SpringBoot MicroServices

Made By:

Bhawish Kumar

With ❤️ For CS

RAD → Rapid Application Development

Stages

① Business Modelling
↳ Product to be developed

② Data Modelling
↳ Relations between data objects are established.

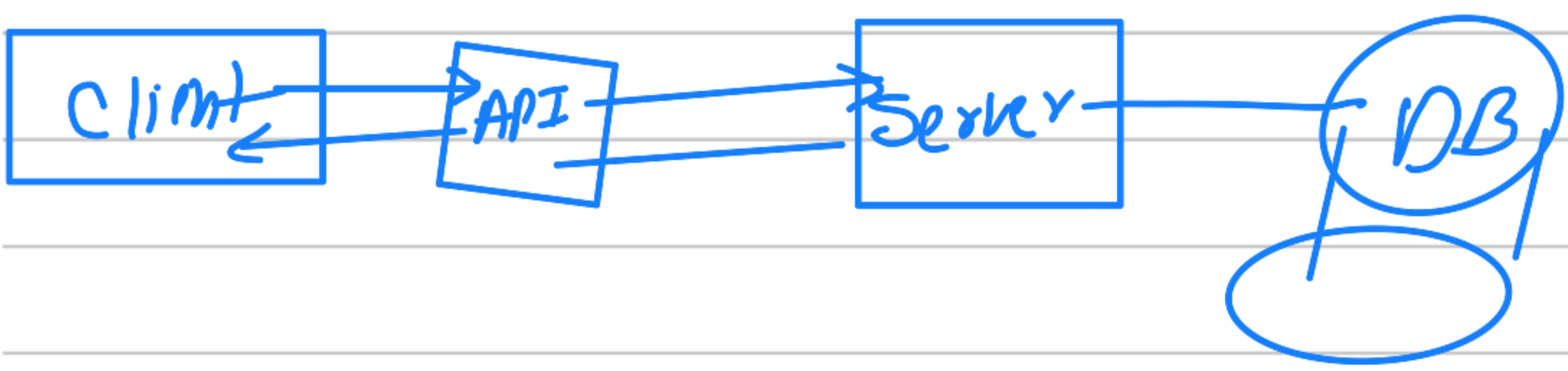
③ Process Modelling
↳ Adding, Deleting, Retrevng, Modifying data objects.

④ Application Actual Product → Prototypy.

⑤ Testing & Turnover :-
Product is designed, tested & updating.

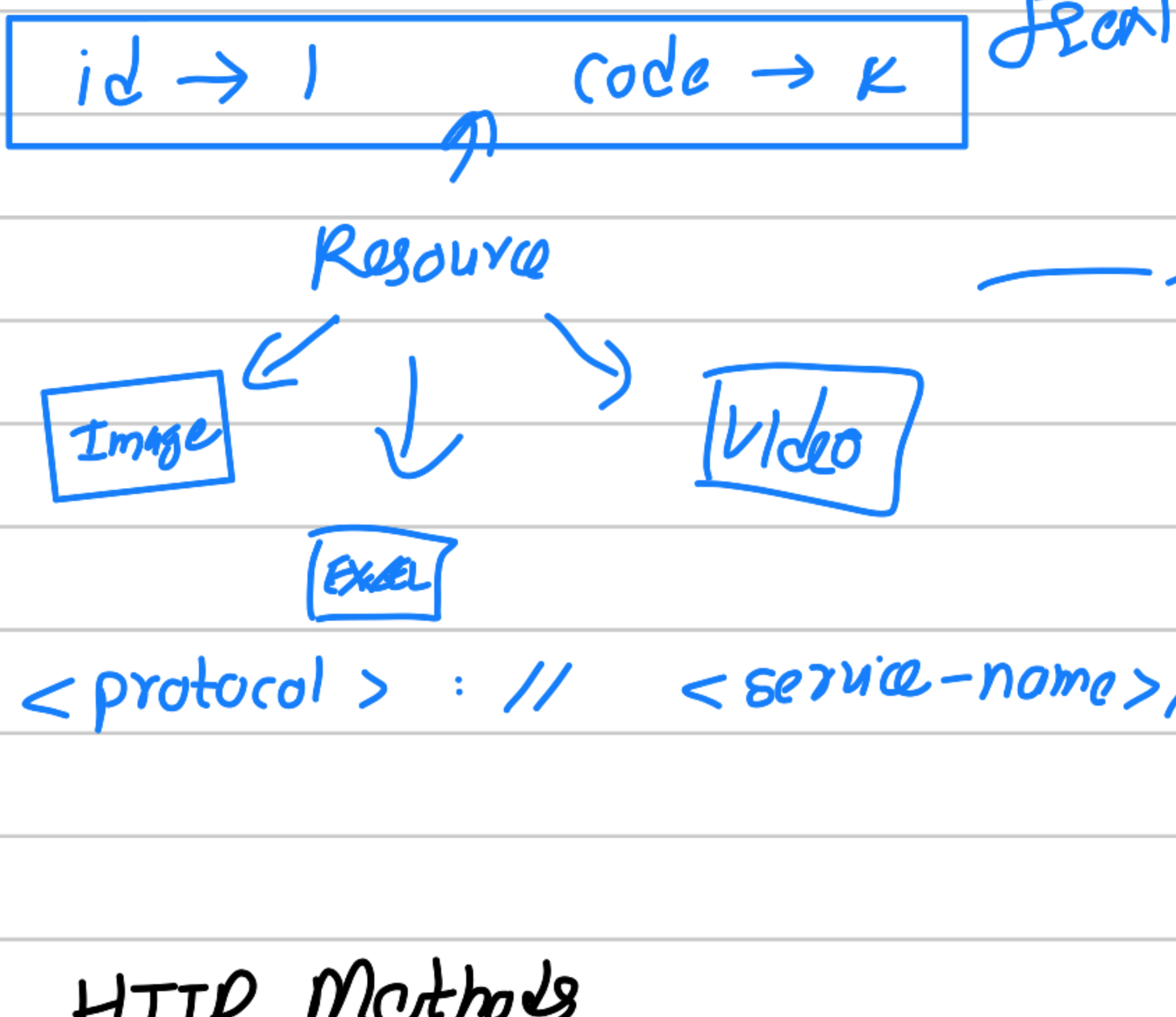
REST → RE presentational State Transfer

RESTFUL Web services follow REST concept (stateless client server architecture)



REST Resource

* every content in REST Architecture is considered a resource.



<protocol> : // <service-name>/ <Resource Type>/
<Resource ID>

HTTP Methods

C ⟶ Create a Resource
R ⟶ Get a Resource
U ⟶ Put a Resource (update)
D ⟶ Delete a Resource.

Statelessness → Client will always send its header
Server does not remember client
Client pases context → Server.

