**Virtual Private Cloud Design using Amazon Web Services**

Virtual Private Cloud (VPC) Design

Objective: Create a private network within a cloud environment that simulates a data center’s network architecture.

Tools: AWS

Purpose: Mimic a real-world private cloud environment where you control the network routing, IP addressing, and security policies.

Documentation: Document the network topology, IP addressing, and firewall rules. Include an explanations of routing and access control policies.

The Layout:

Network Topology Overview (Virtual Private Cloud, Subnets, Internet gateway, NAT Gateway/Instance, Route Tables, Security Groups and Network ACLs),

IP Addressing, Route tables, Security Policies and Access Control Policies, conclusion

Note: when working on an instance, make sure to go to action and select stop instance to avoid excessive billing charges when you are not working on AWS.

1. **VPC Creation:** the first step is to create a VPC utilizing a CIDR Block IP address of 10.0.0.0/16 by going to AWS Management Console and select VPC Dashboard then configure the settings to be tenancy. Under VPC settings: select VPC only, set IPv4 CIDR Bock manual input at 10.0.0.0/16 and select No IPv6 CIDR Block and select Tenancy to Default. Ignore name Tag and tags.
2. **Subnet Creation within VPC:** begin by going to Subnets from the dashboard, create subnets by setting the ipv4 CIDR Block for Public Subnet (App Tier) at 10.0.1.0/24 , then set Private Subnet (DB Tier) at 10.0.2.0/24 and set **Private Subnet (Internal Services)** at 10.0.3.0/24 Then for each subnet make sure to specify them to their own unique availability zone while all subnets need to be linked to the same VPC configuration. 1st subnet 10.0.1.0/24 can have an availability zone of US EAST (N. Virginia) / us-east-1a then set subnet name as Public Subnet (App Tier). Next create the 2nd subnet by selecting add new subnet. Set the name as **Private Subnet (DB Tier)**, choose a different availability zone as US East (N. Virginia) / us-east-1b, set IPV4 VPC CIDR Block as 10.0.0.0/16 and the IPv4 Subnet CIDR Block as 10.0.2.0/24. Next select Add new subnet to create the 3rd subnet. Set the subnet name as **Private Subnet (Internal),** Set availability Zone as US East (N. Virginia) / us-east-1c then set IPv4 VPC CIDR Block as 10.0.0.0/16 and the IPv4 subnet CIDR Block as 10.0.3.0/24. After all, 3 subnet configurations are done, select create subnet. There should be a green prompt that says you have successfully created 3 subnets. The IPv4 subnet addresses should fall within the IP range of 10.0.0.0/16 of the VPC. Another thing to note is that subnet CIDR blocks can not overlap with each other since they need to be distinct and stay within range of the VPC’s IP. Additional subnets can be made to divide the /16 VPC further, such as 10.0.5.0/24 or 10.0.6.0/24, those subnet IP can be reserved for other services, tiers and/or regions if needed.
3. **Internet Gateway Setup:** The 3rd step is to create an Internet Gateway Setup then attach it to the VPC. The purpose of this gateway is to ensure that the internet can access public facing resources (Example: web services). Begin by navigating to the VPC Dashboard from AWS Management Console. On the left sidebar from Virtual Private Cloud, Select Internet Gateways, and choose Create internet gateway, set tag name IGW and select create. Go to Internet Gateways page, where IGW is listed, select the created IGW from the list, press Actions and select Attach to VPC, under the drop down menu select Attach. To verify if IGW is attached to VPC, go to the Internet gateways section and select Attached status next to the IGW. To begin updating the routing tables for public subnet in order to route traffic to the internet via the internet gateway, start with selecting route tables from the VPC dashboard. Identify the route table that associates with the public subnet by looking at the subnet associations. Select the route table and edit it by setting destination to 0.0.0.0/0 (represents all IPv4 traffic) and the Target to be Internet Gateway (IGW) from dropdown list and select save routes. To verify the configuration, begin by launching an EC2 instance in public subnet, assign it a public IP address, try to ping any external website to make sure the EC2 instance is able to access the internet. Summary: Internet Gateway is attached to VPC, routing table is updated, all resources in the public subnet should be able to access the internet. AWS may create a 2nd Internet gateway, 2nd route table and 2nd VPC that you did not manually create. It is best to ignore them since they were not factored in the process of the VPC design. If this happens, find the correct route table that has the proper VPC ID that matches and is associated with the one previously manually created. That VPC should contain the 3 subnets with tag names created and the availability zone they are designated in. Back to the original track, under the correct route table that is linked to the correct VPC, select routes and ensure the destination is 10.0.0.0/16 then check subnet associations and find the 3 subnets with their own IPv4 CIDR Blocks that was previously created with the original VPC, they should be listed under subnets without explicit association. The next step is to edit routes, “add new route”, set the 2nd route destination as 0.0.0.0/0 for all internet traffic and target as the internet gateway, select the IGW that you have created and select save route. The destination 0.0.0.0/16 route indicates it routes internal traffic within the VPC. The creation of the 2nd route within the route table allows instances within the public subnet to communicate with the internet. In summary, public subnet should be associated with public route table and the private subnet should be associated with private route table. The private configurations should be done in the later steps.
4. **Route tables Configuration:** The primary goal is to create a separate route table for both public subnet and private subnets. The Public subnet should be routed to the Internet Gateway by 0.0.0.0/0 and the private subnets should be routed to the NAT (Network Address Translator) Gateway by 0.0.0.0/0. To begin this process, start by selecting Route tables under VPC Dashboard and select Create route table. The 1st table should be named Public-Route-Table, select the VPC that was created from previous steps (ignore the default one created by AWS) and select create to create the route table. In the route table section, select create route table again, name the 2nd table as Private-Route-Table, select the same VPC that was originally created by you and select Create to create the private route table. The 1st table (public route table) needs to be configured by going to the route tab, select edit routes, add route, set destination as 0.0.0.0/0 in order for internet bound traffic to occur and set target as the Internet gateway that you previously created and select save routes. Both public and private route tables should have those 3 subnets listed under subnets without explicit associations. Select private route table under route table dashboard, go to routes tab, select edit routes, add new route, set destination as 0.0.0.0/0 for internet bound traffic and set the target as NAT Gateway (NAT Gateway will be created later) and select save routes. Before configuring that part, you need to create a NAT Gateway and an Elastic IP. Go back to EC2 Dashboard, select Elastic IPs under Network & Security. Then select Allocate Elastic IP address and click allocate in order to generate the new Elastic IP. Once that is done, the NAT Gateway can be created. Go back to VPC dashboard and select NAT gateway, click create NAT Gateway. Then within that menu, under subnet select public subnet (Example 10.0.1.0/24) as the NAT Gateway. This grants internet access for the NAT Gateway via the Internet gateway. For connectivity type: select Public, then under Elastic IP Allocation, select the Elastic IP that was created from previous steps and then select Create NAT Gateway. Doing this will create the NAT Gateway and assign the Elastic IP towards it. Go back to the route table dashboard and select Private Route table and go to routes tab and select edit routes. Select add new route, set destination to 0.0.0.0/0 to ensure all outbound traffic occurs and set the target to be the NAT Gateway that you just created previously and hit save routes. The public subnet is configured to have the internet gateway routed at 0.0.0.0/0 via IGW and the private subnets are configured to have the NAT gateway routed at 0.0.0.0/0 via NAT Gateway along with the NAT ID of the NAT Gateway that was created in previous steps. Finally, ensure that the private subnets (10.0.2.0/24 for DB and 10.0.3.0/24 for Internal services) are explicitly associated with the private route table, in order to allow private resources (like databases) are routing their own internet bound traffic through NAT Gateway, if there are any associations with a public subnet, remove them from the explicit subnet associations list. Also, head to the public route table, under subnet association, ensure that the public subnet (10.0.1.0/24) is explicitly associated with the public route table, this allows any public facing resources such as web servers, to use the internet gateway in order to access the internet and make sure that no private subnets are associated with the public route table, if so remove them from the explicit subnet association. As a result: public route table has public subnet (10.0.1.0/24) listed as explicit subnet association and the private route table has Private subnet (10.0.2.0/24 and 10.0.3.0/24) listed as explicit subnet associations only. Finally, verification: public subnets should access the internet via Internet Gateway, Private subnets can access the internet via the NAT Gateway and the private subnets cannot be directly accessed from the internet. Launch an EC2 Instance in the private subnet to verify it is able to access the internet. Begin by heading over to EC2 dashboard in AWS management console, select launch instance to create an EC2 Instance, select an Amazon Machine Image (example Amazon Linux, Ubuntu, etc), select the Instant Type (example t2.micro for free tier), architecture should be 64-bit (x86), configure instance for Network (select VPC that you created), Subnet (select either private subnets) and disable auto-assign public IP since a public IP is not needed for a private subnet. Set storage setting to be default to maintain free tier eligibility, then for configuring security group, create one that allows inbound traffic from the public subnet (add inbound rule of port 22 as in SSH) to be able to connect an instance from the public subnet. The Inbound rule Configuration should be Type: SSH, Protocol: TCP, Port Range: 22, Source Type Custom, Source 10.0.1.0/24. The Inbound rule is made to ensure that EC2 instance in the private subnet will receive SSH traffic from the public subnet or the machine.

Security groups should be able to allow all outbound traffic (set to 0.0.0.0/0) by default, doing this will make sure that the private EC2 instance is able to access the NAT Gateway, in order to access the internet. There should not be any need to make changes to the default settings of the outbound traffic behavior. Do not alter any settings under Advance network configuration since that is not needed within the scope of this testing. Review all configurations before launching and create a key pair for the SSH into the instance. The next step is to ensure the security group is attached to the EC2 instance. Begin by launching the EC2 instance under the private subnet and attach the security group from there. (Unless the instance is already running, attach the security group by selecting the instance’s security groups section under the EC2 console). Begin verifying if the instance is able to access the internet. The private subnet should route all internet traffic through the NAT Gateway. In order to verify if the instance can access the internet, you can either ping an external IP or try accessing a website. Use an SSH from an instance in the public subnet or your own machine if it allows SSH access. Use the private key associated with the EC2 instance to be able to connect. It should be something like SSH -I /path/to/your/keypair.pem ec2-user@<private-instance-private-ip> . after logging into the EC2 instance, begin by pinging an external ip address to ensure that there is internet access (example: ping 8.8.8.8), if there is a positive response and connectivity then the instance (utilizing private subnet) has internet access through the NAT Gateway. Another way of testing for connectivity is curl <http://google.com>. Successful internet connection indicates that the private subnet, route table configuration and NAT Gateway are set up correctly.

Due to the complexity of trying to access a private ec2 instance that only uses private subnets from a private route table, we will simply use the AWS Systems Manager tool since it bypasses the need to set up an SSH access due to the private e2 instance not having a public IP address. Start by going to the AWS Systems manager, if the EC2 was created from an amazon machine image (amazon Linux) then the SSM agent should have been pre-installed. Ensure the EC2 instance has an IAM role attached with two permissions: AmazonEC2Rolefor SSM policy allows Systems Manager actions and role must allow access to the necessary SSM endpoints. In order to allow the private EC2 instance to utilize the AWS Systems Manager, you will need to attach an IAM role with the proper permissions for that instance. Head over to the IAM Console, select roles and create role. Select AWS Service as the main trusted entity and select EC2 being the use case. Then find the AmazonEC2RoleforSSM policy under the permissions list, additional attachment of AmazonSSMManagedInstanceCore policy can be useful if needed. Name the role “SSM-EC2-Role” and select create role. The next step is to attach the IAM role to the private EC2 instance. Begin by heading to the EC2 Console, instances and select private EC2 Instance, Select action, under security, select Modify IAM Role, there you can either replace the already existing IAM role with SSM-EC2-Role or you can just keep the AmazonSSMRoleForInstancesQuickSetup role that was already in place by default during the creation of that instance. The QuickSetup role is more than sufficient because it contains multiple permissions that allows you to use Systems manager. Such as ssm:DescribeInstanceInformation, ssm:GetParameters, ssm:SendCommand, ec2messages:SendMessage, cloudwatch:PutMetricData. Given that either role can be used, the instance should already have the necessary permissions to be able to utilize AWS Systems manager. The SSM agent should already be installed and running on the instance if the instance was created from amazon Linux or any supported Amazon machine image. Open the AWS Systems Manager Console, go to Instances & Nodes and select Session Manager, press start session, the EC2 instance should be listed under Managed Instances, select the instance and press start session. A terminal session should be prompted in the browser, grant you direct access to the instance. Once you have access to the Session Manager’s terminal inside the EC2 Instance, begin testing the internet connectivity by running ping 8.8.8.8 or enter curl <http://google.com> , now if you entered ping 8.8.8.8 you should get something like: sh-5.2$ ping 8.8.8.8

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.

64 bytes from 8.8.8.8: icmp\_seq=1 ttl=113 time=2.33 ms

64 bytes from 8.8.8.8: icmp\_seq=2 ttl=113 time=2.07 ms

This indicates that the EC2 instance has internet connectivity through the NAT Gateway (or VPC Endpoint if using SSM without an NAT Gateway). Sending an ICMP echo request to 8.8.8.8 and receiving replies indicate that the internet is reachable since the EC2 instance can access google’s public DNS Server. Now if you enter curl <http://google.com> you should get this result: sh-5.2$ curl http://google.com

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">

<TITLE>301 Moved</TITLE></HEAD><BODY>

<H1>301 Moved</H1>

The document has moved

<A HREF="http://www.google.com/">here</A>.

</BODY></HTML>

sh-5.2$

This indicates that the private EC2 instance has internet access through the NAT Gateway. The HTTP 301 has moved permanently redirect, this means that the website has automatically redirect your request to a new URL. Google has redirected you from <http://google.com> to <http://www.google.com>. Now enter curl <http://www.google.com> to confirm full internet access. The result is that you should get a bunch of html codes.

1. **NAT Gateway Setup:** At this point the private EC2 Instance should have an NAT gateway set up since we previously tested and concluded that it has internet access since the private EC2 instance uses both a private subnet & a NAT gateway, the creation & functionality of an NAT gateway is dependent on the creation of an existing elastic IP, which was entirely done throughout the previous steps.
2. **Security Group Configuration:** Given that we have 1 public subnet and 2 private subnets, there needs to be separate security group configurations setup for each of them individually. For the public subnet (Web server) security group, start by ensuring that the Inbound rules have the following: HTTP (port 80) allows all inbound traffic (0.0.0.0/0) since this is a web server, HTTPS (Port 443) allows all inbound traffic (0.0.0.0/0) for accessing the web securely. SSH (Port 22) only allows inbound traffic from trusted sources (your IP or VPC’s CIDR Block), this limits SSH access to specific IP which increases security), that IP could be your own IP address or a specific VPN range. As for outbound rule configurations, you just need to allow all outbound traffics (0.0.0.0/0) because it allows web servers to be able to access external resources if they are needed, such as calling on API or any external data fetch required. For the Private subnet (Database server) security group, there should be limited access to the database instances since there is more control with traffic needed, those instances should not be exposed to the internet. In terms of the private subnet DB configurations, Inbound rules must ensure that traffic from the web server is allowed. In other words, there needs to be inbound database traffic coming from web servers found in the public subnet (application servers need access to database). The web server security group needs to be specified as a source. Although this prevents external access to the private subnet, communications between the database servers and web servers are granted through this configuration. As for outbound rules, traffic can only be allowed to go through the NAT Gateway, the purpose of this set up is designed to ensure that the database server can continue to retrieve updates through internet access. Additionally, additional restrictions can be imposed by limiting the scope of the IP range that handles outbound traffic, if needed. But for our case, we only need to ensure that outbound traffic is handled through the NAT Gateway (0.0.0.0/0). For the private subnet (Internal services) Security Group, this security group is meant to protect backend services within the private subnet, such as application or data cache services. The inbound rule configurations for the private subnet (Internal Services) allow traffic from Public subnet (Web Servers) or any other internal services that need to be able to access critical internal resources. Additionally, allowing traffic from other internal services is needed if there are additional internal backend services that need to be able to communicate to each other. The Outbound rule is to allow outbound internet traffic (0.0.0.0/0) to flow to the NAT Gateway. Ensure to avoid opening any additional ports that do not have any use cases nor purposes within the scope of the original architect of what consists of setting up the VPC. In other words, if there is not a need to open extra ports, do not open them. Allowing VPC flow logs, helps with troubleshooting and identifying any concerns with traffic flow to and from the NAT Gateway as well as in between subnets.

In terms of the actual technical process of conducting the configuration of security groups, we are only going to be working on one EC2 security group, not VPC security groups (since VPC does not directly handle the management of security groups). To begin the EC2 security group configurations, start by ensuring that public subnet (web server) and private subnets (Database Server and Internal Services [backend] Security Group) are each individually being attached to the same EC2 instances that was originally created from the earlier steps. The separation of security groups for each subnet are designated to specify what kinds of inbound and outbound traffic can be allowed to reach those instances. For this case and for simplicity purposes we will be creating one security group, under a private subnet (Internal/Backend services), for a private ec2 instance since we only have created 1 ec2 instance so far. If we take a look at the subnet ID for that same private ec2 instance, we can see that it was created utilizing the private subnet (Internal) IPv4 CIDR 10.0.3.0/24, therefore the creation and attachment of the security group for that instance will be utilizing the same private subnet. The reason we are not creating extra instances is because the AWS free tier only allows 1 instance, additionally we do not have web servers in the public subnet nor a database in the private subnet, therefore there is not a need for additional creation of security groups for those subnets. The Configuration for one security group will ensure that outbound traffic occurs through the NAT gateway, this allows the private ec2 instance to be able to access the internet. Then the inbound traffic should only occur if additional instances within the same VPC are being utilized. The first step is to create a security group for the private ec2 instance. The configuration for inbound rules will be to allow traffic from only other instances within the VPC, this is useful for internal communications. There should not be any inbound traffic allowed coming from the internet since we are using a private ec2 instance with a private subnet. For outbound traffic configuration, all outbound traffics are allowed to travel to 0.0.0.0/0 , this allows the ec2 instance to be able to access the internet through the NAT Gateway. To recap: inbound rules will have the following: Type - Custom TCP Rule, Protocol – TCP, Port Range – 8080 (internal service), Source Type – Custom, Source – 10.0.0.0/16 and Description – Allow internal services to communicate within VPC. We will not be adding an inbound rule for SSH traffic to be opened to the internet since systems manager manages it internally and it does not require any open SSH ports to the public internet. The Outbound rule (what leaves the instance): Type – All traffic, Protocol – All, Port Range – All, Destination Type – Custom, Destination 0.0.0.0/0 and description allow all outbound traffic for internet access going though the NAT Gateway then select Create Security Group. Next step is to attach the new security group to the private ec2 instance, then open the amazon systems management terminal to test for internet connectivity. Begin by heading towards the EC2 Dashboard, select EC2 instance (private subnet), go to action, under security, select change security groups, under Associated security groups, select the newly created security group (Private subnet [Internal] for EC2), select add security groups, if there is a default security group in place leave it be or remove it and select save. Quickly head back and open the AWS Systems Manager Console, go to Instances & Node tools and select Session Manager, press start session, the EC2 instance should be listed under Managed Instances, select the instance and press start session. A terminal session should be prompted in the browser, grant you direct access to the instance. Once you have access to the Session Manager’s terminal inside the EC2 Instance, begin testing the internet connectivity by running ping by going to and check to see if the EC2 instance is still able to access the internet through ping or curl command to test external internet connections. If you are able to get responses from both commands, then it means that the EC2 instance is able to get external internet connections while having the new security group attached to it and the SSM access to the EC2 instance is properly configurated, since the SSM is able to launch a session on the EC2 instance. The workings of an SSM requires the EC2 instance to have an SSM agent installed by default if the instance was created from amazon linux. Additionally, the IAM role needs to have permissions like the AmazonEC2RoleforSSM or a custom role that has AmazonSSManagedInstanceCore policies. Implementing the NAT Gateway allows the private EC2 instance, utilizing private subnet (internal) and with the attachment of the new security group should allow the instance to access the internet through the private subnet without the need to open any SSH ports.

1. **Network Access Control Lists Configuration:** To recap, we are only using 1 private EC2 instance in a private subnet (internal), 1 NAT Gateway to allow outbound internet access for the EC2 instance, security group controls access to the EC2 instance through the NAT Gateway and confirmed it is able to access the internet through ping and curl commands via the SSM terminal without enabling an SSH port. For this section, we will be configurating Network ACLs. Begin by opening an AWS Management console and head over to VPC Dashboard. The purpose of the Network ACLs is to protect and secure the traffic for both the public subnet (NAT Gateway) and the private subnet (EC2 instance). Public subnet tells you where the NAT Gateway is located, NACL for this subnet is needed to allow traffic for the NAT Gateway to be able to route any internet bound requests from the private EC2 instance. Start by going to the Network ACLs, hit create a Network ACL, then select the VPC that was created from previous steps. Select the NACL that you just created, go to Action, select edit Inbound rules, select add rule, then configure the inbound rule to have the following: Rule #100, Type – All traffic, Protocol – All traffic, Port Range – All, Source – 0.0.0.0/0 (allows all inbound traffic coming from any location, allows the NAT Gateway to be accessible) and Allow/Deny is set to Allow. Next go back to the NACL dashboard, select the same NACL previously configured, go to Action, select edit outbound rules and select add new rule. The configuration for the outbound rule should have the following: Rule #100, Type – All traffic, Protocol All traffic, Port Range – All, Destination – 0.0.0.0/0 (allows outbound internet access through the NAT Gateway) and Allow/Deny is set to Allow. Now we have fully created an NACL for Public Subnet (NAT Gateway). The next step is to create a separate NACL for private subnet (EC2 Instance), head back the NACL dashboard, create an NACL while naming it as NACL for private subnet (EC2 Instance) and select the VPC that was created from earlier steps then select create NACL. Under dashboard, select the NACL for private subnet, go to Action, Select Edit Inbound Rules, select add rule and configure the inbound rules under the following: Rule #100, Type – All traffic, Protocol – All Traffic, Port Range – All, Source – public subnet’s CIDR Block (10.0.1.0/24) and Allow/Deny is set to Allow then select save. Then go back to the NACL Dashboard and under the same NACL for Private subnet, select it, go to Action, and select edit outbound rules, add rule and configurate the following rules for outbound: Rule #100, Type – All Traffic, Protocol – All Traffic, Port Range – All, Destination: 0.0.0.0/0 (allows internet-bound traffic to flow through the NAT Gateway, set Allow/Deny rule as Allow and hit save. Now we have created an NACL for private subnet (EC2 Instance). Recap: we have created separate NACLs for both public subnet (NAT Gateway) and private subnet (EC2 Instance). The next step is to associate each NACL with their own subnets. Start by associating the Public subnet NACL with the subnet where the NAT Gateway is locate and make sure to associate the private subnet NACL with the subnet where the EC2 instance is located. Head to NACL section, select the public subnet NACL, go to subnet association, select edit subnet association, find the **public subnet (App tier),** select it and press save change. Back to NACL Section, select private subnet NACL, go to subnet association, select edit subnet association, find the ***private subnet (Internal),*** select it and press save change. The NAT Gateway should have the **public subnet (APP tier)** listed under it’s subnet. The Private EC2 instance should have the ***Private subnet (Internal)*** listed under it’s subnet ID. Test the EC2 instance connection the NAT Gateway by using the SSM terminal for ping and curl, a response from both should indicate that there is connection through the NAT Gateway from the EC2 instance. If you receive this message while trying to open a session under SSM then you need to attach an IAM role with the AmazonSSMManagedInstanceCore or the AmazonSSMRoleForInstancesQuickSetup and theAWSServiceRoleForAmazonSSM policies: The version of SSM Agent on the instance supports Session Manager, but the instance is not configured for use with AWS Systems Manager. The Cause of this message occurred after configurating the inbound and outbound rules for both Public and private subnet NACLs. The next step would be to head to the IAM roles, under roles, find AmazonSSMRoleForInstancesQuickSetup, click on that role, under permissions, ensure that the role has both permissions: AmazonSSMManagedInstanceCore and AmazonEC2RoleforSSM or AmazonSSMManagedEC2InstanceDefaultPolicy. Then head back to SSM to see if you can start a terminal session and perform network connection tests. If problem persists, then we can just narrow everything down to only being the NACL rule configuration being the root cause of the issue since the issue with SSM unable to access the ec2 instance is caused by the NACLs configuration after deleting both public and private subnet’s NACLs configuration for inbound, outbound and subnet associations.

