ENPM 665 CLOUD SECURITY
FINAL PROJECT (IMPLEMENTATION OF CLOUD SECURITY ARCHITECTURE FOR COBRA KAI)

AAKASH RAMAN
UNIVERSITY ID: 119211663

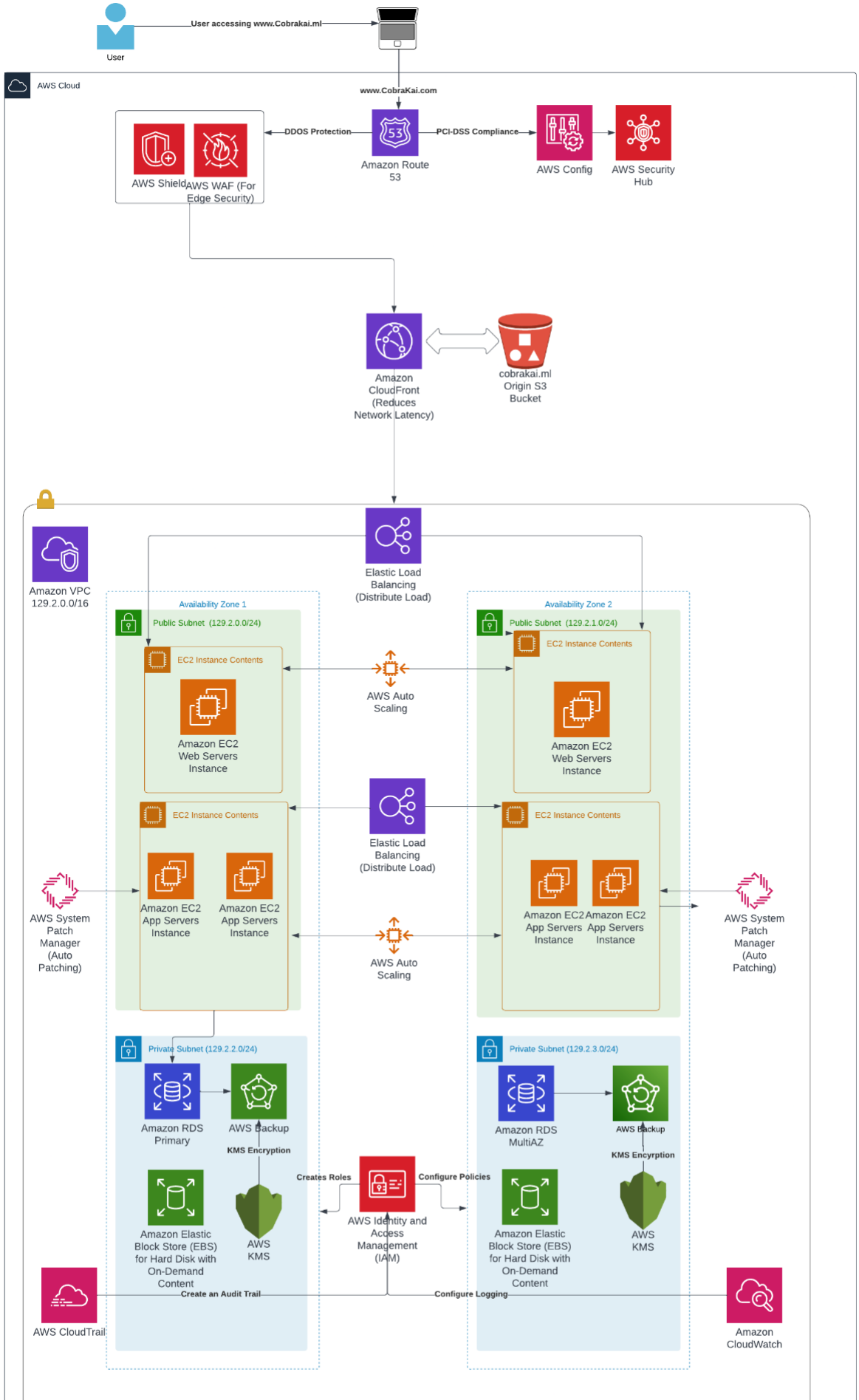
Introduction

Now that I have got approval from the upper management to develop the migration of Cobra Kai's services to the Cloud, there are some important recommendations that I would like to highlight in this report.

Some of these include configuring a proper Identity & Access Management Policy for all the employees at Cobra Kai, maintaining regular backups of the Database server, configuring logging and monitoring of services using AWS Cloudwatch & CloudTrail, deploying DoS protection using tools like AWS WAF & AWS Shield, AWS CloudFront and Route 53 for Content Delivery and Web Hosting, ensuring that connections made remotely are in a secure manner instead of being available globally, configuring a System and Patch Manager, and ensuring compliance with PCI-DSS and using Virtual Private Clouds for Networking and managing them using Security Groups and NACLs (Network Access Control Lists).

Architectural Diagram for Implementation:

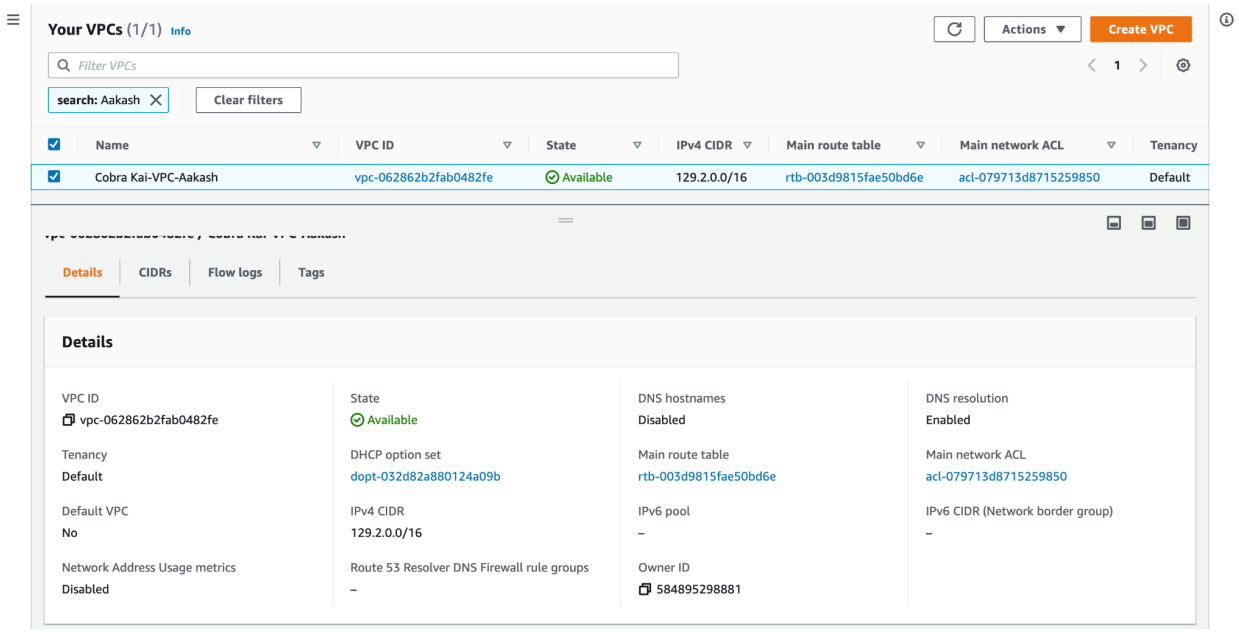
The Architectural Diagram that I am going to implement is shown below.


User
User accessing www.Cobrakai.ml
AWS Cloud
www.CobraKai.com
AWS Shield
AWS WAF (For Edge Security)
DDOS Protection
Amazon Route 53
PCI-DSS Compliance
AWS Config
AWS Security Hub
Amazon CloudFront (Reduces Network Latency)
cobrakai.ml Origin S3 Bucket
Amazon VPC 129.2.0.0/16
Elastic Load Balancing (Distribute Load)
Availability Zone 1
Public Subnet (129.2.0.0/24)
EC2 Instance Contents
Amazon EC2 Web Servers Instance
AWS Auto Scaling
Availability Zone 2
Public Subnet (129.2.1.0/24)
EC2 Instance Contents
Amazon EC2 Web Servers Instance
Elastic Load Balancing (Distribute Load)
EC2 Instance Contents
Amazon EC2 App Servers Instance
Amazon EC2 App Servers Instance
AWS System Patch Manager (Auto Patching)
AWS Auto Scaling
EC2 Instance Contents
Amazon EC2 App Servers Instance
Amazon EC2 App Servers Instance
AWS System Patch Manager (Auto Patching)
Private Subnet (129.2.2.0/24)
Amazon RDS Primary
AWS Backup
KMS Encryption
Amazon Elastic Block Store (EBS) for Hard Disk with On-Demand Content
AWS KMS
Creates Roles
Configure Policies
AWS Identity and Access Management (IAM)
Private Subnet (129.2.3.0/24)
Amazon RDS MultiAZ
AWS Backup
KMS Encryption
Amazon Elastic Block Store (EBS) for Hard Disk with On-Demand Content
AWS KMS
AWS CloudTrail
Create an Audit Trail
Configure Logging
Amazon CloudWatch

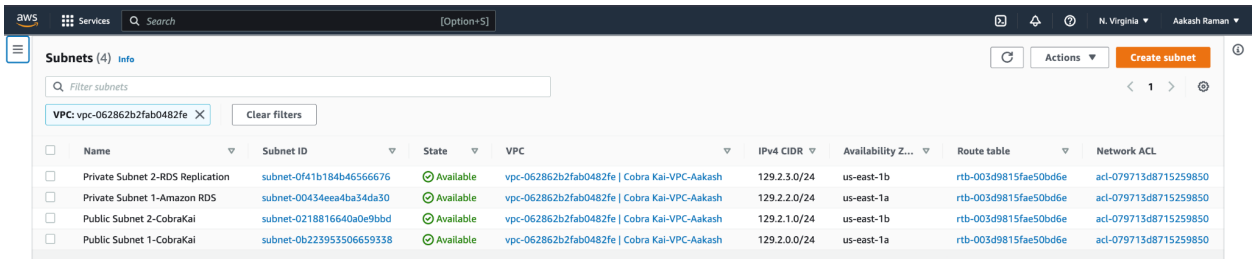
Initial Setup of Cloud Environment:

The below screenshots highlight the initial setup of the Cobra Kai application I performed on AWS:

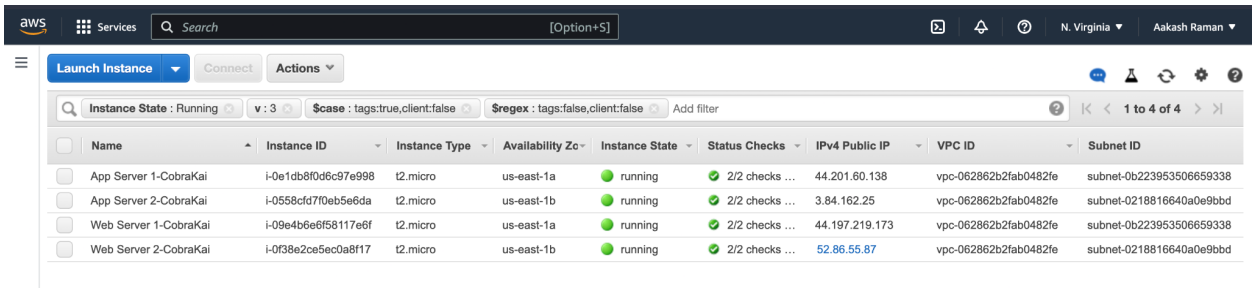
1. The first step is to create the VPC for our Cobra Kai application. This VPC has a CIDR Block of **129.2.0.0/16** and a **Default Tenancy**. It also has a Main Route Table of **rtb-003d9815fae50bd6e** and a Main Network ACL of **acl-079713d8715259850**.



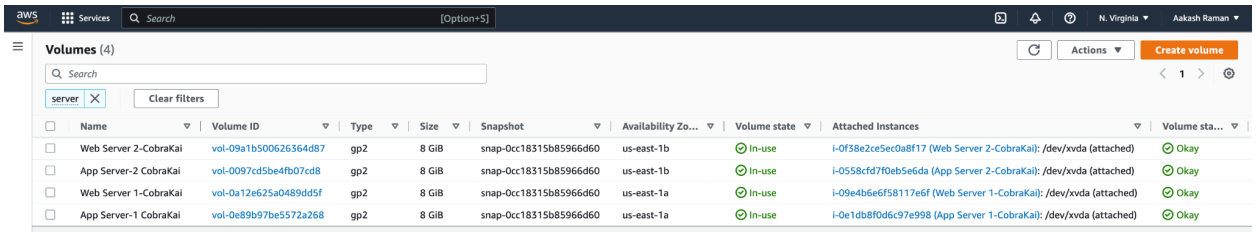
2. The next step is to create subnets for the VPC we created. Subnetting is the process of dividing the CIDR block into smaller segments. My VPCs CIDR Block is 129.2.0.0/16 which I have divided into 4 subnets and their IDs are shown in the below screenshot. In order to ensure high availability, I have created a Public-Private subnet pair in both "**us-east-1a**" and "**us-east-1b**" Availability Zone respectively.



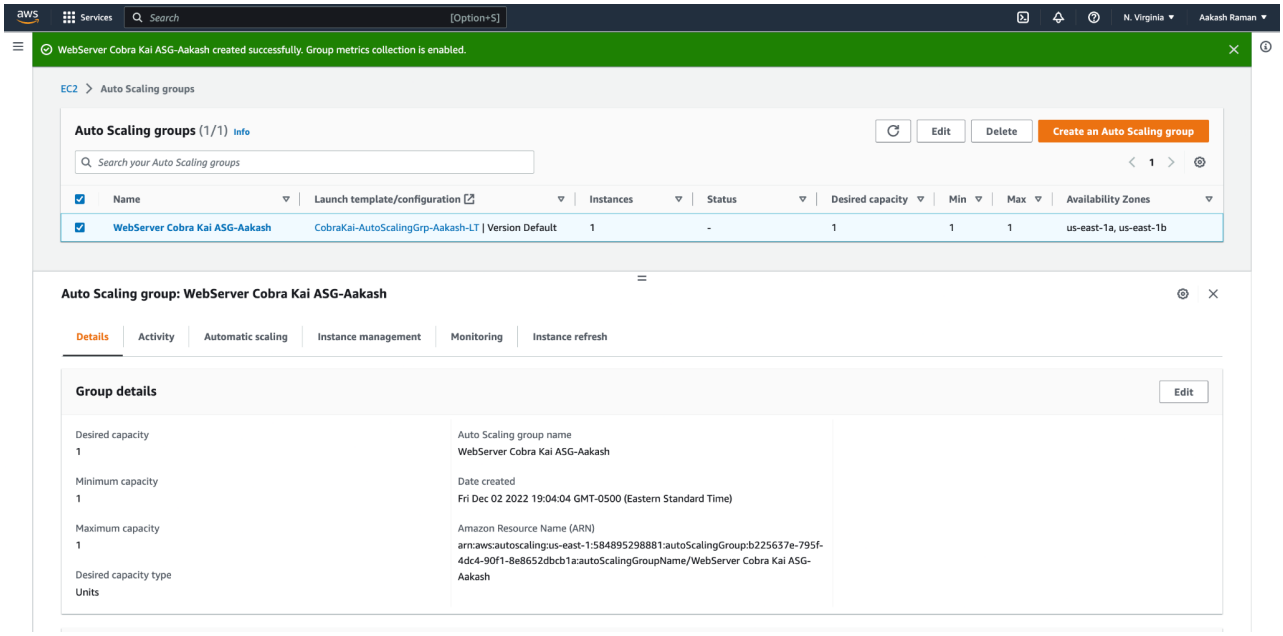
3. The next step is to set up 4 EC2 Instances (2 App Servers & 2 Web Servers) of Cobra Kai and configure them with the initial VPC and Subnet I created. I also designed Auto Scaling groups for each of the instances that enhance scalability depending on the load. One Web-App Server is in Public Subnet 1 & the other Web-App Server is in Public Subnet 2. We also need to set up the storage for each of our instances so we can back them up easily.



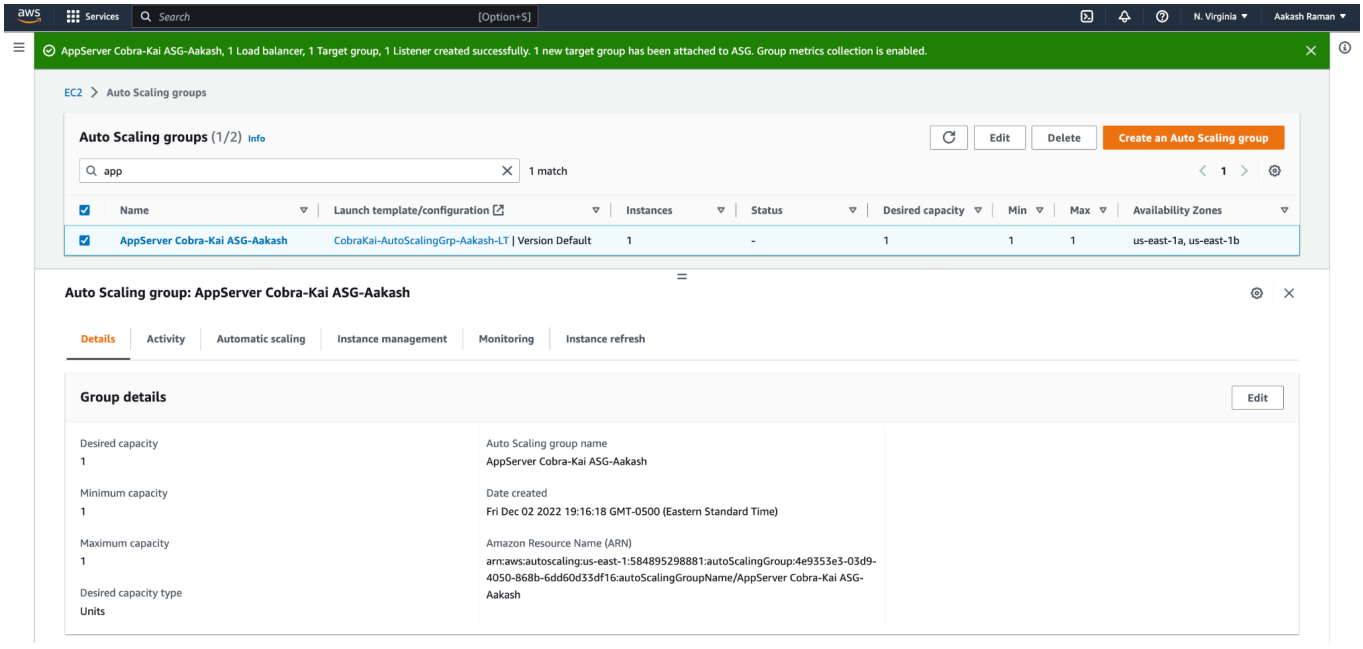
The following screenshot shows the EBS (Elastic Block Storage) volumes that I have attached to each of my server's EC2 Instances (Web & App Servers).



