CSCI 5409

Cloud Computing

Term Assignment

Submitted by:
Tirth Bharatiya (B00955618)

1. Introduction:

   EmailNest application can be used by anyone who wants to connect with their audience through email. It helps users to create links which they can share with their audience. The user will have to put information like title, subtitle, logo (which will be used to create an email capture page) and redirect url to create a new link. Redirect url is where their audience will be redirected after taking their email. So if the user wants to share some amazon product link with their audience but at the same time they want that their user give their email id first before redirecting to the actual product. So they've all the email of their subscribers and then they can send more newsletter to their audience. The user should be able to get their subscriber list without taking much time but at the same time, their audience will be larger so the email capture page should load in no time providing seamless experience to their audience as well. Once the user has their subscribers' emails, they can send their message/newsletter directly to them.

2. Describe how you met your menu item requirements: you will list the services you selected, and provide a comparison of alternative services, explaining why you chose the services in your system over the alternatives.

   AWS services [1] that I used for this assignment are:

   Compute:

   1. AWS Elastic Beanstalk
   2. AWS Lambda

   Storage:

   1. AWS S3

   Network:

   1. AWS API Gateway

   General:

   1. AWS Key Management Service
   2. AWS Secrets Manager
   3. AWS SNS

   - Why I chose Elastic Beanstalk:

I can easily deploy my application over Elastic Beanstalk compared to EC2. With Beanstalk, I can focus on developing and deploying my application without worrying about infrastructure management. Elastic Beanstalk provides built-in auto-scaling capabilities that automatically adjust the number of EC2 instances based on traffic fluctuations. And it would be difficult to manage and deploy different APIs on different Lambda and create APIs for them with API Gateway.

- Why I chose Lambda:

I chose Lambda because of its pay-per-use pricing. For the two APIs experiencing significant traffic, it's strategically beneficial to segregate them from other APIs and deploy them independently on AWS Lambda. Lambda is cheaper compared to Beanstalk so using it is perfect for APIs with high traffic.

- Why I chose S3:

I opted for Amazon S3 over storing hashed images in DynamoDB to store images due to its capabilities in handling large binary objects efficiently and cost-effectively. S3's scalable and durable storage infrastructure ensures reliable storage and retrieval of images, with a simple pricing model. It'll also allow me to use different S3 Storage classes for images which are not being used regularly.

- Why I chose API Gateway:

I selected API Gateway to interface with Lambda functions due to its seamless integration with AWS Lambda, enabling efficient invocation and management of serverless APIs. API Gateway simplifies the process of building and deploying APIs. API Gateway allows for straightforward configuration of endpoints, communication between frontend and Lambda functions.

- Why I chose KMS:

I opted for AWS Key Management Service (KMS) for S3 encryption instead of AWS managed keys to maintain granular control over key management and access policies. It'll allow me to rotate the key for better security.

- Why I chose Secrets Manager

I selected a secret manager to store my mongodb credentials as it allows me to securely store my mongodb credentials at a single central place, which will be used by applications on Elastic Beanstalk and lambda functions. It can integrate with other AWS services easily compared to any other 3rd party application and it also provides encryption at rest and in transit.

- Why I chose SNS

I chose SNS to allow users to send email to their subscribers. I did not have access to use SES, so SNS was the second best choice to send emails.

3. Describe your deployment model. Why did you choose this deployment model?

   For EmailNest, going with the Public Cloud makes perfect sense. It's super scalable, so we can easily handle lots of users without any issue. Plus, it's everywhere, so reaching new users is easy. And also in future we can take advantage of CDN to provide a seamless experience to the users. With the public cloud, we don't have to worry about managing servers or dealing with downtime because it automatically switches to backups if needed. And it's quick to get new stuff up and running, so we can roll out updates fast and keep things moving smoothly. Overall, it's a cost-effective way to run an EmailNest application.

4. Describe your delivery model. Why did you choose this delivery model?

   I chose Software as a Service (SaaS) as the delivery model for EmailNest, eliminating the need for users to install software locally or on a server. With SaaS, users can access the platform through a web browser via the internet. EmailNest is managed and maintained by AWS, ensuring scalability and accessibility without user intervention. This delivery model ensures a seamless experience for users, free from scalability or accessibility concerns.

