Catherine Hawley

4/7/2025

Project Two Conference Presentation: Cloud Development

https://youtu.be/wXA_gtUAJBk

Slide 1

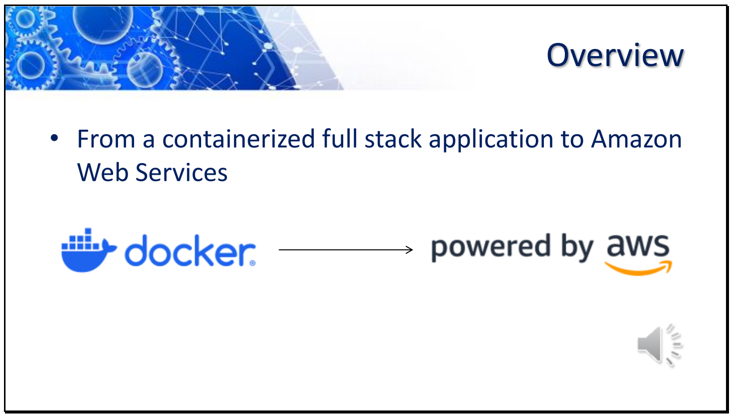
CS 470 Project Two Conference Presentation: Cloud Development

Catherine Hawley

April 2025

My name is Catherine Hawley, and I'm a few short months away from earning my Bachelor's degree in Computer Science. This is my presentation of my final project for CS 470, Full Stack Development Two.

Slide 2



Overview

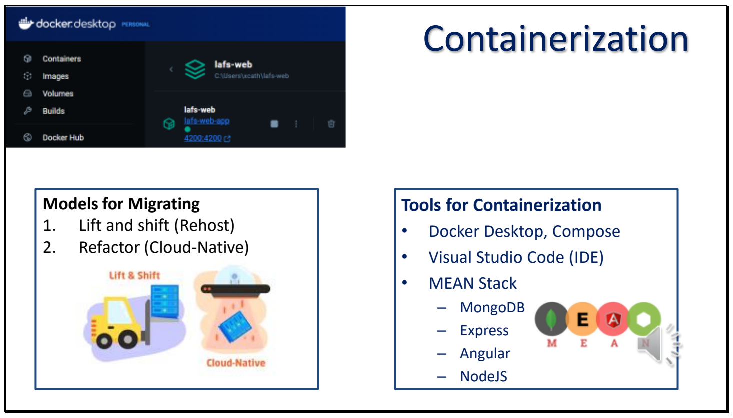
- From a containerized full stack application to Amazon Web Services

docker
powered by aws

Over the last few years I've done a lot of development work for local programs, from quick functions to entire applications. In the last few weeks, I've taken an existing full stack site and migrated it to a cloud-native web application using AWS microservices.

Images: Docker Brand Guidelines, https://www.docker.com/company/newsroom/media-resources/
AWS Trademark Guidelines, https://aws.amazon.com/trademark-guidelines/

Slide 3



Containerization

docker desktop PERSONAL
Containers
Images
Volumes
Builds
Docker Hub
lafs-web
lafs-web
lafs-web-app
4200:4200
Lift & Shift
Cloud-Native
M
E
A

Models for Migrating

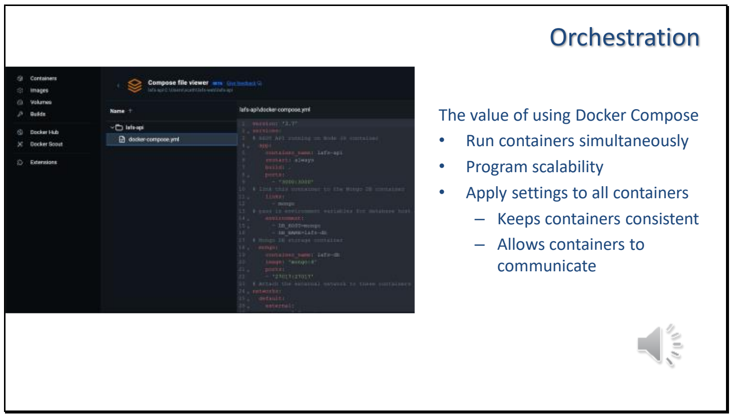
1. Lift and shift (Rehost)
2. Refactor (Cloud-Native)

Tools for Containerization

- Docker Desktop, Compose
- Visual Studio Code (IDE)
- MEAN Stack
  - MongoDB
  - Express
  - Angular
  - NodeJS

