CS-470 Project Two Presentation: Cloud Development
Tyten Perez
October 12, 2024

Presentation Link: https://www.youtube.com/watch?v=vyNykgYThfk

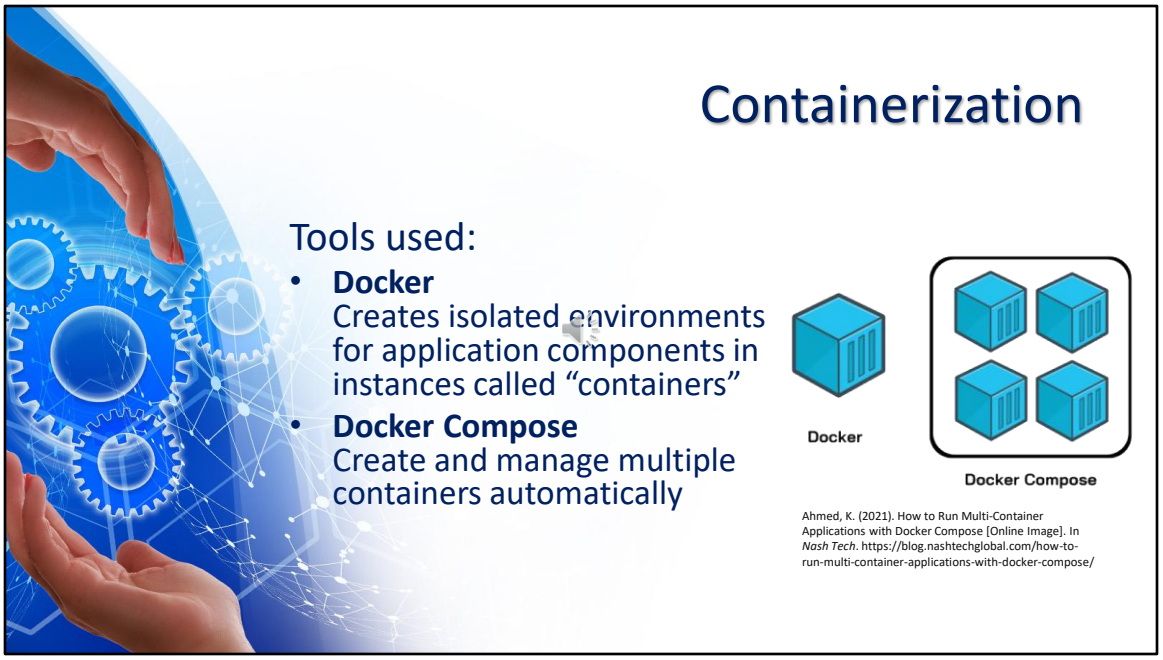
Overview

Reflecting on our AWS migration

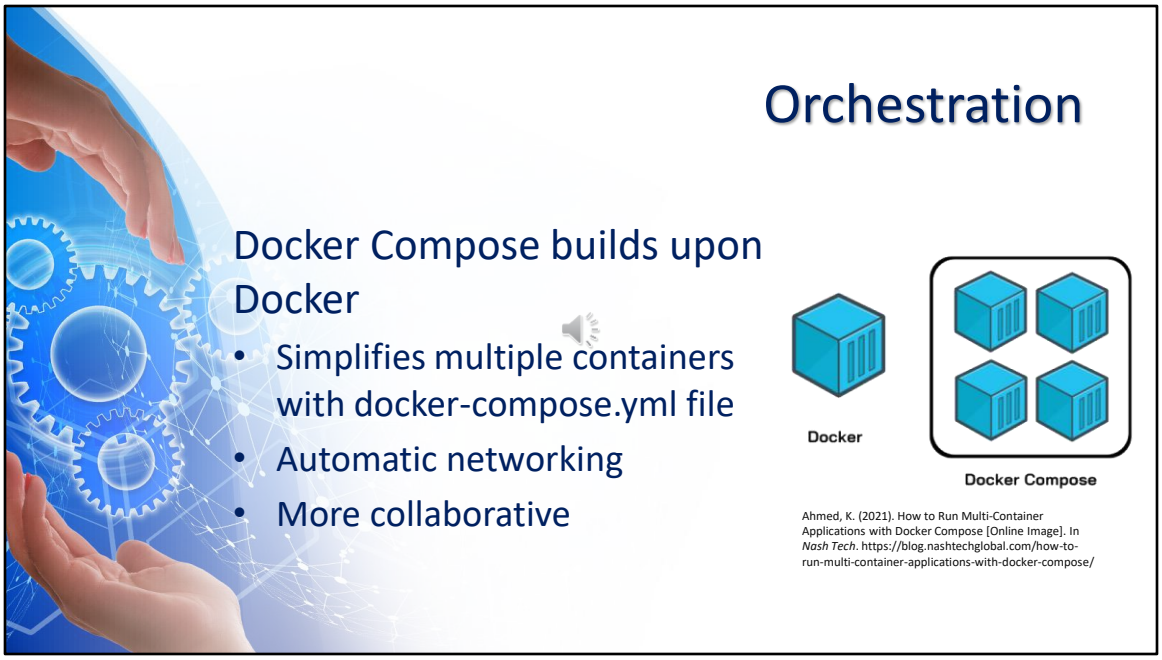
- Containerization
- Serverless Cloud
- Cloud Development Principles

