

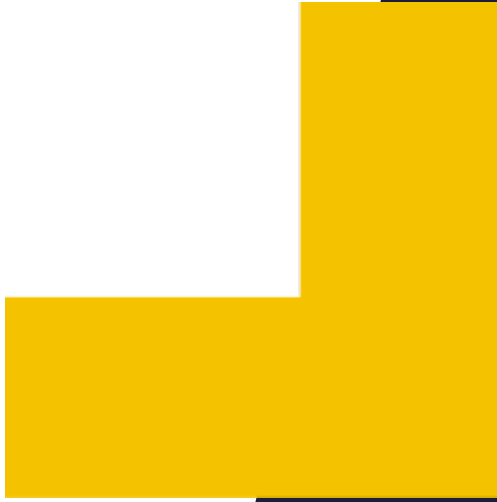


Spring y Apache Kafka

Tecnofor



CONTENIDO:

1. Introducción a Apache kafka
 2. Productor en Apache Kafka
 3. Consumidor en Apache Kafka
 4. Integración con Spring Boot
 5. Procesamiento de mensajes
 6. Arquitectura dirigida a eventos
 7. Microservicios dirigidos a eventos con Spring Boot y Apache Kafka
- 



1. Introducción a Apache Kafka





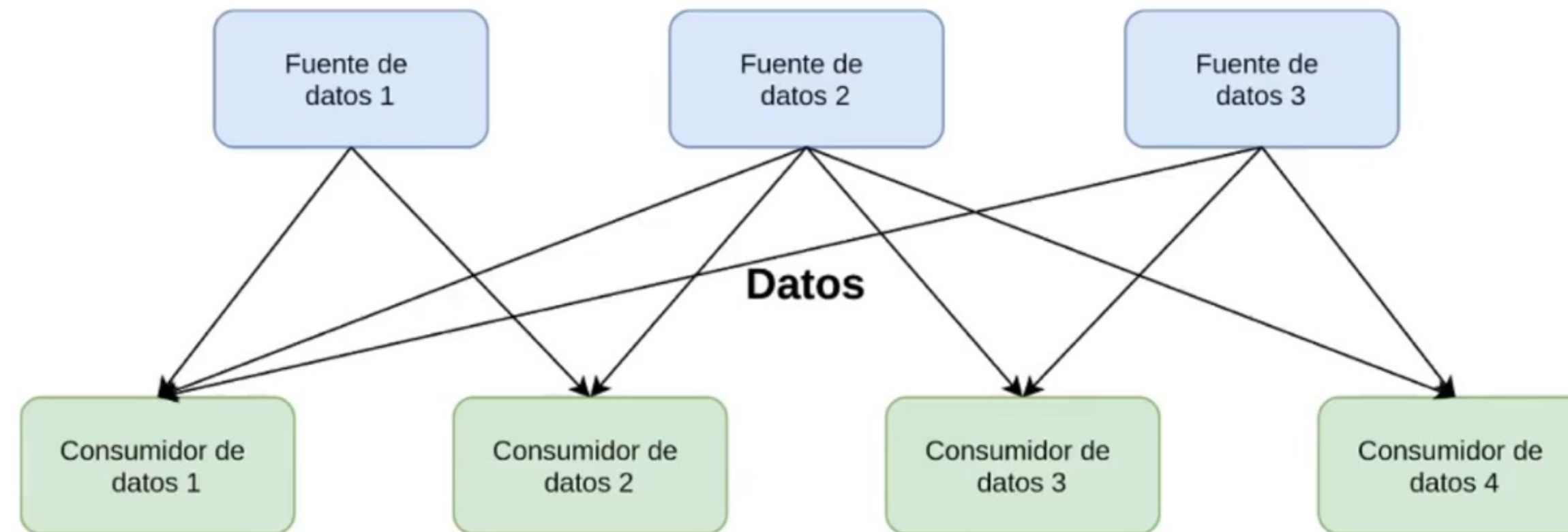
Definición de Apache Kafka

- Apache Kafka es una plataforma distribuida de transmisión de datos que permite publicar, almacenar y procesar flujos de datos en tiempo real
- Está escrito en Java y Scala
- La idea inicial surge en LinkedIn como solución a un problema interno de desarrollo
- En 2011 Apache se hace cargo del proyecto

