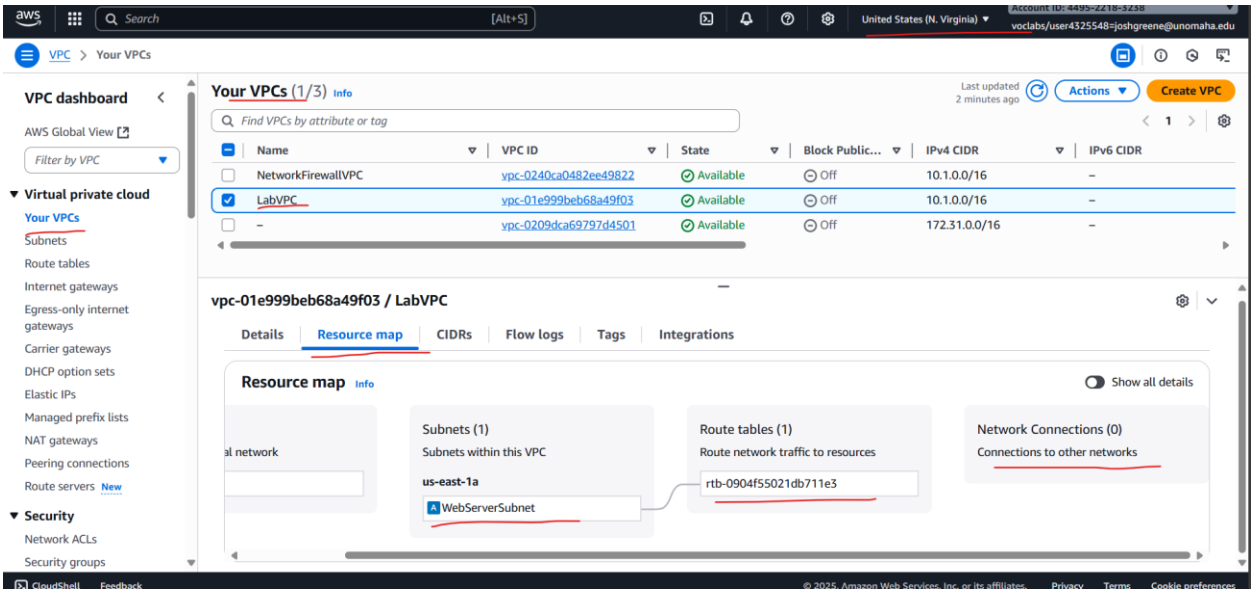
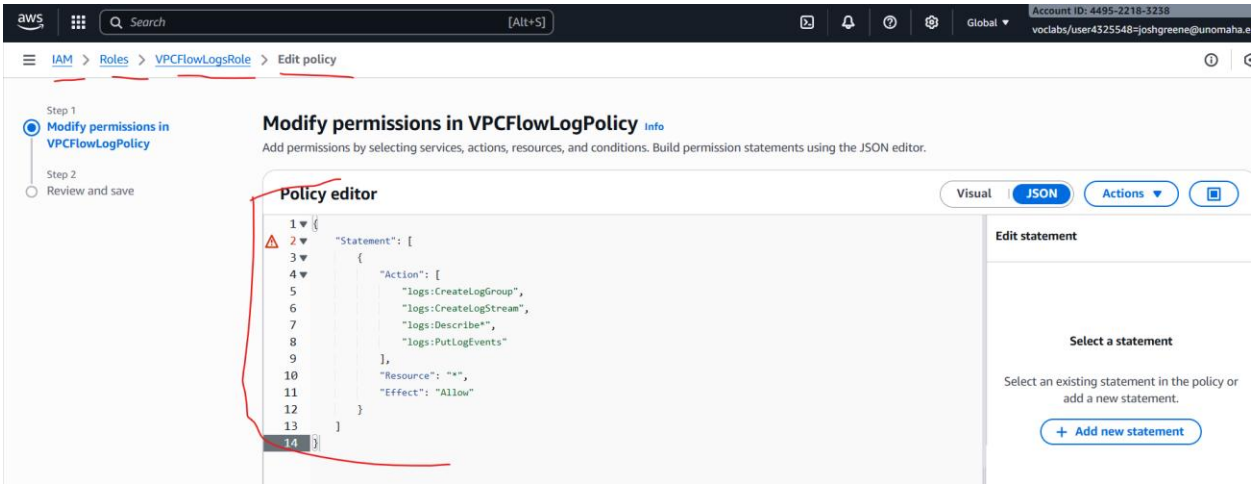
Securing VPCs

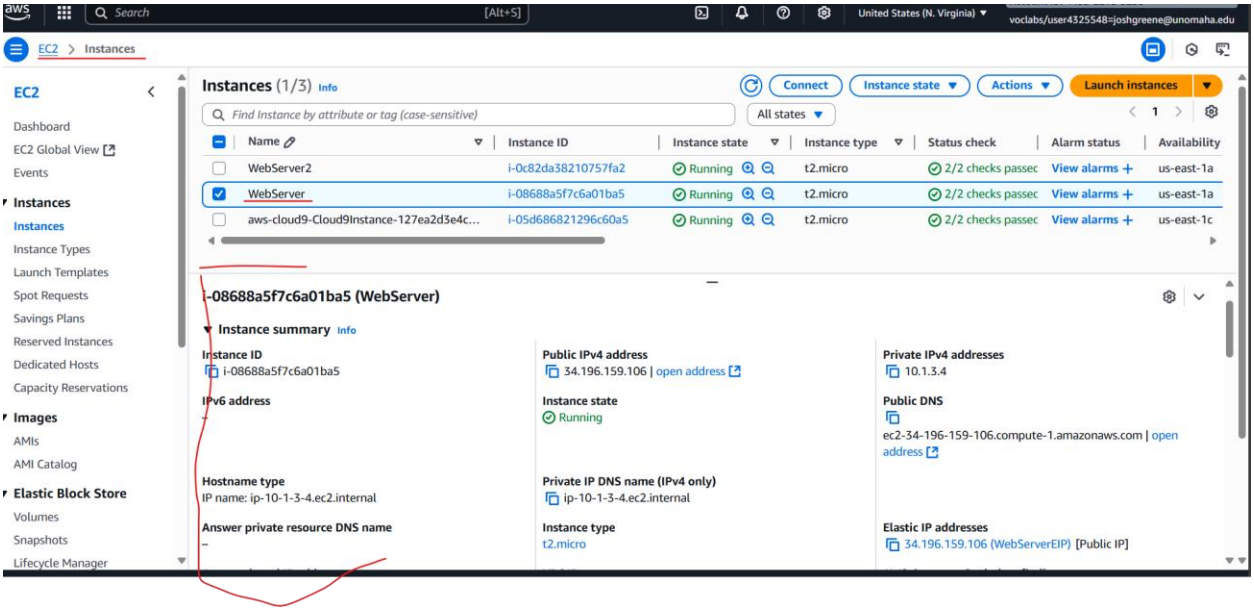
Task 1.1: Review LabVPC and its associated resources



Here I'm reviewing my Lab VPC in the North Virginia (us-east-1) Region. The resource map shows a subnet named WebServerSubnet and a default route table, but the subnet isn't yet connected to the internet gateway.

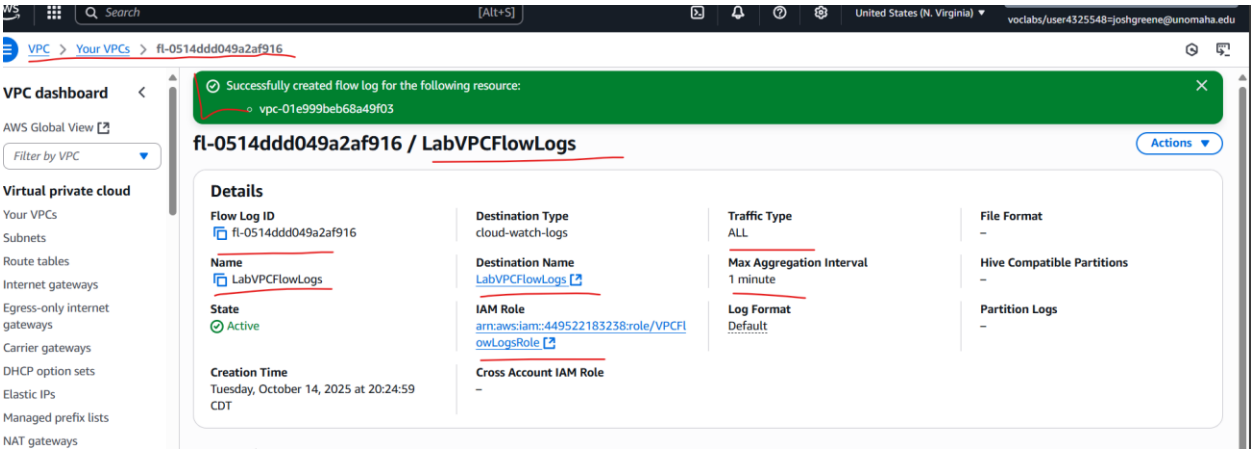


Here I checked the policy for the VPCFlowLogsRole. To get there, I opened the IAM console, went to Roles, searched for *VPCFlowLogsRole*, and clicked to view its policy. The policy includes permissions like logs:CreateLogStream and logs:PutLogEvents, which allow VPC Flow Logs to send data to CloudWatch Logs for monitoring network activity.



