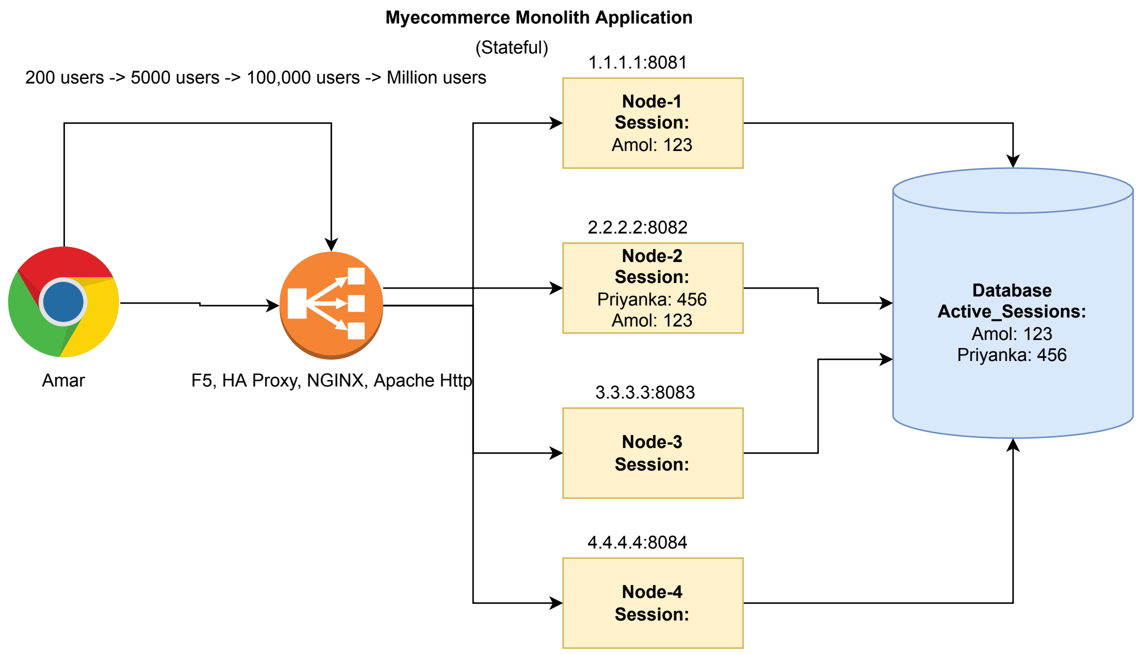
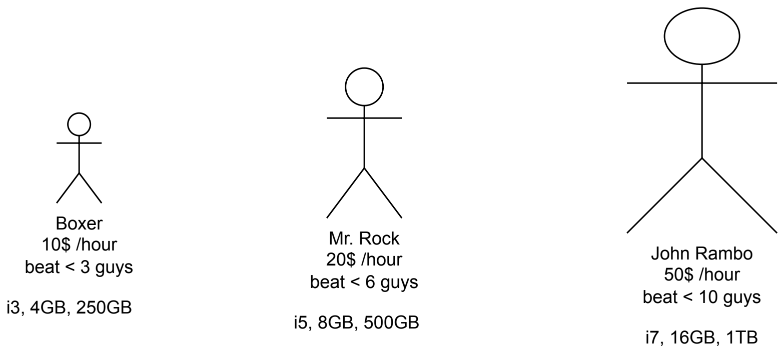
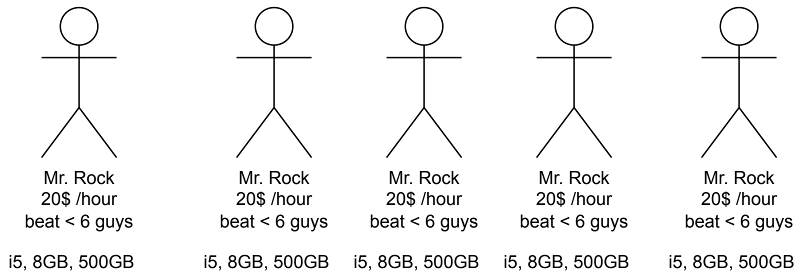

Myecommerce Monolith Application
(Stateful)
200 users -> 5000 users -> 100,000 users -> Million users
Amar
F5, HA Proxy, NGINX, Apache Http
1.1.1.1:8081
Node-1
Session:
Amol: 123
2.2.2.2:8082
Node-2
Session:
Priyanka: 456
Amol: 123
3.3.3.3:8083
Node-3
Session:
4.4.4.4:8084
Node-4
Session:
Database
Active_Sessions:
Amol: 123
Priyanka: 456