4. To achieve automatic scalability for Cobra Kai virtual instances, we can use the concept of Auto Scaling Groups. I have set up 1 Auto Scaling Group "WebServer Cobra-Kai ASG-Aakash" with the desired capacity of 1 instance. This means that 1 EC2 Instance will be created or terminated depending on the load.

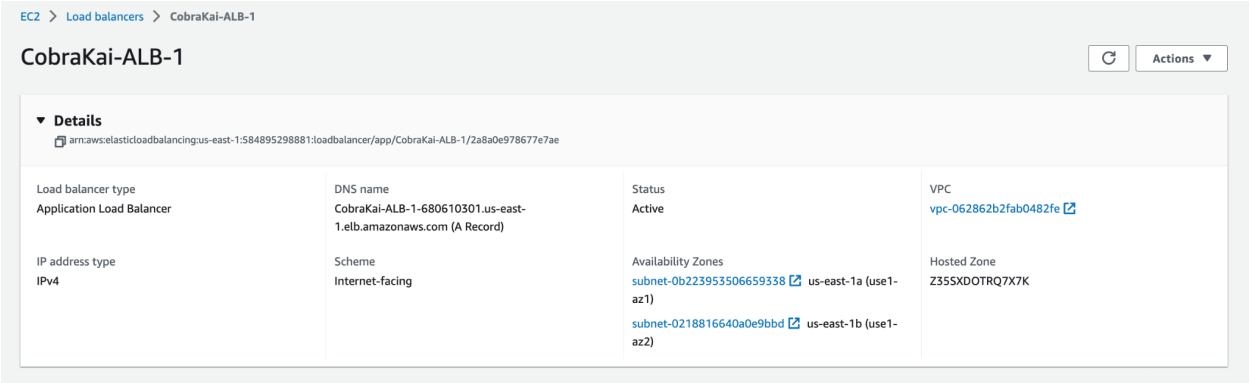


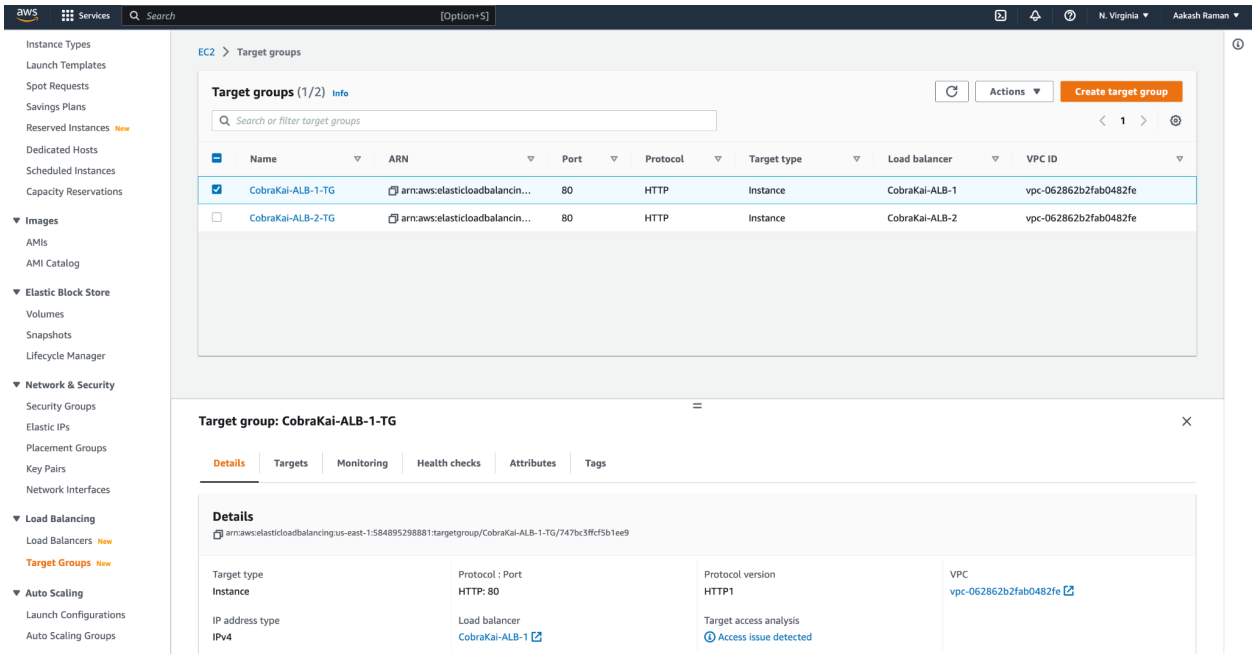
5. Similarly, I have set up 1 more Auto Scaling Group “AppServer Cobra-Kai ASG-Aakash” with the desired capacity of 1 instance. This means that 1 EC2 Instance will be created or terminated depending on the load.



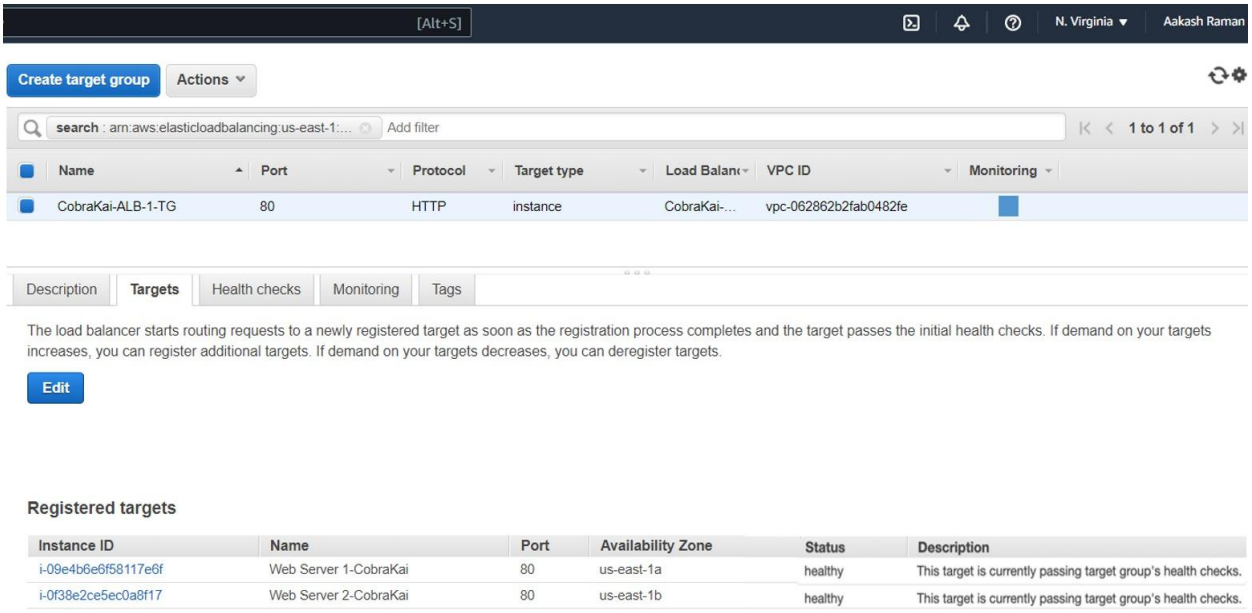
6. To handle any load, I have created 2 Load Balancers to distribute the load among the EC2 Instances which are added to its Target Group. Here, ALB-1 is Load Balancer 1 which

has the Target Group TG-1 attached to it. Similarly, Load Balancer 2 has Target Group TG-2. The Load Balancer will check the health of the instances and check their status by redirecting the requests to the Registered Targets.

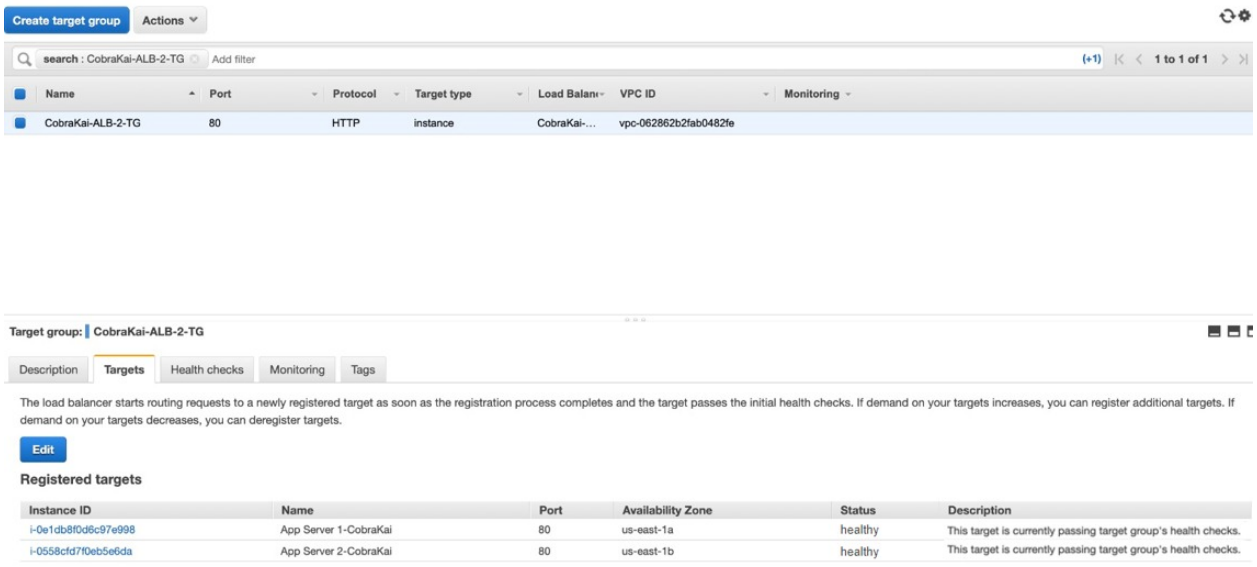




The below screenshot shows the Target Group of **CobraKai-ALB-1** where our Web Servers are added as **Registered Targets** and are "**healthy**".



The below screenshot shows the Target Group of **CobraKai-ALB-2** where our App Servers are added as **Registered Targets** and are "**healthy**".



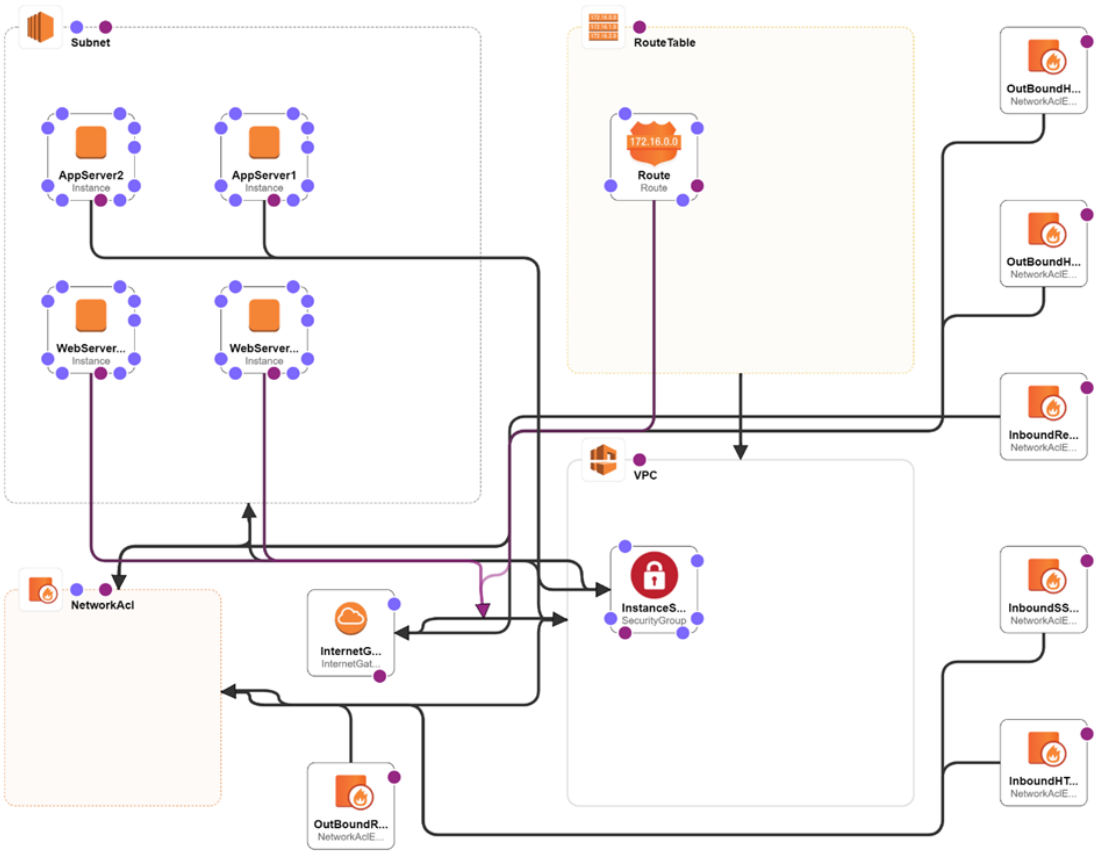
Recommendations for Automatic Patch Management using AWS System Manager:

Patching resources is always a solid Security practice that is very useful to reduce the attack surface and risk of vulnerabilities affecting Cobra Kai's assets. AWS uses the concept of AWS SSM to achieve Patch Management.

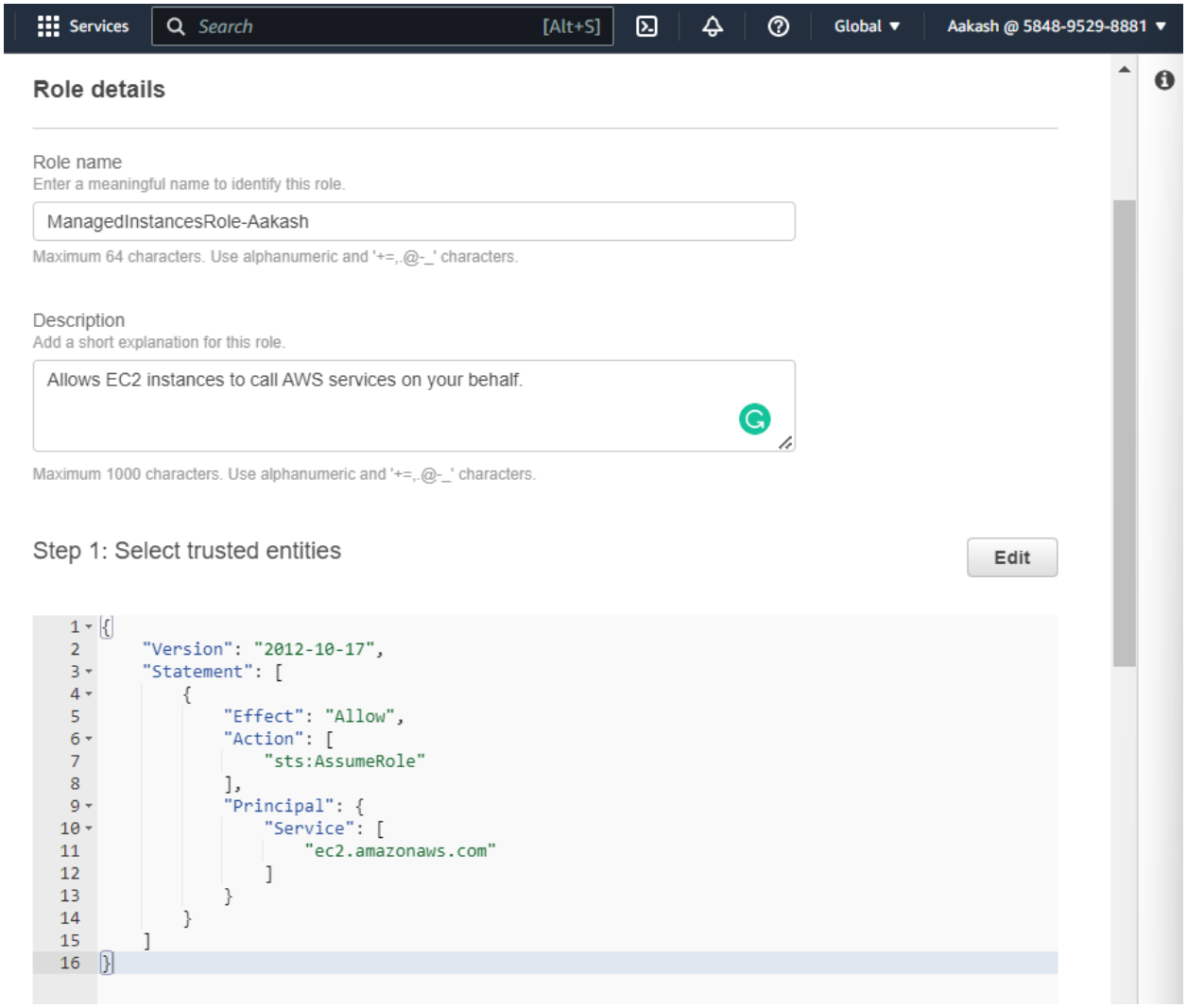
AWS SSM (System Manager) is a platform that provides automated patching capabilities, Fleet, Node & Change Management for all the instances launched that have in-built security controls. AWS SSM can provide automated patching for both Windows & Linux-based instances.

