Micorservices

Microservice

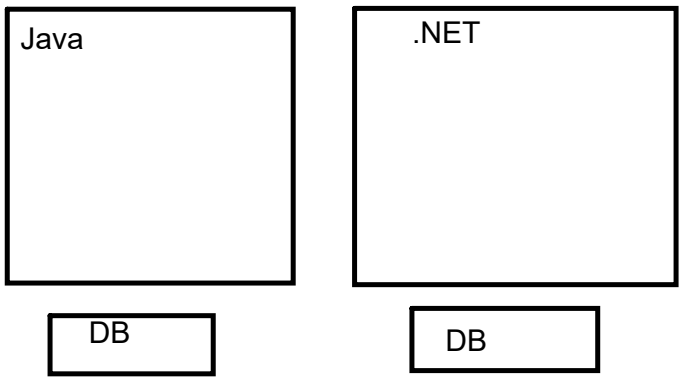
Modularity

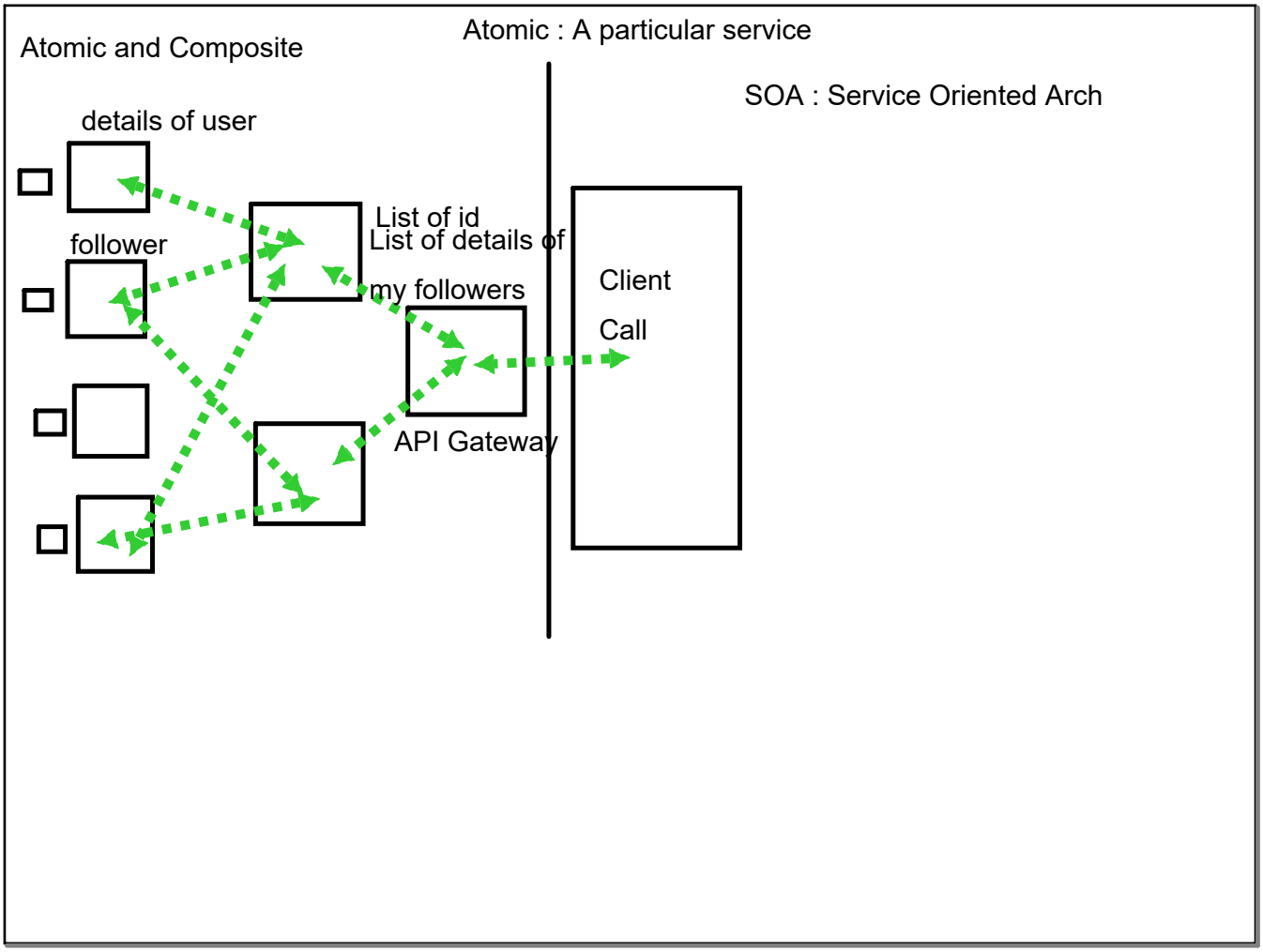
Use technology

Single MS drop

Dynamic Scale up/Scale Down

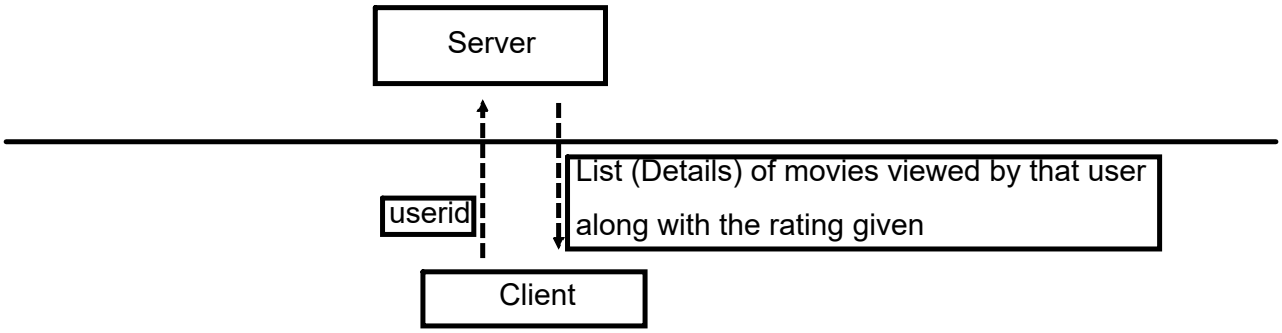
Independent/Autonomous

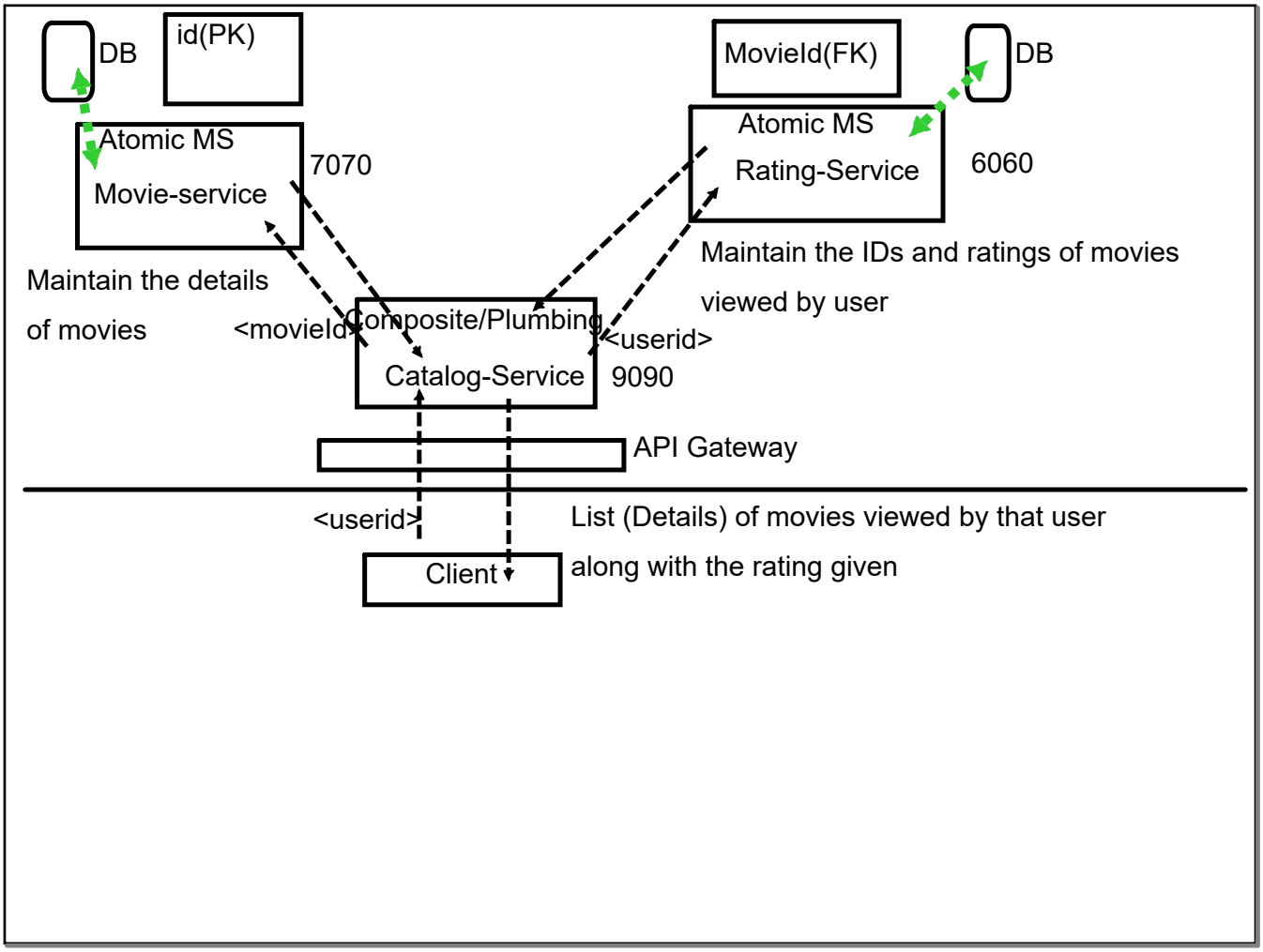


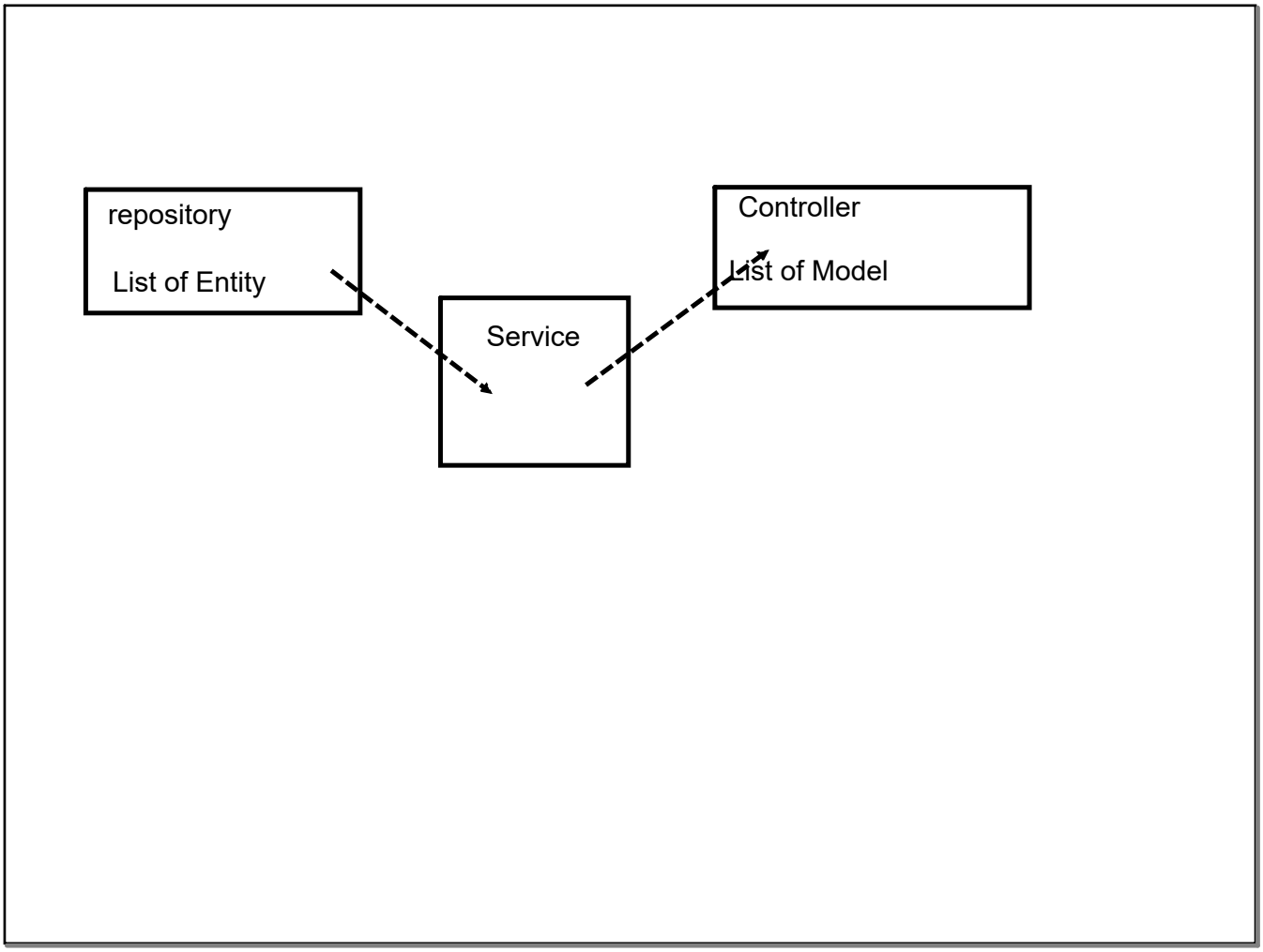

Atomic and Composite
Atomic : A particular service
SOA : Service Oriented Arch
details of user
follower
List of id
List of details of
my followers
Client
Call
API Gateway

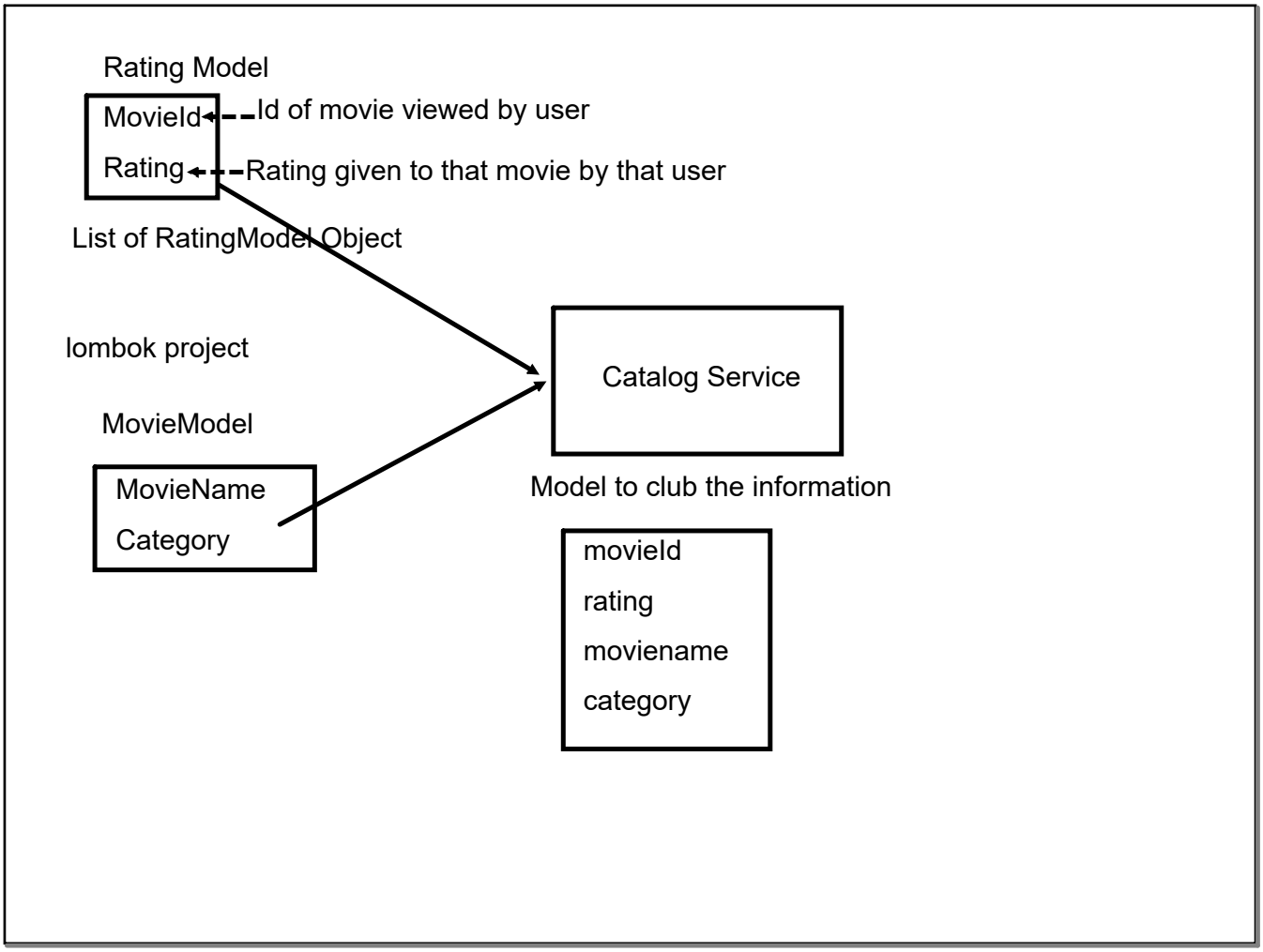
Use Case:

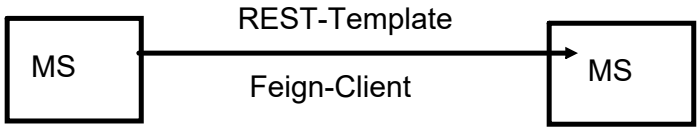
Movie Streaming portal




DB
id(PK)
Atomic MS
Movie-service
7070
Maintain the details
of movies
<movieId>
MovieId(FK)
DB
Atomic MS
Rating-Service
6060
Maintain the IDs and ratings of movies
viewed by user
<userid>
Composite/Plumbing
Catalog-Service
9090
API Gateway
<userid>
List (Details) of movies viewed by that user
along with the rating given
Client


repository
List of Entity
Service
Controller
List of Model


Rating Model
MovieId
Id of movie viewed by user
Rating
Rating given to that movie by that user
List of RatingModel Object
lombok project
MovieModel
MovieName
Category
Catalog Service
Model to club the information
movieId
rating
moviename
category



Best Practice:

1. Should never return a list/collection directly

   # if list is huge, it creates problem in streaming back to calling service

   # it will stop us to update/add any additional info to associate in future

Model class

Collection

<Any additional info>

Naming Server/Discovery Server

==> Eureka Server

1. no need to know the low level of MS (IP/PORT)
2. will auto fetch one of the instances of MS on demand

