CS 470 Project Two Conference Presentation: Cloud Development

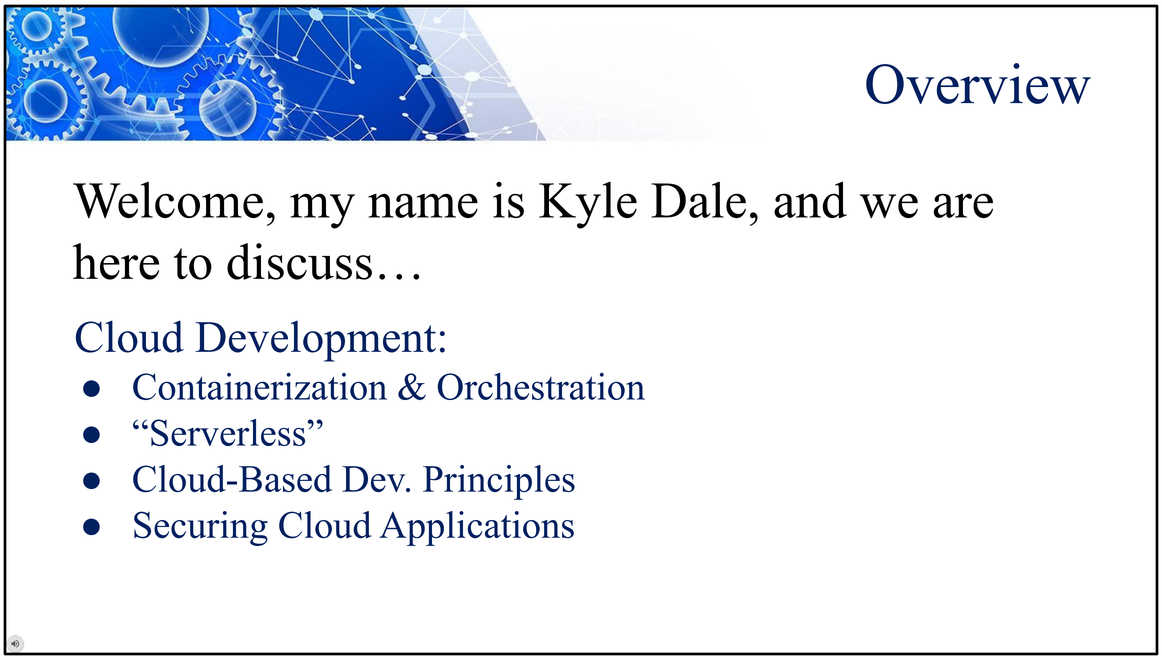
Kyle Dale

April, 2024

Project Two Conference Presentation: Cloud Development

https://youtu.be/XrzVrHMjbS4

Hello, and welcome everyone. Thank you for joining me today...



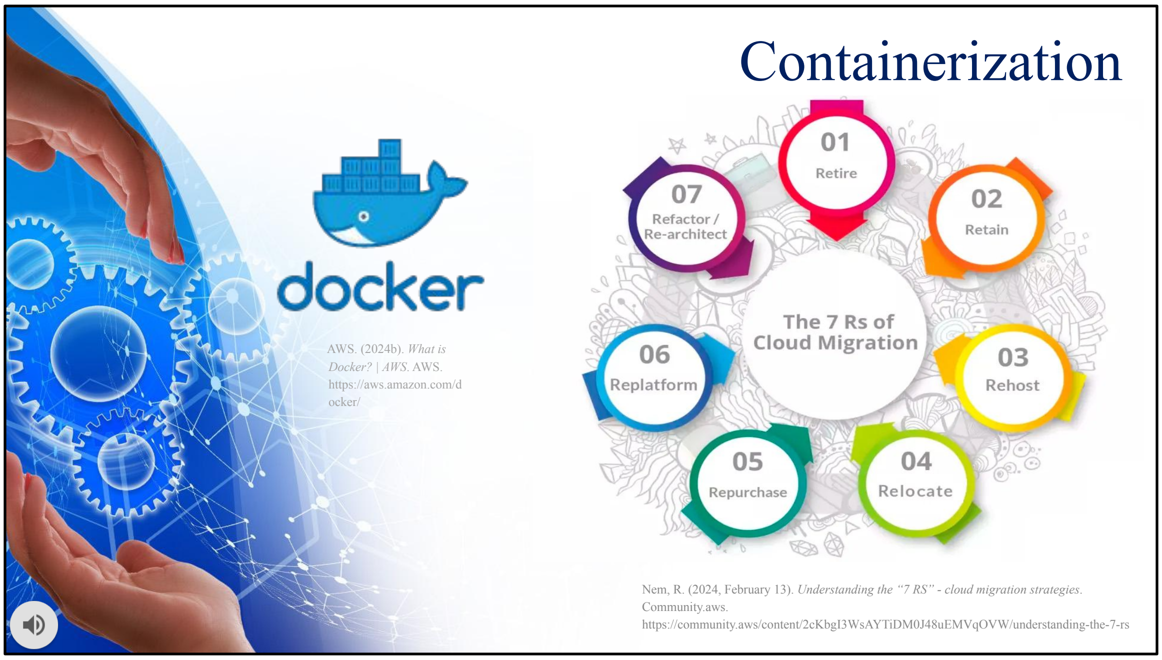
Overview

Welcome, my name is Kyle Dale, and we are here to discuss…

Cloud Development:

- Containerization & Orchestration
- “Serverless”
- Cloud-Based Dev. Principles
- Securing Cloud Applications

My name is Kyle Dale, and I have recently finished successfully migrating a full stack application to a cloud-native web application using AWS and its suite of services. Today, I will be giving a semi-technical presentation on my experience with, and the intricacies involved in cloud-development, particularly when contrasted with more traditional methods of development. Throughout the course of the presentation, we will address topics such as containerization and orchestration, “serverless” systems, cloud based development principles, and securing a cloud application. My hope is that by the end of the presentation, all audience members regardless of technical knowledge can glean useful information for their foray into cloud development.



