CSCI 5409 - Advanced Topics in Cloud Computing

Term Assignment Report

Submitted by-
Yogish Honnadevipura Gopalakrishna
Banner ID – B00928029

Report

ShopVista - Your Shopping Gateway

Project Overview:

The E-Commerce Grocery and Food Delivery Web Application is a comprehensive online platform designed for users to conveniently purchase groceries and food products. The application prioritizes a personalized and secure experience by requiring user registration. Upon successful login, users are greeted with a user-friendly interface presenting a curated catalog of available items. Key features include secure authentication, an extensive product catalog categorizing items, and a streamlined order placement process for effortless selection of desired quantities. After confirmation, the system automatically generates invoices, and users receive email notifications containing payment details, ensuring a seamless and user-centric shopping experience.

Key Objectives:

User-Centric Approach:

- Prioritizing user satisfaction, the application employs a user-centric design to ensure a seamless and enjoyable shopping experience.
- User registration and authentication mechanisms are implemented to create a personalized and secure environment.

Extensive Product Catalog:

- The heart of the application lies in its comprehensive catalog, featuring a diverse range of groceries and food products.
- Items are meticulously categorized and accompanied by detailed descriptions and images to facilitate informed purchasing decisions.

Effortless Order Placement:

- Streamlining the order placement process, users can easily select the desired quantity of each item they wish to purchase.
- Intuitive design and navigation enhance the overall user experience, making it accessible for users of varying technological proficiency.

Automated Invoice Generation:

- Upon order confirmation, the system automatically generates detailed invoices summarizing the selected items and quantities.
- Users receive email notifications containing these invoices, along with payment details, ensuring transparency and accountability.

Order History and Tracking:

- The system maintains a comprehensive order history for users to reference their previous purchases.
- Real-time order tracking enhances transparency, allowing users to monitor the status of their deliveries.

Menu item requirements

Compute(2)

1. Elastic BeanStalk

Elastic Beanstalk stands out as a superior deployment option when compared to Amazon EC2 or Docker in specific scenarios. Its primary strength lies in abstracting the intricacies of infrastructure management, offering a streamlined experience for developers. With Elastic Beanstalk, tasks like capacity provisioning, load balancing, and auto-scaling are automated, allowing developers to concentrate solely on their application code[1]. The platform's ease of use is highlighted by a user-friendly interface and command-line tools, simplifying deployment and management tasks. Furthermore, Elastic Beanstalk excels in automatic scaling, ensuring optimal resource utilization based on application demand, a feature that requires more manual effort with EC2 and additional orchestration tools with Docker.

AWS Elastic Beanstalk excels in simplifying the deployment process by abstracting the complexities of infrastructure management. I can focus on my application code, while the platform takes care of deployment intricacies, including capacity provisioning, load balancing, and auto-scaling. The user-friendly web interface and command-line interface (CLI) make Elastic Beanstalk exceptionally easy to use. It supports a broad spectrum of programming languages, from Java and .NET to Python and Docker, providing developers with flexibility in their language choices.

Another key advantage of Elastic Beanstalk is its managed environment. AWS takes care of the underlying infrastructure, including the operating system, middleware, and runtime. This reduces operational overhead, providing developers with a platform where they can solely focus on their application's logic. The platform seamlessly integrates with various AWS services, facilitating the incorporation of databases, storage (such as Amazon RDS and S3), and monitoring tools like CloudWatch. Elastic Beanstalk seamlessly integrates with various AWS services and Amazon S3, offering a comprehensive solution for diverse application needs.

2. AWS Lambda

I decided to run my code on AWS Lambda since it's an easy and effective approach to run functions. without overseeing the servers[2]. I can concentrate on creating my code as Lambda takes care of server setup and maintenance for me. It adjusts automatically for incoming
requests, thus I won't need to manually modify resources to handle any volume of traffic. Lambda operates on a pay-as-you-go basis as well, so I just have to pay for the actual time that my code utilisation, which lowers the cost for my program.

