



GROUP 6: FINAL REPORT

Group Project – CSCI 5409



APRIL 8, 2021

BALA SUNDEEP KRISHNA DASARI – B00869486

PRASHANT SANJAY SARVI – B00870599

Business Considerations

Considering there would be 10,000 users to our reminder application in the first year and an average user creates 20 reminders every month, we made the following estimates:

Current infrastructure costs:

- These costs include the provisioning of AWS resources and other services like whoisXMLAPI [1] and Redis [3].
- It is clear from Figure 1 that the approximate annual cost of using AWS services is around 750 USD.
- In addition to these costs, we are using Redis [3] service from Heroku [4]. This is a free add-on in Heroku for a limited number of requests. Once the limit is reached, the cost would be around 20 USD per month (240 USD annually).
- We are also using the whoisXMLAPI email verification service [1], which costs around 60 USD per annum.
- Therefore, the total cost of provisioning resources to our application is approximately 1050 USD per annum.

Private Infrastructure costs (Private cloud):

- We primarily need a server environment to host our application. We would require at least two instances with 8GB of primary memory each and a ubuntu operating system. The maintenance cost for this purpose would be 450 USD per annum.
- Additional software like MSOffice, VMWare, Antivirus, etc., would cost a total of 250 USD per annum.
- The next important cost would be our database. Considering the database would require 1TB of storage capacity per annum, the cost would be 120 USD per annum.
- An additional cost of 250 USD is required for other provisionings like firewalls, proxies, domain costs, and security maintenance.
- To host a Redis server [3] in our environment, we would need at least 1TB of memory which would cost 120 USD per annum.

- Other physical and geographical maintenance costs would sum up to 250 USD per annum.
- Other third-party service usages like whoisXMLAPI [1] would cost around 100 USD per annum.
- This would be a total cost of 1440 USD per annum.

It can be concluded that using a shared hosting environment like AWS is a better option for new businesses.

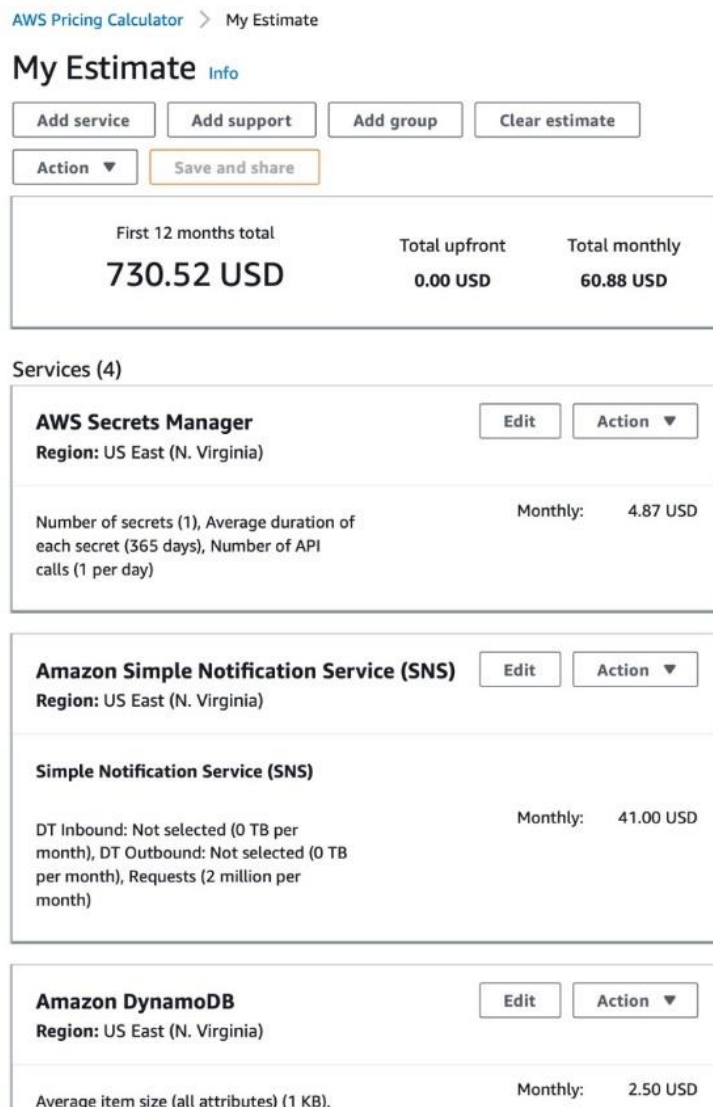


Figure 1: Estimated annual costs of AWS services. Made using AWS calculator [22]

Mechanisms that need most Attention (monitoring):

There are two main mechanisms in our application that might cost a huge amount if not monitored carefully.

