Kevin Schroeder
February 2020
CS 470 Final Reflection
Link to Presentation

CS 470 Final Reflection

Experiences and Strengths:

- What skills have you learned, developed, or mastered in this course to help you become a more marketable candidate in your career field?
    - I learned how to use the Angular framework to develop a dynamic front-end
    - I learned how to use AWS as a cloud-based serverless hosting provider
    - I learned what DynamoDB is and what a single-table database model is
    - I learned how to create, read, write, and update DynamoDB databases
    - I learned what a Lambda is and how to use one
    - I learned how to securely connect a Lambda function with DynamoDB
    - I learned how to use Amazon's API Gateway to trigger a Lambda function when an event happens
    - I learned how to add security and authorization parameters to my cloud-based server, Lambdas, API Gateway, and DynamoDB

    - I developed my API creation skills
    - I developed my skills in both Express and Node.js
    - I developed my skills in creating and manipulating MongoDB databases
    - I developed my skills in connecting a front-end and back-end through a custom-made API
    - I developed my general coding skills, including syntax, commenting, and overall file structure
    - I developed my skills in creating Dockerfiles
    - I developed my skills in managing multiple containers and allowing them to communicate through a docker-compose.yml file
    - I developed my skills in running, stopping, and building Docker containers

- Describe your strengths as a software developer.
    - I am very adaptable, meaning that I can pick up new skills quickly and switch

tech stacks easily
    - I am equally skilled in front-end development as I am in back-end development
    - I am meticulous so I don't push anything to production until I know that it's as quality as possible
    - I am efficient with my time and ability to produce high-quality code in a short amount of time
    - I can read code easily, meaning that I can and am willing to frequently review other team member's code
    - I have some experience in DevOps so I can help set up a serverless architecture and teach others
    - I am curious and value learning new things that will help a product

- Identify the types of roles you are prepared to assume in a new job.
    - Minor DevOps roles
    - Back-end development
    - Front-end development
    - API creation
    - QA/Code review
    - Educating other team members
    - Setting up containers
    - Minor database admin roles

Planning for Growth:

- Identify various ways that microservices or serverless may be used to produce efficiencies of management and scale in your web application in the future. Consider the following:
    - How would you handle scale and error handling?

      Microservices are great tools to help manage scale because they break down a large application into smaller pieces. The smaller pieces handle only part of the application's functionality, meaning that they are less likely to get overwhelmed by usage.

      Elasticity also helps a server scale. Elasticity refers to changing the computing power available on the server over time to meet the usage demand. Traditional servers would be configured it increase to x computing power when the usage hit y. However, this traditional elasticity model is still at risk of sudden and major changes in usage that could cause provisioning. AWS's serverless architecture

takes elasticity to the next level with a pay-for-use model. This model is a payment method that charges based on usage. The practice is similar to that of utility bills, paying for only resources that are needed. In the pay-for-use model, there are no wasted resources, since users only pay for services used, rather than provisioning for a certain amount of resources that may or may not be used. Drastic changes in usage will not affect the end user's experience.

A microservices architecture also helps developers handle errors because it is easier for them to identify the source of an error when they know the specific microservice that was affected. I'd also recommend implementing an error-tracking software like Sentry.io. This is open-source software that will track eros that happens across all microservices and alert the developers via email. It contains detailed logs for the developers to identify exactly where in the code the error occurred. It's a very useful tool and can help organize error handling across multiple services

- How would you predict the cost?

  To predict cost, I would use the AWS pricing calculator. This tool allows us to custom create a quote based on our needs. We can add any AWS service and add details to the services on how much usage we expect. Form there, Amazon will give us an estimate on the amount it will cost to run each service.

  To make an accurate prediction, I'd use a predictive model for workflow cost from monitoring the current application. There is a really useful model made by Ernesto Marquez of Concurrency Labs that would be a great start to make a prediction (here's the link). This model calculates an approximation of monthly AWS charges for all EC2 resources that you've added to the model. It uses an AWS product called CloudWatch to calculate the recent usage for each service. It then projects the recent usage into an approximate monthly usage and calculates the prices using the AWS Price List API. This will be a great approximation of how much we can expect to spend each month.

- What is more cost predictable, containers or serverless?

  Both of these architectures are very hard to predict the pricing. Containers are typically used in serverless architectures, so they are still subject to the challenges of predicting a pay-per-use model. However, they add another level of complexity because calculating/predicting the price per container is breaking up the pricing model into even smaller pieces. Containers could live in clusters, and determining the use of each container in that cluster would be a very difficult task. So the more

cost predictable option is serverless.

- Explain several pros and cons that would be deciding factors in plans for expansion.
    - Pros
        - Cost-effective in the long-run
        - Faster response times with a microservices architecture
        - Modern server architecture keeps up with the times
        - Frequent tasks such as managing the operating system and file system, security patches, load balancing, capacity management, scaling, logging, and monitoring are all offloaded to a cloud provider
        - Don't need to hire someone to manage the servers
        - Data in the cloud is constantly backed up, which decreases the risk of losing data permanently
        - The servers are easily accessible by anyone from anywhere (with the right permissions of course). The only requirements to access the servers are security information and an internet connection
    - Cons
        - Less control over the configuration of the server
        - Less control over server location and hardware
        - Less ability to debug server issues
        - At the mercy of the internet connection of the hosting provider
        - Will typically require companies to split their monolithic applications into microservices. This will come with a learning curve and significant up-front development work
        - Harder to predict the cost

- What roles do elasticity and pay-for-service play in decision-making for planned future growth?

  As discussed in a previous answer, elasticity refers to changing the computing power available on the server over time to meet the usage demand. Traditional servers would be configured it increase to x computing power when the usage hit y. However, this traditional elasticity model is still at risk of sudden and major changes in usage that could cause provisioning. AWS's serverless architecture takes elasticity to the next level with a pay-for-use model. This model is a payment method that charges based on usage. The practice is similar to that of utility bills, paying for only resources that are needed. In the pay-for-use model, there are no wasted resources, since users only pay for services used,

rather than provisioning for a certain amount of resources that may or may not be used. Drastic changes in usage will not affect the end user's experience.

Elasticity and the pay-for-service model give companies peace of mind knowing that no matter what future growth occurs, they will not be over or underpaying for their servers. They are still going to be paying for the exact usage needed and will not be risking over or under-provisioning. However, the payment model may scare companies since they will not be able to accurately predict and budget for future server expenses. If they decide to transition their app to a serverless architecture, they are going to assume that it will have positive cost benefits in the long run without being able to know with 100% confidence.