



**CSCI 5409 – Cloud Computing**

**Mechanisms Critical Analysis and Response**

**Work done by,**

**GROUP 31**

- **Guturu Rama Mohan Vishnu – B00871849**
- **Aditya Deepak Mahale – B00867619**
- **Sumit Singh – B00882103**

The list of mechanisms that our project uses in each category are as follows:

- **Compute:** Amazon EC2
- **Network:** AWS Virtual Private Cloud (VPC)
- **Storage:** Amazon S3
- **Security:** We are using VPC here. So, we would cover Security Groups, NACLs and Private Subnet for security from VPC.
- **Serverless Computing:** Lambda
- **Other mechanisms:** Polly and Translate

## EC2:

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services Cloud. Using Amazon EC2 eliminates our need to invest in hardware up front, in order for us to develop and deploy applications faster. We can use Amazon EC2 to launch as many or as few virtual servers as we need, configure security and networking, and manage storage. Amazon EC2 enables us to scale up or down to handle changes in requirements or spikes in popularity, reducing the need to forecast traffic.

- In our project, we are using EC2 instances for multiple purposes such as to host our frontend project which will be written in ReactJS,
- It is also used to host our backend REST API project and for installing a database either MongoDB or PostgreSQL.
- We will also use it to bastion host for accessing VMs on a private subnet.
- Alternative for EC2 for this project is AWS Elastic BeanStalk, if considering in the AWS range. If we go out of AWS range, the alternatives for EC2 are Google Compute Engine by Google, Microsoft Azure Virtual Machines, IBM Cloud Virtual Servers, DigitalOcean etc.
- The reason we picked AWS EC2 instances over its alternatives is that AWS EC2 is more popular and the most widely used around the world to host web applications. Here are some more reasons why we choose AWS EC2.

Elastic Web Scale Computing: With the help of EC2, we can increase or decrease our capacity within minutes and there is no need to wait for hours or even days for that to happen. We can also commission our server instances simultaneously in the range from one instance to thousands of instances. And since this is all controlled by web service APIs, our application can automatically do the scaling itself.

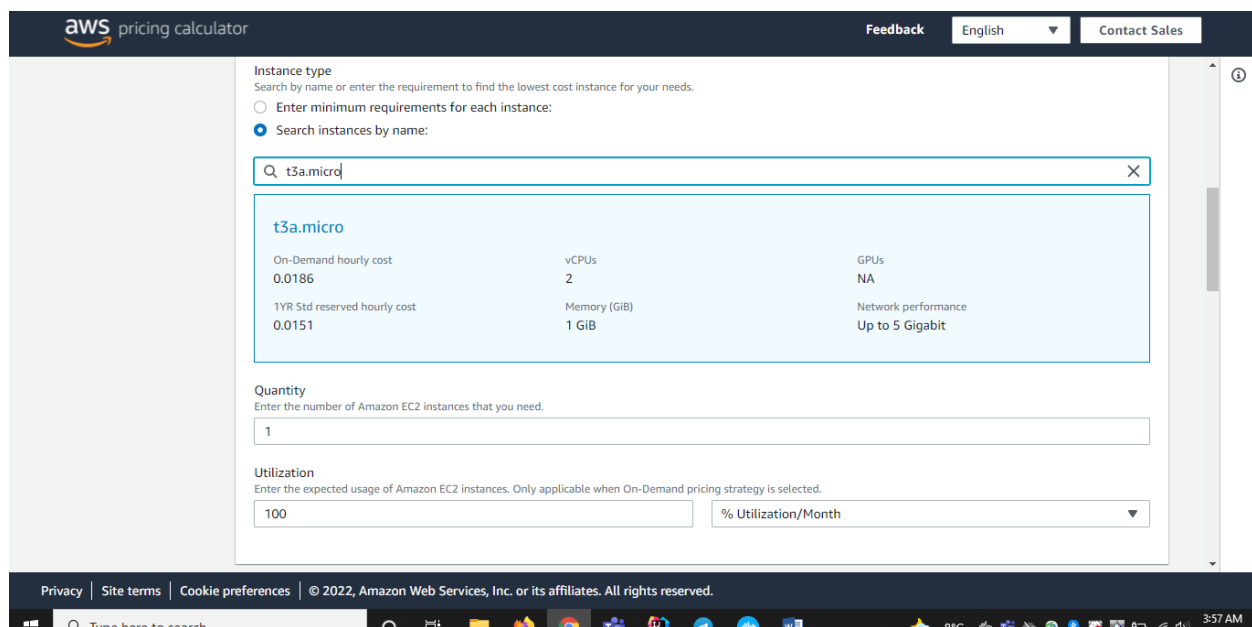
Complete Control: No matter how many number of instances I have on my server, I will have the full control of each and every instance. I will have the root access to each one and I can interact with them whenever I want to. I will also have the access to console output of our instances.