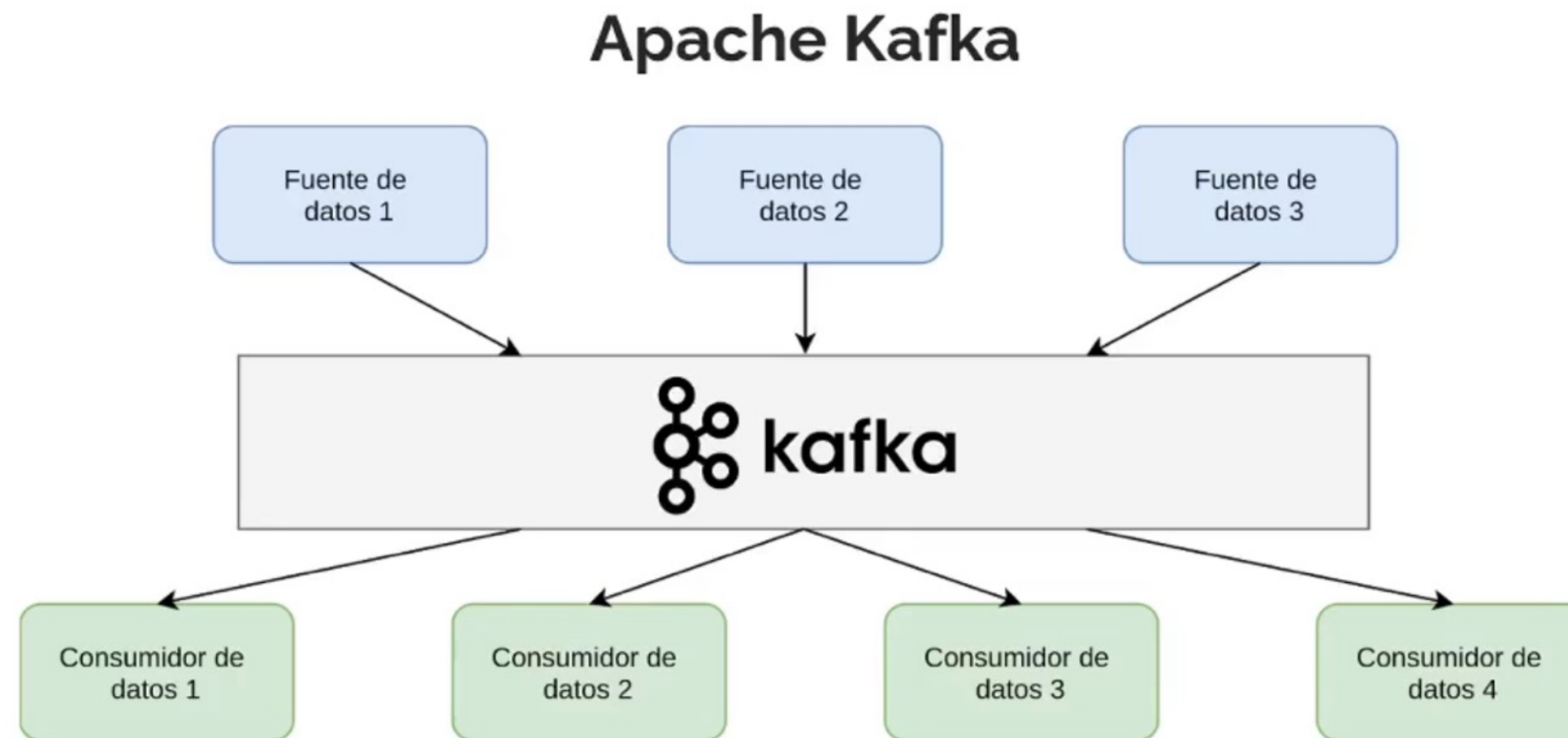


Ventajas de Apache Kafka

El problema



Ventajas de Apache Kafka

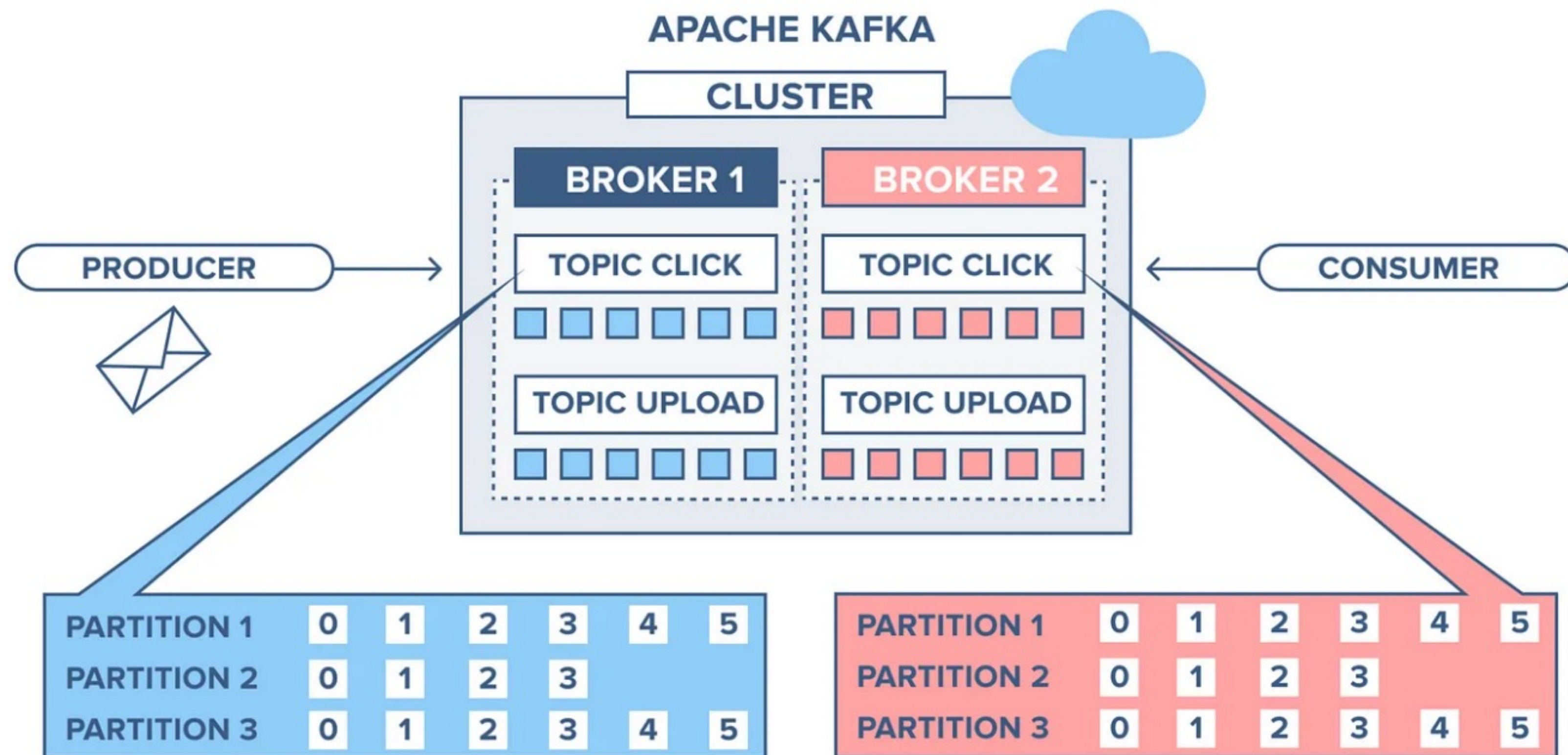




Ventajas de Apache Kafka

- Es open source
- Es escalable, persistente y tolerante a fallos
- Tiene una baja latencia (menos de 10 mseg) lo que hace que sea apto para procesos en tiempo real
- Permite construir flujos de datos en tiempo real entre aplicaciones
- Permite construir aplicaciones que reaccionan a eventos en tiempo real.

