Jonathan Miller

Prof. M. Shah Alam

CS-470

February 26, 2024

Project Two Conference Presentation: Cloud Development

Video Link

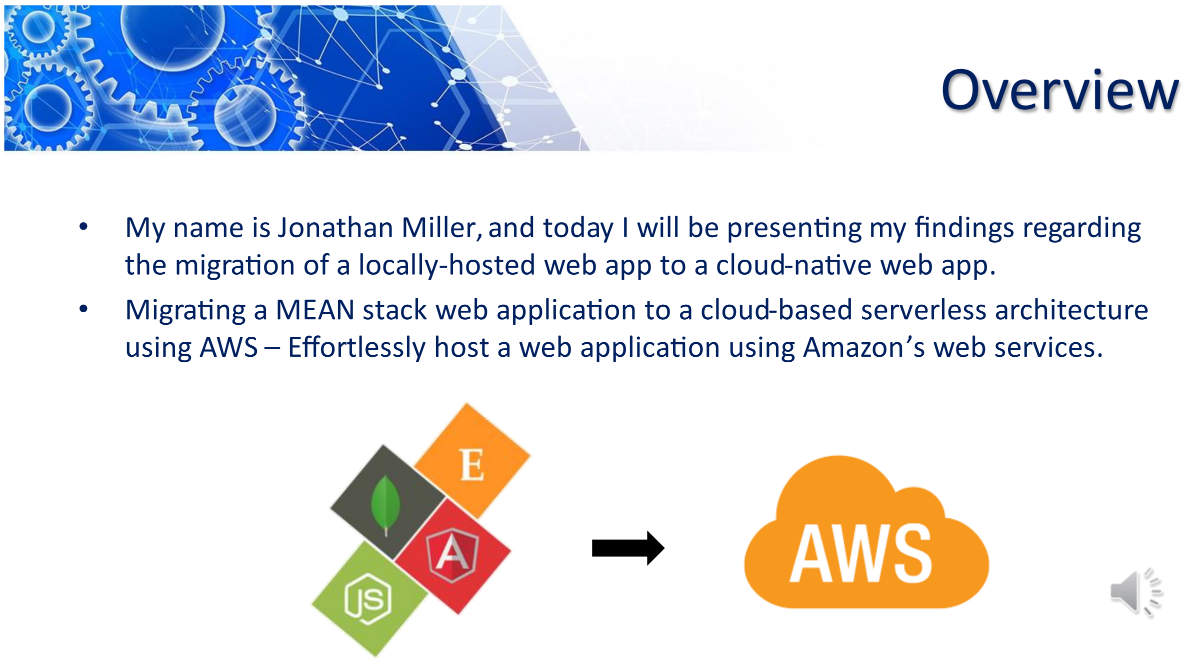
https://youtu.be/MJQWoRc2314

Script

CS 470 Project Two Conference Presentation: Cloud Development

Jonathan Miller

02/2024



Overview

- My name is Jonathan Miller, and today I will be presenting my findings regarding the migration of a locally-hosted web app to a cloud-native web app.
- Migrating a MEAN stack web application to a cloud-based serverless architecture using AWS – Effortlessly host a web application using Amazon’s web services.

Hello,

My name is Jonathan Miller. I recently went through the process of converting a containerized MEAN stack application to a cloud-native web application utilizing a serverless architecture. This was accomplished by utilizing Amazon Web Services, and today I will discuss this migration process along with some information regarding the services that Amazon provides to natively host a full-stack web application.

Containerization

- Local containerization of the MEAN stack with Docker.
- Cloud-based containerization using S3, API Gateway, Lambda, and DynamoDB.

