Agenda:-

1) Project Overview
2) Microservices (vs) Monolithic
3) Intro to Spring framework.
4) Dependency Injection & IOC.
5) SpringBoot
6) Build our first API.

=> Backend arch. of an Ecommerce Appl^n.

UserService
ProductService
SearchService
PaymentService
LogisticsService
OrderService
CartService
NotificationService

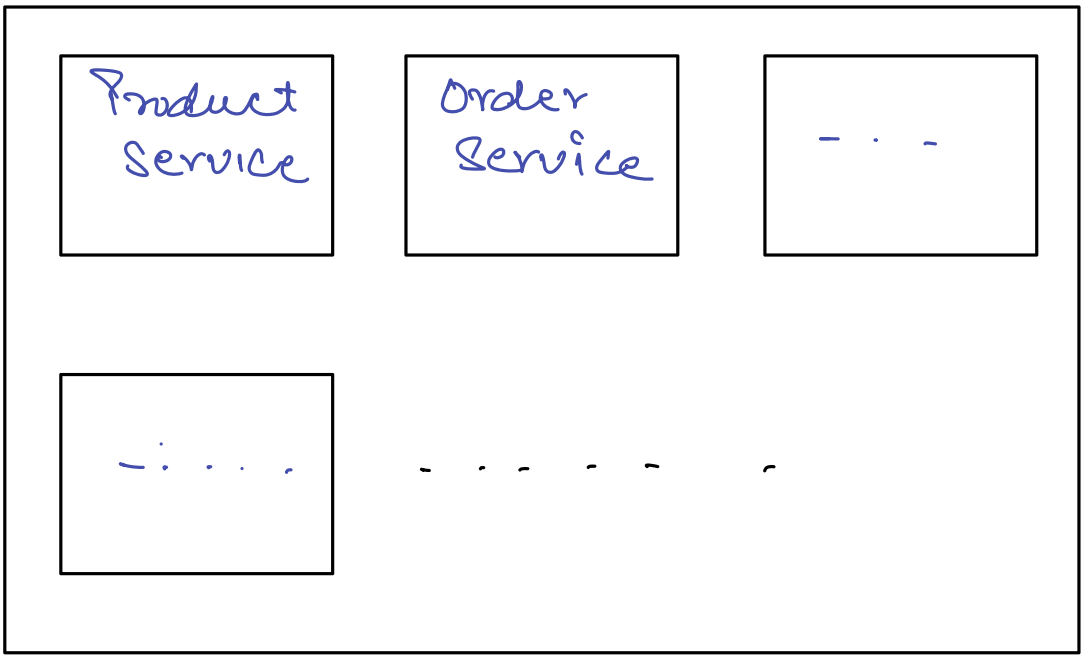
Monolithic Architecture

=> All the services are part of a single project.