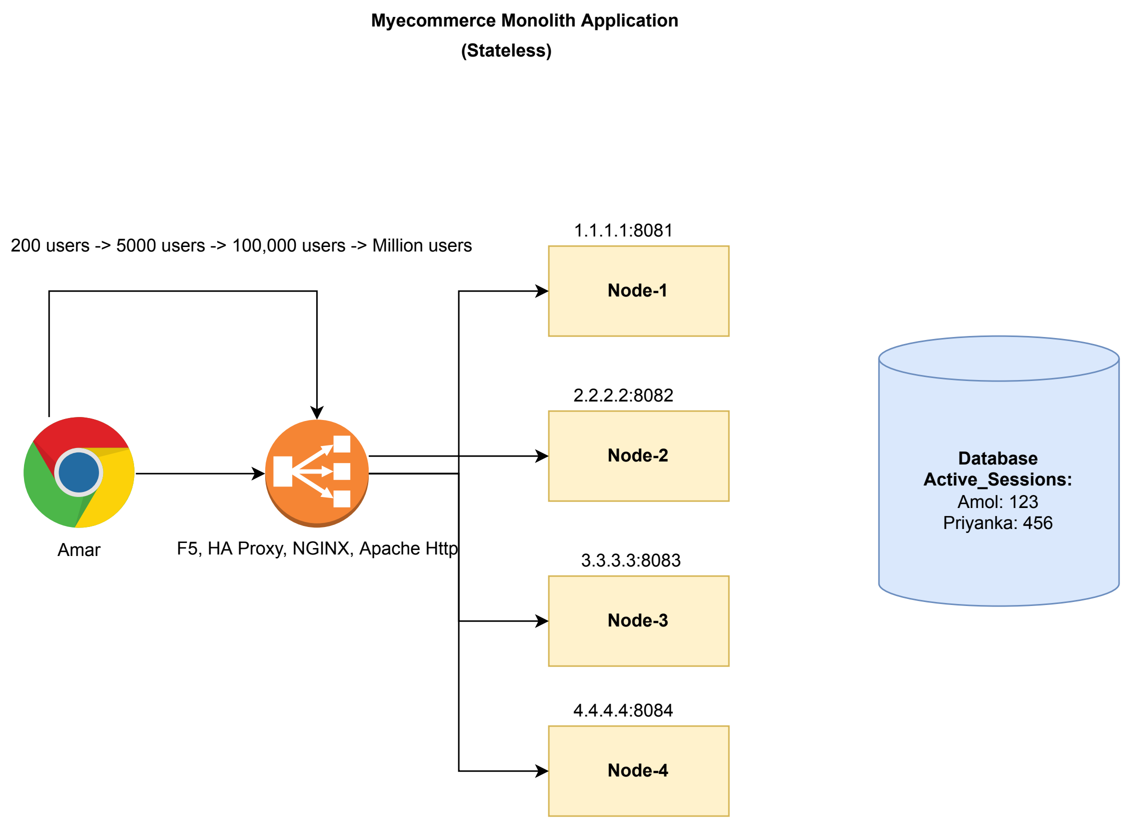
Scaling:
Vertical Scaling: scale up
Horizontal Scaling: scale out

Vertical Scaling



Horizontal Scaling




Myecommerce Monolith Application
(Stateless)
200 users -> 5000 users -> 100,000 users -> Million users
Amar
F5, HA Proxy, NGINX, Apache Http
1.1.1.1:8081
Node-1
2.2.2.2:8082
Node-2
3.3.3.3:8083
Node-3
4.4.4.4:8084
Node-4
Database
Active_Sessions:
Amol: 123
Priyanka: 456

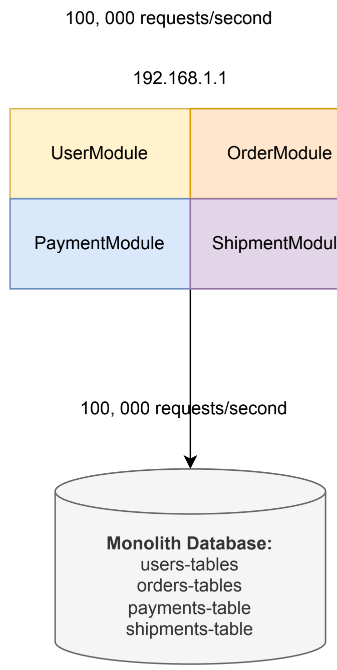
Black Friday/Diwali Sale Season

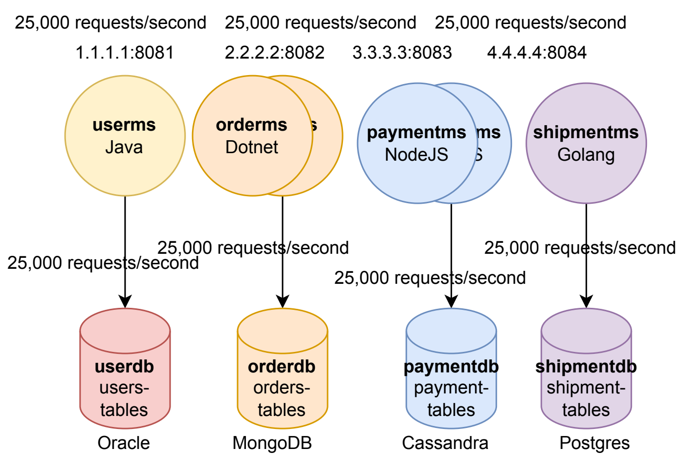
100 microservice x 5 instances = 500 instances

Myecommerce Monolith Application

Microservice Application

(Collection of standalone miniature applications)





(Database per Service)

1. Single Code Base	1. userms Code Base	1. orderms Code Base
2. Single Deployable File	2. userms Deployable File	2. orderms Deployable File
3. Single Database	3. userms Database ie Oracle	3. orderms Database ie MongoDB
4. Single Language ie Java	4. userms Language ie Java	4. orderms Language ie Dotnet

Pros of Microservices:

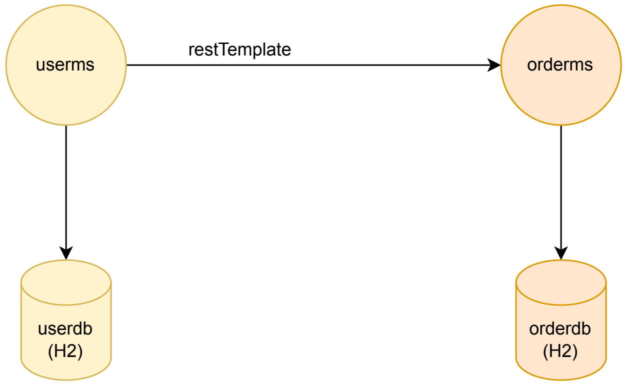
1. Cherry-pick scaling
2. Agility-1: Development is fast
3. Agility-2: Build is fast
4. Agility-3: Testing is fast
5. Agility-4: CI/CD is fast
6. Agility-5: Release is fast
7. Resiliency
8. Distributed Service Load
9. Distributed DB Load
10. Security (Segregation)
11. Technology Hetrogenity
12. DB Hetrogenity

