Table of Content

Introduction

Task 1

For my project, I selected a social media application built on the MERN stack of technologies. This application comprises of a React front end, an Express.js backend, and a MongoDB database. The project was forked from GitHub. The repository has 154 commits and 264 stars. I have chosen this application because of its three-tier architecture, that includes the front end, the backend, and the database. This structure enables for a seamless cloud migration, aligning with the principles of the AWS Well-Architected Framework. The migration process will be implemented using a minimum of five categories and adhering to at least four pillars of the AWS Well-Architected Framework.

The project has a React front end, providing a dynamic responsive user interface. The backend for the project is on NodeJs and Express.js. The backend server handles data requests and facilitates communication between the react front end and the MongoDB database. MongoDB, is a NoSQL database that is hosted on mongo atlas separate from the cloud AWS Infrastructure.

I have chosen to migrate this application to the cloud is because of the various services offered by Amazon Web Services (AWS). AWS has a lot of services for the various tiers of the web application and it align with the requirements of the chosen MERN-based social media platform.

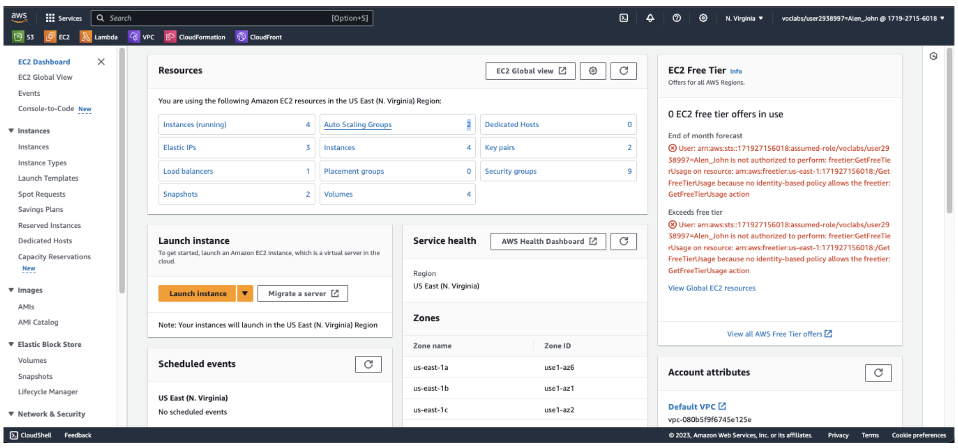
By migrating the application to AWS, I can use the scalable and elastic computing resources services like Amazon EC2 [1], ensuring optimal performance under varying workloads. Amazon S3 will enable me to store raw log files that can be transformed and analysed, while Amazon WAF protects against various application layer attacks. AWS Lambda [2] can be used to trigger events and enhancing the application's responsiveness.

AWS CloudFront, the (CDN) [3] content delivery network can be used to enhance the application's global reach by reducing latency. AWS Elastic Load Balancing ensures the incoming application traffic load is evenly distributed across multiple targets, enhancing fault tolerance and availability. This report provides accurate explanations of the utillized services and outlines the cloud architecture for the entire application.

Services Chosen

Compute

For the compute category for my application I decided to use the **Amazon EC2 instances** [1] to host both the frontend and the backend of my application. The reason i decided to use EC2 was due to the need for scalable, customizable and persistent compute environment. EC2 provided me with the flexibility to configure my servers according to the applications requirements and enabled me to obtain the best performance of my application. Additionally the application is a Social Media Application with a dynamic frontend on react. The front end application has various animations and was frontend heavy. Due to this I chose to use a EC2 for my frontend deployment as well.
(See Figure 1)



Resources

You are using the following Amazon EC2 resources in the US East (N. Virginia) Region:

Instances (running)	4	Auto Scaling Groups	2	Dedicated Hosts	0
Elastic IPs	3	Instances	4	Key pairs	2
Load balancers	1	Placement groups	0	Security groups	9
Snapshots	2	Volumes	4		

Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

Note: Your instances will launch in the US East (N. Virginia) Region

Scheduled events

US East (N. Virginia)
No scheduled events

Service health

Region
US East (N. Virginia)

Zones

Zone name	Zone ID
us-east-1a	use1-az6
us-east-1b	use1-az1
us-east-1c	use1-az2

EC2 Free Tier Info
Offers for all AWS Regions.

0 EC2 free tier offers in use

End of month forecast
User: arn:aws:sts::171927156018:assumed-role/voclabs/user2938997=Alen_John is not authorized to perform: freetier:GetFreeTierUsage on resource: arn:aws:freetier:us-east-1:171927156018:/GetFreeTierUsage because no identity-based policy allows the freetier:GetFreeTierUsage action

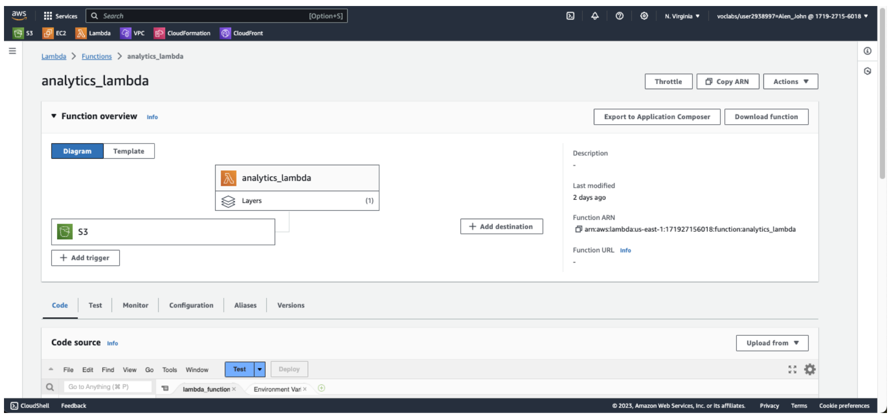
Exceeds free tier
User: arn:aws:sts::171927156018:assumed-role/voclabs/user2938997=Alen_John is not authorized to perform: freetier:GetFreeTierUsage on resource: arn:aws:freetier:us-east-1:171927156018:/GetFreeTierUsage because no identity-based policy allows the freetier:GetFreeTierUsage action

Account attributes

Default VPC
vpc-080b5f9f6745e125e

Figure 1 : EC2 Dashboard

Additionally I also have used serverless compute option provided by AWS, **Lambda** [2] to trigger my analytics pipeline. The serverless option aligned seamlessly with the event-driven nature of the log processing and analytics pipeline. This ensures that the compute resources are only utillized when logs are pushed to the S3 bucket [4]. This approach enhances the cost efficiency as the resources are dynamically allocated and deallocated based on the demand. (See Figure 2)