Reliable: EC2 offers a highly reliable environment where replacement of instances can happen rapidly. The service runs with Amazon's proven network infrastructure and data centers.

Secure: AWS EC2 works in conjunction with Amazon VPC to provide security and robust networking functionality. The instances which we create will be located on a Virtual Private Cloud (VPC) with an IP range that I can specify. I also can decide which instances are to be exposed to the internet and which instances should be kept private.

Low Cost: When compared with its alternatives, EC2 provides us all these features on a pretty low cost for the compute capacity which I actually use.

Pricing: For this project, since we are not making it public and we are only accessing to a limited number of customers, we decided to go with either “t2.micro” or “t3a.micro”. The reason behind our decision is that for both of these instance types, we get one year of free service for up to 750 hours of availability every month.



The screenshot displays the AWS Pricing Calculator interface. At the top, there's a header with the AWS logo, 'pricing calculator', and links for 'Feedback', 'English', and 'Contact Sales'. The main section is titled 'Instance type' and includes a search bar where 't3a.micro' has been entered. Below the search bar, a table lists the instance's specifications: 'On-Demand hourly cost' (0.0186), '1YR Std reserved hourly cost' (0.0151), 'vCPUs' (2), 'Memory (GiB)' (1 GiB), 'GPUs' (NA), and 'Network performance' (Up to 5 Gigabit). The 'Quantity' section shows '1' instance, and the 'Utilization' section shows '100' % Utilization/Month. The footer contains links for 'Privacy', 'Site terms', and 'Cookie preferences', along with a copyright notice for 2022 Amazon Web Services, Inc. The Windows taskbar is visible at the bottom of the screen.

Instance type	On-Demand hourly cost	1YR Std reserved hourly cost	vCPUs	Memory (GiB)	GPUs	Network performance
t3a.micro	0.0186	0.0151	2	1 GiB	NA	Up to 5 Gigabit

If in any case, our utilization crosses the 750 free hours per month, then the monthly cost we would have to pay for our utilization is as follows. So, we can say that pricing is a major factor behind our decision to go with EC2 and we can also say that we don't need to worry much about our money of how much to lose.

aws pricing calculator

FeedbackEnglishContact Sales

EC2 Instance Savings Plans

Compute Savings Plans

Standard Reserved Instances

Convertible Reserved Instances

On-Demand Instances

1 Year

3 Year

No Upfront

Partial Upfront

All Upfront

Show calculations

Unit conversions

EC2 Instance Savings Plans rate for t3a.micro in the US East (N. Virginia) for 1 Year term and No Upfront is 0.0151 USD

Hours in the commitment: 365 days \* 24 hours \* 1 year = 8760.0000 hours

Total Commitment: 0.0151 USD \* 8760 hours = 132.2760 USD

Upfront: No Upfront (0% of 132.276) = 0.0000 USD

Hourly cost for EC2 Instance Savings Plans = (Total Commitment - Upfront cost)/Hours in the term: (132.276 - 0)/8760 = 0.0151 USD

\*Please note that you will pay an hourly commitment for Savings Plans and your usage will be accrued at a discounted rate against this commitment.