SQL vs No SQL

① Read vs Write

better performance.
no dependency
Read Intensive → No SQL ← No joins
Write Intensive → SQL

② Unstructured vs Structured

↓ ↓

No SQL SQL

↓

Concurrency

↳ Hotel Booking

↳ Many people book at same time. Coordination required.

③ Free vs Paid

↓ ↓

No SQL SQL

Monolithic Application

↳ Single unit

Client Side → Next.js / Angular

Server Side → Spring MVC

Database → MySQL or PostgreSQL.

* Codebase gets larger.
* Hard To manage.
* Hard to add features.

* CIC (continuous Deployment) is hard.

What are microservices :-

*Microservices architecture breaks it down to independent standalone small applications, each serving one particular requirement -

* FOOD DELIVERY
  - → Restaurant Listing Service
  - → Food catalogue service
  - → Order Service
  - → User service

entire functionality is split in independent deployable modules communicate using APIs.

*All services are independent of each other.
* Testing & Deployment is easy.
* one bug doesn't ruin our life.
  we need multiple bugs!
* With microservices, it's easy to build complex applications.

Your micro services can also have multiple DBs.

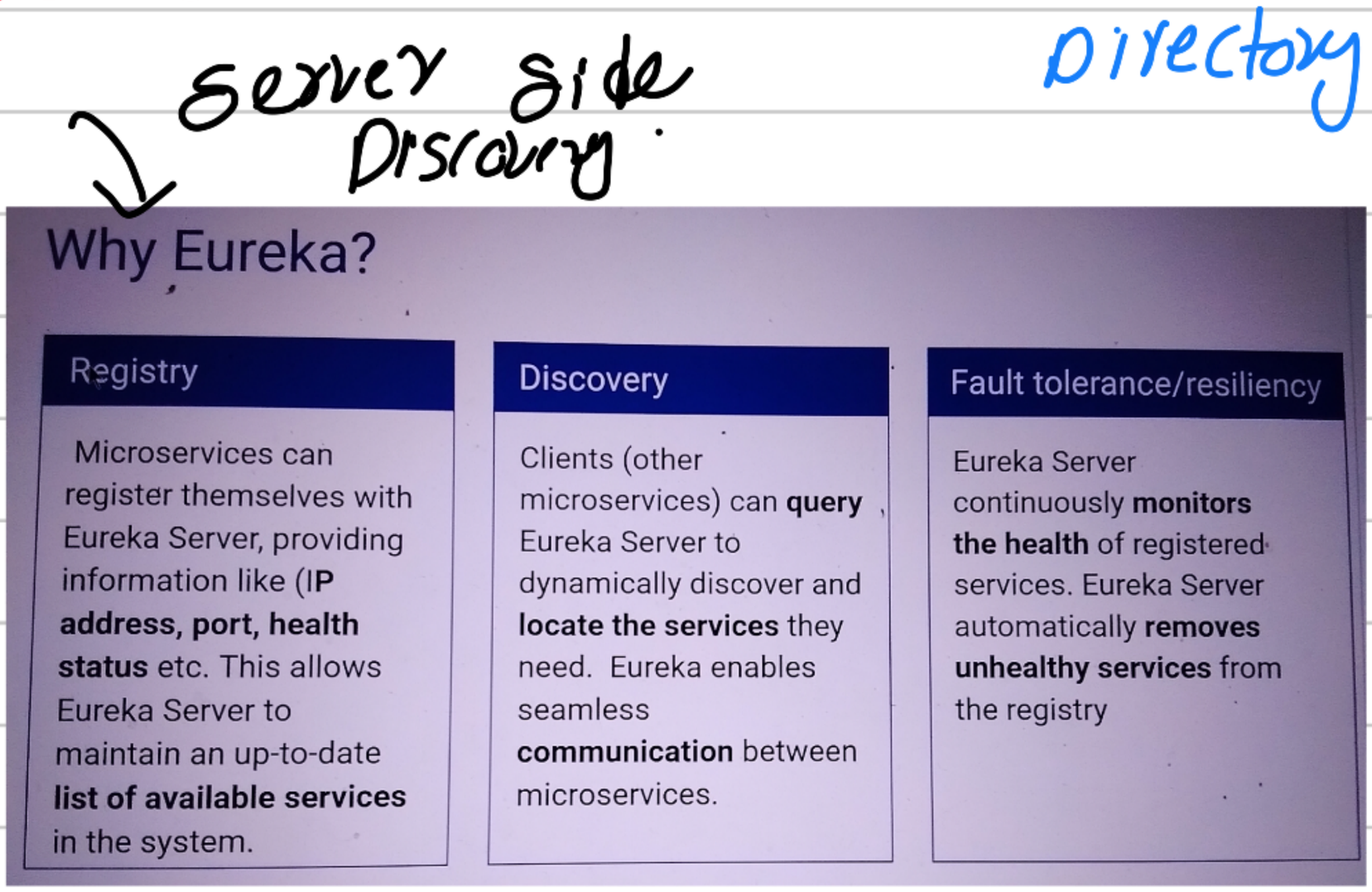
Deployment on cloud is usually on instances.

Due to crash issues, a new IP Address may be assigned by the cloud provider. You will have to change all the urls in existing

micro service who wants to communicate with the created one.

use Eureka → Phone Book directory.

server side Discovery.



Basic Steps

① Create SpringBoot Project.

② Starter dependencies for web eurelca, lombok, mysql, jpa.

③ Configure properties in application.yml.

④ MVC Architecture →

→ Controller Layer

→ Service Layer

→ Repo Layer

→ Entity

→ DTO Layer

* Eureka Server Setup

Steps

* Add eureka server Dependency
* Annotate with @Enable Eurelca Server
* Configure property file.

Go to Application :

add

@ Enable Eurelca Server

Application.yml :-

```
server:
    port: 8761

eureka:
    client:
        fetch-registry: false
        register-with-eureka: false
```

Restaurant listing microservices :-

* web
* lombok
* mysql driver
* JPA
* eureka client.
* mapstruct

Creater

* controller
* repo
* service
* entity
* dto (Data Transfer Object)

Restaurant Entity

@Data ⟶ Getter, Setter, toString, equals, hashCode

@ All Args Constructor ⟶ Generates a constructor with all defined attributes.

@ No Args Constructor ⟶ No attribute implying (parameters) constructors.

They all come from a java library called Lombok that reduces boilerplate code

id (int)
name (str)
address (str)
city (str)
restaurantDescription
rating (int)

what does Bean mean & autowiring?

stream() →

Bean → an annotation used to tell spring to create an object & manage it as a "Spring Bean"

@ Configuration ← init.

@ **Autowired** → please give me an instance of this class from Application context.

stream → process collections using a pipeline.

Note: → You never return an Entity
→ you always return a Data Transfer Object
→ we convert Restaurant object into a Data Transfer Unit.

```
public List<RestaurantDto> getAllRestaurant() {
```

* This method returns a List of RestaurantDto objects.
* RestaurantDto is likely a class with selected fields to return.

