
Aaron Shipley
04/13/2023
Project Two Conference Presentation: Cloud Development
Link: https://youtu.be/rsQKdHmI6aA

Hello Everyone,

My name is Aaron Shipley and today we will be discussing the process of deploying a local full-stack web application with cloud services.

# Overview

- **Local application to the Cloud**
- **Tools Include : Docker and AWS**
- **Containerization**
- **Serverless Cloud**
- **Security**
- **Advantages / Challenges**

First, we start with a local web application. The application was built with the MEAN stack. This stack incorporated MongoDB, Express.js, Angular, and Node.js. The application would then be started and deployed via a localhost server on the local machine. This type of deployment is limited to only the one machine, unless the exact framework, files, and versions are applied to another machine in order to launch it. To make the deployment of the web application easier on other machines, tools like Docker and AWS services can be used to orchestrate and containerize the application files and versions to be loaded automatically to another machine for deployment.

We will also discuss the serverless cloud and how it plays a role in the deployment of the application. Through this we will delve into the security aspects of a cloud-based application, as well as the advantages and disadvantages of this type of deployment.

# Containerization

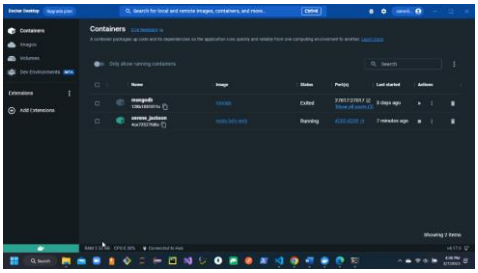
- **Utilize docker to create containers.**
- **Docker Compose to run multiple containers**
- **Easy migration to other machines**
- **All files, libraries and versions contained in containers**
- **Run frontend and backend of application with correct setup**

Containerization is the process of taking all files and resources of an application and bundling all files and dependencies of the application into a container in order to allow the application to be able to be portable, scalable, and more generic in what operating system can run it. This is done via Docker. Docker is a tool used to create such containers for applications.

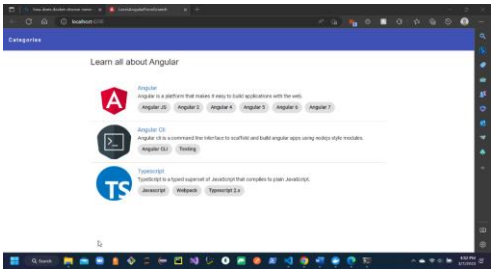
Our application utilized Docker to create containers for the application frontend (client-side), backend (server-side), and the Mongo DB database for the application.

