Abdelladif Moumbagna

08/21/2024

CS470 Final Reflection

https://youtu.be/z1aoJR9A124

Experience and Strength

The CS 470 (Full Stack Development II) course provided me with a general idea of how cloud development is done, its advantages and disadvantages over self-hosting, and some of the considerations to keep in mind when developing a cloud-based application. I have learned the process of containerization, orchestration and how to port an app to the cloud. While porting our application to Amazon Web Services, we learned about some of the services it offers. These services include Amazon S3, DynamoDB, API Gateway, IAM, and Lambda. We've learned about their use cases, benefits, and how to implement them in our application.

My strength as a developer lies in my ability to think critically and logically to solve problems. I am also passionate about continuously learning current and emerging technologies as they become mainstream. From the course, I should be able to assume beginner or entry level cloud application developer role for my knowledge with databases such as NoSQL databases (MongoDB) and SQL, programming languages, and cloud service platform. From previous courses I should be able to get roles as frontend developer and/or backend developer, and tester roles.

Plan for Growth

With serverless, cloud providers taking care of the infrastructure can be used remove some operational overhead and would allow me to focus on the quality of code and functionalities. Serverless applications can also scale automatically with demand, which ensures that resources are allocated dynamically as needed for optimal performance. This ensures a great user experience, by running smoothly regardless of user demand. Billing with serverless applications uses a pay as you go model where you only pay for compute time consumed, which can be more cost effective for applications with fluctuating demand but offers less predictability of cost.

Microservices can be scaled independently based on demand. Scaling here is done by adding more instances of a service to handle increased demand. It provides fault isolation which prevents faulty services from affecting other services or the entire application.

Predicting cost here is based on estimation of the resources (CPU, storage, memory…) that each microservice will consume. Cloud providers often offer tools to help forecast expenses based on usage patterns. This is a more predictable approach in terms of cost since you pay for reserved resources regardless of usage.

Some deciding factors in plans for expansion include

As Pros:

- Scalability: Cloud applications can be easily scaled up or down based on demand. This allows
- Cost efficiency: Cloud providers being responsible for managing the infrastructure reduces the need for significant upfront investments in hardware, security, and maintenance.
- Customer/User reach: Cloud applications can be accessed from anywhere with an internet connection, facilitating user outreach and collaboration.
- Disaster Recovery: Cloud providers offer robust disaster recovery solutions enabling data backups and restorations in case of an emergency.

Cons:

- Lack of control and reliability concerns: While cloud providers offer high availability, they also experience outages, which can be disrupting for operations.
- Vendor Lock-in: Migrating to a cloud provider can sometimes lead to dependencies on that provider's services. This can make it difficult to migrate to another provider or back to self-hosting if needed.

Roles of elasticity and pay for service in decision-making for future growth.

Elasticity ensures that applications can handle varying loads efficiently, while the pay-for-service model helps manage costs and resources effectively, making them both essential for planning future growth.

Elasticity plays a role in applications scalability as it allows applications to scale accordingly based on demand. It ensures the ability to handle high demand without over-provisioning resources during low demand periods. This in turn allows optimal performance of the application by ensuring that resources are available when needed. Additionally, by dynamically allocating resources we only get billed based on time and resources used, which makes for a more cost-effective application.

The pay for service model is a model where you only pay for the resources used. This provides great flexibility in budgeting and is particularly useful for applications with varying

demand. It also provides some flexibility to experiment with different services and configuration without worrying about long-term commitments or waste.