Migration to the AWS Cloud: Plan, Skills, Costs & Software. (n.d.). Www.scnsoft.com. Retrieved October 8, 2024, from https://www.scnsoft.com/aws/migration

Hello everyone, my name is Tyten Perez, and I am here today to explain the company's transition towards Amazon Web Services. I will go over the microservices used to accomplish this and the benefits of doing so. As a QNA developer, I was directly involved in this development, which involved containers, the serverless cloud, and core cloud principles.



To assist in our development, we used containers. Containerization helps us put our application and its dependencies into light-weight packages, ensuring they runs the same everywhere. Using Docker, we contained multiple components of our application like the front-end, back-end, and database. To make this process more efficient, Docker Compose was also used to automatically recreate these containers, allowing them to interact with each-other with ease. This setup made it easy to manage all parts of our application during development and provided a consistent environment. As a result, we avoided issues between our developers which could have caused delays.



Let's explore the value of using Docker Compose. Docker Compose aims to solve the weaknesses of Docker, making container management more convenient and efficient. While Docker on its own allows us to create and run individual containers, handling multiple containers can become complex and time-consuming.

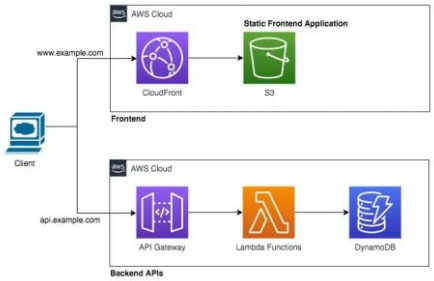
- Docker Compose addresses this issue by simplifying operations involving multiple containers. It does this through the use a simple docker-compose.yml file that allows us to define all the containers our application needs in an easy-to-read format. Instead of running lengthy commands for each container, we can start all of them simultaneously with a single command.
- In our case, this also helped with managing the networking and dependencies between containers automatically. It handles the communication between different services, so we don't have to configure complex networking settings ourselves.
- Docker compose also improves our ability to collaborate together as we can more reliably share configurations with our team members.

As a result, setup time is reduced, and developer environments are more consistent.

The Serverless Cloud

What is “serverless” and what are its advantages?

- More affordable business model
- Enhanced scalability
- Allows us to focus on code



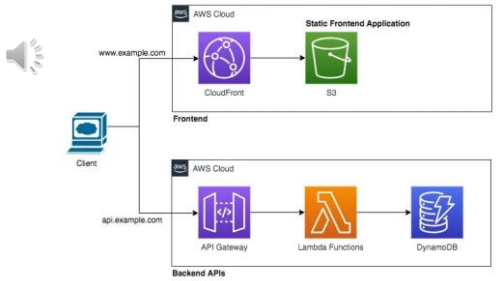
Waswani, N. (2020, November 18). Serverless Architecture Patterns in AWS. Medium. https://waswani.medium.com/serverless-architecture-patterns-in-aws-edeab0e46a32

Serverless allows us to run our applications without managing the servers ourselves at a physical location. This means no on-site setup or presence needed to configure and maintain servers as they are handled by the cloud service.
- This has several other benefits as well, firstly, its more cost-effective for our business as we are not required to invest in server equipment. This also means we don’t have to worry about the cost of running idle servers that may not be used. With the serverless model, we only pay when our code is executed on the service, giving us more predictable expenses.
- Serverless is also more scalable, as it handles scaling for us. It automatically adjusts resources needed, preventing traffic congestion during heavy user traffic.
- Switching to serverless also lets us focus on our application. This gives us more time to focus on development goals rather than worrying about underlying infrastructure issues.

