powered by {CODEMOTION}

Frontend vitaminized from the backend

MIGUEL GARCIA SANGUINO

ING

CloudApps MASTER

Miguel García Sanguino

15 años como developer

Frontend 70% Backend 30%

Software engineer en ING

twitter.com/sanguinoide

github.com/sanguino

linkedin.com/in/sanguinoide

Frontend vitaminized from the backend

Trabajo de investigación TFM
Los procesos se han complicados
Las respuestas ya no son siempre únicas
Tenemos respuestas asincronas
Los datos se actualizan en el tiempo
No podemos tener un solo modelo de datos para front y middle
Relaciones entre muchos squads complican el desarrollo

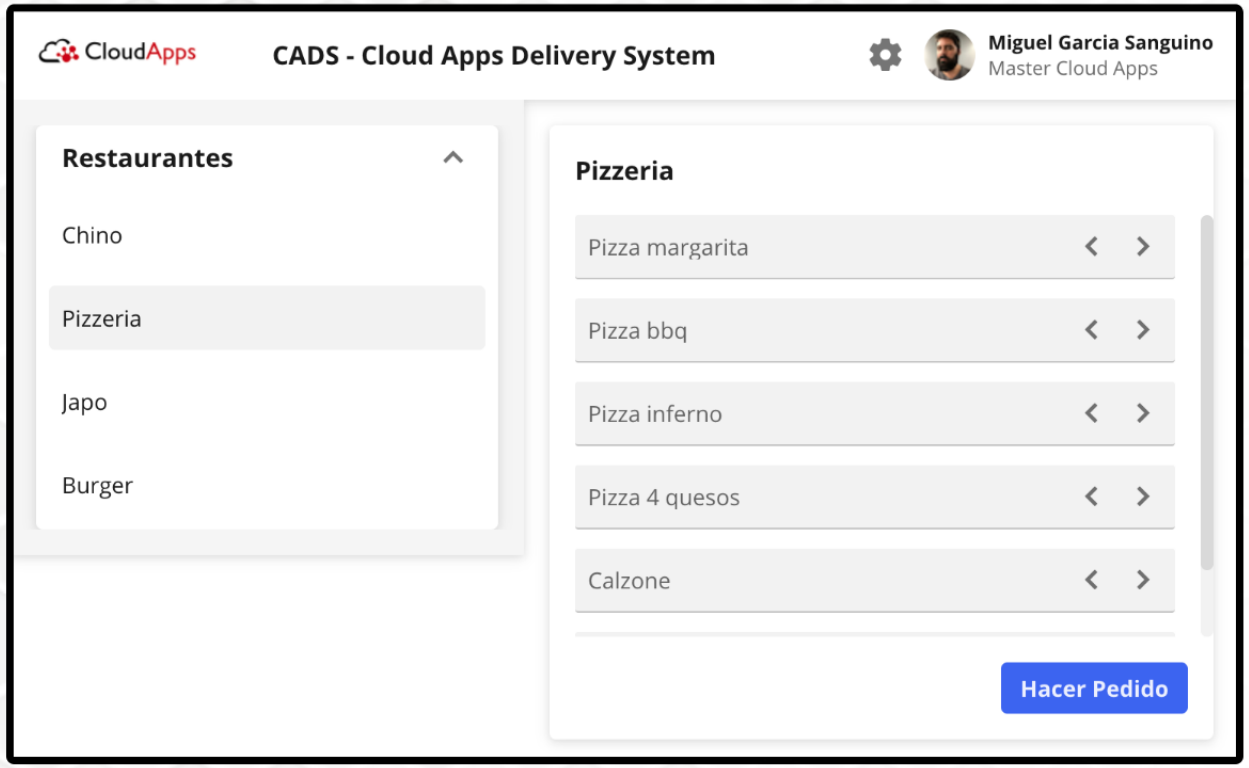
Planteamiento: añadir un servicio solo para el front

Front recibe los datos que necesita,
cuando los necesita,
en el formato que los necesita,
sin entorpecer a los desarrolladores middle,
ni en el modelo,
ni con desarrollos extra

