CSCI 5409
Cloud Computing

Term Project – Final Report

Balaji Sukumaran (bl664064@dal.ca)

Banner ID: B0094897

GitLab Repo: https://git.cs.dal.ca/sukumaran/wallet_watch

Table of contents

Introduction:

In today's fast-paced world, managing personal finances can become a complex task for many individuals. Recent surveys and studies have indicated that a significant portion of the population struggles with budgeting and maintaining a balanced financial lifestyle. The challenges of keeping track of expenses, budgeting efficiently, and saving for future needs are becoming increasingly prevalent. As economic conditions fluctuate and living costs rise, the necessity for effective financial management tools becomes more critical than ever [1].

Objective: Wallet Watch

Recognizing the widespread need for enhanced financial management, our project, Wallet Watch, is designed to empower users with a comprehensive web application that simplifies tracking of expenses, budgeting, and income.

The core functionality of Wallet Watch revolves around providing users with a *user-friendly platform* to monitor their financial activities closely and receive prompt *email alerts* when their spending exceeds predefined thresholds. This proactive approach ensures that users can adjust their spending habits timely and avoid overspending, which can lead to financial strain.

Target Users and Performance Targets

Wallet Watch is aimed at a broad user base, encompassing individuals who seek to improve their financial discipline and gain better control over their financial status. The application is engineered to cater to users with varying degrees of financial literacy, ensuring that its interface is straightforward and accessible for everyone.

To meet the needs and expectations of our users, Wallet Watch is developed with the following performance targets in mind:

1. ***High Availability***: Leveraging the latest cloud computing technologies, the deployment of Wallet Watch prioritizes high availability to ensure that users can access the application at any time. This is crucial for maintaining the effectiveness of the budgeting and alert system, as delayed or missed alerts could lead to unplanned overspending.
2. ***Speed and Efficiency***: Understanding the importance of user experience, Wallet Watch is optimized for speed. The application is designed to allow users to log their expenses swiftly, making the process as efficient as possible to avoid frustration and encourage regular use.
3. ***Simplicity and Usability***: With a focus on inclusivity, the application features a simple and intuitive interface. This design philosophy ensures that Wallet Watch is easily navigable, making financial management an achievable task for users regardless of their tech-savviness.

By addressing the critical need for a user-friendly and efficient financial management tool, Wallet Watch aims to revolutionize the way individuals approach budgeting and expense tracking. Through its innovative use of cloud technology and a keen focus on the user experience, Wallet Watch is positioned to become a pivotal tool in the personal finance management space.

Menu Item Requirements and Service Selection:

Selected Cloud Services

1. ***Compute***:

   AWS EC2 (Elastic Compute Cloud): A virtual server service for running applications on the Amazon Web Services infrastructure [2].

   AWS Lambda: A serverless compute service that runs our code in response to events and automatically manages the compute resources [3].

2. ***Storage***:

   AWS S3 (Simple Storage Service): An object storage service that offers industry-leading scalability, data availability, security, and performance [4].

   AWS RDS (Relational Database Service): A managed relational database service that supports MySQL, PostgreSQL, Oracle, SQL Server, and MariaDB [5].

3. ***Network***:

   VPC (Virtual Private Cloud): A service that provides a logically isolated section of the AWS cloud where we can launch AWS resources in a virtual network that we define [6].

4. ***General***:

   AWS Textract: A service that automatically extracts text and data from scanned documents [7].

   AWS EventBridge: A serverless event bus service that makes it easy to connect applications together using data from our own applications, integrated SaaS applications, and AWS services [8].

   AWS Secrets Manager: A management service for storing, retrieving, and rotating database credentials, API keys, and other secrets throughout their lifecycle [15].

   AWS Simple Email Service (SES): A cost-effective, flexible, and scalable email service that enables developers to send mail from within any application [9].

Comparison with Alternative Services

For ***Compute***, alternatives like AWS Elastic Beanstalk and Docker with AWS Elastic Beanstalk offer automation in running, load balancing, and scaling web applications. However, they ***abstract much of the underlying control***, which can limit customization and optimization based on specific project needs. AWS EC2 was chosen for its ***flexibility and control*** over the computing environment, crucial for hosting both the frontend and backend of the application efficiently [8]. AWS Lambda and AWS Simple Email Service [3][9], was selected for its serverless execution model, ideal for running code in response to events (e.g., generating email alerts) without managing servers, balancing budget constraints with performance.

