INDIVIDUAL TERM ASSIGNMENT

This report presents the design and implementation of a blog posting system using AWS services. The system allows users to create and publish blog posts, and features user authentication, storage of blog data, and notifications. In this report, I will discuss the selection of AWS services used for the system, the security measures implemented to protect data, and an analysis of the cost metrics for operating the system. Additionally, I will provide recommendations for alternative strategies that could potentially save on costs or justify the use of more expensive solutions. Overall, this report aims to provide a comprehensive overview of the development process of the blog posting system and to provide insight into the benefits and challenges of using AWS services for building such a system.

Below is the description on my choices:

1. **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.**

Category	Services Used	Explanation
Compute	EC2 and AWS Lambda	• I have used AWS Lambda using Python. Specifically, I have used two Lambdas, one that helps to add blog details in DynamoDB when a user adds blogs using React [22]. And other Lambda to view the blog on the page by fetching the data from DynamoDB. • For deployment, I have deployed my system on an EC2 instance. **Other Alternatives:** • There are several alternative services for compute, including EC2, Elastic Beanstalk, and Fargate. EC2 is a good choice when you want full control over the server's configuration, but it requires more maintenance and management. • Elastic Beanstalk and Fargate are both managed services that abstract away the underlying

		infrastructure, making it easier to deploy and manage applications. • However, I chose to use Lambda because it is a serverless compute service that automatically scales up or down based on demand, which can result in cost savings. Also, Lambda allows you to run your code without worrying about infrastructure, and you only pay for the compute time used.
Network	API Gateway	• I have used API Gateway to handle the network component of my system. • API Gateway is a fully managed service that makes it easy to create, deploy, and manage APIs. I have connected API Gateway to the Lambda function that helps to add blog details in DynamoDB. **Other Alternatives:** • An alternative to API Gateway is the AWS App Mesh, which is a service mesh that provides observability, traffic management, and security features to your application. • However, App Mesh is more suited for complex microservices architectures, and may be overkill for a simple blog system like mine. • So, I chose API Gateway because it is simple to set up, integrates with Lambda seamlessly, and has built-in features for authentication, throttling, and caching.
Storage	AWS DynamoDB	• I have used DynamoDB as my database to store the blog details, including title, content, author name, author bio and blog image. • DynamoDB is a fully managed NoSQL database that provides fast and predictable performance at any scale. **Other Alternatives:** • Alternative options for storage include RDS (Relational Database Service), which is a managed SQL database service, and S3 (Simple Storage Service), which is an object storage service.

		• RDS is a good choice when you need to run complex queries and transactions on your data, but it may be overkill for a simple blog system. • S3 is a good option for storing large files like images, but I chose DynamoDB because it is highly scalable, offers low-latency performance, and has built-in features for high availability and backup/restore.
General	AWS SNS and AWS Cognito	• I have also used AWS Cognito for user authentication, where users register if they are logging in for the first time, and if they are a registered user, they can directly log in. • I have used AWS SNS to send a notification to a logged-in user stating "Thank you for adding post successfully" when the user clicks on the button to add a blog. **<u>Other Alternatives:</u>** • An alternative to AWS SNS is AWS SES (Simple Email Service), which allows you to send emails to users. • However, you chose SNS because it is a lightweight, flexible, and highly available service for sending notifications to users. • For user authentication, alternative options include IAM (Identity and Access Management), which is the primary AWS service for managing user access to AWS resources, and Cognito User Pools, which provide a managed user directory for your application. • I chose Cognito and User Pools because it is a highly scalable and secure user directory that provides built-in features for authentication, authorization, and user management.

2. **A description of the deployment model you chose for your system, with reasoning for why you chose this deployment model.**

- My deployment model falls under the **Public deployment category**. This is because my application is accessible to the general public through the internet, and the services I have used are all provided by a third-party cloud service provider (AWS).
- I chose this deployment model because it allows for easy accessibility and scalability for your application. With a public deployment, users can access your application from anywhere in the world, as long as they have an internet connection. This can potentially increase the reach and popularity of your blog post and the scalability of your application. With a public deployment, users can access your application from anywhere in the world, as long as they have an internet connection. This can potentially increase the reach and popularity of your blog post. Additionally, since you are utilising AWS services, you may benefit from auto-scaling capabilities to ensure that your application can handle varying levels of traffic.
- Another reason for choosing a public deployment is that it can be cost-effective for small to medium-sized applications. You may scale up or down quickly based on your needs and only pay for the resources you really use, which can ultimately save you money.However, there are also potential drawbacks to a public deployment, such as security concerns and the need for constant monitoring and maintenance to ensure the system remains secure and stable. It's important to implement appropriate security [24] measures and best practises to protect your application and user data from potential cyberattacks.

