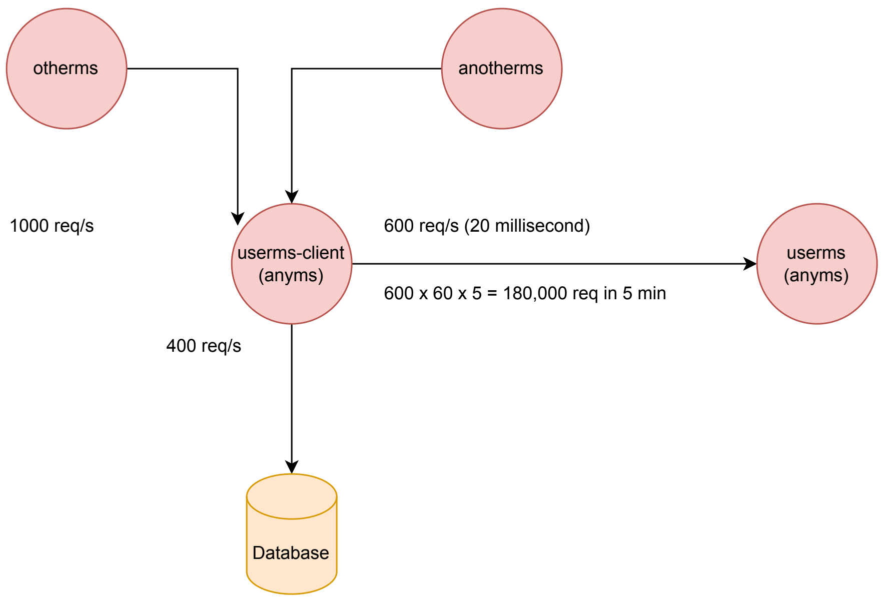
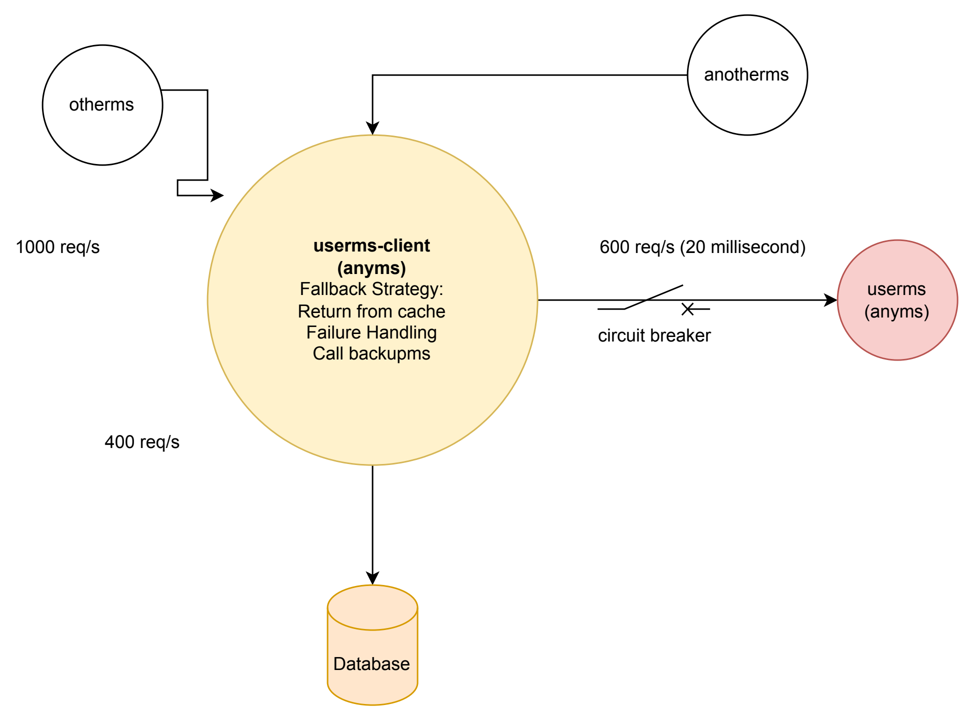
**Microservice Design Patterns**

| Integration Patterns | Database Patterns | Cross-Cutting Patterns | Decomosition Patterns | Observability Patterns |
|---|---|---|---|---|
| API Gateway | Database Per Service | Service Discovery | Strangler Vine | Distributed Tracing |
| Reverse Proxy | Shared Database | Service Registry | | Log Aggregation |
| Aggregator | Saga | Client-side Load Balancing | | Metrics |
| Chained | Event Sourcing | External Configuration | | Health Check |
| Branch | CQRS | Circuit Breaker | | |

