CSCI 5409
Cloud Computing
Sec: 01
(2022/2023 Summer)

Dalhousie University

Term Project (Final Report)

Title: Pinboard
(A stateless asynchronous notice board application.)

August 01, 2023

Pratik Mukund Parmar (B00934515)
pratikparmar@dal.ca

Gitlab Link: https://git.cs.dal.ca/courses/2023-summer/csci4145-5409/pmparmar/-/tree/main/termassignment

1.1 Problem

The stateless notice board application, Pinboard, aims to address the challenge of efficiently disseminating information by providing a cloud-based, serverless solution for posting and accessing notices in real time. Users can instantly post notices, and others can search and retrieve relevant notices based on different parameters. The application utilizes AWS services like EC2, Lambda, API Gateway, DynamoDB, AWS Backup, and AWS SNS to ensure scalability, security, and cost-effectiveness. By implementing this solution, the organization can enhance communication, improve productivity, and streamline the information dissemination process.

1.2 Problems with existing solutions

The existing solutions, such as physical notice boards and basic web-based systems, suffer from various limitations, including a lack of real-time updates, limited accessibility, maintenance overhead, security concerns, scalability issues, and inadequate search capabilities. They may also lack data backup mechanisms, incur unnecessary costs, and offer limited communication channels and collaboration features. By implementing a stateless notice board application using modern AWS services, these challenges can be addressed, providing a more efficient, scalable, and secure solution for information dissemination and notice management.

1.3 Proposed Solution

To overcome the limitations of existing solutions, the proposed solution is to develop a stateless asynchronous notice board application using AWS services. This cloud-based platform will utilize EC2, Lambda, API Gateway, DynamoDB, AWS Backup, and AWS SNS to ensure scalability, real-time updates, secure authentication, and efficient notice posting and retrieval. With a focus on cost optimization, monitoring, and data resilience, the application will streamline information dissemination and notice management, providing a reliable and user-friendly platform for users.

2. Features

The proposed web application, Pinboard, offers a comprehensive set of features to enhance user experience and streamline the process of information dissemination and notice management. The key features include:

1. Stateless Architecture -

   The application's stateless design ensures high scalability, and fault tolerance, and simplifies the management of user sessions.

2. Real-time Notice Posting -

   Users can instantly post notices through the web application, ensuring immediate visibility to all authorized users. The use of AWS Lambda functions allows for quick processing of incoming requests, enabling real-time updates.

3. Efficient Notice Retrieval -

   The application offers efficient notice retrieval based on whether the user wants to read all notices on the board or a specific notice. DynamoDB serves as the primary NoSQL database, enabling fast and reliable data access.

4. Push Notifications -

   AWS SNS is employed to implement a publish-subscribe model, allowing users to get updates for new notices or important updates. This feature enables real-time push notifications as soon as a new notice is posted, or an existing notice is updated.

5. Data Backup and Recovery -

   AWS Backup is configured to automate regular backups of the DynamoDB table. This ensures data resilience and easy recovery in case of accidental data loss.

6. Cost Optimization -

   The architecture is optimized for cost efficiency, leveraging AWS services to only utilize resources as needed, helping reduce operational expenses.

7. User-Friendly Interface -

   The web application offers a simple and intuitive user interface, making it easy for users to navigate, post, and retrieve notices, even for non-tech-savvy individuals.

3.1 Implementation – Why Cloud?

Pinboard was created and delivered using Amazon Web Services (AWS) Cloud Services. The adoption to employ cloud services was taken to minimize the need for real servers and hardware infrastructure, which may be expensive to purchase and maintain.

We decided to implement the Pinboard in the cloud because of the multiple advantages it provides. Cloud solutions, such as AWS, provide the scalability required to accommodate variable user traffic, maintaining a consistent experience during high-traffic periods. Furthermore, the cloud services pay-as-you-go model allows us to better manage expenses by only paying for the resources we use, cutting off the need for large upfront investments. The cloud's high availability and dependability, supported by redundant data centers, ensures the application remains accessible and operating in the event of hardware problems or geographical downtime.

Moreover, security is a primary consideration, and cloud service providers such as AWS build robust security measures that perform better than what organizations can achieve on their own. This includes encryption, access limits, and frequent security upgrades to keep our application safe from potential threats. Additionally, the cloud's flexibility and agility enable rapid deployment of applications, allowing the development team to focus on adding features and improving functionality rather than managing infrastructure. Also, cloud-based applications have an international scope and can be accessed from anywhere, allowing for seamless collaboration among scattered teams or people. Thus, by leveraging cloud services, we can deliver a user-friendly and reliable platform for information dissemination and notice management, ensuring an efficient user experience.