To move an application to the cloud, there are two options. Lift and shift is the non-invasive way to move the pieces of the application onto the cloud, changing the application as little as possible. Refactoring involves rethinking the application, changing its structure or functions to better fit and take advantage of the cloud capabilities. For this foray into cloud-based applications, we used the lift and shift method.
This application began with Docker, a container-based virtualization tool. Docker Compose was used to control those containers. The coding was done with Visual Studio Code, and utilized the MEAN stack of MongoDB, Express, Angular, and NodeJS.
Images: Screenshot of my instance of Docker Desktop
Strategies for Lift & Shift Cloud Migration, https://www.linkedin.com/pulse/strategies-lift-shift-cloud-migration-irene
MEAN Stack Best Practices, https://medium.com/@sureshkrishna75525/mean-stack-best-practices-pro-tips-for-efficient-development-in-2023-e035168eff25

Slide 4



Orchestration

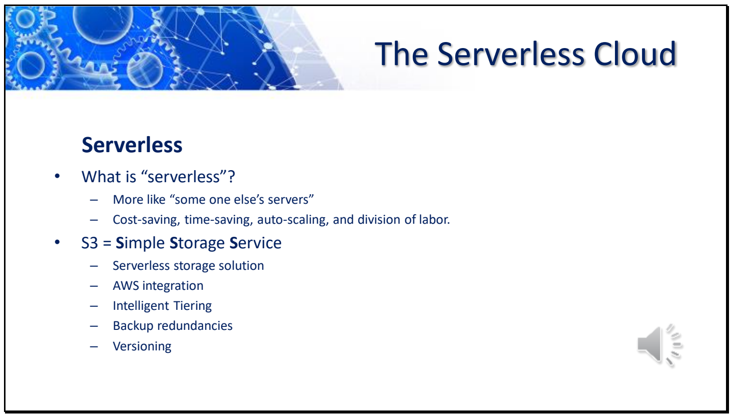
Containers
Images
Volumes
Builds
Docker Hub
Docker Scout
Extensions
Compose file viewer
Give feedback
Name
lafs-api
docker-compose.yml
lafs-api\docker-compose.yml

```
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
    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:
  default:
    external:
```

The value of using Docker Compose
- Run containers simultaneously
- Program scalability
- Apply settings to all containers
  - Keeps containers consistent
  - Allows containers to communicate

Docker Compose allows you to treat all of your containers as a single program, which is helpful because it is. The database container run by Mongo and the API container running on Node can be started and stopped simultaneously using Compose. Compose also offers scalability for container-based applications, which would be complicated if many containers needed to be managed individually.
Settings can be applied to all containers from the docker-compose YAML file, a single convenient place. This ensures settings are compatible between containers, and gives us networking capability, allowing the containers to communicate.

Slide 5



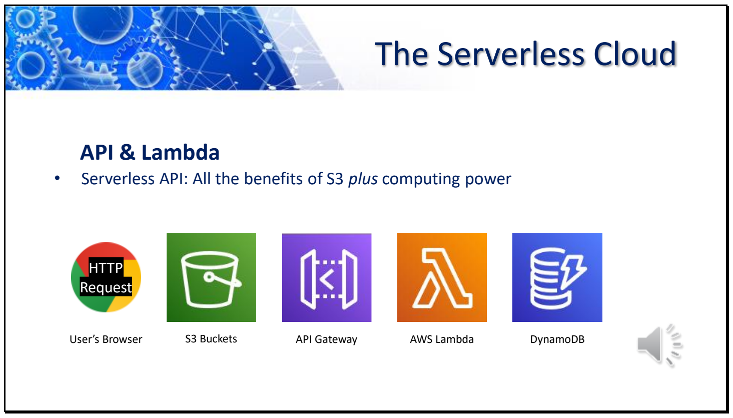
The Serverless Cloud

Serverless

- What is “serverless”?
  - More like “some one else’s servers”
  - Cost-saving, time-saving, auto-scaling, and division of labor.
- S3 = **S**imple **S**torage **S**ervice
  - Serverless storage solution
  - AWS integration
  - Intelligent Tiering
  - Backup redundancies
  - Versioning

What IS serverless?
By “serverless” we mean that our program runs on someone else’s servers. We’re using Amazon Web Services as our cloud provider, so Amazon owns the hardware that we’re using. Server hardware is expensive in terms of time and money, as it requires an initial purchase, setup, and maintenance, along with decisions regarding scale and structure. Allowing AWS to handle those costs and decisions can save a business or individual developer money and time, allowing them to focus on developing the product.
S3 means Simple Storage Service, Amazon’s cloud based storage solution. It brings in all the benefits of serverless architecture, links to other AWS technologies, and a few other S3 perks. Intelligent tiering is a cost-saving measure for less frequently accessed data, backups in multiple locations offer redundancy in case of physical server damage, and versioning allows access to previous versions of the same file.