analytics_lambda

Description
-

Last modified
2 days ago

Function ARN
arn:aws:lambda:us-east-1:171927156018:function:analytics_lambda

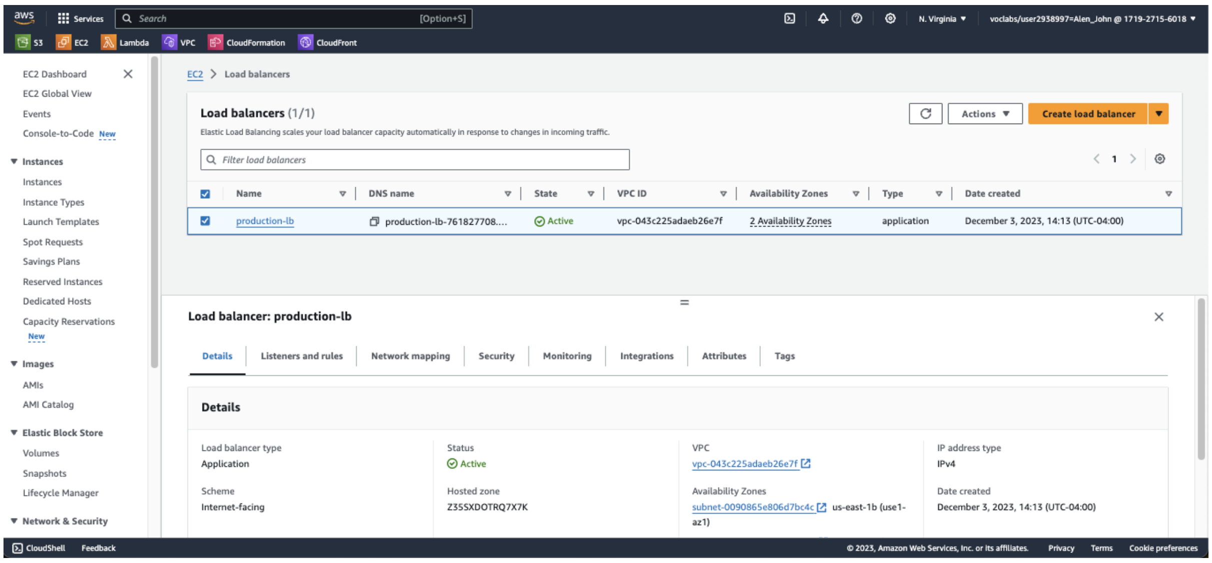
Function URL Info
-

Figure 2 : Analytics Lambda

There were various options considered for the compute category which included options like Amazon S3 for static web hosting, Amazon Elastic Container Service (ECS), Amazon Elastic Kubernetes Service (EKS) and Elastic Beanstalk. The specific demands of a dynamic frontend and a customizable backend led to the selection of EC2s. For the logging and analytics pipeline serverless architecture aligned perfectly with the event-driven requirement of the architecture. By using Amazon EC2 for my architecture i was able to improve on my cost, performance, security and scalability.

Cost: Amazon EC2 instances [1] operate on a pay as you go model. This provides predictability for costs based on the usage. AWS Lambda, incurs charges only when the function is executed thereby optimizing the overall costs. In my implementation I have used the **t2.micro instances** with the On-Demand pricing model for the duration of the migration. This was done to save learner lab credits. In a real world deployment I will use the general purpose instances "m" series. M1.medium instances with a few reserved instances and a few spot instances will be the best for the application [5]. My lambda [2] uses the 128 MB option that is very cost affective. It only has a prove of $0.0000000021per ms. My analytical pipeline required libraries like Matplotlib with were exceeding the 128 MB limit. To avoid increasing my lambda capacity and incurring more cost. I used Lambda Layers to use the Python libraries. I obtained the ARN of a publically deployed layer and just used the layer with the ARN [6].

Performance by setting up a Elastic load balancer and Auto Scaling groups. I directed all the traffic to my application through a elastic load balancer. The load balancer's target group was set up with a auto scaling policy. My application auto scaled its instances when met with high demand. This ensured high performance at all times. The Elastic load balancer ensured that the application load is evenly distributed across all the instances. This ensured that the auto scaling was done only when required by the application. I had saved my AMI images for my backend and the frontend which was used to set up the new instances created by the scaling demands. The use of AMI ensured that the newly created instances were identical to each other. (See Figure 3)



EC2 > Load balancers

Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

Actions | Create load balancer

Filter load balancers

Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
production-lb	production-lb-761827708....	Active	vpc-043c225adaeb26e7f	2 Availability Zones	application	December 3, 2023, 14:13 (UTC-04:00)

Load balancer: production-lb

Details | Listeners and rules | Network mapping | Security | Monitoring | Integrations | Attributes | Tags

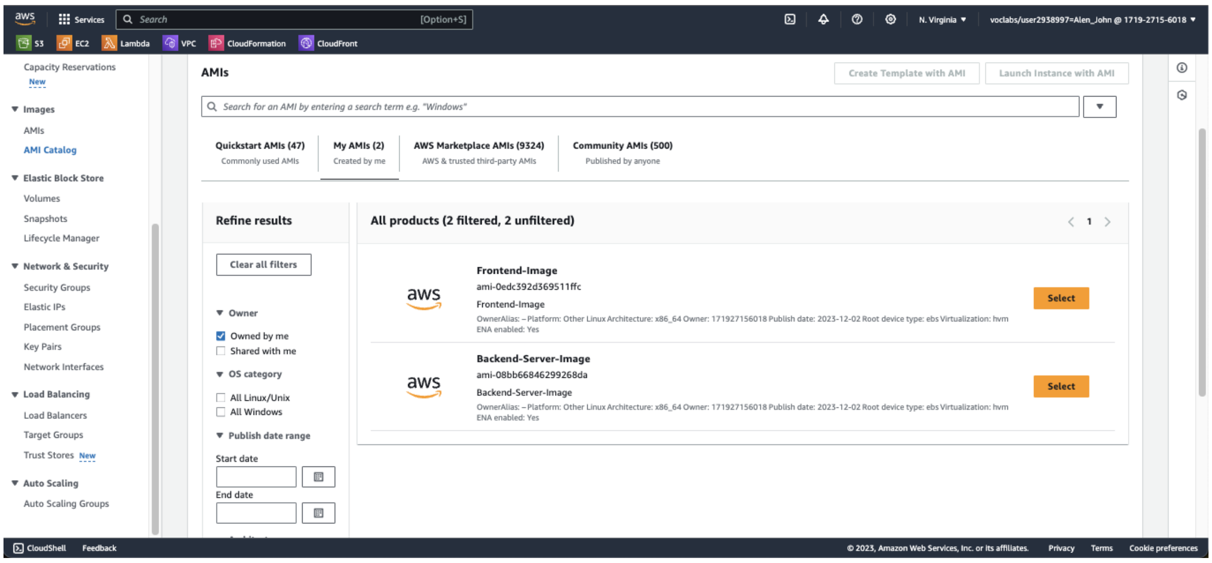
Details

Load balancer type Application	Status Active	VPC vpc-043c225adaeb26e7f	IP address type IPv4
Scheme Internet-facing	Hosted zone Z35SXDOTRQ7X7K	Availability Zones subnet-0090865e806d7bc4c us-east-1b (use1-az1)	Date created December 3, 2023, 14:13 (UTC-04:00)

Figure 3 : Load Balancer

Security was improved by setting up the security groups of thew instances. I ensured that only the ports required by the application we open to be accessed. I also enabled my internal instance firewall to improve my security. I enabled the ubuntu firewall [8] using (sudo ufw enable) and added the ports that were open for traffic. This security was further improved by using approprihate subnets placed in a Virtual Private Cloud [7].

Scalability by setting up auto scaling policy that would add instances in the case of high demand. The scalability was also improved by the usage of AMIs [9] that ensured that the newly created instances were identical to each other. (See Figure 4)



AMIs

Create Template with AMI | Launch Instance with AMI

Search for an AMI by entering a search term e.g. "Windows"

Quickstart AMIs (47) Commonly used AMIs | My AMIs (2) Created by me | AWS Marketplace AMIs (9324) AWS & trusted third-party AMIs | Community AMIs (500) Published by anyone

Refine results

Clear all filters

Owner
- [x] Owned by me
- [ ] Shared with me

OS category
- [ ] All Linux/Unix
- [ ] All Windows

Publish date range

Start date

End date

All products (2 filtered, 2 unfiltered)

Frontend-Image
ami-0edc392d369511ffc
Frontend-Image
OwnerAlias: – Platform: Other Linux Architecture: x86_64 Owner: 171927156018 Publish date: 2023-12-02 Root device type: ebs Virtualization: hvm
ENA enabled: Yes
Select