3.2 Services Used

Compute:

- AWS EC2.
- AWS Lambda.

Storage:

- AWS DynamoDB.

Network:

- AWS API Gateway.

General:

- AWS Backup.
- AWS Simple Notification Service (SNS).

3.2.1 EC2

As part of AWS's Infrastructure as a Service (IaaS) services, EC2 allows users to set up virtual machines (VMs) in the cloud while maintaining complete control over the operating system and application software. It allows users to form security groups and set access control lists (ACLs) as needed, resulting in strong security configurations. Furthermore, EC2 works smoothly with other AWS services, such as AWS CloudFront, to improve content delivery speed and performance.

ReactJS, a popular JavaScript package, is used for the front-end development of the web application. The ReactJS framework succeeds in creating dynamic and interactive web application interfaces that provide a seamless and responsive user experience. The front end of a web application becomes remarkably effective when ReactJS is used, which encourages the rapid development of feature-rich and visually appealing interfaces.

The alternatives to EC2 and their drawbacks are:

EBS - AWS Elastic Beanstalk is a Platform as a Service (PaaS) product that allows application deployment and administration. It performs infrastructure provisioning, load balancing, scalability, and monitoring automatically. However, setting up the front-end environment and managing files in EC2 can be easier than in EBS since EC2 provides more flexibility and control over the setting process. Furthermore, because of its subjective nature, Elastic Beanstalk may have larger deployment times for updates.

Step functions - Step Functions, unlike EC2 instances, do not directly offer computational resources. They primarily function as a workflow coordination service, and while they can communicate with other AWS services that provide computing capabilities, their level of resource control may be limited in comparison to EC2 instances. AWS Step Functions are serverless workflow orchestrators that coordinate several AWS services. As a result, configuring and managing Step Functions might be more difficult than configuring and managing EC2 instances, which are more basic and familiar to traditional server configurations.

To host the front end in AWS EC2, I used the following configurations:

i. Amazon Machine Image

For hosting the front end, the chosen operating system is Amazon Linux 2023 (AMI ID: ami-0f34c5ae932e6f0e4), which is a 64-bit Linux system based on the x86 instruction set. To set up the instance, we use Linux commands through the 'user data' option in the EC2 cloud formation YAML to install Node Package Manager (npm) and AWS CLI.

ii. Instance ID

The web application's front end is dependent on a number of dependencies listed in the package.json file. Thus, npm is used to install all required packages and modules, which downloads and extracts them throughout the installation process. Because of the enormous number of dependencies and modules, this installation process takes a significant amount of processing power and memory.

So since the t2.micro and t2.small instances proved to be too slow for performing npm install and npm start, the t2.large instance with 2v CPUs and 8GiB of RAM was chosen to fix this. We ran into issues while attempting to SSH into the freshly formed t2.small and t2.micro instances during the process some of which were get stuck endlessly, preventing access to the instances.

iii. Security Group

To enhance the security of the instance, we configured the security group to allow SSH (port 22) and Custom port (3000) access from anywhere (CIDR notation 0.0.0.0/0). However, access to other ports has been restricted to prevent unauthorized access and potential security vulnerabilities. By limiting access to specific ports, the aim is to bolster the overall security posture of the instance and protect it from potential threats.

3.2.2 Lambda Functions

Deploying and managing backend apps created with Node.js is made easier by AWS Lambda. Lambda is a good option for Node.js-based applications because it supports many different programming languages and frameworks, including Java, .NET, Node.js, Go, Python, and Ruby. The service only executes the application code when developers upload their Node.js code to Lambda, and only when the lambda is activated. As a result, costs are reduced, and the backend code is not left running unnecessarily.

Node.js is used in the development of my web application Pinboard's backend. While using 64-bit Amazon Linux 2, Lambda supports 3 Node.js platforms: Node.js 18.x, Node.js 16.x, and Node.js 14.x. Also, as a serverless computing service, AWS Lambda enables the execution of code without the need to provision or manage servers, and based on incoming requests, it scales automatically and executes code in response to specific events. For this project, I have created 5 lambda functions with each performing a specific task. The lambdas createNote, readNote, updateNote, and deleteNote are used to perform CRUD operations on the DynamoDB table 'Notes'. The sendNotes lambda is used to send email notifications of Current Notices on the noticeboard to all the users (subscribers) of that noticeboard using AWS Simple Notification Service (SNS).

The alternatives to Lambda and their drawbacks are:

In contrast to EC2 or EBS, Lambda offers a serverless architecture, eliminating the need to manage servers or infrastructure. Additionally, Lambda easily connects with other AWS services, making it the best option for developing responsive and economical apps with little operational overhead.

