Zavalla Huggins

04/23/2023

CS 470 Project Two Conference Presentation: Cloud Development

https://youtu.be/6HuhUPrPo0Y

Slide Two:

Hello everyone. My name is Zavalla Huggins and today we will be reviewing the development of our full stack application and what steps we used to migrate it to the cloud. In our process, we used different AWS services such as S3, Lambda, API Gateway, and DynamoDB which we will expand on later in the presentation.

Slide Three:

There are many different models available to follow when migrating data to the cloud. In this project we used SaaS, also short for Software as a Service. In this instance, we are only migrating services that we need, which mitigate the amount of excess data being transferred. During this process, we containerized the necessary components of our application to transfer it from our local machine to the cloud. Each container will hold pieces of the application and host the environment in which it runs.

To do this, we used Docker Desktop, Docker Compose, and MongoDB. Docker Desktop and Compose helped us containerize our application and MongoDB is where the database data was stored.

Slide Four:

As mentioned previously, we used Docker Compose to build our containers in our application. Some features that Docker Compose allows us to do is single host deployment. Also, the configuration is faster, along with a higher level of security. When using a single host deployment, all parts of the

application can run on one machine. Docker allows us to configure YAML files in which we configure our application. In these files, we can identify the API container and which port is being listened to, then link that port to the database. We then would attach the containers to the cloud network that has been created.

Slide Five:

The serverless cloud is not truly serverless. The server is managed by a larger data service, in this case, Amazon S3. This allows us to save money on storage. Instead of installing our own infrastructure to hold all our data, we can store it using Amazon S3, who will allow us varying amounts of storage space depending on our needs. This also allows our project to be scalable. Upgrading storage space is infinitely easier since the in-house hardware does not need to be modified. It is as simple as changing our subscription plan, in turn saving thousands of dollars in the longevity of the project. Since the storage logistics would be managed by a third-party service, allows for less time to be spent managing storage. The data is also less likely to be compromised since it is the third-party service's responsibility to protect our data.

Slide Six:

In AWS, we can build our REST API using Lambda function. A Lambda function is much like a regular function in coding. We can program it to perform an action, in this case, the CRUD functionality. We added functions to create, read, update, and delete records within our database, held in DynamoDB. The different API models were built with the according GET/PUT/POST/DELETE functions. After creating each function, we needed to add the permissions, which allowed each function to access our DynamoDB table. Then lastly, we needed to enable Cross-Origin Resource sharing. In AWS's API structure, we added an OPTIONS method within each method we created.

Slide Seven:

Both MongoDB and DynamoDB can accept .JSON formatted data, which was what we were inserting. They are both also NoSQL databases. A difference that the two have is that the storage availability in MongoDB is limited by the developer's available storage space on local machines, whereas DynamoDB's storage availability is managed by AWS, which is cloud based, meaning we can scale up the amount of storage needed with ease. To query data in DynamoDB we used the Lambda functions we created.

Slide Eight:

As mentioned previously, the cloud-based storage options are much more flexible than the locally managed ones.

The use of storage is tracked on a month-to-month basis, which means we are only paying for what we use, instead of having to incrementally increase every time our project gets larger in the case it is stored locally. In comparison, if we were to scale up the storage support, the infrastructure is more expensive and has a higher likelihood of not being used, depending on how much of the increased storage space is utilized.

Slide Nine:

Since this data is managed by a third-party source, we want to be sure our app is secure. In that case, we created permission policies using AWS's Identity and Access Management service for each function to limit what the user can do. In our case, this includes things such as the deletion of the entire table and modifying large amounts of data in any given query, and ensuring the user is only accessing the needed database.

When we created the lambda functions, we created roles for each function. We then created one permission that only allowed basic CRUD functionality within the database. For each role we created for the Lambda functions, we attached the permission policies to the functions, which then allows the user to have the necessary permissions to complete their queries.

These permission policies help us secure the Lambda functions to API Gateway services by using multi-factor authentication, user/error logging, and managing user-related data. Lambda API also practices using a transport layer between the endpoints. Using Amazon S3, Lambda is already secure, but we still need to add permissions restricting each user's access a bit more to protect our projects.

Slide Ten:

My main takeaways from this project is that migrating our project to a Cloud environment allows our project to be more scalable, and in turn, cost effective. When creating the Lambda functions and API gateways, I found that using AWS services made each step frictionless, with plenty of documentation to help with any project hangups. This leads to creating security policies. Since understanding how the security policies could be added and tailored to your project's needs, it made for a more secure project.

Thank you for listening to my presentation.