Project Two Transcript

Justin Starr

Department of STEM

CS 470 – Full Stack Development II

Professor George Johnson

October 20, 2024

Project Two transcript

Video Link:
https://youtu.be/5QA41e0MqoM

Slide 2 – Overview
Hello and welcome! My name is Justin Starr. Today, we will discuss the process of migrating a full-stack application to an AWS serverless solution and the intricacies involved.
This includes packaging our application's front end, back end, and database into containers, which can then be deployed to the cloud.

We will discuss how the application is deployed using Amazon S3, how S3 storage compares with local storage, and how we will use AWS API gateway to securely access Lambda functions, which are triggered by events, and used to interact with the database on the backend.
Additionally, cloud-based development principles will be discussed, as well as how we can secure our cloud-based application.

Slide 3 – Containerization
The model that we are using is a common model used when deploying applications to serverless environments, and it is called rehosting, which is a lift and shift model. Essentially, we are going to package up our entire application, along with all of its dependencies into easy-to-manage containers, that can then be deployed to the cloud. By deploying our application using containers, we ensure that the application will run in any environment, regardless of its settings, a problem that containerization aims to solve.

The tools that you will need to accomplish this is Docker, and a Command-Line Interface, such as Windows PowerShell. With Docker Desktop, we can see that we can easily manage containers, which include the front-end and back-end applications, as well as the database that the application will interact with.

Slide 4 – Orchestration
Docker Compose is a tool used for defining and running multi-container applications, which simplifies the control of our full-stack application. Docker Compose brings value to projects, especially when considering that they may require using multiple micro-services, that are deployed using multiple containers. This allows them to interact with each other while being isolated (Docker Inc., 2024). This is accomplished by defining these multi-container applications with a single YAML file, which tells the application which containers it needs to interact with. Then, using the "docker-compose up" command from the application directory, the application is re-built using the YAML file to tell the application which containers to use. Docker Compose files are easily sharable, which promotes efficient collaboration between developers, teams, and stakeholders. Also, because Docker Compose reuses containers when services have not changed, this allows for rapid application development. Compose is highly portable and has extensive community and support available (Docker Inc., 2024).

Slide 5 – The Serverless Cloud
Now we are ready to begin migrating our application to the cloud. The cloud is considered to be "serverless" because the server(s) are managed by third-party cloud services, such as Amazon Web Services, who manage all of the infrastructure to ensure their services run smoothly. This includes

handling the physical hardware, data centers, networking equipment, platform security, service maintenance, compliance, and support, and it is one of the key benefits of serverless computing.
By migrating to the cloud, the burden of maintaining a server is shifted away from an organization to the cloud provider allowing us to focus solely on application development and deployment.
Another key advantage of serverless is that scaling is automatic to meet demand, so as your business needs increase you don't have to worry about performance issues or increasing resources which can be costly.

Also, services are billed on a pay-per-use model, meaning you only pay for the resources you actually use, instead of paying monthly rates even when you aren't using services.
Using AWS S3 (Simple Storage Service), we can upload and deploy our application. S3 is a cloud-based storage solution offered by Amazon and is an object storage service where data is stored in buckets and there is no limit on how many objects can be stored in them, however, an object's size is limited to 5 terabytes (Simplilearn, 2017).

S3 allows for virtually unlimited storage capacity at very low costs and data is stored redundantly across multiple facilities ensuring high durability, high availability, data integrity, and in the event of a disaster, data is easily recoverable.

Unlike S3, with local storage you are limited to the capacity of your resources. This can lead to higher costs associated with managing resources, especially as resources reach their limit, and replacing or upgrading resources can be costly, especially the larger an organization is. Also, with local storage you are responsible for backing up data to prevent loss in situations where in one location data may become unrecoverable. Also, you are responsible for maintaining the security of the data and physical devices. Because of the drawbacks of local storage, S3 is a perfect solution to meet our business needs.

Slide 6 – The Serverless Cloud
Next, we are going to use API Gateway and create Lambda functions that will allow us to integrate the front end of our application with the back end.

The API Gateway is essentially the entry point or front door for our application. It uses routes that trigger Lambda function events which then executes code for our services.

The advantages of using a serverless API are that there is no server management, your API scales up or down automatically based on demand, and it is cost-effective because you only pay for what you use, and when the API is not running you pay nothing.

This enables developers to spend more time writing code, instead of on infrastructure issues, leading to faster development cycles and deployment times.

We create Lambda functions, upload the function code for each Lambda, then set up the API gateway and link it to Lambda functions. This allows us to define endpoints that users can call.

Methods and resources are then defined in API Gateway, such as GET, POST, PUT, and DELETE methods, and the URL resources which will be used to trigger Lambda functions.

Once the functions are deployed, incoming requests for each function trigger the code's execution. The Lambda function executes, processes the event, which then generates and returns a response to the API Gateway, where the response is then sent back to the user.

Scripts are produced for each of our Lambda functions. Each Lambda function performs a specific action. For example, we have a DeleteRecord Lambda function that is executed when the DELETE method is called from the API gateway. This Lambda functions code is executed which will delete the passed in record.

Slide 7 – The Serverless Cloud

DynamoDB is a serverless NoSQL database that was developed by Amazon to respond to queries without the computing overhead needed when making the sometimes complex joins of a relational database (Bonisteel, 2023).

It is a NoSQL key-value item store and each item has a primary key and other attributes. It offers simpler querying that focuses on primary keys and it supports JSON-like document structures.
Since the service is managed by Amazon, its underlying architecture and infrastructure are abstracted away from developers. Also, similar to other AWS services, it operates on a pay-per-use model.

