My name is Luis Cotto
Date: 06/26/2022
CS 470

It has helped me gain a solid understanding of how to create a MEAN stack application, which has made me a more marketable candidate in my sector of work. I've developed skills as a software developer in areas including implementing security, creating databases, and creating APIs. I would assume the tasks of a web developer, web designer, software engineer, and API specialist in this new position. I can succeed in these professions if I develop the talents. I'd create the program's architecture in a way that can manage better error reporting as my first step in tackling scale and error handling in an application. This would involve using naming standards that are consistent with other systems in the same domain so that anyone working on the system can quickly comprehend the source of the error. The version information would then be included so that it is obvious on all systems. In addition, I would gauge application costs first and foremost based on its scope. If the application's actual architecture is more complex than usual, I would design it to be more complex than a small-scale application. The serverless charges, which entail the pay-as-you-go option, would also be included. Serverless is preferable to containers for the most cost-predictable alternative because it helps to reduce resource waste. Users may pay more to use containers because they run continuously. Describe a few benefits and drawbacks that would affect the decision to expand. If I were to expand my software, there are several advantages would involve expanding our consumer base, boosting our inventory and resource levels, and giving you complete control over the expansion. The fact that it will cost money to make money and the length of time it will take to create the expansion are some drawbacks. Finally, elasticity influences price because it helps determine how many resources will be needed to support expected future development. It permits flexibility in expansion for pay-for-service. By doing this, you can expand without having to spend too much or shoulder too many costs.

We learned how to deploy an application and how to use Docker Compose to construct various containers. Additionally, we learned how to manage the various AWS services and how they interact to build cloud-based web applications. Our files and data are stored online in an S3 bucket that we learnt how to create. Functions that will communicate between each service were developed using Lambda. The client could connect to our web application and then to AWS using API Gateway. Finally, we used a single-table model using DynamoDB as our database to store the queries and responses.

Experience and Strengths

More than just how to move a web application to the cloud was taught to me in this training. It taught me how to read from documentation for the earlier sections and to constantly double-check sources from the most recent versions of a software. Additionally, I've gained knowledge with how each company's layout appears and learnt how to traverse a few documentation

pages (Docker Inc. and Amazon). I've learned how to use Lambda for its intended purposes, even though I still don't fully grasp the technical details underlying how the programs communicate with one another. I've acquired the abilities to design and run a I got my first experience defining standards and roles for folks who are developing with the project when I worked on a cloud project on AWS. Additionally, I now understand how API Gateway functions and what distinguishes DynamoDB from its rivals. Additionally, I've acquired certain HTTP protocol tips that I first learned while testing my application through API Gateway.

Debugging and retracing my steps are two of my talents as a software developer. I've learned how to effectively describe services and how they operate from a high-level standpoint, but I'd also like to understand more about the low-level side's specifics. My ability to handle data and visualize it on a chart or graph has always been my greatest strength. When creating an application, I can see myself leaning more toward the back-end elements. Even though I enjoy designing web page layouts because I am an artist, I would also like to study how the back-end functions because my interest in this topic was sparked by my comprehension of it in the previous two classes, I'm constantly learning how to write code that is as effective as possible while adhering to best practices. Aside from writing the yaml files for Docker Compose, I think my best talent as a programmer would have to be my organization in code, but it was not something I had much experience working on.

To create an engaging web page for the front-end, my artistic side would benefit from learning CSS and HTML. My desire to learn more about how the back-end functions and how the program is something I should explore. Overall, the course has provided me with fresh knowledge that will help me in my career as a web or full stack developer.

Planning Growth

The fact that serverless applications focus more on the application itself and less on the server side is among the most crucial things I've learnt about them. Since developers and organizations do not have to handle many server-related tasks including managing outages, the load balancer, traffic, and API connection, I think this is one of the strongest arguments for becoming serverless. Instead, whether it is Amazon's AWS or Microsoft's Azure, different businesses offer different services to help a software go serverless. Additionally, serverless might already include a ton of practical features like graphs to monitor traffic and the volume of requests Going serverless, in my opinion, would be advantageous for a business that needs to focus more on the front end of operations than the back end.

Serverless also scales well, which is an advantage. AWS can scale the incoming requests well enough to prevent the servers from going down and crashing, regardless of how much inbound or outbound traffic is there. Sadly, this is also where some of the disadvantages are found. AWS or other service providers would have to deal with the problem of locating the precise server-side error. I was only able to use Firefox's Developer Tools to debug my program in this course to figure out why some elements wouldn't load correctly. I would need access to the server's source

code to get into the details, but for this project, I had no trouble doing that. Costs are determined by a fixed rate offered by AWS.

It's a tiny amount, but it scales according to how well each service performs, such as API Gateway calls or Lambda executions. Serverless applications have variable costs, while containers have a fixed cost regardless of whether they are running, in my opinion. However, because you only pay for what you use, serverless computing is more affordable and probably easier to maintain. If a serverless or containerized application doesn't receive any traffic but is While both are "operating," the container must pay for the server space it has consumed, whereas serverless is exempt from payment because no requests were made.

I think serverless fulfills the requirements for satisfying user demand when deciding to increase the available space for a server. The benefit of expanding is that it keeps the site from crashing while allowing higher volumes of traffic to pass through. The only drawback I can see to growing a server is the increased complexity it introduces for users with access. Developer access may need to be expanded to add room to new areas of the server, or the new area may conflict with components of the front-end or back-end. I think that a website's sporadic fluctuations can be fairly maintained thanks to the elasticity offered by serverless providers and their pay-for-service business model For instance, it would be wise to employ this strategy so that you don't lose out on possible sales because a server crashed and prevented other users from logging on during the holidays or around Christmas, when many stores may experience a spike in customer demand.