD x Start Producer JCDecaux x JCDecaux Consumer DASHBOARD x +

localhost:8080

WebSocket Dashboard: Votre Remarque :

JCDecaux Consumer DASHBOARD : Observation en temps réel des locations des vélos à chaque station JCDecaux dans le monde.
SV : Support(s) à Vélos disponibles ou occupés

1 SV *** STATION : toulouse *** ADRESSE : 30 RUE GABRIEL PERI

1 SV *** STATION : toyama *** ADRESSE : 大手町 2 番 2 Ote-machi

3 SV *** STATION : nantes *** ADRESSE : 7, rue des Etats

1 SV *** STATION : valence *** ADRESSE : Rio Segre - Rafael Company

11 SV *** STATION : bruxelles *** ADRESSE : MORT SUBITE - RUE MONTAGNE AUX-HERBES-POTAGERES (FACE N° 7)/WARMOESBERG (TEGENOVER N°7)

1 SV *** STATION : luxembourg *** ADRESSE : HALL SPORTIF NICOLAS FRANTZ - RUE DE BERTRANGE / HALL SPORTIF NICOLAS FRANTZ

1 SV *** STATION : valence *** ADRESSE : Quart - Fernando el Católico

2 SV *** STATION : toyama *** ADRESSE : 富山市明輪町1内 Station Plaza South

1 SV *** STATION : toulouse *** ADRESSE : 2 PL DE LA DAURADE

1 SV *** STATION : rouen *** ADRESSE :

1 SV *** STATION : nantes *** ADRESSE : 7, rue des Etats

1 SV *** STATION : lyon *** ADRESSE : Devant le lycée

1 SV *** STATION : bruxelles *** ADRESSE : STADE / STADIUM - AVENUE HOUBA DE STROOPER / HOUBA DE STROOPERLAAN

1 SV *** STATION : seville *** ADRESSE : AVENIDA DE MIRAFLORES - Aprox. C/ Albaida

3 SV *** STATION : bruxelles *** ADRESSE : MORT SUBITE - RUE MONTAGNE AUX-HERBES-POTAGERES (FACE N°



I. Event-Driven Architecture

3. Mise en œuvre : POC App JCDecaux Station avec Spring Kafka, Spring Websocket



Topic List

37.187.88.37:48282/clusters/AfrinnovKafkaCluster/topics

CMAK AfrinnovKafkaCluster Cluster Brokers Topic Preferred Replica Election Schedule Leader Election Reassign Partitions

Clusters / AfrinnovKafkaCluster / Topics

Operations

[Generate Partition Assignments](#) [Run Partition Assignments](#) [Add Partitions](#)

Topics

Show 10 entries

Topic	# Partitions	# Brokers	Brokers Spread %	Brokers Skew %	Brokers Leader Skew %	# Replicas
__consumer_offsets	50	1	100	0	0	1
stationsjcdecaux	1	1	100	0	0	1

Showing 1 to 2 of 2 entries

II. Microservice Architecture

1. Introduction

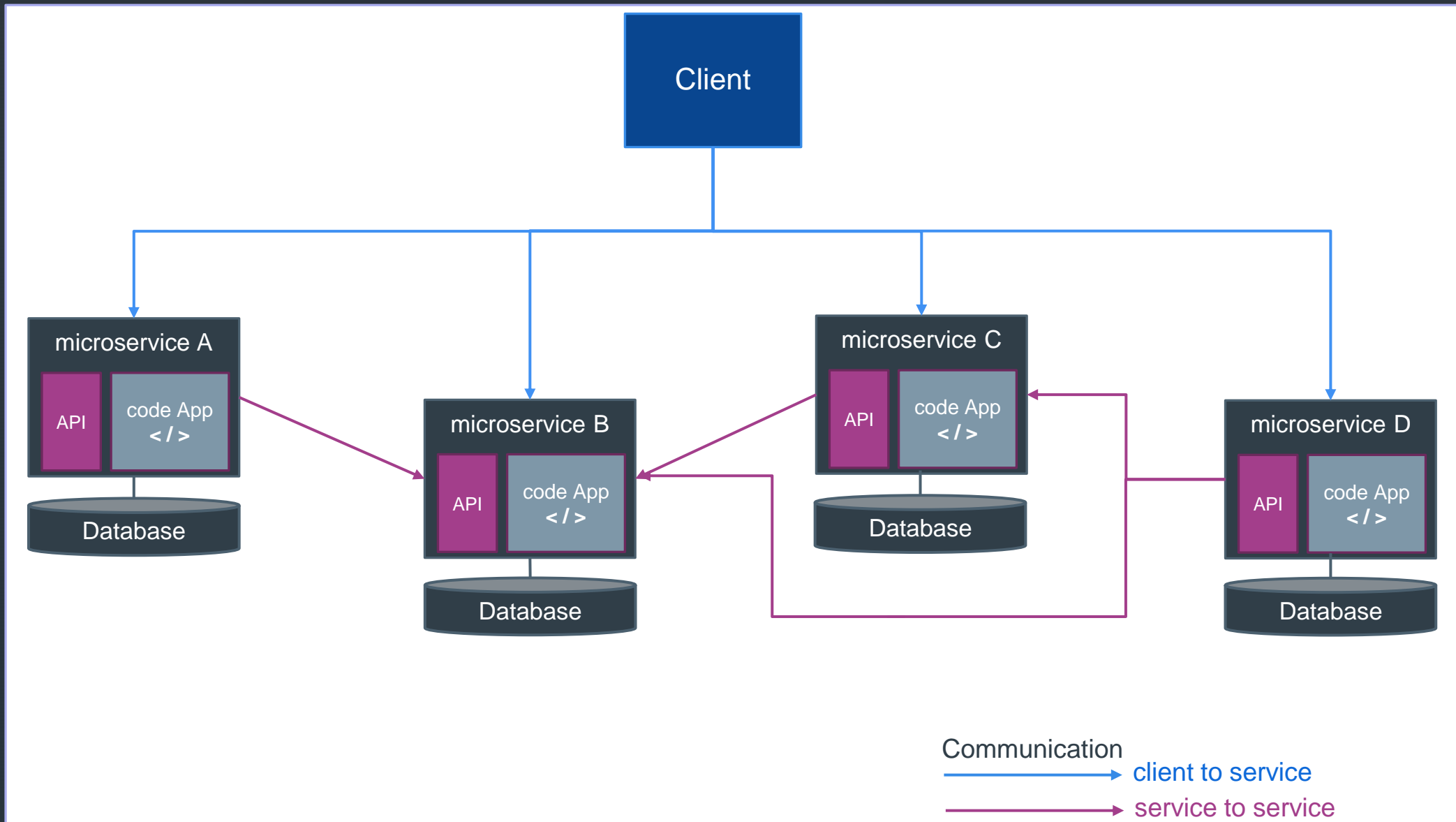
- Architecture Microservice
- « Le style architectural des microservices est une approche permettant de développer une application unique sous la forme d'une suite logicielle intégrant plusieurs services. Ces services sont construits autour des capacités de l'entreprise et peuvent être déployés de façon indépendante »

Martin Fowler

- **Microservices** sont une méthode développement logiciel utilisée pour concevoir une application comme **un ensemble de services modulaires**. Chaque module répond à un objectif métier spécifique et communique avec les autres modules.

II. Microservice Architecture

1. Introduction



II. Microservice Architecture

1. Introduction

- **Réduction du temps de développement**
 - Grâce à un développement distribué, plusieurs microservices peuvent être travaillés et développés simultanément.
- **Évolutivité accrue**
 - Avec les microservices et leur **indépendance** en termes de développement et de déploiement. Cela permet de **faire évoluer l'application** sereinement pour répondre à de nouveaux besoins et de nouvelles demandes sans entraver le fonctionnement du système.
- **Réduction des pannes**
 - Les microservices étant **indépendants**, lorsqu'un élément tombe en panne ou rencontre un problème, l'ensemble de l'application ne cesse pas de fonctionner contrairement aux applications monolithiques. Il est également plus facile d'identifier et de résoudre une panne dans ce type d'eco-système.