For simplicity purposes, we will not be making any configurations to the NACL because the true root cause of the issue is that there was already a default NACL configured by AWS after the creation of the VPC and the same Subnets from earlier steps. The default NACL that was created by AWS has the following configurations: VPC ID is tied to the original VPC we created, Inbound rule number is 100, Type - All Traffic, Protocol – All, Port Range – All and Source – 0.0.0.0/0, Outbound rule number is 100, Type – All Traffic, Protocol – All, Port range – All, Destination – 0.0.0.0/0. The subnet association has both the public and private subnet automatically configured by default. As a result, the SSM is able to create a session terminal since it is able to connect to the EC2 instance, otherwise it would be unable to if additional NACLs were created to configurate the same rules which can cause issues due to overlapping configuration rules and priorities.

The reason we are not creating an application load balancer is because in the midst of creating a security group and a target group you would need to implement some kind of Certificate authority for a functional website that you plan to use to be able to complete the verification process of issuing a new ACM certificate. We do not have a DNS validation for the scope of our project. I was unable to retrieve a certificate ID for DNS validation since it said status failed.

1. **CloudWatch Monitoring & Access Logs**

The purpose of a cloud watch is to monitor all of the AWS resources and applications that you are using. For the scope of our Virtual Private Cloud, use the Cloud Watch tool to monitor network traffic and EC2 Instance Metrics. Begin by going to CloudWatch Console and selecting to create a new log group, implement a Cloudwatch logs agent towards the private EC2 instance (Alternatively configure the EC2 instance to send logs through the AWS CLI/SDK if possible). The next step is to set up alarms for key metrics. Under Cloudwatch console, select create alarms for CPU Utilization, Request Count, Error Rates and Latency for the EC2 Instance. These are all critical metrics to be aware of when setting up the Cloudwatch Alarms. Additionally, ensure that SMS notifications for the alerts are enabled in case a threshold breach has occurred. Under the creation of a new log group, set log group name to CloudWatch-log, set retention setting to 2 weeks (14 Days) however any duration is fine and set log class to standard. Standard log class is ideal for the scope of this project since it provides comprehensive log management in terms of its large capability to provide all of the necessary features needed to monitor an EC2 instance. The offerings provide all of the necessity that an infrequent access log class can provide, as well as Live Tail, Metric Extraction, Alarming, Data Protections and log patterns that provide real time live update on visualizing the application’s health. Select Create Log group and continue.

Before Installing a CloudWatch logs agent on an EC2 Instance using the SSM, Make sure that the EC2 Instance’s IAM role has the following permissions: AmazonSSMManagedInstanceCore (SSM Access) and CloudWatchAgentServerPolicy (CloudWatch logs). Run the following commands on SSM’s Session Manager terminal: “sudo systemctl status amazon-ssm-agent” If the EC2 instance has an SSM Agent installed and running then it should prompt the following result: Loaded: loaded (/etc/systemd/system/amazon-ssm-agent.service; enabled; preset: enabled)

Active: active (running) since Wed 2025-02-05 17:48:48 UTC; 1 week 0 days ago

Main PID: 2151 (amazon-ssm-agen).

To add the cloudwatch agent policy for cloudwatch logs, begin by heading back to the EC2 instance page, select the instance, under details, find the IAM Role that it is using, click on it then edit the permission policies of that policy by clicking add permissions, attach policies, search for CloudWatchAgentServerPolicy, select it and press add permissions. Now we can confirm that both policies are implemented in the IAM role that the EC2 instance is using, as long as those policies are listed and found within the IAM role’s permission list. Now we can begin the process of installing the cloud watch agent on an EC2 instance through the SSM. Head to the AWS Systems manager console, under Nodes, select Run Command and press run command. Under Command Document, select AWS-ConfigureAWSPackage. Then ensure the parameters are properly configured prior to running the command. The Command Parameters should be Action – Install (required), Installation type - uninstall and reinstall (to ensure a clean install), Name – AmazonCloudWatchAgent (Required), Version – Leave blank (It should install the latest version) and Additional Arguments – leave empty. Under Target Selection – Choose Instances manually and select the main Private EC2 instances that we originally created. Set timeout to 600 seconds, leave rate control to it’s default settings, Under output options, Enable an S3 bucket to allow writing all command outputs to an Amazon S3 bucket. Elect choose an S3 Bucket name from the list, choose the default bucket (do not delete SSM Diagnosis-ID), send command outputs to Amazon CloudWatch logs by enabling cloud watch logs. This allows streaming and encrypting log data for all commands in the main account to a cloud watch logs. Set the log group name as Cloudwatch Agent. Under Cloudwatch Alarm – optional, we are going to create a cloudwatch alarm. Begin by selecting Create Cloudwatch alarm, under select metric, we can set it to just a relatively a few weeks, Under metrics and select per instance metrics. The list of metrics should show the exact instance name and instance ID of the same instance we have been working on. Along side it, there are a list of metric names, we will not be selecting all of them, only the ones that are essential for continuous monitoring. The metrics that can be selected to be monitored are CPU-Utilization (CPU usage %), StatusCheckFailed\_Instance (checks if an instance is impaired), NetworkIn and NetworkOut (Network traffic). Since Cloudwatch only lets you create, an alarm using one metric at a time, doing separate configurations for the other metrics can optionally be done later. For the main metric we will use CPU-Utilizations, once you have selected it, there should be a graph containing the time and percentage of utilization of the CPU. The Namespace should be AWS/EC2, Metric name is CPUUtilization, Instance ID and instance name is the same instance we worked on, set statistics to average and period is 5 minutes. Under Conditions, set threshold type to static (use a value as a threshold). Define the alarm condition as Whenever CPU-Utilization is Greater > Threshold. Set the threshold value to 10 for 10% for testing purposes. Under Additional configuration datapoints to alarm, set the datapoints to alarm as in 1 out of 1, this defines the number of datapoints within the evaluation period that needs to be breaching to initiate an alarm to go ALARM state, then under missing data treatment configure it as treat missing data as missing. Press next then there is the configure action page containing Notification, lambda action, auto scaling action, EC2 action, Systems Manager action and investigation action. Since all we are doing is setting up a simple Virtual private cloud, we will not be configuring most of the settings we have here. The only one we need to configure is notification, this allows us to configure and receive email alerts any time an alarm is triggered after you create a new SNS topic. The ones we are skipping are Lambda Action, Auto scaling action, EC2 action, Systems manager action and investigation action. Lambda Action is used for ensuring that AWS Lambda automatically responds to an alarm. Auto Scaling Action is used if you need to scale EC2 instances dynamically. EC2 Action is used to stop, reboot or recover instances automatically. Systems manager action is used if you need AWS to run SSM commands whenever an alarm is triggered. Investigation action is used for advanced setup to trigger AWS Services such as Incident manager. Under the alarm notification configurations, we will be choosing the alarm state trigger to be “in Alarm” the metric or expression is outside of the defined threshold. Example: if the CPU Utilization metric reaches above 10% then an alarm will occur. Under send a notification to the following SNS topic, choose Create new topic, set the topic name to be “VPC-Monitoring-Alerts”. Add an email endpoint that will receive the notification then go to your email account and confirm the subscription for the CloudWatch Alarm SNS. Then refresh the page under Amazon SNS Topics and you should see a status confirmed. Make sure that the alarm notification page says select an existing SNS topic and chooses the newly created SNS topic along with the email you entered for subscription then press continue. Set the alarm name as CPU Utilization and write the description saying this alarm will trigger if it reaches 10% CPU Utilization. Press next then review all of the configurations before creating the alarm and press create alarm. Now head back to the AWS SSM run command, document command page then select the alarm name to be the CPU Utilization which is the one we created. Go to the Ec2 console and find out what the IAM role is attached to the instance. Then go to the IAM Role page for that role “AmazonSSMRoleForInstancesQuickSetup”, under permissions policies, select add policies, attach policies, then add AmazonSSMFullAccess and AmazonSNSFullAccess. Under the current role that is attached to the ec2 instance, the new permissions for SNS and SSM full access should be listed under permissions policies. Back to the SSM run command document command configuration page, under SNS notifications, select enable SNS notifications, then select the SSMroleforinstancesquicksetup as the IAM role to start the SNS notifications. On SNS topic, enter Default\_CloudWatch\_Alarms\_Topicas the SNS topic. The reason this is the SNS topic is because we created this SNS topic during the configuration of our first Cloudwatch Alarm, Only topics belonging to this account are listed here. All persons and applications subscribed to the selected topic will receive notifications. This makes sure that all notifications coming from Cloudwatch and SSM are being sent to the exact same SNS topic, in order to better centralized monitoring. Select all events for the event notifications. Under change notifications, select command status changes, this notifies if a status of a command changes. Under the AWS Command line interface command set the platform to be Linux/Unix/OS X to where the commands will be running from. Additionally, we will leave the settings for rate control to be in its default settings. Then review the CLI command before proceeding, since we’re unable to make any direct changes in the box that provides the entire CLI command, we should assure that AWS has automatically written the CLI Command correctly based on all of the configurations we have made in that SSM run command configuration page. The based on AWS’s autogenerating the CLI Command, it should be

aws ssm send-command --document-name "AWS-ConfigureAWSPackage" --document-version "1" --targets '[{"Key":"InstanceIds","Values":["i-03006537f3a176933"]}]' --parameters '{"action":["Install"],"installationType":["Uninstall and reinstall"],"version":[""],"additionalArguments":["{}"]}' --timeout-seconds 600 --max-concurrency "50" --max-errors "0" --output-s3-bucket-name "do-not-delete-ssm-diagnosis-539247487256-us-east-1-0zn1a" --service-role-arn "arn:aws:iam::539247487256:role/AmazonSSMRoleForInstancesQuickSetup" --notification-config '{"NotificationArn":"Default\_CloudWatch\_Alarms\_Topic","NotificationEvents":["All"],"NotificationType":"Command"}' --cloud-watch-output-config '{"CloudWatchOutputEnabled":true}' --region us-east-1

Select Run and see if any errors occur if not, we should be good to go. If you receive this message: NotificationArn Default\_CloudWatch\_Alarms\_Topic is invalid, should be arn:aws:sns:us-east-1:<accountId>:<topic>. Then You need to head back to the SNS console and find the topic we created, copy the entire ARN which should be arn:aws:sns:us-east-1:539247487256:Default\_CloudWatch\_Alarms\_Topic then paste it under the SNS topic, then rerun to see if it worked. If you receive an error message ServiceRoleArn is not valid: arn:aws:iam::539247487256:role/AmazonSSMRoleForInstancesQuickSetup , even though we already added the AmazonSSMFullAccess and AmazonSNSFullAccess then we need to revise what went wrong with the configuration of the IAM role’s permission that is attached to our EC2 instance. If the Permission policies under the AMazonSSMRoleForInstancesQuickSetup has AmazonEC2RoleforSSM, AmazonSNSFullAccess, AmazonSSMFullAccess, AmazonSSMManagedEC2InstanceDefaultPolicy, AmazonSSMManagedInstanceCore and CloudWatchAgentServerPolicy. Then It could be a trust relationship with the SSM is missing for that role. Head to the IAM role dashboard, head to roles, select AmazonSSMRoleForInstancesQuickSetup then go to the Trust relationship tab and make sure the entire trust policy is written as follows: {

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": {

"Service": [

"ec2.amazonaws.com",

"ssm.amazonaws.com"

]

},

"Action": "sts:AssumeRole"

}

]

}

Once This is done, head back to the SSM run Command Configuration page and rerun it based on all of the configurations we discussed. If you receive this message: Parameters provided in document AWS-ConfigureAWSPackage are invalid or not supported. Then you need to make sure that the parameters are formatted correctly as follows:

Action: Install,

Installation Type: Uninstall and reinstall,

Version:

Additional Arguments: {}

Then select run and you should be prompted by a green message saying Command ID:  was successfully sent. Along with the Overall Status and Detailed status indicating that the run command is successful. Go to Targets and outputs and select the instance and press view output . This indicates that everything we have done up to this point in regard to the SSM run command configuration along with the newly created cloudwatch alarm and SNS is correctly configurated. There are zero error messages shown. The only outputs that should be received are Step execution skipped due to unsatisfied preconditions: '"StringEquals": [platformType, Windows]'. Step name: createDownloadFolder and Initiating arn:aws:ssm:::package/AmazonCloudWatchAgent 1.300052.0b1024 install

Plugin aws:runShellScript ResultStatus Success

install output: Running sh install.sh

create group cwagent, result: 0

create user cwagent, result: 0

Successfully installed arn:aws:ssm:::package/AmazonCloudWatchAgent 1.300052.0b1024

The first output indicates that the step was skipped because we are running a linux instance instead of a window’s instance and the create download folder is designed to download essential files for the cloudwatch agent installation. The plugin aws.runshellscript indicates that the SSM plugin was executed with no issues and the install.sh for the cloudwatch agent ran successfully. The cwagent group and user is included in the installation process. Result: 0 indicates that the commands executions are successful. Retrieving the Cloudwatch agent successfully installed indicates that all of the configurations that were done to create the cloudwatch alarm and SNS are correct. Now that the installation is complete, the next step is to ensure the cloudwatch agent is collecting metrics. Head over to the SSM run command and select AWS-RunShellScript and select the EC2 instance, doing this will allow us to be able to run any script that requires root user permissions, such as Sudo or dnf upgrade, as long as the IAM role has the **SSMManagedInstanceCore** Permission policies that are attached to the main EC2 instance. Once that is done, set Decoument version to 1 (default), set command parameters to

#!/bin/bash

sudo yum update -y

sudo yum install collectd -y

sudo systemctl enable collectd

sudo systemctl start collectd

sudo yum update -y amazon-cloudwatch-agent

sudo systemctl restart amazon-cloudwatch-agent

leave the working directory and execution time out at it’s default settings, choose instances manually and choose the main ec2 instance, set time out (seconds) to 600, then follow the exact same configurations we previously did for the cloudwatch alarm and SNS notifications then press run and you should be prompted with a green message saying command ID was successfully sent and the overall/Detailed status is success. Head to output for that instance and find out what the output and error messages are. You should receive messages saying that the system has successfully installed CollectD and its dependencies but it also detected that the newer version of Amazon Linux 2023 is available. The next step that needs to be done is head over to the SSM Session manager to start session then run the following commands: #!/bin/bash sudo dnf upgrade --releasever=2023.6.20250211 -y sudo systemctl restart amazon-cloudwatch-agent. If the SSM terminal has successfully responded by updating the Amazon Linux then everything up to this point, including the installation of collectD is successful. Next enter sudo reboot to reboot the Amazon Linux on that terminal.

The take away from all of this is that we should be able to perform commands that require root user permissions, along with updating the amazon linux and installing collectD since it had to be done separately in order for us to continue with ensuring that the cloud watch agent can collect metrics using collectD. Next on the SSM session manager, run a terminal session and separately run each of the individual commands as follows: sudo yum list installed | grep amazon-cloudwatch-agent, sudo yum install amazon-cloudwatch-agent, sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard. If all of them are working fine or are already installed then you should be prompted back to the Amazon Cloud watch agent configuration manager page.

================================================================

= Welcome to the Amazon CloudWatch Agent Configuration Manager =

= =

= CloudWatch Agent allows you to collect metrics and logs from =

= your host and send them to CloudWatch. Additional CloudWatch =

= charges may apply. =

================================================================

On which OS are you planning to use the agent?

1. linux

2. windows

3. darwin

default choice: [1]:

1

Trying to fetch the default region based on ec2 metadata...

I! imds retry client will retry 1 timesAre you using EC2 or On-Premises hosts?

1. EC2

2. On-Premises

default choice: [1]:

1

Which user are you planning to run the agent?

1. cwagent

2. root

3. others

default choice: [1]:

[1] this is ideal because it ensures that the cloudwatch agent runs with the least amount of privileges needed to collect any metrics and logs, which aligns with the best practices of basic security needs.

Do you want to turn on StatsD daemon?

1. yes

2. no

default choice: [1]:

[1]

Which port do you want StatsD daemon to listen to?

default choice: [8125]

8125

What is the collect interval for StatsD daemon?

1. 10s

2. 30s

3. 60s

default choice: [1]:

1

What is the aggregation interval for metrics collected by StatsD daemon?

1. Do not aggregate

2. 10s

3. 30s

4. 60s

default choice: [4]:

4

Do you want to monitor metrics from CollectD? WARNING: CollectD must be installed or the Agent will fail tostart

1. yes

2. no

default choice: [1]:

1

Do you want to monitor any host metrics? e.g. CPU, memory, etc.

1. yes

2. no

default choice: [1]:

1

Do you want to monitor cpu metrics per core which can give more granular insight into each core’s performance?

1. yes

2. no

default choice: [1]:

2, for just prefer overall CPU utilization

Do you want to add ec2 dimensions (ImageId, InstanceId, InstanceType, AutoScalingGroupName) into all of your metrics if the info is available?

1. yes

2. no

default choice: [1]:

1, this is helpful for metrics for improve identifying and monitoring

Do you want to aggregate ec2 dimensions (InstanceId)?

1. yes

2. no

default choice: [1]:

1, this is helpful to group and aggregate metrics based on the ec2 instance ID by tracking metrics per instance when monitoring multiple instances and the need to analyze data individually for each

Would you like to collect your metrics at high resolution (sub-minute resolution)? This enables sub-minute resolution for all metrics, but you can customize for specific metrics in the output json file.

1. 1s

2. 10s

3. 30s

4. 60s

default choice: [4]:

4

Which default metrics config do you want?

1. Basic – Includes essential metrics such as CPU, Memory and disk usage

2. Standard – Adds more detailed metrics, like network statistics, disk I/O

3. Advanced – Includes all metrics, potentially with additional configurations for monitoring which allows more fine grained data collection

4. None – no default metrics are included, allowing you to customize fully.

default choice: [1]:

1

Current config as follows:

{

"agent": {

"metrics\_collection\_interval": 60,

"run\_as\_user": "cwagent"

},

"metrics": {

"aggregation\_dimensions": [

[

"InstanceId"

]

],

"append\_dimensions": {

"AutoScalingGroupName": "${aws:AutoScalingGroupName}",

"ImageId": "${aws:ImageId}",

"InstanceId": "${aws:InstanceId}",

"InstanceType": "${aws:InstanceType}"

},

"metrics\_collected": {

"collectd": {

"metrics\_aggregation\_interval": 60

},

"disk": {

"measurement": [

"used\_percent"

],

"metrics\_collection\_interval": 60,

"resources": [

"\*"

]

},

"mem": {

"measurement": [

"mem\_used\_percent"

],

"metrics\_collection\_interval": 60

},

"statsd": {

"metrics\_aggregation\_interval": 60,

"metrics\_collection\_interval": 10,

"service\_address": ":8125"

}

}

}

}

Are you satisfied with the above config? Note: it can be manually customized after the wizard completes to add additional items.

1. yes

2. no

default choice: [1]:

1

Do you have any existing CloudWatch Log Agent (http://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/AgentReference.html) configuration file to import for migration?

1. yes

2. no

default choice: [2]:

2, there is no existing cloudwatch log agent configuration file to import

Do you want to monitor any log files?

1. yes

2. no

default choice: [1]:

1, this allows you to specify what log files you want cloudwatch to track

Log file path:

/var/log/awslogs.log

Log group name:

default choice: [awslogs.log]

We are using this file path because we are running everything on the aws platform through amazon linux for our EC2 instance.

Log group class:

1. STANDARD – this is the most suitable option for frequent access to the log data

2. INFREQUENT\_ACCESS

default choice: [1]:

1

Log stream name:

default choice: [{instance\_id}]

Log Group Retention in days

1. -1

2. 1

3. 3

4. 5

5. 7

6. 14

7. 30

8. 60

9. 90

10. 120

11. 150

12. 180

13. 365

14. 400

15. 545

16. 731

17. 1096

18. 1827

19. 2192

20. 2557

21. 2922

22. 3288

23. 3653

default choice: [1]:

1, we use the default choice since it means the log group will be retained indefinately

Do you want to specify any additional log files to monitor?

1. yes

2. no

default choice: [1]:

2, because we only created the cloud watch alarm and SNS specifically just for one metric which is CPU Utilization. Only select yes if you intend to monitor multiple metrics besides one.

Do you want the CloudWatch agent to also retrieve X-ray traces?

1. yes

2. no

default choice: [1]:

2, this is unreleased to our current use case of monitoring the CPU Utilization Metrics

Existing config JSON identified and copied to: /opt/aws/amazon-cloudwatch-agent/etc/backup-configs

Saved config file to /opt/aws/amazon-cloudwatch-agent/bin/config.json successfully.

Current config as follows:

{

"agent": {

"metrics\_collection\_interval": 60,

"run\_as\_user": "cwagent"

},

"logs": {

"logs\_collected": {

"files": {

"collect\_list": [

{

"file\_path": "/var/log/awslogs.log",

"log\_group\_class": "STANDARD",

"log\_group\_name": "awslogs.log",

"log\_stream\_name": "{instance\_id}",

"retention\_in\_days": -1

}

]

}

}

},

"metrics": {

"aggregation\_dimensions": [

[

"InstanceId"

]

],

"append\_dimensions": {

"AutoScalingGroupName": "${aws:AutoScalingGroupName}",

"ImageId": "${aws:ImageId}",

"InstanceId": "${aws:InstanceId}",

"InstanceType": "${aws:InstanceType}"

},

"metrics\_collected": {

"collectd": {

"metrics\_aggregation\_interval": 60

},

"disk": {

"measurement": [

"used\_percent"

],

"metrics\_collection\_interval": 60,

"resources": [

"\*"

]

},

"mem": {

"measurement": [

"mem\_used\_percent"

],

"metrics\_collection\_interval": 60

},

"statsd": {

"metrics\_aggregation\_interval": 60,

"metrics\_collection\_interval": 10,

"service\_address": ":8125"

}

}

}

}

Please check the above content of the config.

The config file is also located at /opt/aws/amazon-cloudwatch-agent/bin/config.json.

Edit it manually if needed.

Do you want to store the config in the SSM parameter store?

1. yes

2. no

default choice: [1]:

1, this is a good idea for proceed with setting up the cloudwatch agent and using the config across multiple instaces

What parameter store name do you want to use to store your config? (Use 'AmazonCloudWatch-' prefix if you use our managed AWS policy)

default choice: [AmazonCloudWatch-linux]

Trying to fetch the default region based on ec2 metadata...

I! imds retry client will retry 1 times Which region do you want to store the config in the parameter store?

default choice: [us-east-1]

Which AWS credential should be used to send json config to parameter store?

1. ASIAX3DNHVEMOFHDWN47(From SDK)

2. Other

default choice: [1]:

1

Successfully put config to parameter store AmazonCloudWatch-linux.

Program exits now.

