CSCI 5902 Advanced Cloud Architecting (Fall 2023)

Architecting Applications on AWS-Term Assignment

Prepared by:
Vikram Venkatapathi - B00936916

Master of Applied Computer Science (Jan'23 - May'24)
Faculty of Computer Science
Dalhousie University

My GitHub Repository link :
`https://github.com/VikramVenkatapathi/e-commerce-site`

Source GitHub Repository link :
`https://github.com/dinushchathurya/e-commerce-site`

Hosted application's public URL : **`http://react-app-v1-env.eba-swq9qbdg.us-east-1.elasticbeanstalk.com/`**

Contents

Architecting e-commerce-site Application on AWS

1. Introduction

I've picked an open-source application for my project – the "E-commerce Site"[1] developed by Dinush Chathurya. You can find the repository here. It's gained traction with 105 commits and 132 stars, making it a well-regarded project in the developer community. This choice is pretty exciting for a couple of reasons. First off, it's built with technologies like React, React Router, and React components, which are widely used and appreciated in modern web development.

1.1. Significance of Hosting on AWS

The decision to host the "E-commerce Site" on AWS is rooted in the platform's unparalleled scalability and reliability[2]. AWS offers a suite of services tailor-made for modern applications, ensuring seamless handling of growing traffic, enhanced security, and optimal performance. This strategic choice empowers the project to leverage AWS's robust infrastructure for a hosting solution that aligns with the demands of a dynamic e-commerce application.

1.2. Scope and Objectives

This project extends beyond conventional hosting, delving into the augmentation and enhancement of the existing e-commerce application. By leveraging AWS services across various categories, the objective is to optimize performance, fortify security measures, streamline operational processes, and create an environment where the e-commerce site can organically evolve and adapt. The overarching goal is to transform the application into a feature-rich, secure, and scalable platform, showcasing the prowess of AWS in modern application hosting[3].

Furthermore, the project lays the groundwork for future expansions, suggesting the incorporation of additional features. These potential enhancements include integrating payment processing, refining filtering options, introducing user-centric functionalities like reviews and star products, and empowering sellers with advanced tools for effective product management. The intention is to demonstrate not only the immediate improvements but also the extensibility of the application, opening avenues for continuous evolution and enrichment.

2. Application Selection

The selected open-source application is the "E-commerce Site" available at the provided GitHub link . This choice aligns with a personal preference for developing and hosting web applications using React[4], a technology I'm familiar with from multiple projects. The decision to work with this application is rooted in its popularity, as evidenced by the substantial number of stars (132) and commits (105) it has received. Initially offering basic functionalities like product listing, viewing descriptions, cart management, and checkout, the application provided a solid foundation.

Beyond its popularity, the decision to choose this application was bolstered by its relevance to my skill set and project goals. The inclusion of features like user registration and login was an attractive addition, aligning with my objective to extend the application's functionality. This choice sets the stage for a project that combines familiarity, popularity, and the opportunity for personalization and enhancement.

3. AWS Service Selection

3.1. Compute

3.1.1. Amazon Elastic Beanstalk

Factor	**Amazon-Elastic Beanstalk (chosen)**	**Amazon EC2(Alternative)**	**Amazon ECS/EKS(Alternative)**
Abstraction	Fully abstracts[5] infrastructure complexities.	Requires manual provisioning and configuration of instances.[6]	Provides container orchestration but introduces more complexity.[7]
Ease of Use	Simplifies deployment, scaling, and management.[8]	Requires manual setup and configuration, more complex.[9]	Requires containerization knowledge, additional setup.[10]
Scaling	Auto-scales based on demand with minimal intervention.[11]	ASG[12] can be used for scaling instances.	Offers container auto-scaling but adds complexity.[13]
Cost	Lower operational costs due to automation and abstraction.[14]	Costs may be higher due to manual management requirements.[15]	Costs can vary; introduces costs for container orchestration.[16]
Flexibility	Less flexibility as infrastructure details are abstracted.	Provides full control over infrastructure details.	Offers containerization for flexibility but adds complexity.
Security	Simplified security management, fewer opportunities for misconfigurations.[17]	Requires meticulous security configurations.[18]	Security depends on proper containerization and orchestration.[19,20]
Deployment Time	Faster deployment with automated processes.	Deployment time may be longer due to manual steps.	Deployment time may vary, influenced by containerization.

Table 3.1.: Comparison of Amazon Elastic Beanstalk, Amazon EC2, and Amazon ECS/EKS

Elastic Beanstalk aligns seamlessly with the project's objective— "assignment is intended to test my cloud architecting skills, so focus on the architecture, not the application's functionality or the codebase." While EC2 provides more control over infrastructure, it demands more manual effort and introduces complexities that are unnecessary for the project's scope. ECS/EKS, although robust for container orchestration, is considered overkill for the simplicity of the application, adding complexity without substantial benefits.

In summary, Elastic Beanstalk strikes a balance between simplicity and control, optimizing for ease of use, scalability, and cost-effectiveness—making it the most suitable choice for the specified project requirements.

3.1.2. AWS Lambda (with API Gateway)

Factor	**AWS Lambda (Chosen)**	**AWS Step Functions (Alternative)**	**Amazon ECS Tasks (Alternative)**
Cost	Cost-efficient, charged per execution[21].	Incurs costs based on state transitions and executions.[22]	Costs associated with running and managing container instances.[23]
Performance	Swift execution based on events, suitable for short-term tasks.[24]	Manages state transitions, suitable for workflows.[25]	Containerized tasks, suitable for longer-running processes.[26]
Security	Secure environment with proper IAM configurations.[27]	IAM roles for state machine execution.[28]	Security configurations required for running tasks in containers.[29]
Scalability	Automatic scaling based on event triggers.[30]	Scales based on state transitions and workflow complexity.[31]	Manual or automatic scaling of ECS tasks based on demand.[32]

Table 3.2.: Comparison of AWS Lambda with AWS Step Functions and Amazon ECS Tasks

Lambda is a cost-efficient and agile choice for short-term, event-driven tasks like user login and registration. Its serverless model ensures minimal costs and quick execution, making it well-suited for these activities. Step Functions are designed for orchestrating workflows with multiple steps and state transitions. While suitable for complex workflows, they introduce additional costs and complexity, making them less ideal for simple, short-term tasks like login and registration. ECS Tasks involve containerized processes, making them suitable for longer-running activities. However, for short-term tasks like login and registration, the containerization introduces unnecessary complexity and costs compared to the serverless model of Lambda.

In summary, AWS Lambda is the preferred choice for short-term, event-driven tasks in my application due to its cost-effectiveness, swift execution, security features, and automatic scalability. It aligns well with the project's goal of handling user login and registration efficiently.

3.2. Storage

3.2.1. Amazon S3 Selection: Optimal Storage for My Application

Amazon S3[33] stands out as the optimal storage solution for my application's diverse needs. The decision to leverage S3 for image storage aligns with the cost-effective and scalable nature of my cloud infrastructure. By specifically employing the Standard-Infrequent Access(IA)[34] storage class for images, we ensure a balance between accessibility and cost efficiency. Given that my React application utilizes images that aren't accessed frequently, Standard-IA provides a solution that minimizes storage costs while maintaining high data durability and availability. S3's versatility further accommodates various data types, allowing us to seamlessly store log files and other objects that don't require constant access. The adaptability and scalability of S3 make it a pivotal component of my storage strategy, contributing to the overall efficiency and performance of my application in the AWS environment.

3.2.2. Considered Alternative:

In considering alternative storage solutions,

- **Elastic File System (EFS):** Elastic File System (EFS) [35] did not align with the specific requirements of my application. EFS, designed for shared file access, introduces unnecessary complexity to my architecture as my application does not necessitate shared file systems. The simplicity of S3 suffices for my storage needs without introducing the added features provided by EFS. This decision is driven by the fact that my application doesn't require shared file access, and opting for EFS would be an over-engineering choice. By strategically focusing on S3, we maintain a storage infrastructure that caters precisely to my application's demands while avoiding unnecessary overhead associated with alternative solutions like EFS.
- **Elastic Block Store (EBS):** Elastic Block Store (EBS)[36], while a robust solution for block-level storage, was not deemed necessary for my use case. Since my critical data is stored in DynamoDB, and we can leverage the volatile root volume of EC2 instances for temporary data, the need for dedicated EBS volumes for each instance is eliminated. The strategic decision to focus on S3 ensures a storage infrastructure precisely tailored to my application's demands, avoiding unnecessary complexity and overhead associated with alternative solutions like EFS and EBS.

3.3. Database

3.3.1. DynamoDB selection: Optimal choice for simple registration data

The strategic choice of DynamoDB[37] as my database solution for registration data is rooted in a meticulous evaluation of key factors, including cost, performance, security, and scalability. DynamoDB's serverless, NoSQL architecture ensures a cost-effective solution by eliminating the need for provisioning and maintaining database instances. The pay-as-you-go pricing model aligns seamlessly with my usage patterns, providing a cost-efficient approach for managing registration-user information.

Additionally, DynamoDB's exceptional performance in handling high-velocity data and its ability to scale effortlessly make it a compelling choice for my application.

From a security standpoint, DynamoDB's built-in security features[38], including encryption at rest and in transit, contribute to a robust data protection strategy. The fine-grained access control mechanisms allow us to define and enforce access policies, ensuring the confidentiality and integrity of my user data. DynamoDB's scalability is a critical factor, as it accommodates my application's growth without compromising performance. The ability to handle fluctuations in demand seamlessly ensures a responsive user experience.

3.3.2. Considered Alternatives: RDS and Why DynamoDB Prevailed

In contrast to DynamoDB, Amazon Relational Database Service (RDS)[39] was evaluated based on factors such as cost, performance, security, and scalability. While RDS is renowned for its reliability and robust relational database capabilities, the associated costs[41], especially in terms of provisioning and maintaining instances, weighed against its adoption. DynamoDB's serverless model not only proves cost-effective[42] but also enhances performance with its NoSQL design, addressing the specific simplicity and efficiency requirements of my registration data.

From a security perspective, both RDS and DynamoDB provide robust measures, but DynamoDB's managed security features, coupled with its seamless integration with AWS Identity and Access Management (IAM), streamline access control and auditing. Additionally, DynamoDB's inherent scalability[40] without the need for manual intervention aligns well with my application's anticipated growth, offering a nimble solution that adapts effortlessly to changing workloads. The decision to opt for DynamoDB is a testament to its holistic alignment with my cost-efficiency, high performance, security, and scalability objectives.

