



# **CS 470 PROJECT TWO CONFERENCE PRESENTATION: CLOUD DEVELOPMENT**

Isaac Jang

December 2024



Name: Isaac Jang

Date: December 15<sup>th</sup>, 2024

Assignment name: Project Two Conference Presentation: Cloud Development

YouTube Link: <https://youtu.be/vyBYjdBZOVQ>

Welcome everyone to the CS 470 Project Two Conference presentation on cloud development

# INTRODUCTION

- ❖ Name: Isaac Jang
- ❖ Current Senior @ SNHU
- ❖ Majoring in Computer Science
  
- ❖ Discussing Cloud development in both technical and non-technical aspects



My name is Isaac and I am a current senior at SNHU majoring in Computer Science.

Today we will be discussing the intricacies of cloud development to both technical and nontechnical aspects

# CONTAINERIZATION

- Migrating from full stack to the cloud:
  - Lift & Shift, Replatforming, and Refactoring
  - Primarily used Replatforming
  
- Tools We Used:
  - Docker / Docker Compose



In CS 465, we built a full stack application and in this class we wanted to migrate to the cloud.

When migrating to the cloud, there are 3 models or ways to go about it. There is lift & shift, which is taking the application “as is” and hosting on the cloud. Second, there is replatforming, which is making minor adjustments for the cloud. And third is refactoring, which entails redesigning the application to use cloud services.

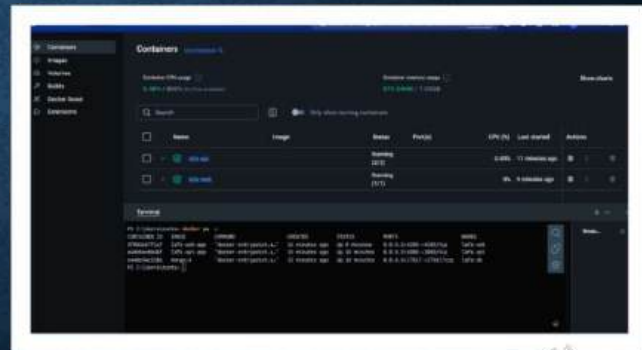
For this class, we primarily used the replatforming model, making small adjustments to take advantage of the cloud

To containerize, the tool that we used was Docker. With docker, we were able to package each part of the app (frontend, backend, and database) into containers that can run anywhere

# ORCHESTRATION

## ➤What is the value of using Docker Compose?

- Multiple Containers
- Configure file: docker-compose.yml
- Easy to Set Up and Share
- Automates Connections
- Saves Time



What is the value of using Docker Compose?

Docker Compose is a tool that helps you manage and run multiple containers as a single system.

### **Multiple Containers:**

Docker Compose lets you define how all the containers work together in one configuration file (docker-compose.yml).

### **Easy to Set Up and Share:**

You can share the docker-compose.yml file with your team, and they can run the entire app setup on their machines without additional steps.

### **Automates Connections:**

Docker Compose automatically connects containers so they can talk to each other (e.g., the backend connects to the database).

### **Saves Time:**

You only need to write one command to start, stop, or restart all the containers, which saves time during development and testing.

# THE SERVERLESS CLOUD

## What is “Serverless”?

- Servers are managed by cloud provider (AWS, Azure, GCP)

Describe the advantages of using a serverless API?

- Advantages:
  - No Server Management
  - Scalability
  - Cost Efficiency
  - High Availability



## What is “Serverless”?

Contrary to what it might seem, serverless doesn't mean there are zero servers. It means that the servers are managed by the cloud provider. For simplicity, I am going to say AWS but all the cloud providers offer a similar services.

There are many advantages to switching to the cloud.

### No Server Management

As mentioned before, you don't need to spend the time or money to set up servers and maintain them. AWS handles it for you

### Scalability

AWS automatically adjusts to handle more traffic and scales down when there is less

### Cost Efficiency

The big reason why many companies are switching is the cost savings. AWS operates on a consumption model, meaning you only have to pay for what you use. If the server is idle, you do not have to pay.

### High Availability

Serverless services are designed to be reliable and highly available without extra effort

# THE SERVERLESS CLOUD

What is S3 storage and how does it compare to local storage?

➤ S3 (Simple Storage Service)

➤ Advantages:

- Accessibility
- Scalability
- Durability
- Cost
- Integration



What is S3 storage and how does it compare to local storage?

S3 stands for Simple Storage Service. In S3, Data is stored in "**buckets**" and can be accessed via the internet.

There are many advantages to using S3:

## **Accessibility:**

S3 is accessible from anywhere with an internet connection. Local storage is limited to the local device.

## **Scalability:**

S3 is virtually unlimited storage capacity whereas local storage are limited by the size.

