CSE-5306 PROJECT 2

Distributed Pharmacy Inventory Management System

Shaoming Pan
Chenchuhui Hu

gRPC Microservice Architecture vs REST Monolith

February 2026 | Distributed Systems

Agenda

01 **System Requirements**

6 functional + non-functional requirements

02 **Architecture Design**

gRPC Microservice (6 nodes) + REST Monolith

03 **Communication Model**

gRPC with Protobuf vs HTTP/JSON

04 **Evaluation Methodology**

Benchmark setup and workload specification

05 **Performance Results**

Latency & throughput comparison

06 **AI Tools & Lessons**

How Claude accelerated development

System Requirements

💊 Add Drug

Register new drug with name, quantity, price, expiry

🔍 Get Drug

Query drug details by ID

📦 Update Stock

Modify inventory quantity

🗑️ Delete Drug

Remove drug record from system

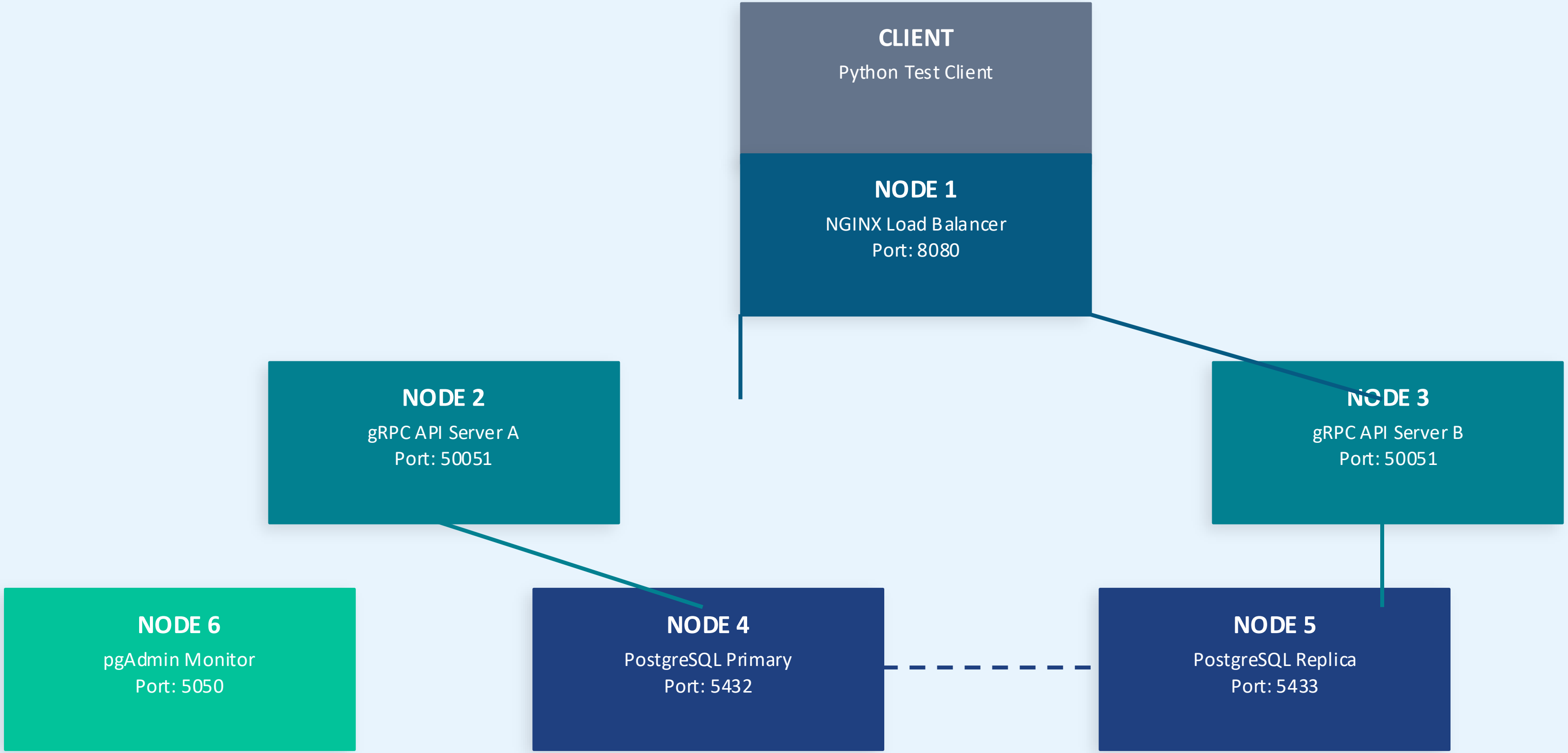
📋 List All Drugs

View complete inventory list

⚠️ Low Stock Alert

Find drugs below threshold quantity

Architecture Design — gRPC Microservice (6 Nodes)



--- Replication | — gRPC calls

Communication Model

gRPC Microservice	REST Monolith
Protocol: HTTP/2 + Protobuf binary	**Protocol:** HTTP/1.1 + JSON text
Format: Binary serialization	**Format:** Human-readable JSON
Speed: Faster encoding/decoding	**Speed:** Slower JSON parsing
Streaming: Bidirectional streams	**Streaming:** Not supported
Contract: Strongly typed .proto file	**Contract:** Flexible, no schema required
Best for: Internal service-to-service calls	**Best for:** Public APIs, browser clients

Evaluation Methodology

Hardware Environment

Machine: MacBook Pro (Apple Silicon)

RAM: 16 GB | OS: macOS

Docker Desktop (all containers local)