.stream → converts a list into pipeline of data flowing.

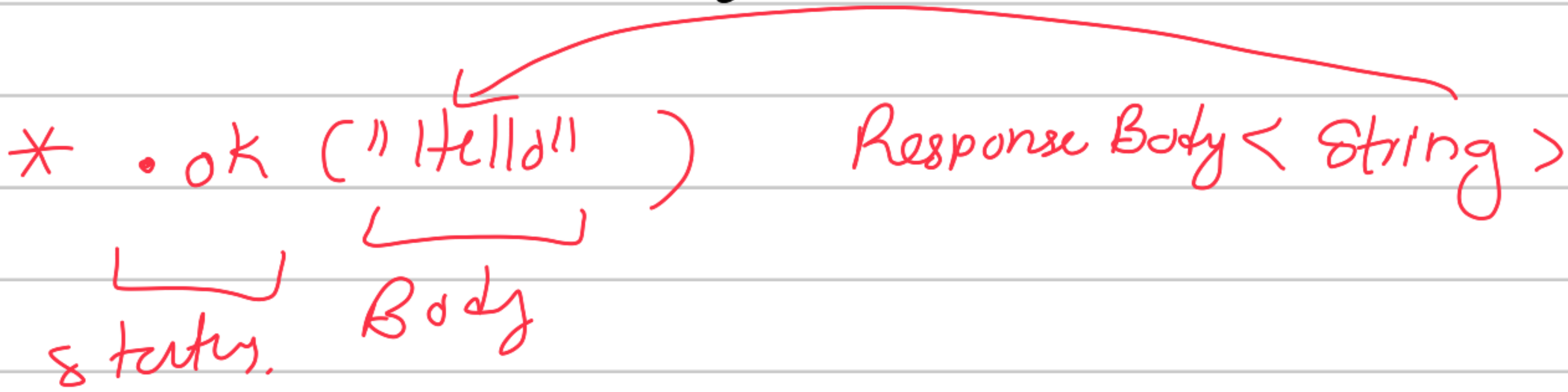
Understanding Response Entity

↳ Handles HTTP responses from APIs.

↳ Set the HTTP status code.

↳ Add Headers

↳ Response Body returned.



Common Static Methods

→ ok(body) Returns 200 OK

→ created(URI) Returns 201 Created

→ noContent() Returns 204 No Content

→ badRequest() Returns 400 Bad Request

→ notFound() Returns 404 Not found

→ status(HTTP Response)

↳ return ResponseEntity.status(HttpStatus.CREATED).body(created)

Handling Edge Cases

* If user not present in the database

```
public ResponseEntity<RestaurantDto> getRestaurantById
        (@PathVariable String id) {
    return restaurantService.getRestaurantById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
```

Syntax ResponseEntity::ok

this is a method reference ↪ in Java.

.map(...) method expects a function that takes a value (the found RestaurantDto) and returns ResponseEntity<RestaurantDto>

actually:

.map(value -> ResponseEntity.ok(value))

when we create / insert something i.e how is a new restaurant added →

@PostMapping("/create")

public ResponseEntity<RestaurantDto> createRestaurant(
@RequestBody RestaurantDto req)

Spring uses a library called Jackson under the hood:-

1. Read the incoming JSON.
2. Use the no argument constructor to create an empty object.
3. Using setters like setName(), setAddress()

to populate values from JSON

No mismatches or anything! Nothing Breaks :)

User Service

handles id, name, password, address

↳ same mysql as previous micro services.

User Entity :-

* id int
* username string
* passwad string
* address string
* city string.

* get all user info on id (POST)

* add usr (post)

Food Catalogue Listing Micro Service.

Description:

On frontend, the user experience is going to be when the user selects one restaurant, list down the particular food list belonging to that restaurant.

Tech Stack same as previous micro services.

Entity:

Food Item

int id
String itemName
String item Description
String Ingredients.
Number price
Integer restaurantId
Integer quantity (intial = 0)

Food Catalogue.
List< FoodItem) ItemList.
Restaurant name;

* addFood Item endpoint.

name, quantity, restaurandId, price, Ingredients, itemDescription

Goal: Return Food Catalogue → foodItem List
→ Restaurant Details.

* Note: Always modularize the code!!!

What is Load Balancing with Eurelca Server?

Eurelca helps microservices register themselves as they start And discover each other without hardcoding Internet Protocol Addresses or ports.

The concept of Load Balancing →

→ Means distributing incoming requests evenly across multiple service instances.

↳ 3 instances of clients ordering food.

Using Round robin of instances.

Benefit:

- No hard coding of URLs
- Automatic failover → If some instance fails then route traffic to some other instance.
- Even Load Distribution
- Easy horizontal scaling

↓

If one instance is unable to handle the traffic then create more instances.

ORDER SERVICE

Overview:

When we hit the order button the user, food details are saved in the backend.

WHY THAT IS THE CASE?