## **Durability**

S3 Data is backed up across multiple AWS locations. Local Data can be lost if the device fails.

## **Cost:**

For S3, you pay only for what you use. Local storage requires you buying physical devices upfront.

**Integration:**

S3 easily integrates with other AWS services. Local storage requires manual setup for connections.



# THE SERVERLESS CLOUD - LAMBDA

## Lambda API Logic:

- Event-Driven Execution
- Stateless Design
- Integration with Services
- Minimal Code for Maximum Output

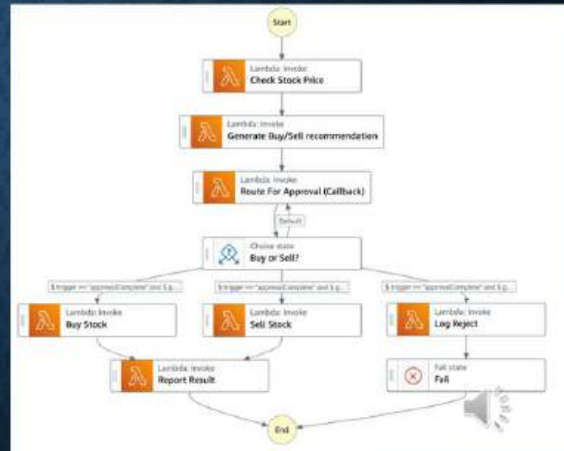


Image Reference:  
Smith, B. (2022, October 27). *Implementing a UML state machine using AWS Step Functions*. AWS Compute Blog. Retrieved from <https://aws.amazon.com/blogs/compute/implementing-a-uml-state-machine-using-aws-step-functions/>

## Event-Driven Execution:

Lambda is triggered by API Gateway requests.

## Stateless Design:

Each function runs independently, without maintaining session state.

## Integration with Services:

Reads/writes data to DynamoDB, processes files in S3, etc.

## Minimal Code for Maximum Output:

Handles requests, processes data, and returns results efficiently.

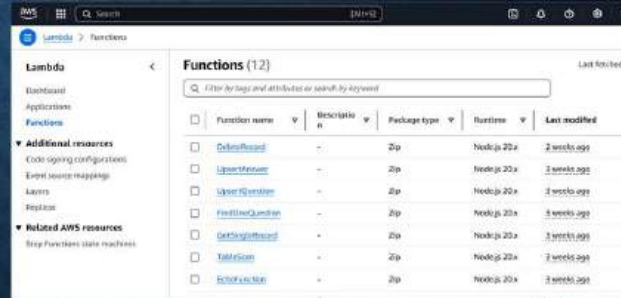
Image Reference:

Smith, B. (2022, October 27). *Implementing a UML state machine using AWS Step Functions*. AWS Compute Blog. Retrieved from <https://aws.amazon.com/blogs/compute/implementing-a-uml-state-machine-using-aws-step-functions/>

# THE SERVERLESS CLOUD - SERVERLESS INTEGRATION

## Serverless Integration:

- Lambda Function Code
  - CRUD Operations
- IAM Role Policies
- API Gateway Configuration



For serverless integration, there are a couple things that we need to do.

First is the lambda function code. Also known as the crud operations, we have to write the functions for creating, reading, updating and deleting data from the DynamoDB.

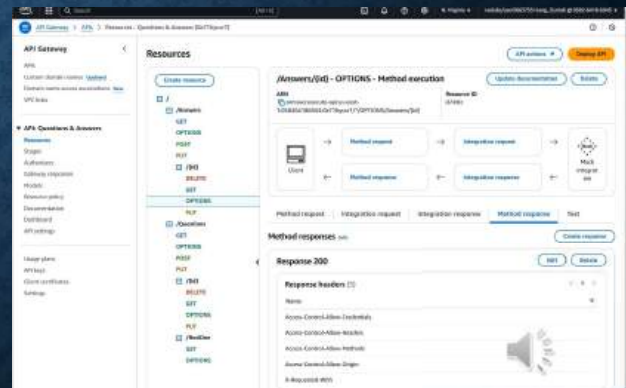
Second is IAM Role Policies. These are Scripts that define permissions for Lambda functions to access resources like DynamoDB or S3.

Third is configuring API Gateway Configuration. This can be doing by script or manual step-up that map endpoints to lambda functions.

# THE SERVERLESS CLOUD - INTEGRATE FRONTEND WITH BACKEND

## Steps to Integrate Frontend with Backend:

- Update Environment Variables
- Enable & Configure CORS
- Deploy Frontend
- Test Integration



To integrate frontend with back end, there are 4 actions that I took.

First is updating the environmental variables. I added API Gateway URL in the frontend config (environment.prod.ts).

Second is Enable CORS. I Configured the CORS headers in API Gateway for secure communication.

