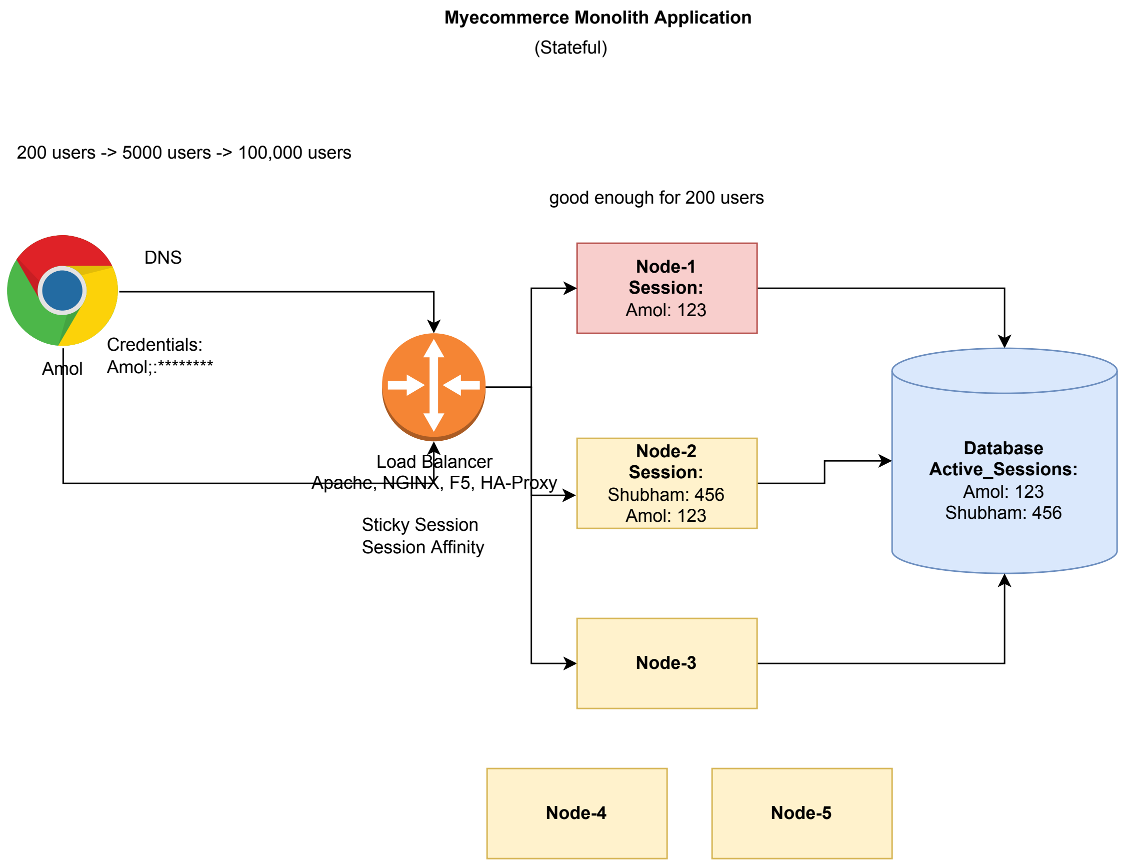
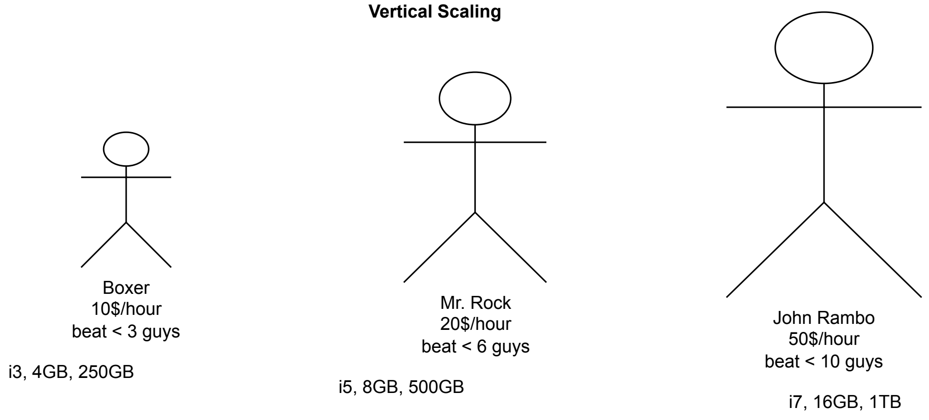
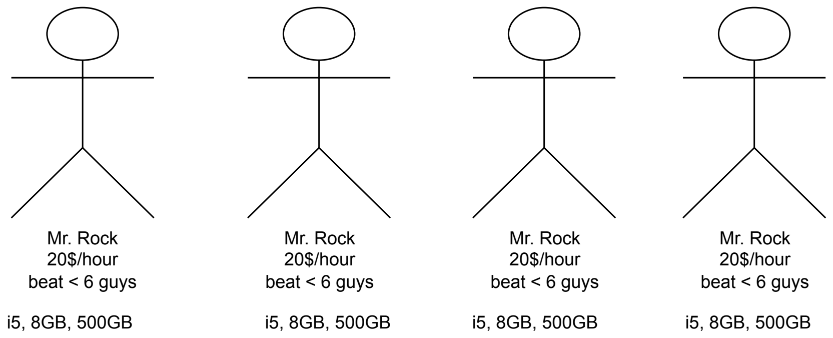

Myecommerce Monolith Application
(Stateful)
200 users -> 5000 users -> 100,000 users
good enough for 200 users
DNS
Amol
Credentials:
Amol;:********
Load Balancer
Apache, NGINX, F5, HA-Proxy
Sticky Session
Session Affinity
Node-1
Session:
Amol: 123
Node-2
Session:
Shubham: 456
Amol: 123
Node-3
Node-4
Node-5
Database
Active_Sessions:
Amol: 123
Shubham: 456

