Terminologies

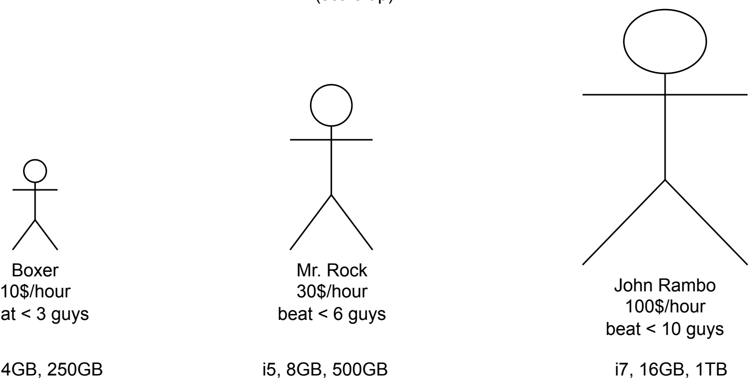
1. Scaling (Scale-up, Scale-out, Scale-in, Vertical Scaling, Horizontal Scaling)
2. Stateless/Stateful

1. Memory (RAM)
2. CPU
3. Disc/Storage
4. NIC (Network Interface Card)

Swayam is a celebrity

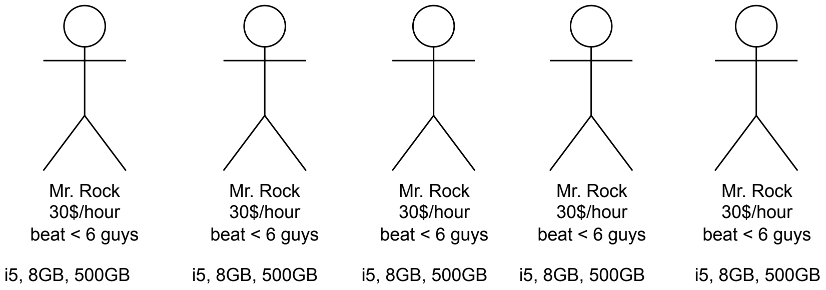
Vertical Scaling

(scale up)



Horizontal Scaling

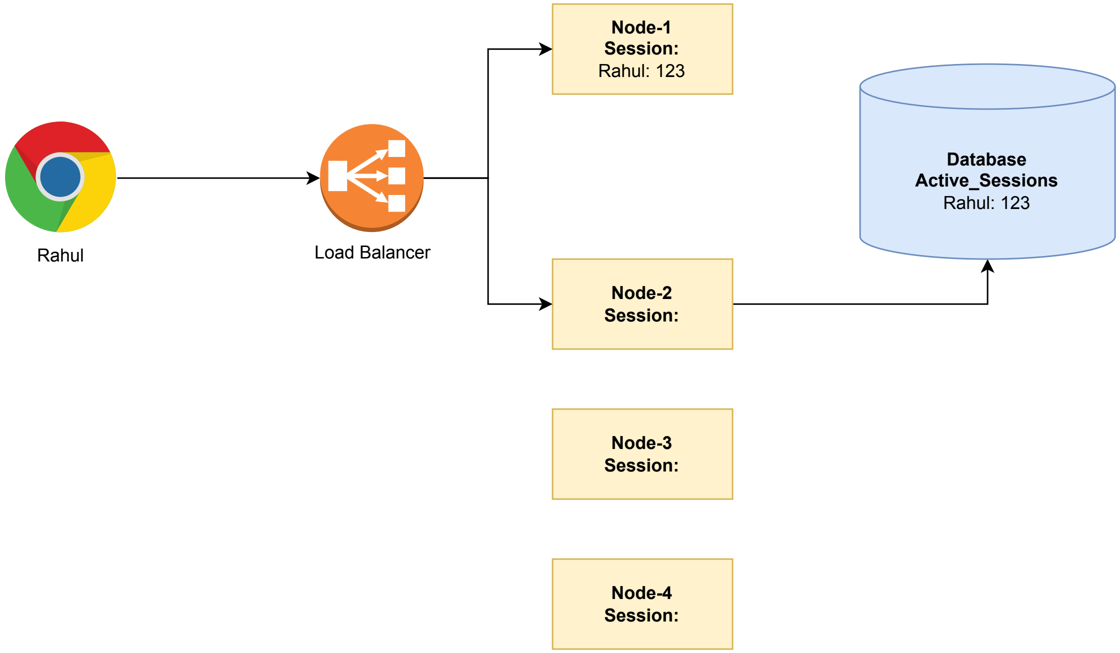
(scale out / scale in)



Myecommerce Monoilth Application

(Stateful)

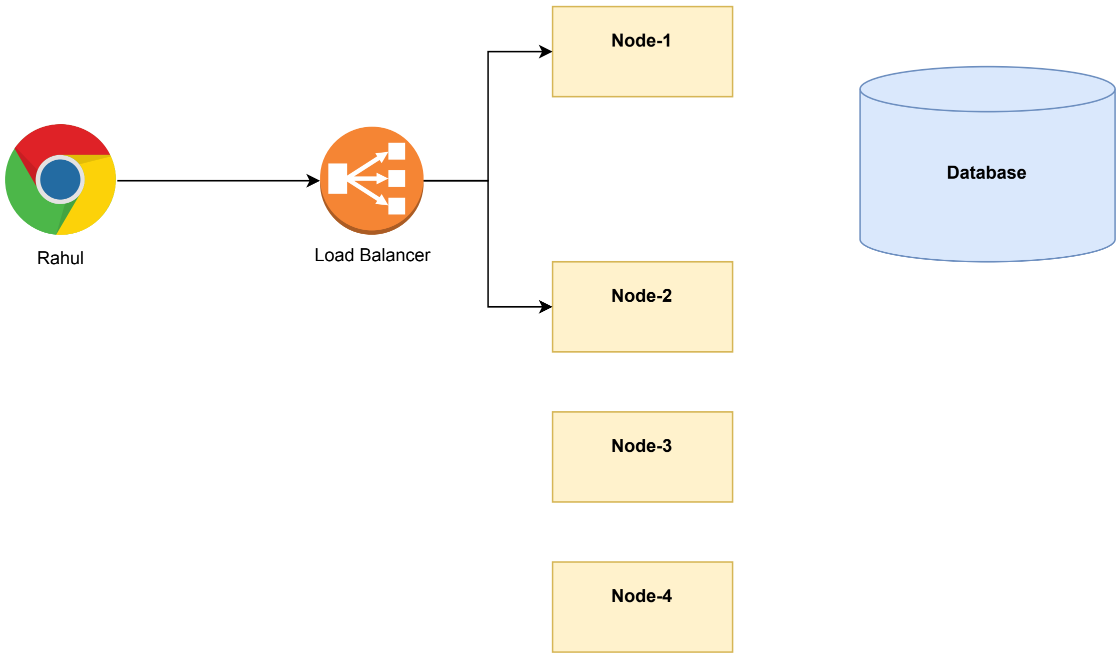
F5, HA Proxy, NGINX, Apache Http Server



Myecommerce Monoilth Application

(Stateless)

