ENPM665: FINAL

Submitted By: -
Ratan Gupta
rgupt21@umd.edu
UID: 118195773

COBRA KAI

STRIKE FIRST – STRIKE HARD – NO MERCY

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

This document serves as a technical plan on moving the Cobra Kai application which is currently hosted on premises to the cloud. Previously, we went over the recommendations for migrating to the cloud and talked about various services AWS offers that could potentially serve as building blocks for the Cobra Kai cloud architecture. In this document, we will go over implementing a few of those services and discuss which configurations would serve Cobra Kai infrastructure the best.

AWS services will be configured in a security-oriented manner and focus will be more on implementing the security services and features AWS offers.

The document will first address building the basic cloud architecture for Cobra Kai. A guide on how to go about creating a VPC, public and private subnets, routing tables, sample security groups, launching EC2 instances, and securely configuring databases will be provided.

Once the base is complete, the document will address all the issues with Cobra Kai's current web architecture by providing a guide on how to go about implementing recommendations.

2. PROPOSED ARCHITECTURE

For context, the proposed cloud architecture is provided in this section.

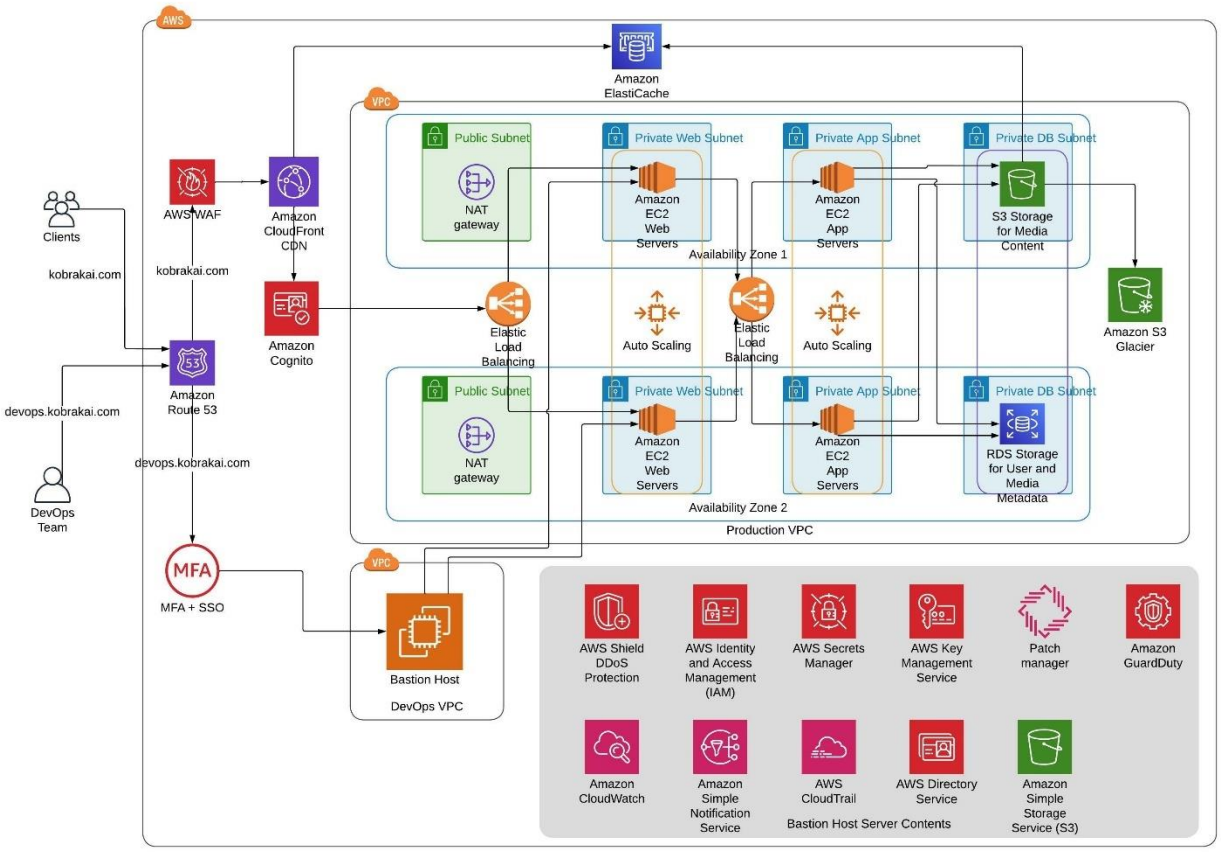

Figure 1: Proposed Cloud Architecture

3. RECOMMENDATIONS

3.1. Building A Basic Architecture

CREATING A VPC

The first thing to do when migrating an on-premises web application to the cloud is to create an AWS Virtual Private Cloud (VPC). According to the approved cloud architecture, within this VPC two (2) availability zones with the public and private subnets exist.

A VPC can be created from the AWS console and configured as per the needs of the organization. For Cobra Kai's purposes, a VPC with the following configuration can be used –

1. IPv4 CIDR block: 129.2.0.0/16
2. Tenancy: Default
3. Number of Availability Zones: 3 – In the architecture we are using 2 Availability Zones and for our purposes that should be enough. However, a third Availability Zone can be kept for future use.
4. Number of Public Subnets: 3 – The NAT Gateways will reside in the public subnets and route traffic to web and application servers.
5. Number of Private Subnets: 6 – The web and application servers will reside in the private subnets.
6. NAT Gateways: 1 per Availability Zone
7. VPC Endpoints: S3 Gateway – This is done to enable requests from the VPC to directly access the SE buckets. This configuration helps in reducing charges that may come our way if we use NAT Gateway for every request.

The following image illustrates how a VPC will be created using AWS console –

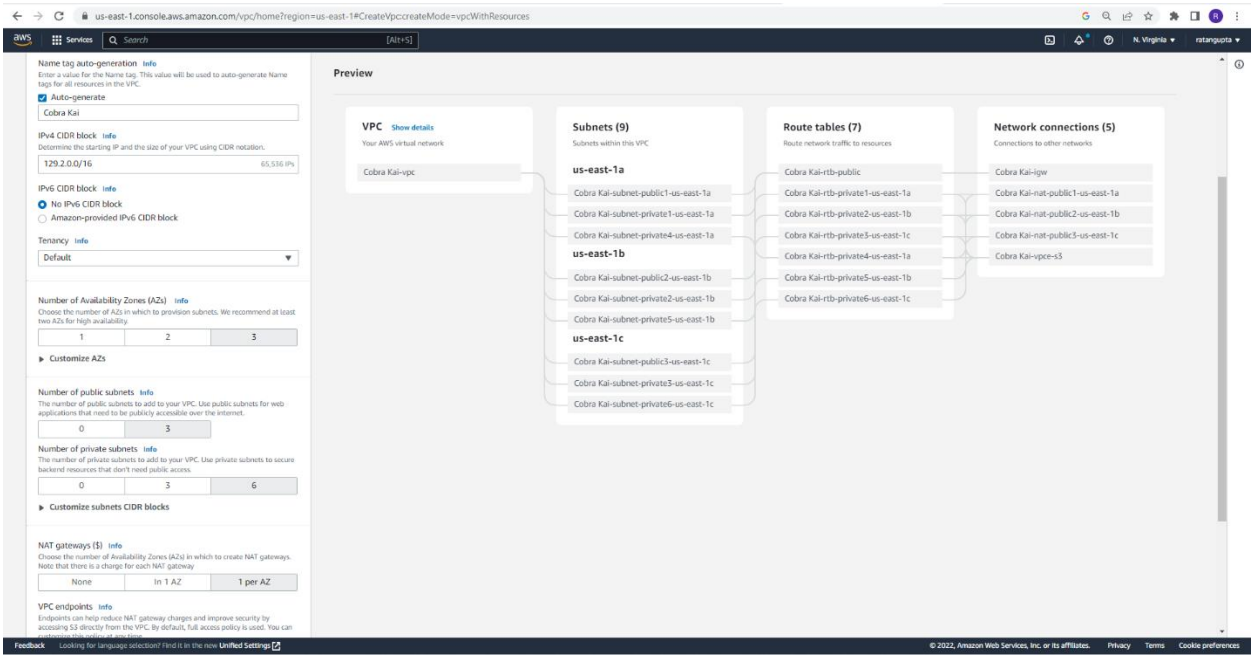

Figure 2: VPC Setup

SUBNETS

The subnets should be up and running at this point.

The routing tables of each subnet can be viewed as follows –

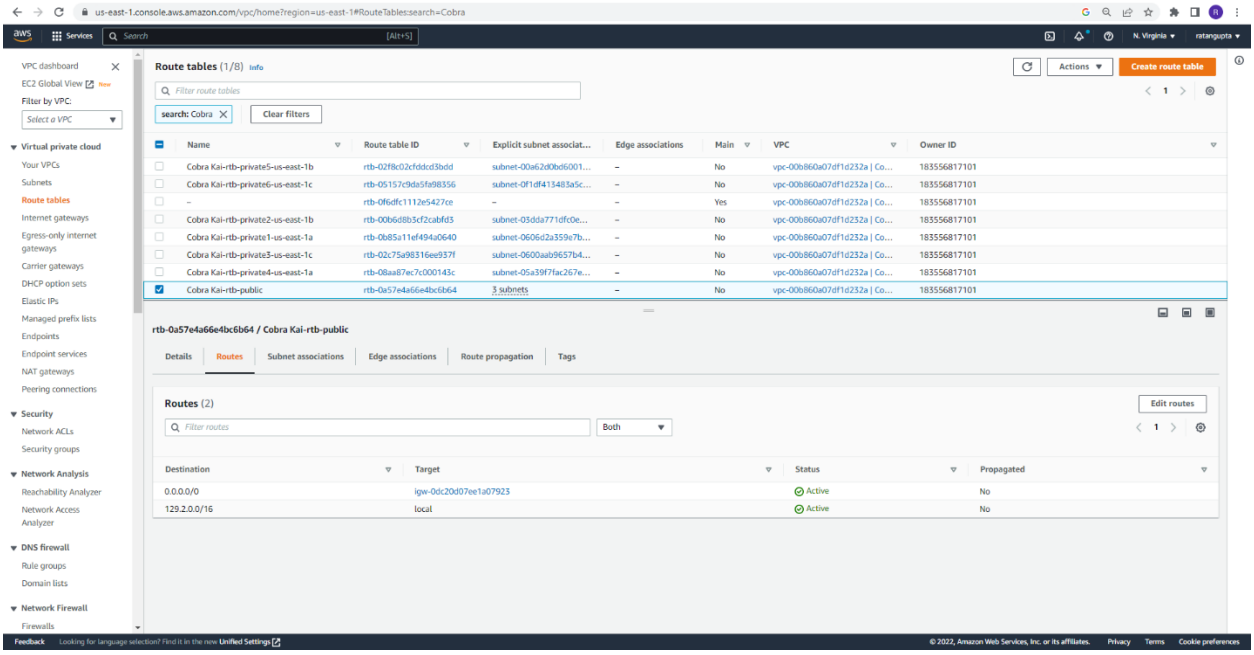

Figure 3: Public Subnets Directly Connected to the Internet

The public subnets should be directly connected to the Internet, and the private subnets should be connected to the Internet through the NAT Gateway. All the private subnets will have an S3 Gateway endpoint attached to them.