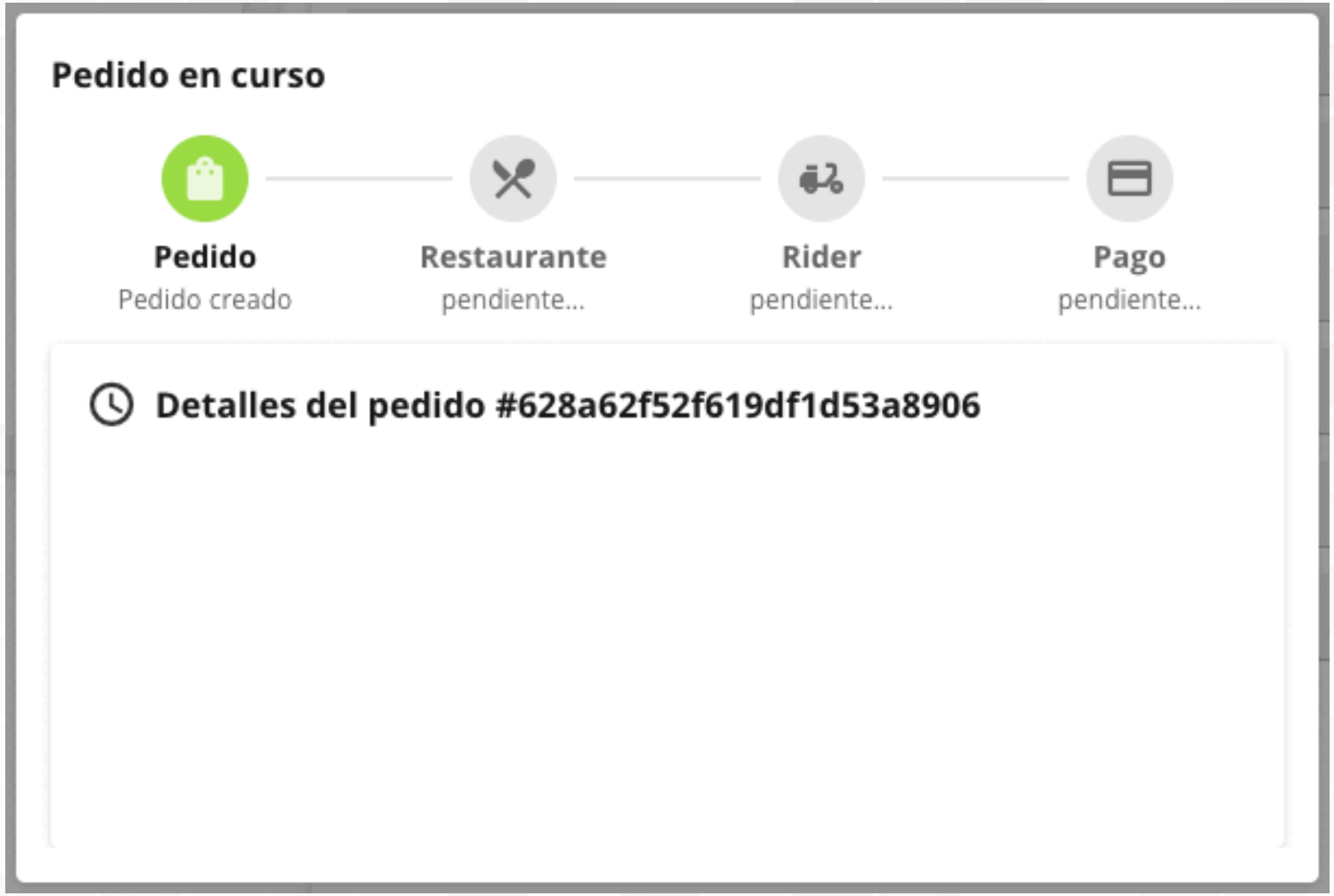
Caso de uso

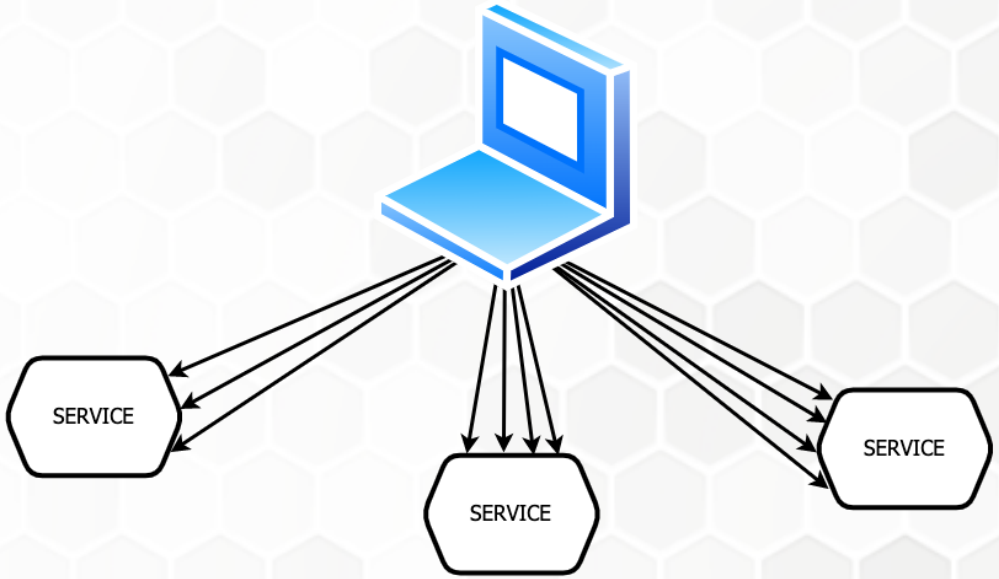
Pedido de comida online

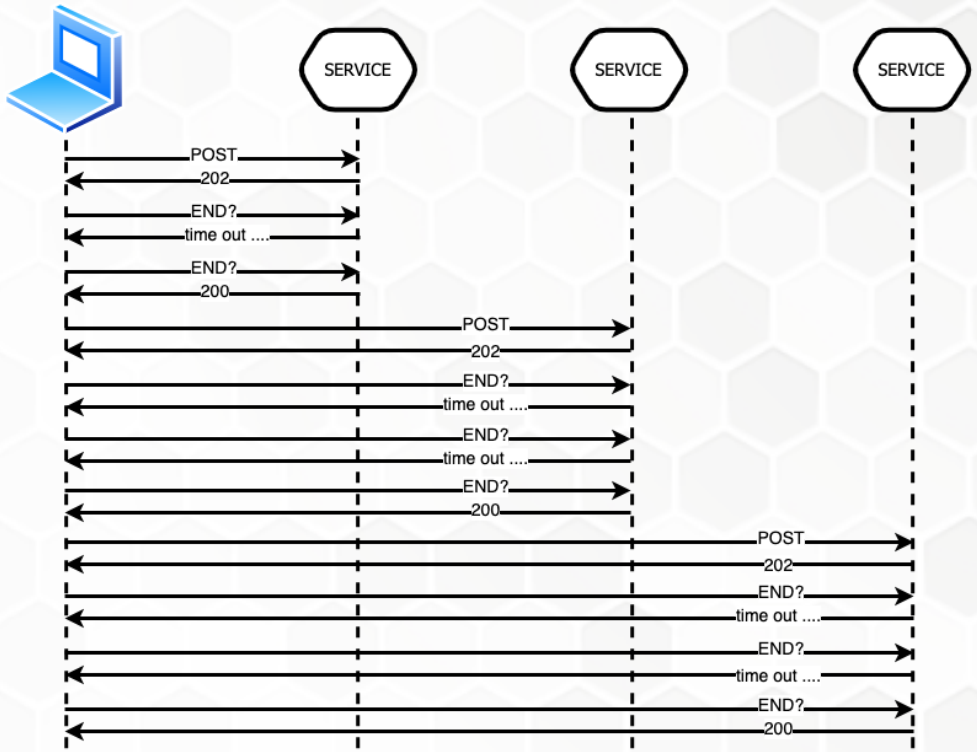
- Realizar pedido
- Responder rápido
- Actualizar estado
- Completar pedidos



Caso de uso


Pedido en curso
Pedido
Pedido creado
Restaurante
pendiente...
Rider
pendiente...
Pago
pendiente...
Detalles del pedido #628a62f52f619df1d53a8906





Antipatrones

Has creado el pedido ya?
Has creado el pedido ya?
Has creado el pedido ya?
Si
Has pedido la comida ya?
Has pedido la comida ya?
Has pedido la comida ya?
Si
Has reservado un rider ya?
Has reservado un rider ya?
...