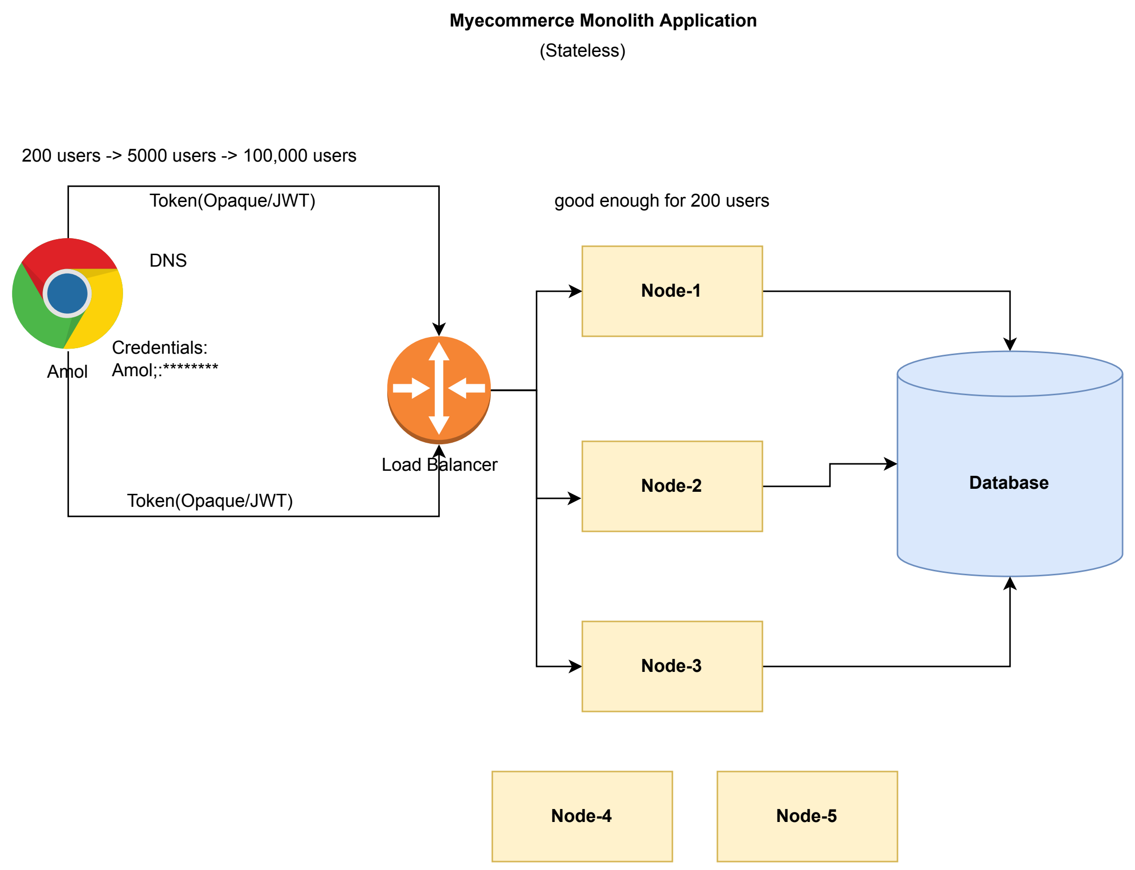
Scaling:
Vertical Scaling: Scale up
Horizontal Scaling: Scale out

Vertical Scaling



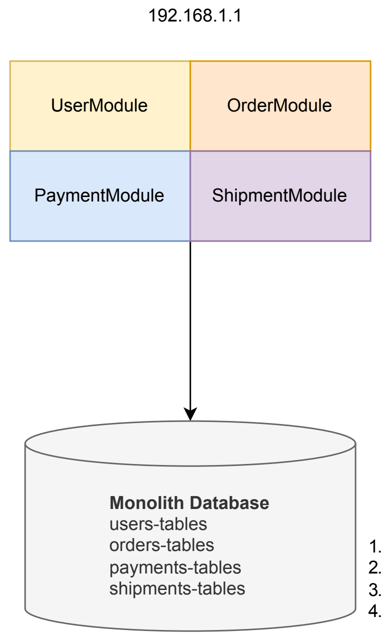
Horizontal Scaling




Myecommerce Monolith Application
(Stateless)
200 users -> 5000 users -> 100,000 users
Token(Opaque/JWT)
good enough for 200 users
DNS
Credentials:
Amol;:********
Amol
Load Balancer
Token(Opaque/JWT)
Node-1
Node-2
Node-3
Database
Node-4
Node-5

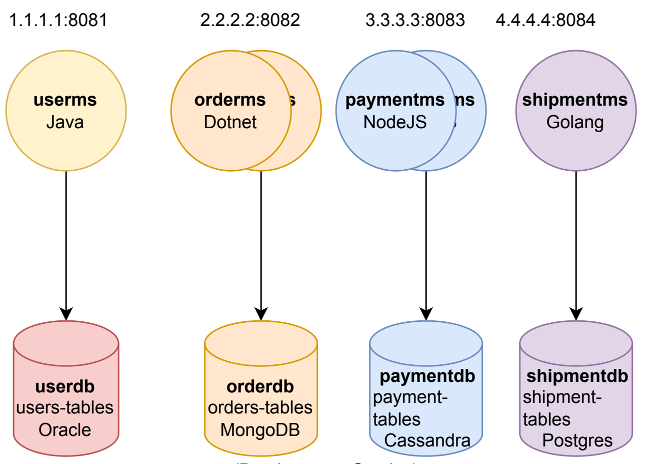
Black Friday/Diwali Sale Season

Myecommerce Monolith Application



Microservice Application

(Collection of standalone miniature applications)


1.1.1.1:8081
2.2.2.2:8082
3.3.3.3:8083
4.4.4.4:8084
userms Java
orderms Dotnet
paymentms NodeJS
shipmentms Golang
userdb users-tables Oracle
orderdb orders-tables MongoDB
paymentdb payment-tables Cassandra
shipmentdb shipment-tables Postgres

(Database per Service)

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

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

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

Pros of Microservices:

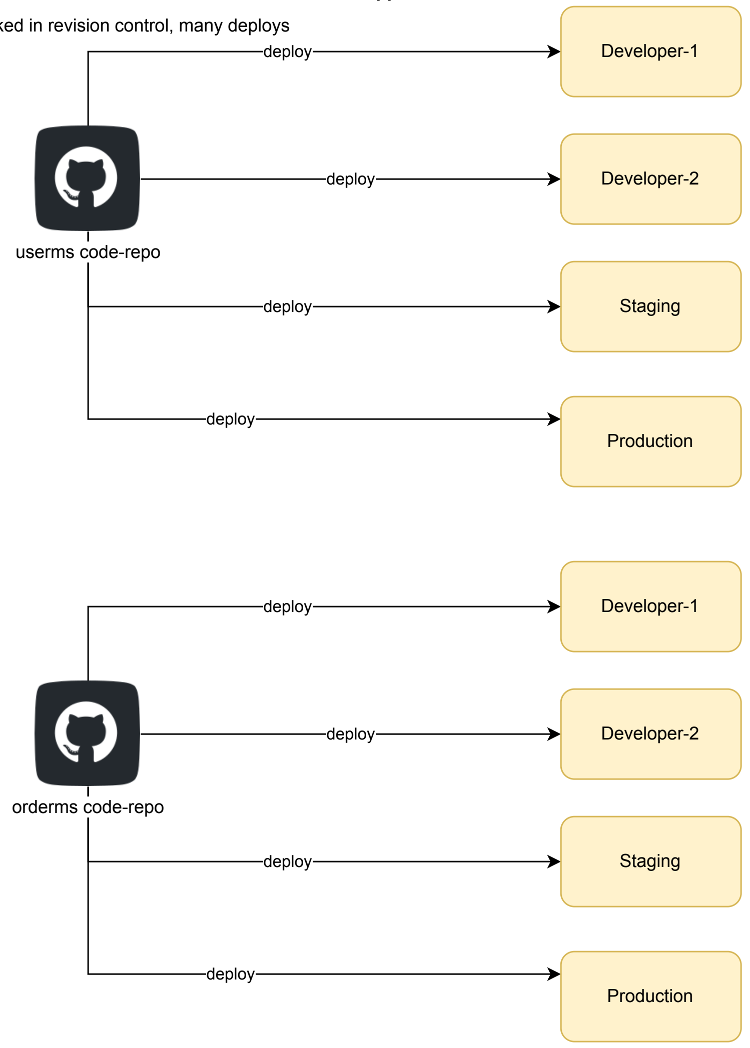
1. Cherry pick scaling
2. Agility-1: Development is fast
3. Agility-2: Build is fast
4. Agility-3: Testing is fast
5. Agility-4: CI/CD is fast
6. Agility-5: Release is fast
7. Resiliency
8. Distributed Service Load
9. Distribute DB Load
10. Security (Segregation)
11. Technology Hetrogenity
12. DB Hetrogenity

Cons of Microservices:

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

12 Factor App

I. Codebase
One codebase tracked in revision control, many deploys



II. Dependencies
Explicitly declare and isolate dependencies

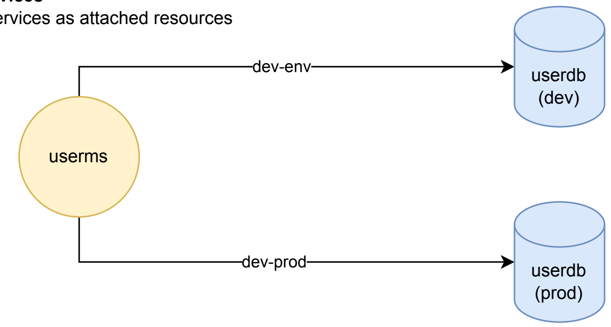
maven: pom.xml

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

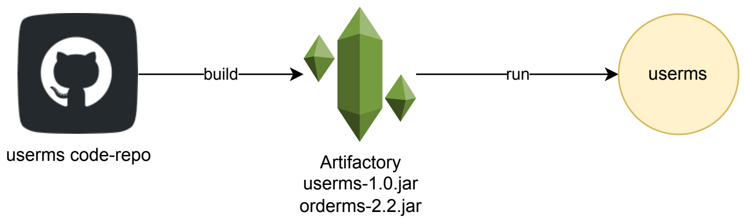
III. Config
Store config in the environment

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

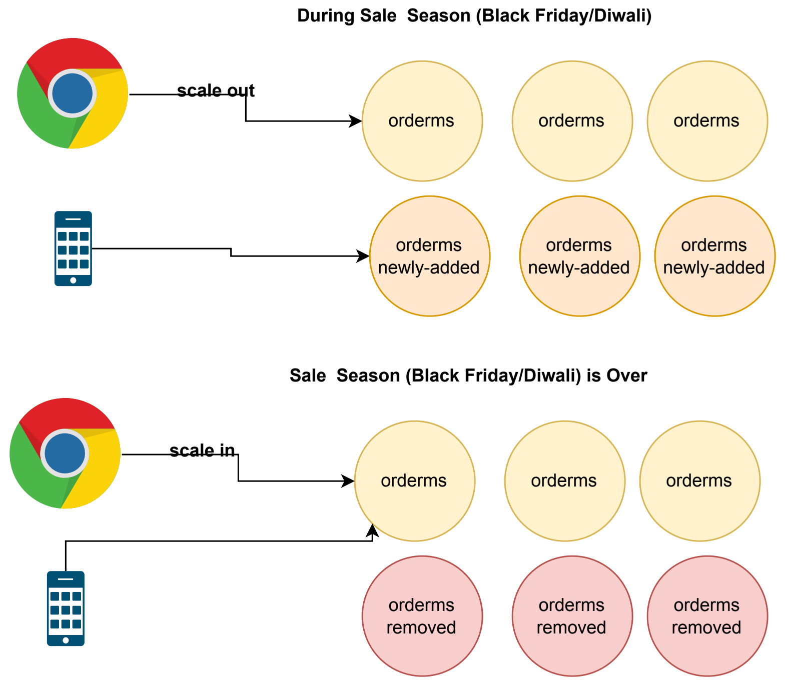
IV. Backing services
Treat backing services as attached resources



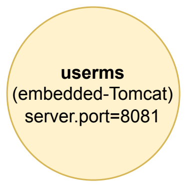
V. Build, release, run
Strictly separate build and run stages



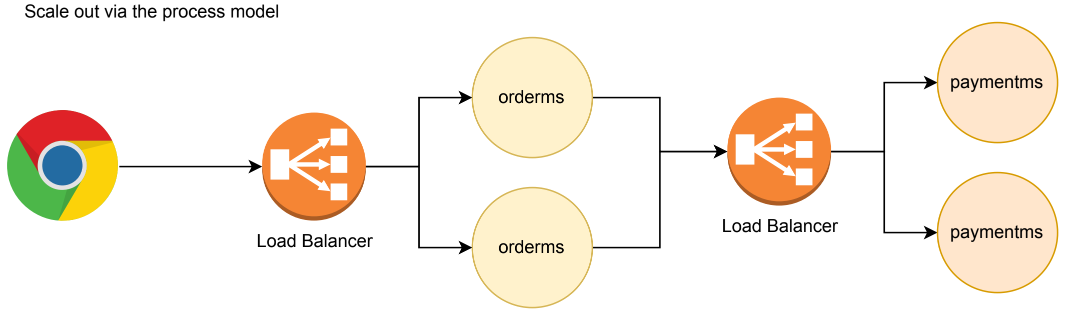
VI. Processes
Execute the app as one or more stateless processes



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 for quick scaling out.
graceful shutdown to keep the application in stable state.