MongoDB on the other hand is a flexible, document-oriented model that stores BSON objects that contain one or more key-value pairs in documents and collections (Bonisteel, 2023).
It offers a wide range of data types including ones that are more complex, such as arrays and embedded documents. It supports complex queries and aggregation through its aggregation pipeline, however, this can lead to performance issues as collections scale largely. Also, the syntax can sometimes be challenging which makes writing and building aggregation pipelines more complex.

Both types of databases are well-suited for different use cases. MongoDB is a more flexible option, especially because it can be deployed on a variety of different platforms, whereas DynamoDB is an AWS service. MongoDB also includes a richer set of tools for storing and querying data, however, DynamoDB is excellent when running smaller databases with reduced infrastructure and lower security overhead as in our case (Bonisteel, 2023).

The types of queries we ran on our Dynamo Database include basic queries that allow us to perform CRUD operations.

To create items in our database, we performed POST method executions for both the Questions and Answers tables using the UpsertQuestion and UpsertAnswer Lambda functions.

We also performed read queries using GET method execution, update queries using PUT method execution, and deleted items using the DELETE method execution. Each of our queries are performed based on the Lambda event that is triggered to perform the desired operation.

The scripts that are used to perform these queries are in each of the Lambda's execution code. We see that for the DeleteRecord Lambda function, a query is structured based on the data that is passed in, then the DeleteCommand is executed on the specified DynamoDB table.

The same applies to all of our Lambda functions, performing the various Create, Read, Update and Delete queries on our Database.

Slide 8 – Cloud-Based Development Principles
With cloud-based development, elasticity refers to the ability of the system to automatically adjust its resources based on its demand. It is a critical aspect of cloud-based development because it is a key feature of deploying your application in the cloud.

When deploying in the cloud we take advantage of the benefits of elasticity by having the ability to automatically scale up when demand is high and down when demand is low. This ensures cost efficiency and the best performance.

This also means that the system is flexible and can quickly adapt to workloads and specific usage patterns automatically, thereby maintaining performance and availability by managing resources efficiently.

The pay-for-use model enables businesses to deploy applications at low costs, ensuring that you are only billed for what you use. Depending on the services being used, you pay for compute time or storage capacity, which is great in cases where your application may be using services that aren't always running.

Additionally, there are no up-front fees and no contracts meaning you have the ability to start and stop services as needed without being billed when services aren't running.
Automatic scaling helps ensure that you always have the necessary amount of resources without overspending on what you don't need.

Slide 9 – Securing Your Cloud App – Access
To secure our cloud application, we can prevent unauthorized access by utilizing AWS's IAM (Identity and Access Management) service to securely control access to AWS resources and set permissions that control these resources and who can access them (Amazon Web Services Inc., 2024).

To accomplish this, we first establish roles, which should be unique to each service, for the various types of users that will need to have access to our resources.

After policies have been defined and attached to roles, these roles can then be attached to our various services.

This ensures that only the users we want accessing our services are given permission and that they are specific to our services.

Slide 10 – Securing Your Cloud App – Policies
For each role, we want to ensure that they are given specific permissions to carry out the tasks that they require.

We can do this by defining specific policies that designate what each role should be allowed to access or perform actions on.

It is also important to follow the principle of least privilege and only grant just enough permission to accomplish the tasks each role requires.

Once these policies are defined, they are attached to the roles, which are then attached to each specific service that the role should be allowed to assume.

Essentially, roles are the who and policies say what they can do when the roles are attached to services.

Here, we created a custom role called LabRole, which has specific policies that determine what the role is allowed to do. This role is then attached to each of our Lambda functions, which allows the role to interact with each of our services

Slide 11 – Securing Your Cloud App – API Security

To secure the API Gateway, we have created roles and applied IAM policies to these roles which determine who can access any one of our services and what actions they can perform.

These roles can be attached to any of our services including but not limited to our Lambda functions, DynamoDB, and S3 bucket.

This applies fine-grained permissions to our AWS services and resources, including who can or cannot access our API Gateway.

Further security measures can be implemented, such as the use of AWS WAF (Web Application Firewall) which helps protect the API Gateway from common web exploits and unusual traffic patterns, enhancing the overall security of the API gateway.

Slide 12 – Conclusion

To bring it all together, we have started by building our application locally and containerizing it to prepare it for deployment in the cloud. Deploying our application in the cloud means that we are able to abstract away the burden of maintaining an infrastructure by shifting to a third-party service provider which allows us to scale automatically based on demand, maintain high performance and availability, and improve cost efficiency by only paying for the services we use.

To deploy our application, we have uploaded it into an S3 bucket where we can host our application, created Lambda functions to interact with our backend DynamoDB tables and Established an API Gateway to act as the front door for our application and trigger our Lambda functions when called.

This enables the front end of our application to interact with the backend, which based on user actions, performs queries on our database to store and retrieve data which is then sent back to the client.

Lastly, we secured our application, including the Lambda functions, API Gateway, and our DynamoDB by creating an IAM role, attaching policies that specify what the Role is allowed to do while keeping the principle of least privilege in mind, and then attaching these roles to our services and functions.

Together we have carefully pieced together a full-stack application and deployed it to the cloud, all while maintaining security, ensuring seamless service to our clients that is elastic and scales on demand, and ensuring resources are maximized and that our application is cost-efficient.

Thank you for your time.

Slide 13 – Image Sources

Slide 14 - References