Cons of Microservices:

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

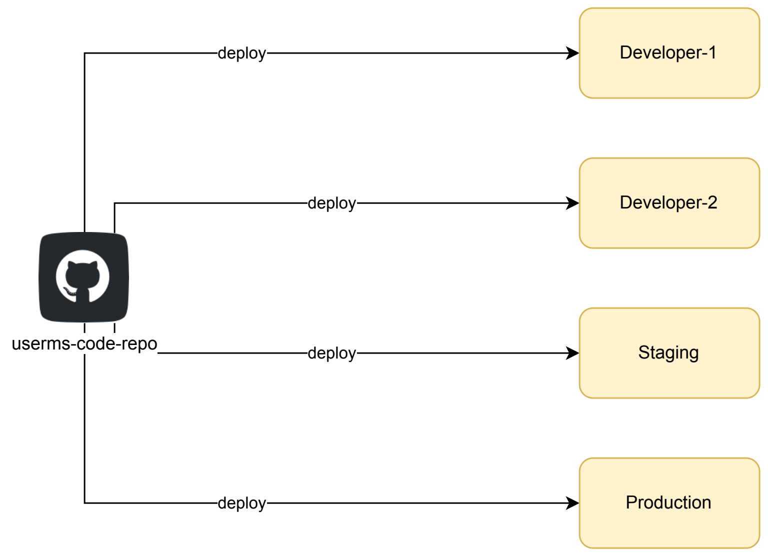
Microservice-to-Microservice Communication

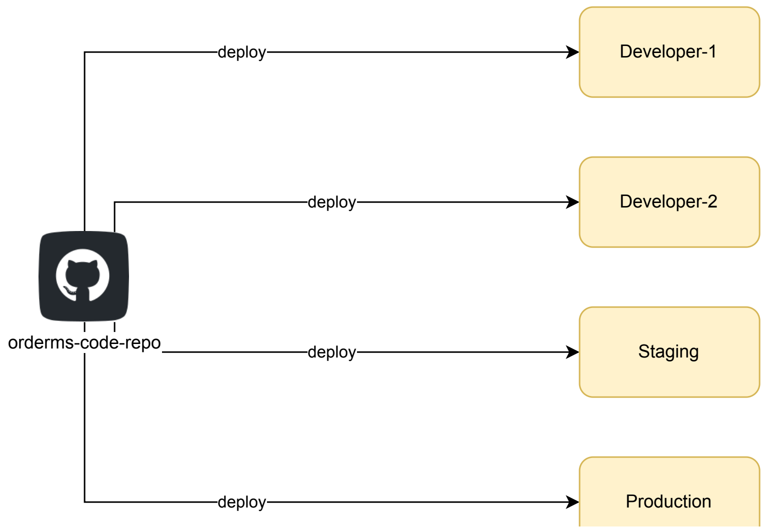


Day-2 **12 Factor App**

I. Codebase

One codebase tracked in revision control, many deploys





II. Dependencies

Explicitly declare and isolate dependencies

maven: pom.xml

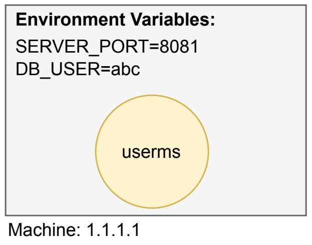
```xml
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

III. Config

Store config in the environment

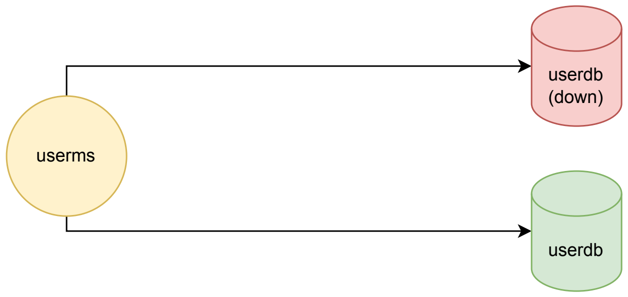
Environment specific properties are supplied during deployment and thus faster and easier deployment without any code change.