X. Dev/prod parity
Keep development, staging, and production as similar as possible

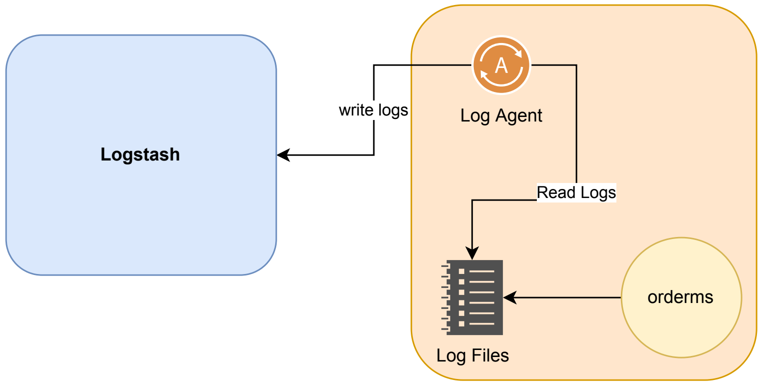
Dev Env:

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

Prod Env:

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

XI. Logs
Treat logs as event streams



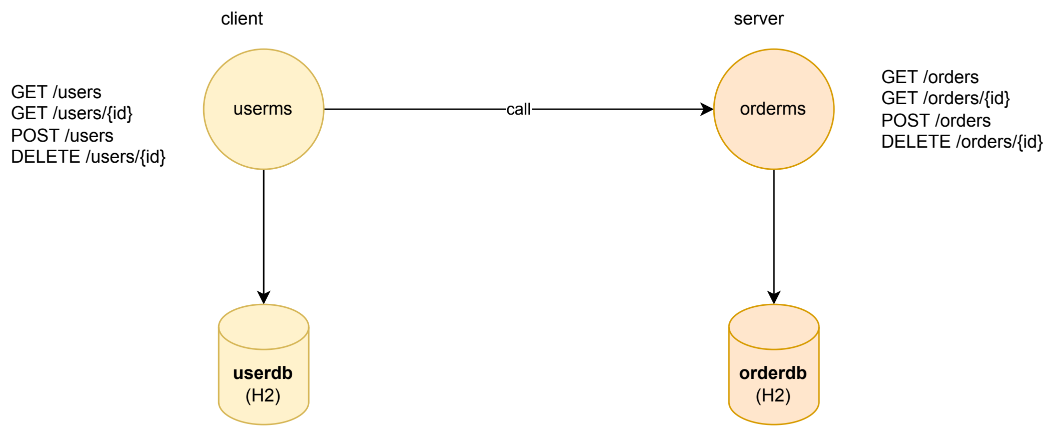
XII. Admin processes
Run admin/management tasks as one-off processes

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

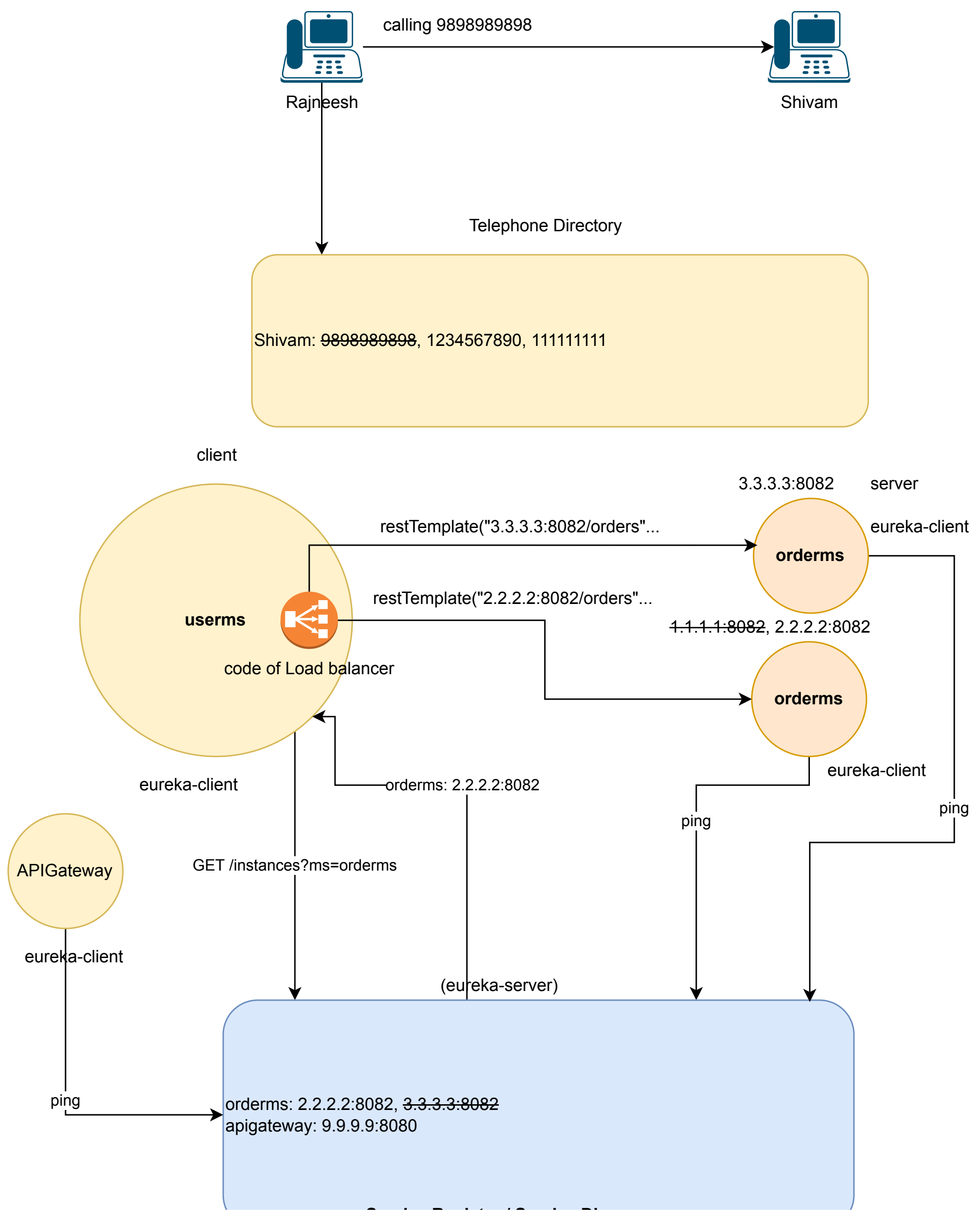
orderms code repo
orderms-code
Management db-scripts
Management APIs