Feign-Client : add dependency

=> outsource URL management (HTTP management)

=> configurable

=> abstract

=> Add additional cloud tool

Feign Client:

Works like a proxy

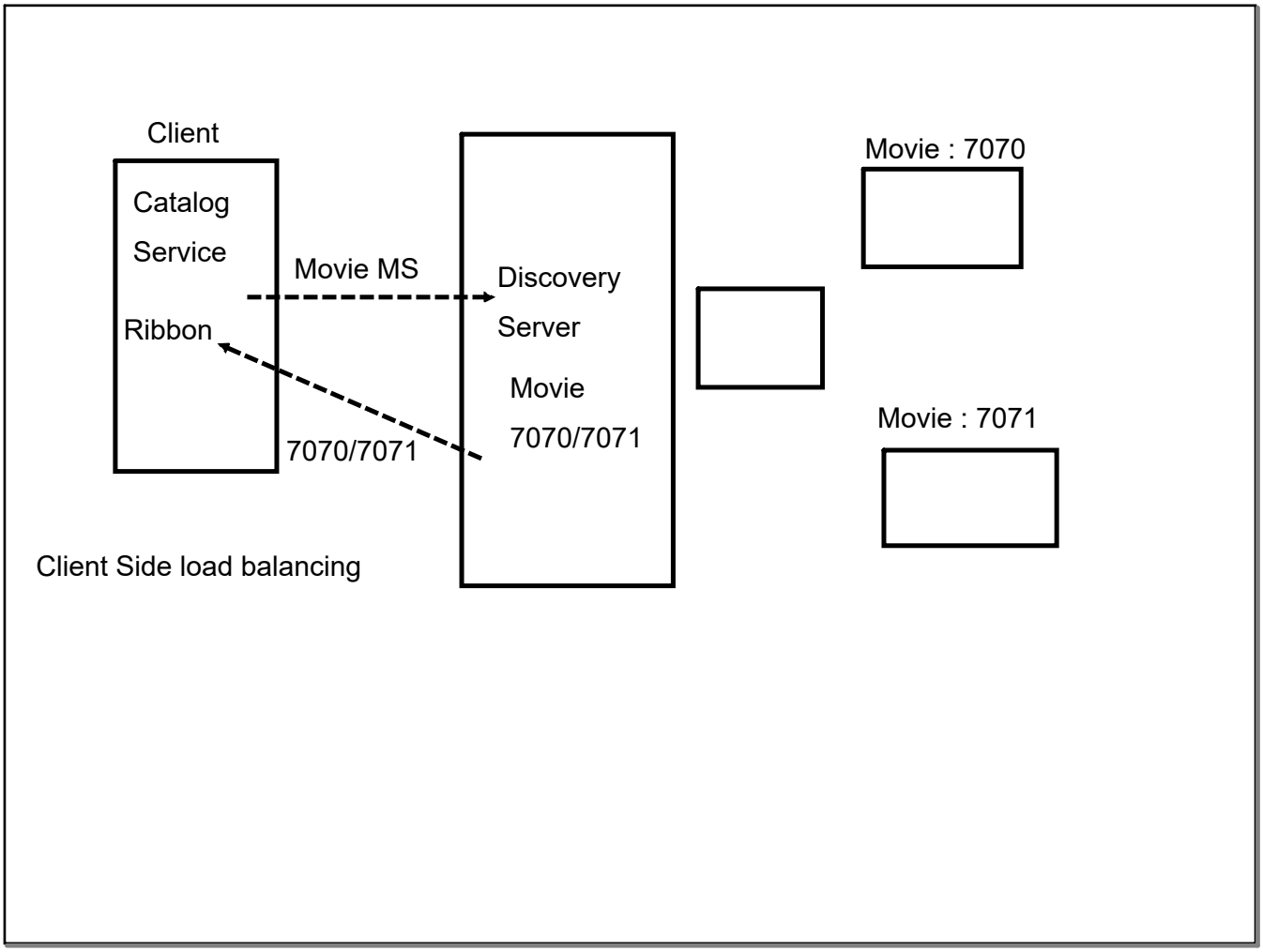
Create interface reflecting the http connection details of other MS (configuration file)

for every MS separate interface

Register all proxies with spring boot application

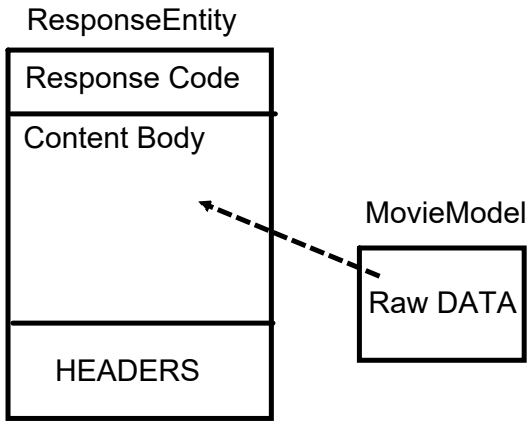
Load Balancing : Ribbon (Client Side load balancer)

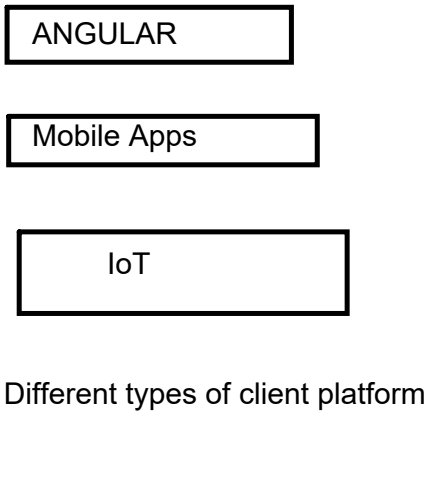
Add dependency

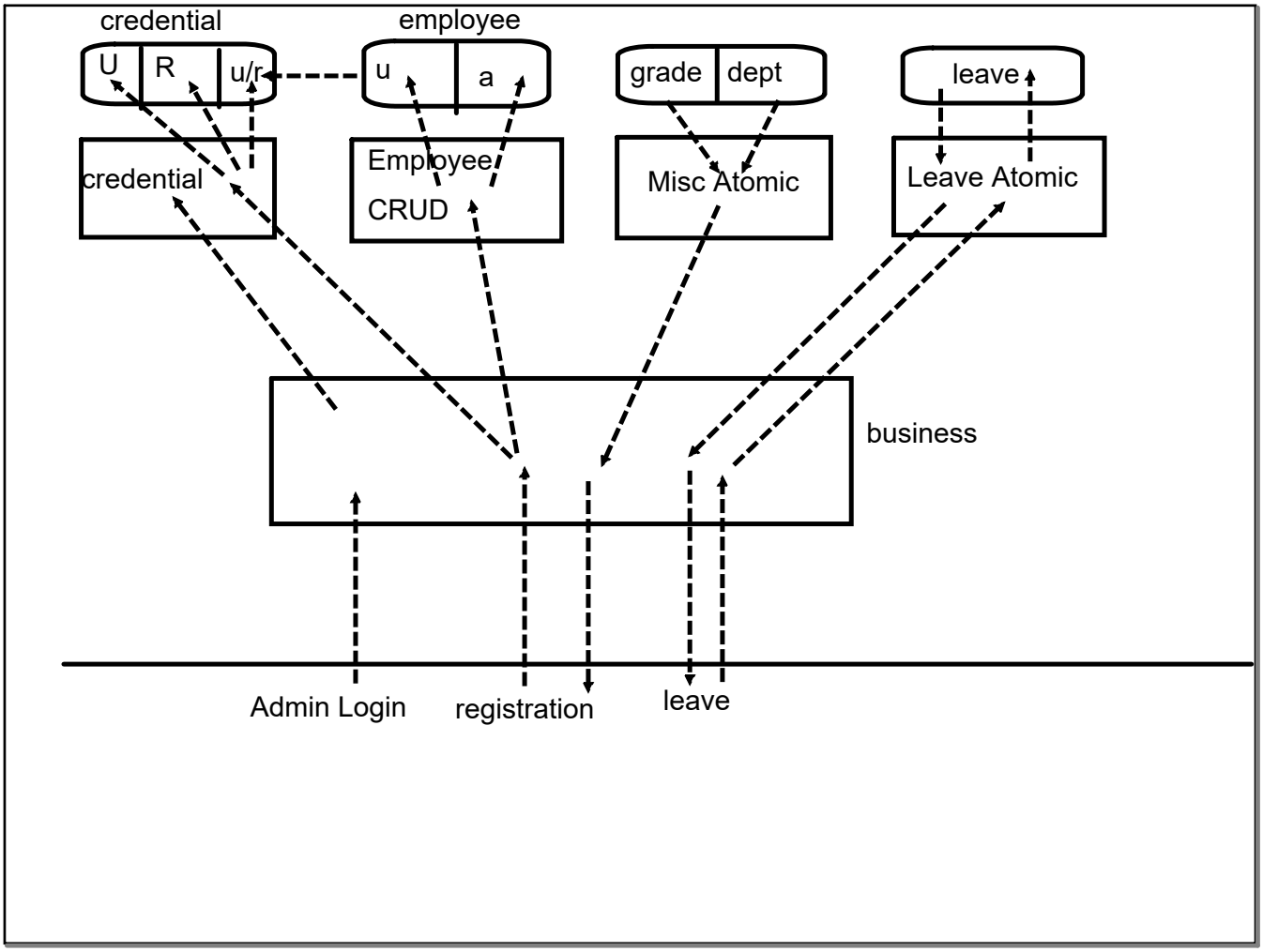

Client
Catalog
Service
Ribbon
Movie MS
7070/7071
Discovery
Server
Movie
7070/7071
Movie : 7070
Movie : 7071
Client Side load balancing