→ Microservice architecture

→ Rest APIs

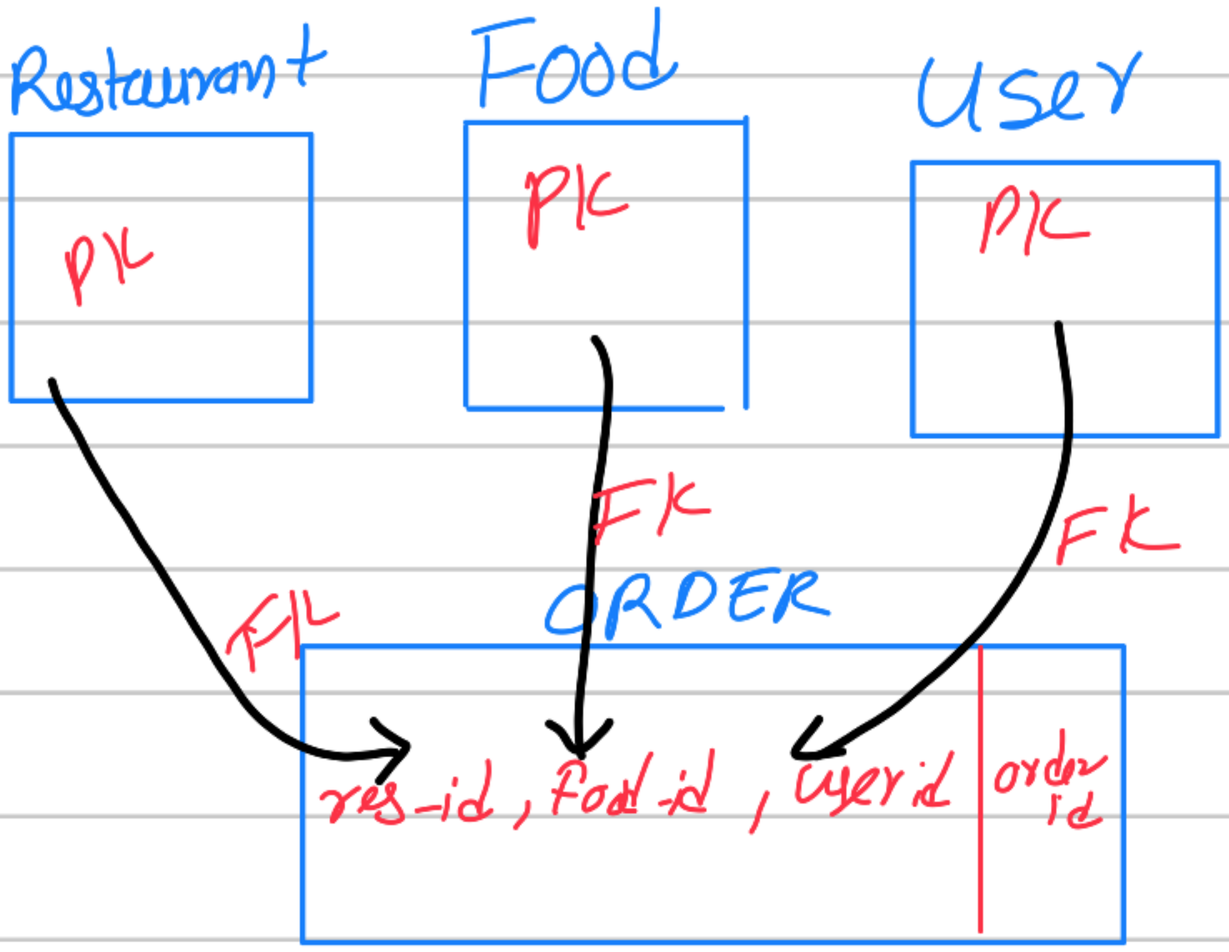
→ Java

→ Spring Boot

→ MongoDB (MAJOR CHANGE)

→ Lombok & Eureka

Why?
SQL →



NOSQL

one big file with basically everything

```
order {
  res {
  }
  user {
  }
}
```

* No foreign Keys
* No Joins
* No multiple Tables
* Faster Read & write.

findAndModify used to find document and modify it in single atomatic operation.

query(where("_id").is("sequence")):

This specifies the query criteria-

It uses the where method to define the condition and is to specify the value of the _id field.
In this case, it searches for a document with _id equal to "sequence".

new update().inc("sequence",1):
This part defines the update operation. The inc methods increaments the value of the "sequence" field by 1. It creates a new update object and specifies the field to update and the increment value.

options().returnNew(true).upsert(true):
Return option indicates that the modified document should be returned as a result.

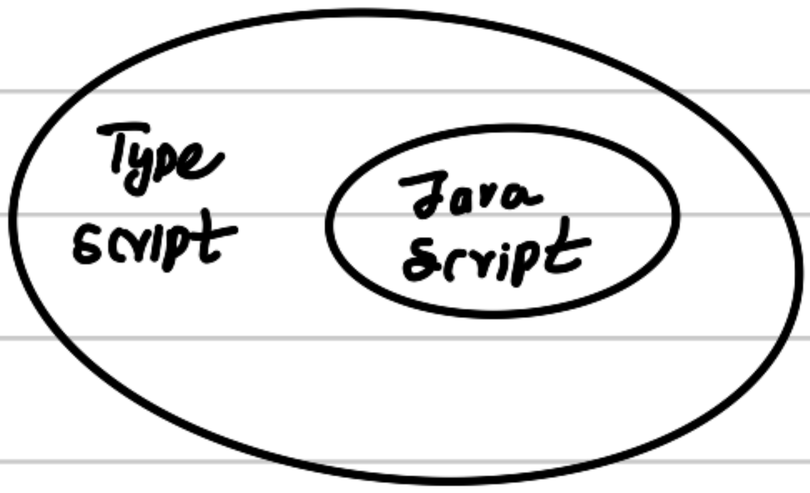
Sequence.class : Sequence class where collection is defined.

counter.getSeq(): Code retrieves the value of the 'seq' field from modified document and returns it.

Overall, just increase seq by 1!!!

Note:
From this point, the next few pages go over some Next.js frontend stuff!

Front End :



VS code

Node Js

* open source runtime env to execute the code to Browser.
* basically a compiler.

npm install -g @angular/cli ← install Angular

ng new my-first-project. ← creates app

ng generate ← generate components

ng serve ← test locally

Package.json & Packase-lock.json.

↳ contains starter dependencies. initially!

Carrot → either this or latest.

