CS 470 Project Two Conference Presentation: Cloud Development

Laura Pittman

Southern New Hampshire University

April 20, 2025

https://youtu.be/AWDbQOZ6-iI

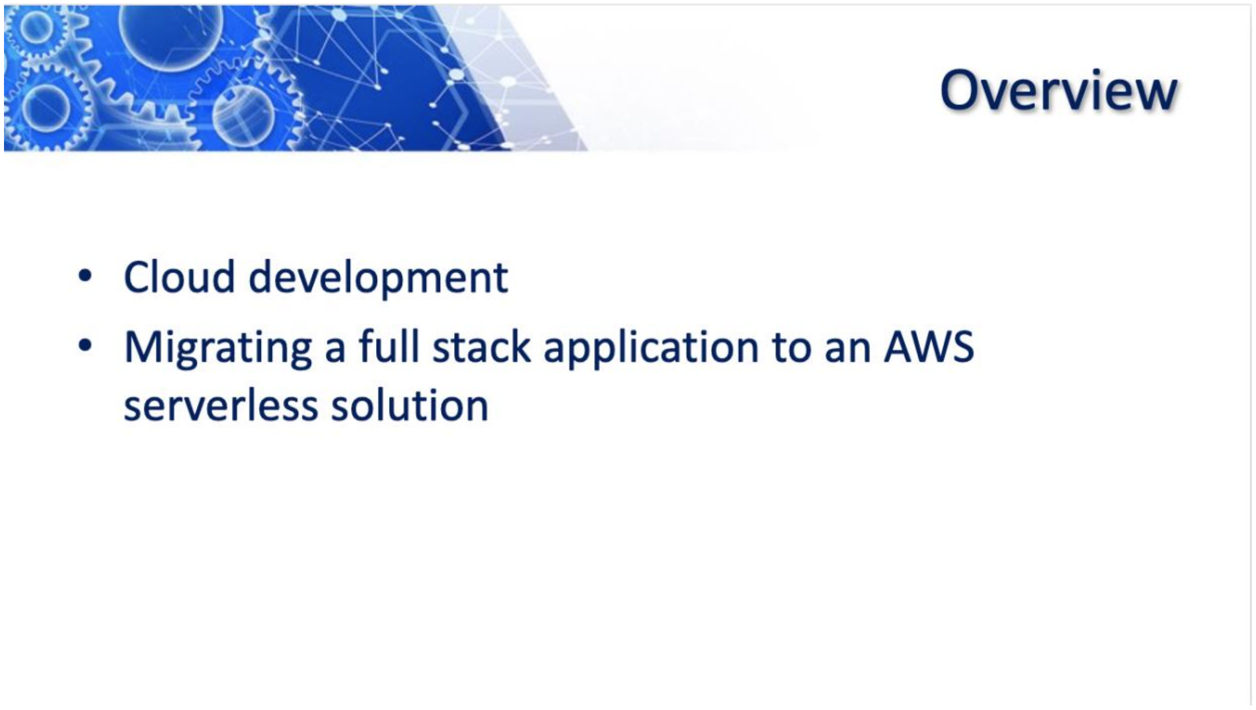
CS 470 Project Two
Conference Presentation:
Cloud Development

