Term Assignment

Introduction:

I am calling my project 'Course Alert', to give a brief introduction before diving into detail, I can say that: This application helps a user follow a course and track its changes.

The landing page of the application lists down all the courses it has in its database, and it allows users to follow the course, the application will prompt the user to enter their email, on successful completion of which the user will get an email to confirm their subscription for that course. Once the subscription is confirmed by the user, anytime there is any change in any of the course's availability, all the users who have subscribed to that particular course will get an email stating the change made to the course's availability.

Services used:

Categories Menu:

1. AWS Lambda
2. AWS EC2

Storage:

1. AWS DynamoDB

Network:

1. AWS API Gateway

General:

1. AWS Secrets Manager
2. AWS SNS

Justification and comparison with alternate services:

Category	Service Used	Alternate Service 1	Alternate Service 2
Compute	AWS Lambda	AWS Elastic Beanstalk	EC2
Compute	EC2	AWS Elastic Beanstalk	AWS ECS
Storage	AWS DynamoDB	AWS RDS	AWS Neptune
Network	AWS API Gateway	-	Lambda URL
General	Secrets Manger	Parameter Store	Environment Variable
General	SNS	SQS	Pinpoint

Category: Compute

Service Name: AWS Lambda

Justification: AWS Lambda is chosen for its serverless architecture, which is famous for its automatic scaling and saving cost, making it a suitable choice for the app's needs. The serverless model reduces operational overhead, and, since I only need operations like fetching course list, adding subscribers and sending emails via AWS SNS, its even driven architecture is best suited here. From a security standpoint, AWS Lambda's managed execution environment and fine-grained permissions reduce the attack surface and potential security risks compared to traditional server setups, ensuring the protection of sensitive user data and application resources [1].

Alternate Service 1: AWS Elastic Beanstalk

Justification: Although AWS Elastic Beanstalk automatically manages the environment, it might not be necessary for the straightforward frontend of 'Course Alert'. Utilizing AWS EC2 directly provides better customization and control [2], and the application can tailor the runtime environment to specific requirements, ensuring optimal performance.

Alternate Service 2: EC2 instances

Justification: Self-managed EC2 instances could be an alternative, but they require manual provisioning, scaling, and maintenance, leading to increased operational efforts. For Course Alert's simple operations, AWS Lambda's serverless approach ensures cost-effectiveness and automatic scaling, relieving the burden of infrastructure management and scaling based on demand.

Service Name: AWS EC2 (Elastic Compute Cloud)

Justification: AWS EC2 is chosen to deploy the frontend application of Course Alert, providing full control over the virtual machine instances for configurations and installations. The flexibility to set up specific software requirements makes EC2 a suitable choice [3]. From a security standpoint, EC2's security groups and IAM roles enhance control, ensuring added protection to the frontend application and its resources.

Alternate Service 1: AWS Elastic Beanstalk

Justification: While Elastic Beanstalk abstracts infrastructure management, it might not be necessary for the straightforward frontend of the application. Using EC2 directly provides better customization and control, and the application can tailor the runtime environment to specific requirements. From a business perspective, direct use of EC2 can potentially be faster and more cost-efficient for a simple frontend application.

Alternate Service 2: AWS Elastic Container Service & Elastic Container Registry

Justification: AWS ECS is a strong contender for hosting the frontend application using Docker containers. It simplifies container deployment, scaling, and management, while Elastic Container Registry (ECR) provides a secure and reliable way to store and manage Docker images [4]. This combination offers a robust and efficient platform for running Docker containers, making it a suitable choice for 'Course Alert's' frontend deployment.

Category: Storage

Service Name: AWS DynamoDB

Justification: AWS DynamoDB, a NoSQL database with a flexible schema, is suitable for Course Alert's diverse course data. Its schemaless design allows easy storage of varying data structures, and easy scalability ensures handling changing workloads easily [5]. From a security standpoint, DynamoDB's built-in encryption and access control help protect the integrity and confidentiality of course’s data.

Alternate Service 1: AWS RDS (Relational Database Service)

Justification: AWS RDS, offering traditional relational database capabilities, might not be the best fit for Course Alert due to the diverse nature of course data. Its fixed schema can be less flexible in handling varying data structures efficiently [6]. From a business perspective, DynamoDB's flexible schema and easy scalability can lead to cost savings, as it can adapt to changes in data requirements without the need for manual schema adjustments.

Alternate Service 2: AWS Neptune (Graph Database)

Justification: AWS Neptune, designed for graph-based data and complex relationships, might introduce unnecessary complexity for Course Alert, which primarily deals with course information. DynamoDB remains a better choice, providing the required flexibility, scalability, and security features without the added complexity of a graph database.