AWS Lambda provides a versatile solution for building event-driven applications by enabling the seamless execution of code in response to various triggers, such as HTTP requests or modifications to data in Amazon S3. This flexibility allowed me to easily construct applications that respond dynamically to specific events. The integration capabilities of AWS Lambda extend beyond its core functionality, allowing for smooth collaboration with other AWS services like DynamoDB and SNS. The strength of AWS Lambda lies in its ability to simplify the execution of code in a scalable and cost-efficient manner.

Storage (1)

1. AWS Dynamo DB

I've opted for Amazon DynamoDB as the storage solution for my project details within the AWS environment. This choice is attributed to the platform's user-friendly nature, eliminating the need for manual server setup or backup management, as AWS handles these aspects seamlessly. DynamoDB's automatic scalability aligns perfectly with the dynamic nature of my application, effortlessly adapting to varying data loads. Its notable speed ensures swift data access for users[3]. The service's flexibility allows for easy modifications and additions to data fields as my application evolves over time. DynamoDB's seamless integration with other AWS services contributes to building a reliable and interconnected application. Additionally, the service prioritizes data security and availability, providing reassurance even in the face of potential issues. In essence, DynamoDB strikes the right balance between simplicity, scalability, speed, and reliability, making it an ideal choice for my application.

2. AWS S3

I have employed Amazon S3 buckets to store essential components of my project, including the zipped file for my React app and Lambda functions. These stored artifacts serve a pivotal role in my configuration setup, where the deployment process for AWS Elastic Beanstalk and Lambda functions is orchestrated. Utilizing S3, I can seamlessly integrate the code for Lambda functions and deployment zip files into my configuration files, ensuring efficient retrieval and deployment processes for both AWS Elastic Beanstalk and Lambda services.

Network(1)

1. API Gateway

An API gateway serves as a crucial component in modern software architecture by providing a unified entry point for clients to interact with various microservices[4]. It simplifies client-server communication, consolidating multiple service endpoints into a single, easily accessible interface. This not only reduces the complexity of interactions for clients but also streamlines the development process. Security is a paramount concern in distributed systems, and API gateways play a pivotal role by centralizing security measures such as authentication, authorization, and encryption. This ensures a secure communication channel between clients and microservices, mitigating potential vulnerabilities.

Load balancing is another key feature of API gateways, optimizing resource utilization by distributing incoming requests across multiple instances of microservices. This not only improves system reliability but also enhances scalability as the application can efficiently handle increased traffic. Additionally, API gateways offer robust monitoring and analytics capabilities, providing real-time insights into API usage, performance metrics, and error rates. This centralized monitoring facilitates proactive issue resolution, ensuring the overall health and performance of the system.

In the context of my Application, the API gateway serves project-specific needs. It acts as a unified endpoint for various E-commerce operations, consolidating endpoints related to user authentication, order processing, and inventory management. This simplifies integration for the frontend application, offering a cohesive experience for users. The API gateway allowed me to easily create APIs to interact with my lambda functions, making it straightforward for web app to access APIs of my application. All things considered, AWS API Gateway simplifies the development and administration of APIs, which makes it a great option for developing scalable and dependable applications.

General (2)

1.AWS SNS (Simple Notification Service)

I have used AWS SNS for sending email to the customers whenever they order items. I create topic using their email id which is unique for each user when they register for the app. I store the ARN of these topics in the customer database. Whenever an user order from the app, I just send the invoice of the items and also the payment details through which the user will make the payment. Other choice was SQS but it requires a lambda to pull the messages from the queue hence defeating the purpose of the immediate mail and payment of the user.

2. Infrastructure as code (Cloud Formation):

CloudFormation is a game-changer for me in AWS, serving as a fundamental tool for Infrastructure as Code (IaC). It empowers me to define and provision AWS resources in a declarative manner, using templates in JSON or YAML format. This approach brings consistency and repeatability to my infrastructure setup, mitigating the risk of configuration errors and making resource management more efficient.