5. Describe your final architecture:

   5.1. How do all of the cloud mechanisms fit together to deliver your application?

   - Beanstalk provides auto scaling making EmailNest highly available and easy to scale. It provides load balancing which is to distribute load between underlying EC2 instances.
   - When the user's audience subscribes to their newsletter, they receive email confirmation for SNS subscription. Once they accept it, after that the user will be able to send their newsletter using SNS.
   - 2 APIs where the email capture page gets loaded and saving subscribers' email id will face heavy traffic, moving them to lambda makes sure that it auto scales with minimal cost. API Gateway is used to trigger lambda functions using API endpoints.

- While user login, registration and new link application will fetch mongoDB credentials from Secrets Manager and connect with MongoDB.
- Encrypted data is stored in S3 which provides in store and in-transit data security.

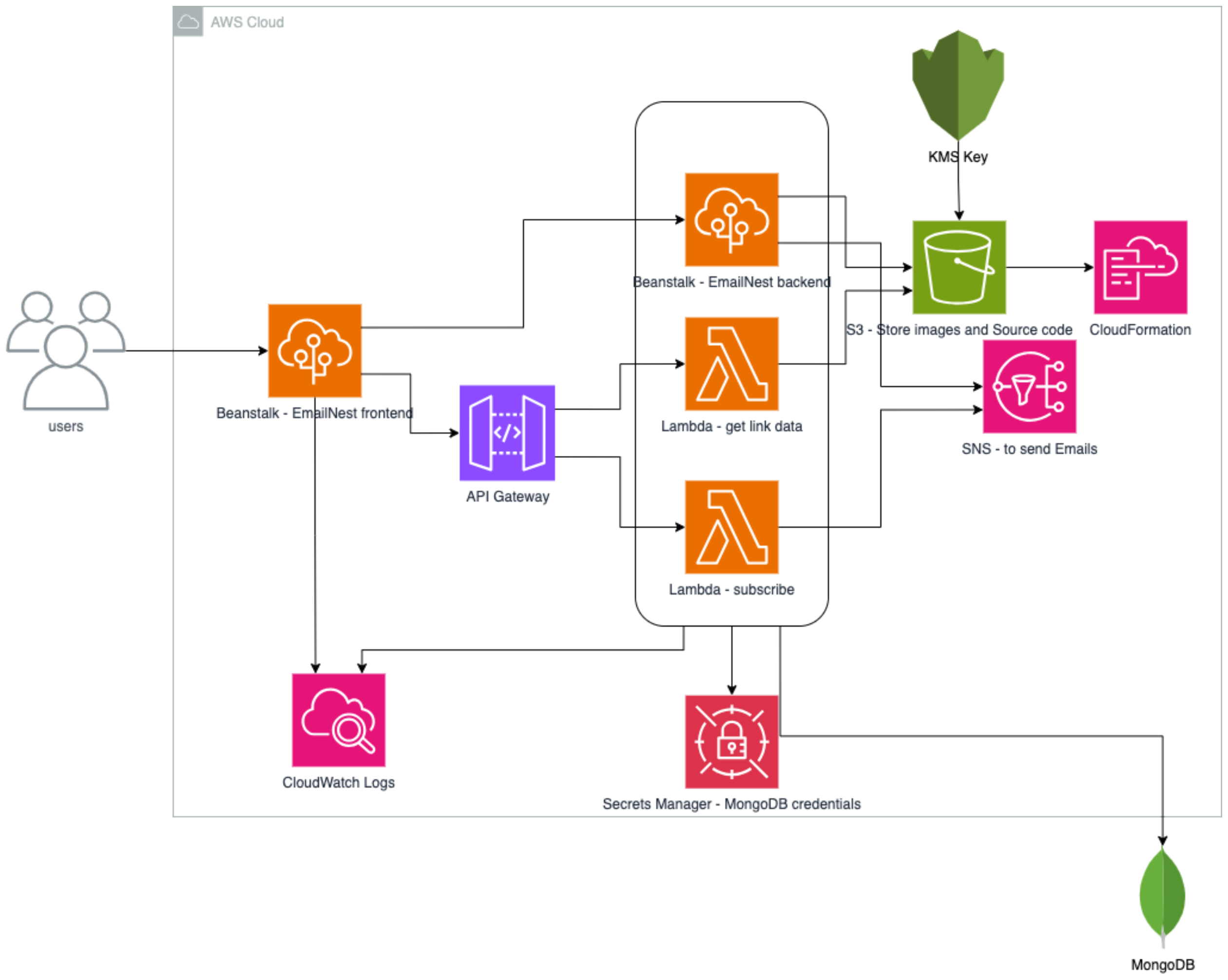
AWS Cloud
KMS Key
Beanstalk - EmailNest backend
S3 - Store images and Source code
CloudFormation
users
Beanstalk - EmailNest frontend
Lambda - get link data
SNS - to send Emails
API Gateway
Lambda - subscribe
CloudWatch Logs
Secrets Manager - MongoDB credentials
MongoDB