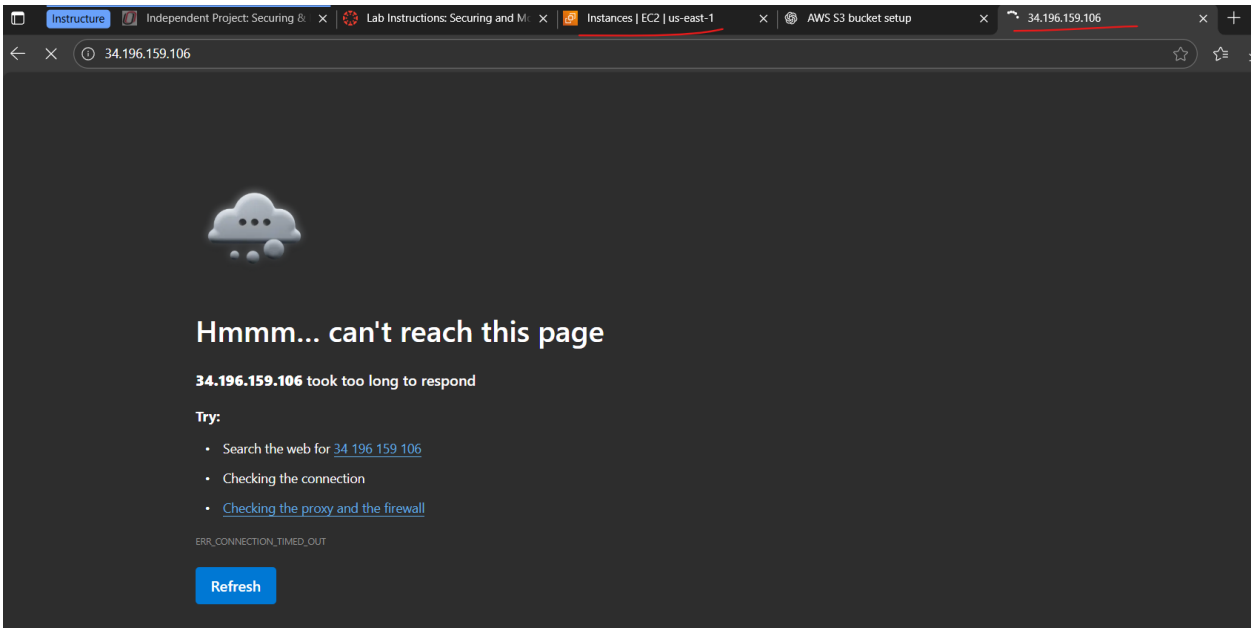
Now I'm reviewing my instance. To do this, I went to the EC2 console and selected Instances. From there, I opened the WebServer instance and looked at its details. I saw that it has a public IPv4 address, is running in the WebServerSubnet, and is using both an IAM role and a security group.

Task 1.2: Create a VPC flow log

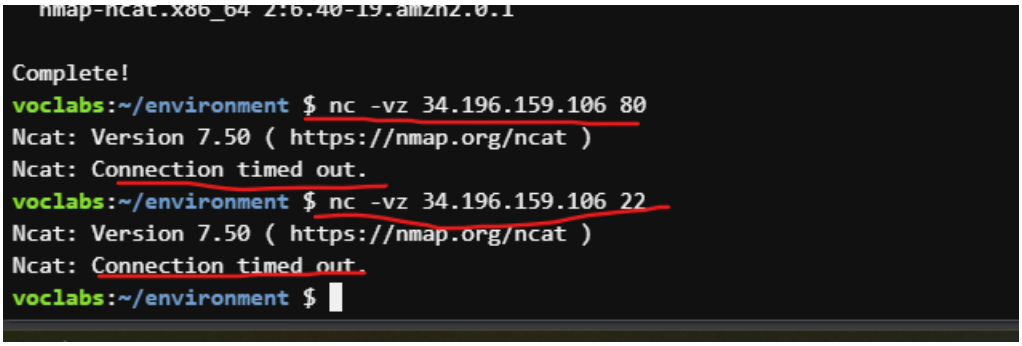


In the screenshot, you can see that I created a VPC Flow Log. To do this, I opened the VPC console, went to my LabVPC, clicked Create Flow Log, and configured the settings. I named it LabVPCFlowLogs, set the filter to All, changed the maximum aggregation interval to 1 minute, created a new destination log group named LabVPCFlowLogs, selected the LabVPCFlowLogs IAM role, and successfully created it.

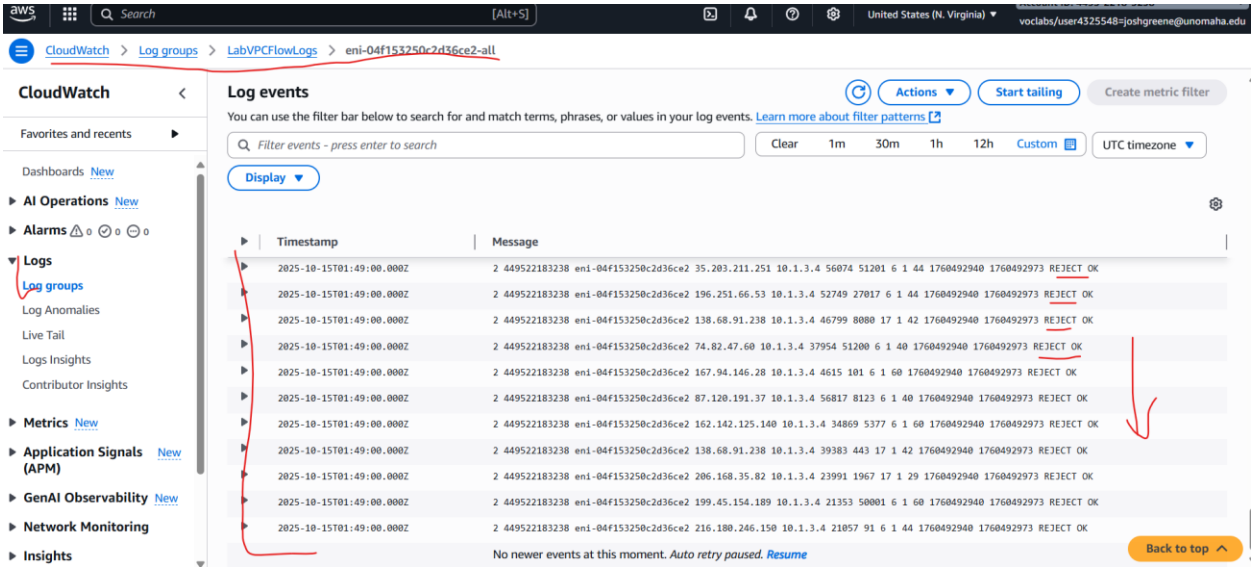
Task 1.3: Access the WebServer instance from the internet and review VPC flow logs in CloudWatch



Here I tried loading my WebServer instance using its public IPv4 address, but it didn't load, which was expected.



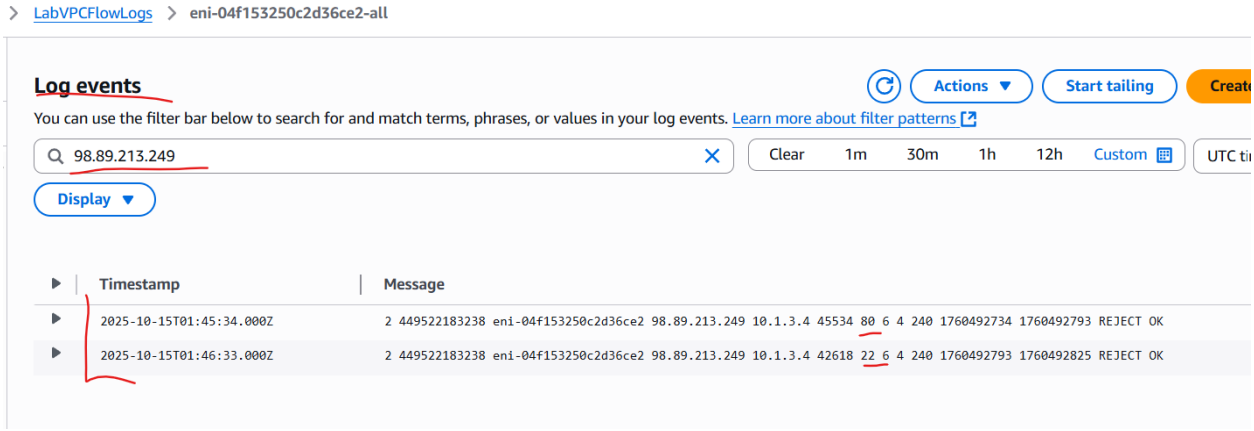
The next thing I did was test with Cloud9. To do this, I searched for the Cloud9 console and opened it. In the terminal, I ran both of the underlined commands, and as shown above, they both failed.



Now I'm checking the Flow Logs. To do this, I opened the CloudWatch console, went to Logs on the left-hand side, clicked Log groups, and selected LabVPCFlowLogs. In the screenshot, you can see a timestamp showing my logs from Cloud9 in the previous step, and it's marked as rejected, which I underlined above.

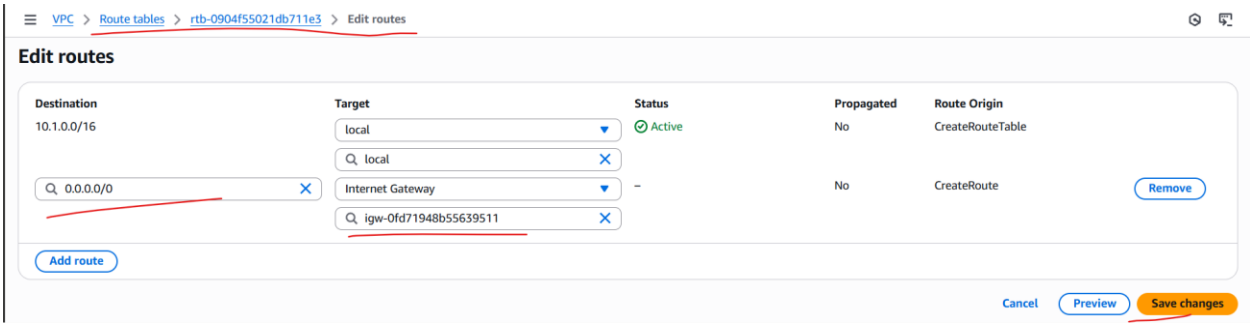
```
voclabs:~/environment $ curl http://169.254.169.254/latest/meta-data/public-ipv4
98.89.213.249voclabs:~/environment $
```

Here I ran this command to give me my public IP.



