



#ASLI ENGINEERING

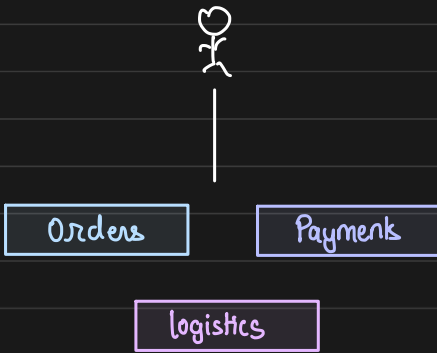
API Composition Pattern In Microservices



BY

ARPIT BHAYANI

API Composition Pattern in Microservices

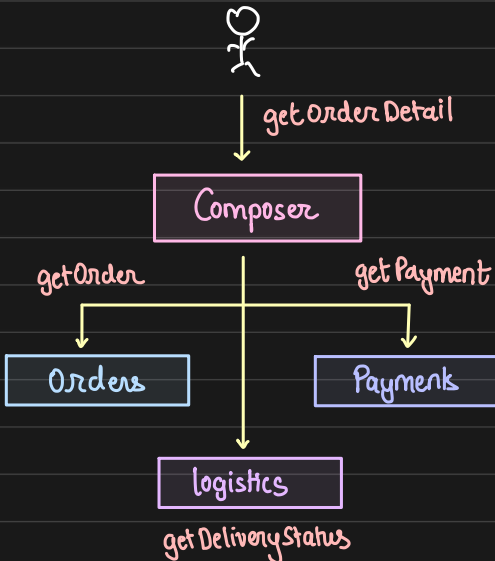


Say we have 3 services, Orders, Payments, and Logistics. A user wants to fetch order detail, how would we get the required data?

Classic problem: Querying Microservices

To query microservices, a common pattern in use is **API Composition**

In the API composition pattern, we have a "composer" that



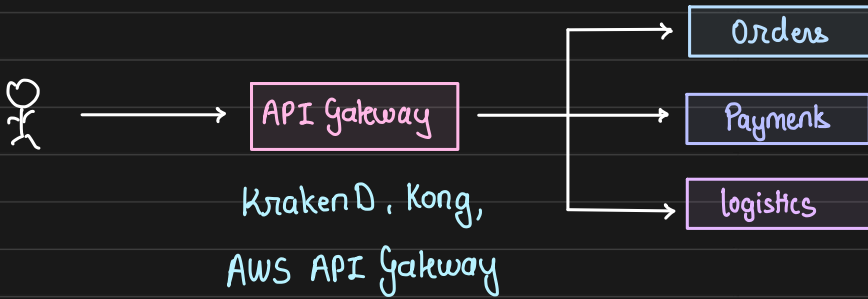
- takes the user query
- identifies the services to talk to
- gets the responses from them
- combines the result
- returns the response

So, how do we implement it?

Do we need to write Composer from scratch? **No!**

API Gateway is a classic example of a "composer"

So, we put an API Gateway in front of our microservices and configure the rules.



The calls to the individual microservices can happen

- Sequentially → Takes longer time to compute
- Parallel → Requires more machine resources

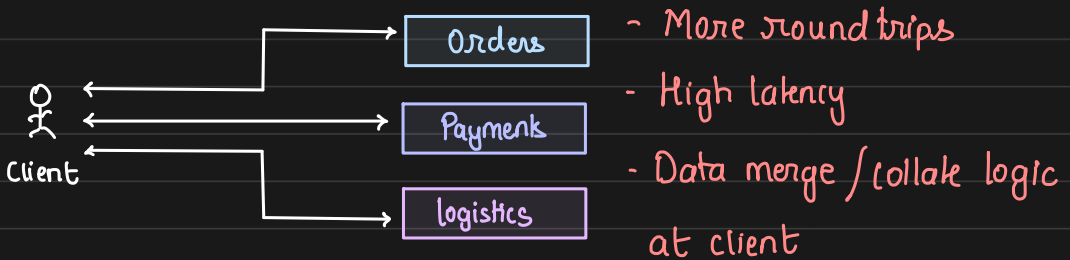
The implementation of the consumer depends on

