Patient Management System

Microservices Architecture with Spring Boot, gRPC & Kafka

Members: |Shahzaman | Saram | Abdul Nafay

PROBLEM STATEMENT

- Scalability Bottlenecks: Legacy monolithic architecture cannot handle high concurrent patient requests.
- Single Point of Failure: Errors in the billing module cause the entire patient portal to crash.
- Tightly Coupled Systems: Direct synchronous communication slows down response times during peak loads.

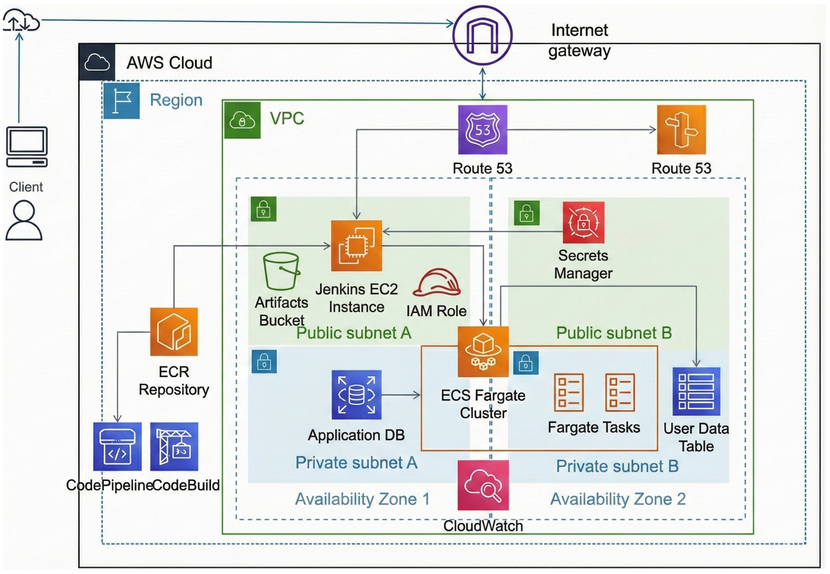
INTRODUCTION

This is the Enterprise Patient Management System. We have engineered a cloud-native solution that replaces fragile legacy healthcare software with a robust Microservices Architecture. By leveraging AWS, gRPC, and Kafka, our platform eliminates single points of failure and ensures zero-downtime scalability for high-traffic hospital environments

METHODOLOGY

- Domain Decomposition: Analyzed the legacy system to identify "Bounded Contexts," splitting the application into isolated Patient, Billing, and Auth domains.
- Microservices Development: Developed independent, loosely coupled services using Java Spring Boot, ensuring each service owns its distinct database (Database-per-Service pattern).
- Hybrid Communication Implementation: Engineered a dual-protocol system using gRPC for low-latency synchronous calls and Apache Kafka for asynchronous, event-driven updates.
- Containerization & Cloud Deployment: Dockerized all services and orchestrated them using AWS ECS Fargate to achieve auto-scaling and high availability.

SOLUTION ARCHITECTURE



Metric	Monolithic System	Microservices (Ours)
Communication	Slow (HTTP/REST)	**Fast (gRPC)**
Scalability	Manual / Difficult	**Auto-Scaling (AWS)**
Fault Tolerance	System-wide Crash	**Isolated Failure**

CONCLUSION

- Successful Migration: Transformed a legacy application into a scalable, fault-tolerant distributed system.
- Optimized Performance: Achieved <100ms inter-service latency using gRPC and asynchronous Kafka messaging.
- Cloud-Native Reliability: Validated production readiness with auto-scaling and self-healing capabilities on AWS ECS.

REFERENCES

- AWS Documentation, "Amazon Elastic Container Service (ECS) & Fargate," aws.amazon.com.
- Spring.io, "Building Microservices with Spring Boot," spring.io/guides.
- gRPC Authors, "gRPC: A high-performance, open-source universal RPC framework," grpc.io.
- Apache Software Foundation, "Apache Kafka Documentation," kafka.apache.org.

TECHNOLOGY STACK

AWS ECS

AWS Fargate

VPC

Amazon RDS