Category: Network Service

Service Name: AWS API Gateway

Justification: AWS API Gateway was chosen for 'Course Alert' due to its powerful API management features and ability to expose RESTful APIs securely. It provides request handling, caching, and security options between the application and clients. The built-in security features, such as AWS IAM authentication, ensure protection against unauthorized access, making API Gateway the ideal choice for secure routing of API requests.

Alternate Service: Using AWS Lambda Directly

Justification: Instead of utilizing AWS API Gateway, one could consider invoking AWS Lambda functions directly by generating their respective URLs. While this approach may be feasible for simple use cases, it lacks API management features that API Gateway provides. Directly using Lambda URLs might lead to challenges in managing authentication, request validation, caching, and rate limiting, which are critical for securing and scaling the 'Course Alert' application effectively. API Gateway's comprehensive capabilities and integration with AWS Lambda make it the more suitable.

Category: General

Service Name: AWS Secrets Manager

Justification: AWS Secrets Manager is chosen to securely store sensitive data, such as APIs URIs and keys, offering centralized management [7]. This makes it an ideal choice for secure secret storage, enhancing the security of Course Alert's sensitive information.

Alternate Service 1: AWS Systems Manager Parameter Store

<u>Justification</u>: Parameter Store is a viable alternative for storing configuration data and small bits of sensitive information [8]. However, AWS Secrets Manager provides more advanced features like fine-grained access control, offering better security for Course Alert's sensitive data.

<u>Alternate Service 2</u>: Environment Variables

<u>Justification</u>: While environment variables are easy to set and access, they are not recommended for storing sensitive data securely. Managing secrets directly as environment variables might lead to potential security risks. AWS Secrets Manager offers a more secure solution for managing secrets in the application, ensuring the protection of sensitive information.

<u>Service Name</u>: AWS SNS

<u>Justification</u>: AWS SNS (Simple Notification Service) is a fully managed pub/sub messaging service, making it an excellent choice for sending course update notifications to subscribers. It allowes multiple subscribers to receive notifications simultaneously, ensuring reliable delivery to all users. From a business perspective, AWS SNS's pub/sub model ensures real-time notifications, keeping Course Alert's users informed and engaged.

<u>Alternate Service 1</u>: AWS SQS (Simple Queue Service)

<u>Justification</u>: While SQS is a message queue service suitable for decoupling application components, SNS's publish-subscribe model is more appropriate for Course Alert's real-time notification requirements. It enables instant delivery of updates to all subscribers, ensuring timely communication with users.

<u>Alternate Service 2</u>: AWS Pinpoint

<u>Justification</u>: AWS Pinpoint, primarily used for sending targeted marketing messages, may not be suitable for Course Alert's event-driven model required for course update notifications [9]. SNS provides more straightforward and efficient notification capabilities, ensuring reliable delivery of updates to all subscribers.

<u>Deployment Model:</u>

Course Alert is deployed on a public cloud infrastructure, specifically utilizing Amazon Web Services (AWS) as the chosen cloud provider. I opted for a public cloud deployment model for several reasons:

<u>Justification</u> for Public Cloud Deployment:

a) Scalability: With a public cloud, Course Alert can easily handle fluctuations in user demand. AWS provides automatic scaling, allowing the application to dynamically adjust resources based on usage patterns.

b) Cost-Effectiveness: By leveraging a public cloud like AWS, Course Alert only pays for the resources it consumes. This cost-effective pay-as-you-go model eliminates the need for large upfront cost.

c) Global Accessibility: Public clouds have a vast network of data centres distributed globally. In future it can be beneficial for the application to have such an option.

d) Reliability and Redundancy: AWS provides high availability and reliable infrastructure. The public cloud's distributed nature ensures that Course Alert remains accessible even if some data centres or servers face issues. This helps enhance the overall reliability of the application.

e) Security and Compliance: Leading public cloud providers like AWS adhere to strict security practices and industry compliance standards. By leveraging the security measures implemented by AWS, Course Alert benefits from a secure environment for storing and processing sensitive user data, providing users with peace of mind [10].

Delivery Model

Course Alert follows a combination of SaaS and PaaS delivery model.

Explanation: The frontend of your application, which includes the web server running on an EC2 instance to serve the frontend application, follows the SaaS model. As a SaaS platform, Course Alert provides its services over the internet, allowing users to access the application without the need for installation or setup. Users can simply navigate to the application through a web browser, making it easy to access and use. The backend components of the app including AWS Lambda, DynamoDB, SNS, and API Gateway, act as a Platform-as-a-Service (PaaS) layer. PaaS enables users to create a subscription list by seamlessly adding themselves as followers of the courses they wish to track [11].