Pricing calculations

1 instances x 0.0151 USD x 730 hours in month = 11.02 USD (monthly instance savings cost)

Amazon EC2 Instance Savings Plans instances (monthly): 11.02 USD

## LAMBDA:

We'll be using IaaS + FaaS delivery model for our project. So, for FaaS we are going forward with lambda function. AWS Lambda is one of the popular Function as a Service (FaaS) implementations in a public cloud. When we use Lambda, we are only responsible for our code. Lambda manages the compute fleet, which provides a good mix of memory, CPU, network, and other resources to run your code. Lambda handles operational and administrative tasks for you, such as managing capacity, monitoring, and logging your Lambda functions.

- In our project we are using lambda function to trigger to store converted and processed audio file into the S3 bucket and we are also using it to stream audio file from S3 bucket up on requested.
- Alternatives of lambda function are AWS Elastic Beanstalk, AWS Step Functions provided by AWS cloud provider. We also have other FaaS supporting functionality provided by other cloud providers few are as follows, Google Cloud Functions by Google Cloud platform, Azure Functions by Microsoft and IBM OpenWhisk by IBM. These all are follow-ups of AWS lambda and introduced in 2016 by respective providers.
- Reason for picking AWS lambda over other alternatives are AWS Lambda is the more mature and most popular.

Pay per use: We pay for what we use with AWS lambda and With Lambda, we can run code for almost any type of application or backend service with no administration and only pay for what we use. You are charged based on the number of requests for your functions as well as the length of time it takes for your code to execute.

Scaling: Auto Scaling with AWS Lambda is known to be the best in the market and it has a capability of handling few requests per months to million requests per second.

Cold Starts: A FaaS instance in an inactive state will require some additional time to respond to a request. This initial delay encountered is known as a cold start.

Service	Average Cold Start (in Minutes)
---------	---------------------------------

AWS Lambda	6-7
------------	-----

Azure Functions	20-30
-----------------	-------

GCP Cloud Functions	15
---------------------	----

Execution time: The maximum execution time is another configurable aspect of FaaS. While most functions in the wild take seconds (or less) to execute, some intensive workloads can potentially take minutes, if not hours (for example, intensive machine learning or data analysis workloads).

Service	Maximum Timeout
---------	-----------------

AWS Lambda	15 minutes
------------	------------

Azure Functions	5 minutes (Consumption Plan) 30 minutes (Premium and Dedicated Plans)
-----------------	--

GCP Cloud Functions	9 minutes
---------------------	-----------

Memory: Memory will need to be adjusted depending on how resource-hungry the code is. If you allocate too little memory, a function will take longer to execute and may time out, but if you allocate too much memory, you may end up overpaying for unused resources.

Cloud providers provide varying maximum memory configurations, while CPU power is configured linearly and automatically in proportion to the amount of memory selected.

Service	Memory
---------	--------

AWS Lambda 128 MB – 10240 MB

Azure Functions 128 MB – 1500 MB (Consumption Plan)  
128 MB – 14000 MB (Premium and Dedicated Plans)

GCP Cloud Functions 128 MB – 4096 MB (in multiples of 128 MB)

Pricing: It costs us around 0.19 Dollar per month for 100,000 requests with 100 milliseconds of execution time and we have allocated 1 GB of memory. For our project pricing won't have that much of impact. And our execution time is lot faster than given for pricing.

**Service settings** [Info](#)

The calculations below exclude free tier discounts.

Architecture

x86

Number of requests

100000

per month

Duration of each request (in ms)

Duration is calculated from the time your code begins executing until it returns or otherwise terminates.