The Serverless Cloud

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

- Accessible anywhere
- Increased durability
- Scalable data storage



Waswani, N. (2020, November 18). Serverless Architecture Patterns in AWS. Medium. https://waswani.medium.com/serverless-architecture-patterns-in-aws-edeab0e46a32

S3, which stands for Simple Storage Service, is a cloud storage solution provided by Amazon Web Services. With it, we can store and retrieve data from anywhere on the internet at any time. Unlike local storage, which relies on physical drives to be managed, S3 is a more convenient, durable and scalable way to store data.

- Firstly, it being accessible anywhere means that it is easy for both developers and users to access. By switching to S3, our application became accessible to users globally without the need for a traditional data server.
- Secondly, the increased durability makes this option more reliable than traditional local storage. This is because it duplicates and creates backups of our data across multiple data servers. This helps protect our database and application files from unrecoverable data loss.
- Thirdly, this option is more scalable as S3 offers unlimited data with a pay per gigabyte business model. Similar to transitioning to serverless, S3 reduces the need for local infrastructure setup and presence.

The Serverless Cloud

API & Lambda

Lambda API Logic:

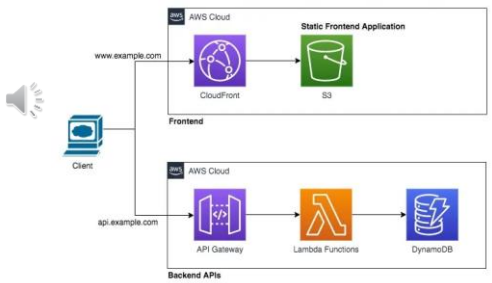
- Event-driven functions
- Interacts with AWS API Gateway

Scripts Produced:

- Lambda function code
- API Gateway configuration

Integrating Frontend with Backend:

- Deployed on AWS S3
- Configured API endpoints in Angular app
- Enabled CORS in API Gateway and Lambdas



Waswani, N. (2020, November 18). Serverless Architecture Patterns in AWS. Medium. https://waswani.medium.com/serverless-architecture-patterns-in-aws-edeab0e46a32

Now let's explore how we applied serverless API in our application.
Using AWS API architecture allowed us to focus on our applications functions.
Lambdas were used to execute these functions through requests from the front-end. This is done through the API Gateway, which acts as an interface to route requests to the appropriate Lambda functions.
Scripts were created for each Lambda function, containing the logic for handling data operations. These operations included creating, reading, updating, or deleting data in our database. This required setting up our API gateway resources and methods to allow these requests.
Connecting our Angular frontend with the backend required several steps.

- First, by hosting the angular application on S3
- Then by configuring endpoints in API gateway to handle frontend calls
- Lastly, Cross-Origin resource sharing was then enabled in both the API Gateway and Lambda function responses, allowing for the frontend to be hosted on a different domain.

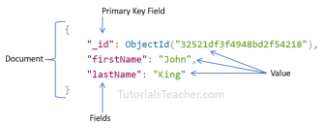
The Serverless Cloud

Database

Data-model differences between MongoDB and DynamoDB

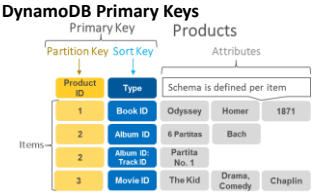
- MongoDB: Flexible schema, unstructured.
- DynamoDB: AWS's NoSQL database service, uses key-value structure.

MongoDB Documents



MongoDB Documents: Document, Array, Embedded Document, n.d.