After going through all of the numerous technical complications with diving deep in to every aspect of being able to finally get permissions to install cloudwatch properly and perform the proper setup, This indicates that the cloudwatch agent configuration is now saved to the SSM Parameter store within the AmazonCloudWatch-Linux and the program exited. The cloudwatch agent is now fully setup with the metrics and logs configurations that we have previously specified. The next thing to do is to start the agent by running this command: sudo amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -c ssm:AmazonCloudWatch-linux -s , this retrieves the configuration from the SSM amazonCloudWatch-Linux, applies it to the cloudwatch agent and starts the agent, additionally you can test to see if its running properly by running this command: sudo systemctl status amazon-cloudwatch-agent then an output should be generated: Active: Active (running). After doing that, the next step is to check the cloudwatch metrics and logs by going to the cloudwatch console under metrics, find the CWAgent and find out if any metrics are being reported, go to cloudwatch logs and find the log group awslogs.log to confirm that logs are being generated. If the CPU Utilization metric is not showing under metrics then we will need to investigate the config.json file to make sure it has CPU Monitoring enabled. Start a session under session manager in SSM and run this command cat /opt/aws/amazon-cloudwatch-agent/bin/config.json then you can edit the file by entering sudo nano /opt/aws/amazon-cloudwatch-agent/bin/config.json then make sure the file contains this: "metrics": {

"metrics\_collected": {

"cpu": {

"measurement": [

"cpu\_usage\_idle",

"cpu\_usage\_user",

"cpu\_usage\_system"

],

"metrics\_collection\_interval": 60,

"resources": [

"\*"

]

}

}

} if not then you need to alter the current existing json file to ensure it includes those CPU measurements. The newly written json file should be :

{

"agent": {

"metrics\_collection\_interval": 60,

"run\_as\_user": "cwagent"

},

"logs": {

"logs\_collected": {

"files": {

"collect\_list": [

{

"file\_path": "/var/log/awslogs.log",

"log\_group\_class": "STANDARD",

"log\_group\_name": "awslogs.log",

"log\_stream\_name": "{instance\_id}",

"retention\_in\_days": -1

}

]

}

}

},

"metrics": {

"aggregation\_dimensions": [

[

"InstanceId"

]

],

"append\_dimensions": {

"AutoScalingGroupName": "${aws:AutoScalingGroupName}",

"ImageId": "${aws:ImageId}",

"InstanceId": "${aws:InstanceId}",

"InstanceType": "${aws:InstanceType}"

},

"metrics\_collected": {

"collectd": {

"metrics\_aggregation\_interval": 60

},

"disk": {

"measurement": [

"used\_percent"

],

"metrics\_collection\_interval": 60,

"resources": [

"\*"

]

},

"mem": {

"measurement": [

"mem\_used\_percent"

],

"metrics\_collection\_interval": 60

},

"statsd": {

"metrics\_aggregation\_interval": 60,

"metrics\_collection\_interval": 10,

"service\_address": ":8125"

},

"cpu": {

"measurement": [

"cpu\_usage\_idle",

"cpu\_usage\_user",

"cpu\_usage\_system"

],

"metrics\_collection\_interval": 60,

"resources": [

"\*"

]

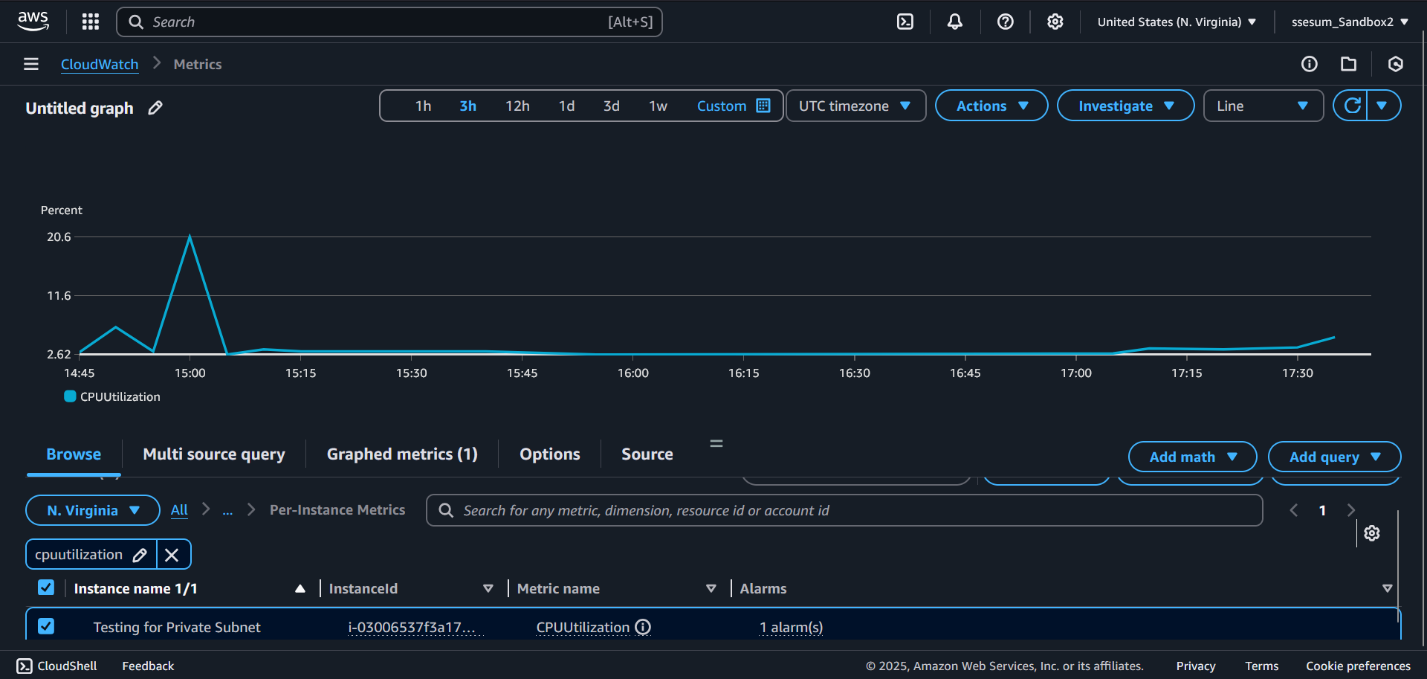
}

}

}

}

Once that is done, save and enter sudo systemctl restart amazon-cloudwatch-agent to restart the agent. Next head back to the cloud watch metrics page and search CPUUtiliation and Select it then a graph should be generated that shows all of the current metrics of data collected.



As we can see here, the CPU Utilization was found under EC2 -> Per instance metric which is the default ec2 monitoring not from cloudwatch agent. To correct this, we will need to do configurations with the json file on SSM again. Another possibility is permission issues with being able to collect those metrics. Head to SSM session manager to start a session and run this command: aws cloudwatch list-metrics --namespace CWAgent --metric-name cpu\_usage\_user

An error occurred (AccessDenied) when calling the ListMetrics operation: User: arn:aws:sts::539247487256:assumed-role/AmazonSSMRoleForInstancesQuickSetup/i-03006537f3a176933 is not authorized to perform: cloudwatch:ListMetrics because no identity-based policy allows the cloudwatch:ListMetrics action

To solve this, we will try adding the Cloudwatchreadonlyaccess policy name within the IAM role attached to the EC2, the reason for this is due to ensuring that the IAM role has access to CloudWatch:ListMetrics. Rerun the terminal to see if the access denied error still persists. If it does not persist then the result should be this:

sh-5.2$ aws cloudwatch list-metrics --namespace CWAgent --metric-name cpu\_usage\_user

{

"Metrics": [

{

"Namespace": "CWAgent",

"MetricName": "cpu\_usage\_user",

"Dimensions": [

{

"Name": "InstanceId",

"Value": "i-03006537f3a176933"

}

]

},

{

"Namespace": "CWAgent",

"MetricName": "cpu\_usage\_user",

"Dimensions": [

{

"Name": "InstanceId",

"Value": "i-03006537f3a176933"

},

{

"Name": "ImageId",

"Value": "ami-0df8c184d5f6ae949"

},

{

"Name": "cpu",

"Value": "cpu-total"

},

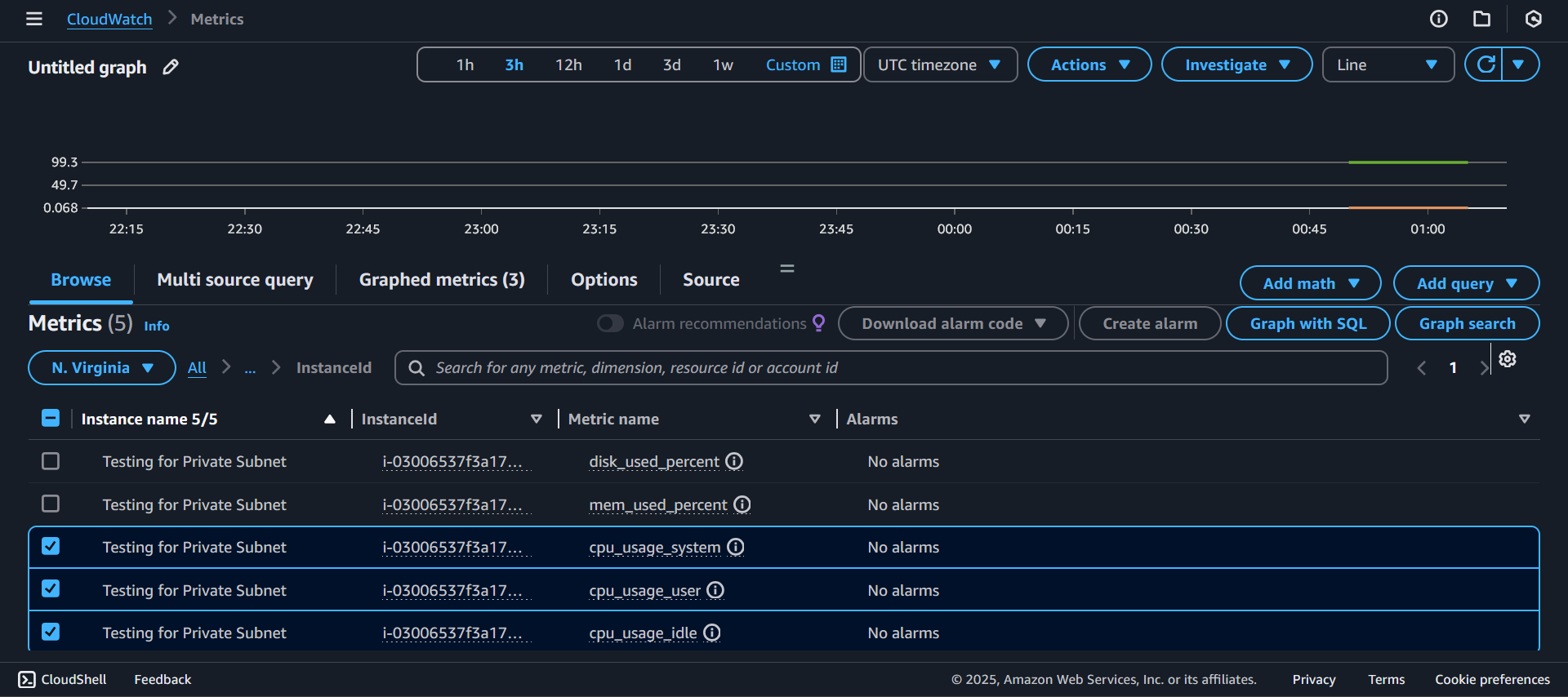
{

"Name": "InstanceType",

"Value": "t2.micro"

}

This confirms that the CloudWatch Agent is now finally able to begin collecting and reporting CPU Utilization Metrics under the CWAgent namespace. Head back to the Cloud watch metrics page to confirm. As we can see the total number of collectable metrics have increased under the CWAgent space and under InstanceID. This is the final result we should see:



Note: if you have deleted an elastic IP and deleted the NAT gateway to prevent additional costs incurred then later created a new elastic IP and new NAT Gateway. Just to later discover that the SSM still has network connectivity issues with the private ec2 instance then its better to try to select the unmanaged Ec2 instance and select diagnoses of network issues then execute to discover the causes. The result of the network diagnosis should show that findings indicate that there is a disabled VPC DNS support and that to resolve this begin by verifying the VPC DNS related attributes. Head over to the VPC dashboard and select the created VPC and press resource map then scroll to where it says network connections. Based on the IDs of NAT gateway and the VPC, if both of them are attached and connected to each other then the root of the problem could be due to the route tables given that the ec2 is a private instance. Head over to the route tables under VPC console, find the route table for the ec2 instance subnet and ensure that the route has the correct NAT Gateway ID. The private route table should be the table we are looking for, under routes we can see that target NAT status is blackhole, to fix that we need to edit the route and replace the old NAT ID with the new NAT Gateway ID and save changes. Now we should go back to the SSM console, wait 5 minutes, refresh and now we should see managed node status instead of unmanaged ec2 instances. As we can see here the biggest take away when reinitiating the entire VPC, we just need to not only create a new elastic IP, new NAT Gateway, reactivate e2 instance but also update the private subnet routing table with a new NAT Gateway ID under its routes in order to restore our VPC back to normal functionality.

1. **Configure Auto Scaling for high Availability**

To begin auto scaling for high availability, head over to ec2 console, under launch templates, select create launch template. Enter name, description, select the box for choosing auto scaling guidance to help us with setting up a template that can be used with EC2 Auto scaling and select the exact same Amazon Machine Image (Application and OS Images), that is currently in use (Amazon Linux 2023), that you use to create your ec2 instance, select the exact instance type that is provided under the drop down menu that should align with the same specifications from your ec2 instance (t2.nano), Under Key pair login, select the existing key pair in case SSH access is needed. Under network settings for subnet, select the private subnet (Internal) since our ec2 instance is private then under firewall (security groups) select existing security group and under common security groups select the one that has a VPC saying private subnet (Internal) for EC2. Adding the ecurity group will be added from all network interfaces. The Storage (volumes) EBS Volumes should already have a default EBS volume (AMI root) containing 8 Gib, EBS, General Purpose SSD (gp3) and 3000 IOPS. Therefore, there is no need to make any changes to that configuration unless additional storage is needed. Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. Under Advance Details, there are some changes that need to be made while many settings will be left to it’s default state. For this case we will need to ensure the IAM Instance profile has the correct IAM role assigned to it, which the role must be the exact same role that is attached to the private ec2 instance that contains policies for SSM and Cloudwatch. This allows ec2 private instance to be able to communicate with AWS Services. The IAM role we choose should be AmazonSSMRoleForInstancesQuickSetup. For the most part many of the advance settings do not apply to the scope of this project and some may add additional incurring costs when not needed. Under Metadata accessible select enabled (must have for auto scaling and SSM to work properly), metadata Version V1 and V2 tokens optional (Prevents existing AWS Services applications or agents that use V1 from breaking if V2 tokens are required), the Metadata Response Hop limit is set to 2 since it is the recommended amount for private instances, and it also decides how many network hopping are needed to allow meta data requests. The allow tags in metadata should be disabled since it prevents any unnecessary accidental exposure of instance metadata. All other settings under advance details that were not mentioned should be left to its default state, disabled or unchecked since they fall outside the scope of our VPC’s overall simplicity design. Once everything is done, select create a launch template. Noice how we are being prevented from creating a launch template due to an advance network configuration setting says subnet-080ec4e69ae93850a, remove subnet for EC2 Auto Scaling templates, Not applicable for EC2 Auto Scaling. Therefore, to resolve this we will need to remove the select subnet under subnet and choose not included in launch template since selecting subnets through the launch template is not applicable for ec2 auto scaling. But under advance network configuration, almost everything should be set to default settings and say not included in launch template while the only security group listed should be the one we selected that is private subnet (Internal) for ec2 VPC. Once that is done, hit create launch template. The launch template should be created, and the next part of this step is to create an auto scaling group. Begin by heading over to the AWS EC2 dashboard and selecting auto scaling groups, then press create auto scaling group. Name it Auto Scaling Group, then under launch template select the newly create “Auto-Scaling-Template” that should outline all of the specifications and configurations that we have previously made to create that launch template then press continue to enter Instance Launch options. Under Instance type requirements, we are going to keep it exactly as it is shown since it is under its default settings that should align with the information provided under the creation of the auto scaling template. Launch template: Auto scaling template, Version: Default, Description: Auto Scaling Template for Private EC2 instance and Instance type: t2.nano. Under network settings we will be choosing the VPC ID that is under created VPC with the IP of 10.0.0.0/16. The selection of the VPC should be based on the way you have set up your network for instance. As for Availability zones and subnets, we are going to choose the ones that are within the VPC where the auto scaling group will be launched. In this case let’s select public subnet 10.0.1.0/24, private subnet (DB) 10.0.2.0/24 and private subnet (Internal) 10.0.3.0/24. Choosing all three of them is the ideal option because we want to ensure there is both high availability and fault tolerance, in case one availability zone goes down, in order to be able to distribute the instances across multiple different availability zones. Public subnets are used for instances that need internet access that need to communicate directly with the internet. While the private subnets do not need direct internet access but can be used specifically for internal infrastructure or things such as databases and application instances that require communication with public subnets (backend applications). Lastly, we should choose Balanced best effort for availability zone distribution because if a launch fails in one availability zone then the auto scaling will try to launch into another health availability zone then press continue. In Integrate with other services page, under load balancer we will select no load balancer since we are not going to add any load in our VPC and also select no VPC Lattice services (unless there is a need due to service to service communication). Selecting no load balancer means traffic to your auto scaling group will not be fronted by a load balancer and no VPC lattice services means VPC Lattice will not manage your auto scaling group’s network access and connectivity with other services. Next, enable Zonal shifts to allow better resilience during availability zone impairments, in other words new instances will get launched and retargeted towards healthy availability zones until the zonal shift gets cancelled. Under health check behavior we will choose to replace unhealthy in order for health check to replace unhealthy instances in an impaired availability zone. Under health checks, we can see that it is set to always enabled for EC2 health checks, this ensures that EC2 auto scaling checks that instances are running and monitors for underlying hardware or software issues that could impair an instance. Since the only available option for us to choose for additional health check types is Turn on Amazon EBS Health checks we will select it. This is useful if the instance depends on EBS volumes. What this does is that the EBS will monitor if an instance root volume or attached volume stalls. It will report unhealthy volume, EC2 auto scaling will replace the instance on its next periodic health check. Note: EBS volumes that are attached to an instance have a DeleteOnTermination attribute. When instances are terminated, this attribute either detaches the volumes (keeping data intact), or deletes the volumes. Detached volumes are not reattached when a new instance starts. To avoid data loss, take frequent snapshots of any critical EBS volumes. You can automate snapshot creation and deletion with [Amazon Data Lifecycle Manager](https://docs.aws.amazon.com/en_us/console/autoscaling/ec2/amazon-data-lifecycle-manager-info-box). For our case there is no need to follow that step since we do not have any real data nor important and sensitive data stored within either the instance nor the EBS volume. Under Health check grace period we can set it to 300 seconds as an ideal initialization time. This is considered a time period that delays the first health check until the instances as finished initializing. It does not prevent an instance from being terminated when placed into a non-running state. Press continues to head to the configure group size and scaling. Under part 4 of creating Auto Scaling group (Configure group size and scaling – optional), we will set the initial size of the auto scaling group to be just one since we are only handling one private ec2 instance. The desired capacity type should be the one that is already set by default “Units (number of instances)”. Under scaling, we will set the scaling limit based on the minimum and maximum desired capacity. The minimum should be one and the maximum should be three. For Automatic Scaling settings, we will be selecting no scaling policies, this ensures that the auto scaling group will retain its initial size and does not dynamically resize to meet any demands since we do not have demands to increase it’s auto scaling capacity needs which falls beyond the scope of our initial VPC’s design. The Functionality of Instance Maintenance policy is designed to ensure that the user has more control over how auto scaling group’s availability works during instance replacement events. The policy covers different functions such as health checks, instance refreshes, maximum instance lifetime features and events that happen automatically to keep the group balanced, also known as rebalancing events. For our case we will be choosing to launch before terminating (prioritize availability), this ensures that the policy will launch new instances and wait until they are ready before terminating other instances that should no longer be needed. This allows you to be able to go above the desired capacity by whatever specified percentage, but this will temporarily increase costs. Under set health percentage we will have the minimum by default to be 100$ and the maximum to be 110% of 1 instance, this should be within the range of the group’s scaling limits. The purpose of the health percentage is to configure the maximum percentage of the desired capacity that can be in service during instance replacement events. Under view capacity during replacements based on your desired capacity, we can see that both the current desired capacity (instance) and temporary minimum healthy instances are set to one, while temporary maximum healthy instances are set to two automatically then the scaling limits of min and max should be 1 – 3. Under Additional Capacity settings we can see that the capacity reservation preference is set to default, this means that the auto scaling uses the capacity reservation preferences based off of the launch template. The purpose of capacity reservation preferences is to ensure that we have the option to decide if we want the auto scaling to launch instances based on either an existing capacity reservation or the capacity reservation resource group. Under Additional settings we will be enabling instance scale-in protection, this allows any newly launched instances to be protected from scale in by default. For monitoring, we will be enabling group metric collection within cloud watch. Then we will also be enabling default instance warmup and set the timer to 60 seconds. The purpose of this is to ensure an adequate amount of time is needed to prevent the cloud watch metrics for new instances from contributing to the group’s aggregated instance metrics, given that there are not any useful usage data to be collected yet. The settings for the configure group size and scaling is to ensure that availability, efficient scaling and cost control are in effect while maintaining the flexibility of our auto scaling group set up. Press continue and skip add notifications and add tags since those are optional until we reach view page before creating the auto scaling group. However, do note that you can optionally choose to add tags to instances (and their attached EBS volumes) by specifying tags in your launch template. We recommend caution, however, because the tag values for instances from your launch template will be overridden if there are any duplicate keys specified for the Auto Scaling group. Press creates auto scaling group. If we head over the auto scaling group page and select the newly created ASG, under activity history we can see that the ASG successfully launched an instance in response to a desired capacity change. Knowing that our EC2 instance is running, has both status checks passed and IP address assigned, we head over to the instance management tab of the ASG and we can see that Instance ID: (ID), lifecycle: Inservice, Instance type: t2.nano, weighted capacity: “-“ , launch template/configuration: auto-scaling-template Version 1, Availability Zone: us-east-1c, Health status: Healthy, Protected From: Scale in, based on everything mentioned we can assure that the “-“ under weighted capacity indicates that it hasn’t been assigned a specific weight to instances in a mixed instance policy. Given that we are using a single instance type t2.nano, the field should remain empty and does not affect scaling behavior. Given that the instance life cycle is in service means its active and running, health status is heathy which passes EC2 health checks, availability zone is in us-east-1c means the instance was launched in a valid Availability zone and protected from scale-in means it prevents ASG from terminating the instance in the event of a down scaling. However, if we head to the main ASG’s status being “-“ as in empty it indicates that it is a normal behavior, this is because there are no scaling events being active. It should not display status unless an action has occurred such as scaling up or scaling down. Given that the instance is running and healthy, the ASG is in an idle functional state. If you are expecting the auto scaling group to scale up or scale down based on demand and fluctuations, but the status remains empty then it could be a configuration scaling issue. The ASG dashboard under activity history should indicate any errors or failed scaling actions that have occurred. If there are no errors listed then the ASG is perfectly fine. Additionally, when comparing total desired capacity to overall running instances make sure that the numbers align. Example: if the desired capacity under ASG is one and it lists one healthy active instance then it should be fine and normal. Since the ASG meet its goal and has no updates to show for its status. Unless there are configuration issues in between the ASG and the EC2 instance then it would show a log of issues under activity history in the ASG dashboard. Overall, the Auto Scaling group is working fine.

