**Thomas Martin**

**CS-470-18332-M01 Full Stack Development II**

**7-1 Submit Project Two**

**Project Two Conference Presentation: Cloud Development**

**Southern New Hampshire University**

**December 13, 2024**

**Talking Points for PowerPoint Presentation**

**Opening Statement:**

*Welcome to this presentation on the migration journey from a traditional full-stack web application to a cloud-native solution powered by AWS microservices.*

**Purpose of the Presentation:**

*This session is designed to bridge the gap between technical and nontechnical perspectives, ensuring the intricacies of cloud development are clear and accessible to all.*

**Topics to Explore:**

**Essential Steps in the Migration Process:**

*Containerization*

*Containerization Platform*

*cloud-native services*

*Orchestration*

*The Serverless Cloud*

*Amazon S3*

*API & Lambda*

*Database*

**Transition Statement:**

*Together, we’ll walk through the transformation process, exploring how modern cloud technologies enable scalability, efficiency, and resilience in web applications.*

**Talking Points for Containerization and Cloud Migration**

**1. Introduction to Containerization**

*What is Containerization?*

*Containerization involves packaging an application and its dependencies into a lightweight, portable container that can run consistently across different environments.*

*Containers ensure that the application works reliably regardless of the underlying infrastructure.*

**2. Migrating a Full-Stack Application to the Cloud**

**Key Considerations:**

*Selecting the right migration model is crucial for aligning with application requirements and leveraging cloud infrastructure benefits.*

*Migration models differ based on the level of changes made to the application.*

**Four Common Cloud Migration Models:**

*Rehosting:*

*Moving the application 'as-is' to the cloud without modifying its architecture.*

*Best for legacy applications or quick migrations.*

*Replatforming:*

*Making minor changes to optimize the application for the cloud.*

*Balances ease of migration with performance gains.*

*Refactoring:*

*Rebuilding significant portions of the application to make it cloud-native.*

*Involves restructuring for scalability and flexibility.*

*Rebuilding:*

*Completely redesigning the application for the cloud from scratch.*

*Offers maximum benefits but requires significant effort.*

*Our Approach for This Project*

*Migration Model: Replatforming*

*For this project, we will be using the replatforming model, making minor changes to optimize the application for the cloud.*

*Focused on balancing migration speed and leveraging cloud capabilities.*

*Containerization Platform: Docker*

**Why Docker?**

*Docker is the most popular containerization platform, used for creating, managing, and running containers.*

**Offers:**

*Portability across environments.*

*Resource efficiency and fast deployment.*

*Extensive community support and integration.*

*Cloud-Native Services and Monitoring Tools*

**Using Docker Compose for Multi-Container Applications:**

*Docker Compose allows us to define and manage multi-container Docker applications with a single configuration file.*

**Benefits include:**

*Simplified deployment for applications with multiple services (e.g., frontend, backend, database).*

*Centralized management of container configurations for efficiency.*

*Importance of These Tools*

*These tools are essential for streamlining the migration process, ensuring that the application performs effectively in the cloud environment.*

**Docker and Docker Compose:**

*Enhance development speed and reliability.*

*Simplify cloud integration and monitoring.*

**Transition to Next Section**

*With these tools and strategies in place, we are well-equipped to ensure a smooth migration process, maximizing the benefits of cloud infrastructure for our application.*

**Talking Points for Orchestration**

*1. Introduction to Orchestration*

*Orchestration refers to the automated arrangement, coordination, and management of containers in complex applications.*

*It ensures that containerized applications run efficiently and are easily manageable as they scale in size and complexity.*

*2. Why Orchestration is Essential*

*Containers provide portability but managing them manually becomes challenging as applications grow.*

*Orchestration simplifies the management of multi-container environments, ensuring consistency and reliability.*

*Key Tasks Automated by Orchestration*

**Deploying and Starting Containers:**

Orchestration automates the deployment and initialization of containers, ensuring all services are up and running seamlessly.

