Agenda.

- → Intro to Project & Project Doc.
- → Intro to Spring framework.
- → Spring (vs) SpringBoot
- → Dependency Injection
- → Inversion of Control.

⇒ Backend of an Ecommerce app^.

⇒ Amazon

- → Authentication
- → Payments
- → Products.
- → Orders
- → Tracking
- → Logistics
- → Cart
- → —
- —
- —

Monolithic

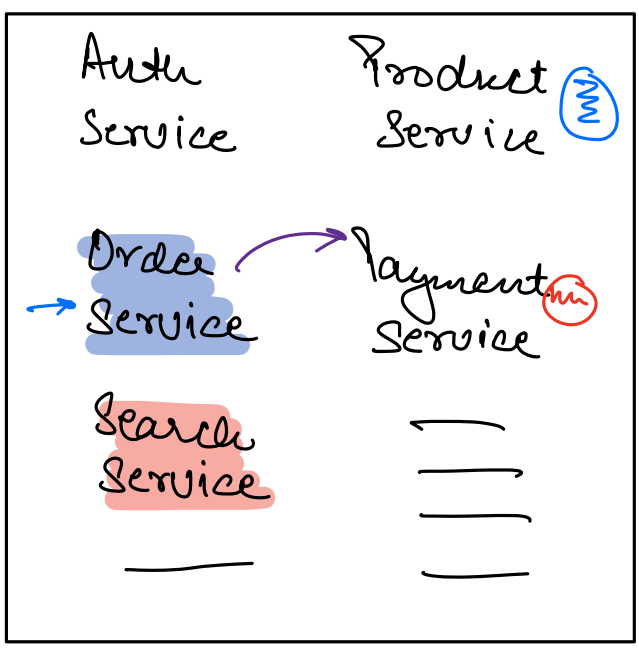
Microservices.

Monolithic architecture :

All the functionalities / services are part of a single application.

- → Single codebase.
- → Huge
- → Slow compilation & high application startup time.
- → No tech stack flexibility.
- → Onboarding new team members will be difficult.
- → No selective scaling is possible.
- → Deployment time is High.

Single → Monolithic App^n



→ One small issue in any service can make the entire service down.

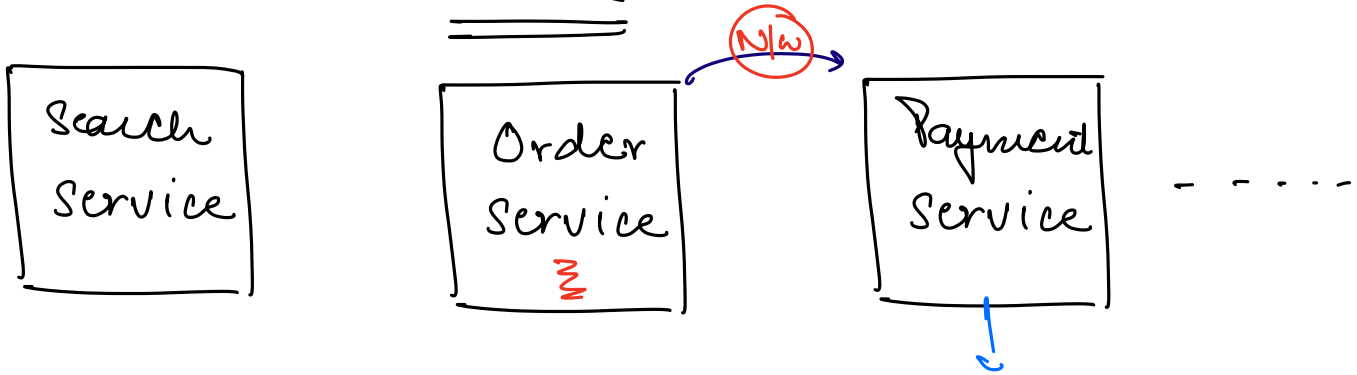
⇒ Cascading failures.

Traffic on Search Service >>7

Traffic on Order Service

⇒ In monolithic architecture, two services will can communicate with each other via a simple function call.

⇒ Divide this single Codebase into small individual codebases.:



⇒ MicroServices.

Codebase (App) / Project.

⇒ Individually deploy services.

⇒ No cascading failures.

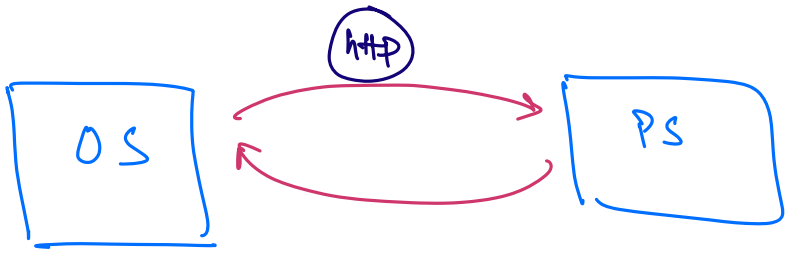
⇒ Tech Stack flexibility.