F5, HA Proxy, NGINX, Apache Http Server



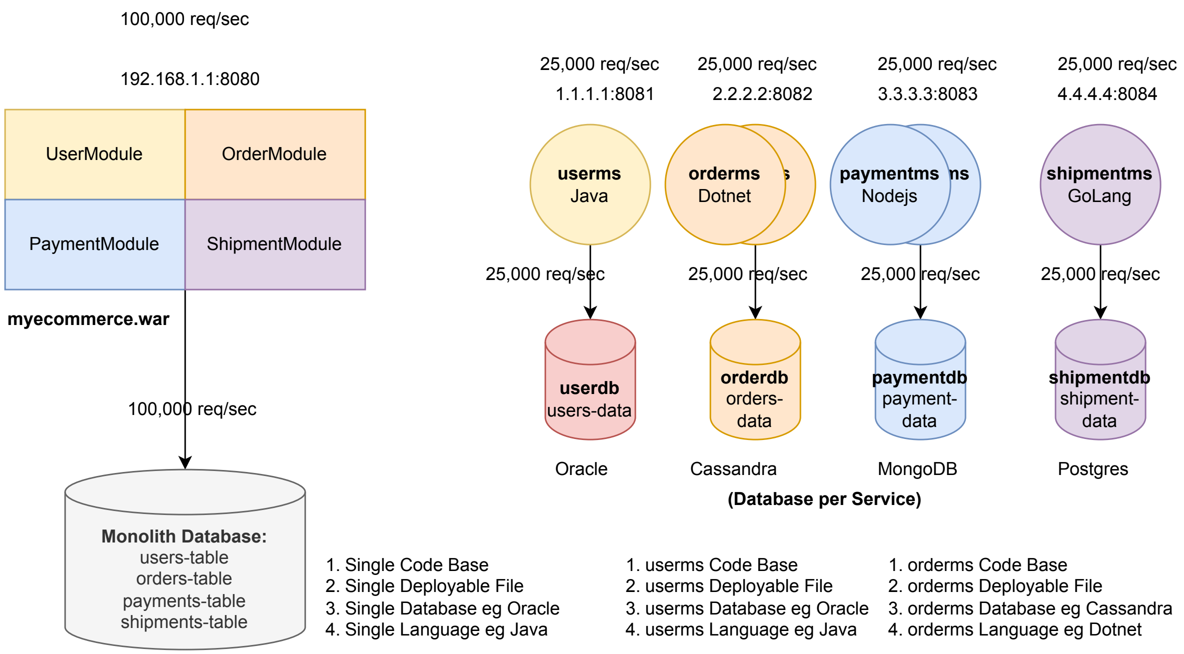
Black Friday / Diwali Sale Season

100 microservice x 5 instances = 500 instances

Myecommerce Monolith Application

Microservice Application

(Collection of standalone miniature applications)



(Database per Service)

1. Single Code Base
2. Single Deployable File
3. Single Database eg Oracle
4. Single Language eg Java

1. userms Code Base
2. userms Deployable File
3. userms Database eg Oracle
4. userms Language eg Java

1. orderms Code Base
2. orderms Deployable File
3. orderms Database eg Cassandra
4. orderms Language eg Dotnet

Pros of Microservices:

1. Cherry-pick scaling
2. Agility-1: Development is fast
3. Agility-2: Build is fast
4. Agility-3: Testing is fast
5. Agility-4: CI/CD is fast
6. Agility-5: Release is fast
7. Resiliency
8. Distributed Service Load
9. DIstributed DB Load
10. Technology Hetrogenity
11. Database Hetrogenity
12. Security (Segregation)

Cons of Microservice:

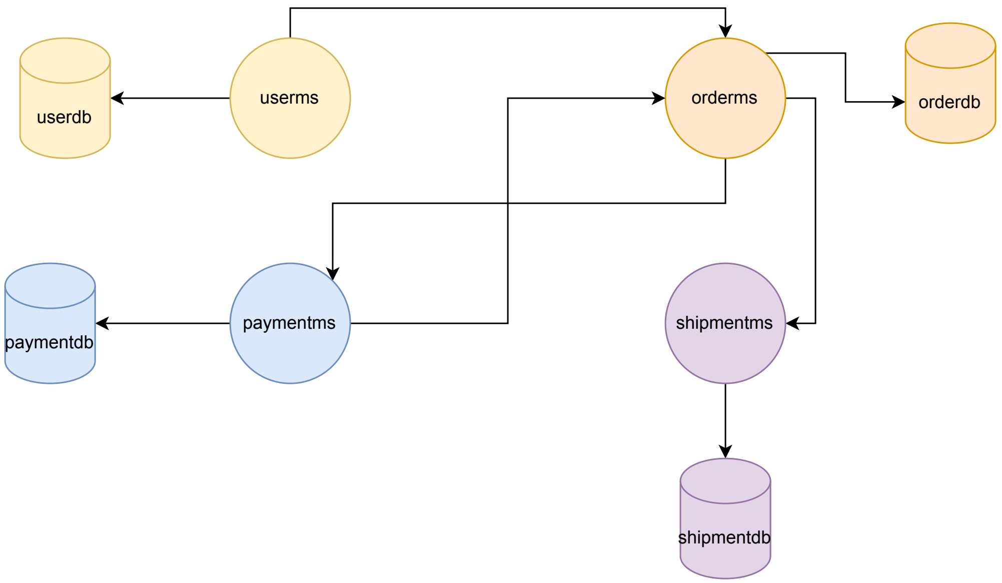
1. Latency between Microservices calls
2. Distributed Database (Aggregation/TxManagement)
3. Complexity in managing Nodes (services + DB)
4. Cost (Infra + Resources)

2-Pizza Team: Team size should be small

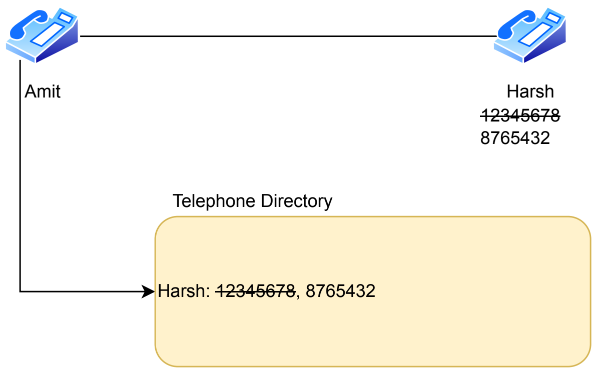
Microservice-to-MIcroservice Communication