I took the IP address that the Cloud9 terminal gave me and pasted it into the search filter bar, which lets me view the specific events that occurred in my Cloud9 instance. Here you can see that both port 80 and port 22 failed.

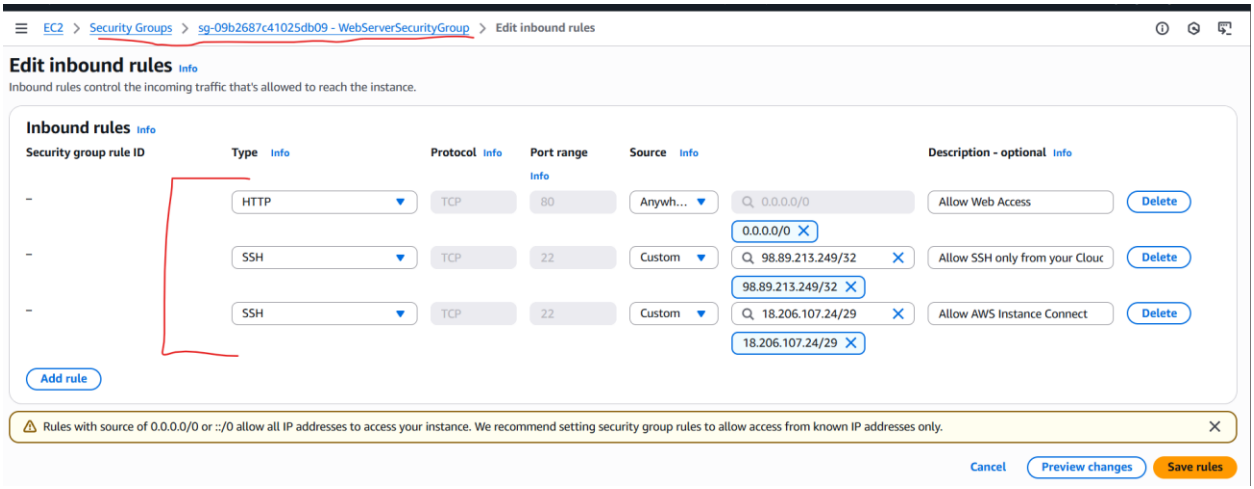
Task 1.4: Configure route table and security group settings



Here I added a destination for my LabVPC and set the target to LabVPCIG. This allows traffic to flow between my subnet and the internet.

```
voclabs:~/environment $ nc -vz 34.196.159.106 80
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connection timed out.
voclabs:~/environment $
```

Now I'm testing my Cloud9 IDE again. It still failed, which is expected. The route is ready, but my security group is still blocking the traffic.



Here I edited my WebServer's security group. I went to the EC2 console, opened Instances, selected my WebServer, followed the linked security group to Inbound rules, then updated and saved the rules shown above.

Hello world from WebServer!

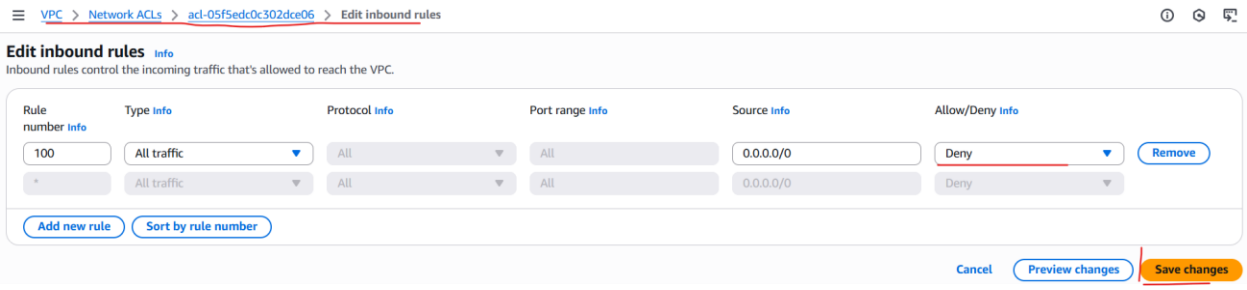
Now that I've finished setting up the security group, I'm testing my public IP. As you can see, it succeeded and displayed the output shown above.

```
voclabs:~/environment $ ping -c 3 www.amazon.com
PING e15316.dsca.akamaiedge.net (23.220.130.101) 56(84) bytes of data.
64 bytes from a23-220-130-101.deploy.static.akamaitechnologies.com (23.220.130.101): icmp_seq=1 ttl=52 time=1.2
6 ms
64 bytes from a23-220-130-101.deploy.static.akamaitechnologies.com (23.220.130.101): icmp_seq=2 ttl=52 time=1.3
3 ms
64 bytes from a23-220-130-101.deploy.static.akamaitechnologies.com (23.220.130.101): icmp_seq=3 ttl=52 time=1.3
5 ms

--- e15316.dsca.akamaiedge.net ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.268/1.317/1.350/0.055 ms
```

From my Cloud9 terminal, I ran the command shown above, and it returned an output of 64 bytes, confirming that my WebServer can now reach the internet.

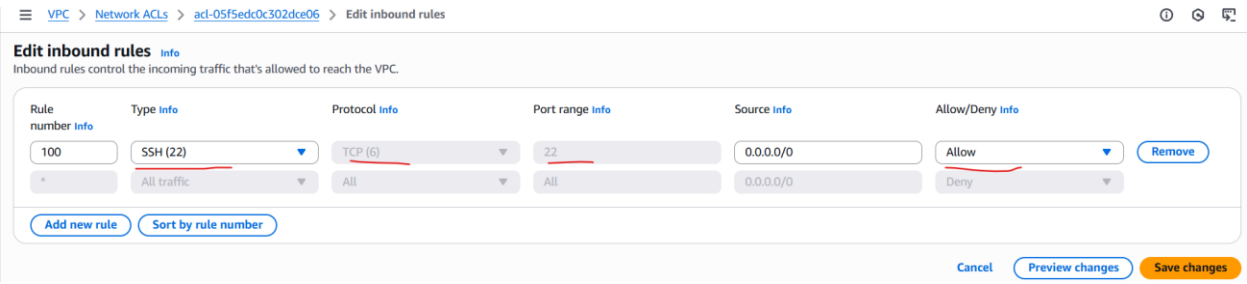
Task 1.5: Secure the WebServerSubnet with a network ACL



In this screenshot, I edited the inbound rules of the network ACL associated with my WebServer. I changed the inbound rule to Deny.

```
voclabs:~/environment $ nc -vz 34.196.159.106 22
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connection timed out.
```

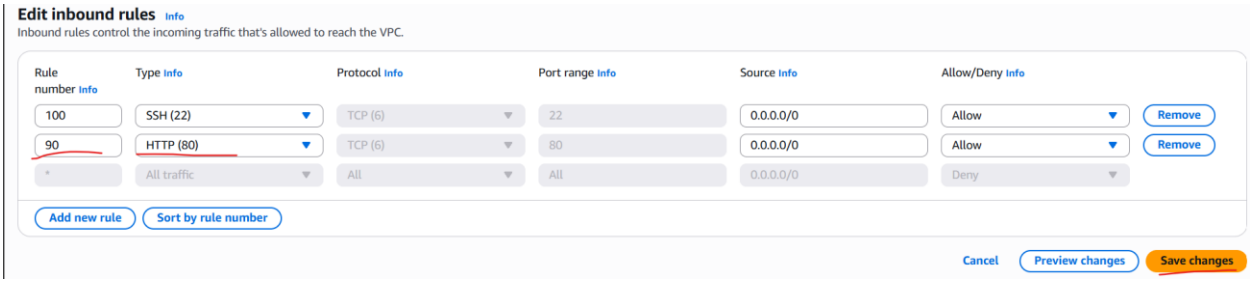
Since I made that change, you can see that when I ran the command in Cloud9, the connection timed out.



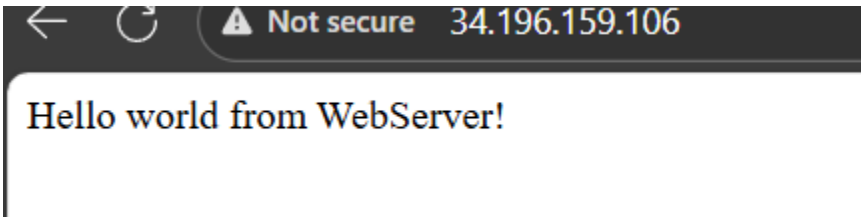
Now I'm editing the inbound rules again. This time, I changed the type to SSH (22).

```
voclabs:~/environment $ nc -vz 34.196.159.106 22
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 34.196.159.106:22.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
```

Here you can see that when I ran the command, it connected successfully this time.

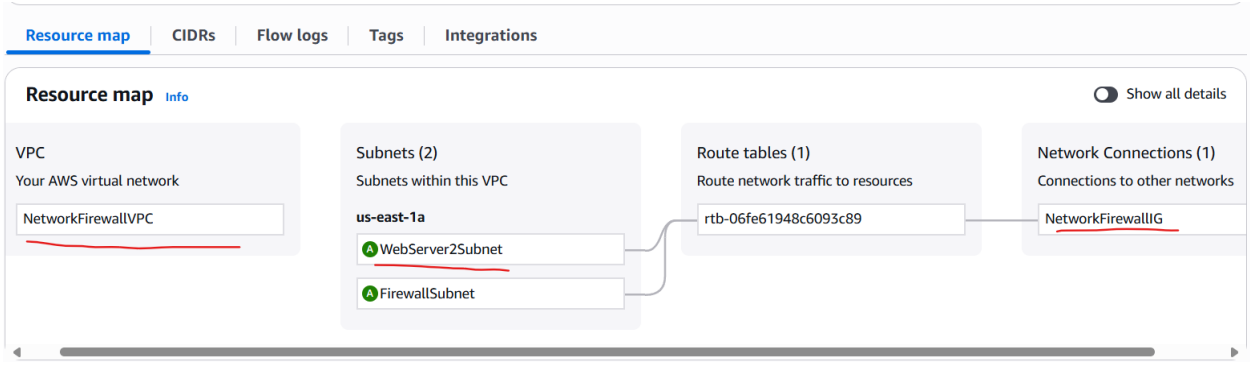


Now I'm adding another inbound rule, setting the rule number to 90 and the type to HTTP (80).