I have used the concepts of ***Infrastructure as Code*** and ***Operations as Code*** to first set up an AWS SSM template from this link from AWS Labs and then used AWS CloudFormation to deploy that template on my instances to set up the patching.

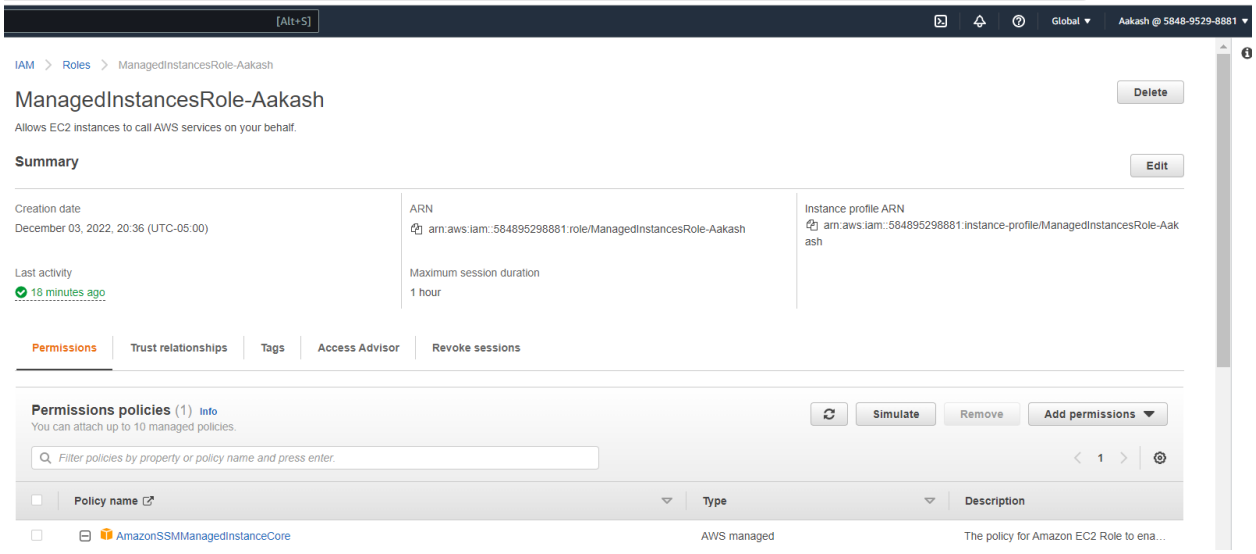
1. The below screenshot highlight the architecture of the AWS System Manager, which I performed on AWS. You can see it in the "**View in Designer**" option while creating a CloudFormation Stack.



2. AWS SSM **requires an IAM Role** in order to allow users to execute commands and instances to process those commands. Hence, the below screenshot depicts a "**ManagedInstancesRole**" that allows EC2 Instances to call AWS services based on the user's needs.



3. The below screenshot depicts the policies that I have added to the above Role. Here, the "**AmazonSSMManagedInstanceCore**" is used.



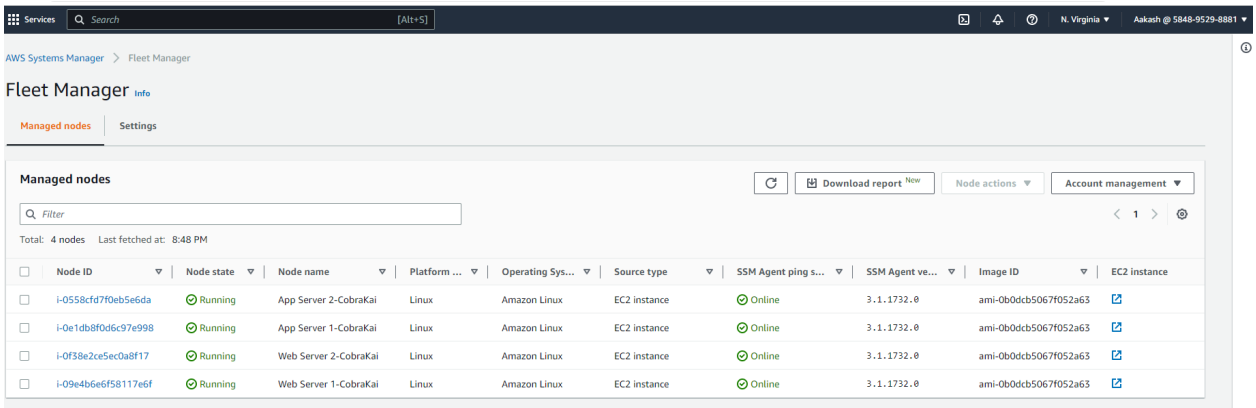
4. The below screenshots show the "**AmazonSSMManagedInstanceCore**" policy in JSON Format.


```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:DescribeAssociation",
                "ssm:GetDeployablePatchSnapshotForInstance",
                "ssm:GetDocument",
                "ssm:DescribeDocument",
                "ssm:GetManifest",
                "ssm:GetParameter",
                "ssm:GetParameters",
                "ssm:ListAssociations",
                "ssm:ListInstanceAssociations",
                "ssm:PutInventory",
                "ssm:PutComplianceItems",
                "ssm:PutConfigurePackageResult",
                "ssm:UpdateAssociationStatus",
                "ssm:UpdateInstanceAssociationStatus",
                "ssm:UpdateInstanceInformation"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateControlChannel",
                "ssmmessages:CreateDataChannel",
                "ssmmessages:OpenControlChannel",
                "ssmmessages:OpenDataChannel"
```

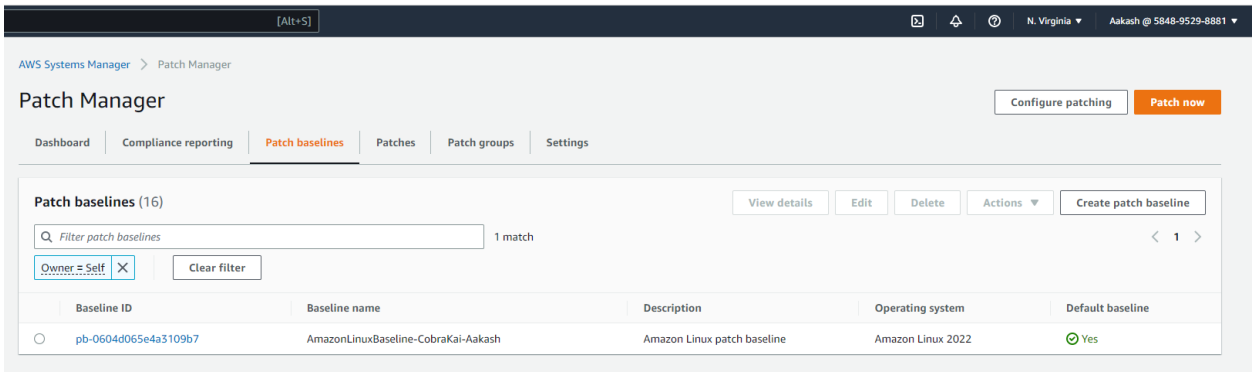
```
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2messages:AcknowledgeMessage",
                "ec2messages:DeleteMessage",
                "ec2messages:FailMessage",
                "ec2messages:GetEndpoint",
                "ec2messages:GetMessages",
                "ec2messages:SendReply"
            ],
            "Resource": "*"
        }
    ]
}
```

5. Now that a Role has been attached to AWS SSM, the next step is to ensure that all the instances have an **SSM Agent installed**. This is crucial for the Patching process to take place. Luckily, as a new feature, AWS already installs SSM Agents for each instance according to the Operating System.

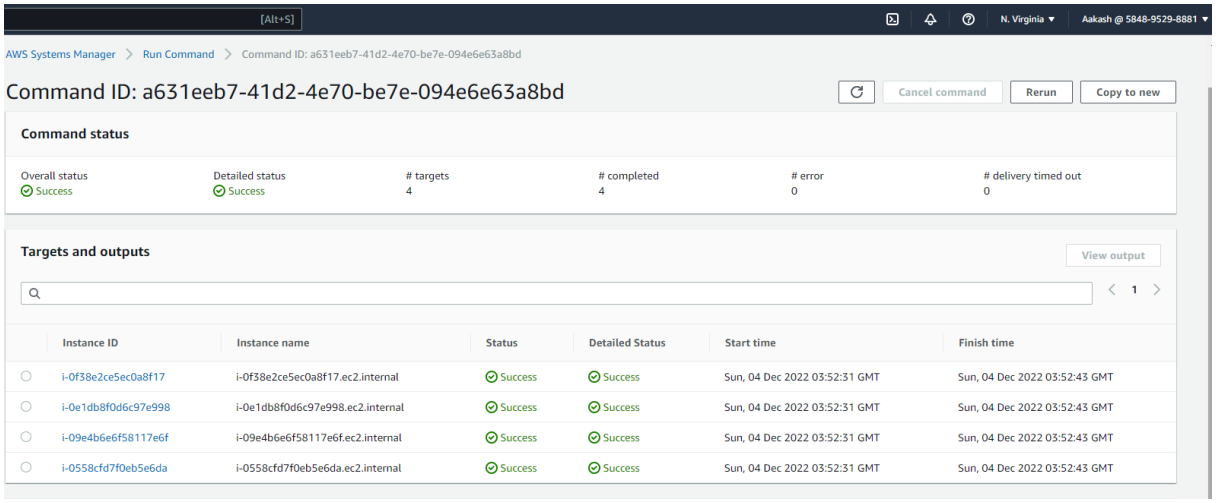
   Once that is done, all the Web and App servers are added to AWS FleetManager and their status is "**Online**" so that they are ready to get patched as shown by the below screenshot:



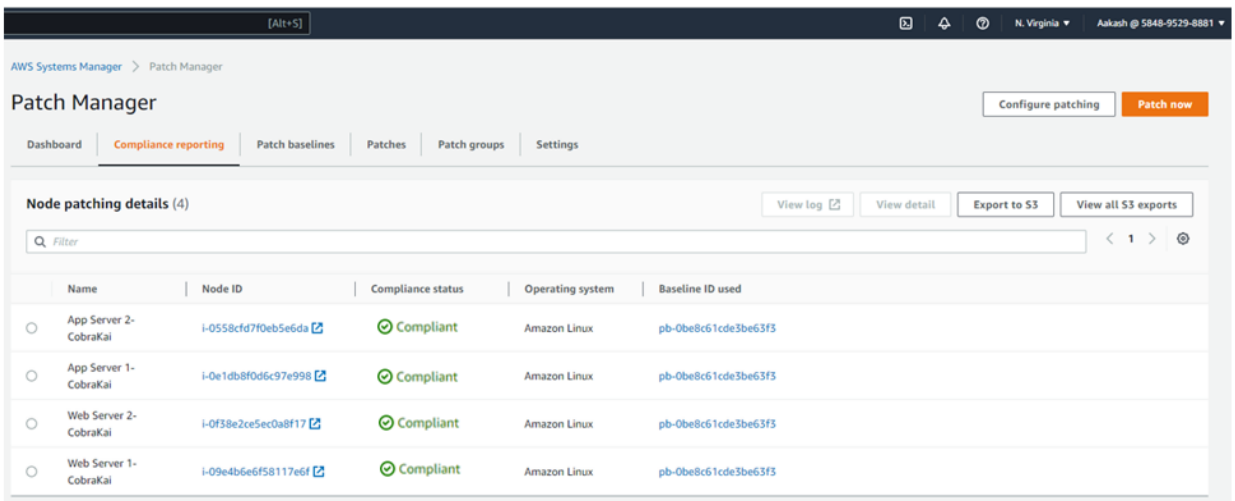
6. Next, a **Patch Baseline** needs to be established depending on the Operating System that defines what patches need to be installed, the rules for the Patching process as well as the timeline for patching. This way CobraKai can choose maintenance hours for patching their servers and reduce their attack surface.



7. AWS also has some predefined Patch Baselines and we can use those Baselines for Patching as well. In the below screenshot, I am using the "**SSM run command**" to perform the Patching process using the predefined **AWS-RunPatchBaseline.** This is used for immediate Patching rather than having a Scheduled Patching.



8. The AWS SSM also produces a **Patch Management Compliance Report** which judges how well the instances that are scheduled to be patched are Patch Compliant or have any configuration inconsistencies. This is shown in the below screenshot.



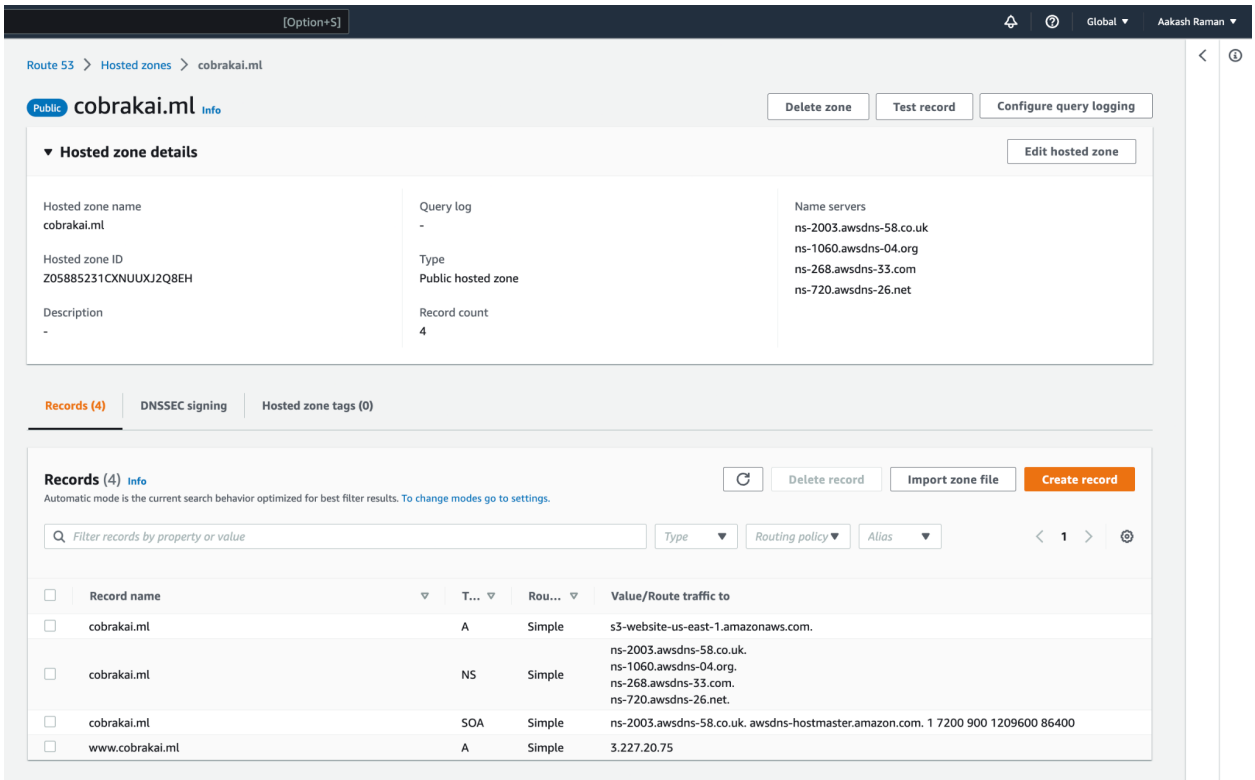
Hence, Cobra Kai can use the AWS SSM to regularly patch all their servers and ensure that updates are performed in a timely manner.