# Examples

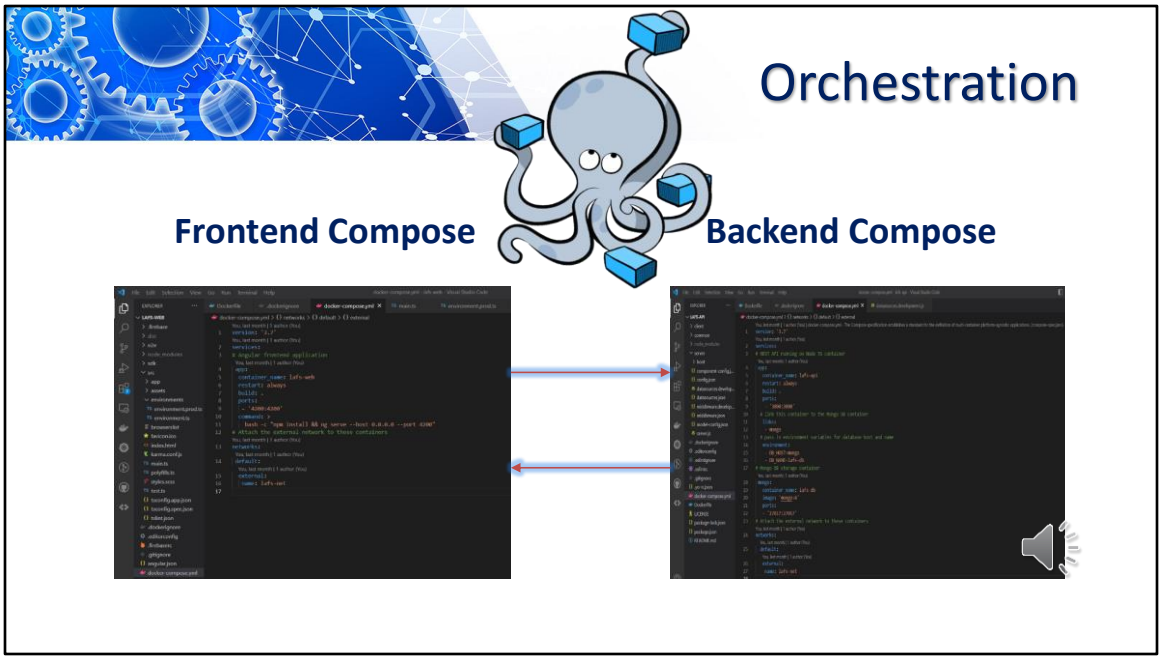
**Containers**




**Web Application**



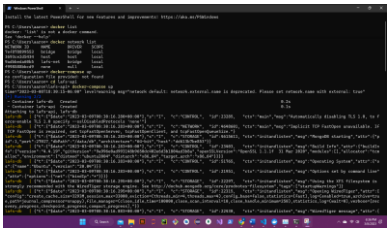

On the left is a screenshot of the containers created with Docker. It shows a MongoDB container for the application's database and a lafs-web container that contains the web application. The second slide shows the web application running properly via the startup of the web app container. This application is just the frontend (client-side) of the application running. It does not have access to the database or the backend of the application.



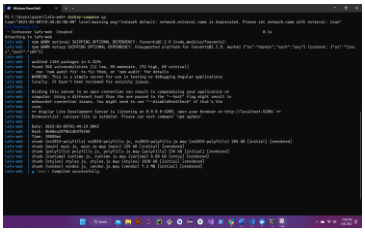

So how do we get the frontend, backend, and database to communicate? By creating the three containers we need a way to run multiple containers under a single command. This process is called orchestration. Docker compose is a tool that allows the orchestration of an app with multiple containers to be shared and used under a single command. Via YAML files created in the application. In the slides you can see the YAML files that were created for the purposes of orchestration. These files tie the frontend, backend, and database together.