Arquitectura de Apache Kafka



Modelo editor-subscriptor

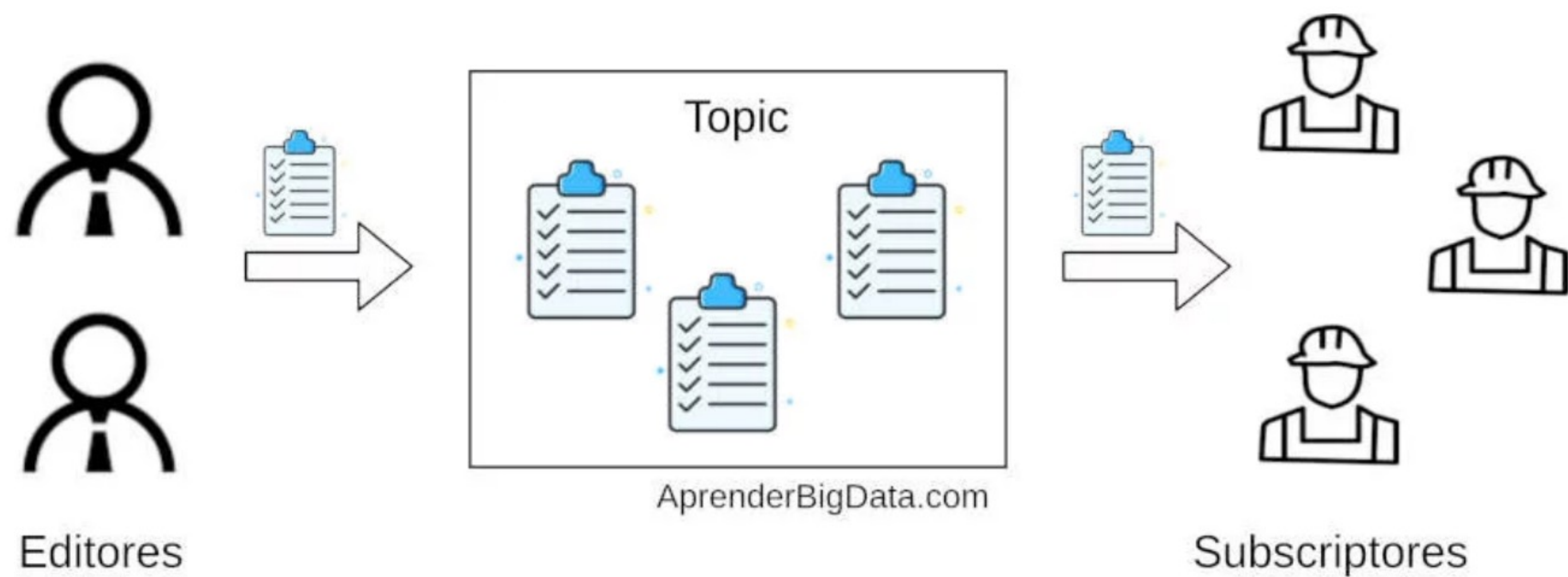


Diagrama patrón editor subscriptor

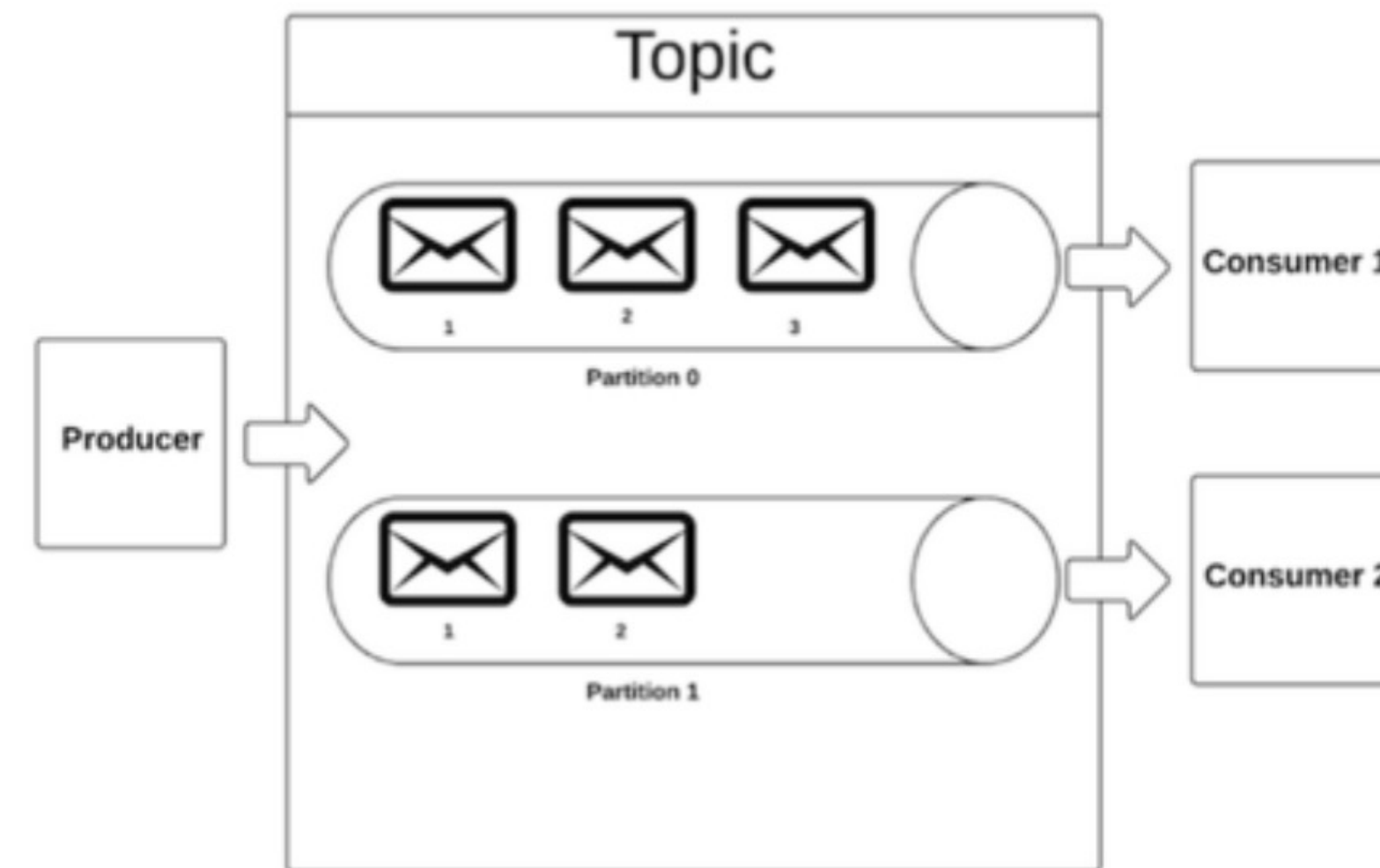


Broker

- Aumenta el desacoplamiento
- Enrutan los mensajes
- Desacoplan las aplicaciones productoras de consumidores
- Organizan y comprueban mensajes
- Almacenan los mensajes
- Tener un cluster de Brokers proporciona escalabilidad y fiabilidad. Si uno falla, esta el resto.

Topics

- Los topics son las categorías en las que se clasifican los mensajes almacenados.
- Son similares a las tablas en bases de datos.





Particiones

- Cada partición es una secuencia de mensajes ordenada e inmutable.
- Permiten distribuir los datos.
- A cada mensaje se le asigna una clave de partición que determina a que partición irá.
- Si no se le asigna, se calcula de forma aleatoria.
- Se intentan repartir de forma balanceada.

Offset

- Es un identificador incremental asignado a cada mensaje.
- Cada partición tiene un offset asociado, que es la posición del último mensaje escrito en la partición.
- Para identificar un mensaje se necesita:
 - Nombre del topic
 - Partición
 - Offset



Configuración básica de un proyecto Apache Kafka

- Descargar y descomprimir Apache Kafka
- Levantar el servidor Apache zookeeper
- Levantar el servidor Kafka
- Crear un topic
- Crear un productor para enviar mensajes
- Crear un consumidor para consumir mensajes



2. Productor en Apache Kafka





Productores

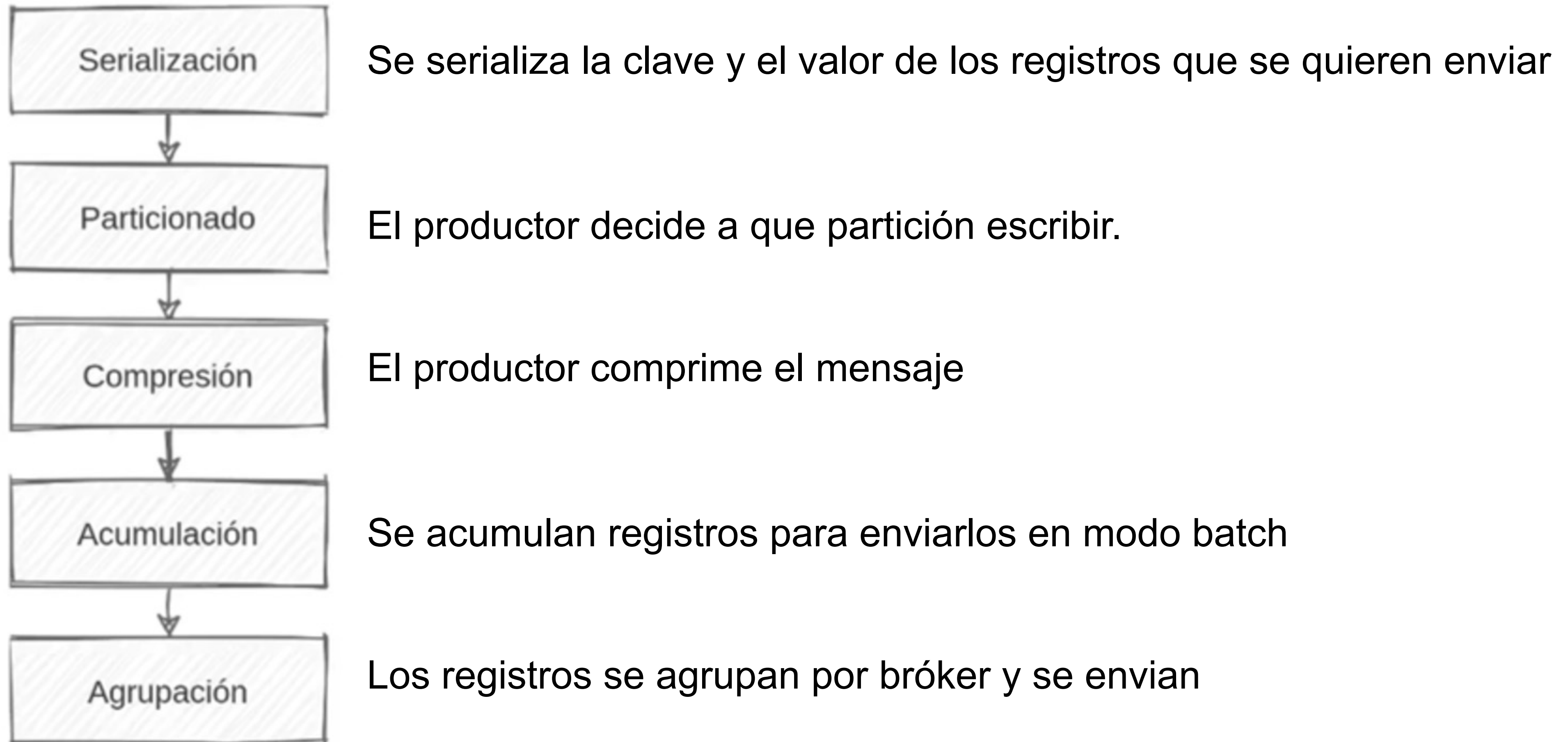
- Son los encargados de publicar mensajes en un bróker
- Son los responsables de serializar, particionar, comprimir y repartir la carga entre los brokers.
- Son los responsables de elegir que mensajes asignar a que partición usando:
 - Round Robin
 - Alguna función semántica que determine la partición

Configuración del productor

```
1 KafkaProducer producer = createKafkaProducer(  
2     "bootstrap.servers", "localhost:9092",  
3     "transactional.id", "my-transactional-id");
```