```
java -jar userms.jar --server.port=8081 --db.user=abc
```

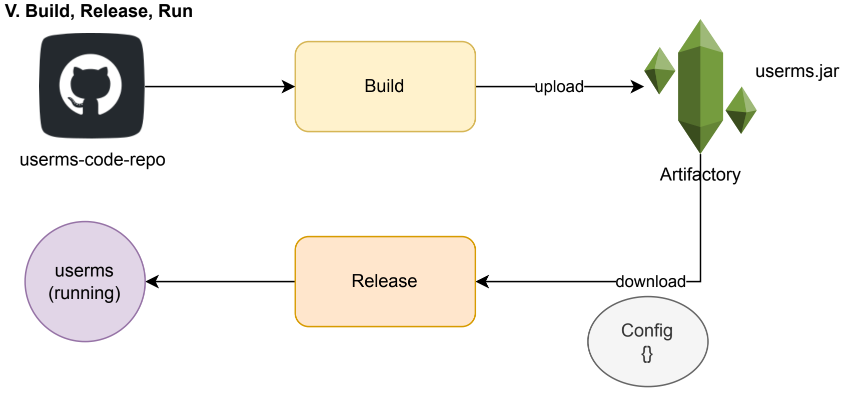


IV. Backing Services

Treat backing service as attached resource

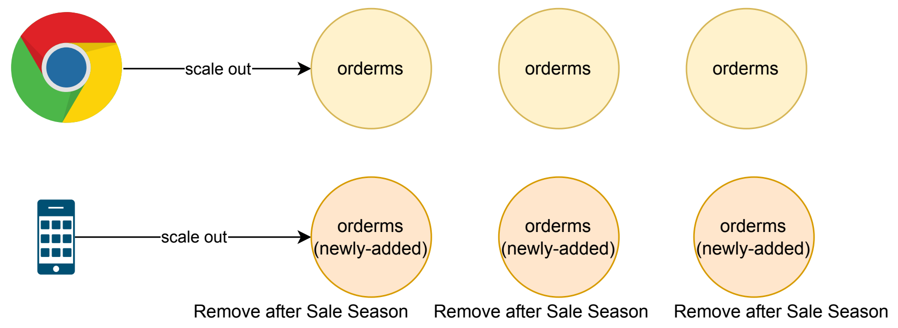


V. Build, Release, Run



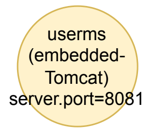
VI. Processes

Execute the app as one or more stateless processes



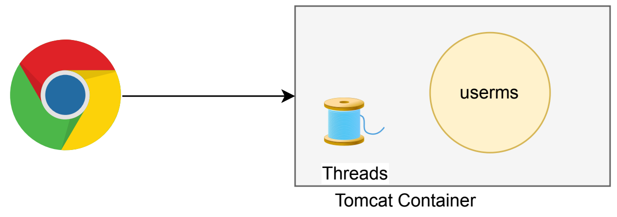
VII. Port Binding

Export services via Port Binding



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:

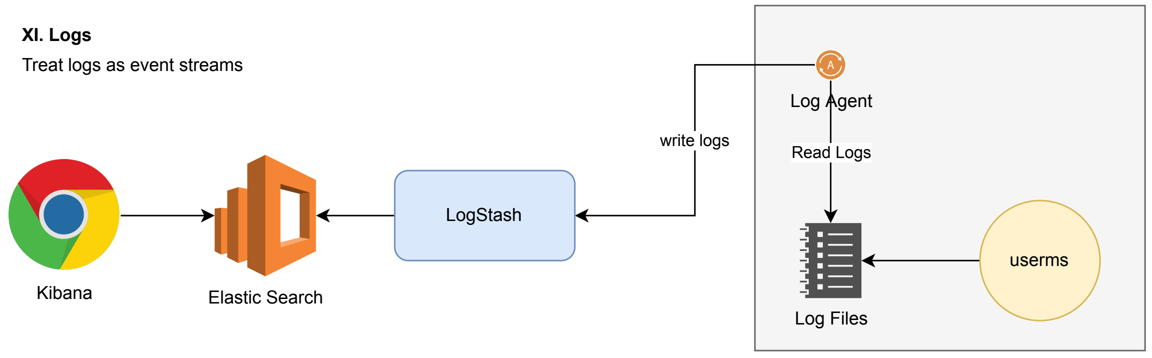
Container(Docker):
userms: Spring Boot Binary (Jar) + Java-8
orderms: Spring Boot Binary (War) + Java-11

Prod Env:

Container(Docker):
userms: Spring Boot Binary (Jar) + Java-8
orderms: Spring Boot Binary (War) + Java-11

XI. Logs

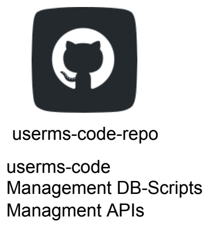