Objetivos

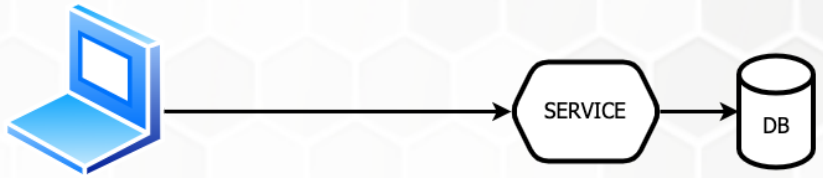
Servicios desacoplados

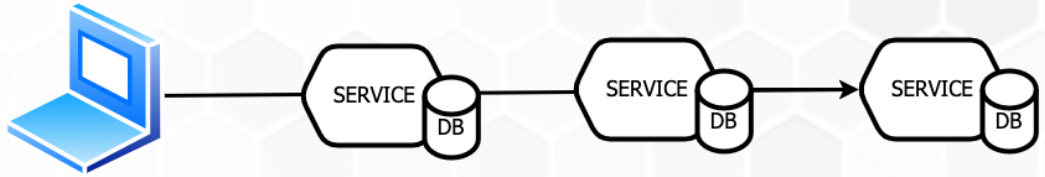
Respuestas multiples asincronas

El middleware no se tiene que preocupar del front

Escalables y resilientes

Desacoplamientos de squads, no solo técnico





Patron Saga

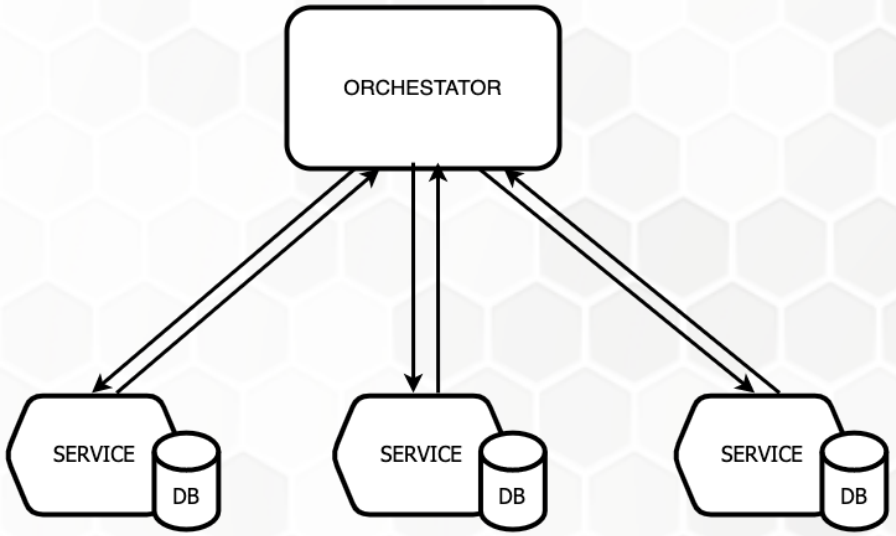
Transacción con microservicios

Cada serivicio hace 1 transacción

Si algo va mal rollback de todo

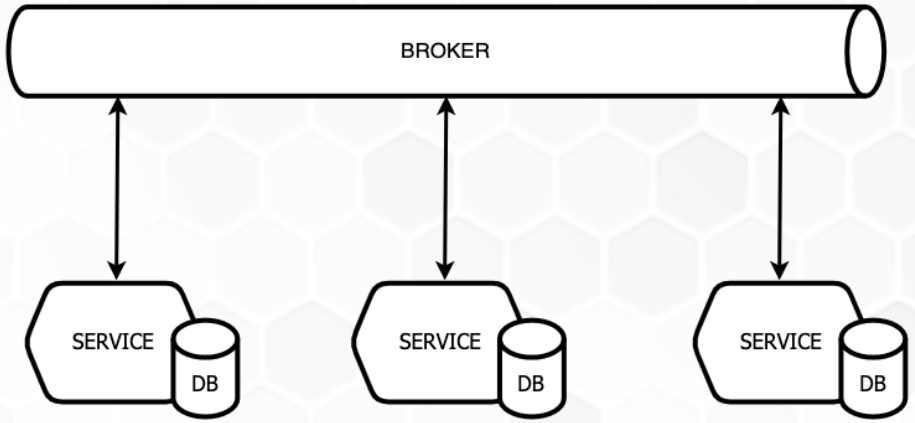
Asegura Consistencia

Orquestadas / Coreografiadas



Saga Orquestada

Orquestador llama y espera
Sencilla en procesos sincronos
Acoplamiento de servicios
+ dificil resiliencia y escalabilidad



Saga Coreografiada

Servicios reciben y envian eventos
Desacoplamiento de servicios
+ facil resiliencia y escalabilidad