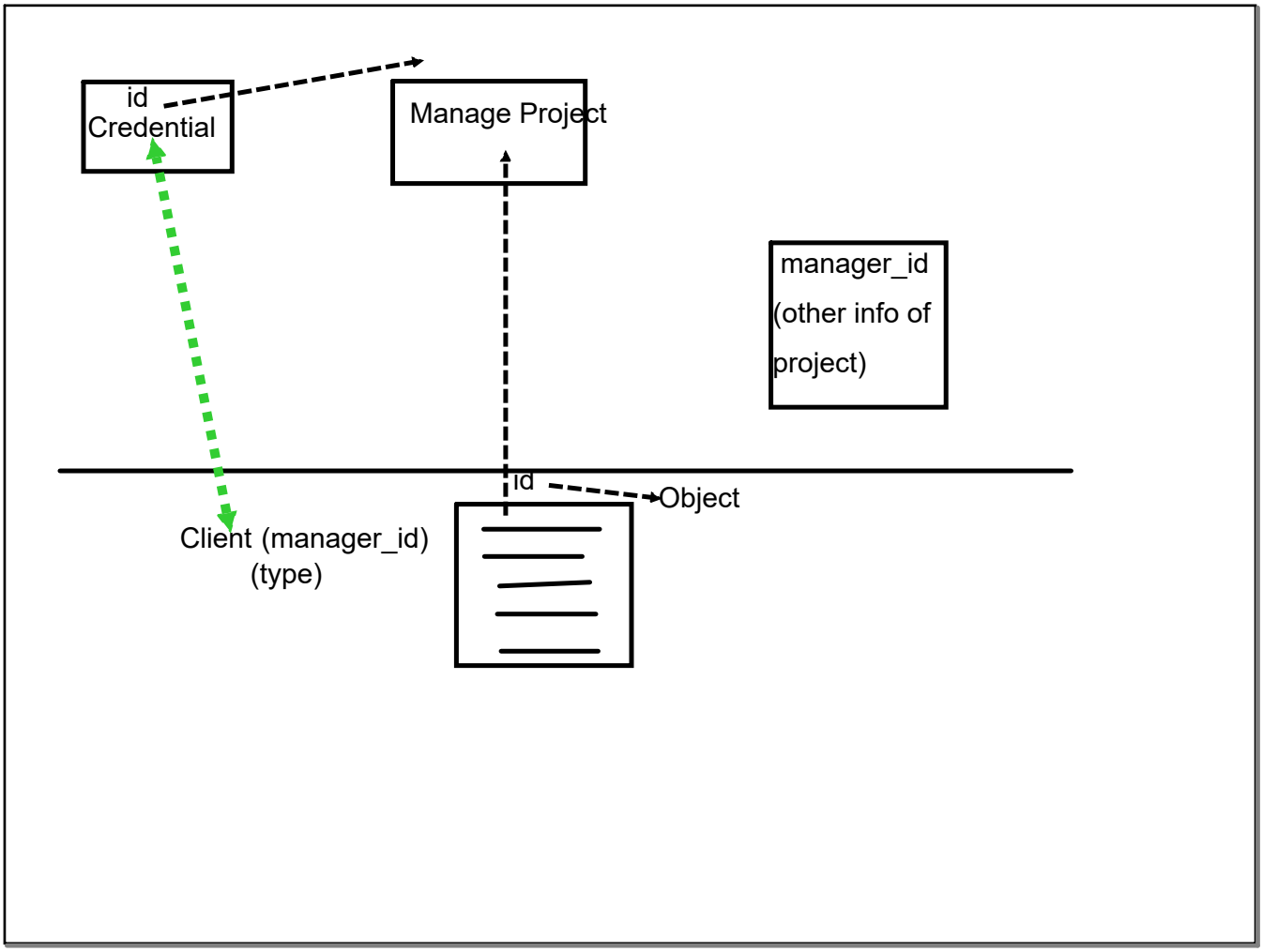
Best Practices

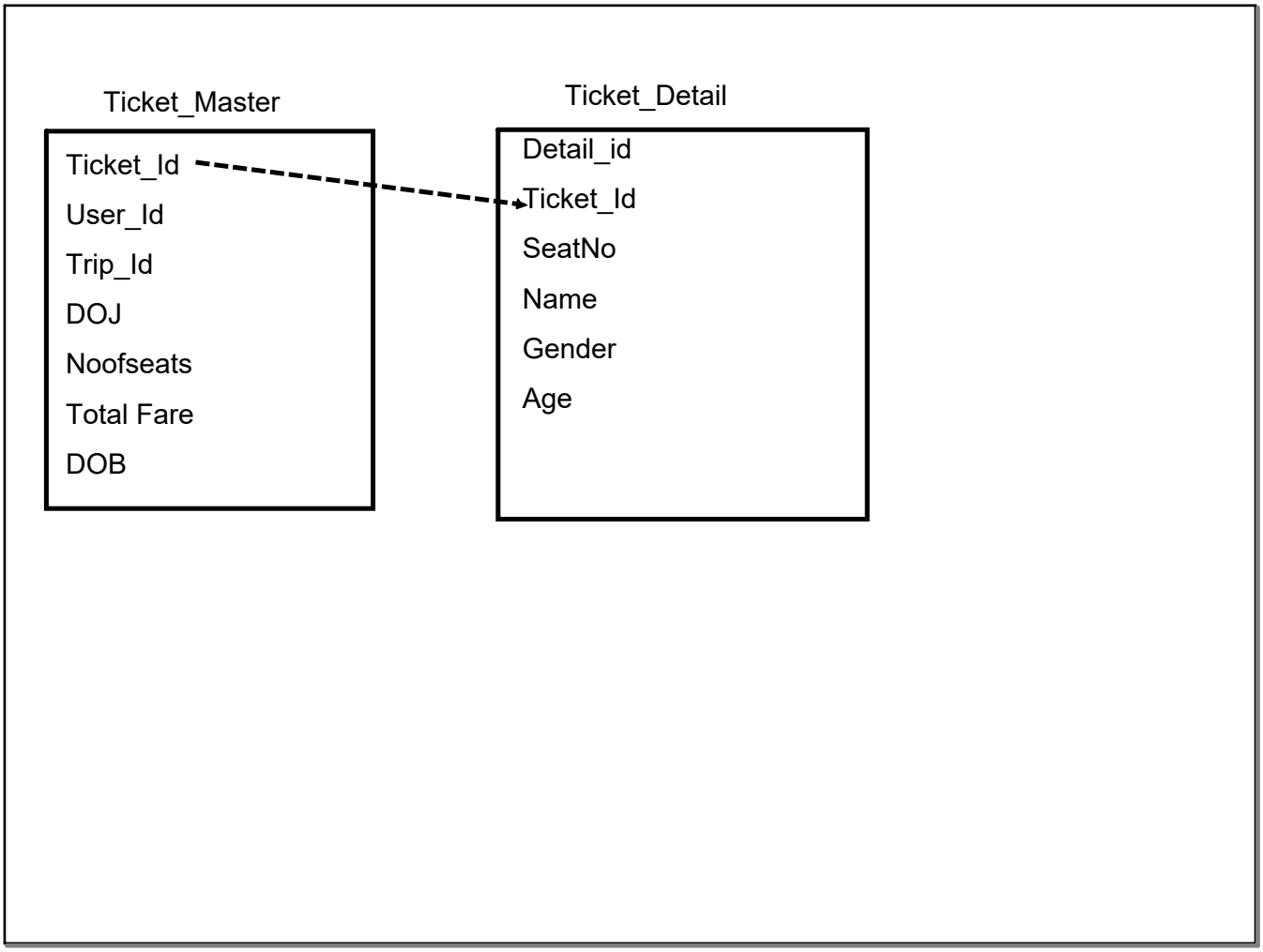
: instead of returning raw data : wrap it in ResponseEntity