EC2 – Lambdas can dynamically scale based on the number of incoming requests, guaranteeing that the application can handle different levels of traffic without manual intervention. However, for auto-scaling, EC2 requires human configuration, which may be inefficient for handling sudden spikes in demand.

EBS – Lambda's serverless nature allows for shorter development cycles and quicker iterative enhancements. On the other hand, EBS might demand more setup and administration, especially for more sophisticated applications.

To host the backend in AWS Lambda, I used the following configurations: -

i. *Runtime* – **Node.js 18.x.**

ii. *Architecture* – **x86_64 architecture.**

iii. *Default Execution Role* – **LabRole.**

iv. *Created Lambdas* – **createNote, readNote, updateNote, deleteNote, sendNotes.**

3.2.3 DynamoDB

AWS offers Amazon DynamoDB, a fully managed NoSQL database service. This high-availability service automatically scales its capacity in response to demand, ensuring efficient and seamless performance.

Pinboard's notice details are stored in an AWS DynamoDB table. When users submit notices through the front end, the data is sent to the backend in JSON format through an HTTPS request, using key-value pairs. This JSON input makes it simple to insert the data into DynamoDB, as it eliminates the need for complex queries when working with a NoSQL document database.

The alternatives to DynamoDB and their drawbacks are:

AWS Aurora – DynamoDB is a NoSQL database that allows flexible data models. As a result, development is easier and more agile, as the schema can evolve to meet the needs of the application. Aurora, on the other hand, is a relational database with a defined schema that may need more planning and effort to adjust when application requirements change.

AWS S3 – S3 is not suitable for storing notice details as it is an object storage service designed for storing and retrieving unstructured data, such as files, images, videos, and backups. Which is why it is primarily used for data storage rather than structured data querying.

I have used the following configuration for DynamoDB -

i. *Table Name* – **Notes**

ii. *Hash Key* –

For ‘notes’ table – **noteid**

iii. *ReadCapacityUnits* – 5

iv. *WriteCapacityUnits* – 5

v. *DynamoDB Stream* – **ON**

vi. *Trigger* – ‘**sendNotes**’ Lambda function.

3.2.4 API Gateway

AWS API Gateway is a practically managed service that enables developers to construct, deploy, and protect APIs. It serves as a front door for backend services, allowing clients to access APIs via specified endpoints. The service supports a wide range of HTTP protocols and authentication methods, as well as capabilities like request and response mapping, rate restriction, and caching. It connects well with other AWS services, making it a powerful tool for developing scalable and secure applications.

The basic function of Pinboard is to effortlessly show notices on the webpage. We may accomplish this by using AWS API Gateway as the service for handling API calls. API Gateway serves as an entry point to the back-end services by efficiently handling traffic and allowing for quicker notice delivery. It offers advantages such as request and response mapping, allowing for smooth integration with other AWS services. This will also help build caching and throttling algorithms with API Gateway, which optimizes speed and assures a seamless user experience. Furthermore, the authentication and access control features of API Gateway could protect API endpoints and ensure that only authorized users could access and post notices on the site.

The alternatives to API Gateway and their drawbacks are:

AWS CloudFront – API Gateway supports request and response transformation, making it possible to easily map and manipulate data between client and backend services. This enables handling data conversions and adjusting to diverse data types easier. While CloudFront is powerful for caching and content distribution, it lacks these built-in API transformation features.

AWS EventBridge – API Gateway is ideal for developing and managing HTTP-based APIs, including RESTful APIs, and is thus compatible with a wide range of client applications. EventBridge, on the other hand, is intended for event-driven architectures and event routing, and it may not be appropriate for direct client interaction over HTTP.

I have used the following configuration for API Gateway –

i. API Name – **ServerlessNotesAPI**

ii. Resource name – **/notes**

iii. Methods under '**/notes**' resource –

- **GET**: get all notices OR a specific notice when the 'noteid' is mentioned.
- **POST**: create a notice on the noticeboard.
- **PUT**: update an existing notice on the noticeboard.
- **DELETE**: delete an existing notice from the noticeboard.

3.2.5 Simple Notification Service (SNS)

SNS (Simple Notification Service) allows users to generate and send messages to a wide group of subscribers via a variety of methods, including email addresses, SMS text messages, and HTTP/HTTPS endpoints. This provides a diverse and simple method for communicating efficiently with multiple receivers through a unified service.

AWS SNS is used in Pinboard to allow users to subscribe to email notifications for new notice postings on the noticeboard. Their email addresses are added to the subscription list. When a new notice is posted or an existing notice is changed, AWS SNS sends an email to all subscribers on the list informing them of the current changes on the noticeboard. Users can unsubscribe by simply clicking on the 'unsubscribe' link supplied in their email.