Figure 1: EmailNest Cloud Architecture [2].

5.2. Where is data stored?

User's logo is stored in S3 and link data (title, subtitle, subscribers emails, image url) are stored in MongoDB. I chose MongoDB over DynamoDB to store user data and link data because MongoDB offers flexibility in schema design. The document-oriented structure of MongoDB aligns well with the nature of my data, enabling efficient storage and retrieval of user information and associated links.

5.3. What programming languages did you use (and why) and what parts of your application required code?

My architecture comprises a client-side application and a server-side application. The frontend is developed using ReactJS, a JavaScript-based library, while the backend is built using NodeJS and ExpressJS for business logic implementation. NodeJS was chosen for its seamless request handling, facilitated by ExpressJS routers. The frontend communicates with the backend via the axios library, facilitating REST API calls. Using JavaScript based frontend and backend makes deployment easier compared to other frameworks.

5.4. How is your system deployed to the cloud?

I've used CloudFormation to deploy my application on the AWS Cloud.Backend APIs for login, signup, new link and frontend application is deployed on Beanstalk. APIs which will have heavy traffic are deployed on Lambda, connected via API Gateway.

5.5. If your system does not match an architecture taught in the course, you will describe why that is, and whether your choices are wise or potentially flawed.

I've developed EmailNest using architectures learnt in lectures. I've used serverless and microservice architecture to develop EmailNest.

6. How does your application architecture keep **data secure at all layers**? If it does not, if there are vulnerabilities, please explain where your data is vulnerable and how you could address these vulnerabilities with further work.

Data Encryption at Rest:

KMS key is used to encrypt and decrypt data at rest in S3 bucket, Bcrypt hashing is used to store user's password in MongoDB.

Data Encryption in transit: HTTPS endpoint is used throug API Gateway to provide data encryption in-transit.

Security mechanism used:

- By default Lambda provides HTTPS endpoint which provides in-transit data protection whereas Beanstalk provides HTTP which makes it vulnerable.
- Encrypted user passwords using Bcrypt are stored in MongoDB.

Vulnerabilities Addressed with Further Work:

- Implementing HTTPS instead of HTTP would enhance security, requiring SSL/TLS certificates for the front end hosted on Beanstalk. HTTPS provides a secure communication channel between the client and server.
- IAM Role-Based Security: The Lab role provided offers default security permissions managed by AWS. However, modifying the Lab role permissions could allow for applying security patches and updates to the EC2 instance, reducing the risk of security flaws and data breaches. Additionally, CloudWatch metrics and alarms can be set up to monitor unusual activities on the EC2 instance.
- Instead of storing passwords in MongoDB, we can use AWS Cognito to manage users.

Server-Side: The backend server, powered by Node.js, is hosted on AWS Elastic Beanstalk and Lambda for isolation and management of server-side operations.

Security mechanism used:

- Security mechanisms in AWS include IAM roles and security groups for both Elastic Beanstalk and Lambda, ensuring controlled access to resources. Lambda functions benefit from resource policies and encryption options like AWS KMS for data protection. These measures collectively safeguard against unauthorized access and potential security risks.