**Circuit Breaker (Resiliency)**



**Circuit Breaker (Resiliency)**



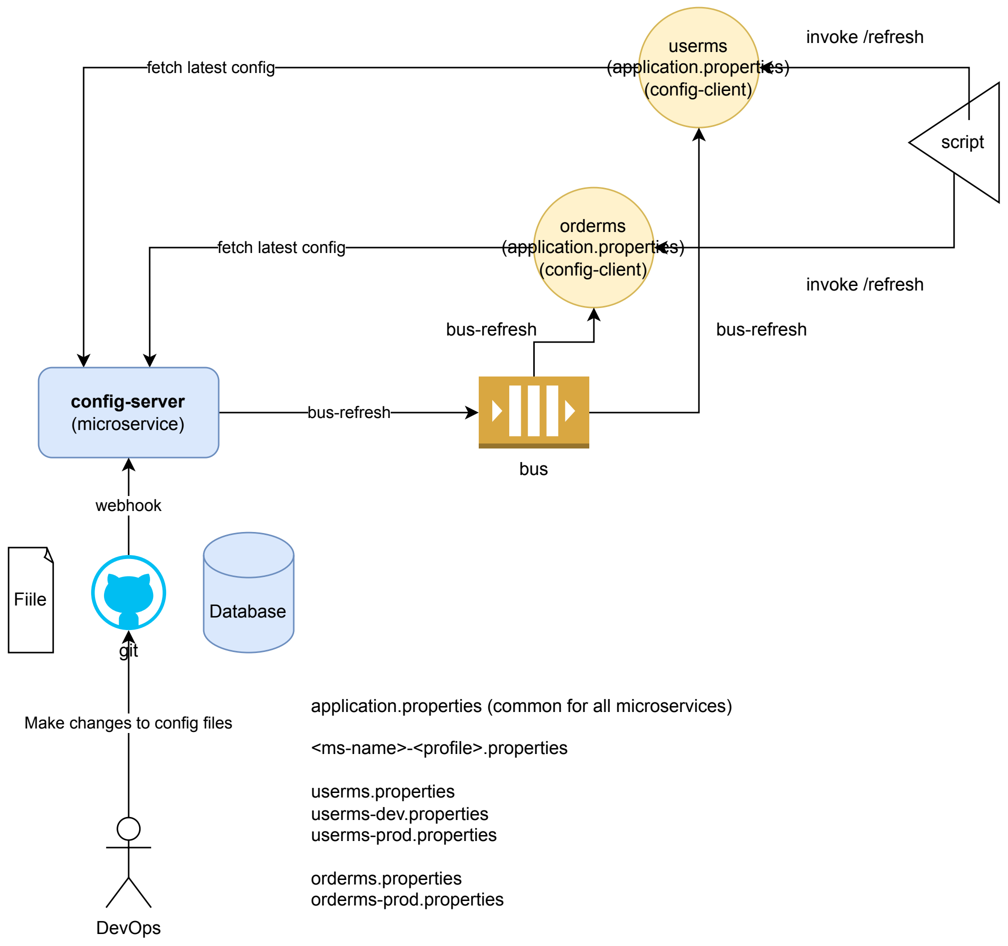
**Config Server**

100 microsevices x 5 instances = 500 instances

spring cloud config server
spring cloud config client

cache-ttl = 10 minutes -> 20 minutes

/refresh



application.properties (common for all microservices)