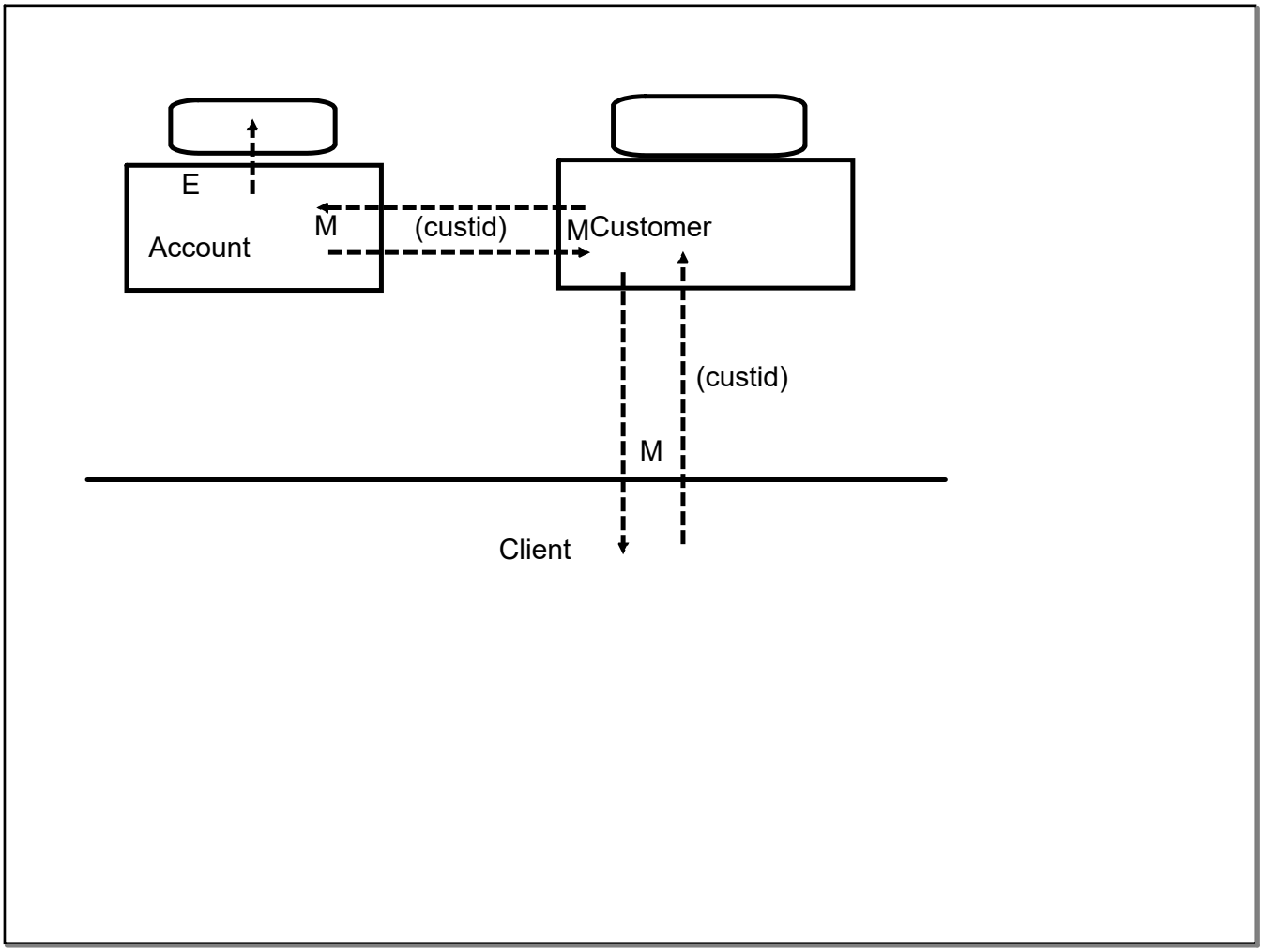





credential
employee
U
R
u/r
u
a
grade
dept
leave
credential
Employee
CRUD
Misc Atomic
Leave Atomic
business
Admin Login
registration
leave

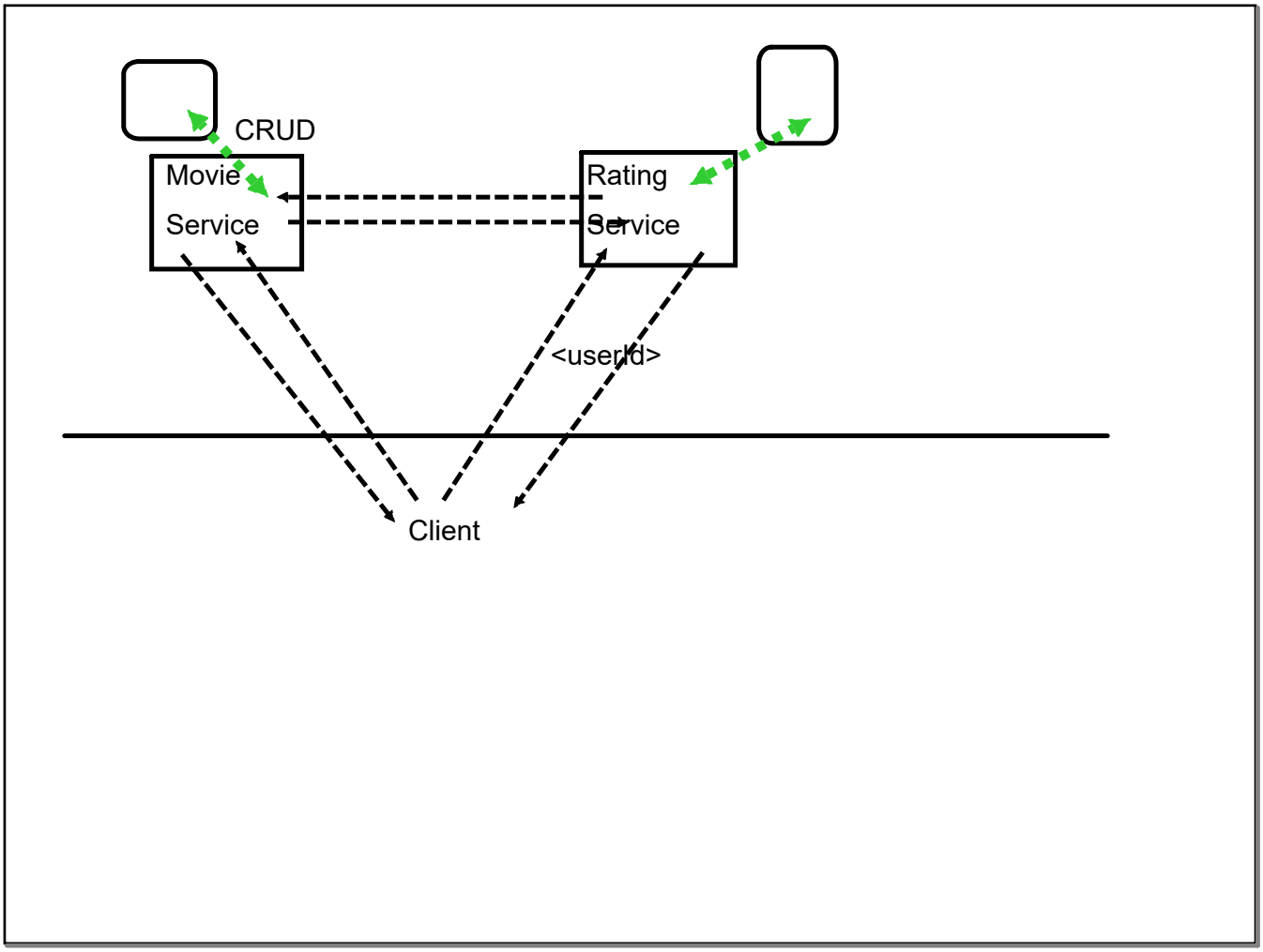

id
Credential
Manage Project
manager_id
(other info of
project)
id
Object
Client (manager_id)
(type)

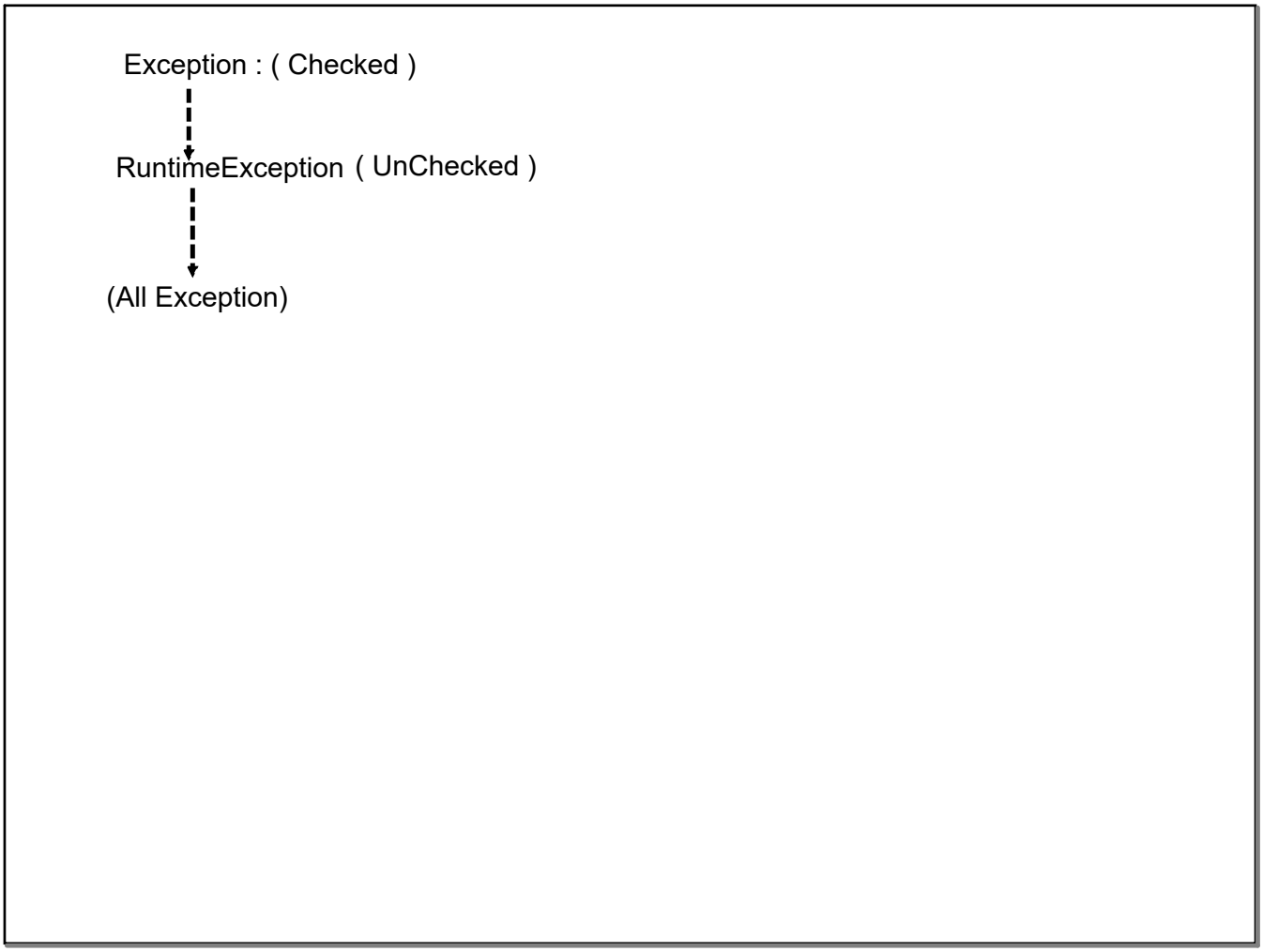

Ticket_Master
Ticket_Id
User_Id
Trip_Id
DOJ
Noofseats
Total Fare
DOB
Ticket_Detail
Detail_id
Ticket_Id
SeatNo
Name
Gender
Age