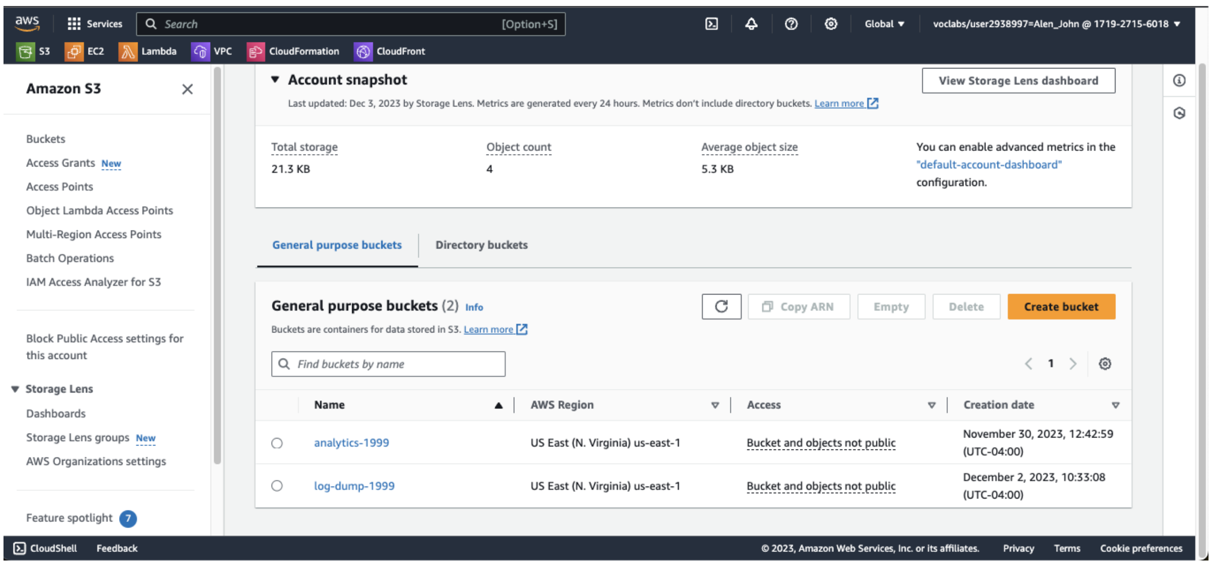
Backend-Server-Image
ami-08bb66846299268da
Backend-Server-Image
OwnerAlias: – Platform: Other Linux Architecture: x86_64 Owner: 171927156018 Publish date: 2023-12-02 Root device type: ebs Virtualization: hvm
ENA enabled: Yes
Select

Figure 4 : My AMIs

In summary, the use of Amazon EC2 [1] and AWS Lambda [2] within the compute category addresses the diverse requirements of hosting a dynamic application, maintaining an efficient backend, and enabling a responsive and cost-effective analytics pipeline.

Storage

For the storage category for my application I choose **Amazon S3** (Simple Storgae Service) [4]. This served as a foundational element for my applications logging and analytical pipeline. My backend server deposits raw logs to into my S3 buckets every 30 minutes via a cron job. The processed logs are stored in a different log bucket. I have selected Amazon S3 due to its durability, saclablity and versatility in handling vast amounts of data. S3 provided an optimal solution for storing log data, offering a reliable, secure, and highly available storage infrastructure. (See Figure 5)



Account snapshot

Last updated: Dec 3, 2023 by Storage Lens. Metrics are generated every 24 hours. Metrics don't include directory buckets. Learn more

Total storage	Object count	Average object size
21.3 KB	4	5.3 KB

You can enable advanced metrics in the "default-account-dashboard" configuration.

General purpose buckets (2) Info

Buckets are containers for data stored in S3. Learn more

Name	AWS Region	Access	Creation date
analytics-1999	US East (N. Virginia) us-east-1	Bucket and objects not public	November 30, 2023, 12:42:59 (UTC-04:00)
log-dump-1999	US East (N. Virginia) us-east-1	Bucket and objects not public	December 2, 2023, 10:33:08 (UTC-04:00)

Figure 5 : My AMIs

I had considered various storage options like Amazon Glacier for long-term archival and Amazon EBS (Elastic Block Store) for block-level storage. However, S3 was the ideal choice due to its seamless integration with other AWS services, notably Lambda [2], that helped in my applications analytical pipeline.

S3 was cost effective in storing real time log data. I used a S3 Standard bucket for the current application architecture. In the real world deployment of my architecture I will be using the S3 Standard-IA (Infrequent access) bucket. S3 Standard-IA [10] is for data that is accessed less frequently, but requires rapid access when needed. My application logs are stored every 30 minuted and S3 Standard IA will be the most cost effective solution for my application. S3 Standard-IA offers the high durability, high throughput, and low latency of S3 Standard, with a low per GB storage price and per GB retrieval charge.

Amazon S3 also has the best performance it automatically scales to high request rates. For example, your application can achieve at least 3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second per partitioned Amazon S3 prefix. [11]

S3 also comes with the best security mesures by default, S3 encrypts all the objects stored in it by default. S3 allowed me to block public access to all my objects in he bucket. S3 also maintains compliance programs, such as PCI-DSS, HIPAA/HITECH, and FISMA, to help in meeting regulatory requirements [12]. I also improved the security by implementing strict bucket policies restricting public access to the buckets. Additionally, the application utilized VPC (Virtual Private Cloud) endpoints for S3 [4], improving data security by ensuring that all interactions between the application and S3 remained within the AWS network.

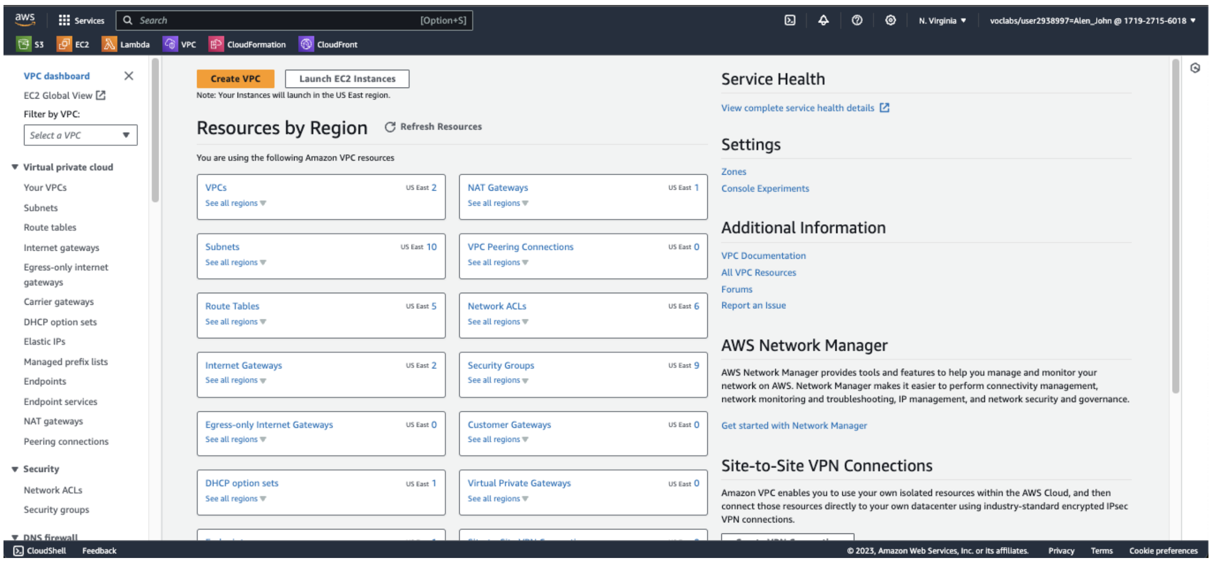
Scalability in storage was a notable feature, with S3 seamlessly handling fluctuations in data volume without compromising performance or availability. The highly scalable nature of S3 facilitated the effortless accommodation of varying data loads and future growth, ensuring the application's storage needs remain met at all times.

In summary, the use of Amazon S3 [4] for storage fulfilled requirements of storing and managing logs and analytics data efficiently. Its reliability, scalability, security, and cost-effectiveness helped in enhancing the overall performance and functionality of the application.