3.4. Networking and content delivery

3.4.1. Virtual Private Cloud(VPC)

In my networking and content delivery architecture, the use of a Virtual Private Cloud (VPC)[43] to host the Beanstalk environment forms the cornerstone of my strategy, optimizing for cost, security, and scalability. The VPC allows us to logically isolate my application resources, providing a secure and dedicated environment in the cloud. This segregation not only enhances security but also enables efficient resource utilization, contributing to a cost-effective infrastructure.

3.4.2. Security Group(SG)

The employment of Security Groups[44] further fortifies my network architecture by controlling traffic to EC2 instances. Security Groups act as virtual firewalls, regulating inbound and outbound traffic based on defined rules. This ensures that only authorized communication occurs, mitigating potential security risks and maintaining the integrity of my application. This approach aligns with my commitment to robust security practices, safeguarding my infrastructure from unauthorized access and potential threats.

3.4.3. Auto Scaling Group(ASG)

The incorporation of Auto Scaling[45] configuration for the Beanstalk environment, coupled with the use of a Load Balancer, exemplifies my commitment to scalability and availability. Auto Scaling dynamically adjusts the number of instances based on demand, optimizing resource utilization and mitigating costs during periods of lower demand. The Load Balancer[46] distributes incoming traffic across multiple instances, enhancing the fault tolerance of my application and ensuring a consistent user experience. This dynamic combination of Auto Scaling and Load Balancer supports my scalability objectives, allowing the application to seamlessly handle varying workloads.

3.4.4. API Gateway

Additionally, the integration of API Gateway[47] in my React app to invoke the Lambda function optimizes content delivery and enhances the efficiency of my serverless architecture. This approach ensures that my application remains responsive and can efficiently handle requests from the React app, contributing to improved performance and a more seamless user experience.

3.5. Management & Governance: Leveraging AWS Services for Enhanced Monitoring, Incident Response, and Cost Management

my management and governance strategy revolves around the effective utilization of AWS services, aimed at monitoring application activity, ensuring robust incident response, and optimizing cost management.

3.5.1. CloudWatch for Comprehensive Application Monitoring

- **Implementation**: CloudWatch[48] is employed to monitor the activity of my applications comprehensively. Lambda functions, integral to my serverless architecture, have dedicated log groups in CloudWatch, providing real-time insights into their execution and performance.
- **Benefits**: This approach enhances my ability to proactively identify and address issues, ensuring optimal performance. CloudWatch enables us to set custom metrics and alarms[49], allowing us to detect anomalies or performance deviations promptly.

3.5.2. Alarming System for Incident Response

- **Implementation**: Alarms are configured within the Beanstalk environment[50] to notify us of any abnormal behavior or potential issues. Additionally, log data from both Lambda functions and the Beanstalk environment is stored in S3, facilitating thorough post-incident analysis and troubleshooting.
- **Benefits**: The integrated alarming system serves as a crucial component of my incident response strategy. Immediate notifications enable prompt action, helping us address and resolve potential issues before they impact the user experience. Storing logs in S3 ensures a comprehensive audit trail for thorough post-incident analysis.

3.5.3. Cost Explorer and Cost and Usage Report for Financial Visibility

- **Implementation**: Cost Explorer[51] and Cost and Usage Report[52] tools are employed to analyze the details of the cost of services. These tools provide a granular breakdown of expenses, aiding in identifying cost drivers and optimizing resource allocation.
- **Benefits**: This approach contributes to efficient financial management. By understanding the cost implications of each service, we can make informed decisions on resource optimization, ensuring that we stay within budget and maintain cost-effectiveness.

3.5.4. AWS Budgets for Proactive Cost Management (Not Implemented)

- **Intention**: While AWS Budgets[53] were considered for proactive cost management, unfortunately, due to limited permissions in the lab account, this feature could not be implemented.
- **Benefits (if implemented)**: AWS Budgets would have allowed us to set custom cost and usage thresholds, receiving alerts when expenditures approached or exceeded predefined limits. This proactive approach to cost management could have provided enhanced control over my financial resources.

In summary, my management and governance strategy leverages AWS services to ensure robust monitoring, incident response, and cost optimization. CloudWatch, alarming systems, and cost analysis tools collectively contribute to the reliability, security, and efficiency of my AWS environment. The intended use of AWS Budgets, if realized in the future, would further enhance my proactive cost management capabilities.

3.6. Security: Safeguarding Web Applications and Data Integrity

3.6.1. Web Application Firewall (WAF) for Application-Level Security

- **Implementation**: WAF[54] is strategically implemented to protect my web applications from common web exploits and malicious traffic. By defining and deploying rules, WAF helps safeguard my applications against various layer 7 attacks.
- **Benefits**: This proactive approach enhances the security posture of my applications by preventing common web vulnerabilities, including SQL injection[55] and cross-site scripting (XSS).[56] WAF acts as a shield at the application level, ensuring the integrity and availability of my web services.

3.6.2. SSL/TLS Encryption for Securing Data in Transit

- **Intention**: Although the acquisition of an SSL/TLS certificate[57] for enabling HTTPS[58] was intended, it couldn't be realized due to the absence of a proper domain.
- **Benefits (if implemented)**: HTTPS provides a secure communication channel over the internet by encrypting data in transit. This ensures the confidentiality and integrity of user data

during transmission, thwarting potential eavesdropping and tampering attempts. The intended implementation of SSL/TLS would have contributed significantly to the overall security of my web applications.

3.6.3. AWS Key Management Service (KMS) for Data-at-Rest Security

- **Implementation**: KMS[59] is utilized to enforce server-side encryption (SSE-KMS)[60] for protecting app images stored in S3. SSE-KMS employs AWS-managed keys, ensuring that data at rest is encrypted and can only be accessed by authorized entities with the necessary permissions.
- **Benefits**: KMS adds an additional layer of security by encrypting app images at rest. This mitigates the risk of unauthorized access to sensitive data, providing a robust defense against potential data breaches. The use of KMS aligns with best practices for securing data stored in S3, bolstering the overall security of my application infrastructure.

In summary, my security measures address different facets of the application environment. WAF protects against web-based attacks, SSL/TLS (if implemented) ensures secure data transmission, and KMS safeguards app images at rest. While the intended SSL/TLS implementation was hindered by domain constraints, the overall security strategy reflects a comprehensive approach to protect both the application layer and data integrity. These measures collectively contribute to a secure and resilient architecture, aligning with best practices for AWS security.

In crafting my AWS architecture, we strategically employed a diverse set of AWS services across storage, database, networking, and management categories. Amazon S3 serves as my primary storage solution, offering scalability and cost-effectiveness, with specific use of Standard-IA for infrequently accessed images. DynamoDB efficiently manages my registration data, aligning with my application's simplicity and scalability requirements. Networking and content delivery are optimized through the implementation of VPC, Auto Scaling, and Load Balancer. For management and governance, Cloud-Watch monitors application activity, while Cost Explorer and Cost and Usage Report enhance cost analysis. This thoughtful service selection ensures a well-rounded, secure, and scalable infrastructure that meets the varied needs of my application.

4. Architecture Design

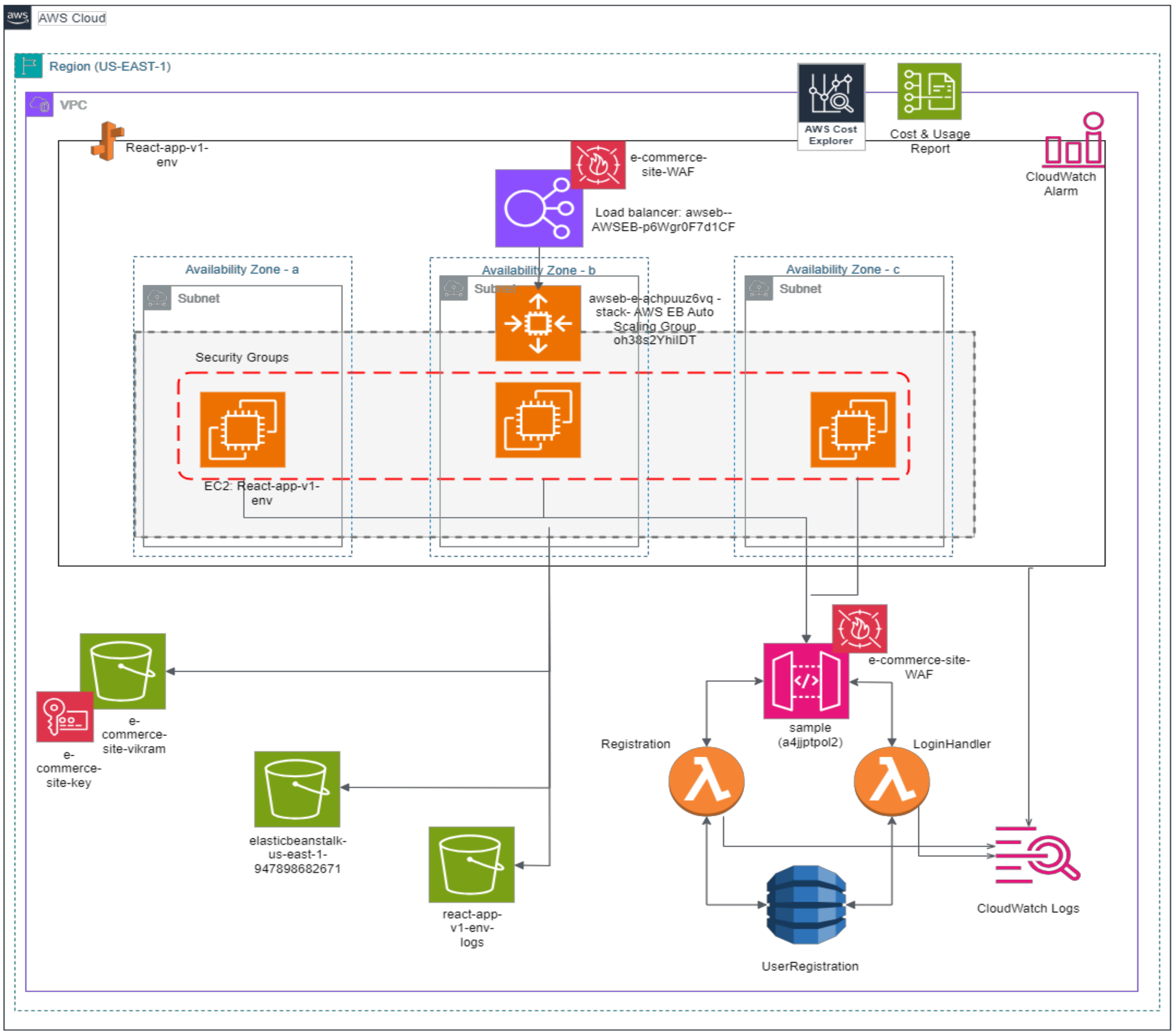


Figure 4.1.: ***Architecture Diagram of my AWS application***

Hosted application's public URL: E-commerce-site

5. Adherence to AWS Well-Architected Framework

5.1. Security pillar

5.1.1. Storage Level (S3 and KMS)

- **Data Encryption at Rest (KMS with S3)**: The use of AWS Key Management Service (KMS)[59] for server-side encryption of data stored in Amazon S3 reflects a robust security measure. This aligns with the well-architected principle of encrypting sensitive information at rest to protect against unauthorized access. KMS provides centralized key management, enhancing control and auditability.

