Spring Microservices

Coupling in Java

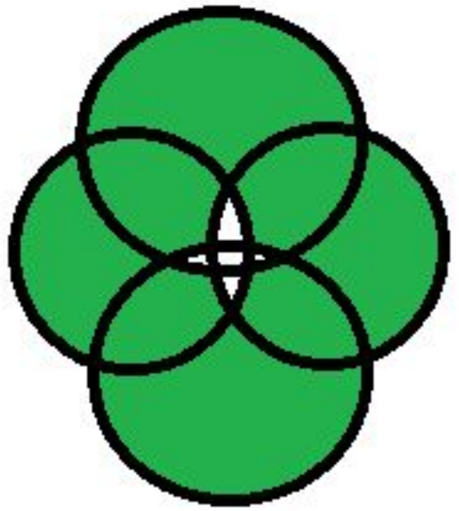
- Tight Coupling
- Loose Coupling

Tight Coupling

Tight coupling is when a group of classes are highly dependent on one another. This scenario arises when a class assumes too many responsibilities, or when one concern is spread over many classes rather than having its own class.

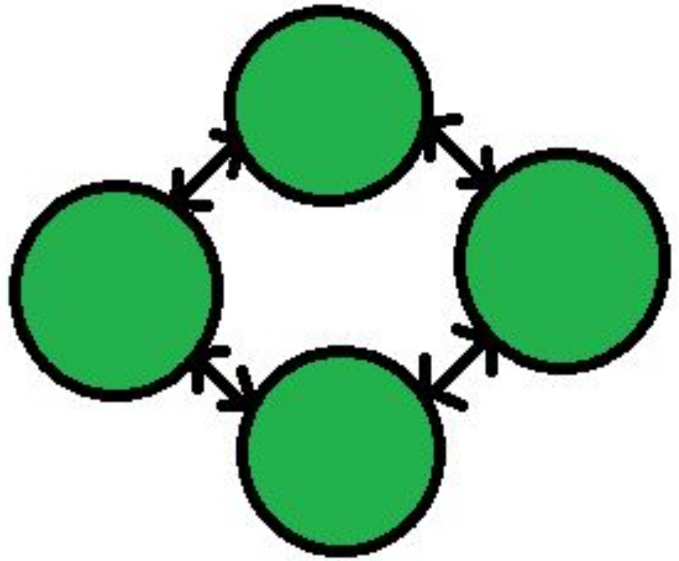
Loose Coupling

Loose coupling is an approach to interconnecting the components in a system or network so that those components, also called elements, depend on each other to the least extent practicable. Coupling refers to the degree of direct knowledge that one element has of another.



Tight coupling:

1. More Interdependency
2. More coordination
3. More information flow



Loose coupling:

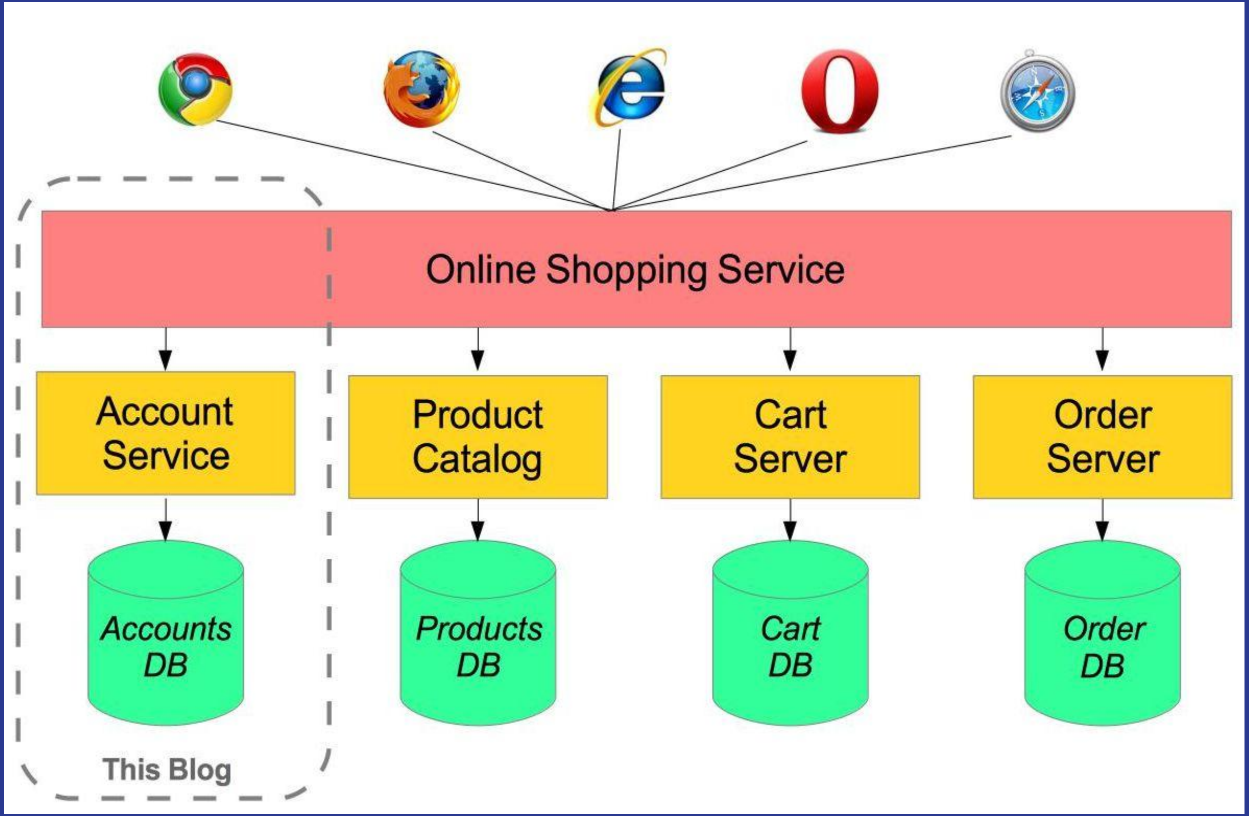
1. Less Interdependency
2. Less coordination
3. Less information flow

Microservices

Microservices allow large systems to be built up from a number of collaborating components. It does at the process level what Spring has always done at the component level: loosely coupled processes instead of loosely coupled components.

A microservice is a stand-alone process that handles a well-defined requirement.

Imagine an online shop with separate microservices for user-accounts, product-catalog order-processing and shopping carts



How we create Microservice

We will be using Spring Boot to create microservices. So,

Number of microservices = Number of Spring Boot applications.

Note: Each Microservices will be running on different port. It is also possible to run different micro services on different machines/servers.

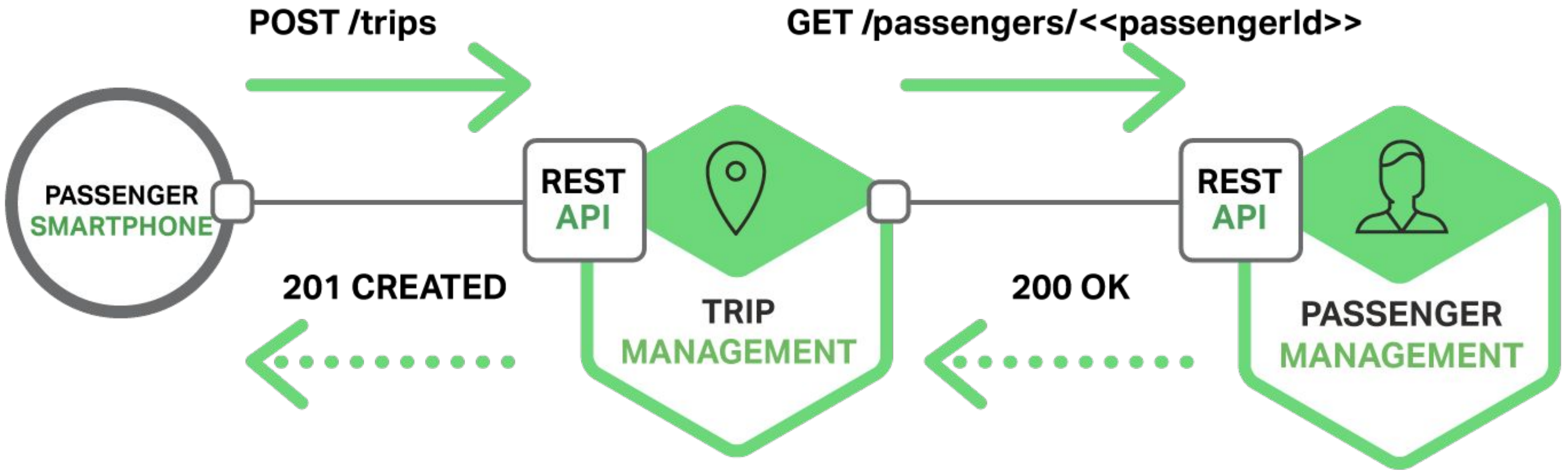
How Microservices talk to each other.