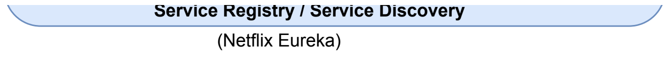
Test these through Swagger

Web, Spring Data JPA, H2, SpringFox(Swagger), Devtools(Optional)

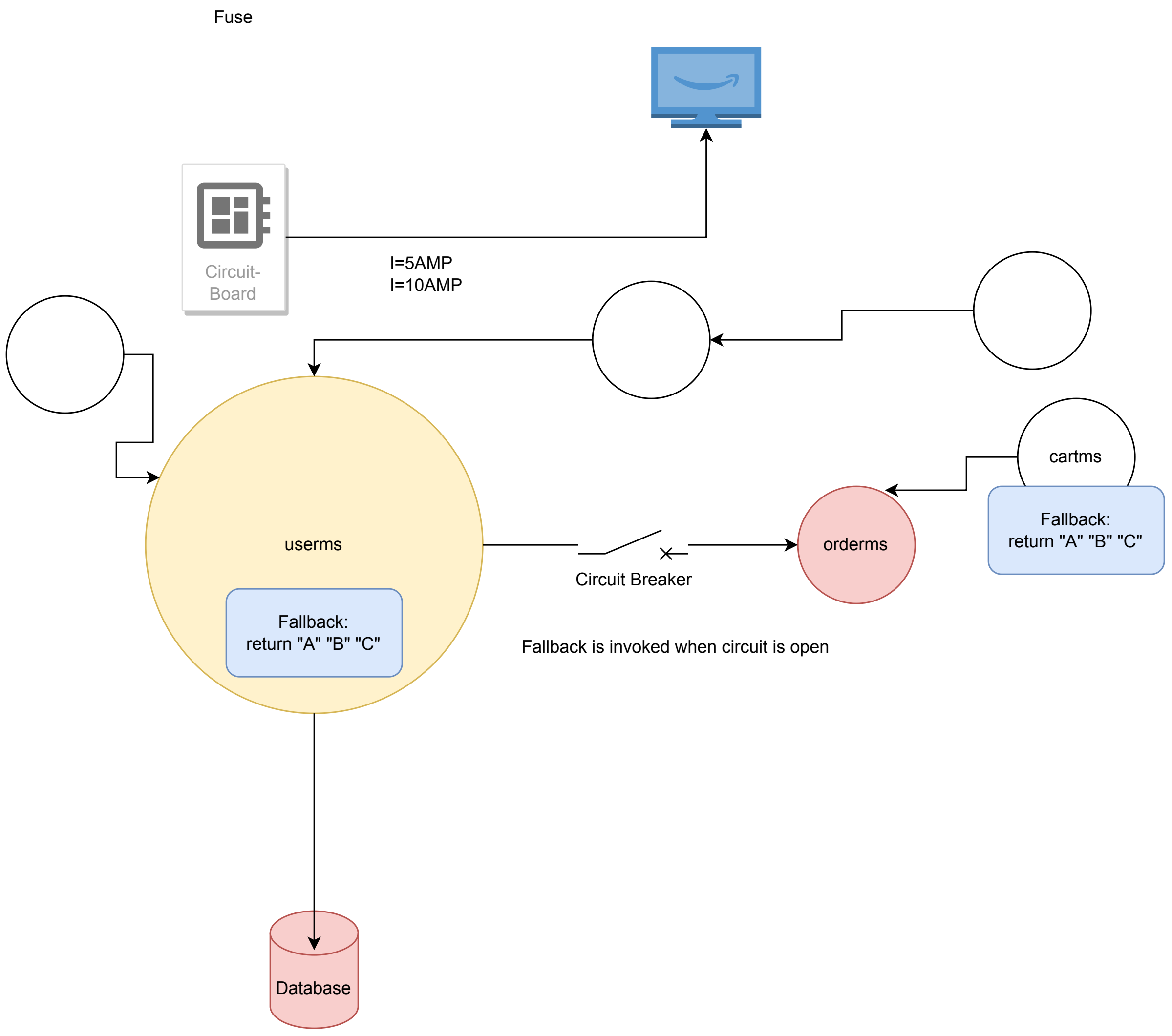


Service Registry / Service Discovery + Client-side Load Balancer



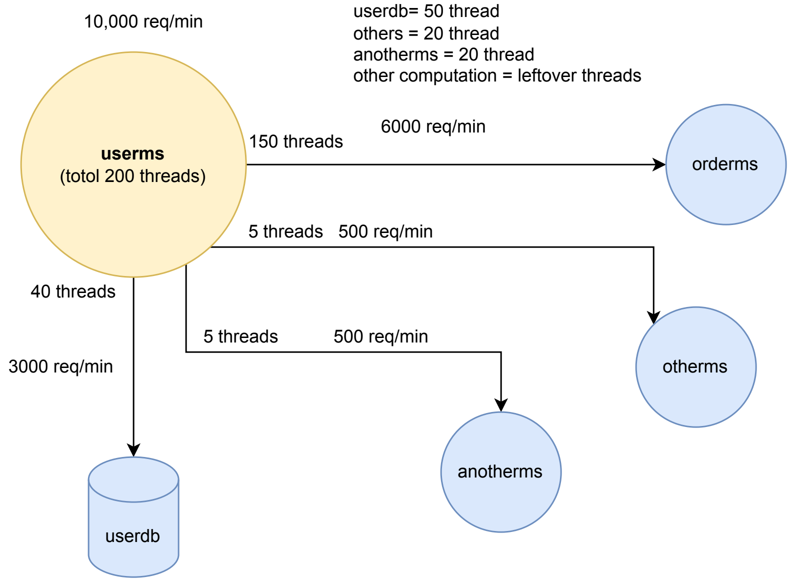

Service Registry / Service Discovery
(Netflix Eureka)