5.1.2. Application Level (WAF, LB, API Gateway)

- **Web Application Firewall (WAF)**: The implementation of WAF[54], coupled with Load Balancer (LB)[46] and API Gateway[47], establishes a robust defense mechanism at the application level. WAF mitigates common web exploits and malicious traffic, fortifying my web applications against layer 7 attacks[62]. This adherence to best practices ensures the protection of information and assets in the application layer.

5.1.3. Network and Instance Level (VPC, Subnet, Security Group)

- **Virtual Private Cloud (VPC)**: Leveraging VPC[43], along with subnets and security groups, strengthens security at the network and instance levels. VPC provides an isolated environment, ensuring a dedicated network space for my applications. Subnets[63] and Security Groups[44] further enforce access controls and traffic filtering, adhering to the principle of least privilege and minimizing the attack surface.

5.1.4. Overall Justification

The adoption of these security measures aligns with the Well-Architected Framework's Security pillar[64] by emphasizing the protection of information, systems, and assets. The strategy incorporates a defense-in-depth approach, addressing security concerns at multiple layers of the application architecture. The use of encryption at rest, application-level firewalls, and network and instance-level controls collectively contribute to a comprehensive security posture, showcasing a commitment to mitigating risks and delivering business value securely. This alignment with the Security pillar ensures that security is an integral part of the application design, providing a strong foundation for delivering business value while safeguarding critical assets and information.

5.2. Operational Excellence pillar

5.2.1. CloudWatch for Monitoring and Improvement:

- **Continuous Monitoring (Logs & Alarms)**: The use of CloudWatch[48] for logging and alarms contributes significantly to the ability to monitor systems continuously. Configuring logs and alarms for the Beanstalk environment and establishing log groups for Lambda functions exemplifies a proactive approach to monitoring system health. By continuously analyzing logs and responding to alarms, you ensure the operational health of my application, aligning with the well-architected principle of real-time visibility.

5.2.2. User Interaction for Continuous Improvement

- **User Interaction and Usage Data Collection**: To enhance operational excellence[66], you recognize the importance of allowing users to interact with the application. Collecting usage data provides valuable insights into user behavior and application performance. This data-driven approach enables continuous improvement by identifying areas for optimization, enhancing user experience, and delivering business value.

5.2.3. Overall Justification

The incorporation of CloudWatch for real-time monitoring and logs aligns with the Operational Excellence pillar[65] by fostering a culture of continuous improvement. By configuring alarms and log groups, you demonstrate a commitment to promptly respond to issues and refine system performance. Furthermore, encouraging user interaction and collecting usage data underscores a commitment to refining processes based on actual user behavior, ensuring the application evolves to meet user expectations and deliver maximum business value. This alignment showcases an operational mindset, where monitoring is not only reactive but also proactive, and where user interaction is leveraged for continual enhancement. It reflects a commitment to operational excellence, ensuring that systems are not only running effectively but are continually refined to deliver sustained business value.

5.3. Reliability pillar

5.3.1. EC2 Instances and Auto Scaling

- **Design for Failure (EC2 and Auto Scaling)**: Running one EC2[6] instance in each Availability Zone (AZ)[68] and utilizing Auto Scaling[12] to dynamically adjust resources based on CPU utilization demonstrates a commitment to designing for failure. By having instances distributed across multiple AZs and scaling up in response to demand, my architecture ensures resilience against individual instances or AZ failures. This design choice contributes to the quick recovery from infrastructure disruptions.

5.3.2. VPC and Security Groups

- **Network Level Reliability (VPC and Security Groups)**: The use of Virtual Private Cloud (VPC)[43] and Security Groups[44] contributes to network-level reliability. By securing communication channels and filtering traffic at each layer, you reduce the attack surface and enhance the reliability of my network. Specific security group configurations, such as permitting only necessary traffic on specific ports[69], contribute to a more reliable and secure architecture.

5.3.3. Serverless Architecture (Lambda, DynamoDB)

- **Guaranteed Reliability in Serverless Services**: Leveraging serverless services like Lambda[21] and DynamoDB[37] inherently provides reliability. AWS Lambda ensures availability by automatically scaling based on the number of incoming requests. DynamoDB, being a fully managed service, guarantees high availability and reliability, minimizing the need for manual intervention. This serverless approach aligns with the well-architected principle of offloading operational responsibilities to AWS, allowing you to focus on building and enhancing my application.

5.3.4. Monitoring and Quick Incident Response (CloudWatch Alarms)

- **Proactive Monitoring for Quick Response**: The implementation of CloudWatch[48] alarms, coupled with notifications to my email, signifies a proactive approach to monitoring. This allows for swift responses to potential incidents, aligning with the well-architected principle of quickly recovering from disruptions. The use of alarms[50] ensures that you can address issues promptly, minimizing downtime and maintaining a reliable system.

5.3.5. Overall Justification

- My architecture systematically addresses reliability[67] at each layer. The distribution of EC2 instances across AZs, coupled with Auto Scaling, provides resilience against infrastructure disruptions. Network-level reliability is enhanced through VPC and Security Groups, minimizing vulnerabilities and reducing the attack surface. The adoption of serverless services guarantees reliability and scalability, while proactive monitoring with CloudWatch alarms ensures quick responses to potential incidents. This comprehensive approach reflects a commitment to the reliability pillar, ensuring that my application can recover quickly from disruptions and dynamically scale to meet demand.

5.4. Performance Efficiency pillar

5.4.1. S3 Storage Efficiency (S3 - IA Tier)

- **Usage-Based Storage Class Selection**: The use of the S3 Infrequent Access (IA)[34] storage tier based on the usage report showcases an efficient approach. By analyzing the usage pattern of my application's images, you tailored the storage class to the access requirements. This not only optimizes storage costs but also ensures that resources align with the actual demand, reflecting a performance-efficient storage strategy.

5.4.2. EC2 Instance Selection (t2.micro)

- **Right-Sizing with Free-Tier Instances**: The choice of the t2.micro EC2 instance[71, available within the AWS Free Tier, demonstrates a right-sizing strategy. This not only optimizes costs for a small-scale application but also aligns with the performance requirements identified in my usage report. The decision to evaluate and potentially upgrade instance types as the application load increases reflects a dynamic approach to maintaining performance efficiency over time.

5.4.3. Auto Scaling and CloudWatch for Dynamic resource Allocation

- **Dynamic resource Allocation (Auto Scaling, CloudWatch)**: The incorporation of Auto Scaling[12] and CloudWatch[48] enhances the dynamic allocation of resources based on demand. This aligns with the principle of efficiently using computing resources to meet system requirements. Auto Scaling ensures that the application can handle varying workloads effectively, optimizing resource utilization and maintaining performance efficiency. Continuous monitoring with CloudWatch allows for data-driven decisions to improve system efficiency over time.

5.4.4. Usage Pattern Analysis and Continuous Improvement

- **Continuous Improvement Through Analysis**: While already utilizing CloudWatch[48] and Cost Usage Reports[72, the acknowledgment of the need for time to analyze usage patterns and improve systems reflects a commitment to continuous improvement. This approach ensures that my architecture evolves based on real data, adapting to changing demands and embracing technological advancements, as outlined in the Performance Efficiency pillar[70].

5.4.5. Overall Justification

My architecture strategically addresses performance efficiency[70] at each layer. The tailored selection of S3 storage classes based on usage patterns optimizes costs. The use of Free-Tier instances ensures cost-effective computing resources, and the dynamic allocation of resources through Auto Scaling and CloudWatch[48] aligns with varying workloads. The focus on continuous improvement, driven by usage pattern analysis, further ensures that my architecture remains efficient over time and evolves with the changing needs of my application. This comprehensive strategy reflects a commitment to achieving and maintaining high-performance efficiency.

5.5. Cost Optimization pillar

5.5.1. Cost Visibility and Analysis (Cost Explorer, Cost and Usage Report)

- **Proactive Cost Management**: The utilization of Cost Explorer[74] and Cost and Usage Reports[72] showcases a proactive approach to cost management. Regularly analyzing service costs provides transparency into resource consumption and allows you to make informed decisions. Even with current costs at zero due to the use of Free Tier options, the practice of monitoring and analysis prepares you for effective cost optimization as my application scales.

5.5.2. Auto Scaling Group (ASG) for Resource Efficiency

- **Preventing Idle Resources (Auto Scaling Group)**: The implementation of Auto Scaling Groups[12] aligns with the principle of preventing idle running resources. By dynamically adjusting the number of instances based on demand, you avoid unnecessary costs associated with idle or underutilized resources. This ensures that my infrastructure scales efficiently to handle varying workloads, optimizing resource usage and reducing costs.

5.5.3. Leveraging Managed Services

- **Shift of Responsibility to Managed Services**: The adoption of managed services[75], such as AWS Lambda[24] and DynamoDB[37, aligns with cost optimization principles. By leveraging managed services, you shift the operational responsibility to AWS, reducing the need for manual maintenance and lowering associated costs. This approach allows you to access advanced technologies at a lower cost while benefiting from AWS's economies of scale and innovations.