In the ***Storage*** category, while options such as AWS DynamoDB offer scalable NoSQL solutions and AWS Aurora provides a managed relational database service, AWS S3 was selected for its ***unparalleled simplicity*** and ***efficiency*** in storing object files, including cloud formation scripts and Lambda jar files. AWS RDS was chosen for our application's database due to its superior managed relational database features, which emphasize simplicity and ***offer greater control***, making other services appear overly complex and ***less suited*** for our specific storage requirements [11].

For ***Network***, AWS Virtual Private Cloud (VPC) was preferred over alternatives like AWS API Gateway and Amazon CloudFront due to its capability to provide a secure and isolated network segment for the application, enhancing security and network management [12].

In the **General** category, AWS Textract and AWS EventBridge were chosen over other services like Amazon Comprehend and AWS Glue for their direct **alignment with project requirements**: Textract for its precise text extraction from images, and EventBridge for facilitating the construction of event-driven applications. Additionally, AWS Secrets Manager is favored for its **superior security management** capabilities, offering seamless secret rotation, tight access controls, and effortless integration with AWS services, thus ensuring enhanced security and operational efficiency in handling sensitive credentials and API keys. These choices reflect a strategic preference for functionality, security, and integration within the AWS ecosystem, streamlining the application's development and deployment processes.

Rationale for Service Selection

The primary considerations in selecting these services were budget, simplicity, control over the environment, and performance.

Control: AWS EC2 was selected to retain full control over the compute environment, essential for customizing the application hosting setup to meet specific performance criteria.

Performance: The selection of AWS EC2 and AWS Lambda ensures that the application can handle requests efficiently, with AWS EC2 providing robust support for the application's frontend and backend, and AWS Lambda facilitating quick and scalable email alert generation to avoid cold starts commonly associated with serverless architectures.

Budget: Services like AWS Lambda for compute and AWS RDS for storage offer cost-efficiency by allowing pay-for-what-you-use pricing models and managed services, reducing the need for extensive infrastructure management.

Simplicity: By choosing managed services such as AWS RDS and AWS Lambda, the complexity of database management and server management is reduced, allowing more focus on application development.

By leveraging these AWS services, the project aims to deliver a highly available, scalable, and secure web application that meets the users' needs while staying within budget and performance targets.

Deployment Model:

The deployment model adopted for our application leverages the robust infrastructure of the AWS cloud, utilizing several AWS components tailored to our specific needs. The application architecture is strategically hosted within an AWS Virtual Private Cloud (VPC), with only the EC2 instance endpoints exposed for public access. This setup positions our application deployment model as a **public cloud deployment** [14]. However, it incorporates elements of privacy and security commonly associated with a private cloud model by restricting direct public access to the database and other internal components.

Justification for This Choice

Opting for this deployment model was a strategic decision aimed at achieving a balance between cost-effectiveness, scalability, and security:

1. Cost-Effectiveness: By hosting all application resources on AWS and utilizing managed services, we significantly reduce the overhead associated with physical infrastructure management and gain access to AWS's scalable resources. This approach allows for an efficient scaling strategy that matches demand without incurring unnecessary costs, taking full advantage of the AWS pay-as-you-go pricing model.
2. Security: The decision to host the application within an AWS VPC and expose only the EC2 instance to the public enhances our application's security posture. It limits potential attack vectors by isolating the database and other critical components from direct public access, ensuring that sensitive data and internal processes are shielded from external threats. Utilizing AWS’s built-in security features further fortifies our application, offering robust protection against unauthorized access.
3. Scalability and Reliability: AWS's cloud infrastructure provides a high degree of scalability and reliability, ensuring that our application remains responsive and available to users even during peak usage times or in the face of component failures.

Delivery Model:

The architecture of our application utilizes a diverse array of AWS services, categorizing our delivery model into a hybrid approach that includes Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Function as a Service (FaaS) [14]. This selection was driven by the distinct functionalities and advantages offered by each service model, tailored to meet the specific requirements of our application components.

IaaS (Infrastructure as a Service): Our use of AWS EC2 and VPC services embodies the IaaS model, providing essential computing resources and network security capabilities. This foundation allows for full control over the virtualized computing environment, tailored to our application's needs.