# Docker Compose

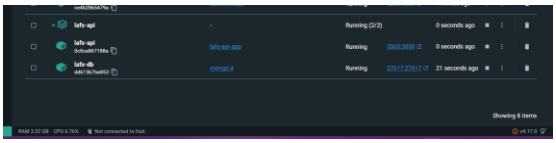
## API compose



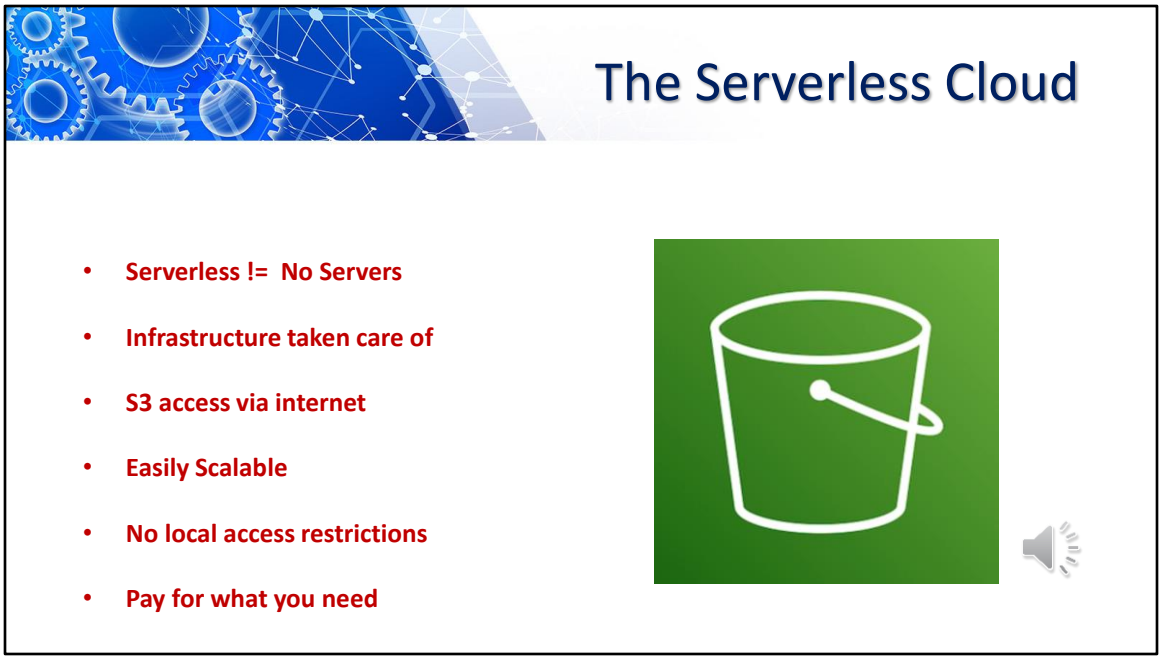
## Web compose



## Virtual Network for Containers




We then will call the compose command in the PowerShell to create the images of the containers in an orchestrated . Before we do this, we need to create a virtual network that will link the composed containers together. Once this is done, the newly composed container images will be linked together and will be able to run under a single command.

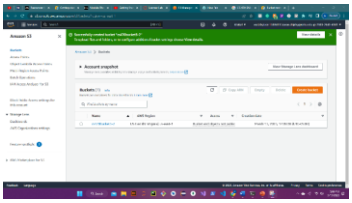



Next, we will look at deploying our application to the cloud-based services. I mentioned in the beginning we will utilize serverless development services. You may be wondering how can it be serverless? Is that possible to run an app without servers? Well in short serverless does not equal no servers. Serverless simply means that the service provides the backend infrastructure, and the user doesn't have to worry about setting it up, maintaining it, or scaling it up or down.

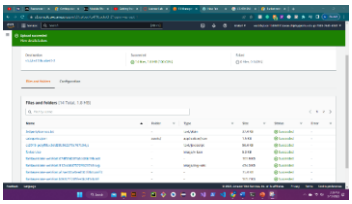
AWS S3 service is easily scalable and accessible anywhere there is an internet connection S3 is a service that stores unstructured data into objects and can scale to the user's needs. Simply clicking a button can scale up or down the storage