100

Amount of memory allocated

Enter the amount between 128 MB and 10 GB

1024

MB

▼ Show calculations

Unit conversions

Amount of memory allocated: 1024 MB x 0.0009765625 GB in a MB = 1 GB

Pricing calculations

100,000 requests x 100 ms x 0.001 ms to sec conversion factor = 10,000.00 total compute (seconds)

1 GB x 10,000.00 seconds = 10,000.00 total compute (GB-s)

10,000.00 GB-s x 0.0000166667 USD = 0.17 USD (monthly compute charges)

100,000 requests x 0.0000002 USD = 0.02 USD (monthly request charges)

0.17 USD + 0.02 USD = 0.19 USD

**Lambda costs - Without Free Tier (monthly): 0.19 USD**



## AWS VPC:

Amazon Virtual Private Cloud gives us full control over virtual networking environment, including resource placement, connectivity, and security. It allows us to define network connectivity and restrictions between web servers, application servers, and databases.

- We are planning to use VPC to host multi-tier web applications. The backend database VMs will reside on the private subnet while frontend VMs will reside on the public subnet.
- Bastion host on the public subnet will give us access to the VMs hosted on the private subnet.
- We'll be able to enforce restrictions by using security groups and NACLs.
- Other cloud providers also provide virtual network services such as Azure virtual network and Google VPC.
- Implementation of VPC across different cloud providers is the same. All three major cloud providers (AWS, Azure, and GCP) offer various services to connect on-premise networks with the cloud.
- Our entire cloud resources would reside on AWS. We are not planning to implement a multi-cloud environment. Hence, we decided to go for AWS VPC.

Pricing: We are planning to use NAT Gateway to allow private VMs to access the internet. For testing, we are estimating that we'll utilize 50GB of data in total. For a month, it would cost us around 35 USD.

The screenshot shows the AWS Pricing Calculator interface. The 'Network Address Translation (NAT) Gateway' service is selected. The settings are configured as follows:

- Number of NAT Gateways: 1
- Data Processed per NAT Gateway: 50 GB per month

The calculations shown are:

- 730 hours in a month x 0.045 USD = 32.85 USD (Gateway usage hourly cost)
- 50 GB per month x 0.045 USD = 2.25 USD (NAT Gateway data processing cost)
- 32.85 USD + 2.25 USD = 35.10 USD (NAT Gateway processing and month hours)
- 1 NAT Gateways x 35.10 USD = 35.10 USD (Total NAT Gateway usage and data processing cost)
- Total NAT Gateway usage and data processing cost (monthly): 35.10 USD**

## **POLLY:**

Amazon Polly is a service that turns text into lifelike speech, allowing you to create applications that talk, and build entirely new categories of speech-enabled products. Polly's Text-to-Speech (TTS) service uses advanced deep learning technologies to synthesize natural sounding human speech. With dozens of lifelike voices across a broad set of languages, you can build speech-enabled applications that work in many different countries.

- We are planning to use AWS Polly to convert text into speech because it provides dozens of languages and a wide selection of natural-sounding male and female voices.
- In our architecture, AWS Polly would be in the second layer to convert translated text to speech.
- One important reason for choosing this service was because it would allow us to do unlimited replays of generated speech without any additional fees.
- There are plenty of alternatives of Polly on other major cloud providers such as Azure Text to Speech API and Google Cloud Text-to-Speech.
- We chose Polly because it would integrate well with other AWS services such as S3 and Lambda. Setting up a text-to-speech converter on other cloud platforms would cost us additional money. Also, the development time would be more to handle integration between different cloud platforms.

Pricing: AWS Polly 5 million free characters in AWS Free Tier. In the case of testing, we are assuming we'll need 5000 requests with a 400-character limit. It would cost us only 8.00 USD.

## Standard Text-to-Speech

**Standard Text-to-Speech (TTS)** [Info](#)

The calculations below exclude free tier discounts.