RESTful services
Messaging: ActiveMQ, RabbitMQ, IBM MQ, Solace Pub/Sub, Kafka

RESTful services: RestTemplate
WebClient
FeignClient



Service Discovery

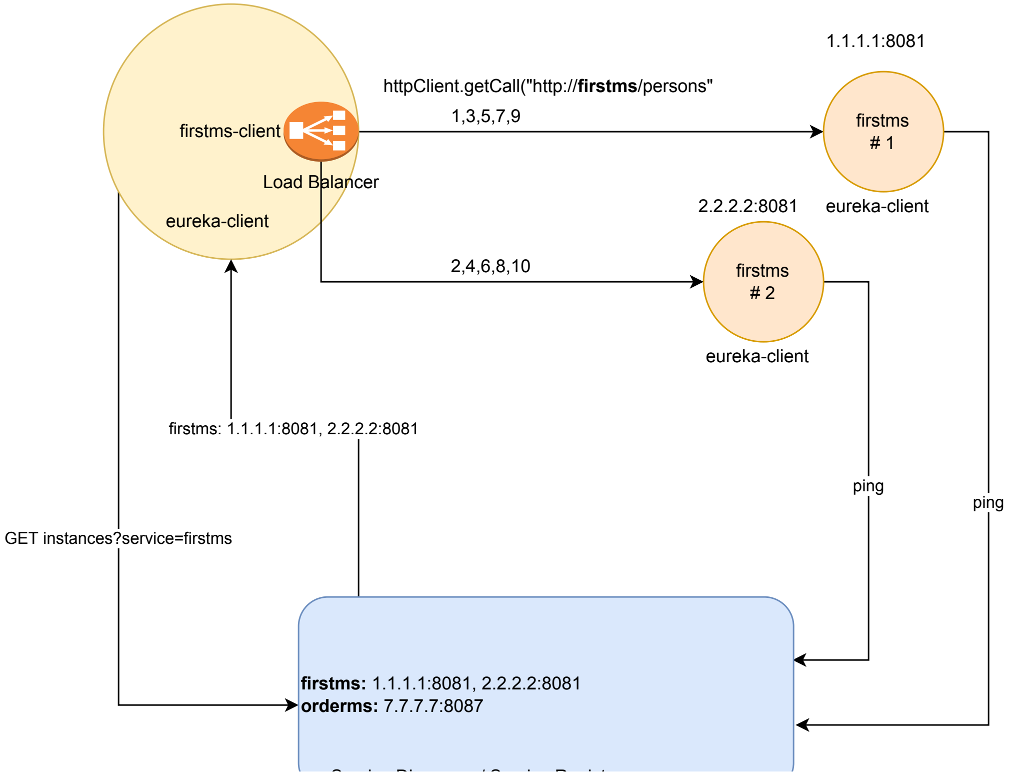


Netflix Eureka Server, Consuul, etc, Apache Zookeeper

100 microservices

Round Robin

cilent-side Load Balancer (Netflix Ribbon, Spring Cloud Load Balancer)



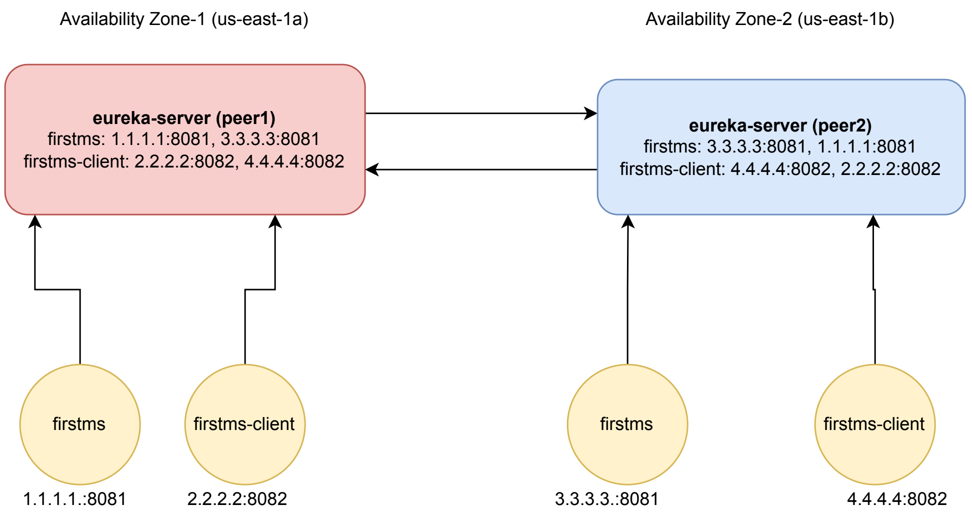
firstms: 1.1.1.1:8081, 2.2.2.2:8081
orderms: 7.7.7.7:8087