Treat logs as event streams



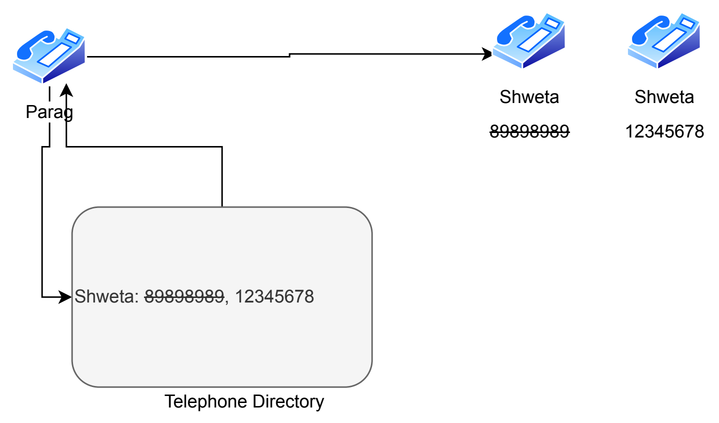
XII. Admin Processes

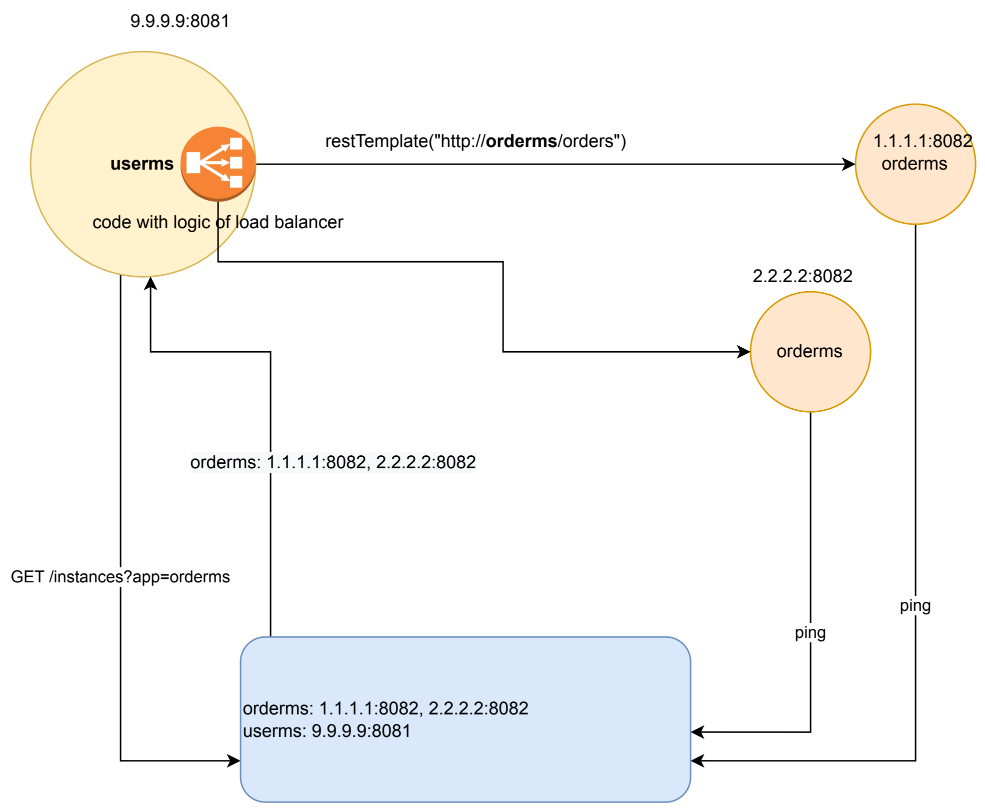
Run admin/management tasks as one-off processes

the script, the APIs, these all should be part of my code



Service Registry / Service Discovery + Client Load Balancer





9.9.9.9:8081
userms
code with logic of load balancer
restTemplate("http://**orderms**/orders")
1.1.1.1:8082
orderms
2.2.2.2:8082
orderms
orderms: 1.1.1.1:8082, 2.2.2.2:8082
GET /instances?app=orderms
ping
ping
orderms: 1.1.1.1:8082, 2.2.2.2:8082
userms: 9.9.9.9:8081

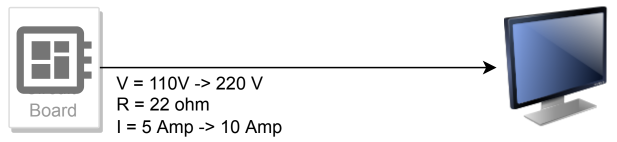
Netflix Eureka (Service Registry/Service Discovery)

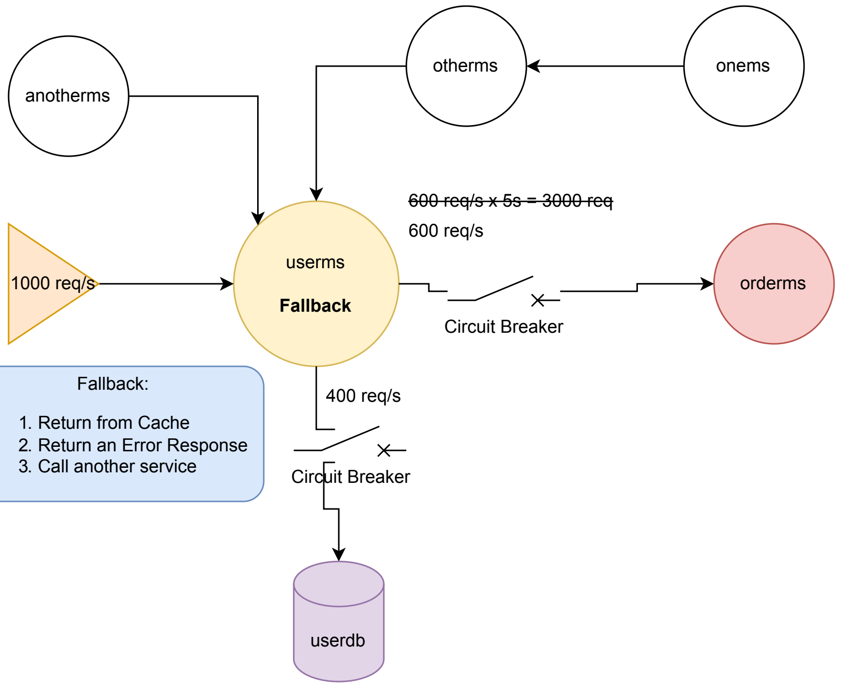
Day-3

Circuit Breaker

(Netflix Hystrix) (High Availability)

Fuse
MCB : Miniature Circuit Breaker