AWS SNS:

- AWS SNS [2] is the mechanism used in our core functionality to send reminders to users. If not monitored properly, it can cost a huge amount when there is high incoming traffic to our application.
- User notifications are affected based on the availability of this service, which will affect the functioning of sending notifications.

Redis:

- Redis [3] is as crucial as AWS SNS [2]. This is because the queue jobs trigger AWS SNS [2] functions and the Redis server [3] maintains the queue jobs. Similar to AWS SNS [2], it may cost a very high amount to our application if the traffic increases.
- The availability of our instance is directly dependent on the availability of this service as it can lead to the crashing of our application.

Final Architecture

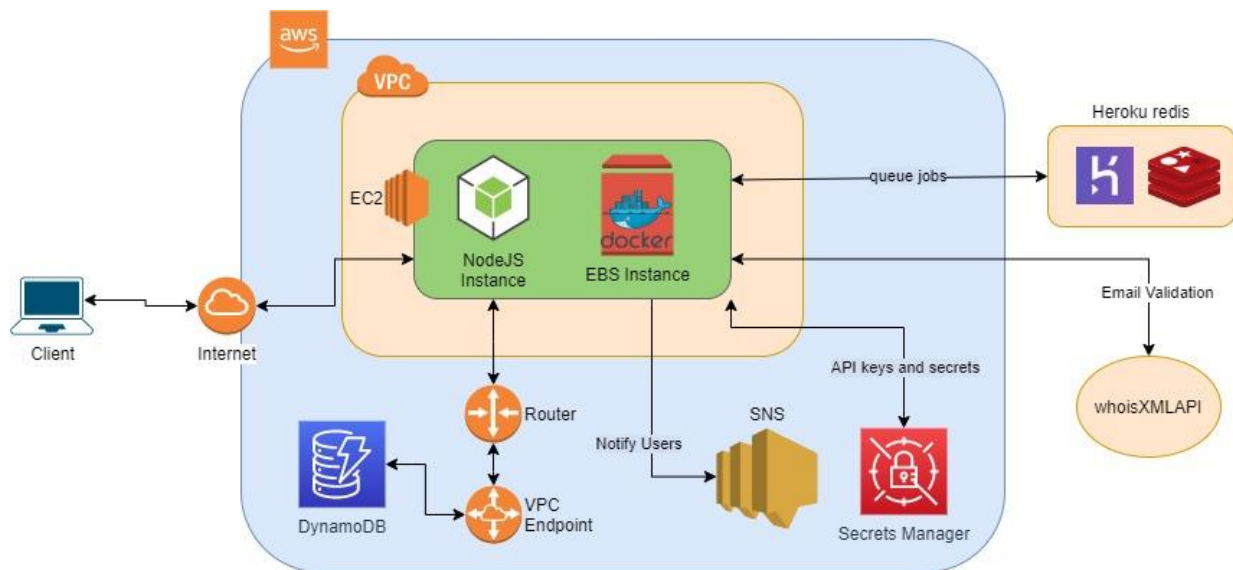


Figure 2 - Reminder application architecture. Drawn with draw.io [21]

Cloud Mechanisms and their purpose

- We deployed our application as an AWS EC2 [5] instance on an *ubuntu* server with a *t2.micro* instance type, as shown in Figure 3. We created a new security rule to allow only port 8000 on all TCP connections.
- We also deployed another instance of our application as a *docker image* on AWS *Elastic Beanstalk* (with NodeJS [7] platform) [6].
- Both of the above deployments lead to having two different URLs (two separate applications).
 - EC2: <http://3.86.240.134>
 - EBS: <http://group6.eba-pj4didhu.us-east-1.elasticbeanstalk.com/>