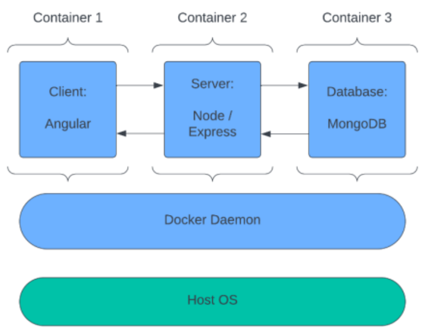


Figure 1: MEAN stack Docker containerization

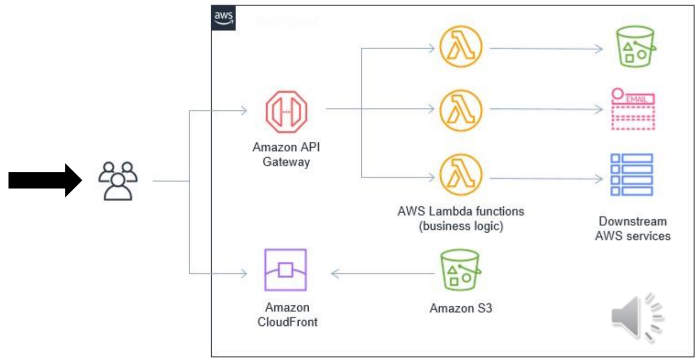


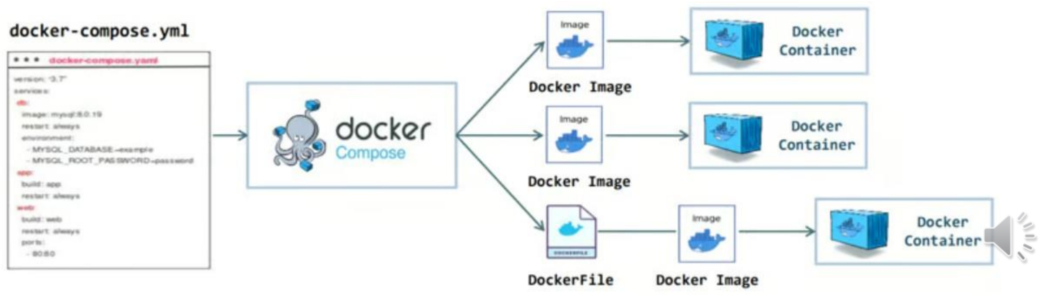
Figure 2: Beswick, J. (2020). AWS. https://aws.amazon.com/blogs/compute/replacing-web-server-functionality-with-serverless-services/

To begin, our locally-hosted MEAN stack web application was containerized with Docker. The front-end, server, and database each had its own container, and this allowed each service to be packaged into a lightweight, standalone, executable image containing the code, dependencies, and settings needed to run the service.

Migrating this containerized web application to Amazon Web Services allows us to effectively containerize our web application in serverless architecture with functionality extending past that provided by Docker. We utilized Amazon S3 to host our Angular front-end, Amazon API Gateway and Lambda to host our back-end API logic, and DynamoDB to host our database. Each of these microservices can securely communicate with each other to form a serverless full-stack web app hosted entirely by Amazon.

Orchestration

- Docker Compose container orchestration for locally-hosted containerized web apps.
- Docker Compose saves time, prevents issues, and enables easy deployment.



```
version: "3.7"
services:
 db:
  image: mysql:8.0.19
  restart: always
  environment:
   - MYSQL_DATABASE=example
   - MYSQL_ROOT_PASSWORD=password
 app:
  build: app
  restart: always
 web:
  build: web
  restart: always
  ports:
   - 80:80
```

Figure 3: Waji (2023). Docker-Compose Management. https://dev.to/waji97/docker-compose-management-1d84

If we wanted to deploy this containerized web application on our own infrastructure, we could use Docker Compose to provide container orchestration. This tool provides the means to effortlessly set up the entire project, and it enables communication between the containers in the project. Docker Compose also allows for scaling and load balancing to distribute traffic across the containers. Docker Compose allows developers to jump straight into development by circumventing tedious project setup, and it enables an efficient devops pipeline through a simple build process.

Although Docker Compose provides many features for deployment of our web application, it doesn’t circumvent the fact the we need to develop and maintain an infrastructure for our web app to be hosted on. This infrastructure can be very costly to develop and maintain if we plan to have our web app scale up as demand increases.

The Serverless Cloud

Serverless Advantages

- Build and run web apps without managing infrastructure.
- Server management done by a third party.
- Developers focus on the product instead of infrastructure.
- Faster time to deployment…saves money.

Amazon S3 Advantages

- Serverless storage.
- Automatic redundancy.
- Accessible.
- Scalable.
- Secure.

