



**DALHOUSIE**  
**UNIVERSITY**

# **Master of Applied Computer Science**

MACS

**CSCI 5409**

**CLOUD COMPUTING**

**CLOUD PROJECT REPORT**

Utsav Singh

Banner ID : B00923487

Email ID : [ut796069@dal.ca](mailto:ut796069@dal.ca)

# 1. Menu Item Requirements

## 1.1. Services Selected

Categories	Services Selected for my application
Compute	AWS EC2, AWS Elastic Beanstalk, AWS Lambda
Storage	AWS S3, AWS DynamoDB
Network	Amazon CloudFront
General	AWS SNS, AWS Secrets Manager

## 1.2 Reasons to Select over other alternatives

### 1.2.1 Compute

#### AWS EC2

Elastic cloud compute comes under infrastructure as a service (IAAS) where AWS provides all the infrastructure required for our computational needs.

#### Reasons to Choose EC2 over other alternatives

Scalability: Given that my AutoHub service includes the ability to upload pictures and that the number of users may increase at any time, I chose EC2 because it dynamically scales up and down according to demand.

Flexibility: The virtual server can be configured using one of the several instance types offered by EC2, such as t2.micro, t2.large, etc., depending on our performance needs and financial constraints. For my AutoHub application as the memory requirement is more, t2.large has 4 times the memory that t2.micro provides and also twice the computing abilities.

Reliability: The built-in redundancy and failover features of an EC2 instance would benefit my application by making it highly available.

Security: Several options are provided by EC2 to protect our application from unauthorized access. To control who can send requests to my AutoHub application, I use security groups for inbound traffic filtering.

Cost: Because EC2 is a pay-as-you-go approach, meaning we only pay for what we use, it would be the most cost-effective solution for AutoHub given the fluctuating traffic to the application.

## **AWS Elastic Beanstalk**

Elastic Beanstalk (EBS) offers three different Node.js version platforms, including Node.js 18, Node.js 16, and Node.js 14. All of them run 64-bit Amazon Linux 2. EBS makes it possible to deploy node applications without having to install reverse proxy programs like Nginx. EBS also provides automatic load balancing through the creation of new instances to guarantee efficient request processing.

### **Reasons to Choose Elastic Beanstalk over other alternatives**

Because the Node JS portion of my AutoHub backend interacts with AWS services, it needs access to AWS. When compared to AWS EC2, EBS makes it easier to manage AWS credentials because it doesn't require the creation of a .env file or the configuration of AWS credentials. The AWS credentials are provided in EBS at the time of creation, and it then controls all of the credentials. There is no requirement to perform configure scripts or make a .env file, as we do in EC2. Also, EBS does load balancing by scaling out and scaling in resources as per the need.

Easier deployment: My Node.js backend application could be deployed more quickly and easily with Elastic Beanstalk due to its simplified deployment procedure.

Architecture Flexibility: Several Node.js web frameworks are supported by Elastic Beanstalk. It benefits my application because Express JS support was required by it.

Monitoring: Elastic Beanstalk provides a variety of monitoring tools. For my application logs are useful as it helps to troubleshoot the issue at a much faster rate.

Security: Several options are provided by EC2 to protect our application from unauthorized access. To control who can send requests to my AutoHub application, I have used a security group for inbound traffic filtering.

Cost: Because EC2 is a pay-as-you-go approach, meaning we only pay for what we use, it would be the most cost-effective solution for AutoHub given the fluctuating traffic to the application.

## **AWS Lambda**

For my AutoHub application, I have the Subscription feature for the user. Whenever a user subscribes to receive notification for every new entry in the cars table a new image is generated under exports and streams which has a trigger that calls a lambda function. The lambda function gets triggered on every new image creation and publishes an email to all the subscriptions within the mentioned SNS topic.

### **Reasons to Choose AWS Lambda over Alternatives**

Cost Saving: The biggest advantage of using Lambda is that we get charged only for the computing time our code uses whereas in the case of EC2 and EBS we are charged for the entire server even if we are not using it fully. Lambda is useful for small workloads like in my case to publish notifications when a change is triggered inside the car table.

Real-time notifications: I can notify subscribers in real-time whenever a new car record is added to the database by utilizing AWS Lambda and SNS. This can improve the user experience and keep users interested in your program.

Scalability: Without having to worry about manually managing infrastructure or scaling resources, AWS Lambda enables my application to handle a high volume of subscribers and notifications.

Customizability: I have complete control over the logic handling my notification behavior because of AWS Lambda, which makes it easier to develop my distinct script. The distribution of notifications can also be tailored using SNS, for example, by delivering them to several endpoints. Currently, AutoHub delivers notifications via Email considering the future scope we can also include SNS notifications.