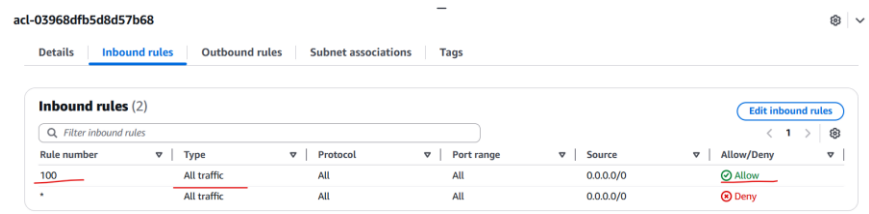


When I run my IP address, it shows the output above. This matters because the work I did with the Network ACLs protects the entire subnet, adding another layer of security on top of my security groups. I made sure that only ports 22 (SSH) and 80 (HTTP) are open, while all other traffic is blocked. Both my instance and subnet are now securely configured.

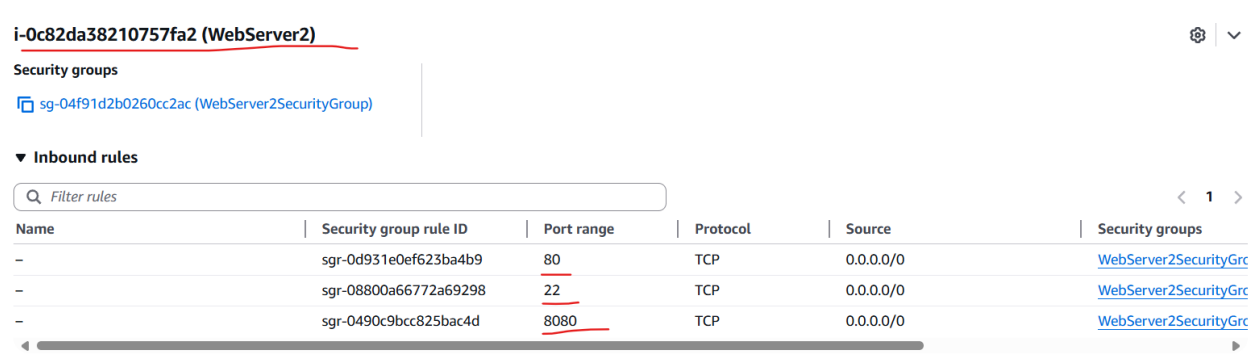
Task 1.6: Review NetworkFirewallVPC and its associated resources



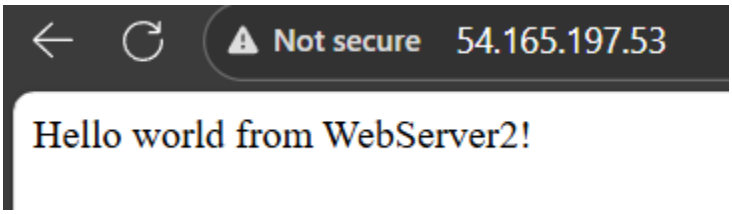
Here I'm reviewing my NetworkFirewallVPC setup. I opened it in the VPC console and looked at the resource map, where I found that it includes a WebServer2Subnet and a NetworkFirewallIG, which serves as the internet gateway for this VPC.



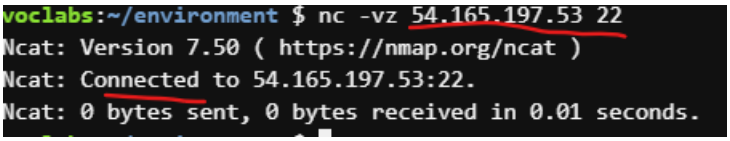
Here I checked the Network ACL associated with my NetworkFirewallVPC. On the details page, I looked at the inbound rules and saw that rule number 100 was set to allow all traffic, which is the default configuration. This rule lets all inbound network traffic reach the subnet until more specific rules are added later for tighter control.



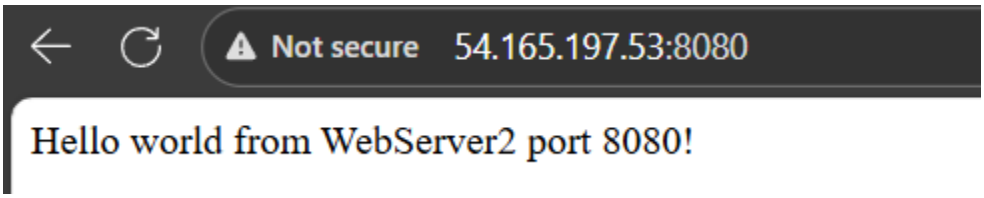
Here I checked the inbound rules for my WebServer2 instance. The screenshot shows that ports 80 (HTTP), 22 (SSH), and 8080 are all open for inbound traffic. These rules allow web, SSH, and custom traffic to reach the server from external sources.



Here I tested port 80, and it was successful. This shows that my web server is reachable and HTTP traffic is allowed.

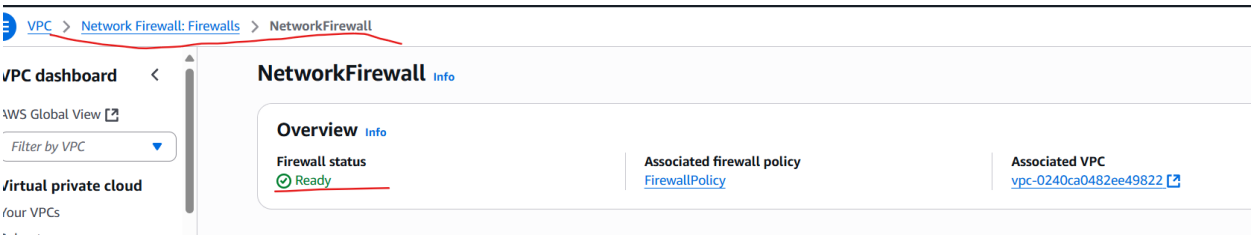


Here I tested port 22, and it connected successfully. This confirms that SSH access to my web server is working as expected.



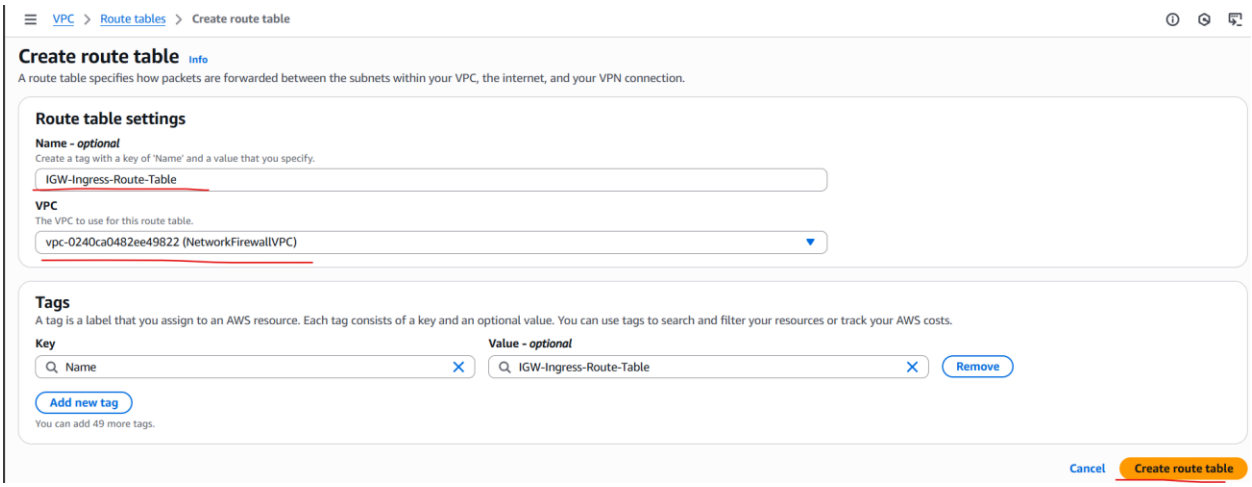
Here I tested port 8080, and it connected successfully. This shows that my web server is also accepting traffic on port 8080.

Task 1.7: Create a network firewall

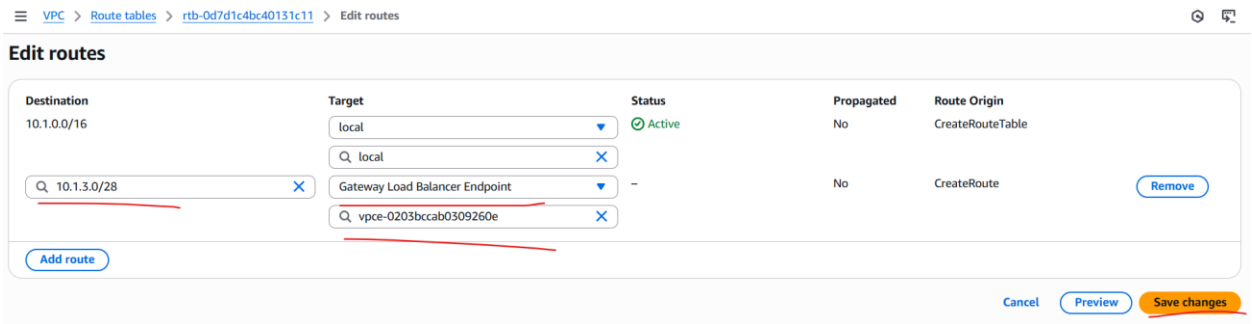


Here I created my Network Firewall in the VPC console. To do this, I opened the left menu, scrolled down to Network Firewalls under Security, and clicked Create firewall. I named it NetworkFirewall, associated it with my NetworkFirewallVPC and Firewall subnet, set the Availability Zone to us-east-1a, and unchecked both delete protection and subnet change protection. After clicking Create firewall, it took a few minutes to finish setting up.