II. Microservice Architecture

1. Introduction

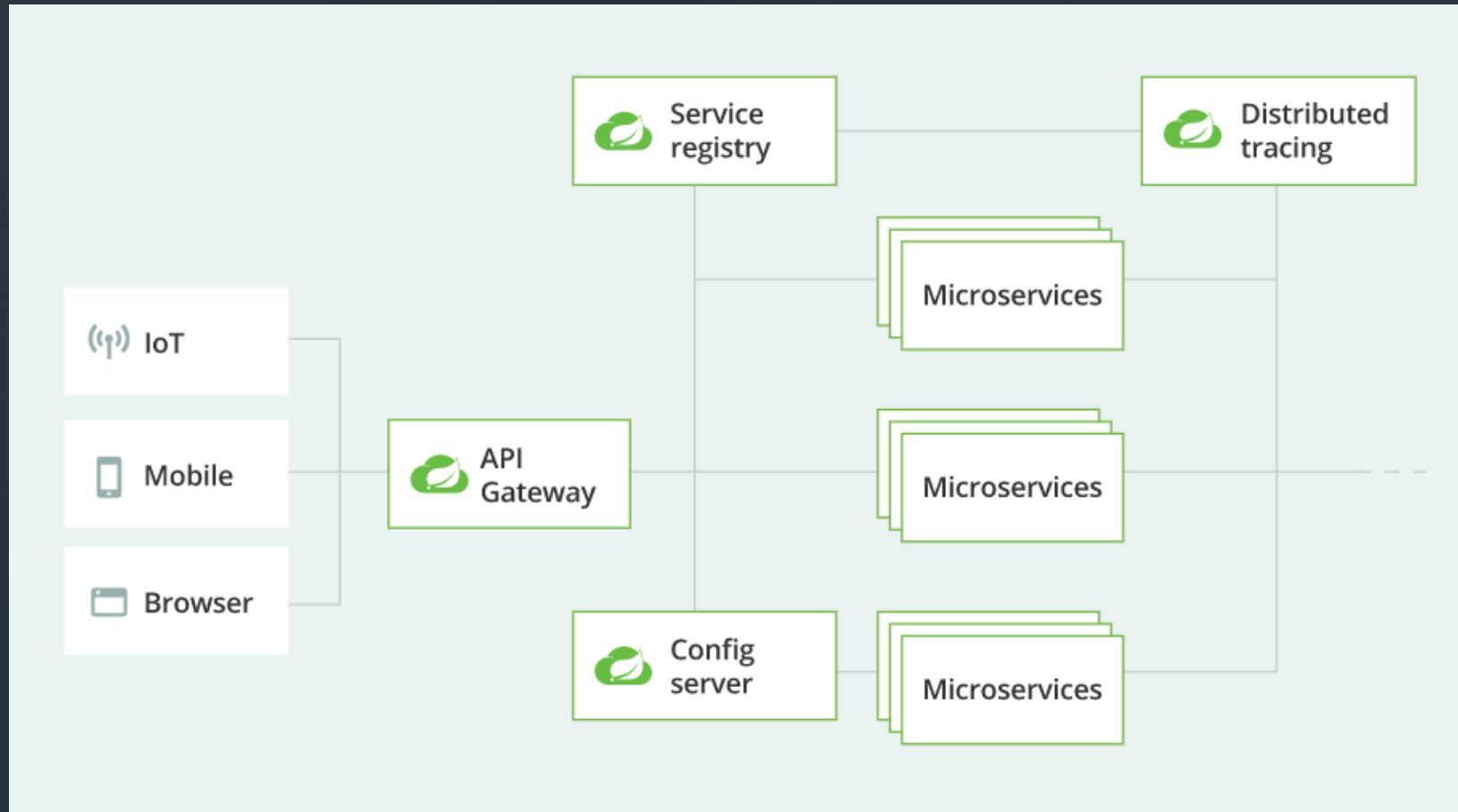
- **L'adaptabilité de chaque microservice aux outils de travail**
 - L'**indépendance** des composants offre la possibilité de paramétrer les **microservices avec différents langages de programmation**. Ainsi, les outils opérationnels utilisés par l'entreprise peuvent être rendus compatibles avec la solution globale.
- **La flexibilité par rapport au cloud**
 - Une entreprise qui utilise l'**architecture en microservices** peut réduire ou augmenter son usage du cloud en fonction de la charge de l'application. L'organisation est ainsi beaucoup plus flexible.
- Le plus important dans l'implémentation des microservices, **c'est de faire simple**.
 - Il faut déléguer le maximum d'exigences non fonctionnelles à l'infrastructure logicielle (conteneurs, Services Mesh, etc...) pour réduire les risques futurs de dettes techniques, et garantir un fonctionnement homogène des services.
 - Pour cela, on se doit de respecter les règles des 12 factors (<https://12factor.net/fr/>).



II. Microservice Architecture

2. Edge Microservice : Spring Cloud

- **Spring Cloud** fournit des fonctionnalités pour les cas d'utilisation typiques tels que l'enregistrement et la découverte de services, le routage, l'équilibrage de charge et les circuit-breakers.



II. Microservice Architecture

2. Edge Microservice : Spring Cloud



- **Spring Cloud Bootstrap** (e.g. Bootstrap context and @RefreshScope)
- **Spring Cloud Config** : Se positionne comme serveur de distribution des fichiers de configuration.
 - **Config Server**
 - **Config Client**
- **Spring Cloud Netflix**
 - **Discovery** : Permet de découvrir les Microservice sur notre serveur. C'est aussi un outil clé pour la gestion des Microservices.
 - **Eureka Server**
 - **Eureka Client**
 - **Load Balancer**
 - **Ribbon** (config et dépendance commenté) : Équilibrer les requêtes entre plusieurs instances pour éviter une surcharge d'un serveur
 - **Circuit Breaker** : Définit un ensemble de seuils qui, une fois dépassés, arrêteront l'exécution d'un bloc de code.
 - **Hystrix** : Une API pour la résilience d'applications.

II. Microservice Architecture

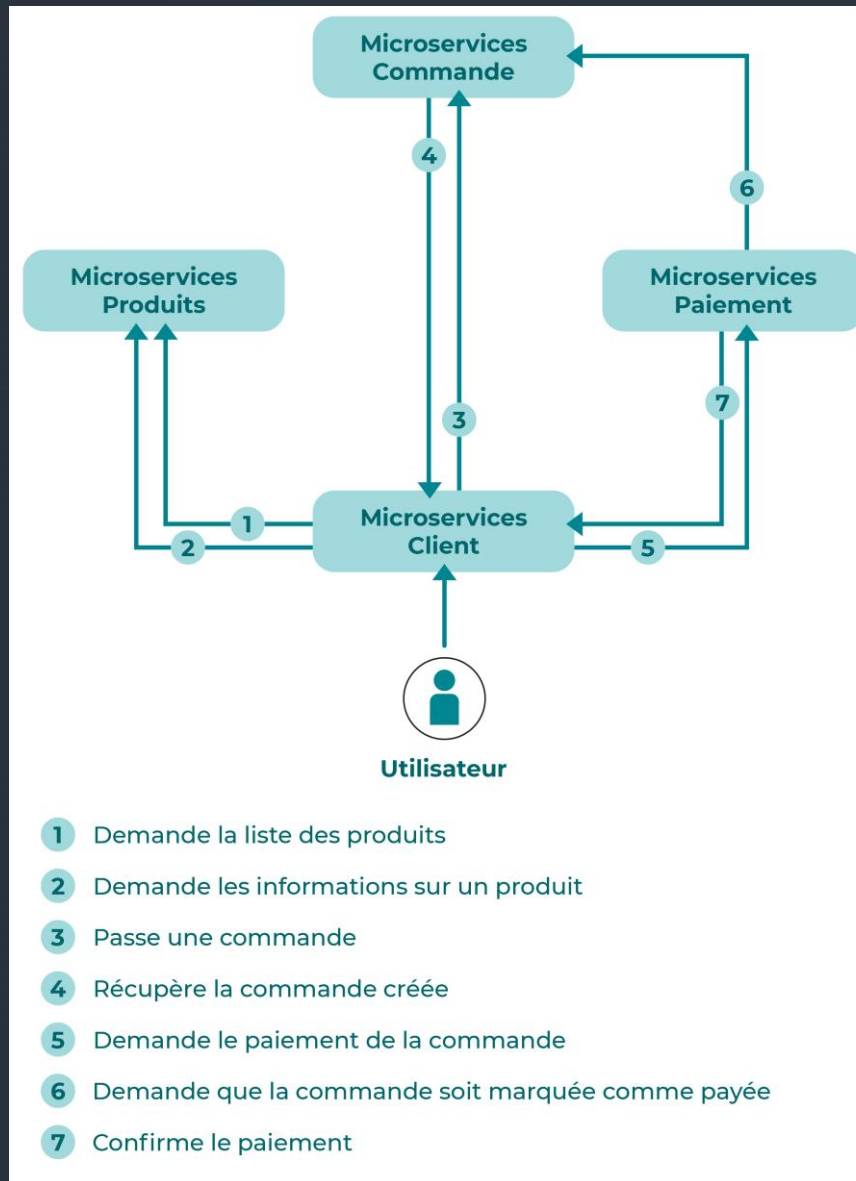
2. Edge Microservice : Spring Cloud

- **Spring Cloud Routing**
 - **Gateway** : Le point d'entrée unique pour les API et Microservices.
 - **OpenFeign** : Faire communiquer (synchrone) les microservices grâce à Feign.
- **Spring Cloud Load Balancer** : Équilibrer les requêtes entre plusieurs instances pour éviter une surcharge d'un serveur
- **Spring Cloud Observability**
 - **Sleuth** : Permet de donner des ID uniques à chaque requête, c'est
 - **Zipkin Client** : permet exposer les traces vers Zipkin Server.
- **Spring Boot Actuator** : expose une API qui donne des informations sur le microservice concerné, mais ne propose pas d'interface graphique.
- **Zipkin Server** : permet de tracer les requêtes de service en service uniquement si ces services intègrent ses dépendances.



