CSCI 5409 - Cloud Computing

Term Project

Submitted by –

Name - Alen Santosh John

Banner ID – B00930528

Email – al283652@dal.ca

Content

Project Specifications 2
Project Description 2
Figure 1: Home page 3
Figure 2: Registration Page 3
Figure 3: Login Page 4
Figure 4: Create a new blog 5
Figure 5: Create a new blog 5
Figure 6: Email Notifications 6
Target Users: 6
Description: 6
Services Incorporated 6
Deployment Model 7
Delivery Model 8
Final Architecture 9
How all the cloud mechanisms fit together: 9
Where is the data stored: 10
What programming language I used: 10
Figure 7: Email Notifications 11
Figure 8: Fetching project artifacts from S3 11
How is my system deployed to the cloud: 11
Figure 9: Cloud Formation Stacks 12
Figure 10: Cloud Formation Stack “Blog-it-UAT” 12
What architecture am I using: 12
Architecture Diagram: 13
Figure 11: Blog it Architecture 14
Application Security 14
Cost to reproduce in a private cloud 15
Cost of cloud resources 16
Upcoming Features for Blog it 17
References 17

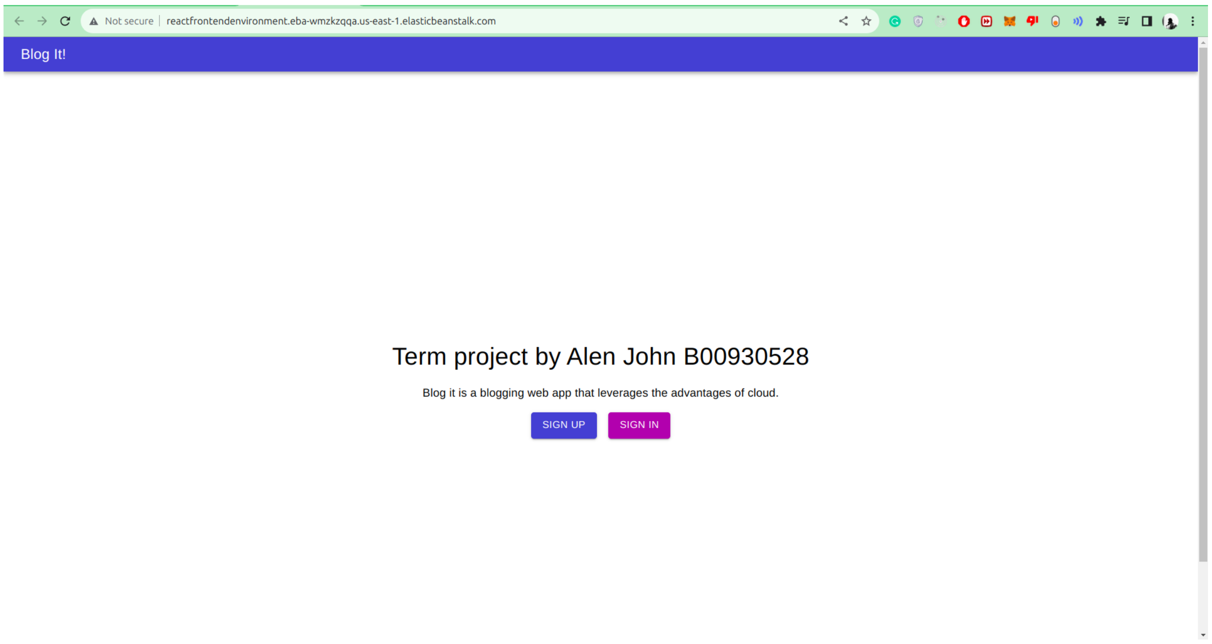
Project Specifications

Project Description

My project is a web application built with react js [6] called “Blog It”. It is a user-friendly social media application that can be used to create blogs. The main purpose of the application is to enable users to create, publish and view blogs easily. The technology stack used to build the application consists of React.js for the frontend, AWS Lambdas [1] for the backend functionality, API Gateway to expose the Lambdas through a REST API, DynamoDB [2] for storing the user information, SNS [3] & SQS [4] for efficient admin operations and S3 [5] bucket to store all the applications static assets.

The core features and functionality of my app are as follows:

1. **User Registration and Authentication:** My application enables users to register with their email address and create a new account. Authenticated users can securely log in with their email and password. **(See Figures 1, 2 & 3)**

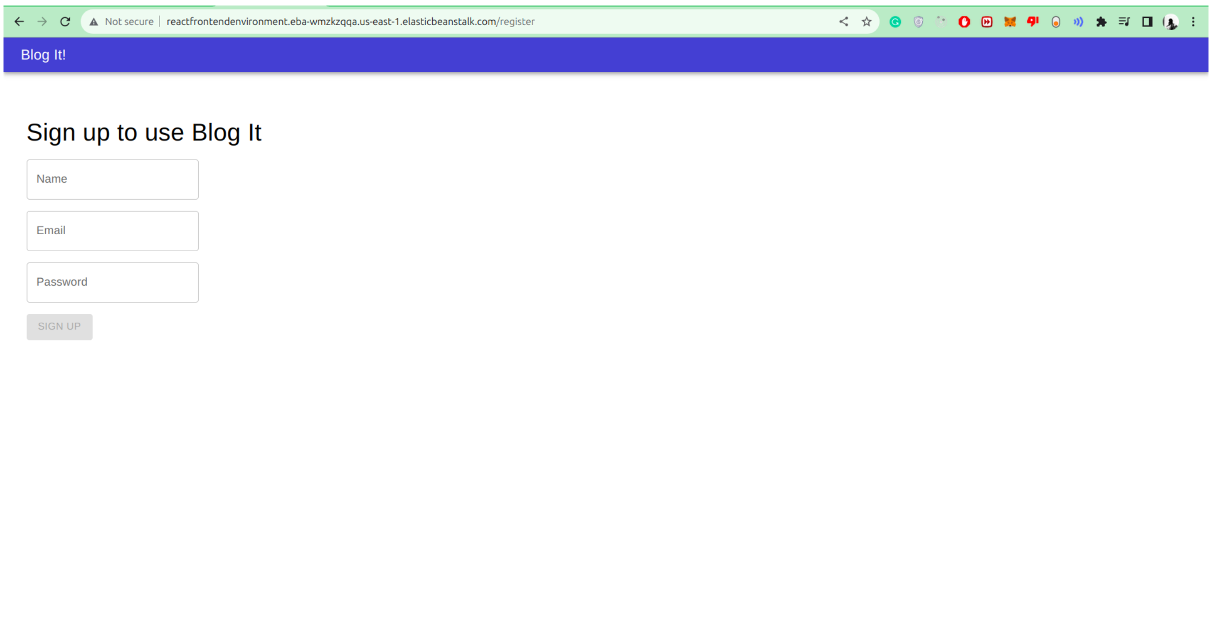


Not secure | reactfrontendenvironment.eba-wmzkzqqa.us-east-1.elasticbeanstalk.com
Blog It!
SIGN UP
SIGN IN

Term project by Alen John B00930528

Blog it is a blogging web app that leverages the advantages of cloud.