The alternatives to AWS SNS and their drawbacks are:

AWS SES – You do not need to manually manage recipient lists while utilizing SNS. Subscribers can instead dynamically subscribe and unsubscribe when they want, this reduces administrative costs. SES, on the other hand, requires manual recipient list administration, which can be time-consuming and error-prone, particularly when dealing with changing recipient lists.

AWS SQS – SNS is ideal for delivering real-time push notifications to multiple subscribers. On the other hand, SQS is designed for asynchronous and reliable message processing and may not provide the same level of real-time delivery as SNS.

The following configuration has been used for SNS –

i. *Topic Name* – **MyNotes**

ii. *Protocol* – **email**

3.2.6 Backup

AWS Backup centrally manages and automates backups across AWS services. In our project, Pinboard, we have implemented a robust backup strategy using AWS Backup to ensure the safety and recoverability of the data stored in the DynamoDB table "Notes." The backup process is scheduled to occur every hour to minimize the risk of data loss. To set up this backup solution, we created a dedicated backup vault to securely store the backup data.

Following that, we created a backup plan, and we designated the "Notes" table as a resource to be backed up by the plan. In the backup plan, we defined a backup rule which defines the specifics of the backup procedure. This involves setting the backup frequency to every hour and starting backups every day at 3 a.m. UTC. The regulation also addresses the backup lifespan, ensuring that older backups are maintained and retained in accordance with our criteria. Furthermore, we identified the backup vault as the backup storage location, guaranteeing that the data is maintained securely and accessible.

Thus, by implementing AWS Backup in our Pinboard project, we have taken proactive measures to safeguard the integrity and availability of our critical data, allowing us to efficiently restore the "Notes" table in case of any unforeseen data loss scenarios. This ensures a resilient and reliable backup solution to support the smooth functioning of the application and maintain data continuity.

The alternatives to AWS Backup and their drawbacks are:

Amazon S3 Lifecycle Policies - AWS Backup is intended to handle data protection and recovery across several AWS services, resulting in an improved solution for backing up and restoring data from numerous resources. While S3 Lifecycle Policies are good for regulating the lifecycle of things stored in S3 buckets, they are only applicable to S3 and do not apply to other AWS services.

Amazon EFS Lifecycle Management - AWS Backup provides an easy and centralized method for managing backup schedules, retention periods, and backup settings across many AWS resources. It offers a uniform view of all backups and supports cross-region and cross-account backup and recovery. In contrast, Amazon EFS Lifecycle Management primarily focuses on migrating files within an EFS file system between different storage classes, such as Standard and Infrequent Access (IA), based on scheduled intervals.

The following configuration has been used for Backup –

i. *DynamoDB table* – **Notes**

ii. *Vault Name* – **dynamodb-backup-vault**

iii. *Backup Plan* – **dbBackupPlan**

iv. *Resource* – **dynamodbTable**

v. Backup Rule – **dbBackupRule**

4. Deployment Model

My web application Pinboard is deployed using the 'Public Cloud' model, in which cloud resources are owned and managed by a third-party cloud service provider such as Amazon, Microsoft, or Google. These resources are available to consumers for a price via the Internet, which allows convenient and scalable access to cloud services.

There are various benefits to using a public cloud deployment model. To begin with, the public cloud is a cost-effective solution because we just pay for the resources we use, which eliminates the need for upfront capital investments in infrastructure. Second, by utilizing a respected cloud service provider such as Amazon, we obtain access to a diverse set of scalable and trustworthy cloud resources. Furthermore, the public cloud offers high availability, worldwide reach, and powerful security features, ensuring that the application is always available and safe from any threats. Furthermore, using the public cloud, it is simple to scale the application based on demand, delivering a consistent user experience during peak periods while optimizing expenses during quieter periods. More importantly, the managed services, automated backups, and regular upgrades provided by the public cloud provider would allow developers to focus on building and upgrading Pinboard's features and capabilities rather than managing the underlying infrastructure. Overall, the public cloud deployment approach fully corresponds with business requirements, giving the flexibility, scalability, and dependability needed to build an efficient and user-friendly web application.

5. Delivery Model

Pinboard uses a Software-as-a-Service (SaaS) delivery approach to provide users with an accessible platform to read, create, update, and delete notices. The website, which is hosted on AWS, is simply accessible via web browsers, eliminating the need for users to install software or manage hardware. However, since Pinboard does not provide APIs or connections with other systems, customization, and integration features are not available. This SaaS strategy offers a smooth and user-friendly experience, allowing consumers to use the service without having to worry about software maintenance or infrastructure management.