Benefits of the combination of SaaS and PaaS delivery model for Course Alert:

a) Scalability and Flexibility: The SaaS frontend effortlessly accommodates a growing user base without manual intervention, while the PaaS backend components (AWS Lambda, DynamoDB, SNS, and API Gateway) efficiently manage resources based on demand, ensuring smooth handling of course updates and user traffic [12].

b) Reduced Maintenance Overhead: SaaS shifts frontend infrastructure management to the service provider, and PaaS handles backend operational aspects, freeing up the development team to focus on enhancing features and delivering new functionalities, leading to a more efficient development process [12].

Architecture:

Design for the application:

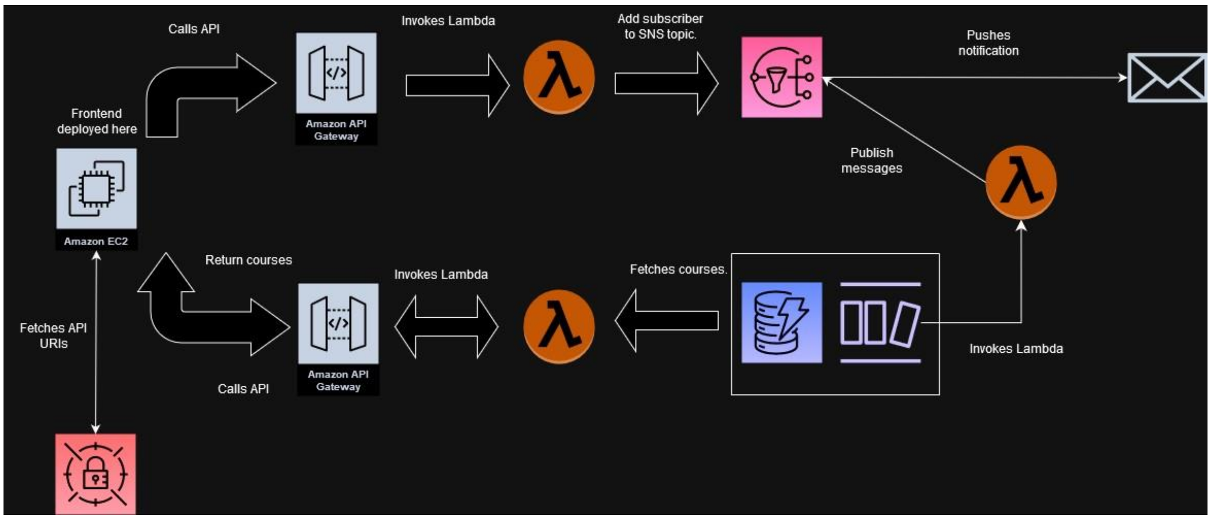

Figure 1: Application design.

Use case diagram:

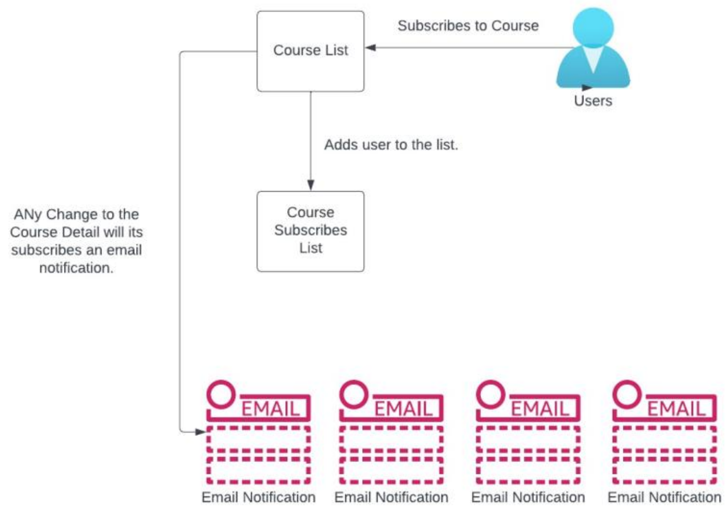

Figure 2: Use case diagram

Explanation:

1. EC2 Instance inside a public subnet, runs the web server i.e., the frontend application exposed to port 80.
2. All the API URIs and API keys are stored in Secrets Manager.
3. Web server fetches the required information from Secrets Manager.
4. Application displays the list of courses by fetching the list of courses from DynamoDB table.
5. User can interact with the application and follow courses; this action will perform a post request to an URI of an API Gateway resource with the information of the user and the followed course.