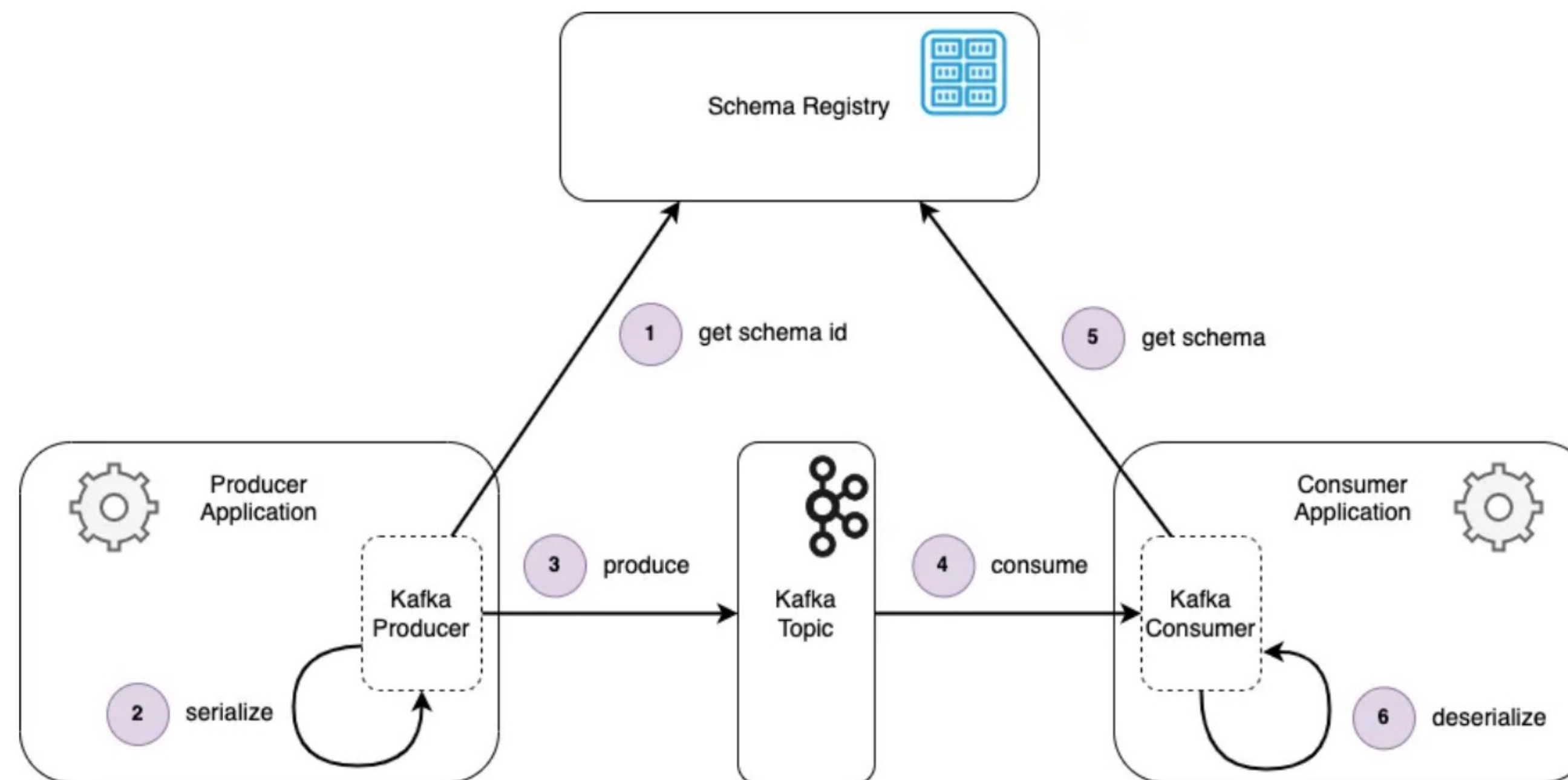



Envío de mensajes con el productor





Serialización y deserialización de mensajes





Serialización y deserialización de mensajes

- **Serialización:** Al serializar objetos de datos en formato binario, los productores de Kafka consultan el Registro de esquemas para obtener el esquema correspondiente al tipo de datos que se está serializando. El esquema se utiliza para codificar el objeto de datos en una representación binaria conforme con el formato del esquema (por ejemplo, el esquema Avro). A continuación, los datos serializados, junto con el ID del esquema, se publican en los temas de Kafka.
- **Deserialización:** Cuando se consumen mensajes de temas de Kafka, >> los consumidores de Kafka recuperan el ID de esquema asociado a cada mensaje a partir de una cabecera de mensaje. >> El ID del esquema se utiliza para obtener el esquema correspondiente del Registro de Esquemas. >> A continuación, los deserializadores utilizan el esquema recuperado para decodificar la carga binaria del mensaje y devolverla a su forma original de objeto de datos.



3. Consumidor en Apache Kafka



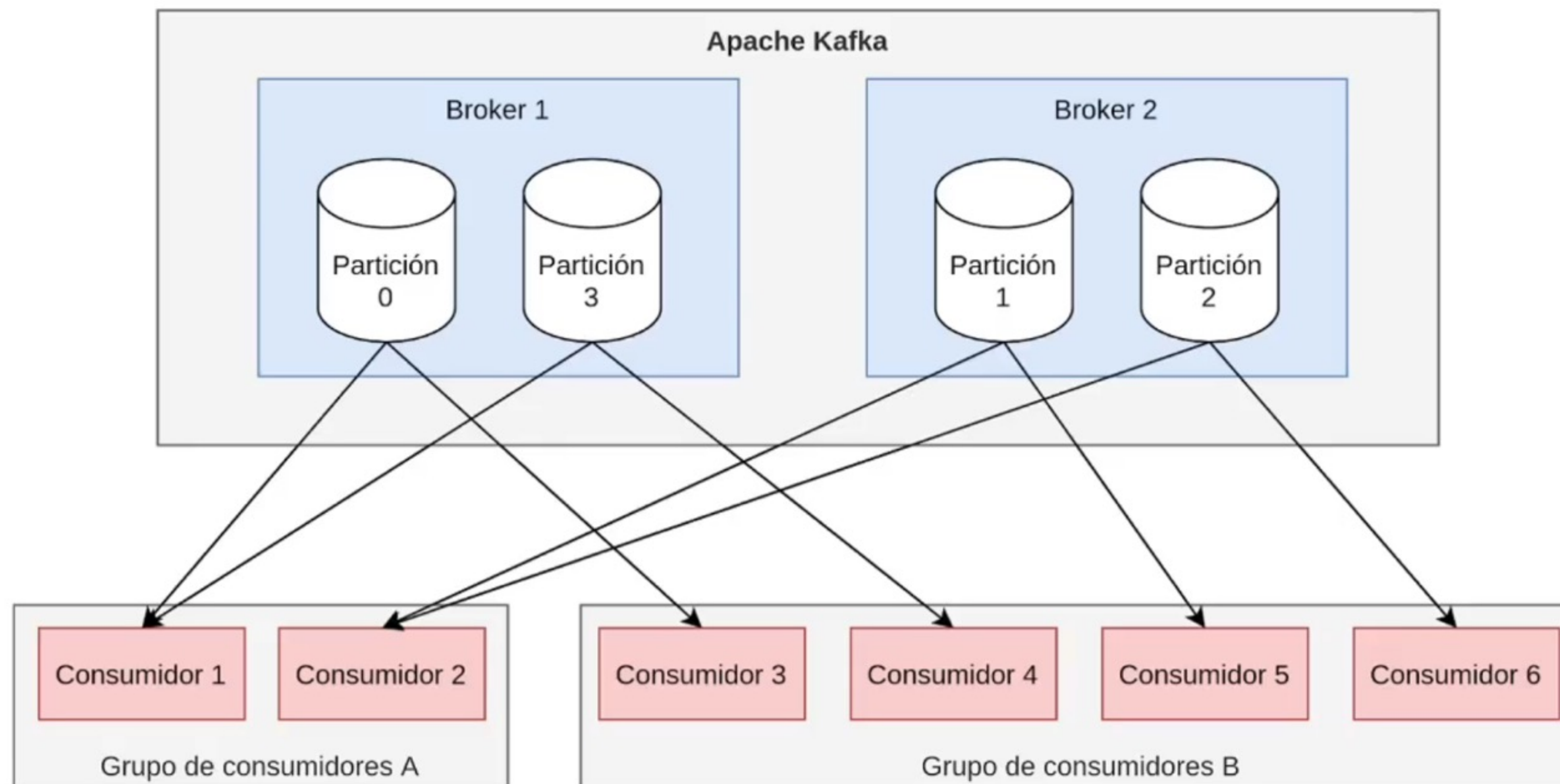


Consumidores y grupos de consumidores

- Son los clientes conectados, suscritos a los topics y consumen los mensajes
- Cada consumidor está asociado a un grupo de consumidores
- Cada mensaje solo es leído por un consumidor de cada grupo
- Cada partición es consumida por un solo consumidor en cada grupo de consumidores.



Consumidores y grupos de consumidores



Configuración del consumidor

```
9  KafkaConsumer consumer = createKafkaConsumer(  
10    "bootstrap.servers", "localhost:9092",  
11    "group.id", "my-group-id",  
12    "isolation.level", "read_committed");
```




Consumo de mensajes

```
15 consumer.subscribe.singleton("inputTopic");
```

```
ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(10));  
  
for (ConsumerRecord<String, String> record : records) {  
    System.out.print("Topic: " + record.topic() + ", ");  
    System.out.print("Partition: " + record.partition() + ",");  
  
    System.out.print("Key:" + record.key() + ", ");  
    System.out.println("Value: " + record.value() + ", ");  
}
```



4. Integración con Spring Boot





Creación de aplicaciones Spring Boot y Apache Kafka

- Es necesario tener:
 - Levantado el servidor zookeeper
 - Arrancado el servidor de Kafka
 - Creado un topic
- Spring Boot nos facilita la tarea de enviar mensajes a través de KafkaTemplates
- Los consumidores serán listeners en Spring Boot, de tal forma que escucharán los eventos.