Number of requests (Standard Text-to-Speech)

per month ▼

Number of characters per request including white spaces, but excluding SSML

## ▼ Show calculations

5,000 request(s) x 400 character(s) = 2,000,000 character(s) per month

2,000,000 character(s) x 0.000004 USD = 8.00 USD (cost for Standard Text-to-Speech)

**Standard Text-to-Speech (monthly): 8.00 USD**

## TRANSLATE:

Amazon Translate is a neural machine translation service that provides fast, high-quality, affordable, and customizable language translation. Neural machine translation is a type of language translation automation that uses deep learning models to deliver more accurate and natural-sounding translation than traditional statistical and rule-based translation algorithms. We can easily translate large volumes of text for analysis using Amazon Translate, and we can efficiently enable cross-lingual communication between users.

- We are planning to use amazon real-time translation, it is best option as ywe want to deliver on-demand translations of content as a feature of our applications. It has a wide range of language support that we can pick from. Currently AWS Translate support translations between 75 languages.
- There are lot of alternatives available for AWS Translate in market, and few exist from long time like google translator. Google translator is one of the most widely used translator worldwide it also supports more than 100 languages and its cloud service is Google cloud translation API. Few other translators are Microsoft translator, IBM Watson language translator, Azure Translator Text API, Yandex Translate.
- After looking at all the available option for translator, AWS Translate and Google Translate are best suitable for our use case. There is no much difference between both, so as our application is mostly built on top of AWS we are planning to go with Amazon translate. Few factors for considerations are as follows

Pricing: Amazon does have a price advantage. Amazon Translate charges \$15 per million characters, while Google Cloud Translate charges \$20 for the same million characters. They both have a free offer. As part of the free tier, Amazon provides 2 million free characters per month, while Google provides half a million free characters per month.

Languages supported: Google distinguishes itself by offering more than 100 languages in comparison to Amazon's 75, as well as a corresponding increase in language pairs. But in our application, we are not providing support to all the languages and we have set of pre-defined languages that we support and AWS Translate supports those.

Language Identification: AWS Translate has this cool feature of identifying the source language, when its not specified. This feature will be a good fit for us if user forgets to give the type of input language.

Pricing: For million words per month AWS Translate costs us around 15 dollars USD. This is one of the biggest factors influencing the choice to pick AWS Translate. When compared with its competition Google Cloud Translate, AWS Translate is cheap and also gives 2 million free words per month, where as Google just gives .5 million words per month and costs around 20 dollars USD for one million words per month.

**Standard Real-Time Translation** [Info](#)

The calculations below exclude free tier discounts.

Number of characters including white spaces and punctuations (Standard Real-Time Translation)  
Average number of characters translated per month

▼ Show calculations

1,000,000 characters x 0.000015 USD = 15.00 USD (cost for Standard Real-Time Translation)

**Standard Real-Time Translation cost (monthly): 15.00 USD**

### S3:

Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can store and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, we can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.

To understand the structure of Amazon S3, here is the block diagram:



- We are planning to use Amazon S3 in our project to store the converted audio files and to host images for our front end. We also use that to store the input data which we get from the user and the result of that data.
- Talking about the alternatives of S3, there are many alternatives out there for storage purpose and AWS DynamoDB and AWS Aurora are two of them considering in the AWS range. Out of AWS range, Google Cloud Storage, Azure Blob Storage, DigitalOcean Spaces, IBM Cloud Object Storage, Oracle Cloud Infrastructure Object Storage and many others are example.
- After looking at all the available options for storage, we have concluded that AWS S3 and Google Cloud Storage would be the better picks of all. Since we are sticking to the range of AWS, the only option is AWS S3. Factors for considering it might be a few as follows.

Storage Management and Monitoring: All our files (objects) are stored in S3 buckets and can be organized and shared names called Prefixes. We can append up to 10 key-value pairs called S3 object tags to each object, which can be created, updated, and deleted throughout an object's lifecycle. To keep track of objects and their respective tags, buckets, and prefixes, we can use an S3 Inventory report that lists your stored objects within an S3 bucket or with a specific prefix.