PaaS (Platform as a Service): AWS RDS, our chosen relational database service, represents the PaaS category, offering a managed database platform. This service abstracts and automates the underlying infrastructure management tasks, enabling us to focus on application development without the complexity of database administration.

FaaS (Function as a Service): AWS Lambda is employed for event-driven, serverless computing, epitomizing the FaaS model. It enables running backend code in response to events (such as triggering email alerts), simplifying deployment and scaling with no server management.

Justification for This Choice

The adoption of a hybrid delivery model, incorporating IaaS, PaaS, and FaaS, was strategically chosen for its multifaceted benefits:

1. Flexibility and Scalability: The model offers the necessary flexibility to select specific services best suited to each aspect of the application, ensuring easy scalability and adaptability to evolving requirements.
2. Cost Efficiency: By leveraging managed services like PaaS and FaaS, operational costs are minimized. FaaS's pay-as-you-go model further reduces expenses by charging only for the compute resources actually used.
3. Simplified Development and Operational Management: Utilizing PaaS and FaaS simplifies operational management and accelerates the development process, allowing the team to concentrate on application logic and user experience rather than infrastructure maintenance.

Final Architecture:

Programming Languages

Wallet watch employs a **React frontend** and a **Spring Boot backend**, providing a responsive user interface and a robust server-side environment, respectively. React was chosen for its efficiency in updating and rendering the right components when data changes, enhancing the user experience with dynamic content. Spring Boot, utilized for the backend, offers a comprehensive platform that simplifies the development of stand-alone, production-grade Spring-based applications.

The backend is further supported by **Java-based AWS Lambda** functions, tasked with sending alerts to users. This choice leverages Java's portability and AWS Lambda's serverless architecture to efficiently manage alerting mechanisms without provisioning or managing servers.

Data Storage Solutions

The application's data storage architecture comprises a relational database hosted on **AWS RDS (MySQL)**, chosen for its reliability and ease of management within the AWS ecosystem. For object storage, such as files or media, **AWS S3** is utilized for its scalability and data availability. **AWS Secrets Manager** securely handles the application's secrets, including database credentials, ensuring sensitive information is stored and accessed securely.

Integration of Cloud Mechanisms

The integration strategy centers around deploying the frontend and backend on an AWS EC2 instance, which resides within a **Virtual Private Cloud (VPC)**. This setup allows public access to the application via the internet while maintaining controlled access to the backend processes. The AWS RDS instance, containing the application's database, is placed within a private subnet of the VPC, accessible only by the EC2 instance. This configuration enhances security by limiting direct exposure to the internet. Application secrets managed by **AWS Secrets Manager** are integral to this architecture, providing a secure method for storing and accessing sensitive configuration details.

Cloud Deployment Strategy

Our deployment strategy utilizes **AWS CloudFormation** for provisioning and managing the AWS components, ensuring a repeatable and consistent infrastructure setup. For the application deployment, a **Gitlab CI/CD pipeline** facilitates automatic updates and deployment processes, pushing the application into a **Docker container hosted on the EC2 instance**. This approach ensures that the deployment process is both efficient and scalable, capable of adapting to future application updates and infrastructure changes.