Containerization

AWS. (2024b). *What is Docker? | AWS*. AWS. https://aws.amazon.com/docker/

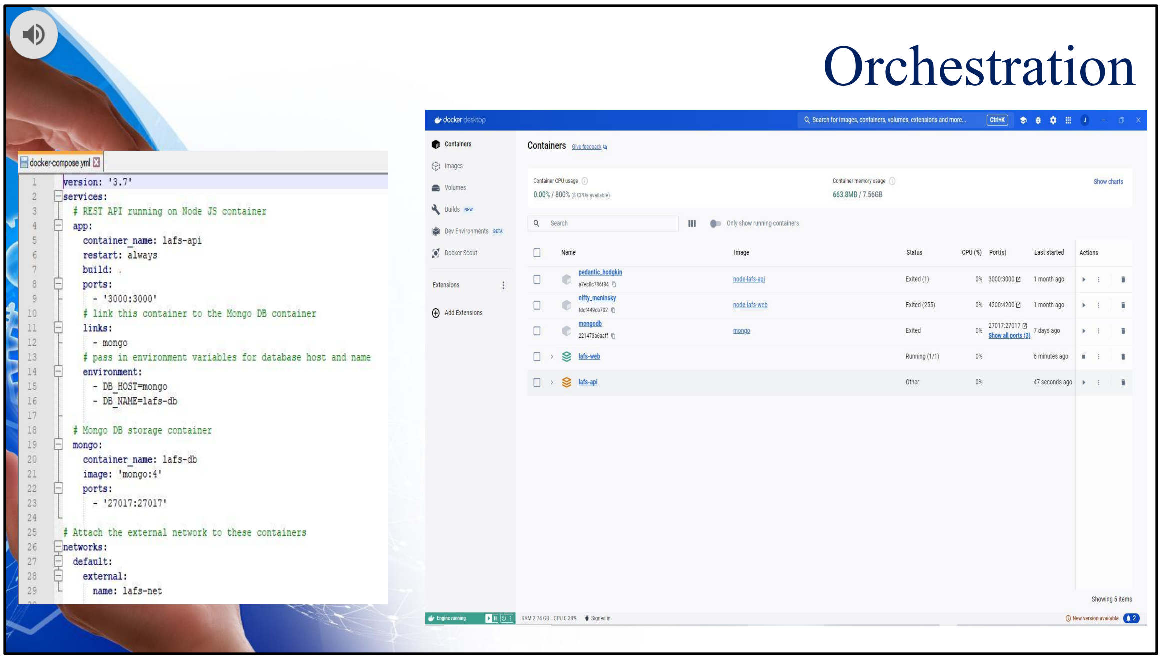
Nem, R. (2024, February 13). *Understanding the "7 RS" - cloud migration strategies*. Community.aws. https://community.aws/content/2cKbgI3WsAYTiDM0J48uEMVqOVW/understanding-the-7-rs

Containerization:

There are many models by which a full stack application can be migrated to the cloud, including the 7 R's or: Retire, Retain, Rehost, Relocate, Repurchase, Replatform, and Refactor (AWS, 2024a). However, we will focus on the two most relevant for my specific use case, Relocate, or transferring a number of servers, comprising one or more applications to a cloud-native platform, and Replatform (or lift, tinker, and shift), which involved migrating an application to the cloud and introducing some level of optimization to reduce cost, or take advantage of cloud capability (AWS, 2024a). The process involved in migrating my Angular application to the cloud involved repackaging the application code into containers via Docker, and facilitating communication between these containers via Docker Compose. Once the application had been containerized it could be built for deployment and uploaded to an AWS S3

Bucket, with subsequent functionality being built in the cloud-native AWS environment. This process will become more clear throughout the presentation.

At the start of the process was containerization. Containerization is a method of packing, distributing, and running applications in isolated environments known as containers, and is accomplished through the use of a program like Docker. Containerization offers numerous advantages in modern software development, including ensuring portability by encapsulating an application along with all its dependencies and libraries into a single unit. This allows containers to run consistently across different environments, eliminating compatibility issues and ensuring a seamless deployment process. This encapsulation ensures each container operates independently of its host system and other containers, preventing conflicts and ensuring that changes or issues within one container do not affect others, enhancing security and stability. Moreover, containerization offers efficient resource usage by leveraging the shared kernel of the host operating system. Unlike traditional virtualization methods, which require separate operating systems for each virtual machine, containers share the underlying resources, resulting in lower overhead and higher resource utilization. This enables rapid scalability to handle fluctuating workloads.



Orchestration

docker-compose.yml

```yaml
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:
  default:
    external:
      name: lafs-net
```

Containers

Container CPU usage 0.00% / 800% (8 CPUs available)