Laura Pittman
April 2025

CS 470 Project Two
Conference Presentation:
Cloud Development

Laura Pittman
April 2025

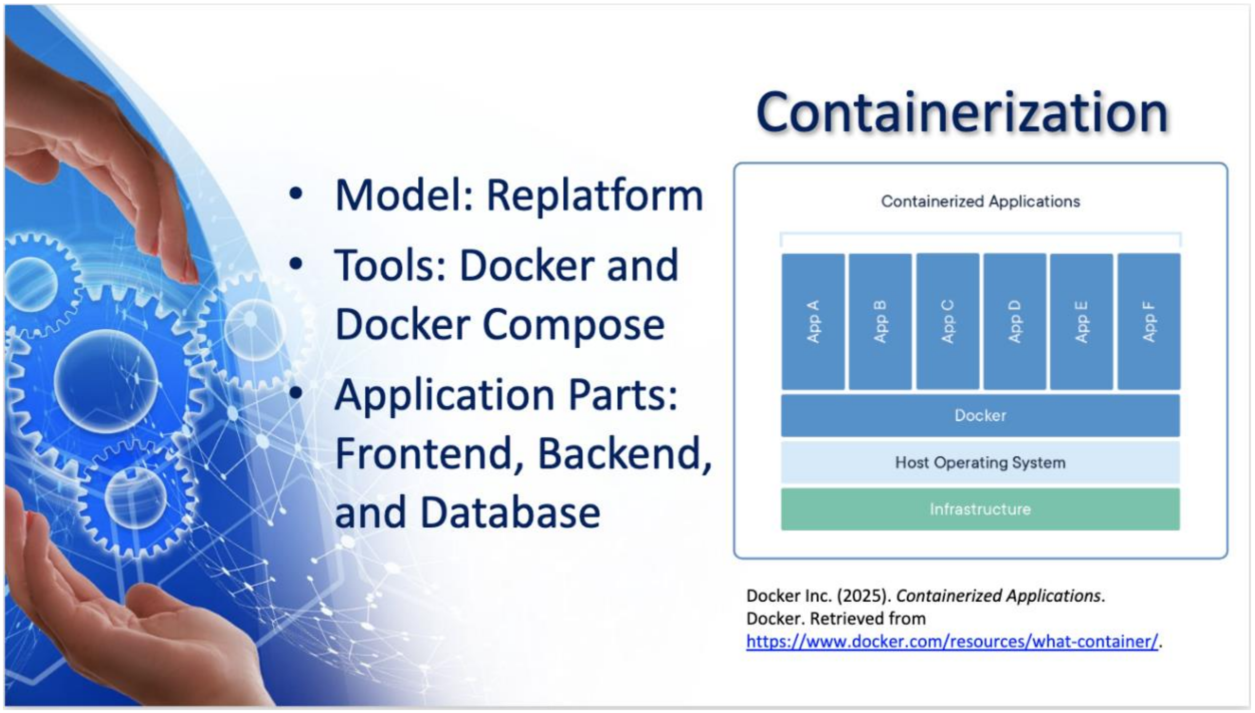
Hello! My name is Laura Pittman, and I am a senior in my last term at Southern New Hampshire University working on my bachelor's degree in Computer Science.



Overview

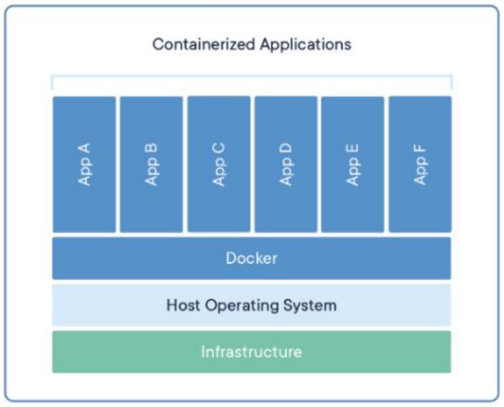
- Cloud development
- Migrating a full stack application to an AWS serverless solution

This presentation will assess and discuss the intricacies of cloud development, specifically the process of migrating from a full stack application to a cloud-native web application using AWS microservices, in a way that is easily understood by both technical and non-technical audiences.