**Scaling Applications:**

*It adjusts the number of containers automatically to meet changes in demand, scaling up during peak times and scaling down when demand decreases.*

**Ensuring High Availability and Fault Tolerance:**

*Orchestration ensures applications remain available by restarting failed containers or redistributing workloads.*

**Managing Networking Between Containers:**

*It handles communication between containers, linking services securely and efficiently.*

*Updating Applications with Minimal Downtime:*

*Orchestration supports rolling updates, ensuring that new versions of applications are deployed without disrupting service.*

**Monitoring and Logging:**

*It tracks container performance, providing metrics and logs for troubleshooting and optimization.*

**4. Benefits of Orchestration**

*Reduces manual intervention, improving developer productivity.*

*Enhances system reliability and uptime through automated fault recovery.*

*Simplifies complex deployments, making applications more scalable and resilient.*

**5. Transition to Tools**

*With orchestration at the core of container management, tools like AWS ECS provide powerful platforms to automate these tasks effectively, making cloud-native application management simpler and more robust.*

**Talking Points for Serverless Computing and Amazon S3**

**1. Introduction to Serverless Computing**

*Serverless is a cloud computing execution model where the cloud provider dynamically manages the allocation and provisioning of servers.*

*Removes the need for server management, allowing developers to focus entirely on writing code and building applications.*

**2. Benefits of Serverless Computing**

**Scalability:**

*Serverless applications automatically scale based on demand, efficiently handling variable workloads.*

*No need for manual intervention to adjust capacity.*

**Cost Efficiency:**

*With a pay-as-you-go model, users only pay for the compute resources consumed during execution, avoiding idle server costs.*

**Rapid Deployment:**

*Serverless platforms provide streamlined deployment pipelines, accelerating the development lifecycle.*

*Developers can deploy code in minutes without configuring servers.*

**Global Availability**:

*AWS replicates serverless functions across regions, ensuring low-latency and high-availability services worldwide.*

**3. Amazon S3 Overview**

*Amazon S3 (Simple Storage Service) is an object storage service provided by AWS, designed to store and retrieve any amount of data, anytime, from anywhere on the web.*

**Features include:**

*Scalability: Handles vast amounts of data seamlessly.*

*Durability: Designed for 99.999999999% (11 9's) durability by replicating data across multiple servers and facilities.*

*Security: Provides advanced security features, including encryption and fine-grained access controls.*

**4. Comparing S3 with Local Storage**

*Advantages of S3:*

*S3 offers unmatched scalability, durability, and global accessibility, making it ideal for distributed applications and massive datasets.*

*Supports high-availability use cases like backups, media storage, and content delivery.*

**Advantages of Local Storage:**

*Local storage is better suited for low-latency, offline operations, or secure, localized data handling.*

*Useful for applications requiring fast access without relying on an internet connection.*

**5. Use Cases for S3 in Cloud Applications**

*Content delivery for websites or mobile apps.*

*Storage of large-scale backups or disaster recovery data.*

*Hosting media files, logs, or other unstructured data for analytics.*

**6. Transition Statement**

*By combining serverless computing with services like Amazon S3, modern cloud applications achieve unprecedented levels of scalability, cost efficiency, and global reach, transforming the way we build and deploy software solutions.*

**Talking Points: Advantages of Using a Serverless API**

**1. Introduction to Serverless APIs**

*Serverless APIs, powered by cloud services like AWS Lambda, offer an efficient and modern approach to building and scaling applications.*

*Eliminates the need for traditional server management, allowing developers to focus solely on application logic.*

**2. Key Advantages of Serverless APIs**

**Scalability:**

*Serverless APIs automatically adjust to traffic demands, effortlessly handling spikes without any manual intervention.*

*Perfect for unpredictable workloads or rapidly growing user bases.*

**Cost Efficiency:**

*With a pay-as-you-go model, you only pay for the compute resources consumed during execution, avoiding costs for idle servers.*

*Cost-effective for startups and enterprises alike.*

**Reduced Operational Overhead:**

*No need to manage, patch, or configure servers—the cloud provider handles the underlying infrastructure.*

*Frees up development resources for innovation and faster delivery.*

**Global Reach:**

*Easily deploy APIs across multiple regions, ensuring low latency and a better user experience for customers worldwide.*

**3. Lambda API Logic**

*AWS Lambda is a widely adopted choice for building serverless APIs because of its seamless integration with other AWS services and ability to execute backend logic efficiently.*