DynamoDB Primary Keys



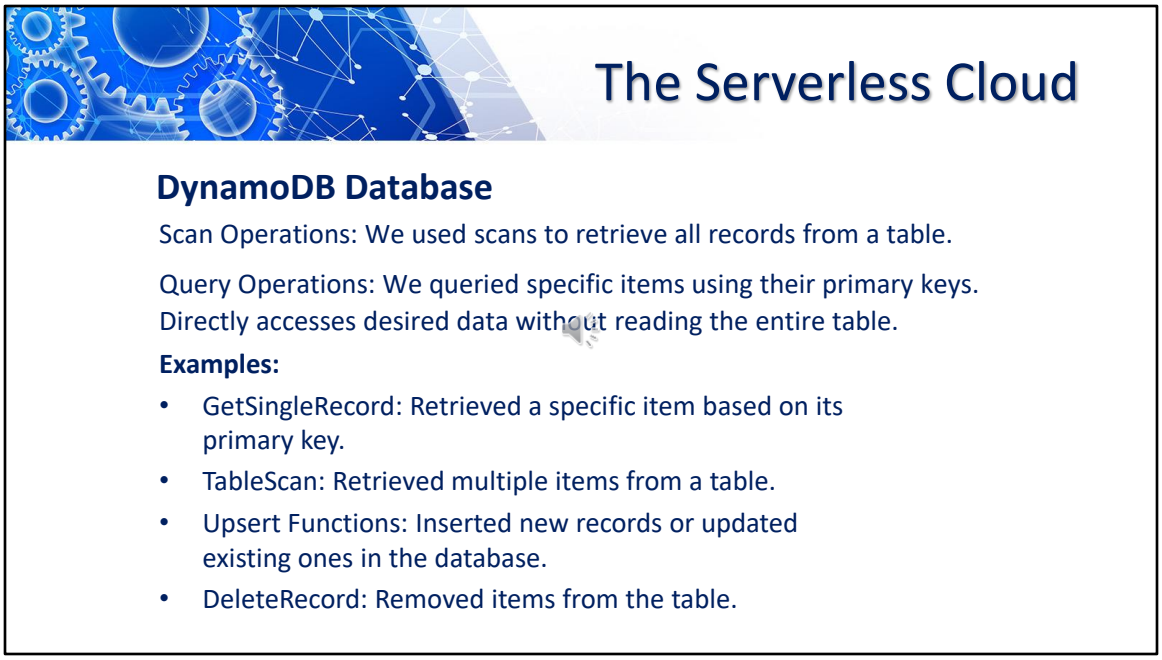
Amazon AWS, n.d.

Amazon AWS. (n.d.). What Is a Key-Value Database? Amazon Web Services, Inc. Retrieved October 8, 2024, from https://aws.amazon.com/nosql/key-value/

MongoDB Documents: Document, Array, Embedded Document. (n.d.). Tutorialsteacher.com. Retrieved October 8, 2024, from https://www.tutorialsteacher.com/mongodb/documents

Initially, our application used MongoDB, a document-based database that stores data in flexible, JSON-like documents. This flexibility meant that we didn't need to define a strict schema upfront. This allowed for rapid development and easy modifications.

When we migrated to the cloud, we switched to Amazon DynamoDB, AWS's NoSQL database service. DynamoDB requires us to define primary keys for each item. Primary keys require planning data more carefully, but it also helped optimize data retrieval and storage efficiency. This structure is key to DynamoDB's ability to provide fast and consistent performance, even in larger scales.



To interact with DynamoDB, we performed various queries. This includes:

Scan Operations, which were used to retrieve all records from a table.

Query Operations were used to query specific items using their primary keys. This method directly accesses desired data without reading the entire table.

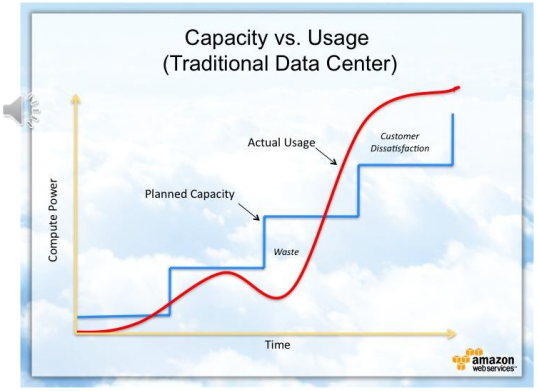
Several Lambda functions were created to interact with the DynamoDB database. For example, GetSingleRecord was used to retrieved a specific item based on its primary key.
Other examples include TableScan, which retrieved multiple items from a table and DeleteRecord, which removes items from the table.