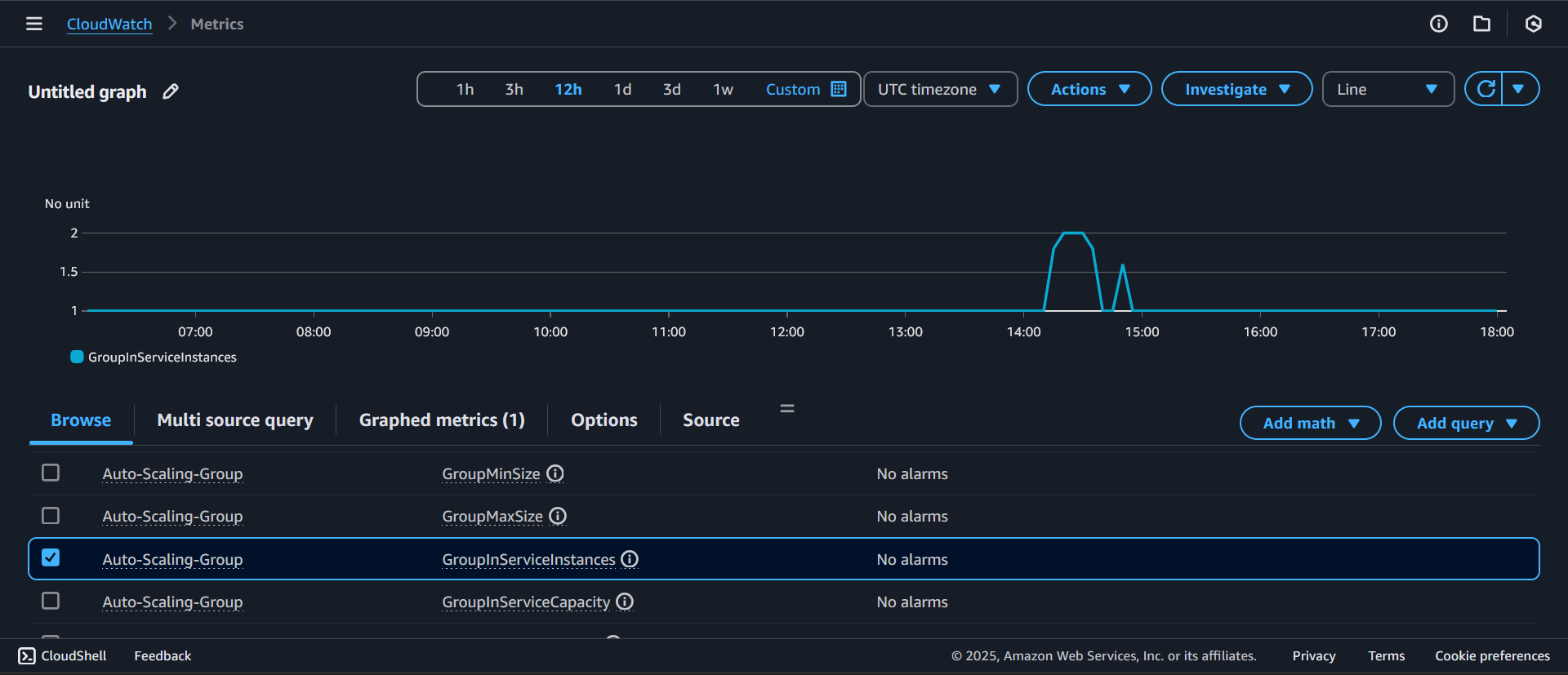
1. **Final Testing and Validation**

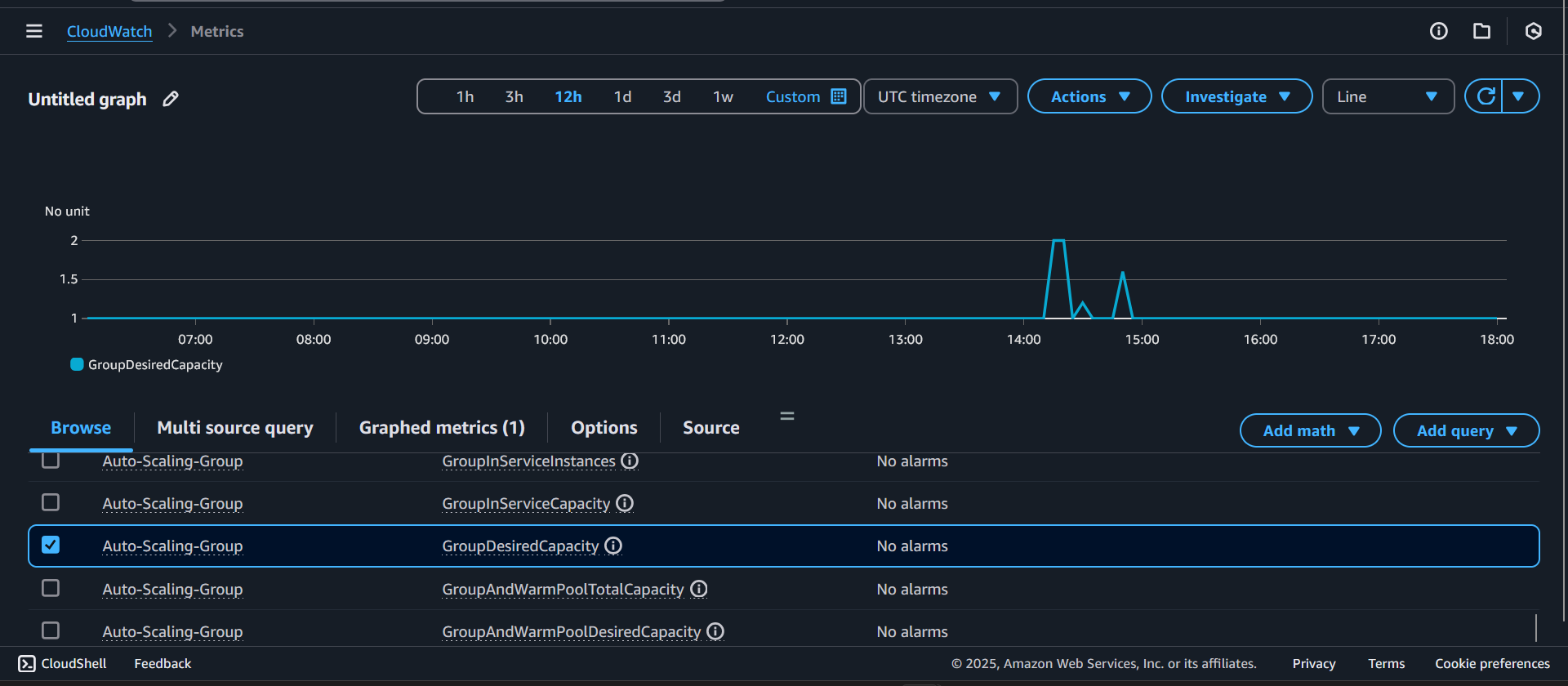
If you received an email about an alarm about the CPU Utilization reaching passed 10% then that means the configurations for the alarm and SNS for that ec2 instance are working properly. Now that the ASG is able to connect with the ec2 instance since it is able to acknowledge that the instance is healthy and the activity history contains no error or failed messages, we can confirm that the ASG is functioning normally. For the final phase of this VPC project we just need to Test Auto Scaling Behavior, Network Connectivity and Security Testing, Monitor Logs and Metrics in CloudWatch. When testing auto scaling behavior if we alter the desired capacity by a number higher or lower than the current number then it should alter the status of the Auto Scaling Group. We begin by heading over to the auto scaling group page and selecting the ASG that was created. Then under Action go to edit and edit the desired capacity by changing from 1 to 2 then press update and you should see a status that says updating capacity. Setting the desired capacity to be 1 number higher than the current number of instances should cause a new activity to be generated in the history log. Wait a few minutes then go to the activity history log under ASG and we can see that a new message has prompt saying at 2025-03-10T14:15:19Z a user request update of AutoScalingGroup constraints to min: 1, max: 3, desired: 2 changing the desired capacity from 1 to 2. At 2025-03-10T14:15:27Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2. If we head over to instance management we can see that the lifecycle should say In service for the instances that were created by the ASG. Now that we scaled up, we are going to scale down to one. Note that we have a main private instance, and the extra instance was created by ASG. Repeat the steps of configuring the desired capacity but change the number to one for desired capacity and set it to one for max desired capacity then press update. As we can see that under activity history we got a message saying cancelled “Could not scale to desired capacity because all remaining instances are protected from scale-in. At 2025-03-10T14:25:17Z a user request update of AutoScalingGroup constraints to min: 1, max: 3, desired: 1 changing the desired capacity from 2 to 1. At 2025-03-10T14:25:31Z group reached equilibrium.” This confirms that the scale in protection works. In order for us to get the ASG to terminate an instance we will need to disable to the scale in protection under the ASG settings then update it again. Alter the numbers by getting it to scale up then scale down and wait a few minutes then refresh the activity history to get a new status logged. If that does not work then try detaching from ASG and deleting the extra instances under EC2 instance dashboard then start a new slate with getting the ASG to scale up and scale down. Wait a few minutes for each action and the activity history should generate log. Also under the ASG configuration under termination policies set it to the oldest instance, termination should occur due to scale in. Under the activity log it should say Detaching EC2 instance: i-0b7235b7944fed91d At 2025-03-10T14:39:15Z instances i-0841aa6281610e0c4, i-0b7235b7944fed91d were detached in response to a user request. This confirms that the ASG is responding to manual actions. As for terminating an EC2 instance and launching a new one, we should receive these messages: Terminating EC2 instance: i- 048cb827b756619be At 2025-03-10T14:40:16Z an instance was taken out of service in response to an EC2 health check indicating it has been terminated or stopped. Launching a new EC2 instance: i-0afd37b755fe3a6e3 At 2025-03-10T14:40:16Z an instance was launched in response to an unhealthy instance needing to be replaced. Given that we disabled the scale in protection prior to creating another instance from scaling up, we should be able to get the ASG to terminate an EC2 instance by scaling down by both changing the desired capacity back to 1 and setting the termination policy to oldest whenever scale in occurs. We should be prompted a scale down message that says this: Terminating EC2 instance: i-0afd37b755fe3a6e3 At 2025-03-10T14:53:29Z a user request update of AutoScalingGroup constraints to min: 1, max: 3, desired: 1 changing the desired capacity from 2 to 1. At 2025-03-10T14:53:41Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2025-03-10T14:53:41Z instance i-0afd37b755fe3a6e3 was selected for termination. This concludes all of the necessary testing needed to ensure the ASG is working properly based on the basic use cases being done. Additionally, if we head over to instance management we can see that there is only one instance remaining and it aligns with our desired capacity.

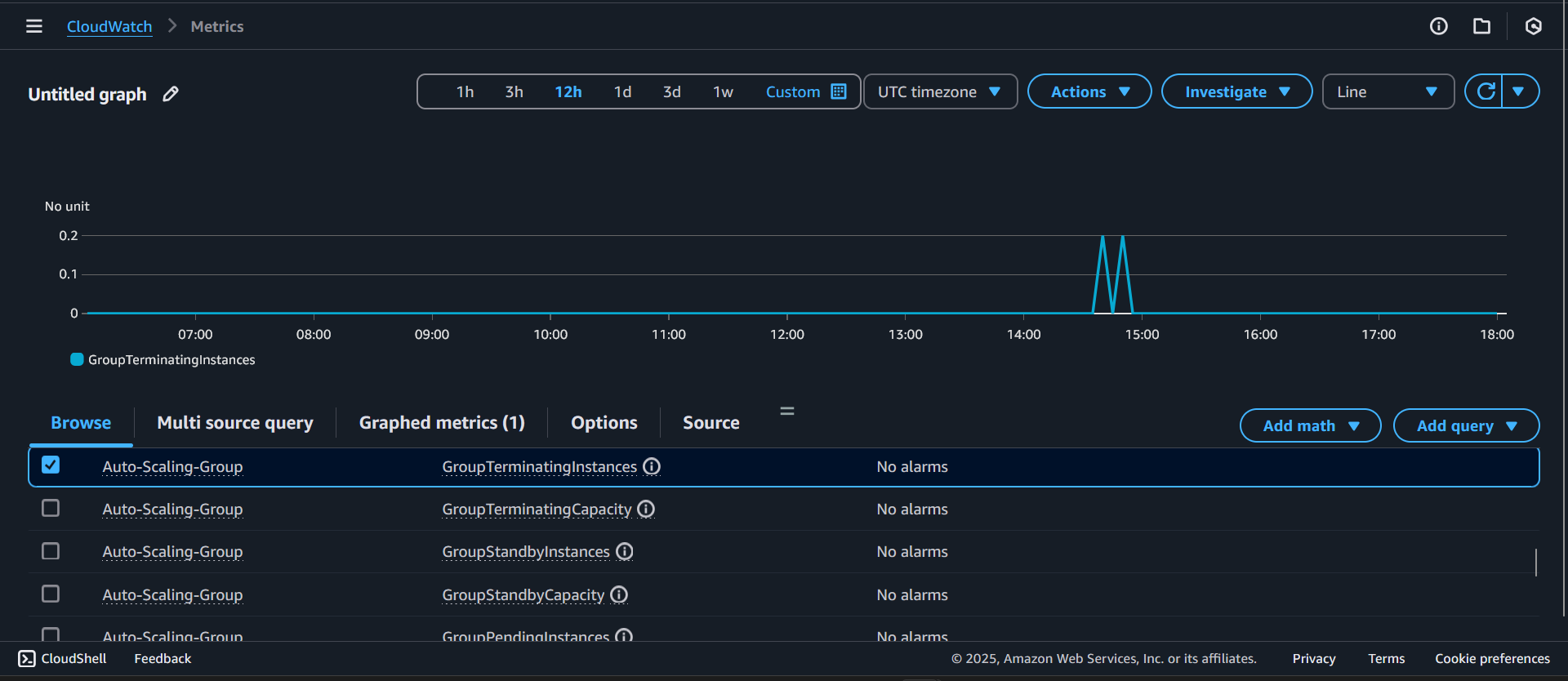
In the next part of Network connectivity and security testing we will head over to the EC2 instance dashboard and confirm if the instance is deployed in the correct private subnet. We can see that the instance’s private IPv4 address is 10.0.3.32 and it is attached to the private subnet (internal) which also has an IPv4 CIDR of 10.0.3.0/24. To confirm that the instance is correctly distributed across different availability zone, we need to head over to the instance dashboard then under availability zone we can see that it says us-east-1c. If we compare that main ec2 instance to the previous ones that were created under ASG then we can see that those have different availability zones such as us-east-1a, us-east-1b and us-east-1c. They should have their own subnets and ip addresses, this concludes that the instances have been distributed correctly across different availability zones. If we head over to the SSM, under session management then attempt to launch a session. We should be able to access an SSM terminal through our instance. This is possible since the IAM role attached to the instance should have more than one SSM role such as AmazonEC2RoleforSSM. If you are able to generate responses with curl and ping command then this also concludes that not only do we have proper internet connection for the private instance but also that the security group has been configured correctly. We can double check the configurations to see what was implemented in the security group. For the inbound rule we can see that HTTPS TCP port 443 and Custom TCP port 8080 have been implemented under IP 10.0.0.0/16. The outbound rule is configured to accept all traffic going out through IPv4 address 0.0.0.0/0, this ensures that all outbound traffic for internet access through NAT Gateway. Additionally we can also see that the Network ACL contains all 3 subnets that it is associated with as well as finding out how each individual subnet is configured for both inbound and outbound rules. If we enter curl -I http://example.com # Check outbound HTTP connectivity

ping -c 4 google.com # Check DNS resolution and are able to generate responses then all of the configurations that we have done relating to network have been correct so far.

The last verification testing we need to do is monitor logs and metrics in cloud watch. Head over to cloud watch console under metrics, find and monitor each of the following metrics: GroupInServiceInstances, GroupDesiredCapacity and GroupTerminatingInstances.

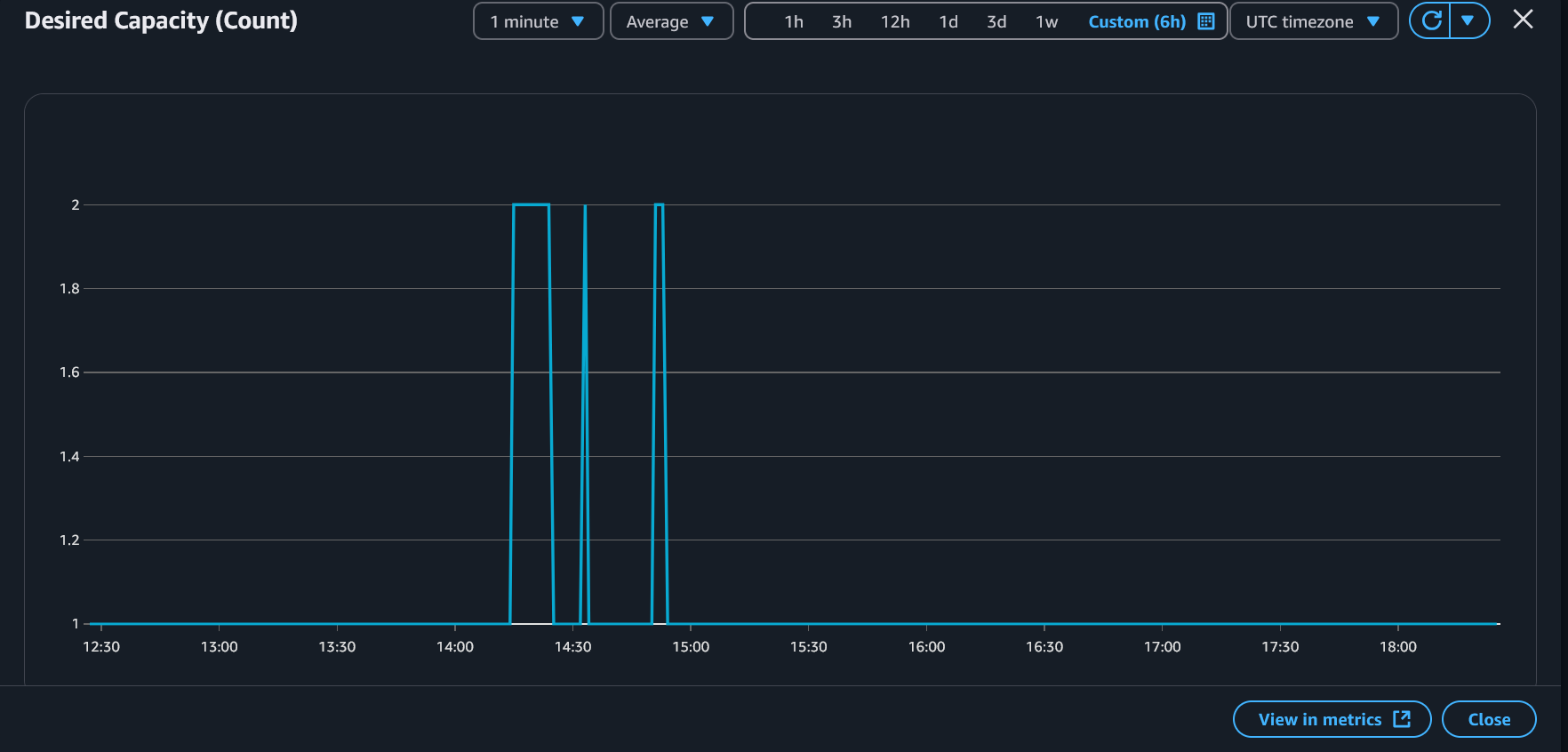
****

****

****

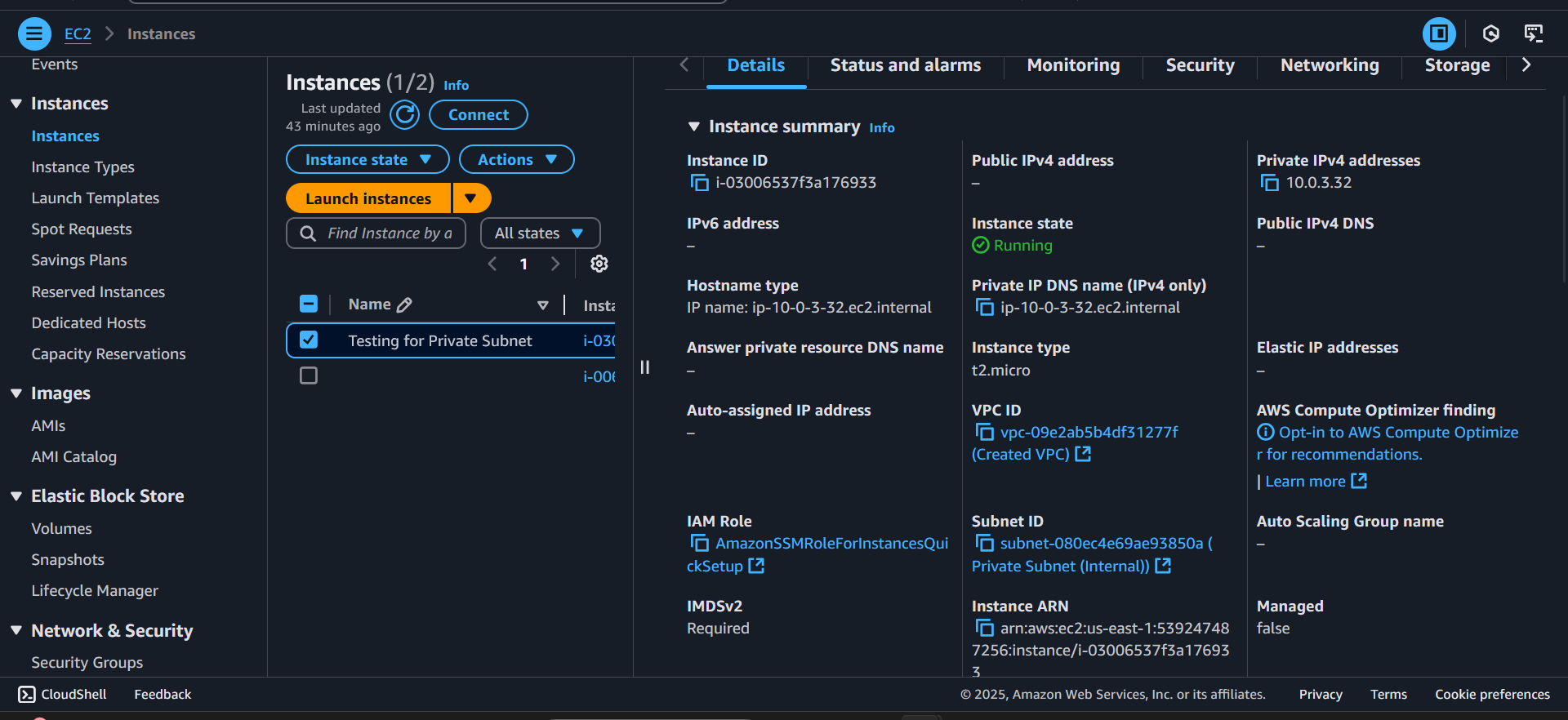
Head over to Auto Scaling group, under monitoring, select enable group metrics collection and find Total instances (count) and total capacity (count).



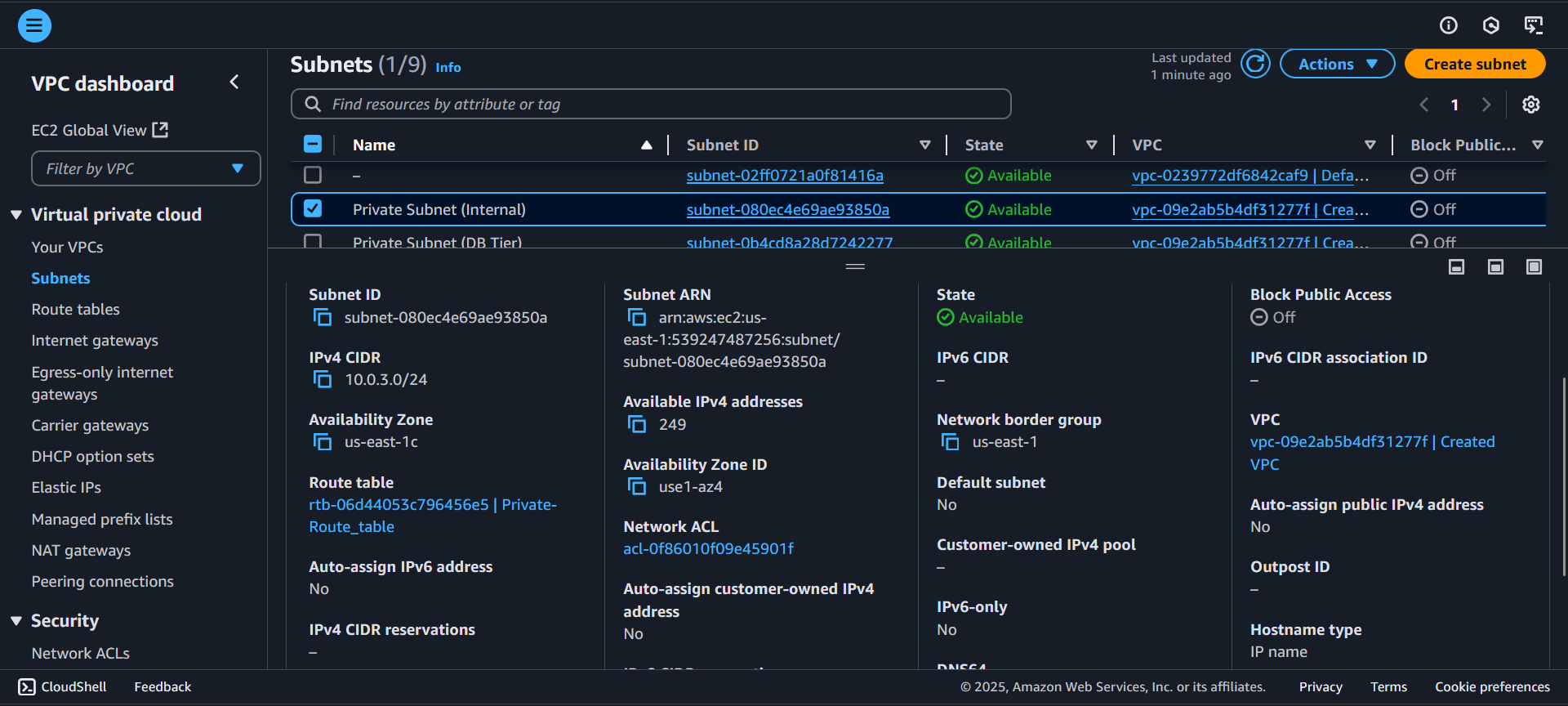


Some Screenshots of each section that were utilized throughout the creation of the VPC project from each AWS Service console can be useful for documenting what has happened.