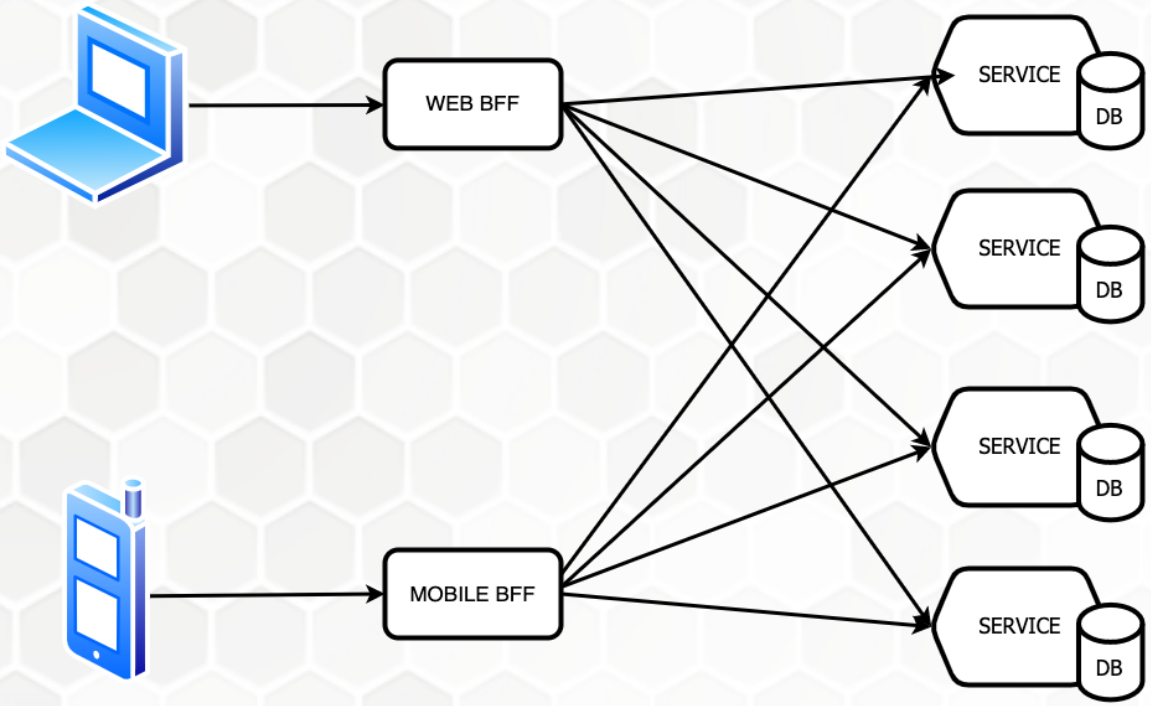
Patron Backend for Frontend

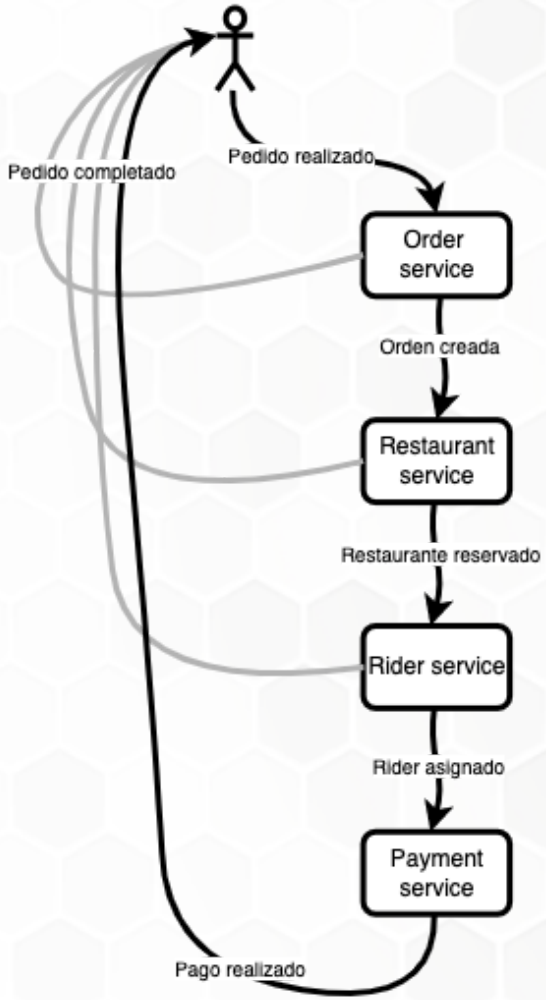
Uno por cada tipo de cliente

Adapta el api a cada consumidor

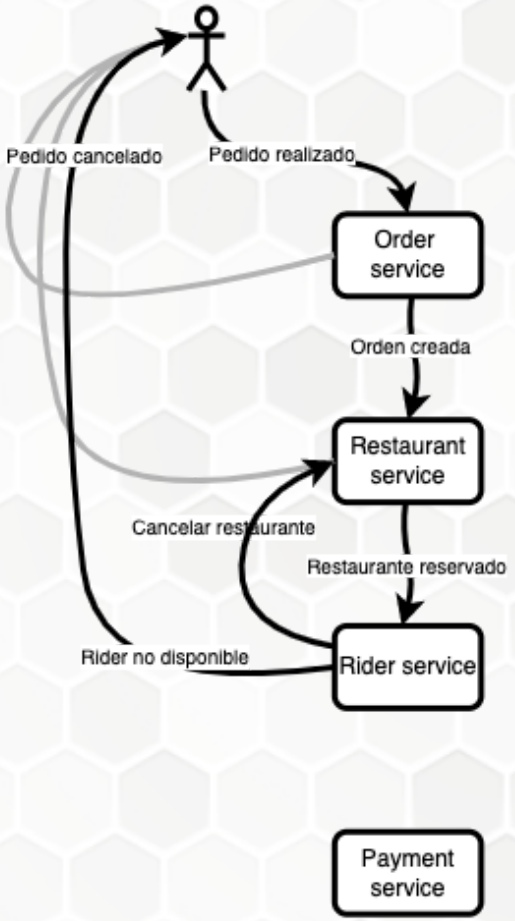
Simplifica clientes

Elimina la sobrecarga de servicios





Completada



Cancelada

Caso de uso

Cada paso un servicio

Rollback en caso de fallo

Informar al usuario en cada paso

Objetivos:

Servicios desacoplados
Respuestas multiples asincronas
El middleware no se tiene que preocupar del front
Escalables y resilientes
Desacoplamientos de squads, no solo técnico