Configuración de propiedades

```
spring.application.name=Ejemplo2_Kafka_Spring_Boot

# Configuración de los servidores Kafka
spring.kafka.bootstrap-servers=localhost:9092,localhost:9093,localhost:9094

# Configuración del grupo de consumidores
spring.kafka.consumer.group-id=grupo1

# Configuración del topic
message.topic.name=topic-test

# Configuración de otros parametros
spring.kafka.consumer.max-poll-records=100
spring.kafka.consumer.max-poll-interval-ms=1000
spring.kafka.consumer.heartbeat-interval-ms=1000
```





Gestión de dependencias


```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

5. Procesamiento de mensajes





Transformación y validación de mensajes

- **Manipulación de mensajes:** Los interceptores permiten manipular los mensajes antes de que sean enviados o recibidos, lo que permite agregar funcionalidad adicional a los productores y consumidores de Kafka.
- **Reacción a eventos:** Los interceptores pueden ser utilizados para detectar y reaccionar a eventos específicos en tiempo real, como la recepción de un mensaje que cumpla con ciertos criterios.
- **Escalabilidad:** Los interceptores pueden ser utilizados para escalar el sistema, permitiendo la detección y procesamiento de eventos en tiempo real.
- **Flexibilidad:** Los interceptores permiten una gran flexibilidad en la implementación de la lógica de negocio, ya que pueden ser utilizados para realizar operaciones específicas y personalizadas.



Uso de interceptores

- **Producer Interceptor:** Interceptores que se aplican a los productores de Kafka, permitiendo la manipulación de los mensajes antes de que sean enviados a los brokers. Estos interceptores pueden ser utilizados para agregar metadatos, realizar transformaciones de datos, o incluso rechazar mensajes que no cumplan con ciertos criterios.
- **Consumer Interceptor:** Interceptores que se aplican a los consumidores de Kafka, permitiendo la manipulación de los mensajes antes de que sean procesados por el consumidor. Estos interceptores pueden ser utilizados para filtrar, transformar o rechazar mensajes, o incluso para realizar operaciones de negocio específicas.



6. Arquitectura dirigida a eventos





Qué es la arquitectura dirigida a eventos

- La arquitectura dirigida a eventos (EDA) es un paradigma de diseño de software que se enfoca en la producción, detección, consumo y reacción a eventos.
- En este enfoque, los componentes del sistema se comunican a través del intercambio de eventos, lo que permite una mayor flexibilidad, escalabilidad y desacoplamiento entre los diferentes módulos.



Qué es la arquitectura dirigida a eventos

- **Conceptos clave**
 - **Evento:** Un evento es algo que sucede en el sistema y que puede ser de interés para otros componentes. Puede ser una acción del usuario, un cambio de estado, un error, etc.