Slide 6



The Serverless Cloud

API & Lambda

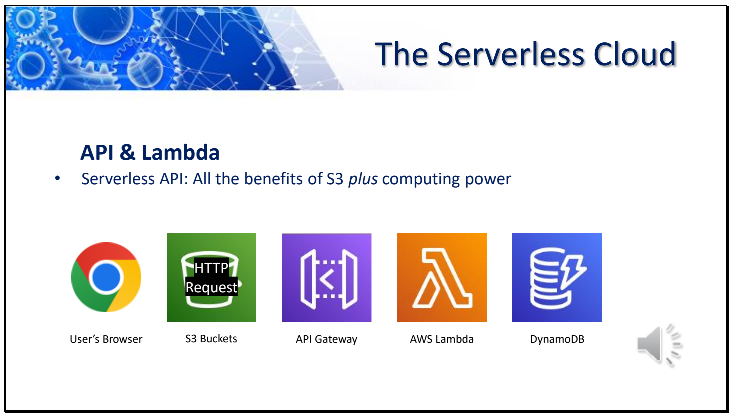
- Serverless API: All the benefits of S3 *plus* computing power

HTTP Request
User's Browser
S3 Buckets
API Gateway
AWS Lambda
DynamoDB

Lambda is event-driven computing. We use NodeJS to write the scripts for our Lambda functions. Each function gets its own block of code, separate from the other functions. This standalone method keeps the code maintainable and understandable, and ensures that each function can be correctly triggered by the API Gateway. The frontend and the backend are integrated by Cross-Origin Resource Sharing, or CORS, which allows our website to pull information through the API in a secure way.

The user's browser sends an HTTP request to the S3 bucket, which returns website files. Then, the user is able to interact with the website. That interaction is sent to the API Gateway in the form of an HTTP request. The gateway routes that request as an event for Lambda. Our Lambda functions are all CRUD operations, which are sent to the database. Dynamo processes that operation and sends a response back through Lambda and the API gateway to the user's browser, giving them the requested information or letting them know the database has been updated.