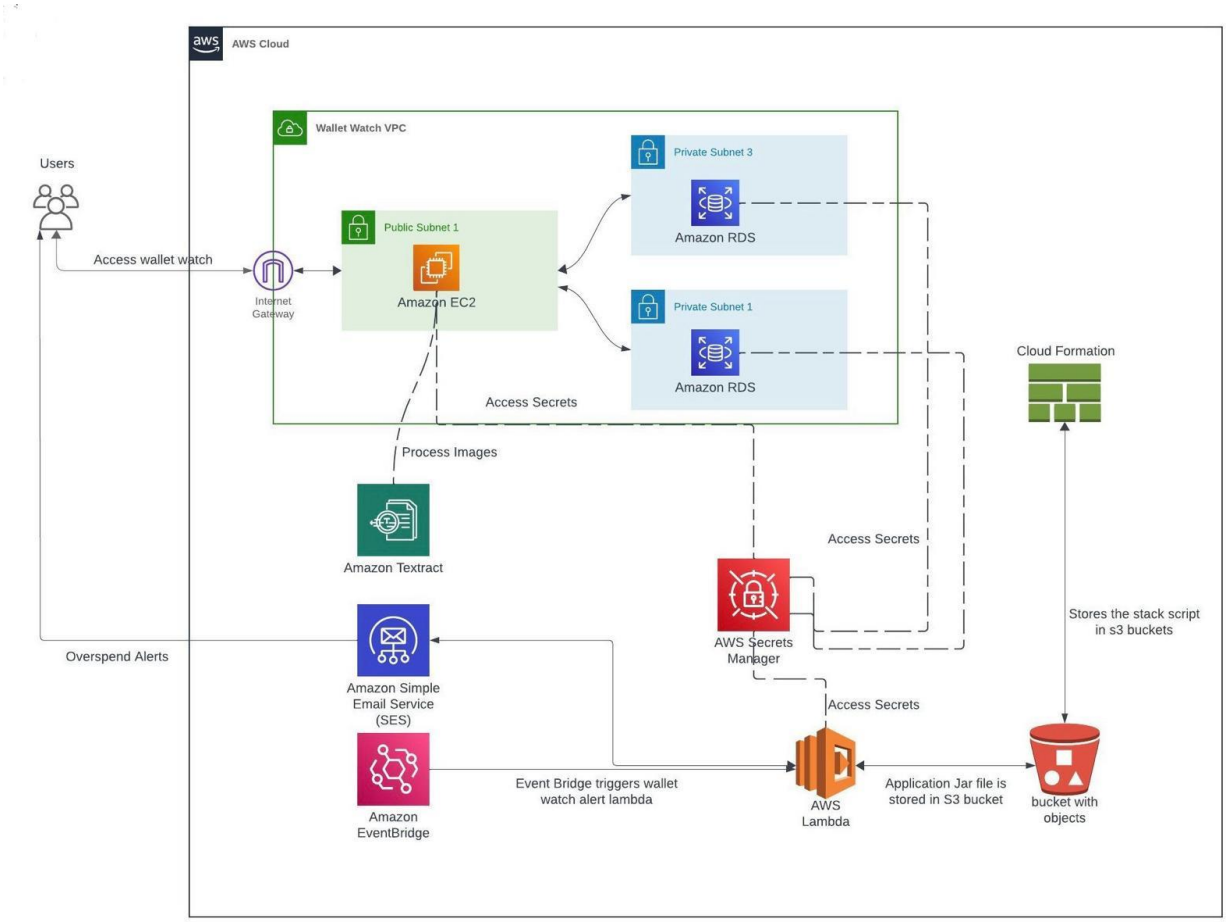

Figure 1: Wallet watch architecture

Architecture Choice and Potential Improvements

The chosen architecture aims to balance scalability, security, and operational efficiency. By leveraging managed services and serverless computing, the architecture reduces the operational burden and allows for a focus on application development and user experience.

However, potential improvements could include exploring the use of

1. **Amazon Elastic Kubernetes Service (EKS)** for better orchestration of containerized applications.
2. **Amazon CloudWatch and AWS X-Ray** for enhanced monitoring and logging capabilities.

These enhancements would further optimize application performance, monitoring, and scalability, adapting to the evolving needs of users and the application itself.

Security:

Our application leverages **AWS Virtual Private Cloud** (VPC) to ensure a secure networking environment, with the application frontend being the only component directly exposed to users. The backend, including the AWS RDS database, resides within a private subnet, accessible exclusively by the EC2 instance and designated AWS components. This setup significantly limits potential attack surfaces by restricting direct internet access to sensitive components.

For authentication, the application employs **JWT tokens**, which provide a secure and efficient way to transmit information between parties as a JSON object. This method enhances the security of user sessions by ensuring that each request is verified and authenticated.

AWS Secrets Manager is utilized for managing sensitive data, such as database credentials and API keys. By centralizing the storage of application secrets, we enhance security through encryption, access control, and audit capabilities.

Vulnerabilities and Potential Improvements

While our current security measures are robust, continuous improvement is essential to address emerging threats. Potential vulnerabilities include:

Exposure to DDoS attacks: Given the internet-facing nature of the EC2 instance, it's susceptible to Distributed Denial of Service (DDoS) attacks, which could compromise availability.

Insider Threats: Mismanagement of access rights within the AWS environment could lead to unauthorized access.