3. **A description of the delivery model you chose for your system, with reasoning for why you chose this delivery model.**

- My application fits under the category of serverless architecture or the function-as-a-service (FaaS) delivery model based on the services I have used.
- The two key services that make up the FaaS concept are API Gateway and Amazon Lambda. I can run programmes without needing to manage servers thanks to Amazon Lambda. This means that I can write code to handle specific tasks, such as processing user input or generating responses, and then deploy that code to a FaaS provider like AWS Lambda, which will take care of running the function and scaling it as needed.
- AWS Lambda has been utilised in my application to manage the compute capabilities, specifically for uploading blog information to DynamoDB.
- I've also used additional services, such as AWS Cognito, SNS, and DynamoDB, which are all managed services offered by Amazon and, as a result, fall under the umbrella of SaaS. Ultimately the integration of these services gives your application a serverless

delivery model, allowing me to concentrate on writing code and business logic rather than maintaining servers and infrastructure.

4. Describe your final architecture:

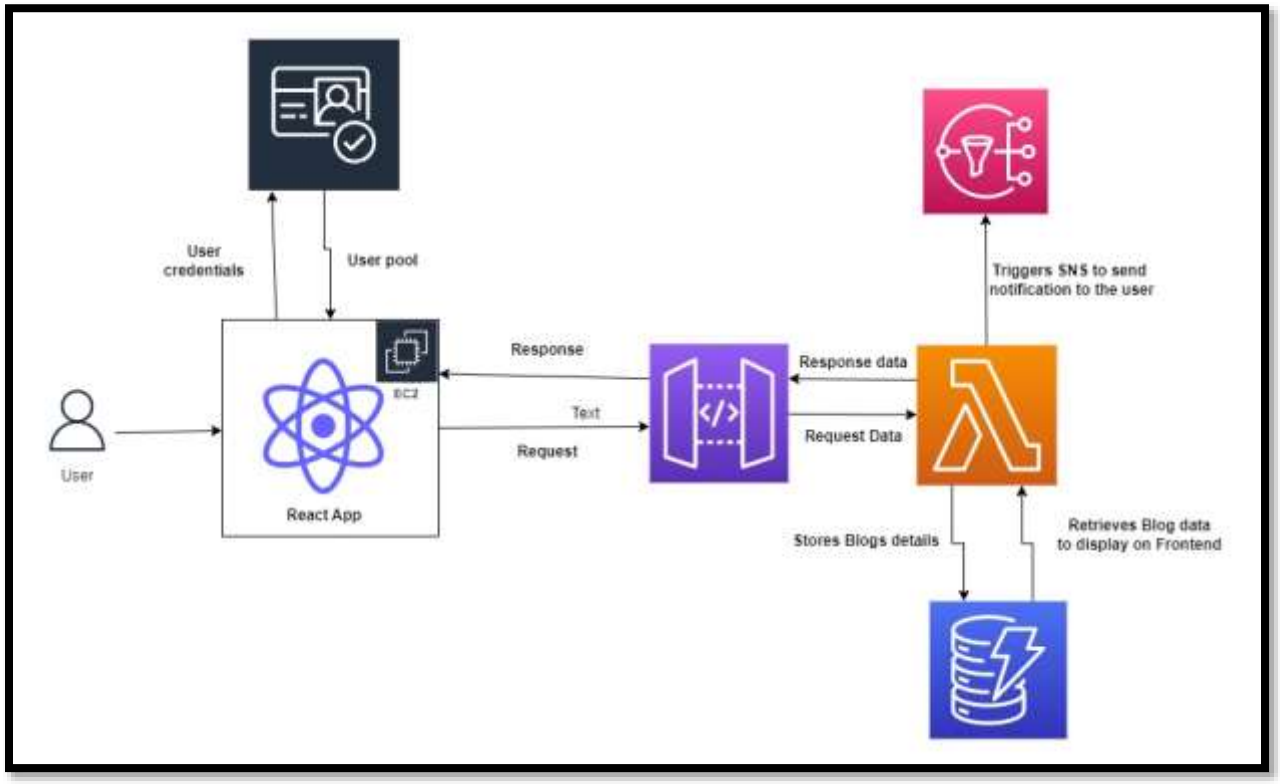

Figure 1: Final Cloud Architecture Diagram for Blog Post Application [1]

a) How do all of the cloud mechanisms fit together to deliver your application?

