PROJECT TWO CONFERENCE PRESENTATION:

CLOUD DEVELOPMENT

SOUTHERN NEW HAMPSHIRE UNIVERSITY

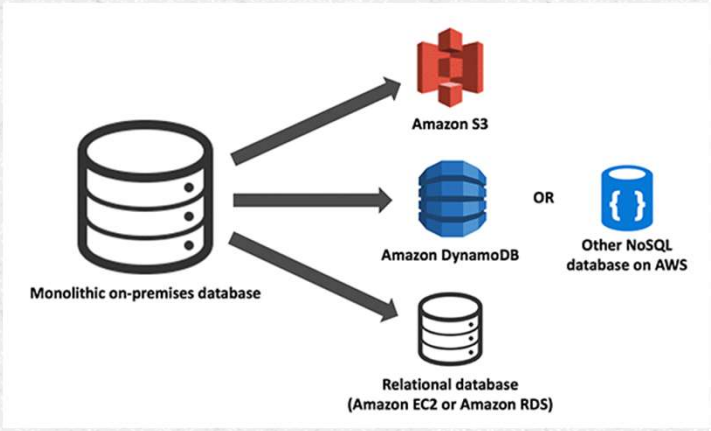
CS-470 FULL STACK DEVELOPMENT II

17TH AUGUST 2025

MIGRATING A FULL STACK APP TO AWS SERVERLESS

Presented by: Kursheeka Milburn

Hello everyone, my name is Kursheeka Milburn. I am currently completing my final term at Southern New Hampshire University toward earning my degree in Computer Science. Today I will share my experience migrating a traditional full stack application into a cloud-native serverless architecture on AWS. This presentation is for both technical and nontechnical audiences, and I will cover the tools, processes, and lessons learned along the way.



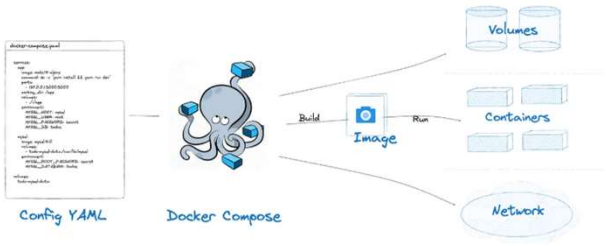
PURPOSE & OVERVIEW

- How a full stack application was containerized
- How it was migrated to AWS serverless
- Benefits, challenges, and results

The purpose of this presentation is to explain the process of moving a full stack application from a traditional hosting environment to AWS serverless. We will review containerization, the migration process, and the results. I will also discuss the benefits and challenges we encountered and how AWS services made the transition easier.

CONTAINERIZATION

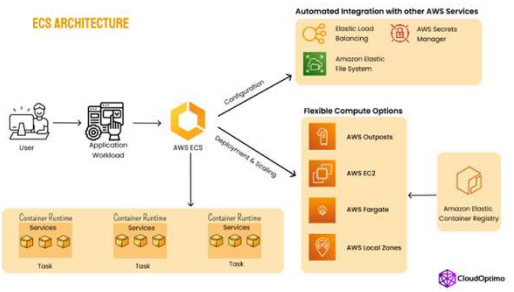
- Docker for container creation
- Docker Compose for orchestration
- MEAN stack: MongoDB, ExpressJS, Angular, NodeJS



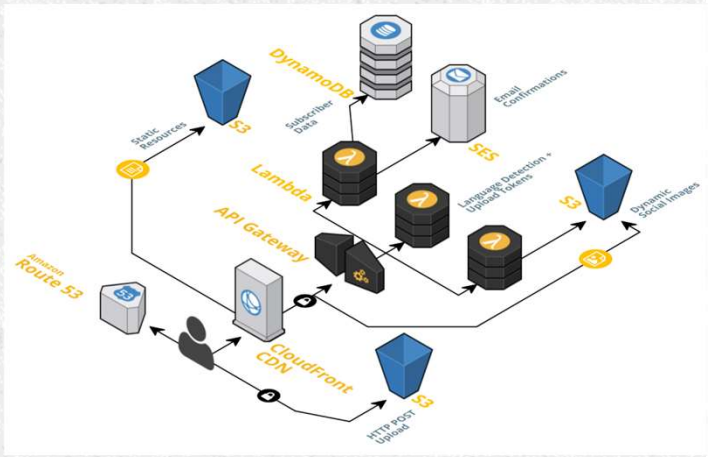
We began with containerization to ensure consistency across development, testing, and production. Docker was used to build container images for the frontend, API, and database. Docker Compose managed multiple containers, allowing them to work together seamlessly. The MEAN stack formed the foundation of the application.

ORCHESTRATION

- Manage multi-container apps
- Define how services connect and scale
- Easy to adjust for demand



Orchestration ensures services work together efficiently. With Docker Compose, we could define deployment rules, service connections, and scaling instructions. This made it simple to adjust resources for increased traffic or scale down during quieter periods.



