CS 470 Final Reflection

CS-470-T4248 Full Stack Development II 23EW4

by. Corey Nance

4/20/2023

https://youtu.be/lIr71TWguak

The focus of this course is to learn full stack development with technologies such as Docker Containers and Amazon Web Services which are two of the latest technologies that will help to become a more marketable candidate in my career field. Through this course I have learned exactly how Docker Containers work, which developed my understanding of its importance. Understanding AWS and how to migrate applications to those services will help very much to become a more marketable candidate. Being able to develop an application in multiple ways would be an asset to most if not all development companies.

Some of my biggest strengths as a software developer start with the mindset of not being willing to fail. If I do not currently have the knowledge to solve a problem, I can research many possible solutions to solve or partly solve the task. Understanding what I need to get done will help me to research each step and then be able to combine them together for the exact task I am looking to get done. The types of roles I am prepared for range from just doing databases to creating an entire full stack application. To go along with the full stack development, another role would be creating AWS applications or using Dockers to create containers.

Overall, microservices and serverless architectures are able to help with scaling a web application due to the service user not needing to obtain and maintain their own servers. Security

wise, with the microservice and serverless providers, they would handle part of the security, but the developer should still develop their own security measures. It is very hard to predict what the cost would be, but using a pay as you go set up would help manage the cost as one grows. With containers one would have to pay up front and somewhat know what the demand will be but with serverless, it grows based on demand. With that being said, containers are more predictable because it's not an endless supply. The containers can only hold so much but with the serverless set up being able to take on all traffic, random spikes could make it so that serverless is a little less predictable.

When it comes to expansion, the objective would be to be able to scale the application with as little down time as possible. With containers a huge benefit would be that the existing application can be easily converted into containers and then scaled and deployed fairly quickly. This approach allows the application to still be able to run on any OS without major refactoring and be more resource efficient. The downside of using containers is that due to them being very lightweight, the security of the containers is not very good, each container uses the same kernel, so if the kernel becomes vulnerable then the containers also become vulnerable. Another downside is that the cost savings gained using containers could be spent training or acquiring developers that can work with containers. Containers have a very difficult learning curve. When it comes to using a serverless set up for expansion, The main benefits are that the security is usually very good and the bulk of it is handled by the serverless provider. With containers one has to spend money on decent hardware that can handle the containers but with serverless, that need is not there, one can acquire more resources as needed and on demand. The downside to using a serverless set up is that an application at times may have to be completely refactored in order to be serverless, which could take longer to develop, which takes longer to deploy. Other

downsides would consist of reliance on the serverless provider and their services and the requirement to always be connected online. Overall, for any expansion, it all depends on what the end goals are for the application, those goals would be what determines which expansion plan is better.

Overall, when a company is looking into possible expansions, usually the driving factor is based on resource demand and application traffic. This planning can involve the need to purchase additional resources whether it is hardware or staffing. With expansion, there is always the risk that the plan does not go perfectly which can cause an over or under allotment of resources. This is where having an architecture that has enough elasticity to be able to adapt quickly to changes to make the expansion more efficient. Understanding that at times expansion or application traffic could have spikes and not be a linear increase, using a pay-for-service can help account for the changes in growth without the organization needing to be over resourced to account for the spikes. Being over resourced can help to handle spikes but then the additional resources then go unused. For these reasons, using an architecture that can respond to changes quickly and allow for resources on demand is a crucial part of decision making for planned future growth.