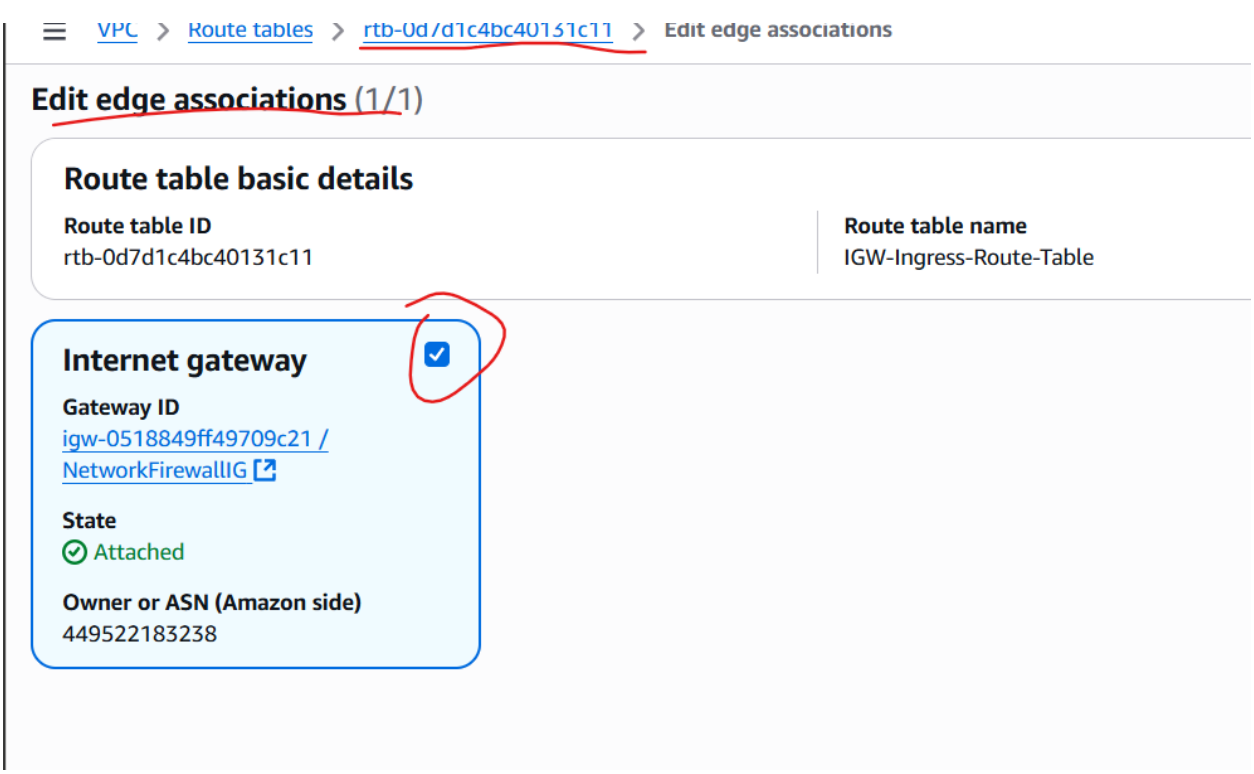
Task 1.8: Create route tables



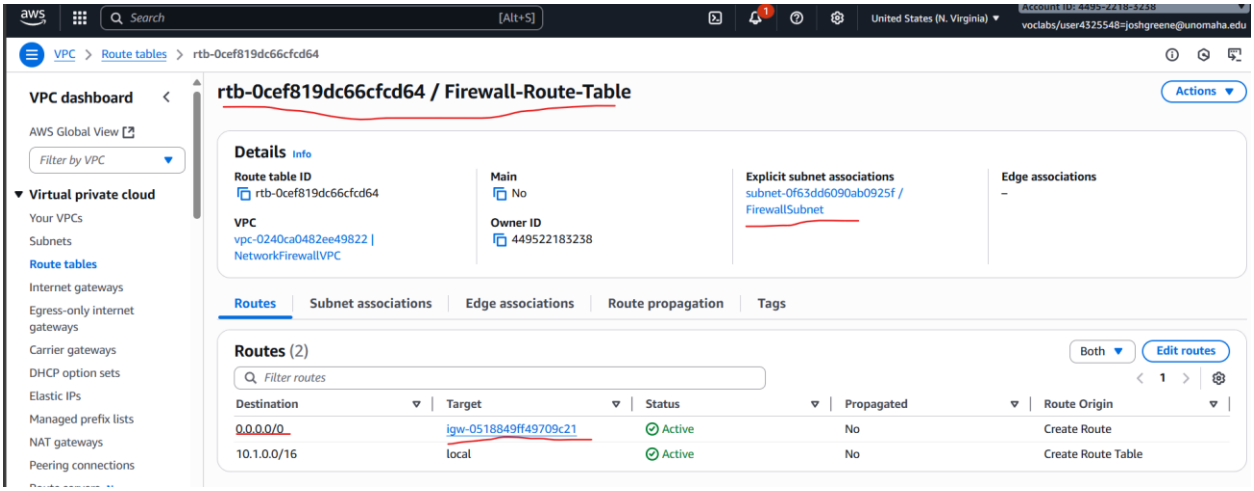
Here I created my route table in the VPC console. I went to Route tables on the left menu, clicked Create route table, and entered the information shown above. I made sure to associate this route table with my NetworkFirewallVPC, so it connects to the correct network environment.



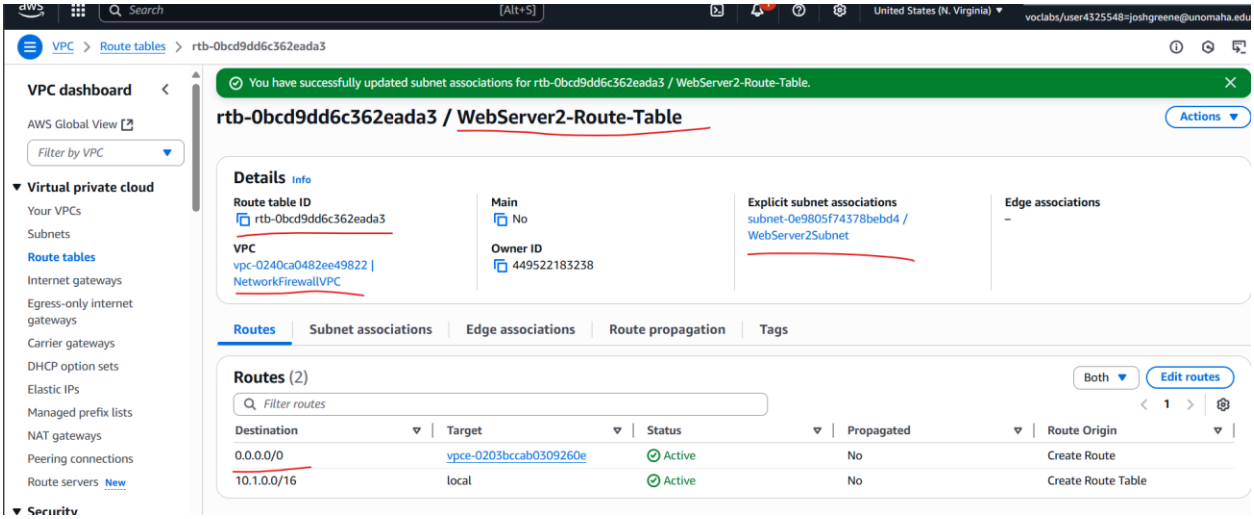
Here I added another route in my route table. For the destination, I used the CIDR block that matches my WebServer2Subnet, and for the target, I selected the Gateway Load Balancer Endpoint with the VPC endpoint (vpce) associated with my Network Firewall. This setup makes sure traffic from the internet can flow correctly through the firewall before reaching the web server subnet.



Here I edited the edge associations for my route table. I selected my internet gateway and attached it to the route table. This step links the internet gateway with the ingress route table so incoming traffic from the internet can route properly through the firewall and reach the internal network.