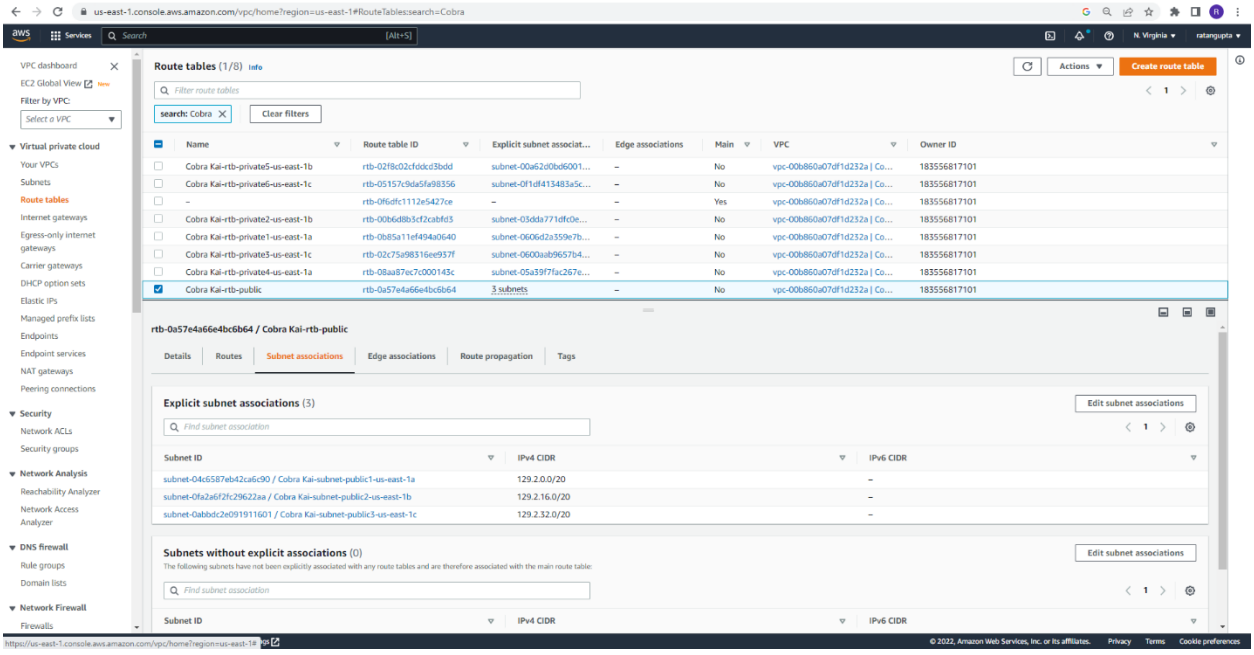

Figure 4: Public Subnet Association

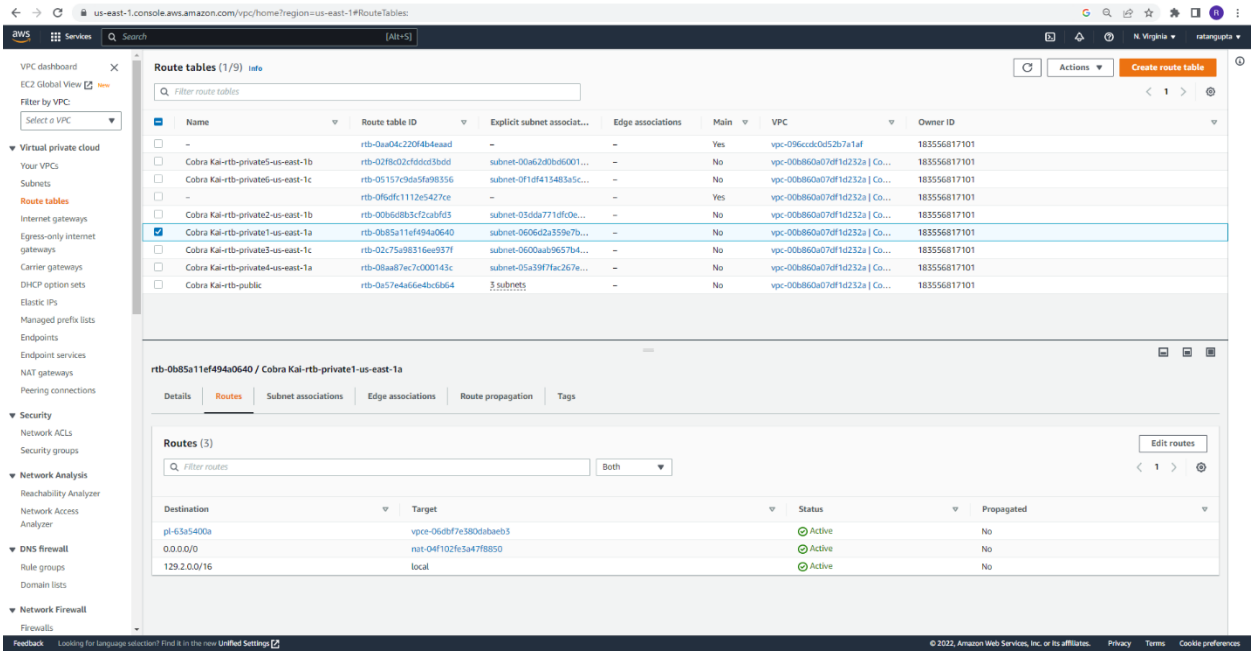

Figure 5: Private Subnet Routes

SAMPLE SECURITY GROUPS

It is recommended to create a security group for port restriction. In our case we have restricted all the ports other than port 443 (HTTPS) for inbound requests. The outbound requests must remain open to access the database and other services.

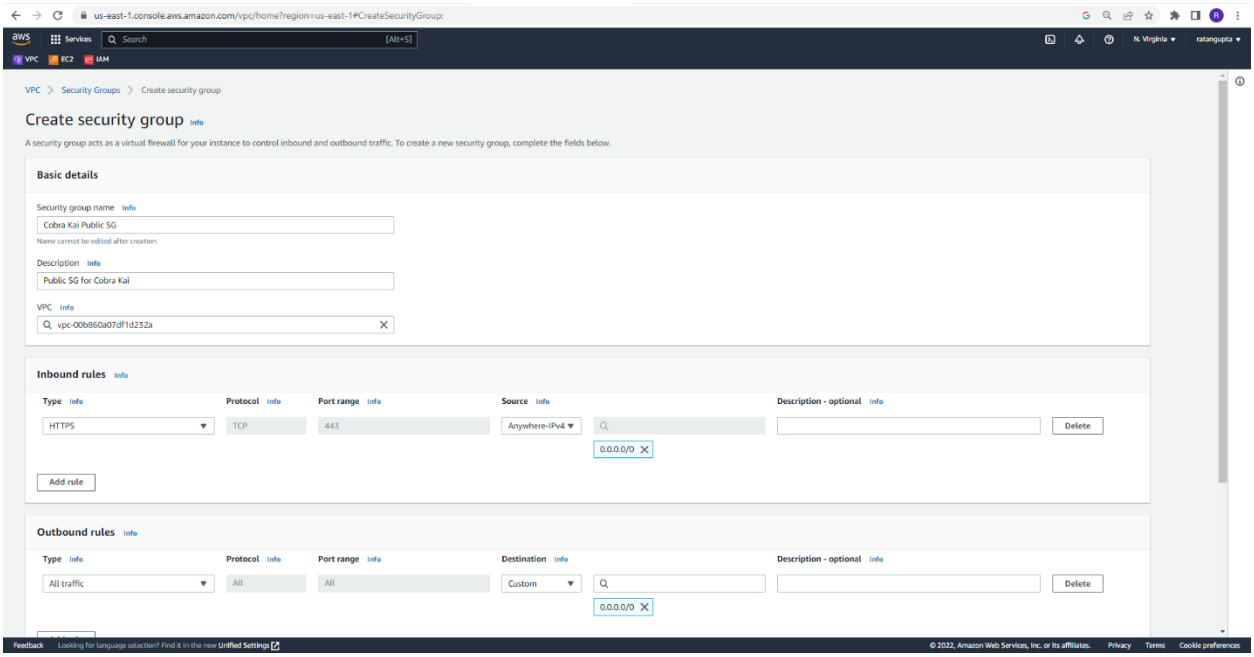

Figure 6: Sample Security Group

When the application goes live, we must make sure that the application runs on port 443 because it is more secure than port 80. The certificates for the same can be created when we configure Route 53.

LAUNCHING AN EC2 INSTANCE

The web and app servers are going to run in the private subnets inside the VPC. We will use EC2 instances to host web and app servers. An EC2 instance can be created as illustrated below –

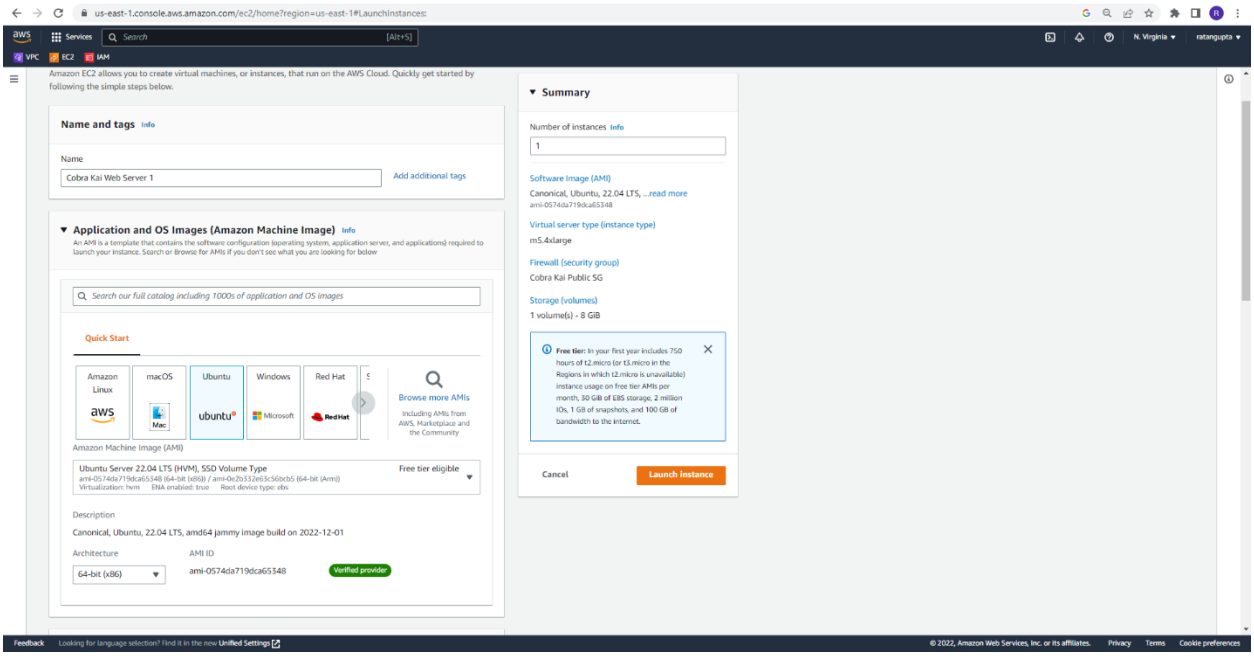

Figure 7: Launching an EC2 Instance

AWS recommends using m series instances for production workloads. My recommendation is to use m5.4xlarge instances. Using this instance ensures a balance between CPU and RAM resources. 4xlarge instances offer 16 CPUs and 64GB RAM, therefore it is advised to have one (1) instance running. We can scale up as and when required. When launching an instance, make sure to select the Security Group that was created by us and not the Default Security Group.

AUTO SCALING AND LOAD BALANCERS

To scale up or down dynamically, create an auto-scaling and run the EC2 instances in the auto-scaling group. We can start with 1 instance of type m5.4xlarge deployed and then set upper and lower limits after analyzing the amount of traffic received on the application.

The load balancers we use will perform health checks on application ports every 3-5 mins to see if they are responsive. If any application becomes unresponsive, the auto-scaling group will shut the instance down and spin up a new one to replace it.

CREATING AN S3 BUCKET

S3 buckets will be used to store all on-demand media content such as Cobra Kai's videos.

