Nate Bennett

Cemal Tepe

Full Stack Development II

Southern New Hampshire University

27 October 2024

CS 470 Final Reflection

https://youtu.be/9cKjsoj5Hpk

This course taught Full Stack Development skills and concepts, the topics that I learned about include Docker, Docker Compose, AWS, AWS API Gateway, AWS Lambda functions, and AWS IAM roles. Also, how to migrate a Full Stack web application to the cloud with AWS. Some of my strengths as a software developer might include efficiency, problem-solving, and adapting to different coding languages. Professional roles that I could assume based on the topics learned from this course could include a Full Stack developer and or a cloud based software developer.

Microservices and serverless architectures support scaling by default. AWS Lambda for example can scale based on traffic by default. Also, traffic is evenly distributed across multiple services, which can help services run efficiently. AWS Lambda functions can be configured to get potential errors, so that they can be viewed, which can help manage errors. Also, AWS CloudWatch offers monitoring metrics which could help with detecting errors. AWS uses a Pay-for-Use model that could help with predicting costs, as AWS customers pay based on cloud resources that are used. To try to predict costs, customers could find the average daily users of the application using AWS, as well as how many resources an average user uses daily through the application and how much that costs, and multiply the two. Containers could offer more predictable costs when resources are used constantly. Serverless can be more cost-efficient for unpredictable resource demand, based on the Pay-for-Use model.

Pros for serverless could include minimal operational overhead due to AWS handling infrastructure, swift scaling for fluctuating traffic, and the Pay-for-Use model that has customers pay based on resource use. Cons for serverless could include a limited runtime for functions, potentially more complex to debug, and higher costs for applications with constant demand. Pros of containers could include more control over the environment and software, better for complex applications with constant usage, and the ability to run the container across different environments potentially without configurations of the software. Cons of containers could include infrastructure management, fixed operational costs, and complex software management.

Elasticity could help applications scale based on demand, preventing resources from being over provisioned, and potentially limiting resource downtime. The pay-for-service model has customers pay based on service usage, which offers flexibility and limits starting costs. This could encourage efficient architecture designs. These models potentially support scalable, cost-effective application growth by scaling resources by default, and having customers pay based on the usage of resources.