Michelle Powers
December 15th, 2025
CS 470 Final Reflection
Presentation video: https://www.youtube.com/watch?v=zKHCEIwMUEc

This full stack development course has taught me many specific technical details about the way current cloud architecture works, and I think that the result will be that this experience will make me a lot more well-rounded as a software developer. Knowing exactly how all the moving parts such as lambdas, API endpoints, and database tables work and interact and how they can be configured for usability and security makes me feel confident that I have the foundation needed to design a professional full stack cloud project from scratch.

As a software developer, I think that one of my personal strengths is that I test code thoroughly and traverse through a project in very small increments. Moving onto the next part of a project only happens when I'm fully happy with how the last turned out, and I tend to fine-tune small details (maybe a bit *too much*) until it's practically perfect in terms of usability. Because of these qualities, I think that I would be most comfortable in a job working with low-level systems or UI design – admittedly two completely different subsets of software development, but I've learned that I enjoy parts of both after working on academic and personal projects.

I've learned that one large benefit of using a cloud platform such as AWS is that it handles scaling automatically. It's still important to monitor traffic to make sure requests aren't being throttled too frequently, and CloudTrail can be very helpful for monitoring API calls in a high traffic application. Scaling up an application also means that user permission management and error handling scale up with it, and for this the first measure I would personally take is to enable database backups to prevent potential data loss. CloudWatch can again be helpful for error logging as well, and for more debug data try-catch statements can be added to lambda functions to find out exactly *where* something went wrong.

Michelle Powers
December 15th, 2025
CS 470 Final Reflection
Presentation video: https://www.youtube.com/watch?v=zKHCEIwMUEc

Since serverless cost is based on usage rather than uptime, one way I would attempt to estimate this is by looking at the statistics given by AWS. Each identity page features usage data and how often functions are called (or endpoints are accessed) among other statistics, which can be used as an estimate of how they might linearly scale. Containers on the other hand rely on uptime to determine costs, and this can be predictable if a service is expected to be running constantly. The event-driven nature of lambdas and other serverless identities means that the traffic can differ day by day, and especially for a cloud application with many concurrent users this makes it best for services that are mostly "on-demand", otherwise a container may be ideal.

One large benefit of expansion using serverless cloud services is less manual management of maintenance tasks such as software and operating system updates, backups, and hardware costs and decisions. This gives developers more time to spend on planning and securely coding core functions. A downside is that there is a lot of configuring that needs to happen up front, and a relative learning curve when transitioning from MongoDB to DynamoDB without prior experience on how its queries and its key-value model works. Once everything is set up, however, the flexibility of being able to set up features case-by-case for each service is a net positive in the long run, and DynamoDB also becomes simpler through experience.

Overall, knowing exactly how an application is expected to grow is important for deciding whether to go serverless or stick with a traditional local approach using containerization. Pay-for-Service is great for unpredictability since you know you're never paying for more than you use, but otherwise elasticity can be cheaper if the audience or market for the application is known before/during development. Ultimately, it's important to consider both approaches, and I plan to use this new knowledge for my own projects in the future!