Project Two Conference Presentation: Cloud Development

Winnie Kwong

Southern New Hampshire University

CS-470 Full Stack Development II

Professor Ogoh

August 18th, 2024

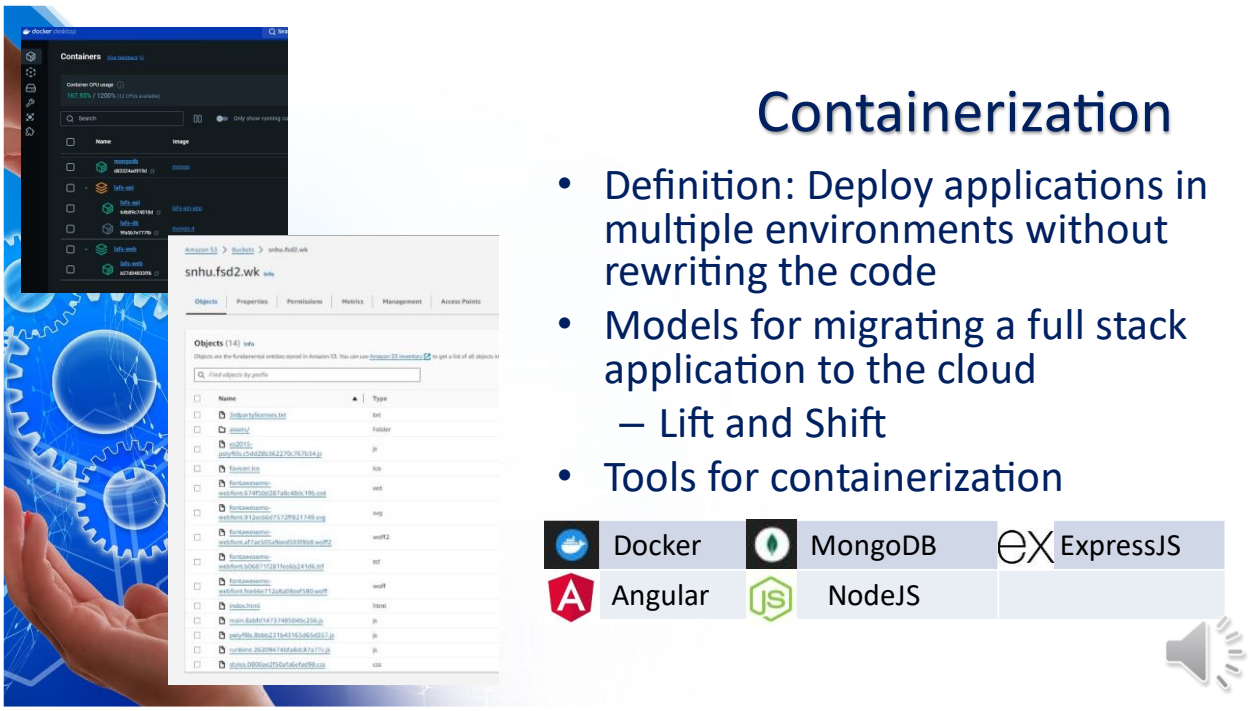
https://youtu.be/iQcXYdPEK6w

Imagine trying to transfer massive amounts of data from a rapidly growing application. Slow speeds, security risks, and skyrocketing costs can quickly become overwhelming. That's the challenge we faced at Southern New Hampshire University. Hello, everyone. I'm Winnie Kwong and I'll demonstrate how we successfully overcame these hurdles by migrating our full-stack application from Docker to AWE services. Let's dive in.

Overview

- About Me
  - Status: Senior at SNHU
  - Majoring in Computer Science
- Purpose of Presentation
  - Understanding the fundamentals of developing and migrating a full stack application using containers to a cloud - based system.

