CS-470 Final Reflection: Cloud Development

Christian Wallace
Southern New Hampshire University
Video Presentation: https://youtu.be/795_sdsOUhk

Experiences and Strengths

Throughout CS-470: Full-Stack Development II, I learned how to design, implement, and deploy a cloud-native full-stack application using modern AWS serverless technologies. This course not only strengthened my technical skills but also taught me how to think critically about scalability, automation, and efficiency in distributed systems.

The most valuable skills I developed include:

- **Serverless Architecture Design: I learned to replace traditional backend servers with AWS Lambda functions that scale automatically and minimize maintenance.**
- **API Gateway Integration: I configured RESTful APIs that communicate securely between the frontend and cloud functions.**
- **Cloud Database Management: I transitioned from MongoDB to DynamoDB, learning how to structure key-value data models for performance and scalability.**
- **Security and IAM Policies: I applied the principle of least privilege to AWS IAM roles and policies, ensuring that each Lambda function had limited, task-specific access.**
- **Containerization with Docker: I packaged applications using Docker and orchestrated services with Docker Compose, which streamlined both local testing and cloud deployment.**
- **Elasticity and Pay-for-Use Cloud Concepts: I gained a deep understanding of how cloud providers dynamically allocate resources and charge based on real usage.**

Together, these experiences made me a stronger and more versatile developer, capable of architecting, deploying, and managing production-grade cloud applications.

My greatest strength as a developer lies in my ability to integrate full-stack systems—from the user interface to the database layer—while keeping scalability, performance, and maintainability in focus. I also bring strong problem-solving and debugging skills from my experience developing enterprise applications, where reliability and performance are critical.

These skills position me well for roles such as:

- **Full-Stack Developer**
- **Cloud Application Developer (AWS / Azure)**
- **Backend or API Engineer**
- **DevOps / Cloud Infrastructure Engineer**

Each of these roles requires knowledge of API design, deployment pipelines, and the use of cloud-native services—all areas that this course directly enhanced.

Planning for Growth

As the application and its user base grow, planning for scalability and efficiency becomes essential. This course gave me the foundation to plan for elasticity, cost efficiency, and fault tolerance in cloud-based systems.

Microservices and Serverless Efficiencies

Using microservices or serverless functions allows future versions of my application to scale specific components independently.
For example, user authentication, reporting, and data analysis could each be deployed as separate Lambda functions or microservices, minimizing interdependence and simplifying updates. This modularity also allows for rolling out new features without affecting other parts of the system.

Handling Scale and Error Management

To handle scale effectively:

- **Auto-Scaling: Serverless Lambda functions automatically scale based on request volume, removing the need for manual intervention.**
- **Error Handling: I would implement retry logic with exponential backoff and centralized logging through CloudWatch.**

- **Monitoring and Alerts: CloudWatch metrics and alarms would detect function failures or latency issues, automatically triggering responses or notifications.**

Predicting and Managing Cost

Predicting cost in serverless environments is more straightforward because AWS bills per execution and per millisecond of compute time.
In contrast, containerized workloads require continuous running infrastructure, leading to higher idle costs.
For applications with variable or unpredictable traffic, serverless is far more cost-predictable and efficient.

If my application were to scale to millions of requests, I could still calculate costs by estimating the number of Lambda invocations and the execution time of each. This makes financial forecasting simpler compared to managing container clusters.

Pros and Cons for Future Expansion

Approach	Pros	Cons
Serverless (AWS Lambda)	Automatic scaling, low maintenance, pay-per-use, no server management	Cold starts, limited execution time, potential vendor lock-in
Containers (Docker + ECS/Fargate)	Full control, predictable performance, portable across clouds	Higher maintenance, continuous cost, manual scaling configuration

For a fast-growing startup application, the serverless model offers the best balance between cost, scalability, and speed of deployment. However, for applications requiring long-running processes or complex networking, a container-based approach may be better suited.

Elasticity and Pay-for-Service in Growth Decisions

Elasticity ensures that computing power automatically adjusts to demand, which prevents wasted resources and ensures high availability.
The pay-for-service model complements this elasticity by charging only for what is used, making scaling financially viable.
Together, these two concepts are essential to designing sustainable, cost-

effective architectures that can handle future traffic growth without extensive redesign.

Conclusion

This course solidified my understanding of cloud development principles—from containerization to serverless computing—and provided hands-on experience building scalable, secure, and efficient web applications.
It directly strengthened my career readiness, giving me confidence in deploying real-world applications that can grow dynamically with user demand.

The combination of technical depth and cloud strategy thinking from this project has made me a more capable developer, ready to take on advanced cloud-based projects in professional environments.