Overall, Pinboard is an obvious SaaS delivery model due to its lack of user control over the infrastructure and software stack, as well as its lack of customization choices.

Each individual service used in this project has its own delivery model. The table below includes all the services used in the project as well as their delivery models.

Table 1: Services and their delivery model.

Service	Delivery Model
EC2	Infrastructure as a Service
Lambda	Software as a Service
DynamoDB	Database as a Service
API Gateway	Platform as a Service
Backup	Software as a Service
SNS	Software as a Service

6. Final system architecture

The system's cloud deployment is accomplished by "Infrastructure as Code" with AWS CloudFormation. This procedure entails creating a YAML file that describes the resources, their dependencies, and their configurations. This YAML file is used as an instruction manual by CloudFormation to automatically create and deploy the Pinboard web application in the cloud environment.

The system architecture in **Figure 1** depicts the final design of the web application. To successfully manage different levels of website traffic, we use the "Workload Distribution Architecture." This method involves dividing the workload across various components to efficiently manage various aspects of the application's operation.

AWS EC2 instances serve as the entry point where users interact with the application. API Gateway acts as a front-end interface, routing requests to the appropriate Lambda functions (CRUD Lambdas) responsible for handling Create, Read, Update, and Delete operations. CRUD Lambdas interact with the DynamoDB table to perform the respective CRUD operations based on User's request and Lambda's logic. **DynamoDB's "Notes" table** is where the **data** of notices is **stored**, and the Notes table is integrated with AWS Backup to perform regular backups of the data. A separate Lambda function is used to trigger notifications through AWS SNS, which sends notifications to subscribed users about any changes to the noticeboard.

Here's why the architecture fits the definition of Workload Distribution Architecture of horizontally scaled IT resources:

1. Addition of Identical IT Resources: The architecture involves the use of AWS Lambda functions for different CRUD operations, and these Lambda functions can be added in a horizontally scalable manner. As the workload increases, additional identical Lambda functions can be deployed to handle the increased demand.

2. Load Balancer for Even Workload Distribution: The architecture incorporates API Gateway as a load balancer that distributes incoming requests among the available Lambda functions. API Gateway provides runtime logic to ensure an even distribution of the workload among the Lambda functions, ensuring that the application can handle varying levels of traffic efficiently.

Overall, this architecture demonstrates horizontal scalability by adding more identical IT resources (in this case, Lambda functions) to handle the increased workload, and the load balancer (API Gateway) evenly distributes the incoming requests among the available resources.

Node.js was chosen as the programming language for Pinboard because of its asynchronous, non-blocking I/O structure, making it effective for managing concurrent tasks in web applications. Its wide ecosystem of packages (npm) provides the required modules and libraries, allowing for simple integration of third-party capabilities. The use of JavaScript for both frontend and backend development facilitated maintenance. Furthermore, Node.js's scalability and capacity to handle many concurrent connections makes it appropriate for handling changing traffic loads.

The Pinboard web application required code implementation, which included both front-end and back-end development. The front end oversaw rendering the user interface, interacting with users, and performing API requests to the backend. The backend, powered by Node.js, on the other hand, processed API requests, interacted with the DynamoDB database, and executed the essential business logic for core functionalities, such as lambdas to do CRUD operations for notices and notifications via AWS SNS. Node.js was critical across the application stack, handling user requests, maintaining data, and facilitating connectivity with external services.

