Victor Jarvis

CS 470

February 2025

Project Two Conference Presentation: Cloud Development

CS 470 Project Two Conference Presentation: Cloud Development

Victor Jarvis

February 2025

Overview

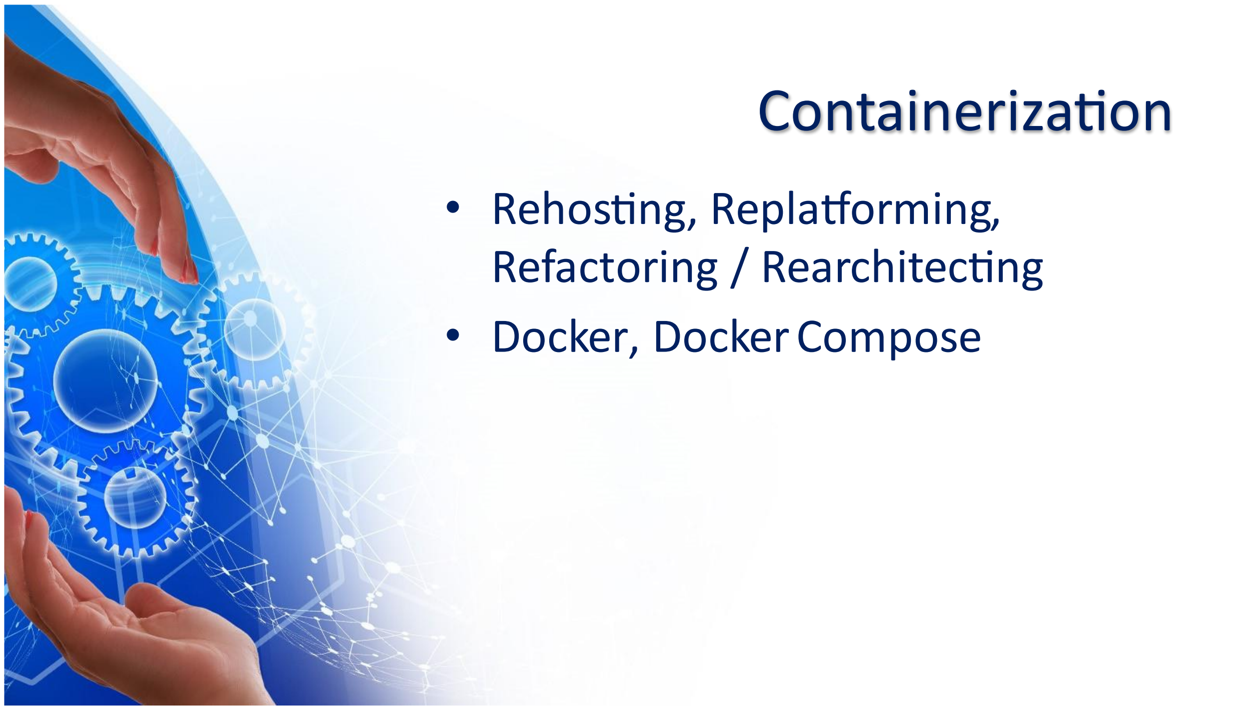
- Who am I?
  - I'm Victor
- Why are we here?
  - To discuss cloud development. (For both technical and nontechnical audiences.)

Introduction

Hello everyone, my name is Victor, and today we'll be walking through the journey of migrating a full stack application to a cloud environment using AWS. Really quickly, about myself, I'm a software engineer based in the Midwest, and I'm wrapping up my Computer Science degree.

So, why are we here? To discuss cloud development and migration, and methodologies.

We're going to begin by talking about Containerization, which is foundational for many modern cloud deployments.