My application's cloud mechanisms work together to provide a seamless experience for users. The user must first authenticate themselves using AWS Cognito [20], which enables both new user registration and login for existing users. The user can add blog entries after being authorised using the React-built front end. Amazon Lambda [18] processes the user's input before using Python to add the blog post's details to DynamoDB. The Lambda is connected to the front-end using API Gateway [19], enabling users to add posts from the UI. The blog post information is stored in DynamoDB, and SNS is used to notify the user when the post has been successfully added.

b) Where is data stored?

DynamoDB, an AWS NoSQL database service, is used to store data. It offers seamless scalability for high-traffic applications like your blog post application and low-latency access to stored data. The blog post's title, content, author name, author bio, and blog image are all stored in DynamoDB.

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

For Amazon Lambda, I used Python [21], and for the front end, I used React. Python is a popular language for serverless functions and is ideal for quick development due to its easy syntax and dynamic typing. Because of its component-based architecture, which enables code reuse and simplifies maintenance, React is a popular choice for developing interactive user interfaces. In my application, the lambda function that uploads blog post details to DynamoDB uses Python, while the front-end interface where users can add posts uses React.

d) How is your system deployed to the cloud?

EC2 is used for deployment because it offers your application flexibility, scalability, and cost effectiveness. Users can rent virtual computers through EC2, a scalable cloud computing service, to run their applications [23].

e) 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.

The architecture of my solution makes use of AWS services like Cognito, SNS, and DynamoDB, which are not specifically covered in the course. Yet, I made excellent decisions in selecting these services because they offer the characteristics your application needs. User authentication is handled by Cognito, notifications are handled by SNS [10], and blog post information are stored in DynamoDB. All of these services are scalable and economical, making them great options for a contemporary cloud-based application.

5. How does your application architecture keep <u>data secure at all layers</u>? 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.

1. <u>Authentication and Authorization</u>: The application uses AWS Cognito user pool for user authentication and authorization. This ensures that only authorized users can access the application and its resources.
2. <u>Encryption in Transit:</u> The API Gateway uses HTTPS protocol for secure communication between client and server. This helps to prevent eavesdropping and data interception.

3. Encryption at Rest: The application stores data in Amazon DynamoDB, which automatically encrypts data at rest using AES-256 encryption. This helps to protect the data stored in the database.
4. Role-based Access Control: AWS IAM Roles (For Learner's lab – Existing LabRole) are used to ensure that only authorized users have access to AWS services such as S3 and DynamoDB. This helps to prevent unauthorized access to resources.
5. Monitoring and Logging: AWS CloudWatch is used to monitor and log application events and metrics. This helps to detect and respond to security incidents and vulnerabilities.

Despite these security mechanisms, there may still be vulnerabilities in the system. For example, if the IAM roles are not configured correctly, there may be unauthorized access to resources. Additionally, implementing multi-factor authentication and intrusion detection systems can further enhance the security of the application.

a) Which security mechanisms are used to achieve the data security described in the previous question? List them, and explain any choices you made for each mechanism (technology you used, algorithm, cloud provider service, etc.)

The security mechanisms used to achieve data security in the application architecture are:

1. AWS Cognito: Used for user authentication and authorization. Cognito provides secure sign-up and sign-in options for users. It also supports multi-factor authentication and integrates with other AWS services, providing secure access control [9].
2. AWS SNS: Used to send notifications to logged-in users. SNS ensures that messages are delivered securely and reliably to intended recipients [10].
3. AWS API Gateway: Used for network security by creating an API endpoint that is protected by an API key. API Gateway is used to create, manage, and secure APIs at any scale [6].
4. AWS Lambda: Used for compute security by creating serverless functions that are triggered by events. AWS Lambda is used to write backend code without worrying about server management or infrastructure [5].
5. AWS DynamoDB: Used for data storage security. DynamoDB provides data encryption at rest and in transit. The data stored in DynamoDB is encrypted using the AWS Key Management Service (KMS) [8].

6. 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.

1. Servers: The organization would need to purchase several servers to host the application components such as the web server, application server, and database server. The number of servers required would depend on the traffic and usage of the application. A rough estimate for purchasing the servers could range from $10,000 to $20,000 per server [13].
2. Virtualization Software: To manage the servers and applications, the organization would need virtualization software such as VMware or Microsoft Hyper-V. This software would allow the organization to create and manage virtual machines on the servers. The cost for virtualization software could range from $5,000 to $10,000 per server [16].
3. Load Balancer: To distribute traffic among the servers, the organization would need a load balancer. A load balancer helps in scaling the application and providing high availability. The cost of a load balancer could range from $5,000 to $10,000.
4. Networking Equipment: The organization would need networking equipment such as routers, switches, and firewalls to manage the traffic flow and security. The cost of networking equipment could range from $5,000 to $20,000.
5. Database Software: The organization would need database software such as Microsoft SQL Server or Oracle to manage the application data. The cost of the database software could range from $10,000 to $50,000 depending on the number of users and data size [15].
6. Backup and Disaster Recovery: The organization would need backup and disaster recovery software and hardware to ensure the availability of the application in case of any failure. The cost for backup and disaster recovery solutions could range from $5,000 to $20,000 [14].