Slide 7

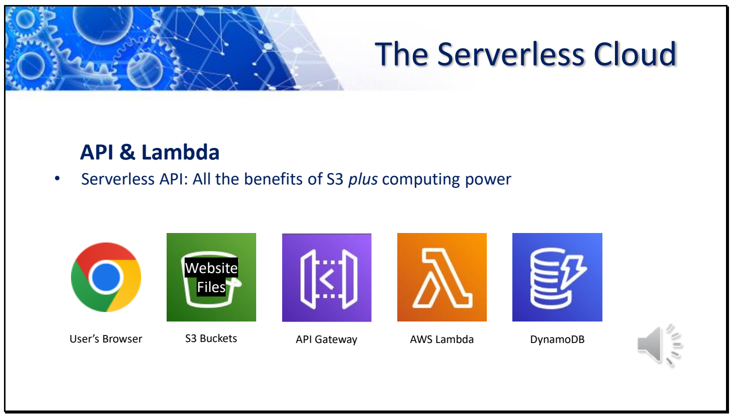


The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Slide 8

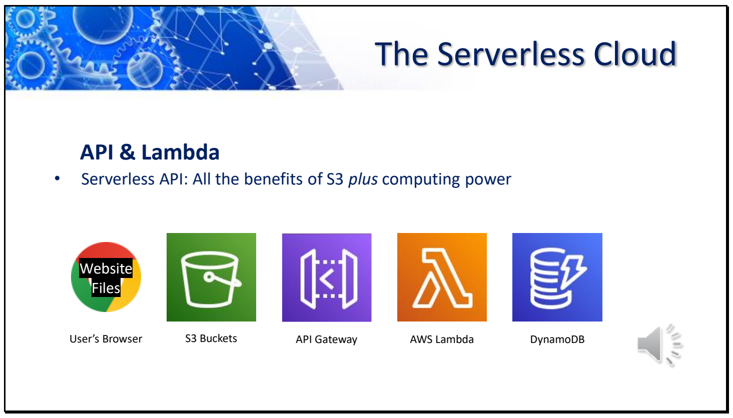


The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Slide 9

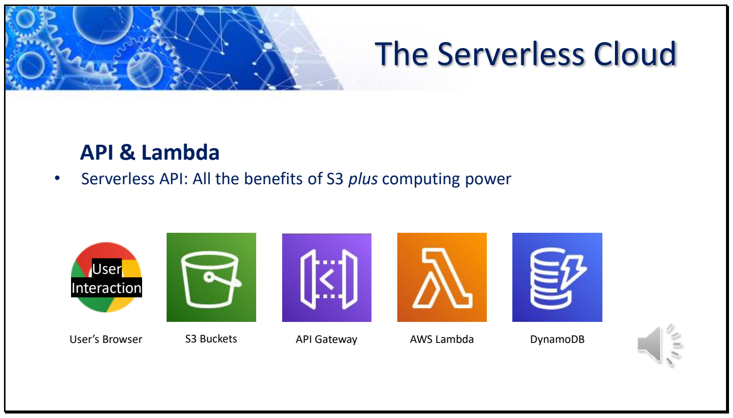


The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Slide 10



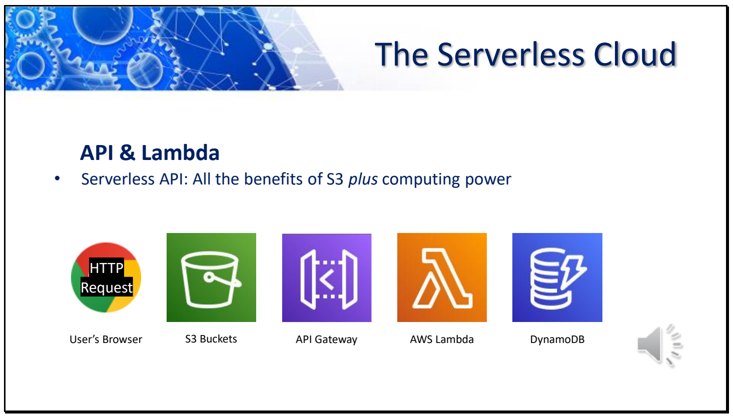
The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

User Interaction
User's Browser
S3 Buckets
API Gateway
AWS Lambda
DynamoDB

Slide 11



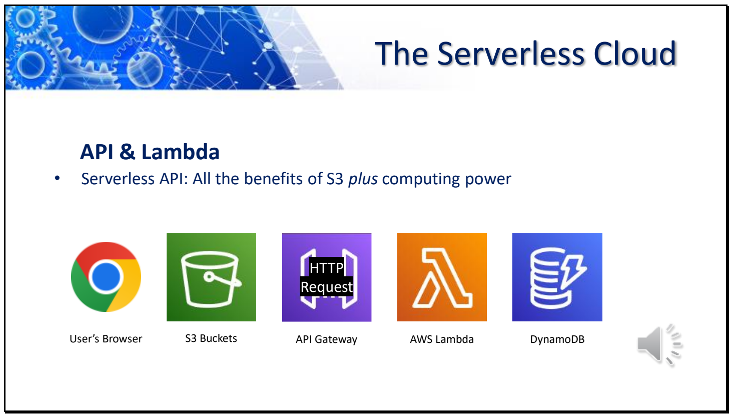
The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

HTTP Request
User's Browser
S3 Buckets
API Gateway
AWS Lambda
DynamoDB

Slide 12



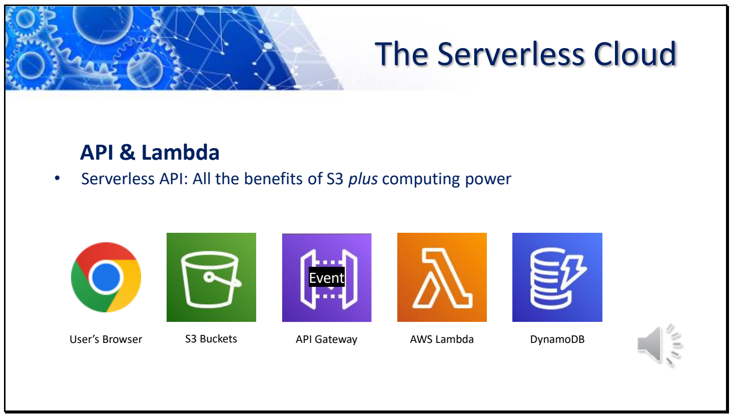
The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