Recommendation for Handling DDoS Attacks & Alternative Routes using Route 53 & AWS WAF:

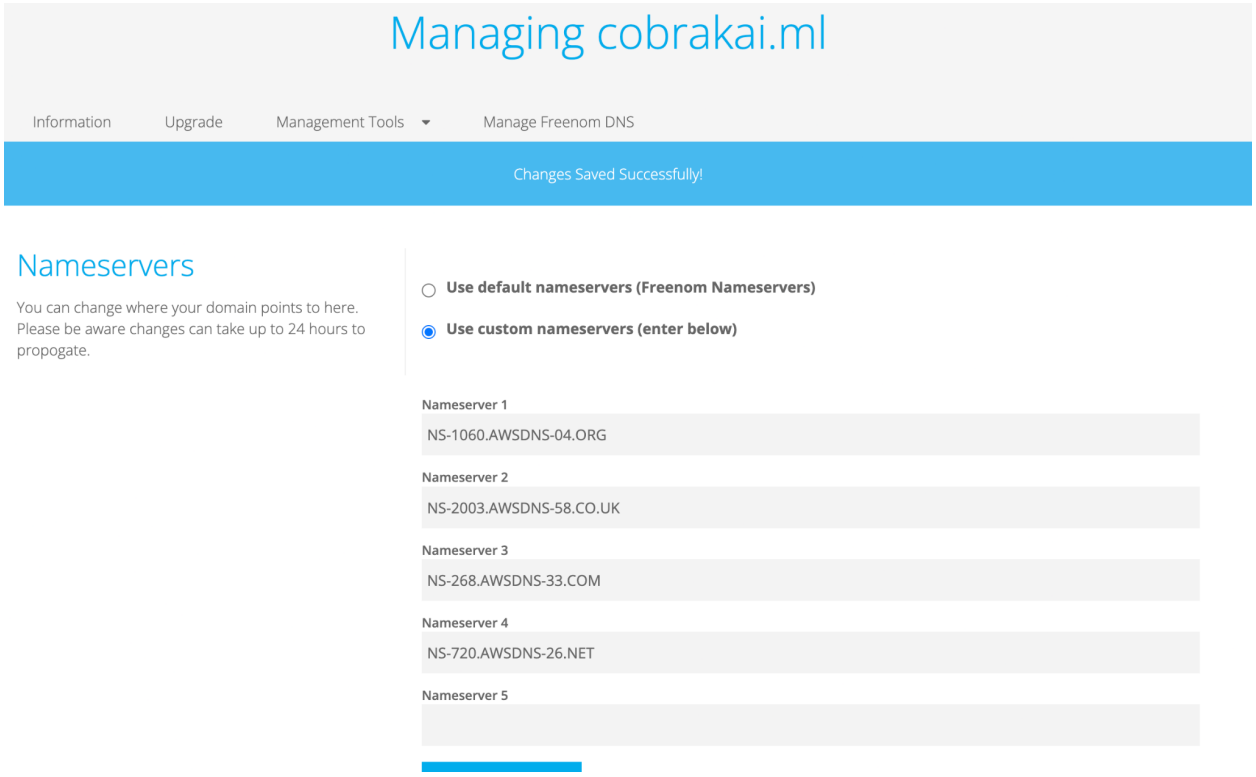
The main idea to handle DDoS attacks and create a secure alternative route to the website, "www.cobrakai.ml" is to route traffic using Route 53 and then set up AWS WAF/Shield to protect that incoming traffic and enhance resiliency and reduce Network Latency, use a Content Distribution Network, AWS CloudFront.

1. I am planning to host the Cobra Kai Website using a Free Domain Registrar "www.Freenom.com". The URL I used was "www.cobrakai.ml". The process to set up AWS Static Web Hosting along with the Sample Website was given in https://github.com/kts262/enpm809j.

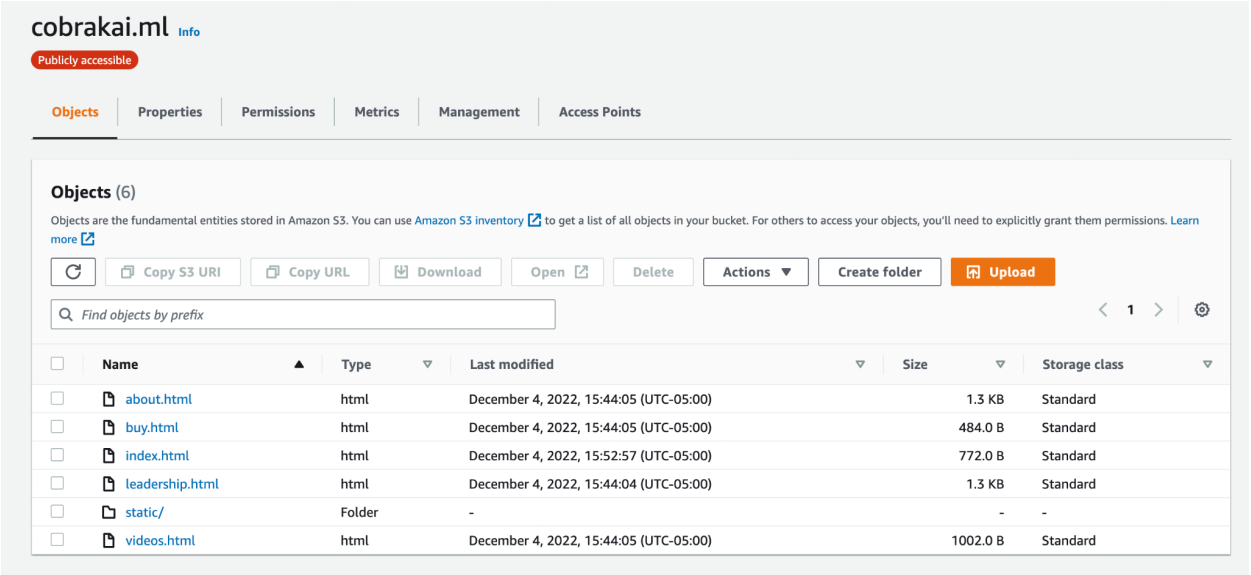
A. The below screenshot shows all the DNS Records (A, SOA & NS) of "www.cobrakai.ml" of the hosted zone I created in Route 53:



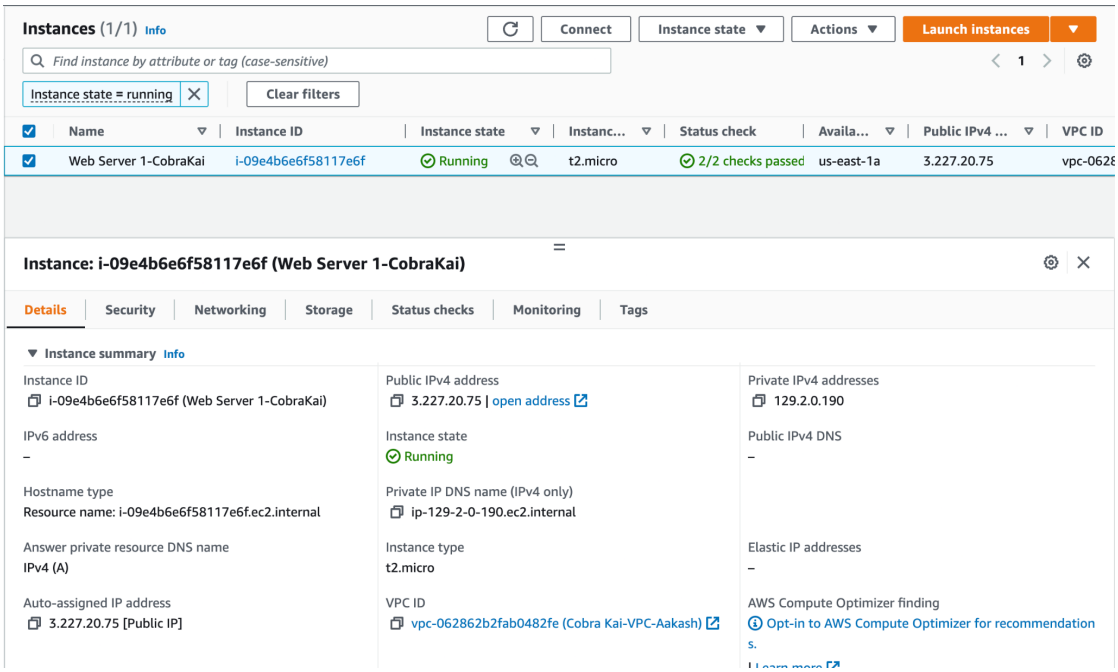
B. Then I added the Name Servers that AWS provided in Freenom.com's Name Servers as shown below.



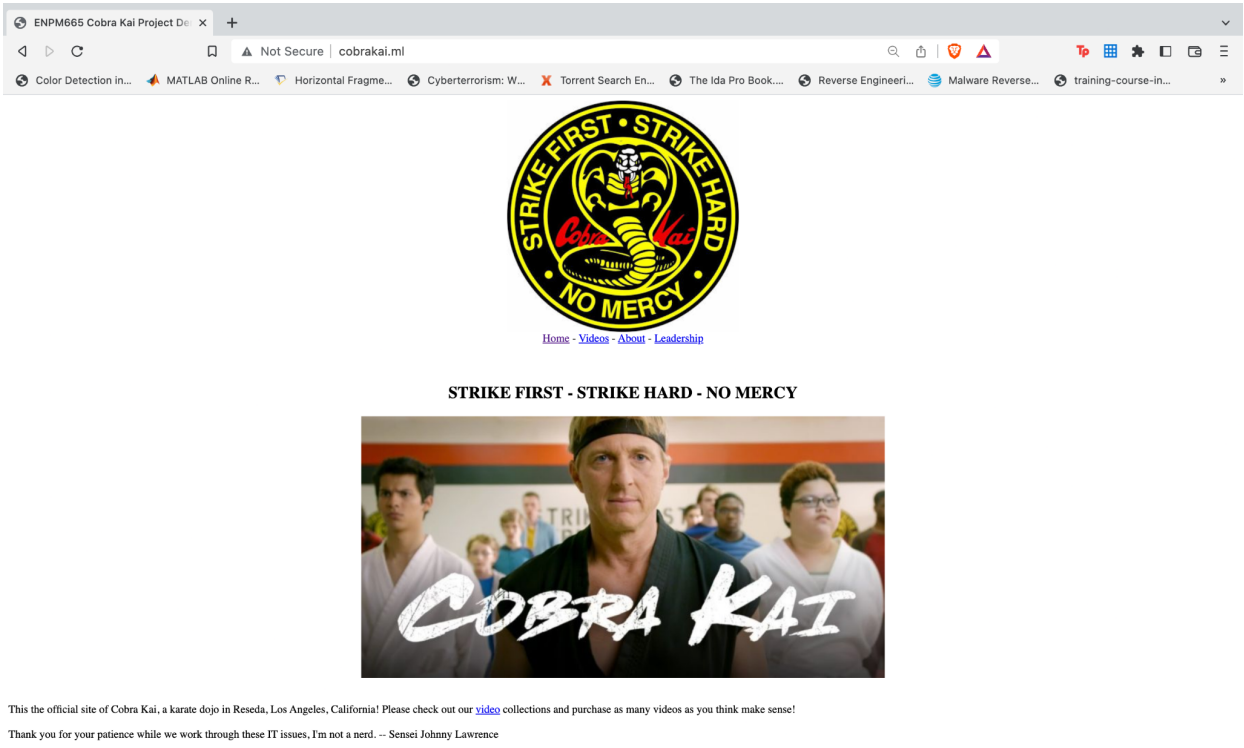
C. Now we have to create an S3 bucket to statically host " www.cobrakai.ml". Along with NS Records, A Records also need to be added to route traffic correctly via Route 53.



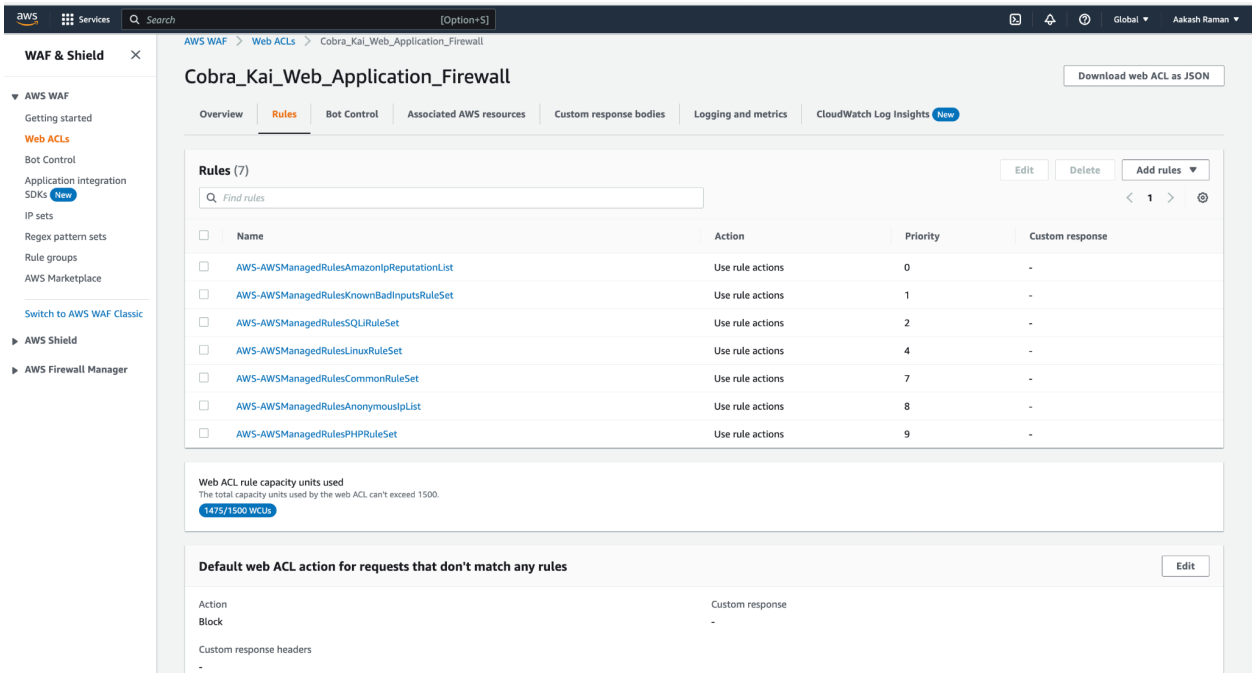
D. This is done by adding Web Server 1's Public IP Address as a Value in the A Record so that traffic will be redirected there (refer to point A). This is depicted by the below screenshot:



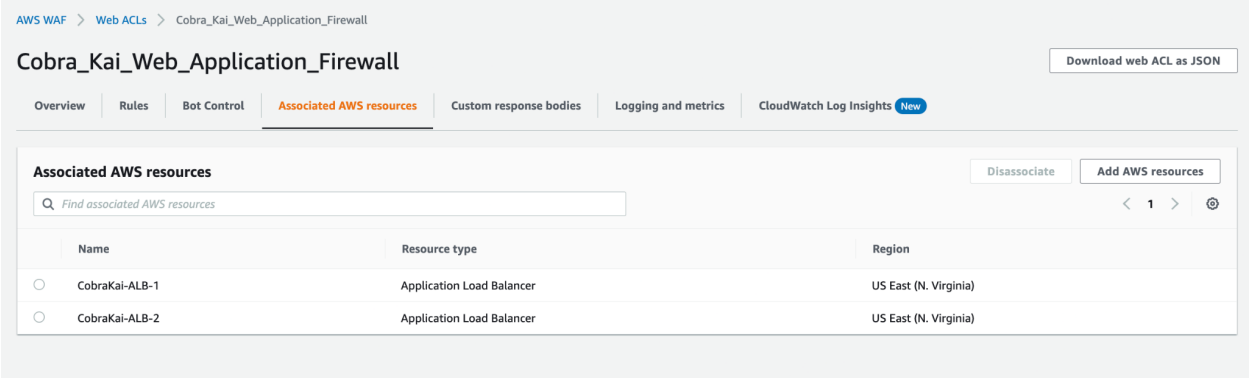
E. After doing this, I managed to host the Cobra Kai website by my domain "www.cobrakai.ml" as shown in the screenshot below.



2. Now, that the static website is hosted, we need to add AWS Rule Sets for the AWS WAF & Shield (Web Application Firewall) so that all common Web Application Vulnerabilities, such as SQL Injection, Cross-Site-Scripting, CSRF, etc are checked for and prevented. The AWS Rule Sets I have added are shown in the below screenshot:



3. The below screenshot shows the associated AWS Resources for the WAF. There are mainly 4 types of resources under AWS WAF, namely, Amazon API Gateway, AWS AppSync, Application Load Balancer, and Amazon Cognito User Pools. In this case, I have chosen our Internet-facing Load Balancers of Cobra Kai.



This way all the traffic entering Cobra Kai will be routed via Route 53 and will have to pass AWS WAF, hence reducing the chances of a DDoS attack.

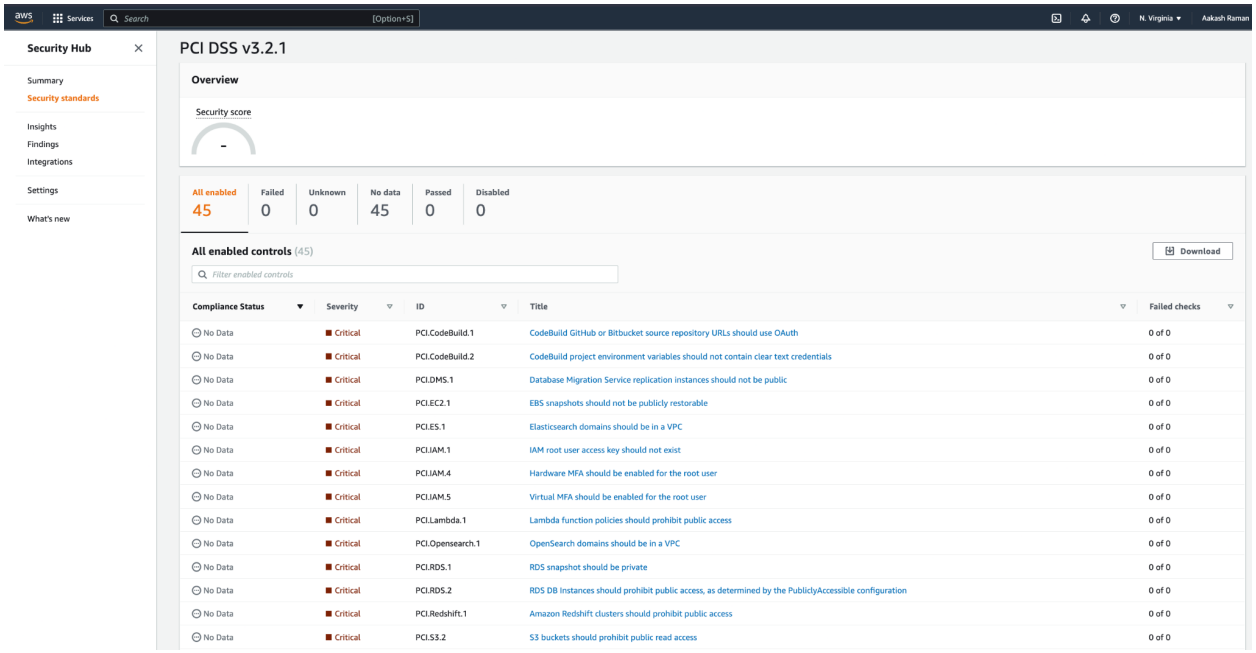
Recommendation for Handling PCI-DSS Compliance using AWS Security Hub:

Since the clients' credit card information is exposed to attacks, the current Cobra Kai infrastructure does not meet PCI-DSS compliance criteria, which is a major problem.