5.5.4. Overall Justification

My architecture systematically addresses cost optimization[73] at each layer. Through the use of Cost Explorer and Cost and Usage Reports, you establish a foundation for informed decision-making regarding resource consumption. The implementation of Auto Scaling Groups ensures that resources are utilized efficiently, preventing unnecessary costs. Leveraging managed services not only enhances operational efficiency but also aligns with the principle of avoiding expenses associated with maintaining infrastructure. This comprehensive strategy reflects a commitment to cost optimization, ensuring that my architecture remains efficient and cost-effective as my application evolves.

5.6. Sustainability pillar

5.6.1. Load Distribution Efficiency (Elastic Load Balancer)

- **Environmental Impact Reduction**: The use of Elastic Load Balancer (ELB)[46] contributes to the sustainability pillar by efficiently distributing the load across instances. By evenly distributing traffic, ELB enhances resource utilization and reduces the need for over-provisioning, leading to lower energy consumption. This load distribution efficiency aligns with sustainable practices, ensuring optimal use of resources.

5.6.2. Auto Scaling Group (ASG) for Resource Efficiency

- **Idle Resource Prevention**: Auto Scaling Groups (ASG)[12] play a crucial role in sustainability by preventing the idle running of resources. By dynamically adjusting the number of instances based on demand, ASG ensures that only necessary resources are active, reducing overall energy consumption. This dynamic resource allocation aligns with sustainability principles, promoting efficient resource usage.

5.6.3. Leveraging Managed Services for Multi-Tenant Control Planes

- **Multi-Tenant Environmental Impact Distribution**: The use of managed services, such as AWS Lambda[24] and DynamoDB[37], contributes significantly to the sustainability pillar. Managed services distribute the environmental impact of the service across many users due to their multi-tenant control planes[76]. This shared infrastructure model allows AWS to optimize resource usage at scale, resulting in reduced overall environmental impact. Leveraging managed services aligns with sustainability goals by maximizing resource efficiency and minimizing the carbon footprint[77] associated with individual instances.

5.6.4. Overall Justification

My architecture showcases a commitment to sustainability across various layers. The use of Elastic Load Balancer[46] and Auto Scaling Group[12] ensures efficient resource utilization, reducing energy consumption and promoting sustainable practices. Additionally, leveraging managed services further enhances the sustainability[78] of my architecture. The multi-tenant control planes of these services enable AWS to optimize environmental impact at a larger scale, aligning with the principles of shared responsibility and efficient resource usage. This comprehensive approach reflects a commitment to building a sustainable and environmentally conscious architecture.

While my architecture may not fully adhere to all the pillars at every layer, earnest efforts have been made to incorporate best architectural practices to the extent feasible. This holistic approach signifies a commitment to constructing an architecture that aligns with the principles of the AWS Well-Architected Framework. I have strived to satisfy each pillar to the best of my ability, recognizing the importance of continuous improvement and the dynamic nature of cloud architecture.

6. Implementation on AWS

6.0.1. Implementation Journey: Crafting a Robust Architecture on AWS

The initial implementation of the application focused on essential functionalities like product browsing and cart management. As the scope expanded to include user-specific features such as login and registration, the need for a database service became apparent. This led to the integration of Amazon DynamoDB to manage user registration data seamlessly. However, during this phase, a noteworthy challenge surfaced in the form of CORS errors[79] when connecting the React app to Lambda through API Gateway. This challenge was promptly addressed by configuring the appropriate headers in the API Gateway's response, ensuring smooth communication and data flow between components.

A pivotal decision in the implementation phase involved the choice of hosting service. Initially considering Amazon EC2 for its configurability, attempts to distribute traffic across instances encountered challenges with health checks by target groups. This obstacle led to a strategic pivot to AWS Elastic Beanstalk, providing a seamless deployment solution for the application. The subsequent inclusion of a Load Balancer, coupled with an Auto Scaling Group, allowed for dynamic scaling based on CPU utilization metrics[80], optimizing resource usage and enhancing overall reliability.

Security implementation posed another significant challenge, which was met with a comprehensive approach. The integration of AWS WAF's web ACL played a crucial role in filtering and protecting against web-based attacks. Additionally, stringent security measures were applied, including ensuring that objects in Amazon S3 were not publicly accessible, configuring security groups for EC2 instances to control inbound and outbound traffic, and leveraging AWS Key Management Service (KMS) for server-side encryption to protect data at rest.

Throughout the implementation process, each challenge was met with iterative problem-solving, emphasizing the importance of adaptability in crafting a resilient architecture. The transition from EC2 to Elastic Beanstalk, resolution of CORS errors[79], and the implementation of robust security measures collectively showcase the commitment to overcoming obstacles and building an architecture that aligns with best practices and principles. The implementation journey reflects a dynamic approach, ensuring the successful integration of diverse AWS services to meet the evolving needs of the application.

7. Lessons Learned

7.0.1. Navigating Cloud Architecture

1. **Complexity of Best Practices**:

   Insight: Following AWS best practices and design patterns demanded a deep understanding. Simplified documentation is essential for effective implementation.

2. T**he Enigma of Scale**:

   Insight: Even on a small scale, realizing the complexities faced by major platforms highlighted the importance of anticipating growth and planning for scalability.

3. **Continuous Improvement in Security and Cost Optimization**:

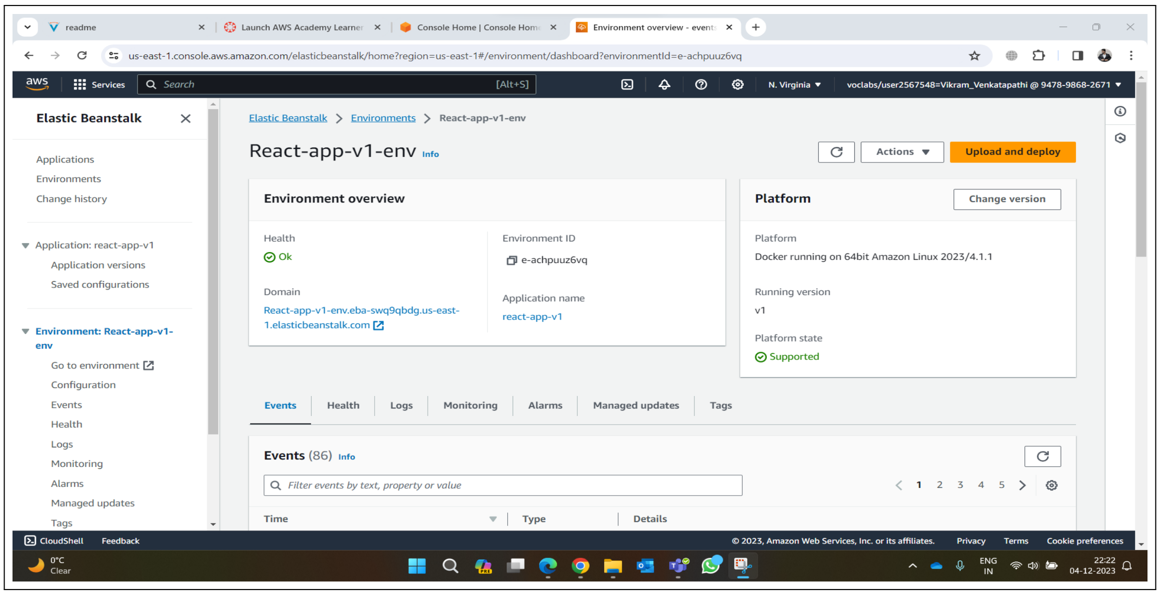
   Insight: There's always room for improvement. Regularly refining security measures and cost management strategies ensures resilience and cost-effectiveness.

4. **Complexity of IAM Environment**:

   Insight: Working in a restricted IAM environment requires a balance between security and functionality. Continuous fine-tuning is crucial.

In conclusion, the journey underscored the importance of simplicity in documentation, scalability considerations, continuous improvement, and the delicate balance between security and functionality in a restricted IAM environment.

8. Screenshots



readme | Launch AWS Academy Learner | Console Home | Console Home | Environment overview - events

us-east-1.console.aws.amazon.com/elasticbeanstalk/home?region=us-east-1#/environment/dashboard?environmentId=e-achpuuz6vq

N. Virginia | voclabs/user2567548=Vikram_Venkatapathi @ 9478-9868-2671

Elastic Beanstalk

- Applications
- Environments
- Change history
- Application: react-app-v1
  - Application versions
  - Saved configurations
- Environment: React-app-v1-env
  - Go to environment
  - Configuration
  - Events
  - Health
  - Logs
  - Monitoring
  - Alarms
  - Managed updates
  - Tags

Elastic Beanstalk > Environments > React-app-v1-env

React-app-v1-env Info

Actions | Upload and deploy

Environment overview

Health: Ok

Environment ID: e-achpuuz6vq

Domain: React-app-v1-env.eba-swq9qbdg.us-east-1.elasticbeanstalk.com

Application name: react-app-v1

Platform | Change version

Platform: Docker running on 64bit Amazon Linux 2023/4.1.1

Running version: v1

Platform state: Supported

Events | Health | Logs | Monitoring | Alarms | Managed updates | Tags

Events (86) Info

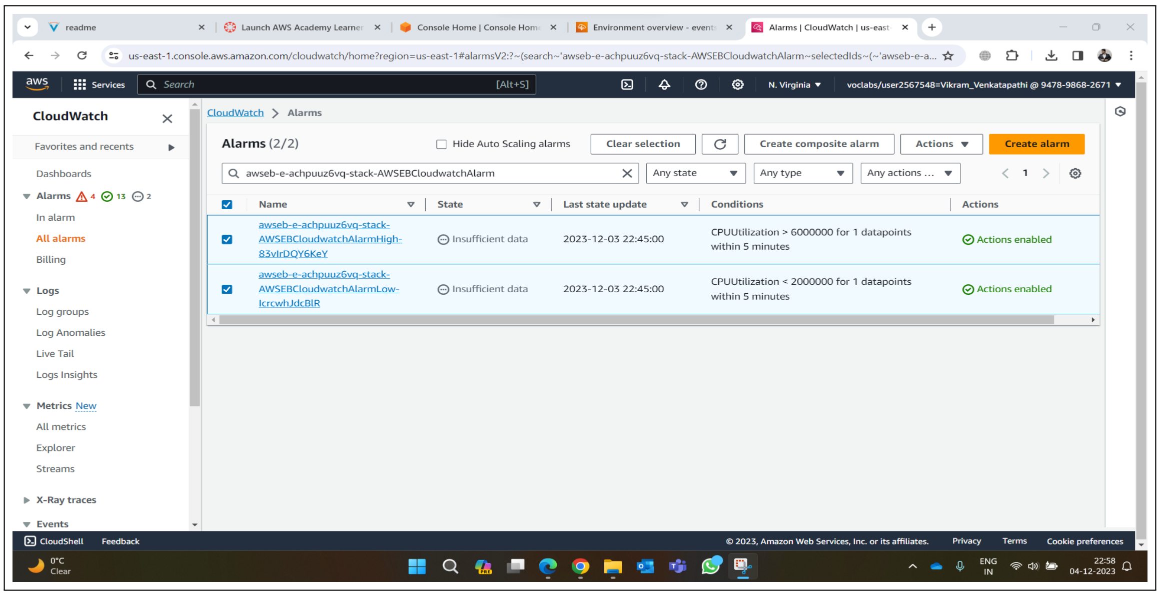
Filter events by text, property or value