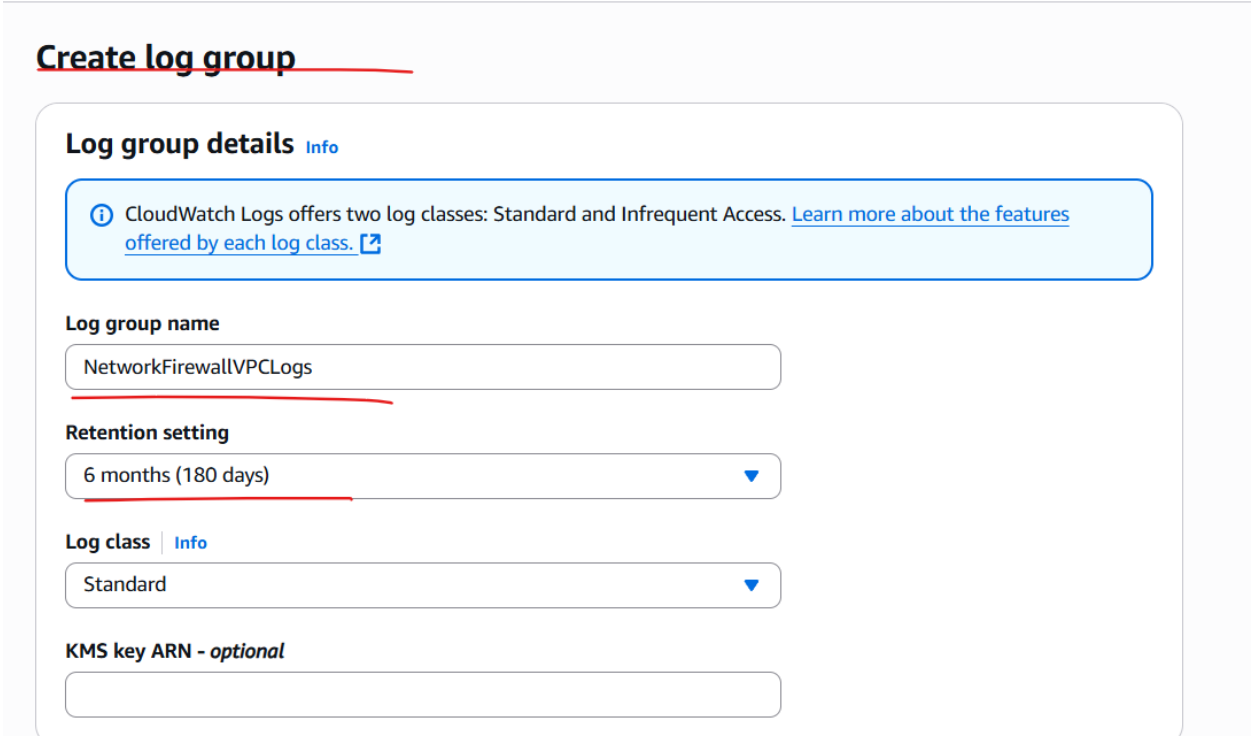


Here I created my firewall route table the same way as before. The only difference was that I associated it with my FirewallSubnet. This allows the subnet where the firewall sits to route traffic out to the internet through the correct path, keeping all outbound traffic monitored and controlled.

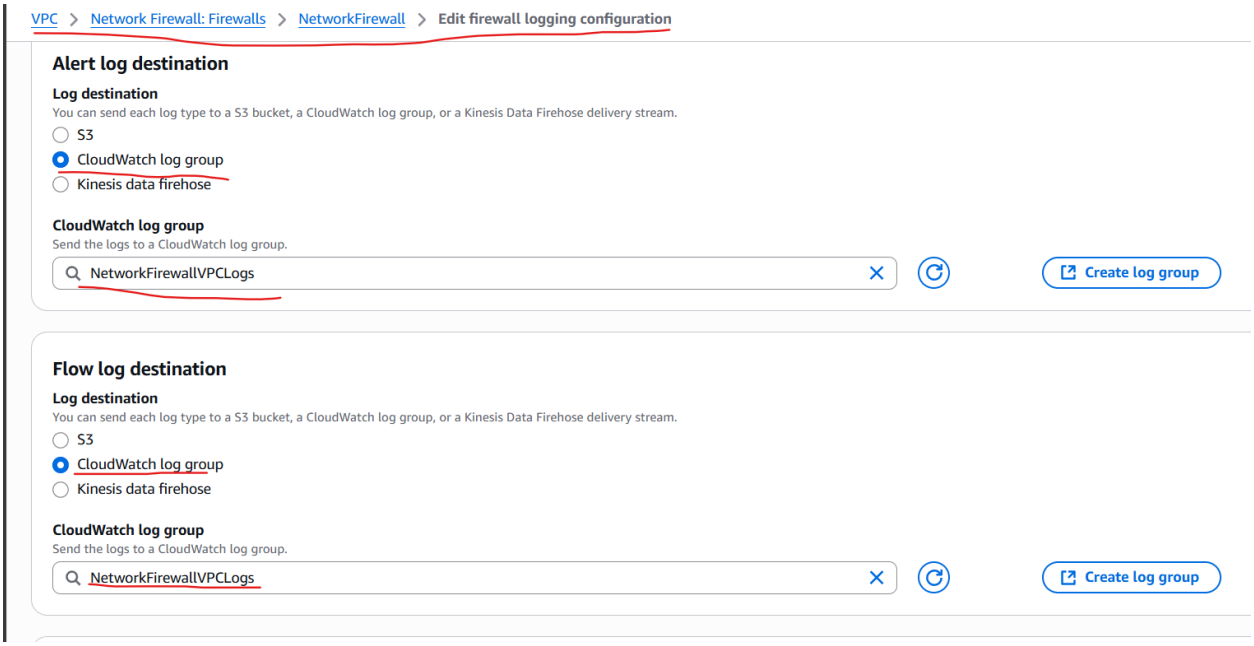


Here I created a route table for my WebServer2 using the same process as before. This time, I associated it with the WebServer2Subnet. With this setup, all traffic from my web server now routes through the Gateway Load Balancer Endpoint to the firewall before reaching the internet, ensuring that all network traffic is inspected and filtered for security.

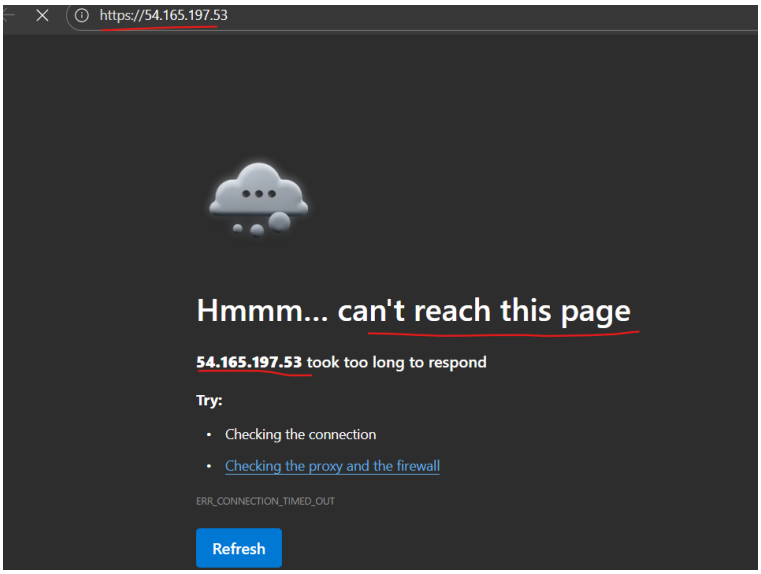
Task 1.9: Configure logging for the network firewall



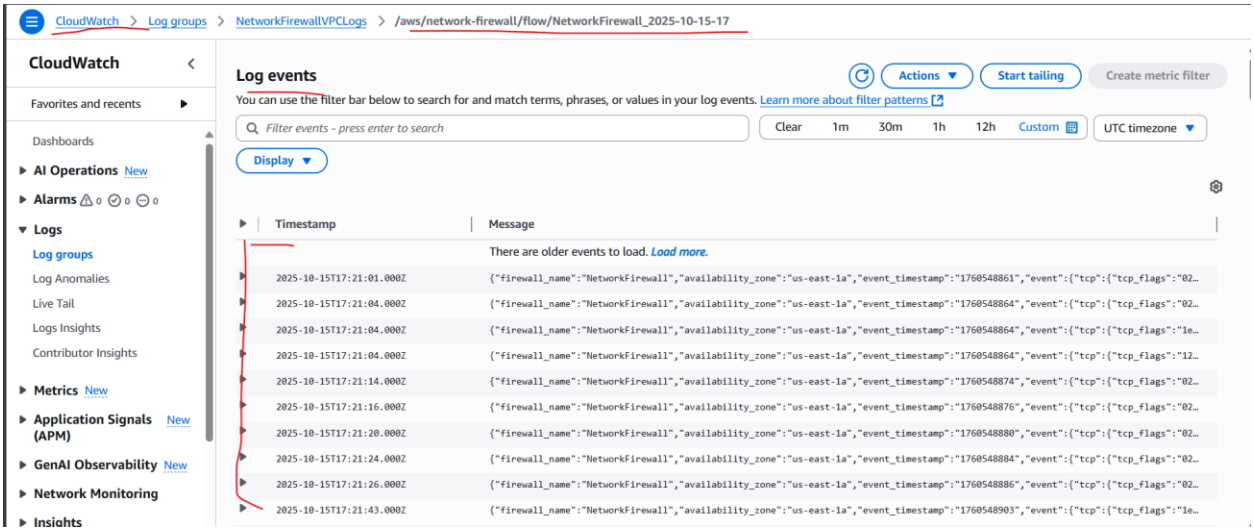
Here I created a CloudWatch log group named NetworkFirewallVPCLogs. While setting it up, I chose a retention period of 6 months so that logs are automatically deleted after that time. This log group will store both alert and flow logs from the firewall, helping me monitor and analyze network traffic over time.



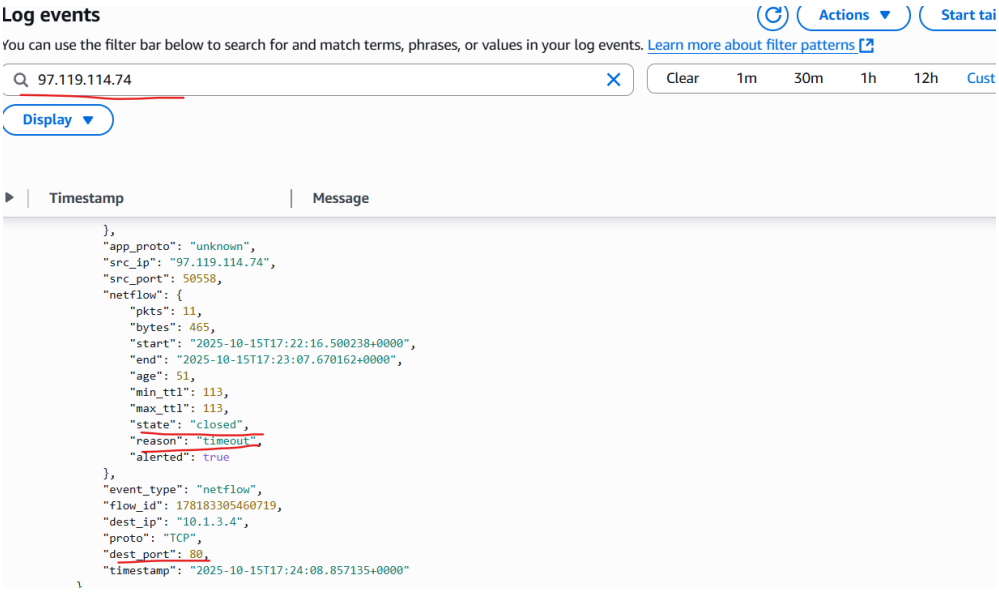
I connected the Network Firewall to CloudWatch by setting both Flow and Alert logs to go to my new log group. This lets me record every connection attempt and any blocked or suspicious traffic automatically.