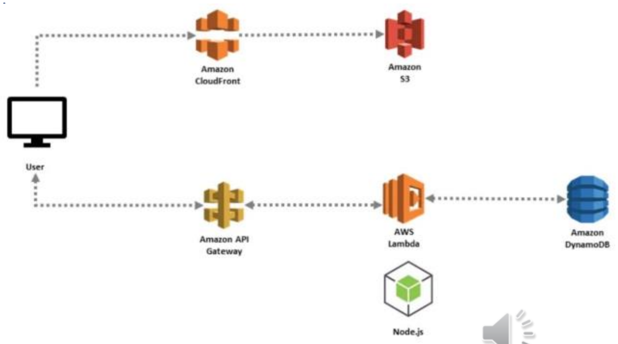


Figure 4: Patel, A. (2020). AWS Serverless. https://medium.com/awesomecloud/aws-serverless-services-and-serverless-computing-in-aws-a1298ace0e3c

Amazon Web Services allows our web app to be hosted entirely on Amazon's servers. This allows the developers to not concern themselves with the infrastructure to host it on, and it allows them to solely focus on the development of the product. The serverless architecture also enables developers to simultaneously work on the microservices, leading to a faster time to deployment. This lack of infrastructure management and faster time to deployment will ultimately lead to cost savings while also providing automation for scalability.

The front-end of our serverless web app was migrated to Amazon's S3 storage solution. Amazon S3 is a cloud storage solution that enables the storage of objects in buckets. The objects are files and metadata, and the buckets are containers for the objects. When creating a bucket, you can choose the AWS datacenter region that it will reside in, and then you can set permissions, static website hosting, events, versioning, cross-region replication, and so on. S3 provides automatic redundancy across regions, easy access from all over the world, scalability through virtually limitless storage, and security through encryption and access control mechanisms.

The Serverless Cloud

API & Lambda

- Provides CRUD functionality for database interaction.
- Front-end uses API Gateway URL endpoint & Lambda to interact with DynamoDB databases.
- HTTP requests resolved through API Gateway.
- Lambda functions contain business logic to process API calls and interact with the database.
- Scalable.
- Cost-savings.

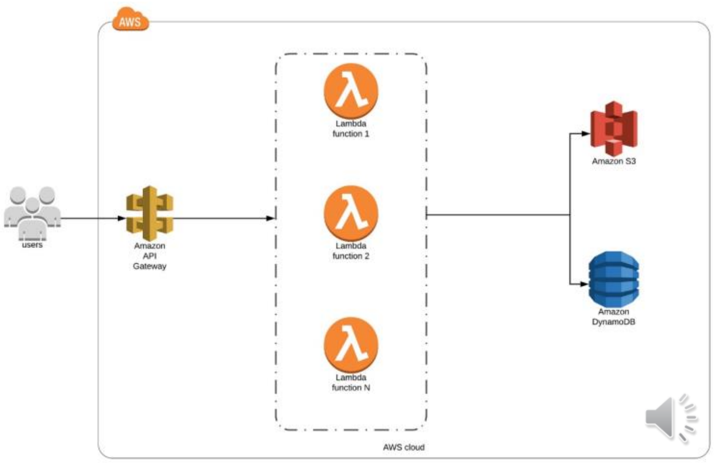


Figure 5: Phuyal, A. (2019). Attendance System AWS Lambda. https://www.genesesolution.com/finland/blog/2019/05/03/genese-attendance-system-overview/

The back-end for our serverless web app was migrated to Amazon's API Gateway and Lambda. This allowed us to implement a RESTful API with create, read, update, and delete functionality to manipulate our databases. The API URL is defined in our Angular front-end, and Angular directs all HTTP requests to this API endpoint. API Gateway then directs the HTTP requests to the corresponding Lambda function, and this Lambda function processes the request to retrieve the requested information from the DynamoDB database. Then the retrieved database item is passed back through the API to the S3-hosted Angular front-end.

Since Amazon API Gateway and AWS Lambda are fully-managed services, developers can easily create, publish, maintain, monitor, and secure APIs at any scale. These services allow for the processing of thousands of concurrent API calls while handling traffic management, authorization and access control, versioning, and monitoring all while automatically scaling as needed. This leads to cost-savings due to the lack of resources needed for architecture management purposes.

The Serverless Cloud

Database

- MongoDB and DynamoDB are both NoSQL database management engines.
- MongoDB is document-oriented with optional schema.
- MongoDB is cross-platform.
- DynamoDB utilizes a key-value model with flexible schema.
- DynamoDB is a single-table design database.
- DynamoDB is only available through AWS.
- DynamoDB enables fast, consistent access.
- DynamoDB enabled queries for Questions and Answers in our serverless web app.