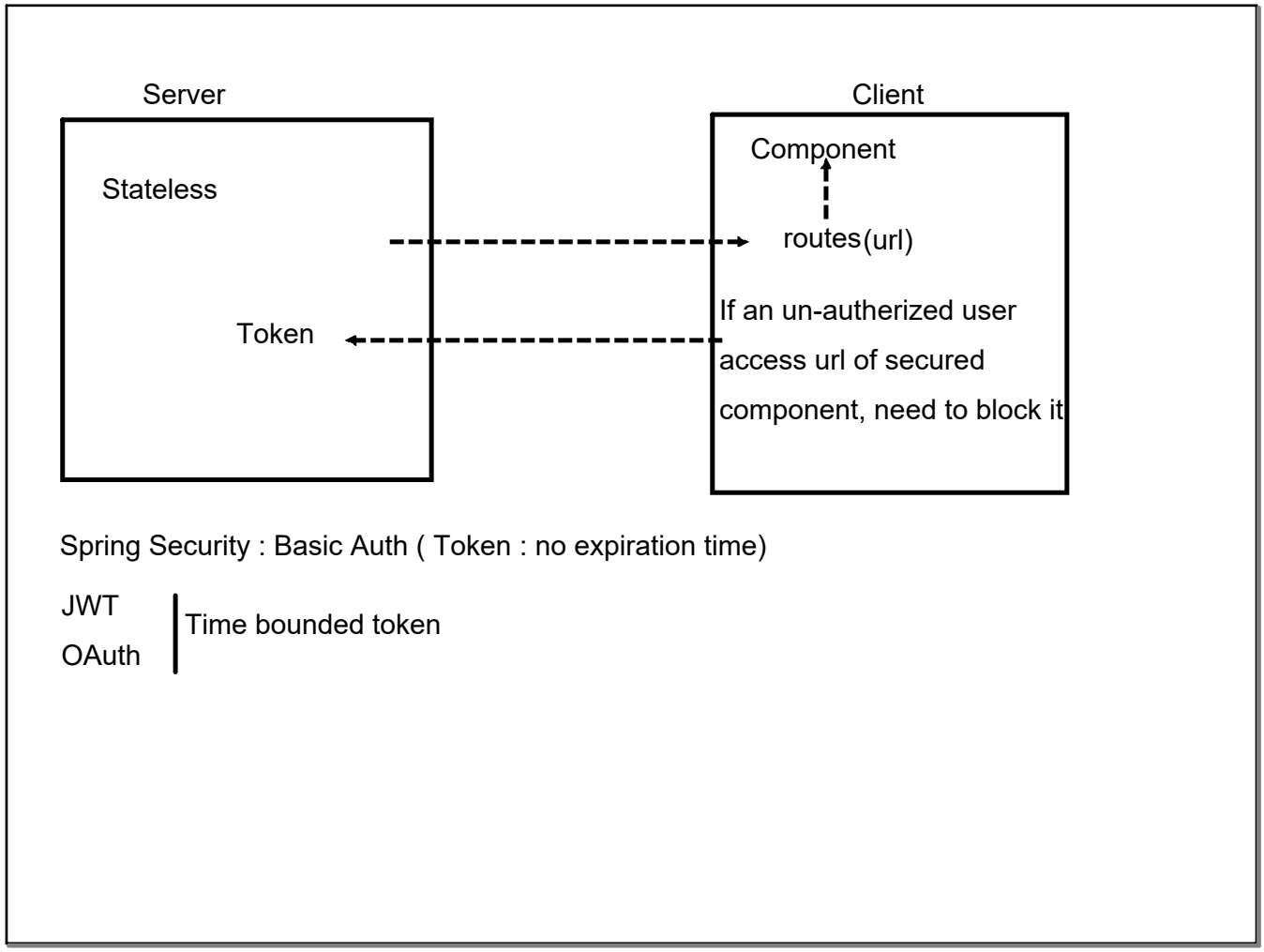

E
Account
M
(custid)
M
Customer
(custid)
M
Client

Custom repository methods

1. Create an interface
2. Add custom method prototype
3. Create a class implementing methods of that interface (using EntityManager)
4. inherit custom interface in repository interface