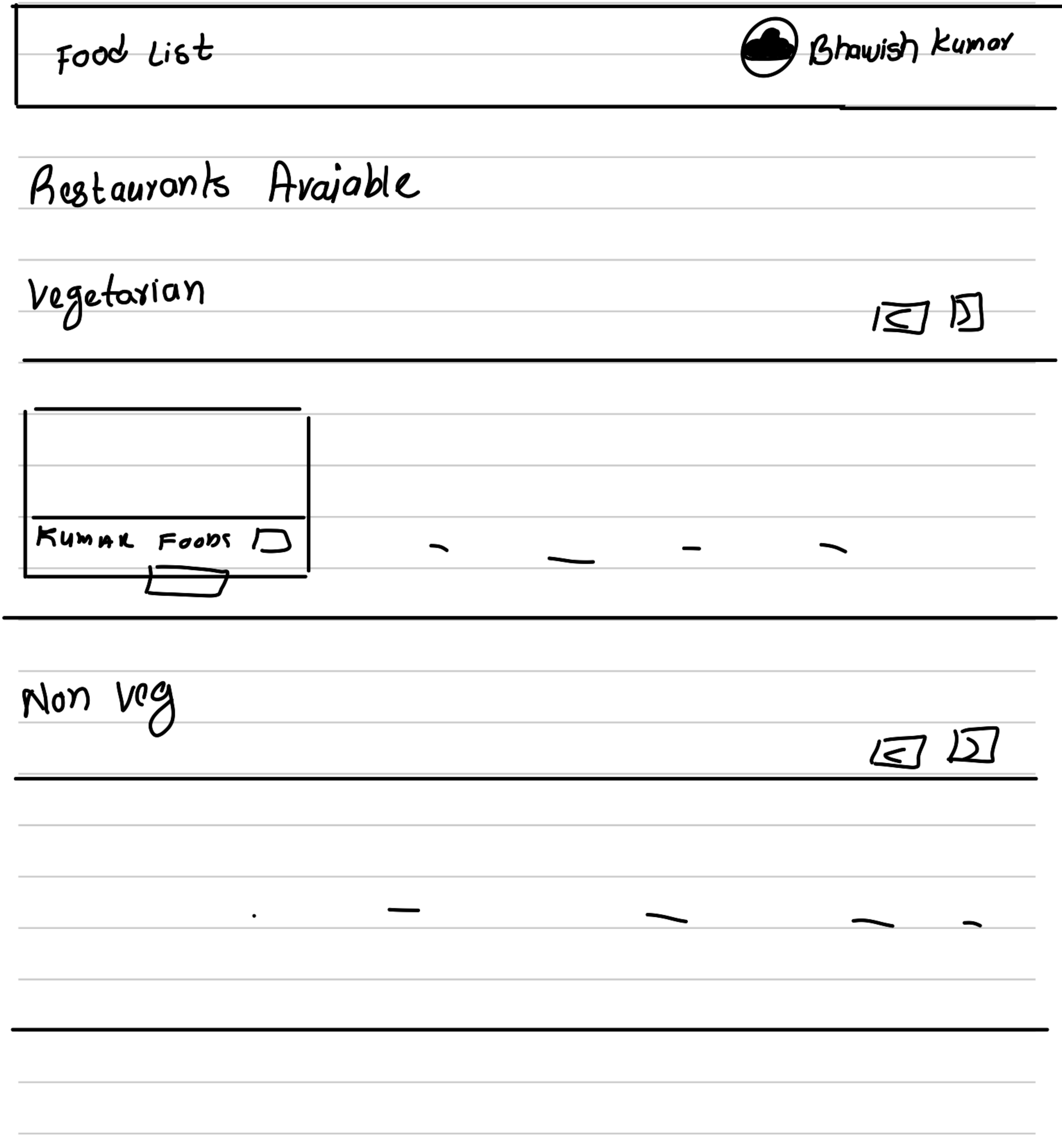
Tilde →

Shifting Back to Next j's →

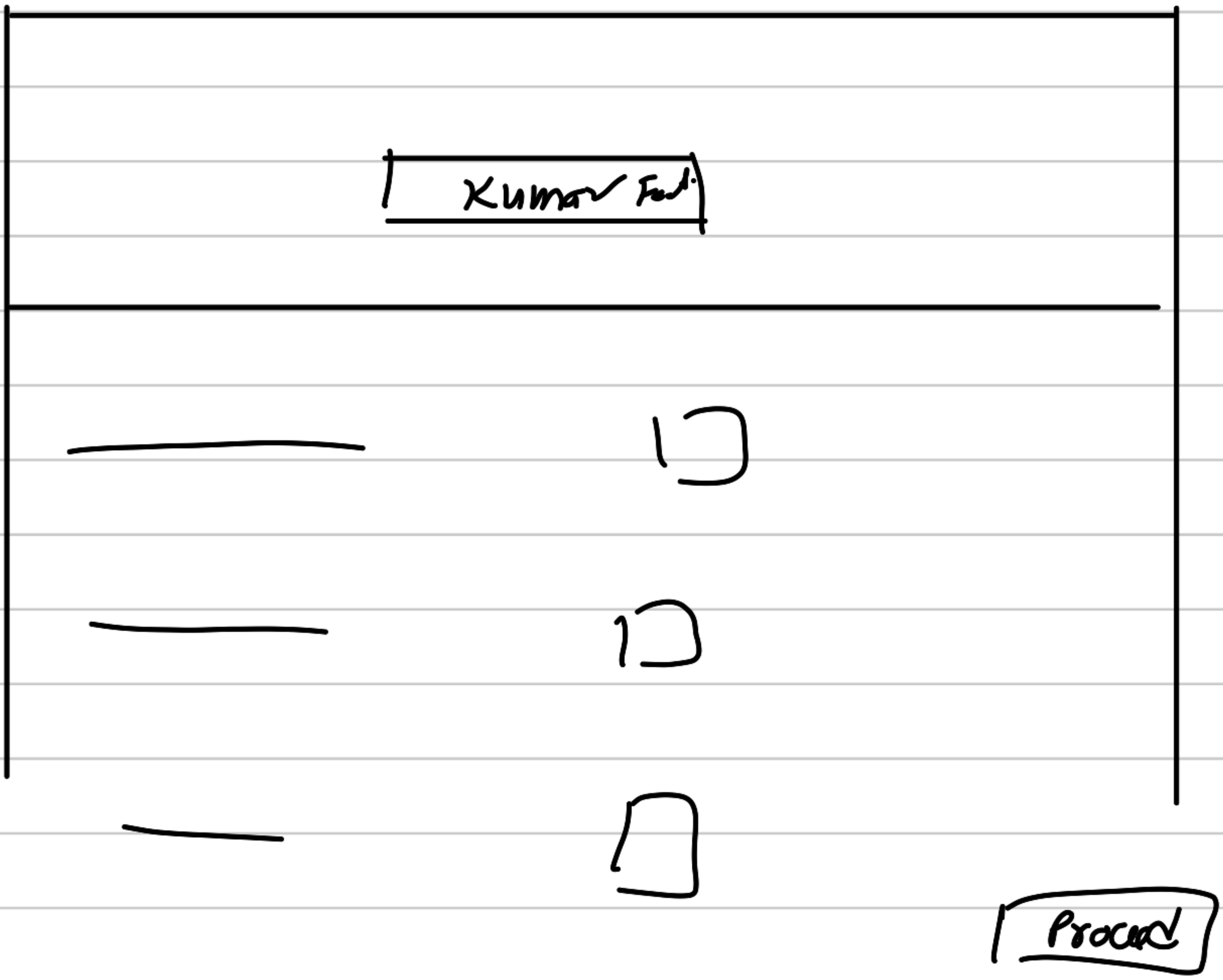
Pages →

Wire frame pages :

① Restaurant Listing Page :-



Menu Page



Order Checkout

Use Effect in React — a detailed study

→ use Effect (() => { ... })

* after each render

→ use Effect (() => { ... }, [ ]);

// mount → run

unmount → Cleanup

→ use Effect () => { ... }, [a, b]);

when a or b change.

*Effects runs only on clients.

Mental Timeline:

* React renders your component (pure calculation → JSX).

* DOM commits (user sees it)

* React runs your effects(s).

* Next render happens → before running the new effect, React runs cleanup of previous effect.

* Unmount → React runs cleanup!

Whenever one of the values in the dependency array changes, React reruns the effect.

For state or props put them under dependency array!

Unpacking the simple

```
onChange = {(e) => ...}
```

well not that simple maybe

→ is a react prop for an input that runs whenever the input's value changes!

(e) => ... is an arrow function that takes event object e.