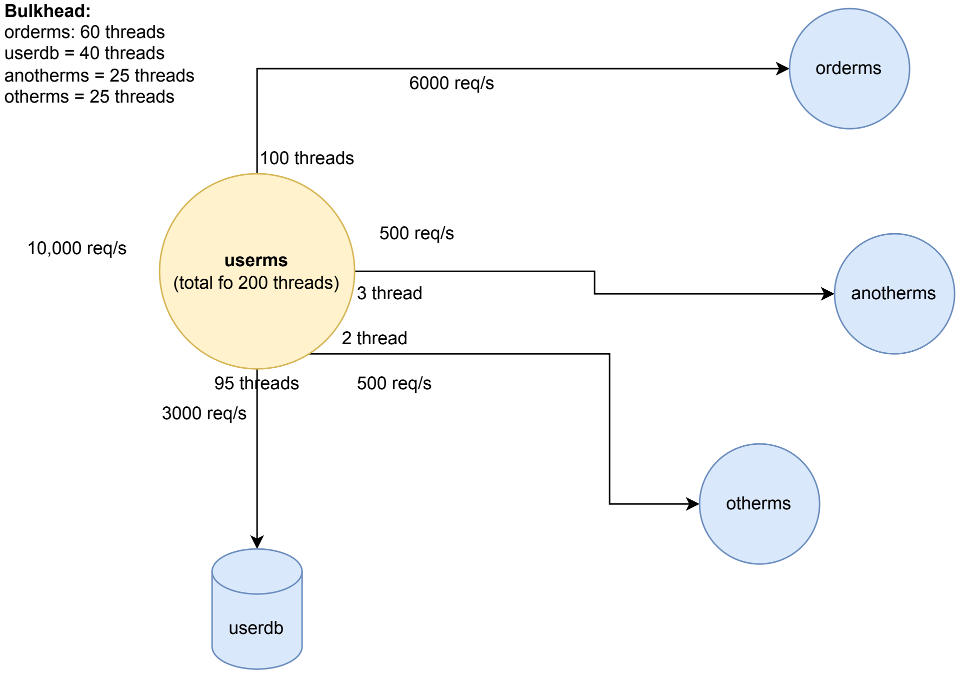


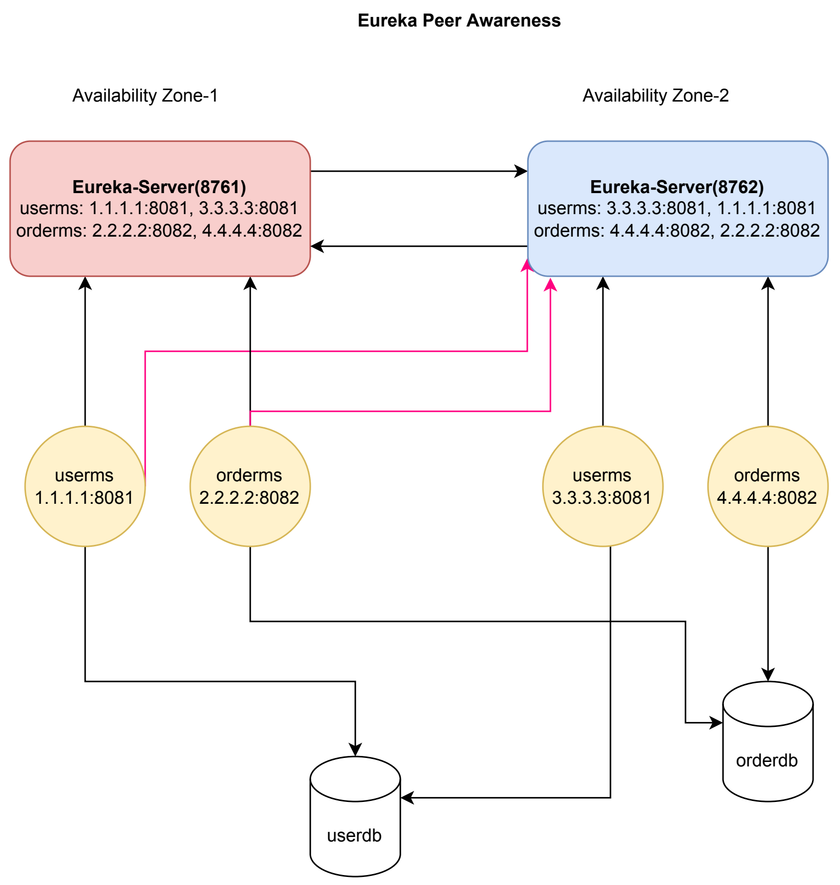
Fallback:

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

BulkHead

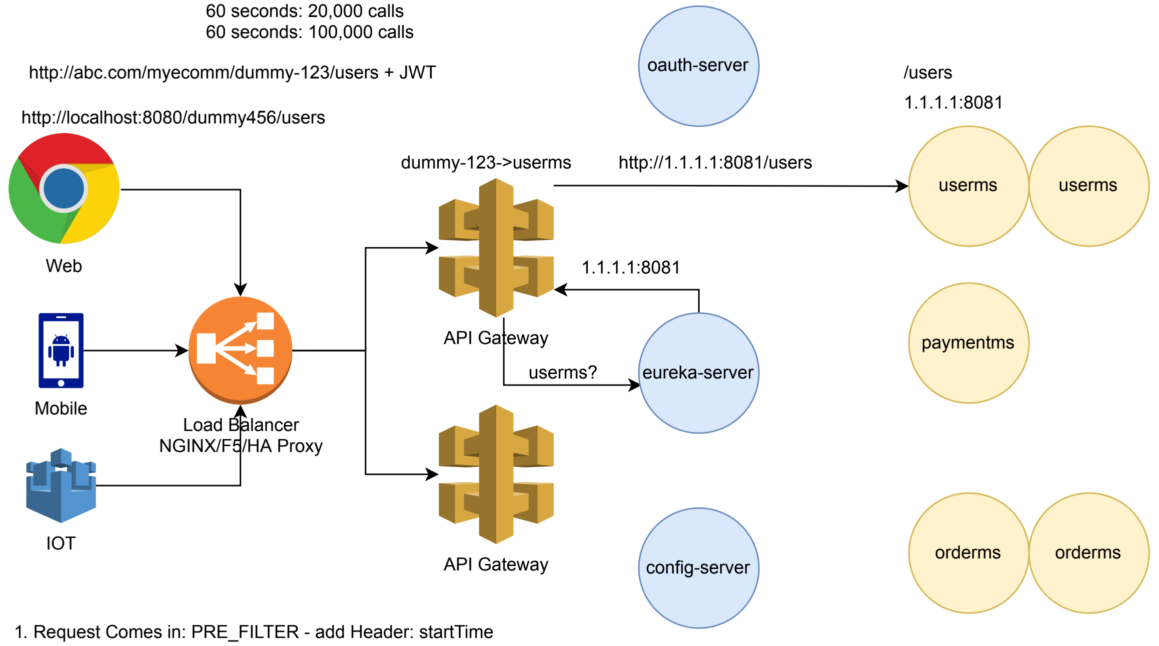
Bulkhead:
orderms: 60 threads
userdb = 40 threads
anotherms = 25 threads
otherms = 25 threads




Eureka Peer Awareness
Availability Zone-1
Availability Zone-2
Eureka-Server(8761)
userms: 1.1.1.1:8081, 3.3.3.3:8081
orderms: 2.2.2.2:8082, 4.4.4.4:8082
Eureka-Server(8762)
userms: 3.3.3.3:8081, 1.1.1.1:8081
orderms: 4.4.4.4:8082, 2.2.2.2:8082
userms 1.1.1.1:8081
orderms 2.2.2.2:8082
userms 3.3.3.3:8081
orderms 4.4.4.4:8082
orderdb
userdb

API Gateway (Edge Service)
(Netflix Zuul, Spring Cloud Gateway)

RateLimit:
Free:
60 seconds: 10 calls
Paid:
60 seconds: 10,000 calls
60 seconds: 20,000 calls