Networking and Content Delivery

For networking, the architecture used Amazon Virtual Private Cloud (VPC) [7] and for content delivery AWS CloudFront this ensured a robust and efficient network infrastructure. The choice of these services was driven by the need for a secure and scalable networking environment and a content delivery solution that enhances user experience.

I had various options to use along with the VPC and the Cloudfront [3] services like AWS route 53 DNS services, but due to learner lab restrictions I couldn’t use these services. I also had options like API Gateway, but these options would require me to modify the existing code infrastructure and make major code changes. (See Figure 6)



Note: Your Instances will launch in the US East region.

Resources by Region

You are using the following Amazon VPC resources

VPCs	US East 2	NAT Gateways	US East 1
Subnets	US East 10	VPC Peering Connections	US East 0
Route Tables	US East 5	Network ACLs	US East 6
Internet Gateways	US East 2	Security Groups	US East 9
Egress-only Internet Gateways	US East 0	Customer Gateways	US East 0
DHCP option sets	US East 1	Virtual Private Gateways	US East 0

Service Health

Settings

Additional Information

AWS Network Manager

AWS Network Manager provides tools and features to help you manage and monitor your network on AWS. Network Manager makes it easier to perform connectivity management, network monitoring and troubleshooting, IP management, and network security and governance.

Site-to-Site VPN Connections

Amazon VPC enables you to use your own isolated resources within the AWS Cloud, and then connect those resources directly to your own datacenter using industry-standard encrypted IPsec VPN connections.

© 2023, Amazon Web Services, Inc. or its affiliates.

Figure 6 : VPC Dashboard

Amazon VPC:

Amazon VPC [7] provided the foundational networking framework, allowing me to create an isolated virtual network in the AWS Cloud. This VPC was established in the US East (N. Virginia) region, consisting of two subnets in us-east-1a (public and private) and two subnets in us-east-1b (public and private). This segmentation into public and private subnets facilitated the isolation of critical resources, with the frontend hosted in the public subnet and the backend in the private subnet.

The decision to use VPC was due to the requirement for a custom and controlled network environment (to enhance security). This isolation provided an added layer of security, preventing unauthorized access to sensitive backend resources. Each subnet was configured to suit its designated role, contributing to a well-organized and secure networking setup.

AWS CloudFront:

To optimize content delivery and enhance end-user performance, I used AWS CloudFront, a content delivery network (CDN) service. CloudFront [3] cached content at edge locations distributed globally, ensuring reduced latency and faster load times for users accessing the application from different parts of the world.

The choice of CloudFront provide users with a seamless and accelerated user experience. By having users interact with the CloudFront link, the CDN helped offload traffic from the application servers, reducing the load on the backend infrastructure. This improved performance and contributed to enhancing the security by adding as a shield between users and the underlying application load balancer.

Cost: Both Amazon VPC and AWS CloudFront offer a cost-effective approach to networking and content delivery. VPC has no additional charge for creating and using an Amazon Virtual Private Cloud (VPC) by itself. It charges for the various aws services used in this VPC. NAT gateways are chargeable (Price per NAT gatewayis $0.045 $/hour) [13]. My architecture used a NAT gateway to direct internet traffic from my private instances to the public internet (through the internet gateway). I also have used a single NAT gateway in one of subnets from a availability zone [13]. This helps in further optimizing costs. Cloudfront does not charge for data transfer from an AWS region to the edge location, it only charges when a transfer occurs from the edge location to the internet. It also has 10,000,000 HTTP or HTTPS Requests per month in its always free tier [14].

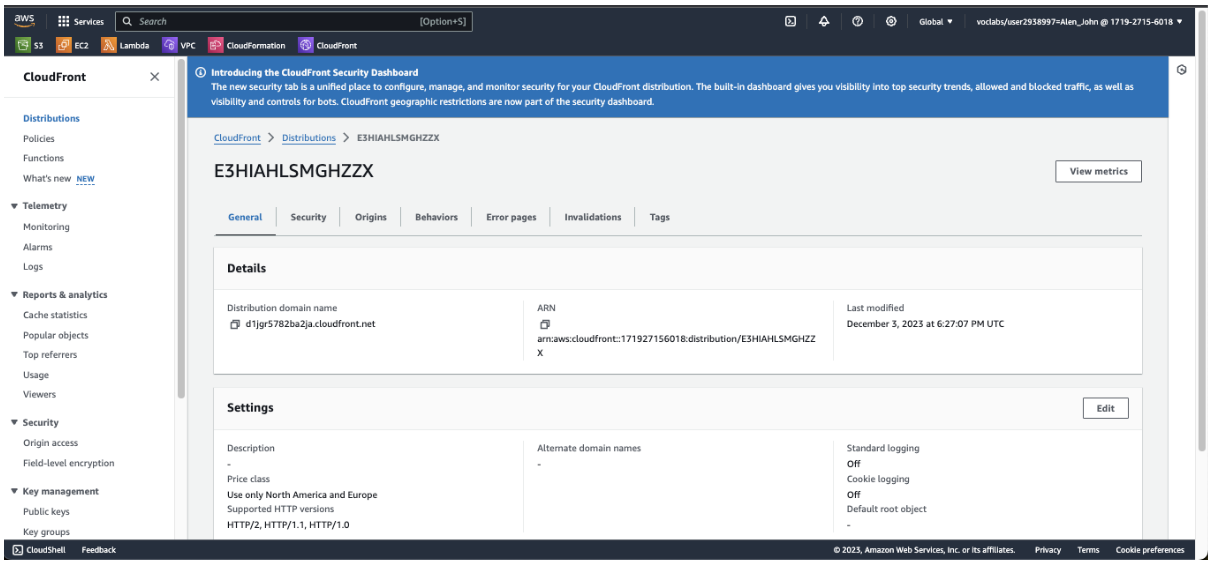
Performance: VPC ensured a reliable and low-latency network environment, while CloudFront's global edge locations significantly improved content delivery performance. The combination of these services contributed to a responsive and efficient user experience. Cloudfront [3] caches static content in amazon edge locations around the world. When a user tries to access the web application the edge location delivers the cached content from edge location closet to the user. This reduces load time and improves the performance. It also offloads load from the application server hence improving on the overall performance of the application.

Security: VPC's isolation of resources into public and private subnets enhanced security by restricting direct access to backend resources. CloudFront acted as a protective layer, distributing content from edge locations rather than the origin servers. I also enabled WAF [15] (web application firewall) to further prevent mallicious request that may compromise the security of the application.

Scalability: VPC provides with numerous option to improve on scalability. The VPC can be extended to all the Availablity zones of a region. AWS also provides options for a multi-availablity zone VPC architecture through organizational units. AWS Cloudfront can help in scaling our architecture around the world. It provides options while setting up to choose between that will enable the caching for all the regions. This will ensure that the data is cached at all the edge locations where the data is accessed, providing a scalable architecture [15].

Security

Security: To ensure high level of security for the application, I utilized various aspect in AWS like, integrating **Network Access Control Lists (NACLs), Security Groups**, **Elastic Load Balancer** [16]**,** (ELB), **VPC endpoints,** and **AWS CloudFront** [3]. For the public subnet NACL, three rules were strategically configured to allow HTTP traffic, MongoDB communication on port 27017, and facilitate SSH access solely from the local machine (my machine). The private subnet NACL had the same configuration as public NACL, permitting HTTP and MongoDB traffic, and SSH access **but restricting** the source to the public instance's IP. (See Figure 7)



Introducing the CloudFront Security Dashboard
The new security tab is a unified place to configure, manage, and monitor security for your CloudFront distribution. The built-in dashboard gives you visibility into top security trends, allowed and blocked traffic, as well as visibility and controls for bots. CloudFront geographic restrictions are now part of the security dashboard.