Time | Type | Details

CloudShell | Feedback | © 2023, Amazon Web Services, Inc. or its affiliates. | Privacy | Terms | Cookie preferences

0°C Clear | ENG IN | 22:22 04-12-2023

Figure 8.1.: ***Beanstalk environment where the react application is hosted***



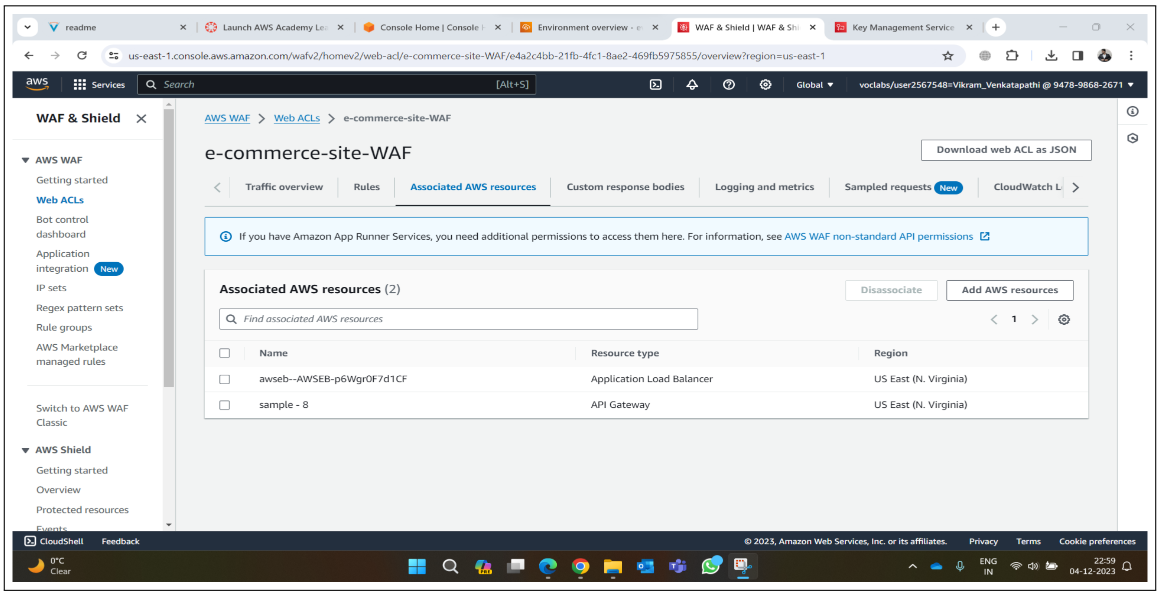
CloudWatch > Alarms

Alarms (2/2)

awseb-e-achpuuz6vq-stack-AWSEBCloudwatchAlarm

	Name	State	Last state update	Conditions	Actions
✓	awseb-e-achpuuz6vq-stack-AWSEBCloudwatchAlarmHigh-83vIrDQY6KeY	Insufficient data	2023-12-03 22:45:00	CPUUtilization > 6000000 for 1 datapoints within 5 minutes	Actions enabled
✓	awseb-e-achpuuz6vq-stack-AWSEBCloudwatchAlarmLow-IcrcwhJdcBIR	Insufficient data	2023-12-03 22:45:00	CPUUtilization < 2000000 for 1 datapoints within 5 minutes	Actions enabled

Figure 8.2.: *Cloud Watch alarm to trigger based on CPU utilization(used the default value of metric)*



AWS WAF > Web ACLs > e-commerce-site-WAF

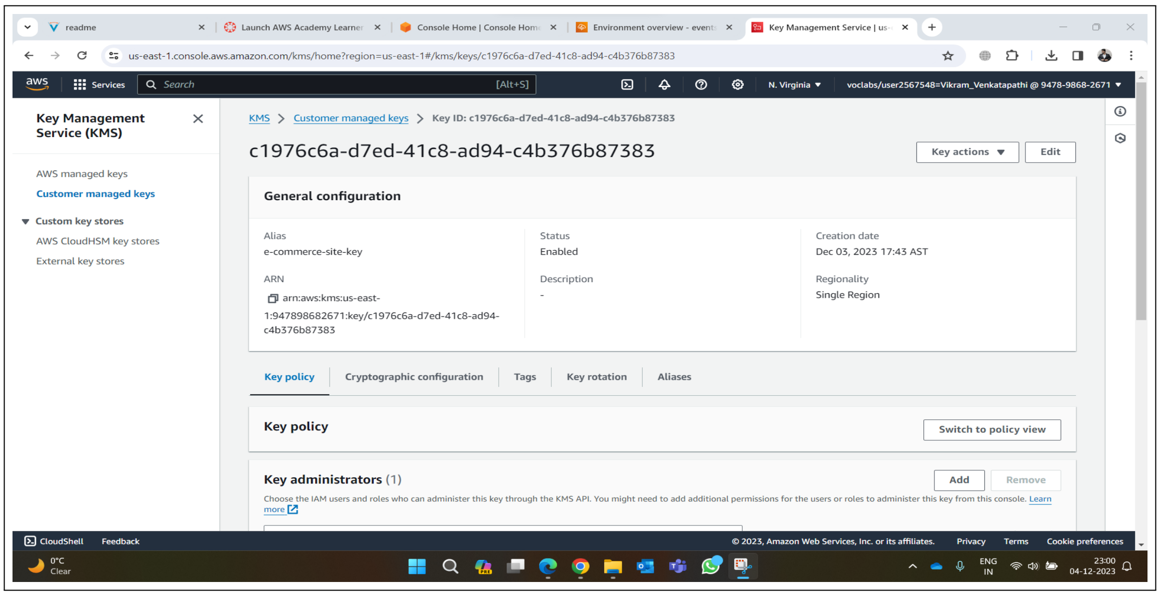
e-commerce-site-WAF

If you have Amazon App Runner Services, you need additional permissions to access them here. For information, see AWS WAF non-standard API permissions

Associated AWS resources (2)

	Name	Resource type	Region
	awseb--AWSEB-p6WgrOF7d1CF	Application Load Balancer	US East (N. Virginia)
	sample - 8	API Gateway	US East (N. Virginia)

Figure 8.3.: *WEB ACL to protect the Load Balancer and API Gateway*


us-east-1.console.aws.amazon.com/kms/home?region=us-east-1#/kms/keys/c1976c6a-d7ed-41c8-ad94-c4b376b87383

Key Management Service (KMS)

AWS managed keys
Customer managed keys
Custom key stores
AWS CloudHSM key stores
External key stores

KMS > Customer managed keys > Key ID: c1976c6a-d7ed-41c8-ad94-c4b376b87383

c1976c6a-d7ed-41c8-ad94-c4b376b87383

General configuration

Alias	Status	Creation date
e-commerce-site-key	Enabled	Dec 03, 2023 17:43 AST
ARN	Description	Regionality
arn:aws:kms:us-east-1:947898682671:key/c1976c6a-d7ed-41c8-ad94-c4b376b87383	-	Single Region

Key policy | Cryptographic configuration | Tags | Key rotation | Aliases

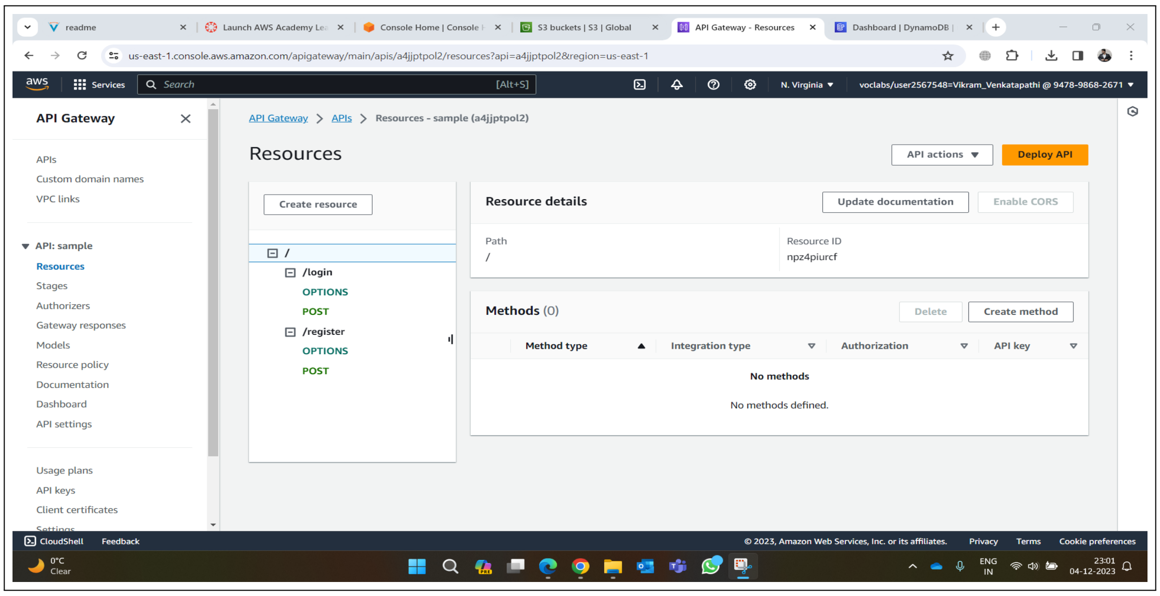
Key policy

Key administrators (1)

Choose the IAM users and roles who can administer this key through the KMS API. You might need to add additional permissions for the users or roles to administer this key from this console. Learn more

© 2023, Amazon Web Services, Inc. or its affiliates.