Third is Deploying Frontend. I Built the Angular app and upload it to the S3 bucket.

Fourth is Test Integration. I Ensured that the frontend API calls routed to backend Lambda functions through API Gateway.

# THE SERVERLESS CLOUD - DATABASES

## MongoDB vs. DynamoDB

Feature	MongoDB	DynamoDB
<b>Data Model</b>	Document-based (JSON-like BSON documents)	Key-value and document-based
<b>Schema</b>	Schema-less, flexible, and dynamic	Requires a primary key and optional sort key
<b>Indexes</b>	Manual indexing required for performance	Built-in indexing (global and local indexes)
<b>Storage</b>	Self-hosted or cloud options available	Fully managed by AWS with auto-scaling

What are the differences between MongoDB and DynamoDB?

### MongoDB

- Data Model:** Document-based (JSON-like BSON documents).
- Schema:** Schema-less, flexible, and dynamic.
- Indexes:** Manual indexing required for better query performance.
- Storage:** Self-hosted or cloud options available.

### DynamoDB

- Data Model:** Key-value and document-based.
- Schema:** Requires a primary key (partition key) and optionally a sort key.
- Indexes:** Built-in indexing (global and local secondary indexes).
- Storage:** Fully managed by AWS with automatic scaling and backups.

# THE SERVERLESS CLOUD – DATABASE QUERIES

## MongoDB Queries

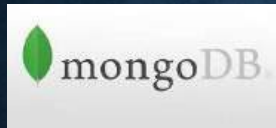
```
1 Find Documents:
2 db.collection('questions').find({ category: 'AWS' });
3
4 Insert a Record:
5 db.collection('questions').insertOne({ question: "What is
  DynamoDB?", category: "AWS" });
6
```

## DynamoDB Queries

```
1 Get an Item by Key:
2 const params = {
3   TableName: 'Questions',
4   Key: {
5     id: '12345'
6   }
7 };
8 dynamoDB.get(params).promise();
9
10 Scan for All Items:
11 const params = { TableName: 'Questions' };
12 dynamoDB.scan(params).promise();
```

Here you can see the queries for MongoDB and DynamoDB

## THE SERVERLESS CLOUD – DATABASE SCRIPTS



### MongoDB

- Node.js code for connecting to MongoDB using the MongoDB client.
- CRUD operations written for interaction with collections.



### DynamoDB

- Lambda Functions: Scripts to handle GetItem, PutItem, Scan, and Query requests.
- IAM Role Policies: Scripts granting Lambda permissions to perform DynamoDB actions (dynamodb:Query, dynamodb:Scan, etc.).

**MongoDB uses** Node.js code for connection. CRUD operations are written for interaction with collections

**DynamoDB uses** Lambda Functions to handle GetItem, PutItem, Scan, and Query requests.

IAM Role Policies grant Lambda permissions to perform DynamoDB actions (dynamodb:Query, dynamodb:Scan, etc.).

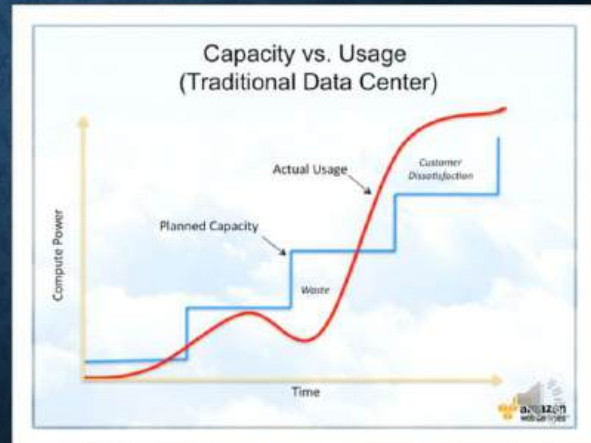
# CLOUD-BASED DEVELOPMENT PRINCIPLES

## Elasticity:

- Auto-Scaling
- Benefits

## Pay-for-Use Model:

- Pay for what you consume
- Benefits



## Elasticity

### Definition:

Elasticity refers to the ability of cloud systems to automatically adjust resources (like compute power or storage) based on demand.

### How It Works:

When demand increases (e.g., during traffic spikes), more resources are automatically allocated.

When demand decreases, resources are scaled down to avoid waste.

### Benefits:

Cost savings: You only use what you need.

Seamless user experience: High traffic doesn't slow down your application.

## Pay-for-Use Model

### Definition:

The pay-for-use model charges you only for the resources and services you actually consume.



**How It Works:**

Cloud providers track usage (e.g., compute time, storage space, data transfer) and bill based on consumption.

**Benefits:**

No upfront costs: No need to invest in physical hardware.