When it comes to Continuous Integration and Continuous Deployment (CI/CD), CloudFormation takes center stage in my workflow. By representing my infrastructure as code, I seamlessly integrate CloudFormation templates into my CI/CD pipelines. This integration ensures that changes to my infrastructure are treated with the same level of control and automation as changes to my code, fostering collaboration between my development and operations teams.

Examining my CloudFormation script, I see how it orchestrates the creation of various AWS resources – DynamoDB tables, Lambda functions, an API Gateway, and an Elastic Beanstalk environment – all of which are essential for my web application. In the CI/CD context, every modification triggers a pipeline where the CloudFormation template is deployed automatically. This ensures that infrastructure changes align with code changes, creating a unified and efficient deployment process.

CloudFormation's ability to version my infrastructure is a crucial aspect of my CI/CD practices. It allows me to track, test, and deploy changes incrementally. If issues arise, I can easily roll back to previous configurations, following the best practices of CI/CD. In essence, CloudFormation is my go-to tool for achieving automated, repeatable, and traceable infrastructure management in AWS, perfectly aligning with the principles of CI/CD.

Deployment Model

I choose to public cloud to host my application as it comes with a lot of benefits. As it is an ecommerce platform made for the public to shop from their homes, public cloud is opt as anyone from the internet can access the app and shop. Additionally, because the cloud automatically scales up resources to manage traffic spikes, it guarantees that my website remains online even when there are large numbers of users at once. I won't have to worry this way. concerning my website stuttering or crashing when it becomes crowded. Additionally, I feel very secure using the public cloud, which is comforting for my personal data and initiatives. Strong security measures are taken care of by the cloud provider for my data. safeguards such as firewalls and encryption. An additional benefit is that the cloud provider oversees everything the server-related tasks, saving me from having to handle upgrades or maintenance.

Delivery Model

I've opted for a Platform as a Service (PaaS) delivery model for my frontend using AWS Elastic Beanstalk, and it feels like having a hassle-free assistant for my website! The choice of Elastic Beanstalk eliminates the need for me to delve into the intricacies of server management and other technical complexities. It streamlines the deployment process, allowing me to focus more on refining my website without the burden of handling server-related intricacies. Elastic Beanstalk serves as a reliable and user-friendly companion, making the overall experience of managing and deploying my application notably smoother and more efficient.

For the backend of my application, I've embraced a Function as a Service (FaaS) delivery model by leveraging Lambda functions. Lambda's serverless architecture liberates me from concerns about server setup and maintenance, enabling me to channel my efforts directly into writing code. The beauty of Lambda lies in its automatic scaling capabilities, dynamically adjusting resources in response to incoming requests. This means I can confidently handle varying levels of traffic without the need for manual intervention. Lambda's seamless scalability provides a flexible and efficient backend solution, allowing me to prioritize development and functionality over infrastructure management.

For the database component of my project, I've adopted a Platform as a Service (PaaS) delivery model, utilizing Amazon S3 for file storage and DynamoDB for data storage. Opting for a PaaS approach means I can operate without the burden of managing the intricate infrastructure supporting these services. The interaction with the database is seamlessly facilitated through the boto3 for python, offering a convenient and efficient means of accessing and manipulating data. This choice not only simplifies the overall database management process but also ensures a streamlined experience in developing and interacting with the storage services essential for my project.