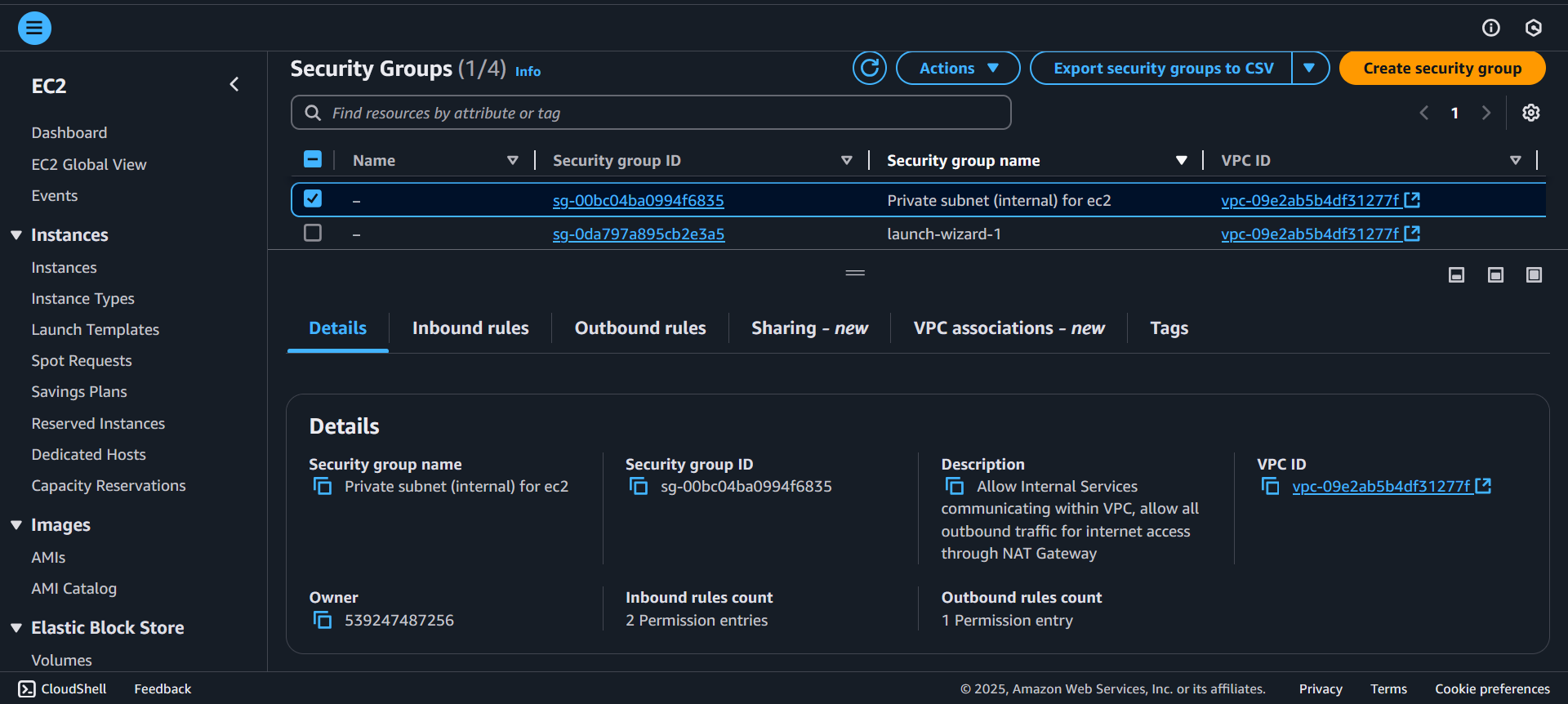
**EC2 Instance Dashboard:**

****

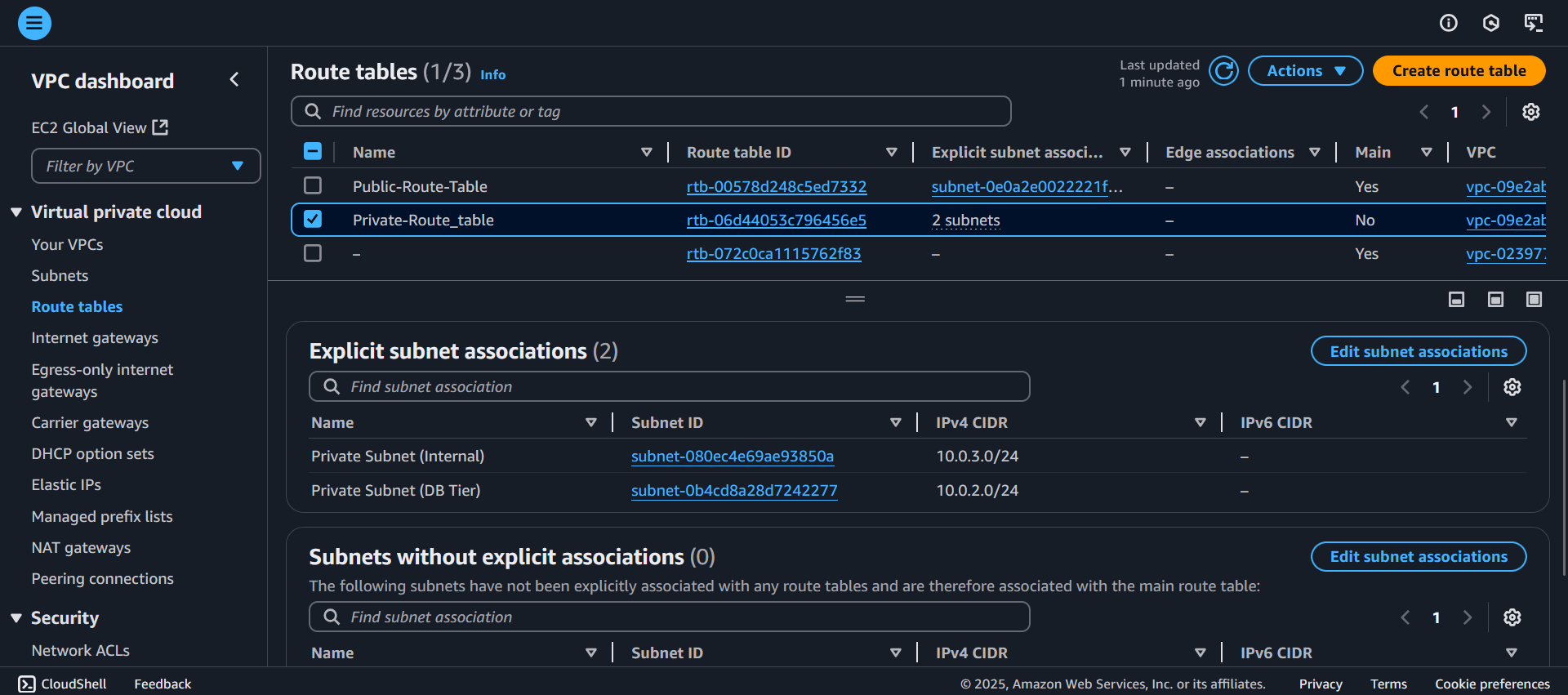
**Subnets:**

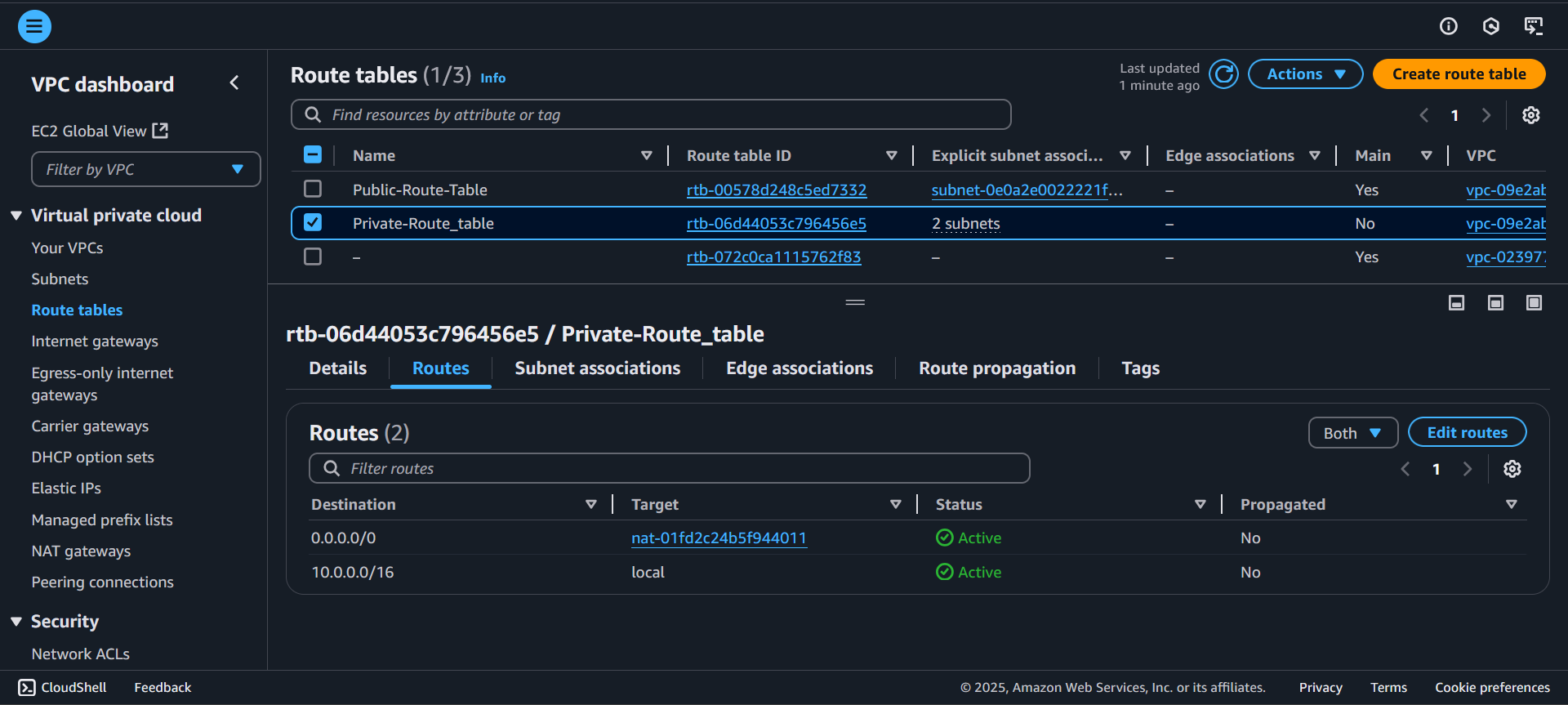
****

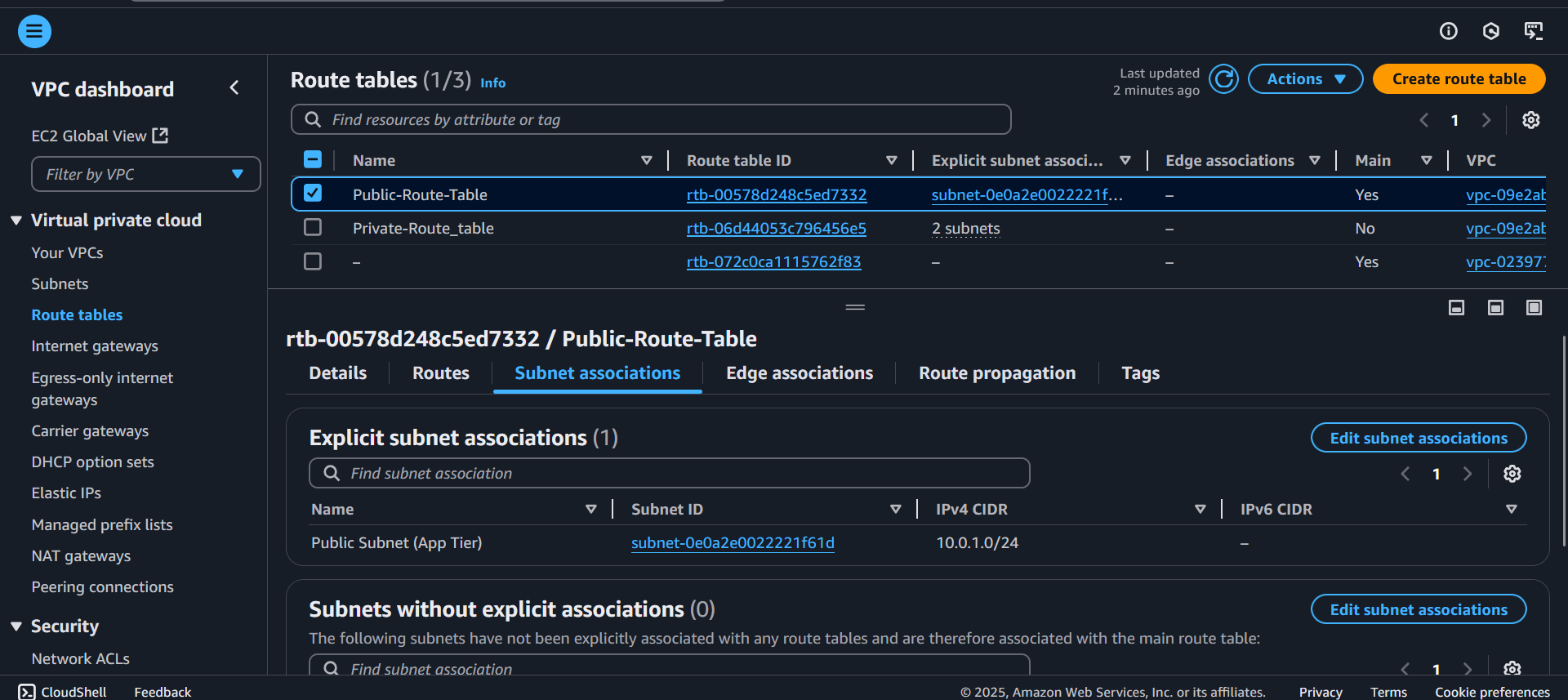
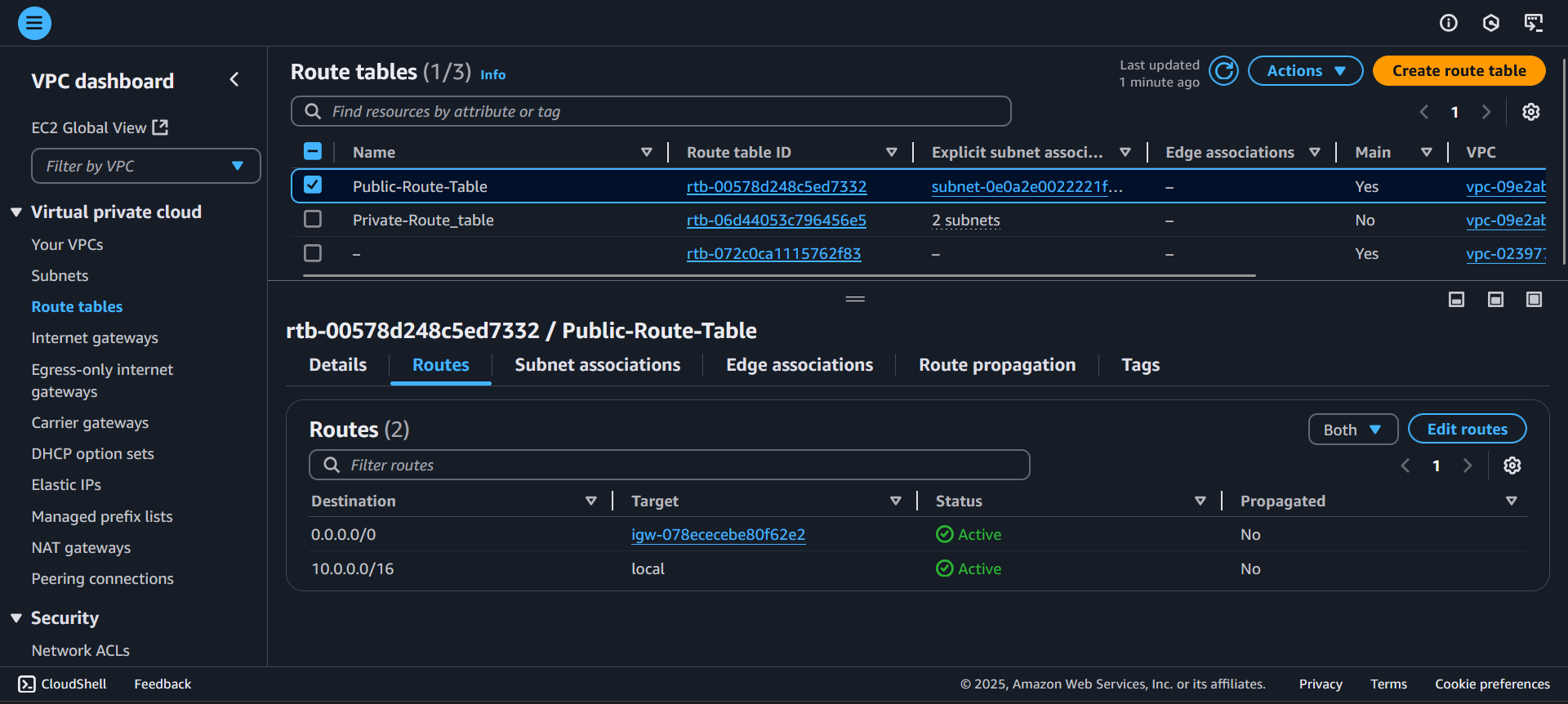
Security Groups (EC2 Feature):