Containerization

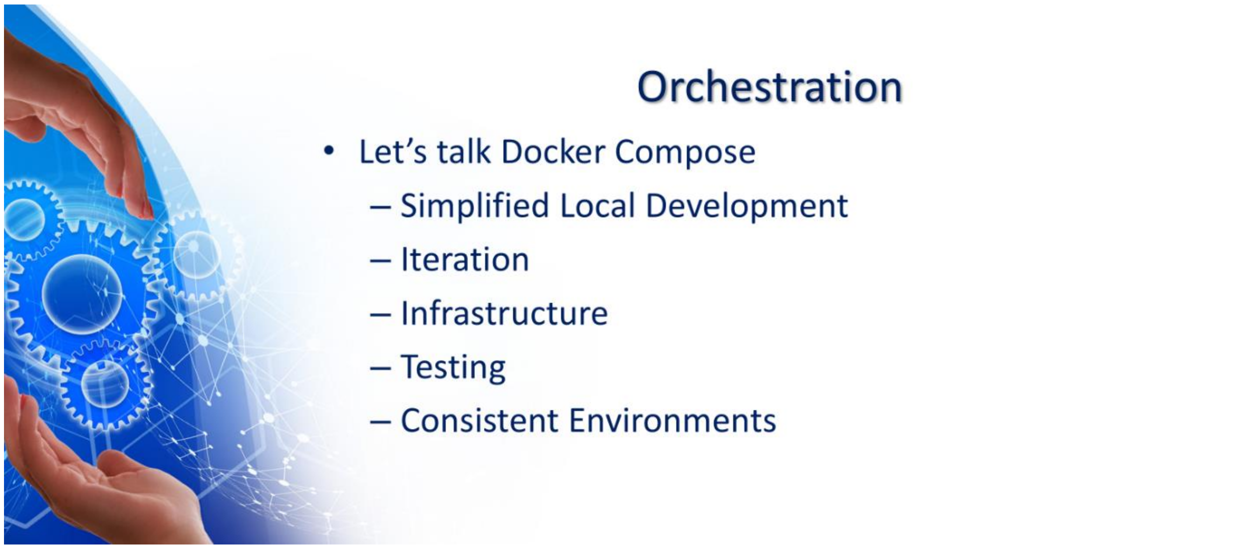
- Rehosting, Replatforming, Refactoring / Rearchitecting
- Docker, Docker Compose

There are multiple different migration models, but these are a few that I want to highlight:

Rehosting, or the "Lift and Shift", which involves taking an existing application and just redeploying it in the cloud with minimal changes. Replatforming involves updating without completely redesigning your app. Refactoring or Rearchitecting involves rewriting or significantly altering your application to be cloud-native, often to embrace microservices or serverless architecture.

Key Containerization Tools

Docker bundles your app code, dependencies, and runtime into containers, and Docker Compose simplifies spinning up multi-container environments locally. With these tools, you can standardize your environment and accelerate deployment cycles.



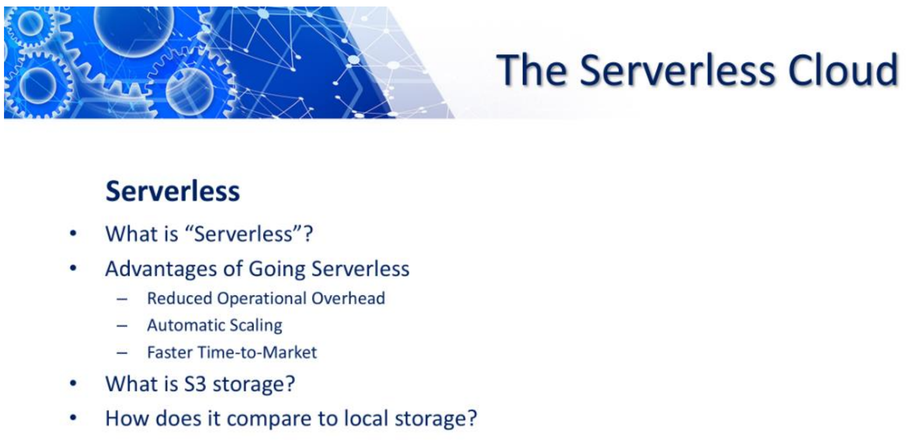
Orchestration

- Let's talk Docker Compose
  - Simplified Local Development
  - Iteration
  - Infrastructure
  - Testing
  - Consistent Environments

Now we're going to discuss Orchestration. Orchestration is about automating the deployment, management, and scaling of your containerized apps. While many devs often use Kubernetes, Amazon ECS, or AWS Fargate, but for local development, we are going to rely on Docker Compose.

A very valuable aspect of Compose is Simplified Local Development. One command can brings up all containers, such as web, API, or DB. It features Rapid Iteration, as you can quickly rebuild or restart services without manual steps. The docker-compose.yml file defines the entire environment, version-controlled alongside your application code. Compose also helps replicate production-like setups locally, keeping development environments consistent.

Next up, we're going to talk about serverless solutions, which remove even more infrastructure concerns from the developer's plate.


The Serverless Cloud

Serverless

- What is "Serverless"?
- Advantages of Going Serverless
  - Reduced Operational Overhead
  - Automatic Scaling
  - Faster Time-to-Market
- What is S3 storage?
- How does it compare to local storage?

Serverless computing means you don't manage servers at all—your cloud provider (AWS) takes care of provisioning and auto-scaling. With AWS Lambda, your code runs only when invoked, and you pay per execution rather than for idle time.

Let's discuss advantages of serverless computing. Not having servers to patch or scale manually means a reduced operational overhead. With automatic scaling, the platform seamlessly handles spikes in traffic. Freed from infrastructure tasks, you can deploy features more quickly.

Amazon S3, or Simple Storage Service, is an object-based storage system that is highly durable and scalable. Data is automatically replicated across multiple Availability Zones.