Cost efficiency: You're not paying for unused resources.

**Chart****Planned Capacity (Blue Line):**

- Represents the fixed amount of compute power allocated to meet expected usage.
- Capacity is added in steps because scaling infrastructure in traditional data centers requires planning and investment.

**Actual Usage (Red Line):**

- Shows the real demand for compute power over time.
- Demand fluctuates and often grows unpredictably.

**Waste (Between Blue and Red Lines):**

- When planned capacity exceeds actual usage, resources are wasted.
- Organizations pay for compute power that is not being used.

**Customer Dissatisfaction:**

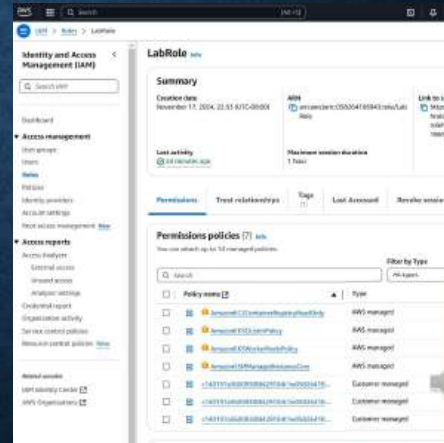
- When actual usage exceeds planned capacity, there is not enough compute power to meet demand.
- This results in slow performance or system failures, leading to customer dissatisfaction.



# SECURING YOUR CLOUD APP - ACCESS

## How can you prevent unauthorized access?

- Use IAM roles and policies to control access to AWS resources.
- Implement least privilege access to grant only necessary permissions.
- Enable Multi-Factor Authentication (MFA) for critical accounts



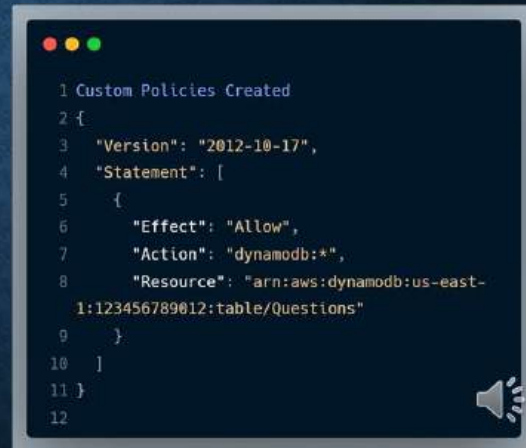
## How can you prevent unauthorized access?

- Use IAM roles and policies to control access to AWS resources.
- Implement least privilege access to grant only necessary permissions.
- Enable Multi-Factor Authentication (MFA) for critical accounts

# SECURING YOUR CLOUD APP - POLICIES

## Roles vs Policies

- IAM roles define who can access resources, while policies define what actions they can perform.

A code editor window with a dark background and light-colored text. It displays a JSON policy document for AWS IAM. The text is as follows:

```
1 Custom Policies Created
2 {
3   "Version": "2012-10-17",
4   "Statement": [
5     {
6       "Effect": "Allow",
7       "Action": "dynamodb:*",
8       "Resource": "arn:aws:dynamodb:us-east-
9         1:123456789012:table/Questions"
10    }
11  ]
12 }
```

A small speaker icon is visible in the bottom right corner of the code editor window.

## Roles vs Policies

- IAM roles define who can access resources, while policies define what actions they can perform.

On the right, you can see one of the custom policies that were created

# SECURING YOUR CLOUD APP - API SECURITY



**Secure Lambda and API Gateway Connection**



**Lambda and the Database**



**S3 Bucket**



For Api security, we did a couple things.

## **Secure Lambda and API Gateway Connection:**

We used IAM authorization to restrict API Gateway access to specific users or services.

## **Lambda and the Database:**

We also attach IAM roles to Lambda to grant access to DynamoDB with minimal permissions.

## **S3 Bucket:**

Lastly, we Used bucket policies to allow public read access only for frontend files and disabled public write access to prevent unauthorized uploads.

# CONCLUSION

**So what? Why is this important?**



**LIFE IS  
EASIER**



**SAVES  
MONEY**



**MORE  
SECURE**



**So what? Why is this important?**

## **Life is easier with Serverless Architecture**

By leveraging AWS Lambda and API Gateway, you eliminate the need to manage servers. This reduces overhead, ensures automatic scaling, and allows developers to focus on writing business logic.

## **Consumption model saves money**

Cloud services like AWS automatically scale resources up or down based on demand. With the pay-for-use model, you only pay for the resources consumed, saving costs compared to traditional infrastructure.

## **More Secure Through IAM and Policies:**

IAM ensures secure access to resources. By defining roles and fine-grained policies, you can prevent unauthorized access and enforce the principle of least privilege.