Container memory usage 663.8MB / 7.56GB

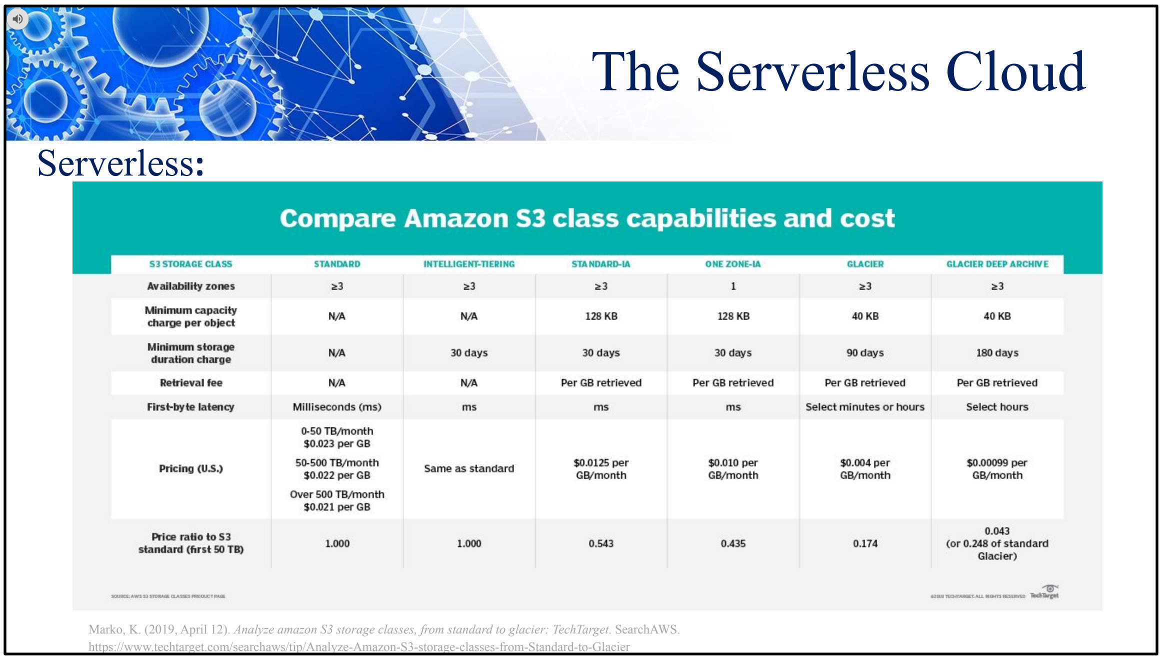
Name	Image	Status	CPU (%)	Port(s)	Last started
pedantic_hodgkin a7ec8c786f84	node-lafs-api	Exited (1)	0%	3000:3000	1 month ago
nifty_meninsky fdcf449cb702	node-lafs-web	Exited (255)	0%	4200:4200	1 month ago
mongodb 221473a6aaff	mongo	Exited	0%	27017:27017 Show all ports (3)	7 days ago
lafs-web		Running (1/1)	0%		6 minutes ago
lafs-api		Other	0%		47 seconds ago

Orchestration:

When considering why to use Docker Compose, it is important to gain an understanding of what Docker Compose is. Basically, Docker Compose is a tool that allows one to define and run multi-container Docker applications. Docker Compose provides a way to define complex applications composed of multiple services, each running in its own container, and allows the entire application stack to be managed from a single configuration file. This principles is demonstrated in the above image where we can see the containers for lafs-web, lafs-api, and the DB respectively, as well as the YAML configuration file for lafs-api. Docker Compose optimizes resource utilization by allowing one to define resource constraints and limits for specific containers. Done properly, this will aid in efficient resource allocation and ensure one container does not monopolize resources.

Concerning orchestration, Docker Compose simplifies the orchestration of multiple containers by providing commands that manage the lifecycle of the entire application stack, including starting, stopping, and restarting specific containers (IBM, 2019).

Lastly, Docker Compose is both scalable and flexible. As touched upon previously, the system is easily scaled horizontally to aid in increasing resources or aid with fault protection. Docker Compose also supports the scaling of services on demand (IBM, 2019). This key ability will allow the application to handle traffic spikes without downtime. Docker Compose provides many advantages and QoL tools that will not only increase the efficiency of development, but allow for a more robust and compatible application to be created and managed.



The Serverless Cloud

Serverless:

Compare Amazon S3 class capabilities and cost

S3 STORAGE CLASS	STANDARD	INTELLIGENT-TIERING	STANDARD-IA	ONE ZONE-IA	GLACIER	GLACIER DEEP ARCHIVE
Availability zones	≥3	≥3	≥3	1	≥3	≥3
Minimum capacity charge per object	N/A	N/A	128 KB	128 KB	40 KB	40 KB
Minimum storage duration charge	N/A	30 days	30 days	30 days	90 days	180 days
Retrieval fee	N/A	N/A	Per GB retrieved	Per GB retrieved	Per GB retrieved	Per GB retrieved
First-byte latency	Milliseconds (ms)	ms	ms	ms	Select minutes or hours	Select hours
Pricing (U.S.)	0-50 TB/month $0.023 per GB 50-500 TB/month $0.022 per GB Over 500 TB/month $0.021 per GB	Same as standard	$0.0125 per GB/month	$0.010 per GB/month	$0.004 per GB/month	$0.00099 per GB/month
Price ratio to S3 standard (first 50 TB)	1.000	1.000	0.543	0.435	0.174	0.043 (or 0.248 of standard Glacier)

SOURCE: AWS S3 STORAGE CLASSES PRODUCT PAGE

©2019 TECHTARGET. ALL RIGHTS RESERVED TechTarget

Marko, K. (2019, April 12). *Analyze amazon S3 storage classes, from standard to glacier: TechTarget*. SearchAWS. https://www.techtarget.com/searchaws/tip/Analyze-Amazon-S3-storage-classes-from-Standard-to-Glacier