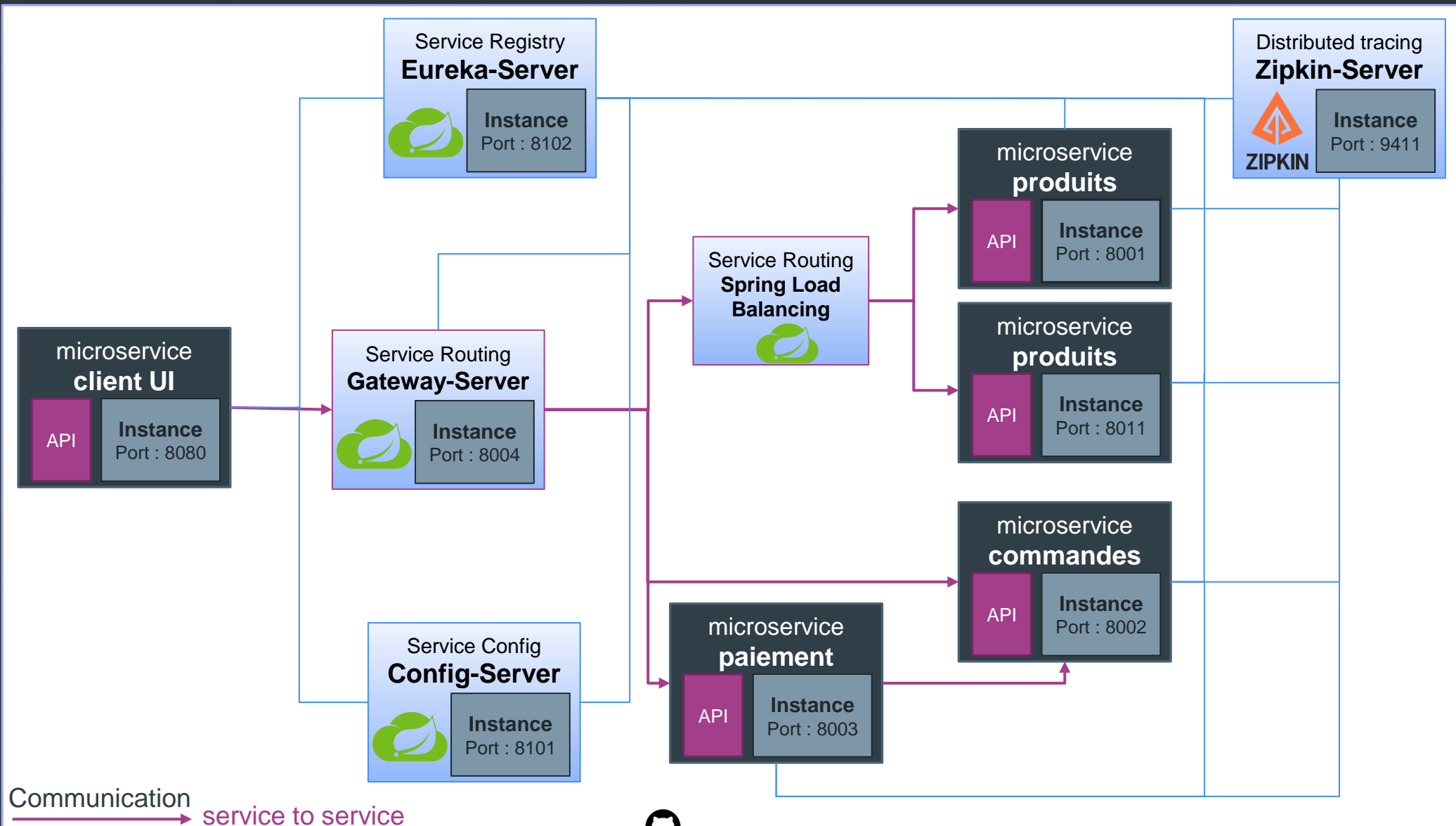
II. Microservice Architecture

3. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud



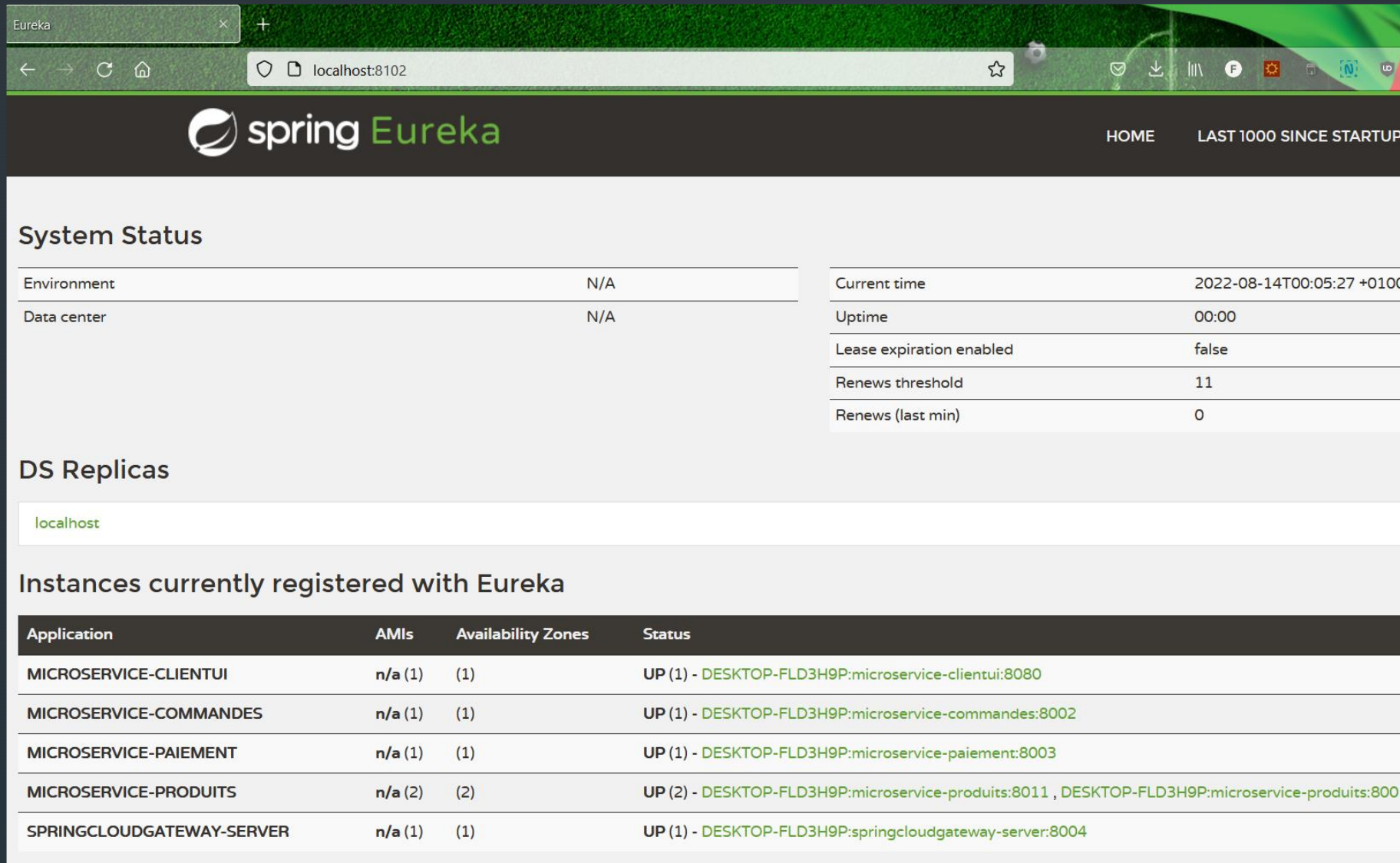
II. Microservice Architecture

3. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud



II. Microservice Architecture

3. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud



The screenshot shows the Spring Eureka web interface in a browser window. The address bar shows 'localhost:8102'. The page has a dark header with the 'spring Eureka' logo and navigation links 'HOME' and 'LAST 1000 SINCE STARTUP'.