Middleware

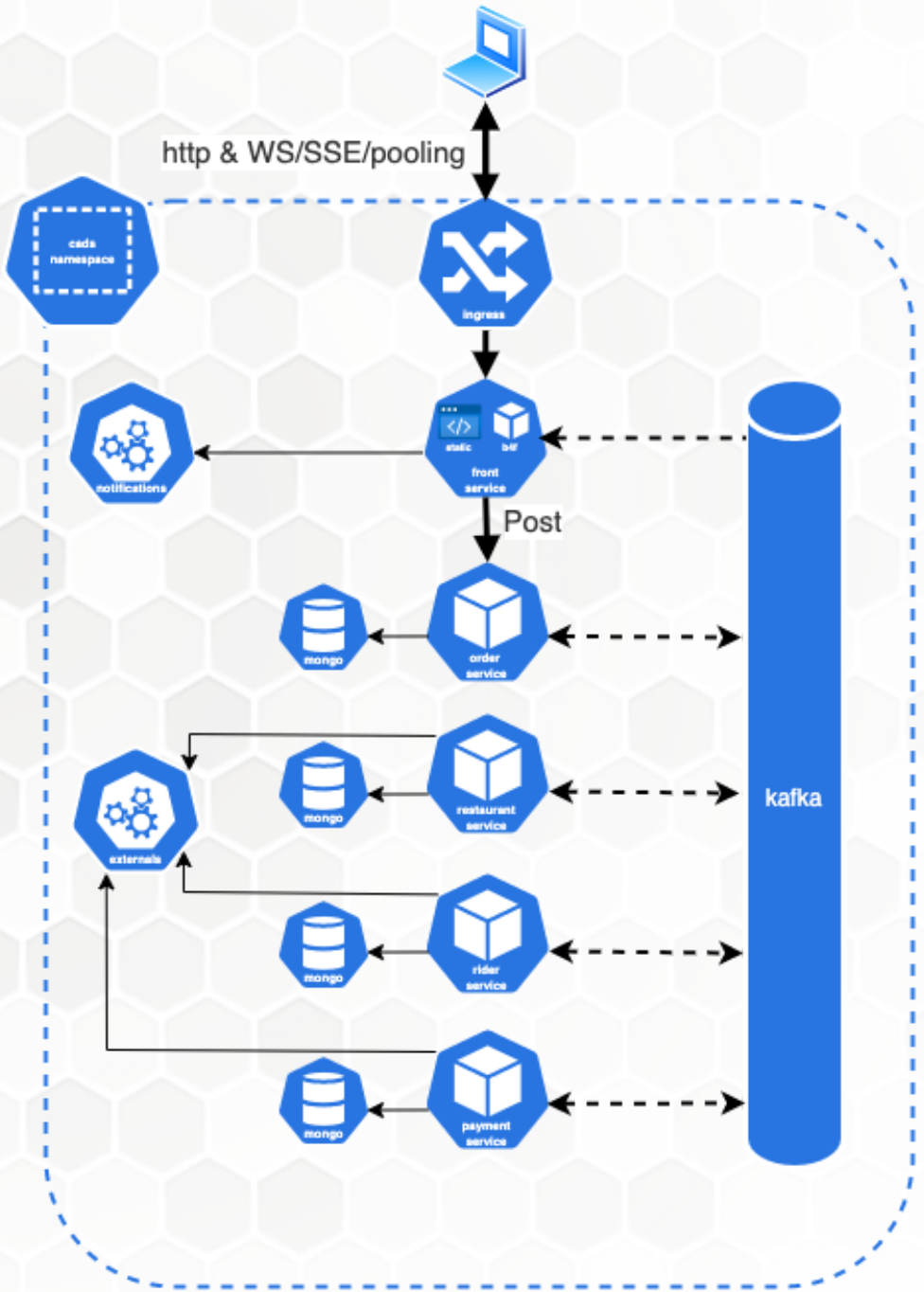
- servicios desacoplados
- saga coreografiada, sin estados
- comunicacion por eventos
- escalables y resilientes

Frontend

- pervertir patrón BFF
- consume de los eventos
- independiente y asíncrono
- notificaciones online / offline

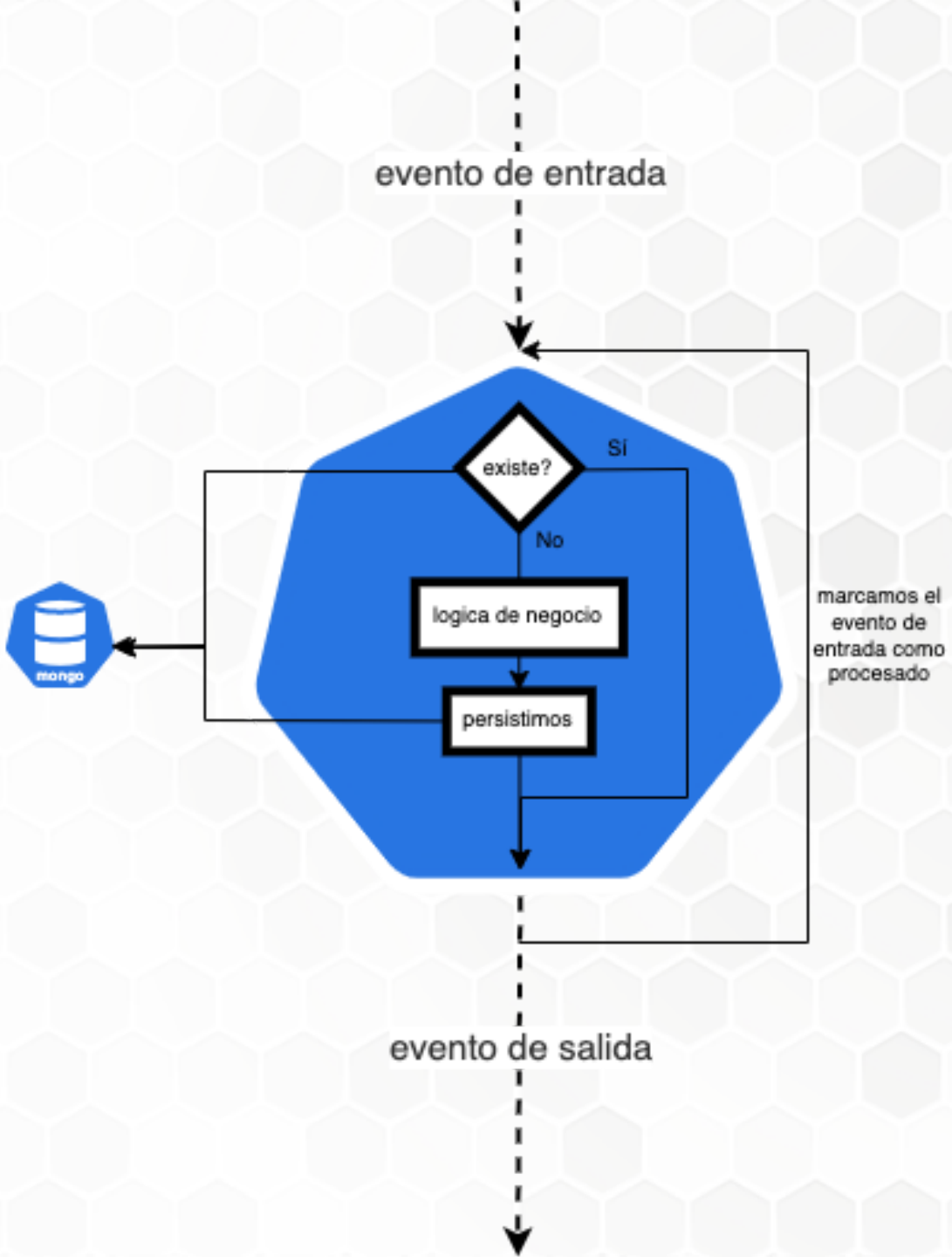
Arquitectura

- ingress
- front
- servicios + bases de datos
- externals ~ Mocks
- notificaciones ~ Mocks
- Zookeeper y kafka



Stack

- Kubernetes
- Kafka
- Mongo DB
- Nodejs
- kafkajs
- express
- mongoose
- Rollup como builder
- Lit
- Kor-ui