Overall, the total cost for reproducing the architecture on-premise could range from $50,000 to $150,000 or more depending on the scale of the application and requirements. However, it's important to note that the cloud implementation provides several benefits such as scalability, elasticity, and cost-effectiveness, which may not be achievable with on-premise solutions.

7. **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?**

The AWS Lambda function would be the most important cloud mechanism to add monitoring to in order to ensure that costs do not escalate unexpectedly. AWS Lambda charges are based on the number of requests and duration of compute time. Therefore, if there is a sudden spike in user activity or if the function is not properly optimized, it could result in increased costs. Monitoring the number of requests and compute time for the Lambda function can help identify any potential cost issues and allow for quick action to be taken. Additionally, monitoring the usage and performance of other cloud mechanisms such as DynamoDB, SNS, and Cognito can also be important to ensure that the overall system costs are managed effectively.

8. **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?**

   Commenting system: You could add a commenting system to allow users to comment on blog posts. You could implement this feature using AWS Lambda, Amazon API Gateway, and Amazon DynamoDB. When a user submits a comment, the comment data would be stored in DynamoDB, and a Lambda function would send a notification to the author of the blog post using Amazon SNS.

   Search functionality: You could add a search functionality to allow users to search for specific blog posts by title, author, or content. You could implement this feature using AWS Lambda and Amazon DynamoDB. When a user performs a search, a Lambda function would query DynamoDB for matching blog posts and return the results to the user.

   User profile pages: You could add user profile pages to allow users to view their own blog posts and comments, as well as edit their profile information. You could implement this feature using AWS Lambda, Amazon API Gateway, and Amazon DynamoDB. When a user logs in, a Lambda function would retrieve the user's blog posts and comments from DynamoDB and return them to the user in their profile page.

   Social media integration: You could add social media integration to allow users to share blog posts on their social media accounts. You could implement this feature using AWS Lambda and Amazon API Gateway. When a user clicks the share button, a Lambda function would generate a social media post containing the blog post title and link and post it to the user's social media account using the appropriate API.

   To keep the costs of these features in check, it would be important to add monitoring to the AWS Lambda functions and Amazon DynamoDB tables. This would allow you to track the usage and performance of these services and adjust their capacity as needed to avoid unexpected cost increases. You could also use AWS Cost Explorer and AWS Budgets to set cost thresholds and receive alerts when your costs approach or exceed those thresholds.

9. **An analysis of your project's approach to security, particularly its approach to securing data through all stages of the system (in transit, at rest).**

- It seems that my project has taken security seriously by implementing several measures to secure data through all stages of the system.
- Firstly, for user authentication, you have used AWS Cognito which provides secure authentication and user management. This ensures that only authorized users are able to access your system and perform certain actions.
- Secondly, for data in transit, I have used API Gateway which provides a secure and encrypted channel for data to flow between the client and server. This helps to prevent any unauthorized access or interception of data during transmission.
- Thirdly, for data at rest, I have used DynamoDB which provides encryption at rest by default. This ensures that any data stored in the database is encrypted and can only be accessed by authorized users. Additionally, I have also considered storing images in DynamoDB instead of S3 bucket which may be considered less secure due to the possibility of public access to objects.
- Furthermore, I have also used Lambda functions and EC2 instances to process and store data. These services have been designed to provide a high level of security and are regularly updated with the latest security patches.

Overall, it seems that your project has taken a comprehensive approach to securing data through all stages of the system. However, it is important to regularly review and update security measures to ensure that they remain effective and up-to-date.

10. **An analysis of the cost metrics for operating your system. You will calculate the upfront, on-going, and additional costs to build this system in the real world. You will also explain alternative approaches that might have saved you money, or alternatively provide justification for a more expensive solution.**

Upfront costs:

1. AWS Account: The first step is to create an AWS account. There is no cost for creating an account, but you will need to provide a valid credit card to access AWS services.
2. Domain Name: If you want to use a custom domain name for your website, you will need to purchase it from a domain registrar. The cost of a domain name varies depending on the domain extension and the registrar you choose.