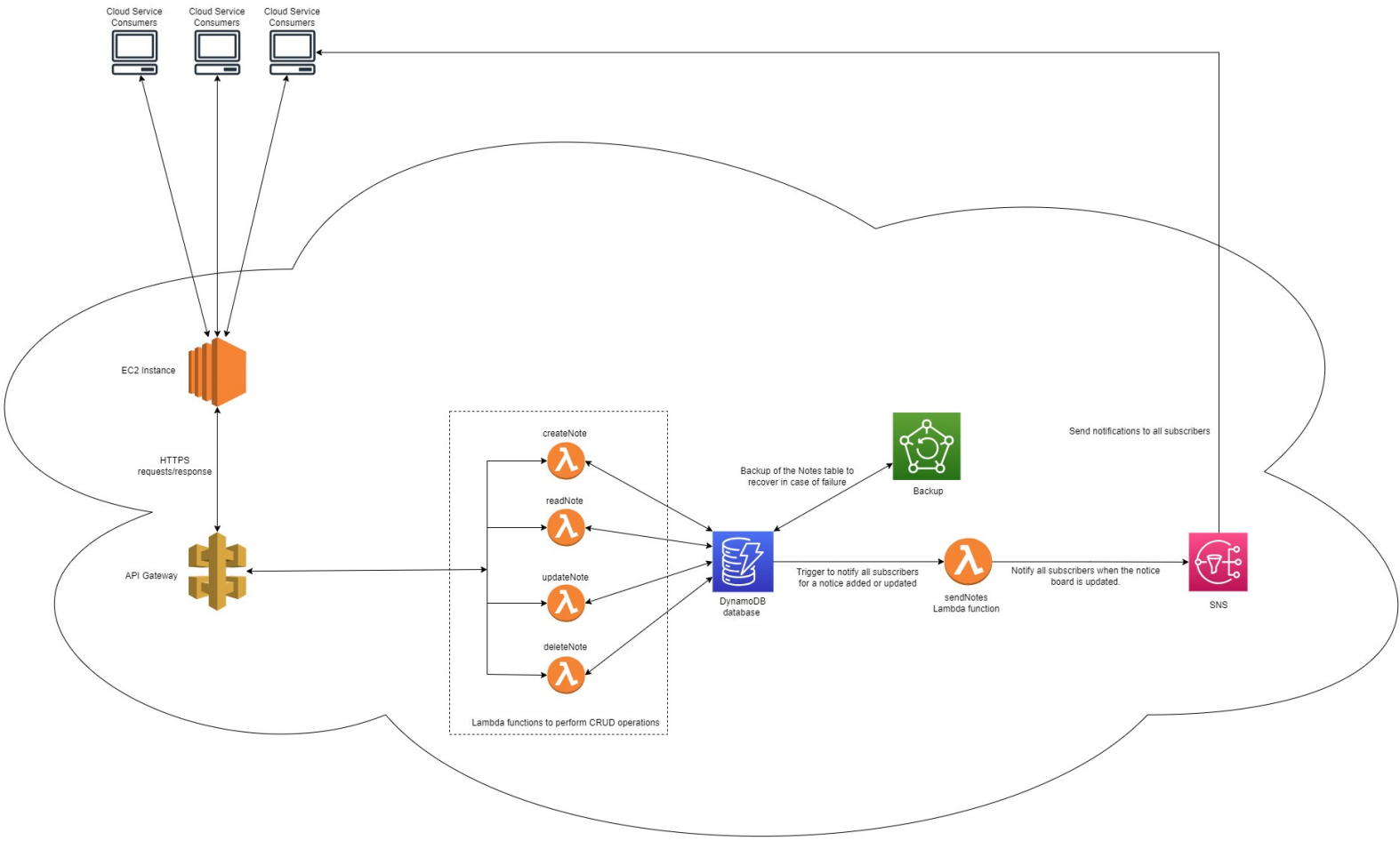

Figure 1: System Architecture of Pinboard.

7. Security Analysis

A security group has been created to function as a virtual firewall on the network, restricting access to only the appropriate ports. This setting prevents unauthorized access to the EC2 instance, hence improving the overall security of our application.

The HTTPS (HTTP safe) protocol has been used to ensure the safe transfer of user data. HTTPS protects sensitive information, such as user emails, against interception by possible attackers who use network sniffing tools like Wireshark. This approach ensures the security of user data while in transit and improves the overall security of the application.

The Lambda backend is shielded from website visitors, giving an extra degree of protection by restricting prospective attackers' access. To improve security, we use NGINX as a reverse proxy, allowing the Lambda backend to listen to port 80 rather than any custom port. The attack surface is lowered because of this technique, making it more difficult for potential attackers to target specific ports and exploit vulnerabilities. By employing these measures, we ensure that our web application has a more robust and secure design while protecting it from potential dangers.

The application utilizes a Lambda trigger in order to get access to DynamoDB's "Notes" table and the SNS topic ARN containing subscribed email addresses. Furthermore, sensitive information, including AWS credentials, is not saved in a vulnerable.env file that may be hacked. Thus, by using such methods, the application ensures possible attackers are unable to obtain access to crucial and confidential data. In general, the usage of Lambda triggers and secure processing of sensitive data improves the entire system's overall security profile.

To ensure the safety of data, our web application's architecture combines multiple security methods at various levels. Although these improvements considerably upgrade the application's security, several vulnerabilities still remain. For example, unauthorized access to the EC2 instance could endanger sensitive data by breaching the virtual firewall. So, we could reduce this risk by introducing more security controls, such as multi-factor authentication, systems for preventing and detecting intrusions, and meticulous log monitoring on the EC2 instance. Furthermore, encrypted data can be used to protect data held in storage. This means that even if an attacker gains access to the storage, they cannot read or modify the encrypted data, offering an extra degree of security to important information. By adopting these safeguards, the application's security can be strengthened as well as minimize potential dangers.