*Automatically scales based on demand and supports event-driven workflows.*

**Why Choose Lambda?**

*Integrates with AWS services like DynamoDB, S3, and API Gateway.*

*Executes code in response to events such as HTTP requests or database updates.*

*Reduces operational overhead and speeds up development cycles.*

**4. Integrating Frontend with Backend**

*For this project, the integration process includes:*

**Developing Backend Logic:**

*We used frameworks like Node.js to create efficient backend services.*

**Creating RESTful API Endpoints:**

*These endpoints handle data flow between the frontend and backend, enabling structured communication.*

**Enabling Cross-Origin Resource Sharing (CORS):**

*CORS ensures secure data exchange between the frontend and backend, especially when hosted on different domains.*

**Building the Frontend:**

*Frameworks like Angular were used to develop a responsive UI, connected to the API for seamless data exchange.*

**5. Transition to Cloud-Native Benefits**

*By leveraging serverless APIs and modern frameworks, we’ve created a scalable, cost-efficient, and globally accessible solution that simplifies the development process while enhancing user experiences.*

**Talking Points: Amazon DynamoDB Overview**

**1. Introduction to Amazon DynamoDB**

*Amazon DynamoDB is a fully managed NoSQL database service provided by AWS.*

*Designed for applications requiring low-latency, high-throughput performance, and scalability.*

*Ideal for managing key-value and document-based data with the flexibility to adapt to diverse use cases.*

**2. Key Features of DynamoDB**

**Serverless Architecture:**

*DynamoDB eliminates the need to manage servers or infrastructure, allowing developers to focus on application logic.*

*Fully managed by AWS, reducing operational overhead.*

**On-demand Scaling:**

*DynamoDB automatically scales to handle traffic spikes and adjusts capacity as needed.*

**Offers two modes:**

*Provisioned Capacity: Ideal for predictable workloads.*

*On-demand Capacity: Suitable for unpredictable traffic patterns.*

*Data Model:*

*Key-Value Pairs:*

*Efficient for simple lookups and straightforward data retrieval.*

**Document Storage:**

*Supports flexible, JSON-like documents for unstructured or semi-structured data.*

**3. Queries Used in This Project**

*For this project, we utilized standard CRUD operations via HTTP methods to interact with DynamoDB.*

**Operations Include:**

*GET: Retrieve items from the database.*

*POST: Add new items to the database.*

*PUT: Update existing items.*

*DELETE: Remove items from the database.*

*These methods ensure seamless and efficient interaction between the application and the database.*

**4. Scripting with JavaScript**

*We used JavaScript scripts to connect the frontend to DynamoDB, leveraging the AWS SDK for JavaScript to perform CRUD operations.*

*Capabilities of the Scripts:*

*Querying data for retrieval.*

*Inserting new data entries.*

*Updating existing data records.*

*Deleting unnecessary or obsolete data.*

*Example: The AWS SDK simplifies integration and automates interactions with the database.*

**5. Why DynamoDB for This Project**

*Scalability and performance tailored to the needs of cloud-native applications.*

*Serverless architecture eliminates infrastructure management, allowing faster development cycles.*

*Flexibility to handle both structured and unstructured data efficiently.*

**6. Transition Statement**

*By leveraging DynamoDB, we created a robust and efficient backend capable of handling real-time data demands with minimal operational overhead.*

**Talking Points: Cloud-Based Development Principles**

**1. Introduction to Cloud-Based Development Principles**

*Cloud-based development leverages the unique advantages of cloud infrastructure to build scalable, cost-efficient, and agile applications.*

*Two key principles that drive cloud efficiency are Elasticity and the Pay-for-Use Model.*

*2. Elasticity*

**Definition:**

*Elasticity refers to the ability of a cloud-based system to automatically scale resources up or down based on demand.*

**Key Benefits:**

*Demand-Driven Scaling:*

*Resources automatically expand during traffic spikes and shrink during low usage periods.*

*This ensures optimal performance without over-provisioning resources.*

**Cost Optimization:**

*With elasticity, organizations pay only for the resources they actually use, reducing waste.*

**Enhanced User Experience:**

*Applications remain highly responsive, even under variable workloads, ensuring consistent performance.*