Availability: Compared to EC2 and EBS which requires manual setup for high availability and tolerance, Lambda automatically provides high availability and tolerance by replicating our code automatically to multiple availability zones. If one availability zone fails, the code runs on another availability zone.

Step functions under function as a service could have been an alternative, but they are best used for a series of tasks or when a decision needs to be made based on user input, which is not the case with my feature, which only needs to publish notifications when new data is added to the car table.

### **1.2.2 Storage**

#### **AWS S3**

In my application AutoHub, I am using AWS S3 to store car images that are being uploaded by the user who wants to sell a car and list it on our portal. I am storing the image inside the AWS S3 bucket.

#### **Reasons to Choose AWS S3 over other alternatives**

AWS S3 is the most cost-effective solution for storing a wide range of unstructured data like images, videos, logs, etc,

In my application durability matters as for a user to view a car images play an important role to know the condition of the car and visual representation. With an object durability of 99.999999999% (11 9s), S3 has exceptional durability. To guarantee data availability in the event of a disaster, it also offers data replication across different availability zones.

Other alternatives like AWS DynamoDB, AWS Neptune, AWS Aurora, and AWS Athena are mostly suited for handling structured data due to their powerful query capabilities.

#### **AWS DynamoDB**

In AutoHub, I have 2 tables one for the user details and one for car details.

In the registration feature, the user enters his details which are stored inside the user details table. Username is considered as the partition key.

In the feature of selling a car the user enters all the details about the car and uploads the Images the image is stored in S3 and the AWS CloudFront link pointing to the image is also stored in the DynamoDB table.

### Reasons to choose AWS DynamoDB over other alternatives

The advantage of DynamoDB over alternatives like AWS SimpleDB and AWS DocumentDB is that it is highly scalable and optimized for low latency performance compared to the alternatives.

### 1.2.3 Network

#### Amazon CloudFront

The content delivery network (CDN) service Amazon CloudFront is offered by Amazon Web Services. (AWS). It is intended to provide consumers all around the world with minimal latency and fast transfer speeds while delivering content, such as web pages, movies, applications, and APIs.

In CloudFront, content is cached at edge locations, which are AWS data centers spread across the globe. When a user requests content, CloudFront directs the request to the closest edge location that already has the requested content cached, reducing the amount of time the requested content must travel and speeding up response times.

### Reasons to Choose Amazon CloudFront over other alternatives

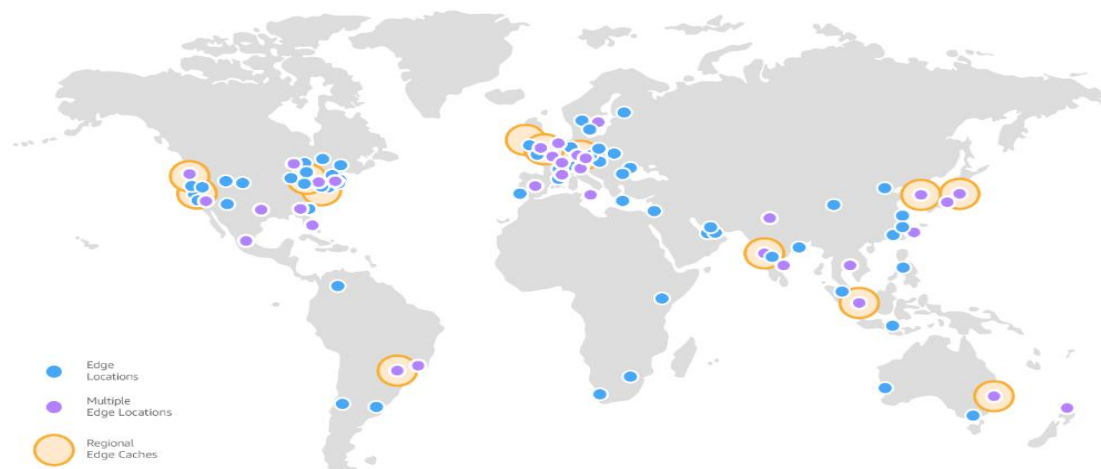


Figure 1: CloudFront Edge Locations Graph

## Why I use Amazon CloudFront

I am using Amazon CloudFront to retrieve images from S3. It helps to retrieve images at a faster rate by referring to the nearest edge location and delivering the result.

### Reasons to Choose AWS CloudFront over other alternatives

Easy integration ability with a variety of other AWS services. In my case I am integrating CloudFront with my AWS S3 bucket to replicate it to multiple edge locations and retrieve it from the nearest edge location thus reducing the latency.

To improve delivery tactics, CloudFront offers real-time data and monitoring that would help me better understand the performance and utilization of content.

Other alternatives like Route 53 and Elastic load balancing do not have the same caching ability as CloudFront. The number of edge locations is comparatively higher in AWS CloudFront compared to others.