Python 3.9 benchmark scripts

Workload Specification

Write: concurrent AddDrug requests

Read: concurrent ListDrugs requests

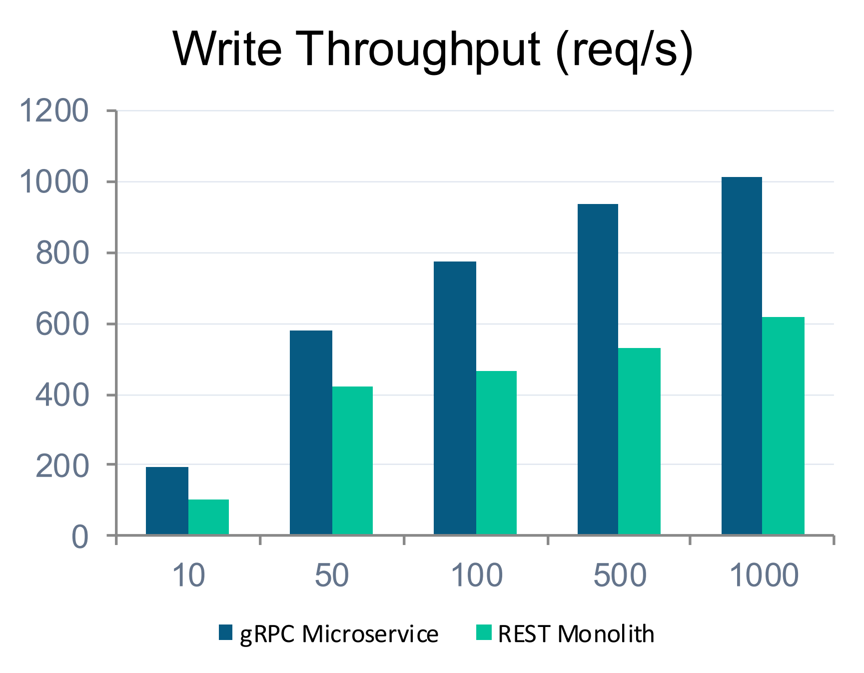
Concurrency: 10, 50, 100, 500, 1000

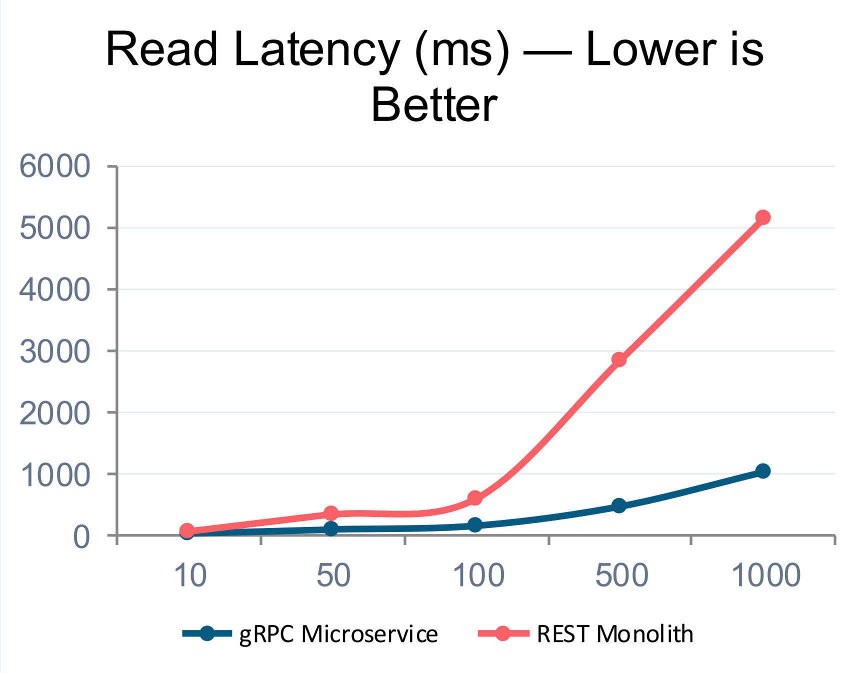
Metrics: Avg Latency (ms) + Throughput (req/s)

```
Latency = T_response − T_request          Throughput = Total Requests / Total Time
```

gRPC: 6 nodes | REST Monolith: 2 containers | Both running simultaneously

Performance Results





At 1000 users: gRPC read latency 1023ms vs REST 5150ms — 5× better | gRPC write throughput 63% higher

Trade-off Analysis

Dimension	gRPC Microservice	REST Monolith
Performance	Higher throughput, lower latency	Degrades sharply at scale
Scalability	Horizontal via load balancer	Single bottleneck
Fault Tolerance	Redundant servers + DB replication	Single point of failure
Complexity	Higher — multi-node orchestration	Lower — one container
Dev Speed	Slower — proto definitions needed	Faster — standard REST

AI Tools — How Claude Helped

Architecture Design

Designed the 6-node system layout and Docker Compose configuration

Proto File

Generated all gRPC .proto definitions for 6 service methods

Debugging

Diagnosed Dockerfile COPY path error and fixed build context

Benchmarking

Generated concurrent Python benchmark and plotting scripts

Documentation

Wrote README, final report, and this presentation

Prompting Strategy: provide exact errors + architecture constraints → targeted, working solutions

Conclusion

- ✅ 6 functional requirements fully implemented and tested
- ✅ 6-node gRPC microservice architecture deployed with Docker
- ✅ gRPC delivers 5× lower latency and 63% higher throughput at scale
- ✅ Microservice design provides fault tolerance REST cannot match
- ✅ AI tools (Claude) accelerated development at every stage

Thank you! | Q&A