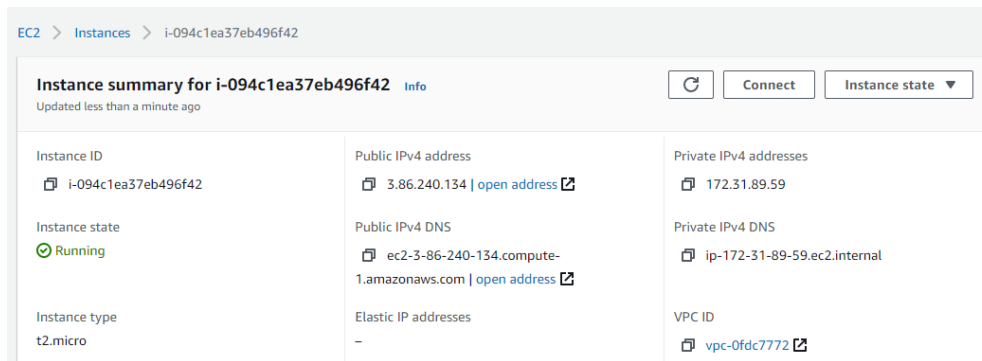


Figure 3 - EC2 instance configuration

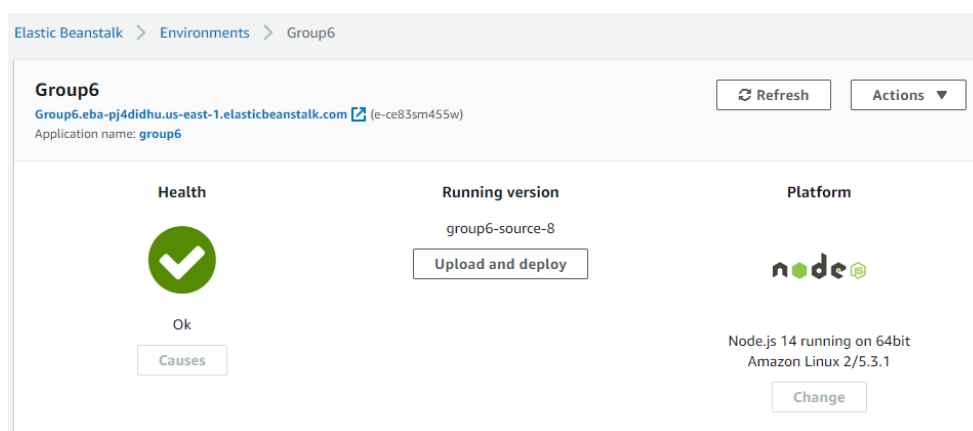


Figure 4 - EBS instance health check

- AWS VPC [8] is automatically provisioned when an EC2 [5] instance is successfully created. We updated the VPC [8] by adding a new endpoint to our DynamoDB [9] service. This makes our storage secure by allowing access of our data only through the instances inside our VPC [8] as shown in Figure 2.

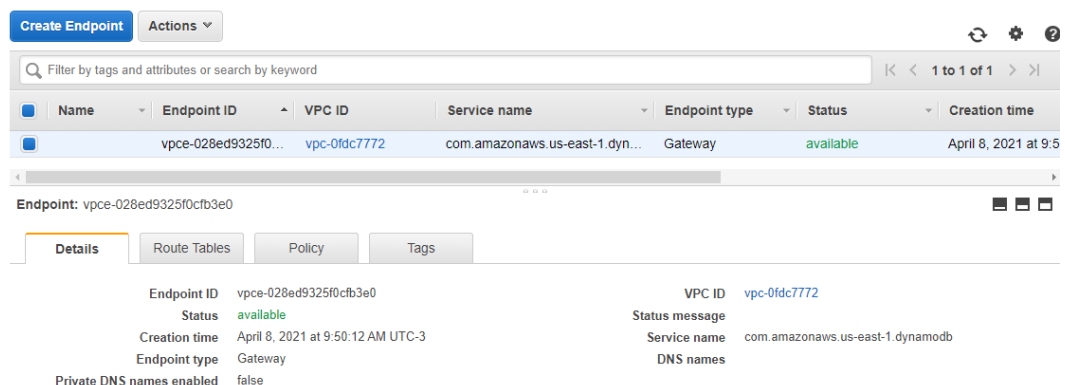


Figure 5 - VPC endpoint for DynamoDB

- AWS Secrets Manager [10] is used to store all our application related configurations including API keys and encryption secrets.

- Our application mainly focuses on providing the capability of setting reminders via email to our users. For this purpose, AWS SNS [2] is used for user subscriptions, publishing messages (sending emails), and maintaining the subscription status of the users.
- We use DynamoDB [9] as our backend data storage. A VPC [8] endpoint is added, as shown in Figure 5. This is to ensure that there is no scope for unauthorized access to the database, which would affect the availability and integrity of the data stored.
- Our application needs the capability of running background jobs (specifically queue jobs) to keep track of when a user needs to be notified. A Heroku-Redis addon hosted in the Heroku platform [4] serves this purpose by maintaining the queue jobs.
- We also use a third-party service named whoisXMLAPI [1] to verify the correctness of emails provided by users during the sign-up process.