User's Browser
S3 Buckets
HTTP Request
API Gateway
AWS Lambda
DynamoDB

Slide 13

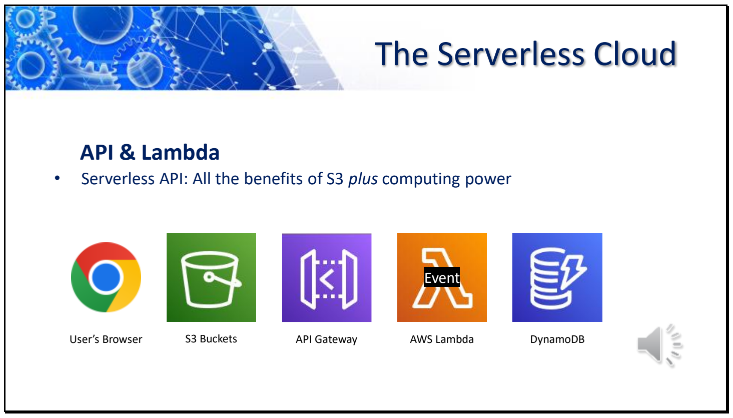


The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Slide 14

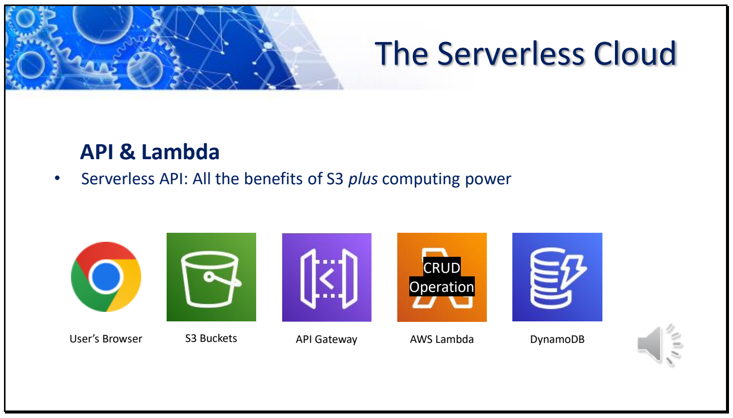


The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Slide 15

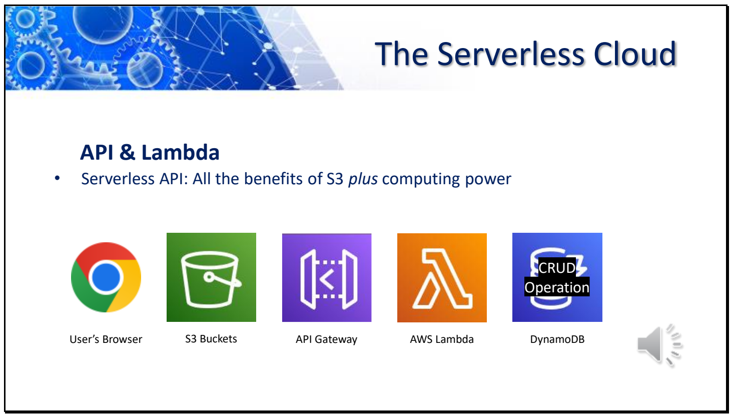


The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Slide 16



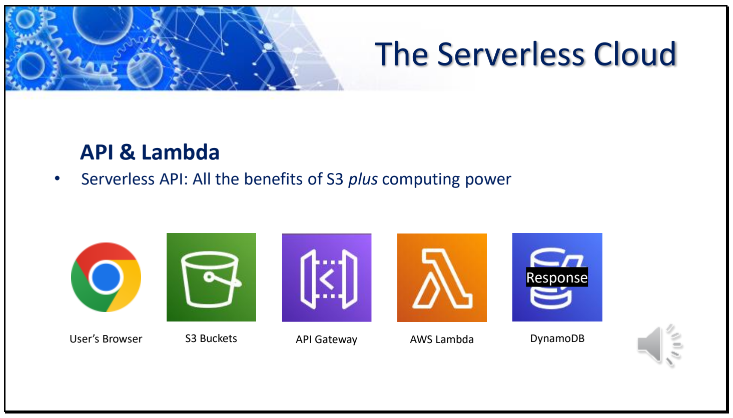
The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