AWS CloudFront also provides encryption of data while transferring.

Performance testing in CDN for retrieving a car image store in AWS S3bucket

URL

Test











LOCATION	STATUS	DNS	CONNECT	TLS	TTFB	
 Frankfurt	301	11.07 ms	0.65 ms	--	1.53 ms	▼
 Amsterdam	301	29.64 ms	13.39 ms	--	25.46 ms	▼
 London	301	17.42 ms	2.84 ms	--	4.36 ms	▼
 New York	301	23.79 ms	2.32 ms	--	4.15 ms	▼
 Dallas	301	53.22 ms	6.62 ms	--	13.19 ms	▼
 San Francisco	301	25.93 ms	8.89 ms	--	16.83 ms	▼
 Singapore	301	15.49 ms	4.27 ms	--	5.96 ms	▼
 Sydney	301	12.72 ms	1.67 ms	--	2.81 ms	▼
 Tokyo	301	11.87 ms	1.54 ms	--	9.33 ms	▼
 Bangalore	301	41.65 ms	3.51 ms	--	5.76 ms	▼

Figure 2: Performance Testing in CDN

#### **1.2.4 General**

##### **AWS SNS**

###### **Reasons to Choose AWS SNS over other alternatives**

For my application AutoHub, I choose AWS SNS for my subscription feature for which if a user subscribes to receive notifications then his email is added to the subscription list from the Node JS backend, and then SNS publishes the email to all the subscribers.

SNS was the most suitable in my case as in my case email needed to be broadcasted on every new entry to all the subscribers.

Also considering the future scope of the application numerous communications protocols are supported by SNS, including HTTP, HTTPS, SMS, email, and push notifications for mobile devices. As a result, integrating with various endpoints and clients would be easy.

Easy to integrate with other services like AWS lambda. In the AutoHub application whenever there is a new car entry lambda triggers the desired SNS to broadcast emails to all the subscribers.

Compared to other AWS messaging services like Amazon MQ (managed message broker) and Amazon SQS (Simple Queue Service), which can handle a variety of different endpoints, SNS offers a more flexible and robust messaging solution.

##### **Amazon Secrets Manager**

Why I choose Amazon Secrets Manager

AutoHub makes use of Dynamo table, S3 Bucket, and CloudFront so to secure access to these resources I have used Amazon Secrets Manager which stores the credentials for the following resources.

###### **Reasons to choose Amazon Secrets Manager**

Sensitive data is protected from unauthorized access and breaches with the help of AWS Secret Manager, which offers highly secure and encrypted storage for secrets.

It provides a way to rotate the secret after a certain time frame which enhances the security. It can integrate with other services so that secret retrieval becomes easy and safe.

It provides a way to rotate credentials manually as well as automatically which helps the secret from being compromised and getting outdated.

You can keep track of who is accessing your secrets and when with the help of the comprehensive logging and monitoring options provided by AWS Secret Manager.

## 2. Deployment Model

For my AutoHub application Public Cloud would be the best-suited deployment model as my application would cater large number of users and to address this issue public cloud would be the best as it would easily scale up or scale down depending on the requirement. Another advantage of public cloud computing is its global reach. Public cloud service providers are widespread, with data centers and services available in numerous nations. Reaching out to new users becomes simple as a result. The public cloud model would eliminate the overhead of managing servers and maintaining them. Also if the server goes down it would automatically switch to another which would increase the availability of software to the users. The public cloud offers faster time-to-market. The public cloud provides faster deployment which helps businesses to bring new applications and services to market more quickly. Overall, it provides a highly scalable, cost-effective way to deploy and manage their infrastructure.

## 3. Delivery Model

### 3.1 Describe your delivery model. Why did you choose this delivery model?

For our application AutoHub, **Software as a Service** would be the best-suited delivery model as users/customers would not require to install software locally or on a server. They can access through a web browser using the internet. AutoHub is managed by and maintained by AWS in my case. By delivering applications as software as a service users won't face scalability, or accessibility issues.

With SAAS, the cost of purchasing hardware, software licenses, and maintenance is eliminated. SAAS comes with inbuilt security features which make sure that data are protected from unauthorized access and data breaches.