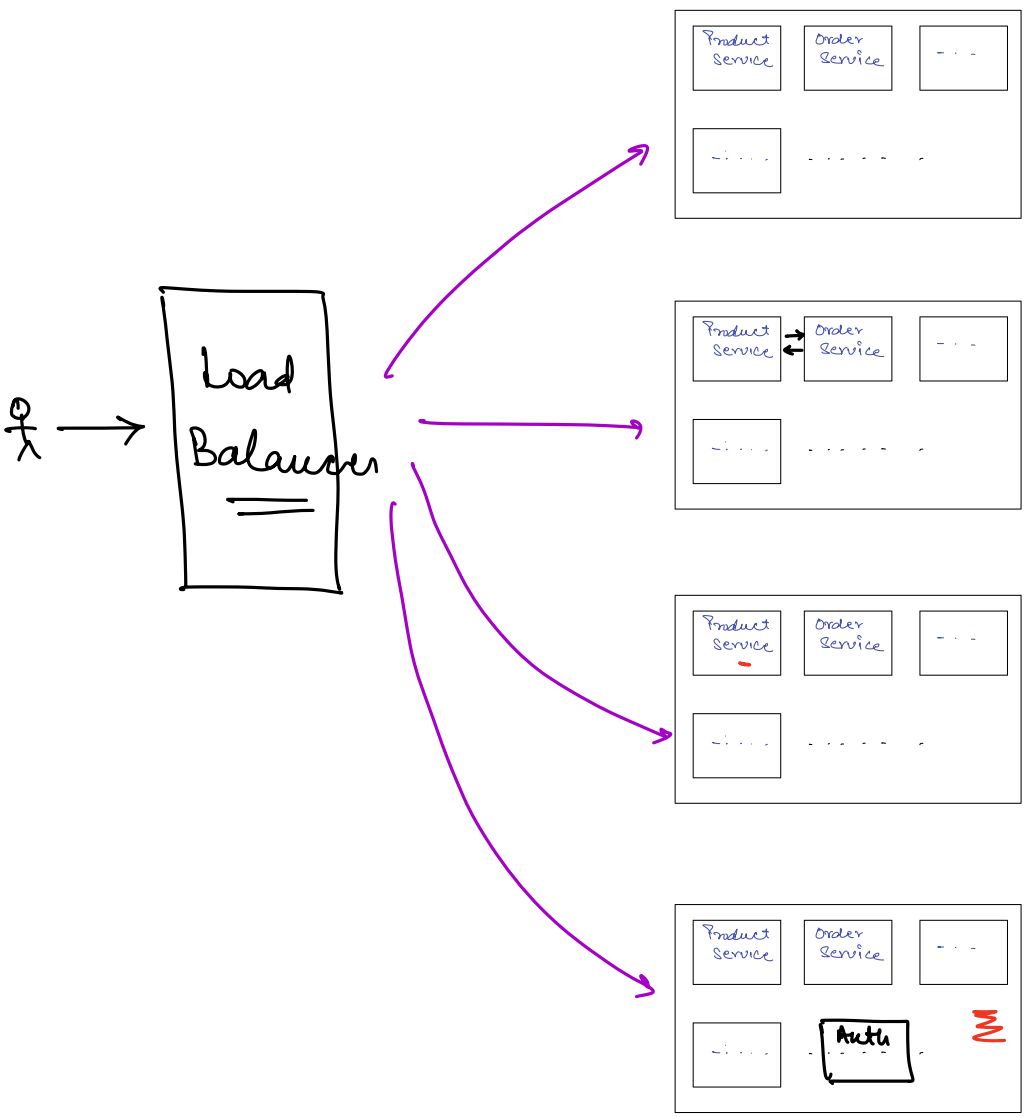
MicroService

=> Different project for each service.

Amazon.



Monolith.



Pros

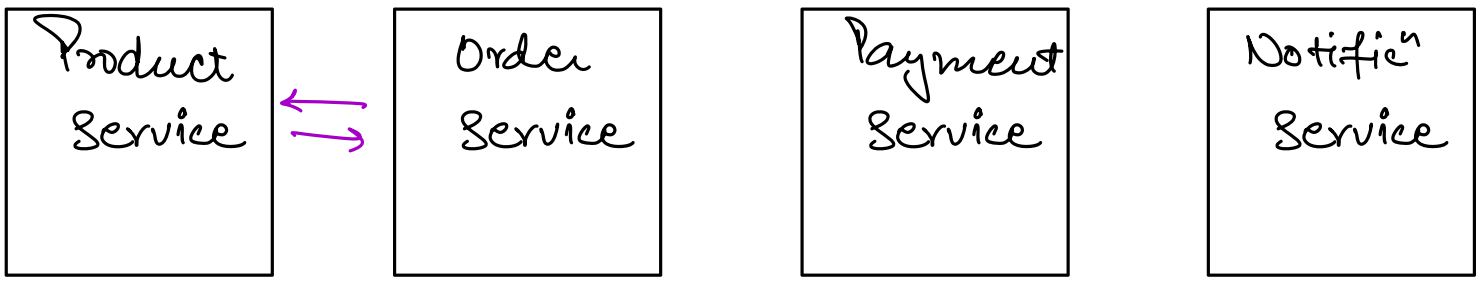
1) Single deployment.
2) No n/w latency.
3) Easy debugging

Cons.

1) More deployment time.
2) No tech stack flexibility
3) No selective scaling.
4) A small bug can get the entire app^n down.

Traffic of SearchService >>> Payment Service

MicroServices.



...

Pros.