To mitigate these and other vulnerabilities, I recommend:

AWS Shield: For advanced protection against DDoS attacks.

Enhanced Access Control: Regular audits of IAM roles and policies to ensure that access is granted based on the principle of least privilege.

Cost of Private Cloud Replication:

Below, I provide a rough estimate for setting up a small to medium-sized private cloud environment and the equivalent AWS components as a comparison.

Private Cloud Infrastructure Cost Estimate:

Virtualization Software: Solutions like VMware vSphere or Microsoft Hyper-V could cost anywhere from $1,000 to $5,000 for licensing, depending on the features and the number of CPUs [17].

High-Performance Servers: For compute and storage, assuming a medium-sized setup, the cost could range from $5,000 to $20,000 per server. A minimal setup might require at least 3 servers for redundancy and load balancing, totaling $15,000 to $60,000 [18].

Networking Hardware: Switches, routers, and firewall appliances could cost between $10,000 and $30,000, depending on the network's complexity and security requirements [19].

Software Licenses: Database management, monitoring tools, and security software could add an additional $2,000 to $10,000 (approx).

Security Appliances: Hardware and software for firewalls, intrusion detection/prevention can range from $5,000 to $20,000 [20].

AWS Cloud Service Cost Estimate (Monthly):

EC2 Instances: Depending on the instance type (e.g., t3.medium for general purposes), costs can range from $50 to $200 per instance per month [16].

RDS Database: A db.t3.medium MySQL instance could cost approximately $100 to $200 per month, depending on the region and storage requirements [16].

S3 Storage: Pricing can vary widely based on the amount of data stored and accessed, but a modest bucket might cost around $20 to $50 per month [16].

Lambda: The cost depends on the number of requests and execution time but could range from $20 to $100 for moderate usage [16].

Secrets Manager: Roughly $0.40 per secret per month, plus $0.05 per 10,000 API calls. For a small number of secrets, this might amount to a few dollars per month [16].

VPC, CloudFormation, and CloudWatch: These services come with no additional charge for their basic usage.

Comparison

A small to medium-sized private cloud could require a significant upfront investment of $32,000 to $125,000 for hardware and software, plus ongoing operational costs.

In contrast, a similar setup on AWS would incur a monthly cost that could range from $200 to $750, excluding potential savings from reserved instances or cost optimization strategies.

Monitoring and Cost Management:

Although EC2 and RDS are not the most budget-friendly options within the AWS ecosystem, their cost structures are relatively transparent. Expenses associated with these services are unlikely to increase unless we decide to upgrade our resources or configurations. This predictability offers a level of financial control and planning security, allowing for more straightforward budget management.

However, it is advisable to remain vigilant when it comes to monitoring AWS Lambda, S3, and Textract.

AWS Lambda, for example, charges based on the number of requests and the duration of execution. This model, while cost-effective for applications with inconsistent workloads, can lead to significant cost surges if not properly monitored. Sudden increases in usage or inefficient code can unexpectedly inflate costs.

AWS S3's pricing is influenced by storage volume, requests, and data transfer out of the AWS environment. Without careful monitoring, high data retrieval rates or extensive data transfer activities can result in unforeseen expenses.

AWS Textract, designed to extract text and data from scanned documents, also operates on a pay-per-use basis. The costs here can escalate based on the volume of processed data and the complexity of the documents being analyzed. For services like Textract, where processing demands can fluctuate widely, close oversight is essential to prevent budget overruns.

Future Development:

Potential Features and Enhancements

A mobile application version is envisioned to provide users with enhanced accessibility and convenience, allowing them to manage their finances on-the-go.

Cloud Mechanisms for Future Development

1. For future development, particularly for the mobile application and ensuring high availability, the use of Kubernetes for container orchestration will facilitate seamless scaling and management of containerized applications [23].
2. AWS Load Balancers for traffic distribution will ensure that the application remains responsive under varying load conditions, enhancing the user experience [22].