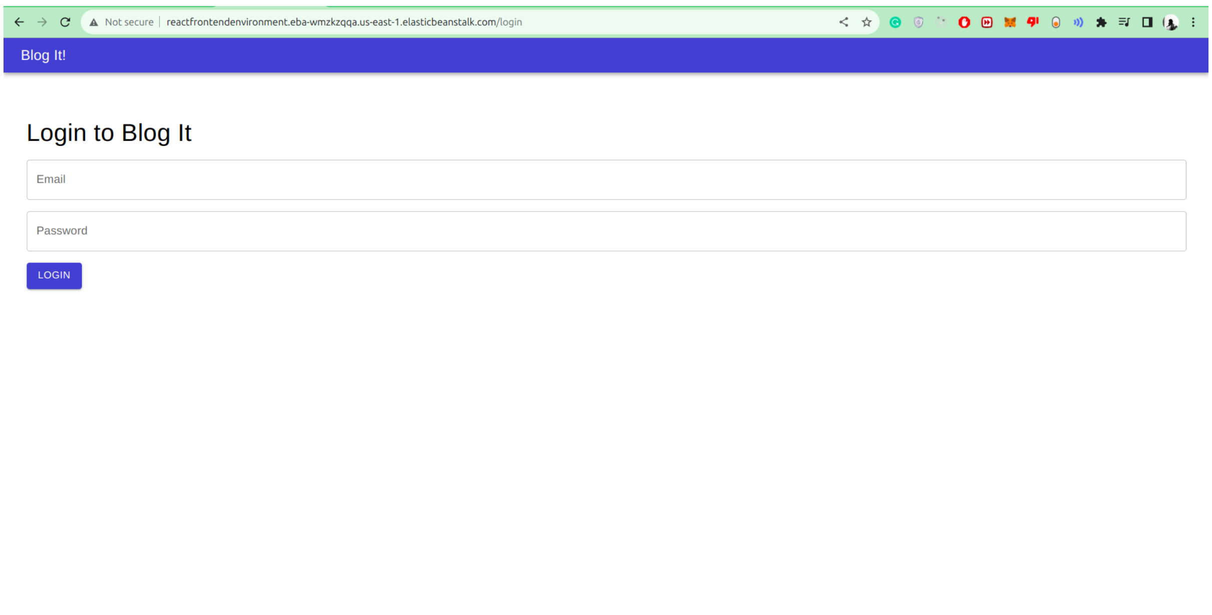
Figure 1: Home page


Not secure | reactfrontendenvironment.eba-wmzkzqqa.us-east-1.elasticbeanstalk.com/register

Blog It!

Sign up to use Blog It

Figure 2: Registration Page

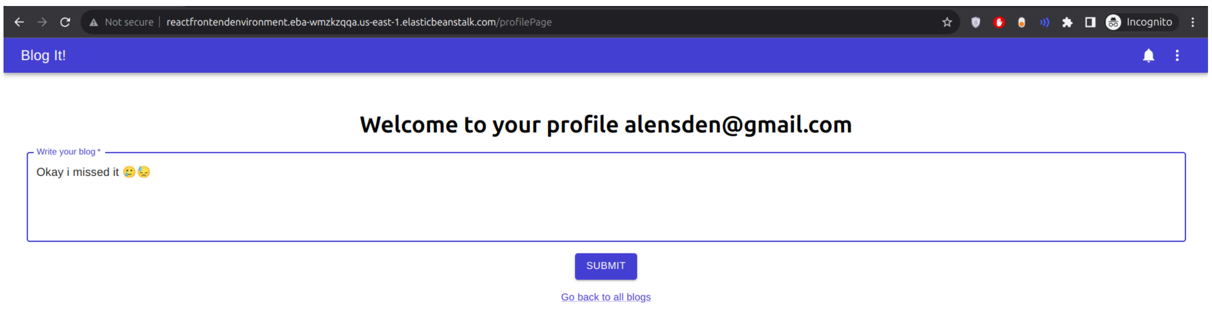

Not secure | reactfrontendenvironment.eba-wmzkzqqa.us-east-1.elasticbeanstalk.com/login

Blog It!

Login to Blog It

Figure 3: Login Page

2. **Blog Creation:** Logged-in users can create a new blog and post it so that others can see the blog. **(See Figure 4)**



Not secure | reactfrontendenvironment.eba-wmzkzqqa.us-east-1.elasticbeanstalk.com/profilePage

Blog It!

Welcome to your profile alensden@gmail.com

Write your blog *

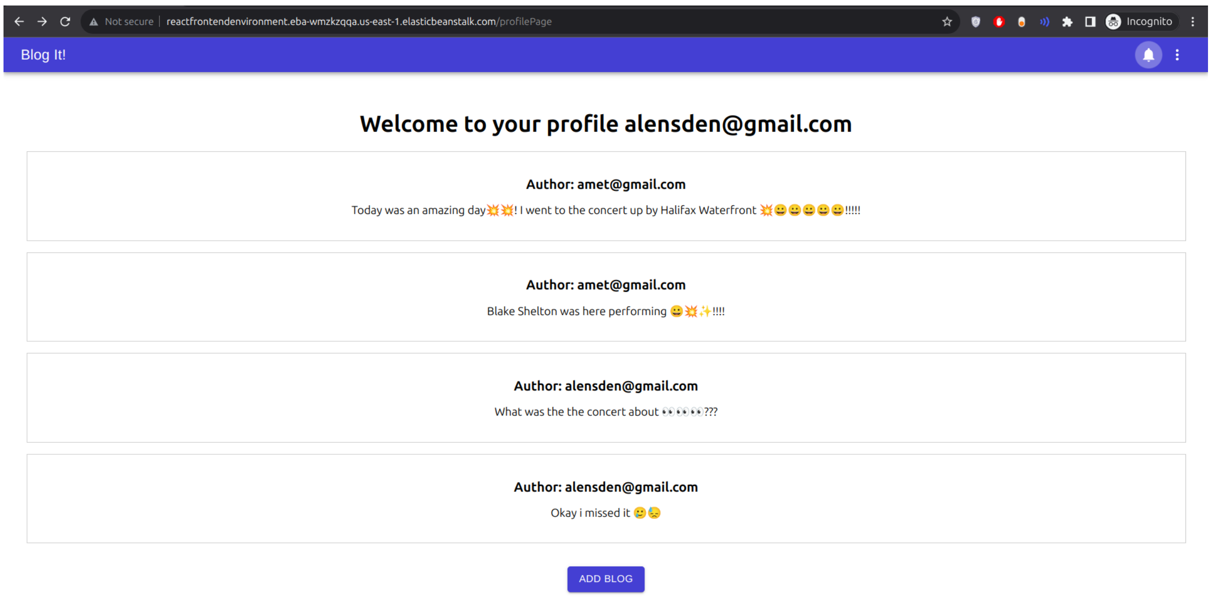
Okay i missed it 🥲😓

SUBMIT

Go back to all blogs

Figure 4: Create a new blog

3. **Blog Viewing**: Logged-in users are shown all the blogs posted by other users in the app. **(See Figure 5)**



Not secure | reactfrontendenvironment.eba-wmzkzqqa.us-east-1.elasticbeanstalk.com/profilePage

Blog It!