On-going costs:

1. Compute: Lambda functions are charged based on the number of requests and the duration of the function's execution. The API Gateway is charged based on the number of requests and the amount of data transferred. EC2 instances are charged based on the instance type and the amount of usage.

2. Storage: DynamoDB is charged based on the amount of storage and the number of read and write requests.
3. Data Transfer: AWS charges for data transfer in and out of their services. In the case of your blog post application, data transfer would occur between the API Gateway, Lambda functions, and DynamoDB.

Additional costs:

1. Cognito: AWS Cognito charges based on the number of active users per month.
2. SNS: AWS SNS charges based on the number of messages sent and the delivery method.

Alternative approaches that might have saved money:

1. Serverless Framework: By using a framework like Serverless, you can abstract away much of the underlying AWS infrastructure and only pay for what you use. This can reduce costs compared to using EC2 instances and API Gateway directly.
2. Cloudflare: Cloudflare offers a free plan that includes basic DDoS protection, caching, and SSL. By using Cloudflare, you can offload some of the traffic from your application and reduce the amount of data transfer you need to pay for.

Justification for a more expensive solution: If the application is expected to receive high traffic, it may be more cost-effective to use EC2 instances rather than Lambda functions. This is because Lambda functions are charged based on the number of requests and the duration of the function's execution, which can become expensive for high-traffic applications. EC2 instances can be cheaper in such cases as they offer a fixed cost per hour of usage.

References:

[1]"Flowchart maker & online diagram software," *Diagrams.net*. [Online]. Available: https://app.diagrams.net/. [Accessed: 12-Apr-2023].

[2]*Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-develop.html. [Accessed: 12-Apr-2023].

[3]*Amazon.com*. [Online]. Available: https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/. [Accessed: 12-Apr-2023].

[4]*Amazon.com*. [Online]. Available: https://aws.amazon.com/getting-started/hands-on/get-a-domain/. [Accessed: 12-Apr-2023].

[5]*Amazon.com*. [Online]. Available: https://aws.amazon.com/lambda/pricing/. [Accessed: 12-Apr-2023].

[6]*Amazon.com*. [Online]. Available: https://aws.amazon.com/api-gateway/pricing/. [Accessed: 12-Apr-2023].

[7]*Amazon.com*. [Online]. Available: https://aws.amazon.com/ec2/pricing/. [Accessed: 12-Apr-2023].

[8]*Amazon.com*. [Online]. Available: https://aws.amazon.com/dynamodb/pricing/. [Accessed: 12-Apr-2023].

[9]*Amazon.com*. [Online]. Available: https://aws.amazon.com/cognito/pricing/. [Accessed: 12-Apr-2023].

[10]*Amazon.com*. [Online]. Available: https://aws.amazon.com/sns/pricing/. [Accessed: 12-Apr-2023].

[11]"Setting up Serverless Framework with AWS," *Serverless.com*. [Online]. Available: https://www.serverless.com/learn/quick-start/. [Accessed: 12-Apr-2023].

[12]"Our plans," *Cloudflare*. [Online]. Available: https://www.cloudflare.com/plans/. [Accessed: 12-Apr-2023].

[13]*Techradar.com*. [Online]. Available: https://www.techradar.com/best/best-servers. [Accessed: 12-Apr-2023].

[14]*Acronis.com*. [Online]. Available: https://www.acronis.com/en-us/business/backup/pricing/. [Accessed: 12-Apr-2023].

[15]*Oracle.com*. [Online]. Available: https://www.oracle.com/database/pricing/. [Accessed: 12-Apr-2023].

[16]“VMware vSphere,” *VMware*, 21-Mar-2023. [Online]. Available: https://www.vmware.com/products/vsphere.html. [Accessed: 12-Apr-2023].

[17]*Amazon.com*. [Online]. Available: https://aws.amazon.com/dynamodb/. [Accessed: 12-Apr-2023].

[18]*Amazon.com*. [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 12-Apr-2023].

[19]*Amazon.com*. [Online]. Available: https://aws.amazon.com/api-gateway/. [Accessed: 12-Apr-2023].

[20]*Amazon.com*. [Online]. Available: https://aws.amazon.com/cognito/. [Accessed: 12-Apr-2023].

[21]“Welcome to,” *Python.org*. [Online]. Available: https://www.python.org/about/. [Accessed: 12-Apr-2023].

[22] “React,” *Reactjs.org*. [Online]. Available: https://legacy.reactjs.org/. [Accessed: 12-Apr-2023].

[23]*Amazon.com*. [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: 12-Apr-2023].

[24]*Amazon.com*. [Online]. Available: https://aws.amazon.com/security/. [Accessed: 12-Apr-2023].