Report: Lambda-Based SDN Controller for Dynamic Traffic Policy Enforcement in AWS

1. Introduction

As modern cloud environments grow in scale and complexity, traditional static security controls often fall short in responding to dynamic threats. To address this challenge, this project explores the design and implementation of a Software-Defined Networking (SDN)-inspired controller using AWS Lambda. The goal is to create a lightweight, automated system capable of analyzing VPC Flow Logs and enforcing real-time network security policies by programmatically modifying security groups and network ACLs.

By leveraging serverless architecture and custom traffic policies written in Python, the solution aims to provide dynamic response capabilities—such as blocking unauthorized SSH attempts—without manual intervention. The system is tested against simulated attack scenarios to evaluate its responsiveness, reliability, and compliance readiness. This report outlines the design decisions, implementation steps, and key findings from the simulation, with a focus on scalability, automation, and security best practices in a cloud-native context.

2. Objective

This project aims to build a serverless SDN controller in AWS that dynamically enforces network security policies by analyzing VPC Flow Logs in real-time with AWS Lambda. Using a Python-based DSL, it allows customizable traffic rules to automatically detect and respond to suspicious activities—such as blocking unauthorized access—by programmatically updating security groups and network ACLs. The system also generates detailed logs for auditing and compliance, and it is tested through simulated attacks to ensure reliable policy enforcement and rapid incident response. By automating network traffic control, the solution reduces the need for manual intervention and enhances security agility in cloud environments. Ultimately, this approach demonstrates how event-driven cloud services can improve threat detection and mitigation while supporting scalable and flexible network management.

Key goals include:

- **Real-Time Traffic Analysis Using VPC Flow Logs**
  The system should continuously ingest and analyze VPC Flow Logs to monitor network activity across the AWS environment. This real-time analysis enables the identification of suspicious traffic patterns, such as repeated failed connection attempts, traffic from unknown IPs, or unexpected access to sensitive ports. The goal is to ensure timely detection of potential threats without relying on manual log review.

- **Automated Policy Enforcement with AWS Lambda**
  A core objective is to automate the enforcement of network policies using AWS Lambda. Based on predefined rules, the Lambda function should trigger actions—such as blocking a source IP or adjusting port access—without human intervention. This allows for faster incident response and supports scalability by removing the need for constant manual oversight.

- **Customizable Policy Logic via Python DSL**
  To provide flexibility and control, the project includes a Python-based domain-specific language (DSL) that allows administrators to define custom traffic rules. These rules dictate how the system reacts to specific network conditions, making it easier to tailor security behavior to different use cases or compliance requirements.