Welcome to your profile alensden@gmail.com

Author: amet@gmail.com

Today was an amazing day💥💥! I went to the concert up by Halifax Waterfront 💥😀😀😀😀😀!!!!!

Author: amet@gmail.com

Blake Shelton was here performing 😀💥✨!!!!

Author: alensden@gmail.com

What was the the concert about 👀👀👀???

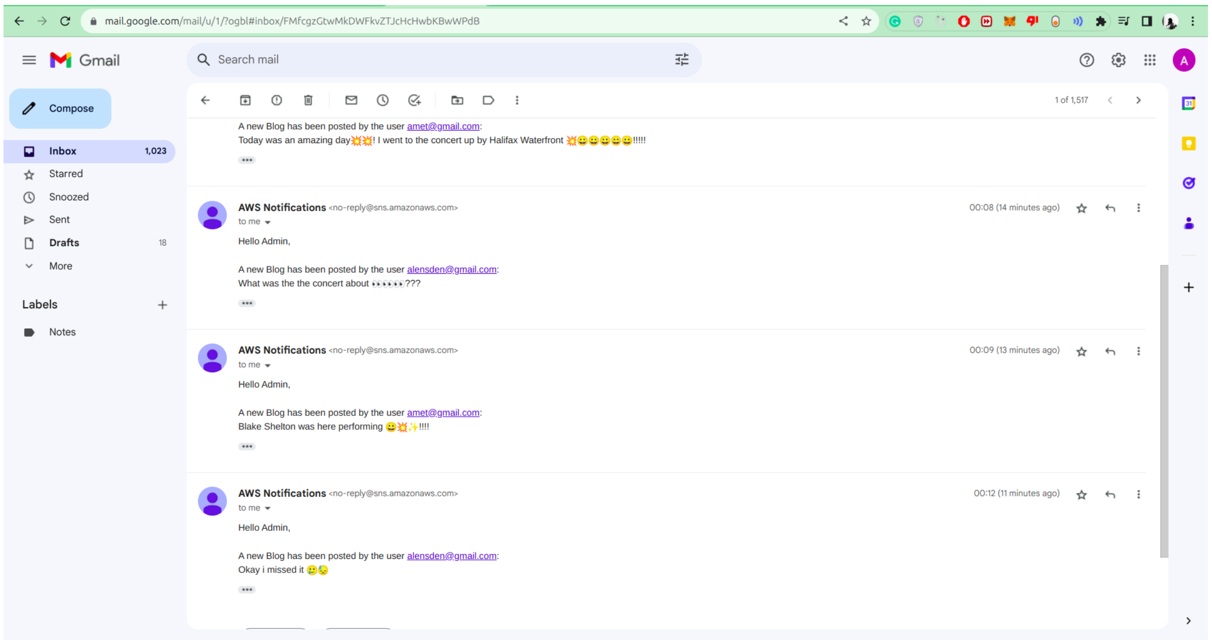
Author: alensden@gmail.com

Okay i missed it 🥲😓

ADD BLOG

Figure 5: Create a new blog

4. **Content Moderation:** With the increase of cyberbullying and hate speech, each of the blogs posted is reviewed by the administrator to ensure acceptable language and content on the platform. The admin gets an email notification whenever a user posts a new blog. **(See Figure 6)**



A new Blog has been posted by the user amet@gmail.com:
Today was an amazing day💥💥! I went to the concert up by Halifax Waterfront 💥😀😀😀😀😀!!!!!

AWS Notifications <no-reply@sns.amazonaws.com> 00:08 (14 minutes ago)
to me

Hello Admin,

A new Blog has been posted by the user alensden@gmail.com:
What was the the concert about 👀👀👀???

AWS Notifications <no-reply@sns.amazonaws.com> 00:09 (13 minutes ago)
to me

Hello Admin,

A new Blog has been posted by the user amet@gmail.com:
Blake Shelton was here performing 😀💥✨!!!!

AWS Notifications <no-reply@sns.amazonaws.com> 00:12 (11 minutes ago)
to me

Hello Admin,

A new Blog has been posted by the user alensden@gmail.com:
Okay i missed it 🥲😓

Figure 6: Email Notifications

5. **Performance:** The application uses a serverless architecture allowing it to scale automatically based on user demands and with no manual intervention.

6. **High Availability:** The application is deployed using AWS services due to this it has high availability and minimal downtown.

Target Users:

Blog It targets individual users, bloggers and content creators who wish to post their thoughts, stories and expertise on a user-friendly reliable platform. It caters to new bloggers and experienced bloggers providing a good platform for content creation.

Description:

Overall, Blog It is a user-friendly platform that enables users to focus on content creation while enjoying a seamless hassle-free experience.

Services Incorporated

For my web application, I have used all the required services as mentioned in the document. I have selected the services that best align with the requirements of my project. The following are the services that I selected for my application.

1. **Compute:**
    a. **Selected Service:** Elastic Beanstalk and AWS Lambda
    b. **Alternate Service:** EC2 and AWS Lambda

I choose AWS Elastic Beanstalk over AWS EC2 [7] as EBS [9] enables my application to automatically run, load balance and scale based upon demands without the need to manage the underlying resources. EBS has helped to abstract the complexity of individually setting up and configuring EC2 instances. This provided me with a streamlined approach to hosting my application. Using EBS helped me to focus on the application code while leaving the infrastructure management to AWS. This saves me time and effort.

2. **Storage:**
    a. **Selected Service:** AWS DynamoDB
    b. **Alternate Service:** AWS S3 Bucket

I choose AWS DynamoDB [2] over AWS S3 [5] because DynamoDB is more suitable for my application requirements. I can easily store and access user authentication information and blogs using DynamoDB. The key-value store and automatic scaling are well suited for my application requirements. Moreover, S3 is good object storage and can store large objects seamlessly hence I have used it to store my application artifacts.

3. **Network:**
    a. **Selected Service:** AWS API Gateway
    b. **Alternate Service:** AWS VPC

I chose AWS API Gateway as it provided me with a secure and scalable way to expose my Lambdas to external resources. It also helped me to effectively manage traffic and enable cross-origin-resource sharing. API Gateways [8] also enabled me to directly connect my AWS Lambdas to my API methods, this allowed me to easily integrate my serverless backend to my frontend. Using API Gateway let me abstract the complexity of setting up and configuring a virtual private network, by directly invoking my resources through REST API endpoints.

4. **General:**
    a. **Selected Service:** AWS SNS and AWS SQS
    b. **Alternate Service:** AWS Polly

I choose AWS SNS and SQS as I can easily streamline the process of the admin receiving emails regarding the blog content for content moderation. The SQS stores each blog object in a queue to be processed and the SNS processes the blog object and sends the required notification. AWS Polly is a powerful service to provide text-to-speech but my project requirements made me choose SNS and SQS.