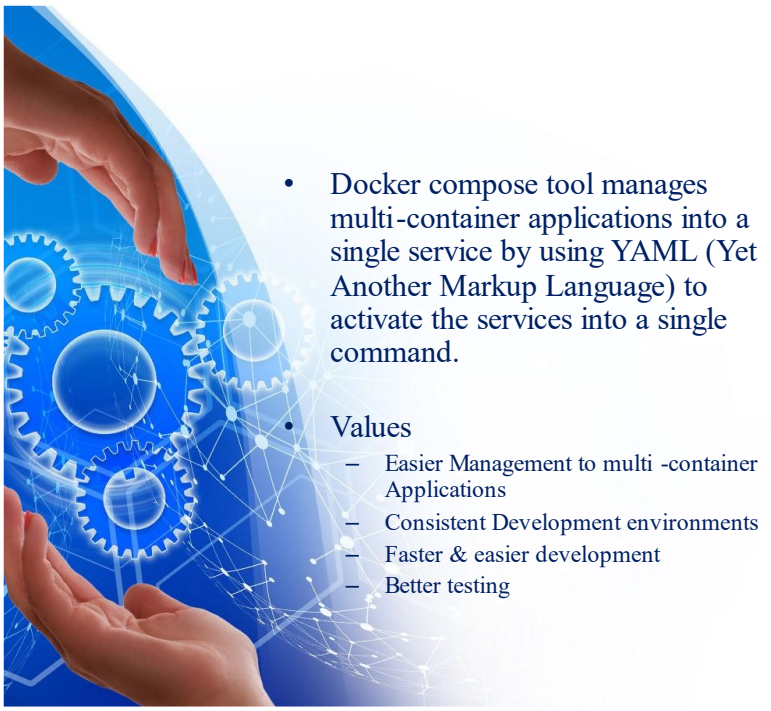
A little about me: I am a senior, and this is my last term before graduation. I majored in computer science with a core concentration in software engineering. Focusing back on the presentation, I'll walk you through the migration process, the benefits we achieved, and key lessons learned to fully understand how developing and migrating a full-stack application can be achieved by using containers to a cloud-based system.



Containerization. We can define it as a deployment application in multiple environments without rewriting code. This means, we can bundle applications and their dependencies under one roof, anywhere and anytime. Having containers enables developers to work on the front end and back end, saving time and resources. Some benefits include isolation, where the units can run independently without worrying if they would interfere with one another, scalability to optimize resource utilization and performance, and faster time-to-market.

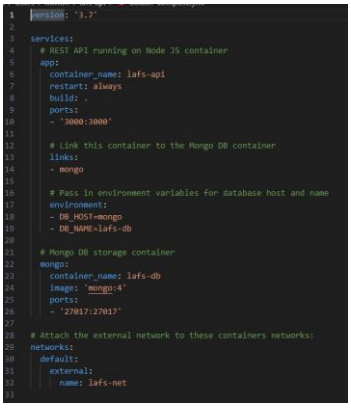
The tool used to migrating a full stack application to the cloud is the lift and shift. The lift and shift model, or rehosting, is used to migrate a full-stack application to the cloud, without alterations to the code. We migrated all of the containers from the Docker desktop and moved all the data into the AWS S3 bucket. It's a simple process that causes minimal disruption to the operation, like picking up a book and setting it on the table.

The first tool for containerization is docker, a platform to help build, ship, and run applications for the front-end, API, and database containers. The following tools are the MEAN stack, which stands for MongoDB, the NoSQL document database; ExpressJS, the Node.js framework to build applications; Angular, the open-sourced web application; and NodeJS, the JavaScript runtime to build the application. Together, the MEAN stack helps to find a solution to creating a dynamic web application from the database to the user interface.



Orchestration

- Docker compose tool manages multi-container applications into a single service by using YAML (Yet Another Markup Language) to activate the services into a single command.
- Values
  - Easier Management to multi -container Applications
  - Consistent Development environments
  - Faster & easier development
  - Better testing



```
version: '3.7'

services:
  # REST API running on Node JS container
  app:
    container_name: lafs-api
    restart: always
    build: .
    ports:
    - '3000:3000'

    # Link this container to the Mongo DB container
    links:
    - mongo

    # Pass in environment variables for database host and name
    environment:
    - DB_HOST=mongo
    - DB_NAME=lafs-db

  # Mongo DB storage container
  mongo:
    container_name: lafs-db
    image: 'mongo:4'
    ports:
    - '27017:27017'

# Attach the external network to these containers networks:
networks:
  default:
    external:
      name: lafs-net
```

Orchestration. Docker-compose tool manages multi-container applications into a single service by using YAML, short for yet another markup language, to activate the services into a single command. Using YAML means we can easily configure the application's services, networks, and volumes into a single file, making it more manageable for complex applications. There are a ton of benefits to using docker-compose. The first is the management of multiple containers, enabling you to start, stop, and rebuild all the services with a single command. Secondly, there's a lot of consistency because the development environment mirrors the production environment, making it easier to debug and troubleshoot issues. The third is faster and easier deployment, which allows you to quickly iterate your application and test changes without jumping into the containers individually. Lastly, docker-compose offers better testing to identify and fix issues during development.