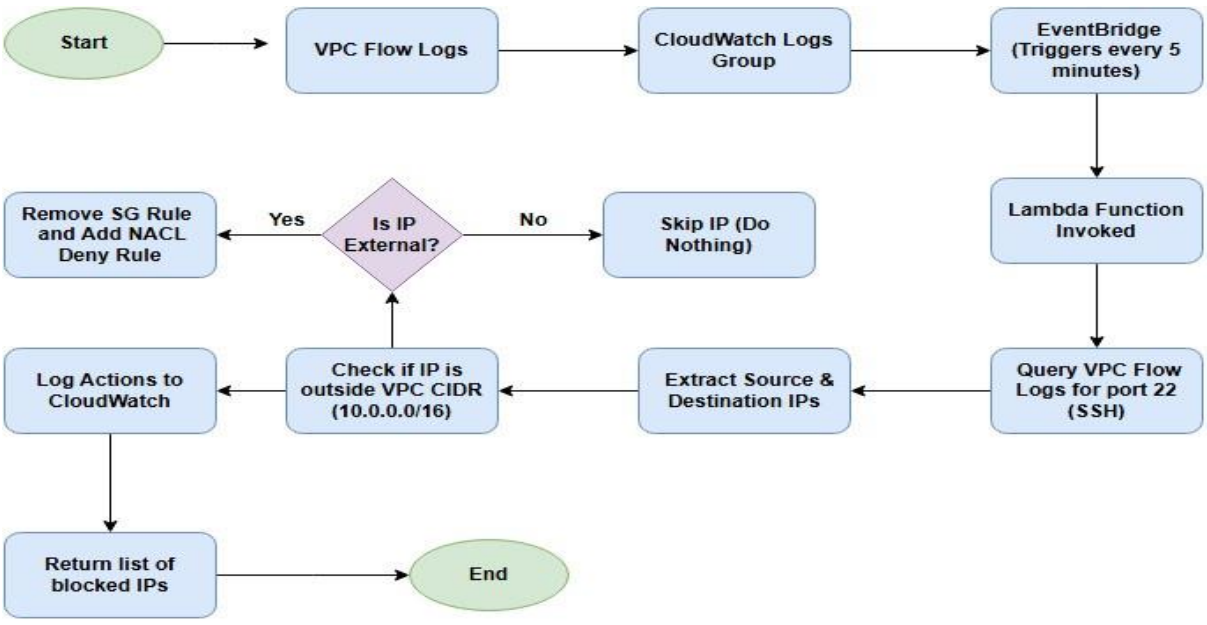
- **Dynamic Modification of Security Groups and NACLs**
  The solution must be capable of modifying AWS security groups and network ACLs programmatically based on analysis results. This ensures that unauthorized or malicious traffic can be swiftly blocked, and legitimate traffic can be allowed without delay. Dynamic reconfiguration helps maintain strong security postures even in rapidly changing environments.

- **Comprehensive Logging and Audit Reporting**
  To support compliance and post-incident analysis, the system should generate detailed logs of all actions taken, including traffic evaluations, rule matches, and security group changes. These logs will be used to produce audit-ready reports that provide insight into system behavior, policy enforcement outcomes, and overall security effectiveness.

3. Tools And Techniques

- **AWS Lambda** – Serverless compute service used to process VPC Flow Logs and execute traffic policy logic in real-time.
- **VPC Flow Logs** – Network traffic logs from AWS VPCs used for monitoring and analysis.
- **Python DSL** – Custom domain-specific language implemented in Python for defining dynamic traffic control policies.
- **AWS Security Groups & NACLs** – Network access control mechanisms dynamically modified via code to enforce security policies.
- **AWS SDK (boto3)** – Python library used to programmatically interact with AWS services for policy enforcement.
- **SSH Login Attempts** – Used to simulate unauthorized access attempts for validating policy enforcement.
- **AWS CloudWatch** – Service used for logging, auditing, and monitoring system activities.

4. Workflow Diagram



5. Methodology

This project focuses on building a dynamic Software-Defined Network (SDN) controller using AWS Lambda to monitor and automatically adjust network security policies in real-time. The objective is to create a secure cloud environment that detects unauthorized SSH login attempts via VPC Flow Logs, processes these logs with Lambda functions, and dynamically updates security groups and Network ACLs (NACLs) to block malicious IP addresses. The methodology is divided into distinct phases encompassing network setup, monitoring, automation, attack simulation, and validation.