Service Discovery / Service Registry

eureka-server

Day-2

Eureka Peer Aware

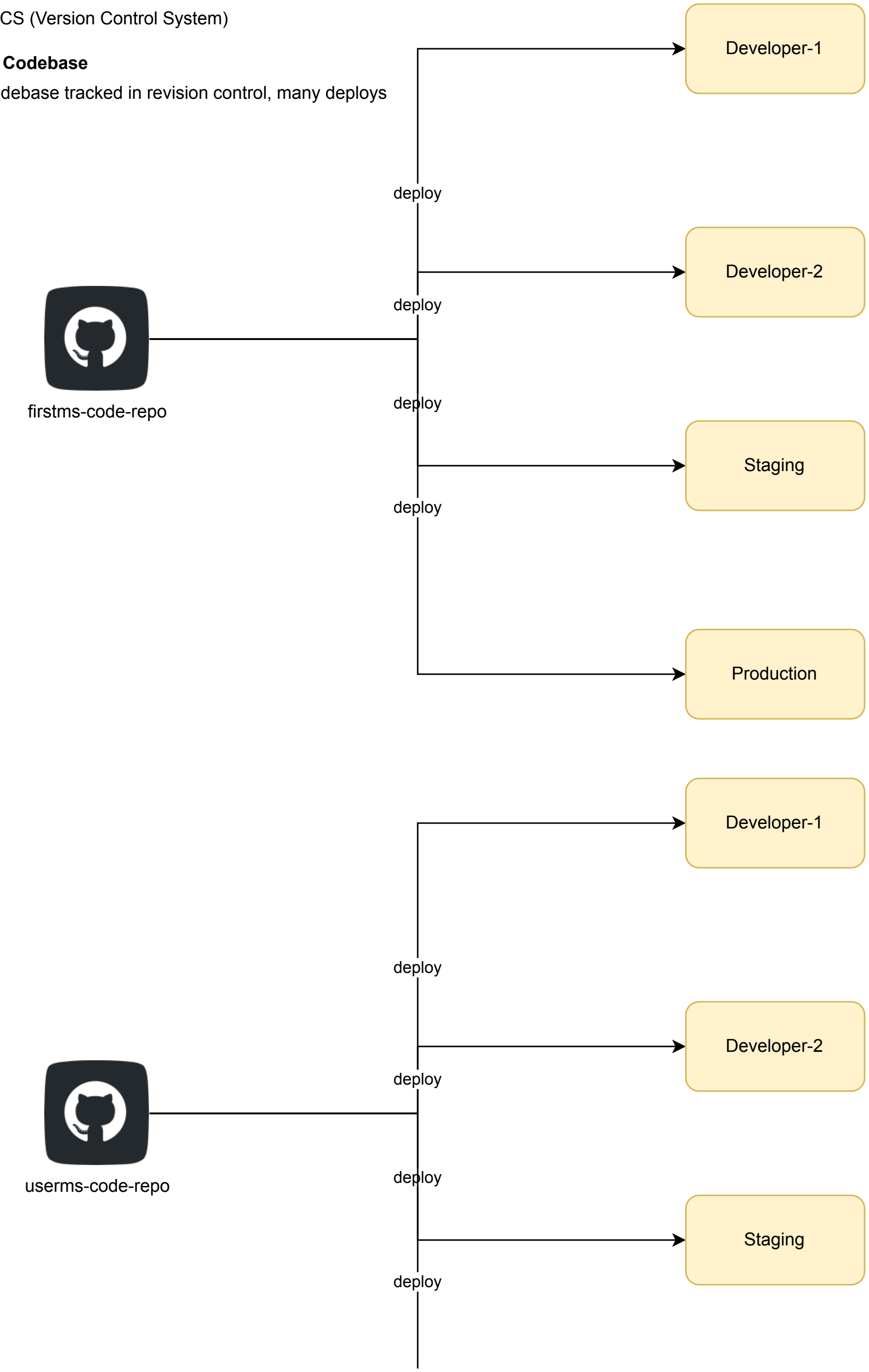

Availability Zone-1 (us-east-1a)
Availability Zone-2 (us-east-1b)
eureka-server (peer1)
firstms: 1.1.1.1:8081, 3.3.3.3:8081
firstms-client: 2.2.2.2:8082, 4.4.4.4:8082
eureka-server (peer2)
firstms: 3.3.3.3:8081, 1.1.1.1:8081
firstms-client: 4.4.4.4:8082, 2.2.2.2:8082
firstms
firstms-client
firstms
firstms-client
1.1.1.1.:8081
2.2.2.2:8082
3.3.3.3.:8081
4.4.4.4:8082

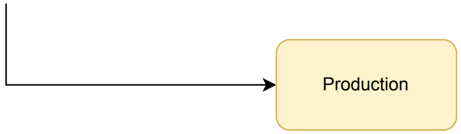
12 Factor App

VCS (Version Control System)

I. Codebase

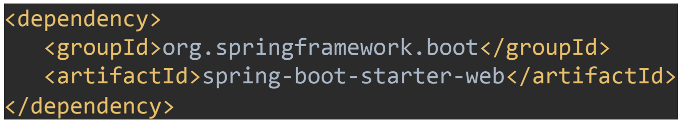
One codebase tracked in revision control, many deploys