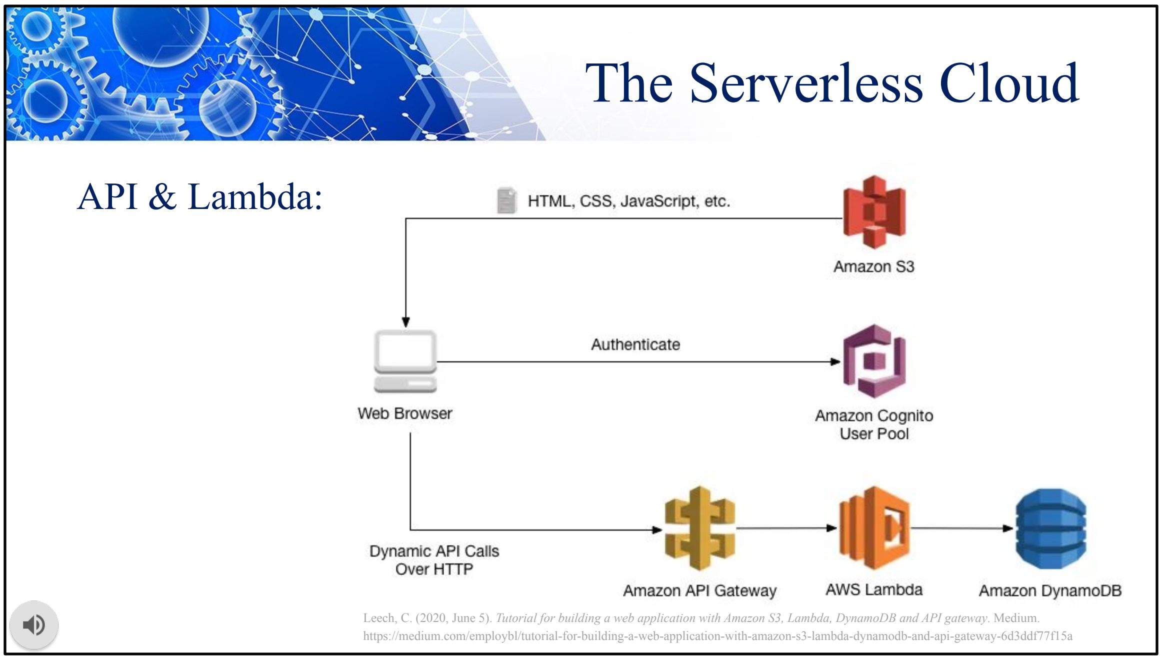
The Serverless Cloud:

"Serverless" refers to a cloud computing model where the cloud provider dynamically manages the allocation of machine resources. In the case of many of the AWS services, this means the 'under the hood' configuration and maintenance is abstracted away and handled by AWS.

AWS S3 is a cloud storage service that allows data storage and retrieval from anywhere on the web and is designed to be accessible and scalable. S3 offers multiple types of storage depending on need, as seen in the chart above. All versions of S3 scale automatically to accommodate growing or shrinking data requirements (Amazon Web Services, 2015). This is beneficial in situations where demand may be variable, and one is unsure of how much hardware to actually acquire. S3 and cloud-based storage mitigate this issue by providing on-demand and pay-on-demand resources.

Additionally, S3 is designed for 11-9's durability. Said plainly, even with one billion objects, you would be unlikely to lose a single one in a hundred years (Google, 2021). Cloud-based solutions like S3 can be more affordable than local disk options, as S3 operates on a pay-as-you-use model with no upfront infrastructure costs, in some situations, this can be more affordable.

Lastly, S3 offers easy to use and robust RBAC mechanisms, allowing for permissions to be set at the bucket level (Simplilearn, 2017). S3 supports encryption for data at flight and at rest. Although these capabilities can be accomplished with local disk solutions, configuration is often more complicated and error-prone.



The Serverless Cloud

API & Lambda:

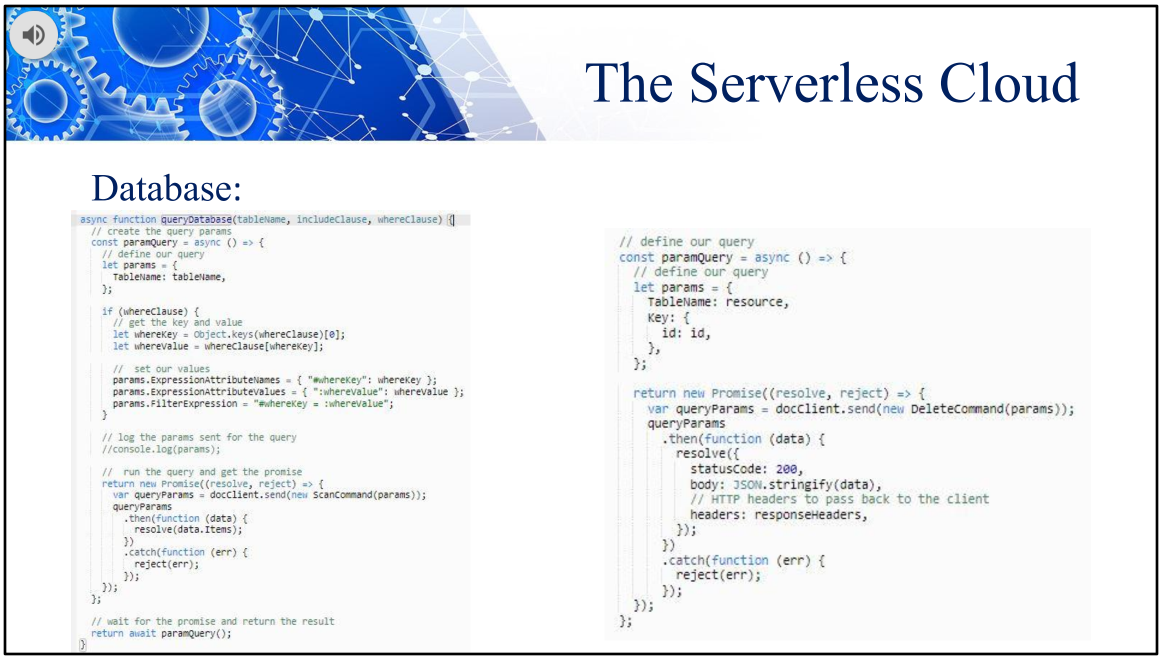
Leech, C. (2020, June 5). *Tutorial for building a web application with Amazon S3, Lambda, DynamoDB and API gateway*. Medium. https://medium.com/employbl/tutorial-for-building-a-web-application-with-amazon-s3-lambda-dynamodb-and-api-gateway-6d3ddf77f15a

Many of the advantages inherent in API Gateway can be seen in Lambda as well, as the two work closely together to provide application functionality within a serverless environment. API Gateway is a fully managed service that allows developers to publish, maintain, monitor, and secure APIs at any scale, while Lambda is a serverless computer service that allows one to run code in response to events without provisioning or managing servers (Amazon Web Services, 2015; AWS, 2024b).