## 4. Final architecture

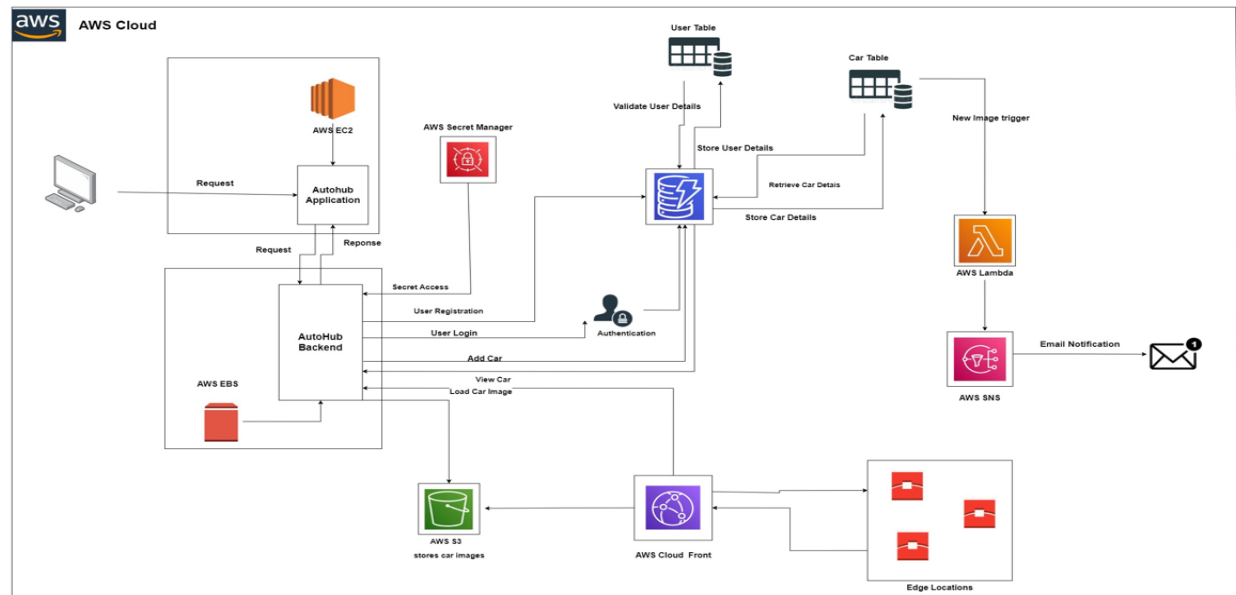


Figure 3. Application Flow

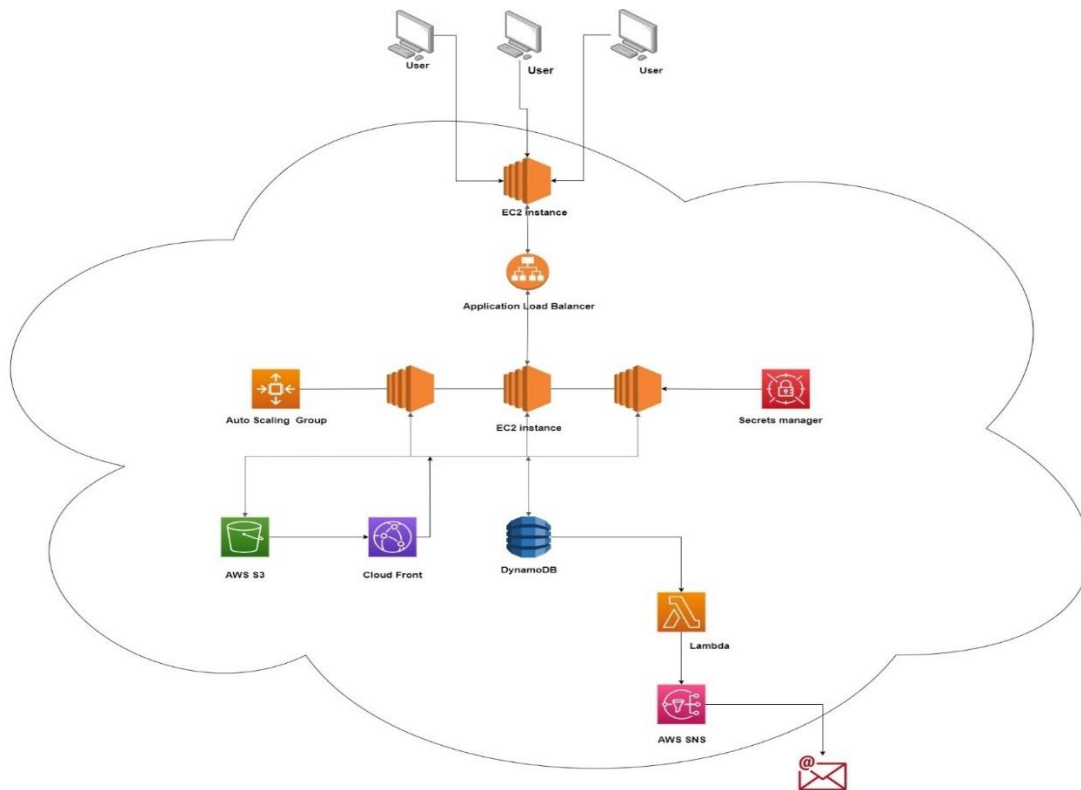


Figure 4. Service Load Balancing Architecture

## 4.1 How do all of the cloud mechanisms fit together to deliver your application?

### Service load balancing architecture

Referring to Figure[4], whenever the user sends requests to the frontend hosted on EC2 the requests are then forwarded to the backend using load balancer. The only condition in this architecture is that all the instances should replicate themselves i.e. all nodes should be working on the same data on the same data. In our backend which is hosted on EBS autoscaling has been configured so that all EC2 instances replicate themselves and work on the same data.