Production

II. Dependencies

Explicitly declare and isolate dependencies

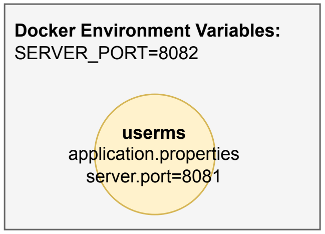


```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

III. Config

Store in the environment

Environment specific properties are supplied during deployment and thus easier, faster deployment without code changes



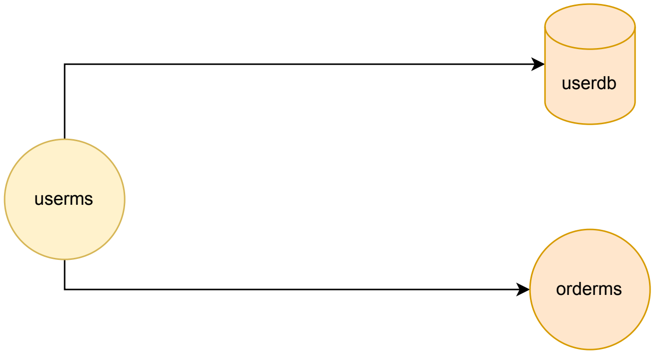
Docker Environment Variables:
SERVER_PORT=8082

userms
application.properties
server.port=8081

java -jar --server.port=8083
docker run -e SERVER_PORT=8084 userms-docker-image

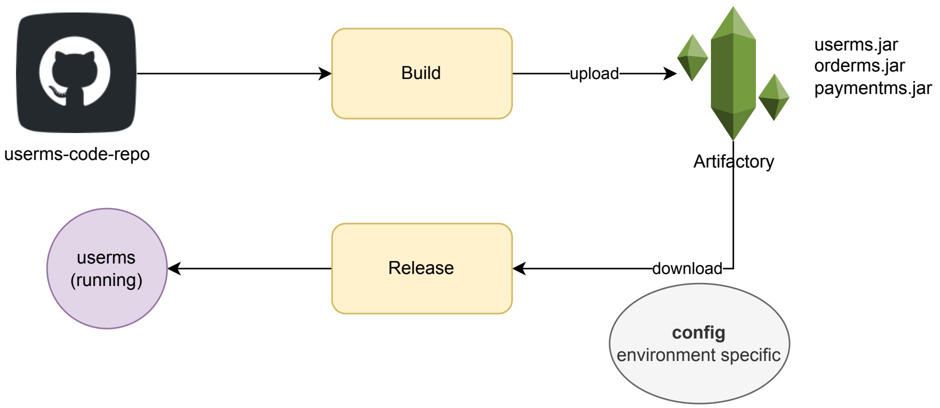
IV. Backing Services

Treat backing services as attached resource



V. Build, Release, Run

Strictly separate build and run stages



userms-code-repo

Build

upload

Artifactory

userms.jar
orderms.jar
paymentms.jar

download

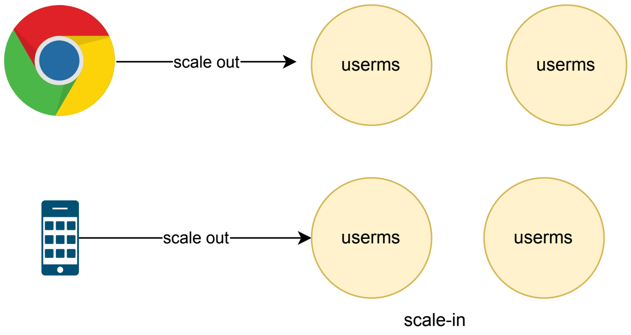
Release

userms (running)

config
environment specific

VI. Processes

Execute the app as one or more stateless processes



scale out

userms

userms

scale out

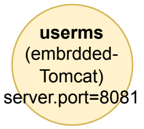
userms

userms

scale-in

VII. Port Binding

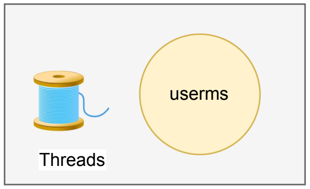
Export services via Port Binding



userms