With API Gateway specifically the main advantages lie in efficiency, scalability, security, and capability of the API to work in conjunction with other AWS services. API Gateway allows users to run multiple versions of the same API simultaneously, allowing for the quick refactoring and testing of new versions (AWS, 2024c). Additionally, API Gateway takes advantage of AWS's global network of edge locations and uses CloudFront to throttle traffic and authorize API calls seamlessly and efficiently (AWS, 2024c). This ability allows Amazon API Gateway to gracefully withstand traffic spikes and ensure back-end systems are not unnecessarily called. Like with many AWS services, you only pay for the traffic you utilize, further increasing scalability and reducing needless expenditure on maintenance and upkeep. The next major benefits to Amazon API Gateway is the integration it has with CloudWatch, IAM, and Amazon Cognito. These integrations allow users to monitor performance and metric information on API calls, data latency, error rates, and the ability to set alarms, all from the API dashboard via CloudWatch (AWS, 2024b). Finally, API Gateways primary ability is to route incoming API requests to the appropriate backend services or Lambda functions (AWS, 2024b). With Amazon API Gateway, the user can define RESTful APIs and configure different endpoints to handle various resource paths and HTTP methods (AWS, 2024b; AWS, 2024c). We can see this concept demonstrated in the diagram displayed.

The primary characteristics of Lambda include the serverless state, event-driven architecture, stateless execution, automatic scaling, integrated logging and monitoring flexibility of language choice, and pay-per-use billing. Event-driven architecture means that Lambda functions are triggered by various events, Lambda supports a wide range of event sources, enabling complex development easily (Amazon Web Services, 2015). The Lambda functions are typically written to handle HTTP requests or responses, and API gateway can be configured to route incoming HTTP requests to specific Lamanda functions, allowing for the CRUD functionality seen on the Angular application.

Now that we understand API Gateway and Lambda we can go over the basic steps followed to integrate the front-end with the back-end of the Angular application. The process began with defining API endpoints, including specifying HTTP methods and request/response formats, then implementing the front-end and back-end logic within AWS, and finally testing integration. The overall structure of the application is summarized well through the diagram seen above. By the completion of the migration I had created Lambda functions responsible for finding one question in a table, upserting a question or answer, deleting a record, scanning the table or DB, and getting a single record from the DB. These functions were invoked in response to specified events, and interacted with the DB, as illustrated in the image, to support the sites functionality.



The Serverless Cloud

Database:

```
async function queryDatabase(tableName, includeClause, whereClause) {
  // create the query params
  const paramQuery = async () => {
    // define our query
    let params = {
      TableName: tableName,
    };

    if (whereClause) {
      // get the key and value
      let whereKey = Object.keys(whereClause)[0];
      let whereValue = whereClause[whereKey];

      //  set our values
      params.ExpressionAttributeNames = { "#whereKey": whereKey };
      params.ExpressionAttributeValues = { ":whereValue": whereValue };
      params.FilterExpression = "#whereKey = :whereValue";
    }

    // log the params sent for the query
    //console.log(params);

    //  run the query and get the promise
    return new Promise((resolve, reject) => {
      var queryParams = docClient.send(new ScanCommand(params));
      queryParams
        .then(function (data) {
          resolve(data.Items);
        })
        .catch(function (err) {
          reject(err);
        });
    });
  };

  // wait for the promise and return the result
  return await paramQuery();
}
```

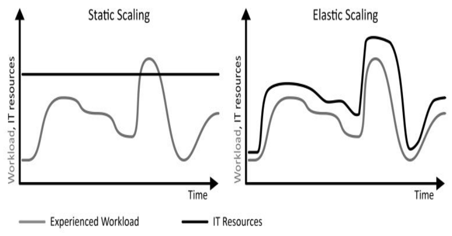
```
// define our query
const paramQuery = async () => {
  // define our query
  let params = {
    TableName: resource,
    Key: {
      id: id,
    },
  };

  return new Promise((resolve, reject) => {
    var queryParams = docClient.send(new DeleteCommand(params));
    queryParams
      .then(function (data) {
        resolve({
          statusCode: 200,
          body: JSON.stringify(data),
          // HTTP headers to pass back to the client
          headers: responseHeaders,
        });
      })
      .catch(function (err) {
        reject(err);
      });
  });
};
```

The major differences in MongoDB and DynamoDB lie in accessibility, manageability, contrasting data models, and recoverability. Dynamo is limited to configurement through AWS, while Mongo is platform-agnostic and can be deployed from anywhere (MongoDB, 2024). However, Dynamo is a fully managed DB that allows users to quickly configure the database, as AWS will manage auto scaling, availability, and updating, by contrast, Mongo requires the user to manage the infrastructure and configuration personally (Wickramasinghe, 2021). MongoDB utilizes JSON-based document storage of up to 16MB document size and the additional capability of advanced BSON types (int, long, date, timestamp, etc) (MongoDB, 2024)). By contrast, Dynamo is limited to key-value stores with JSON support, limited data type (number, string, and binary only), and a 400KB single item limit on size. Mongo offers users more flexibility in querying data, as it supports single keys, ranges, graph traversals, JOINs, and more (Wickramasinghe, 2021). By contrast, Dynamo only has the capability to carry out complex aggregation using other AWS services.

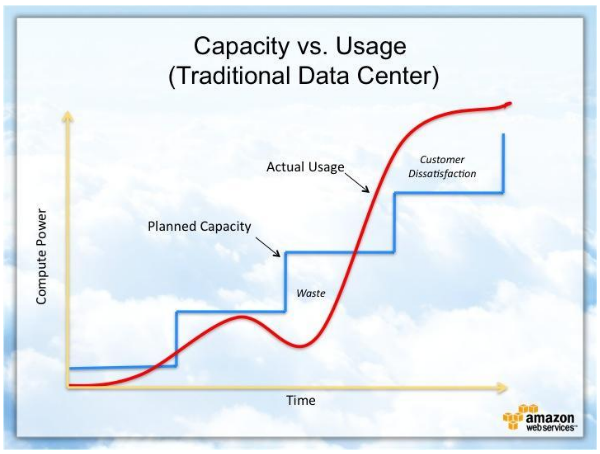
Lastly, when it comes to security, backup, and recovery, it is extremely difficult to beat the out-of-the-box functionality and ease of configuration present with AWS platforms. Dynamo comes with baked-in security best practices through AWS, and flexible RBAC through IAM (Wickramasinghe, 2021). Additionally, Dynamo offers multi-region data replication baked-in as part of the AWS service, supporting both on-demand and automated backups with point-in-time recovery (Wickramasinghe, 2021). By contrast, with Mongo, users are responsible for most all of the security and recovery architecture.