Architecture

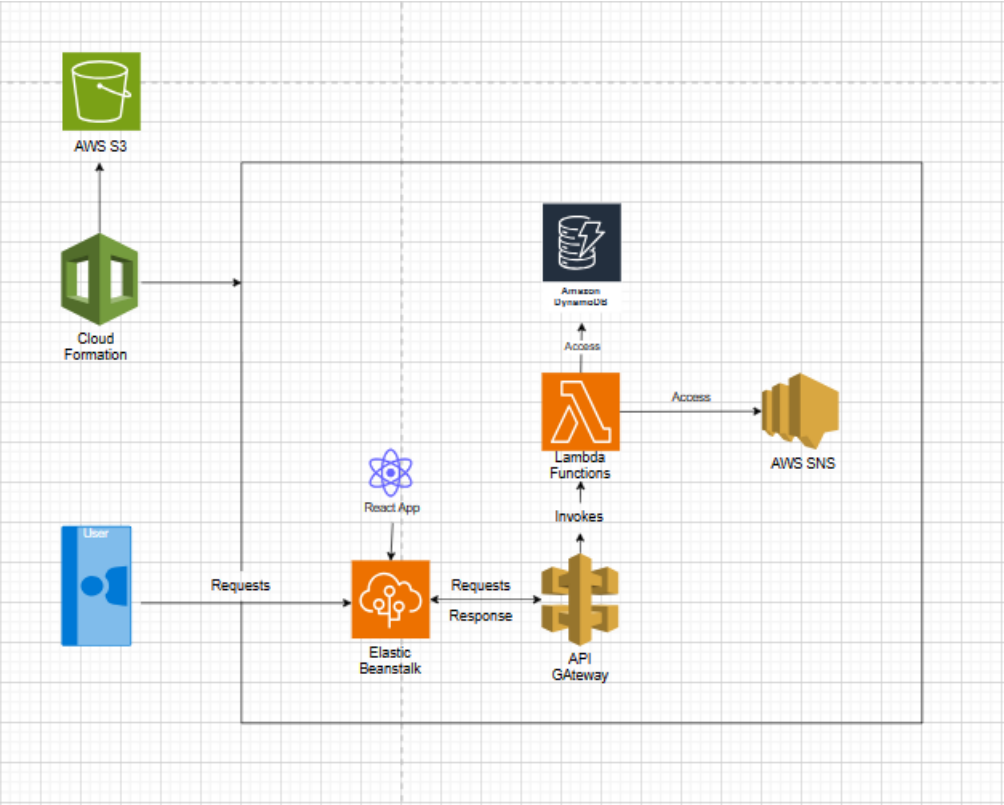

Fig 1. Service Architecture. It was created using [6]
Source: From Author

I have used React for my frontend and have used python which uses boto3 library to communicate with the cloud as my backend in my lambdas. S3 and DynamoDB are the storage services I have used in my project.

The above are the cloud services I am using to build the application. The entry point of the users request is the elastic beanstalk which contains the react app hosted in it. The react App will make required API calls to the gateway which invokes the appropriate lambda required. The lambda then will execute the code and accesses dynamo DB and AWS SNS whenever required. This whole architecture will be built by cloud formation which will use S3 Bucket to place the react app in the elastic beanstalk and also python code in the appropriate lambdas.

Security

In terms of security, my project has implemented a robust approach to safeguarding data across all stages of the system, addressing both data in transit and at rest.

For data in transit, the user will have to register with the app at the start. This data will be stored in DynamoDB which is secured for data storage. When the user inputs any details, the react app is having input validations to carefully check and parse the data.

When it comes to data at rest, the project adopts encryption mechanisms to safeguard information stored in the database and file storage. In the case of DynamoDB, data is encrypted at rest by default, providing an additional layer of protection. Amazon S3, used for file storage, supports server-side encryption, ensuring that stored files remain confidential and tamper-resistant.

Regular security audits and monitoring are integral to the project's security posture. AWS CloudWatch, along with other monitoring tools, helps in the detection of any unusual activities or potential security threats. Additionally, AWS provides a secure environment for deploying Lambda functions and Elastic Beanstalk applications, contributing to the overall security framework.

By adopting a comprehensive security strategy that encompasses encryption, access controls, and continuous monitoring, the project ensures the confidentiality and integrity of data throughout the system's lifecycle, fostering a secure environment for both users and the application.