I tested the firewall by trying to reach the web server. The timeout proves that the firewall is blocking HTTP traffic for now, which confirms it’s filtering properly.

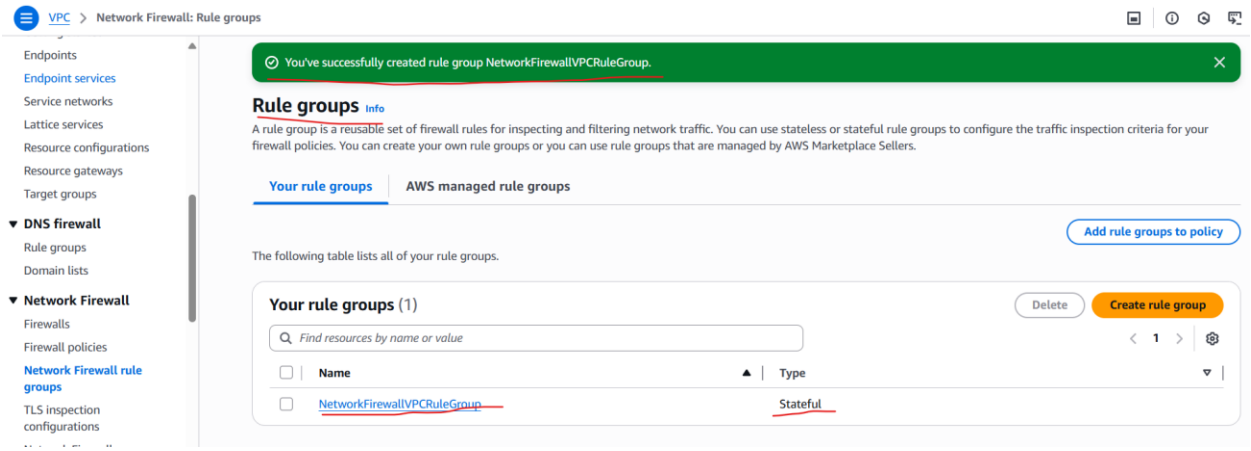


I checked the log group and saw new entries created by my blocked request. This shows that CloudWatch is successfully receiving data from the Network Firewall.

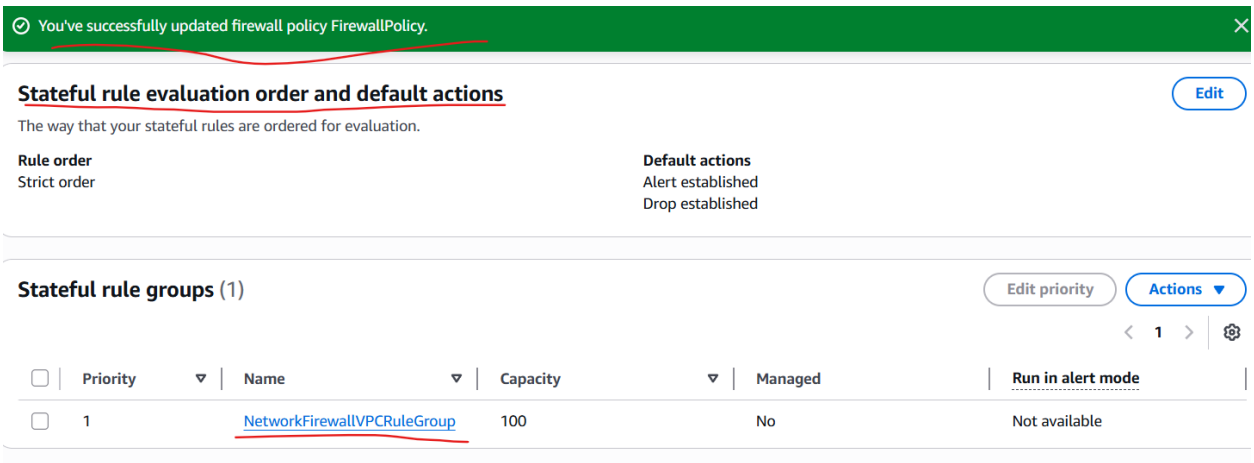


I filtered the logs by my IP to confirm my web request was recorded. The results show my IP and port 80, proving the firewall detected and logged my connection attempt.

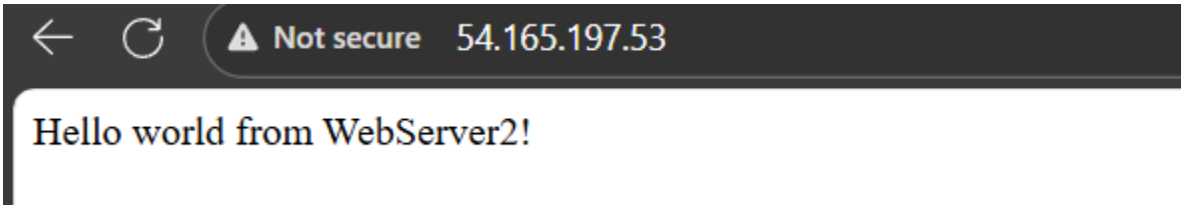
Task 1.10: Configure the firewall policy and test access



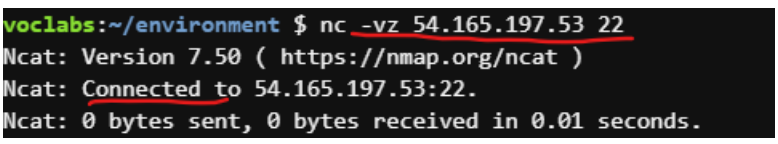
Here I've created a stateful rule group in the Network Firewall to control what traffic is allowed and denied in my VPC. To get here, I first went to the VPC console, selected Network Firewalls, and opened my existing firewall called *NetworkFirewall*. Then I clicked Create rule group, chose Stateful, set the rule order to strict, named it *NetworkFirewallVPCRuleGroup*, and gave it a capacity of 100. After that, I added five rules — one to drop traffic on port 8080 and four to allow traffic on ports 80 (HTTP), 22 (SSH), 443 (HTTPS), and ICMP (ping). This setup ensures that only secure and necessary connections can reach my web server while blocking unwanted ones.



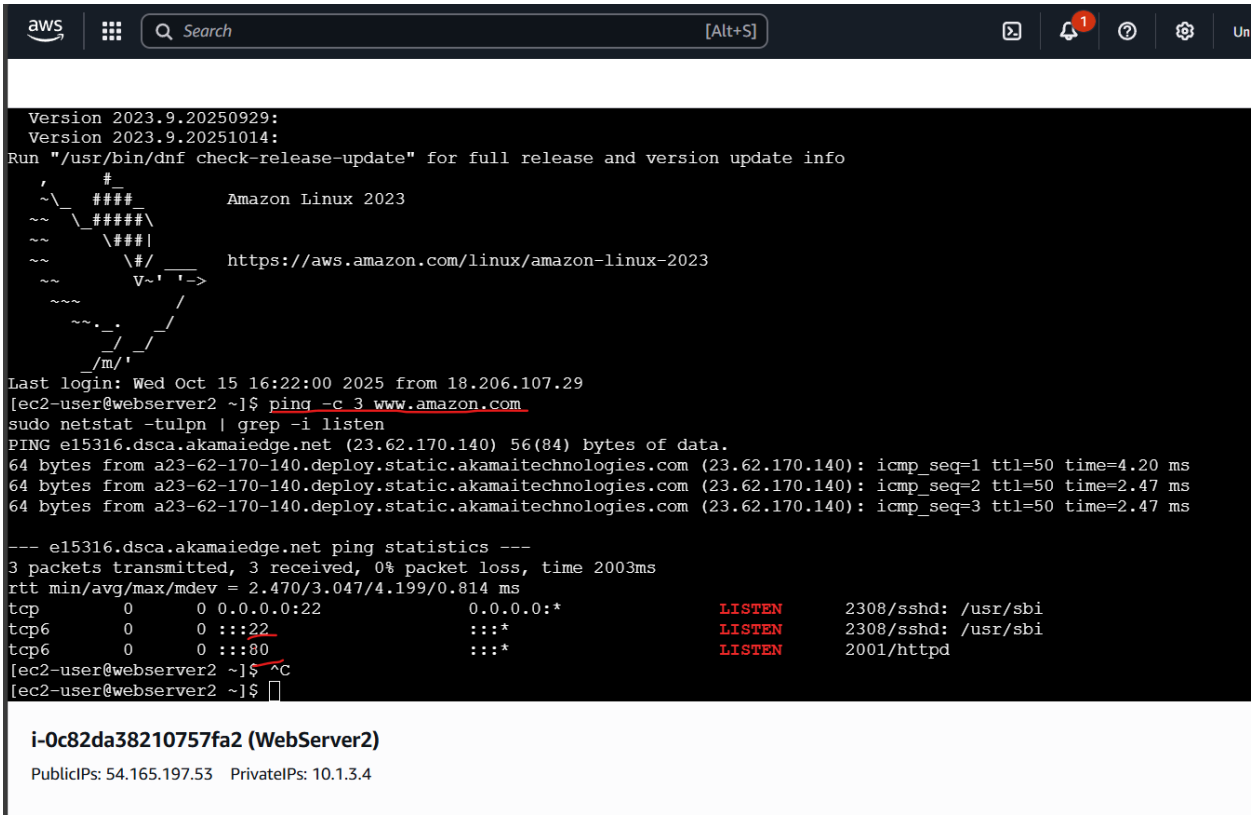
Here ive added my stateful rule group to the FiewallPolicy