CRUD Operation
User's Browser
S3 Buckets
API Gateway
AWS Lambda
DynamoDB

Slide 17



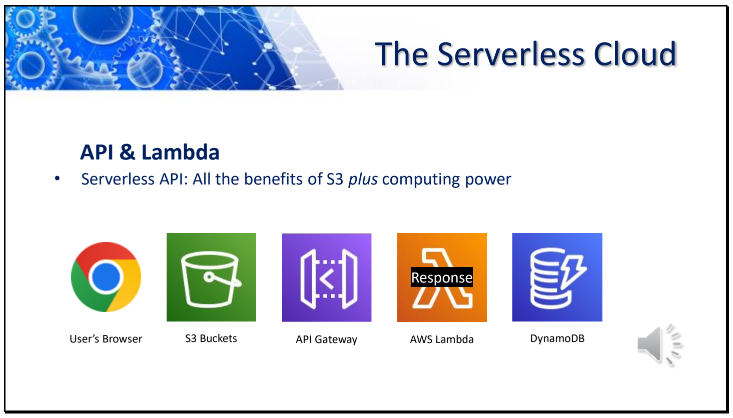
The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Response
User's Browser
S3 Buckets
API Gateway
AWS Lambda
DynamoDB

Slide 18



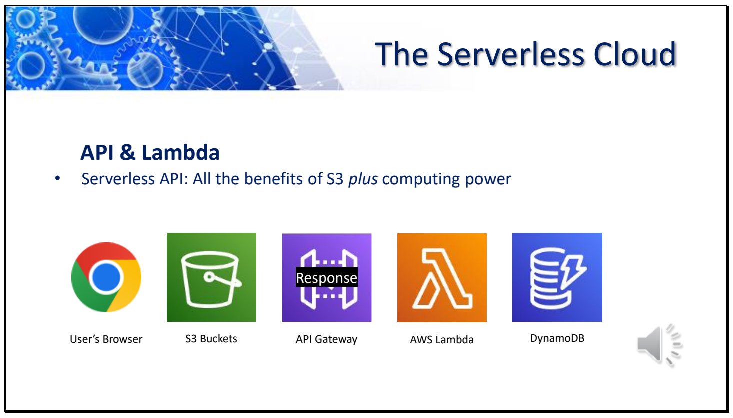
The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Response
User's Browser
S3 Buckets
API Gateway
AWS Lambda
DynamoDB