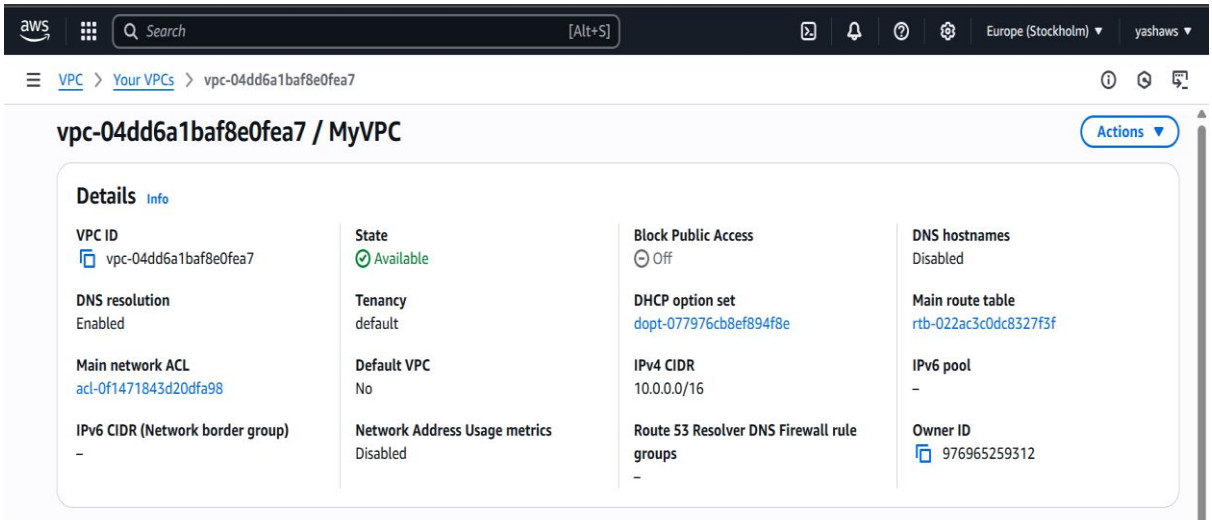
5.1 Network Setup and Resource Configuration

This phase establishes the cloud network infrastructure necessary for hosting resources and capturing traffic data. It ensures a well-architected environment where monitoring and policy enforcement can operate reliably.

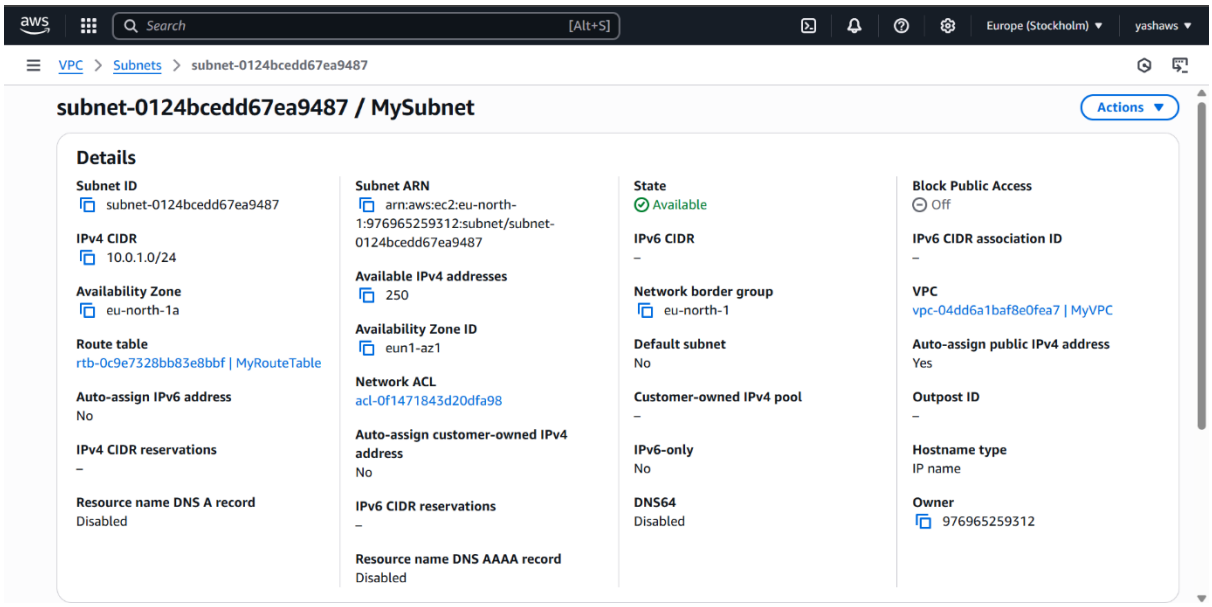
Purpose: Establish a foundational and fully functional AWS networking environment to simulate real-world scenarios and enable monitoring.