Storage Classes: With Amazon S3, we can store data across a range of different S3 storage classes purpose-built for specific use cases and access patterns: S3 Intelligent-Tiering, S3 Standard, S3 Standard-Infrequent Access (S3 Standard-IA), S3 One Zone-Infrequent Access (S3 One Zone-IA), S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, S3 Glacier Deep Archive, and S3 Outposts. Every S3 storage class supports a specific data access level at corresponding costs or geographic location.

Access Management: To protect our data in S3, users will only have access to the resources created by them. We can grant access to other users also by following either AWS Identity and Access Management; or Access Control Lists (ACL); or Bucket Policies; or S3 Access Points; or Query String Authentication.

Data Transfer: AWS provides a portfolio of data transfer services to provide the right solution for any project. The level of connectivity is a major factor in data migration, and AWS offers services that can handle hybrid cloud storage, online data transfer and offline data transfer.

Performance: S3 supports parallel requests, which means I can scale up my S3 performance by the factor of my compute cluster, without making any customizations to my application. There are no limits to the number of prefixes, which directly scales the performance. S3 performance supports at least 3500 requests per second to add data and 5500 requests per second to retrieve the data.

Pricing: We pay for storing our objects in S3 buckets. The rate we are charged depends on our objects size, how long we store the objects during the month and the storage class. There are per-request ingest charges when using PUT, COPY or lifecycle rules to move data into any S3 storage class.

▼ S3 Standard [Info](#)

The calculations below exclude Free Tier discounts.

S3 Standard storage

PUT, COPY, POST, LIST requests to S3 Standard

GET, SELECT, and all other requests from S3 Standard

Data returned by S3 Select

Data scanned by S3 Select

▼ Show calculations

## ▼ Show calculations

Tiered price for: 100 GB

100 GB x 0.0230000000 USD = 2.30 USD

Total tier cost = 2.3000 USD (S3 Standard storage cost)

10,000 PUT requests for S3 Storage x 0.000005 USD per request = 0.05 USD (S3 Standard PUT requests cost)

10,000 GET requests in a month x 0.0000004 USD per request = 0.004 USD (S3 Standard GET requests cost)

100 GB x 0.0007 USD = 0.07 USD (S3 select returned cost)

100 GB x 0.002 USD = 0.20 USD (S3 select scanned cost)

2.30 USD + 0.004 USD + 0.05 USD + 0.07 USD + 0.20 USD = 2.62 USD (Total S3 Standard Storage, data requests, S3 select cost)

**S3 Standard cost (monthly): 2.62 USD**



## References:

- [1] <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
- [2] <https://www.amazonaws.cn/en/ec2/>
- [3] <https://calculator.aws/#/createCalculator/EC2>
- [4] <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>
- [5] <https://stackshare.io/aws-lambda/alternatives>
- [6] <https://www.sentinelone.com/blog/aws-lambda-use-cases/>
- [7] <https://acloudguru.com/blog/engineering/serverless-showdown-aws-lambda-vs-azure-functions-vs-google-cloud-functions>
- [8] <https://iamondemand.com/blog/aws-lambda-vs-azure-functions-ten-major-differences/>
- [9] <https://www.stackshare.io/stackups/aws-lambda-vs-aws-step-functions>
- [10] <https://wonderproxy.com/blog/amazon-vs-google-translate/>
- [11] <https://aws.amazon.com/translate/details/>
- [12] <https://calculator.aws/#/createCalculator/Translate>
- [13] <https://aws.amazon.com/polly/>
- [14] <https://www.g2.com/products/amazon-polly/competitors/alternatives>
- [15] <https://aws.amazon.com/vpc/>
- [16] <https://aws.amazon.com/s3/>
- [17] <https://aws.amazon.com/s3/features/?nc=sn&loc=2>