To facilitate the function of the application, I utilized a variety of queries within my Lambda functions mentioned earlier. These queries interacted with the DB to facilitate scanning the table, finding specific records, deleting records, upserting questions or answers, and performing other operations as needed. An example of two queries can be seen in the provided code, for FindOneQuestion and DeleteRecord respectively. FindOneQuestion performs a query against the DB to find records matching the specified criteria, while DeleteRecord deletes a record from the DB based on the provided id.

Cloud-Based Development Principles



Skylines Academy. (2024, February 15). *AZ-900: Cloud concepts - scalability and elasticity - Skylines Academy.* Refactored - Top Rated Cloud Training. https://www.skylinesacademy.com/blog/2020/3/6/az-900-cloud-concepts-scalability-and-elasticity



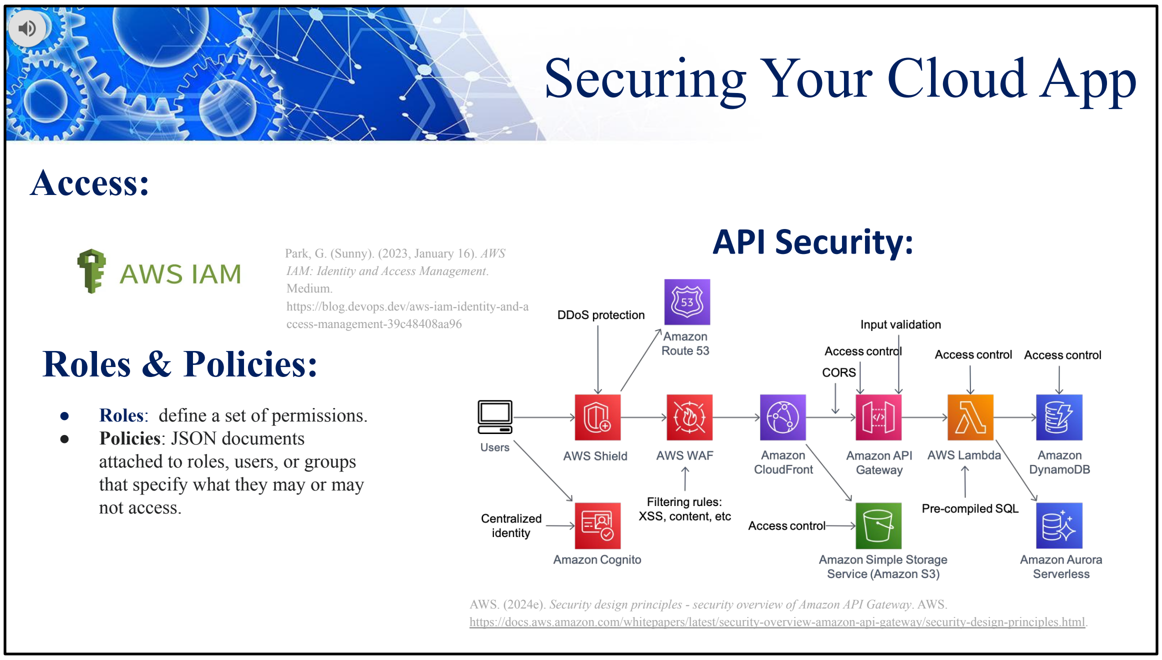
Cloud Based Development Principles:

Elasticity refers to the ability to acquire resources as you need them and release resources when you no longer need them. Many AWS services do this automatically as part of the service, and this capability is essential for scalability, allowing systems to handle variable workloads efficiently (AWS, 2024d). For example, AWS offers services like S3, SQS, SNS, and SES, which automatically adjust resource capacity based on demand. (AWS, 2024). Additional AWS services like Dynamo, EC2, ECS and more, utilize auto-scaling. AWS auto scaling monitors applications and automatically adjusts capacity to maintain steady and predictable performance at the most efficient cost possible. A graph of this efficient resource usage can be seen in the left image.

One of the major benefits of AWS is its pay-for-use model, where users only pay for the resources and services they actively use. This model eliminates the need for upfront investment in physical hardware and ensures cost efficiency by charging users based on their actual usage.

Users are not responsible for wasted downtime or underutilized resources, resulting in significant cost savings.

The elasticity inherent in AWS, combined with the pay-for-use model, forms an incredibly efficient method of hosting and deploying applications. Users have the flexibility to scale resources up or down based on demand, ensuring optimal performance and cost efficiency. By only paying for what they use, users can optimize their resource allocation and minimize unnecessary expenses, unlike traditional setups where upfront investments are required regardless of actual usage.



Securing Your Cloud App

Access:

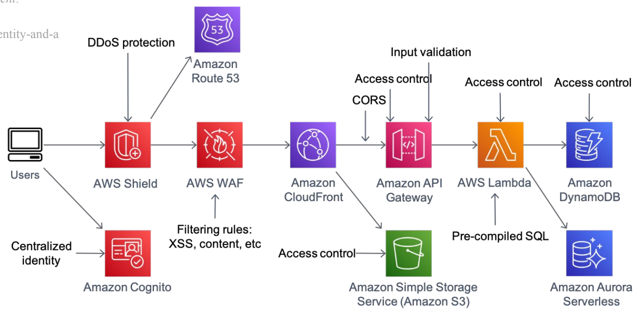
AWS IAM

Park, G. (Sunny). (2023, January 16). *AWS IAM: Identity and Access Management*. Medium. https://blog.devops.dev/aws-iam-identity-and-access-management-39c48408aa96

Roles & Policies:

- **Roles**: define a set of permissions.
- **Policies**: JSON documents attached to roles, users, or groups that specify what they may or may not access.