CloudFront > Distributions > E3HIAHLSMGHZZX

E3HIAHLSMGHZZX

Details

Distribution domain name
d1jgr5782ba2ja.cloudfront.net

ARN
arn:aws:cloudfront::171927156018:distribution/E3HIAHLSMGHZZ
X

Last modified
December 3, 2023 at 6:27:07 PM UTC

Settings

Description
-
Price class
Use only North America and Europe
Supported HTTP versions
HTTP/2, HTTP/1.1, HTTP/1.0

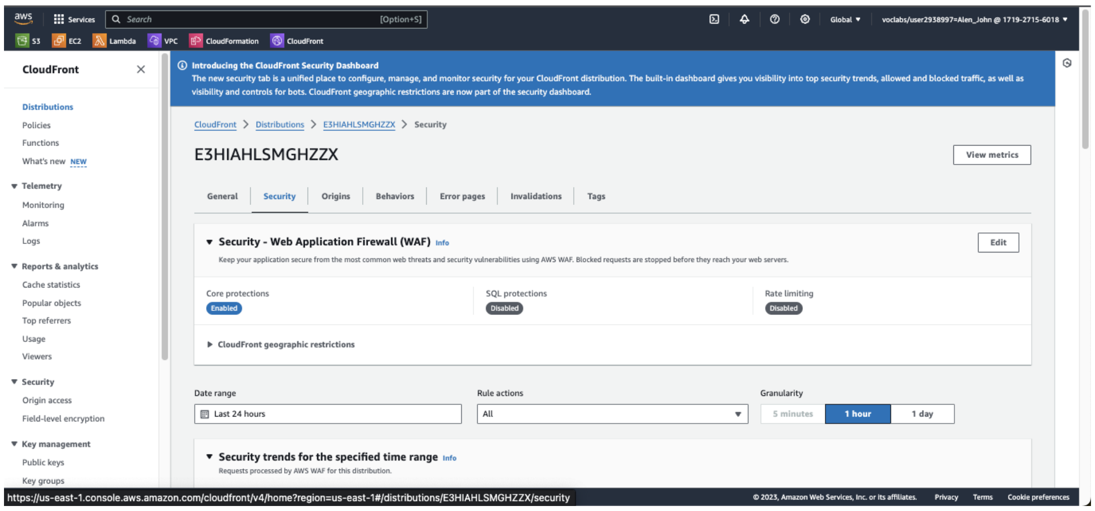
Alternate domain names
-

Standard logging
Off
Cookie logging
Off
Default root object
-

Figure 7 : CloudFront Dashboard

Security Groups were configured to allow only essential ports, restricting unnecessary access. The public and private subnets were used to safeguard critical assets, segregating the application server in the private subnet. An Elastic Load Balancer was utilized to further enhance security, intercepting user requests before reaching the application server directly.

The security measure involved the implementation of AWS CloudFront, this provided a protective shield. Users' interaction with the application was through CloudFront, ensuring a secure and optimized experience. Additionally, the Web Application Firewall (WAF) [15] for CloudFront was implemented to secure against malicious requests. This also helped to get a free SSL certificate, establishing a secure HTTPS connection. (See Figure 8)



Introducing the CloudFront Security Dashboard
The new security tab is a unified place to configure, manage, and monitor security for your CloudFront distribution. The built-in dashboard gives you visibility into top security trends, allowed and blocked traffic, as well as visibility and controls for bots. CloudFront geographic restrictions are now part of the security dashboard.

CloudFront > Distributions > E3HIAHLSMGHZZX > Security

E3HIAHLSMGHZZX

Security - Web Application Firewall (WAF)
Keep your application secure from the most common web threats and security vulnerabilities using AWS WAF. Blocked requests are stopped before they reach your web servers.

Core protections
Enabled

SQL protections
Disabled

Rate limiting
Disabled

CloudFront geographic restrictions

Date range
Last 24 hours

Rule actions
All

Granularity
5 minutes | 1 hour | 1 day

Security trends for the specified time range
Requests processed by AWS WAF for this distribution.

https://us-east-1.console.aws.amazon.com/cloudfront/v4/home?region=us-east-1#/distributions/E3HIAHLSMGHZZX/security

Figure 8 : CloudFront WAF

For my analytical pipeline I specifically used VPC endpoints [18] while invoking my buckets to avoid my request going through the public internet. This ensures that the security is maintained at all times.

The architecture provides a comprehensive secure solution that aligns with AWS best practices, emphasizing least privilege access, network segmentation, and robust protection against potential threats. The integrated security measures collectively contribute to a scalable security framework.

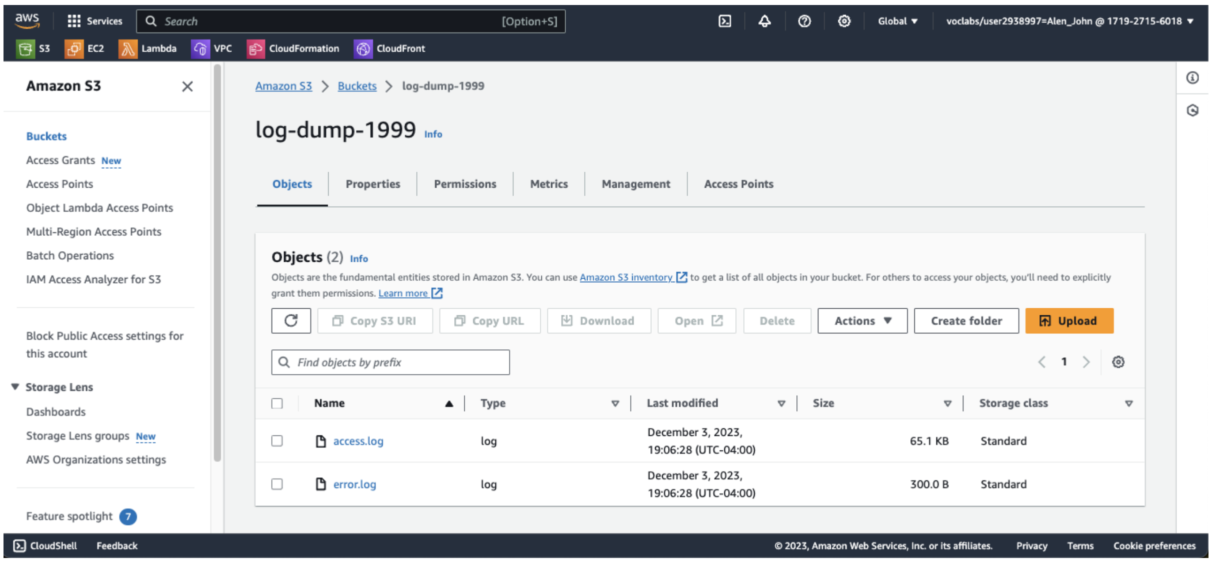
Cost: AWS resources like VPC has no cost associated by it has cost to services used in the VPC. My application uses a NAT Gateway in one of the public subnet. The cost for the gateway comes to about $0.045 per hour. Cloud front prices depend on the no of data transferred. The first First 10TB transfered out to the internet for United States, Mexico, and Canada is $0.085. It also has free tier for requests for requests under a TB [17].

Performance: The performance of the application is improved. Cloudfront [3] caches static content in amazon edge locations around the world. When a user tries to access the web application the edge location delivers the cached content from edge location closet to the user. This reduces load time and improves the performance. It also offloads load from the application server hence improving on the overall performance of the application.

Scalability: VPC provides with numerous option to improve on scalability. The VPC can be extended to all the Availablity zones of a region. AWS also provides options for a multi-availablity zone VPC architecture through organizational units. AWS Cloudfront [3] can help in scaling our architecture around the world. It provides options while setting up to choose between that will enable the caching for all the regions. This will ensure that the data is cached at all the edge locations where the data is accessed, providing a scalable architecture.