Containerization

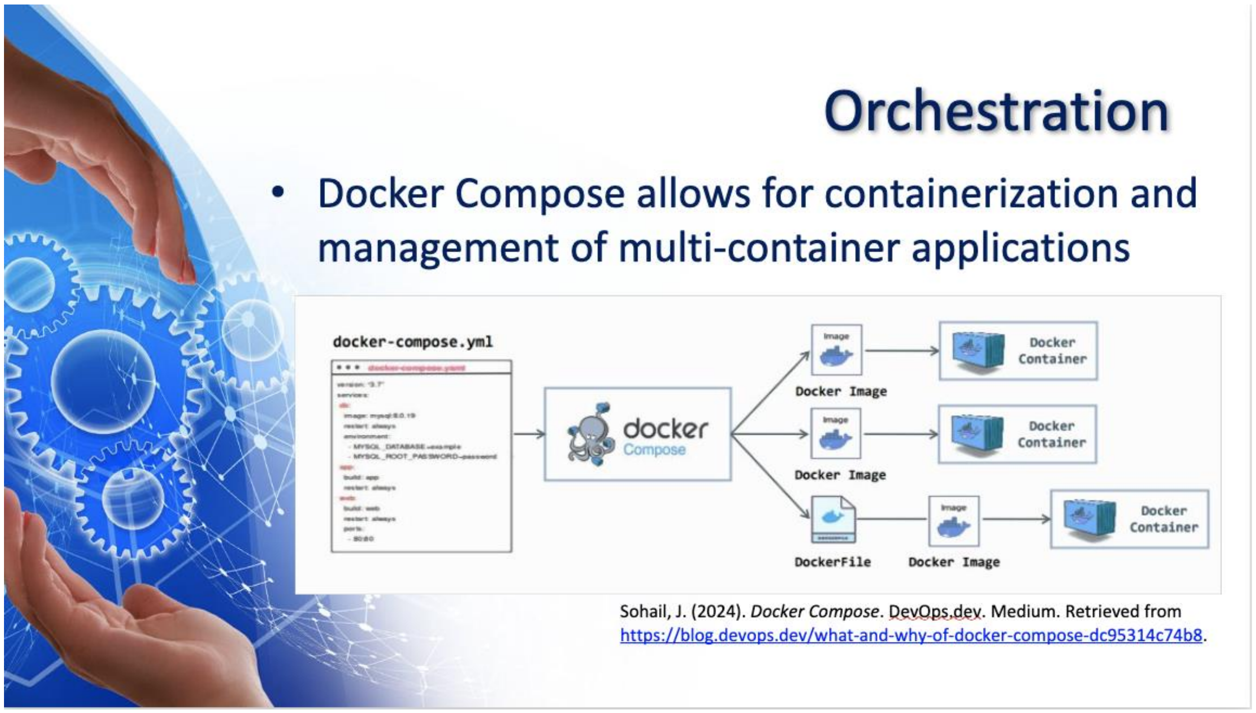
- Model: Replatform
- Tools: Docker and Docker Compose
- Application Parts: Frontend, Backend, and Database



Docker Inc. (2025). *Containerized Applications*. Docker. Retrieved from https://www.docker.com/resources/what-container/.

The model used to migrate the full stack application to the cloud was replatforming which included making very minimal changes to optimize it for the cloud platform and updating the application's database and other dependencies to cloud-native versions. Docker and Docker

Compose are necessary tools for this task as they allow for containerization. This is the process that allows the original application to be broken down into smaller parts like the frontend, backend, and database and packaging each part into a separate container that can later be deployed to the cloud.



Orchestration

- Docker Compose allows for containerization and management of multi-container applications

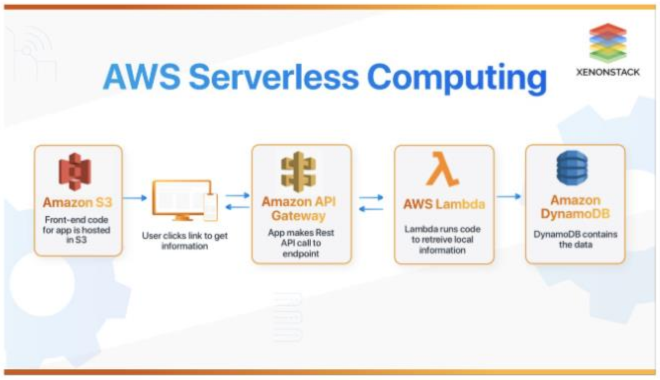
docker-compose.yml

```
docker-compose.yaml
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
```

Sohail, J. (2024). *Docker Compose*. DevOps.dev. Medium. Retrieved from https://blog.devops.dev/what-and-why-of-docker-compose-dc95314c74b8.

The value of using Docker Compose is unmatched as it necessary to make the management of multi-container Docker applications even easier. It allows you to define and run multiple services in a single configuration file, also known as docker-compose.yml. This process makes it much easier to manage dependencies, environments, and scaling if needed.