8. Cost Analysis

On Going Costs on AWS

The ongoing cost of running the Pinboard web application is calculated using AWS Calculator. The detailed report is available at: https://calculator.aws/#/estimate?id=8369ac17b1a05ff819849102ef3a1e43ee178d95

EC2 (Frontend)

Specifications – Shared instance, Linux O.S., 1 t2.large instance, on-demand pricing option for maximum flexibility.

Monthly price – USD 67.74

Name	Group	Region	Upfront cost	Monthly cost
Amazon EC2	No group applied	US East (N. Virginia)	0.00 USD	67.74 USD

Status: -
Description:
Config summary: Tenancy (Shared Instances), Operating system (Linux), Workload (Consistent, Number of instances: 1), Advance EC2 instance (t2.large), Pricing strategy (On-Demand Utilization: 100 %Utilized/Month), Enable monitoring (disabled), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month)

AWS Lambda (Perform CRUD operations and Trigger SNS)

Specifications – Invoke Mode (Buffered), Architecture (x86), Architecture (x86), Number of requests (10000 per month), Amount of ephemeral storage allocated (512 MB), Concurrency ().

Monthly Price – USD 0.00

AWS Lambda	No group applied	US East (N. Virginia)	0.00 USD	0.00 USD

Status: -
Description:
Config summary: Invoke Mode (Buffered), Architecture (x86), Architecture (x86), Number of requests (10000 per month), Amount of ephemeral storage allocated (512 MB), Concurrency ()

API Gateway (Handle API requests)

Specifications – REST API request units (millions), Cache memory size (GB) (None), WebSocket message units (thousands), HTTP API requests units (thousands), Average size of each request (34 KB), Requests (50 per month), Average message size (32 KB).

Monthly Cost – USD 0.05

Amazon API Gateway	No group applied	US East (N. Virginia)	0.00 USD	0.05 USD

Status: -
Description:
Config summary: REST API request units (millions), Cache memory size (GB) (None), WebSocket message units (thousands), HTTP API requests units (thousands), Average size of each request (34 KB), Requests (50 per month), Average message size (32 KB)

DynamoDB (Storing notice details)

Specifications – Table class (Standard), Average item size (all attributes) (1 KB), Data storage size (10 GB) Number of DynamoDB Streams GetRecord API requests (50000 per month) On-demand backup data storage (20 GB).

Monthly Price – USD 4.51

Amazon DynamoDB	No group applied	US East (N. Virginia)	0.00 USD	4.51 USD

Status: -
Description:
Config summary: Table class (Standard), Average item size (all attributes) (1 KB), Data storage size (10 GB) Number of DynamoDB Streams GetRecord API requests (50000 per month) On-demand backup data storage (20 GB)

Backup (Creating and storing the backup of DynamoDB Table)

Specifications – Hourly - Transition to cold storage (7 Days), Daily - Transition to cold storage (0 Days), Weekly - Transition to cold storage (0 Days), Monthly - Transition to cold storage (0 Days), Total DynamoDB storage (20 GB), Hourly backup retention period (21 Days).

Monthly Cost – USD 2.75

AWS Backup	No group applied	US East (N. Virginia)	0.00 USD	2.75 USD

Status: -
Description:
Config summary: Hourly - Transition to cold storage (7 Days), Daily - Transition to cold storage (0 Days), Weekly - Transition to cold storage (0 Days), Monthly - Transition to cold storage (0 Days), Total DynamoDB storage (20 GB), Hourly backup retention period (21 Days)

Simple Notification Service

Specifications – Requests (10000 per month), EMAIL/EMAIL-JSON Notifications (20000 per month), Amazon Web Services Lambda (20000 per month).

Monthly Cost - USD 0.38

Amazon Simple Notification Service (SNS)	No group applied	US East (N. Virginia)	0.00 USD	0.38 USD

Status: -
Description:
Config summary: Requests (10000 per month), EMAIL/EMAIL-JSON Notifications (20000 per month), Amazon Web Services Lambda (20000 per month)

Total Ongoing Cost

The total monthly cost for running the Pinboard web application is **USD 75.43** and the estimated yearly cost is USD **905.16** including the upfront cost.

Export date: **8/1/2023** Language: **English**

Estimate title: **My Estimate**

Estimate URL: **https://calculator.aws/#/estimate?id=cd79a9448831673ad081f7b89d29d1144d4e21cb**

Estimate summary

Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	75.43 USD	905.16 USD Includes upfront cost

8.1 Cost to reproduce the architecture in a private cloud.

Reproducing the architecture of the Pinboard web application on-premises in a private cloud would require a combination of hardware and software resources. While providing an accurate cost estimate is difficult, an approximate figure is provided below.