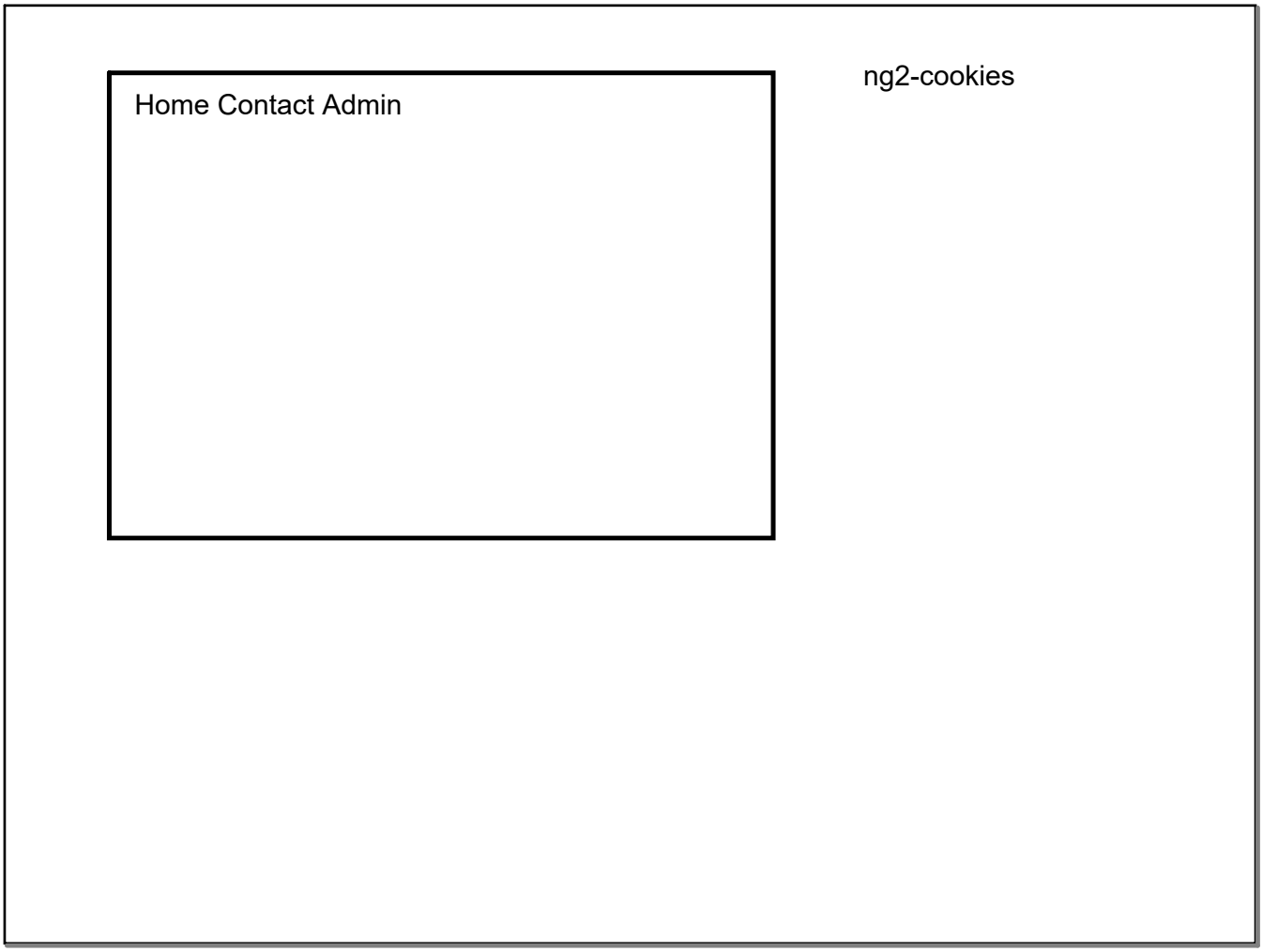

CRUD
Movie
Service
Rating
Service
<userId>
Client

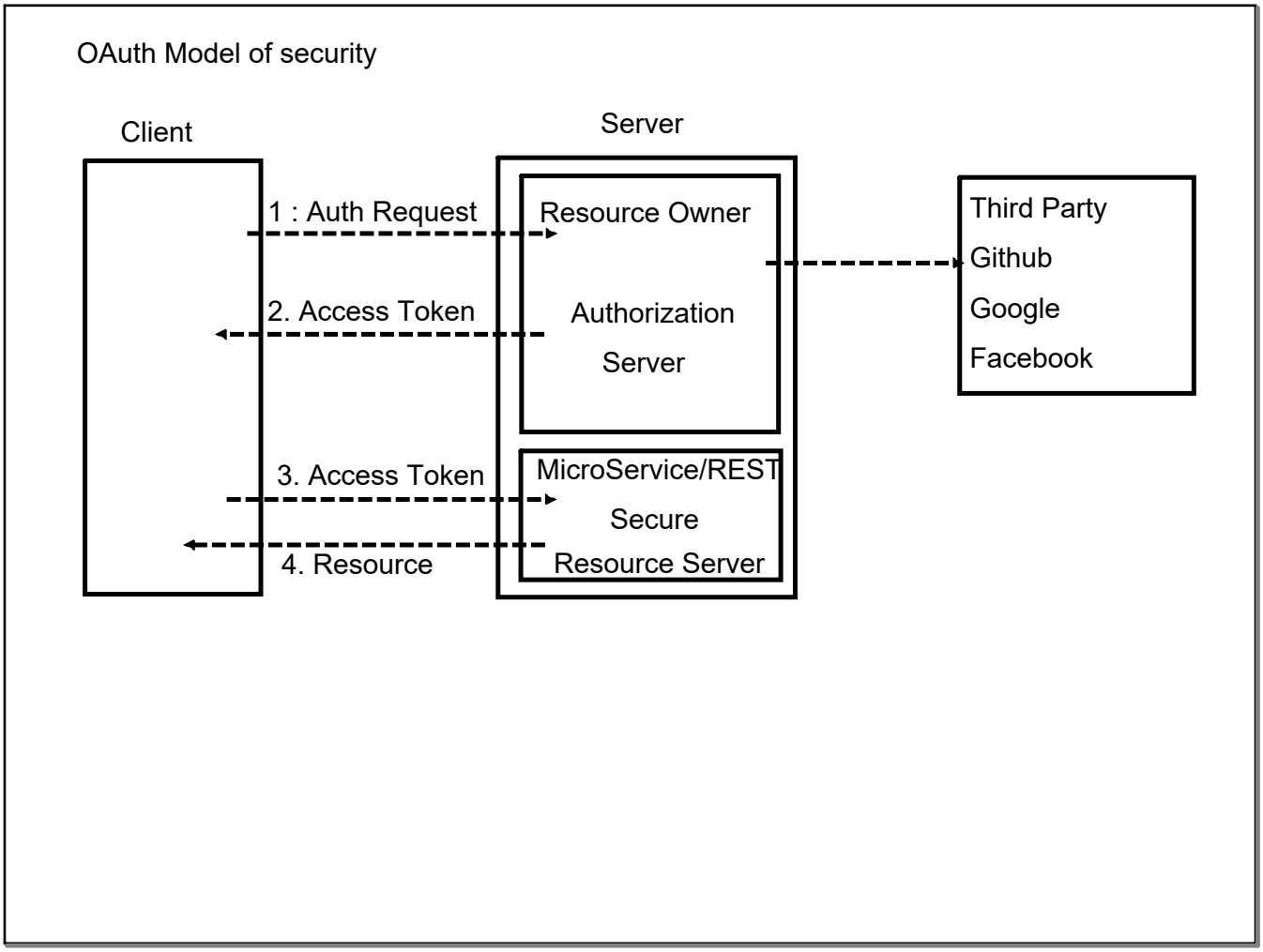

Exception : (Checked)
RuntimeException (UnChecked)
(All Exception)



Spring Security : Basic Auth (Token : no expiration time)

JWT
OAuth
| Time bounded token


Home Contact Admin
ng2-cookies


OAuth Model of security
Client
Server
1 : Auth Request
Resource Owner
Third Party
Github
Google
Facebook
2. Access Token
Authorization
Server
3. Access Token
MicroService/REST
Secure
Resource Server
4. Resource

Authorization Server

Two config file

1. Config the standard spring security
2. Config the OAuth (Third party rules)

OAuth requires passwords to be encoded

OAuth requires the AuthenticationManager from spring security

OAuth provided its own internal REST Endpoints

OAuth Credential (Basic Authentication)

Application Credentials (form data)

Resource Server (Microservice/REST Service)

1. Resource Server Configuration

AGILE : Increment Product ---> AGILE way for CI/CD

Containerization

development env

Staging env

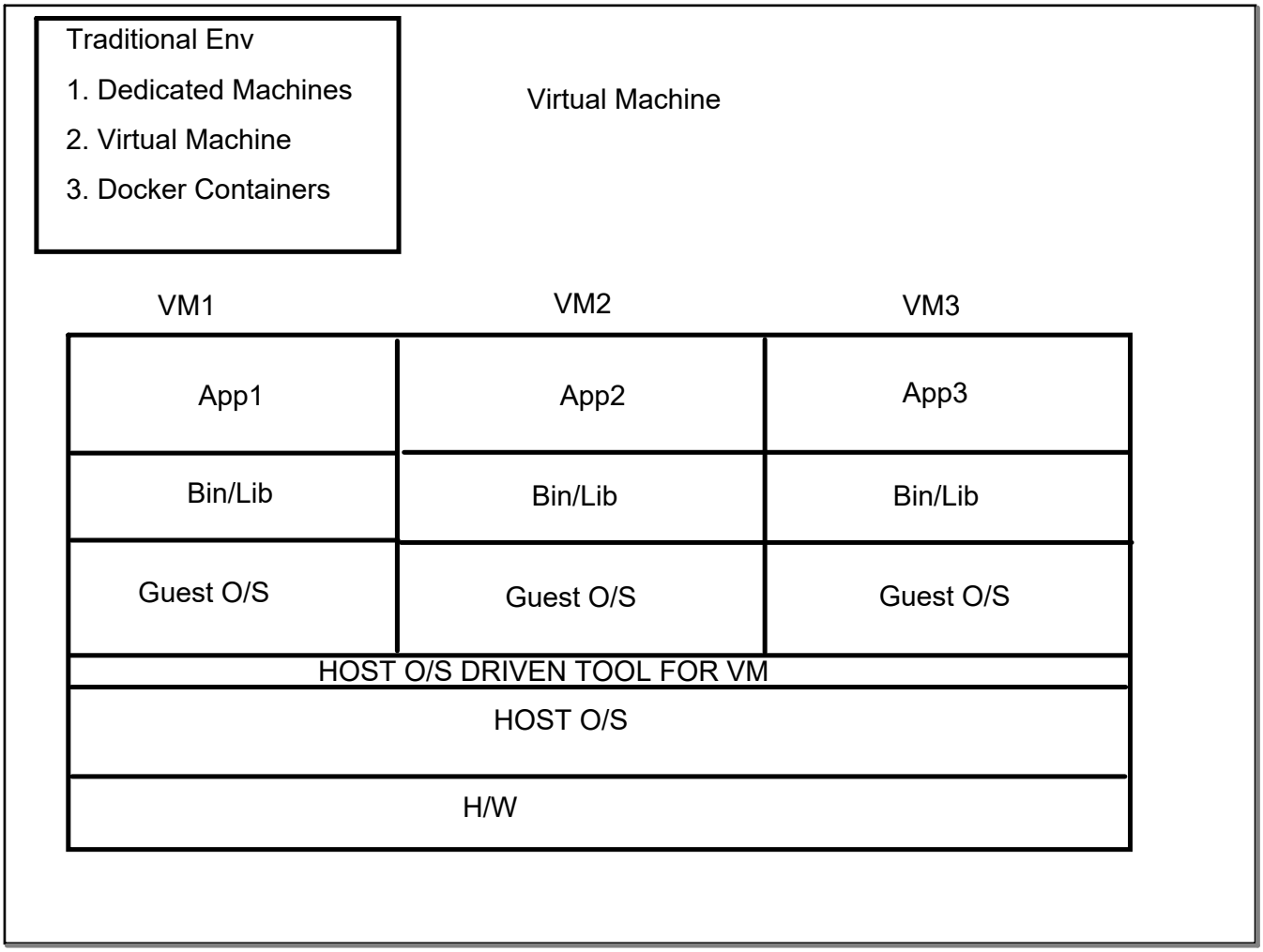
QA env

Testing env

Prod env

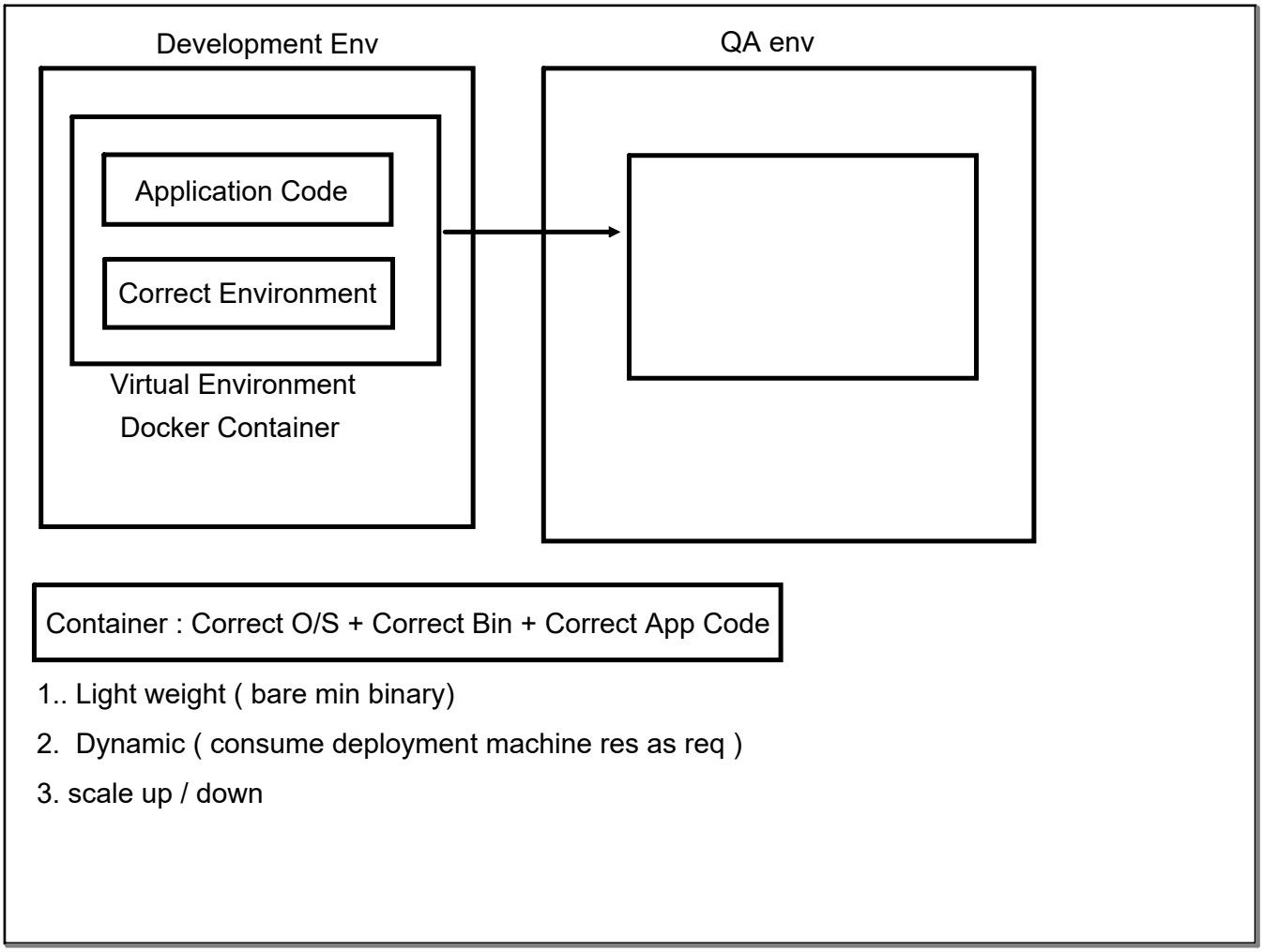
Problem : Application works on my machine (only)