****

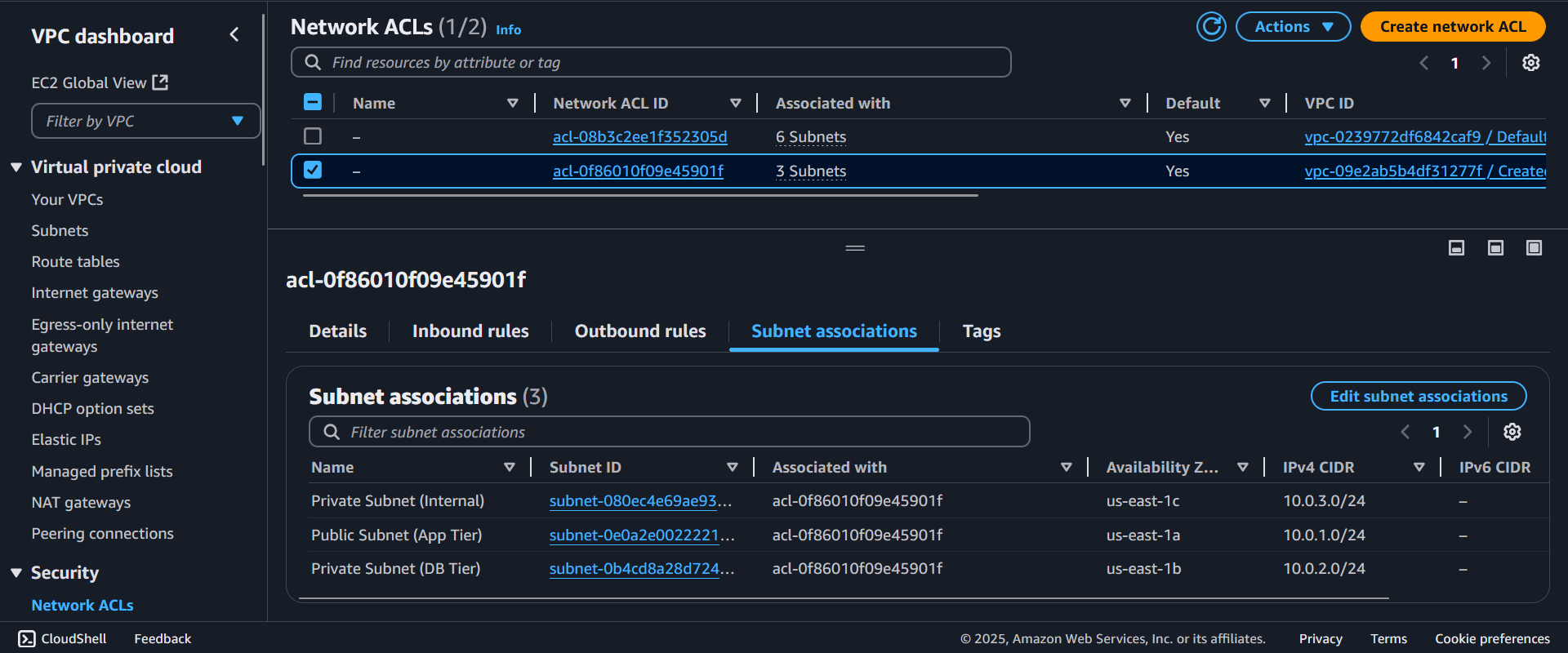
**Route Table:**

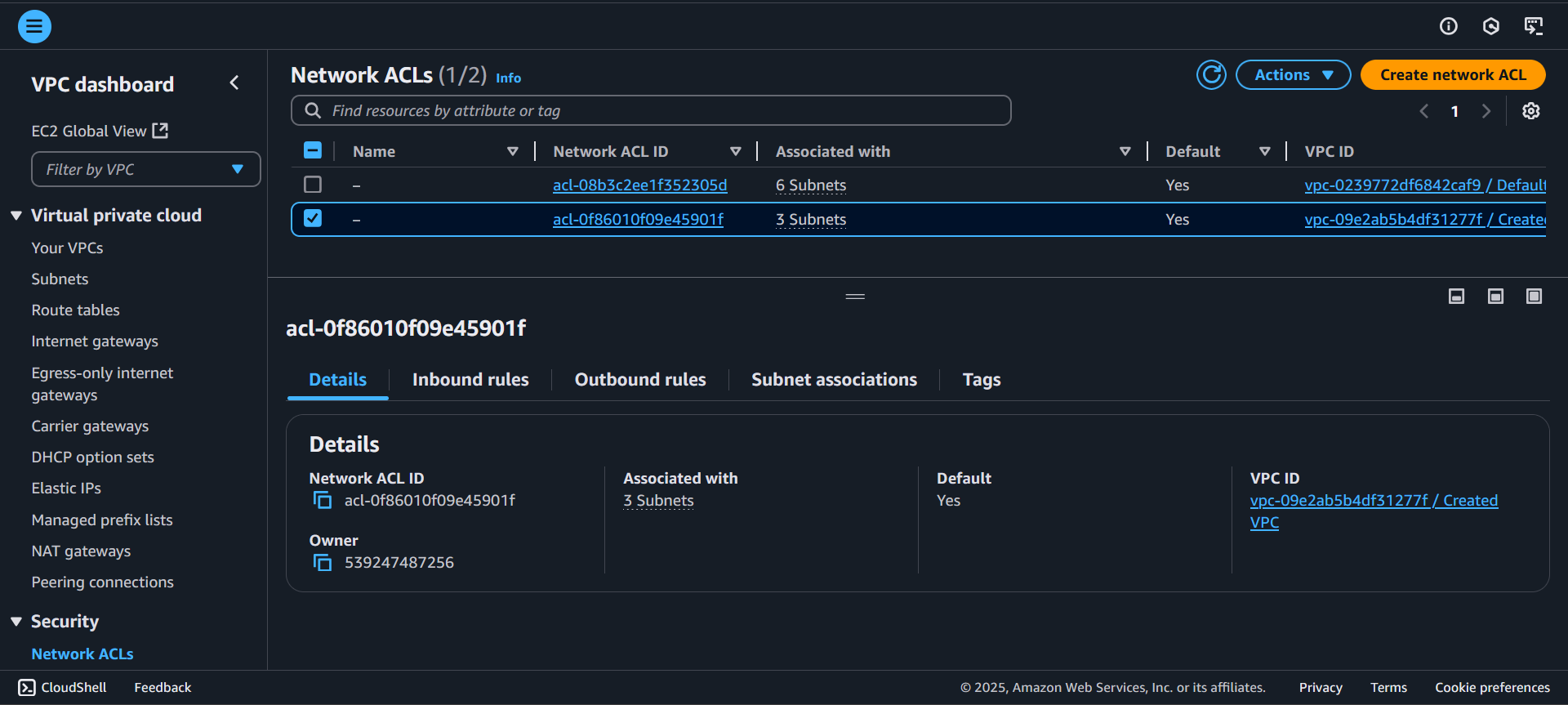
****

****

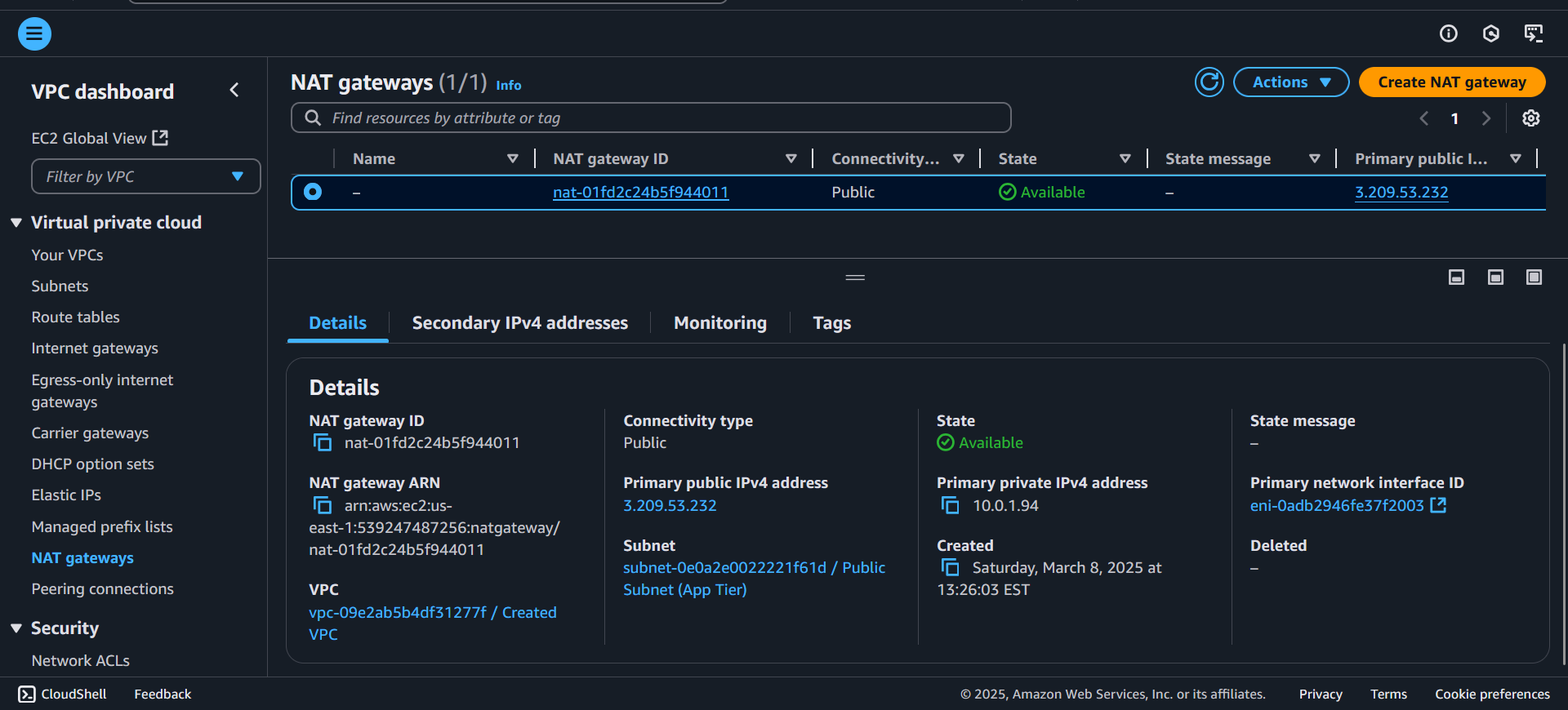
****

**Network Access Control List:**

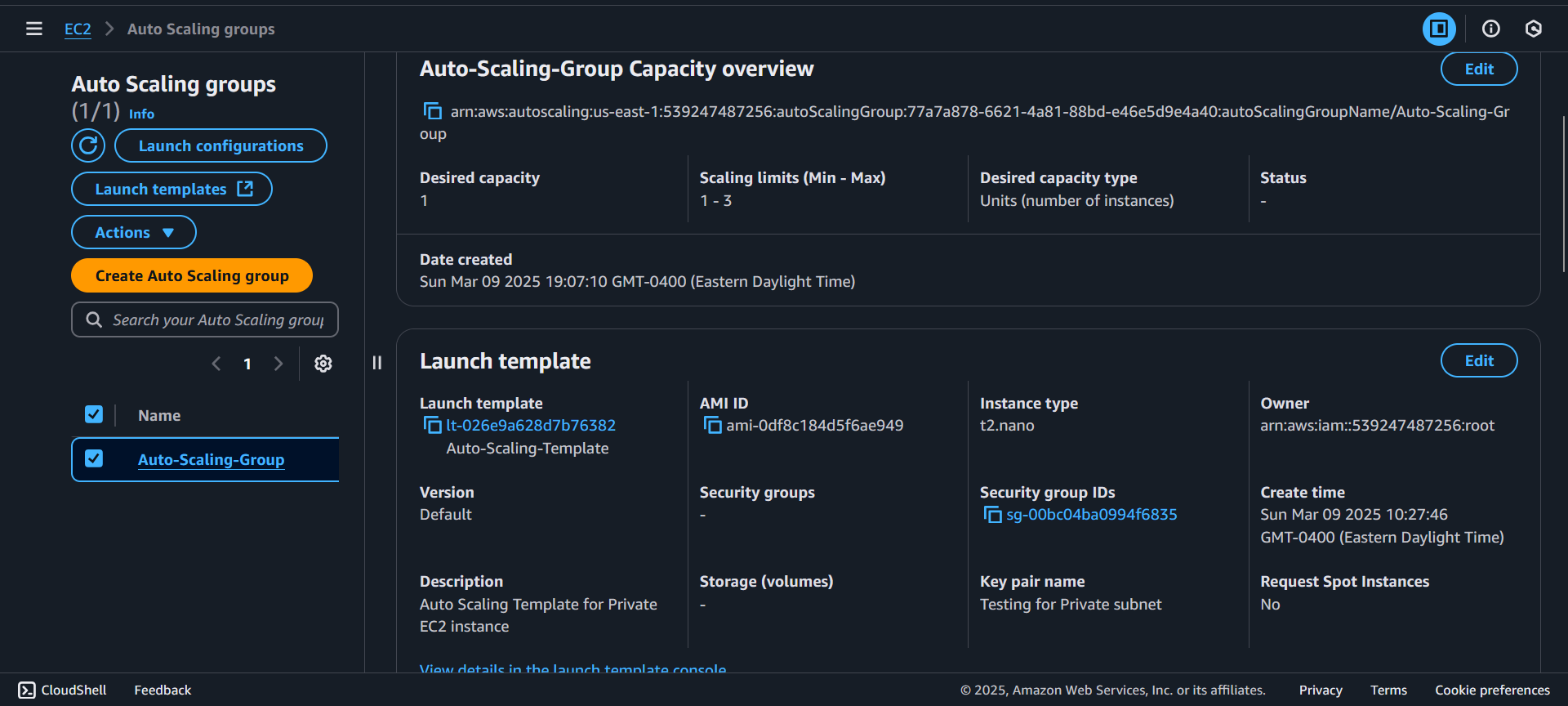
****

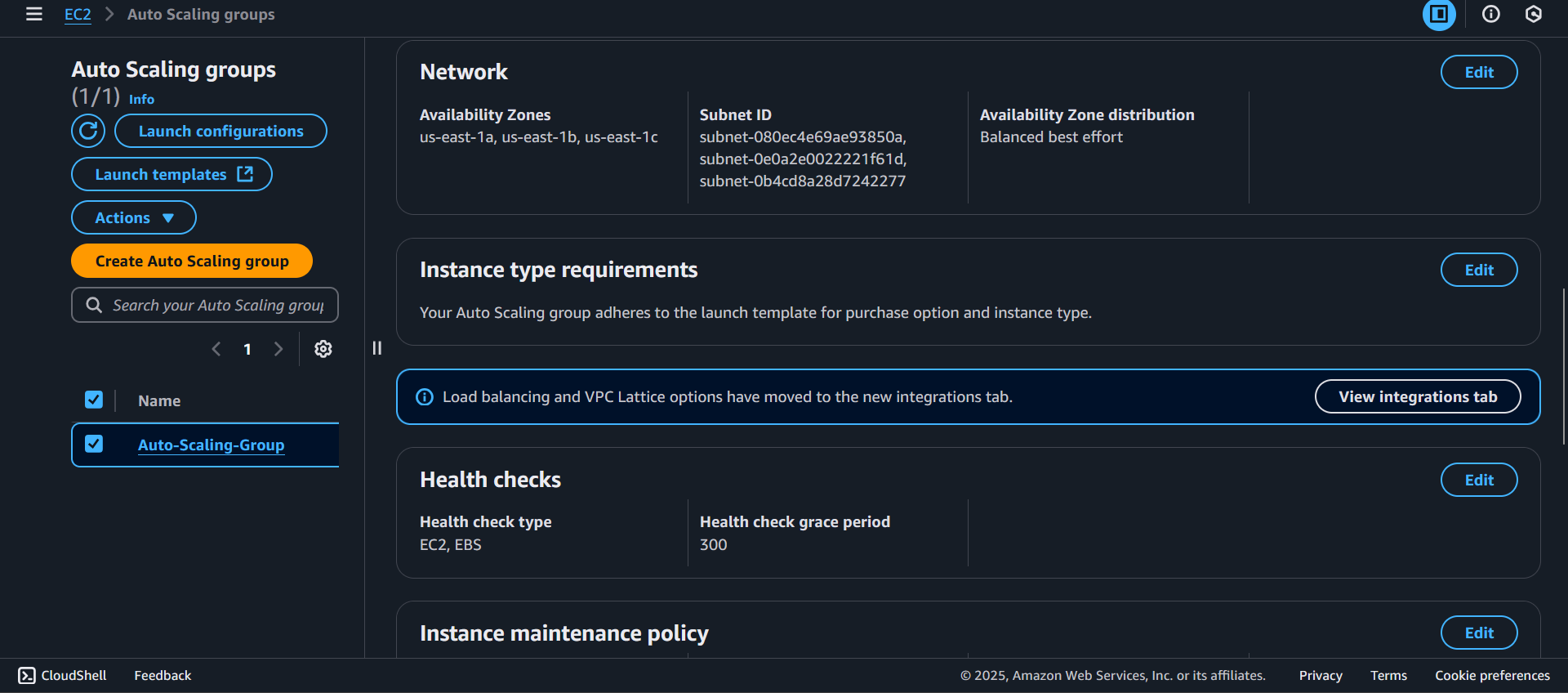
****

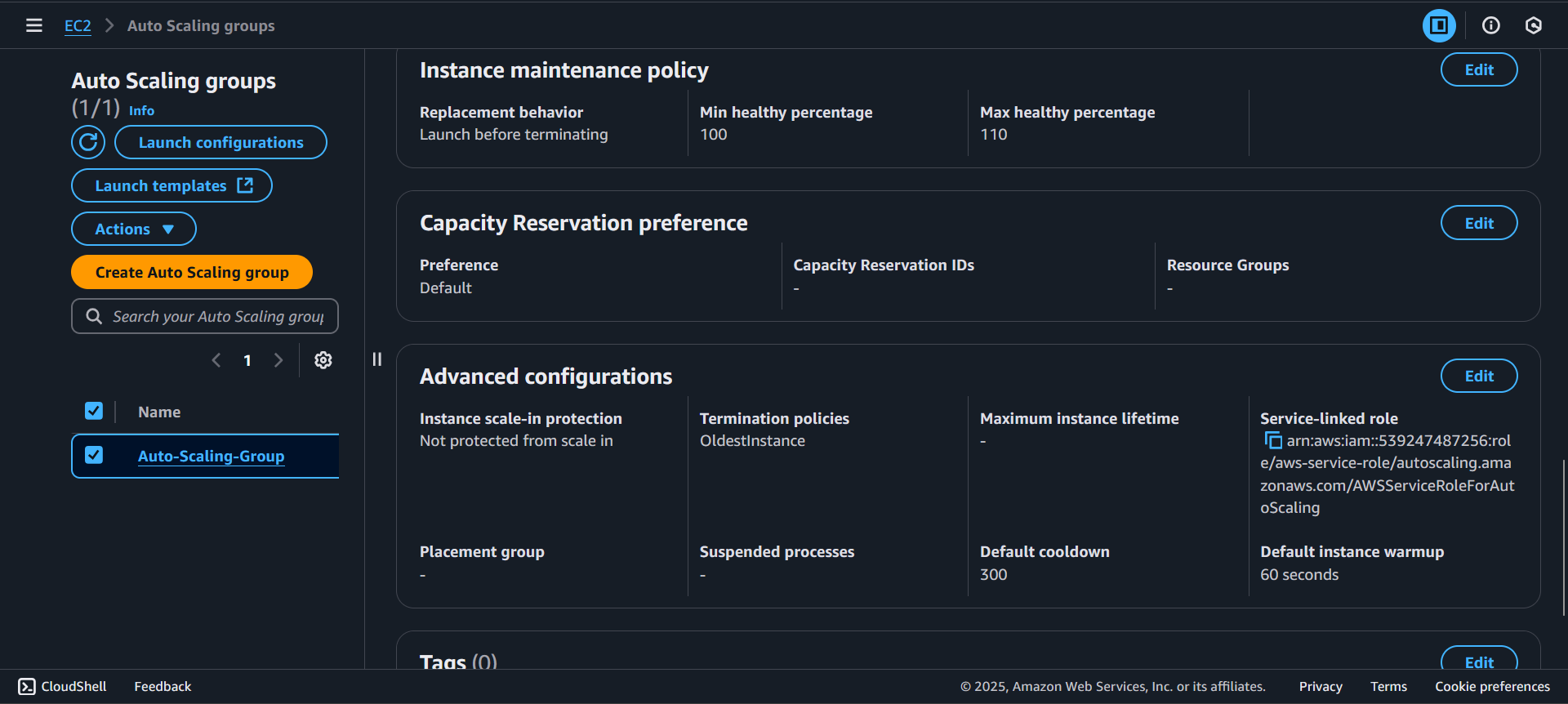
**Network Address Translator Gateway:**

****

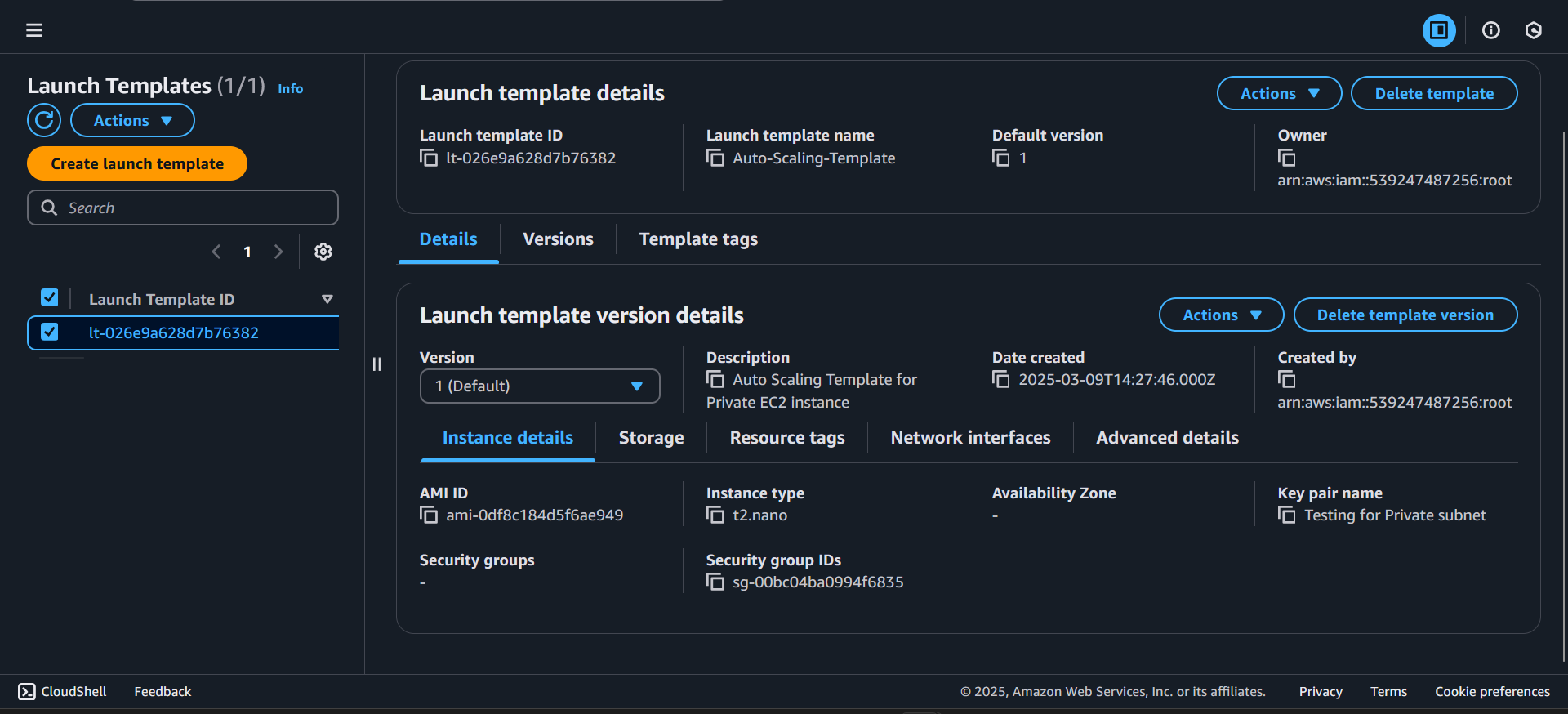
**Auto Scaling Group:**

****

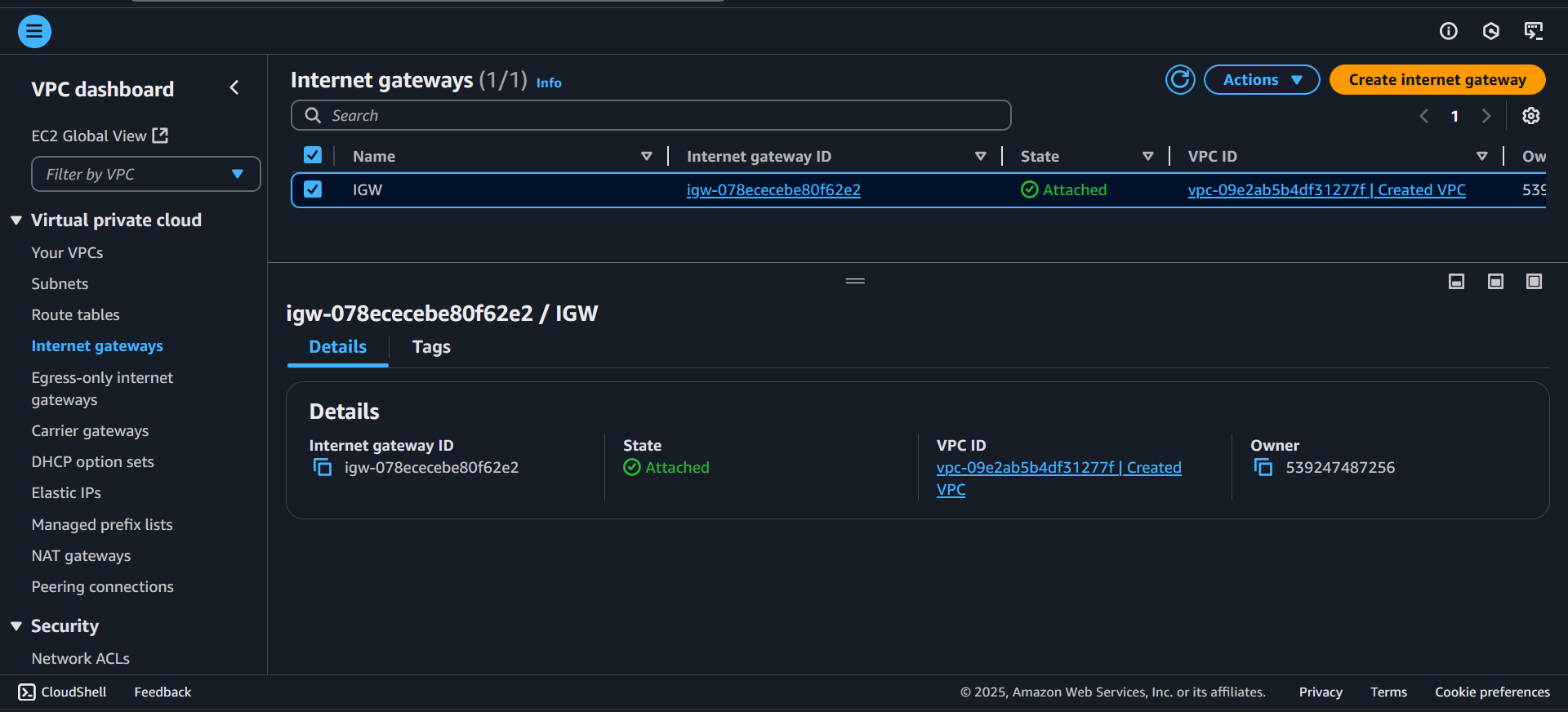
****

****

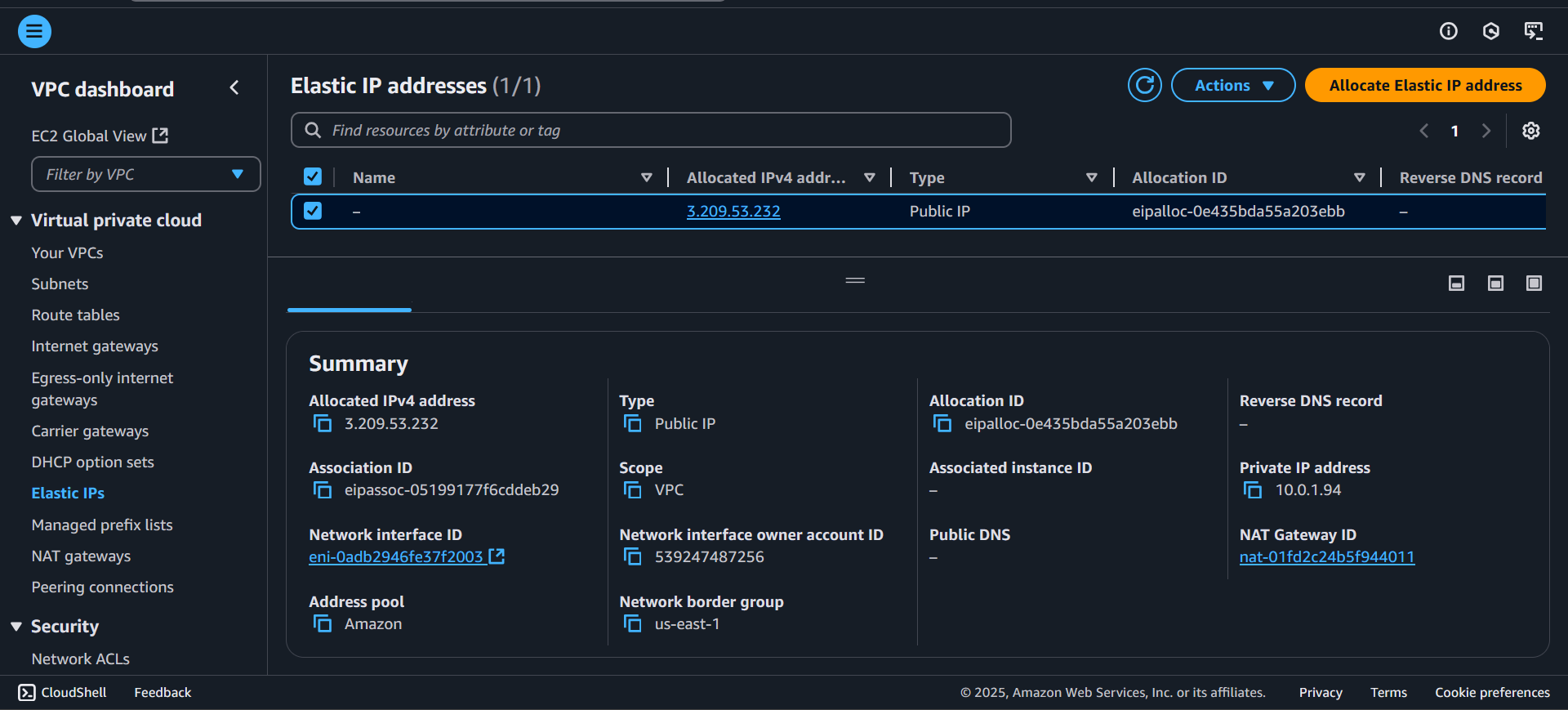
**Launch Template:**

****

**Internet Gateway:**

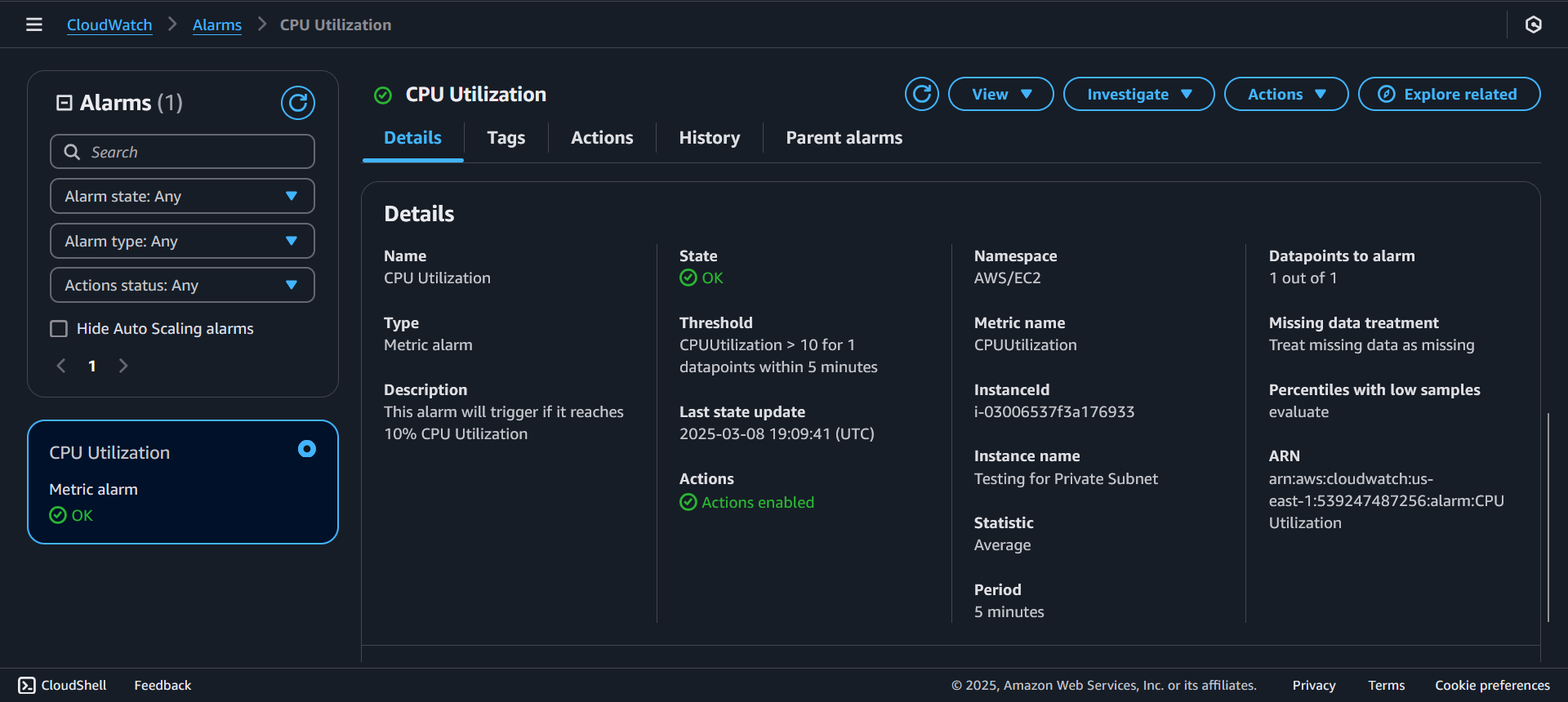
****

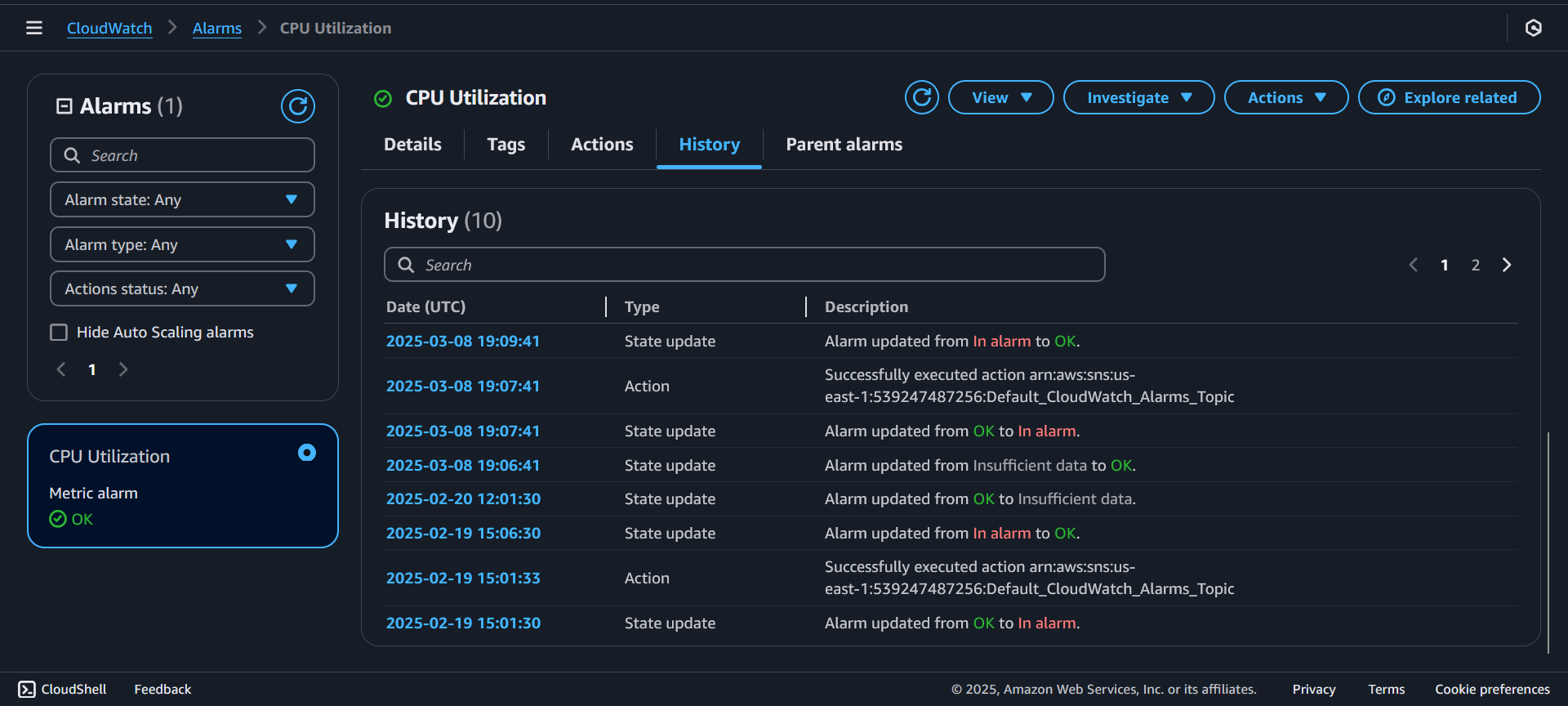
**Elastic IP:** VPC and EC2 dashboards should provide the exact same info about the Elastic IP

****

**Cloud Watch Alarm:** CPU Utilization

****

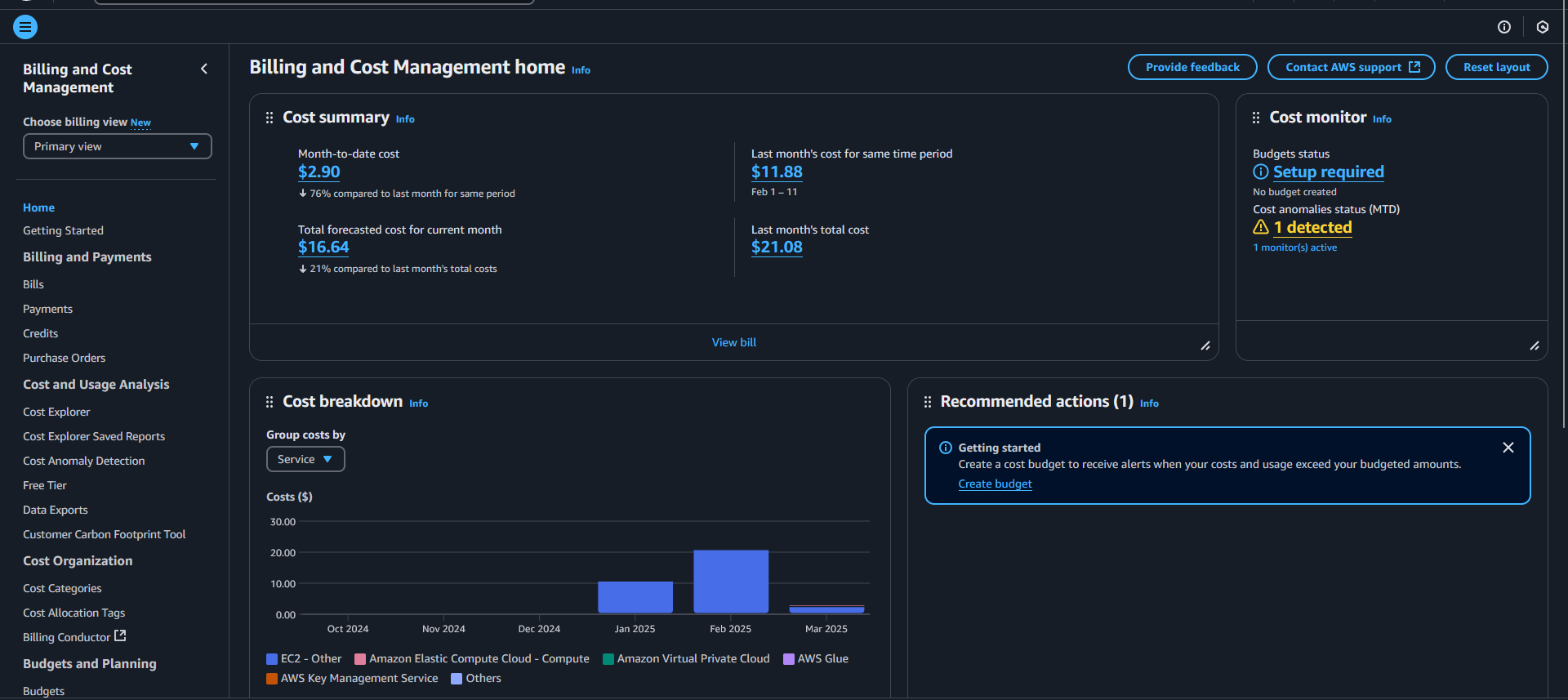
****

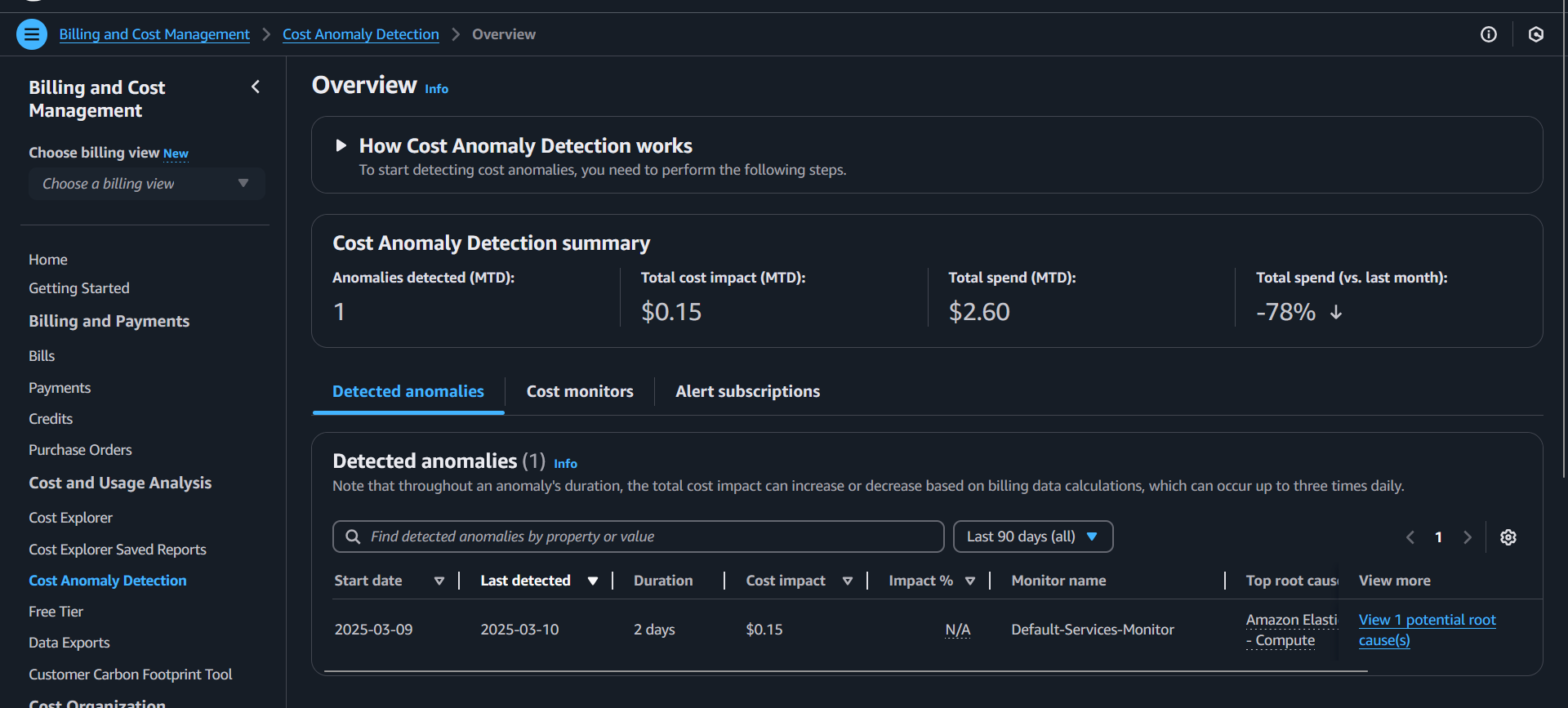
****

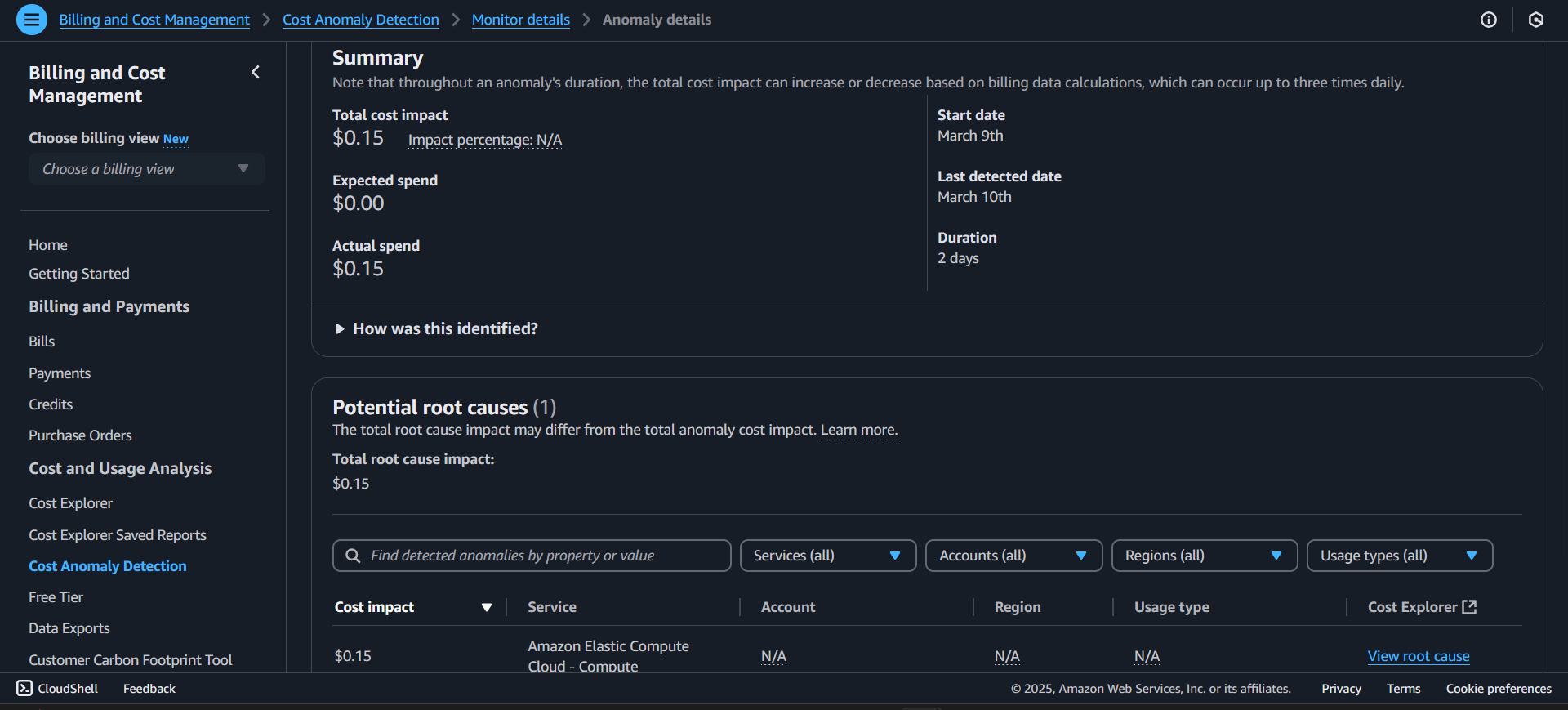
1. **Clean up Resources**

For this portion we just need to delete the EC2 Instance, Subnets, Security Groups, Route table, Network Access Control List, Network Address Translator Gateway, Auto Scaling Group, Launch Template, Internet gateway, release Elastic IP and Cloud Watch Alarm as well.

Begin by deleting the EC2 Instances by heading over to ec2 instance dashboard, select the instances that need to be deleted, press action, instance state and terminate instance and press confirm termination. Next head to auto scaling groups dashboard, select the ASG that was used, press action and select delete, this will also terminate all instances managed and previously created by the ASG. Next head over to Launch Templates dashboard, select the launched template that was created to be used by ASG, press action and select delete template. After that head over to Cloud Watch Dashboard, select any and all alarms that were configured, press action and select delete. Under Elastic IP’s Dashboard which is found under EC2’s Network & security, select the allocated Elastic IP that was used for the creation of Network Address Translator, press action and select release Elastic IP Address. That should stop any incurring costs caused by having the elastic IP. Next Head over to NAT Gateway dashboard, select the NAT Gateway, press action and delete NAT Gateway. That should also stop any incurred costs associated with the account. Head over to the Internet Gateway Dashboard, select the IGW, press action, select detach from VPC and