My selection depends on the following:

1. **Scalability:** I have chosen AWS Services that can scale based on demand. This enables my application can handle varying workloads and can provide a smooth experience for its users
2. **Cost-effectiveness:** AWS Lambdas is used for the backend of my application and is cost-effective. I am only charged for the actual compute time used thereby minimizing costs and ensuring high performance.
3. **Ease of management:** By opting for managed services like EBS and AWS Lambdas, I have reduced the operational burden and can focus more on the application's core functionalities.

Deployment Model

I have chosen the public deployment model and have used AWS services like AWS Lambdas, SNS & SQS, DynamoDB and API Gateway. All of these services are public cloud offerings. My front end is deployed through Elastic Beanstalk. This is a Platform-as-a-Service and allows my application to run in isolation from other users.

The various services that I have used like AWS Lambdas for my backend, API Gateway for managing my APIs, AWS DynamoDB for my database and AWS SNS & SQS for pub/sub messaging are AWS Public cloud offerings. While Elastic Beanstalk allowed me to deploy my web application to a fully managed environment as a public cloud offering it provided each user with an isolated environment to work with. I choose this model as it is **flexible and scalable.** The combination of Lambdas, API Gateways, DynamoDB SNS and SQS allows for high scalability and a very flexible architecture. AWS's pay-as-you-go feature ensures cost-effectiveness and easy scalability as the application's demand fluctuates.

AWS Elastic Beanstalk provides **Easy Management and Deployment** for the front-end architecture without worrying about the underlying infrastructure. It simplifies the deployment process. By using AWS services it allows for **Integration with other services** and makes it easier to build a cohesive application. By using Elastic Beanstalk I have provided additional **Security and Isolation** to my application environment as EBS maintains isolated resources for each user.

To summarize the benefits discussed above let me choose a public deployment model over other models.

Delivery Model

For my application, I am utilizing a combination of several delivery models deployed in a public cloud environment.

1. **AWS Elastic Beanstalk**: To streamline the deployment and management of the front end of my application I used the Platform as a service (PaaS) delivery model that provided me with an environment that automatically handles infrastructure setup up, load balancing, scaling and health monitoring. By using this I could focus on developing the front-end code without worrying about the underlying infrastructure complexities.

2. **AWS Lambdas:** The backend of my application is developed through AWS Lambdas. AWS Lambdas follows the Function as a Service delivery model. This enabled me to develop individual functions catered to specific tasks for my application. The Lambdas automatically manage the infrastructure and scale the function based on demand.

3. **AWS API Gateway:** API Gateway is a Platform as a Service (PaaS). To expose my backend in a secure way for external resources I used the API Gateway. This service enabled me to make HTTP calls to my Lambdas. API Gateway supports various authentication methods and facilitates traffic management. This makes it an ideal choice for the applications API Layer.

4. **AWS SNS:** AWS SNS (Simple Notification Service) is a fully managed notification service and is a type of Platform as a Service (PaaS). It abstracts the complexity of sending notifications to multiple subscribers. Using this I could easily send email notifications to my admin.

5. **AWS SQS:** AWS SQS (Simple Queue Service) is a fully managed queuing service that is a type of Platform as a Service. It enables asynchronous communications between multiple microservices.

6. **DynamoDB:** DynamoDB is a fully managed NoSQL database. It is a type of PaaS called Database as a Service (DBaaS). It provides an environment to create and manage data.

Overall adopting a delivery model consisting of various cloud services hosted within the public cloud, I have achieved a scalable resilient and cost-effective infrastructure.

Final Architecture

How all the cloud mechanisms fit together:

My final architecture is a combination of various cloud mechanisms and services provided by AWS. Starting from my front end which is deployed in AWS Elastic Beanstalk. EBS automatically handles infrastructure setup, load balancing, scaling and health monitoring. By using this I could focus on developing the front-end code without worrying about the underlying infrastructure complexities. The backend is implemented using AWS Lambdas. This enabled me to develop individual functions catered to specific tasks for my application. The Lambdas automatically manage the infrastructure and scale the function based on demand. The Front end and back end are connected using API Gateway. This service enabled me to make HTTP calls to my Lambdas. API Gateway supports various authentication methods and facilitates traffic management. This makes it an ideal choice for the applications API Layer.

When a user creates a new blog an email notification is sent to the admin regarding the content of the blog. The admin can then review the content and ensure the content policy is being followed. The email notification is enabled through SNS. AWS SNS is a fully managed notification service and is a type of Platform as a Service (PaaS). It abstracts the complexity of sending notifications to multiple subscribers. Using this I could easily send email notifications to the admin to review the blog content. Since multiple blogs can be posted by a lot of users I also use a SQS queue to direct the blogs in a streamlined manner. AWS SQS is a fully managed queuing service that is a type of Platform as a Service. It enables asynchronous communications between multiple microservices. In this case, it is used to call the lambdas.

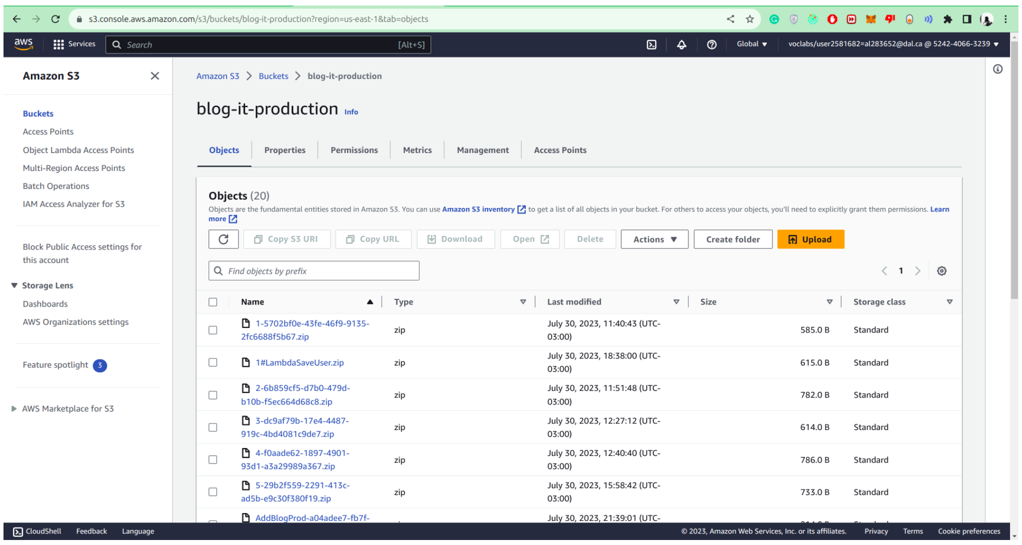
Where is the data stored:

The data in my program can be divided into two parts. One part is the user authentication data, which contains the email, name and password. This data is stored in the dynamo DB table *"Authentication_Table"*. The second part of the data is regarding the blogs the user posts. The data regarding this is stored in the table named *"Blogs"*. This data contains the email of the user, a unique *blog_id* and the blog content.