60 seconds: 100,000 calls



1. Request Comes in: PRE_FILTER - add Header: startTime
2. Response going out: POST_FILTER - calculate duration based on startTime

PRE_FILTER vs POST_FILTER:

Cross Cutting Concerns:

1. Security(Authenticatin/Authorization)
2. Security - URL Hiding
3. Proxy/Reverse Proxy
4. Audit (Collect API usage, duration of Requests)
5. Client-Specific Response
6. RateLimit(DDoS, monetize)
7. Distributed Tracing

API Gateway (Edge Service)

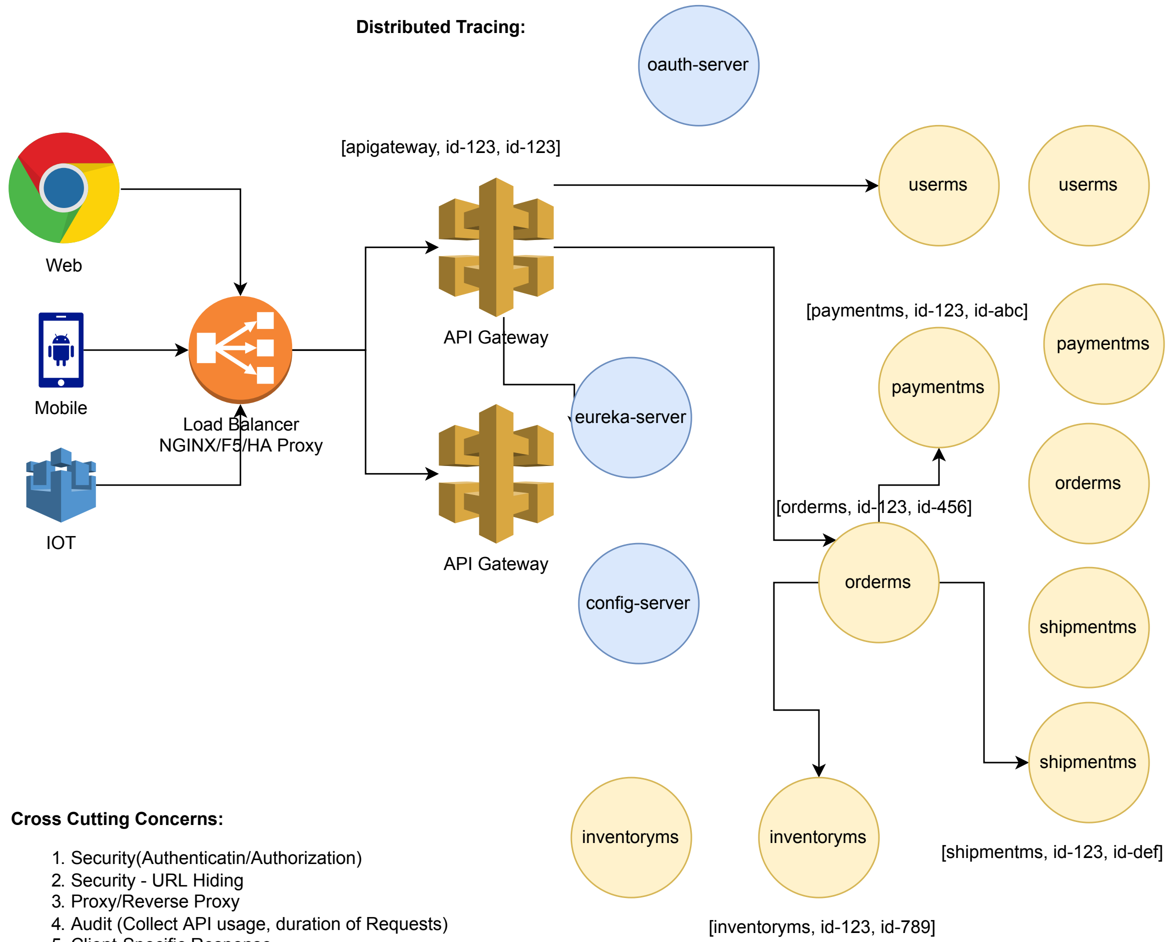
(Netflix Zuul, Spring Cloud Gateway)

Three pillars of Observability:

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

Trace: [microservice-name, requestId, spanId]

Distributed Tracing:

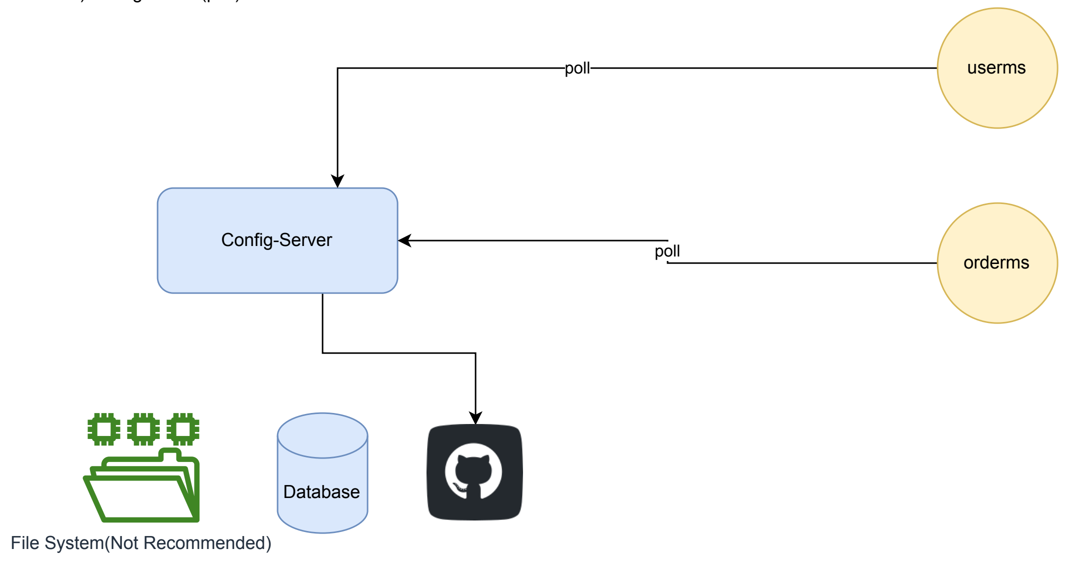


Cross Cutting Concerns:

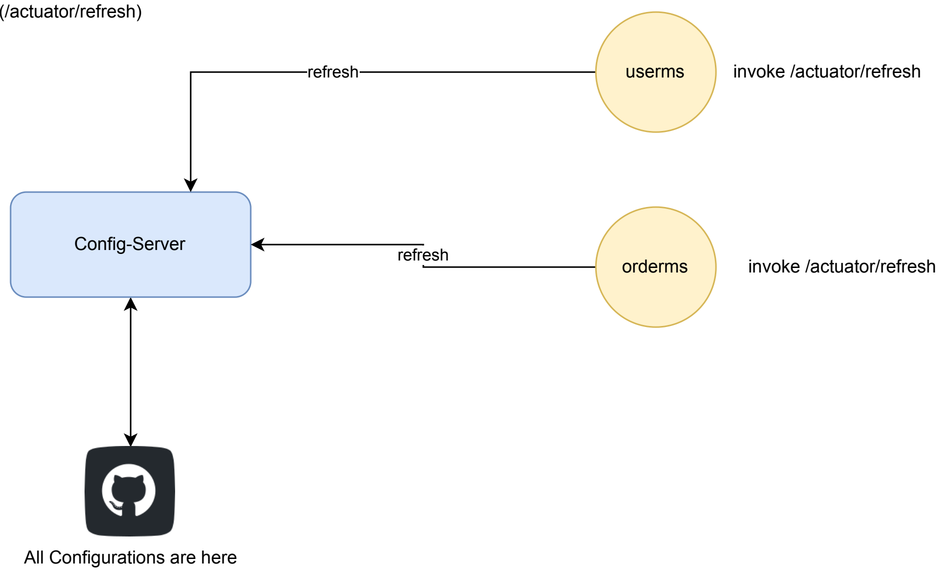