Idempotencia

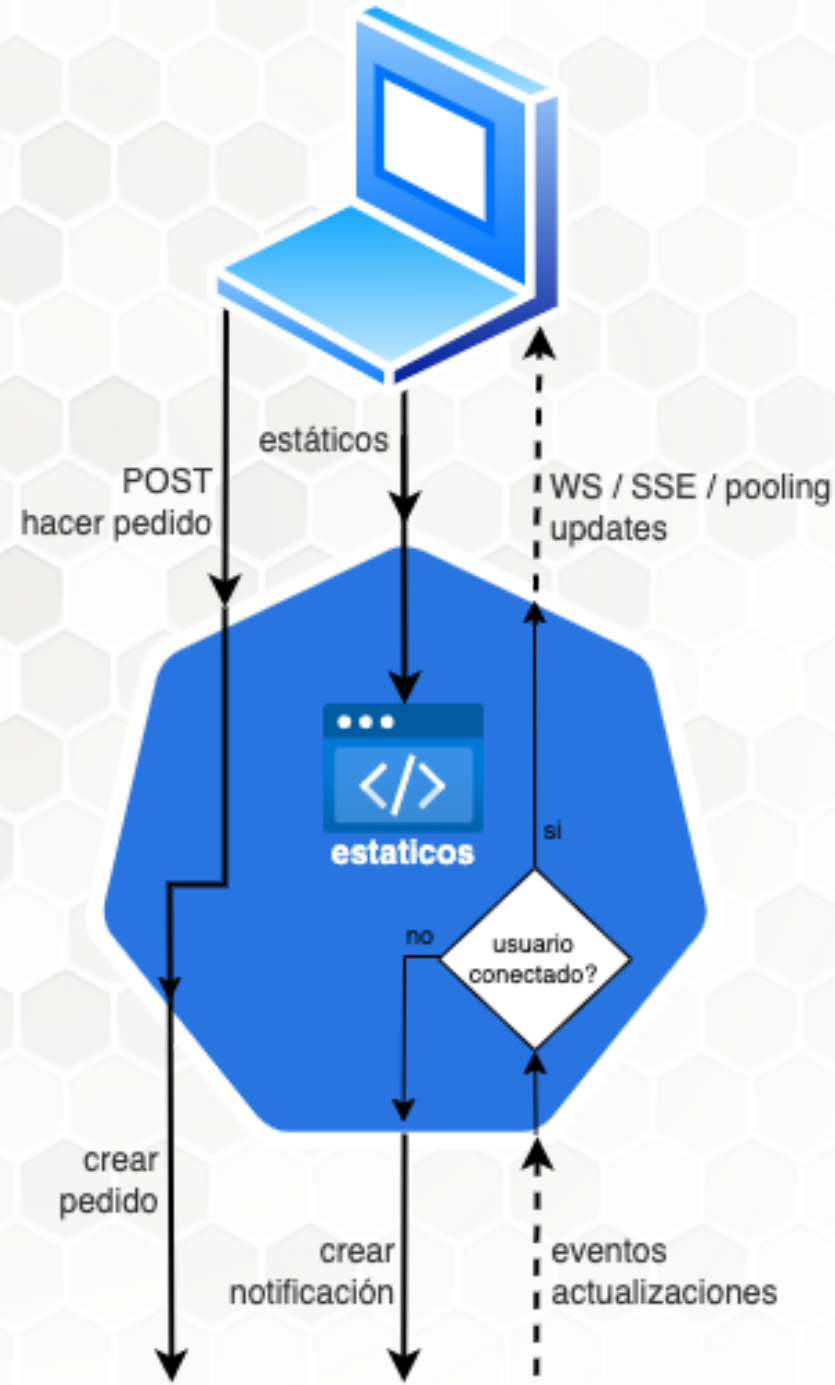
Marcamos el offset después de enviar la salida.
Deben ser idempotentes:

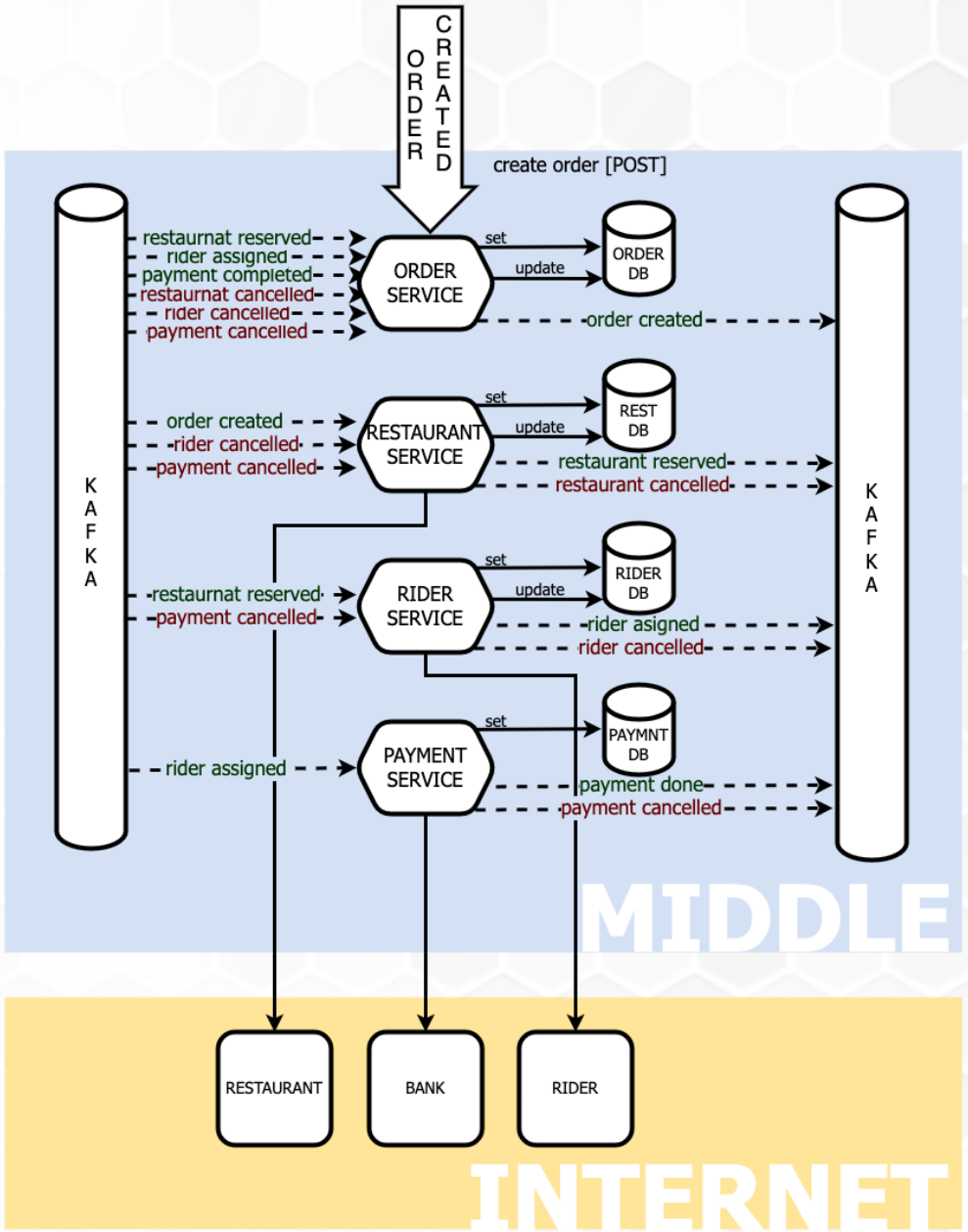
- todos los servicios de la saga
- los servicios externos
- los rollback de la saga