The other data is my project artifacts. This is comparatively a large amount of data and is stored in the S3 object storage provided by AWS. This is because I am referring to the resource on my cloud formation script using the S3 key. This way I can easily store my source code and work with it during project deployment.

What programming language I used:

My application mainly consists of javascript as my programming language used. For the front end of my application, I have used the Reacj.js library. This uses node js which is a javascript runtime (javascript outside a web browser). I also have used HTML and CSS within my .jsx file to define certain aspects of the UI. The backend of my code is AWS Lambdas and is written in Node.js as well. The front end and the back end code are zipped and uploaded on the S3 bucket. **(See Figure 7)**


Amazon S3 > Buckets > blog-it-production

blog-it-production

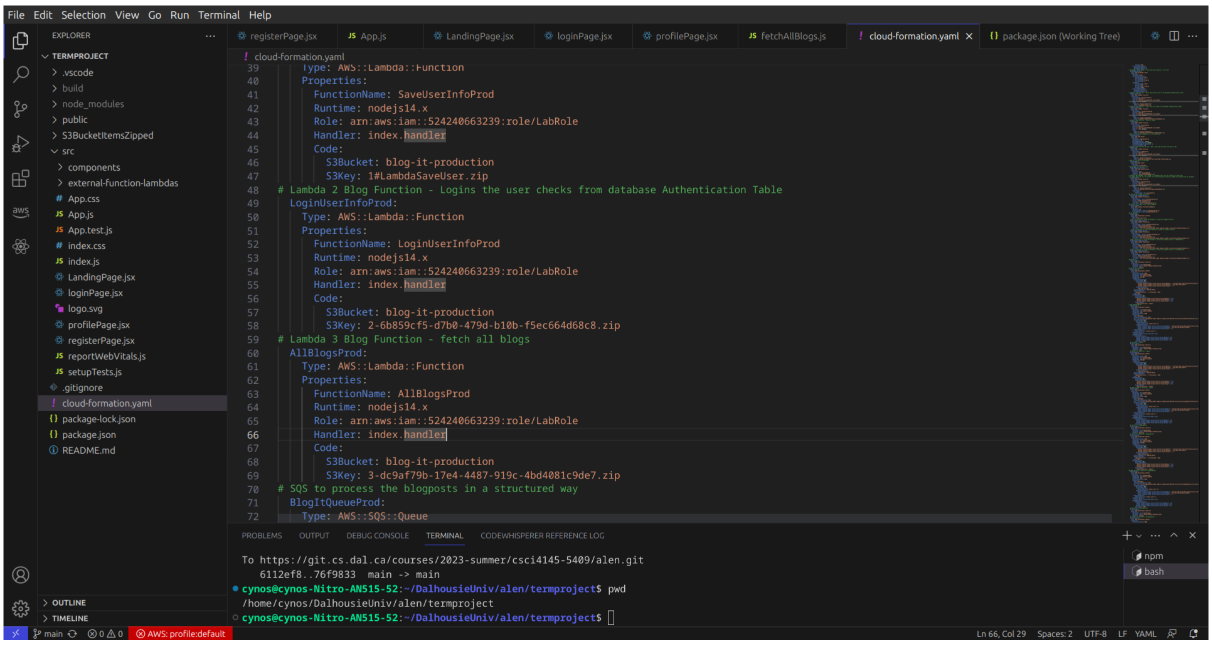
Objects (20)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
1-5702bf0e-43fe-46f9-9135-2fc6688f5b67.zip	zip	July 30, 2023, 11:40:43 (UTC-03:00)	585.0 B	Standard
1#LambdaSaveUser.zip	zip	July 30, 2023, 18:38:00 (UTC-03:00)	615.0 B	Standard
2-6b859cf5-d7b0-479d-b10b-f5ec664d68c8.zip	zip	July 30, 2023, 11:51:48 (UTC-03:00)	782.0 B	Standard
3-dc9af79b-17e4-4487-919c-4bd4081c9de7.zip	zip	July 30, 2023, 12:27:12 (UTC-03:00)	614.0 B	Standard
4-f0aade62-1897-4901-93d1-a3a29989a367.zip	zip	July 30, 2023, 12:40:40 (UTC-03:00)	786.0 B	Standard
5-29b2f559-2291-413c-ad5b-e9c30f380f19.zip	zip	July 30, 2023, 15:58:42 (UTC-03:00)	733.0 B	Standard
AddBlogProd-a04adee7-fb7f-		July 30, 2023, 21:39:01 (UTC-		

Figure 7: Email Notifications

Then I can write my cloud-build yaml script. This script contains all the resources I have used for my project. I can easily refer to the zipped resources using the S3 bucket key. When I execute my cloud build yaml file all of my resources are provisioned and all the lambdas and my frontend code are downloaded from the bucket and used in the deployed application.
(See Figure 8)


```yaml
      Type: AWS::Lambda::Function
      Properties:
        FunctionName: SaveUserInfoProd
        Runtime: nodejs14.x
        Role: arn:aws:iam::524240663239:role/LabRole
        Handler: index.handler
        Code:
          S3Bucket: blog-it-production
          S3Key: 1#LambdaSaveUser.zip
  # Lambda 2 Blog Function - Logins the user checks from database Authentication Table
    LoginUserInfoProd:
      Type: AWS::Lambda::Function
      Properties:
        FunctionName: LoginUserInfoProd
        Runtime: nodejs14.x
        Role: arn:aws:iam::524240663239:role/LabRole
        Handler: index.handler
        Code:
          S3Bucket: blog-it-production
          S3Key: 2-6b859cf5-d7b0-479d-b10b-f5ec664d68c8.zip
  # Lambda 3 Blog Function - fetch all blogs
    AllBlogsProd:
      Type: AWS::Lambda::Function
      Properties:
        FunctionName: AllBlogsProd
        Runtime: nodejs14.x
        Role: arn:aws:iam::524240663239:role/LabRole
        Handler: index.handler
        Code:
          S3Bucket: blog-it-production
          S3Key: 3-dc9af79b-17e4-4487-919c-4bd4081c9de7.zip
  # SQS to process the blogposts in a structured way
    BlogItQueueProd:
      Type: AWS::SQS::Queue
```