The recommended way to configure an S3 bucket securely is as follows –

1. Create a bucket using the AWS console.
2. Select the region where the bucket would exist.
3. Disable ACLs while setting up the bucket to let the access remain with the admin who is creating the bucket. Policies and endpoints can be created later to determine who gets access to the bucket and how.
4. Block all public access to the bucket. Secure channels must be defined for requests to reach S3 buckets.
5. Enable bucket versioning to keep multiple versions of the same object in storage. This prevents an older version of the object from being overwritten.
6. Use Amazon Key Management Service (KMS) to encrypt objects in storage.
7. After creating the bucket, logging and event notifications can be set up by the admin.

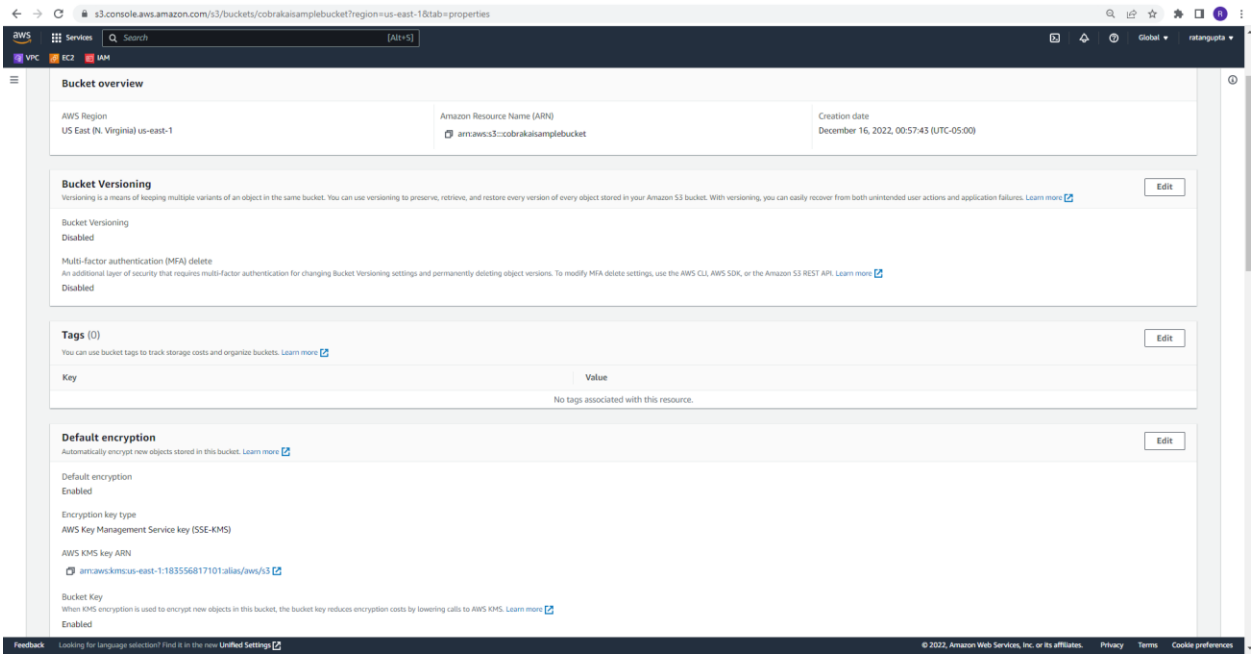

Figure 8: S3 Bucket Versioning and Encryption

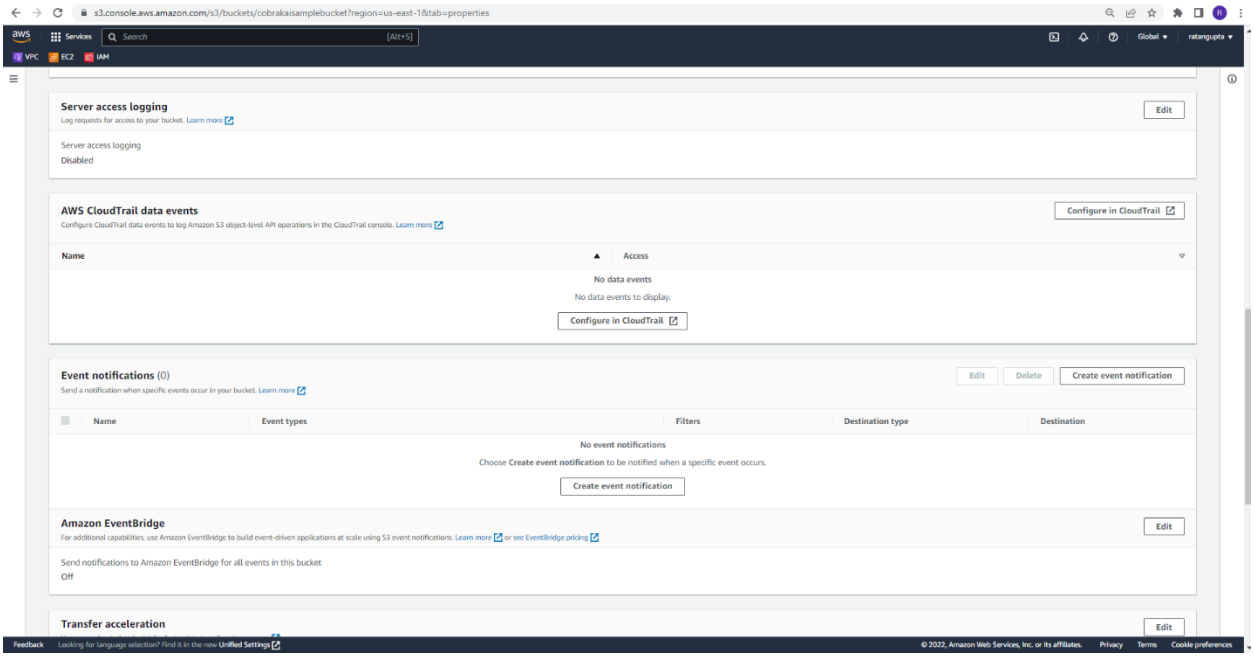

Figure 9: Logging and Event Notifications

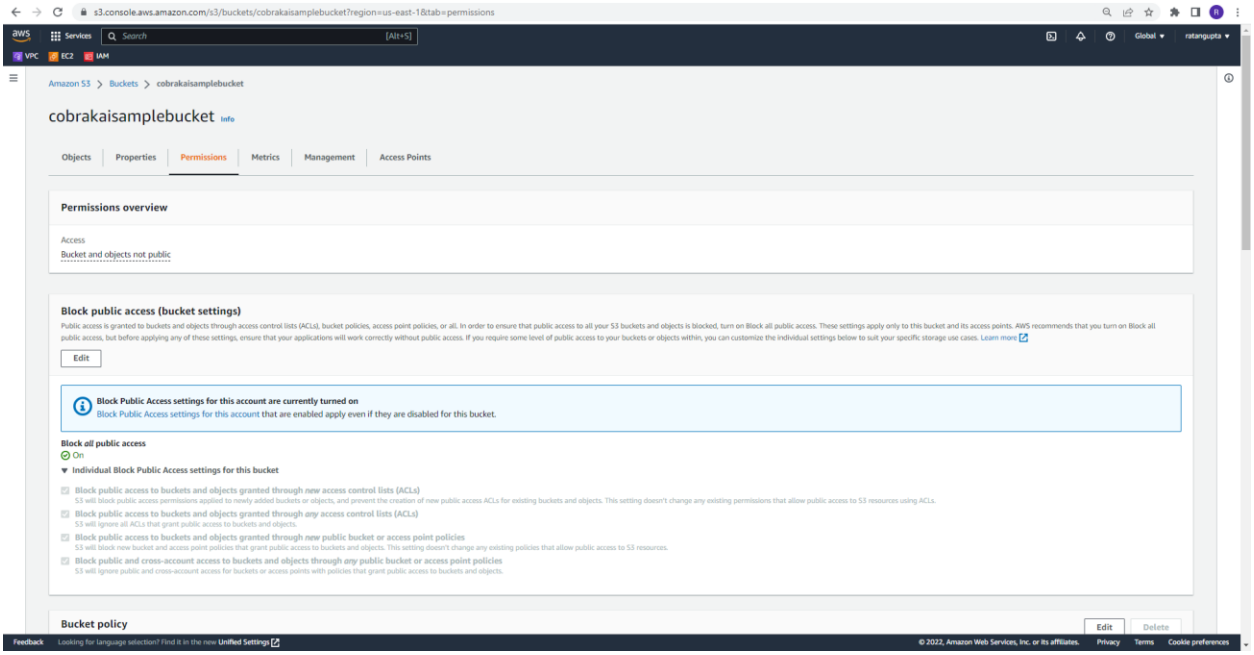

Figure 10: Public Access Blocked

CREATING AN RDS DATABASE

Amazon Relational Database Service (RDS) will be used to store user data and on-demand media metadata.

The recommended way to configure an RDS securely is as follows –

1. Recommended engine to use is MySQL. It is the most popular open-source RDBMS and offers great support for using a relational database service.
2. Use the production template with multiple DB instances (creates a standby DB instance) for backup and high availability.
3. Set up master credentials for the database. The master user can define permissions, objects, and users.
4. The DB instance class can be configured to use 'db.m5d.xlarge'. This configuration offers a good balance between processing power and storage.
5. Enable autoscaling in case the predefined threshold is reached.
6. Connect to the Cobra Kai VPC previously created, use the Cobra Kai security group, and disable public access.
7. Set up database authentication to use 'Password and IAM database authentication'. This is a way to implement 2FA (Two Factor Authentication).
8. Encrypt the database using the encryption key managed by AWS KMS.
9. Enable automated backups with maximum retention time and set a maintenance window.
10. Lastly, enable deletion protection to prevent the accidental deletion of the database.

3.2. Patching Strategy

Configuring a patching strategy can be done in several ways. One recommended way is to create a Patch Group and connect it to all the EC2 Instances.

CREATE A SSM ROLE FOR EC2 INSTANCE

The first step is to have an IAM role in place. Create a role with the following permission – AmazonEC2RoleforSSM. This allows EC2 instances to call AWS services on our behalf.

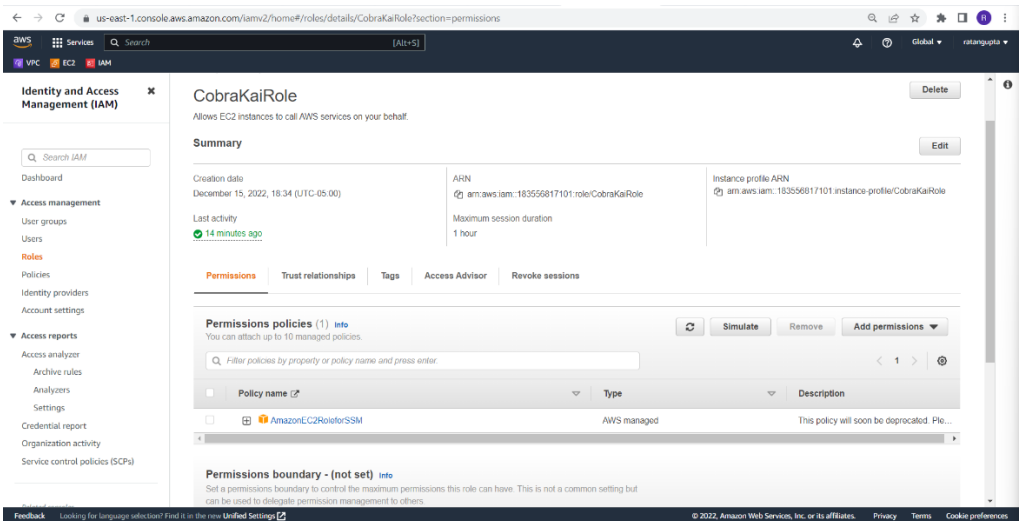

Figure 11: Sample IAM Role

CREATE A PATCH GROUP

Secondly, create a Patch Group by going to Systems Manager -> Patch Manager -> View Predefined Patch Baselines. Click on the baseline image that best suits our need and select the action to modify its patch groups.