Figure[3] above represents the entire application flow of my cloud-based application “AutoHub”

“AutoHub” comprises of client-server architecture. The client side of the application is based on ReactJS. It is deployed on EC2 sever with instance type “t2.large”. The reason for selecting t2.large is that my application requires more memory as it has the file upload feature and also I have images used for visual representation in the application. “t2.micro” took more time to load the images and so decided to take advantage of scalability. The security group associated with the EC2 instance gives access to ports 22 and 3000 respectively. The EC2 instance communicates with the backend (Server side) of the application that comprises NodeJS.

The Server side consists of the NodeJS application deployed on AWS Elastic Beanstalk. The application is configured to listen to requests on port 5000. The EBS is configured with all the required AWS and environmental parameters in the cloud formation template during the time of creation. The server listens to requests and forwards them to the respective services. It has access to a secret manager that stores all the access credentials to access the DynamoDB table, S3 bucket, and CloudFront URL.

When a /register POST request is received by the server it stores all the information in the user table and also if the user decides to subscribe to receive notifications it also sends a /subscribe post request to which the server connects to the SNS service and adds the user email to the subscription list of the respective topic and a confirmation link is sent to the user.

CloudFront makes use of the “Redundant Storage” where it replicates all the images stored inside AWS S3 to multiple edge locations. This helps in reducing the latency while loading up an image

On receiving a /login POST request the server connects to the user DynamoDB table by getting access from the Secrets manager and checks whether the user is a valid user or not.

On receiving a /addcar POST request the server connects to the secrets manager and stores the images to S3 and other car details to the DynamoDB. It also makes use of the CloudFront URL that is linked to the S3 bucket and stores the image link to the same DynamoDB table. As mentioned above, if the user has subscribed to receive notifications then for every new insertion in the cars table a new image is created and a lambda function is triggered that sends a broadcast email with a customized email body to all the subscribers.

When the server receives a view car POST request from the then the server using the Secrets manager connects to the DynamoDB and retrieves all the information from the cars table. As mentioned for retrieving the image the server does not directly connect to the AWS S3 instead it uses CloudFront URL to retrieve the images by selecting the edge location that is nearest among all the edge locations on which the CloudFront has replicated the images.

AWS Simple Notification Service is triggered by the Lambda function to send emails to all subscribers via SMTP protocol.

#### **4.2 Where is data stored?**

In AutoHub data is stored in 2 tables in DynamoDB

Users table – consisting of user details entered during registration.

Cars table – consist of details added during

Images are stored inside AWS Simple Storage (S3) bucket that is being retrieved using CloudFront.

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

A client-side application and a server-side application make up my architecture. For my front end I am using ReactJS which is a JavaScript-based framework.

For the backend part of the application, I am using NodeJS and ExpressJS. All the business logic is written using these languages only and the major reason for using NodeJS is that it makes request handling easy due to the use of routers in ExpressJS.

Connecting the frontend and backend parts is done using the axios library that is used to make Rest API calls to the backend hosted using AWS Elastic Beanstalk and running on an EC2 instance.

#### **4.4 How is your system deployed to the cloud?**

I have made use of AWS Cloud Formation (Infrastructure as a service) to deploy the application to the AWS cloud. AWS cloud

Figure [5] represents the design of my Cloud Formation Design used to deploy the infrastructure and services to the cloud.

