Cody Lake

12-22-2024

CS 470 Final Reflection

https://youtu.be/UfIwqPIXa8o

Some of the things I learned in this class were to create a serverless application, and to create applications using AWS. I think some of the things that made this job easy was using the AWS services like lambda and S3. These made my job easy because as a programmer there is lots of times that you need to be precise in your coding and naming. With the premade AWS services there is little code that I have to manually write, meaning there are less areas that I have to mess up because if I didn't name something the same that is an easy fix. Some of my strengths is that I like everything to look clean so I have a tendency to create using the same capitalizations and characters on all of the projects. In my mind it is easier to keep a naming scheme rather than implement a second different one so even if it is a scheme that I wouldn't personally use I would rather keep up the scheme to keep everything uniform and clean. Currently I feel that I understand the how and why of how things work and even though I think I could put this into practice for a job I would need more time to get everything I need done correctly. I would prefer to be an apprentice so to speak to someone for a while and learn and grow my skills on the job under the tutelage of an experienced programmer.

To grow my application we can employ microservices or serverless architecture to increase the efficiency and help to manage scaling the application. By using microservices we can allow individual services to scale without effecting the others should that one service get more traffic than the others. We can also use them to deal with error handling. By implementing circuit-breakers we can prevent cascading failures, and by using distributed tracing and centralized logging we can find and fix problems quicker. If we used a serverless architecture scaling is automatic if the service is idle then it is not being used at all and it can be ramped up based on the demand of the service. In terms of cost I would recommend using serverless as the cost is tied to the actual time of use of the service instead of paying for an allotted amount of resources at all times. It is easier to predict the cost using microservices as the rate you pay is what you get even if you don't use all of it. With serverless you could have a very cheap cycle as opposed to one that had a lot of traffic and cost more to operate. Some pros to expanding using microservices are that you have a very fine control over the scaling and resource allocation when scaling. It is easier to integrate with legacy systems, and if is more flexible in a technology stack for different services. Some cons to using microservices to scale are that the code required to scale gets more and more complex and harder to manage, and the time to implement and deploy updates is higher compared to serverless. Some pros to scaling with serverless is that the scaling is automatic. This means that if there is a large load on the server then it will scale up to the demand but if nobody is online at the time then it will scale down to reduce cost. This pay as you go model minimizes the upfront costs. Cons to serverless include the potential for a higher cost on sustained workloads, and sometimes there are limited amounts of execution time and constraints on resource usage. To be elastic in future growth opportunities serverless is the way to go as it scales automatically. This is more advantageous for unpredictable or a burst of workloads, where as to remain elastic microservices will require and explicit

configuration to be elastic. A good way for startups or smaller teams to get started without a lot of capital investment is to use a pay for service. This is essentially what a serverless architecture is.