Cost Metrics

Current estimate URL:
https://calculator.aws/#/estimate?id=90e11fe35d4edfd17de7e9005e7a38d9ccde8d60

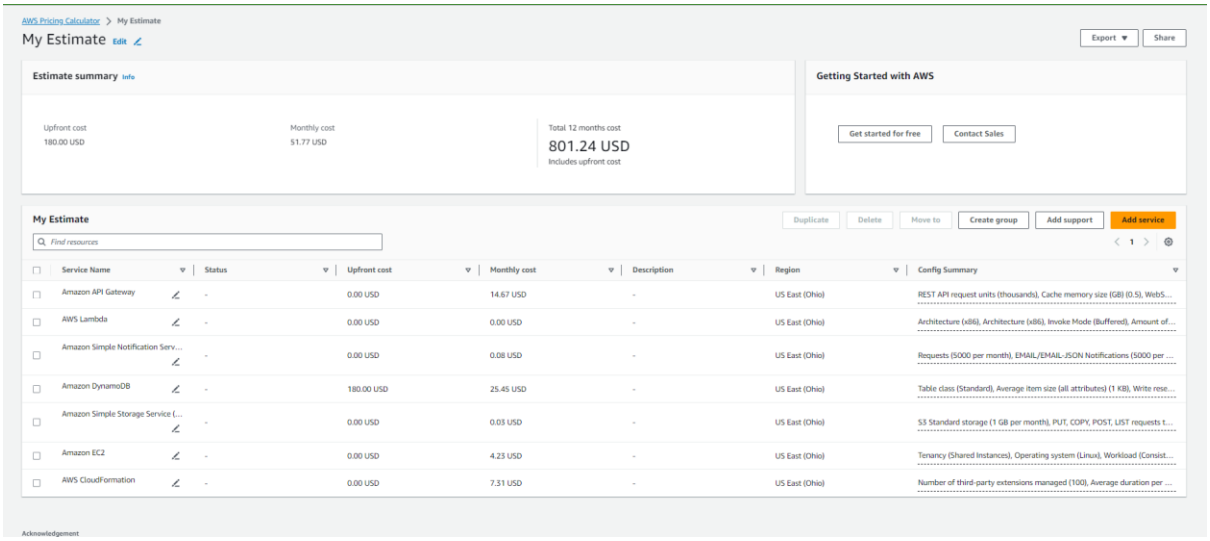

Fig 2. Cost Estimation of AWS Services. It was created using [5]
Source: From Author

According to the estimation the upfront cost is 180 USD with monthly cost coming upto 52 USD and the yearly cost will be around 800 USD.

Alternatives:

I was trying to see to store the data in RDS because the data stored in RDS can execute complex SQL queries, data structures with relationships, and robust transactional support. RDS, with its support for various relational database engines, offers mature features such as automatic backups, point-in-time recovery, and adherence to SQL standards. It excels in scenarios where a flexible schema, ACID compliance, and a well-established ecosystem are critical for the application's success.