6. The action above will trigger a <u>Lambda</u> and add the user as one of the subscribers of the course.
7. The subscription here means that the <u>Lambda</u> will add the user to the <u>SNS</u> topic.
8. The action in the previous step will send an email to the user, asking to confirm the subscription to the course.
9. Now with all the subscribers and courses in place, whenever there is change in the 'availability' or number of seats of any course, <u>DynamoDB Stream</u> will trigger a <u>Lambda</u>.
10. The <u>Lambda</u> triggered in the previous step will publish a message to the <u>SNS</u> topic that changed its 'availability'.
11. The action of publishing message to the <u>SNS</u> topic will send an email to all the subscribers of the course with the relevant information about the change in the course.

<u>Data Storage</u>: Data is stored in two main services within the AWS cloud. The course information, including the list of courses, their availability, and the title, is stored in DynamoDB, a NoSQL database. Additionally, the subscribers' information, such as user details and their followed courses, is stored as a part of SNS topics. Each topic represents a specific course, and the subscribers are added to the corresponding SNS topic when they choose to follow a course. This architecture ensures that data related to courses and subscribers is efficiently managed and maintained.

<u>Language used</u>: Node.js was chosen as the primary programming language to write AWS Lambdas. Node.js is well-suited for serverless architectures and event-driven applications, making it a suitable choice for the Lambdas that handle various backend functionalities of Course Alert. The asynchronous nature of Node.js fits seamlessly with AWS Lambda's event-driven execution model, allowing the Lambdas to respond efficiently to events such as user subscriptions and course availability changes. For frontend application, 'React' library, was used. React is widely used for building responsive web applications.

<u>Deployment:</u>

The frontend application, built using JavaScript and React, is deployed on AWS EC2 instances hosted in a public subnet. These EC2 instances run the web server, making the frontend application accessible to users on port 80. EC2 directly allows for better customization and control over the runtime environment, tailored to specific requirements.

On the backend, the serverless architecture is implemented using AWS Lambda functions written in Node.js. These Lambdas are triggered in response to various events, such as user subscriptions and course availability changes. The Lambdas communicate with DynamoDB to fetch and store course data and manage subscriptions. Additionally, SNS is utilized to send notifications to subscribers when changes in course availability occur.

Overall, this deployment model leverages the scalability and cost-effectiveness of the AWS cloud services, allowing 'Course Alert' to efficiently handle user interactions, course data, and real-time notifications.

<u>Security:</u>

<u>Web Tier</u>: In the web tier, the EC2 instance hosting the frontend application is deployed within a public subnet. To ensure data security at this layer, the following measures are taken:

- Security Groups**:** The EC2 instance is associated with a security group that acts as a virtual firewall, inbound and outbound traffic has been carefully defined to expos certain ports so that the application can prevent unauthorized access to the EC2 instance.
- By hosting frontend application on EC2 instance it minimizes the attack surface. Unnecessary services, ports are disabled to reduce vulnerabilities. Additionally, regular patching and updates are applied to the OS and software running on the instance.

Application Tier (Lambda and API Gateway): The application tier consists of AWS Lambda functions and the API Gateway. To maintain data security at this layer, the following steps are taken:

- Secrets Management**:** Sensitive API URIs and keys are stored securely in AWS Secrets Manager. This helps prevent accidental exposure of sensitive information and restricts access to authorized Lambda functions that require this data.
- Lambda Execution Role**:** Each Lambda function is associated with an execution role, which determines its permissions and access to AWS resources. By carefully configuring these roles, the principle of least privilege is enforced, limiting Lambda functions' access to only the necessary resources.

Database Tier (DynamoDB): AWS DynamoDB is used to store course data in the database tier. To maintain data security at this layer, the following steps are taken:

- Encryption at Rest**:** DynamoDB offers encryption at rest, which ensures that the data stored in the database is encrypted, reducing the risk of unauthorized access to data even in case of physical compromise.
- Least Privilege Principle**:** In the Course Alert application, access permissions are configured using the principle of least privilege. Each AWS Lambda function and other components interacting with DynamoDB are granted only the minimum set of permissions required to perform their specific tasks.

SNS:

- Secure Message Delivery**:** AWS SNS ensures secure message delivery to all subscribers. With SNS, messages are transmitted securely over HTTPS, preventing interception during transmission.
- Avoiding Direct Exposure of Subscriber Data**:** Since SNS handles the delivery of messages, subscribers' email addresses are not directly exposed to other components of the application. This helps prevent potential leaks of the subscribers’ data.