The Serverless Cloud

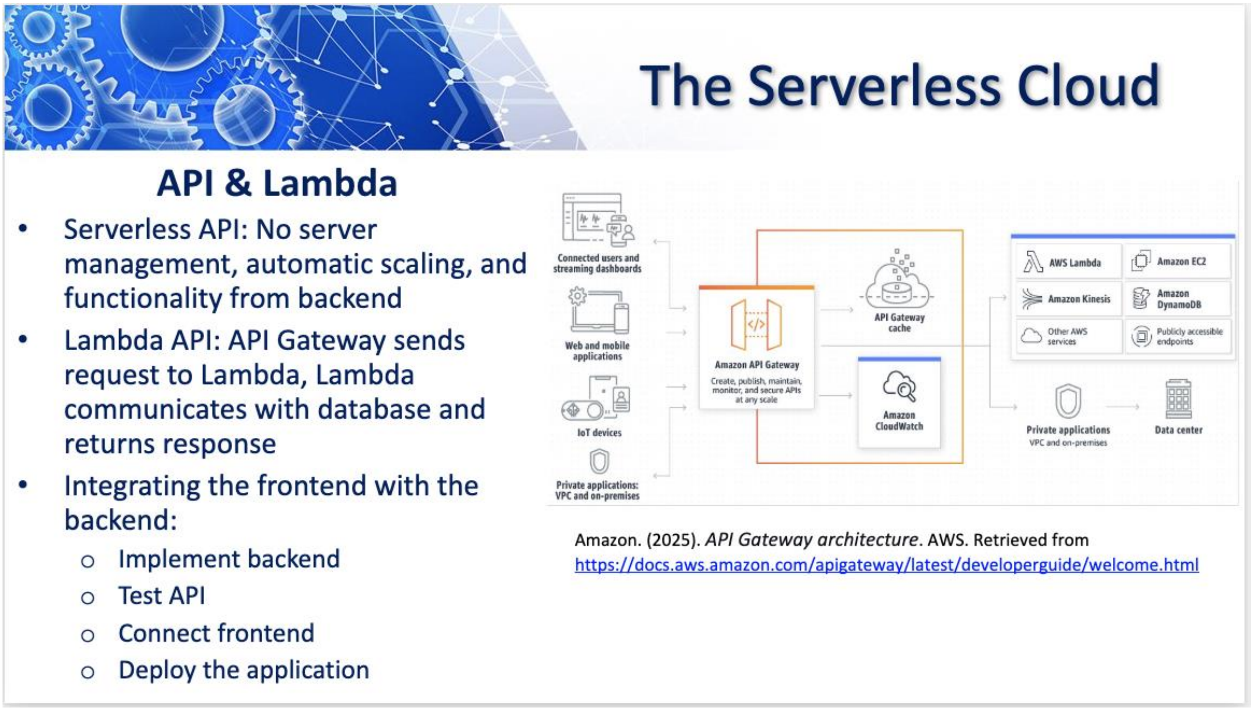

AWS Serverless Computing
XENONSTACK
Amazon S3
Front-end code for app is hosted in S3
User clicks link to get information
Amazon API Gateway
App makes Rest API call to endpoint
AWS Lambda
Lambda runs code to retreive local information
Amazon DynamoDB
DynamoDB contains the data

Gill, J. K. (2025). *AWS Serverless Computing*. XenonStack. Retrieved from https://www.xenonstack.com/blog/aws-serverless-computing/.

Serverless

- Cloud computing model
- No server management
- Authentication with JSON Web Tokens (JWT)
- **S3 storage** is a cloud-based object storage

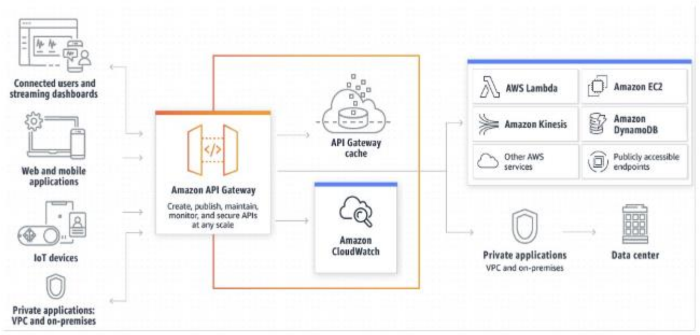
Serverless computing is a cloud computing model that eliminates the need for users to manage and provision servers as they are monitored by another entity, for this particular instance that is AWS. This type of application has various advantages such as no need to maintain or provision servers, authentication through JSON Web Tokens which are a compact and self-contained way for securely transmitting information by verifying the client's credentials and what actions they are allowed to perform. Furthermore, S3 storage is a cloud-based object storage built to retrieve any amount of data from anywhere that offers scalability, data availability, security, and performance, and it is rather different from local storage as it is not limited by the physical device's capacity, but rather the server capacity.