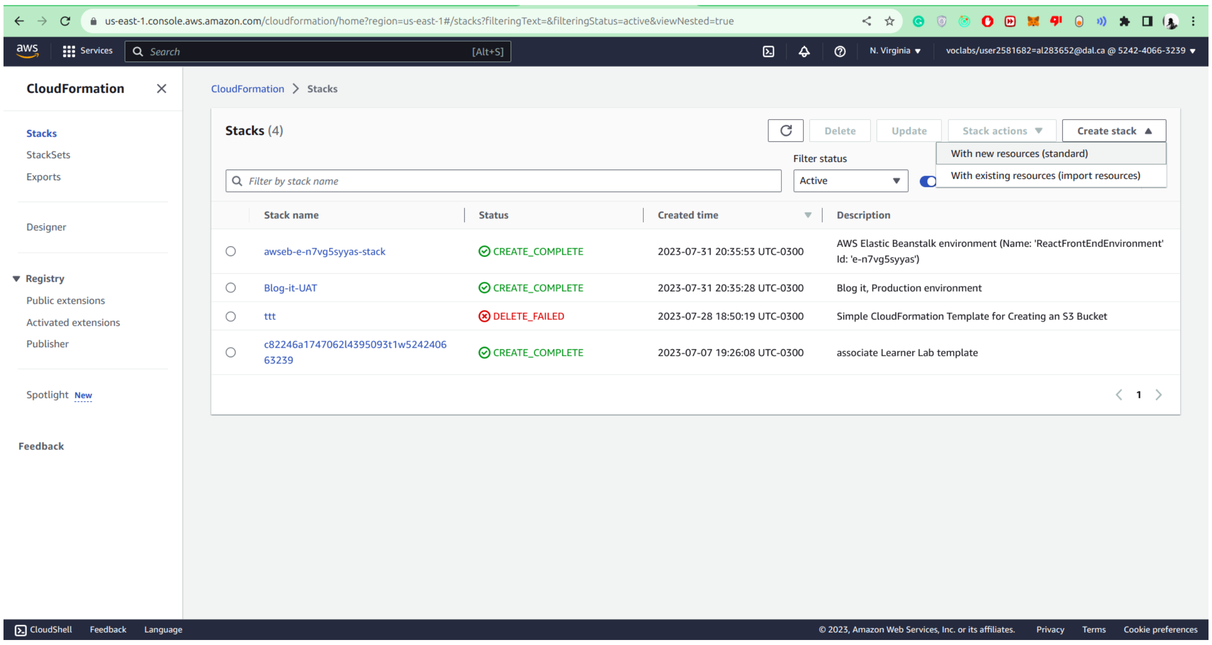
```
To https://git.cs.dal.ca/courses/2023-summer/csci4145-5409/alen.git
   6112ef8..76f9833  main -> main
cynos@cynos-Nitro-AN515-52:~/DalhousieUniv/alen/termproject$ pwd
/home/cynos/DalhousieUniv/alen/termproject
cynos@cynos-Nitro-AN515-52:~/DalhousieUniv/alen/termproject$
```

Figure 8: Fetching project artifacts from S3

The yaml file is where the infrastructure for my project is set up and executing this brings all of my resources alive.

How is my system deployed to the cloud:

My application is deployed using AWS cloud formation. I have written a YAML script with just over 500 lines of code that consists of all of my services. I have set the services up and connected them with each other in this script. Upon executing the script the provisioning of my system begins. All of my artifacts are downloaded from the S3 bucket and unzipped to be used for my infrastructure. AWS cloud formation just requires the YAML file to be uploaded to the console and it provisions everything else on its own. **(See Figure 9)**

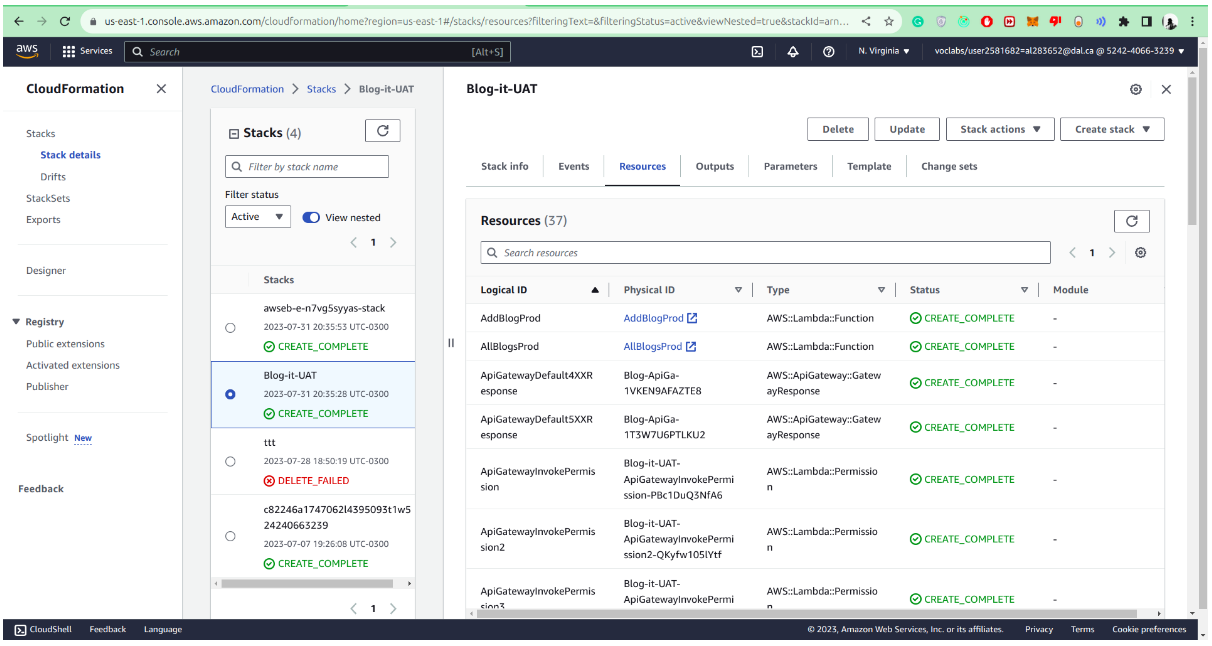


Stacks (4)

Stack name	Status	Created time	Description
awseb-e-n7vg5syyas-stack	✓ CREATE_COMPLETE	2023-07-31 20:35:53 UTC-0300	AWS Elastic Beanstalk environment (Name: 'ReactFrontEndEnvironment' Id: 'e-n7vg5syyas')
Blog-it-UAT	✓ CREATE_COMPLETE	2023-07-31 20:35:28 UTC-0300	Blog it, Production environment
ttt	✗ DELETE_FAILED	2023-07-28 18:50:19 UTC-0300	Simple CloudFormation Template for Creating an S3 Bucket
c82246a1747062l4395093t1w5242406 63239	✓ CREATE_COMPLETE	2023-07-07 19:26:08 UTC-0300	associate Learner Lab template

Figure 9: Cloud Formation Stacks

I start with creating my DynamoDB collections and end with downloading my frontend zip from the S3 bucket and deploying it on AWS Elastic Beanstalk. AWS shows me all of the created resources under the resource tab. **(See Figure 10)**



CloudFormation > Stacks > Blog-it-UAT

Stacks (4)

Filter status: Active — View nested

Stacks
awseb-e-n7vg5syyas-stack 2023-07-31 20:35:53 UTC-0300 CREATE_COMPLETE
Blog-it-UAT 2023-07-31 20:35:28 UTC-0300 CREATE_COMPLETE
ttt 2023-07-28 18:50:19 UTC-0300 DELETE_FAILED
c82246a1747062l4395093t1w524240663239 2023-07-07 19:26:08 UTC-0300 CREATE_COMPLETE