- programming lang
- usecase at hand

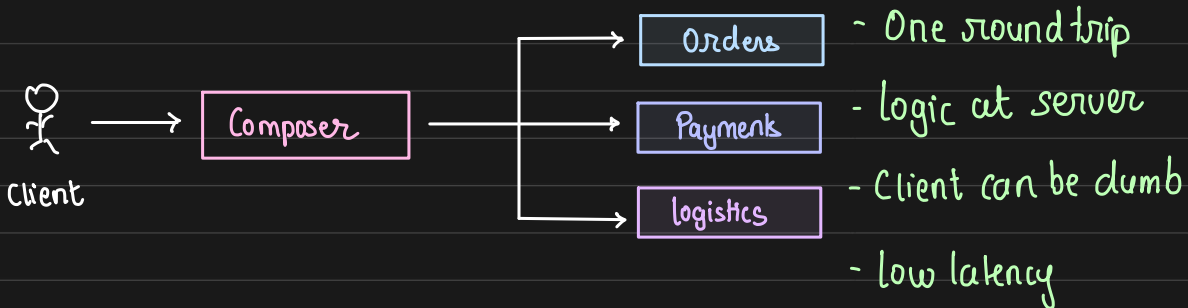
Improving end user experience

The impact of API composition is not just limited to microservices it has huge role to play in giving a great UX

Imagine this,



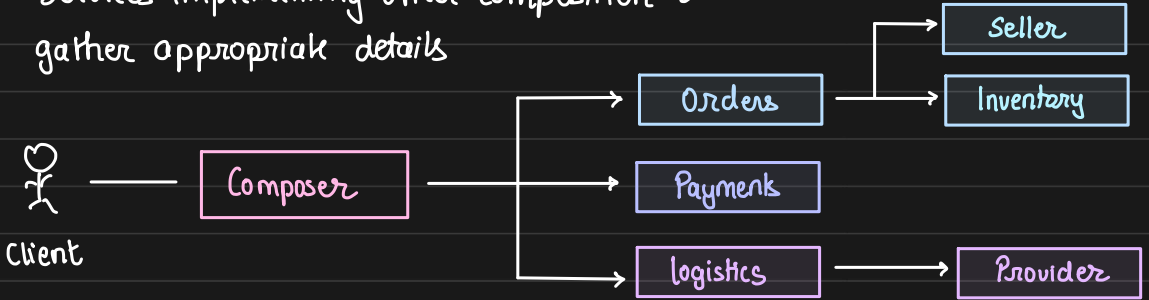
Hence, to make experience better, we should compose at server
+ Better use of user's data



End user thus gets pretty solid experience
with just one data round trip.

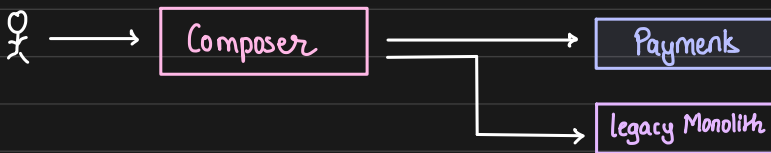
Branch Composition

Branch composition is Multi-level API composition
Services implementing other composition to
gather appropriate details



Advantages of API Composition

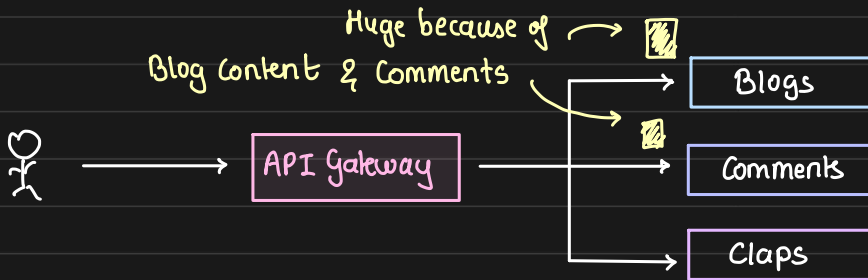
- Simple to implement
- User has a single point to interact
- Hides the implementation detail and complexities ↗ First line of defence
- Security and limiting can be implemented at composer
- Can cover "bad" design decisions and wrap them with a new interface
- Hides legacy system and we can replace it gradually



Disadvantages of API Composition

- What if we fetch **large data** from microservices

↳ Joining them on API Gateway would be **inefficient**



- Overall **availability** is challenged when the number of services composer interacts with increases
- Having **transactional data consistency** is very difficult
eg: All or none across services Distributed Transaction
- composer needs to be **managed and maintained**
- composer may become a **bottleneck**