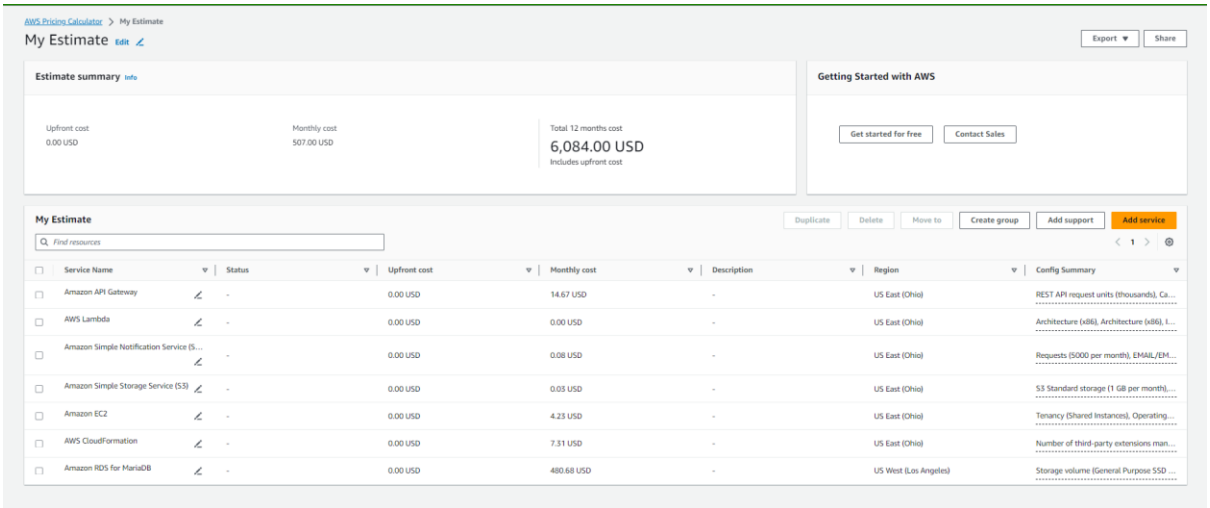

Fig 3. Cost Estimation of Expensive approach. It was created using [5]
Source: From Author

The alternate monthly cost is 500 USD which makes it around 6000 USD per year. Although this is expensive, as the application grows the complexity of the database can be handled better in RDS.

Reproducing the architecture in private cloud

Reproducing the architecture in a private cloud would require procuring the necessary hardware, networking equipment, and software licenses. The estimate would include servers for hosting, networking infrastructure for connectivity, storage solutions, and licenses for any software components used. Additionally, the organization would need to invest in security measures, such as firewalls and intrusion detection systems, to ensure the same level of availability and data protection. The cost could vary significantly based on factors like the size of the infrastructure, specific hardware and software choices, and any additional security measures implemented. A rough estimate for a moderately sized deployment could range from tens of thousands into account both initial setup costs and ongoing operational expenses.

Cost Monitoring tools

Effectively monitoring costs in a cloud environment is crucial for maintaining financial control and optimizing resource usage. Cloud providers offer dedicated tools for this purpose, such as AWS Cost Explorer. These tools provide insights into spending patterns, allow users to set budgets, and offer alerts to prevent unexpected cost escalations. Additionally, cloud-native monitoring services like AWS CloudWatch can be employed to track costs alongside performance metrics. These services enable users to analyze resource utilization, identify cost drivers, and receive notifications when predefined spending thresholds are approached. I believe that DynamoDB and Ec2(Elastic beanstalk) would cost the most money in my architecture. By using the above tools I can monitor the usage and costs of the services.

Upcoming Features

In the ongoing evolution of the application, the focus would be on enhancing both security and user experience. First and foremost, implementing a robust user authentication and authorization system becomes paramount to secure user data and control access effectively. Amazon Cognito provides a seamless solution for integrating these identity management features into the application, ensuring a secure user environment.

Additionally, expanding the application's feature set could include functionalities like creating a complete payment gateway instead of sending a mail with the payment details. I can add features like adding items from different categories and giving an option to search and filter the items which would save the time for the user.

References

[1] "AWS Elastic Beanstalk FAQs - Amazon Web Services (AWS)," *Amazon Web Services, Inc.*, 2016. https://aws.amazon.com/elasticbeanstalk/faqs/ (accessed Nov. 28, 2023).

[2] "Best practices for working with AWS Lambda functions - AWS Lambda," *Amazon.com*, 2023. https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html (accessed Nov. 28, 2023).

[3] "Accessing DynamoDB - Amazon DynamoDB," *Amazon.com*, 2023. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AccessingDynamoDB.html (accessed Nov. 29, 2023).

[4] "Choosing between REST APIs and HTTP APIs - Amazon API Gateway," *Amazon.com*, 2023. https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html (accessed Nov. 30, 2023).

[5] "AWS Pricing Calculator," *Calculator.aws*, 2023. https://calculator.aws/#/estimate (accessed Nov. 30, 2023).

[6] free, "Flowchart Maker & Online Diagram Software," *Diagrams.net*, 2023. https://app.diagrams.net/ (accessed Nov. 30, 2023).