#ASLI ENGINEERING

Designing workflows with microservices

SWIPE

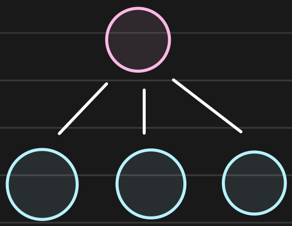
BY

ARPIT BHAYANI

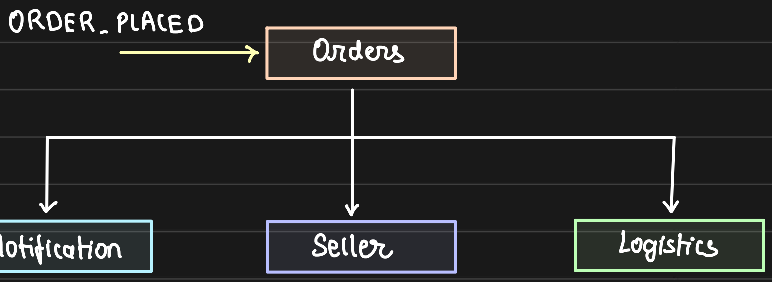
Designing workflows in microservices

Say we are building an e-commerce website, and whenever a user purchases something, we have to

1. Send an email confirmation to the user
2. Notify the seller to keep the shipment ready
3. Find an assign a logistic delivery partner to ship



How do we model and implement this flow ?



Notify the user | Notify the seller | Assign a delivery partner

So, how would the involved 3 services - Notification, Seller, Logistics - get to know about it to take necessary actions ?

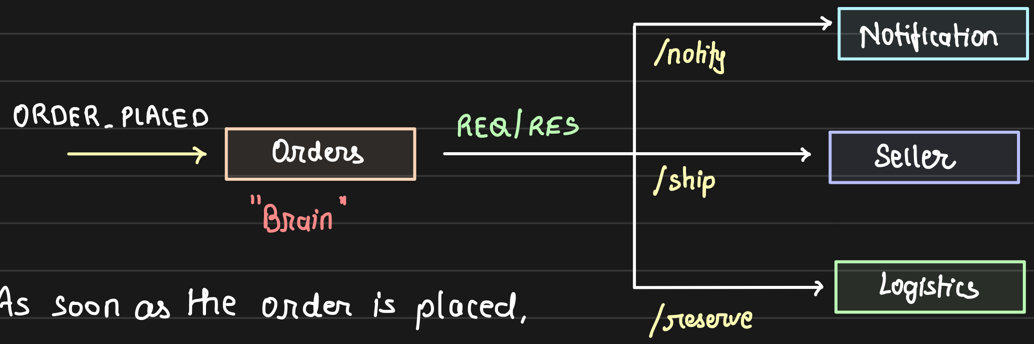
There are two ways to model this

1. Orchestration
2. Choreography

ARPIT BHAYANI

Orchestration [decision logic is centralized]

let there be a single brain that exactly tells others what to do.

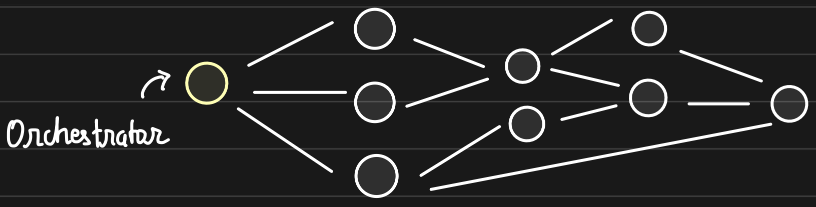


As soon as the order is placed,
the order service invokes the API
of other involved services to do what needs to be done.

* All the 3 calls need not be one after the other

Core Idea: Services are dumb, orders service knows what needs to be done on each involved service, and it thus orchestrates the required actions.

* In some cases, the orchestrator needs to handle, manage, and track a much complex workflow

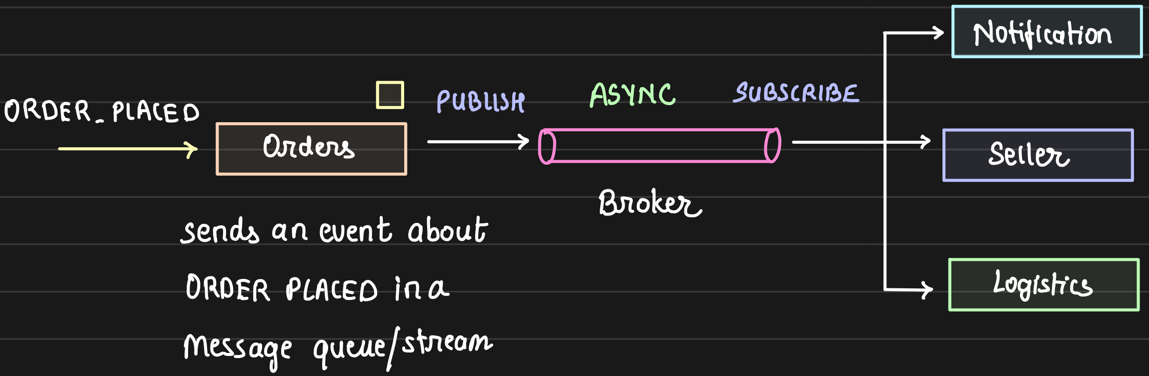


ARPIT BHAYANI

Choreography [decision logic is distributed]

Instead of having a single "brain", let each service be independent to think what needs to be done upon getting to know what happened.

This lays the foundation of Event Driven Architecture



When an order is placed, the orders service emits an event

Notification, Seller, and logistics have subscribed to ORDER_PLACED

Once the services receives the event, they do what they are supposed to

Thus, all the 4 involved services are decoupled

and if want to extend our system to do something more, the new service can simply just subscribe to relevant events and handle them

ARPIT BHAYANI

So, which one should we use, when, and why?

Most modern systems are inclined towards **choreography**

- loosely coupling - the core principle of microservices
- extensible - adding new type of service is simple
- flexible - services are independent to drive their own changes
- robust - workings not affected no matter the number of subscribers

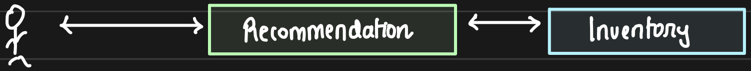
But with choreography approach we need to

- implement complex observability and track what's happening

Note: although a lot of people adopt & prefer choreography, this does NOT make orchestration bad

Because orchestration is REQ/RES type flow, we can use it at

- services need to be invoked transactionally - distributed transactions
- sending OTP for logging in happens synchronously
- rendering details in recommendation requires a sync communication



ARPIT BHAYANI