Blog-it-UAT

Delete | Update | Stack actions | Create stack

Stack info | Events | Resources | Outputs | Parameters | Template | Change sets

Resources (37)

Logical ID	Physical ID	Type	Status	Module
AddBlogProd	AddBlogProd	AWS::Lambda::Function	CREATE_COMPLETE	-
AllBlogsProd	AllBlogsProd	AWS::Lambda::Function	CREATE_COMPLETE	-
ApiGatewayDefault4XXResponse	Blog-ApiGa-1VKEN9AFAZTE8	AWS::ApiGateway::GatewayResponse	CREATE_COMPLETE	-
ApiGatewayDefault5XXResponse	Blog-ApiGa-1T3W7U6PTLKU2	AWS::ApiGateway::GatewayResponse	CREATE_COMPLETE	-
ApiGatewayInvokePermission	Blog-it-UAT-ApiGatewayInvokePermission-PBc1DuQ3NfA6	AWS::Lambda::Permission	CREATE_COMPLETE	-
ApiGatewayInvokePermission2	Blog-it-UAT-ApiGatewayInvokePermission2-QKyfw105lYtf	AWS::Lambda::Permission	CREATE_COMPLETE	-
ApiGatewayInvokePermission3	Blog-it-UAT-ApiGatewayInvokePermi	AWS::Lambda::Permission	CREATE_COMPLETE	-

© 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Figure 10: Cloud Formation Stack "Blog-it-UAT"

What architecture am I using:

My application uses an event-driven serverless architecture [10]. It used AWS Lambdas for the backend, API Gateways to integrate it to my front end, AWS SNS and SQS to show email notifications to the admin, and dynamo DB to store all the data. The architecture is event-driven because it responds to certain events such as user creation, Blog creation or user login. When a user requests for a specific service a lambda is triggered using the API Gateway. All of the operations are event-driven for my application.

1. When a user signs up for an account. The saved user information lambda is triggered and this saves the user information that is received from the API gateway.
2. The user details are now stored in the authentication table. When a user tries to log in their information is checked again through a different lambda that is invoked.
3. When a user creates a new blog, the add new blog lambda is triggered. This lambda stores the blog in the dynamo DB and forwards the blog object to the SQS queue. This queue processes the object and then triggers a lambda function. This lambda function checks if a blog object is present in the queue.
4. If an object is present it processes it and sends it to the SNS topic. The SNS topic then published the message as an email to the subscriber
5. In this case to the admin on their email for the content of the blog to be reviewed.

The AWS auto-generated architecture diagram is as follows **(See Figure 11)**

Architecture Diagram:

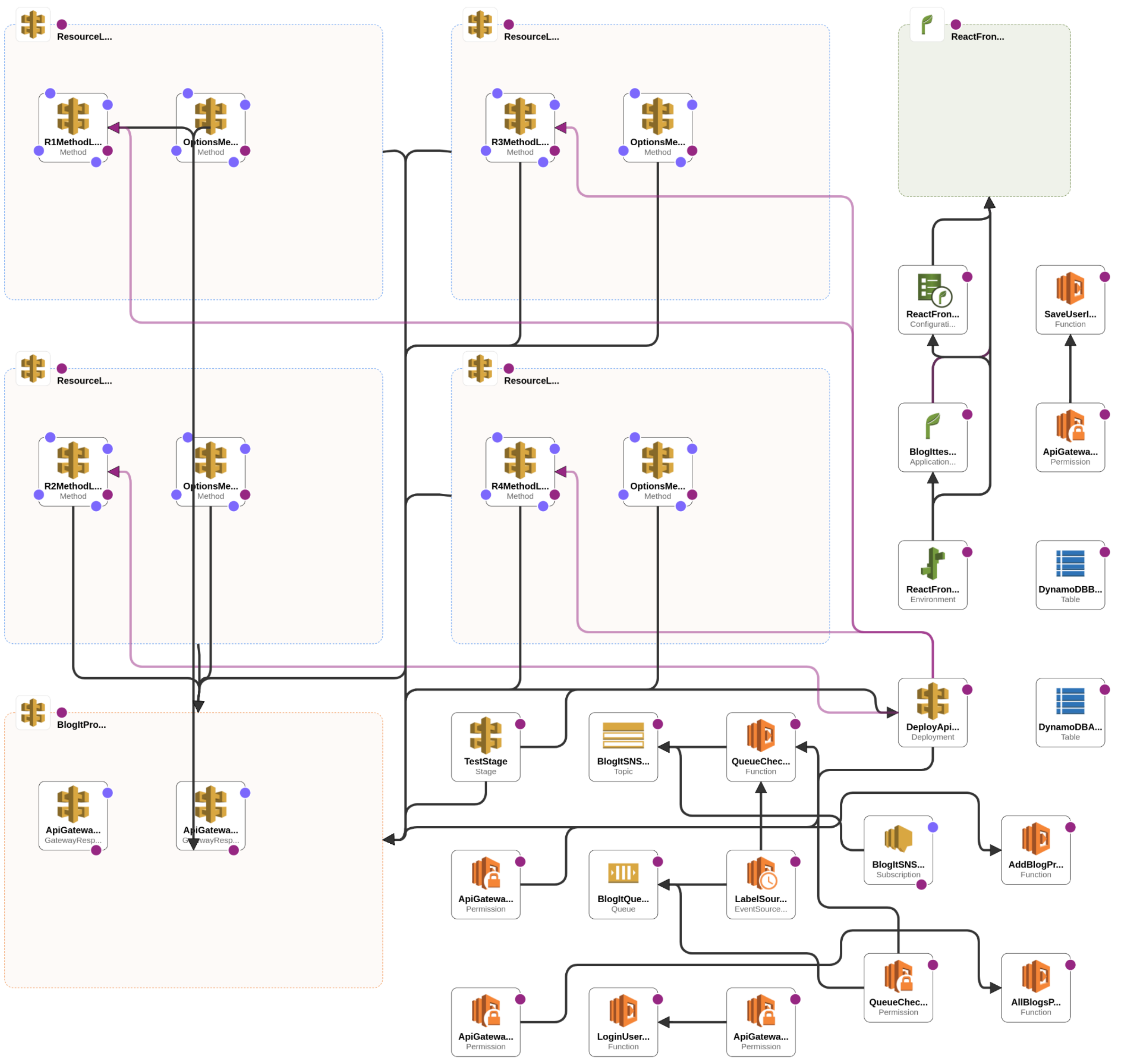


Figure 11: Blog it Architecture

Application Security

My application uses simple authentication that requires an email and a password. I also have not hard-coded my API endpoints on my front-end application as they can be easily accessed through the developer console and be exploited. My backend is also built using AWS Lambdas which further enhances security. I have tried to use environment variables and avoid hard coding to avoid security issues. All of my infrastructure is securely stored in an AWS S3 bucket that has blocked all access from public IP addresses.