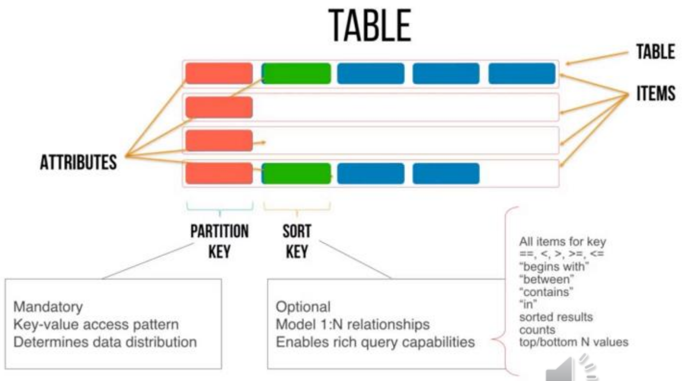


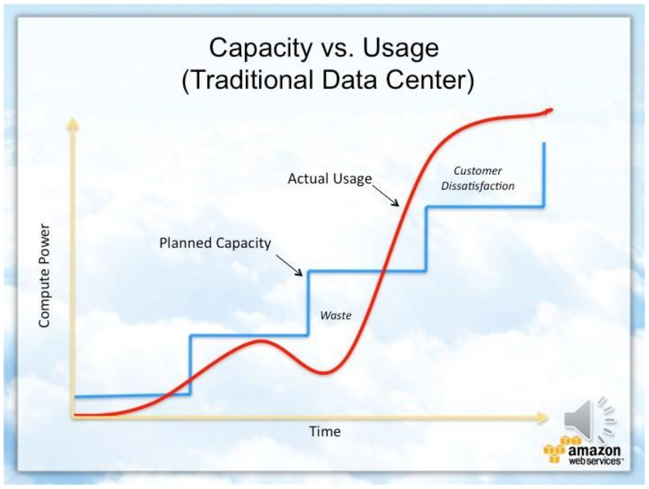
Figure 6: Semyon (2020). Single Table Design. https://pisarev.us/blog/2020-05-29-single-table-design

The databases of our serverless web app was migrated to DynamoDB from MongoDB. Both of these NoSQL database management engines offer similar functionality, but they differ in many aspects. DynamoDB stores data in a key-value model utilizing a single-table design. It is great for consistent, low-latency access at any scale. As a result of this single-table design, the query language is not as rich as MongoDB, but it is a fast, secure, self-managed database service that perfectly integrates with other AWS microservices.

MongoDB, on the other hand, utilizes a document-oriented data model to store and manage structured, semi-structured, and unstructured data with optional schema. MongoDB databases are organized into collections of JSON-like documents that contain fields. Unlike DynamoDB, MongoDB is a general-purpose database solution that is cross-platform, open-source, and supports many programming languages.

Cloud-Based Development Principles

- Elasticity and scalability.
- Resiliency.
- Pay-for-use model.
- Managed services.
- Security.



Cloud-based development follows many principles that help companies overcome the pitfalls of the traditional data center infrastructure. For instance, it is time consuming and costly to meet the demand of the customer in a traditional data center. However, with cloud-based development, scaling with customer demand is much easier and less costly.

Cloud-based architectures allow for elasticity to meet the sudden ups and downs in the workload during a small period of time. They are also scalable to meet static increases in workload. Cloud-native architecture is also resilient with redundancy leading to virtually no application downtime. They also utilize pay-for-use models and fully-managed services that lead to significant cost savings over the traditional data center architecture. And lastly, cloud-based development is secure, which leads us into our next topic.

Securing Your Cloud App

Access

- Prevent unauthorized access with permissions for users and roles with AWS IAM.
- Utilize the principle of least privilege.

Policies

- Policies can be attached to IAM Entities.
- These policies define permissions for IAM Entities through JSON documents.
- To secure our databases, we used IAM policies to restrict access for Lambda functions.
- Read and write permissions were set for each Lambda function.

API Security

- The connection between API Gateway and Lambda is secured by HTTPS and TSL.
- The connection between Lambda and DynamoDB is secured through IAM policies.
- S3 Buckets are secured with IAM policies, access control lists, automatic encryption, and audit logs.