The Serverless Cloud

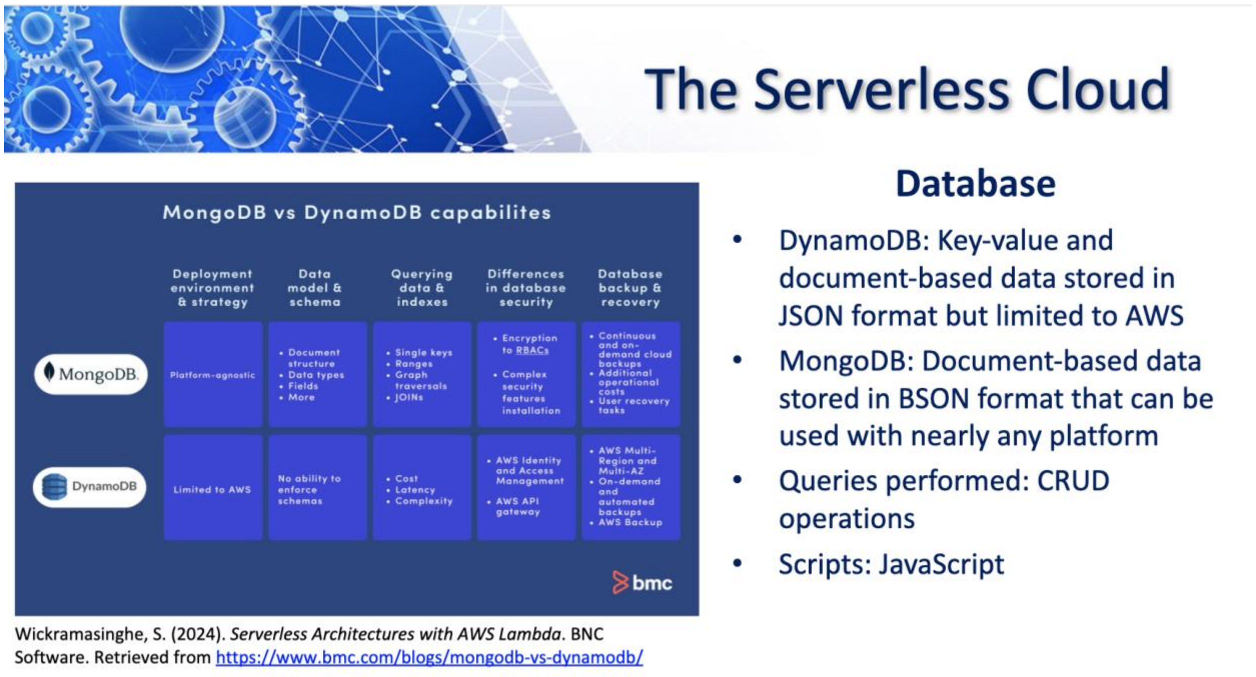
API & Lambda

- Serverless API: No server management, automatic scaling, and functionality from backend
- Lambda API: API Gateway sends request to Lambda, Lambda communicates with database and returns response
- Integrating the frontend with the backend:
  - Implement backend
  - Test API
  - Connect frontend
  - Deploy the application


Connected users and streaming dashboards
Web and mobile applications
IoT devices
Private applications: VPC and on-premises
Amazon API Gateway
Create, publish, maintain, monitor, and secure APIs at any scale
API Gateway cache
Amazon CloudWatch
AWS Lambda
Amazon EC2
Amazon Kinesis
Amazon DynamoDB
Other AWS services
Publicly accessible endpoints
Private applications
VPC and on-premises
Data center

Amazon. (2025). *API Gateway architecture*. AWS. Retrieved from https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

The advantages of using a serverless API are no server management, automatic scaling based on demand, and improved functionality from backend services, such as code running on AWS Lambda. The Lambda API Logic starts with the API Gateway which routes the request to a Lambda function that runs the appropriate logic, communicates with the database, and returns a response. Integrating the frontend with the backend of the application involves using API Gateway endpoints to interact with the Lambda functions, making the backend accessible through HTTP requests. It starts with implementing the backend, then testing the API, and connecting the frontend where you can then deploy the application.



