1. System Overview

This project implements a simple synchronous microservice architecture using Spring Boot.

The system consists of three independent services:

- InventoryService (port 8081)
- OrderService (port 8080)
- NotificationService (port 8083)

Communication Flow: Client → OrderService → InventoryService → NotificationService → OrderService → Client

OrderService calls InventoryService synchronously.
If the inventory reservation succeeds, OrderService then calls NotificationService synchronously.

Since the calls are synchronous, OrderService blocks and waits for each downstream service to respond before returning a result to the client.

2. Latency Results (N = 5)

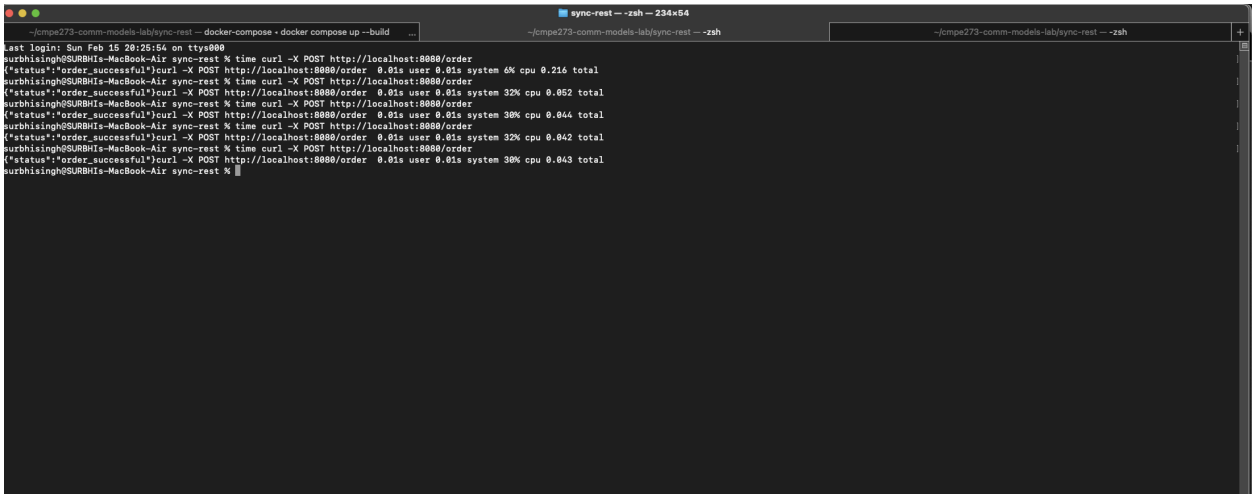
Latency was measured using: time curl -X POST http://localhost:8080/order

Each scenario was tested 5 times.

Scenario 1 – Baseline (No Delay) : InventoryService responds immediately.

Run	Latency (seconds)
1	0.216
2	0.052
3	0.044
4	0.042
5	0.042

Average is 0.079s

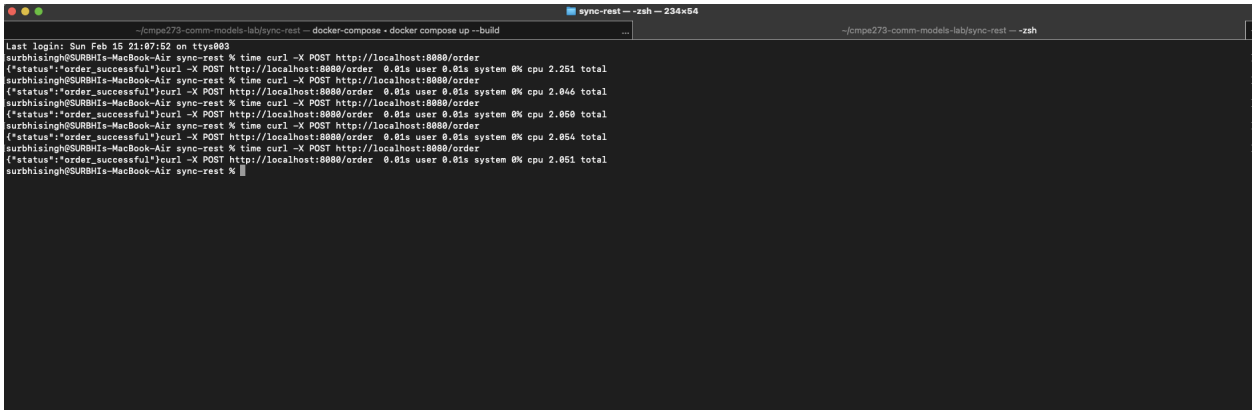


OrderService waits synchronously for InventoryService and NotificationService. Since both services respond immediately, total latency is low.

Scenario 2 – Injected 2sec (Delay) : InventoryService artificially delayed using Thread.sleep(2000);

Run	Latency
1	2.251
2	2.046
3	2.050
4	2.054
5	2.051

Average is 2.09 s



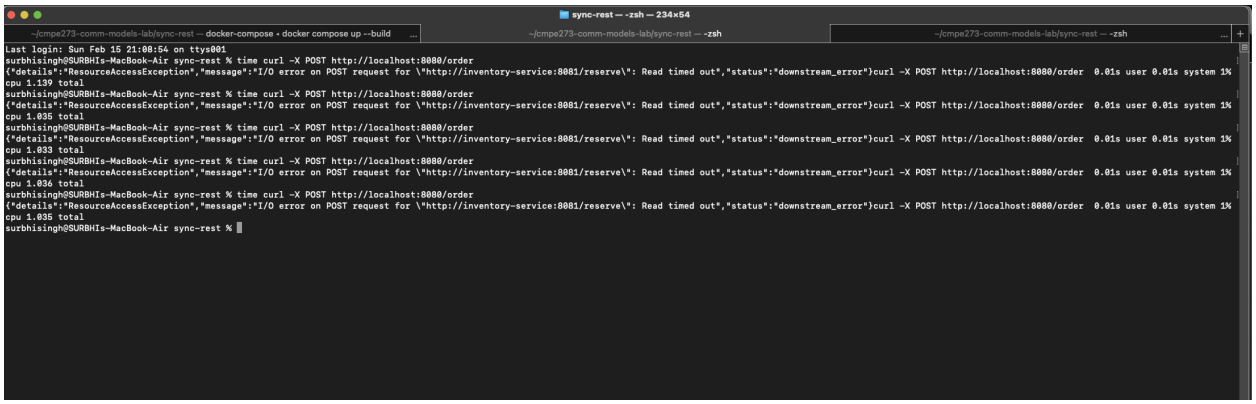
When a 2-second delay is injected into InventoryService, OrderService latency increases accordingly. Since OrderService calls InventoryService synchronously, it blocks and waits for the response. Therefore, downstream delay directly propagates to upstream latency.

Scenario 3 – Inventory failure and Timeout

InventoryService simulated hang. OrderService configured with 1-second timeout.

Run	Latency
1	1.139
2	1.035
3	1.033
4	1.036
5	1.035

Average is 1.06s



When InventoryService becomes unresponsive, OrderService waits only until its configured timeout (1 second).

Instead of blocking indefinitely, it returns a downstream error response. This demonstrates failure handling in synchronous communication and the importance of timeout protection.