Circuit Breaker



Bulkhead

Bulkhead:
orderms = 50 threads
userdb= 50 thread
others = 20 thread
anotherms = 20 thread
other computation = leftover threads

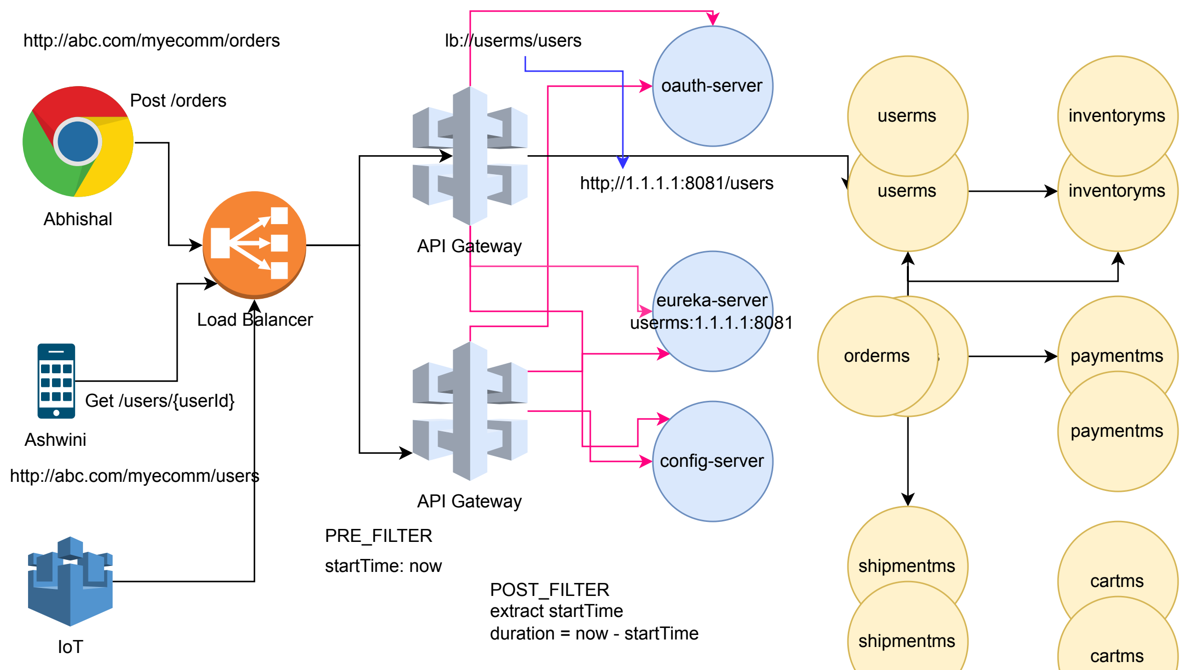


API Gateway (Edge Service)

(Netflix Zuul, Spring Cloud Gateway)

orderms: Post http://1.1.1.1:8082/orderms/api/v2/orders
userms: Get http:1.1.1.1:8081/userms/api/v1/users/{userId}

100 microservice x 5 instances = 500 instances



Cross Cutting Concerns:

1. Security (Authentication/Authorization)
2. Security - url hiding
3. Proxy/Reverse Proxy
4. Audit (Collect API usage, duration of Request)
5. Client-Specific Response
6. Distributed Tracing
7. RateLimit (DDoS, monetize)

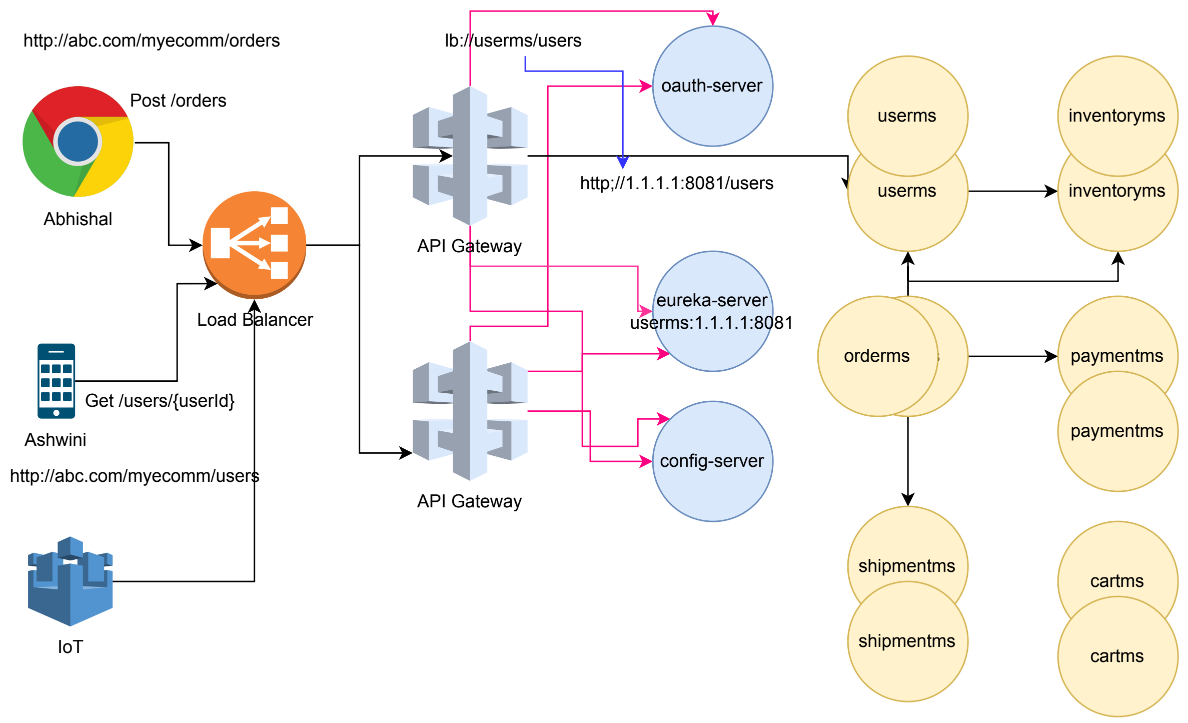
API Gateway (Edge Service)

