Morgan Getkin

12/21/2024

CS 470 Final Reflection

https://youtu.be/qYWizHT-XF4

Experiences and Strengths

This course has significantly enhanced my skills as a software developer and equipped me with knowledge that will help me achieve my professional goals. One of the most valuable skills I've gained is proficiency in cloud-native development, particularly using AWS services like Lambda, S3, DynamoDB, and API Gateway. I've also mastered containerization tools like Docker and orchestration with Docker Compose, which are essential for modern application development and deployment. Additionally, I've become adept at designing and implementing serverless APIs, understanding their logic, and integrating them with frontends and databases.

My strengths as a software developer lie in my ability to adapt to new technologies, troubleshoot complex problems, and think critically about scalability and efficiency. Through this project, I've demonstrated a strong ability to architect applications using microservices and serverless models, which makes me well-suited for roles like cloud developer, backend engineer, or DevOps specialist. I'm prepared to assume responsibilities that involve designing scalable systems, optimizing cloud applications, and implementing robust security measures in a new job.

Planning for Growth

Cloud services provide a foundation for scaling and managing web applications effectively, and I've learned how to leverage these technologies for future growth. Microservices and serverless architectures, like AWS Lambda, allow applications to scale effortlessly based on demand while simplifying error handling. For example, AWS automatically provisions more Lambda instances during high-traffic periods and retries failed executions. Similarly, DynamoDB's auto-scaling feature adjusts read and write capacities to handle workload changes without manual intervention. These tools create efficiencies by reducing the need for constant monitoring and manual scaling.

Predicting costs involves understanding usage patterns and leveraging monitoring tools like AWS CloudWatch. Between containers and serverless, serverless is generally more cost-predictable because you only pay for execution time, while containers often require paying for reserved resources even when idle. However, serverless costs can spike unexpectedly with massive usage, so monitoring tools and budget alerts are essential for cost management.

When planning for growth, the pros of serverless include automatic scaling, cost-efficiency, and reduced infrastructure management. However, cons such as cold-start latency and potential vendor lock-in must also be considered. Containers, while slightly less dynamic, provide more control over the runtime environment and can be better suited for applications with specific configuration needs.

Elasticity and pay-for-service models play pivotal roles in future growth decisions. Elasticity ensures the application can scale up during peak times and scale down during quiet periods, optimizing resource usage and maintaining performance. The pay-for-service model allows for flexibility in budgeting, making it easier to allocate resources based on actual application needs, which is particularly beneficial for startups or businesses with variable traffic patterns. By applying these principles, I can ensure that the web application remains efficient, scalable, and cost-effective as it grows.