(embrdded-Tomcat)
server.port=8081

VIII. Concurrency

Scale out via the process model



IX. Disposability

Maximize robustness with fast startup and graceful shutdown

Fast startup is for quick scaling out.
Graceful shutdown is to keep the application in steady state.

X. Dev/Prod Parity

Keep development, staging and Production as similar as possible

Dev Env:

Container(Docker):
userms: SpringBoot Jar + Java-8
orderms: SpringBoot Jar + Java-11

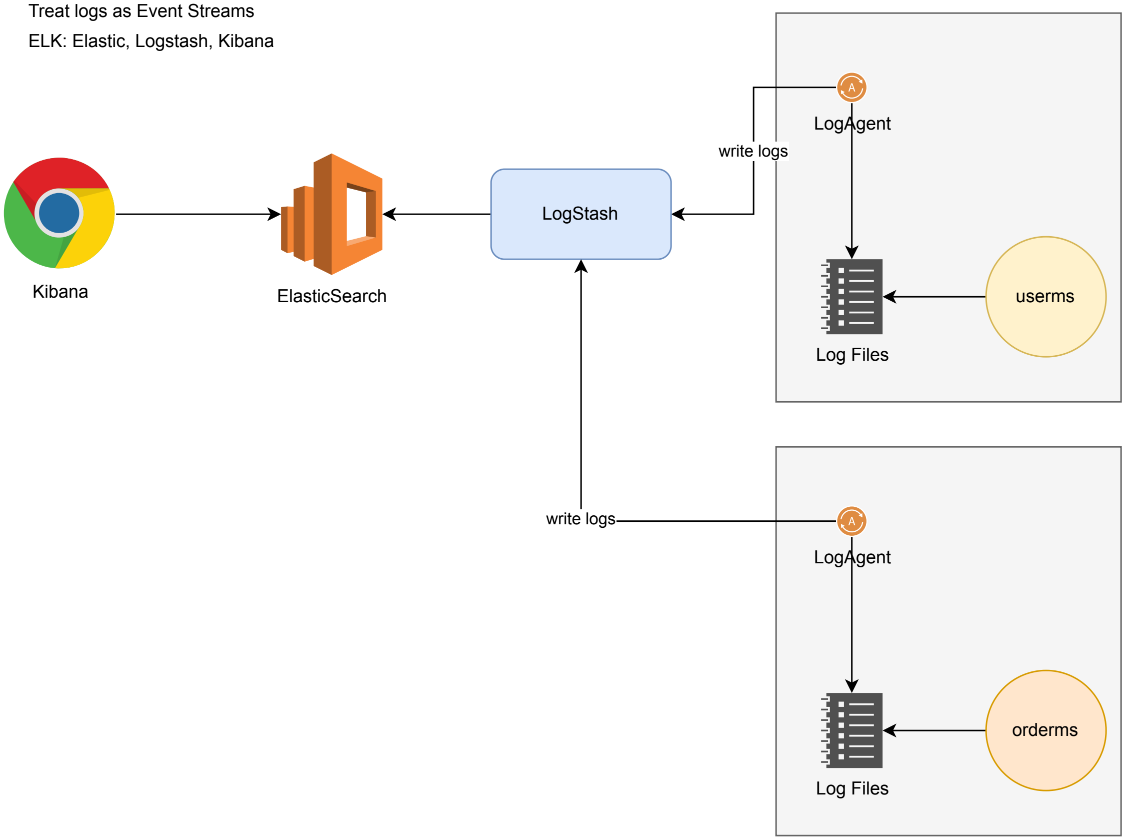
Prod Env:

Container(Docker):
userms: SpringBoot Jar + Java-8
orderms: SpringBoot Jar + Java-11

XI. Logs

Treat logs as Event Streams

ELK: Elastic, Logstash, Kibana



XII) Admin Processes

Run admin/management tasks as one-off processes
the script, the APIs. these all should be part of my code

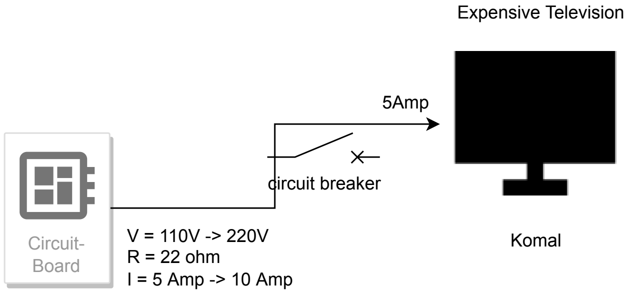
userms-code-repo

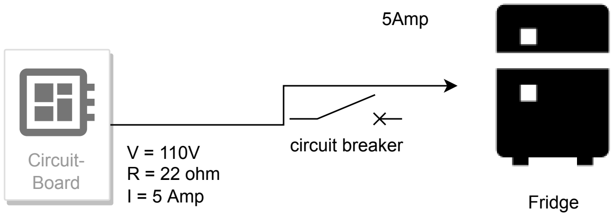
userms-code
Management DB-Scripts
Management APIs

Circuit Breaker

(Resiliency)

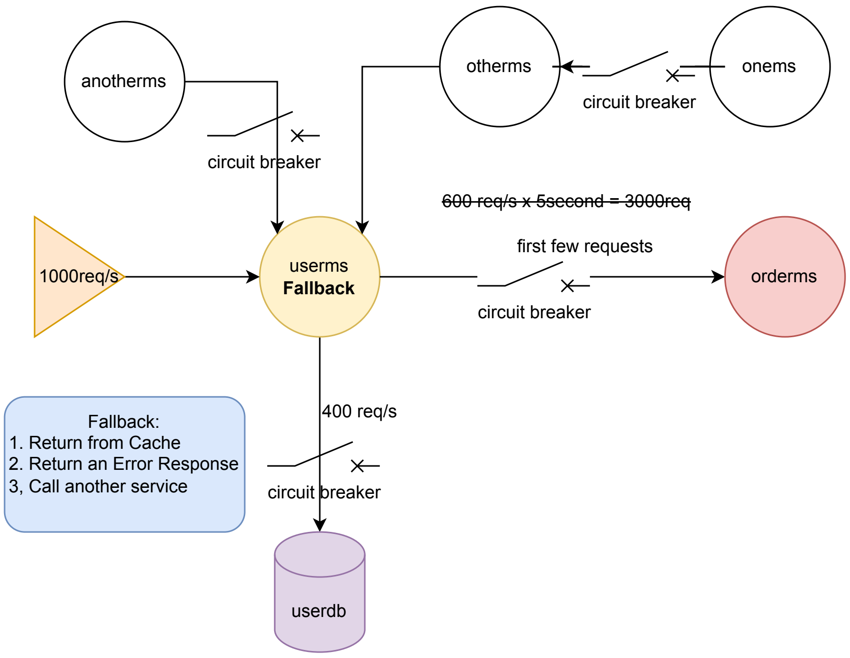
MCB: Miniature Circuit Breaker





Circuit Breaker

(Resiliency)



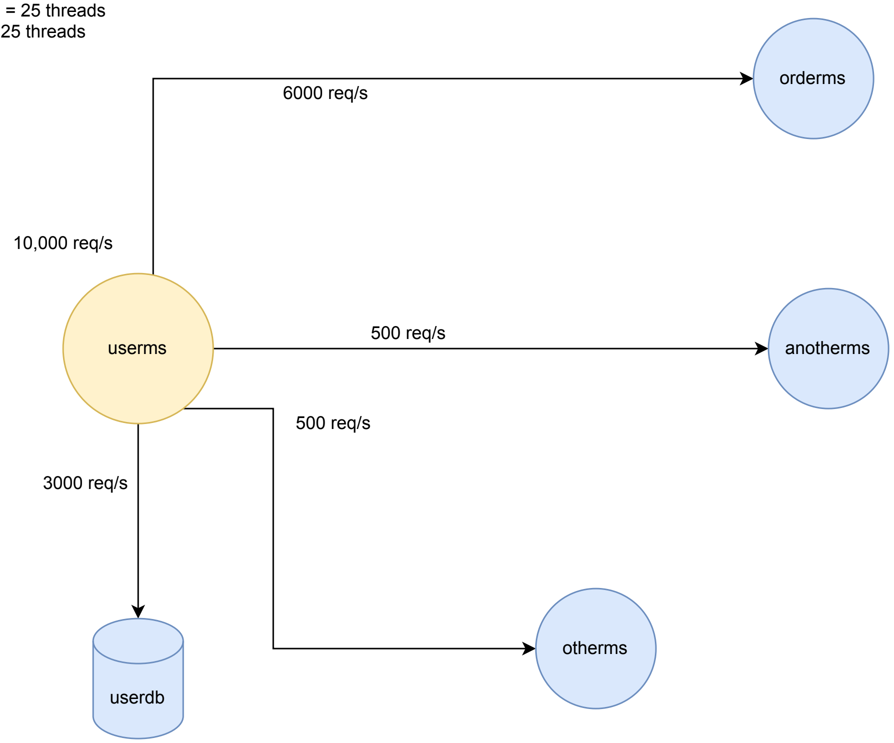
Fallback:
1. Return from Cache
2. Return an Error Response
3, Call another service