Analytics

I had various options to choose from like cloudwatch with S3 [4] and lambda [3] that had a lot of parameters that could be used for analytics for my application's analytics pipeline. But due to learner lab restrictions I coudnt use these services. The analytics pipeline within the application serves as a critical component, providing valuable insights through log processing and analysis. This pipeline is powered by **AWS Lambda** [3] and **S3** [4], offering a serverless and event-driven architecture that aligns seamlessly with the nature of log processing. To generate logs i used **nginx logs** [19] and then i had a cron job that pushed this logs to the s3 bucket every 30 mins. This triggers my AWS Lambda function that processes logs triggered by the event, specifically the recurring dump of raw logs from the backend server to an Amazon S3 bucket. The serverless model ensures efficient resource utilization, as the compute resources are invoked only when logs are deposited in the designated S3 bucket. (See Figure 9)

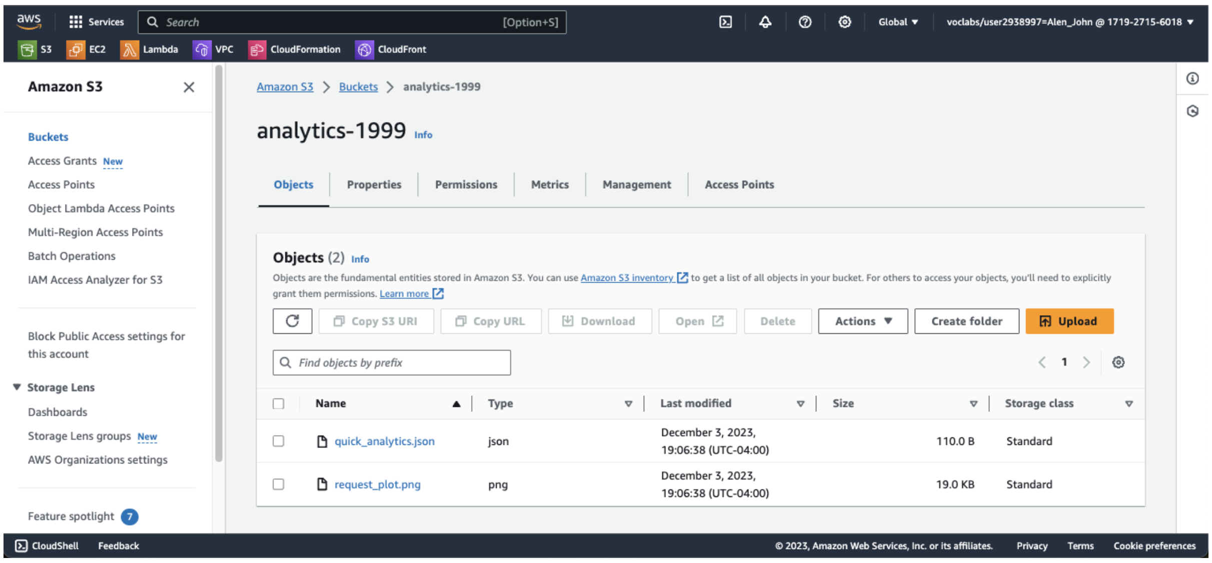


Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
access.log	log	December 3, 2023, 19:06:28 (UTC-04:00)	65.1 KB	Standard
error.log	log	December 3, 2023, 19:06:28 (UTC-04:00)	300.0 B	Standard

Figure 9 : S3 Bucket Log Dump

The AWS lambda generated a json file with all the logs, and a bar plot with the no of requests from each server. This processes analytics is present in the analytics 1999 bucket. (See Figure 10)



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
quick_analytics.json	json	December 3, 2023, 19:06:38 (UTC-04:00)	110.0 B	Standard
request_plot.png	png	December 3, 2023, 19:06:38 (UTC-04:00)	19.0 KB	Standard

Figure 10 : S3 Bucket End Logs

Cost: AWS Lambda [2] provides pay-as-you-go pricing model that aligns with the event driven nature of log processing. This offers very predictability in costs based on usage. My lambda [20] uses the 128 MB option that is very cost affective. It only has a prove of $0.0000000021per ms. My analytical pipeline required libraries like Matplotlib with were exceeding the 128 MB limit. To avoid increasing my lambda capacity and incurring more cost. I used Lambda Layers to use the

Python libraries. I obtained the ARN of a publically deployed layer and just used the layer with the ARN. for the storage of these logs i use a standard S3 bucket that has $0.023 per GB for First 50 TB / Month.

Performance: AWS lambdas [2] have autoscaling enabled. As the function receives more requests, Lambda automatically scales up the number of execution environments to handle these requests. This ensures high performances at all times

Security: AWS Lambda operates within the secure AWS environment, and the architecture ensures that only necessary resources are allocated during function execution. This enhances the overall security of the analytics pipeline. S3 [4] also comes with the best security mesures by default, S3 encrypts all the objects stored in it by default. S3 allowed me to block public access to all my objects in he bucket. S3 also maintains compliance programs, such as PCI-DSS, HIPAA/HITECH, and FISMA, to help in meeting regulatory requirements. [21]

Scalability: AWS Lambda has auto-scaling enabled and it allows for seamless scalability based on demand. This ensures scalability of the application.

Architecture Implemented

The following figure shows my architecture on aws cloud. (See figure 11)

