John Brungard

11/29/2023

CS470 Final Reflection

Video Link:

https://youtu.be/AW7lhsrwh_U?feature=

shared

After completing the development of a full stack web application in the cloud, I have learned and developed skills to help me become a more marketable candidate in my career field. One such skill is familiarization with using Docker Desktop. I became familiarized with the Powershell commands to create a default container such as: docker pull, docker run and docker exec. Using these, I was able to successfully run MongoDB inside a default container. I also was able to learn how to create containers for an Angular Frontend and a MEAN REST API backend. From there I developed skills to create a shared Docker Network and a Docker compose script for backend containers and frontend containers. I did this by learning how to setup YAML files to load into a container to define the services, network, and storage volumes the application would use. I also was introduced to concepts of preparing angular code for Amazon Web Services. This included creating an S3 Bucket, deploying my application, and configuring my S3 bucket as a website as well as the security for it. Within AWS, I also developed skills to create a serverless compute function in the form of Lambda that I was also able to test. It was also within AWS that I learned how to create and test an echo function from the console. However, one of the biggest developments I made in this class was creating an API using API Gateway. This included wiring my Lambda to my API, testing my API with its call to the Lambda, and deploying the API. One skill I built upon is my interactions with DynamoDB and its attributes. This was done by creating a questions table, adding items and attributes, and creating an answer table. I revisited Lambdas after doing this to build a Lambda called TableScan that can query either the questions or answers including the filter logic. It was during this event that I also learned how to create a security policy that defined permissions for my Lambda and the involved role to define what can be done in the role and under what conditions. Finally, I learned how to establish an S3 server to

host a static Angular website on the front end and how to migrate all the server-side business logic for the backend and API.

Some of the strengths I feel I had as a Software Developer through learning and developing these skills was the ability to debug and troubleshoot. As with nearly any application, I ran into issues that I had not fully expected. One of such was issues with CORs not showing my questions and answers both when I ran the application from Docker and when I ran it inside AWS. Because I had an idea of why the issue was occurring, I found the solution by running docker compose up for my containers and redeployed my API which solved the problem. When faced with different steps needed than what was in the guide involving policies, I also troubleshooted and found that the lab role did not need policies attached as per the instructions. Another strength I have is attention to detail. Because the guides were very detailed on what actions to perform, I had to follow them thoroughly and perform the actions exactly as described. I also found issues with code formatting from the guides that I was able to successfully edit to have the project run successfully. This also included frequently testing the application for functionality.

I am prepared to assume a few different roles in a new job. One would be as a cloud developer due to the development of the full stack web application into the cloud. Another role could be a cloud security engineer as I worked on IAM policies and permissions that helped dictate the authorization of certain roles within the web application. This was also evident when creating the policy for the S3 Bucket and Lambdas. Another role could be a cloud network engineer as I was able to configure a bridge network using docker so containers could communicate with one another. There was also the integration of the existing networks between Lambda and DynamoDB for the questions and answers table using API. An additional role could

be a cloud consultant as I learned issues on performance, security and data migration and could help with the technical expertise and strategy with deploying AWS cloud solutions. Finally, a cloud data architect would be a role I would be interested in and could take on in a new job with my exposure to both MongoDB with Docker Desktop and with DynamoDB using AWS. I learned the differences and capabilities of both NOSQL databases and would now be able to make better informed decisions about using either of them.

Though I have learned much from the course, there is still some planning for growth. I would handle scale and error handling by following AWS documentation concerning the error and the best mitigation strategies. For example, when you invoke a function, there are two types of errors that can occur which are invocation errors and function errors. Invocation errors typically involve issues with the request, caller, or account. They include the error type and status code in the response so you can pinpoint the error and make the correct adjustments accordingly. Function errors happen when the code or runtime returns an error. They do not return a status code like invocation errors but by returning the error showing a header named X-Amz-Function-Error and a response formatted in JSON with the message and other details. Knowing the type of error and being able to refer to the documentation for how to resolve it is how I would handle error handling. Concerning scaling, using AWS Auto Scaling is one option as it monitors your system to match your given performance levels. When your demand skyrockets, auto scaling dynamically increases the size of constrained resources so you can continue a high quality of service. There are four primary kinds of autoscaling: manual scaling, scheduled scaling, dynamic scaling, and predictive scaling. Knowing which one would best suit your needs can be crucial to your application's success. Manual scaling is where the number of instances is scaled manually. Scheduled scaling is best used when traffic occurs at a specific time as you can set your scaling

to happen during these moments. Dynamic scaling is when instances are changed automatically, and predictive scaling is using machine learning algorithms to program the desired instances and future traffic is predicted for the correct number of instances.

AWS helps predict costs differently than other services. Costs are compiled based on the resources that you use up each month instead of being a flat monthly or weekly billed service. With that said, predicting costs with AWS usually involves a forecast. A forecast is the prediction of how many resources you will use over a forecasted period based on past usage. It will create a bill as well as set alarms and budgets based on these predictions. The prediction interval of 80% is common for cost explorer forecasts. If AWS doesn't have enough data on your usage to forecast the 80% interval, then a forecast is not provided. Another option is to use the AWS Pricing calculator, which provides an estimate of cost by month. The AWS Pricing calculator is free to use and comes with many features. One such feature is that you can export your estimates in pdf or csv format to share locally with others. Estimates can also be sorted into groups and saved to view later. If I had to predict costs, these would be the two options I would plan on using.

When looking at cost predictions, I would argue that containers are more predictable as they charge you for the entire time your servers are running while serverless applications are more cost effective because you only pay for the time your application is executing. Containers also cost more resources such as CPU and memory so your bill will likely be higher compared to serverless. This advantage for serverless comes at a disadvantage of not being quite as portable and scalable as containers, however.

There are several pros and cons when deciding factors in plans for expansion. One of the deciding factors would be security. As an application would grow, so would the need for security

as more popularity would involve more likely cases of breaches and other attacks. Security costs money, which leads into another con in planning for expansion which is cost. Even though some applications like serverless applications have you pay for resources as you use them, there is usually a positive correlation between popularity and total cost. That's why it is careful to plan your growth gradually or at a pace that permits the budget of the company. A final con I will share is the introduction of bugs to the application. As the application grows and more features are added, there is always a risk for bugs and errors to be present that can affect even the older features of the application. Employees may have a harder time troubleshooting this during a expansion transition and this can be harmful to sales. Despite the many cons I shared, there are also some deciding factors that are considered a pro in expansion. The first one is the flipside to the security con in that updated and new software is typically more secure as cybercriminals have not yet had the time to find and exploit security vulnerabilities. This also leaves time for employees to reverse engineer the software before unauthorized users do and find possible areas of weakness within the new software. Also, even though costs can be considered a con, upgrading to newer software can also be considered a pro as software without any more manufacturer support can quickly fail and cause issues as vulnerabilities are exposed but not resolved. Efficiency and productivity can also be boosted with software upgrades as growth brings in different workers with different strengths and software also becomes more intuitive and streamlined.

Factors like elasticity and pay-for-use/pay-for-service modeling come into play when considering planned growth. Elasticity is the ability to manipulate the resources a cloud-based application uses. For this, the available resources try to match the current demand as closely as possible. That means deprovisioning when demand is low and provision when additional

resources are needed. The Pay-for-use model does not assign a fixed fee per a certain time iteration. You pay for what resources or compute power you use. This means you will likely pay more when demand spikes and resources are allocated to meet demand but can pay less when demand lowers. These factors make planned growth adaptable and dynamic so there is not as high a need for a fixed number of resources to be available during growth even though predictions and a minimum number of resources to forecast may be useful.