There are two ways microservices can talk to each other.

1. Synchronous
2. Asynchronous

Synchronous Communication

- Synchronous communication is a style of communication where the caller waits until a response is available.
- Each microservice provides a RESTFul interface over HTTP.
- Other microservice will use RestTemplate to call RESTFul API/Interface exposed by targeted microservice.

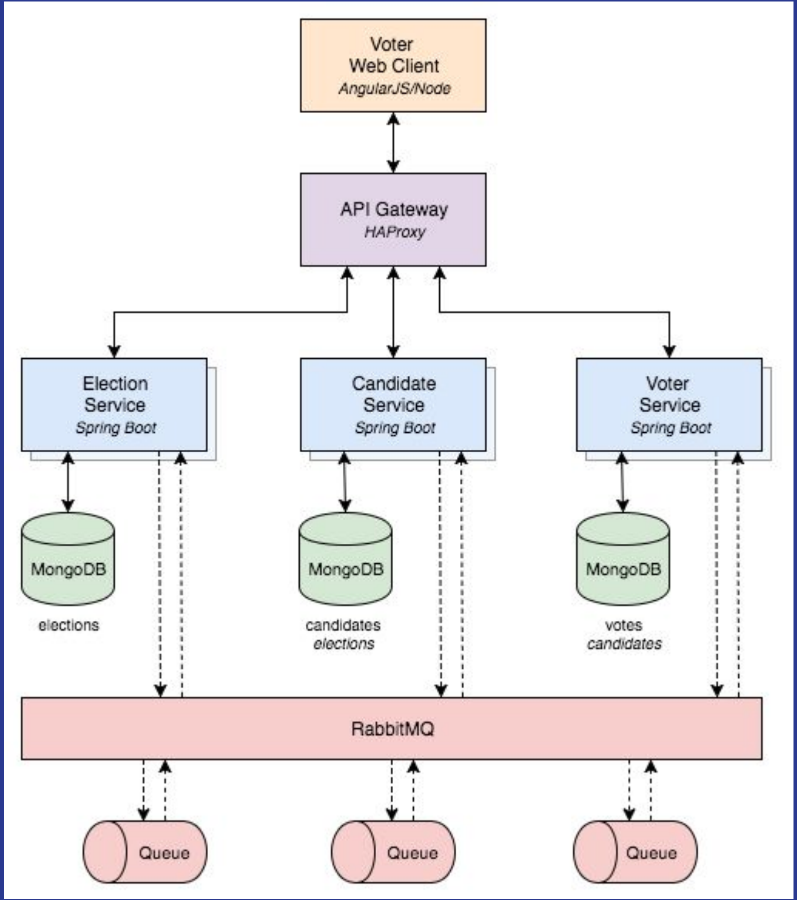

POST /trips
GET /passengers/<<passengerId>>
PASSENGER
SMARTPHONE
REST
API
TRIP
MANAGEMENT
REST
API
PASSENGER
MANAGEMENT
201 CREATED
200 OK

Synchronous Communication - Drawbacks

- This mode of communication works well when the response arrives almost immediately. However, it puts restrictions on the microservices that are involved. In order that Microservice1 be available, Microservice2 also needs to be available.
- In certain scenarios, synchronous communication might not even be user friendly.
- Let’s say Microservice2 is down. In that case, the user who submitted the request needs to be intimated about it. You don’t always want to do that. In such situations, asynchronous communication provides a better alternative.

ASynchronous Communication

ASynchronous Communication Continue...



ASynchronous Communication Continue...

- In a system that involves asynchronous messaging, the server does not need to be up and running, all the time. Messages that are put into the message queue can be processed in batches, at a later time.

- Great thing is as long as we fix the errors and reprocess the message, the users of services that initiated the requests with errors do not even need to know about them.