(Netflix Zuul, Spring Cloud Gateway)

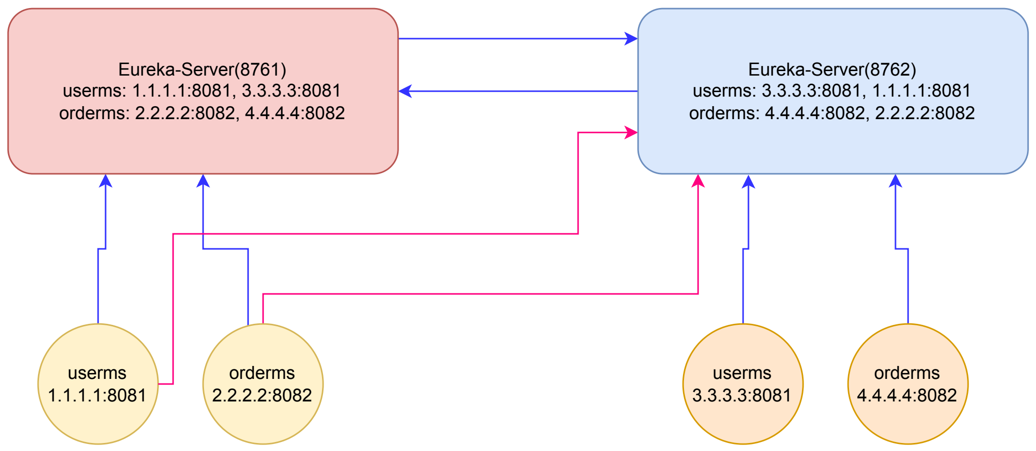
Distributed Tracing

[ms-name, requestId, spanId]

APIGateway: [apigateway, id-123, id-123]
Orderms: [orderms, id-123, id-456]
inventoryms: [inventoryms, id-123, id-789]
Paymentms: [paymentms, id-123, id-abc]
Shipmentms: [shipmentms, id-123, id-def]

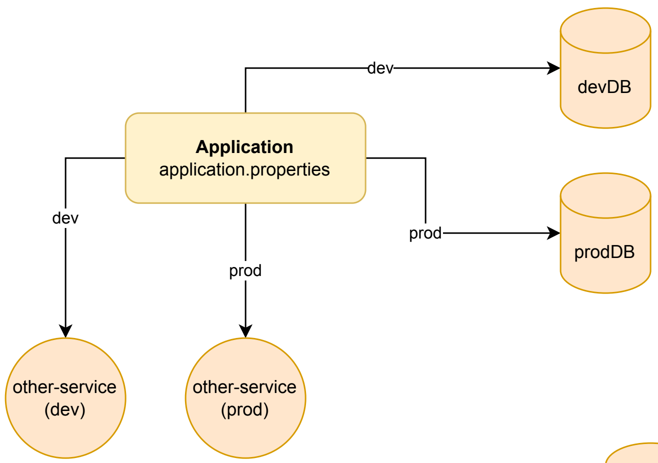


Eureka Peer Aware

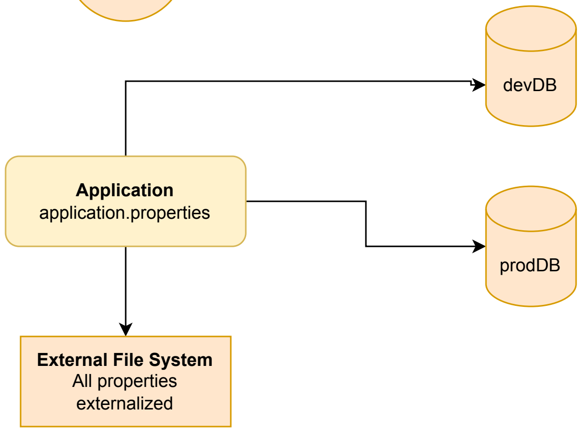

Eureka-Server(8761)
userms: 1.1.1.1:8081, 3.3.3.3:8081
orderms: 2.2.2.2:8082, 4.4.4.4:8082
Eureka-Server(8762)
userms: 3.3.3.3:8081, 1.1.1.1:8081
orderms: 4.4.4.4:8082, 2.2.2.2:8082
userms 1.1.1.1:8081
orderms 2.2.2.2:8082
userms 3.3.3.3:8081
orderms 4.4.4.4:8082