 - **Productor de eventos:** Un componente que genera y publica eventos en el sistema.
 - **Consumidor de eventos:** Un componente que escucha y reacciona a los eventos publicados.
 - **Intermediario de eventos:** Un componente que facilita la comunicación entre productores y consumidores de eventos, como un bus de eventos o un broker de mensajes.



Funcionamiento

1. Producción de Eventos

- **Productor de eventos:** Un componente del sistema, como un servicio o una aplicación, genera un evento cuando algo importante sucede. Por ejemplo, un servicio de autenticación puede generar un evento cuando un usuario se autentica correctamente.
- **Evento:** El evento contiene información relevante sobre lo que ha sucedido, como el tipo de evento, los detalles del evento y cualquier otra información necesaria para su procesamiento.

2. Publicación de Eventos

- **Intermediario de eventos:** El productor de eventos publica el evento en un intermediario de eventos, como un bus de eventos o un broker de mensajes. Este intermediario almacena y gestiona los eventos hasta que sean consumidos.

3. Suscripción a Eventos

- **Consumidor de eventos:** Un componente del sistema, como un servicio o una aplicación, se suscribe a los eventos que le interesan. Por ejemplo, un servicio de notificaciones puede suscribirse a eventos de autenticación para recibir notificaciones cuando un usuario se autentica correctamente.



Funcionamiento

4. Consumo de Eventos

- Consumidor de eventos: El consumidor de eventos escucha los eventos publicados en el intermediario de eventos y reacciona según sea necesario. Por ejemplo, el servicio de notificaciones puede recibir el evento de autenticación y enviar una notificación al usuario.

5. Reacción a Eventos

- Consumidor de eventos: El consumidor de eventos procesa el evento y toma medidas adecuadas. Por ejemplo, el servicio de notificaciones puede enviar una notificación al usuario cuando se autentica correctamente.



Ventajas

- **Desacoplamiento:** Los componentes del sistema están desacoplados, lo que permite una mayor flexibilidad y escalabilidad.
- **Asincronía:** La comunicación entre componentes es asíncrona, lo que significa que los productores y consumidores de eventos no necesitan estar activos al mismo tiempo.
- **Escalabilidad:** Es más fácil escalar el sistema agregando más consumidores de eventos para procesar eventos en paralelo.
- **Extensibilidad:** Es más fácil agregar nuevos tipos de eventos y consumidores sin afectar el funcionamiento existente del sistema.



7. Microservicios dirigidos a eventos con Spring Boot y Apache Kafka

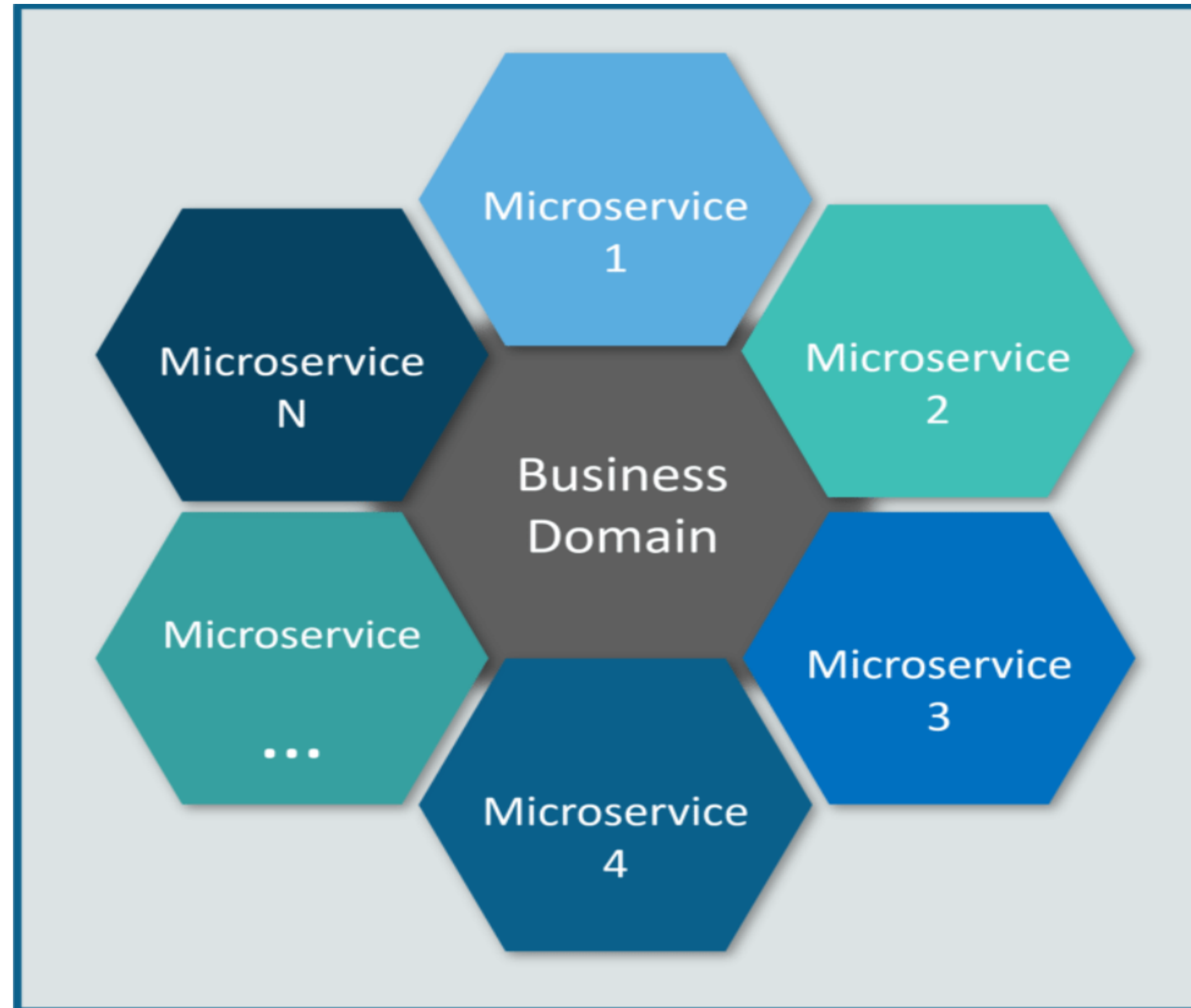




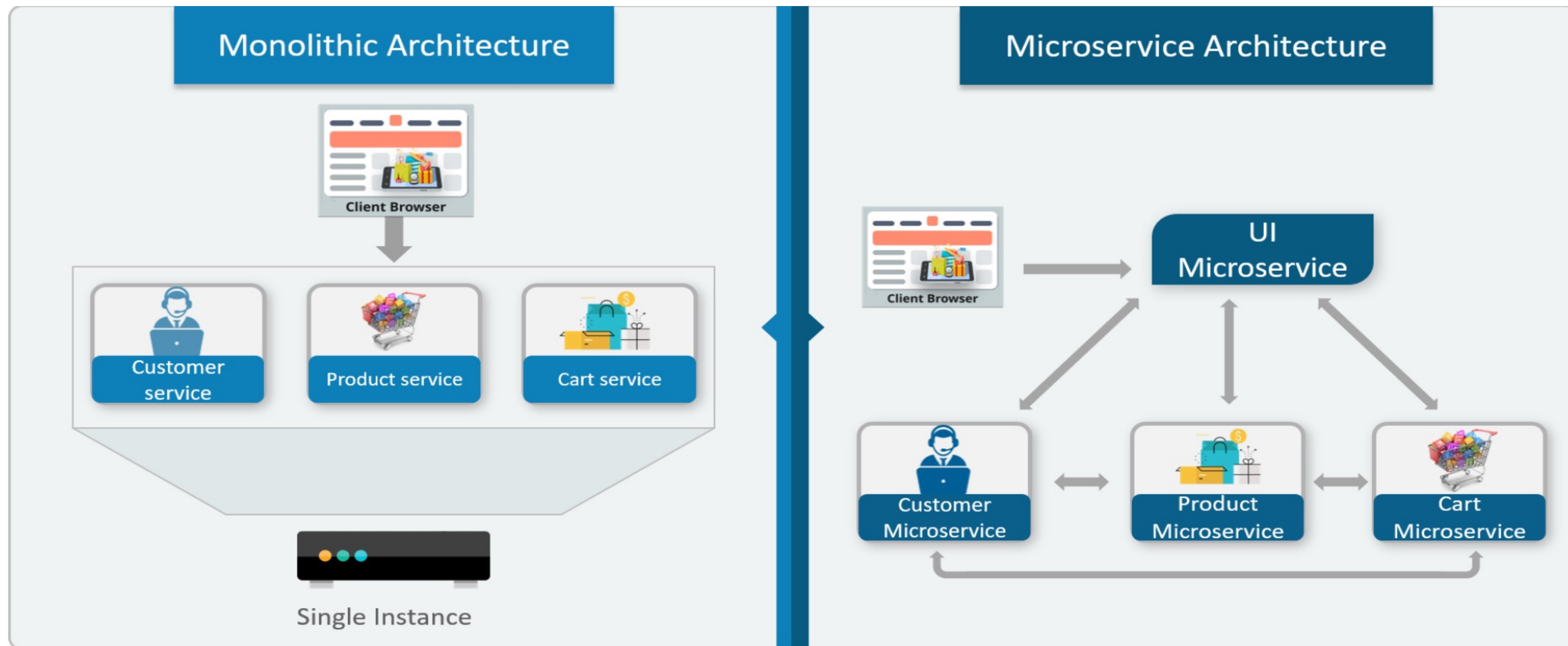
Que es un microservicio?