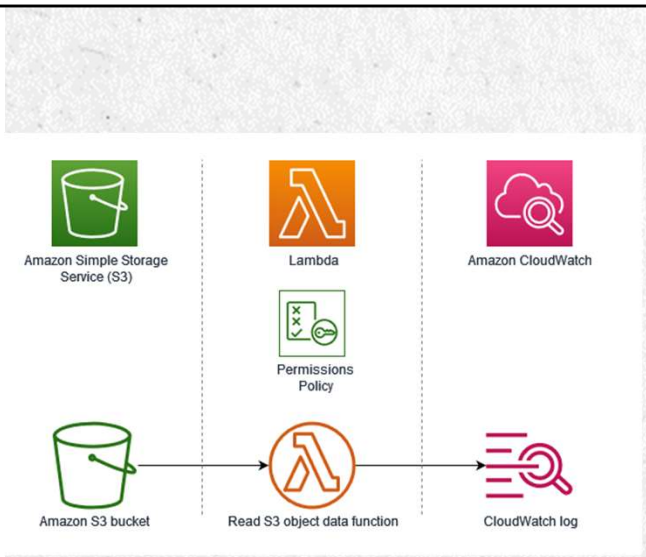
WHAT IS SERVERLESS?

- Runs on managed servers
- No hardware maintenance
- Automatic scalingPay only for usage

Serverless computing means the cloud provider handles all server maintenance, scaling, and patching. AWS services like Lambda and API Gateway allow us to focus solely on application code. We pay only for the resources used, making it cost-effective and highly scalable.

S3 STORAGE VS LOCAL STORAGE

- Local Storage:
  - Fixed size, hardware-dependent
  - Manual expansion
- Amazon S3:
  - Unlimited capacity
  - Redundant storage across regions
  - Intelligent-Tiering for cost savings



Local storage is limited by physical disk space and requires manual upgrades. Amazon S3 offers virtually unlimited storage, high redundancy, and cost optimization through Intelligent-Tiering. It integrates easily with other AWS services, making it ideal for storing static assets and application content.

API GATEWAY & LAMBDA

- API Gateway: Manages endpoints
- Lambda: Runs backend logic
- Supports GET, POST, PUT, DELETE
- CORS enabled for frontend access

API Gateway defines API endpoints and routes requests to Lambda functions that run our backend logic. We set up endpoints for CRUD operations and enabled CORS to allow communication between the frontend and backend. Access policies ensure only authorized requests are processed.

Feature	MongoDB	DynamoDB
Data Model	Document-based (JSON-like structure)	Key-value store
Hosting	Can run locally or in cloud (e.g., Atlas)	AWS-managed service only
Scalability	Sharding for scaling, manual configuration	Automatic scaling
Management	Requires manual updates and patches	Fully managed by AWS
Security	Configured manually	Integrated with AWS IAM roles and policies
Best Use Case	Flexible schema applications	High-performance AWS-native applications

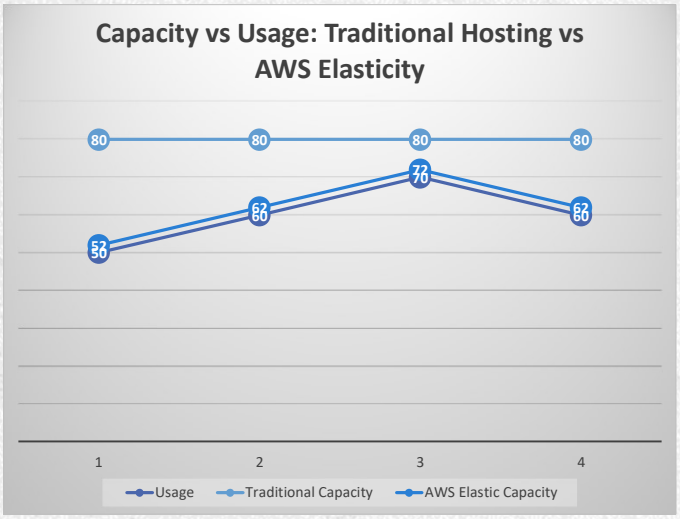
DYNAMODB VS MONGODB

- **MongoDB:**
  - Document-based
  - Requires manual updates
- **DynamoDB:**
  - Key-value store
  - AWS-managed service
  - Scales automatically

We moved from MongoDB to DynamoDB because DynamoDB is fully managed by AWS and integrates with IAM for secure access. While both are NoSQL databases, DynamoDB's auto-scaling and AWS integration made it the better choice for our serverless setup.