⇒ Selective Scaling

⇒ Difficult to manage so many services.

⇒ Two services can communicate with each other via an API call.

↳ N/W ⇒ Slower.



→ ProductService

↳ Product Catalog Service

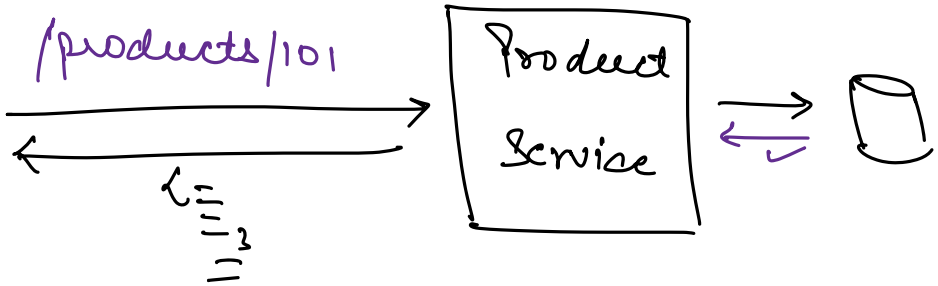
→ PaymentService

→ AuthService

→ EmailService

→ Service Discovery.

⇒ JAVA + SpringBoot.



⇒ CRUD operations.

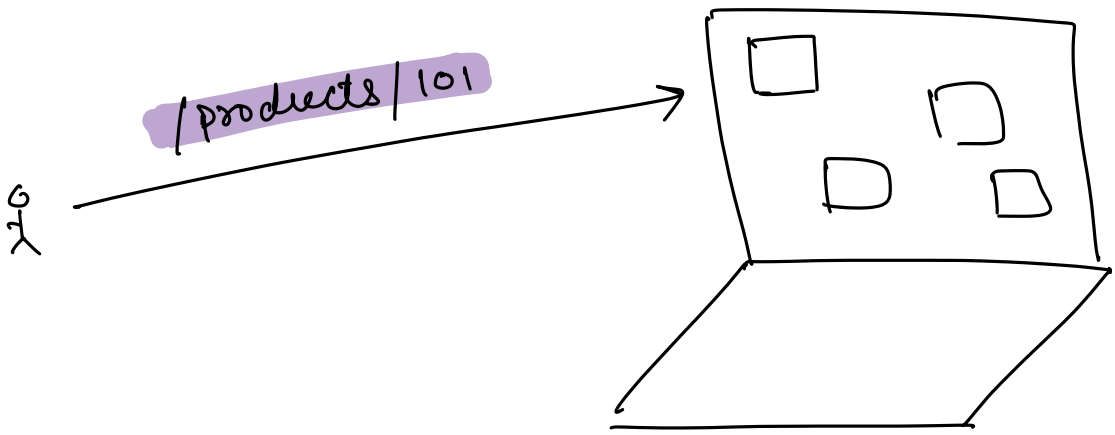
⇒ framework.

- Write APIs
- Connecting with DB
- —
- —
- —
- —

⇒ Java + Spring framework.

⇒ framework provides ready to use implementation for the most common functionalities that we need in our Applications.

=> As a Software Engineer, We need to focus more on writing the core business logic



=> Don't Reinvent the wheel.

=> framework : functionalities provided in a ready to use way to make commonly occuring problems easy to do.

=> Spring framework.

Set of projects that allows easy creation of enterprise level JAVA application.

↓

Production / industry.

Spring

Core framework

[Core set of functionalities]

+

Add Ons

(Authentication | Kafka | DB | Redis)

- - - .

Dependency Injection.

```
Class A {
    B b;
}
```

A has a dependency on B.

I)

```
Class A {
    B b = new B();
}
```

II) Create an Object of b outside the class and pass it inside A.

```
A {
    B b;

    A(B b) {
        this.b = b;
    }
}
```

```
B b = new B();
A a = new A(b);
```

Constructor Injection

OR

```
A a = new A();
a.setB(b);
```

Setter Injection

#

```
UserService {
    DB db = new DB();
    ========
}
```

```
ProductService {
    DB db = new DB();
    ========
}
```

```
OrderService {
    DB db = new DB();
    ========
}
```

⇒ Creating so many DB objects is a complete waste of resources.

Dependency Injection allows us to Reuse Objects

⇒ Create one DB object outside the class and let it to different classes via Constructor or setter method.