- Según Martin Fowler y James Lewis explican en su artículo Microservices, los microservicios se definen como un estilo arquitectural, es decir, una forma de desarrollar una aplicación, basada en un conjunto de pequeños servicios, cada uno de ellos ejecutándose de forma autónoma y comunicándose entre si mediante mecanismos livianos, generalmente a través de peticiones REST sobre HTTP por medio de sus APIs.

Que es un microservicio?



Arquitectura monolitica vs Arquitectura microservicios



Tendencia en el desarrollo

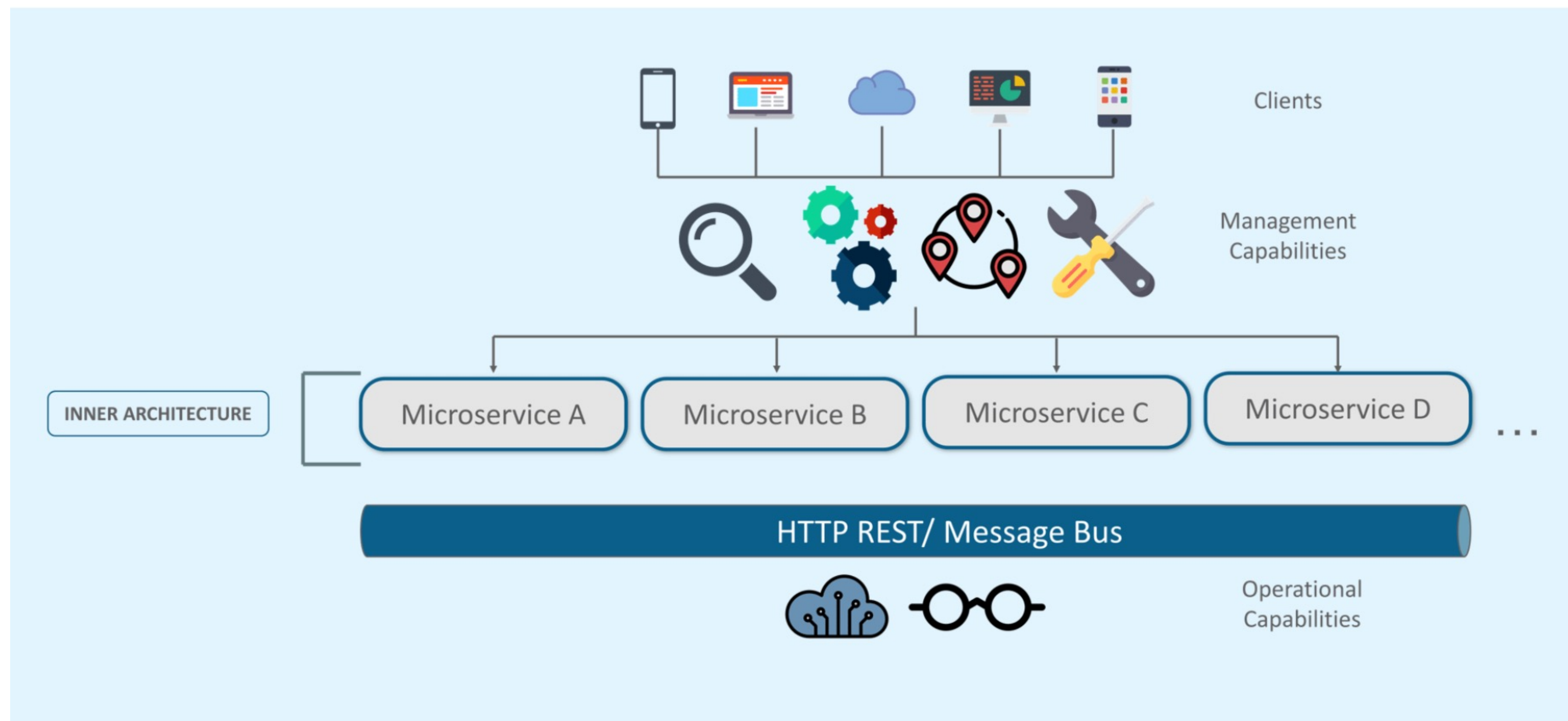
- La tendencia es que las aplicaciones sean diseñadas con un enfoque orientado a microservicios, construyendo múltiples servicios que colaboran entre si, en lugar del enfoque monolítico, donde se construye y despliega una única aplicación que contenga todas las funcionalidades.



Características de los Microservicios

1. Pueden ser auto-contenidos, de tal forma que incluyen todo lo necesario para prestar su servicio
2. Servicios pequeños, lo que facilita el mantenimiento. Ej: Personas, Productos, Posición Global, etc
3. Principio de responsabilidad única: cada microservicio hará una única cosa, pero la hará bien
4. Políglotas: una arquitectura basada en microservicios facilita la integración entre diferentes tecnologías (lenguajes de programación, BBDD...etc)
5. Despliegues unitarios: los microservicios pueden ser desplegados por separado, lo que garantiza que cada despliegue de un microservicio no implica un despliegue de toda la plataforma. Tienen la posibilidad de incorporar un servidor web embebido como Tomcat o Jetty
6. Escalado eficiente: una arquitectura basada en microservicios permite un escalado elástico horizontal, pudiendo crear tantas instancias de un microservicio como sea necesario.

Arquitectura microservicios





Arquitectura microservicios

1. Diferentes clientes de diferentes dispositivos intentan usar diferentes servicios como búsqueda, creación, configuración y otras capacidades de administración
2. Todos los servicios se separan según sus dominios y funcionalidades y se asignan a microservicios individuales.
3. Estos microservicios tienen su propio balanceador de carga y entorno de ejecución para ejecutar sus funcionalidades y al mismo tiempo captura datos en sus propias bases de datos.
4. Todos los microservicios se comunican entre sí a través de un servidor sin estado que es REST o Message Bus.
5. Los microservicios conocen su ruta de comunicación con la ayuda de Service Discovery y realizan capacidades operativas tales como automatización, monitoreo
6. Luego, todas las funcionalidades realizadas por los microservicios se comunican a los clientes a través de la puerta de enlace API.
7. Todos los puntos internos están conectados desde la puerta de enlace API. Por lo tanto, cualquiera que se conecte a la puerta de enlace API se conecta automáticamente al sistema completo

Características de los microservicios

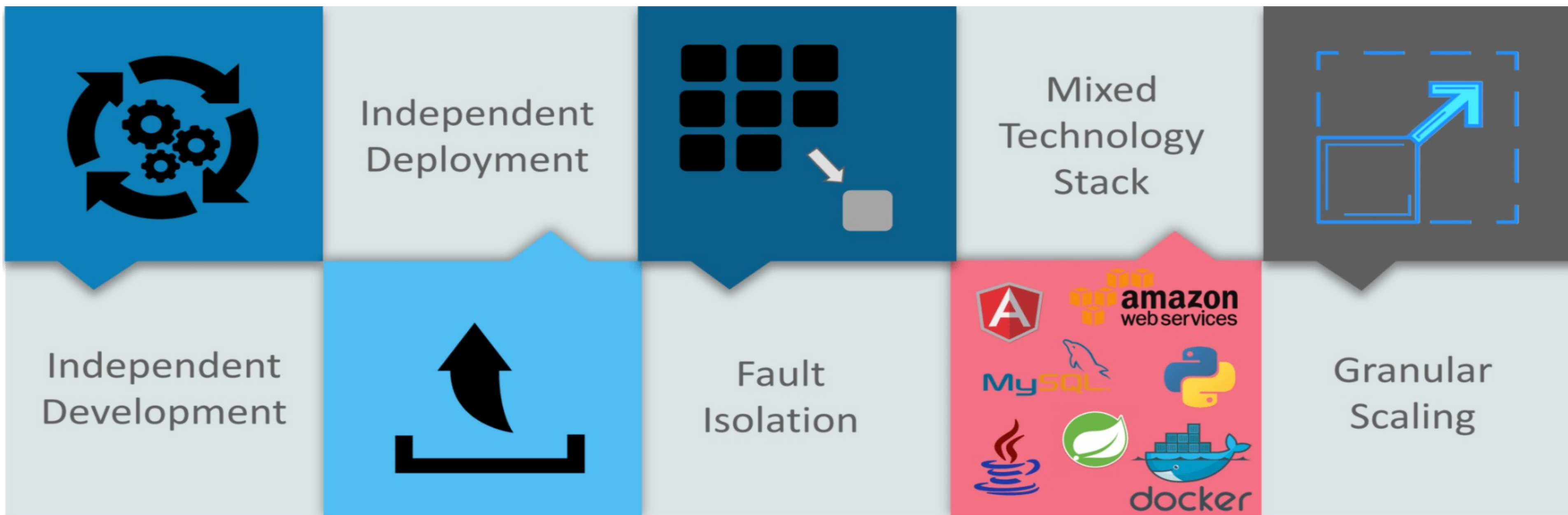




Características de los microservicios