# AWS S3

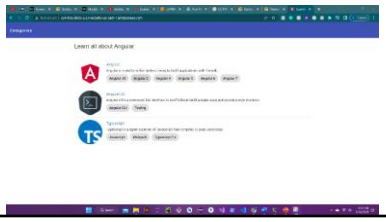
## Created Bucket



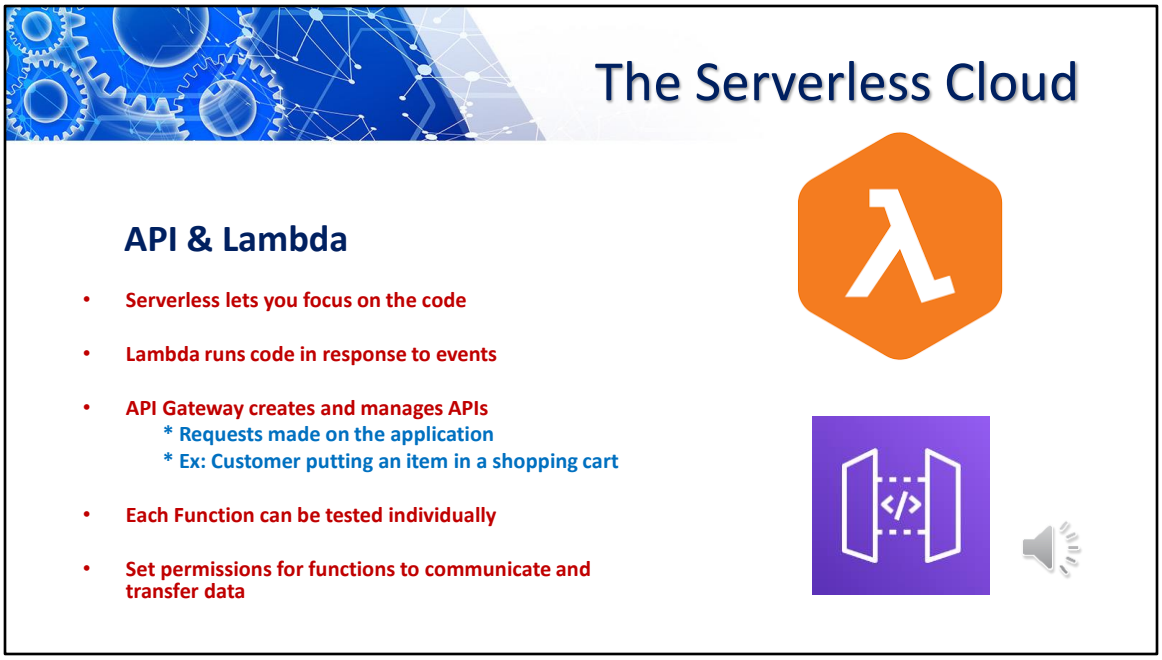
## Upload files for Web Hosting



## Web Application Running



The top left slide shows an object storage in S3 called a bucket. This bucket is where the unstructured data is stored for our application. The bucket contains the files from the web application. The application in essence can be run from the bucket (with small adjustments and accessibility changes). By default, the bucket will be set to private and will need to be set to public in order to run the application.




Next, we will look at the serverless compute model. Now we will look another AWS service for this, Lambda. Lambda simply runs code in response to an event. Lambda stores the code for responses to an event in functions. These functions are created to respond accordingly. As an example, one could create a function that find one dataset in a database that meets a specific criteria. This function will have code that essentially tells the application how to respond to that event.

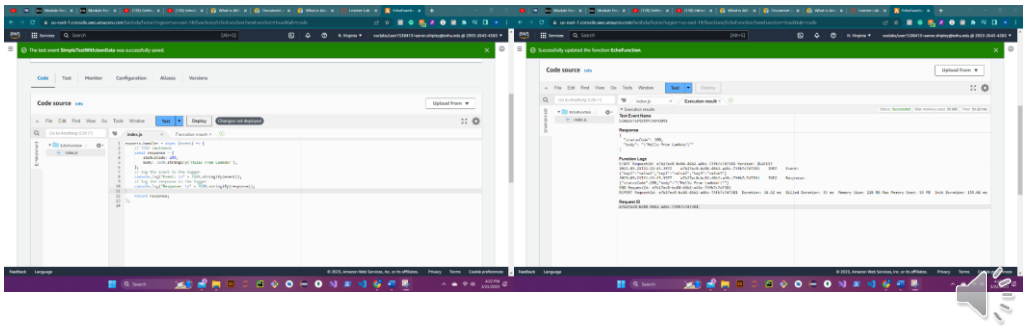
Now we must have a way for the functions to communicate and connect to the database (which we will discuss in further detail later). Requests made on the application can vary, and API Gateway by AWS allows the user to create what type of requests can be made on the application.

Each function can be tested individually by creating test cases in each function to make sure the function is operating as expected. We will see examples of this on the next slide.