System Status

Environment	N/A	Current time	2022-08-14T00:05:27 +0100
Data center	N/A	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	11
		Renews (last min)	0

DS Replicas

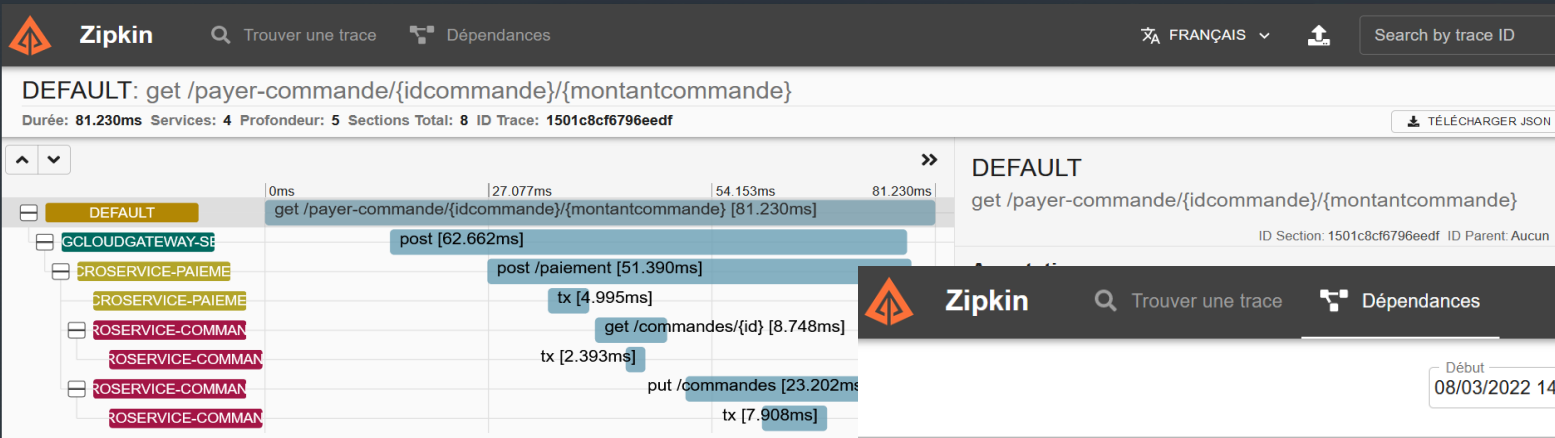
localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MICROSERVICE-CLIENTUI	n/a (1)	(1)	UP (1) - DESKTOP-FLD3H9P:microservice-clientui:8080
MICROSERVICE-COMMANDES	n/a (1)	(1)	UP (1) - DESKTOP-FLD3H9P:microservice-commandes:8002
MICROSERVICE-PAIEMENT	n/a (1)	(1)	UP (1) - DESKTOP-FLD3H9P:microservice-paiement:8003
MICROSERVICE-PRODUITS	n/a (2)	(2)	UP (2) - DESKTOP-FLD3H9P:microservice-produits:8011 , DESKTOP-FLD3H9P:microservice-produits:8001
SPRINGCLOUDGATEWAY-SERVER	n/a (1)	(1)	UP (1) - DESKTOP-FLD3H9P:springcloudgateway-server:8004

II. Microservice Architecture

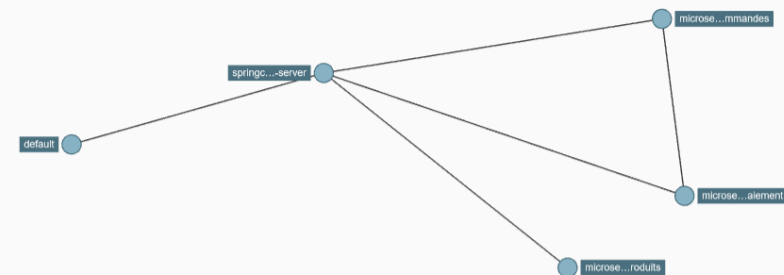
3. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud



Application Mcommerce

Bougie fonctionnant au feu

Chaise pour s'asseoir



III. Event-Driven Microservice Architecture

1. Introduction

- De nombreux facteurs avantageux et déterminants caractéristiques de **l'architecture microservices** se retrouvent dans **une architecture orientée événements**, qui permettent toutes les deux de relever des défis similaires.
- Cependant, prenez une **architecture microservices** adoptant une approche d'intégration demande/réponse synchrone traditionnelle ; ce type d'architecture active un service atomique indépendant doté d'un cycle de vie indépendant et, par conséquent et de manière générale, une équipe indépendante qui en est responsable. Si ce service reposait sur un flux d'intégration demande/réponse avec un autre microservice (sans raison spécifique), alors le service ne serait pas couplé de manière lâche et n'en serait pas véritablement indépendant. Vous ne pourriez pas mettre à niveau/faire évoluer le service sans vous préoccuper du service couplé.

III. Event-Driven Microservice Architecture

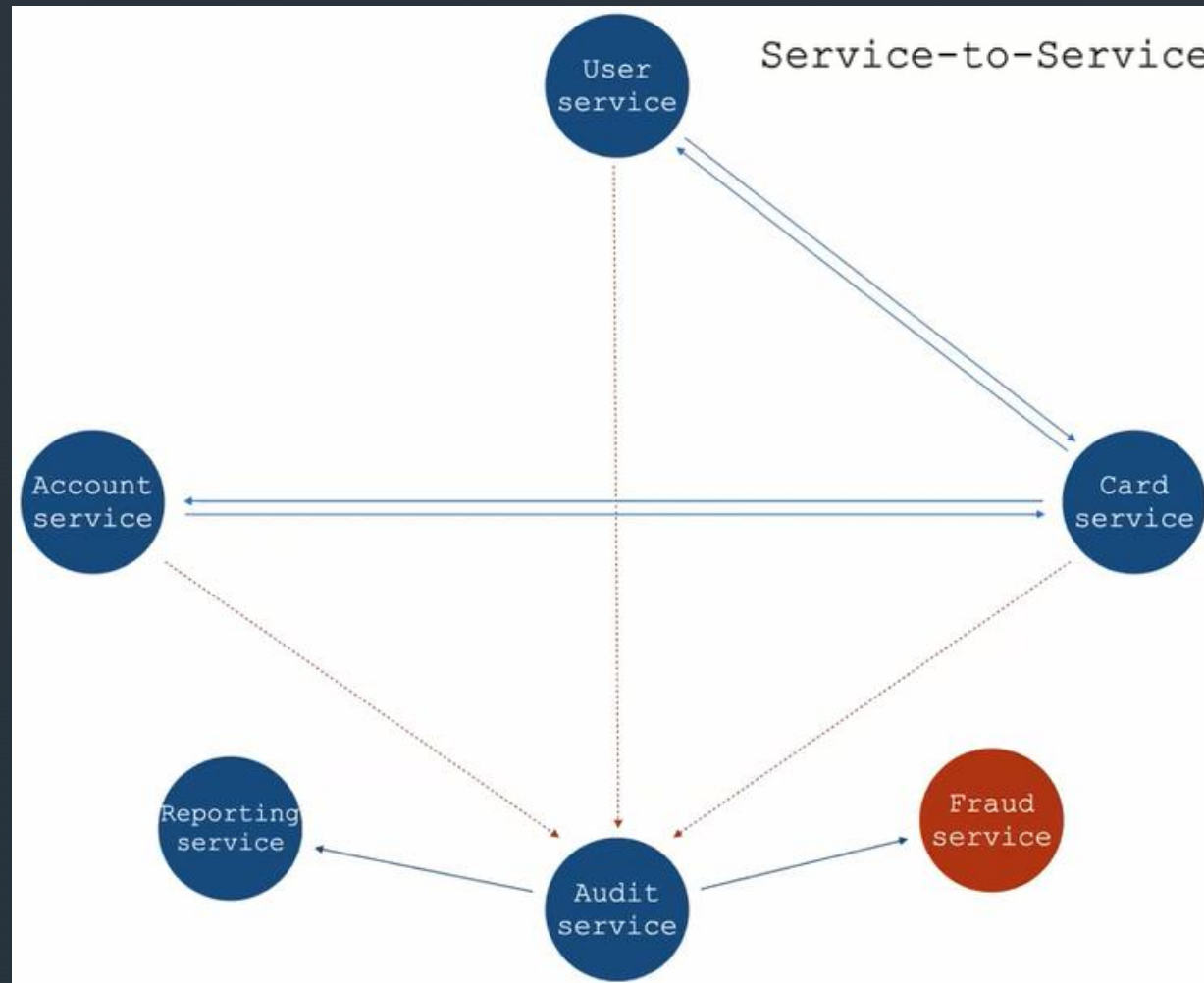
1. Introduction

- Il n'est pas indispensable de disposer d'une architecture **orientée événements** pour surmonter ce problème, car un modèle asynchrone alternatif, comme une file d'attente de messages, constituerait une solution efficace dans cette situation. Cependant, cela démontre dans quelle mesure les caractéristiques d'une architecture orientée événements et la diffusion d'événements peuvent interagir de manière positive avec les microservices.