One tool that performs Security Best Practices checks, automated retrieval, and aggregation of logs is AWS Security Hub.

I have set up AWS Security Hub to monitor whether Cobra Kai is compliant with PCI-DSS. To set up AWS Security Hub you need to, enable AWS Config and then set up AWS Security Hub.

The below screenshot shows the output of the PCI-DSS Report generated by AWS Security Hub. Many of the vulnerabilities are Critical, High, Medium, Low & Informational.

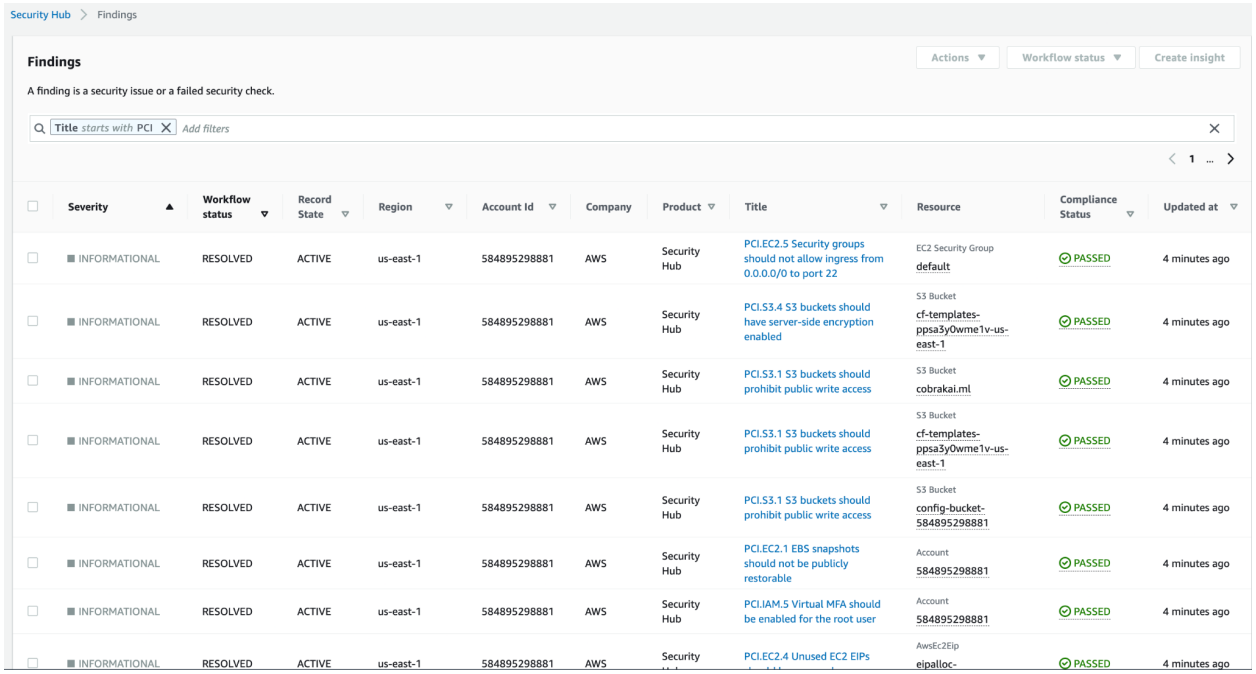


No Data	High	PCI.IAM.3	IAM policies should not allow full "*" administrative privileges	0 of 0
No Data	High	PCI.SSM.1	EC2 instances managed by Systems Manager should have a patch compliance status of COMPLIANT after a patch installation	0 of 0
No Data	High	PCI.SageMaker.1	Amazon SageMaker notebook instances should not have direct internet access	0 of 0
No Data	Medium	PCI.CloudTrail.1	CloudTrail should have encryption at-rest enabled	0 of 0
No Data	Medium	PCI.CloudTrail.3	CloudTrail log file validation should be enabled	0 of 0
No Data	Medium	PCI.Config.1	AWS Config should be enabled	0 of 0
No Data	Medium	PCI.EC2.6	VPC flow logging should be enabled in all VPCs	0 of 0
No Data	Medium	PCI.ELBv2.1	Application Load Balancer should be configured to redirect all HTTP requests to HTTPS	0 of 0
No Data	Medium	PCI.ES.2	Elasticsearch domains should have encryption at-rest enabled	0 of 0
No Data	Medium	PCI.IAM.8	Password policies for IAM users should have strong configurations	0 of 0
No Data	Medium	PCI.IAM.6	MFA should be enabled for all IAM users	0 of 0
No Data	Medium	PCI.IAM.7	Unused IAM user credentials should be removed	0 of 0
No Data	Medium	PCI.KMS.1	AWS KMS key rotation should be enabled	0 of 0
No Data	Medium	PCI.Opensearch.2	OpenSearch domains should have encryption at rest enabled	0 of 0

No Data	Medium	PCI.S3.6	S3 Block Public Access setting should be enabled	0 of 0
No Data	Medium	PCI.S3.4	S3 buckets should have server-side encryption enabled	0 of 0
No Data	Medium	PCI.S3.5	S3 buckets should require requests to use Secure Socket Layer	0 of 0
No Data	Medium	PCI.SSM.3	EC2 instances should be managed by AWS Systems Manager	0 of 0
No Data	Low	PCI.AutoScaling.1	Auto scaling groups associated with a load balancer should use load balancer health checks	0 of 0
No Data	Low	PCI.CloudTrail.4	CloudTrail trails should be integrated with Amazon CloudWatch Logs	0 of 0
No Data	Low	PCI.CW.1	A log metric filter and alarm should exist for usage of the "root" user	0 of 0
No Data	Low	PCI.EC2.4	Unused EC2 EIPs should be removed	0 of 0
No Data	Low	PCI.IAM.2	IAM users should not have IAM policies attached	0 of 0
No Data	Low	PCI.Lambda.2	Lambda functions should be in a VPC	0 of 0
No Data	Low	PCI.S3.3	S3 buckets should have cross-region replication enabled	0 of 0
No Data	Low	PCI.SSM.2	EC2 instances managed by Systems Manager should have an association compliance status of COMPLIANT	0 of 0

AWS Security Hub enables us to check for PCI-DSS Compliance by monitoring Cobra Kai's instances and then using the predefined benchmarks to check if they are compliant or not.

The below screenshot shows how I have set Cobra Kai's Web & App Server instances to follow PCI DSS Compliance benchmarks and they have passed them based on AWS Security Hub's Observation.

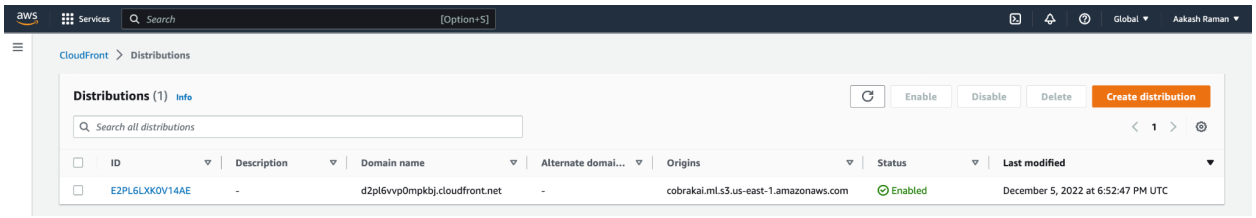


Thus, by incorporating AWS Security Hub into each of Cobra Kai's resources, proper benchmarks can be followed to ensure that PCI-DSS Compliance is maintained regularly.

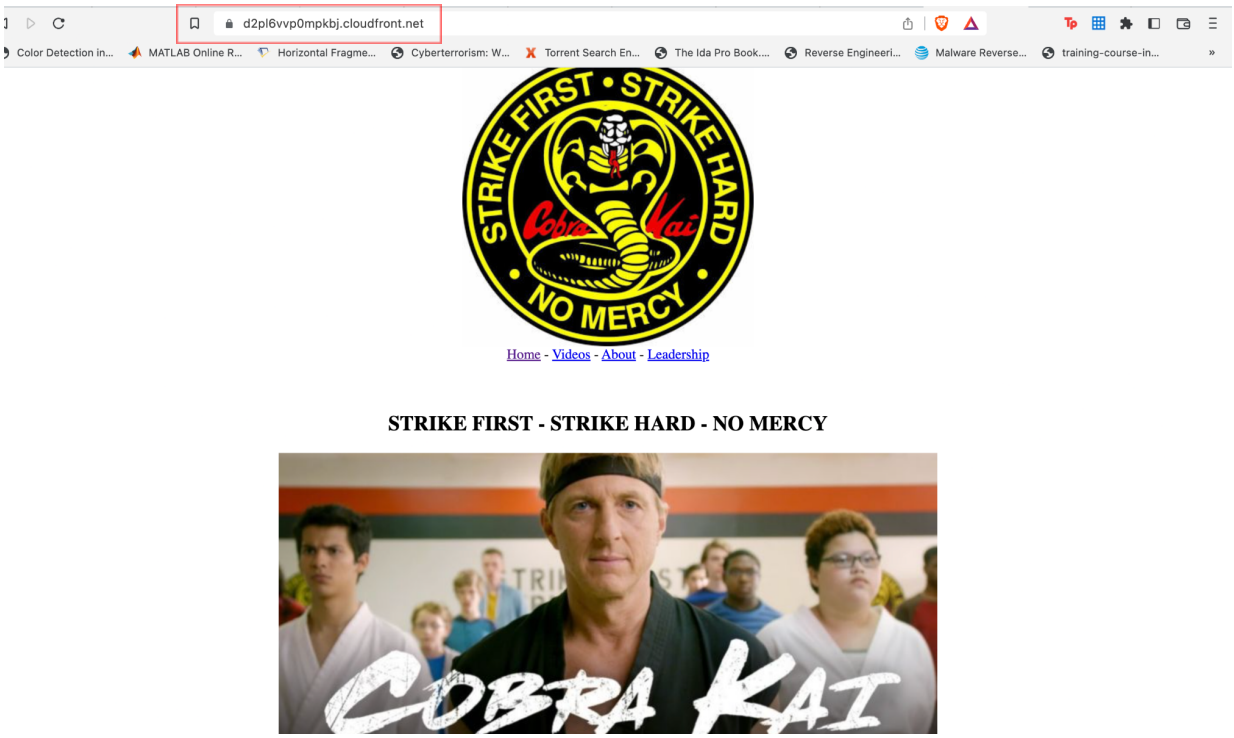
Recommendations for Handling Resiliency & Reduced Network Latency using Amazon CloudFront:

Amazon CloudFront is a Content Delivery Network that provides a global network of proxy servers that cache content which can be videos, audio, images, etc thus reducing the Network Latency and speeding up the delivery of any application to its customers. Logically, incorporating this tool will help Cobra Kai provide its high-demand videos easily and efficiently to its customers.

1. Through Amazon CloudFront, I can access my Cobra Kai website that I deployed in the S3 Bucket securely. The below screenshot depicts the Amazon CloudFront Distribution I used to set up the "www.cobrakai.ml" website.



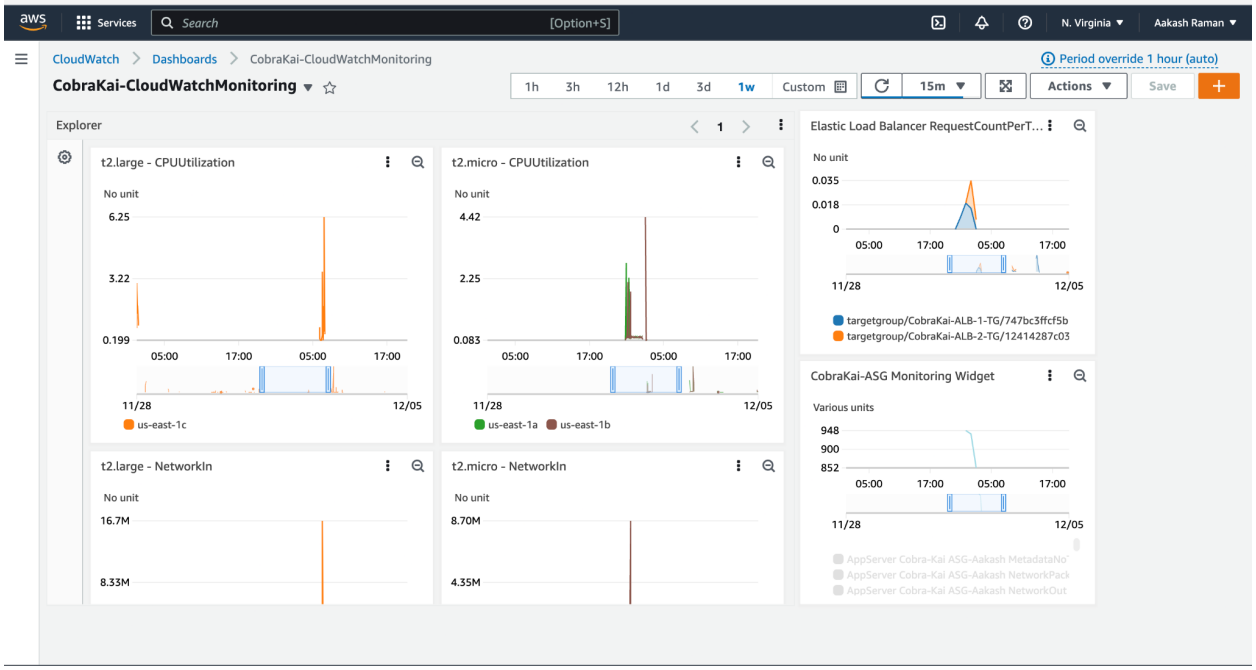
2. By using the domain name from CloudFront distribution, I can load a **secure** "index.html" page of Cobra Kai's website.



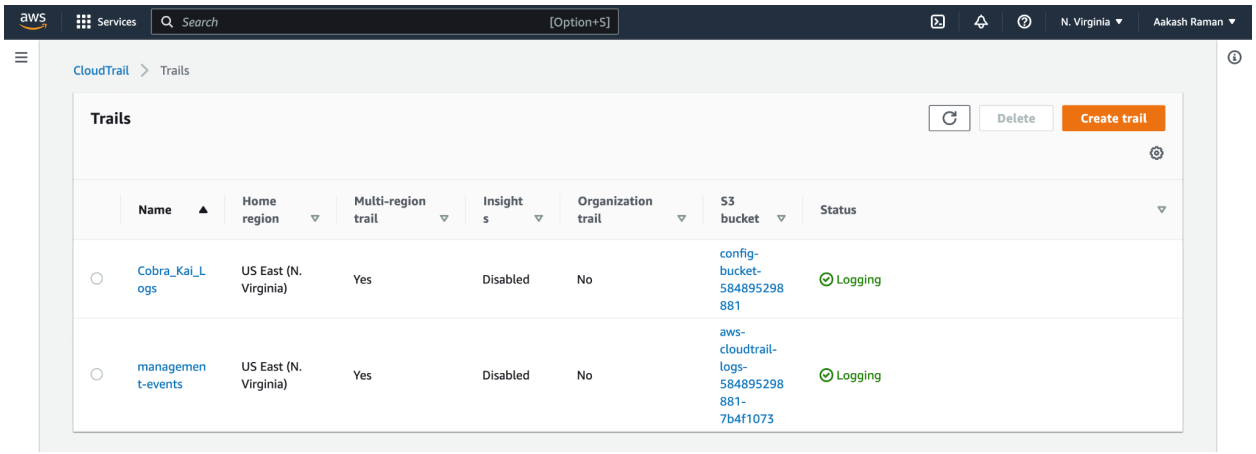
Recommendations for Continuous Logging & Monitoring using Amazon CloudWatch & Amazon CloudTrail:

Amazon CloudWatch is Amazon's continuous monitoring service that provides valuable insights into how resources that are deployed in the Cloud environment are used regularly. Amazon CloudTrail is a similar tool that handles auditing, troubleshooting, and security monitoring by keeping track of a user's activity along with API usage and calls.