Once the patch group is created it will look something like this –

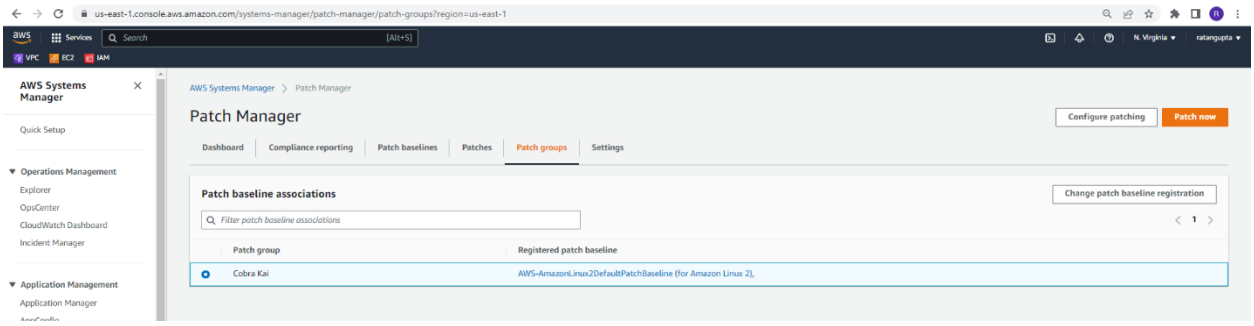

Figure 12: Sample Patch Group

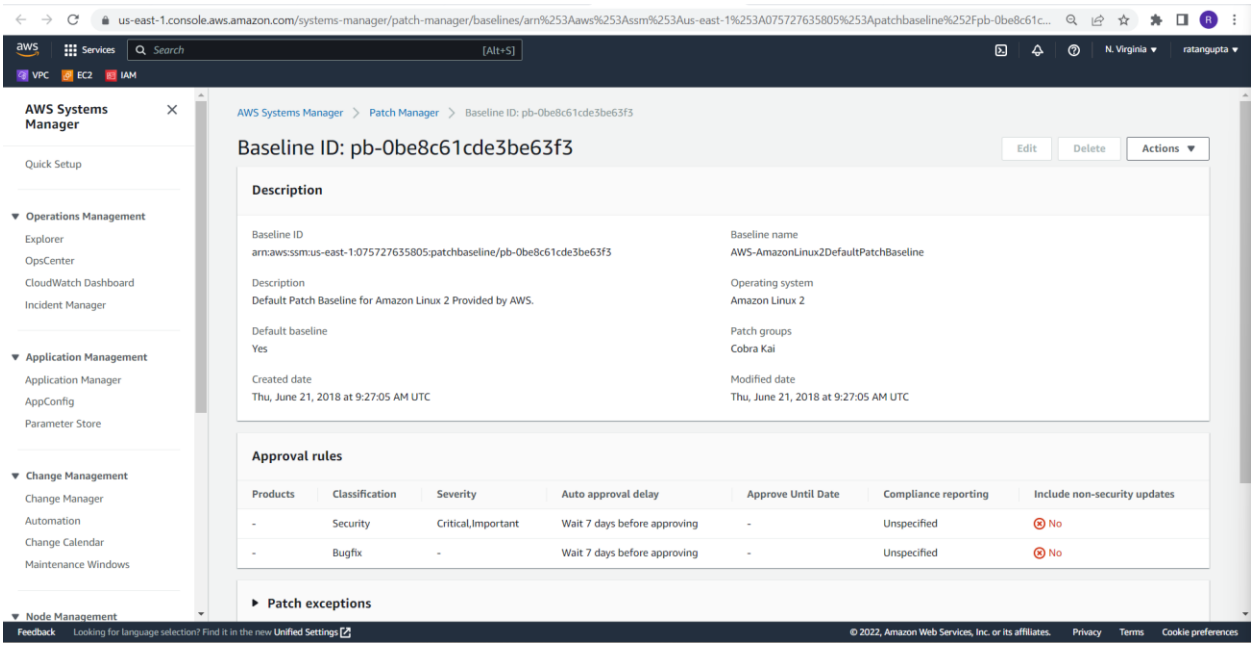

Figure 13: Patch Group Details

For this method of patching to work, make sure to add the specific IAM role and Patch Group when launching EC2 instances.

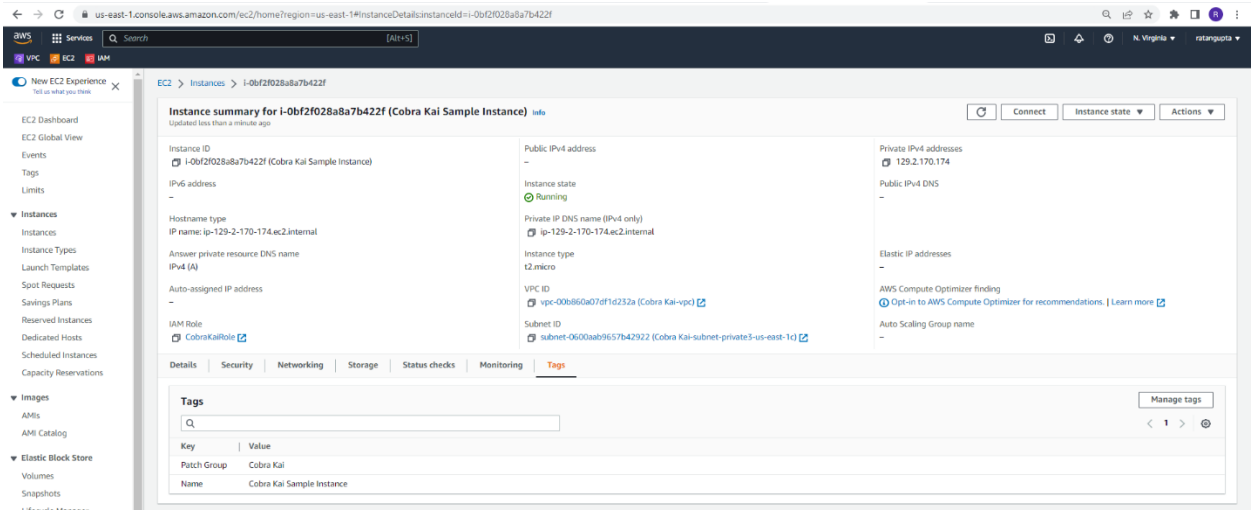

Figure 14: IAM Role and Patch Group Added to EC2 Instance

CONFIGURE PATCH MANAGER

Finally, patching can be configured by browsing to Systems Manager -> Patch Manager -> Configure Patching.

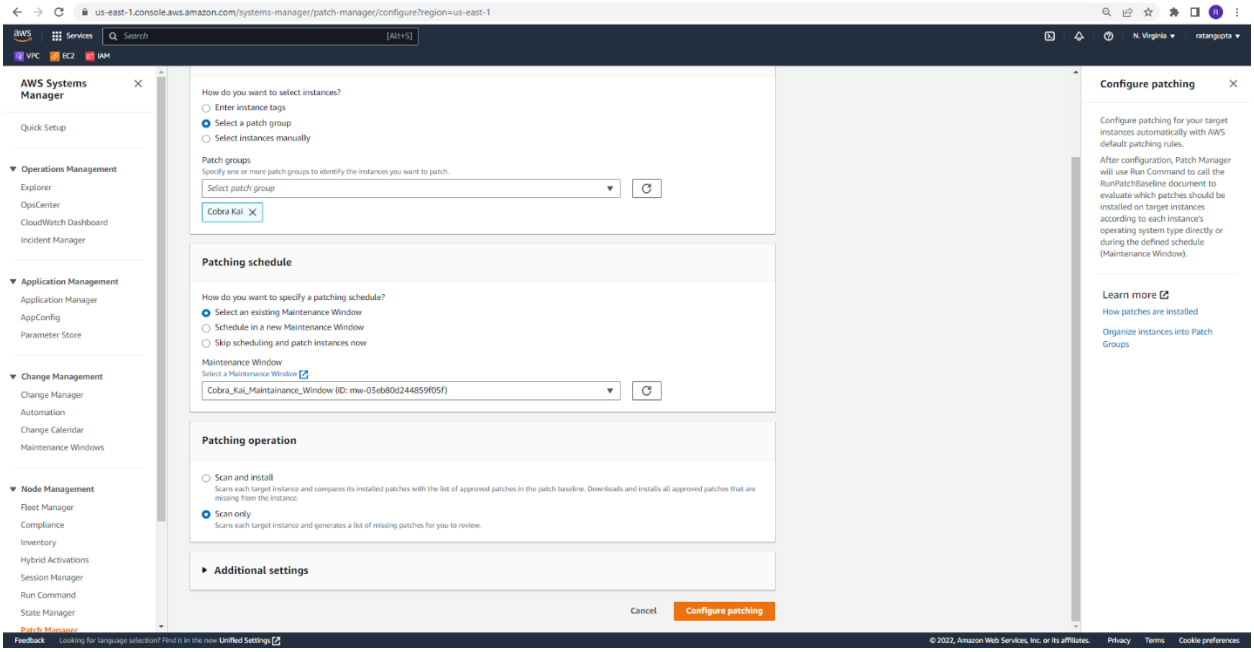

Figure 15: Configure Patching

The recommended configuration is as follows –

1. Select the Patch Group as created in the previous steps.
2. Create a maintenance window if not already created or just select an already created window. For our purposes, we created a CRON job to run every week On Sunday

midnight. This ensures that patching is done when the load on the application is minimal.

3. The patching operation is set to scan only, which allows for the admin team to study the list of missing patches generated by the Patch Manager.

As the needs and expectations of the organization change, these configurations can always change with them.

3.3. Backup Strategy

S3 BACKUPS

The recommendation is to use S3 Glacier to backup data from S3 buckets. These backups will be of the CloudTrail logs stored in the S3 buckets. The video files will not be archived to Glacier, they till be backed up in S3 buckets only using S3 Versioning.

An S3 Glacier backup can be configured by adding a Lifecycle Configuration rule to S3 buckets as illustrated in the following image –

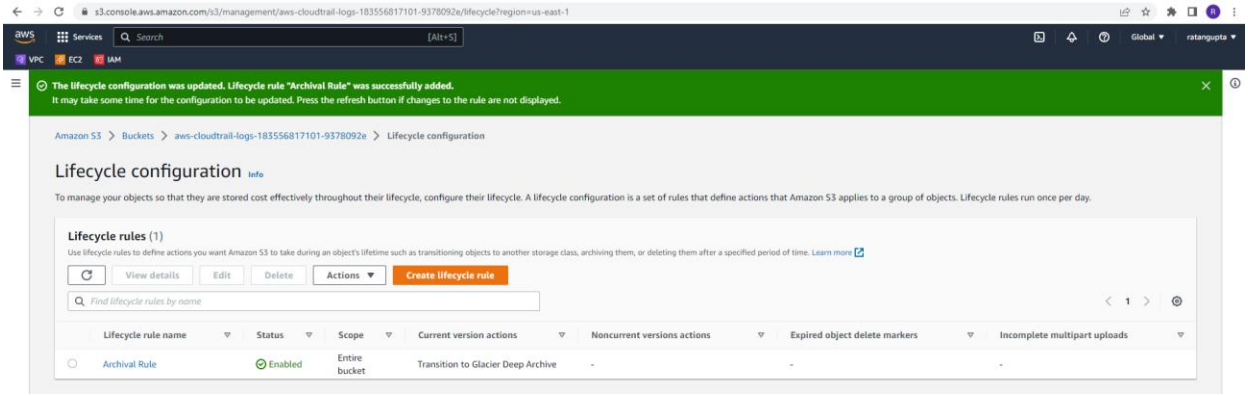

Figure 16: Transition to Glacier

To add this rule –

1. Go to the S3 bucket associated with CloudTrail and add a lifecycle configuration rule.
2. Give a name for the rule and configure which versions of objects should be transitioned.
3. Set the transition to be 'Transition to Glacier Deep Archive' and set the time to be 90 days. This would ensure that all logs will be kept in the S3 bucket for 3 months and then archived to Glacier.