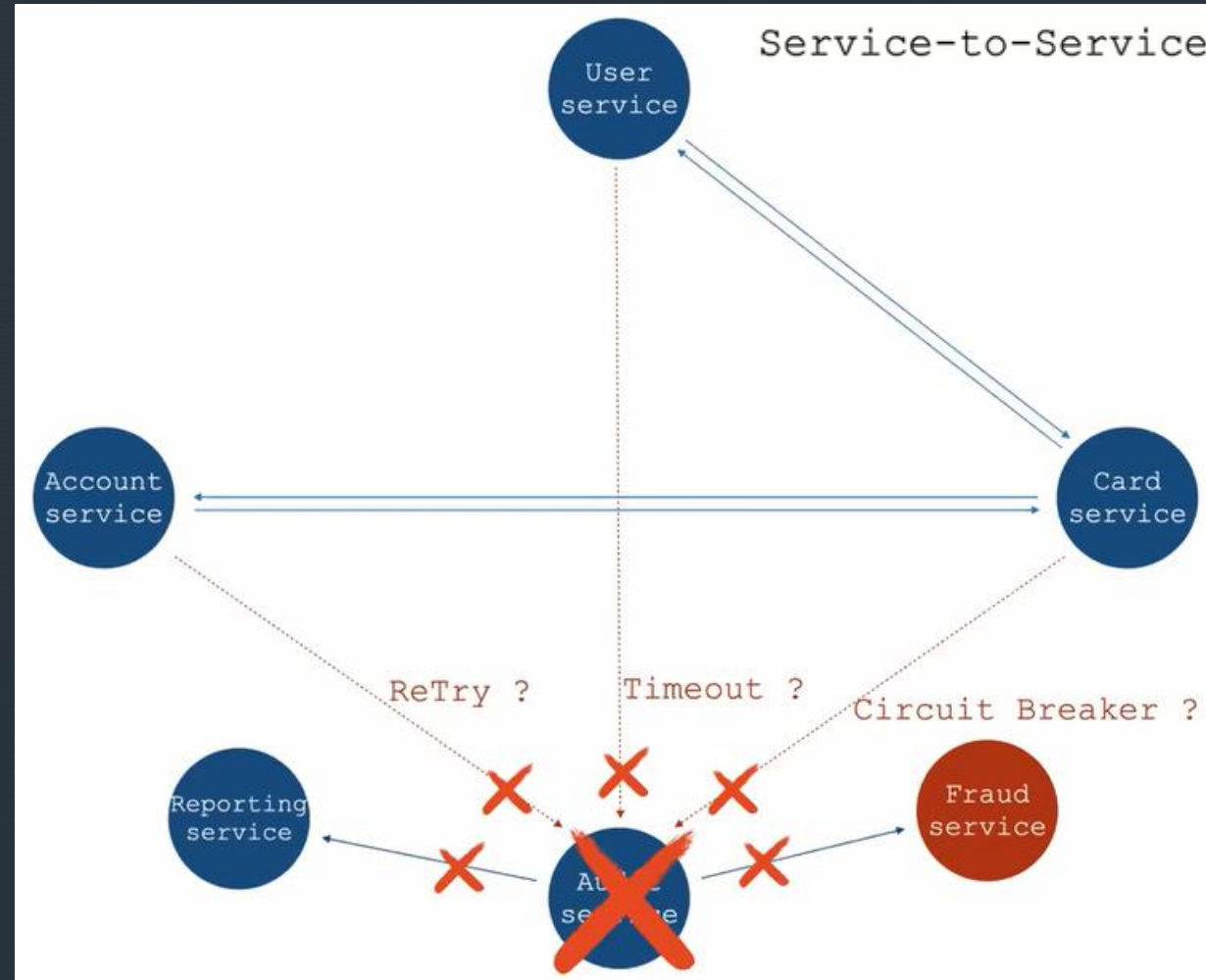
III. Event-Driven Microservice Architecture

1. Introduction



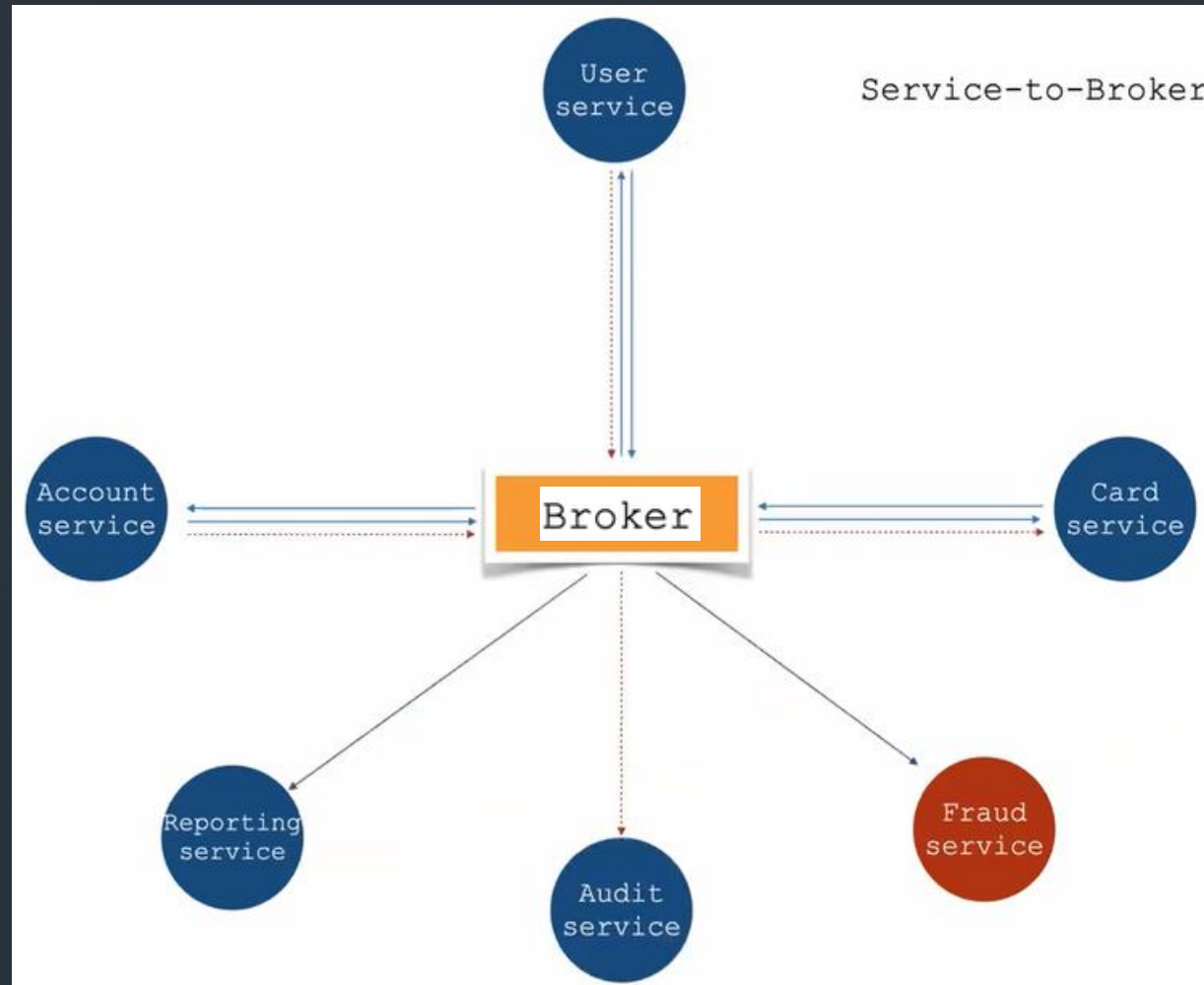
III. Event-Driven Microservice Architecture

1. Introduction



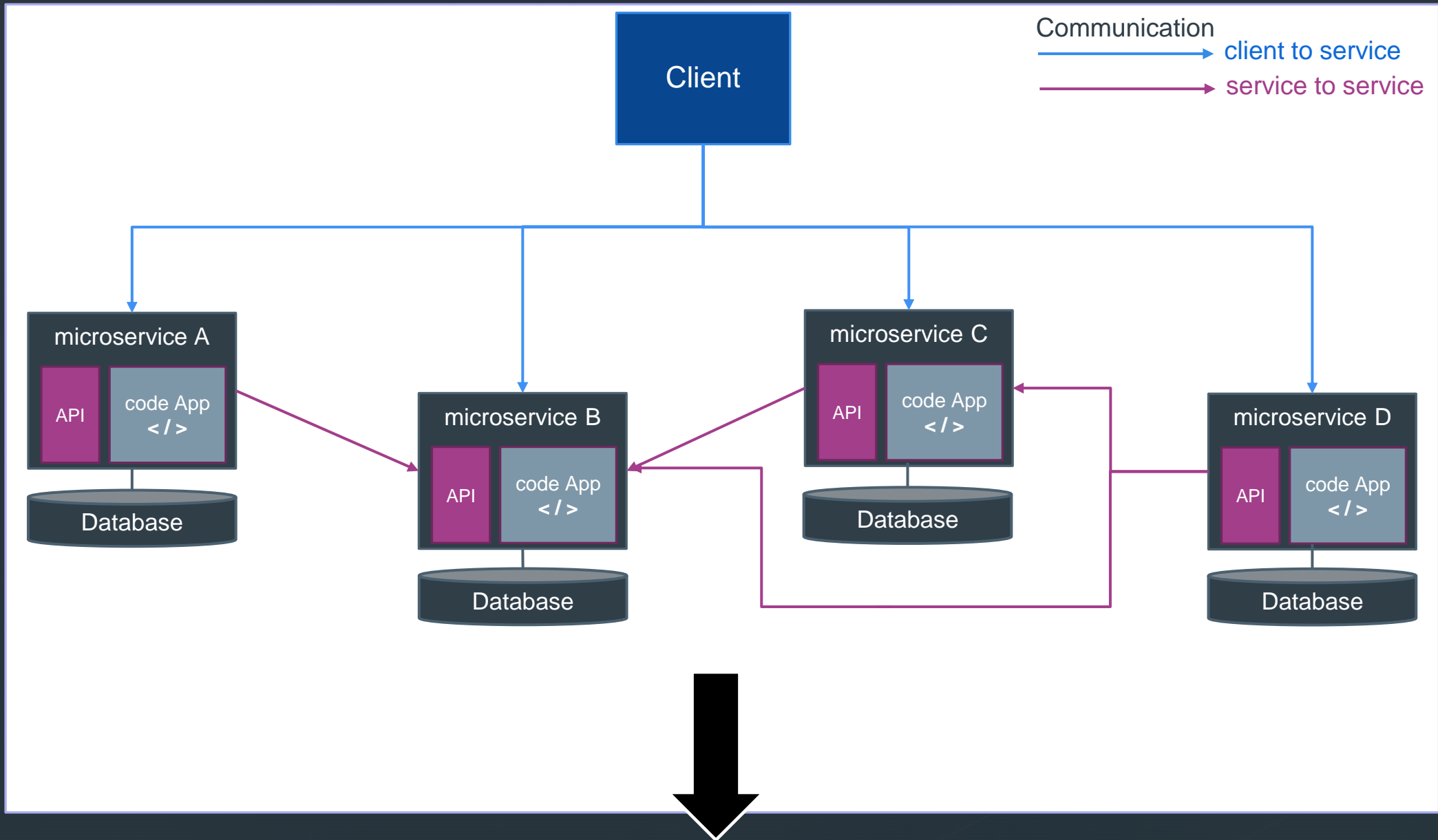
III. Event-Driven Microservice Architecture