I can improve my application security by incorporating the following -

1. **AWS IAM Roles:** I can improve security by assigning different levels of access to different Lambdas following the principle of least privilege. This will ensure that each lambda has its own permissions and it can access only the resources it is supposed to access.
2. **DynamoDB:** I can further improve security by hashing the stored passwords. I can use libraries like bcrypt to only store hashed passwords. Considering the learning objectives I have not implemented hashing but in a real-world scenario, the password is securely hashed before it is stored in the database.
3. **AWS Cognito:** To further enhance security I can also use AWS Cognito to manage the user authentication. This ensures that the passwords are hashed before storing and also securely stores the user information. This service can easily be integrated with my current architecture and improve the security of my application. It also provides multi-factor authentication for an added security benefit.

Cost to reproduce in a private cloud

Reproducing my architecture in an on-premise setting will require a significant investment in hardware and software. To provide a relatively same level of availability as in AWS public cloud, I will have to ensure redundancy, fault tolerance and scalability in my private cloud setup. On-premise solutions typically have higher upfront costs due to high acquisition costs for IT resources and employees.

1. **Servers:** I would need a bunch of servers to host my application components including the front end, backend and database servers. To ensure high availability, fault tolerance and load balancing I will also need to configure a load balancer. The average cost of a server is $5000 eg the Lenovo think system starts at $4408. For a moderate setup, I will need 10 servers, which would cost $50,000.

2. **Storage:** To host application data like AWS DynamoDb I will need to invest in a Network-Attached Storage (NAS) or Storage Area Network (SAN) for data storage and backup. The WD NAS which is a 54 TB storage option starts at $3704. I might initially need just two and this can go up as demand increases. This will cost around $10,000.

3. **Network Equipment:** I would require Network equipment like routers, switches and firewalls. A good quality switch can cost $10,000. A Cisco router and firewall can cost $6,628 per piece of equipment. This brings the total to $22,000 for a single network configuration. To have high availability I would at least need 2 networks set up bringing the cost to around $44,000.

4. **Software:** I would also need to buy various software to ensure everything works. I would need virtualization software to virtualize my servers and a single license for one can go up to $5,000. I will also need database software licenses to host my databases which can cost up to $5,000 per license. This brings my total cost to about $20,000 for 10 servers.

5. **Power and Cooling:** I will have to also ensure my on-premise solution is equipped with a reliable power source and cooling systems to ensure high availability and uptime. This might be in the range of $20,000 for the entire cooling setup. This will include the industrial server room air conditioning systems and rack cooling for the servers.

6. **Security:** I will also have to invest in robust security solutions like intrusion detection/ prevention systems, access controls and security cameras. All of this can cost up to $ 15,000.

7. **Employees:** I will also need an IT support analyst, Cloud developers and associates to manage and maintain all the resources. This will cost me an average of $75,000 per year per employee. I would at least need 4 employees to maintain my resources in the initial days. This will bring the cost to $300,000.

The total cost to maintain a private cloud solution like AWS will go up to $459,000. Remember this is just the upfront cost and this price will be increasing as the demand increases. Each year I will have to spend money on updates, minor fixes and new software. A majority of my expenses will be on my employees. More than half of my monetary resources are spent on my employees every year! Even after all of this, it may be challenging to manage and maintain the on-premise solution.

Cost of cloud resources

1. **AWS Lambdas:** The cost of the Lambdas depends on the number of invocations and the total execution time. Since I have several Lambdas in my architecture the costs can increase based on the frequency of the functions being invoked. The more the amount of users interacting with my application, the more the costs. Hence appropriate monitoring is required for this service.

2. **AWS API Gateway:** The API Gateway cost is determined by the number of HTTP calls made to the API. With the increase of users using my application, I would get more costs for this resource. Hence monitoring is required for this service as well. Monitoring can also help detect suspicious HTTP API calls.

The other resources like DynamoDB and SQS SNS have a more predictable cost associated with them; their costs are generally related to the amount of storage used and the number of read/write operations performed. I can further optimize the costs by adding cloud watch monitoring by cloud watch. Cloud watch provides the necessary insights required for cost monitoring.

Upcoming Features for Blog it

1. **User Authentication and Authorization:** I can use AWS Cognito in the future for authentication and authorization. This will improve my application's overall security.

2. **User Profile and Settings:** Users can store user-specific details like their likes and interests. Users will also have their own profile page that can be viewed by others.

3. **Image and File Uploads:** Blog It will have support to upload images and files to the platform to make it more appealing to the users. For this, I can use the S3 object storage to store user images and files.

4. **Personalized Recommendations:** I can utilize AWS Personalise to recommend content to users based on their reading history and interest.

References

[1] Amazon Web Services, "AWS Lambda Documentation," [Online]. Available: https://docs.aws.amazon.com/lambda/index.html. [Accessed: July 4, 2023].

[2] Amazon Web Services, "Amazon DynamoDB Documentation," [Online]. Available: https://docs.aws.amazon.com/dynamodb/index.html. [Accessed: July 4, 2023].

[3] Amazon Web Services, "Amazon SNS Documentation," [Online]. Available: https://docs.aws.amazon.com/sns/index.html. [Accessed: July 4, 2023].

[4] Amazon Web Services, "Amazon SQS Documentation," [Online]. Available: https://docs.aws.amazon.com/sqs/index.html. [Accessed: July 4, 2023].

[5] Amazon Web Services, "Amazon S3 Documentation," [Online]. Available: https://docs.aws.amazon.com/s3/index.html. [Accessed: July 4, 2023].

[6] React, "React Documentation," [Online]. Available: https://legacy.reactjs.org/docs/getting-started.html. [Accessed: July 4, 2023].

[7] Amazon Web Services, "Amazon EC2 Documentation," [Online]. Available: https://docs.aws.amazon.com/ec2/index.html. [Accessed: July 4, 2023].

[8] Amazon Web Services, "Amazon API Gateway Documentation," [Online]. Available: https://docs.aws.amazon.com/apigateway/index.html. [Accessed: July 4, 2023].

[9] Amazon Web Services, "Amazon EBS Documentation," [Online]. Available: https://docs.aws.amazon.com/ebs/index.html. [Accessed: July 4, 2023].

[10] Confluent, "Event-Driven Microservices," [Online]. Available: https://www.confluent.io/resources/event-driven-microservices/?utm_medium=sem&utm_source=google&utm_campaign=ch.sem_br.nonbrand_tp.prs_tgt.technical-research_mt.mbm_rgn.namer_lng.eng_dv.all_con.event-driven-architecture&utm_term=%2Bevent%20%2Bdriven%20%2Barchitecture&creative=&device=c&placement=&gad=1&gclid=Cj0KCQjw2qKmBhCfARIsAFy8buJYb3EeOLyYoQLtvf_6MhGOtxeCfD-uQqXMLxvKuma7TA00-yteoxMaAvXPEALw_wcB. [Accessed: July 4, 2023].