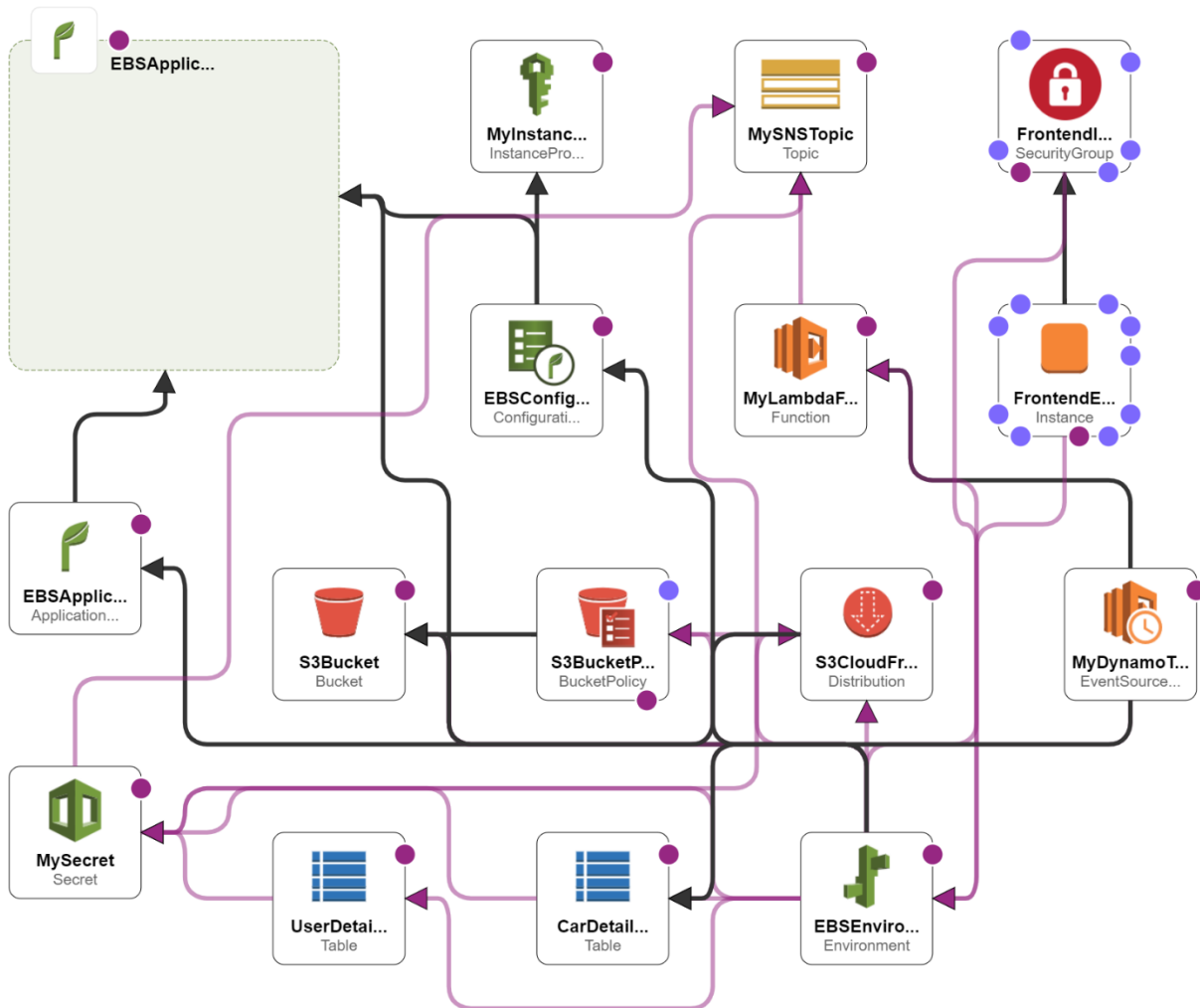


Figure 5. Cloud Formation Design

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

The AutoHub application makes use of the Service Load Balancing Architecture that was taught in class.

## 5. Security

### 5.1. How does your application architecture keep data secure at all layers?

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.)

## **Presentation Layer:**

In my application, the front end is isolated from the backend and database layers. To achieve this I have deployed each tier of the architecture on a separate resource.

## **Security Mechanism Used**

**1. Network-Based Security:** I have made use of security groups to control inbound traffic to my EC2 instance. It allows me to restrict access to only necessary ports and protocols. Security groups in AWS enable network-level security for EC2 instances. They act as virtual firewalls to control the flow of traffic into and out of the instances. Depending on the defined rules, security groups either permit or deny traffic. All incoming traffic is by default blocked but all outgoing traffic is permitted on an EC2 instance.

I have configured my security group in the EC2 instance to listen on Port number 22 when the admin logs in using SSH (and Port number 3000 where our EC2 instance is hosted)

## **Vulnerabilities that could be addressed with further work**

We can make use of HTTPS instead of HTTP for that reason we will require to configure SSL/TLS certificates for our front end hosted on EC2. HTTPS offers a safe communication route between the client and the server.

**2. IAM role-based security:** We have been provided with the Lab role which provides security by default. It consists of permissions managed by AWS as well as custom permissions managed by Vocareum Labs which we do not have permission to view.

## **Vulnerabilities that could be addressed with further work**

Currently I am not able to modify the existing configurations in the Lab role as it is designed by Vocareum Labs so having the permissions to modify would allow me to apply security patches and updates to your EC2 instance. The danger of security flaws and data breaches can be decreased as a result.

In case there are unusual activities on our EC2 instance we can keep a watch by using CloudWatch metrics and setting up alarms and notifications.

## **Business Layer (Server Side):**

To isolate my backend server, I have hosted my Node JS application on AWS Elastic Beanstalk.

