Rodrigues Mavita

February, 27 2026

CS 470 Final Reflection

https://youtu.be/e6fU8u9dYb0

Experiences and Strengths

Taking CS 470 has genuinely pushed me to think like a professional developer, not just a student. Deploying a full stack application in the cloud — from containerizing services to configuring S3 buckets and working with AWS — gave me hands-on experience I couldn't have gotten from a textbook alone. The skills I've built here, like working with cloud infrastructure, RESTful APIs, Docker, and serverless architecture, make me a stronger candidate for full stack or cloud-focused roles.

As a developer, my strengths lie in problem-solving and adaptability. I'm comfortable working across the stack — from the database layer up to the frontend — and I've gotten better at thinking about how systems scale, not just whether they work. I'm also detail-oriented when it comes to security, which is something I want to grow into professionally.

With this degree, I feel prepared to step into roles like **full stack developer**, **cloud application developer**, or even a **junior DevOps engineer**. Long-term, I'm aiming toward security and reverse engineering, and this course gave me a foundation that connects directly to those goals.

Planning for Growth

Thinking about the future of my web application, microservices and serverless both offer real advantages depending on what the app needs.

Microservices would let me break the app into independently deployable pieces — so if the booking service gets hammered with traffic, I can scale just that service without touching everything else. **Serverless** (like AWS Lambda) is great for event-driven tasks — things like sending confirmation emails or processing payments — where I don't want to manage infrastructure at all.

- **Scale and error handling**: With serverless, AWS handles scaling automatically. I'd use dead-letter queues and retry logic to handle errors gracefully without losing requests.
- **Cost prediction**: Serverless is harder to predict at high volume since you pay per invocation, while containers (ECS/Fargate) give more predictable monthly costs once

traffic patterns stabilize. For early-stage growth, **serverless is cheaper**; for consistent high traffic, **containers become more cost-predictable**.

- **Pros of serverless**: No server management, auto-scaling, pay only for what you use.
- **Cons of serverless**: Cold start latency, harder to debug, cost unpredictability at scale.
- **Pros of containers**: Consistent performance, easier local testing, predictable pricing.
- **Cons of containers**: More operational overhead, requires orchestration (e.g., Kubernetes or ECS).

Elasticity is a huge factor in my planning — the ability to scale up during peak demand and scale back down to save money is exactly what makes cloud-native architecture worth it. Pay-for-service means I'm not locked into over-provisioning infrastructure "just in case," which keeps costs lean while the app grows.

Overall, I'd likely start serverless for cost efficiency and simplicity, then migrate high-traffic services to containers as usage patterns become predictable.