While local storage is bound to a single machine and requires manual backups, while S3 is virtually unlimited, replicated by default, and pay-as-you-go, so no server management is required.

S3 allows us to store things like static files, media, or backups in a more cost-effective and reliable manner, which is very important for a serverless architecture.

The Serverless Cloud

API & Lambda

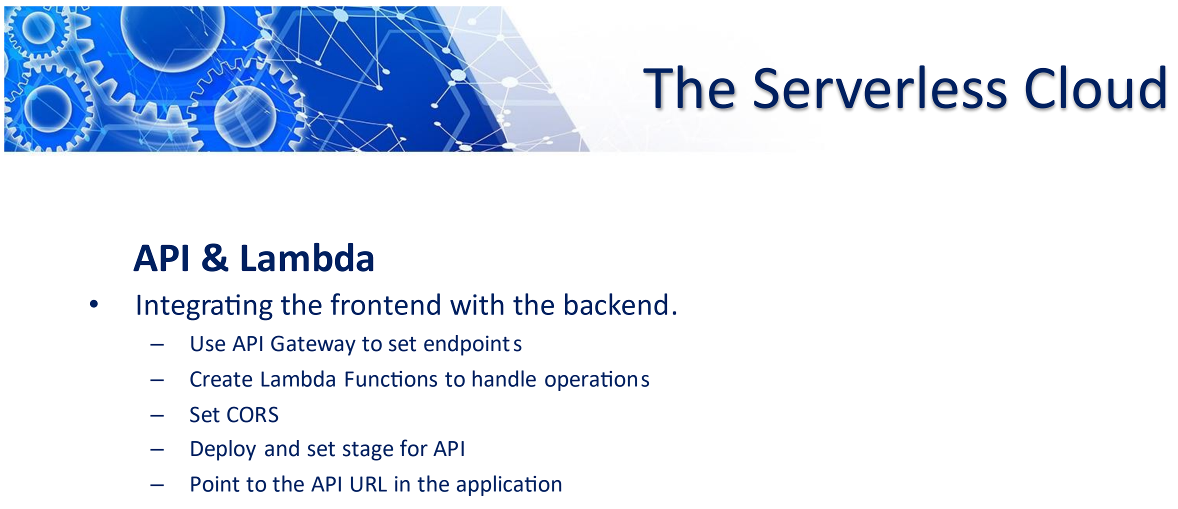
- Advantages a serverless API:
  - On-Demand Scaling
  - Cost Efficiency
  - Rapid Deployment
- Lambda API Logic
- Build & Deployment Scripts

Continuing with the serverless cloud, we're going to discuss APIs and Lambda.

Serverless APIs offer several advantages, like On-Demand Scaling, where it auto-scales based on traffic, and Cost Efficiency, due to the fact that you pay only for each API call and Lambda invocation. Rapid Deployment allows you to update or roll back your APIs in minutes. AWS managing the backend infrastructure reduces the maintenance needed.

Lambda API Logic

Lambda functions trigger in response to API Gateway requests. It is also stateless, so each invocation is independent, and data is stored externally, such as, in a database. Through its handler structure, code processes the request sent through, accesses data if needed, and returns the appropriate response.



The Serverless Cloud

API & Lambda

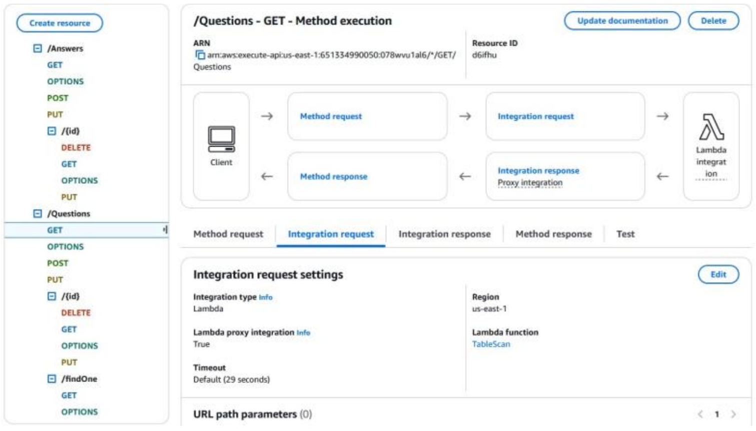
- Integrating the frontend with the backend.
  - Use API Gateway to set endpoints
  - Create Lambda Functions to handle operations
  - Set CORS
  - Deploy and set stage for API
  - Point to the API URL in the application

In our case, we utilized API Gateway to set our endpoints revolving around basic CRUD operations, with Lambda functions being set to handle these operations. CORS attributes and settings are established, and after ensuring the API URL is set in the application, make sure the proper deployments have been handled, and the stage is set in Gateway so the two ends can communicate.