The Serverless Cloud

MongoDB vs DynamoDB capabilites

	Deployment environment & strategy	Data model & schema	Querying data & indexes	Differences in database security	Database backup & recovery
MongoDB	Platform-agnostic	• Document structure • Data types • Fields • More	• Single keys • Ranges • Graph traversals • JOINs	• Encryption to RBACs • Complex security features installation	• Continuous and on-demand cloud backups • Additional operational costs • User recovery tasks
DynamoDB	Limited to AWS	No ability to enforce schemas	• Cost • Latency • Complexity	• AWS Identity and Access Management • AWS API gateway	• AWS Multi-Region and Multi-AZ • On-demand and automated backups • AWS Backup

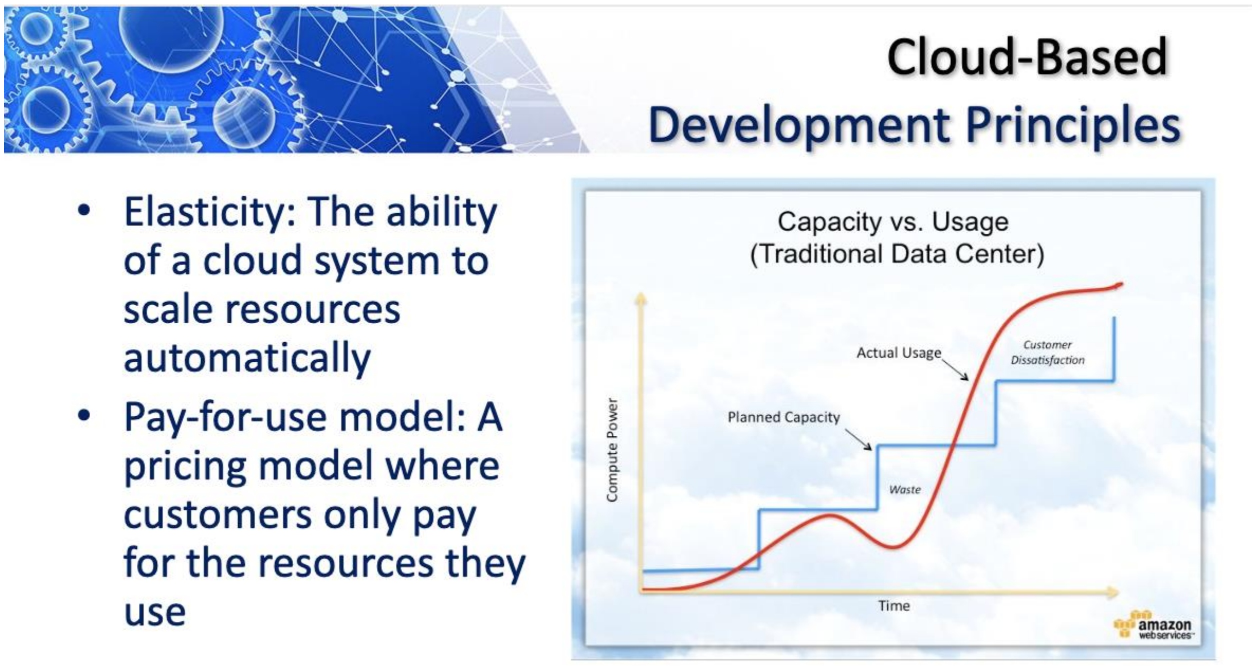
bmc

Wickramasinghe, S. (2024). *Serverless Architectures with AWS Lambda*. BNC Software. Retrieved from https://www.bmc.com/blogs/mongodb-vs-dynamodb/

Database

- DynamoDB: Key-value and document-based data stored in JSON format but limited to AWS
- MongoDB: Document-based data stored in BSON format that can be used with nearly any platform
- Queries performed: CRUD operations
- Scripts: JavaScript