↳ .target.value

DOCKER

containerization platform which packages your application and dependencies together in the form of containers to ensure that your application works seamlessly in any environment (test, dev or prod)

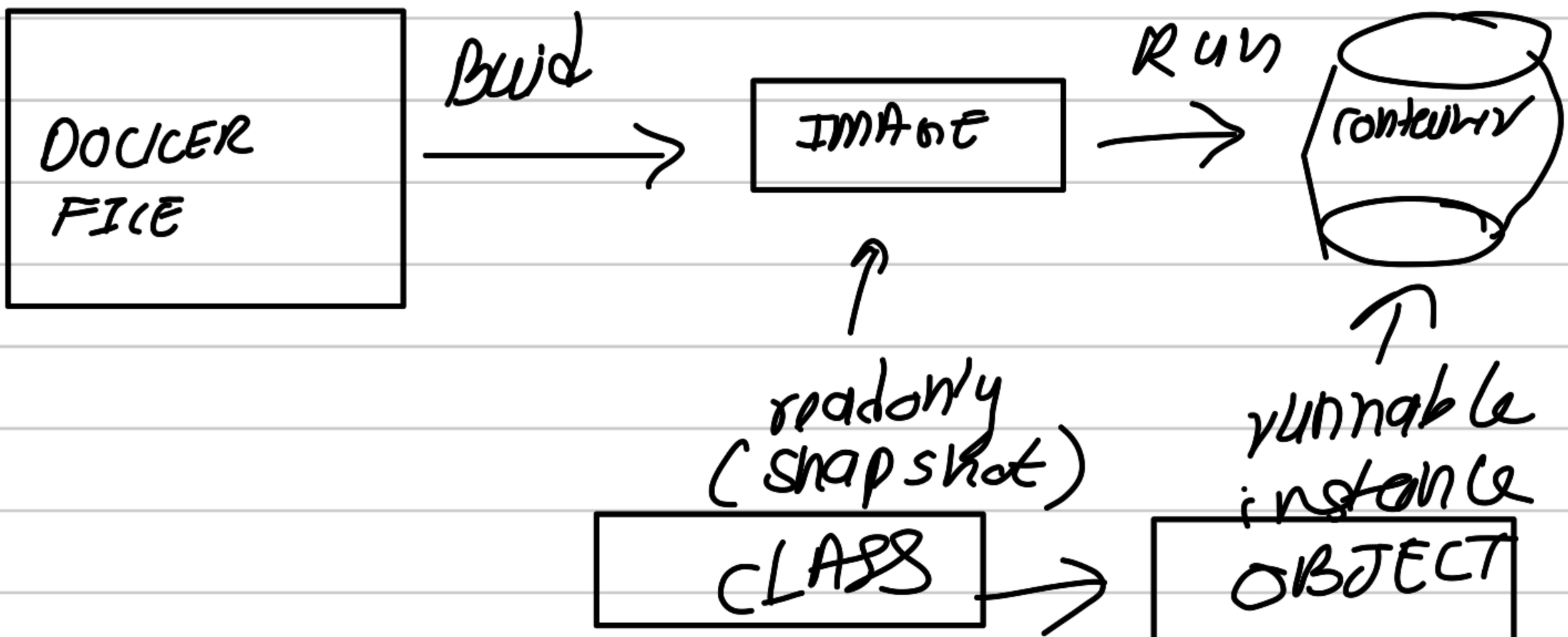
Hence a container is complete ecosystem of your application.

Including the application and all of its dependencies and everything to run.

So regardless on environment, the application runs!

Container has items (application + dependencies)

Dockerfile → Instructions.



DOCKER FILE →

* This text file provides a set of instructions to build a Docker Image including the operating system, env variables, file locations, network ports, other components it needs to run.

* Each time a command is run in a Docker file, a new layer is created on top of previous layer. Docker image is built incrementally.

example →

```
FROM        openjdk:17-alphine      ] Parent Image.
WORKDIR     /opt                    ] define working directory of a Docker container at any given time.
ENV PORT 8080                       ] define environmental variables.
COPY target/*.jar /opt/app.jar
ENTRYPOINT exec java $JAVA_OPTS -jar app.jar
                                    ↳ RUN Application.
```

WHAT IS DOCKER IMAGE?

* Docker images are read only binary template/snapshot used to build containers.

* Create a docker image using Docker build command whenever you pass a Dockerfile to the docker build command then the docker daemon will create a docker image according to the file.

* Run the docker image using **docker run**

Whenever we pass the command to docker client then the docker client passes this command to the docker daemon then docker daemon will create the container for the image.

* Push the docker image to public registry like Docker Hub using **docker push** command after pushed you can acess these images from anywhere using **docker pull** command.

DOCKER CONTAINER

*A container is a runnable instance of an image. You can create, stop, start, move or delete container using Docker API or CLI.

* Containers provide you a light weight & platform-indp way of running your applications.

* Container is volatile, it means whenever you remove/kill the container then all of its data will be lost.
Use docker storage to persist.

* Containers only have acess to

resources that are defined in image unless additional access is defined.

Docker Hub?

How to create a docker file and push it

Docker Hub → BACKEND

Eureka :-

```
FROM adoptopenjdk/openjdk11:jdk-11.0.2.9-slim
WORKDIR /opt
COPY target/*.jar /opt/app.jar
ENTRYPOINT exec java $JAVA_OPTS -jar app.jar
```

We are not using Expose & env instructions

WHY?

* EXPOSE instruction is used to inform Docker that a container process listens on a specific network port at runtime.

The purpose of EXPOSE INSTRUCTION is to provide metadata about the intended network ports that the container process will use.

When using kubernetes for deployment, you do not rely on the EXPOSE instruction in Dockerfile Instead you define desired ports in

Kubernetes itself.

↳ Deployment configuration.

FRONTEND :-

Stage 1 : Build Angular Application

FROM node:16 as build

WORKDIR /app app

COPY package.json ./ package.json.

RUN npm install

COPY .] All Remaining files.

RUN NPM RUN BUILD.

↓
Install all dependencies

Stage 2: Serve app using nginx

FROM nginx : alphine

COPY --from=build /app/dist / food-ordering-gui / usr / share / nginx / html

→ is an HTTP server (high performance)

EXPOSE 80

CMD ["nginx", "-g daemon off;"]

Redmi

In this stage, the base image is set to nginx alphine which includes nginx, a light weight web server.

* The copy command copies the build artifacts from the previous stage (named build) to nginx.

EXPOSE instruction indicates that the container will listen on port 80 default HTTP port used by Nginx.

* docker build -t codedecode25/app-name:0.0.1
* represents current directory

docker build -t Redmi/eurika-service:0.0.1 .
run docker desktop first.

docker login
docker push <image name>

Spring Boot Profiling

feature that allows developers to configure & customize their application's behaviour based on different runtime environments.

our projects often use urls / urls such as database connections, server connections However these are for local thus when we move to prod or test we end up losing them.

By using spring boot profiling, you can define different sets of properties, configuration & dependencies.

This enables you to have a separate configuration for each profile

* Spring Boot provides a way to define multiple configuration files each associated with specific profile.

* You can activate one by:

spring.profiles.active

Eureka server's endpoint with Kubernetes cluster _

service url of the eurekaserver in Kubernetes

http://eureka-0.eureka-service.default.svc.cluster.local:8761/eureka

Amazon Web Servias

*Introduction:-

→IaaS : Infra structure as a Servie

Blank OS
Do whatever.