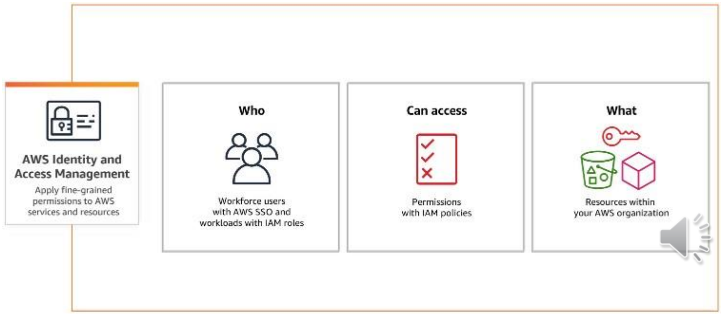
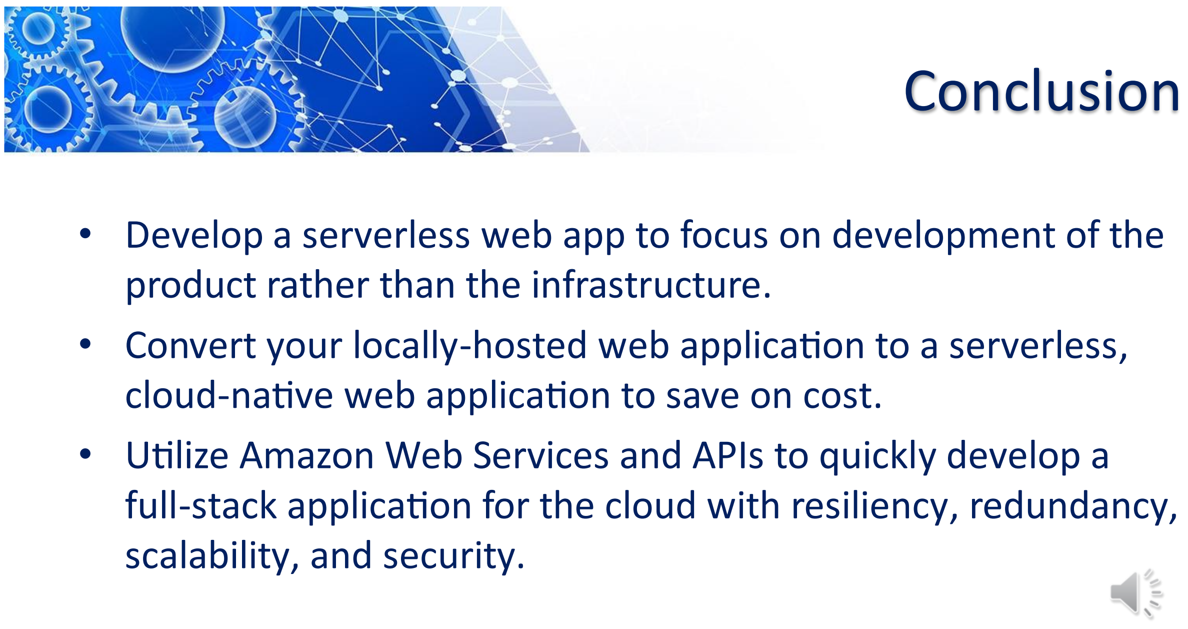

AWS Identity and Access Management
Apply fine-grained permissions to AWS services and resources
Who
Workforce users with AWS SSO and workloads with IAM roles
Can access
Permissions with IAM policies
What
Resources within your AWS organization

Figure 7: Amazon. (2024). AWS Identity and Access Management. https://aws.amazon.com/iam/getting-started/

The migration to Amazon Web Services allows us to take advantage of the many security mechanisms on offer. Access control can be implemented with permissions for users, roles, resources, and organizational units. These permissions are delegated such that entities are only granted the permissions that are needed to perform a task. These permissions are attached to entities through AWS Identity and Access Management policies.

For our serverless web app, IAM policies were defined for various entities through JSON documents. These IAM roles and policies define what our Lambda functions have access to. In order for our Lambda functions to access our databases, resource-based policies needed to be created for each Lambda function to define the read and write permissions. These policies allowed our Lambda functions to get, query, scan, put, update, and delete items in our databases.

The connection between API Gateway and Lambda is secured by HTTPS and TSL, leading to additional security in our back-end. The S3 bucket that hosts the front-end is secured with automatic encryption.



Conclusion

- Develop a serverless web app to focus on development of the product rather than the infrastructure.
- Convert your locally-hosted web application to a serverless, cloud-native web application to save on cost.
- Utilize Amazon Web Services and APIs to quickly develop a full-stack application for the cloud with resiliency, redundancy, scalability, and security.

So, in conclusion, the migration from a locally-hosted web app to a cloud-native web app allowed development to focus on the product rather than the underlying infrastructure and architecture management. This leads to significant cost savings in the short and long term. The serverless architecture also allowed our full-stack web app to be fully integrated into the Amazon Web Services platform with increased resiliency, redundancy, scalability, and security.