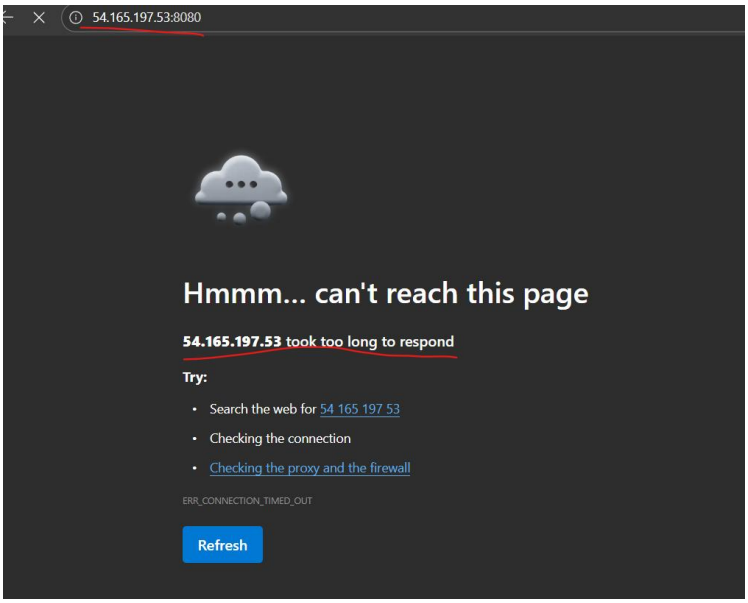
Here, I tested my new firewall rules by loading the WebServer2 webpage. It successfully displayed the "Hello world" message, confirming that HTTP (port 80) traffic is allowed.



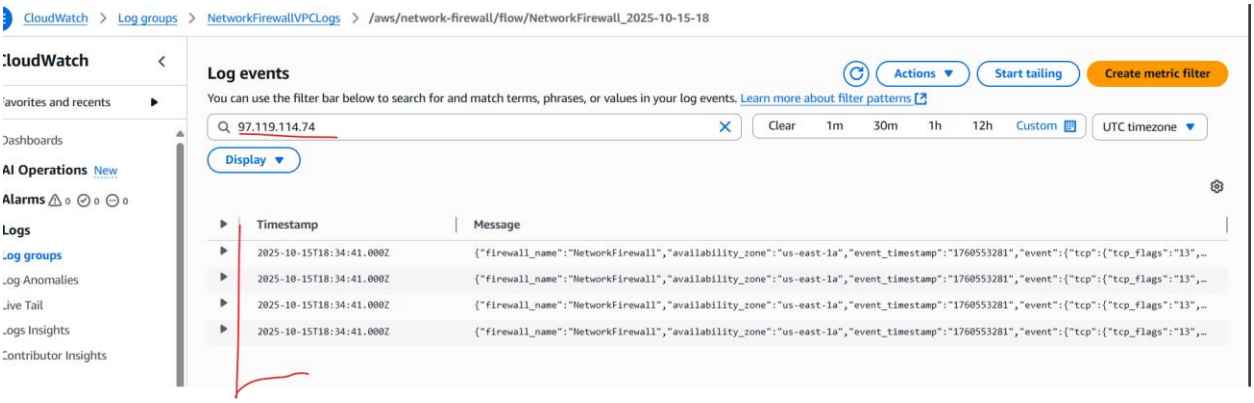
I used the nc command from Cloud9 to test SSH connectivity. The connection succeeded, confirming that SSH is open through the firewall.



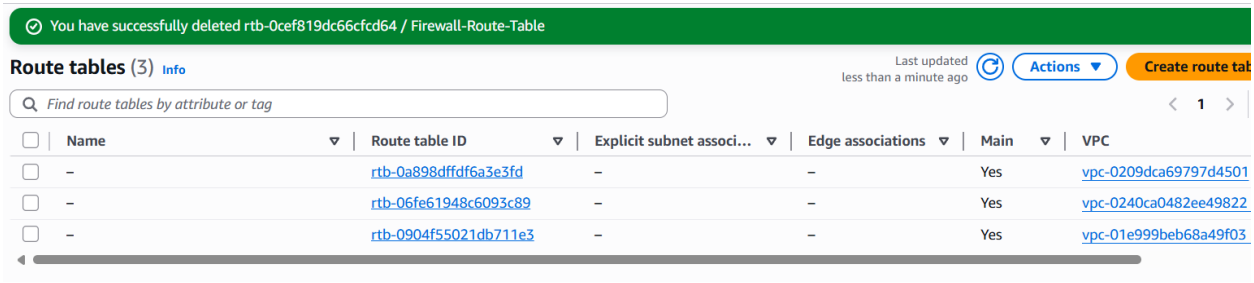
Here I verified outbound internet access and saw which services are listening on the instance, confirming both inbound and outbound traffic flow.



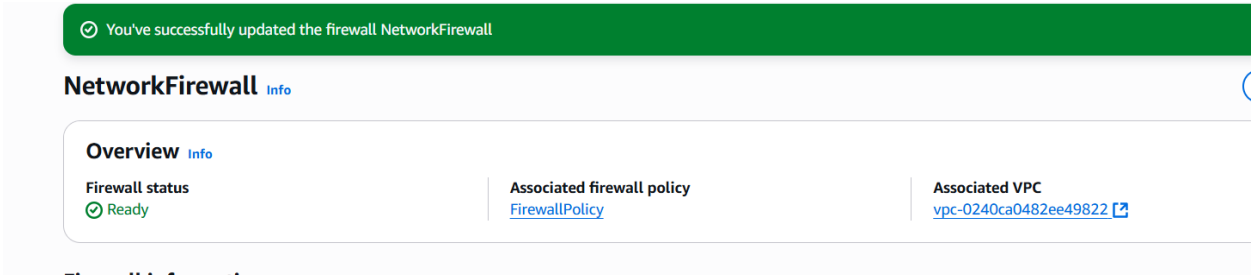
I confirmed that the deny rule for port 8080 works. Even though the service is running, the firewall blocks access as intended.



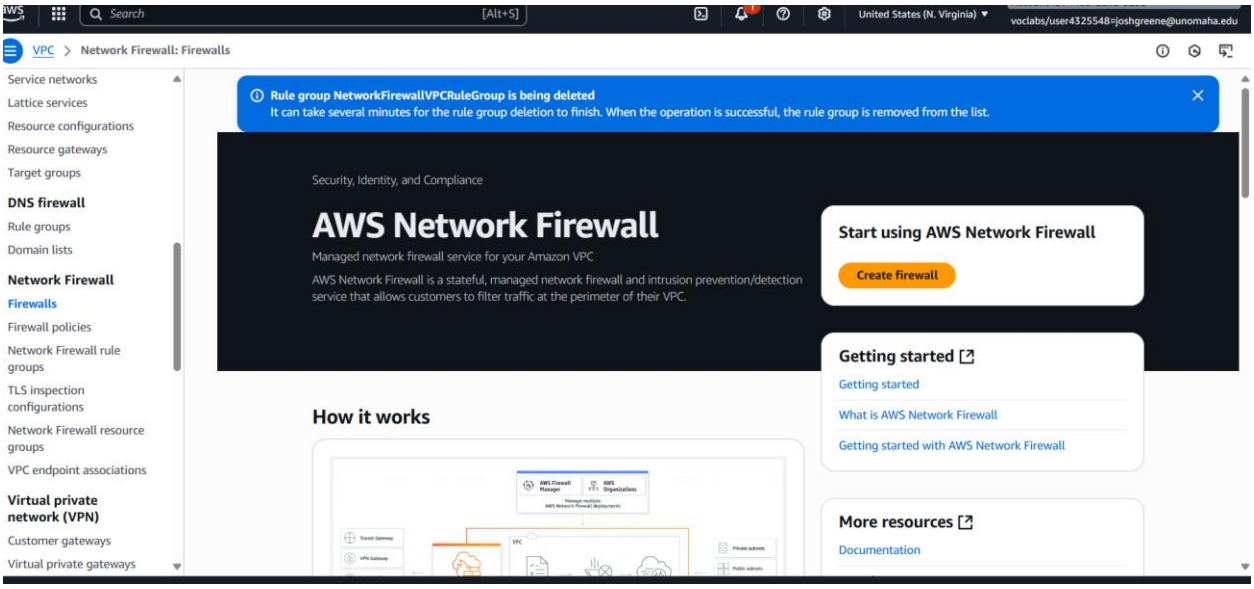
Here I verified that the firewall is logging traffic correctly. The log entries show accepted and denied connections, confirming monitoring is active.



Here I removed all subnet and edge associations from the three route tables, then deleted them to clean up my VPC routing setup.



Here I removed CloudWatch logging for the firewall to prevent new log data from being written and avoid extra charges.



Here I deleted my network firewall, its policy, and rule group to fully remove firewall resources from the VPC.

I deleted the CloudWatch log group to clean up stored logs and stop further billing for log retention.

Project Summary

In this project, I worked on making my VPCs more secure and testing how traffic moves through them. I started by checking my setup and creating flow logs to see what kind of traffic was being allowed or blocked in CloudWatch. Once I saw the rejected traffic logs, I updated my route tables and security groups so my web server could safely allow HTTP and SSH connections. I also added a network ACL to give the subnet extra protection.

Then I set up an AWS Network Firewall for another VPC, added route tables, turned on logging, and made a rule group that allowed ports 22, 80, 443, and ICMP but blocked 8080. When I tested it, my webpage and SSH worked fine while 8080 was blocked as expected. I checked CloudWatch to make sure the firewall was logging traffic correctly, and finally cleaned up everything to avoid charges. This phase helped me understand how AWS layers like security groups, NACLs, and firewalls work together to keep a network safe.