Vulnerabilities Addressed with Further Work:

- Furthermore, we can assign SSL/TLS certificates to Beanstalk to make it secure using AWS Certificate Manager (ACM).

Storage: S3 is used to store the images and mongoDB is used to store link and user data.

Security mechanism used:

- All the images stored in AWS are encrypted using KMS. I have my own KMS key to be used for encryption to images in S3.
- MongoDB credentials are stored in the Secret manager to make it secure.

Vulnerabilities Addressed with Further Work:

- To increase the security we can send encrypted data while sending image data between frontend and backend.
-

7. What would your organization have to purchase to reproduce your architecture in a private cloud while providing relatively the same level of availability as your cloud implementation? Try to give a rough estimate of what it would cost, don't worry if you are far off. These systems are complicated and you don't know all the exact equipment and software you would need to purchase. Just explore and try your best to figure out the combination of software and hardware you would need to buy to reproduce your app on-premise.

   Below is the breakdown of cost to replicate the same architecture in a private cloud with the same level of availability:

   Server: The approximate cost for an Intel E3-1240 v5 (3.50 GHz) server is $1,476, with additional expenses for RAM ($147) and HDD ($95), totaling approximately $1,800 per server. For 3 servers, the total upfront cost would be $5,400.

   Virtualization Software: Virtualization software like VMware ranges from $4,000 to $8,000 per server, resulting in a total cost of $12,000 to $24,000 for 3 servers.

   Load Balancer: An Enterprise VA 1G load balancer is priced at approximately $4,500 for a one-time purchase.

   Storage Cost: Storing 1 Terabyte of data would incur monthly costs of $100 to $150, totaling $1,200 to $1,800 annually.

   Cooling and Electricity: Factoring in cooling and electricity, which contribute to around 50% of server costs, the expenses would be approximately $4,000 to $5,000 yearly.

   Network Requirements: Routers like the CISCO ISR4221/K9 Router cost around $1,200 each, while firewalls such as the Fortinet fortigate range from $250 to $2,000. Bandwidth for handling application traffic could cost approximately $800 to $1,000 monthly, totaling $12,000 yearly for 500 Gbps.

   Other Costs: Additional expenses for staff and maintenance tools might amount to $12,000 to $15,000 yearly.

   Operating System License: A Windows Standard Edition license costs approximately $950 to $1,100 per server.

   SendGrid: 3rd party email service will cost around $1000.

8. Which cloud mechanism would be most important for you to add monitoring to in order to make sure costs do not escalate out of budget unexpectedly? In other words, which of your cloud mechanisms has the most potential to cost the most money?

   If we compare different services used in building EmailNest, Beanstalk, Lambda, S3, SNS are the most expensive services which will take the majority of the budget for the application. To make sure that the bill doesn't exceed the budget we can set up CloudWatch with email notification and rules to terminate beanstalk application if it exceeds the costs defined in the budget.

9. How would your application evolve if you were to continue development? What features might you add next and which cloud mechanisms would you use to implement those features?

   If I've time and budget, here are few things I would do to improve security and performance of the application:

- Private VPC and subnet to increase the security.
- Use AWS Cognito to manage the user and permission. It's better to manage application users.
- CloudFront to deliver a seamless experience to the user. As mentioned, users create links for their audience and their audience can be around the world. And the application provides email capture pages to take their email id so it is important that they don't have to wait to move to the actual url.
- SES to send emails instead of SNS.
- I would implement CI/CD pipeline using AWS CodeCommit, CodeBuild, CodeDeploy and CodePipeline.
- Implementation of caching using CloudFront and Elasticache, would surely improve the user experience.

References:

[1] “Welcome to AWS Documentation,” AWS, [Online]. Available: https://docs.aws.amazon.com. [Accessed: Apr 9, 2024].

[2] “Flowchart maker and Online Diagram Software,” Draw.io, [Online]. Available: https://app.diagrams.net. [Accessed: Apr 9, 2024].