RDS BACKUPS

Backups for RDS database are configured during its creation.

3.4. Account Permission Strategy

Every user of the application must be given a properly defined role in the application. A role will determine the permissions a user will have to perform actions on the application, and function within a well-defined wall of privileges. It is good practice to ensure that all users actions are functioning on the principle of least privilege.

A good way to define a role is to create user groups and assign permissions to a group. Then, users can simply be added to a group and inherit the group's permissions.

In the context of Cobra Kai, the recommendation is to have three (3) user groups –

1. Lead@CobraKai
2. Admins@CobraKai
3. Dev@CobraKai

LEADERSHIP TEAM

Since Johnny Lawrence is the Founder of Cobra Kai, and Miguel Diaz is the Chief Operating Officer (COO), they are the leadership team. As Johnny and Miguel are not technical, they are given the AWS Credential type as 'Password - AWS Management Console access' so they can login to the AWS console and perform their respective tasks.

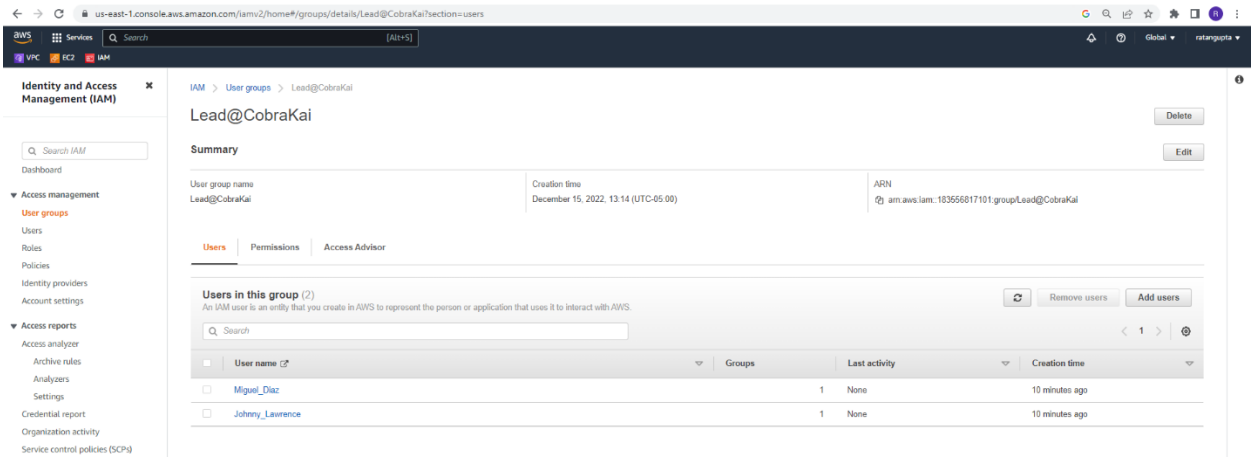

Figure 17: Leadership Team Users

The leadership team will have the following AWS permissions –

1. Billing: Provides access to account billing information and tools. Enables users to handle billing services and manage costs for the AWS services being used to host the application.
2. ReadOnlyAccess: With this permission, users have read-only access to all services.

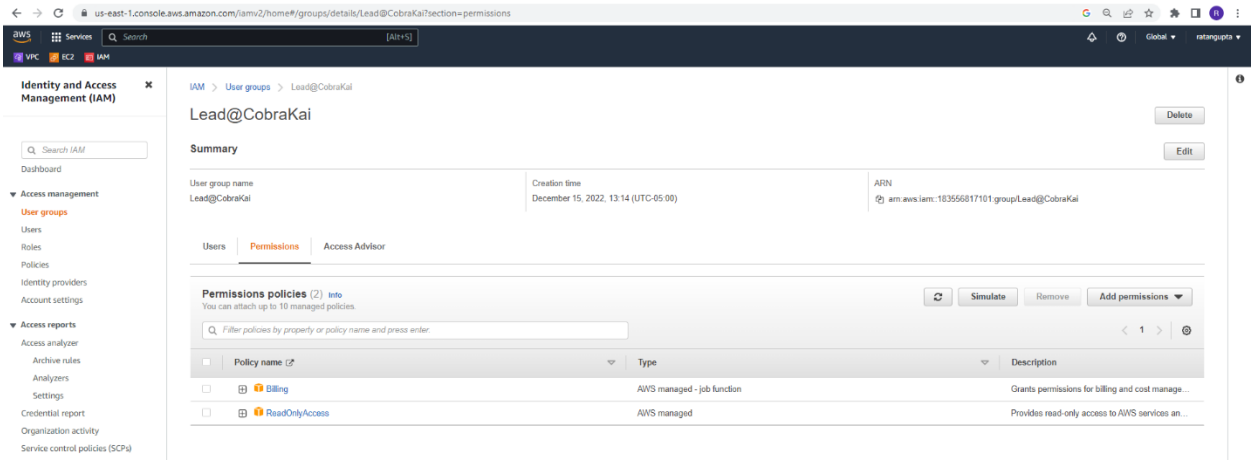

Figure 18: Leadership Team Permissions

ADMIN TEAM

The admin group will comprise of Aisha Robinson (CISO), El “Hawk” Moskowitz (CIO), and Bert (System Administrator). These users are given credential types –

1. Password - AWS Management Console access: Enables them to log into the AWS console.
2. Access key - Programmatic access: Enables them to use the access key ID and secret access key to AWS CLI and other development tools.

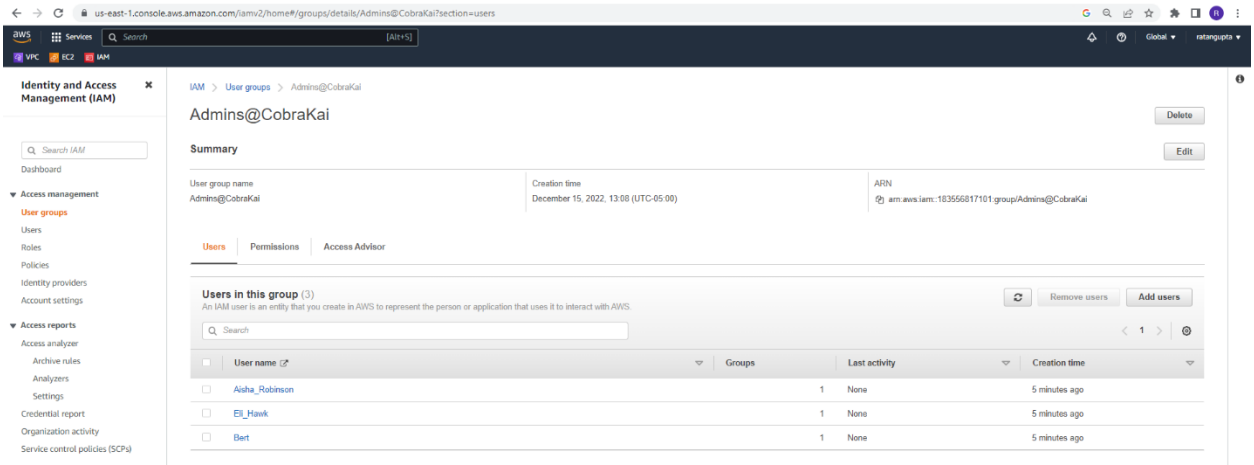

Figure 19: Admin Team Users

The admin team will have the following AWS permissions –

1. AdministratorAccess: This permission gives the admin full access to all AWS resources, services, and tools. They can manage everything that AWS has to offer and that is being used for the application.

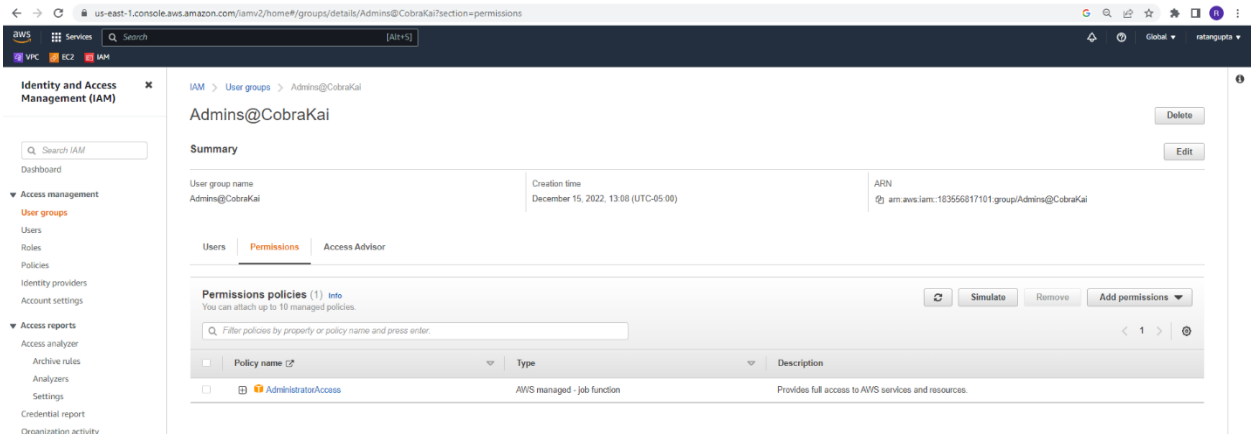

Figure 20: Admin Team Permissions

DEVELOPMENT TEAM

The development team will have Demetri as he is the web developer for Cobra Kai. Even though Eli has access to all AWS resources, it is better to add him to the development team as well since he is the brain behind the development of Cobra Kai's streaming platform. This brings clarity to the employee structure of the organization. Just like the admin team, this team will have both types of credentials AWS offers –

1. Password - AWS Management Console access: Enables them to log into the AWS console.
2. Access key - Programmatic access: Enables them to use the access key ID and secret access key to AWS CLI and other development tools.

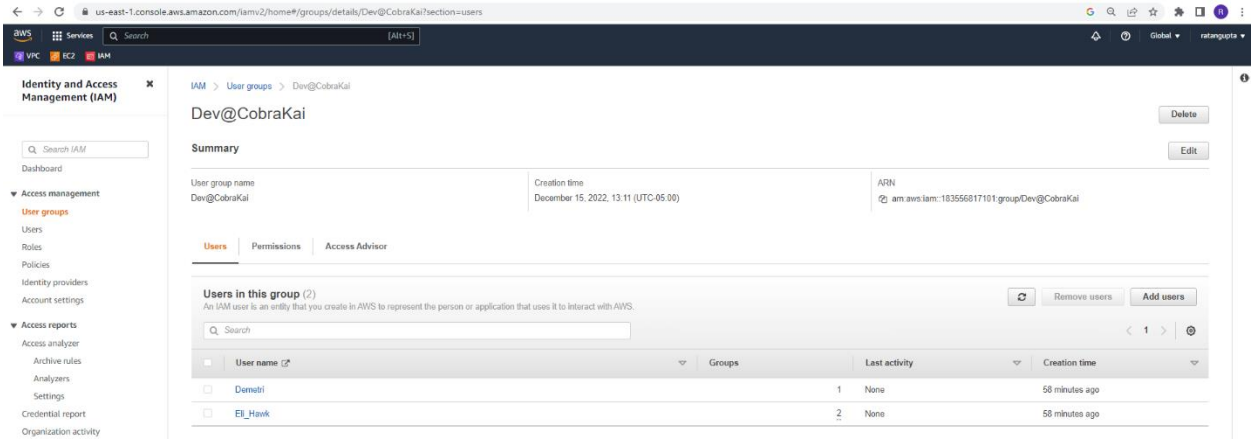

Figure 21: Development Team Users

The development team will have the following AWS permissions –

1. AmazonEC2FullAccess: This permission provides full access to all EC2 instances and other related AWS services like Elastic Load Balancer (ELB), CloudWatch, EC2 Auto Scaling, etc.