1) Selective Scaling.
2) faster deployment
3) Techstack flexibility.

Cons.

1) Difficult Debugging
2) N/w call b/w microservices Communication.

Microservices.

ProductService

UserService

Auth

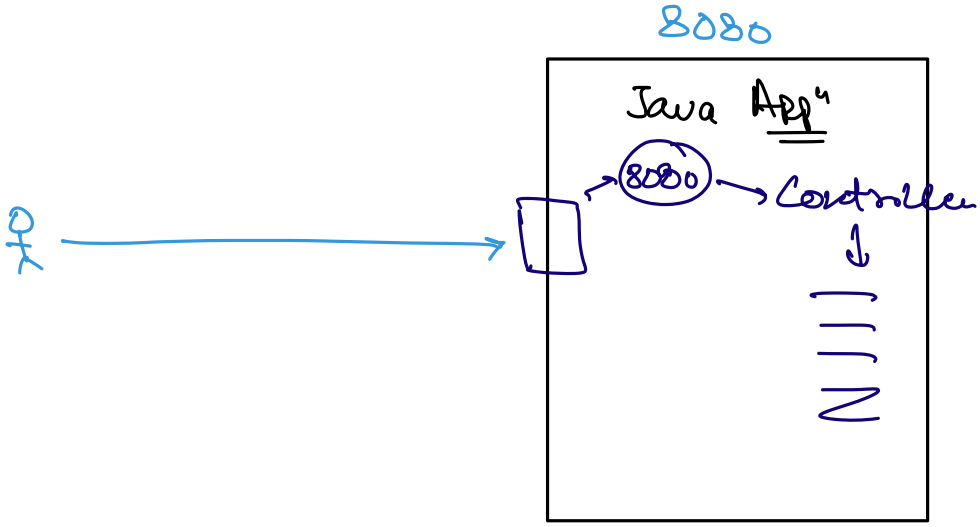
Notific^n Service

API GW.

Frameworks.

↳ Provides us ready to use implementation of the most common thing that we generally do as a Software Engineer.

⇒ We should mostly spend time on implementing the business logic.



Frameworks provides ready to use functionalities to implement most common things :-

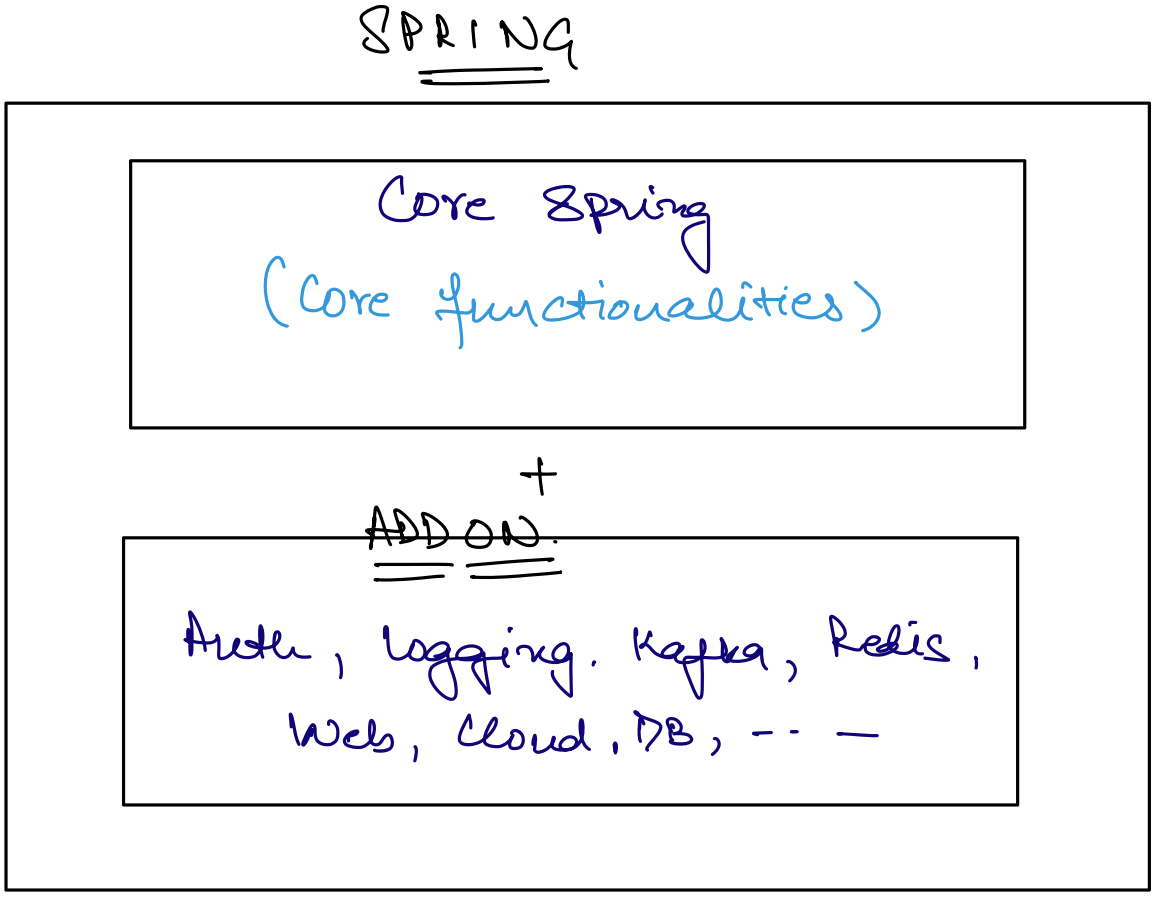
1) Creating API's
2) Connecting to DB
3) Auth
4) Logging