A. Hardware:

- High-performance servers or virtual machines to host the web application, database, and other services. Estimated cost: $5,000 to $8,000 per server.
- Network equipment, such as switches and routers, to connect the servers and ensure proper data flow. Estimated cost: $3,000 to $6,000.
- Storage systems for the database and other data. Estimated cost: $2,000 to $3,000.

B. Software:

- Operating systems and licenses for the servers. Estimated cost: $5,000 to $6,000.
- Web server software (e.g., Nginx, Apache) for serving web pages and handling API requests. Estimated cost: $1,000 to $2,000.
- Database management system (e.g., MySQL, PostgreSQL) for storing and managing data. Estimated cost: $2,000 to $4,000.

- Security software, including firewalls, intrusion detection/prevention systems, and antivirus. Estimated cost: $6,000 to $8,000.
- Monitoring and logging tools for tracking system performance and detecting issues. Estimated cost: $2,000 to $3,000.
- Backup and disaster recovery solutions to protect data and ensure data availability. Estimated cost: $3,000 to $4,000.

C. Networking and Internet Connectivity:

- Internet service provider (ISP) subscription for high-speed internet connectivity. Estimated cost: $1,000 per month.
- Network security appliances for secure access to the application. Estimated cost: $5,000 to $7,000.

D. Power Consumption:

- Servers: The server might consume around 300-500 watts on average. So, for a small cluster of servers, let's assume 5 servers, the total power consumption would be around 500-1000 watts (0.5 kW - 1 kW).
- Networking Equipment: Network switches and routers usually consume a few hundred watts. Let's assume 500 watts for networking equipment.
- Storage Systems: The power consumption of storage systems varies based on their type (HDD, SSD, etc.) and capacity. For a small storage setup, let's assume 500 watts for storage systems.

Total estimated power consumption: 1500 watts (1.5 kW) to 2000 watts (2 kW).

This much power consumption would roughly cost around $500/Month.

The total cost to re-produce the architecture on a private could be.

Up-front costs: USD 5408 – USD 7086

Monthly recurring costs: USD 2904.01 – USD 3279.17

Total yearly costs: USD 43560.29 – USD 44750.02

8.3 Cost Monitoring

AWS Lambda and EC2 instances are the two most significant cloud methods to monitor in the Pinboard web application to avoid unexpected cost escalation. AWS Lambda charges are based on the number of requests and the execution duration, whereas EC2 instances have expenses related to running and the instance type selected. Close monitoring of Lambda functions using AWS CloudWatch metrics and billing alerts, as well as monitoring EC2 instance utilization and cost via CloudWatch, can aid in the tracking of resource consumption and expenses. This allows for proactive optimization in order to stay within budget and avoid expense overruns.

9. Future Advancements

The following features could be added in the future:

i. User Profiles: Introducing user profiles where users can create accounts, customize their profiles, and manage their notices, preferences, and subscriptions.

ii. Admin Notice Deletion - Admin could delete notices that seem to be fake.

iii. Advanced Search and Filters: Enhancing the search functionality with advanced filters, tags, and categories to help users find specific notices quickly and efficiently.

iv. Location-Based Notices: Implementing location-based services to enable users to view notices specific to their geographical area, enhancing relevance and local engagement.

v. Advanced Notice Management: Offering users the ability to schedule notices for future publication and manage the duration of notice visibility.

vi. Social Sharing: Implementing social media integration to allow users to share interesting notices with their friends and followers on platforms like Facebook, Twitter, and LinkedIn.

REFERENCES

[1] "AWS Documentation," *Amazon.com*. [Online].
Available: https://docs.aws.amazon.com/ [Accessed: August 01, 2023].

[2] P. Vadapalli, "AWS architecture explained: Function, components, deployment models & advantages," *Knowledgehut*, 01-Aug-2023. [Online].
Available: https://www.upgrad.com/blog/aws-architecture/ [Accessed: July 25, 2023].

[3] "Flowchart maker & online diagram software," *Diagrams.net*. [Online].
Available: https://app.diagrams.net/ [Accessed: July 31, 2023].

[4] T. Rehemägi, "API Gateway as an Application Load Balancer?" *Dashbird*, 04-May-2021. [Online]. Available: https://dashbird.io/blog/can-api-gateway-act-load-balancer/
[Accessed: July 29, 2023].

[5] "Cloud Computing Architectures," *Brightspace* [online].
Available: https://dal.brightspace.com/d2l/le/content/274266/viewContent/3647785/View
[Accessed: July 24, 2023].

[6] "Fundamental Cloud Architectures," *Informit.com*. [Online].
Available: https://www.informit.com/articles/article.aspx?p=2093407
[Accessed: July 25, 2023].