Figure 8.4.: ***KMS key used to encrypt images in S3 bucket***


us-east-1.console.aws.amazon.com/apigateway/main/apis/a4jjptpol2/resources?api=a4jjptpol2&region=us-east-1

API Gateway

APIs
Custom domain names
VPC links
API: sample
Resources
Stages
Authorizers
Gateway responses
Models
Resource policy
Documentation
Dashboard
API settings
Usage plans
API keys
Client certificates

API Gateway > APIs > Resources - sample (a4jjptpol2)

Resources

```
/
  /login
    OPTIONS
    POST
  /register
    OPTIONS
    POST
```

Resource details

Path	Resource ID
/	npz4piurcf

Methods (0)

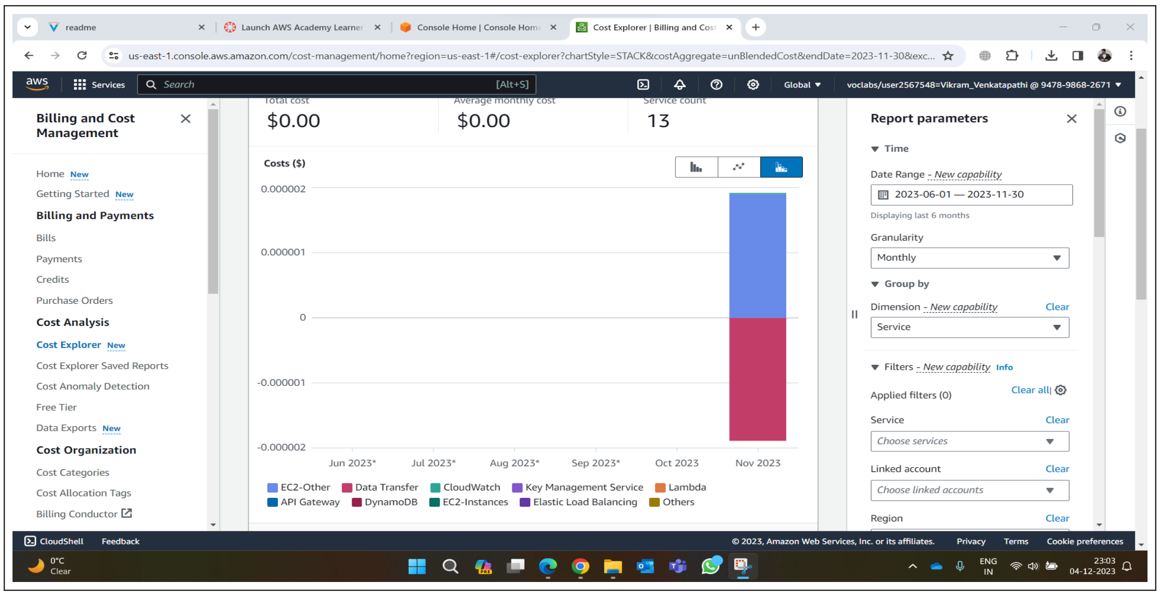
Method type	Integration type	Authorization	API key

No methods

No methods defined.

© 2023, Amazon Web Services, Inc. or its affiliates.

Figure 8.5.: ***API Gateway with routes***



us-east-1.console.aws.amazon.com/cost-management/home?region=us-east-1#/cost-explorer?chartStyle=STACK&costAggregate=unBlendedCost&endDate=2023-11-30&exc...

Total cost $0.00
Average monthly cost $0.00
Service count 13

Report parameters

Time

Date Range - New capability: 2023-06-01 — 2023-11-30
Displaying last 6 months

Granularity: Monthly

Group by

Dimension - New capability: Service

Filters - New capability

Applied filters (0)

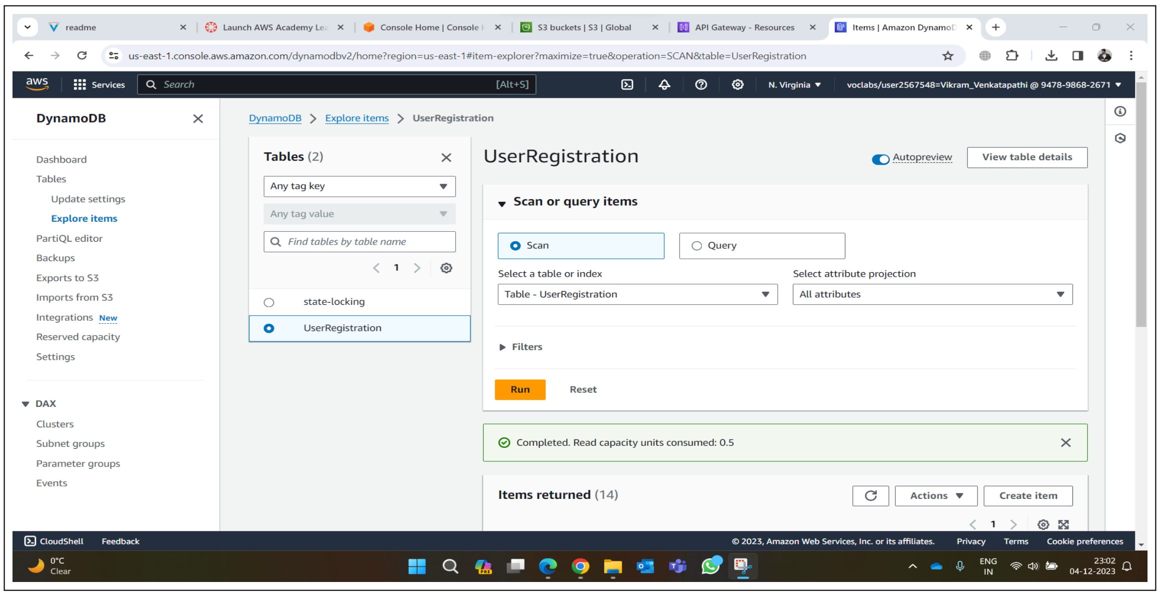
Service: Choose services

Linked account: Choose linked accounts

Region

© 2023, Amazon Web Services, Inc. or its affiliates.

Figure 8.6.: ***Cost explorer***



us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#item-explorer?maximize=true&operation=SCAN&table=UserRegistration

DynamoDB > Explore items > UserRegistration

Tables (2)

state-locking

UserRegistration

UserRegistration

Scan or query items

[✓ Scan] Query

Select a table or index: Table - UserRegistration

Select attribute projection: All attributes

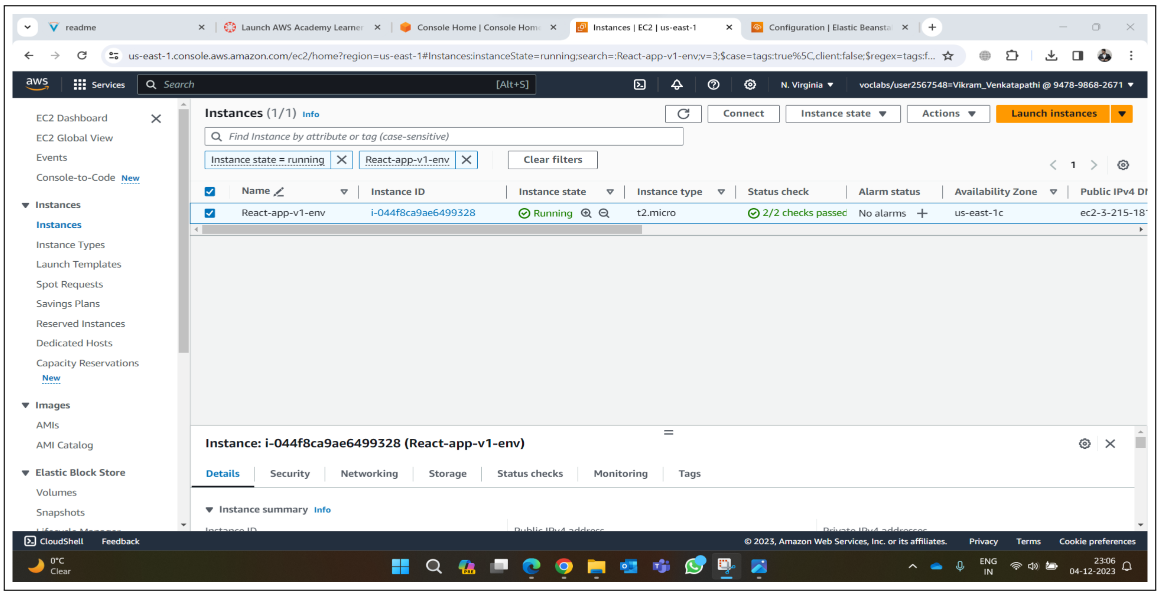
Filters

Completed. Read capacity units consumed: 0.5

Items returned (14)

© 2023, Amazon Web Services, Inc. or its affiliates.