Estáticos y servicio juntos

El contendor front lleva dentro el servicio BFF y los estáticos

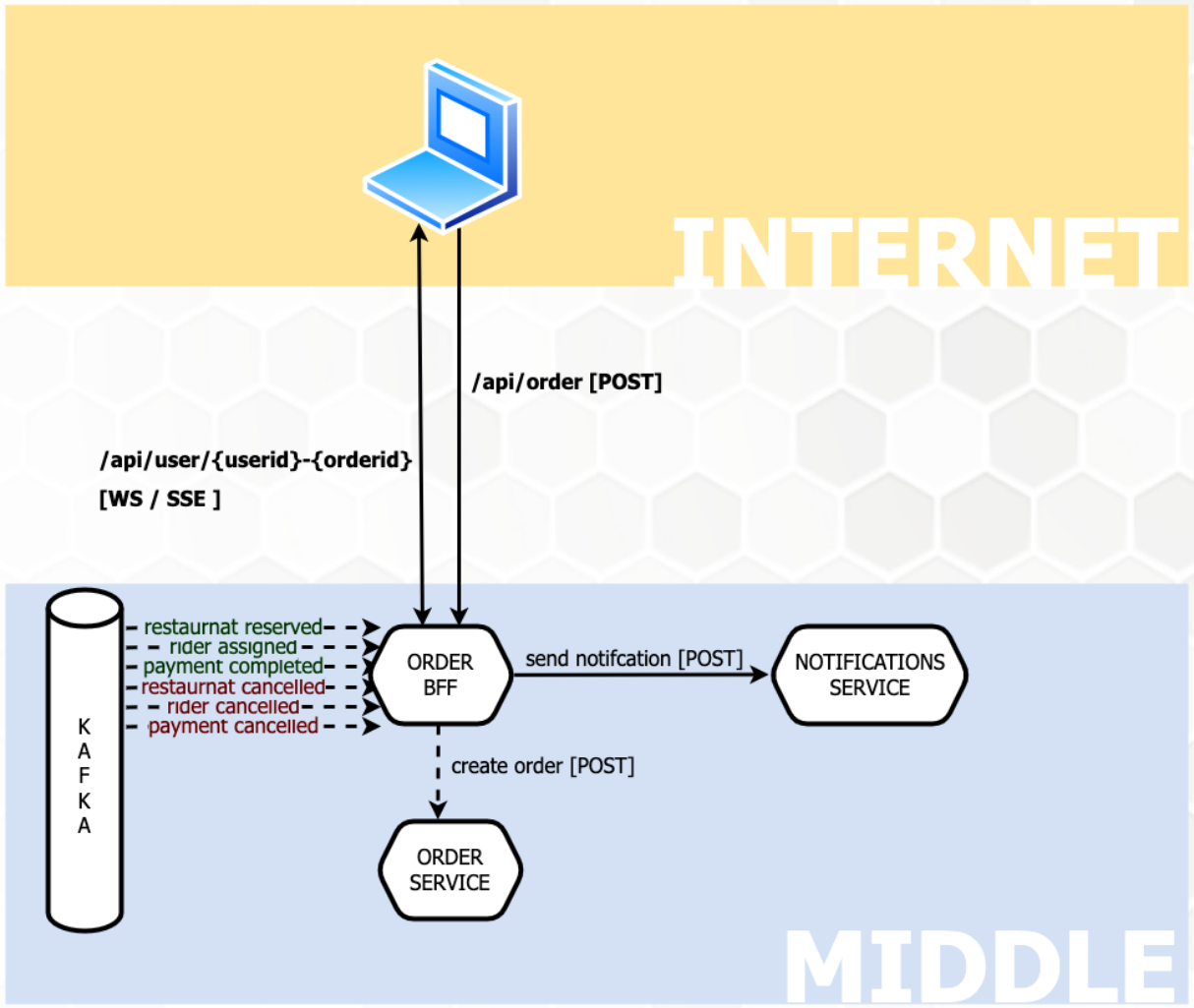
- Los desarrolla el equipo front a sus necesidades
- Agiliza el CI/CD y el testing
- Decide si envía online / offline





Flujo middleware

- único punto de entrada, independiente del consumidor
- servicios no tienen conexiones entre ellos
- Order genera el orderId y audita
- OrderId como correlation id
- Variables de entorno cambiar orden de la saga
- resiliencia y escalabilidad