⇒ Spring framework + Java.

⇒ Python + Django. ⇒ Recorded Videos.

Spring framework.

↳ Set of Projects that allows creation of Production level java application by providing ready to use functionalities.



Dependency Injection.

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

⇒ A is depending on B.

1) ✗

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

Creating the dependency within the class.

2) ✓

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

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

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

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

⇒ Dependency Injection.

#

1

UserService {

DB db = new DB()

=

}

SearchService {

DB db = new DB()

=

}

ProdService {

DB db = new DB()

=

}

=> We can Reuse the DB instance

main()

DB db = new DB()

US us = new US(db);

PS ps = new PS(db);

}

(2)

UserService {

DB db = new MySQL()

PSQL()

}

SearchService {

DB db = new MySQL()

PSQL()

}

ProdService {

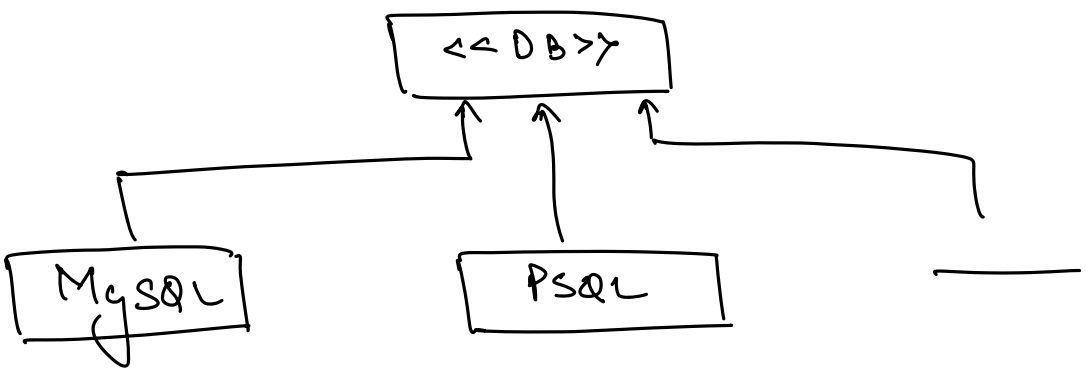
DB db = new MySQL()

PSQL()

}

⇒ Tightly coupled.

Migrate from MySQL to PSQL.