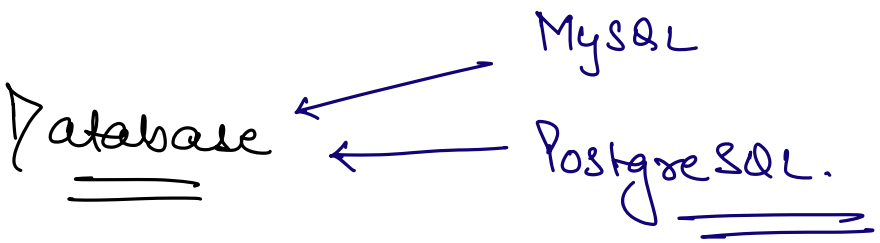
```
UserService {
    DB db;
    ---
    ---
    ---
}
```

```
ProductService {
    DB db;
    ---
    ---
    ---
}
```

```
OrderService {
    DB db;
    ---
    ---
    ---
}
```

⇒ DB db = new DB()

```
UserService us = new UserService(db);
ProductService ps = new ProductService(db);
---
---
---
```




```
UserService {
    DB db = new MySQL();
    ---
    ---
    ---
    ---
}
```

Tightly Coupled.

```
ProductService {
    DB db = new MySQL();
    ---
    ---
    ---
    ---
}
```

⇒ If we want to migrate from MySQL to PostgreSQL.

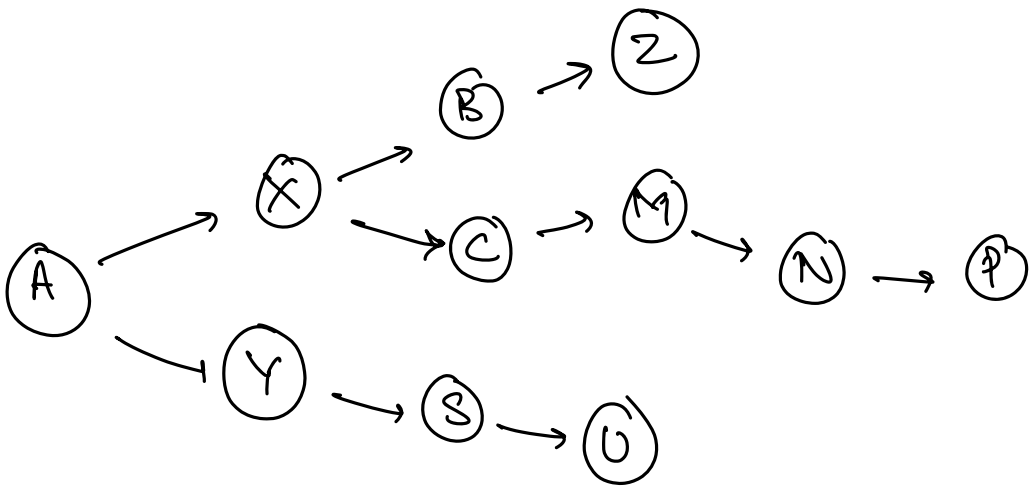
⇒ In the above implementation, We'll have to go to all the classes and change the object from MySQL to PostgreSQL.

PostgreSQL()

```
DB db = new MySQL();
UserService us = new UserService(db);
ProductService ps = new ProductService(db);
```

⇒ loosely coupled.

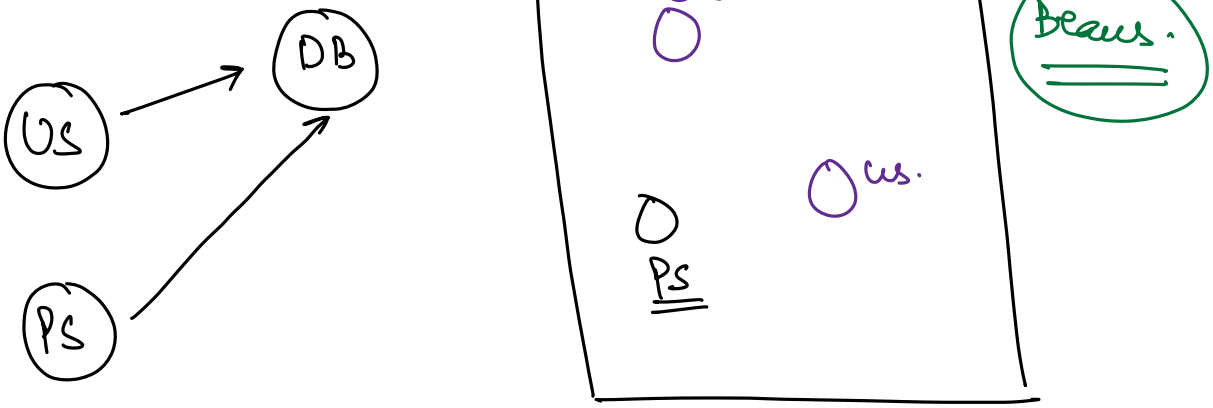
Amazon's Codebase.



IOC (Inversion of Control)

↳ Spring fw provides a way to resolve dependencies automatically.

⇒ Spring now takes the responsibility of object creation in the correct order.



Application Context / Spring Container / IOC Container

Bean: Special Objects that spring stores inside Application Context.

for every addon, we had to create a new Configuration file (xml) in the appn.

↳ Earlier

SpringBoot.

↳ No need to add configuration file for each addon.

→ SpringBoot automatically creates configuration for the dependencies.

⇒ SpringBoot provides us an opinionated view of using a particular dependency in the right way but also it provides us a way to configure it.