→PaaS: Platform as a servie

Azure ← manage
Elastic Kubernetes Services.

→SaaS: Software as a service

Entire servie provided.

Servias we use :-

① AWS RDS →
This Database AWS service is easy to set up, operate & scale a relational database in Cloud.

② AWS EKS → Allows you to use kubernetes on AWS without installation.

③ AWS ALB → Load Balancer

④ AWS EC2 Instanle → EC2 is

a virtual machine in the Cloud on which you have OS level control - You can run this Cloud server.

AWS RDS

* Databases can also be deployed just like other micro services. But we don't do it!

WHY ?

↳ Production apps that work on Kubernetes do end up crashing! It's weird why industry still adopts such a crashable instance service. Any ways, that is not the topic today.

But Yeah, because they crash, data can be lost! henceforth the database does not serve its purpose.

Pod → smallest unit that runs something in kubernetes

one pod runs MySQL

one pod runs Node.js app

Why we don't usually run database itself inside Kubernete?

Database → Stateful

Pods → Stateless

No backups, no updates → really hard

All manual!

What people should do instead?

* inside cluster → Stateful set + Persistent Volumes

* Cloud → AWS RDS / Aurora

outside cluster and apps connect.

Cloud provider runs your database DB.

AWS RDS → Amazon Relational Database Servias

is a fully managed database servie. By leveraging provision database instance, scale resouras up or down as needed and easily!

With AWS RDS →

MySQL
Postgre SQL
Maria DB
Oracle
Microsoft Mysql Server.

Same & simple procedure to setup on AWS. Very simple.
Connect to mysql and replace the urls.

Mongo db Atlas:-

multi-cloud database servie by mongodb.

Mongodb Atlas internally uses EC2 CAUS on the infrastructure level) but actualy this will reduce our work!

KUBERNETES:-

open source container orchestration platform that automates the deployment, scaling, management of containerized apps.

managing lifecycle of containers

deployment

scaling

networking

storage

It ensures that the desired state of application or system is maintained.

Kubernetes provides a framework for managing & running these containers across a cluster of machines.

Basically avoiding all manual work!

Why Kubernetes!-

Simplified Deployment
Automated Scaling
Load Balancing
Fault Tolerance.
Rollouts & Rollbacks.
easier to deploy, develop & manage applications.

less focus on infra

Components of Kubernetes :-

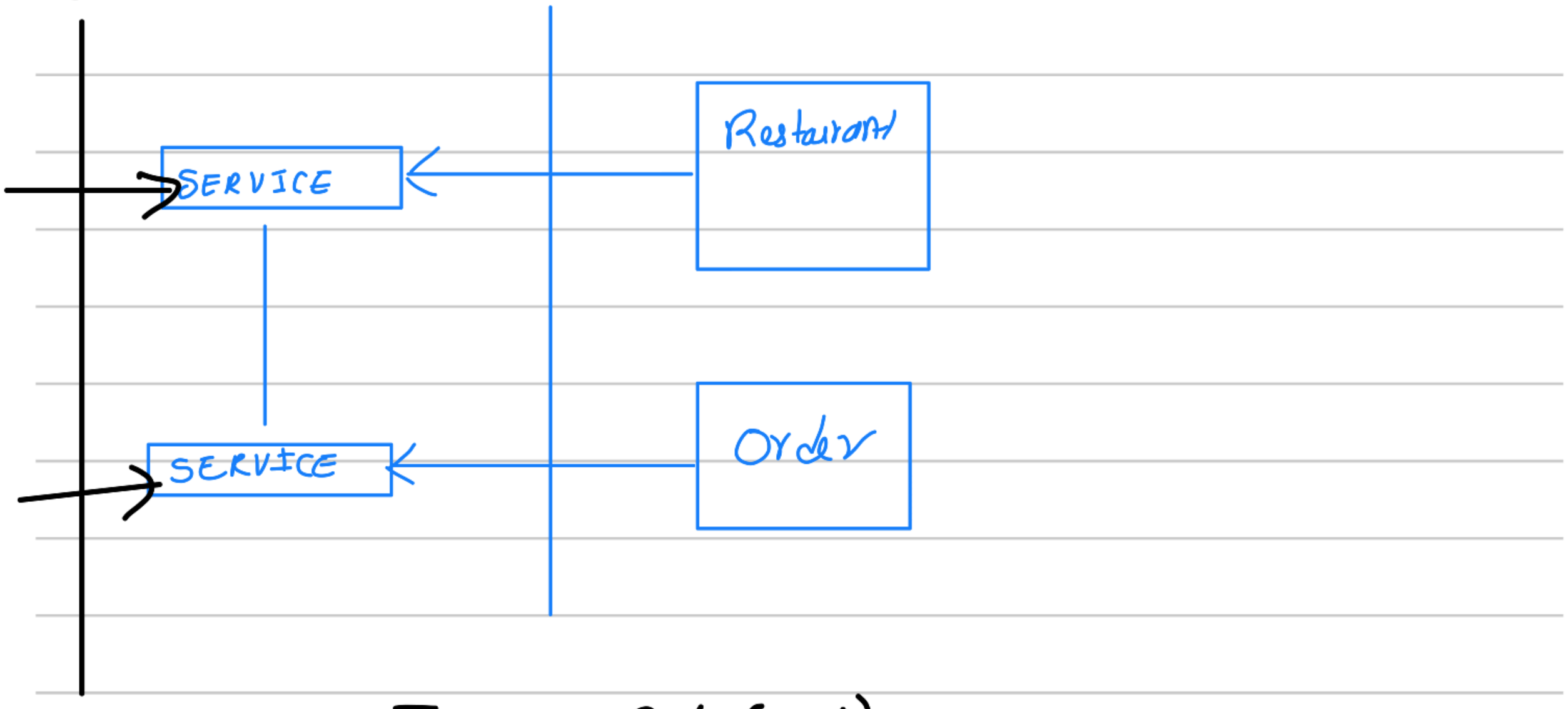
Node :- You need to deploy your app on a physical server or VM and that machine is called "node or worker node" as this works!

PODS :- smallest deployable unit and the basic building block. K8 → 

→ each pod has its own IP address
→ pods are designed to be crashed and disposable.
→ K8s basically poke & register the stats & handle different circumstances.