Slide 19

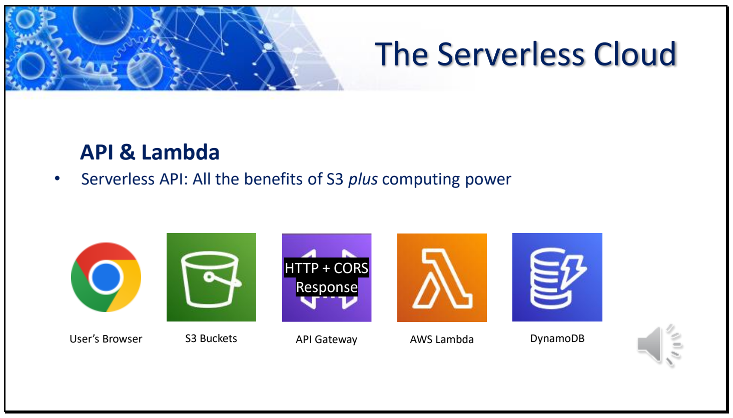


The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Slide 20

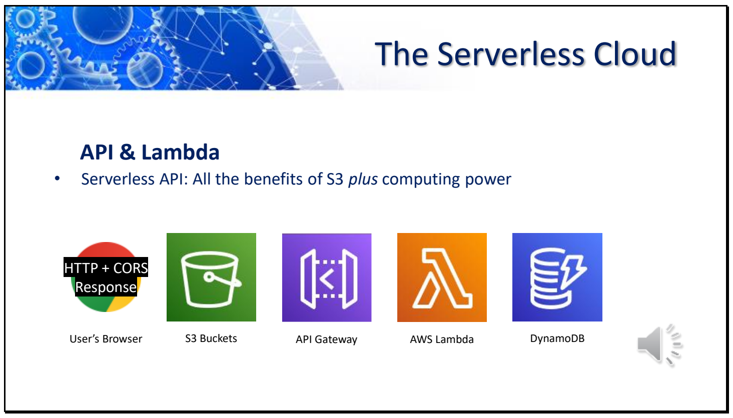


The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Slide 21

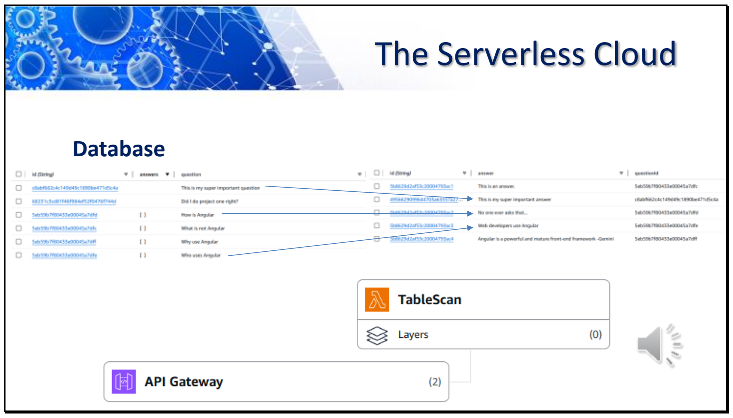


The Serverless Cloud

API & Lambda

- Serverless API: All the benefits of S3 *plus* computing power

Slide 22



The Serverless Cloud

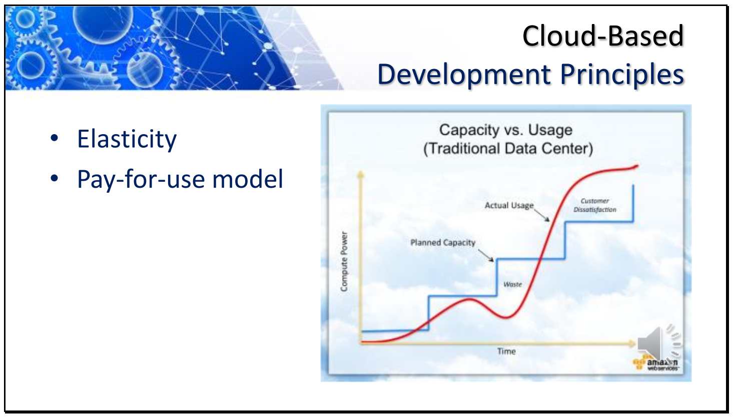
Database

	id (String)	answers	question
☐	c8abf662c4c149d49c1890be471d5c4a		This is my super important question
☐	68251c3cd81f46f984ef52f0476f744d		Did I do project one right?
☐	5eb59b7f80433e00045a7dfd	[ ]	How is Angular
☐	5eb59b7f80433e00045a7dfc	[ ]	What is not Angular
☐	5eb59b7f80433e00045a7dff	[ ]	Why use Angular
☐	5eb59b7f80433e00045a7dfe	[ ]	Who uses Angular

	id (String)	answer	questionId
☐	5b8629d2af55c20004793ac1	This is an answer.	5eb59b7f80433e00045a7dfc
☐	4956629099644703a63557a27...	This is my super important answer	c8abf662c4c149d49c1890be471d5c4a
☐	5b8629d2af55c20004793ac2	No one ever asks that...	5eb59b7f80433e00045a7dfd
☐	5b8629d2af55c20004793ac3	Web developers use Angular	5eb59b7f80433e00045a7dfe
☐	5b8629d2af55c20004793ac4	Angular is a powerful and mature front-end framework -Gemini	5eb59b7f80433e00045a7dff

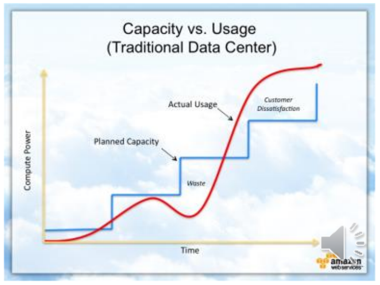
MongoDB has a multi-table document structure, while DynamoDB has a single-table key-value structure. In DynamoDB, the table automatically scales as items are added, columns are created as needed and unique id strings identify every item.
We query the question and answer items for our website using the unique id keys, with answers referencing the question’s key to link the two.
The scripts that manage this database manipulation are Lambda functions, such as TableScan for GETting all the items, findOne for finding the specific answers for a single question, Upsert functions for POST and PUT requests, and DeleteRecord for DELETE functionality.