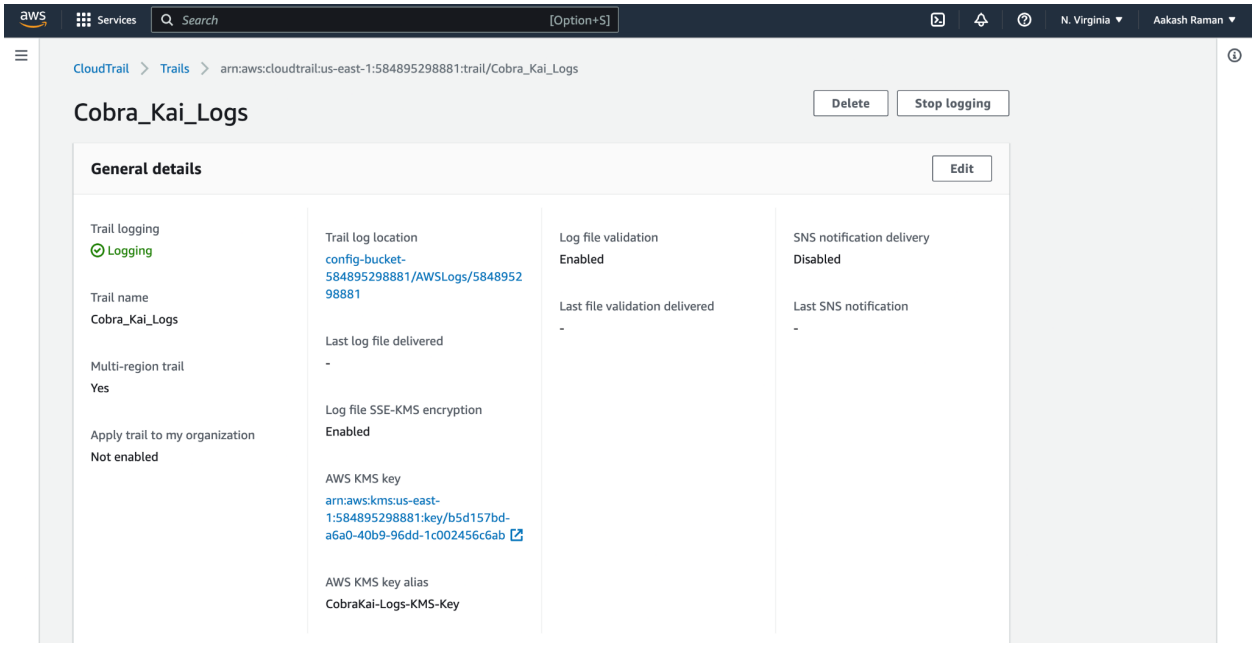
1. To monitor the performance of Cobra Kai's App and Web Servers, I have also configured a Dashboard in Amazon CloudWatch to monitor their network parameters so it can be a centralized location for continuous monitoring of all the critical assets of Cobra Kai.



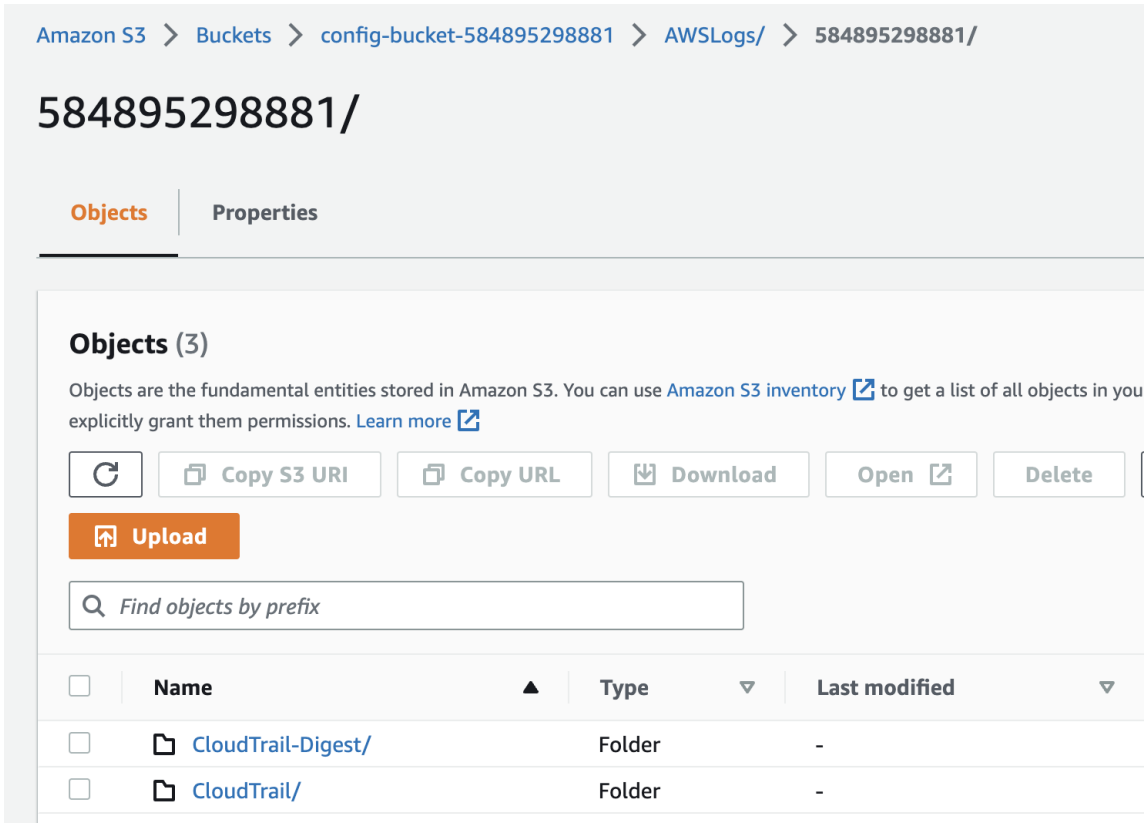
2. Along with the dashboard I created, I have set up logging using AWS CloudTrail Logs that uses KMS (Key Management Service) Encryption where I have created a Customer Managed Key to create a Trail, and the logs are stored in an S3 bucket as shown below in the Trail Log:



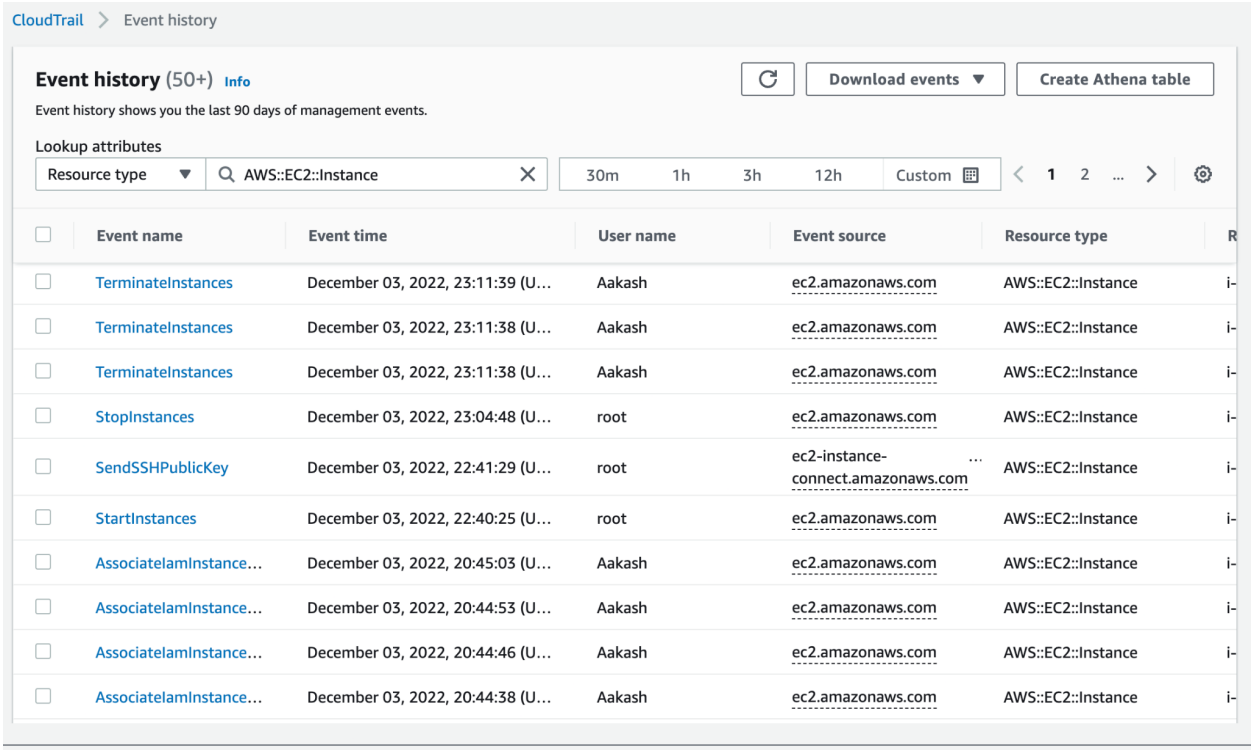
3. The below screenshot depicts the location of the Cobra_Kai_Logs in AWS and it can also be seen that logging is configured and operational for the VPC I set where all the events taken by the users in that VPC get logged. Also, Management & Data Events get logged too.



4. This is the location where the S3 Bucket that we configured for logging is stored.



5. Additionally, as AWS CloudTrail is an auditing tool, all the audits can be seen in the **Event History** section of AWS CloudTrail as shown below:



Hence, Cobra Kai can maintain a constant flow of operations and logs efficiently and the Security team can track in case any attack is waged against their resources too.

Initial Setup of the AWS RDS Replication for High Availability (Relational Database Service)

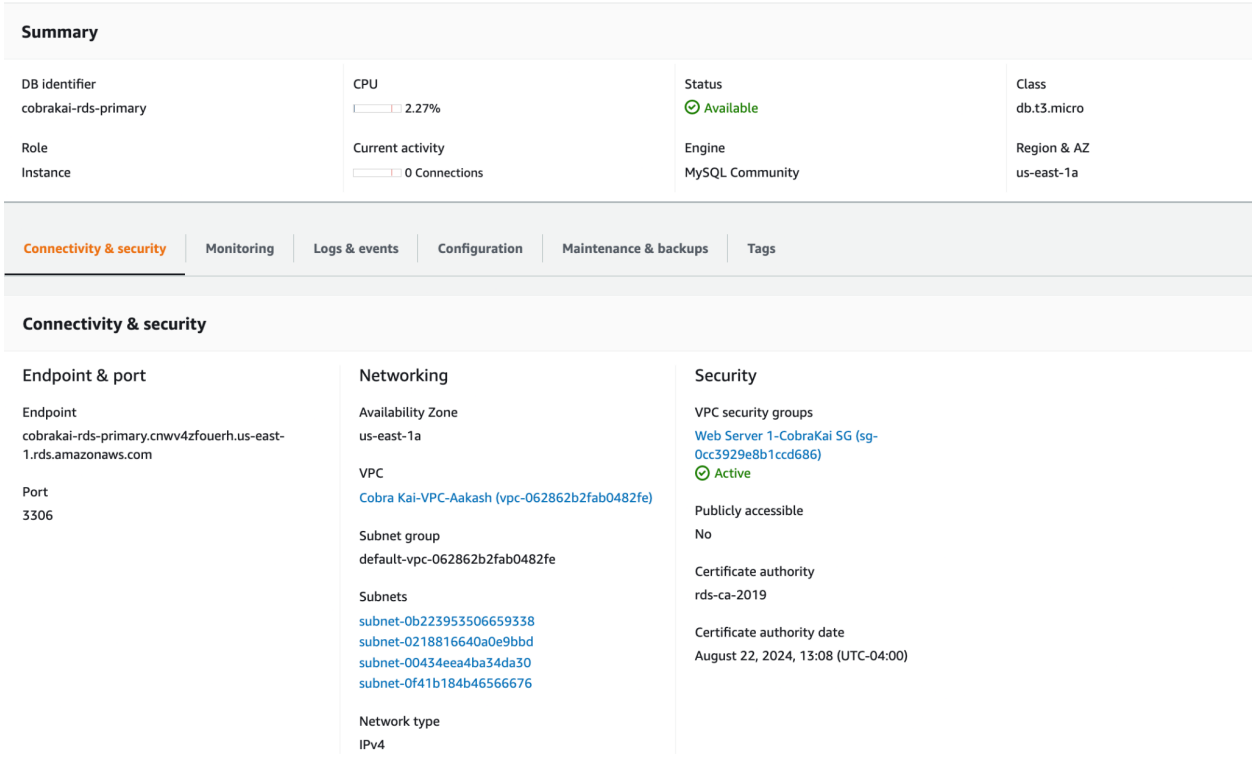
AWS RDS is a Distributed Relational Database Service that Amazon uses for setting up a Relational Database in the Cloud.

As Cobra Kai maintains a Database of all its customers along with their credit card data, I have created a Primary RDS Database and replicated that to another Availability Zone for high availability for Disaster Recovery and Business Continuity.

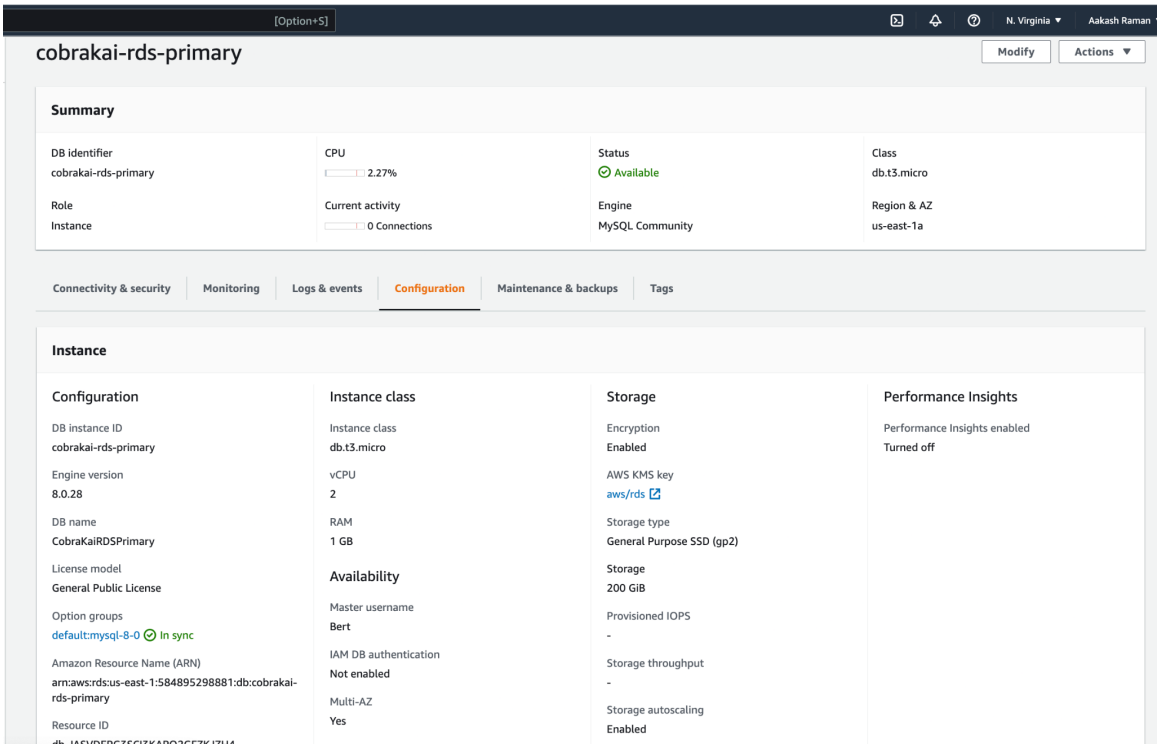
The environment I have created is a **Multi-AZ (Availability Zone)** environment where I have added Bert, the System Administrator of Cobra Kai to have access to the Database. Bert would

need to have constant interaction with the Database to retrieve valuable customer information and perform routine operations. He is a highly technical person too.

I have also connected the Database to our initial Subnets for easy Replication as shown below:



The below screenshot shows the Primary RDS Server that will be replicated in another Availability Zone. I have also encrypted the Primary RDS Server with the **AWS/RDS** KMS Key as shown below.

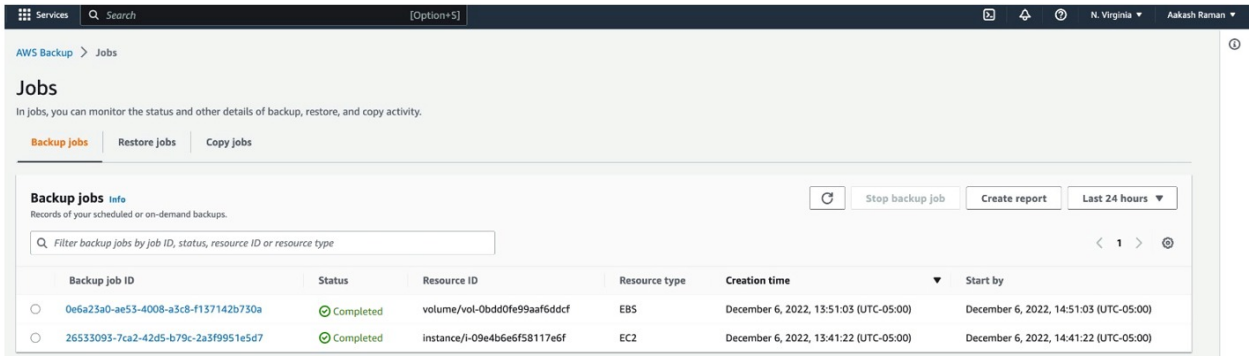


Recommendations for Backing Cobra Kai's Resources using AWS Backups

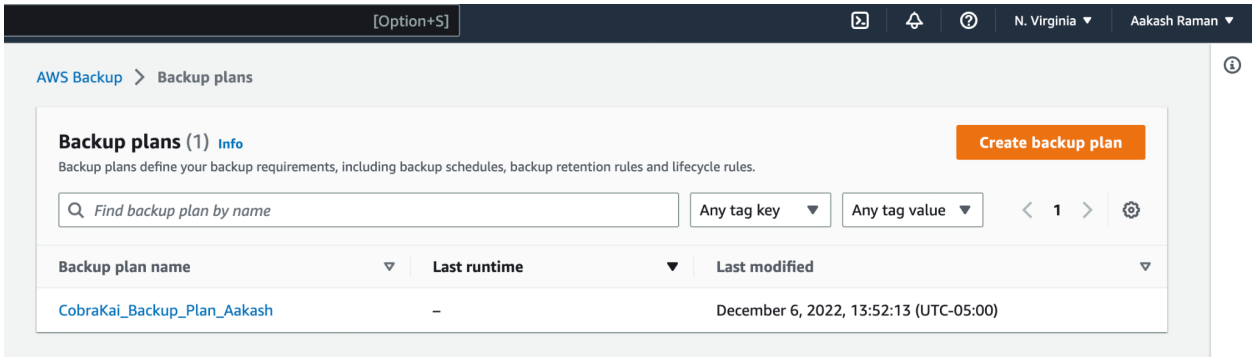
Backing up resources is a strong Security practice and is always recommended when companies face a cyber or natural-related attack. In the Cloud, AWS uses AWS Backups to perform this.