The Serverless Cloud

Serverless

- Definition: A cloud computing execution model that manages applications and services.
- Advantages
  - Management Overhead
  - Scalability
  - Cost efficient
  - High availability
  - Developer productivity

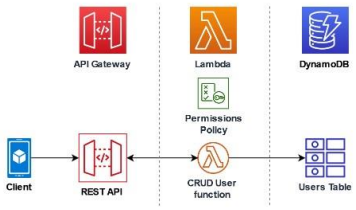
Comparison	S3 (Simple Storage Service)	Local Storage
Definition	Cloud-based storage service used to store and retrieve data wherever and whenever.	Data storage that is directly attached to a computer or device.
Scalability	Unlimited (Virtual storage capacity)	Limitations based on physical storage capacity
Performance	High performance, suitable for heavy workloads	Faster access speed, best for read/write operations
Cost	Pay-as-you go	Upfront investments

The serverless cloud. The serverless cloud is a computing execution model that manages applications and services. Some advantages of serverless clouds include management overhead, scalability, cost-efficiency, high availability, and developer productivity. Let's look at the example comparing S3 and local storage. While S3 has virtual storage capacity, local storages have limitations depending on the developer's physical storage capacity. Next, looking at performance, S3 is best for working with heavy workloads. While local storage can access data faster, it's best for reading and writing operations. Lastly, with cost, S3 offers a pay-as-you-go that makes it easier for developers to handle traffic, whether it's scaling up or down, while local storage requires you to pay everything upfront, making it difficult to manage traffic and a chance of wasting resources.

The Serverless Cloud

API (Application Programming Interfaces) & Lambda

- Advantages of API
  - Reliability
  - Robust Security
  - Seamless Integration
- Lambda API Logic.
  - Creating RESTful API to handle incoming API request and responses.



- Steps to integrate the frontend with the backend
  1. API Gateway receives API Request
  2. API Gateway routes request to Lambda function
  3. Lambda function processes the request to perform all executed actions
     1. CRUD (Create, Read, Update, Delete)
  4. Function generates response and returns to API Gateway then client.

APIs offer numerous advantages, including reliable access to data and functionality, robust security through authentication, authorization, and encryption, and seamless integration across systems and applications.

The Lambda API logic handles all API requests and generates responses. A RESTful API uses HTTP methods of GET, POST, PUT, and DELETE to perform CRUD operations that adhere to REST architectural principles. Each Lambda function was configured with API keys to ensure only authorized users could interact with the API and data.

API Gateway serves as a bridge between frontend and backend as shown in the image. It receives API requests from the frontend, routes them to the appropriate Lambda function based on defined endpoints. The Lambda function processes the request and performs all actions to generate a response, which is then returned to the client via the API Gateway.

The Serverless Cloud

Database

Comparison	MongoDB	DynamoDB
Data Model	Flexible (JSON-like), multi-table model	Key-value pairs with fixed schemas, uses single-table model
Use Cases	Complex data structure	Low-latency application with large datasets
Scalability	Potential underutilization	On demand scaling
Cost	Higher costs due to provisioned resource	Cheaper due to pay per-use
Performance	Affected by data size and complexity	High performance optimized for specific workload

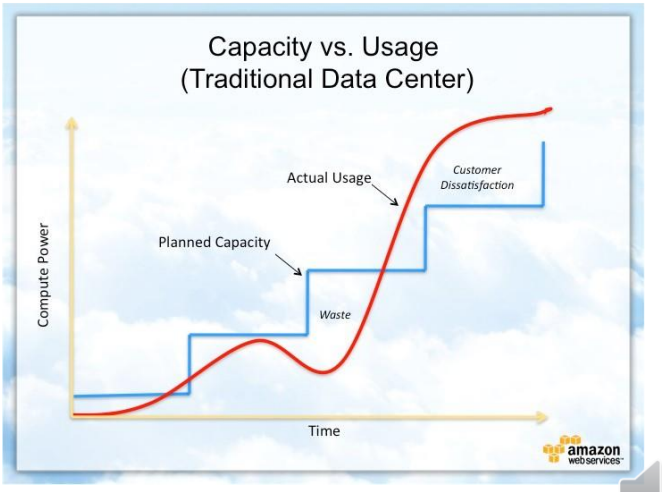
- Queries performed
  - Find, create, read, update, and delete record from database
  - Written in AWS Lambda functions to perform the serverless tasks
- Scripts used
  - JavaScript

