#ASLI ENGINEERING

How to scope a microservice

SWIPE

BY

ARPIT BHAYANI

How to scope a microservice

It is always exciting to creak a new microservice, there are so many things to look forward to like

- fresh codebase
- new tech stack
- clean CICD setup

Plus, you would think to write this new service the best way possible & not repeat past mistakes.

But, designing and fencing the service well is extremely important..

Too few of them... and you still have multiple teams collaborating on same codebase & breaking each other's flow

Too many of them... and you have a mess of inter-connected and inter-dependent sub-systems

A good micro-service is always designed by keeping the following two concepts in check

Loose Coupling

↓

change in one service should not require change in another

High Cohesion

↓

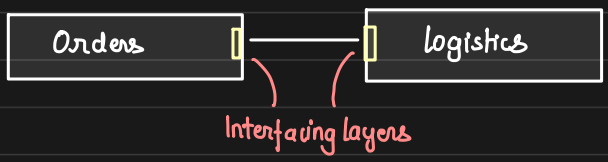
Related functionalities should be part of the same microservice

1. Loose Coupling

- change in one service should NOT require change in other

↑

core ideology behind microservices



So long as the interfacing layer and API contracts remain the same, chang in the orders service should be transparent to the logistics.

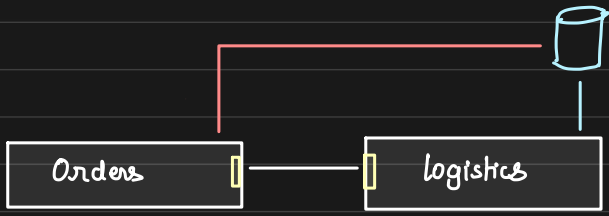
But, how to achieve loose coupling?

A service should know as little as it needs about other services

eg: Public APIs	YES	Rate limits	YES
Authentication	YES	Communication protocol	YES
~~Database used~~	No		

If a service knows too much about other service then it might tightly couple them. eg.

Orders service directly talking to DB of logistics



2 High Cohesion

- related behaviour sit together
- unrelated behaviour sit separately

CORE IDEA: Service should operate independently

If your orders microservice also holds and is responsible for customer data, then deploying changes will require consent of both the involved teams and their respective consumers.

If your unrelated components are "sitting" together
then it might need you to deploy everything connected

Two microservices sharing the same codebase

eg: when transitioning from monolith to microservices
you tend to reuse the monolith codebase & fork out services

Now, when some changes in monolith codebase,
you will need to re-deploy other services sharing the codebase.

Deploying multiple services at once is risky

Hence, fence the responsibilities & make them
as loosely coupled as possible.