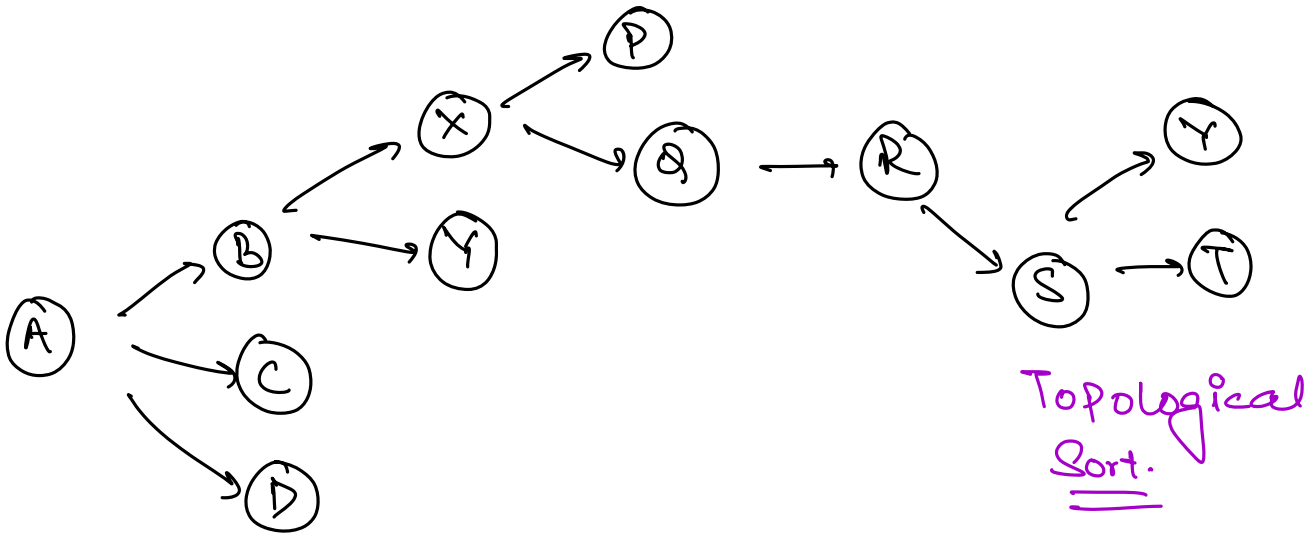
⇒ If we want to migrate to another DB then we'll have to go & change at every place.

```
main() {
    DB db = new MySQL();
    VS vs = new VS(db);
    PS ps = new PS(db);
}
```

PSQL()

Loosely coupled

3



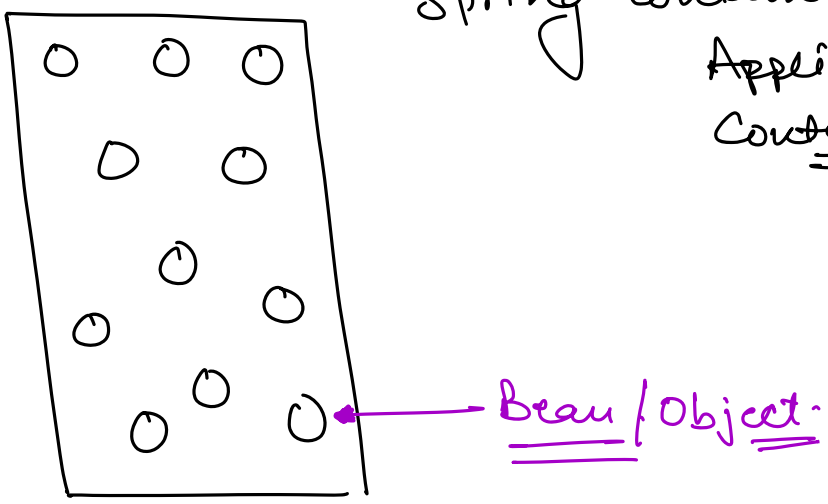
Topological Sort.

Spring : Dependency Injection.

Inversion Of Control

framework does the dependency Injection on our Behalf.

⇒



Bean ≡ Object

↳ Objects managed by Spring and used automatically whenever.

⇒ An object that spring creates, manages & uses whenever required is called as Bean.

1) Start the App"
2) Spring creates all the beans.
3) Spring stores all the beans inside a container / Application context.

⇒ Earlier if we want to use any addon with Spring, lot of configurations (XML) were required.

⇒ Spring Boot: Package on top Spring framework that allows easy usage of Addons.

→ framework based on Spring that provides us easy usage of Addons.