Although MongoDB and DynamoDB have similarities, they have slight differences as well. Looking at the table, the first is the data model. MongoDB is more flexible with its multi-table model using JSON-like documents, meaning data can be altered without existing data, making it easy to adapt to unpredictable structures. DynamoDB uses key-value pairs and a single table to optimize read and write operations. In terms of scalability and cost, MongoDB is more expensive because of its provisioned resources, risking potential underutilization. DynamoDB is much cheaper as it offers a pay-per-use and on-demand scaling. When focusing on performance, MongoDB has a disadvantage because it could be affected by data size and complexity, while DynamoDB automatically scales for the best optimization.

When creating the queries performed, the application managed database records using CRUD operations. These operations were implemented as AWS Lambda functions triggered by API Gateway endpoints. The Lambda functions were written in JavaScript, which executed the necessary query logic for each operation.

Cloud-Based Development Principles

- Elasticity
  - Ability to adjust its resource allocation to accommodate changes in a workload.
  - Provides flexibility and efficiency to scale up or down.
- Pay-for-use model
  - Customers only pay for what is used.
  - No upfront costs
  - Scalability and Flexibility



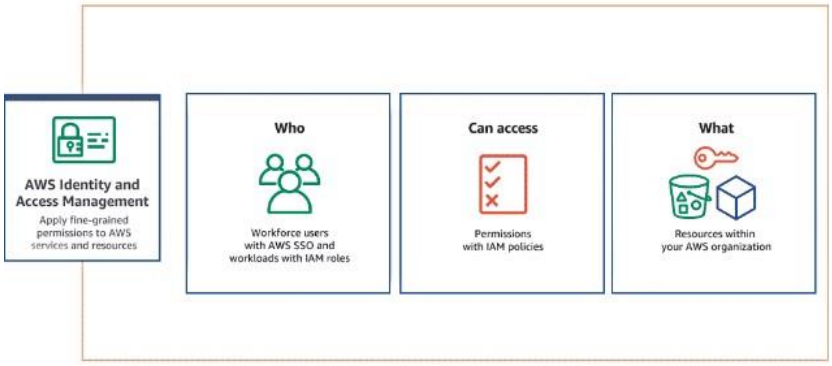
Elasticity is the ability to adjust resource allocation to accommodate changes in a workload. It provides flexibility and efficiency to scale up or down. The pay-for-use model enables customers to only pay for what is being used. Some of the key characteristics of the model include no upfront fees or long-term contracts, scalability on demand, and flexibility to adjust to the developer's needs.

The capacity versus usage chart illustrates the challenges of traditional data centers. The x-axis represents time, while the y-axis denotes compute power. The stepped blue line signifies planned capacity, acquired in discrete hardware increments. In contrast, the fluctuating red line depicts actual usage, demonstrating the dynamic nature of computing demands. This discrepancy often leads to either underutilized resources or insufficient capacity, highlighting the inefficiencies of traditional data center models.

Securing Your Cloud App

Access

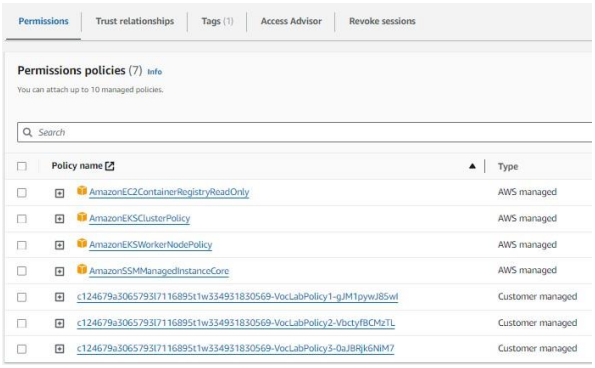
- Prevent unauthorized access
  - Strong password policy
  - Two-factor authorization
  - AWS Identity and Access Management (IAM)
  - Apply least privilege principle



Access. To improve security, robust password policies and two-factor authentication are essential. Implementing Identity and Access Management (IAM) further strengthens protection by identifying, verifying, and authorizing users and devices. The principle of least privilege, a cornerstone of IAM, ensures users are granted minimal access necessary to perform their job, which significantly reduces the risk of unauthorized actions and data breaches.