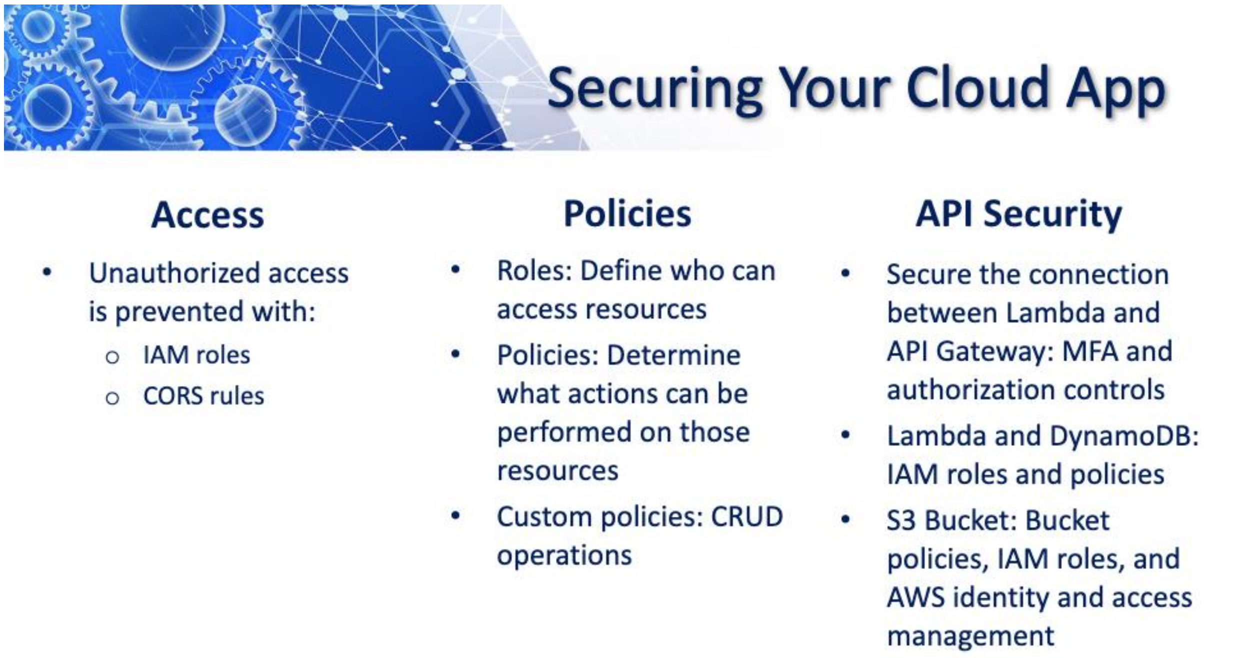
DynamoDB is a serverless, NoSQL, fully managed database by Amazon that supports key-value and document data while storing it in JSON format with single-digit millisecond performance at any scale. Meanwhile, MongoDB is a general purpose, document-based distributed NoSQL database that stores data primarily in BSON format with some JSON formatted data. This means that the data in MongoDB is stored in binary whereas DynamoDB stores data in a text file format. DynamoDB was used to perform the CRUD operations, or Create, Read, Update, and Delete which easily allows for access and modification to the data being stored through JavaScript.


Capacity vs. Usage
(Traditional Data Center)
Compute Power
Time
Actual Usage
Planned Capacity
Customer Dissatisfaction
Waste
amazon web services

Cloud-Based Development Principles

- Elasticity: The ability of a cloud system to scale resources automatically
- Pay-for-use model: A pricing model where customers only pay for the resources they use

Cloud-based development principles include elasticity and pay-for-use model. Elasticity refers to the ability of a cloud system to scale resources automatically in response to demand changes. A pay-for-use model refers to a pricing model in which customers only pay for the resources they use. These principles work together to allow businesses to optimize both performance and cost.