For my server security mechanisms considered are:

## Security Mechanism Used

**1. Access Control:** EBS consists ability to integrate with AWS Identity and Access Management. We could create IAM roles and policies to restrict access to our application.

While deploying our backend server using AWS Elastic Beanstalk I have configured the security using parameters that consist of my AWS credentials and also attached the Lab Role IAM profile to the EBS so that users without the permissions don't have access to the server. Also the cloud private key that is unique has been configured during the creation. All this was done using the AWS CloudFormation template.

### Vulnerabilities that could be addressed with further work

The major data breach in the cloud occurs due to irregularity in IAM roles. Not providing appropriate permissions or missing out on some permissions. To address this IAM roles and policies that are associated with respective IAM roles must be implemented properly.

**2. Regular Automatic updates:** EBS automatically applies security fixes to the underlying operating system, the web server of your application environment. This enables us to ensure that our application is shielded from known security problems.

For my server side of the application, I am using 64bit Amazon Linux 2 v5.8.0 running Node.js 16 platform. EBS applies automatic updates on my platform based on the version that is running.

### Vulnerabilities that could be addressed with further work

With each update, there are chances that our system and code may not be compatible with the new updates. To encounter this issue we can test the updates in the staging environment and then deploy it to the production environment. This will ensure that our system is always available to the authorized users.

**The business Layer communicates with multiple AWS services each service provides additional security as follows:**

#### AWS DynamoDB

**Encryption:** DynamoDB automatically provides encryption to data at rest as well as data in transfer through AWS Key Management Service(KMS). During the transfer of data, DynamoDB protects data from unauthorized access by making use of HTTPS protocol.

**Backup and Restore:** In case there is a loss of data in DynamoDB we can configure and easily create and manage backups in DynamoDB

## AWS S3

**Versioning:** Simple Storage Service provides a way to keep multiple versions of the same object in the same bucket which would be useful in case of accidental deletion of car images in my application.

**Bucket Policies:** With bucket policies, I can provide deny and permit access to objects inside a bucket. In my application, I have allowed the s3:GetObject action so that images could be retrieved to render on the client side

## AWS CloudFront

**Origin Access Identity:** CloudFront uses an origin access identity to prevent unauthorized access to your origin server. (OAI). CloudFront to authenticate requests to your origin server.

**Access Control:** To manage who has access to your material, CloudFront can interact with AWS Identity and Access Management (IAM) and Amazon S3 bucket policies. I can limit access to S3 Bucket content for my application depending on the AWS account, IAM roles, or S3 bucket restrictions.

## AWS SNS

**Monitoring:** SNS provides a way by which we can know which emails were not delivered using Dead letter queues. This will help us the way to monitor delivery failures.

**Encryption:** SNS supports both at-rest and in-transit encryption. Using AWS Key Management Service we can create and manage encryption keys for SNS.

## AWS Lambda

**No Direct Access:** In my case, AWS Lambda avoids direct access to the SNS. Whenever new data is entered in the DynamoDB table Lambda is triggered that further triggers the SNS.

**Runtime Isolation:** Lambda runs code in a container that is isolated from other infrastructure. This help to prevent malicious code from affecting other resources.

## 6. On-Premise Cost

**6.1 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?**

Our application would have a user base of approximately 50000-60000 users.  
To reproduce the architecture in a private cloud without compromising the original public cloud implementation in terms of availability, redundancy, and many other factors we would need multiple resources in our private environment.

To serve the number of users mentioned above we might need multiple servers(assuming 5) that can handle the traffic load. Considering the cost of an Intel E3-1240 v5 (3.50 GHz) Server the total approximate would be

#### **Server**

The approximate cost that would be required are [3].

Approximate cost:

Intel E3-1240 v5 (3.50 GHz) Server

The upfront cost for 1 Server:

Server - \$1476

Ram - \$147

HDD - \$95

Total = approx. \$ 1800

For 5 servers it would be,

$1800 * 5 = \$ 9000$

#### **Virtualization Software**

The cost of virtualization software such as VMware ranges from \$4000 - \$8000 per server depending on the vendor we select.

The total cost of virtualization software for our 5 servers would range from \$20000-\$40000 approximately.

#### **Load Balancer**

The cost of an Enterprise VA 1G price is approx. \$4500 for a one-time purchase. Assuming that we might require 2 such devices, The total cost of a Load Balancer would be \$9000 - \$10000 approximately [4].

#### **Storage Cost**

If we want to store user and car details data we would need approximately 1 Tera byte of data for the AutoHub application.

The storage cost would be around \$100-\$150 per month which would be approximately \$1200 - \$1800 yearly.

#### **Cooling and Electricity**

The modern server comes with an integrated cooling system but still, we might consider it. Cooling and Electricity contribute to approximately 50% of the server cost.