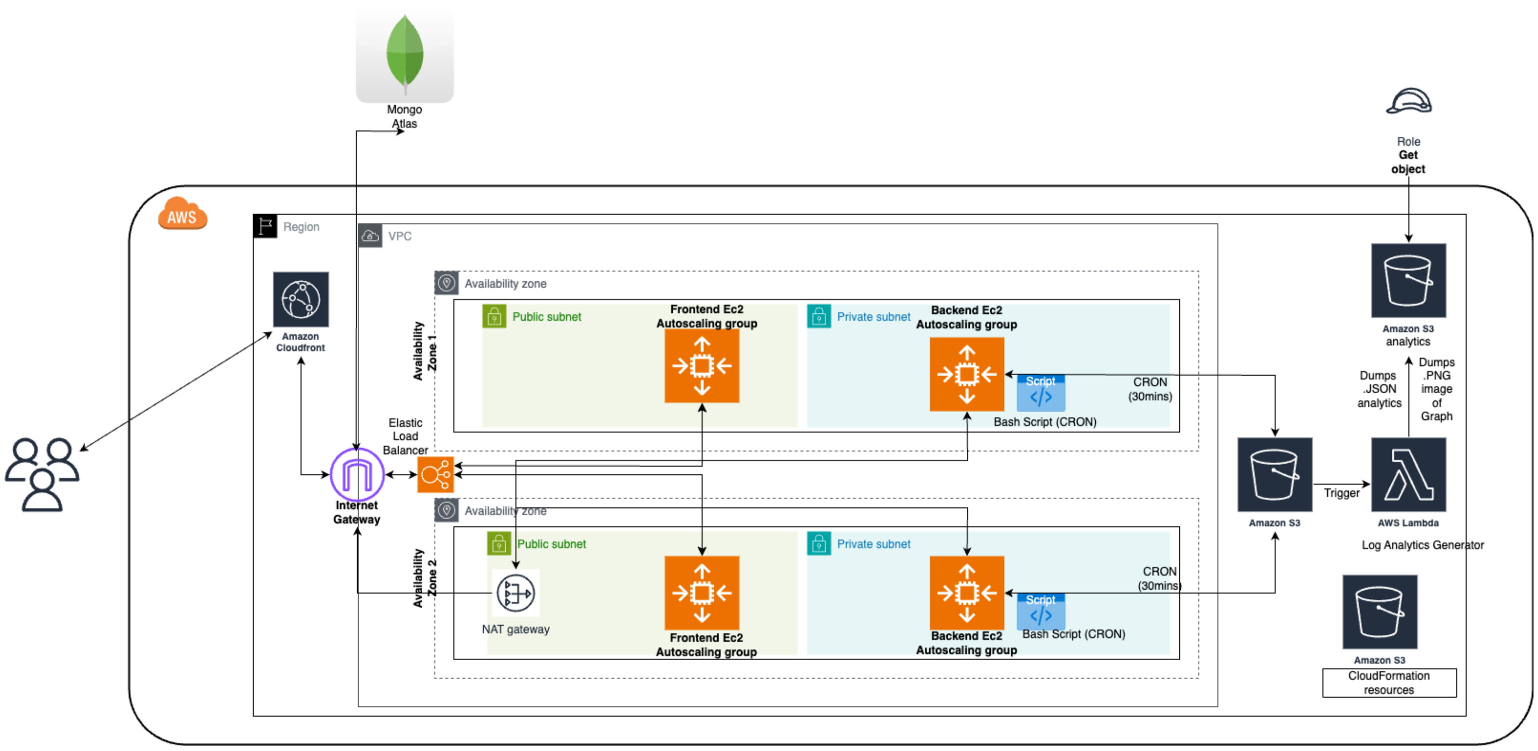


Figure 11 : Architecture Diagram

My architecture having a Virtual Private Cloud (VPC) [7] that encompasses four subnets, divided into private and public subnets across two availability zones within the US East 1 region. The design features a front-end Auto Scaling EC2 [1] group positioned in the public subnet and a corresponding back-end Auto Scaling EC2 group in the private subnet. This arrangement is replicated in two availability zones, with internet traffic facilitated through the Internet Gateway integrated into the VPC.

To manage internet access for the private subnet, a NAT Gateway is strategically placed in Availability Zone 2. This gateway efficiently routes all traffic from the private subnet to the public internet, specifically handling database requests involving GET, PUT, and POST operations. Enhancing the architecture's scalability, an Elastic Load Balancer (ELB) evenly distributes the application load across both availability zones. The ELB's target group is configured with autoscaling enabled, ensuring dynamic scaling in response to fluctuating traffic demands.
Furthermore, my architecture incorporates two Amazon S3 [4] buckets dedicated to the log pipeline. Each of the backend instances contributes to this pipeline, facilitating efficient log management and analysis.

Working of the Architecture

When a user requests the web application, they're first directed to the CloudFront [3] edge location. This location holds cached data. If the CloudFront location doesn't have the needed data, the request goes to the Elastic Load Balancer [16]. This balancer is located behind the Internet Gateway in my Virtual Private Cloud [7]. The load balancer spreads the traffic evenly across availability zones based on their compute resource usage. The EC2 instances then communicate with the backend EC2, fetching user data from the MongoDB Atlas database, which is on the internet. This communication is facilitated using the NAT Gateway in Availability Zone 2.

Working of the Log/Analytical Pipeline

All my servers use NGINX server configuration and they produce logs. These logs are stored in the local storage of these instances. I've developed a bash script utilizing AWS CLI, that executes every 30 minutes via a cron job. This script uploads raw NGINX logs [19] to an S3 bucket. The S3 bucket triggers a Lambda function, scripted in Python and uses various Python libraries for data analytics. This Lambda cleans and processes the data, converting crucial parameters into a JSON file and generates a bar graph. The analytics JSON file and the Graph are then stored in a new bucket for further viewing or operational use.

AWS Well Architected Framework

For my application I have tried to follow the AWS well architected framework principles to ensure that my architecture is scalable secure and a well-designed infrastructure the following are some of the pillars that my architecture aligns to

1. **Operational Excellence:** My architecture has implemented a robust logging strategy that utilizes the nginx logs generated by the back-end server. These logs are sent to an S3 bucket, which triggers a Lambda function for data cleaning, processing, and analysis. This approach helps me monitor and troubleshoot my application in case of errors.
   In addition, I have automated various tasks in my application. For example, my application automatically increases the number of instances present if the load increases. My application is attached to a load balancer, which ensures that each target group has an equal load. I have utilized AMI images to automate the setting up and deployment of instances. I have also configured user data to run these instances.

2. **Security:** For my architecture I have ensured that security place a vital role. I have created my resources in a custom VPC and have created various subnets. My private subnets consist of my critical resources, such as my back-end service, whereas my public subnet has my Bastion host and my front-end instances. I have also ensured that I have configured the network access control list and the security groups, allowing only the rules or ports that require subnet access. My private subnet is also secure and does not have direct internet access. I have used a NAT gateway to provide access.
   I have also ensured that I use VPC endpoints for my S3 operations to ensure that the traffic is routed from the internal Amazon network. All the data that is stored inside my S3 is also encrypted, and the rest of my data in transit is also encrypted. I have also ensured that I have a Bastion host for my private subnet instances to regularly maintain and update them with the latest security patches.

3. **Reliability:** My architecture is highly available this is achieved by using a multi-availability zone deployment and elastic load balancers alongside auto-scaling groups. This ensures high availability and fault tolerance, even if one availability zone becomes unavailable. The elastic load balancers can use saved AMIs to regenerate healthy instances and provide high availability
   In addition, I have provided caching solutions by using Amazon's content delivery network, CloudFront. This caching solution enhances the performance and reliability of

my application by reducing latency and offloading traffic from my back end.

4. **Performance Efficiency:** My application utilizes caching and content delivery by implementing Amazon CloudFront, a content delivery network (CDN). This has improved the overall performance of my application by caching static content closer to end-users. My application also has autoscaling enabled, which ensures high performance at all times based on the application load and traffic.

5. **Cost Optimization:** My application has auto-scaling groups and load balancing set up. This ensures that instances are scaled only when the traffic or network load is high. This approach ensures that my costs are optimized. I have also implemented an elastic load balancer that ensures that both target groups are evenly distributed to ensure optimized and cost-effective compute.

   I am also using AWS Lambda for my log and analytics pipeline. AWS Lambda is a serverless function and is only charged when triggered by an event. This helps me optimize my costs as I am only paying for the computer time used during the execution of my function.

6. **Sustainability:** My application is sustainable as I use resources only when necessary. My EC2 instances are configured for auto-scaling and they are only scaled when the network traffic is high. I don’t have warm instances running all the time for high traffic, thereby saving energy utilization. I have also utilized AWS Lambda functions for my log pipeline because my log pipeline only has one execution every 30 minutes. This ensures that I have saved energy, thereby being more sustainable and green.

Deployed Application Screenshot and Details

Name	Link
Github Repo	https://github.com/jm-shi/mern-social-network
Deployed Link	https://d1jgr5782ba2ja.cloudfront.net/login