2. AmazonS3FullAccess: This permission provides full access to all S3 buckets and Lambda functions.
3. AmazonRDSFullAccess: This permission provides full access to RDS database along with limited access to other related AWS resources and services such as CloudWatch (read/write access), CloudWatch Logs (list/read access), EC2 (list/read access), etc.
4. IAMReadOnlyAccess: Provides read only access to IAM users, groups, and policies.

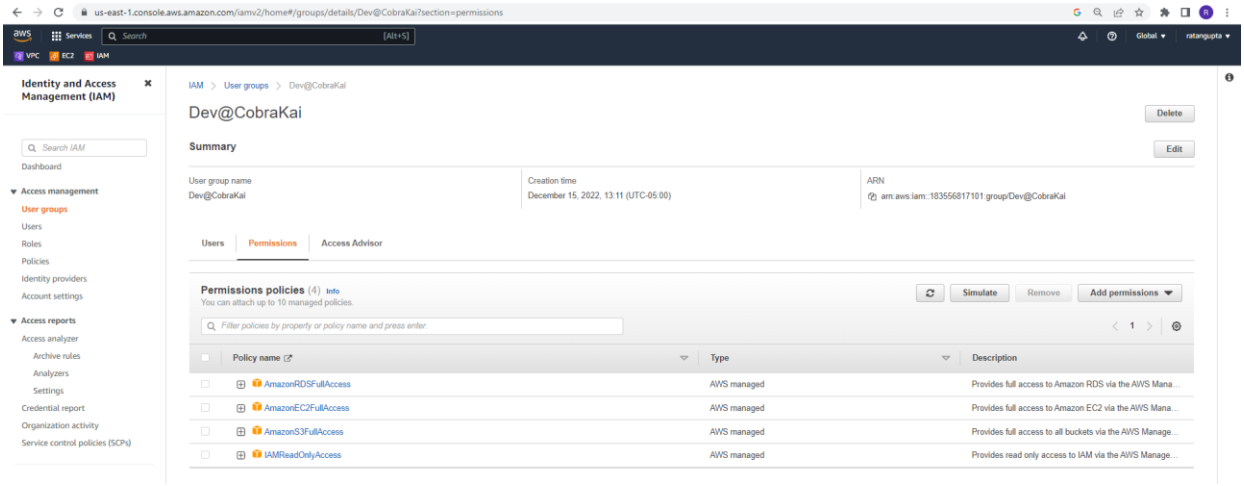

Figure 22: Development Team Permissions

The aforementioned user groups and policies are a good place to start the migration with. As time progresses, user groups and policies can be added or modified if the need arises.

AWS COGNITO

It is highly recommended to use AWS Cognito to set up sign-in and sign-up features and it incorporates security. It sets up identity stores (user pools) where it stores user login data and provides options to use federated login mechanisms.

Cognito can be setup as follows –

Create a user pool and set up sign-in features.

1. It is recommended to give users the option to sign-in with a username or an email address.
2. Set up strong password requirements like a password must have a minimum of 12 characters, at least one (1) uppercase character, at least one (1) lowercase character, at least one (1) special character, and at least one (1) number.
3. Set up MFA using third party authenticator apps. It is recommended to have MFA as a requirement for security reasons (defense in depth).
4. Setup email as the password recovery option.
5. Cognito provides an option to customize token configuration such as expiration and refresh time, and other advanced app settings. These can be left with the default values at this stage.

6. Lastly, Cognito can be integrated with AWS Web Application Firewall as well.

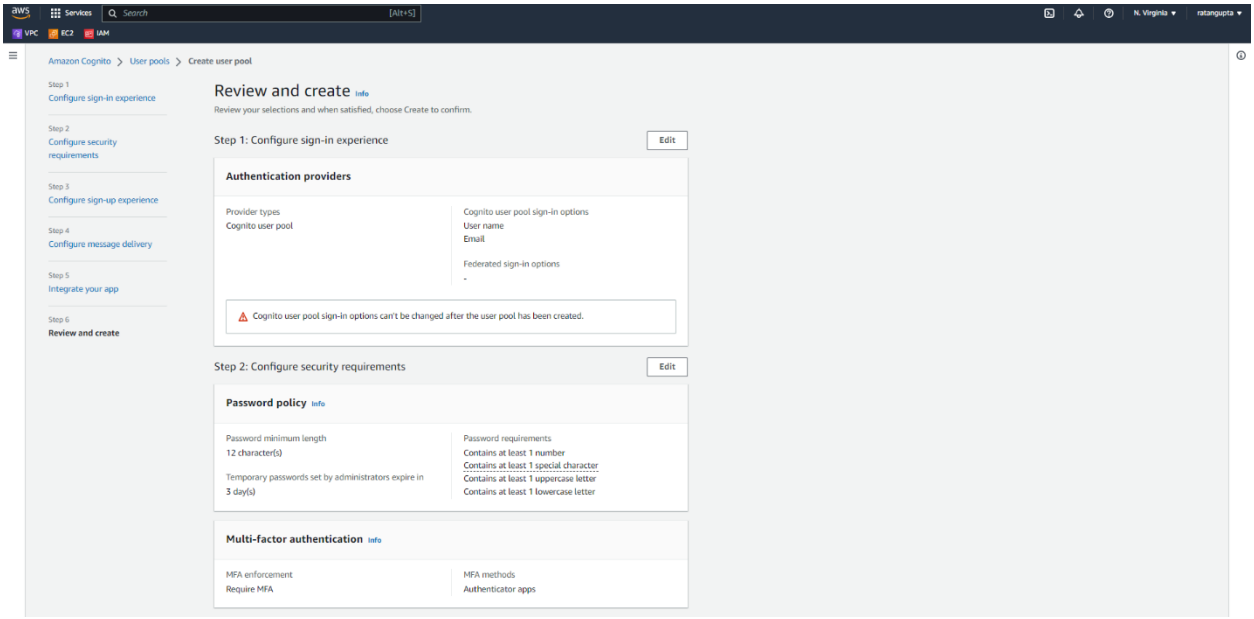

Figure 23A: Configuring Cognito

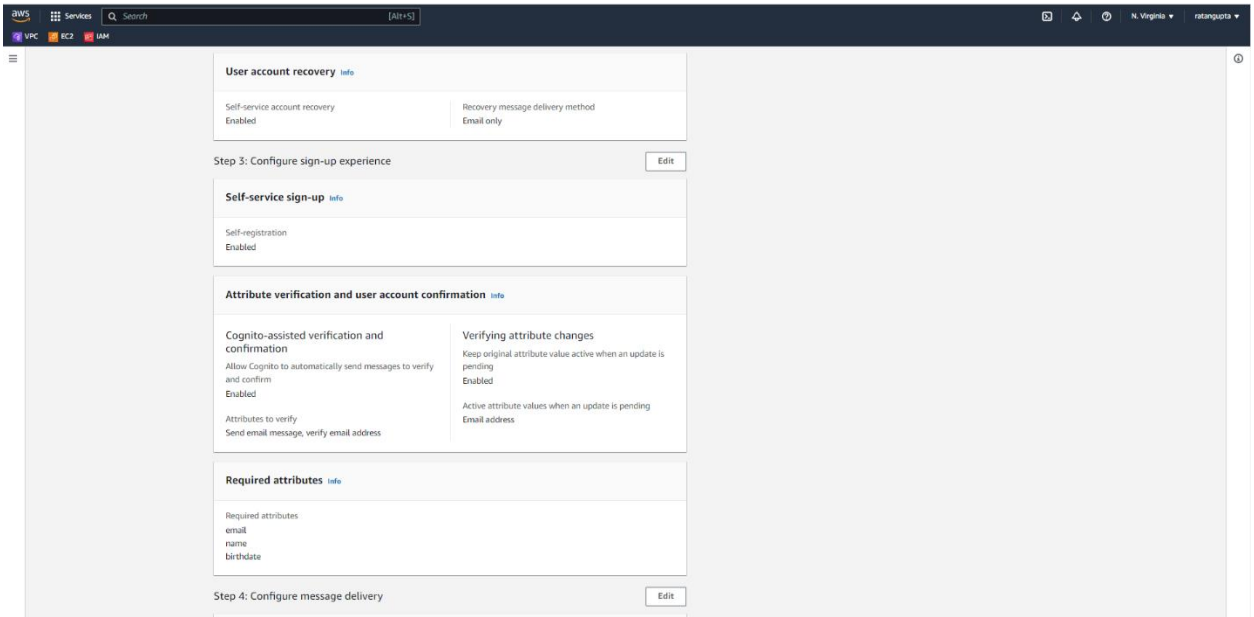

Figure 23B: Configuring Cognito

3.5. Mitigate DDoS, hardware failures, and human errors

AWS SHIELD

AWS Shield should be used to prevent DDoS attacks. It can automatically detect Denial of Service attacks and mitigate the attempts. AWS offers a free tier version of Shield and a paid version called Shield Advanced. The free tier is automatically enabled once an account is created but it offers limited protection and is not recommended.

Shield Advanced offers more security assurance than the free version and is the one that Cobra Kai should consider using.

To setup AWS Shield Advanced –

1. Subscribe to Shield Advanced using the AWS console and agree to the terms and conditions.
2. Add the resources from the different regions that need to be protected from the threat of DoS attacks. These would normally be Load Balancers, CloudFront resources, Route 53 routes, etc.
3. Configure ACLs, rate limiters, and Route 53 health checks.
4. Setup the AWS Shield Response Team for 24/7 support. The SRT team offers mitigation support in case the system is under attack. SRT support can be set up by editing the SRT access option and creating a new role “drt.shield.amazonaws.com”.
5. Add contacts which can be used by SRT if they find anything out of the ordinary during health checks.

AWS WAF (WEB APPLICATION FIREWALL)

AWS Web Application Firewall (WAF) must be set up and deployed between the VPC and the rest of the internet.

WAF is recommended to help protect against DoS attacks and other security exploits that might be caused by malicious requests such as CSRF, phishing, SQLi, and open redirection attacks.

When configuring WAF, make sure to do the following –

1. Create web ACLs to filter incoming HTTP(S) requests and allow or block requests depending on whether the requests conform to predefined (by the admins) security rules. Use a white-list approach to filter incoming requests.
2. Define conditions to allow or block incoming requests such as setting limits on the size of requests, blocking certain IP address ranges, implementing REGEX pattern matching, and detecting, and sanitizing user input to prevent CSRF and SQL Injections, etc.
3. Make use of AWS managed rule groups which can be found in AWS marketplace.

WAF will help protect AWS resources such as CloudFront, Load Balancer, Cognito, etc.

3.6. Performance Enhancement

AMAZON CLOUDFRONT

Amazon CloudFront will be used to deliver the on-demand content to the users.

Configure the CloudFront distribution so that users can access the content stored in S3 bucket only through CloudFront.

Give then origin domain as the S3 bucket instance. Then create an Origin Access Control Setting using the S3 bucket so that the bucket restricts access to only CloudFront. This creates a way for CloudFront to send authenticated requests to the S3 bucket.