Cloud-Based Development Principles

Serverless

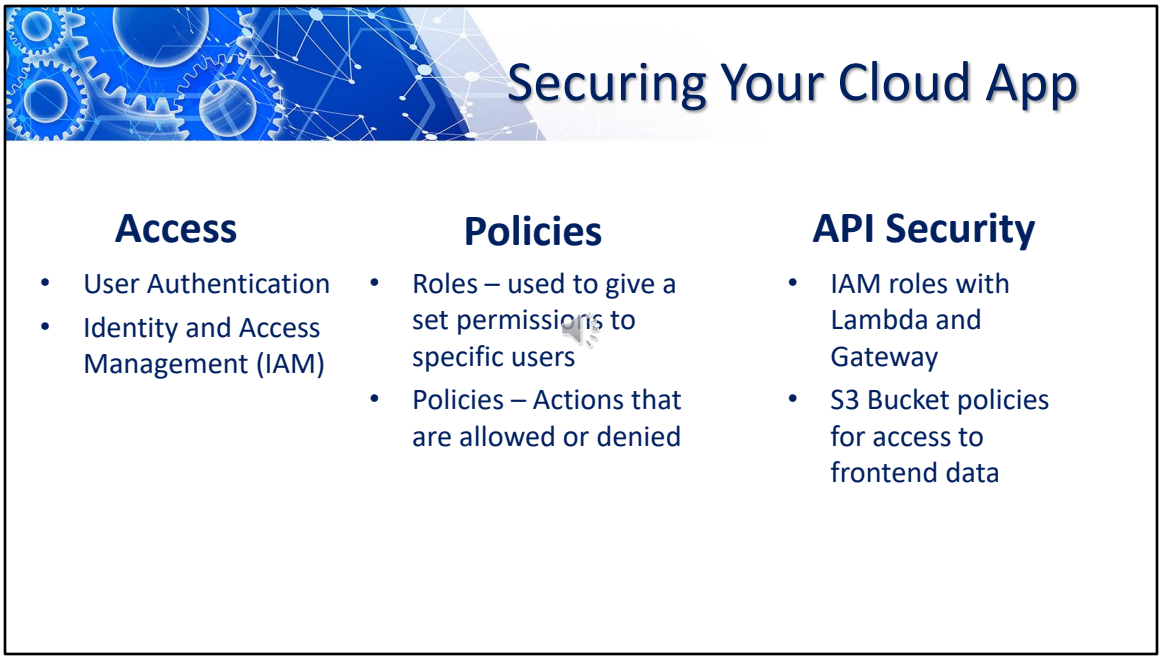
- **Elasticity**
  Automatic scaling as needed
- **Pay-for-use model**
  Only pay for resources used

Traditional Servers



- Let's compare core serverless principles to traditional servers. Note the graph depicting traditional data center capacity planning. In this graph, the planned capacity is represented by the stepped line, indicating how companies traditionally allocate server resources based on expected future demand. The actual usage is shown by the red curved line and rarely matches the planned capacity. The area between these two lines demonstrates a problem of wasted resources due to over-provisioning which wastes capacity. We can also see that when too few resources are allocated, the customer experiences issues using our service. With that in mind, we still don't want to over-provision because it leads to unnecessary expenses for the company.
- Elasticity is the cloud's solution to this problem, with the ability to automatically adjust resources to match our company needs. In our application, we utilized AWS services like Lambda and DynamoDB, which scale as user activity increased or decreased. When more users accessed our application, AWS automatically allocated more resources to handle the load without needing oversight from us. As a result, we are able to deliver a consistent experience to our users.

- The cloud uses a pay-for-use business model, which means we only pay for the resources we actually use. With AWS Lambda, we are billed only for how long it takes to run each function. This means that during low site traffic, our costs decreased since there were fewer function executions. Using this model reduces our overall infrastructure costs compared to traditional servers. This is because we don’t have to invest in hardware and pay for it to run continuously, regardless of usage.



We secured our cloud application by preventing unauthorized access through AWS Identity and Access Management (IAM). This lets us control who can access resources by assigning roles to users and services, and also by defining permissions with policies.
We created custom policies to ensure specific Lambda functions could access only the DynamoDB tables they needed. For example, we had a policy allowing a Lambda function to perform dynamodb:GetItem on the 'Questions' table.
Our application was secured through configuring cloud components like Lambda, API Gateway, DynamoDB, and S3.
- Between Lambda and API Gateway, we used IAM roles so the API Gateway could invoke our Lambda functions securely.
- Lambda and DynamoDB interactions involved assigned roles to Lambda functions for secure database access.
- The S3 Bucket had set bucket policies to control access to our frontend files hosted on S3.
These actions result in a more secure cloud application as each component has the minimal privileges needed to function.

CONCLUSION

To summarize:

- Using containerization
- Our migration to serverless
- Secure microservices with IAM and policies

Thank you for your time.

In summary we've discussed containerization, embracing serverless architecture, and cloud security.

Firstly, we streamlined our development with containerization using Docker and Docker Compose. This made our development environments more consistent, simplifying the management of multiple application components.

Secondly, we embraced serverless architecture using microservices like Lambda, API Gateway, and S3. This allowed us to adopt a more scalable application that was also more cost-effective. Additionally, infrastructure being handled by the service let us focus on development.

And lastly, we enhanced our security by using specific IAM roles and policies, protecting our resources from unauthorized access.

Its these key aspects that enhanced our application development, resulting in improved scalability, efficiency, and security.