References

[1] "70% of Americans feel financially stressed, new CNBC survey finds", *CNBC*, Available: https://www.cnbc.com/2023/04/11/70percent-of-americans-feel-financially-stressed-new-cnbc-survey-finds.html, [Accessed: April 2, 2024]

[2] "Cloud Compute Capacity - Amazon EC2", *AWS*, Available: https://aws.amazon.com/ec2/, [Accessed: April 2, 2024]

[3] "Serverless Function, FaaS Serverless - AWS Lambda", *AWS*, Available: https://aws.amazon.com/lambda/, [Accessed: April 2, 2024]

[4] "Cloud Object Storage - Amazon S3 - AWS", *AWS*, Available: https://aws.amazon.com/s3/, [Accessed: April 2, 2024]

[5] "Managed SQL Database - Amazon Relational Database Service (RDS)", *AWS*, Available: https://aws.amazon.com/rds/, [Accessed: April 2, 2024]

[6] "What is Amazon VPC? - Amazon Virtual Private Cloud", *AWS Docs*, Available: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html, [Accessed: April 2, 2024]

[7] "What is Amazon Textract? - Amazon Textract", *AWS Docs*, Available: https://docs.aws.amazon.com/textract/latest/dg/what-is.html, [Accessed: April 2, 2024]

[8] "Event Listener - Amazon EventBridge - AWS", *AWS Docs*, Available: https://aws.amazon.com/eventbridge/, [Accessed: April 2, 2024]

[9] "Bulk Cloud Email Service - Amazon Simple Email Service - AWS", *AWS*, Available: https://aws.amazon.com/ses/, [Accessed: April 2, 2024]

[10] M. Mickiewicz, "AWS Elastic Beanstalk vs EC2: A Detailed Comparison", *SitePoint*, (30 Mar. 2023), Available: https://www.sitepoint.com/aws-elastic-beanstalk-vs-ec2/, [Accessed: April 2, 2024]

[11] Cloud Academy Team, "Amazon RDS vs DynamoDB: 12 Differences You Should Know", *Cloud Academy*, (28 Jul. 2023), Available: https://cloudacademy.com/blog/amazon-rds-vs-dynamodb-12-differences/, [Accessed: April 2, 2024]

[12] "What is the differences between VPC endpoint and gateway endpoint?", *Medium*, (4 Apr. 2020), Available: https://gkzz.medium.com/what-is-the-differences-between-vpc-endpoint-gateway-endpoint-ae97bfab97d8, [Accessed: April 2, 2024]

[13] "ETL Service - Serverless Data Integration - AWS Glue" *AWS*, Available: https://aws.amazon.com/glue/ [Accessed: April 2, 2024]

[14] "Deployment & Delivery Models - CSCI4145 CSCI5409 - Cloud Computing (Sec 01) - 2023/2024 Winter" *Brightspace*, Available: https://dal.brightspace.com/d2l/le/content/310726/ [Accessed: April 2, 2024]

[15] "Cloud Password Management, Credential Storage - AWS Secrets Manager" *AWS*, Available: https://aws.amazon.com/secrets-manager/ [Accessed: April 2, 2024]

[16] "AWS Product and Service Pricing" *AWS*, Available: AWS Product and Service Pricing | Amazon Web Services, [Accessed: April 2, 2024]

[17] "Virtualization Cost Comparison" *WintelGuy*, Available: https://wintelguy.com/2024/virtualization-cost-comparison.html, [Accessed: April 2, 2024]

[18] "What's the Cost of a Server for Small Business" *Servermania*, Available: https://www.servermania.com/kb/articles/how-much-does-a-server-cost-for-a-small-business, [Accessed: April 2, 2024]

[19] "Computer Networking Products & Equipment", *cdw*, Available: https://www.cdw.com/category/networking/?w=R, [Accessed: April 2, 2024]

[20] "WEB SECURITY APPLIANCE Price - Cisco Global Price List", *itprice*, Available: WEB SECURITY APPLIANCE Price - Cisco Global Price List (itprice.com), [Accessed: April 2, 2024]

[21] "APM Tool - Amazon CloudWatch - AWS", *AWS*, Available: APM Tool - Amazon CloudWatch - AWS, [Accessed: April 2, 2024]

[22] "Load Balancer - Elastic Load Balancing (ELB) - AWS", *AWS,* Available: Load Balancer - Elastic Load Balancing (ELB) - AWS (amazon.com), [Accessed: April 2, 2024]

[23] "Kubernetes on AWS", *AWS,* Available: Kubernetes on AWS | Amazon Web Services, [Accessed: April 2, 2024]