want a Solution : develop, ship and run it anywhere without getting involved in config


Traditional Env
1. Dedicated Machines
2. Virtual Machine
3. Docker Containers
Virtual Machine
VM1
VM2
VM3
App1
App2
App3
Bin/Lib
Bin/Lib
Bin/Lib
Guest O/S
Guest O/S
Guest O/S
HOST O/S DRIVEN TOOL FOR VM
HOST O/S
H/W

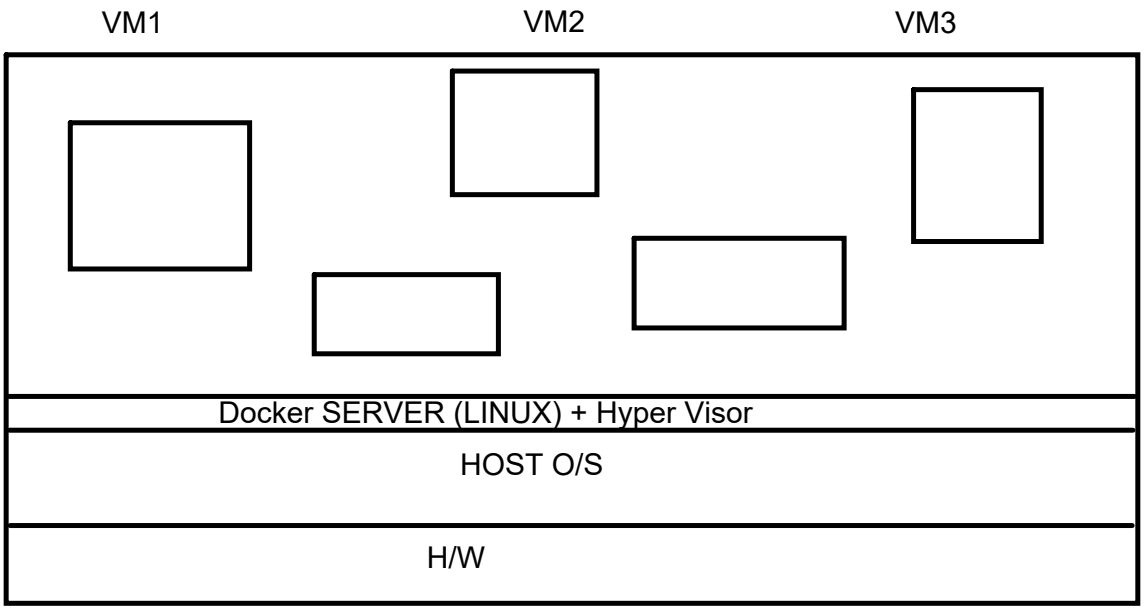
Challenges with VM

1. Resource intensive
2. Static in resource distribution
3. Can't dynamically scaled up/down
4. Need to update VM as per new requirement

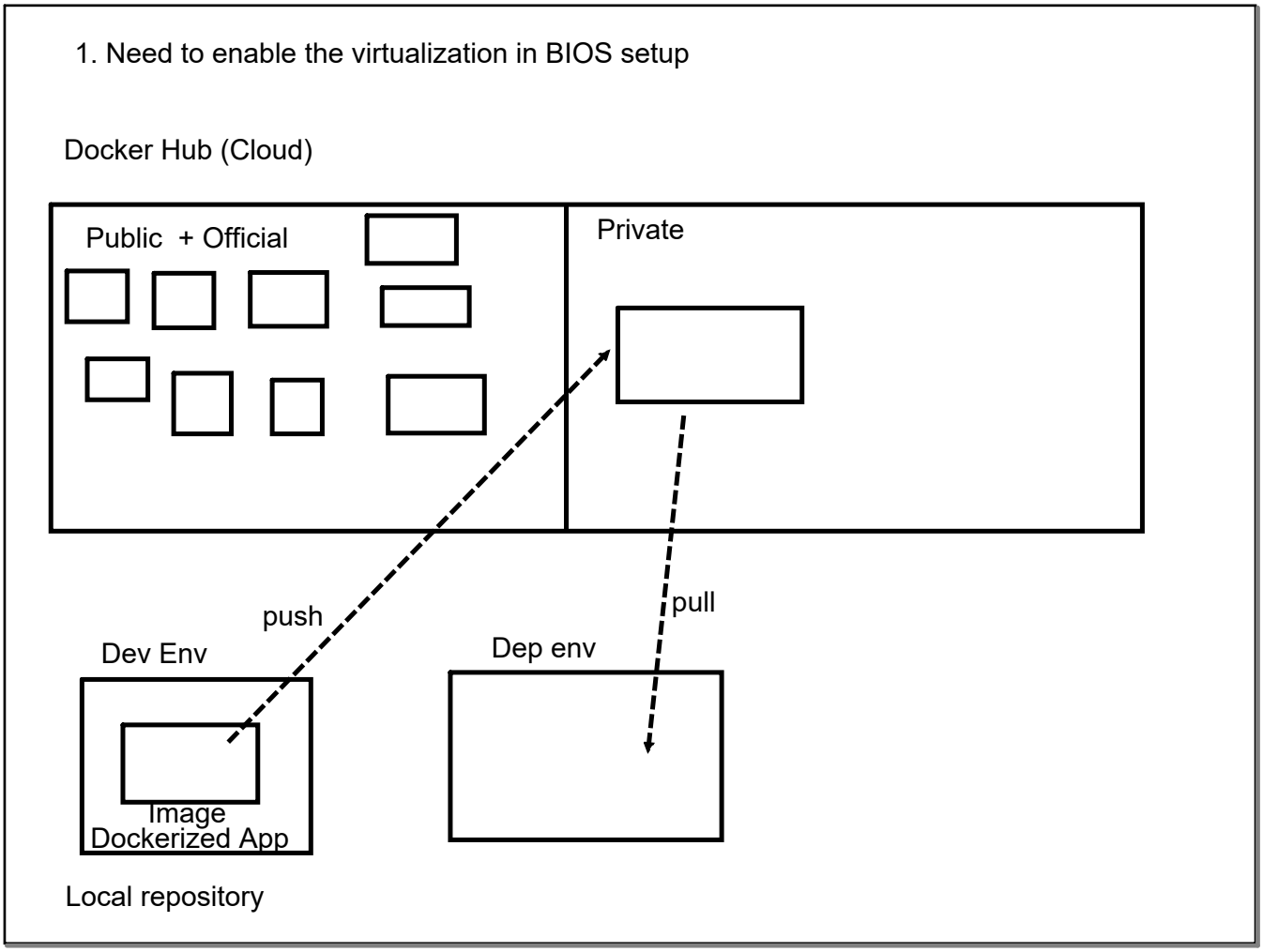

Development Env
QA env
Application Code
Correct Environment
Virtual Environment
Docker Container
Container : Correct O/S + Correct Bin + Correct App Code
1.. Light weight (bare min binary)
2. Dynamic (consume deployment machine res as req)
3. scale up / down

Docket Tool

- Server (to run docker container)
- Client (to dockerize an application)



1. Need to enable the virtualization in BIOS setup



==> docker images

lists all images in local repository

Image : blueprint(instruction to launch) + resources (organised environment way)

Container : runtime env

Create an image :

Create a manifest file: contains instruction what an image will contain

manifest file : "Dockerfile"

DOCKER COMMAND

FROM : which other Docker image we need to create our image file

LABEL : non-technical : add info

RUN : Run certain command to setup env (running bash command inside the env of Container)

eg : RUN mkdir ..

COPY <host machine> <container vm>

Microservice :

1. jdk setup with Linux O/S (official docker image)
2. jar file of our application

Launch MS : java -jar <jar>

Build a Docker image

==> docker build -t <image-name> <location of Dockerfile>

while we launch container :

map the internal port to any external port number

Run a container

docker container run -p <external> : <internal> <image-name>

=> docker container ls

command to list running container