1. Introduction



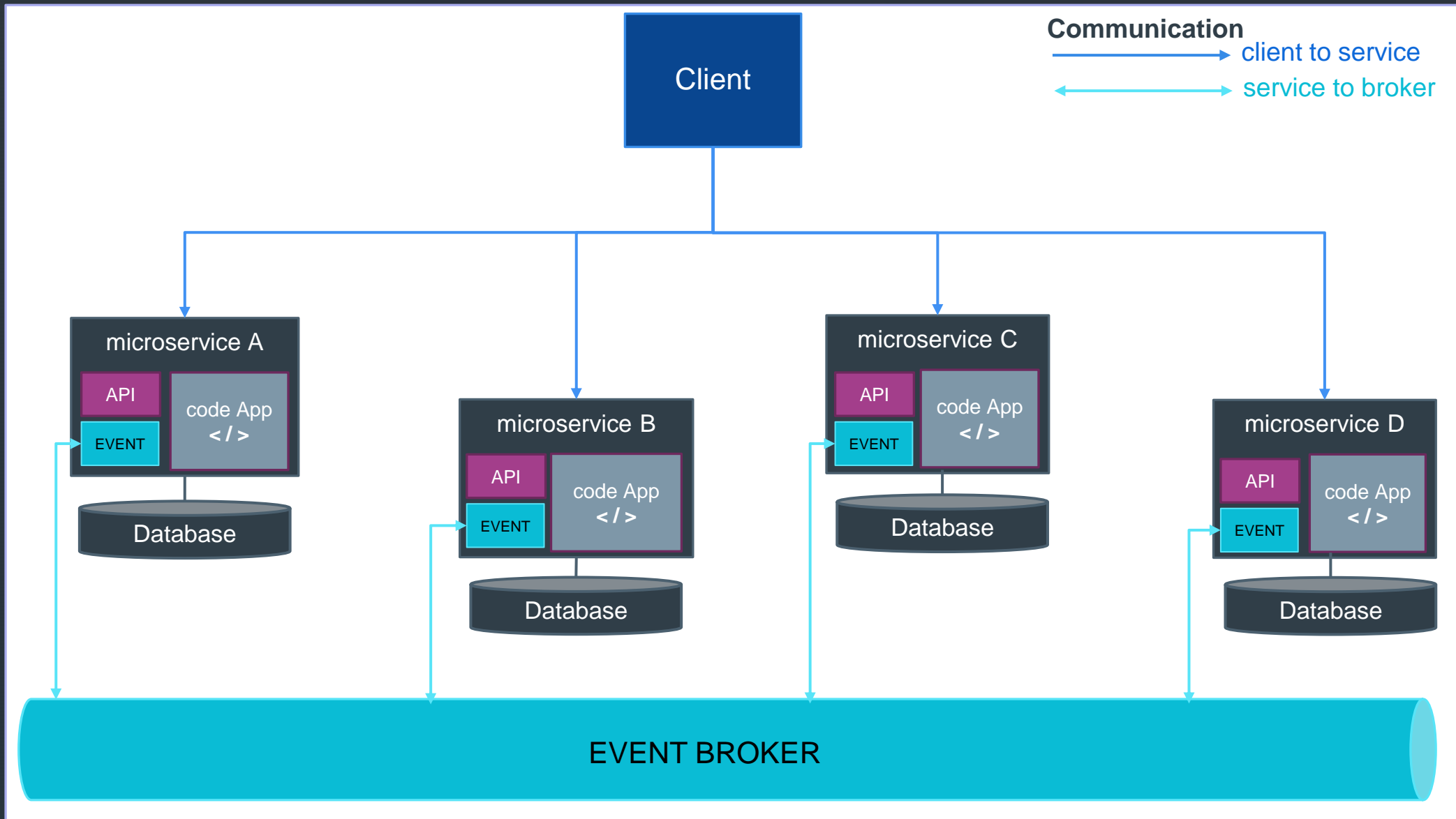
III. Event-Driven Microservice Architecture

1. Introduction



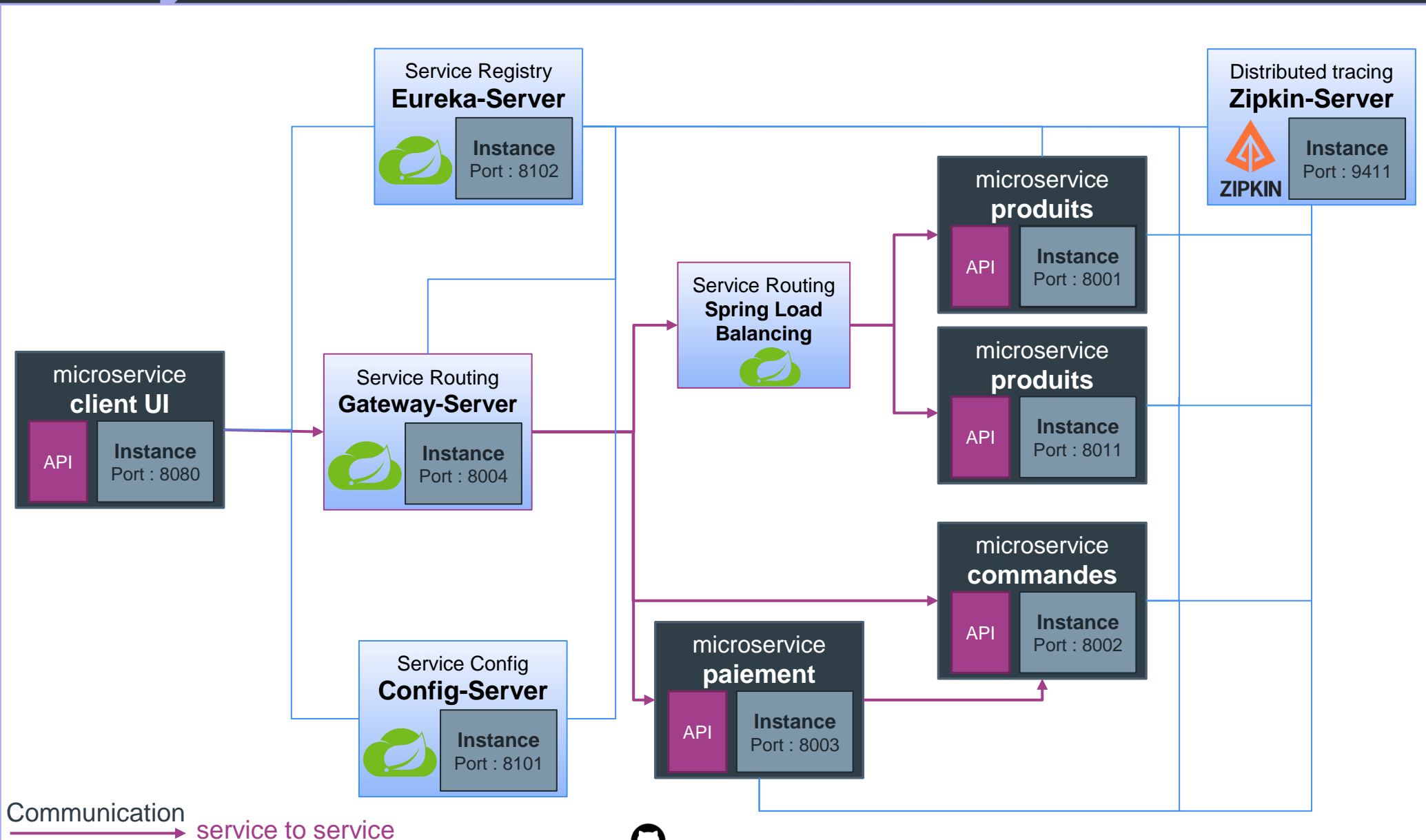
III. Event-Driven Microservice Architecture