Securing Your Cloud App

Policies



- Relationship between polices and roles
  - Roles: Type of IAM identity
  - Policies: Define permission of the IAM identity
- Custom policies created
  - LabRole
    - AWS modified the roles to include needed permissions

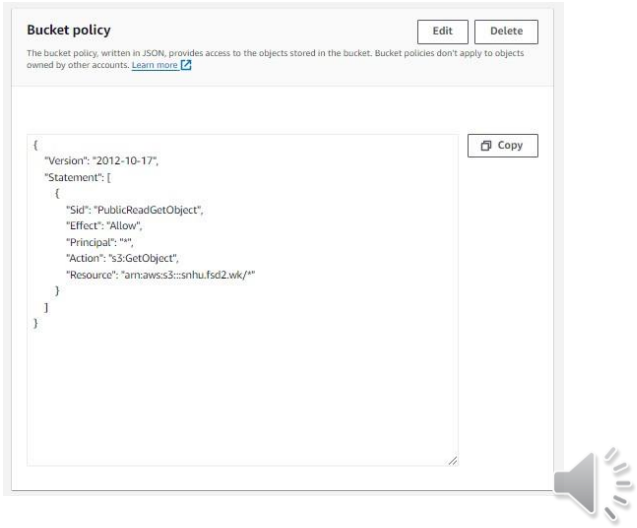
Policies. IAM roles and policies define permission to access AWS resources. The critical difference between the two is that roles are a type of IAM identity used for authentication and authorization to use AWS resources. At the same time, policies define the permissions of the IAM identity.

A custom role was created, LabRole. Custom policies were automatically applied to the role to include all necessary permissions, as shown in the image.

Securing Your Cloud App

API Security

- The connection between Lambda and Gateway can be secure by the API access and IAM roles
- Lambda and the database can be secure by IAM roles, database encryption, and least privilege principle.
- S3 Bucket can be secure by bucket policy, IAM roles, and encryption



API Security. Securing connections involves multiple layers. API Gateway and Lambda communication is fortified through API keys and IAM roles. Database interactions rely on IAM roles, encryption, and the principle of least privilege. For S3 buckets, security is upheld by bucket policies, (shown in the image to the right), IAM roles, and encryption. This approach ensures robust protection for data and applications within the serverless architecture.

CONCLUSION

- Three main points to demonstrating cloud development
  - Serverless architecture with AWS Lambda and Gateway can build scalable applications without managing the technical complexities, enabling developers to focus on application logic.
  - Security is the foundation in cloud environments and requires a multi layered approach, including IAM, encryption, and network security measures to safeguard data and applications.
  - Serverless cloud are cost efficient that quickly and easily manage traffic flow and pay based on used resources needed.

Thank you for your time.

Conclusion. The three main points that demonstrating cloud development are:

- number 1. Serverless architecture with AWS Lambda and Gateway can build scalable applications without managing the technical complexities, enabling developers to focus on application logic.
- Number 2. Security is the foundation in cloud environments and requires a multi-layered approach, including IAM, encryption, and network security measures to safeguard data and applications.
- And lastly, number 3. Serverless cloud are cost efficient that quickly and easily manage traffic flow and pay based on used resources needed.
- I hope you've enjoyed this presentation. Thank you for your time.

REFERENCES

[Angular_full_color_logo.svg]. Wikipedia. https://en.m.wikipedia.org/wiki/File:Angular_full_color_logo.svg

[APIG_tut_resources]. AWS. https://docs.aws.amazon.com/lambda/latest/dg/services -apigateway -tutorial.html

[IAM -aws]. Geeksforgeeks . https://www.geeksforgeeks.org/identity -and -access -management -iam -in -amazon -web -services -aws/

[Deploy -Docker -Container -to -AWS _ -A -Straightforward -Guide -1]. Mobilise.Cloud . https://www.mobilise.cloud/deploy -docker -container -to -aws /

[Expressjs_logo_icon_169185]. Icon -Icons. https://icon -icons.com/icon/expressjs -logo/169185

[Logo, Icon, and Brand Guidelines _ Docker]. Docker. https://www.docker.com/company/newsroom/media -resources/

[Mongodb -icon -1]. WorldVectorLogo . https://worldvectorlogo.com/logo/mongodb -icon -1

[_Node.js_ Icon - Download for free – Iconduck]. Iconduck . https://iconduck.com/icons/27728/node -js