1. **Desacoplamiento:** los servicios dentro de un sistema se desacoplan en gran medida. Por lo tanto, la aplicación en su conjunto se puede construir, modificar y escalar fácilmente.
2. **Componentes:** los microservicios se tratan como componentes independientes que se pueden reemplazar y actualizar fácilmente.
3. **Capacidades empresariales:** los microservicios son muy simples y se centran en una sola capacidad
4. **Autonomía:** los desarrolladores y los equipos pueden trabajar de forma independiente, lo que aumenta la velocidad.
5. **Entrega continua:** permite lanzamientos frecuentes de software, a través de la automatización sistemática de la creación, prueba y aprobación del software.
6. **Responsabilidad:** Los microservicios no se centran en aplicaciones como proyectos. En cambio, tratan las aplicaciones como productos de los que son responsables.
7. **Gobernanza descentralizada:** el enfoque está en usar la herramienta adecuada para el trabajo correcto. Eso significa que no hay un patrón estandarizado o ningún patrón tecnológico. Los desarrolladores tienen la libertad de elegir las mejores herramientas útiles para resolver sus problemas
8. **Agilidad** - Los microservicios apoyan el desarrollo ágil. Cualquier nueva característica puede ser desarrollada rápidamente y descartada nuevamente

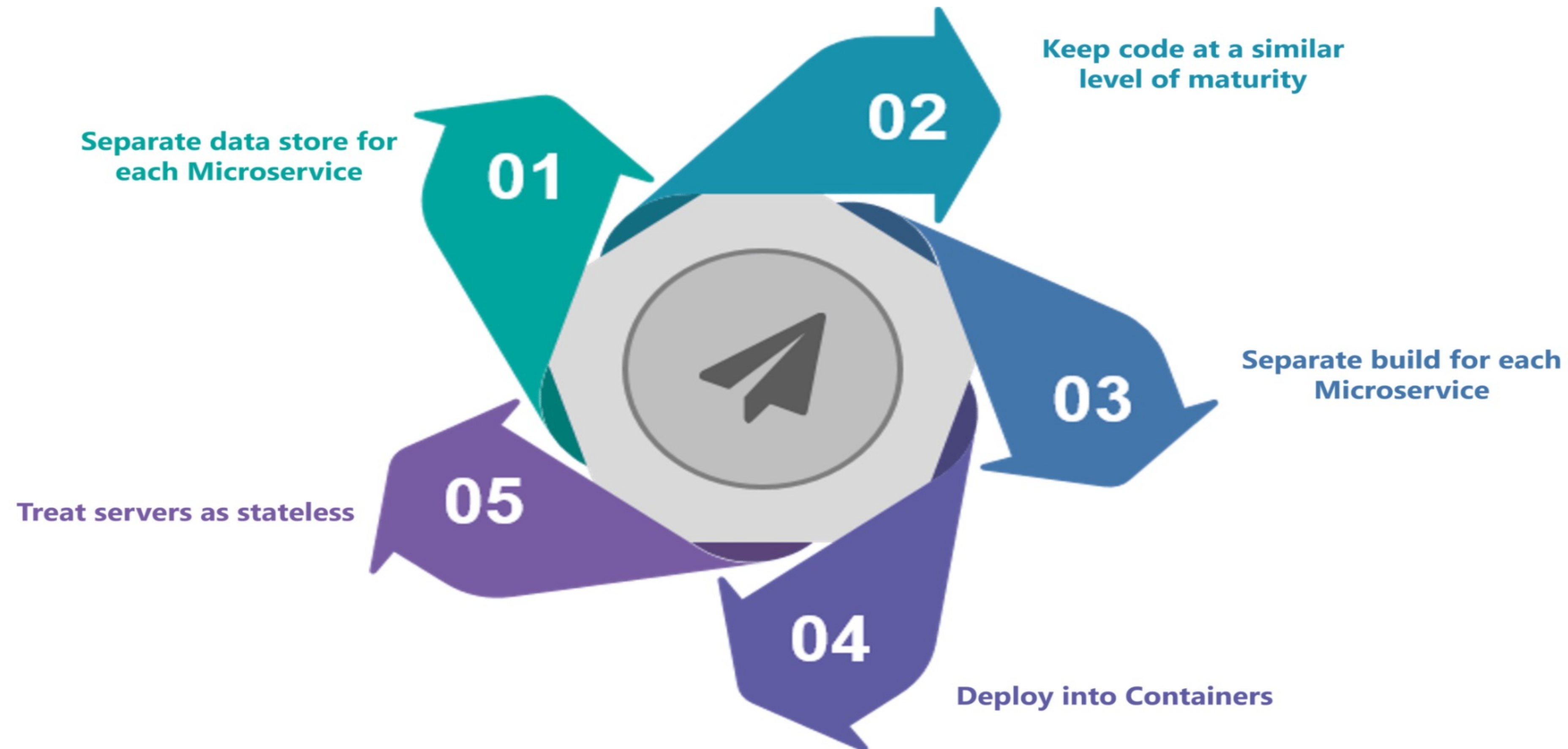
Ventajas de los microservicios



Ventajas de los microservicios

- **Desarrollo independiente:** todos los microservicios se pueden desarrollar fácilmente según su funcionalidad individual
- **Implementación independiente:** en función de sus servicios, se pueden implementar individualmente en cualquier aplicación
- **Aislamiento de fallos:** incluso si un servicio de la aplicación no funciona, el sistema continúa funcionando
- **Pila de tecnología mixta:** se pueden utilizar diferentes lenguajes y tecnologías para crear diferentes servicios de la misma aplicación
- **Escalado granular:** los componentes individuales pueden escalarse según la necesidad, no es necesario escalar todos los componentes juntos

Buenas practicas diseño



Quien los utiliza?

amazon.com®

NETFLIX

GILT



ebay

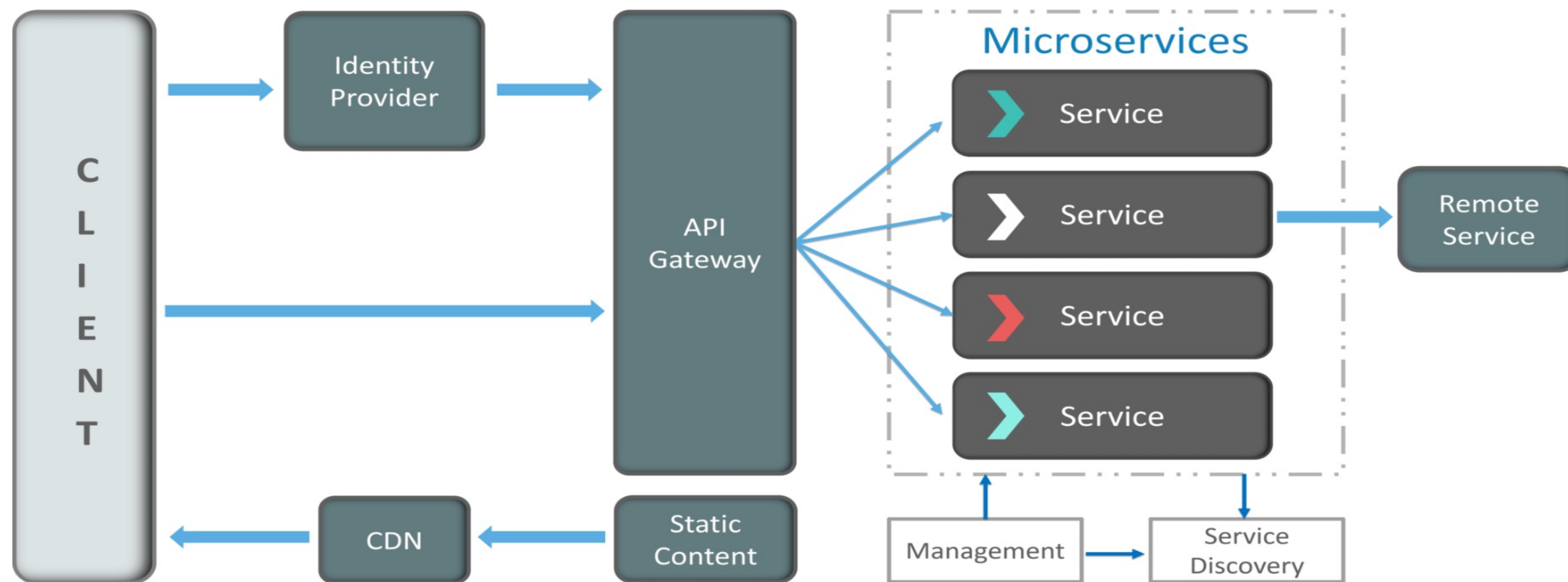


NORDSTROM

theguardian

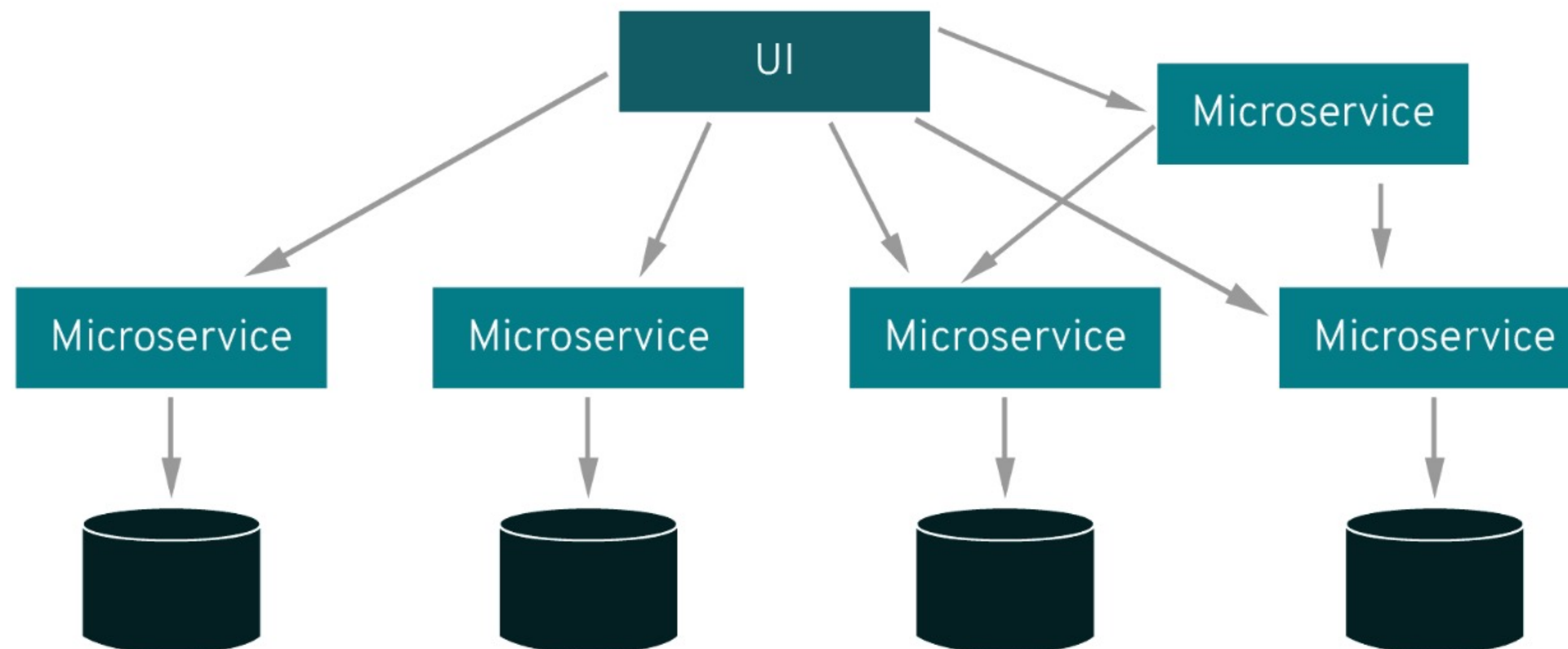
Tecnofor

Componentes de arquitectura





Integración de Kafka para comunicación asíncrona y desacoplada



Uso de Kafka para la gestión de eventos en tiempo real

```
@Service
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage(String message) {
        kafkaTemplate.send("topic-test", message);
    }
}

@Service
public class KafkaConsumer {

    @KafkaListener(topics = "topic-test")
    public void consume(String message) {
        System.out.println("Consumed message: " + message);
    }
}
```

Tecnofor



¡Gracias!

Plaza de la Independencia, 8
28001-Madrid

www.TecnoFor.es