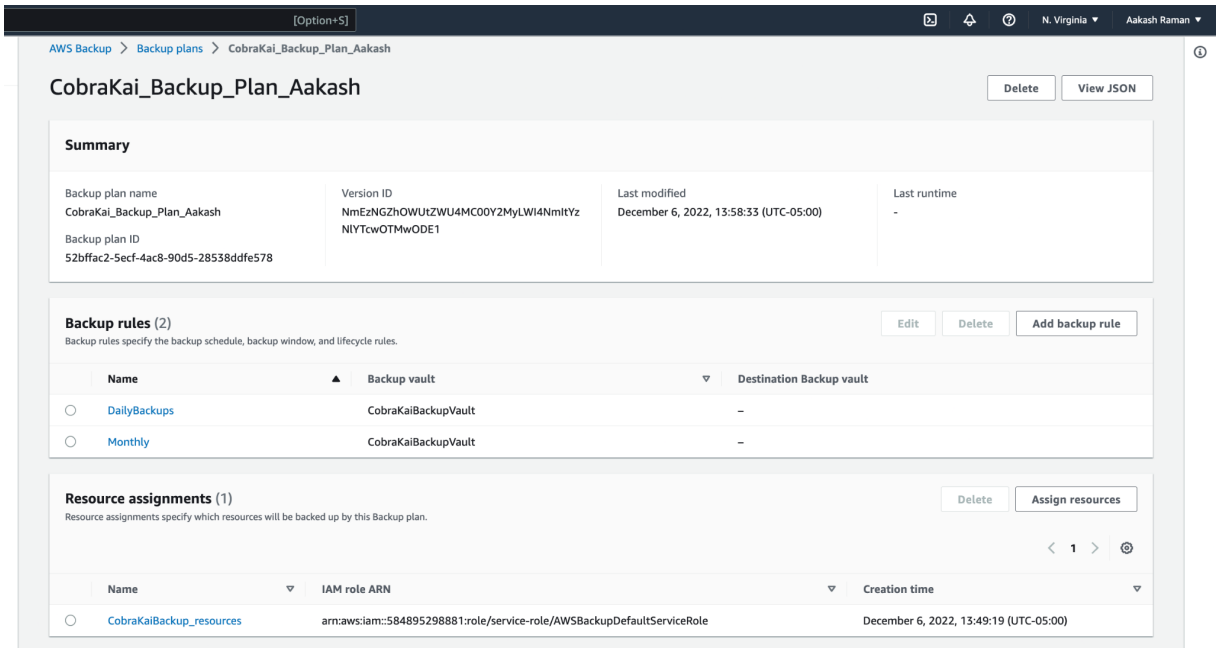
AWS Backups are a cost-effective way of backing up your Cloud resources using a fully-managed, policy-based service that simplifies Data Protection and is a good practice to always maintain high availability.

1. AWS Backups allow the creation of periodic backups of the resources of our choice. I tried this approach on Web Server 1 & App Server 1 of Cobra Kai. I also backed up the EBS Volume, "**vol-0bdd0fe99aaf6ddcf**" as shown by the below screenshot.

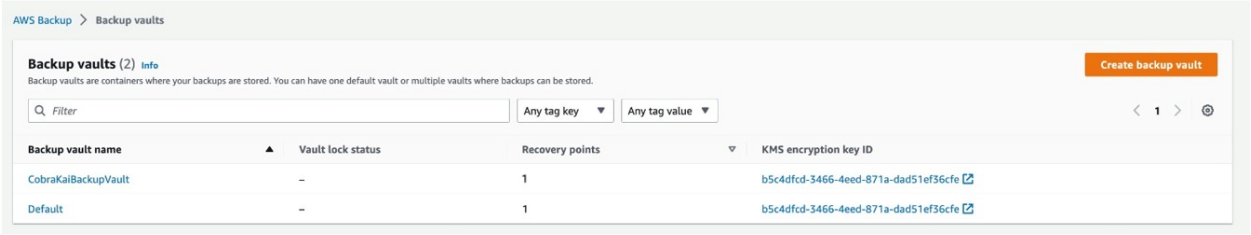


2. The below screenshots show the Backup Plan I have created for backing up the instance's Daily and Monthly Backup rules.





3. To protect the backups that are taken, AWS uses the concept of "**Backup Vaults**". Backup Vaults are encrypted using a **KMS Encryption Key** which is depicted by the screenshot below.



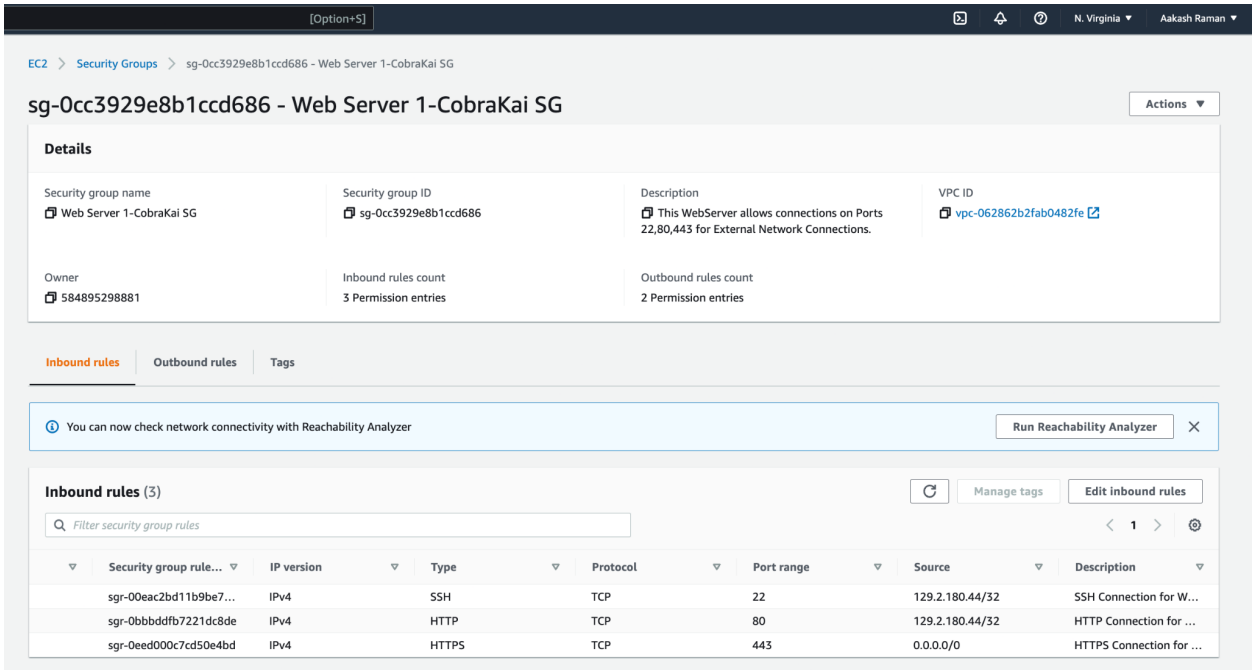
Hence, Cobra Kai can use AWS Backups to back up their resources and encrypt those backups too all in the Cloud.

Recommendations for Handling Network Firewalls using Security Groups & NACLs (Network Access Control Lists)

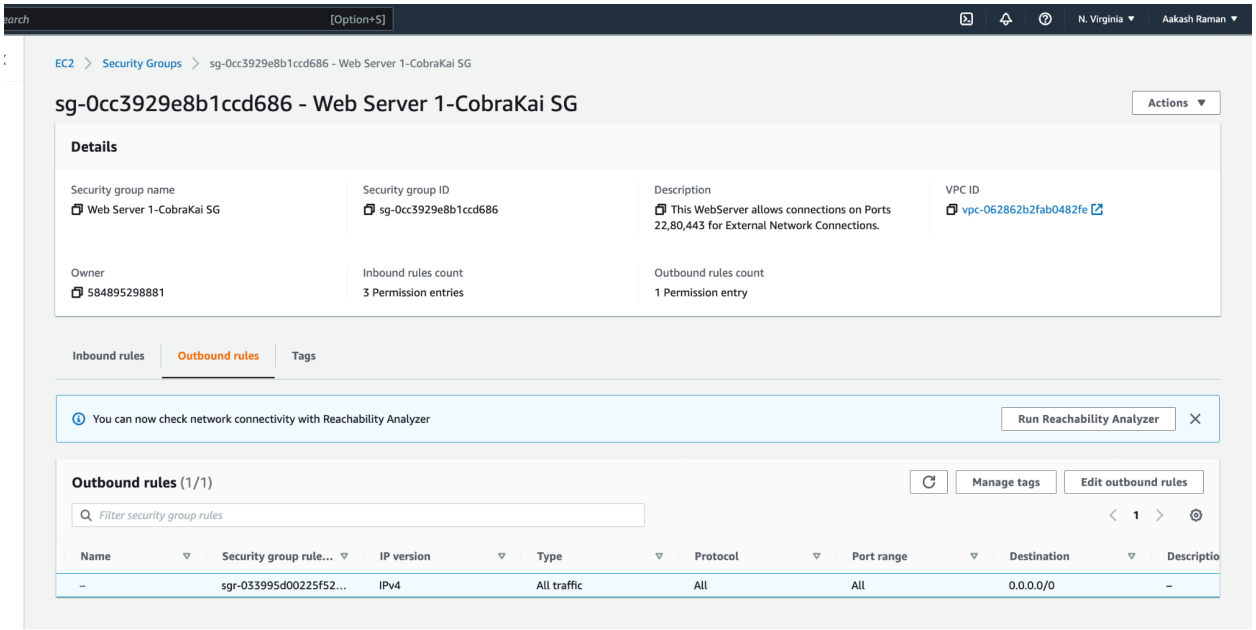
Security Groups help to control the traffic that is allowed to enter or leave that instance. Security Groups are a good way to control traffic as they are the equivalent of a Network Firewall, but work specifically on the Cloud. When you create a VPC, it comes with a default Security Group. They are **implicit**, **stateful**, and **generic** hence it's an easy way to enforce basic Network Security on the Cloud.

I have created the Security Group "**Web Server 1-CobraKai SG**" for all the EC2 Instances.

1. The below screenshot depicts the Inbound Rules for the Security Group. I have added SSH & HTTP connections from only "**My IP**" so that connections are securely made to the Server and are not directly accessible by the Internet. The HTTPS connection is open to the public as it is HTTP over a Secure channel.

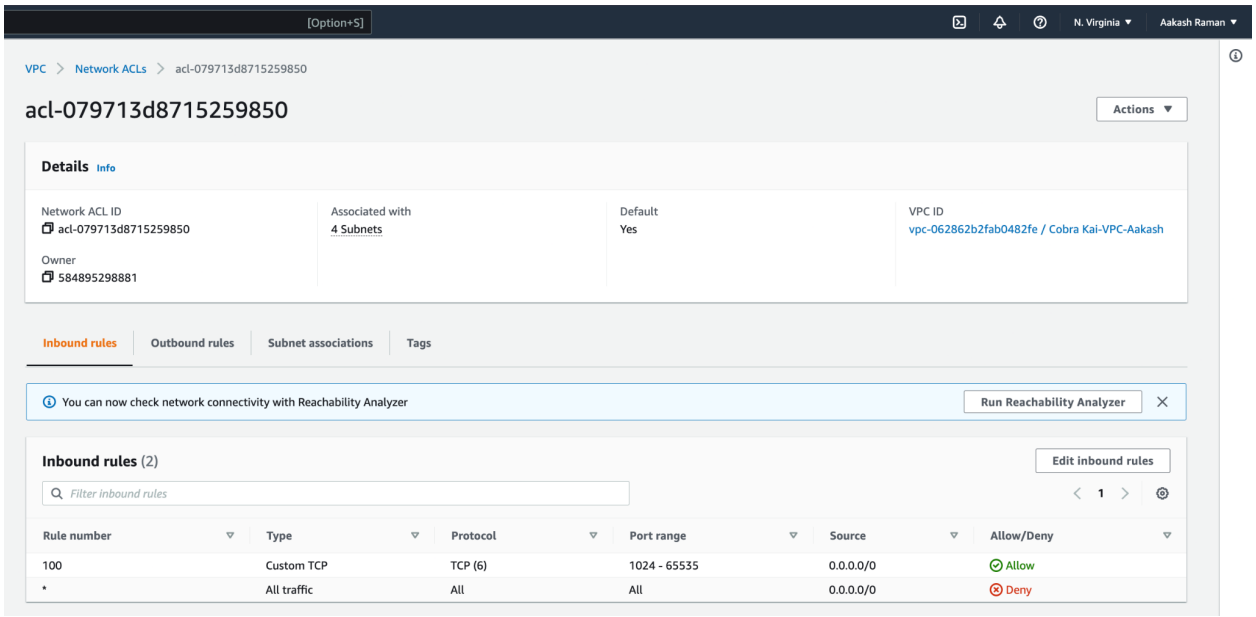


2. The below screenshot shows the Outbound Rules for the Security Group. In this case, I have kept the default Security Group rules where all connections can be broadcasted.

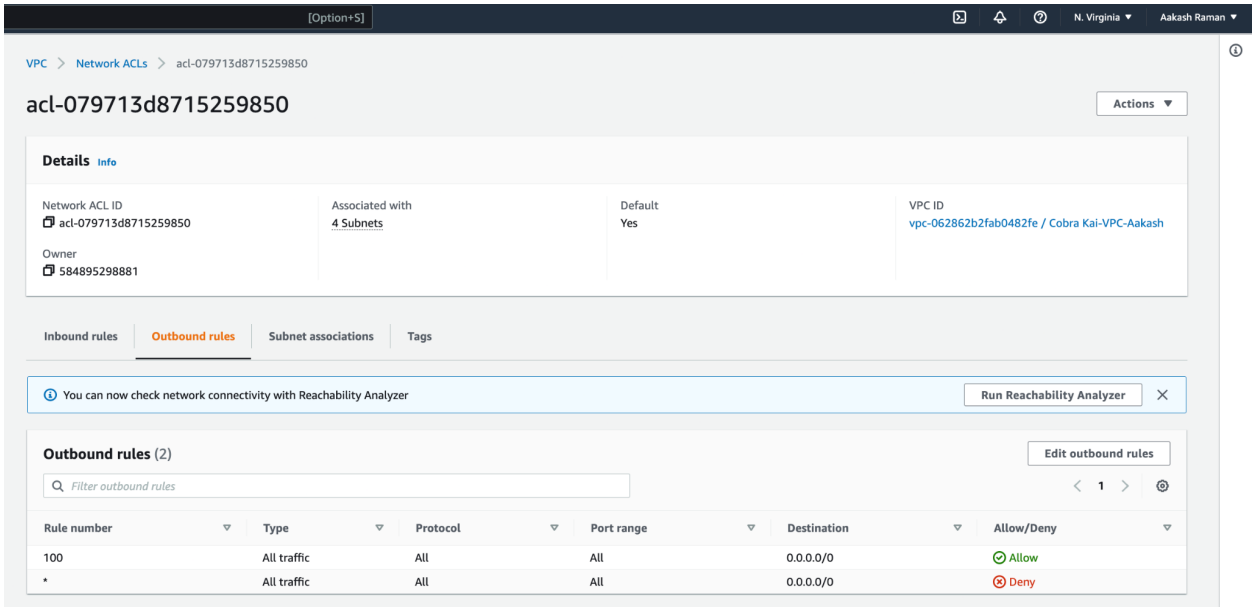


Network ACLs are an optional layer of Security in the Network that controls traffic too. They are **explicit**, **stateless**, and more **specific** than Security Groups. They are also a form of Firewall and follow the "**Implicit Deny Principle**", and the order of Precedence is from **Top to Bottom**.

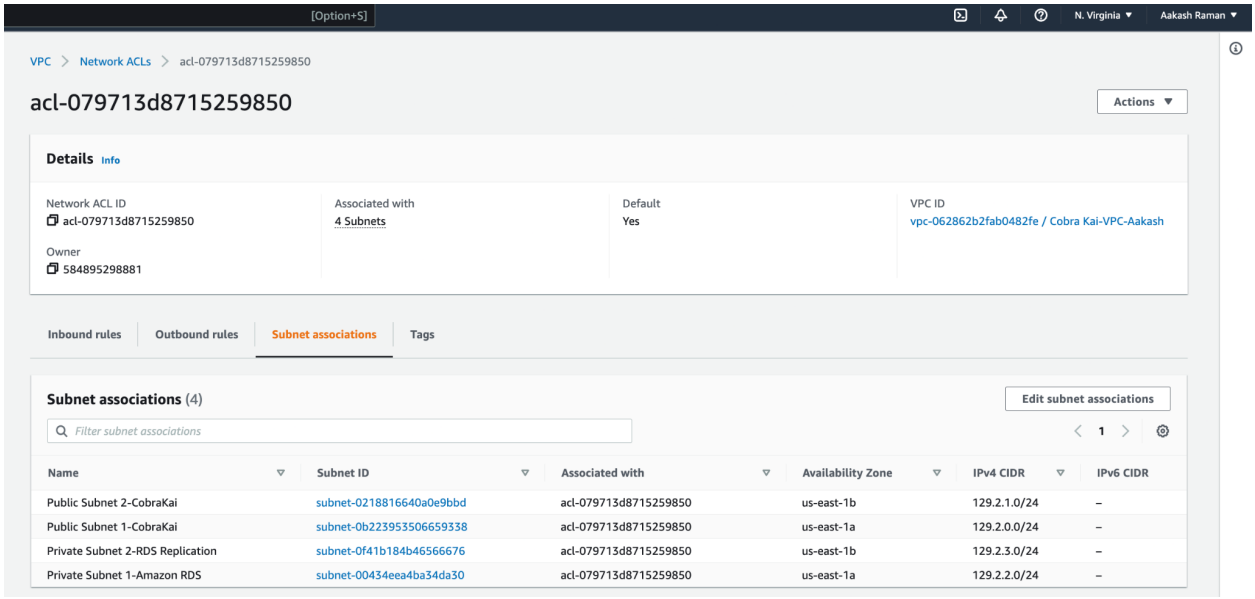
I have configured a NACL and configured Inbound Rules that allow custom TCP traffic over a particular port range from 1024 - 65535.