Securing Your Cloud App

Access

- Unauthorized access is prevented with:
  - IAM roles
  - CORS rules

Policies

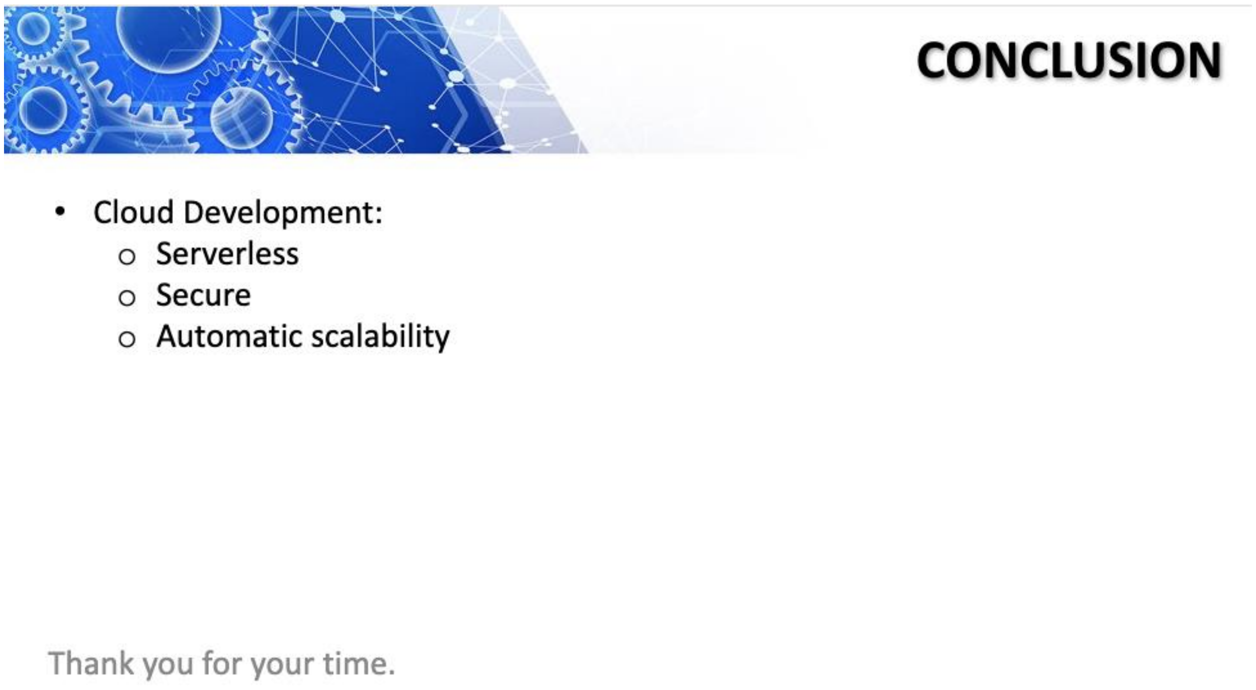
- Roles: Define who can access resources
- Policies: Determine what actions can be performed on those resources
- Custom policies: CRUD operations

API Security

- Secure the connection between Lambda and API Gateway: MFA and authorization controls
- Lambda and DynamoDB: IAM roles and policies
- S3 Bucket: Bucket policies, IAM roles, and AWS identity and access management

To prevent unauthorized access, you must implement CORS (Cross-Origin-Resource-Sharing) rules and IAM roles to define identities, for example Lambda functions and users, that can access

AWS resources. Roles define who (or what) can access resources whereas policies determine what actions those identities can perform on those resources. The custom policies created for this project include CRUD operations which are create, read, update, and delete. Securing the connection between Lambda and Gateway includes MFA (Mutli-Factor Authentication) and authorization controls. Securing the connection between Lambda and DynamoDB includes IAM roles and policies. Securing the S3 bucket includes employing bucket policies that ensure no unauthorized access can occur, implementing IAM roles to manage temporary credentials for applications/services that need to access S3, and ensuring AWS identity and access management.



CONCLUSION

- Cloud Development:
    - Serverless
    - Secure
    - Automatic scalability

Thank you for your time.

Overall, it is clear to see that cloud development is superior to other migration methods due to it being serverless which frees up time for developers to focus on code, secure with easy implementation of various levels of authentication and authorization controls through IAM roles and CORS rules, and automatically scales in response to the demand changing through the pay-for-use model which ensures that cloud development is cost-effective as well as efficient.