Config Server

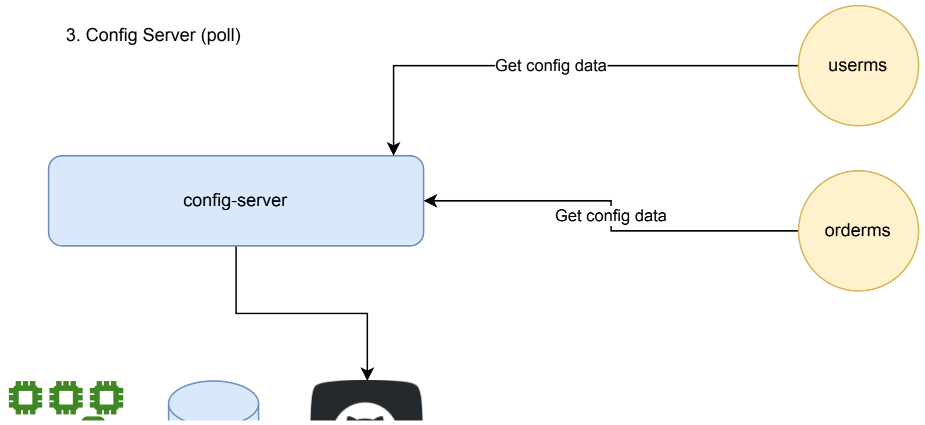
1. properties as part of code

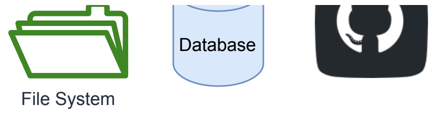


2. Externalize the properties



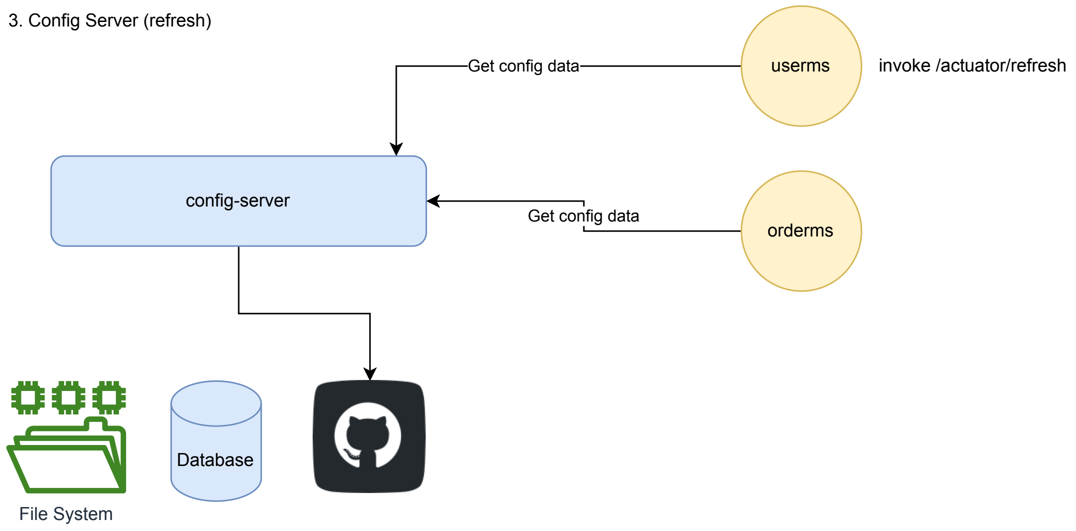
3. Config Server (poll)



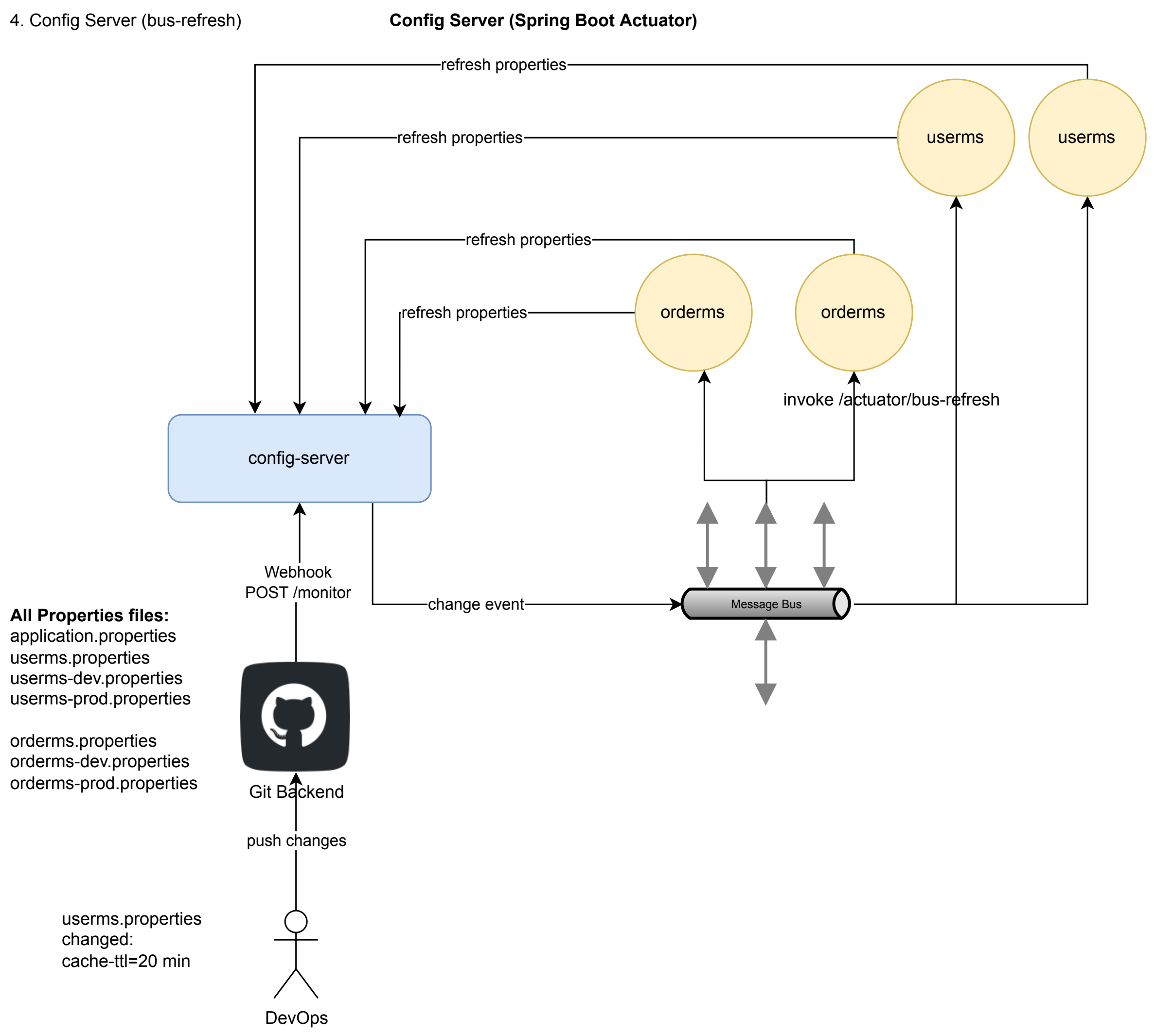

Database
File System

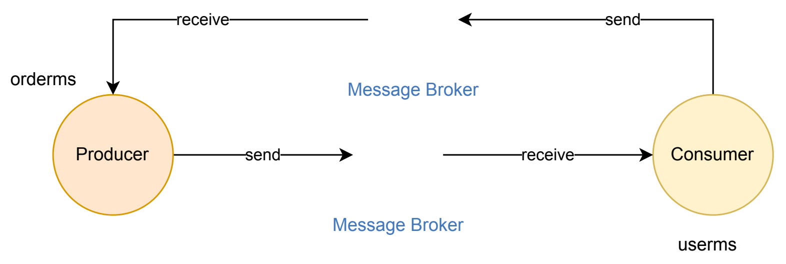
Config Server (Spring Boot Actuator)

3. Config Server (refresh)



4. Config Server (bus-refresh)




receive
send
orderms
Message Broker
Producer
send
receive
Consumer
Message Broker
userms