David Varella

12/18/2024

CS 470 Final Reflection

https://www.youtube.com/watch?v=PakUTRVnWRw

Throughout CS-470, we worked to migrate a full stack application using cloud storage and a cloud database with AWS. The skills I personally learned, while not mastered, include the use of S3 storage buckets, APIs, and Lambda functions. Also, I believe that the knowledge of AWS functionality, such as testing, CORS, deployment, and how to set up the backend and frontend are beneficial to my future in software development.

As a developer, I believe I understand most services and functionalities that I have come across so far, such as the ones we learned during this course. Having experience with creating full stack applications, and knowledge of serverless architecture is also in demand, and I consider these areas to be among my strengths. A couple other strengths I have that aren't necessarily related to this course are things such as programming language knowledge (mostly Python), and understanding secure coding, etc.). I believe secure coding is extremely important in a software environment.

In a new position, I would like to think that I am prepared for a backend or frontend development position. While I have experience from this course and the previous course with full stack, I don't know if I would be fully prepared for a full stack dev position.

Using microservices or serverless architectures can help handle scale and error handling. With scaling, developers can add more instances of microservices across multiple machines or

containers, which is considered horizontal scaling. Horizontal scaling includes strategies such as auto-scaling, which automatically changes the number of instances based on requests or CPU, DevOps using CI/CD pipelines, and load balancing. To handle errors, some serverless-based options include function execution timeouts can prevent blocked resources and limit the number of function executions to prevent overloading, while microservice-based options can include health checks with Kubernetes, isolating services to prevent other microservices from failing, etc.

Microservices and serverless architectures have different cost structures, making it challenging to predict overall expenses. Microservices are typically billed based on factors like the number of containers, resource allocations, and storage usage. Tools like the AWS Pricing Calculator can help estimate costs under various scenarios, allowing for better planning and scaling. In contrast, serverless architectures operate on a pay-per-use model, where charges are based on the number of function invocations and memory consumption. This pricing model is particularly beneficial for applications with fluctuating or unpredictable traffic, as it ensures you're only paying for actual usage.

When comparing containers to serverless for cost predictability, containers are most likely more predictable. Containers charge based on memory, storage, and other resources, which is easier to predict compared to serverless, where the number of function calls is more variable.

If a development team is planning to expand, there are different pros and cons for both microservices and serverless architectures. Since microservices are independent of each other, they can be scaled up or down separately without affecting other services. These microservices are also easily updated and easy to develop and deploy since teams can work on numerous services at the same time. Developers can also have different security features/settings to

prevent unauthorized access to the entire system, in case of a breach. While there are many benefits, having a great deal of microservices can also be difficult to manage. Communication between services can become confusing, and having many services can increase long-term cost.

Serverless architecture is very efficient with its pay-per-use model. It is very cost effective, and automatically scale functions based on traffic. Having serverless architecture also removes the need for managing the infrastructure, so developers can focus solely on the logic and functionality needed. It's also extremely reliable and has high availability of data. Some cons, though, include "cold start" times, which occurs when functions and code are not constantly running, so it must "boot up". This may cause degraded performance. There's also a concern about security, a term known as multitenancy. This occurs because the serverless provider often shares their cloud with multiple companies; data could potentially be exposed if the servers aren't configured correctly.

Elasticity and pay-for-service are very important in decision making for planned future growth. Elasticity is the ability to dynamically scale up or down to manage a workload efficiently. This helps with traffic management and cost efficiency and allows can even help microservices optimize where resources are being used. Elasticity is also generally already implemented with a serverless architecture, since serverless automatically scales based on traffic needs.

Pay-for-service is a pricing model used in both microservices and serverless architectures, where companies are only charged for the resources they actually use, such as storage, memory, and compute power. While this model is mostly known to be used with serverless architectures, where functions are billed based on their usage to ensure cost efficiency, it can also be applied to certain microservices. In microservices, individual services can be

configured to use the pay-for-service model, incurring charges only when resources are utilized. This approach helps avoid paying for idle resources, ensuring that costs align with actual usage.