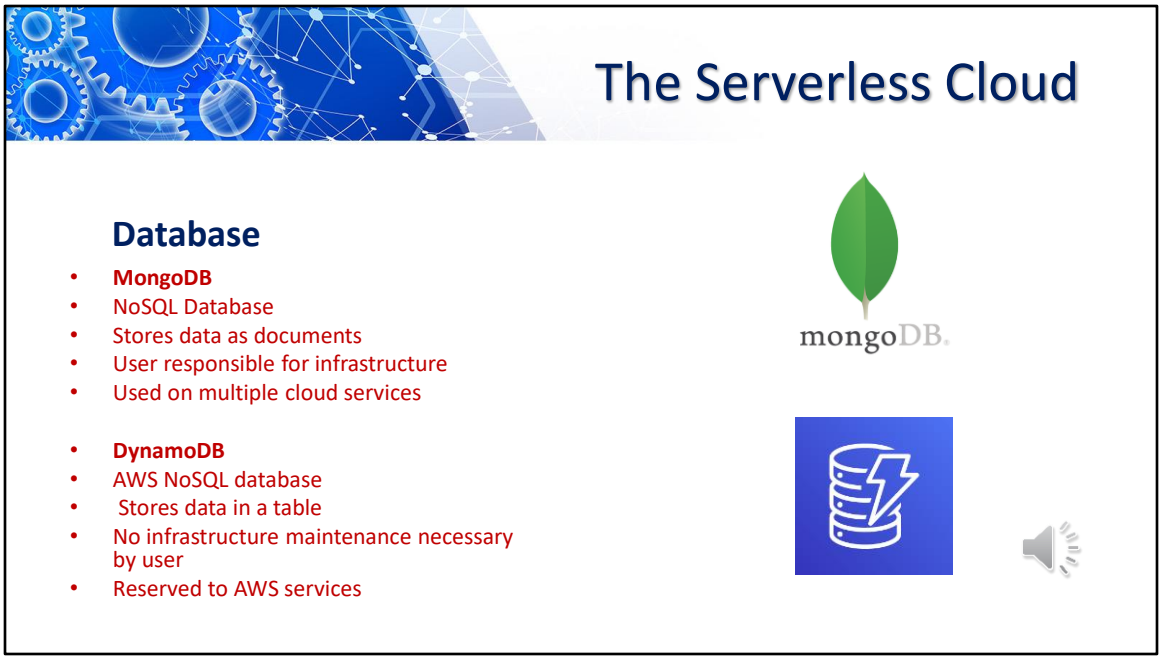
# Lambda Test Case

## Code for Test Case

## Test Case Results



Here are examples of a test case from our application function. It is creating a test event to see if the event gets the expected response. The right slide shows the results of the test, achieving the expected output from the function.



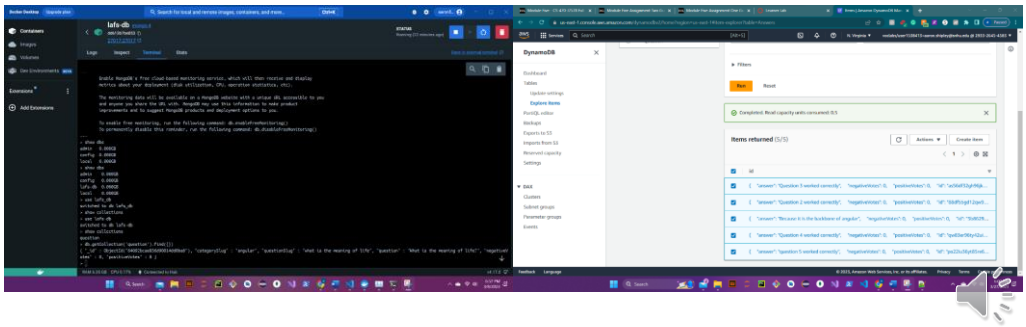

Now we will talk about Databases. We discussed our use of MongoDB briefly at the beginning when our application was still local. But what is Mongo DB? Mongo DB is a NoSQL database that stores unstructured data into documents. With MongoDB the user is responsible for setup, maintenance, and security for the database.

DynamoDB is also a NoSQL database. It stores data into single tables. Tables are then accessed via requests and the data is then retrieved via a key (partition key) value pair. With DynamoDB, there is no responsibility on the user to setup, maintain, or secure it. This is done by AWS. Unlike MongoDB, which can be deployed to many cloud services, DynamoDB is reserved strictly to AWS.

# Database Models

## MongoDB

## DynamoDB

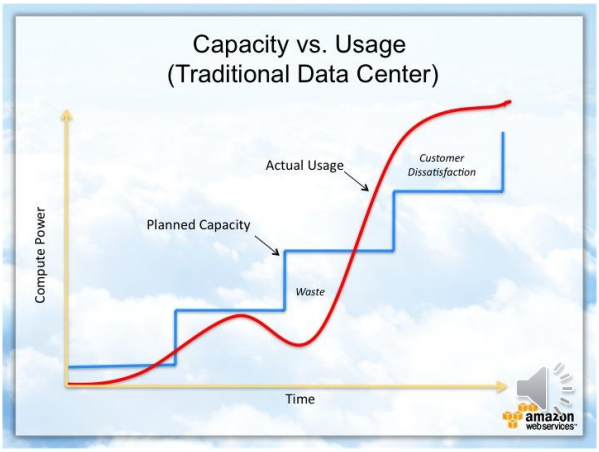


The slide on the left shows our MongoDB that was created for our docker Containers. The database has collections within it where the documents are stored.

The slide on the right shows our DynamoDB table created. The data is stored in the table and given a unique id that can be called up onto retrieve the data. In the case for our app, we created the Question table and Answer table. Each table was linked via ids. When a particular question was requested, the linked answer from the other table was the response. This tested our tables for functionality.