API Security:



AWS. (2024e). *Security design principles - security overview of Amazon API Gateway*. AWS. https://docs.aws.amazon.com/whitepapers/latest/security-overview-amazon-api-gateway/security-design-principles.html

Securing Your Cloud App:

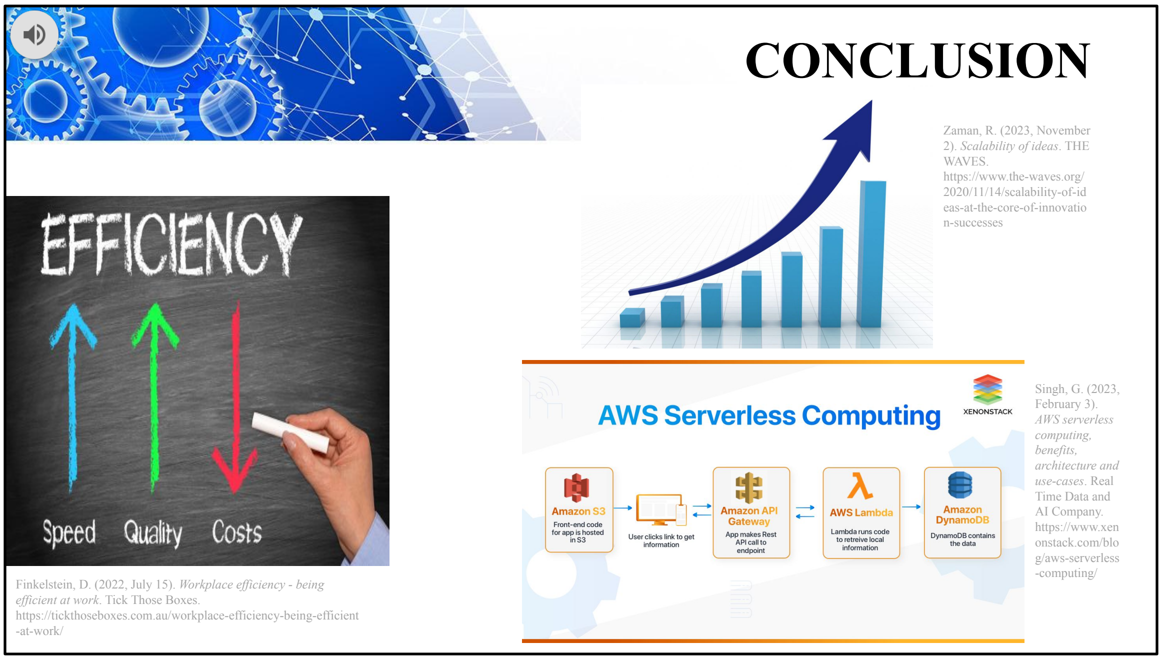
AWS Provides many services and tools to secure ones cloud application and implement comprehensive defensive in depth. In terms of access, to preface, it is important to remember to adhere to best security principles like triple-A and least privilege when dictating roles and policies. IAM provides users with a few ways to ensure RBAC within each AWS service. When creating a role and determining policies, users are provided with either a visual or JSON format to edit and configure the role's permissions. Role permissions are defined by each service, such as Dynamo or Lambda, ensuring that privileges are granular and specific to the needs of that service.

IAM roles and policies were utilized to establish CRUD capability across our application. Roles define a set of permissions that grant access to AWS resources, while policies are JSON documents attached to roles, users, or groups that specify what they may or may not access. Specifically, we required a role that allowed six actions within the

Dynamo service: GetItem, Query, and Scan actions from the ‘Read’ section, and DeleteItem, PutItem, and UpdateItem from the ‘Write’ section. This provided access to the necessary services to create or delete questions or answers from the respective tables.

To secure the connection of the API, and specifically, Lambda to API Gateway and Lambda to the S3 bucket, there are a few methods one can employ within AWS. The first step is ensuring RBAC and controlling access to the resources properly, as discussed above. Additionally logging and monitoring should be implemented during RBAC implementation for proper application monitoring. The next step to take would be to utilize the encryption capabilities provides by most AWS services, including S3 and Lambda. Additionally AWS offers flexibility key management options for encryption, with the option to oversee keys personally or allow AWS to manage the keys (AWS, 2024e). Finally, AWS offers encrypted message queues for the transmission of sensitive data using server side encryption (AWS, 2024e). An illustration of how the various AWS services interact to secure the API can be seen in the image under API Security.

Ultimately the ease of configuration and depth of layers provided by AWS inherent security allows for safer, smoother, and more efficient development of an application where pipeline efficiency is increased while incorporating security by design.



CONCLUSION

EFFICIENCY
Speed
Quality
Costs
AWS Serverless Computing
XENONSTACK
Amazon S3
Front-end code for app is hosted in S3
User clicks link to get information
Amazon API Gateway
App makes Rest API call to endpoint
AWS Lambda
Lambda runs code to retreive local information
Amazon DynamoDB
DynamoDB contains the data

Zaman, R. (2023, November 2). *Scalability of ideas*. THE WAVES. https://www.the-waves.org/2020/11/14/scalability-of-ideas-at-the-core-of-innovation-successes

Singh, G. (2023, February 3). *AWS serverless computing, benefits, architecture and use-cases*. Real Time Data and AI Company. https://www.xenonstack.com/blog/aws-serverless-computing/

Finkelstein, D. (2022, July 15). *Workplace efficiency - being efficient at work*. Tick Those Boxes. https://tickthoseboxes.com.au/workplace-efficiency-being-efficient-at-work/

As we have explored throughout the presentation, there are a multitude of advantages that come with cloud-based applications and developments that make it a powerful option if relevant to one's needs. Ultimately three overarching points that can be gleaned about cloud-based development from this presentation are the increase in efficiency, scalability of the system, and the implications of a “serverless” state.