press action again and select delete internet gateway, after it has been detached from the VPC. After that is done, head over to the Route table’s Dashboard, select the route table that are not the default ones created by AWS, check to make sure they were the ones that are designed to be attached to the custom made subnets that was created for the VPC project, press action and select Delete route table. Once that is done, go to the security group under VPC Dashboard, select the custom made security groups that was designed for this project, press action and select delete security group. If the security group is still actively associated with an instance then you will need to disassociate the security group from the instance prior to deletion. Since AWS does not allow you to delete them if they are still actively associated with an instance or network interface. Next, head over to the Network Access Control List (NACL) under VPC Dashboard, select the custom made ACL that was used for this project and press action then delete Network ACL. AWS will not let you delete the default ones created by AWS. After that is all done, head over to Subnet located under VPC Dashboard, select the custom made subnets that were used for the VPC Project, press action and select delete subnet. AWS may not let you delete the subnets until you delete all instances associated with the subnets. After that is done, head over to the VPC Dashboard, select the main custom made VPC that was created for the project, press action and select delete VPC then confirm deletion. That should be the final resource that needs to be cleaned up. If you receive any messages saying you cannot delete them due to association with other resources then those resources needed to be deleted first. Example: the route table and the Network ACL will not let you delete them individually instead you need to delete the VPC first and you will be prompted a message saying the following resources will be deleted with it. Additionally, it is always best practice to head over to the billing and cost management page under your account to identify all sources of all billing costs that are associated with the account. Head over to your account settings then select billing and cost management. Under the billing and cost management home dashboard, you can see all of the main metrics will tracking costs. If you head over to the Cost anomalies status 1 detected and dig further you will see some graphs relating to the cost each day.