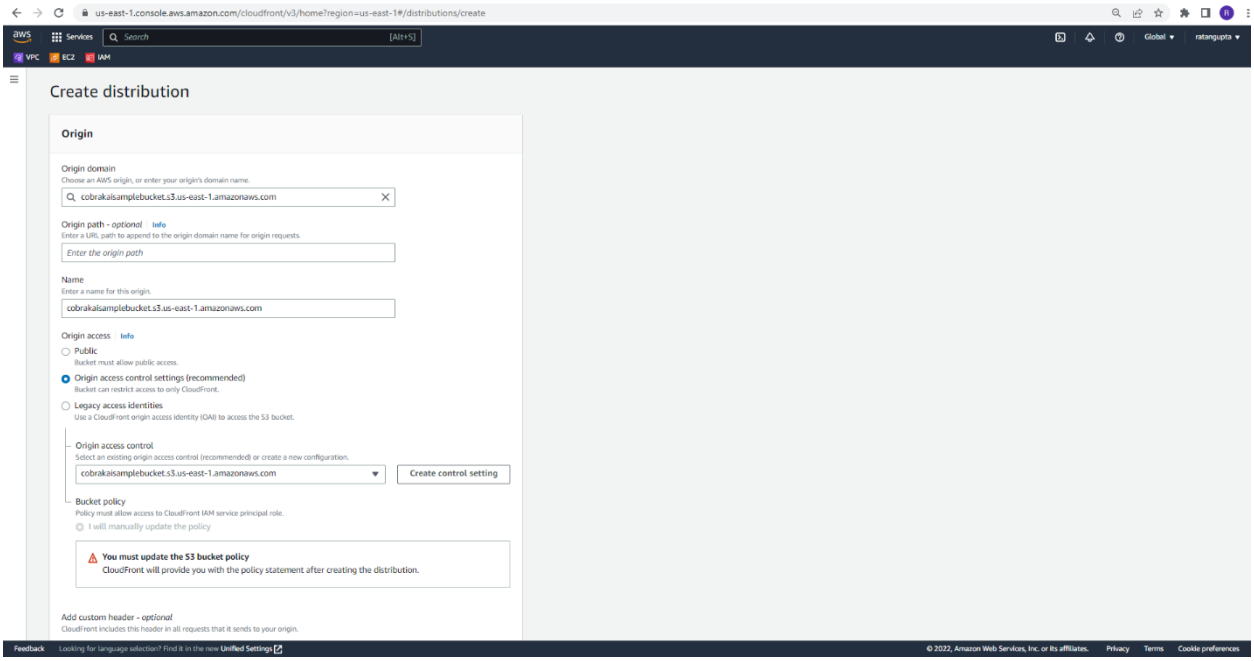

Figure 24: Setting up Origin Domain and Access Control Settings

Set the cache settings as illustrated in the image below –

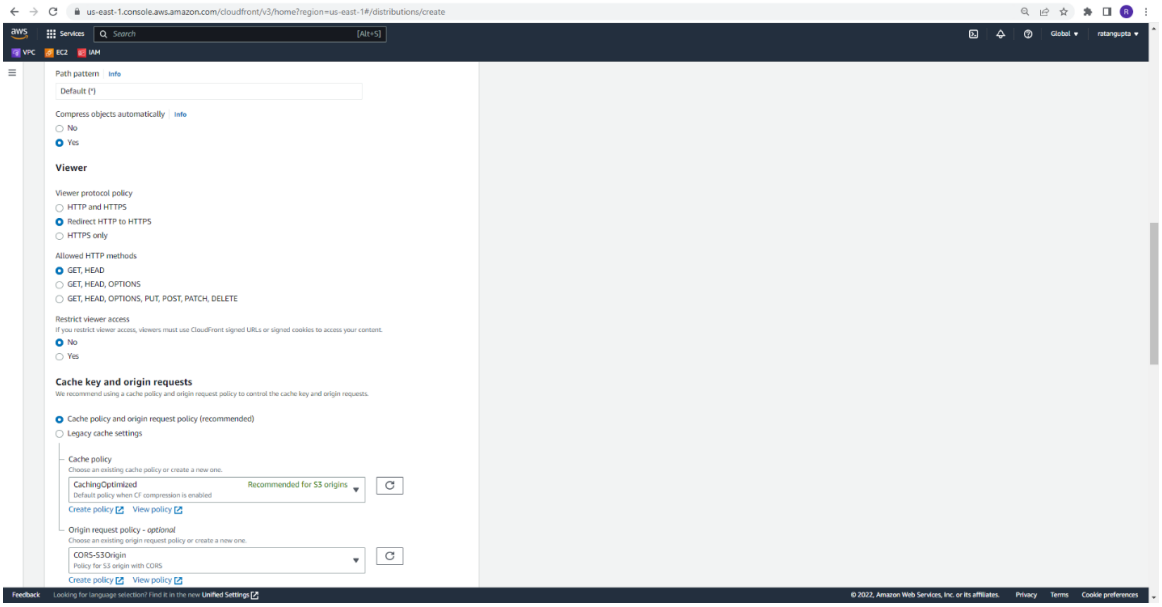

Figure 25: Cache Settings

The rest of the settings can be left to defaults –

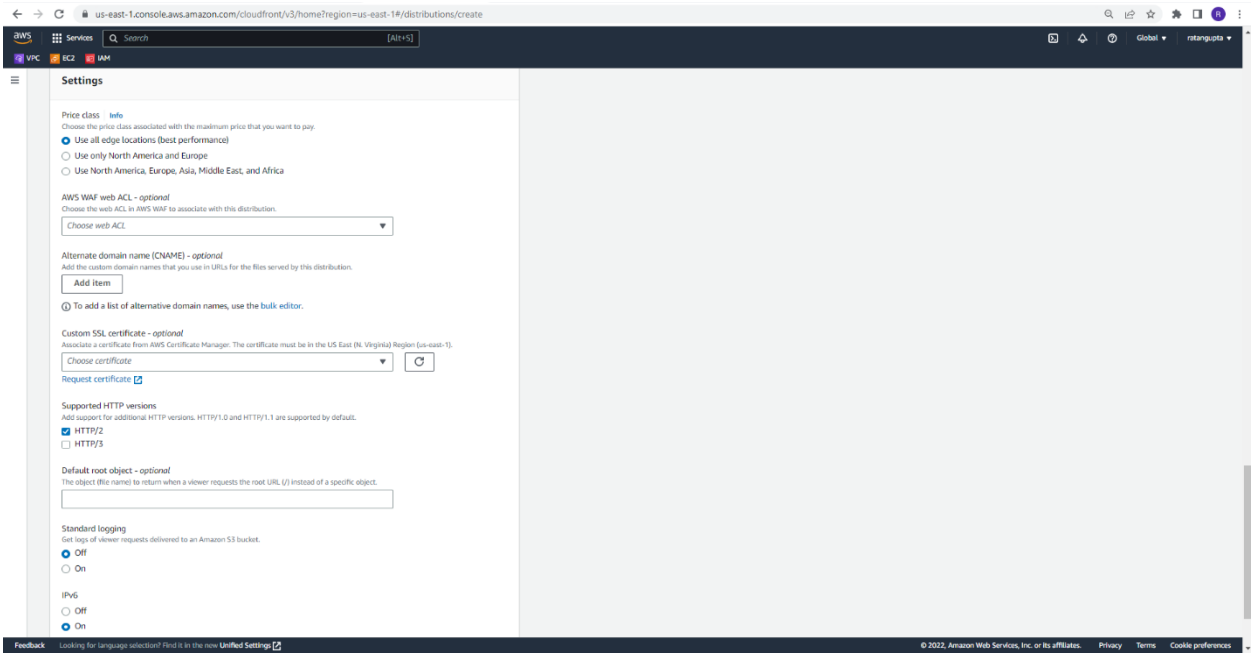

Figure 26: Default Settings

After creating the distribution, CloudFront generates a policy. Attach that policy to the S3 bucket as follows –

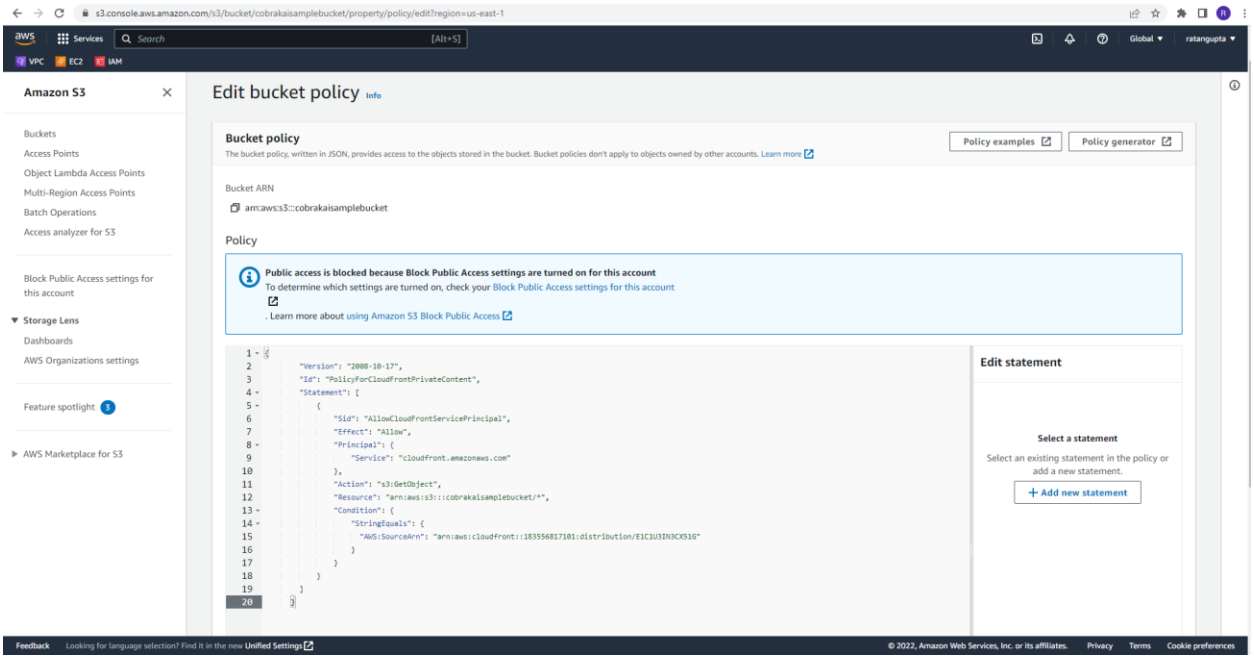

Figure 27: Updating S3 bucket policy

This restricts access to the S3 bucket and makes it inaccessible through any other channel.

Now, the content of the S3 bucket can only be accessed through CloudFront.

3.7. Logging and Monitoring

AWS CLOUDTRAIL

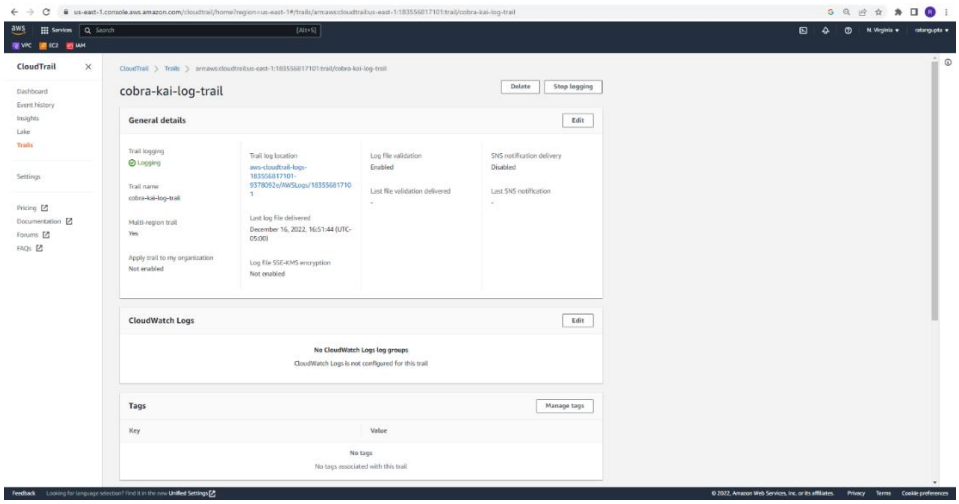

Figure 28: CloudTrail

In order to monitor the use of different resources and keep a track of all events to spot any abnormalities, it is recommended to use AWS CloudTrail.