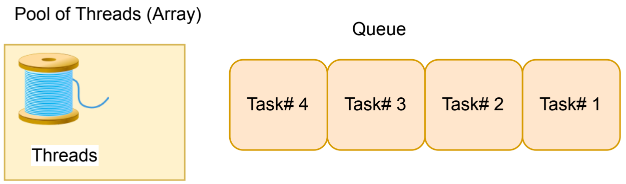
tota
ord
use
anc
oth

Day-3

Bulkhead

l threads = 200
erms = 60 threads
rdb = 40 threads
therms = 25 threads
ersm = 25 threads





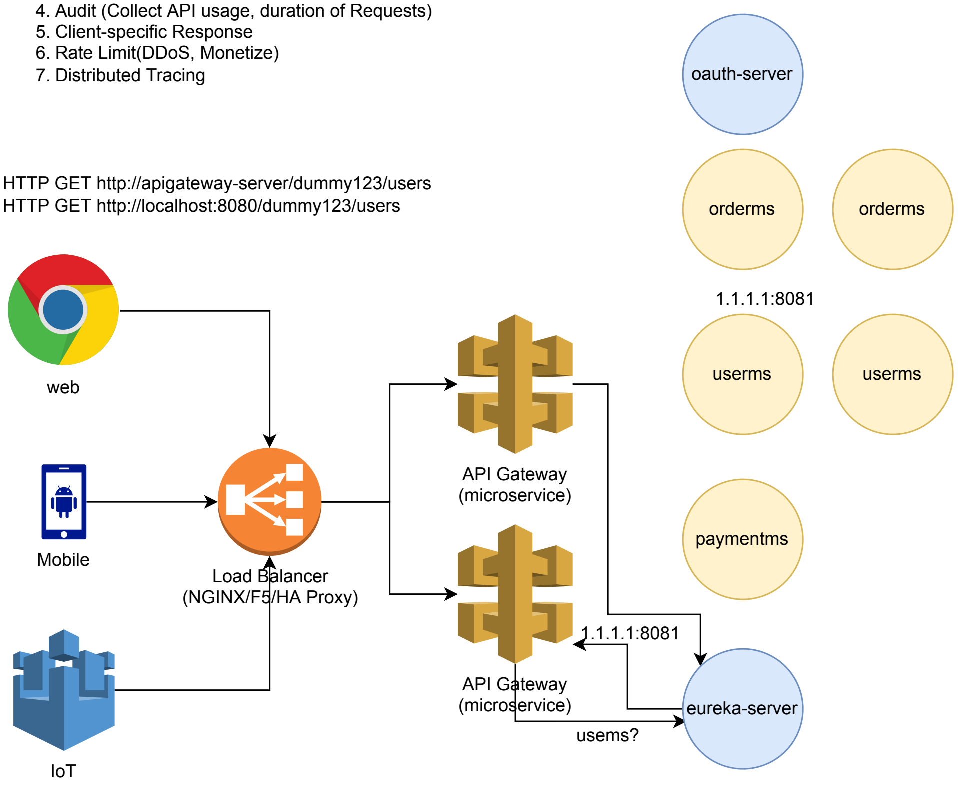
API Gateway (Netflix Zuul or Spring Cloud Gateway)

100 microservices x 5 instances = 500 instances

Cross Cutting Concerns:

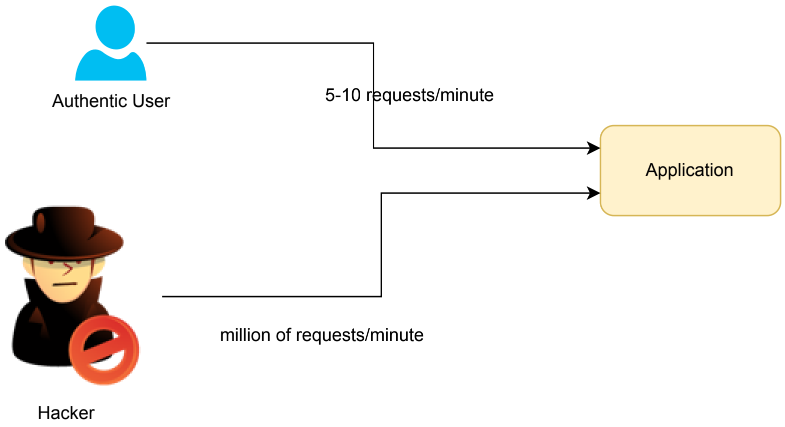
1. Security (Authentication/Authorization)
2. Security - URL Hiding
3. Proxy
4. Audit (Collect API usage, duration of Requests)
5. Client-specific Response
6. Rate Limit(DDoS, Monetize)
7. Distributed Tracing

HTTP GET http://apigateway-server/dummy123/users
HTTP GET http://localhost:8080/dummy123/users



/dummy123/** -> userms/**
/dummy456/** -> orderms/**
dummy789/** -> paymentms/**

DoS (Denial of Service) & Rate Limit

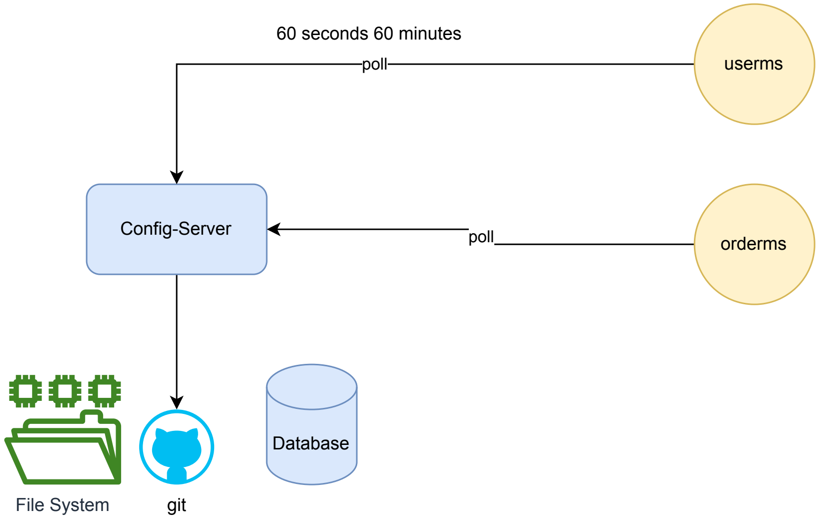


Rate Limit:

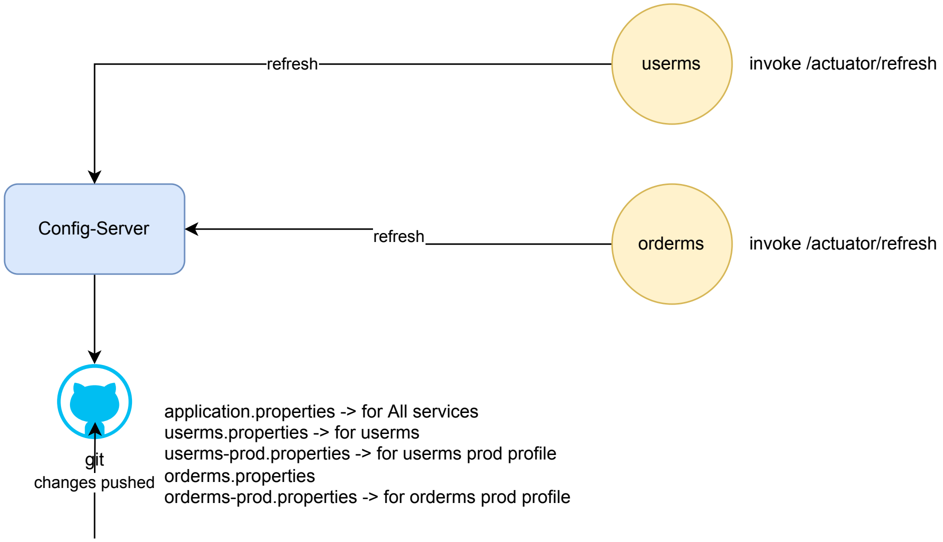
Free: 60 seconds - 10 calls
Paid:
60 seconds: 10,000 requests - 10$
60 seconds: 20,000 requests - 20$
60 seconds: 100,000 requests: 50$

Config-Server

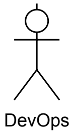
1) Config Server (poll)



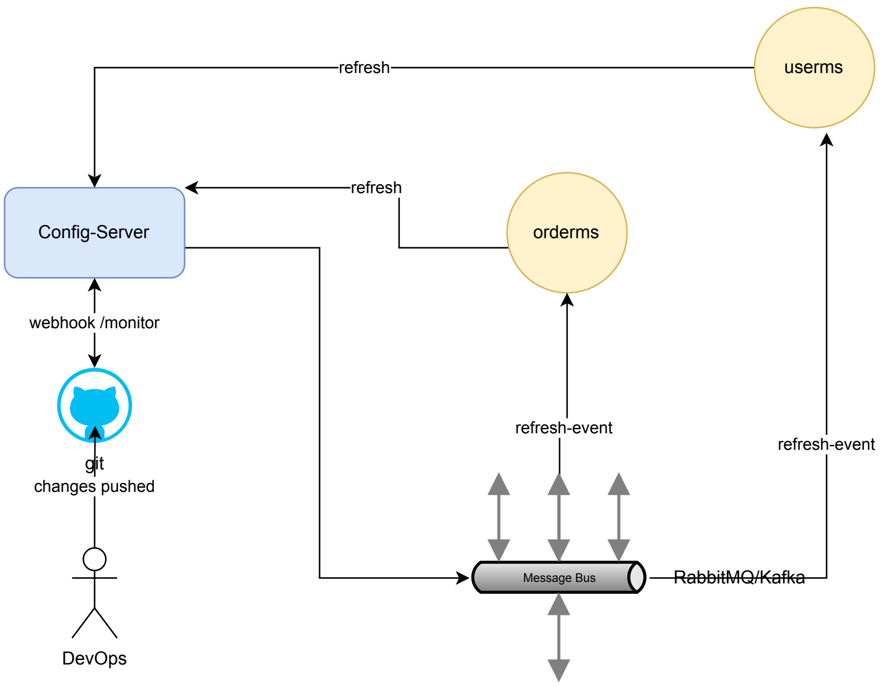
2) Config Server (/actuuator/refresh)