## Cloud-Based Development Principles

- **Elasticity**
- **Pay-for-use model**

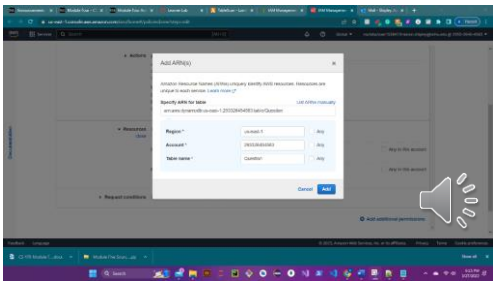

As we delve into the different AWS services, we need to look at financial implications of deploying our application through AWS cloud services. A standard approach to deploying an application locally would be to buy the servers needed, rack them, and store them. Each step of this process is quite expensive. The cost of servers is quite taxing on the bottom line as well as paying for installation and storage of the servers. This can work successfully and has in the past. But what do we do if you need more servers to meet demand for your application? You will have to do this process over to scale up your application. Another issue is having a drop in app traffic and now you have too many servers than what is needed. You have waisted components.

With AWS cloud services, you use a pay as you need model. This model allows users to pay for the storage, servers, and all other hardware on a need basis. AWS lambda, DynamoDB, and S3 are serverless, allowing the app to be automatically scaled up or down as needed. With Lambda, a user will pay per request on the app. This model saves a customer money when hardware isn't needed, scales up when it is needed, and leaves no excess or deficit on the app's needs.

# Securing Your Cloud App

## Access

- **IAM Roles and Policies**
- Select who has access to what in the app
- Principle of Least Privilege
- Add new policies as needed to roles

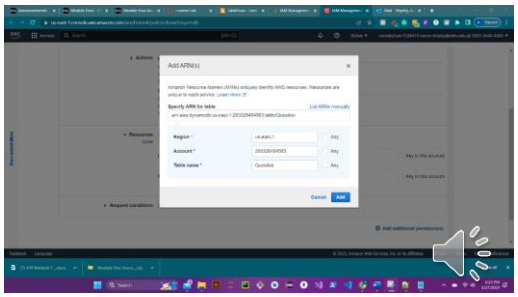


Now what about security for our application when deploying through AWS? With AWS, another service offered is IAM (Identity and Access Management). With IAM a user can create permission and select which users have access with the permission created. Let's investigate this more.

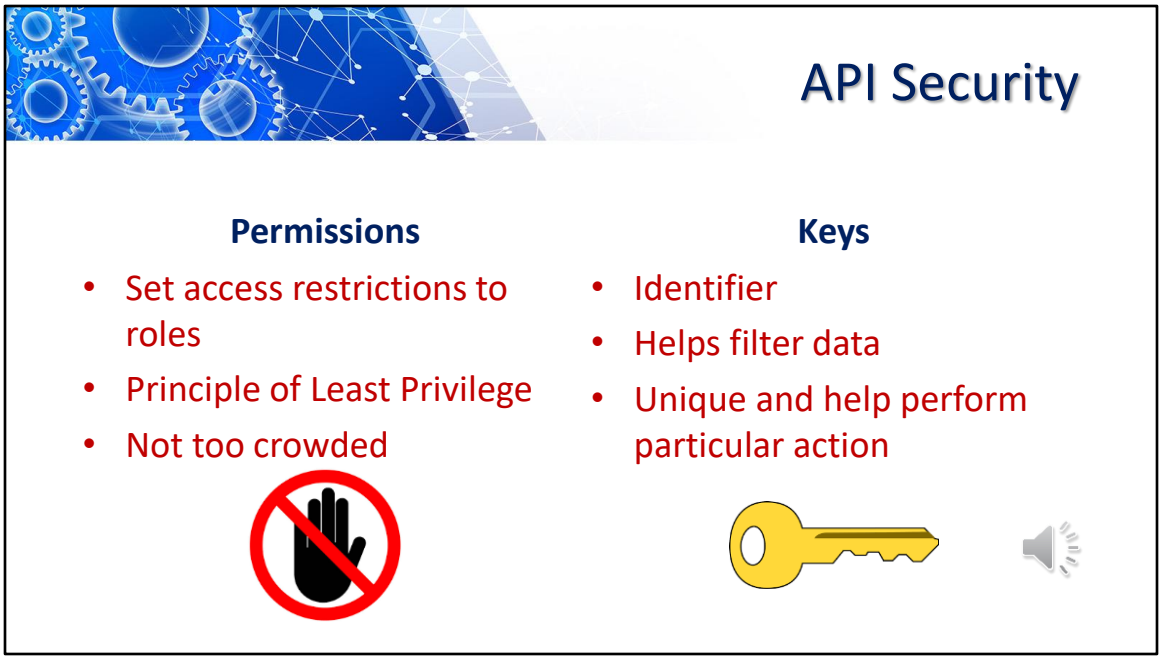
# Policies

## CRUD Functionality

- Specific permissions made for CRUD commands
- Create, Read, Update, Delete
- Manipulate data through requests