**3. Pay-for-Use Model**

**Definition:**

*The pay-for-use model ensures users are charged only for the compute, storage, or network resources they consume.*

**Key Benefits:**

**Cost Efficiency:**

*No upfront costs for unused resources, allowing businesses to align expenses with actual usage.*

**Flexibility:**

*Businesses can scale their operations without committing to fixed infrastructure investments.*

*Accessibility for All Sizes:*

*Startups and enterprises alike can leverage cloud resources without large capital expenditures.*

*Example in Practice:*

*A startup using AWS Lambda only pays for the execution time of its functions, avoiding the costs of idle servers.*

**4. Why These Principles Matter**

*Elasticity and the pay-for-use model enable businesses to respond quickly to market demands, reduce operational costs, and ensure high availability, making cloud-based development a cornerstone of modern applications.*

**5. Transition Statement**

*By adhering to these cloud-based development principles, we can create applications that are not only efficient and scalable but also cost-effective and highly adaptive to user needs.*

**Talking Points: Securing Your Cloud Application**

**1. How Can You Prevent Unauthorized Access?**

*Identity and Access Management (IAM):*

*Use IAM to control access to AWS resources.*

*Implement least privilege access to limit permissions to only what is necessary.*

**Resource Policies:**

*Apply policies on resources like S3 buckets and API Gateway endpoints to restrict access.*

**2. Explain the Relationship Between Roles and Policies**

*Roles:*

*Roles provide temporary credentials for AWS services to perform specific tasks without hard-coded credentials.*

*Example: A Lambda function uses a role to access a DynamoDB table.*

*Policies:*

*Policies define permissions in JSON format, specifying allowed or denied actions on AWS resources.*

*Relationship:*

*Roles assume the permissions defined in policies.*

*Policies can be attached to roles, users, or groups, ensuring consistent and secure access control.*

**3. What Custom Policies Were Created?**

**Custom Policies Implemented in the Project:**

**Lambda to S3 Access Policy:**

*Allows Lambda functions to read and write to specific S3 buckets.*

**API Gateway Resource Policy:**

*Restricts API access to specific IP ranges or authenticated users via IAM.*

**Database Access Policy:**

*Grants Lambda read and write permissions for a specific DynamoDB table, limiting access to required operations only.*

**4. Securing the Connection Between Lambda and API Gateway**

*Implement resource policies on API Gateway to restrict access to trusted sources.*

**5. Securing the Connection Between Lambda and the Database**

**Best Practices:**

*Grant Lambda access to the database with a specific IAM role and a custom policy.*

**6. Securing the S3 Bucket**

*Key Security Measures for S3:*

*Bucket Policies:*

*Restrict access to specific IAM roles, users, or trusted IP addresses.*

*S3 Block Public Access:*

*Prevent accidental public exposure of data by enabling block public access settings.*

*Versioning:*

*Enable versioning to safeguard against accidental overwrites or deletions.*

*Access Logging:*

*Enable S3 logging to monitor and audit access to the bucket.*

**7. Transition Statement**

*By implementing these security best practices, we can ensure a robust and secure cloud application, safeguarding data integrity, confidentiality, and availability while adhering to AWS’s shared responsibility model.*

**Talking Points: Three Main Points About Cloud Development**

**1. No Server Management**

*One of the core advantages of cloud development, especially with serverless models, is the elimination of server management.*

*Developers no longer need to provision, patch, or maintain physical or virtual servers.*

*Cloud providers, like AWS, handle all infrastructure-related tasks, allowing teams to focus on writing and deploying code.*

**2. Elasticity and Scalability**

*Cloud development ensures applications can adapt seamlessly to varying workloads.*

*Elasticity:*

*Automatically scales resources up during high demand and scales down during low usage periods.*

*Ensures optimal resource utilization and performance.*

*Scalability:*

*Cloud services can scale to accommodate millions of users or scale back to minimal resources for small workloads.*

**3. Cost-Efficiency Through the Pay-for-Use Model**

*The pay-as-you-go model is a hallmark of cloud development, offering significant cost advantages.*

*Users pay only for the compute, storage, or network resources they consume, avoiding costs for idle infrastructure.*