1. Introduction



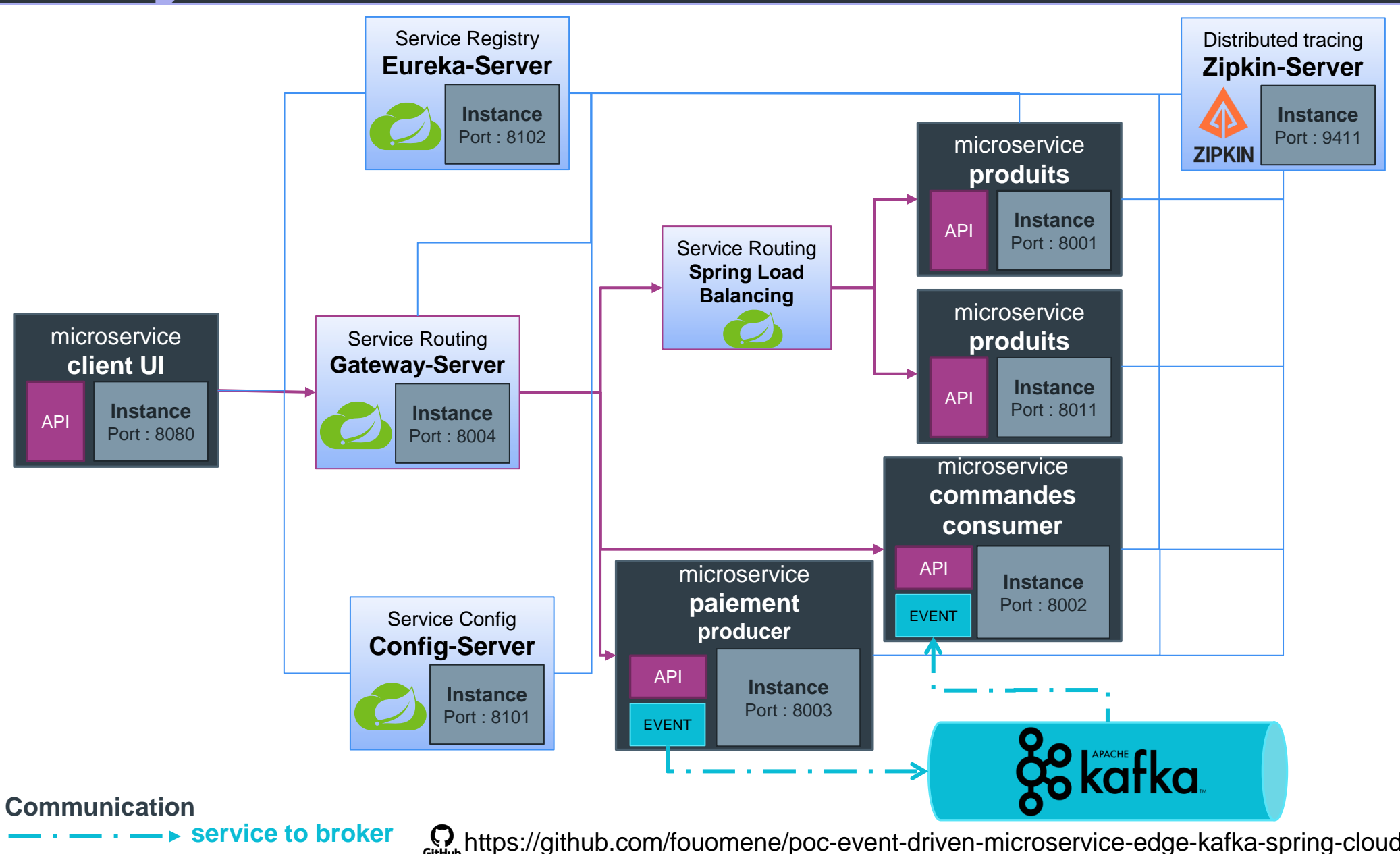
III. Event-Driven Microservice Architecture

2. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud, Spring Kafka



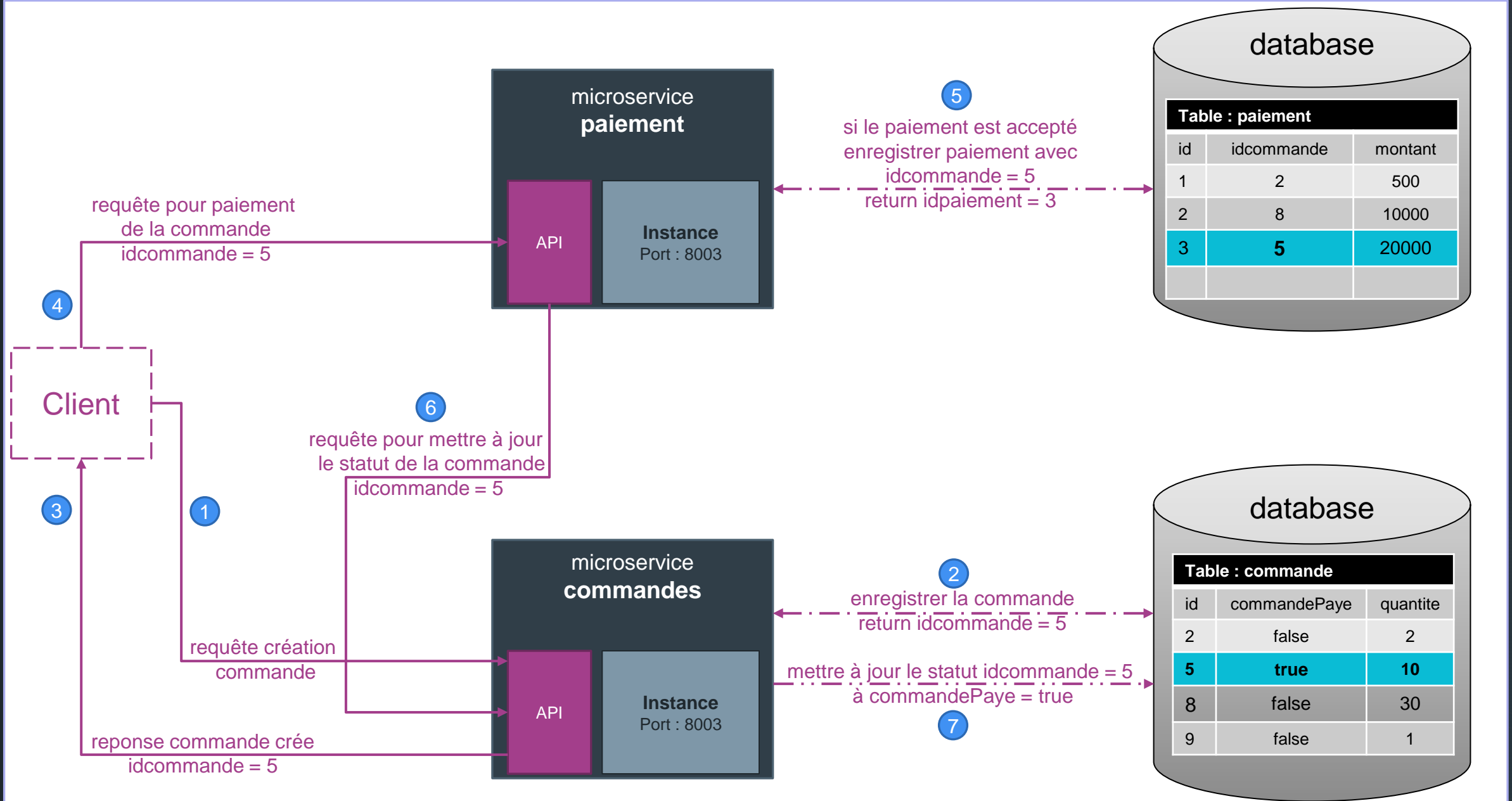
III. Event-Driven Microservice Architecture

2. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud, Spring Kafka



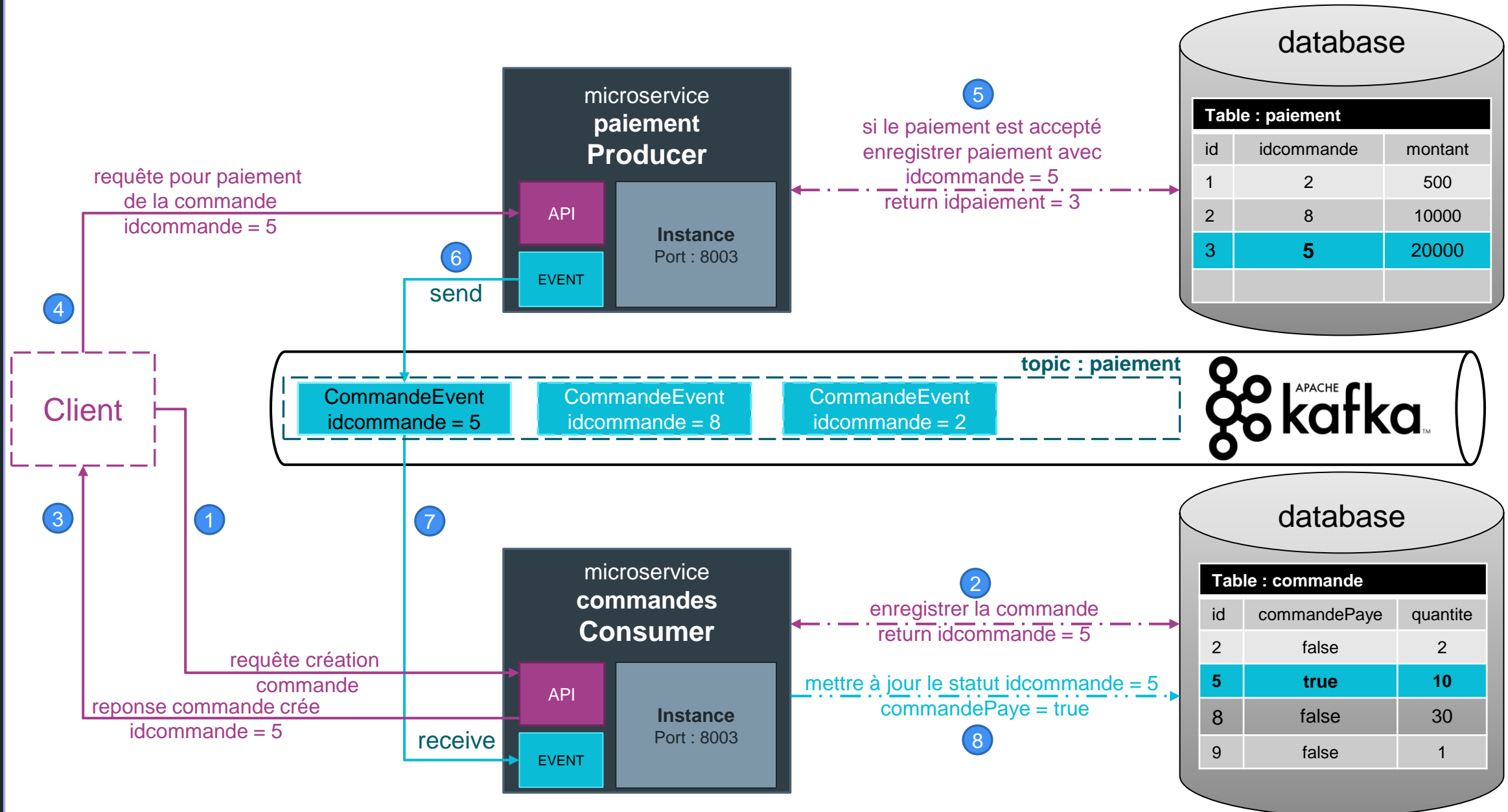
III. Event-Driven Microservice Architecture

2. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud, Spring Kafka



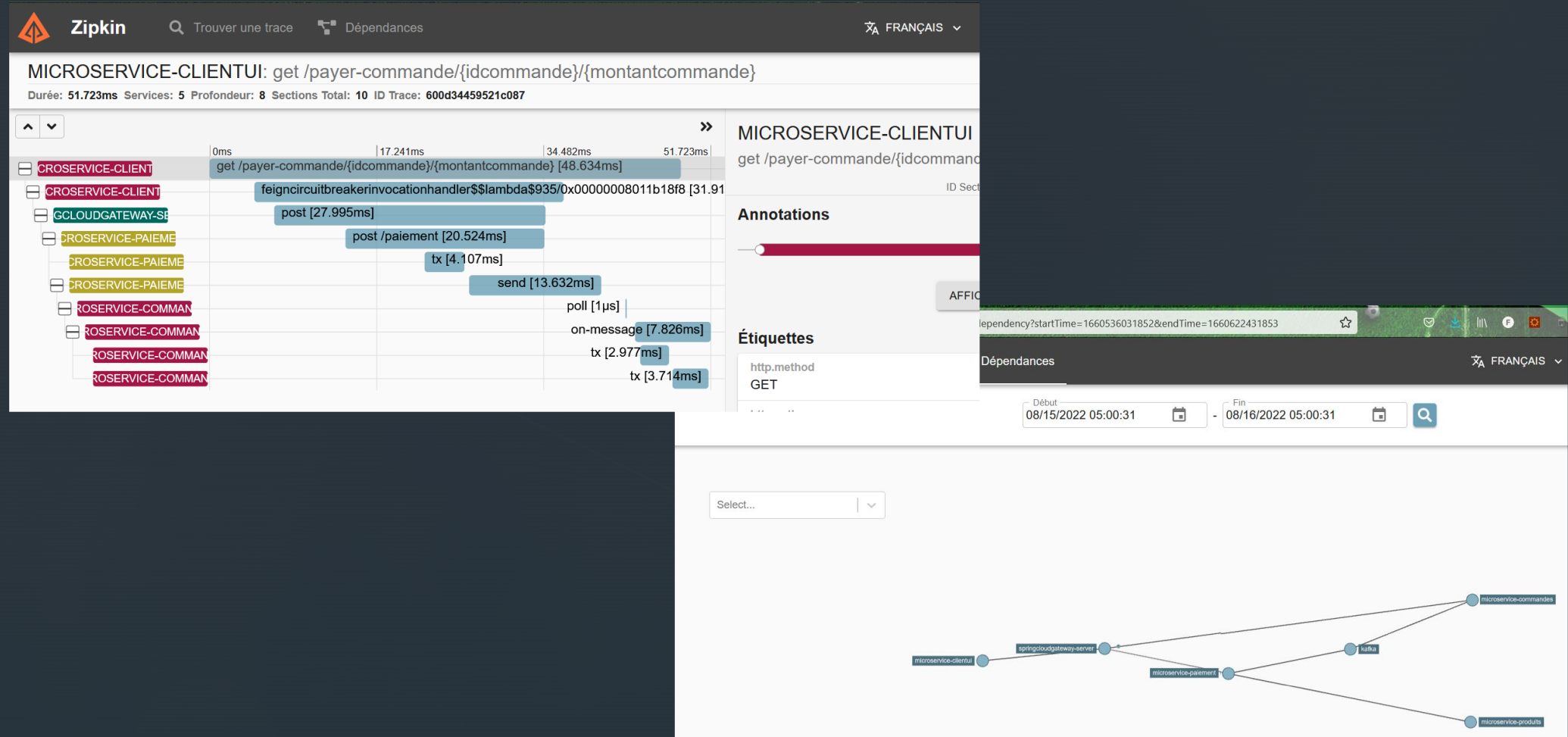
III. Event-Driven Microservice Architecture

2. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud, Spring Kafka



II. Event-Driven Microservice Architecture

2. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud, Spring Kafka



<https://github.com/fouomene/poc-event-driven-microservice-edge-kafka-spring-cloud>




II. Event-Driven Microservice Architecture

2. Mise en œuvre : POC App Mini-Ecommerce avec Spring Cloud, Spring Kafka


← → ↻ ⓘ localhost:8080

Android Asset Studi... Developers - How t... how to encode and... Nouvel onglet Home Home 4 Home 3 https://demo.them...

Application Mcommerce



Bougie fonctionnant au feu



Chaise pour s'asseoir



<https://github.com/fouomene/poc-event-driven-microservice-edge-kafka-spring-cloud>

IV. Références

- <https://github.com/fouomene/poc-jcdecaux-spring-kafka-websocket>
- <https://github.com/fouomene/poc-microservice-edge-spring-cloud>
- <https://github.com/fouomene/poc-event-driven-microservice-edge-kafka-spring-cloud>
- <https://kafka.apache.org>
- <https://spring.io/projects/spring-kafka#overview>
- <https://spring.io/guides/gs/messaging-stomp-websocket>
- <https://www.redhat.com/en/topics/integration/what-is-event-driven-architecture>
- <https://blog.ippon.fr/2021/06/29/comment-se-lancer-avec-kafka-partie-1>
- <https://nexworld.fr/kafka-et-architectures-fast-data>
- <https://www.confluent.io/2017-apache-kafka-report>
- <https://www.tibco.com/fr/reference-center/what-is-event-driven-architecture>
- <https://www.leanix.net/fr/wiki/vsm/architecture-microservices>
- <https://nexworld.fr/les-microservices-cest-quoi>
- <https://blog.octo.com/une-vision-sur-le-service-mesh-service-mesh-versus-librairies-applicatives-leexemple-de-spring-cloud>
- <https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh>
- <https://12factor.net/fr/>
- <https://www.chakray.com/fr/architectures-orientees-evenements-panacee-autre-tendance-consommateurs>