The efficiency gains from migrating to a cloud based platform like AWS are not limited to workflow alone. The pay-per-use cost structure discussed previously makes cloud-computing a much more cost-efficient option for some development needs. Additionally the lack of management and configuration present in AWS systems allows for developers to spend more time refining and improving upon the application, rather than configuration and set-up.

Although scalability is part of efficiency, the inherent scalability seen in AWS services is one of the primary factors contributing to usage.

The focus on ease of scalability and elasticity are so thoroughly ingrained in the AWS products it is not limited to a single service like Lambda, but is a hallmark of them all.

Finally, the “serverless” state of cloud-based development and platforms like AWS provide numerous advantageous over traditional systems. This ability, along with the lack of maintenance mentioned earlier, allows developers to maintain as much up-time as possible, with the capability to backup programs across regions and ensure minimal loss of data.

Cloud-based development represents a revolutionary capability, while it will not completely replace traditional development, it is seeing increased use-case and popularity across many sectors. Leveraging the powerful capability while decreasing traditional downsides has made cloud-based development an important consideration before beginning any development scenario.

Thank you all for your time and please take a moment to review the following reference slides…

References

Amazon Web Services. (2015, May 19). *Introduction to Aws Lambda - Serverless Compute on Amazon Web Services*. YouTube. https://www.youtube.com/watch?v=eOBq__h4OJ4&t=2s.

Amazon Web Services. (2015, September 17). *Introduction to amazon simple storage service (S3) - cloud storage on AWS*. YouTube. https://www.youtube.com/watch?v=77lMCiiMilo.

AWS. (2024a). *About the migration strategies - AWS prescriptive guidance*. AWS. https://docs.aws.amazon.com/prescriptive-guidance/latest/large-migration-guide/migration-strategies.html.

AWS. (2024b). *Amazon API gateway - AWS documentation*. AWS. https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html.

AWS. (2024c). *API management - amazon API gateway - AWS*. AWS. https://aws.amazon.com/api-gateway/.

AWS. (2024d). *Elastic - reactive systems on AWS*. AWS. https://docs.aws.amazon.com/whitepapers/latest/reactive-systems-on-aws/elastic.html

AWS. (2024e). *Security design principles - security overview of Amazon API Gateway*. AWS. https://docs.aws.amazon.com/whitepapers/latest/security-overview-amazon-api-gateway/security-design-principles.html

References Continued…

Docker. (2024, February 9). *Docker service scale*. Docker Documentation. https://docs.docker.com/reference/cli/docker/service/scale/.

Google. (2021, January 28). *Understanding cloud storage 11 9s durability target | google cloud blog*. Google. https://cloud.google.com/blog/products/storage-data-transfer/understanding-cloud-storage-11-9s-durability-target.

IBM. (2019, April 17). *Container Orchestration explained*. YouTube. https://www.youtube.com/watch?v=kBF6Bvth0zw.

MongoDB. (2024). Comparing dynamodb and mongodb. MongoDB. https://www.mongodb.com/compare/mongodb-dynamodb.

Schachte, R. (2019, February 10). *Introduction to Docker and Docker containers*. YouTube. https://www.youtube.com/watch?v=JSLpG_spOBM.

Simplilearn. (2017, July 14). *AWS S3 tutorial for Beginners | Amazon S3 Tutorial | Amazon Simple Storage Service | simplilearn*. YouTube. https://www.youtube.com/watch?v=LfBn5Y1X0vE.

Wickramasinghe, S. (2021, July 14). *MongoDB VS dynamodb: Comparing nosql databases*. BMC Blogs. https://www.bmc.com/blogs/mongodb-vs-dynamodb/.

Image References

AWS. (2024e). *Security design principles - security overview of Amazon API Gateway*. AWS. https://docs.aws.amazon.com/whitepapers/latest/security-overview-amazon-api-gateway/security-design-principles.html.

AWS. (2024f). *What is Docker? | AWS*. AWS. https://aws.amazon.com/docker/

Finkelstein, D. (2022, July 15). *Workplace efficiency - being efficient at work*. Tick Those Boxes. https://tickthoseboxes.com.au/workplace-efficiency-being-efficient-at-work/.

Leech, C. (2020, June 5). *Tutorial for building a web application with Amazon S3, Lambda, DynamoDB and API gateway*. Medium. https://medium.com/employbl/tutorial-for-building-a-web-application-with-amazon-s3-lambda-dynamodb-and-api-gateway-6d3ddf77f15a.

Marko, K. (2019, April 12). *Analyze amazon S3 storage classes, from standard to glacier: TechTarget*. SearchAWS. https://www.techtarget.com/searchaws/tip/Analyze-Amazon-S3-storage-classes-from-Standard-to-Glacier.

Image References Continued…

Nem, R. (2024, February 13). *Understanding the "7 RS" - cloud migration strategies*.

Community.aws.https://community.aws/content/2cKbgI3WsAYTiDM0J48uEMVqOVW/understanding-the-7-rs.

Park, G. (Sunny). (2023, January 16). *AWS IAM: Identity and Access Management*. Medium.

https://blog.devops.dev/aws-iam-identity-and-access-management-39c48408aa96.

Singh, G. (2023, February 3). *AWS serverless computing, benefits, architecture and use-cases*.
Real Time Data and AI Company. https://www.xenonstack.com/blog/aws-serverless-computing/.

Skylines Academy. (2024, February 15). *AZ-900: Cloud concepts - scalability and elasticity -*

Skylines Academy. Refactored - Top Rated Cloud Training.
https://www.skylinesacademy.com/blog/2020/3/6/az-900-cloud-concepts-scalability-and-elasticy.

Zaman, R. (2023, November 2). *Scalability of ideas*. THE WAVES.
https://www.the-waves.org/2020/11/14/scalability-of-ideas-at-the-core-of-innovation-successes/.