Vulnerabilities:

Secure Communication between EC2 and API Gateway**:** While the frontend application uses HTTPS to encrypt communication between users and the EC2 instance, the communication between the EC2 instance and the API Gateway may still be exposed to potential risks. Implementing mutual TLS (Transport Layer Security) authentication between EC2 and API Gateway can enhance security by ensuring that both endpoints authenticate each other during communication.

<u>Reproduce the architecture in private cloud:</u>

<u>Platform:</u>

1. Private cloud platform such as OpenStack can be used to create and manage your private cloud infrastructure.

<u>Compute: OpenStack Compute (Nova) can be used to manage compute resources.</u>

2. Web server (frontend application): Deploy web server software (Nginx) on virtual machines within the private cloud to host and serve the frontend application.
3. Application Server: Set up virtual machines or containers running the application server to handle user requests and implement business logic.
4. Cost:
    a. Requirement: For Ngnix and application server we can estimate for a server with 1 CPU with dual core for small business with 8 GB RAM, and since my frontend and backend application is small, disk storage of around 50 GB would be more than enough.
    b. The estimated cost of such server could be approximately $100 per month, depending on the cloud provider and specific configurations chosen.

<u>Storage/Database:</u>

5. Database Server: Deploy a database server on virtual machines or containers to store and manage course data securely. Almost all NoSQL databases provide data stream which can be utilized here.
6. We can add a listener which will monitor for the data changes via data stream and will call an API which will send an email to all the subscribers.
7. Cost [14]:
    a. Requirement: We are planning to keep course list for the current semester only. If we assume to have 1000 courses with 100 subscribers per course on average.
    b. We can plan to go with MongoDB Dedicated Standard Server, majorly because we need security and also change stream feature of MongoDB, which is not supported by MongoDB serverless.
    c. Assuming we will need around 1000 changes/month, we will have to go with Standard clusters (M30/M40).
    d. If we go with M30 with 40GB storage, it can cost around $0.5/hour i.e., $360 per month.

<u>Networking: Neutron (OpenStack networking system) can be utilized here</u>

8. A network can be created using OpenStack.
9. Web server will have floating IP addresses bought from ISP.
10. DNAT can be used for application and database server, a rule can be added for DNAT that it won't allow any inbound rules from internet.
11. Routers can be established which will allow webservers to communicate with application servers etc.
12. We can use name resolution to help communicate between servers within same network in OpenStack.

Security:

1. We are already using public and private network using OpenStack for our web and application & Database servers respectively.
2. We can use OpenStack security groups and configure rules to specify network access rules.
3. Most NoSQL databases provides encryption for data at rest.
4. For in transit data in NoSQL databases, we can configure it with SSL/TLS to ensure data security.

Total Cost:
Since we are using OpenStack which is a open source, we can say that adding up the cost mentioned above, the total cost of total infra in private cloud could be around $460/month.

Components' cost & Monitoring:

These components/elements can cost most money:

- AWS Lambda: As the project uses Lambda extensively to trigger various actions, the number of Lambda function invocations and execution time can impact costs.
- DynamoDB: Storing and managing course data in DynamoDB, especially with increased data and read/write operations, can contribute significantly to expenses.
- Amazon SNS: Frequent messaging and email delivery through SNS may lead to additional costs, especially as the number of subscribers increases.

Monitoring tools:

Cost Explorer can be used to track the costs of all three services mentioned above. By filtering it by service name, it enables monitoring of costs related to that service. By filtering by service in Cost Explorer, you can gain insights into the individual cost components of each service and efficiently manage your overall cloud expenses. We can also use tags with our services and filter via tags.

Features for future:

1. Personalized Alarms: Each subscriber of a course can set thresholds for themselves, under what conditions they want to be notified.
2. View peers' profile: User can have the option to public their profile, if it is public, others can see their name under the subscriber list for the course.

References:

1. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
2. https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html
3. https://aws.amazon.com/ec2/instance-types/
4. https://docs.aws.amazon.com/AmazonECR/latest/userguide/ECR_on_ECS.html
5. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
6. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html
7. https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html

8. https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-parameter-store.html
9. https://docs.aws.amazon.com/pinpoint-email/latest/APIReference/Welcome.html
10. https://www.pluralsight.com/resources/blog/cloud/the-basic-advantages-of-public-cloud
11. https://blog.hubspot.com/service/iaas-paas-saas
12. https://www.sam-solutions.com/blog/iaas-vs-paas-vs-saas-whats-the-difference/
13. https://www.mongodb.com/pricing