application.properties -> for All services
userms.properties -> for userms
userms-prod.properties -> for userms prod profile
orderms.properties
orderms-prod.properties -> for orderms prod profile

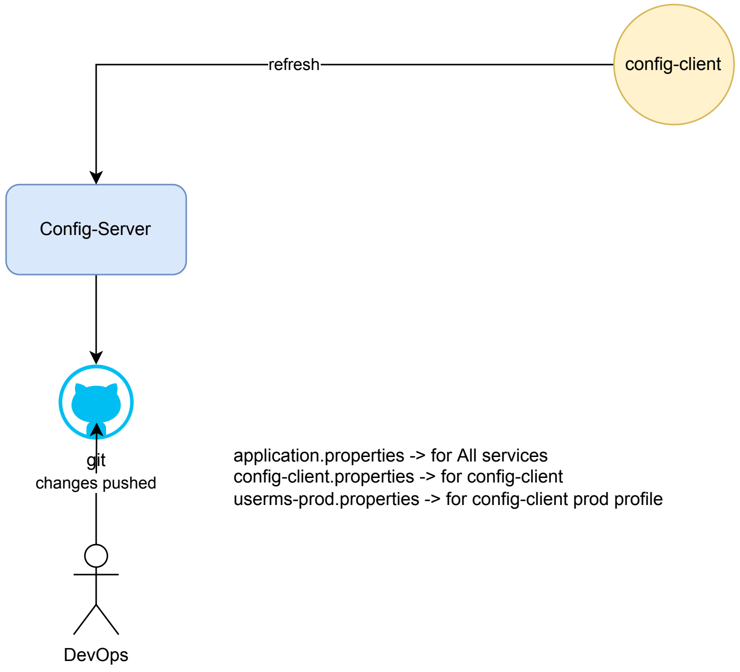

DevOps

Config Server (/bus-refresh)



application.properties -> for All services
userms.properties -> for userms
userms-prod.properties -> for userms prod profile
orderms.properties
orderms-prod.properties -> for orderms prod profile

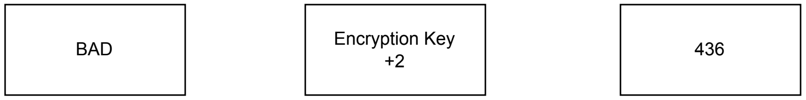
Config-Server Handson



invoke /actuator/refresh

application.properties -> for All services
config-client.properties -> for config-client
userms-prod.properties -> for config-client prod profile

BAD = 214



Config Server Handson
(Message Bus)

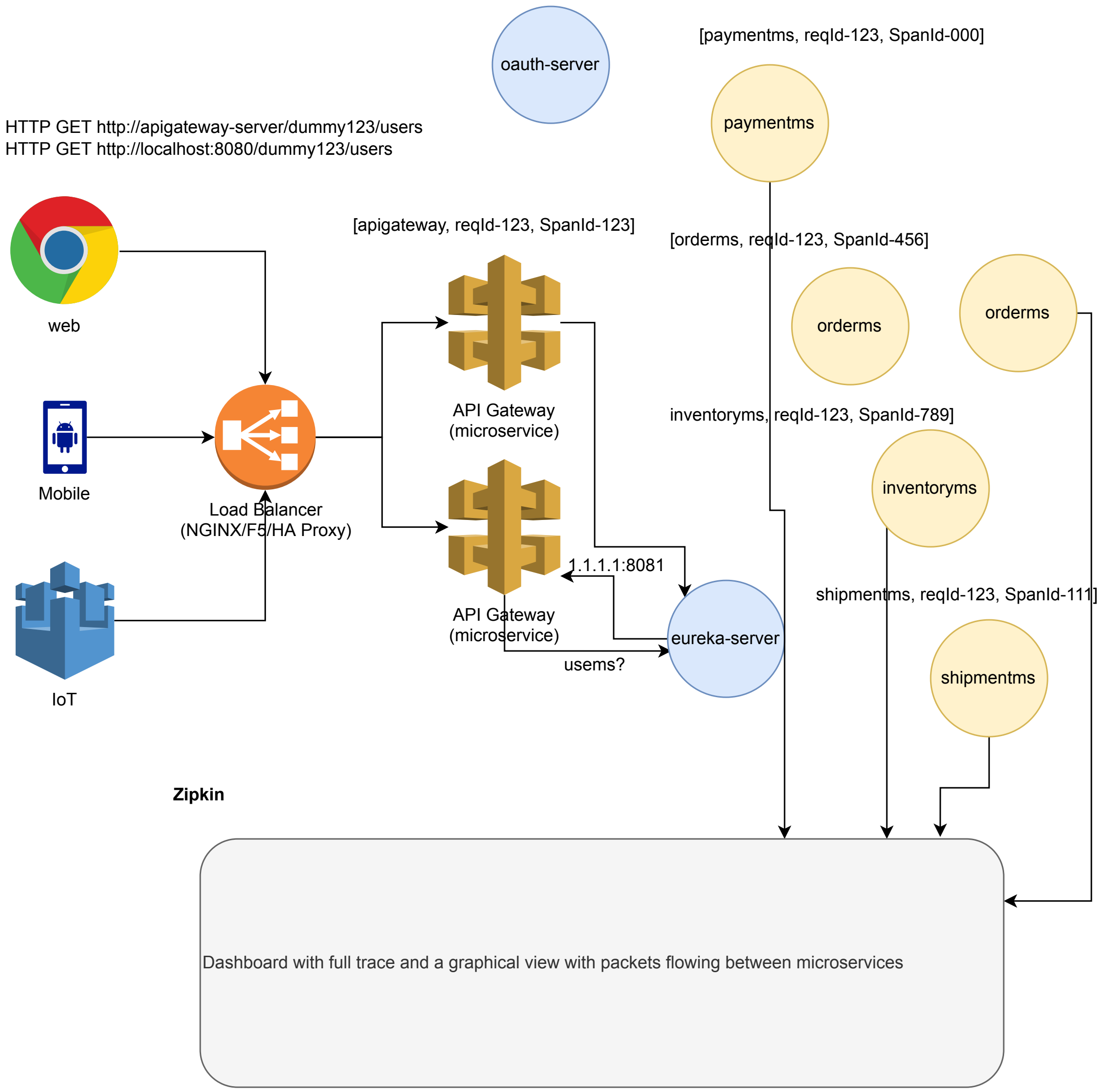
config-server
config-client
eureka-server
rabbitmq

Distributed Tracing

(Sleuth + Zipkin)

Three pillars of Observability:

1. Distributed Tracing
2. Centralized Logging
3. Metrics (Actutator)



HTTP GET http://apigateway-server/dummy123/users
HTTP GET http://localhost:8080/dummy123/users

Dashboard with full trace and a graphical view with packets flowing between microservices