Slide 23



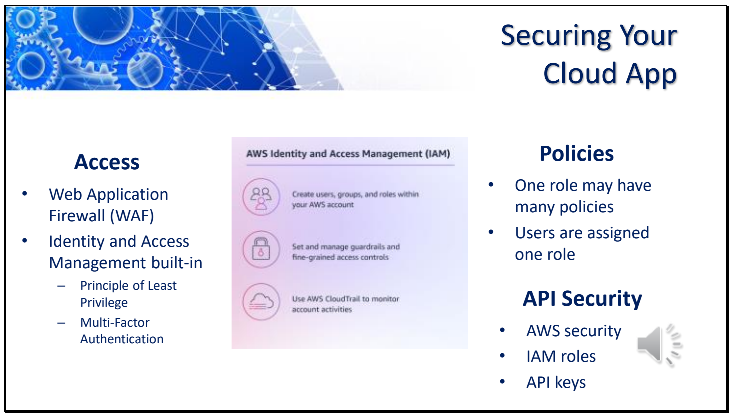
Cloud-Based Development Principles

- Elasticity
- Pay-for-use model



Elasticity describes the need for compute power to adjust in real-time based on user demand. The capacity can be planned ahead of time, and servers can be turned on or off as needs change, but a smaller server may not be able to handle large spikes in user demand. AWS manages huge amounts of server hardware, and can quickly reroute requests to servers with higher capacity to meet demand, then limit that server's production when demand tapers off to reduce waste.
The pay-for-use model is a way of giving the AWS user value, so that a smaller developer only pays for the small amount of power they're using, and a large company with thousands of requests can pay for their usage. There is a cost analysis to be done, but generally the pay-as-you-go is going to provide optimal value for any project that can't afford a server maintenance employee, doesn't have steady demand on their server, or is just starting out and doesn't know what their capacity needs will be.

Slide 24



Securing Your Cloud App

Access

- Web Application Firewall (WAF)
- Identity and Access Management built-in
  - Principle of Least Privilege
  - Multi-Factor Authentication

AWS Identity and Access Management (IAM)

Create users, groups, and roles within your AWS account

Set and manage guardrails and fine-grained access controls

Use AWS CloudTrail to monitor account activities

Policies

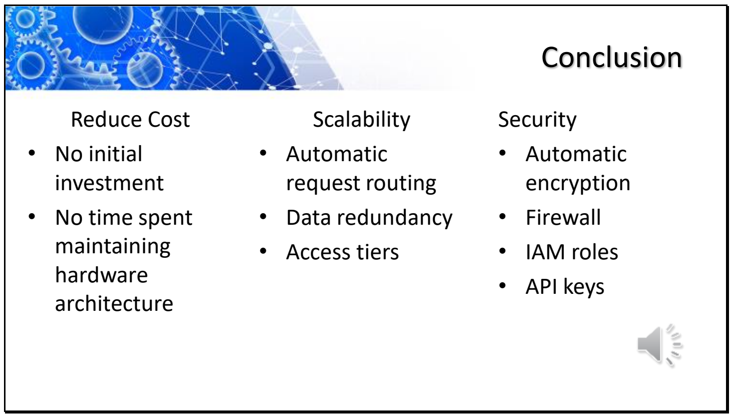
- One role may have many policies
- Users are assigned one role

API Security

- AWS security
- IAM roles
- API keys

AWS automatically sets many security features into their products. AWS Web Application Firewall automatically prevents SQL injection, cross-site scripting, and DDoS attacks. The Identity and Access Management, IAM, features allow the developer to set permissions for individuals or groups, allowing for the principle of least privilege to be easily maintained, and supporting multi-factor authentication to prevent common user-side access abuse like phishing or keylogging.
With IAM allowing us to assign roles to users, those roles let us dictate the security policies that allow users to access the data. Roles and policies are complex, and with the LabRole assigned by AWS we were able to do a lot within the lab environment. Unfortunately, some IAM features were not available so my understanding of the policies and roles is limited.
AWS security has many capabilities. For securing API Gateway and Lambda, AWS offers SSL certificates, the web application firewall, MFA, request throttling, and API key options. AWS expects the user to maintain control over resources using these tools, while they protect the infrastructure. AWS automatically protects S3 and database data with Amazon GuardDuty and Amazon Macie, as well as a wide range of security logging capabilities.

Slide 25



Conclusion

Reduce Cost
- No initial investment
- No time spent maintaining hardware architecture

Scalability
- Automatic request routing
- Data redundancy
- Access tiers

Security
- Automatic encryption
- Firewall
- IAM roles
- API keys

Lifting and shifting a MEAN stack web application into AWS microservices was a quick and easy process. Using the serverless architecture that AWS offers is a way to save time and money, particularly for smaller projects and startups. The initial investment is virtually zero, which is a big jump from the time and money invested with traditional server-required projects. Time can be spent developing the product, rather than maintaining server architecture.
AWS scales automatically, routing requests through their expansive server network as necessary while only charging for what is used. As user demand changes, the backend changes with it. Similarly, AWS stores data in multiple places, allotting more space as more data is added. Data is moved through the tiers of storage to make the more volatile data more easily accessible.
AWS security is robust, with all the industry-standard automatic protections such as data encryption and web application firewall, as well as user-implementable protections such as IAM roles and API keys.