CLOUD DEVELOPMENT PRINCIPLES

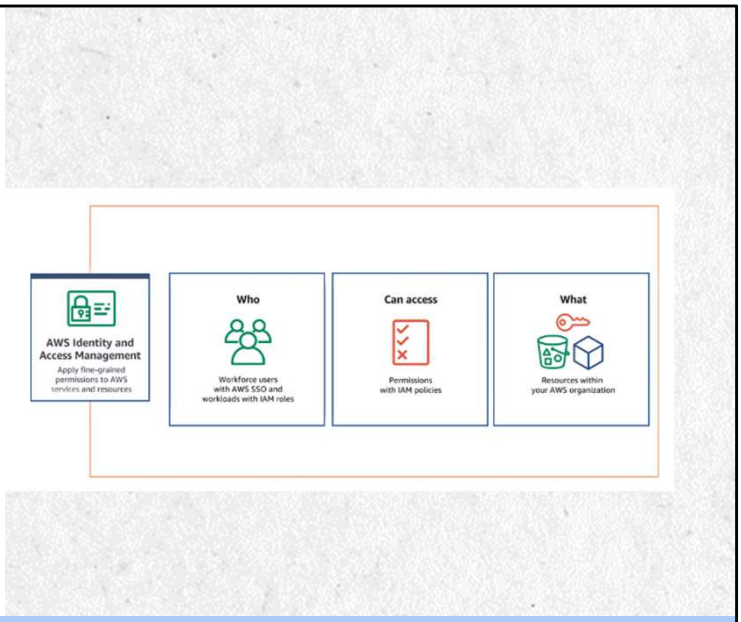
- Elasticity: Auto-scale with demand
- Pay-for-use: Only pay for what is consumed



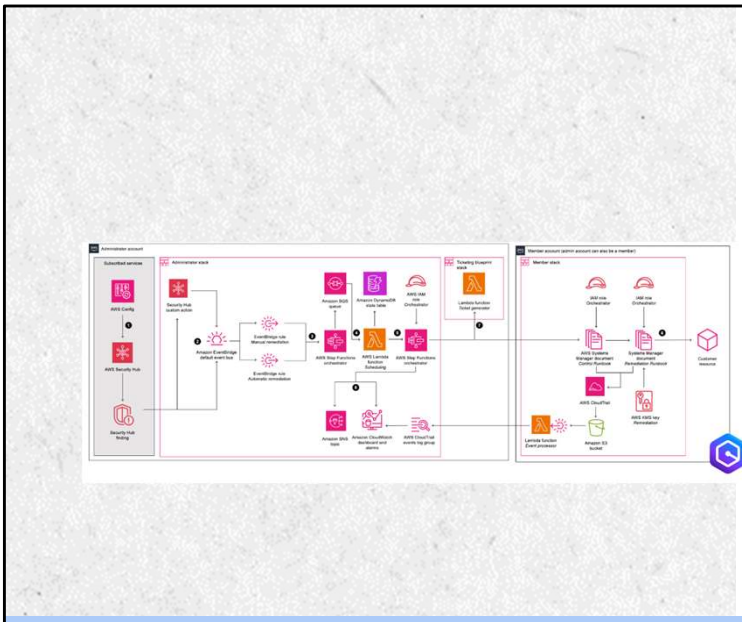
Elasticity allows our application to adjust resources automatically based on traffic. The pay-for-use model ensures we only pay for the exact resources we consume, avoiding waste and reducing costs.

SECURING THE CLOUD APPLICATION

- IAM roles and least privilege
- Custom Lambda access policies
- API keys for secure API Gateway access
- Controlled S3 bucket permissions



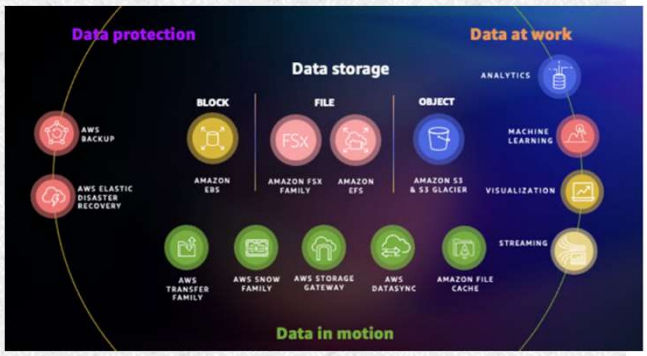
Security is a priority in cloud development. We used IAM roles with the principle of least privilege to limit access. Custom policies-controlled Lambda permissions, API keys secured API Gateway endpoints, and S3 bucket settings ensured the right level of public or private access.



API & DATABASE SECURITY

- S3 → Lambda: Secured by AWS
- API Gateway → Lambda: Secured with API keys
- Lambda → DynamoDB: IAM roles and policies

We moved from MongoDB to DynamoDB because DynamoDB is fully managed by AWS and integrates with IAM for secure access. While both are NoSQL databases, DynamoDB's auto-scaling and AWS integration made it the better choice for our serverless setup.



CONCLUSION

- Flexible and scalable architecture
- Reduced costs with pay-for-use
- Strong security with AWS-managed services and custom policies

Migrating to AWS serverless improved scalability, reduced costs, and strengthened security. By combining containerization, AWS-managed services, and proper security policies, we created a robust, maintainable, and efficient full stack solution. Thank you for your time.

THANK YOU