1. CloudTrail can be set up using the AWS console.
2. It automatically creates an S3 bucket (where the logs will be stored) when a Trail is created and starts monitoring all the AWS resources of the account.
3. It is recommended to encrypt the log files using SSE-KMS encryption.
4. Enable log file validation to keep a track of the log files. This feature helps determine if a log file was modified or deleted after it has been delivered by CloudTrail. This is achieved by having log file digests which are also stored in S3 bucket.

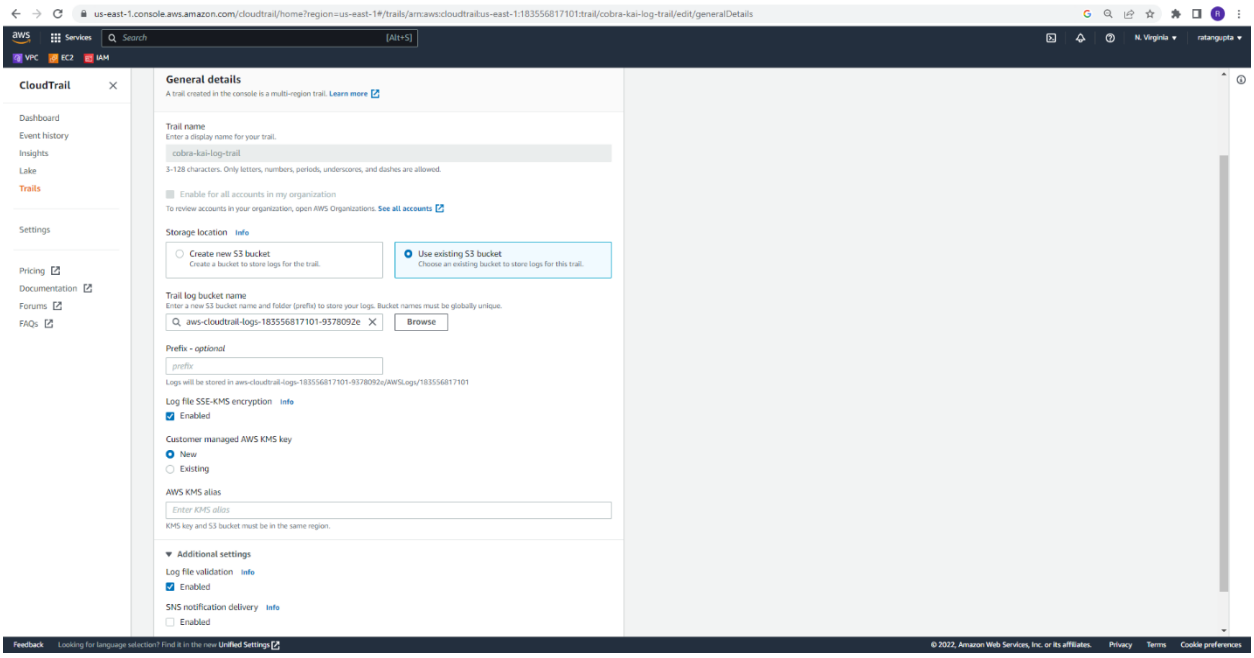

Figure 29: CloudTrail Configurations

AWS CLOUDWATCH

Use CloudWatch to monitor resources and get notified if certain predetermined events transpire.

For example, an alarm can be set up if an EC2 instance's CPU utilization exceeds 85%. An admin will get notified of the event and can take the necessary actions on it.

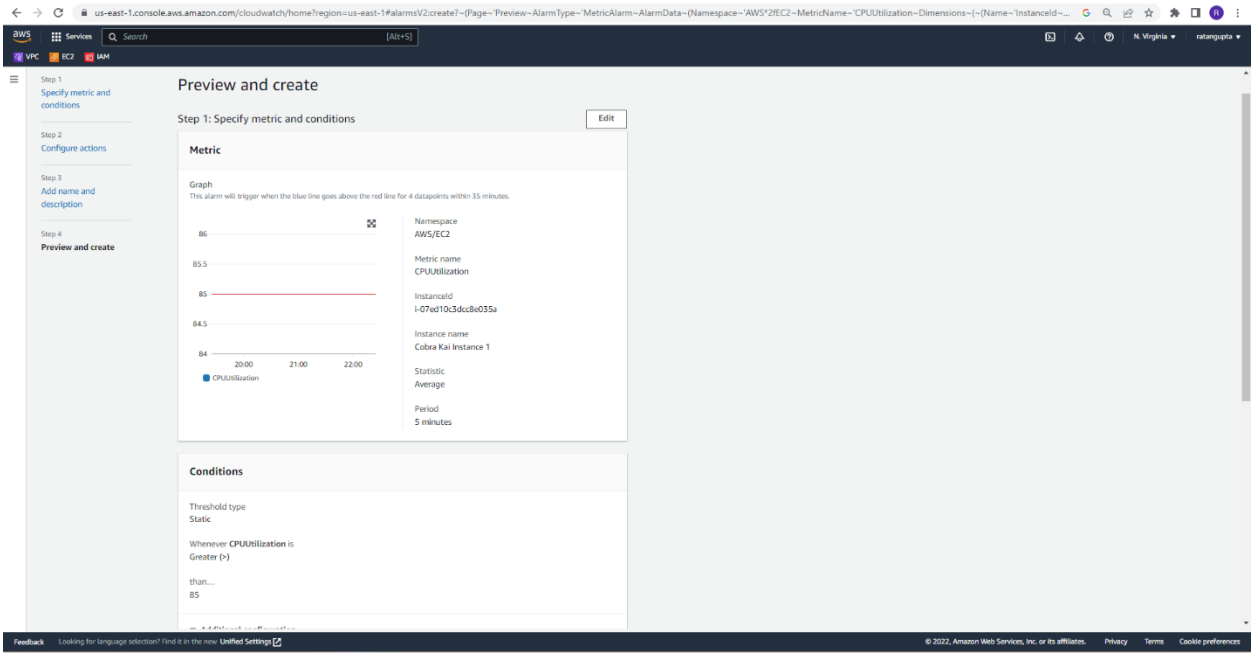

Figure 30A: CloudWatch Alarm

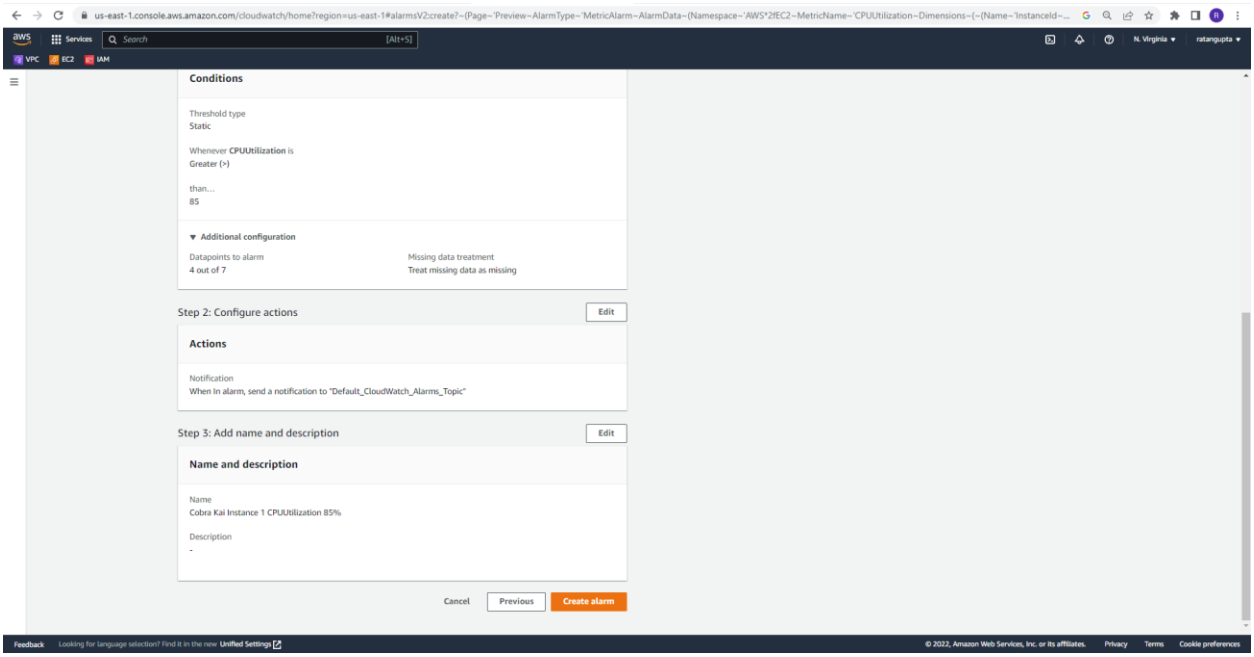

Figure 30B: CloudWatch Alarm

3.8. PCI DSS COMPLIANCE

In order to comply with the PCI DSS Requirements, we make the following recommendations. Some of these recommendations have been implemented in the document, and the others should be implemented by the admin and development team when migrating to the cloud.

1. Restrict ports 22 and 80 in the cloud architecture.
2. Use KMS for encryption for CloudTrail logs.
3. Integrate CloudWatch logs with CloudTrail.
4. Load Balancers in the auto-scaling groups must perform health checks.
5. Maintain a firewall to protect user's card data. AWS Firewall Manager can be used for this purpose. It comes with a myriad of services including AWS WAF, Shield, GuardDuty, etc. These services help protect PII and other sensitive data.
6. Encrypt all the databases. Using KMS for encryption is a good idea.
7. In order to keep track of every user on the network, assign every individual with a unique ID. This helps with easier logging and auditing of all user activities.
8. Use an anti-virus solution as AWS does not have its own antivirus software. AWS has a partnership with Bit Defender. Using Bit Defender is recommended.
9. Use AWS Inspector to check operating systems for vulnerabilities frequently. Use the knowledge to patch the systems immediately.
10. Archive all logs (stored in S3 buckets) in Glacier Deep Archive.
11. Use AWS Config to enable recording of the resources against which PCI DSS performs checks.

4. CONCLUSION

This document provided implementations and configurations for most of the recommendations that were pitched to the Cobra Kai team. All the implementations have been made keeping security the focal point of the discussion.

VPCs were created to isolate the Cobra Kai environment. Secure configurations of EC2 instances, S3 buckets and RDS databases were implemented. Security Groups were created. IAM user groups and policies were created to give a sense of the different roles the Cobra Kai team should have along with different permissions. Patching strategy was discussed with the implementation of Patch Manager. The document showed how to archive logs from S3 buckets to S3 Glacier. Logging and monitoring were achieved using AWS CloudTrail and AWS CloudWatch. AWS Shield and AWS WAF were used to offer protection against DoS attacks and other exploits. AWS CloudFront was used to deliver on-demand media content to users. Lastly, PCI DSS requirements were discussed and specific services that have not been mentioned in the document were recommended.

With the context of the proposed architecture, a few services like ElastiCache, SNS, Bastion Host, and Directory Service are recommended to use.

5. REFERENCES

1. VPC: https://docs.amazonaws.cn/en_us/vpc/latest/userguide/what-is-amazon-vpc.html
2. IAM: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html
3. Shield: https://docs.aws.amazon.com/shield/
4. Patch Manager: https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-patch.html
5. Cognito: https://aws.amazon.com/cognito/
6. CloudFront: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html
7. PCI DSS: https://www.controlcase.com/what-are-the-12-requirements-of-pci-dss-compliance/
8. PCI DSS: https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-pci-controls.html
9. AWS Documentation: https://docs.aws.amazon.com/index.html