Data Storage:

- AWS DynamoDB [9] is used for storing all the data related to our application.
- We configured a VPC [8] endpoint to our DynamoDB [9] to restrict its access to the outside world. Only the instances hosted in our VPC [8] can access our DynamoDB [9] service.
- We also configured a global secondary index for all the tables to make efficient queries. Without this secondary index, every query scans all the records and then filters the response, which is not efficient.

Tech Stack:

We used JavaScript technologies to build our application as they are developer-friendly and, easy to implement and maintain.

- Front-end: We used ReactJS [11] and Bootstrap [12] for our front end. The main reason for using these is to make our UI responsive and render components in a dynamic and modular way rather than in a static way.
- Back-end: We used ExpressJS [13] and NodeJS [7] for our backend. This is because ExpressJS [13] has a variety of capabilities for making API call using REST and NodeJS [7] comes with npm (node package manager) that can be used for a variety of integrations like “aws-sdk”, and “redis”.

Features implemented through code:

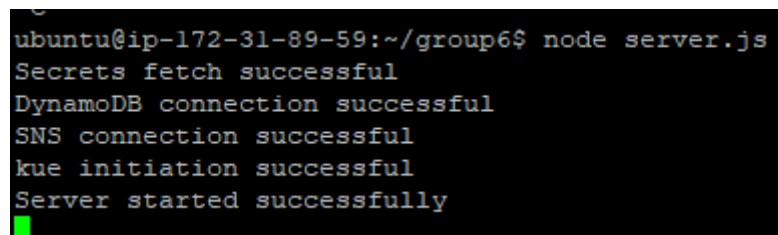
- We used the aws-sdk library to implement functionalities of SNS [2] like publishing mails, sending subscription requests, and fetching the status of a subscription.
- We use the aws-sdk library to implement the feature of getting secrets from Secrets Manager [10] in AWS.

Cloud Deployment:

We deployed our application as two different instances using the following services:

AWS EC2:

- We configured an AWS EC2 [5] instance with Ubuntu AMI and an instance type of t2.micro and the instance is successfully running on the IP address "3.86.240.134".
- We connected from our laptop with an SSH login using the secret key generated while building our application. We created our project directory in the remote instance and copied all the necessary elements of code into it.
- We then installed NodeJS [7] and npm on our ubuntu instance and started our application successfully on port 8000, as shown in Figure 6.

A terminal window with a black background and green text. The prompt is 'ubuntu@ip-172-31-89-59:~/group6\$'. The command 'node server.js' has been executed, resulting in the following output: 'Secrets fetch successful', 'DynamoDB connection successful', 'SNS connection successful', 'kue initiation successful', and 'Server started successfully'. A green cursor is visible at the end of the last line.

```
ubuntu@ip-172-31-89-59:~/group6$ node server.js
Secrets fetch successful
DynamoDB connection successful
SNS connection successful
kue initiation successful
Server started successfully
```

Figure 6 - EC2 instance successful start logs

AWS Elastic Beanstalk:

- We first created a new directory on our laptop and copied all the required contents to that directory.
- We then created a Dockerfile to represent the requirements of our application in the remote host, as shown in Figure 7.


```
File Edit Format View Help
FROM node:14

WORKDIR /group6

COPY package*.json /group6

RUN npm install

COPY . /group6

EXPOSE 8000|

CMD [ "node", "server.js" ]
```

Figure 7 - Dockerfile contents

- Figure 7 shows the requirements of our application to be deployed as a docker image. This file tells the Elastic Beanstalk to install NodeJS (version 14) [7], create a new directory named “group6” and copy all the content to that directory and start the server on 8000 port.
- We then uploaded our zip file to AWS Elastic Beanstalk [6], and the instance got created successfully, as shown in Figure 3.

Final Architecture vs Project Proposal

AWS SQS:

- Initially, we planned to use SQS [14] as the queuing service for our application. Later, we realized that the maximum delay for a queue job in SQS [14] is just 15 minutes.
- Our users can set reminders even for the next year. Hence, AWS SQS [14] is not feasible for our application which is why we chose to use Redis for running queue jobs.

Twilio API:

- We also planned to use Twilio [15] service to validate phone numbers provided by the users and send SMS using AWS SNS [2].
- We removed this feature from our application due to the requirement of more research related to pricing strategies of sending push notifications and other time constraints.