Figure 8.7.: ***UserRegistration DynamoDB table***



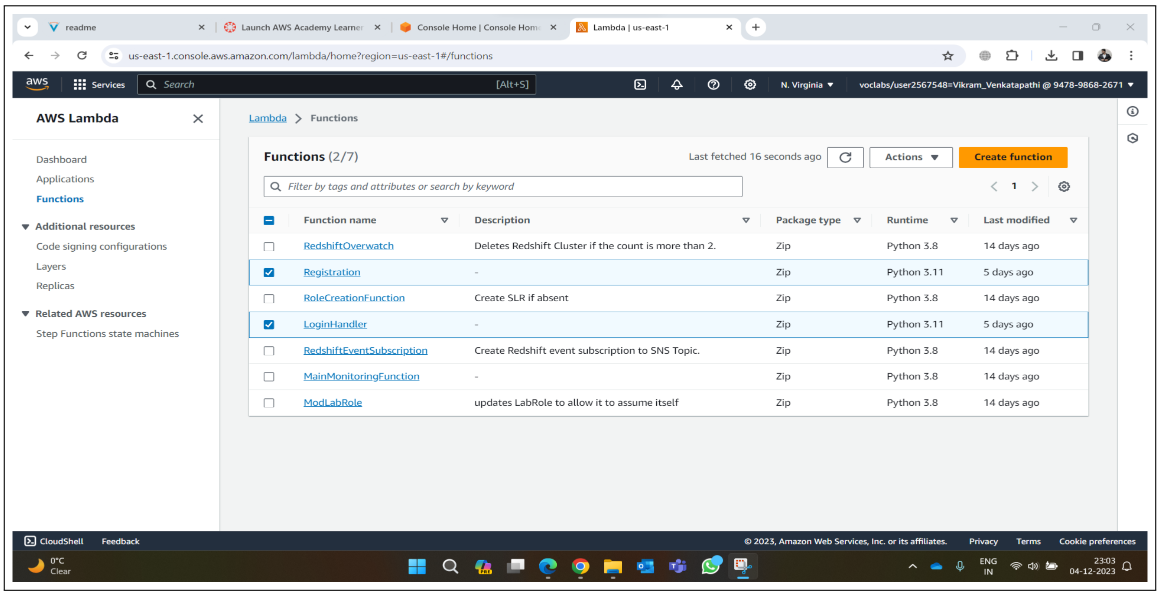
Instances (1/1)

Instance state = running | React-app-v1-env

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DI
React-app-v1-env	i-044f8ca9ae6499328	Running	t2.micro	2/2 checks passed	No alarms	us-east-1c	ec2-3-215-18

Instance: i-044f8ca9ae6499328 (React-app-v1-env)

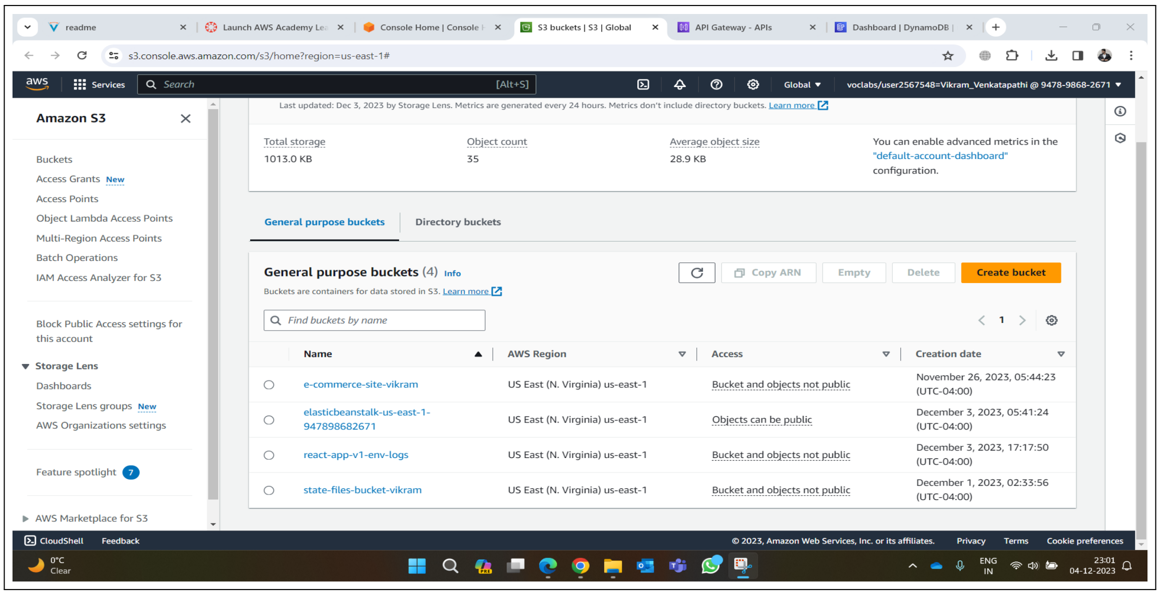
Figure 8.8.: ***EC2 instance running on us-east-1c***



Functions (2/7)

	Function name	Description	Package type	Runtime	Last modified
☐	RedshiftOverwatch	Deletes Redshift Cluster if the count is more than 2.	Zip	Python 3.8	14 days ago
☑	Registration	-	Zip	Python 3.11	5 days ago
☐	RoleCreationFunction	Create SLR if absent	Zip	Python 3.8	14 days ago
☑	LoginHandler	-	Zip	Python 3.11	5 days ago
☐	RedshiftEventSubscription	Create Redshift event subscription to SNS Topic.	Zip	Python 3.8	14 days ago
☐	MainMonitoringFunction	-	Zip	Python 3.8	14 days ago
☐	ModLabRole	updates LabRole to allow it to assume itself	Zip	Python 3.8	14 days ago

Figure 8.9.: ***Registration and LoginHandler lambdas***



Last updated: Dec 3, 2023 by Storage Lens. Metrics are generated every 24 hours. Metrics don't include directory buckets. Learn more

Total storage	Object count	Average object size	
1013.0 KB	35	28.9 KB	You can enable advanced metrics in the "default-account-dashboard" configuration.

General purpose buckets (4) Info

Buckets are containers for data stored in S3. Learn more

Name	AWS Region	Access	Creation date
e-commerce-site-vikram	US East (N. Virginia) us-east-1	Bucket and objects not public	November 26, 2023, 05:44:23 (UTC-04:00)
elasticbeanstalk-us-east-1-947898682671	US East (N. Virginia) us-east-1	Objects can be public	December 3, 2023, 05:41:24 (UTC-04:00)
react-app-v1-env-logs	US East (N. Virginia) us-east-1	Bucket and objects not public	December 3, 2023, 17:17:50 (UTC-04:00)
state-files-bucket-vikram	US East (N. Virginia) us-east-1	Bucket and objects not public	December 1, 2023, 02:33:56 (UTC-04:00)

Figure 8.10.: ***S3 buckets***

9. Conclusion

In the journey of architecting and implementing an application on AWS, this project has been a rewarding endeavor. Embracing AWS best practices, design patterns, and principles such as the least privilege, the success of the chosen architecture is evident in its resilience, scalability, and security. Despite challenges, the application now stands as a testament to the lessons learned and continuous improvement.

As a cloud enthusiast, this project has been a significant step forward, surpassing the knowledge gained in prior courses. The understanding of intricate cloud architectures, the complexities of best practices, and the delicate balance between security and functionality have all enriched my expertise. This passion for cloud computing remains unwavering, and I aspire to evolve into a cloud architect in the future. The project not only signifies a successful implementation but also marks a personal milestone in the journey towards mastering cloud technologies.

Part I.

REFERENCES

References

[1] Dinushchathurya Dinushchathurya/e-commerce-site: E-commerce application using react.JS, GitHub. Available at: https://github.com/dinushchathurya/e-commerce-site (Accessed: 04 December 2023).

[2] Amazon Web Services, "Architecting for Reliable Scalability," AWS Architecture Blog, [Online]. Available: https://aws.amazon.com/blogs/architecture/architecting-for-reliable-scalability/. [Accessed: Dec. 4, 2023].

[3] Amazon Web Services, "Modern Applications," Amazon Web Services, [Online]. Available: https://aws.amazon.com/modern-apps/. [Accessed: Dec. 4, 2023].

[4] React, "React – A JavaScript library for building user interfaces," React, [Online]. Available: https://react.dev/. [Accessed: Dec. 4, 2023].

[5] Amazon Web Services, "Introduction to DevOps on AWS," AWS Whitepapers, [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/aws-elastic-beanstalk.html. [Accessed: Dec. 4, 2023].

[6] Amazon Web Services, "Amazon EC2 FAQs," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ec2/faqs/. [Accessed: Dec. 4, 2023].

[7] Amazon Web Services, "Compute Abstractions on AWS: A Visual Story," AWS Architecture Blog, [Online]. Available: https://aws.amazon.com/blogs/architecture/compute-abstractions-on-aws-a-visual-story/. [Accessed: Dec. 4, 2023].

[8] Amazon Web Services, "Introducing Elastic Beanstalk," AWS News Blog, [Online]. Available: https://aws.amazon.com/blogs/aws/introducing-elastic-beanstalk/. [Accessed: Dec. 4, 2023].

[9] Amazon Web Services, "Getting Started with Amazon EC2," Amazon Elastic Compute Cloud, [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html. [Accessed: Dec. 4, 2023].

[10] Amazon Web Services, "Amazon Elastic Container Service," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ecs/. [Accessed: Dec. 4, 2023].

[11] Amazon Web Services, "Managing Applications in Elastic Beanstalk," AWS Elastic Beanstalk Developer Guide, [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.as.html. [Accessed: Dec. 4, 2023].

[12] Amazon Web Services, "Auto Scaling Groups," Amazon Elastic Compute Cloud, [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-groups.html. [Accessed: Dec. 4, 2023].

[13] Amazon Web Services, "Cluster Auto Scaling," Amazon Elastic Container Service, [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/cluster-auto-scaling.html. [Accessed: Dec. 4, 2023].

[14] Amazon Web Services, "Elastic Beanstalk Pricing," Amazon Web Services, [Online]. Available: https://aws.amazon.com/elasticbeanstalk/pricing/. [Accessed: Dec. 4, 2023].

[15] Amazon Web Services, "Amazon EC2 Pricing," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ec2/pricing/. [Accessed: Dec. 4, 2023].

[16] Amazon Web Services, "Theoretical Cost Optimization by Amazon ECS Launch Type: Fargate vs. EC2," AWS Containers Blog, [Online]. Available: https://aws.amazon.com/blogs/containers/theoretical-cost-optimization-by-amazon-ecs-launch-type-fargate-vs-ec2/. [Accessed: Dec. 4, 2023].

[17] Amazon Web Services, "Security in Elastic Beanstalk," AWS Elastic Beanstalk Developer Guide, [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/security.html. [Accessed: Dec. 4, 2023].

[18] Amazon Web Services, "Amazon EC2 Security Groups for Linux Instances," Amazon Elastic Compute Cloud, [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security.html. [Accessed: Dec. 4, 2023].

[19] Amazon Web Services, "Security in Amazon ECS," Amazon Elastic Container Service, [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/security.html. [Accessed: Dec. 4, 2023].

[20] Amazon Web Services, "Security in Amazon EKS," Amazon Elastic Kubernetes Service, [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/security.html. [Accessed: Dec. 4, 2023].

[21] Amazon Web Services, "AWS Lambda Pricing," Amazon Web Services, [Online]. Available: https://aws.amazon.com/lambda/pricing/. [Accessed: Dec. 4, 2023].

[22] Amazon Web Services, "AWS Step Functions Pricing," Amazon Web Services, [Online]. Available: https://aws.amazon.com/step-functions/pricing/. [Accessed: Dec. 4, 2023].