<ms-name>-<profile>.properties

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

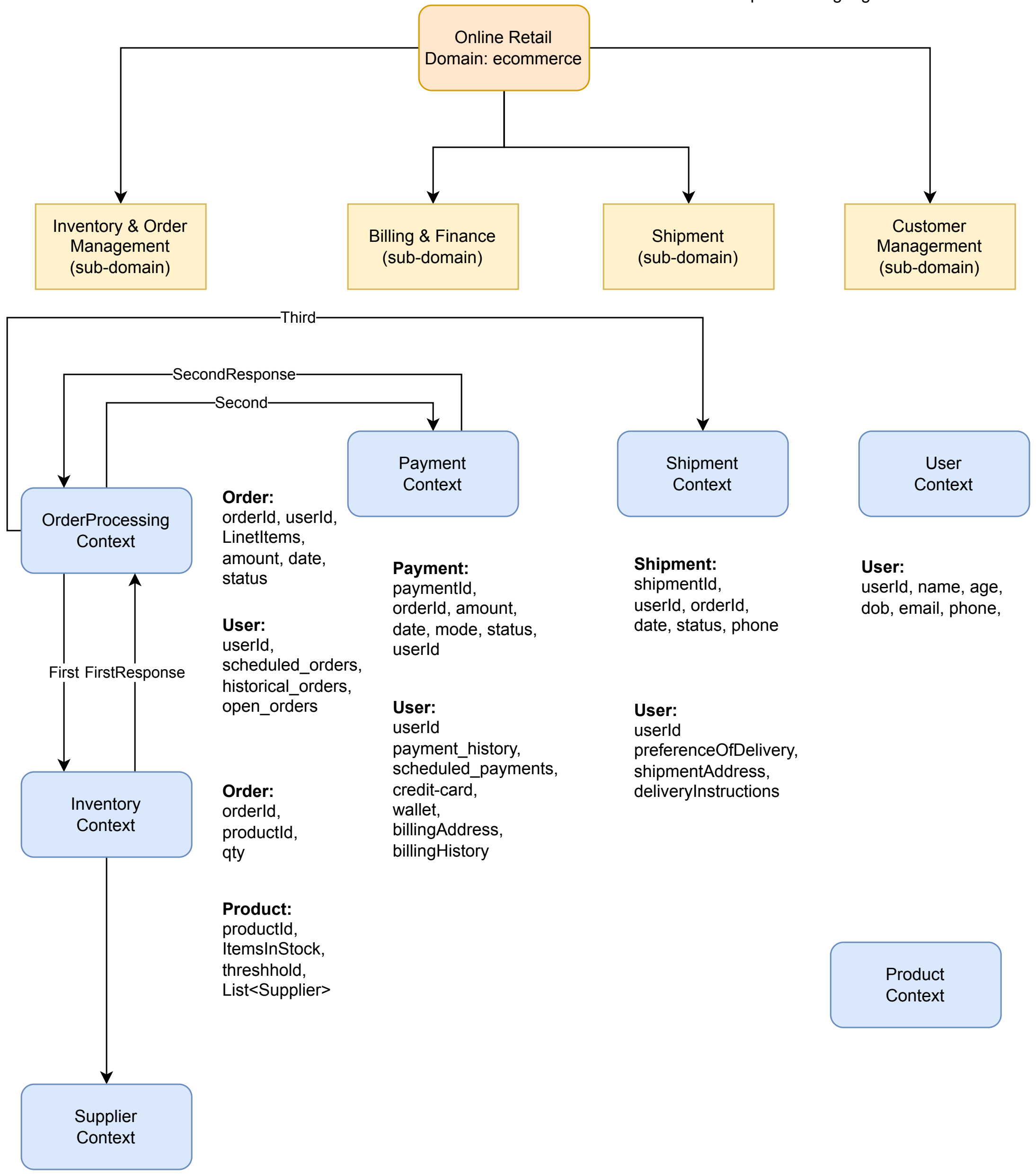
orderms.properties
orderms-prod.properties

**Database Techniques in Microservices**

DRY: Do not repeat yourself
WET: Write Everything Twice

**Domain Driven Design (DDD)**

1. Domain/SubDomain
2. Context/Bounded Context
3. Model
4. Ubiquitous Language



**Order:**
orderId, userId,
LinetItems,
amount, date,
status

**User:**
userId,
scheduled_orders,
historical_orders,
open_orders

**Order:**
orderId,
productId,
qty

**Product:**
productId,
ItemsInStock,
threshhold,
List<Supplier>

**Payment:**
paymentId,
orderId, amount,
date, mode, status,
userId

**User:**
userId
payment_history,
scheduled_payments,
credit-card,
wallet,
billingAddress,
billingHistory

**Shipment:**
shipmentId,
userId, orderId,
date, status, phone

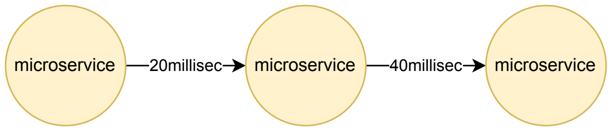
**User:**
userId
preferenceOfDelivery,
shipmentAddress,
deliveryInstructions