Don't use IP Address !!! on CRASH

SERVICES :-

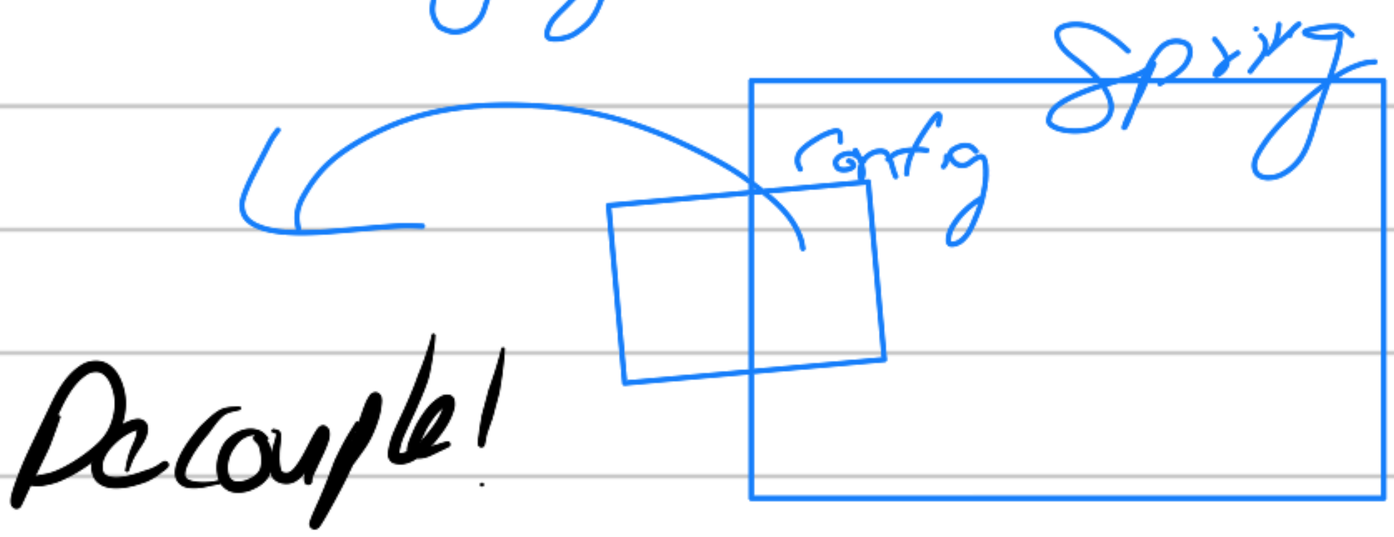


① Cluster IP :- (default) accessible only within one cluster. Exposes service on an internal IP within the cluster and only accessible via one!

② NodePort :-
This type exposes the service on a specific port of each cluster node's IP address.
It makes service accessible from outside the cluster by using node's IP address & assigned NodePort.

③ Load Balancer (ext): assigns ext IP address and provided by cloud provider!
↳ Need More Details!!!

Config Map → is used to store configuration data that can be consumed by containers running in pods.
* It provides a way to decouple the configuration from the application code, allowing for easy configuration changes without modifying the container image!



Secret: can't store password in text in config map. It stores password in base64 encoded format!

Volumes:- are the way to provide persistent storage to containers running pods.

They enable containers to work with shared and persistent data in Kubernetes.

Means if your database in pod crashes, your data is safe in volumes.

What is data persistence?

Ability of data to outlive the process that created it!
i.e the data remains without any crash-

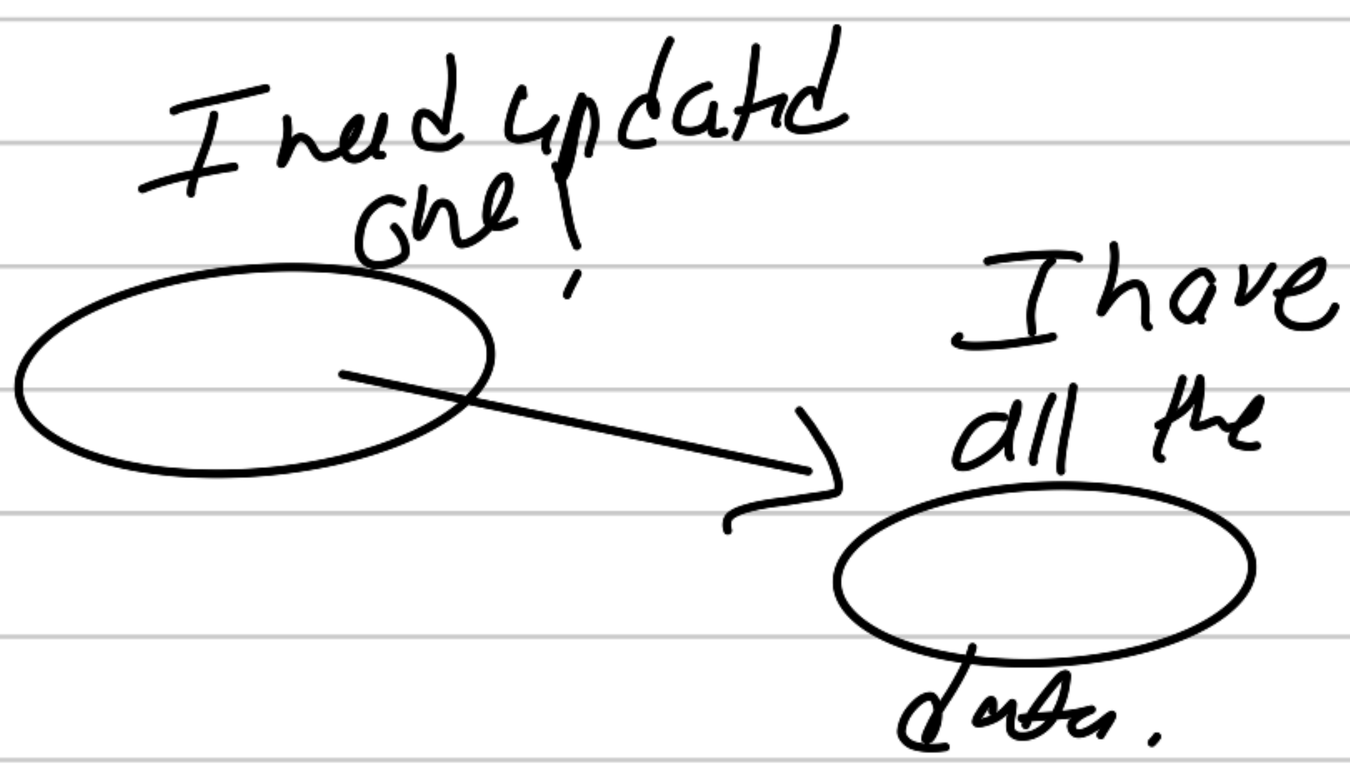
What is persistent volumes (PV) :-

(PV) → cluster resource → RAM

Kubernetes objects that represent storage resources in your cluster. PVs work in conjuction with Persistent Volume Claims (PVs)-

Storage that doesn't depend

You need to configure kubernetes'



honestly I don't fully understand but these notes don't require me to go in Kg much tbh:

* Pods crash! so we need the replicas so thus KB can still work.

* we never manually create pods! we always do deployments.

What is a deployment?
Manager of application basically.

has no relation to config maps, secrets, etc.

For stateful (DB) → state ful set
For Stateless (App) → Deployment.

We End here for now because I don't have AWS

In Next Part we dive into KB, AWS, Junits and production.