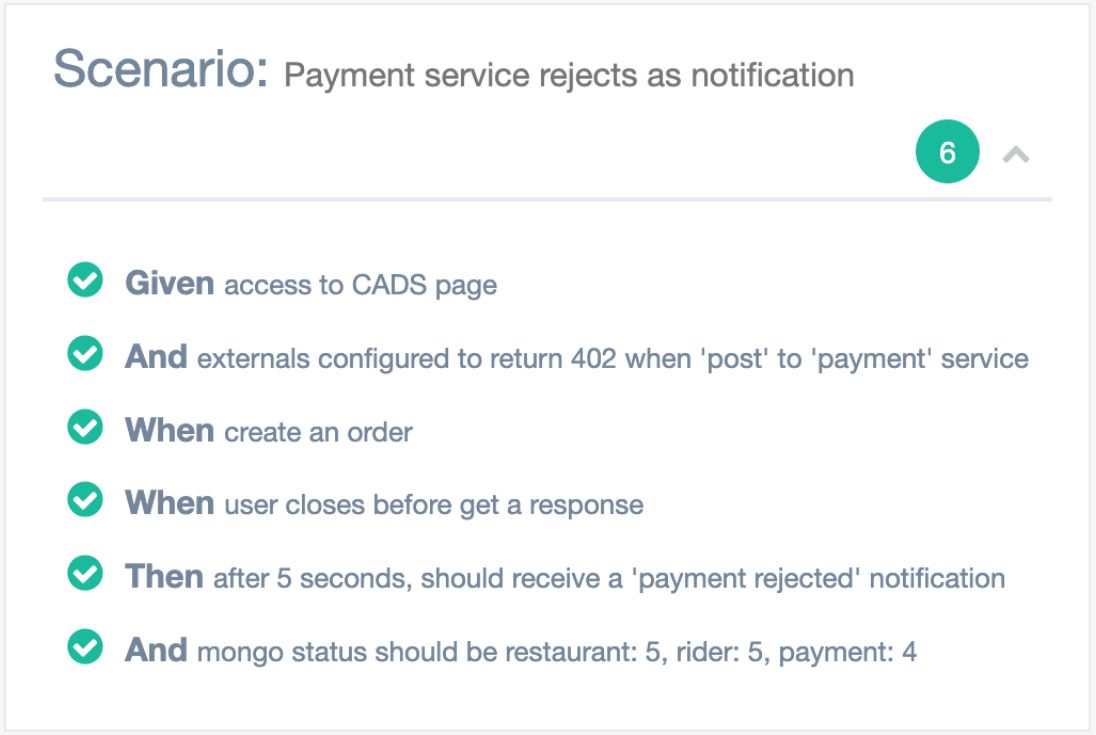
Flujo Frontend

- backend for frontend
- sin bbdd
- reenvía eventos de middle a front
- convierte eventos en notificaciones
- adapta modelos

Web Sockets vs Server Sent Events vs pooling

- En los 3 casos se queda una conexión abierta, tiempos muy similares
- Pooling descartado por dejar 1 hilo y porque a los 30 seg se repite la petición
- Server Sent Events es REST
- Web Sockets permite bidireccionalidad y datos complejos


DEMO TIME!



E2E test

Test E2E en cypress con gherkin.

Cada test configura el api externals: tiempo y response code (banco, restaurante, rider).

Tests con el usuario online y offline, check de notificaciones.

Tests comprueban el rollback en las bbdd.

> reportes mocha y cucumber

E2E test video

Conclusiones

Objetivos conseguidos:

- Saga coreografiada con eventos en kafka
- Servicios idempotentes, resilientes, escalables e independientes.
- El frontend consume actualizaciones sin afectar a los servicios
- Las piezas y la arquitectura son muy simples, y muy mantenible
- El patron BFF esta pervertido pero es fundamental
- Los squads apenas tienen dependencias entre ellos más haya del contrato de los eventos
- SSE gana sobre WC y Pooling, aunque por poco

Otros casos de uso para el BFF

- Actualizar datos cuando llegan a middleware
  - Usuario actualiza sus datos en el momento
  - Evita la sobrecarga del middleware
  - Evita recargas o gestion de cache
  - Desacoplamiento middle - frontend
  -

VENTAJAS

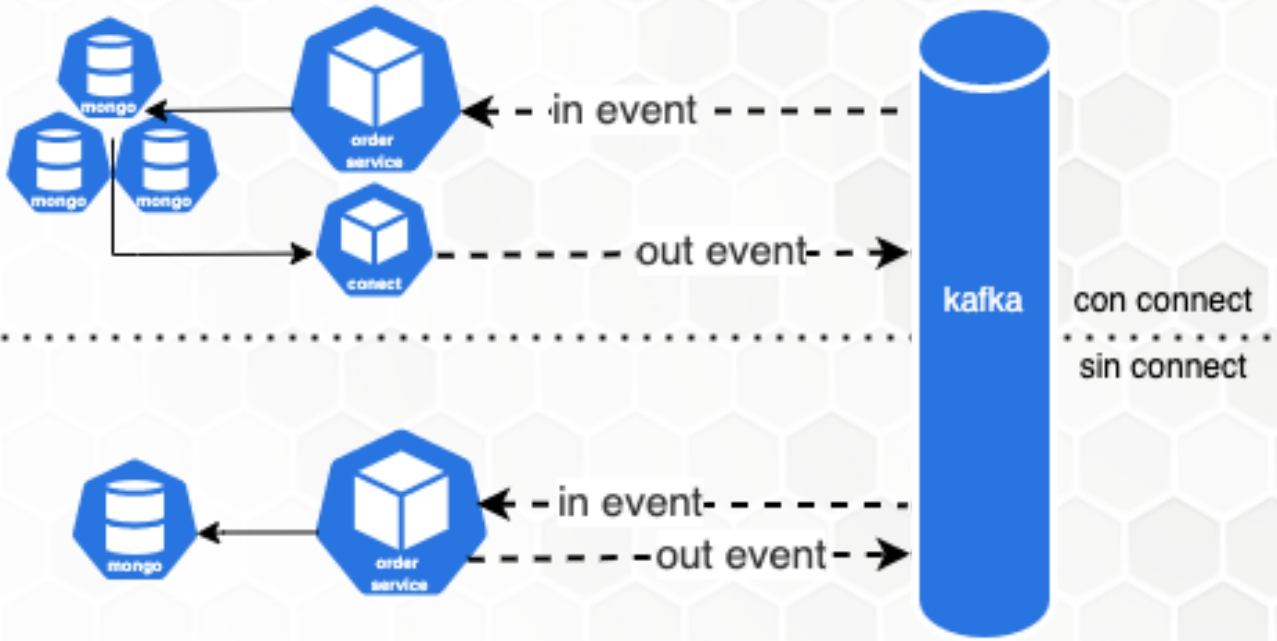
Cada servicio pueder esta en una tecnologia u otra
Podemos cambiar el orden de la saga cambiando los eventos

Consas malas

Performance conexión BFF

Testing
CI/CD

Observabilidad
Seguridad
Conciliaciones
Frontend – UX
Escalabilidad BFF



Kafka Mongo connect

Pros

- envía eventos al persistir en bbdd
- simplifica idempotencia

Cons

- un servico más
- obliga a tener mongo en replica set de al menos 3 instancias

> Finalmente se descarta, no merece la pena

GRACIAS!