**User:**
userId, name, age,
dob, email, phone,

**Granularity of Microserives**

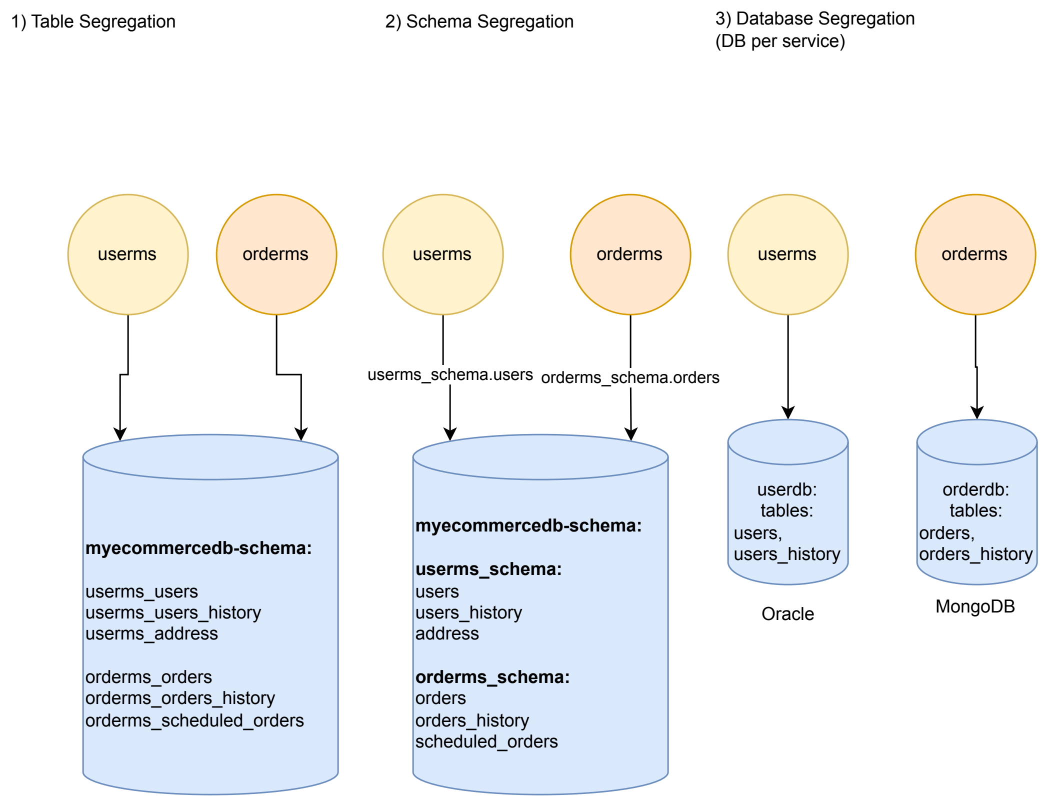
1. Domain Driven Design
2. Thumb Rule
3. Benchmark

**2. Thumb Rule:** A request should not span across more than 3-5 microservices calls in chain



**3. Benchmark:** How many time my request take on average

**Database Strategies**



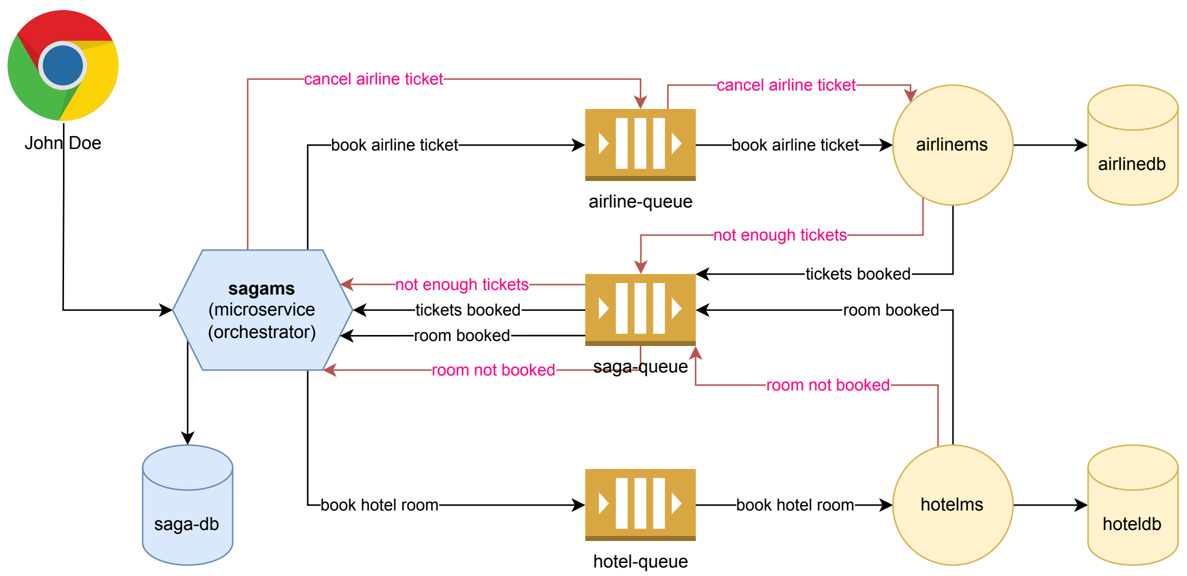
**Transaction Management**

Use Cases:

1. Both airlinems and hotelms successfully update the Database
2. airlinems fails but hotelms succeeds to update the Database
3. airlinems succeeds but hotelms fails to update the Database
4. Both airlinems and hotelms fail to update the Database

Saga Pattern
(Eventual Consistency)

SEC: Saga Execution Coordinator


John Doe
cancel airline ticket
cancel airline ticket
book airline ticket
book airline ticket
airline-queue
airlinems
airlinedb
not enough tickets
sagams (microservice (orchestrator)
not enough tickets
tickets booked
tickets booked
room booked
room booked
saga-queue
room not booked
room not booked
saga-db
book hotel room
hotel-queue
book hotel room
hotelms
hoteldb