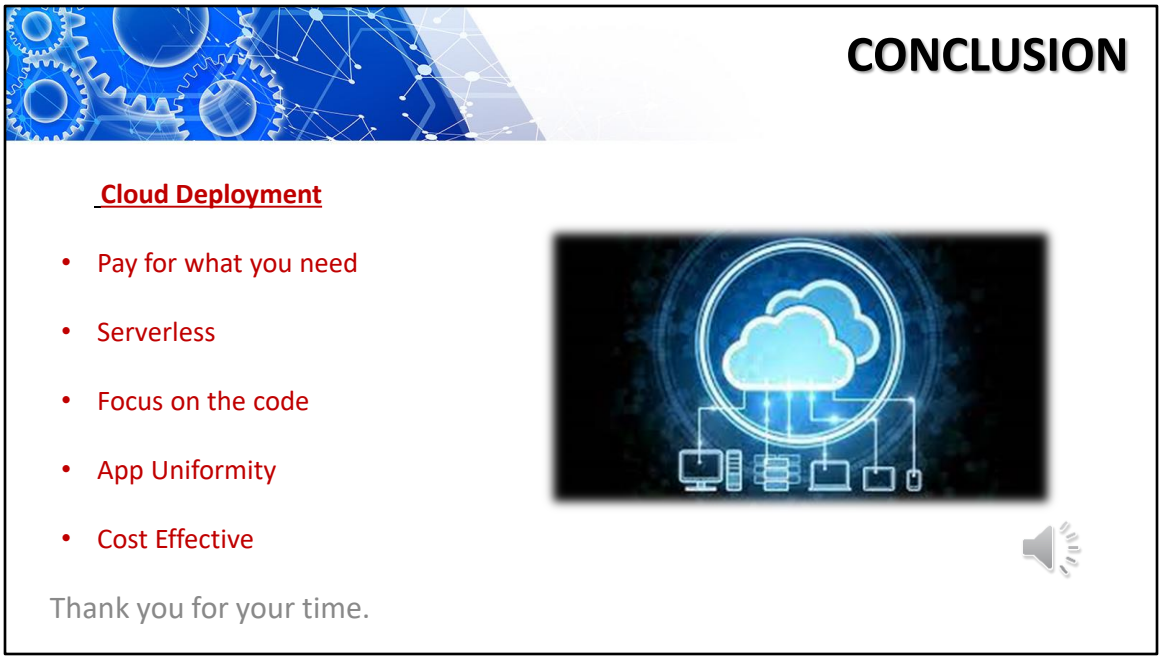


With our application, we use IAM to create permissions for roles that can use the CRUD functions we developed earlier. By creating these permissions, we were able to assign them to specific users, in our case the LabRole role we used, which allowed us to get access to the application in order to manipulate data through these functions.




With the permissions we created, we use the Principle of Least Privilege. This principle states that a user has the minimal amount of access to the system needed to complete their tasks. These permissions are granted to them on a need basis and should not exceed the minimal amount necessary.

Another use of security on the system is with the DynamoDB database generated. Each item within the table has a unique id associated with it. When a request on the application is made, that contains filters, the frontend request will be sent to the backend then on to the database. If the filter contains an id associated with an item within the table, the request will execute for the user. If it is not a match, the request will terminate and provide the user with a message stating this. This is a form of data validation to add yet another layer of security to our application.




In conclusion, by deploying our application with a cloud-based service like AWS, we will only pay for what we need in terms of hardware (servers, storage, etc.). Also, the application will be serverless, allowing us to focus on the code of the application and not the backend infrastructure. All of this will be taken care of by the AWS services used. By deploying through AWS, the app will be uniform in nature due to it being containerized on all parts through docker compose. This will broaden the number of users that can successfully access the application. Finally, by deploying through AWS, it will be cost-effective in terms of paying only for what we need as well as saving time and money by not having to worry about the backend infrastructure of the application as much. Security on the application will allow it to run efficiently and help reduced the threat of unauthorized access or use on the app. Thank you all for time. I hope this presentation proved to you the benefits of AWS app deployment and how it is the right choice for our application and possibly any other future applications.