AWS IAM:

- Initially, we thought to implement AWS IAM [16] service for role-based access to our application and we realized that our accounts do not support using IAM [16] for creating roles due to insufficient permissions.
- We then planned to use AWS VPC [8] for securing our DynamoDB [9] access to meet the project requirements of using at least one network service from the given list.

Future Enhancements

Application Functionalities:

- *Friends*: This is an enhancement for our reminder application where users can send friend requests to other users. Once the other person accepts the request, they can create group reminders and share group notes.
- *Reminder inside Notes*: As of now, our notes feature supports only making notes with title and message. We are planning to integrate reminders inside notes as well. This enables every user to efficiently manage their schedule in a full-fledged manner.
- *Notifications*: We are sending only email notifications as of now. This can be easily extended to sending in-app notifications and SMS by adding a couple of existing cloud services like Twilio [15].
- *Mobile App*: Our application is a responsive web app. It is developed with the mobile-first design in mind. This makes the launching of our application in the App Store and Play store easier.

Cloud Services:

- *AWS Lambda*: We implemented most of the services using APIs and libraries inside the code. We are planning to automate some of our services like publishing SNS [2] messages using AWS lambda [17]. This makes our application less vulnerable to bugs.
- *AWS SES [18]*: We are sending emails using AWS SNS [2], which does not provide the flexibility of having our subject to emails and supporting multipart data in messages.
- *AWS CloudWatch [19]*: This service provides a very useful functionality of monitoring the usage of our cloud services. This is a must-have feature to reduce costs and resource utilization.
- *AWS API Gateway*: When we implement lambda functions in the future, we can use the API Gateway service [20] to build safe and secure endpoints to our lambda functions and provide role-based access.

References

- [1] "WhoisXML API: #1 for Domain, WHOIS, IP, DNS & Threat Intelligence", Whoisxmlapi.com, 2021. [Online]. Available: <https://www.whoisxmlapi.com/>.
- [2] "Amazon Simple Notification Service (SNS) | Messaging Service | AWS", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/sns/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc>.
- [3] "redis/redis", GitHub, 2021. [Online]. Available: <https://github.com/redis/redis>.
- [4] "Cloud Application Platform | Heroku", Heroku.com, 2021. [Online]. Available: <https://www.heroku.com/>.
- [5] "Amazon's EC2 vs Google Compute Engine", Cloud Academy, 2021. [Online]. Available: <https://cloudacademy.com/blog/ec2-vs-google-compute-engine/>.
- [6] "Amazon Elastic Block Store (EBS) - Amazon Web Services", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/ebs/>.
- [7] "Node.js", Node.js, 2021. [Online]. Available: <https://nodejs.org/en/>.
- [8] "Amazon Virtual Private Cloud (VPC)", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/vpc/?vpc-blogs.sort-by=item.additionalFields.createdDate&vpc-blogs.sort-order=desc>.
- [9] "Amazon DynamoDB | NoSQL Key-Value Database | Amazon Web Services", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/dynamodb/>.
- [10] "AWS Secrets Manager | Rotate, Manage, Retrieve Secrets | Amazon Web Services (AWS)", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/secrets-manager/>.
- [11] "React – A JavaScript library for building user interfaces", Reactjs.org, 2021. [Online]. Available: <https://reactjs.org/>.
- [12] "React-Bootstrap", React-bootstrap.github.io, 2021. [Online]. Available: <https://react-bootstrap.github.io/>.
- [13] "Express - Node.js web application framework", Expressjs.com, 2021. [Online]. Available: <https://expressjs.com/>.
- [14] "Amazon SQS | Message Queuing Service | AWS", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/sqs/>.
- [15] "Twilio - Communication APIs for SMS, Voice, Video and Authentication", Twilio, 2021. [Online]. Available: <https://www.twilio.com/>.
- [16] "AWS Identity & Access Management - Amazon Web Services", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/iam/>.
- [17] "AWS Lambda – Serverless Compute - Amazon Web Services", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/lambda/>.
- [18] "Amazon Simple Email Service | Cloud Email Service | Amazon Web Services", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/ses/>.
- [19] "Amazon CloudWatch - Application and Infrastructure Monitoring", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/cloudwatch/>.
- [20] "AWS Identity & Access Management - Amazon Web Services", Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/iam/>.
- [21] "Flowchart Maker & Online Diagram Software", App.diagrams.net, 2021. [Online]. Available: <https://app.diagrams.net/>.
- [22] "AWS Pricing Calculator", Calculator.aws, 2021. [Online]. Available: <https://calculator.aws/>.