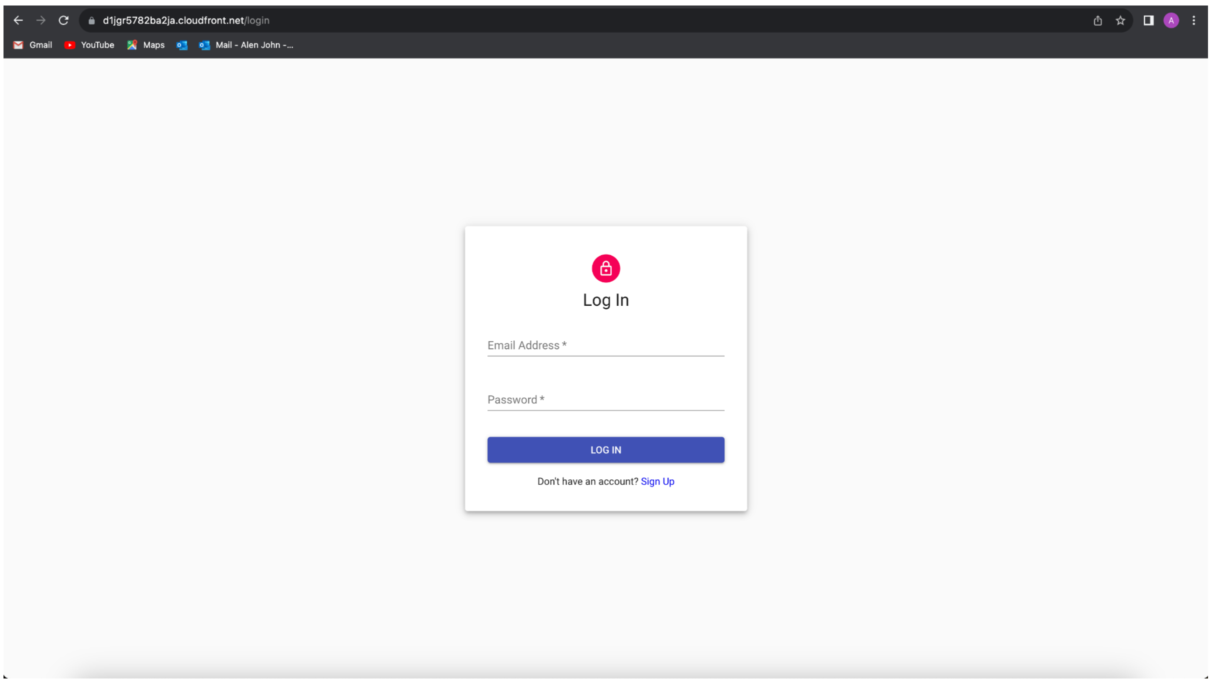


Figure 12 : Deployed Application

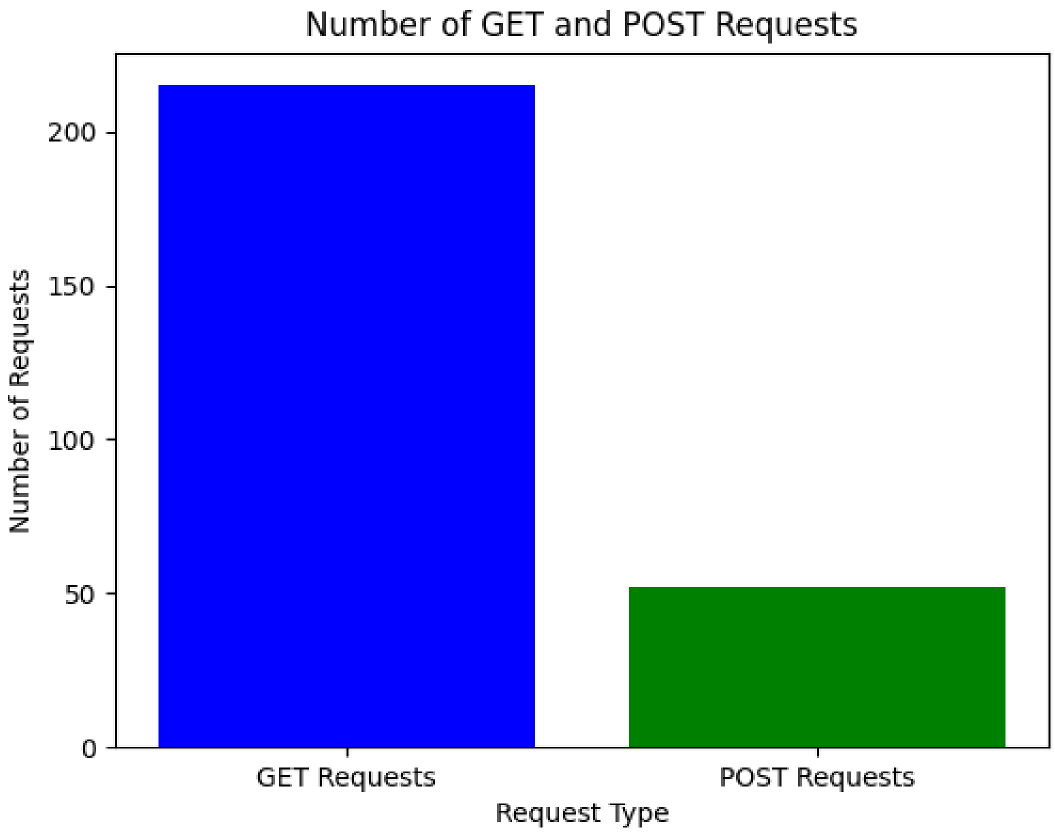


Figure 13 : Generated Bar Plot showing the no of request on server

References

[1] “Amazon Elastic Compute Cloud (Amazon EC2) - Amazon Web Services.” [Online]. Available: https://docs.aws.amazon.com/ec2/. [Accessed: 04-Dec-2023].

[2] “AWS Lambda - Amazon Web Services.” [Online]. Available: https://docs.aws.amazon.com/lambda/. [Accessed: 04-Dec-2023].

[3] “Amazon CloudFront - Amazon Web Services.” [Online]. Available: https://docs.aws.amazon.com/cloudfront/. [Accessed: 04-Dec-2023].

[4] “Amazon Simple Storage Service (S3) - Amazon Web Services.” [Online]. Available: https://docs.aws.amazon.com/s3/. [Accessed: 04-Dec-2023].

[5]“General Purpose Instances - Amazon Elastic Compute Cloud.” [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/general-purpose-instances.html. [Accessed: 04-Dec-2023].

[6] “Klayers - List of ARNs.” [Online]. Available: https://github.com/keithrozario/Klayers#list-of-arns. [Accessed: 04-Dec-2023].

[7] “Amazon Virtual Private Cloud - Amazon Web Services.” [Online]. Available: https://docs.aws.amazon.com/vpc/. [Accessed: 04-Dec-2023].

[8] “Uncomplicated Firewall - Ubuntu Wiki.” [Online]. Available: https://wiki.ubuntu.com/UncomplicatedFirewall. [Accessed: 04-Dec-2023].

[9] “Amazon Machine Images (AMI) - Amazon Elastic Compute Cloud.” [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html. [Accessed: 04-Dec-2023].

[10] “Amazon S3 Storage Classes - Amazon Simple Storage Service.” [Online]. Available: https://aws.amazon.com/s3/storage-classes/. [Accessed: 04-Dec-2023].

[11] “Optimizing Performance for Amazon S3 - Amazon Simple Storage Service.” [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/optimizing-performance.html. [Accessed: 04-Dec-2023].

[12] “Amazon S3 Security - Amazon Simple Storage Service.” [Online]. Available: https://aws.amazon.com/s3/security/. [Accessed: 04-Dec-2023].

[13] “Amazon VPC Pricing - Amazon Web Services.” [Online]. Available: https://aws.amazon.com/vpc/pricing/. [Accessed: 04-Dec-2023].

[14] “Amazon CloudFront Pricing - Amazon Web Services.” [Online].
Available: https://aws.amazon.com/cloudfront/pricing/. [Accessed: 04-Dec-2023].

[15] “CloudFront Features -
AWS WAF, AWS Firewall Manager, and AWS Shield Advanced.” [Online].
Available:https://docs.aws.amazon.com/waf/latest/developerguide/cloudfront-features.html.
[Accessed: 04-Dec-2023].

[16] “Elastic Load Balancing - Amazon Web Services.” [Online].
Available: https://aws.amazon.com/elasticloadbalancing/. [Accessed: 04-Dec-2023].

[17] “Amazon CloudFront Pricing - Amazon Web Services.” [Online].
Available: https://aws.amazon.com/cloudfront/pricing/. [Accessed: 04-Dec-2023].

[18] “What are VPC Endpoints? - AWS PrivateLink.” [Online].
Available:
https://docs.aws.amazon.com/whitepapers/latest/aws-privatelink/what-are-vpc-endpoints.html
. [Accessed: 04-Dec-2023].

[19] “Logging and Monitoring - NGINX.” [Online].
Available: https://docs.nginx.com/nginx/admin-guide/monitoring/logging/. [Accessed: 04-Dec-2023].

[20] “AWS Lambda Pricing - Amazon Web Services.” [Online].
Available: https://aws.amazon.com/lambda/pricing/. [Accessed: 04-Dec-2023].

[21] “Amazon S3 Security - Amazon Simple Storage Service.” [Online].
Available: https://aws.amazon.com/s3/security/. [Accessed: 04-Dec-2023].