The Serverless Cloud

Database

- Comparing MongoDB and DynamoDB
- CRUD Queries:
  - GET
  - PUT
  - DELETE
  - POST
  - OPTIONS for CORS



Create resource

- /Answers
  - GET
  - OPTIONS
  - POST
  - PUT
  - /{id}
    - DELETE
    - GET
    - OPTIONS
    - PUT
- /Questions
  - GET
  - OPTIONS
  - POST
  - PUT
  - /{id}
    - DELETE
    - GET
    - OPTIONS
    - PUT
  - /findOne
    - GET
    - OPTIONS

/Questions - GET - Method execution

Update documentation | Delete

ARN
arn:aws:execute-api:us-east-1:651334990050:078wvu1al6/*/GET/Questions

Resource ID
d6ifhu

Method request | Integration request | Integration response | Method response | Test

Integration request settings

Edit

Integration type Info
Lambda

Region
us-east-1

Lambda proxy integration Info
True

Lambda function
TableScan

Timeout
Default (29 seconds)

URL path parameters (0)

Continuing once more with The Serverless Cloud, we’re going to look at databases, and quickly compare MongoDB & DynamoDB

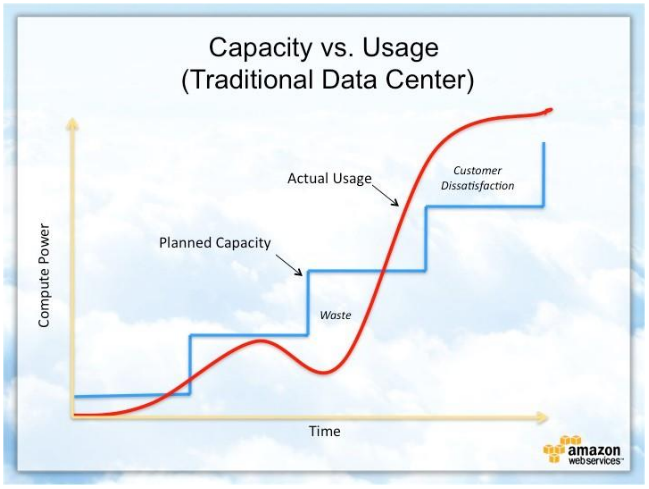
MongoDB is document-based, has a flexible schema, rich queries, and can be self-hosted or run on MongoDB Atlas, while DynamoDB utilizes key-value NoSQL with partition and sort keys, which are fully managed by AWS. DynamoDB offers higher performance and more seamless scalability.

CRUD Operations

- Create: UpsertQuestion and UpsertAnswer.
- Read: GET options for Questions and Answers, including finding singular entries.
- Update: Upsert functions used with PUT.
- Delete: by ID

Cloud-Based Development Principles

- Elasticity
  - Definition
  - Example
  - Why It Matters
- Pay-For-Use
  - Definition
  - Example
  - Why It Matters



Elasticity is a system’s ability to scale resources up or down automatically in response to load. A sudden surge in user activity can be handled well with proper planning and management.

Why It Matters: It saves on cost and ensures your app stays responsive under sudden spikes, with no over-provisioning.

In the Pay-for-Use Model, you’re charged only for the resources you consume. Lambda bills per request and runtime, S3 bills per stored gigabyte, and EC2 bills per time used.

Why It Matters: Lowers the barrier to entry, fosters experimentation, and aligns costs with actual usage.

Securing Your Cloud App

Access

- Authentication & Authorization
- IAM Roles

Policies

- Roles vs. Policies
- Custom policies

API Security

- Securing the connection between Lambda and Gateway
- Lambda and the database
- S3 Bucket

To prevent unauthorized access, you want to secure your internal services with IAM roles.

What are roles? Roles are assumable identities for AWS services or external users that carry temporary credentials. Policies are JSON documents defining allowed/denied actions on resources. You attach policies to roles to grant minimal permissions needed, following the principle of least privilege.

Setting up an IAM Role for Lambda grants your function permission to access DynamoDB, S3, or other resources, while an API Gateway Integration Role allows API Gateway to invoke Lambda functions. By enforcing strict roles and policies, and encrypting data, you ensure your application remains both functional and secure.

CONCLUSION

- **Efficiency**
- **Choice**
- **Security**

Thank you for your time.

In conclusion, our key takeaways are going to center on Efficiency, choice, and security. Migrating your application to the cloud means you only pay for what you use, and the platform auto-scales as traffic grows. Selecting the right data storage and leveraging services like S3 can drastically simplify your operations while increasing your application's scalability and reliability. And finally, security. Properly configuring your roles and policies, coupled with proper authentication processes, keeps your application safe and ensures compliance. These principles are key foundations to successfully migrating your full stack application to AWS.