[23] Amazon Web Services, "Amazon ECS Anywhere Pricing," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ecs/anywhere/pricing/. [Accessed: Dec. 4, 2023].

[24] Amazon Web Services, "Performance optimization," AWS Lambda Operator Guide, [Online]. Available: https://docs.aws.amazon.com/lambda/latest/operatorguide/perf-optimize.html. [Accessed: Dec. 4, 2023].

[25] Amazon Web Services, "Best practices for Step Functions," AWS Step Functions Developer Guide, [Online]. Available: https://docs.aws.amazon.com/step-functions/latest/dg/sfn-best-practices.html. [Accessed: Dec. 4, 2023].

[26] Amazon Web Services, "Best practices for running tasks on Amazon ECS," Amazon Elastic Container Service, [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/task.html. [Accessed: Dec. 4, 2023].

[27] Amazon Web Services, “AWS Lambda Security,” AWS Lambda Developer Guide, [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda-security.html. [Accessed: Dec. 4, 2023].

[28] Amazon Web Services, “Security in AWS Step Functions,” AWS Step Functions Developer Guide, [Online]. Available: https://docs.aws.amazon.com/step-functions/latest/dg/security.html. [Accessed: Dec. 4, 2023].

[29] Amazon Web Services, “Security in Amazon ECS,” Amazon Elastic Container Service, [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/security.html. [Accessed: Dec. 4, 2023].

[30] Amazon Web Services, “Managing Concurrency,” AWS Lambda Developer Guide, [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html. [Accessed: Dec. 4, 2023].

[31] Amazon Web Services. ”AWS Step Functions Pricing”. _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/step-functions/pricing/. [Accessed Dec. 4, 2023].

[32] Amazon Web Services. ”Amazon ECS Anywhere Pricing”. _Amazon Elastic Container Service_. [Online]. Available: https://aws.amazon.com/ecs/anywhere/pricing/. [Accessed Dec. 4, 2023].

[33] Amazon Web Services. ”Optimizing Lambda Performance”. _AWS Lambda_. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/operatorguide/perf-optimize.html. [Accessed Dec. 4, 2023].

[34] Amazon Web Services. ”Best Practices for AWS Step Functions”. _AWS Step Functions_. [Online]. Available: https://docs.aws.amazon.com/step-functions/latest/dg/sfn-best-practices.html. [Accessed Dec. 4, 2023].

[35] Amazon Web Services. ”Best Practices for Amazon ECS Tasks”. _Amazon Elastic Container Service_. [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/task.html. [Accessed Dec. 4, 2023].

[36] Amazon Web Services. ”AWS Lambda Security Best Practices”. _AWS Lambda_. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda-security.html. [Accessed Dec. 4, 2023].

[37] Amazon Web Services. ”Security in AWS Step Functions”. _AWS Step Functions_. [Online]. Available: https://docs.aws.amazon.com/step-functions/latest/dg/security.html. [Accessed Dec. 4, 2023].

[38] Amazon Web Services. ”Security in Amazon ECS”. _Amazon Elastic Container Service_. [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/security.html. [Accessed Dec. 4, 2023].

[39] Amazon Web Services. "Managing Concurrency for a Lambda Function". _AWS Lambda_. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html. [Accessed Dec. 4, 2023].

[40] Amazon Web Services. "AWS Step Functions Features". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/step-functions/features/. [Accessed Dec. 4, 2023].

[41] Amazon Web Services. "Cluster Auto Scaling". _Amazon Elastic Container Service_. [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/cluster-auto-scaling.html. [Accessed Dec. 4, 2023].

[42] Amazon Web Services. "Amazon S3". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/s3/. [Accessed Dec. 4, 2023].

[43] Amazon Web Services. "Amazon S3 Storage Classes". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/s3/storage-classes/. [Accessed Dec. 4, 2023].

[44] Amazon Web Services. "Amazon Elastic File System". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/efs/. [Accessed Dec. 4, 2023].

[45] Amazon Web Services. "Amazon Elastic Block Store". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/ebs/. [Accessed Dec. 4, 2023].

[46] Amazon Web Services. "Amazon DynamoDB". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/dynamodb/. [Accessed Dec. 4, 2023].

[47] Amazon Web Services. "Security in Amazon DynamoDB". _Amazon DynamoDB_. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/security.html. [Accessed Dec. 4, 2023].

[48] Amazon Web Services. "Amazon RDS". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/rds/. [Accessed Dec. 4, 2023].

[49] Amazon Web Services. "Scaling Amazon DynamoDB Part 1: How Partitions Work in DynamoDB". _AWS Database Blog_. [Online]. Available: https://aws.amazon.com/blogs/database/part-1-scaling-dynamodb-how-partitions-hot-keys-and-split-for-heat-impact-performance/. [Accessed Dec. 4, 2023].

[50] Amazon Web Services. "Amazon RDS Pricing". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/rds/pricing/. [Accessed Dec. 4, 2023].

[51] Amazon Web Services. "Amazon DynamoDB Pricing". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/dynamodb/pricing/. [Accessed Dec. 4, 2023].

[52] Amazon Web Services. "How Amazon VPC Works". _Amazon Virtual Private Cloud_. [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/how-it-works.html. [Accessed Dec. 4, 2023].

[53] Amazon Web Services. "Security Groups for Your VPC". _Amazon Virtual Private Cloud_. [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-groups.html. [Accessed Dec. 4, 2023].

[54] Amazon Web Services. "Auto Scaling Groups". _Amazon EC2 Auto Scaling_. [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-groups.html. [Accessed Dec. 4, 2023].

[55] Amazon Web Services. "Elastic Load Balancing". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/elasticloadbalancing/. [Accessed Dec. 4, 2023].

[56] Amazon Web Services. "Amazon API Gateway". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/api-gateway/. [Accessed Dec. 4, 2023].

[57] Amazon Web Services. "Amazon CloudWatch". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/cloudwatch/. [Accessed Dec. 4, 2023].

[58] Amazon Web Services. "Creating an Amazon CloudWatch Alarm That Sends Email". _Amazon CloudWatch_. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html. [Accessed Dec. 4, 2023].

[59] Amazon Web Services. "Using Elastic Beanstalk with Amazon CloudWatch Alarms". _AWS Elastic Beanstalk_. [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.alarms.html. [Accessed Dec. 4, 2023].

[60] Amazon Web Services. "AWS Cost Explorer". _Amazon Web Services_. [Online]. Available: https://aws.amazon.com/aws-cost-management/aws-cost-explorer/. [Accessed Dec. 4, 2023].

[61] Amazon Web Services, "Architecting for Reliable Scalability," AWS Architecture Blog, [Online]. Available: https://aws.amazon.com/blogs/architecture/architecting-for-reliable-scalability/. [Accessed: Dec. 4, 2023].

[62] Cloudflare, "Application Layer DDoS Attack," Cloudflare, [Online]. Available: https://www.cloudflare.com/en-in/learning/ddos/application-layer-ddos-attack/. [Accessed: Dec. 4, 2023].

[63] Amazon Web Services, "Configure Subnets," Amazon Virtual Private Cloud, [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/configure-subnets.html. [Accessed: Dec. 4, 2023].

[64] Amazon Web Services, "Security Pillar," AWS Well-Architected Framework, [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/welcome.html. [Accessed: Dec. 4, 2023].

[65] Amazon Web Services, "Operational Excellence Pillar," AWS Well-Architected Framework, [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/operational-excellence-pillar/welcome.html. [Accessed: Dec. 4, 2023].

[66] Amazon Web Services, "Amazon CloudWatch Insights Metrics," Amazon CloudWatch, [Online]. Available: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-IM-insights.html. [Accessed: Dec. 4, 2023].

[67] Amazon Web Services, "Reliability Pillar," AWS Well-Architected Framework, [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html. [Accessed: Dec. 4, 2023].

[68] Amazon Web Services, "AWS Global Infrastructure," Amazon Web Services, [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/. [Accessed: Dec. 4, 2023].

[69] Amazon Web Services, "Security Group Rules," Amazon Elastic Compute Cloud, [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/security-group-rules.html. [Accessed: Dec. 4, 2023].

[70] Amazon Web Services, "Performance Efficiency Pillar," AWS Well-Architected Framework, [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/performance-efficiency-pillar/welcome.html. [Accessed: Dec. 4, 2023].

[71] Amazon Web Services, "Amazon EC2 T2 Instances," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ec2/instance-types/t2/. [Accessed: Dec. 4, 2023].

[72] Amazon Web Services, "What Is AWS Cost and Usage Report?," AWS Cost and Usage Report, [Online]. Available: https://docs.aws.amazon.com/cur/latest/userguide/what-is-cur.html. [Accessed: Dec. 4, 2023].

[73] Amazon Web Services, "Cost Optimization Pillar," AWS Well-Architected Framework, [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/cost-optimization-pillar/welcome.html. [Accessed: Dec. 4, 2023].

[74] Amazon Web Services, "AWS Cost Explorer," Amazon Web Services, [Online]. Available: https://aws.amazon.com/aws-cost-management/aws-cost-explorer/. [Accessed: Dec. 4, 2023].

[75] Amazon Web Services, "AWS Managed Services," Amazon Web Services, [Online]. Available: https://aws.amazon.com/managed-services/. [Accessed: Dec. 4, 2023].

[76] Amazon Web Services, "Fairness in Multi-Tenant Systems," AWS Builders' Library, [Online]. Available: https://aws.amazon.com/builders-library/fairness-in-multi-tenant-systems/. [Accessed: Dec. 4, 2023].

[77] Amazon Web Services, "Optimizing Your AWS Infrastructure for Sustainability – Part I: Compute," AWS Architecture Blog, [Online]. Available: https://aws.amazon.com/blogs/architecture/optimizing-your-aws-infrastructure-for-sustainability-part-i-compute/. [Accessed: Dec. 4, 2023].

[78] Amazon Web Services, "Sustainability Pillar," AWS Well-Architected Framework, [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/sustainability-pillar/sustainability-pillar.html. [Accessed: Dec. 4, 2023].

[79] Amazon Web Services, “API Gateway CORS Errors,” AWS Knowledge Center, [Online]. Available: https://repost.aws/knowledge-center/api-gateway-cors-errors. [Accessed: Dec. 4, 2023].

[80] Amazon Web Services, “Auto Scaling Groups,” Amazon Elastic Compute Cloud, [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html. [Accessed: Dec. 4, 2023].