CS470 Full Stack II Final Reflection

Brad Mills

Professional Development and Goals

This course has greatly expanded my understanding of cloud-native development, preparing me for a future in roles that prioritize scalability, resilience, and efficiency. Mastering cloud service concepts, especially those related to microservices and serverless architectures, aligns well with my goal of becoming a skilled cloud software developer. These skills are highly sought after in tech-driven fields, and proficiency with cloud platforms like AWS enhances my employability.

Skills Learned and Marketability

Key skills acquired include building and deploying applications in a cloud environment, using microservices and serverless functions, and managing containerized applications with Docker and orchestration tools like Docker Compose. I also gained expertise in API design and integration, crucial for ensuring seamless interaction between the frontend and backend. These skills make me a strong candidate in today's market, where cloud-native and API-driven architectures are increasingly important.

Developer Strengths

My strengths as a software developer include adaptability, a solid understanding of cloud-based architectures, and an analytical approach to problem-solving. I am skilled in designing modular applications that can scale easily and efficiently. Additionally, my knowledge of cloud services, especially AWS, enables me to make strategic decisions regarding storage, compute resources, and cost optimization.

Prepared Roles

Given my experience, I am well-prepared for roles such as Cloud Application Developer, Backend Developer (specializing in serverless and microservices), and Full Stack

Developer. My skill set positions me to contribute effectively to teams focused on cloud-native projects, where scalability and cost-efficiency are paramount.

Planning for Growth

Future Use of Microservices and Serverless for Efficiency and Scale

To achieve efficient scaling, microservices and serverless functions offer compelling benefits. Microservices allow each service to be deployed and scaled independently, making it easier to manage updates and add new features. Serverless functions, like AWS Lambda, further enhance efficiency by executing code in response to specific events, which is cost-effective and scales automatically.

Handling Scale and Error Management

For scaling, I would implement autoscaling policies that monitor traffic and dynamically allocate resources as demand increases. AWS services like CloudWatch and Lambda's error-handling mechanisms can monitor application health, handle retries for transient errors, and alert on issues to maintain reliability. Using distributed logging and monitoring tools would further streamline troubleshooting and support error recovery.

Cost Prediction

Predicting costs involves analyzing usage patterns, estimating requests to serverless functions, and monitoring storage needs in services like S3. Containers may be more predictable for applications with stable workloads, as they rely on allocated resources. However, serverless architectures provide cost savings for unpredictable or seasonal applications due to their pay-as-you-go model, charging only during active use.

Pros and Cons of Containers vs. Serverless

Containers provide more control over runtime environments and resource usage, which is advantageous for applications with continuous usage. However, they require maintenance and can lead to overprovisioning. Serverless, in contrast, offers lower costs for variable workloads and requires no infrastructure management, but it may face limitations in execution time and environment configurations. Deciding factors will depend on the application's workload predictability and resource demands.

Elasticity and Pay-for-Service in Planning Future Growth

Elasticity and pay-for-service models are crucial for planning growth. Elasticity allows applications to handle surges in demand by dynamically allocating resources, ensuring reliability without overspending. The pay-for-service model makes scaling cost-effective, as charges are based on usage rather than fixed allocations. Together, these models allow me to focus on growth without the risk of overprovisioning or under-resourcing, providing flexibility for future expansions while managing costs efficiently.