1. Security(Authenticatin/Authorization)
2. Security - URL Hiding
3. Proxy/Reverse Proxy
4. Audit (Collect API usage, duration of Requests)
5. Client-Specific Response
6. RateLimit(DDoS, monetize)
7. Distributed Tracing

Config-Server

1) Config Server(poll)



1) Config Server (/actuator/refresh)



Config Server (Spring Boot Actuator)

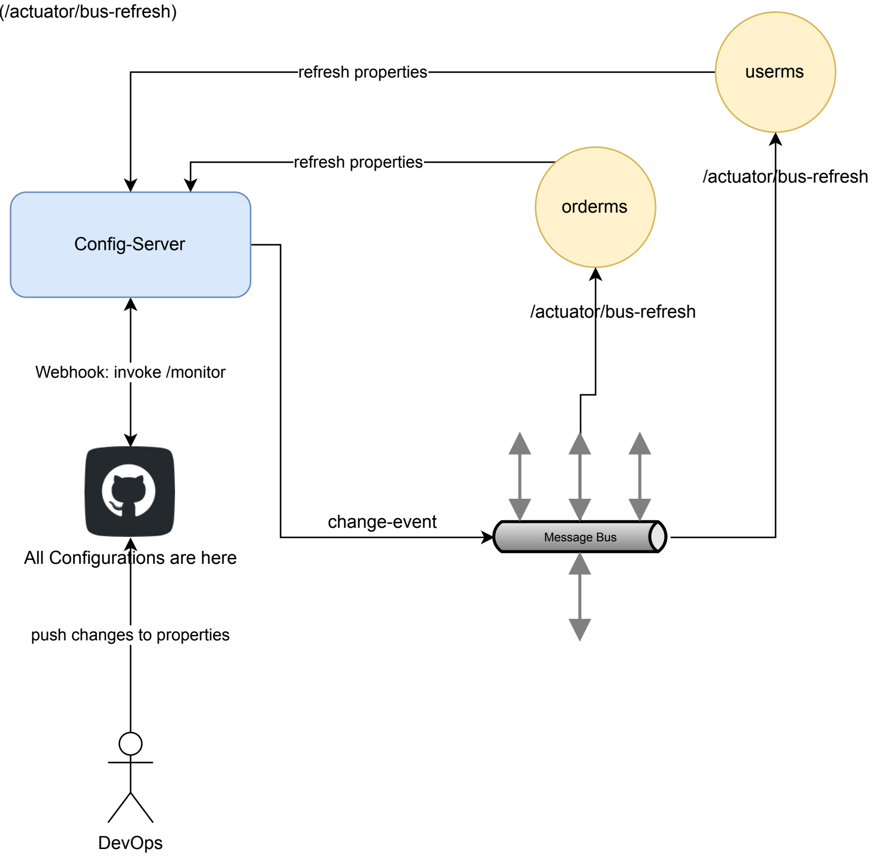
3) Config Server (/actuator/bus-refresh)

All Properties Files:

application.properties

userms.properties
userms-dev.properties
userms-prod.properties

orderms.properties
orderms-dev.properties
orderms-prod.properties


refresh properties
userms
refresh properties
orderms
/actuator/bus-refresh
Config-Server
/actuator/bus-refresh
Webhook: invoke /monitor
change-event
Message Bus
All Configurations are here
push changes to properties
DevOps