The Cooling and Electricity cost would be around \$4000 - \$5000 approximately



## **Network Requirements**

### **Routers**

CISCO ISR4221/K9 Router: \$1200 approx [5].

### **Firewalls**

Firewalls play an important role in deciding which traffic should enter or not enter. They enhance the security of the system.

Fortinet fortigate : \$250-\$2000 approx [6].

### **Bandwidth**

To handle the traffic in the AutoHub application we might require approximately 500 Gbps which would cost approx. \$800 - \$1000 monthly which sums up to \$12000 yearly.

### **Other Costs**

As we have migrated on-premise we might require staff and tools for maintenance tasks of the servers and other resources which would cost around \$12000 -\$15000 yearly.

### **Operating System License**

Windows Standard Edition: \$950-1100 approximately [7].

## **7. Monitoring**

**Which cloud mechanism would be most important for you to add monitoring 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?**

Comparing all of the AWS services, the cloud computing and storage technologies used in our AutoHub application have the potential to be the most expensive. The project would incur substantial costs because the backend of the application is hosted on EBS and can auto-scale to multiple EC2 resources

AWS Elastic Beanstalk would require monitoring to make sure that costs do not exceed, Data movement between AWS EBS and other AWS services, such as Amazon EC2 instances, is subject to fees. To prevent unforeseen charges while using AWS EBS with EC2 instances monitoring is a must.

We can make use of AWS CloudWatch for monitoring purpose. We can keep an eye on a AWS resources, such as EC2 instances and EBS volumes, with the help of CloudWatch, which also offers a thorough analysis of their usage and performance.

We can set up CloudWatch to track important cost-related variables, like EBS volume utilization and data transfer usage patterns that affect your costs, to make sure that costs don't rise abruptly.

## **8. Applications Future Scope(Evolution)**

**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?**

Currently, AutoHub is developed to connect car sellers and buyers but considering the future scope of the application new features could be added.

Features that could be added in the future:

### **1. Chat Box**

Currently, the only way of communication is email notification but considering the future we could implement a chat box on the website to connect sellers and buyers more flexibly.

Target Cloud Services

#### **Amazon Lex**

This could be used to build conversational interfaces on which chat applications could be implemented.

Amazon Lex comes with integrated SDKs. Amazon Lex SDKs provide support for numerous programming languages which gives us a lot of options. Programming languages include Java, Python, and Node.js. It becomes easy to integrate with our existing application if it's already based on one of the following languages.

### **2. Product Geo-Location:**

This feature could help the buyers and sellers to calculate the cost and distance to reach out to each other in case of an in-person meet-up.

Target Cloud Services

#### **Amazon Location Service**

Amazon Location Service, a fully-managed service provided by AWS, makes it easy to add location-based features and capabilities to your application. It offers several APIs that allow programmers to integrate maps, places of interest, navigation, geocoding, and other features into their applications. The ability to determine on a map the location of the vehicles (or other specified products) would be helpful for my application.

## References:

- [1] "Key Features of a Content Delivery Network – Performance, Security – Amazon CloudFront," *Amazon Web Services, Inc.*, 2013. <https://aws.amazon.com/cloudfront/features/?whats-new-cloudfront.sort-by=item.additionalFields.postDateTime&whats-new-cloudfront.sort-order=desc> (accessed Apr. 11, 2023).
- [2] "Performance Test - Check URL Speed From 10 Global Locations | KeyCDN Tools," *KeyCDN*, 2023. <https://tools.keycdn.com/performance?url=Netflix.com> (accessed Apr. 11, 2023).
- [3] "What's the Cost of a Server for Small Business? - ServerMania," *Servermania.com*, Feb. 10, 2023. <https://www.servermania.com/kb/articles/how-much-does-a-server-cost-for-a-small-business/> (accessed Apr. 11, 2023).
- [4] Load Balancer Enterprise ADC, "Load Balancer Enterprise ADC Pricing, Packages & Plans 2023 | G2," *G2*, 2023. <https://www.g2.com/products/load-balancer-enterprise-adc/pricing> (accessed Apr. 11, 2023).
- [5] "Routers Overview, Solutions, Brands and Products," *Router-switch.com*, 2020. <https://www.router-switch.com/routers.html> (accessed Apr. 11, 2023).
- [6] J. Ferguson, "Network Firewalls Pricing Comparison," *TrustRadius for Vendors*, Oct. 05, 2020. <https://www.trustradius.com/buyer-blog/network-firewall-pricing-comparison> (accessed Apr. 11, 2023).
- [7] "How Much Does a Windows Server License Cost? - ServerMania," *Servermania.com*, Feb. 11, 2022. <https://www.servermania.com/kb/articles/how-much-does-a-windows-server-cost/> (accessed Apr. 12, 2023).