I have configured Outbound Rules that allow all traffic to leave the network as shown below.



The below screenshot shows the Subnet Associations for the NACL.



Hence, by incorporating Security Groups & NACLs, Cobra Kai can control Network Traffic entering and leaving the Network and also use secure protocols such as SSH to establish Remote Connections. Other protocols, like SFTP, can also be incorporated in the future in the same manner.

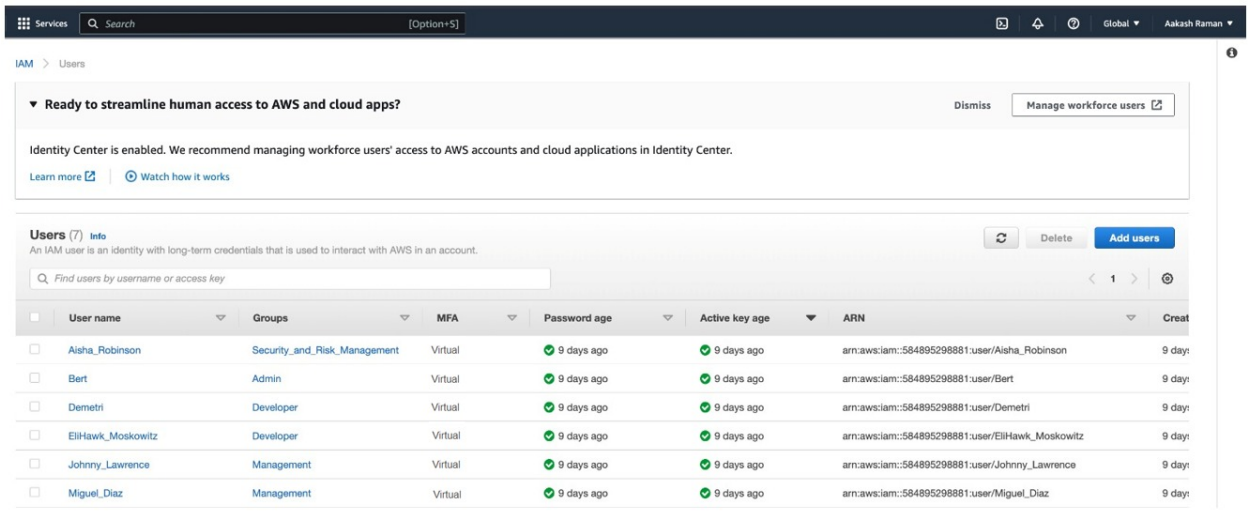
Recommendations for Handling Identity & Access Management

Identity & Access Management is a critical domain for providing security in the Cloud. AWS has an IAM Dashboard that allows creating "**User Groups**", "**Roles**" & attach "**Policies**" to those users. This way the "**Principle of Least Privilege**" will be followed, which is a strong practice both in traditional and Cloud environments.

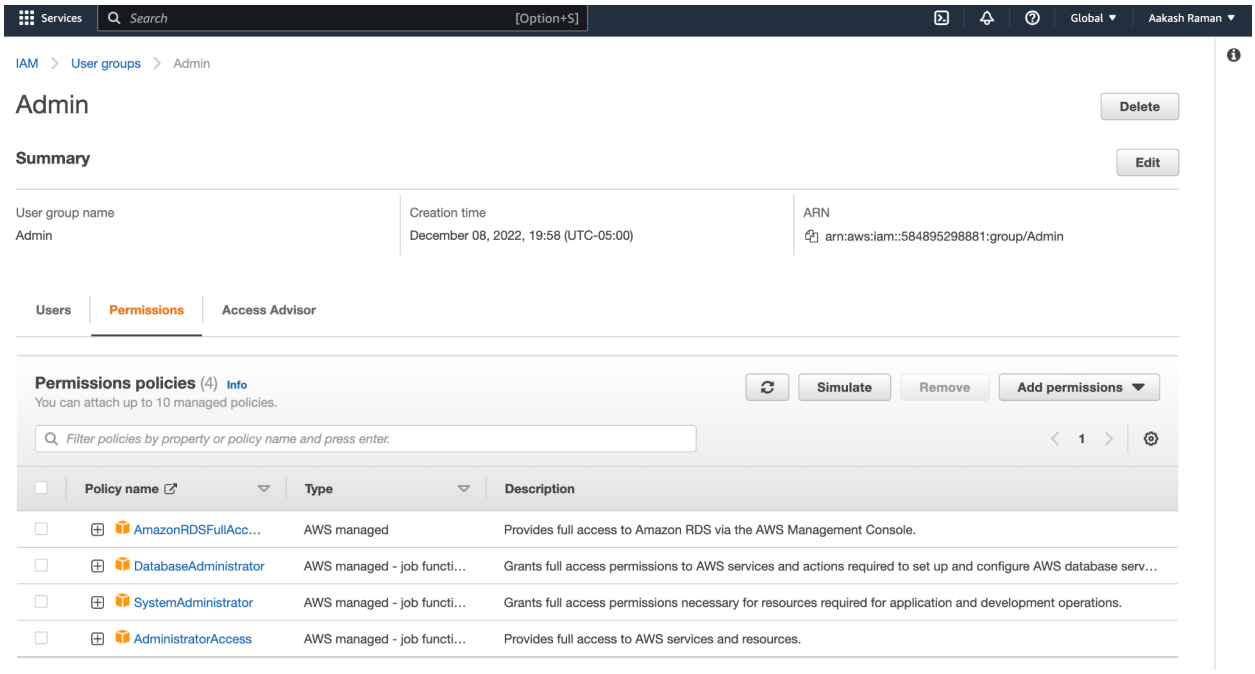
I have created User Groups for all the employees of Cobra Kai as shown below screenshot:
The User Groups include:

- **Admin**
- **Developer**
- **Management**
- **Security_and_Risk_Management**

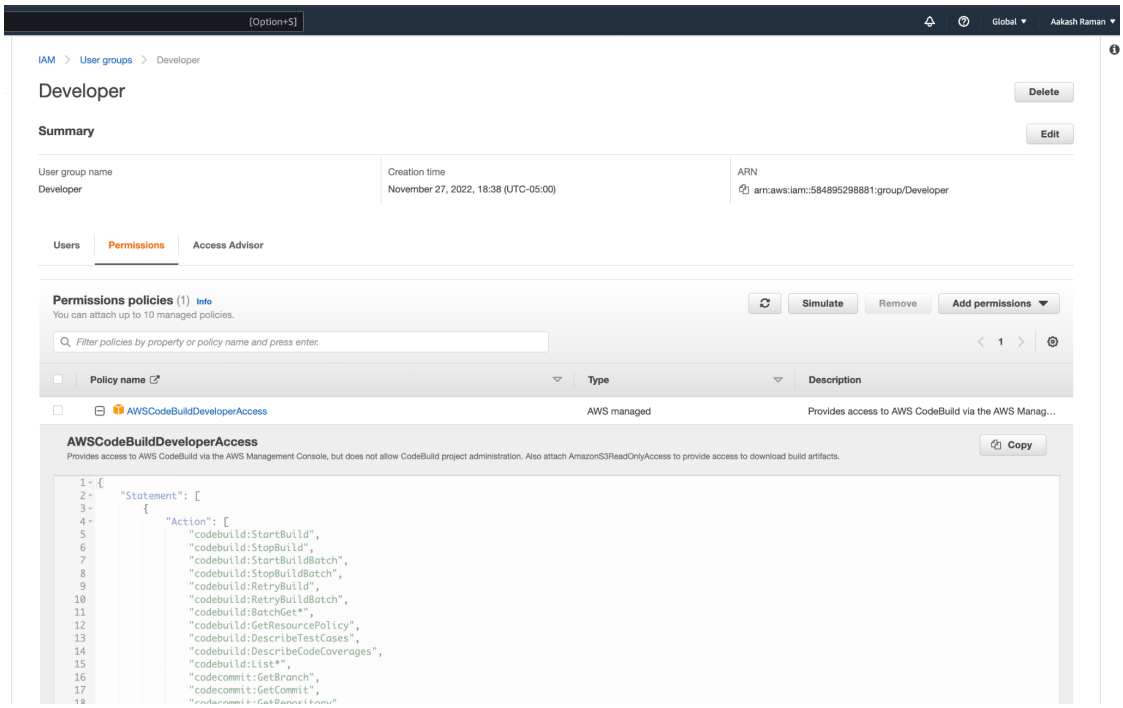
These are shown in the below screenshot:



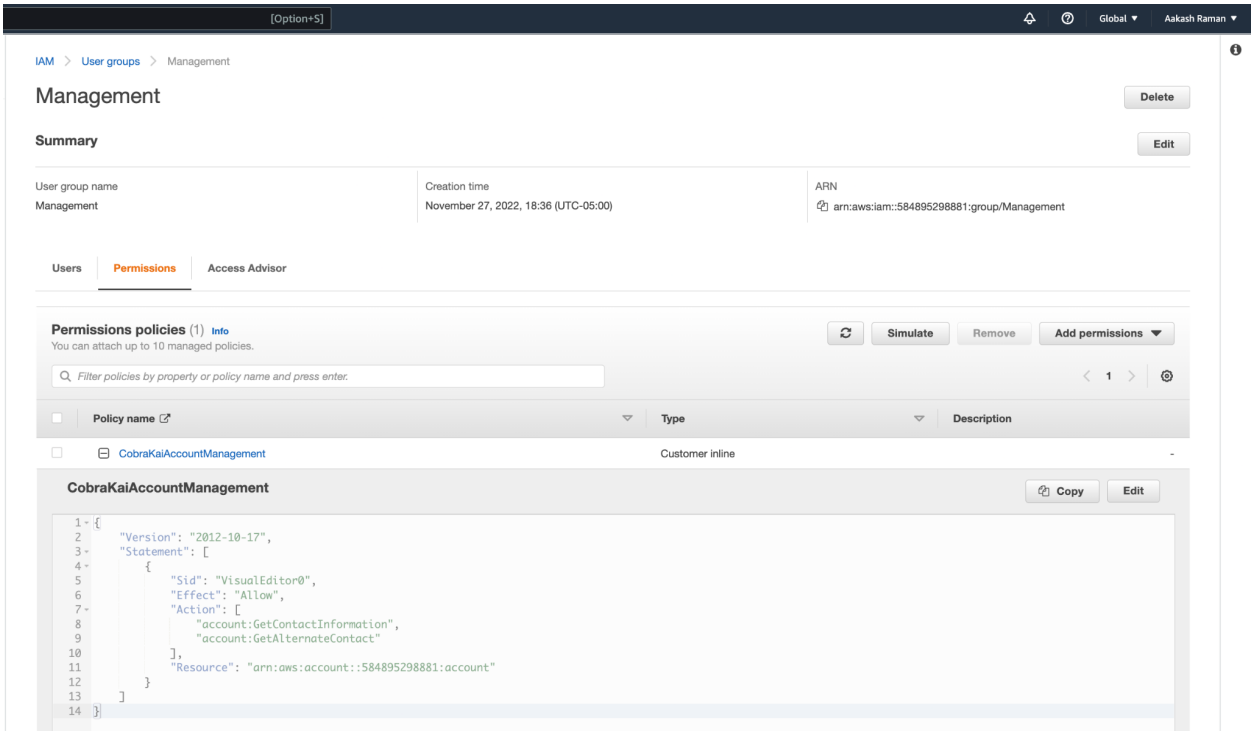
1. I have added **Bert**, the **System Administrator**, to the **Admin** group because he would need to interact with the Databases & Servers regularly along with his technical knowledge. No other employee would require that much access to perform his/her daily functions. He has the policies of "**AdministratorAccess**", "**AmazonRDSFullAccess**","**DatabaseAdministrator**" and "**SystemAdministrator**". This is shown in the below screenshot.



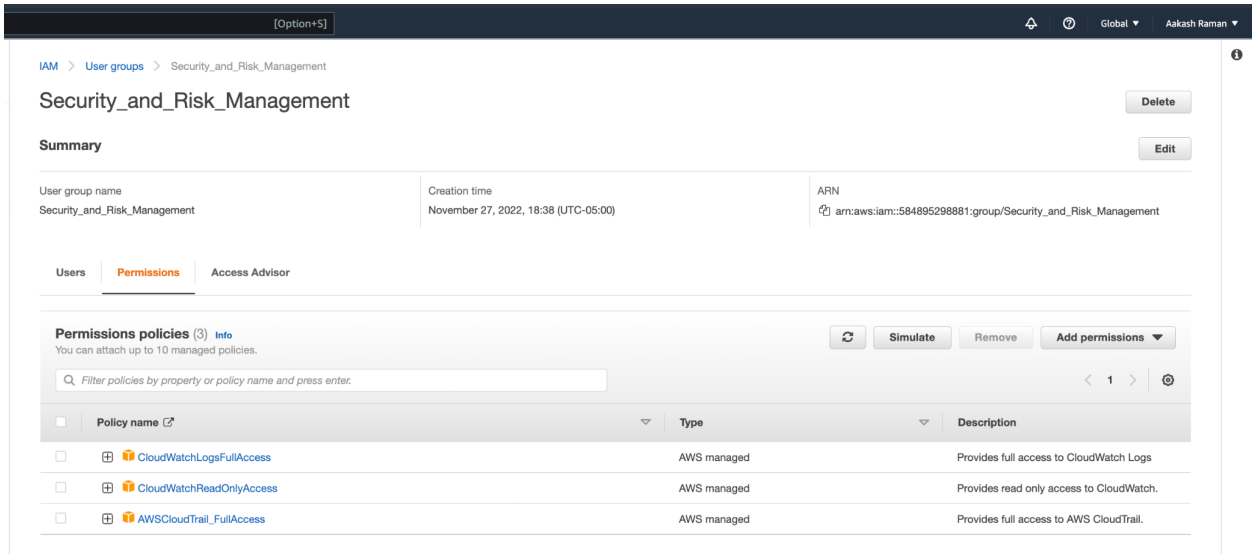
2. I have added **Eli Hawk Moscowitz**, the **Chief Information Officer** & **Demetri**, the **Web Developer** in the **Developer** group, and given them the "**AWSCodeBuildDeveloperAccess**" because both of them require access to a Development environment to perform their daily activities.



3. I have added **Johnny Lawrence**, the Chief Executive Officer, and **Miguel Diaz**, the Chief Operating Officer in the **Management** group and they have the policy of "**CobraKaiAccountManagement**". As these are C-Level executives they are kept in a common group and also they are Non-Technical but must be in charge of knowing who has how much access to data.



4. I have added **Aisha Robinson**, the Chief Information Security Officer (CISO) in the **Security_and_Risk_Management** group and she has 3 policies, namely, "**CloudWatchLogsFullAccess**", "**CloudWatchReadOnlyAccess**" and "**AWSCloudTrailFullAccess**". This is because the CISO should have full access to the logs and dashboards of CloudWatch and CloudTrail but need not perform any setup or implementation with those logs. This is shown in the below screenshot.



By following these practices, Cobra Kai can implement strong IAM Policies within their Cloud environments to prevent both insider threats and maintain proper audits of all their user's actions.

Conclusion

Overall, the process of designing all the required recommendations and the architecture in the AWS Cloud Environment can be a bit challenging for Cobra Kai, especially during its initial stages, but almost all of its problems can be easily solved in a very small time frame. Hence, by following a proper Security Baseline, Best Practices, and Compliance Frameworks, Cobra Kai can truly leverage the Cloud to its full capacity and boost its business!

References

These are the references I have used for this Project in APA Format:

- AWS, A. (n.d.). *AWS Management and governance tools workshop*. AWS Management and Governance Tools Workshop. Retrieved December 8, 2022, from https://mng.workshop.aws/ssm/use-case-labs/inventory_patch_management/patch.html
- Team, W.-A. (n.d.). *Cloudfront with S3 Bucket Origin*. CloudFront with S3 Bucket Origin :: AWS Well-Architected Labs. Retrieved December 8, 2022, from https://www.wellarchitectedlabs.com/security/100_labs/100_cloudfront_with_s3_bucket_origin/
- Ribeiro, A. (2021, July 4). *Build a serverless website using Amazon S3 and Route 53*. Medium. Retrieved December 8, 2022, from https://towardsdatascience.com/build-a-serverless-website-using-amazon-s3-and-route-53-c741fae6ef8d
- Team, W.-A. (n.d.). *Create CloudWatch dashboard*. Create CloudWatch Dashboard :: AWS Well-Architected Labs. Retrieved December 8, 2022, from https://www.wellarchitectedlabs.com/performance-efficiency/100_labs/100_monitoring_linux_ec2_cloudwatch/3_creating_cloudwatch_dashboard/
- KnowledgeHut. (2020, January 2). *AWS relational database- how to create RDS instance in 11 easy steps*. Knowledgehut. Retrieved December 8, 2022, from https://www.knowledgehut.com/tutorials/aws/creation-of-rds-instance
- Conn, M., Conn, M., Conn, M., Conn, M., Conn, M., Conn, M., Conn, M., Conn, M., Conn, M., & Conn, M. (2021). *Getting started*. Amazon. Retrieved December 8, 2022, from https://aws.amazon.com/getting-started/hands-on/amazon-ec2-backup-and-restore-using-aws-backup/