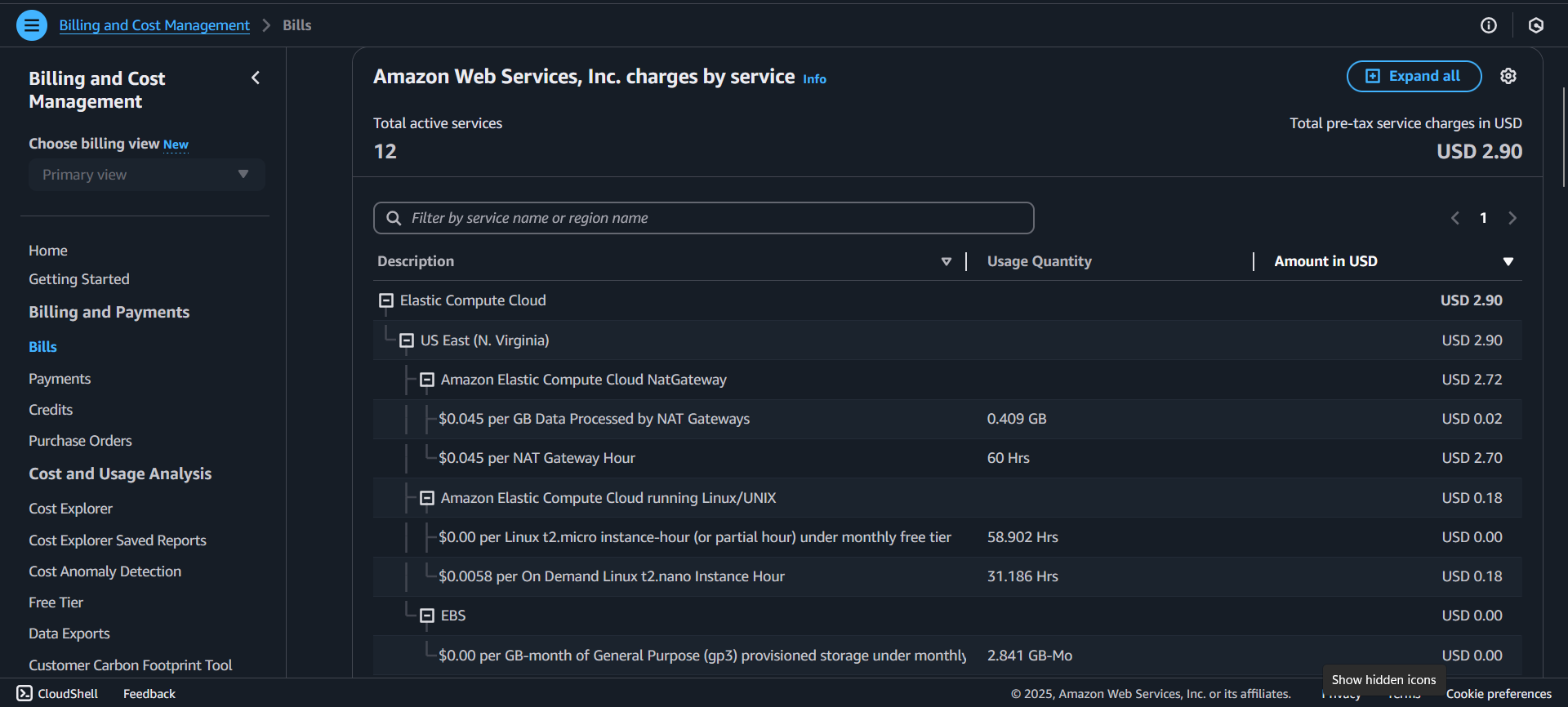


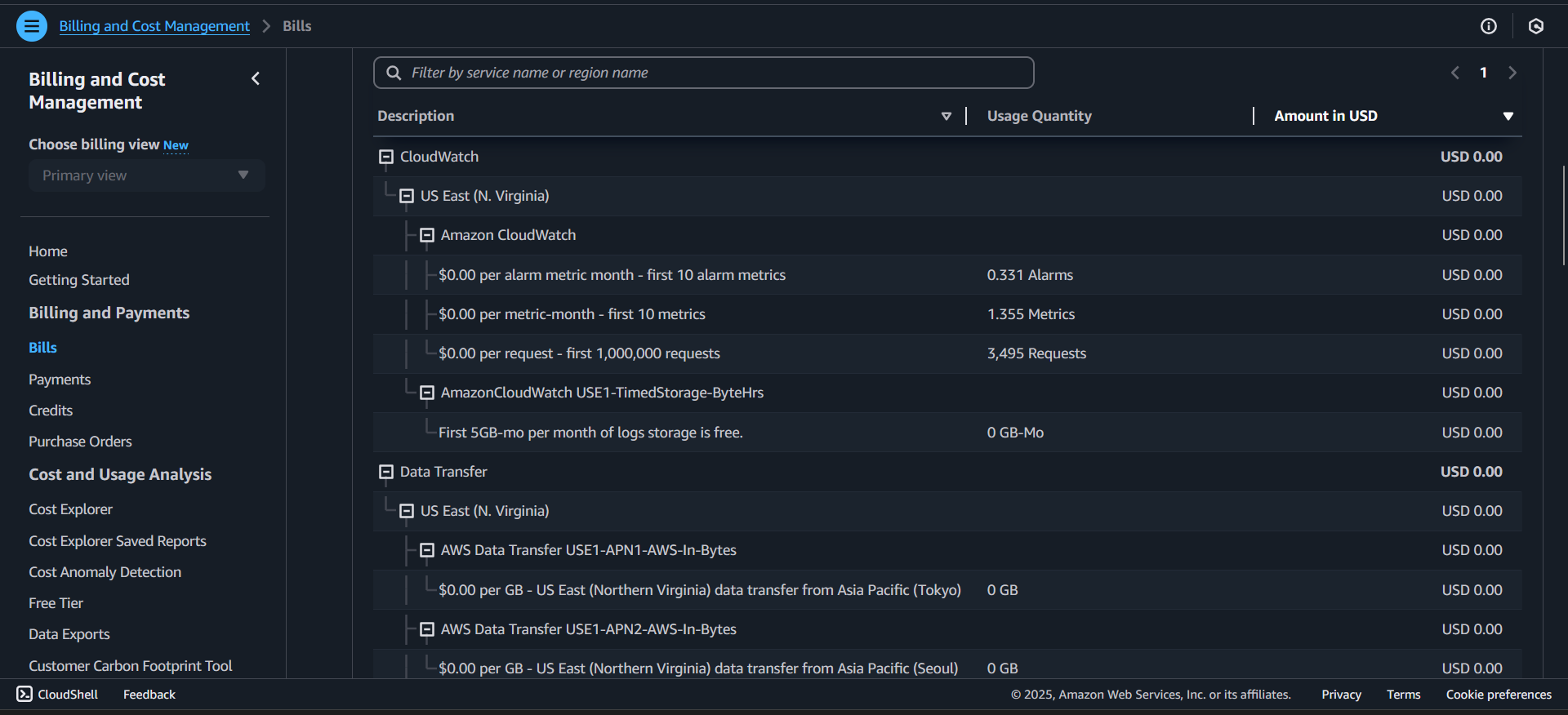


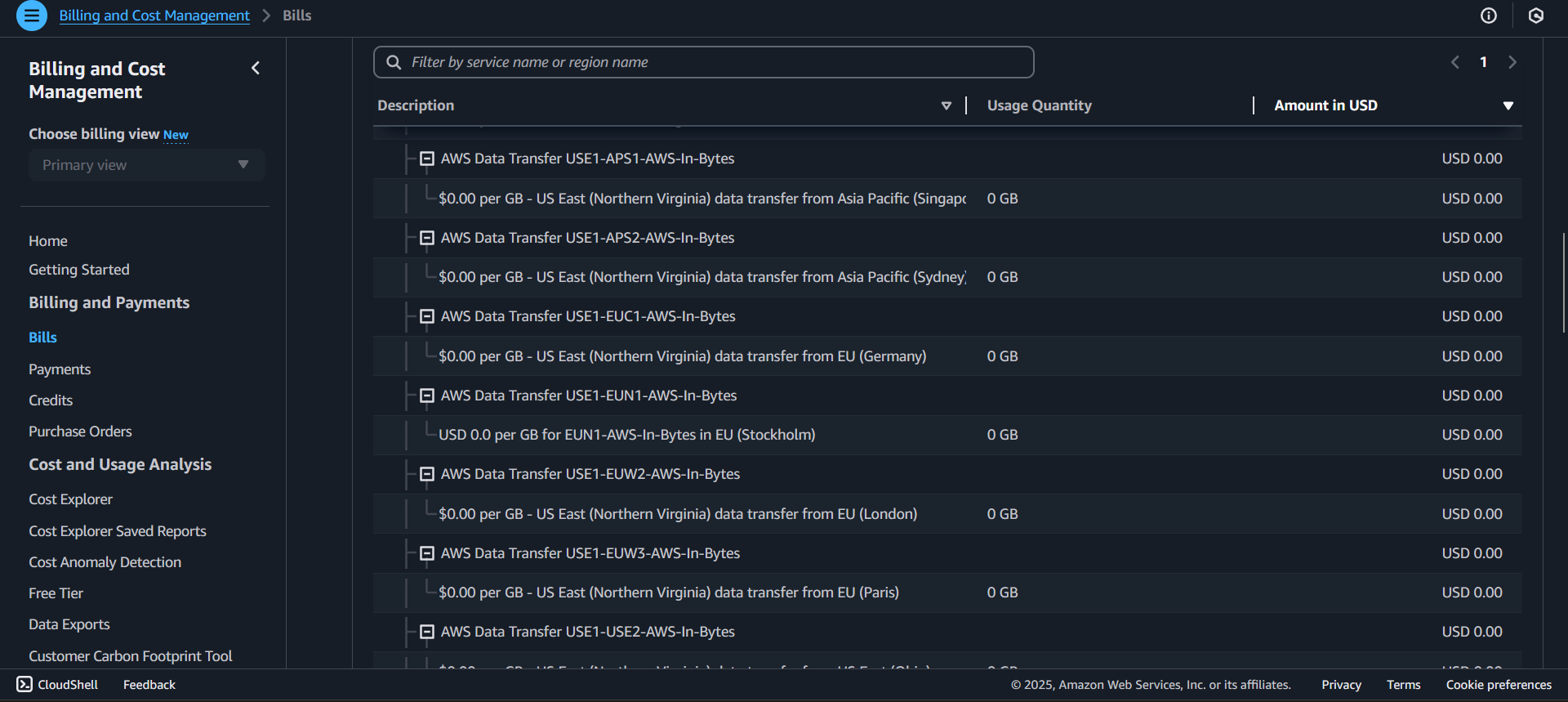


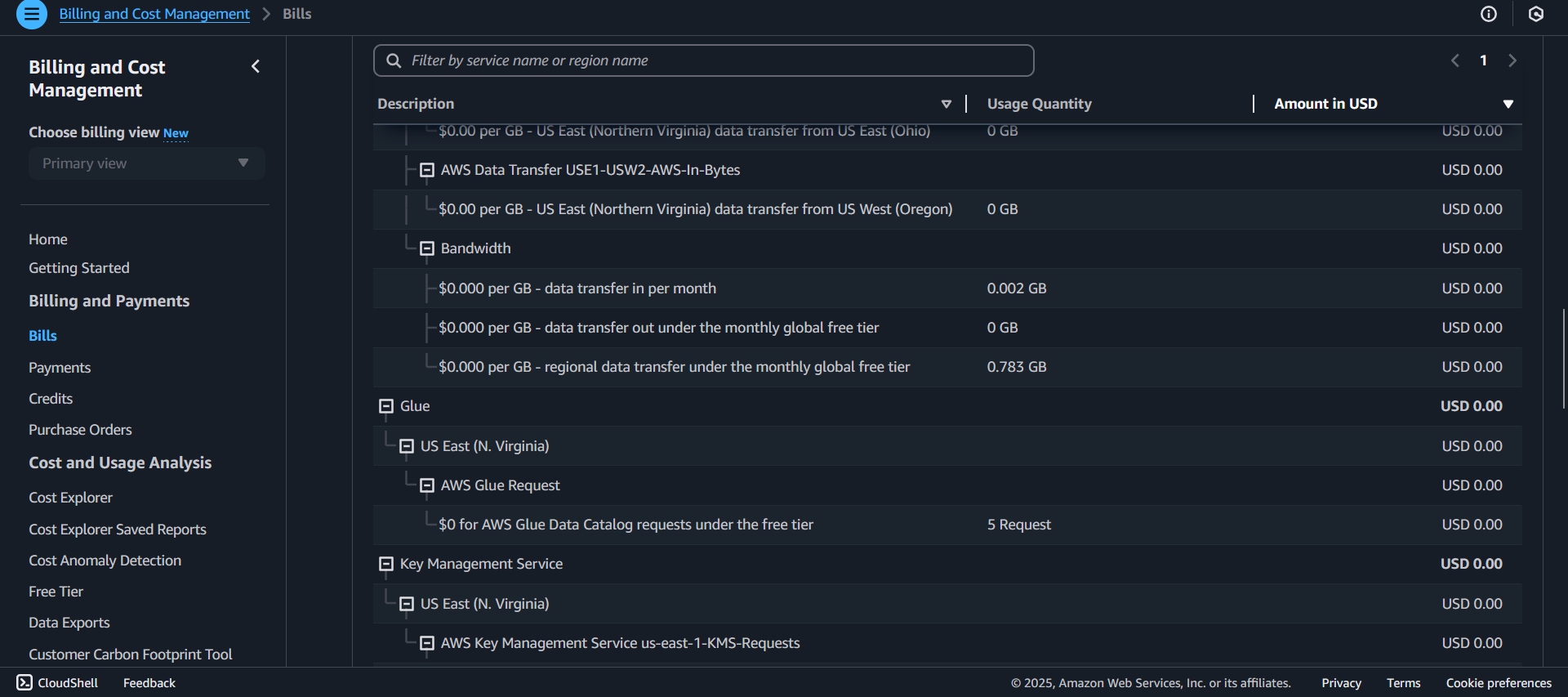


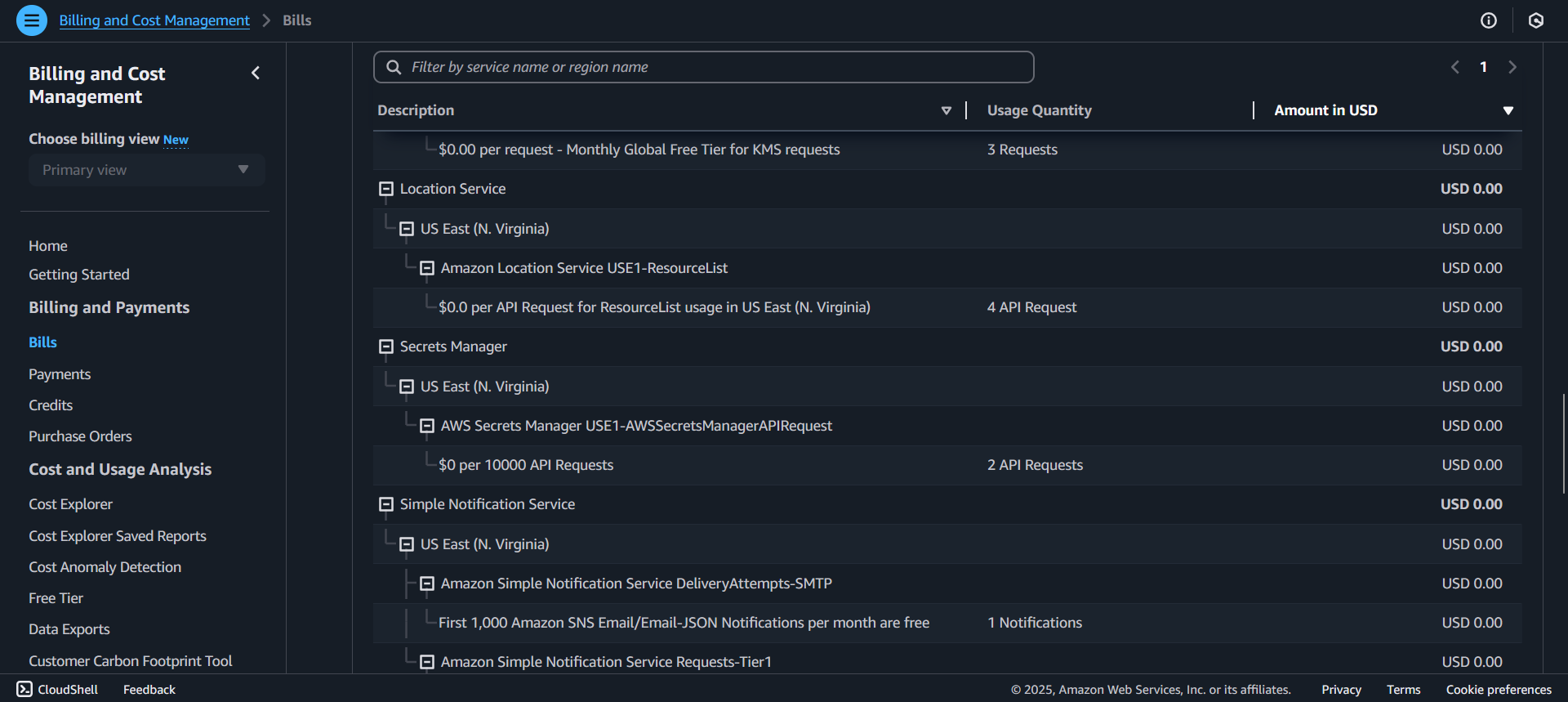
In order to truly find out the root cause of the cost increment with AWS services, we need to head over to the Bills section of the billing and cost management page. Then scroll down until you find Amazon Web Services, Inc. Charges by Services. If you navigate each item listed, it will tell you the actual root cause of the cost of using AWS as a cloud service. If you Press Elastic Compute cloud, under US East (N. Virginia), you will find out that the Amazon Elastic Compute Cloud NAT Gateway cost $2.72 since it cost $0.045 per GB Data Processed by NAT Gateways and the usage quantity is 0.409GB. Additionally, there’s 0.045 per NAT Gateway Hour at 60 hours of usage. Under Amazon Elastic Compute Cloud running Linux/UNIX, you will see it cost $0.18. We can see that $0.00 per Linux t2.micro instance-hour (or partial hour) under monthly free tier with 58.902 hours. At $0.0058 per On Demand Linux t2.nano Instance Hour with 2.841 hours of usage it cost $0.18. For the entire scope of the project, these are the main items listed that have caused the incurring costs of using AWS Cloud platform as a service. For our purposes, it is also best practice to show the rest of the possible incur costs that have been listed under charges by service, for future references. The complexity and duration of an AWS Project/Usage will greatly determine the overall cost of using AWS as a Cloud Provider Service.

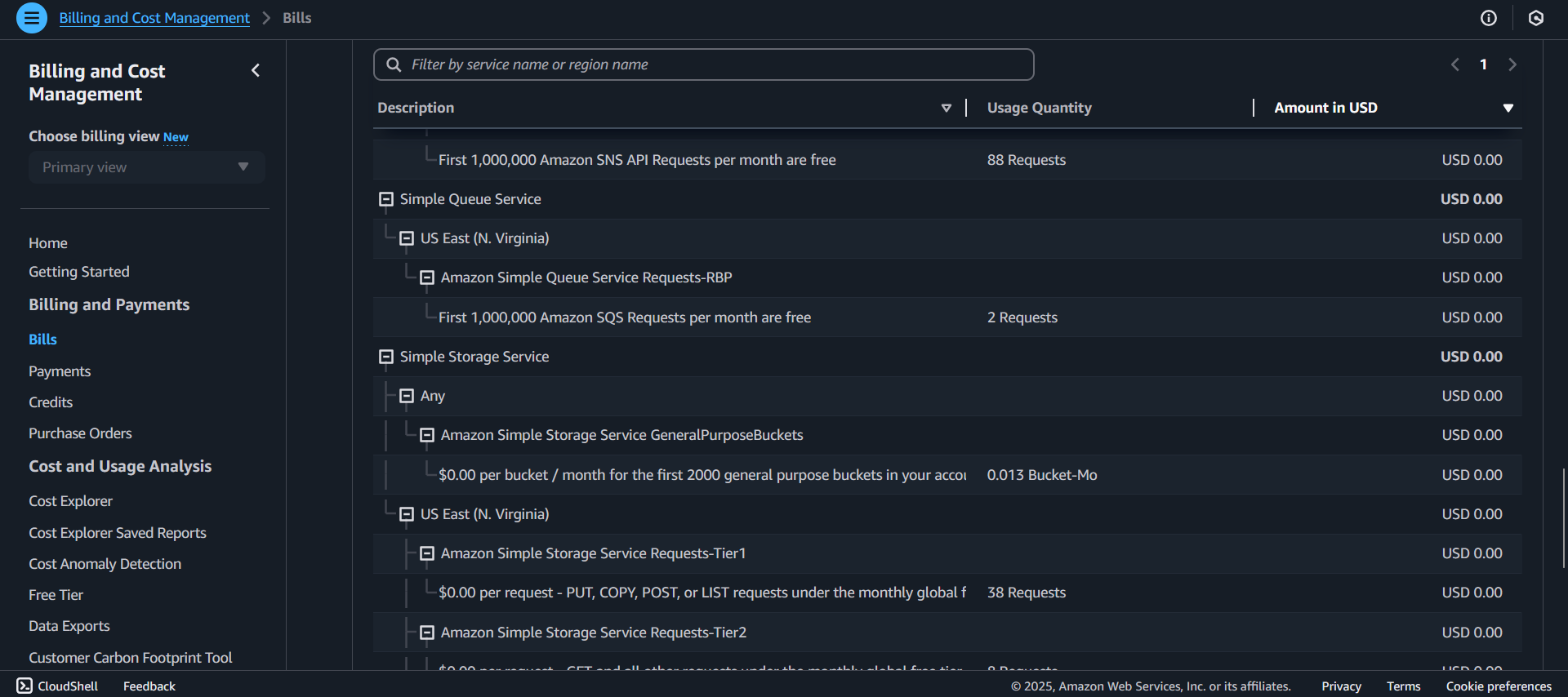


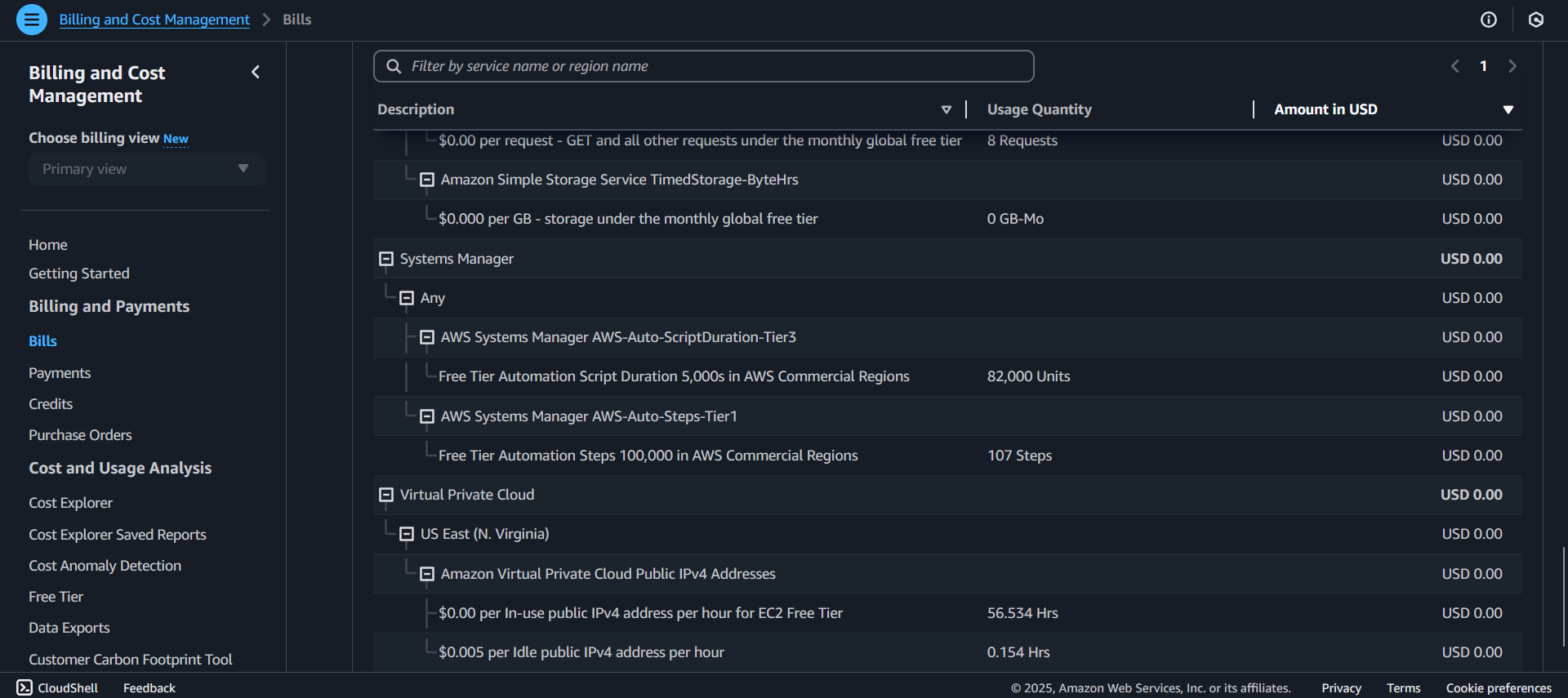












Final Thoughts: The purpose of this AWS Virtual Private Cloud Project was to demonstrate the ability to design and deploy a secure, high availability AWS VPC, configuring private and public subnets, create a firewall also known as a security group for configurating inbound and outbound traffic, establish a route table and utilizing an Elastic IP address for an NAT Gateway that helps control internet access for the Private EC2 instance. Implemented Auto Scaling for Fault Tolerance, monitored performance using Cloud Watch metrics and validated network security with Network Access control lists and security groups. Utilized AWS Systems Management (SSM)’s terminal for CLI commands, streamlining access management by eliminating open SSH ports and centralizing access management through IAM roles. Gained an insight into analyzing incur costs and billing optimization, as well as multiple AWS Service Interconnecting and communicating with each other.