- Created a Virtual Private Cloud (VPC), subnet, Internet Gateway (IGW), route tables, and attached them appropriately to establish a functional networking infrastructure.

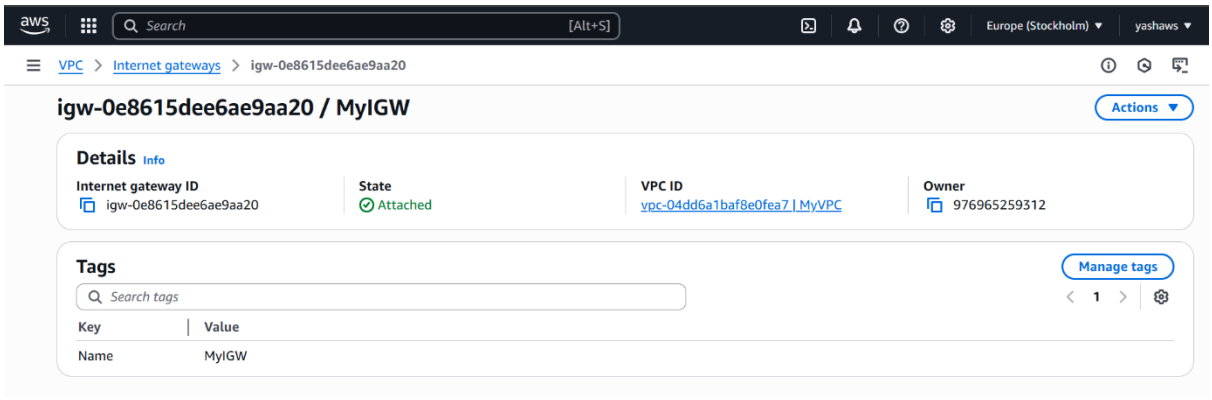
(A VPC named "MyVPC" was created.)



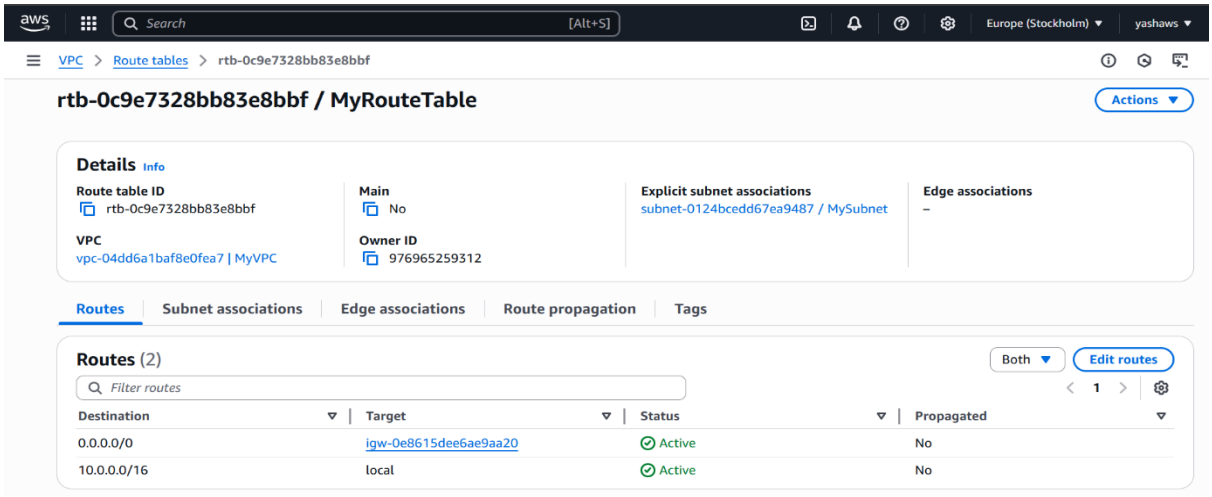
(Created a subnet and added the VPC to it.)



(Created an Internet Gateway and attached it to the VPC.

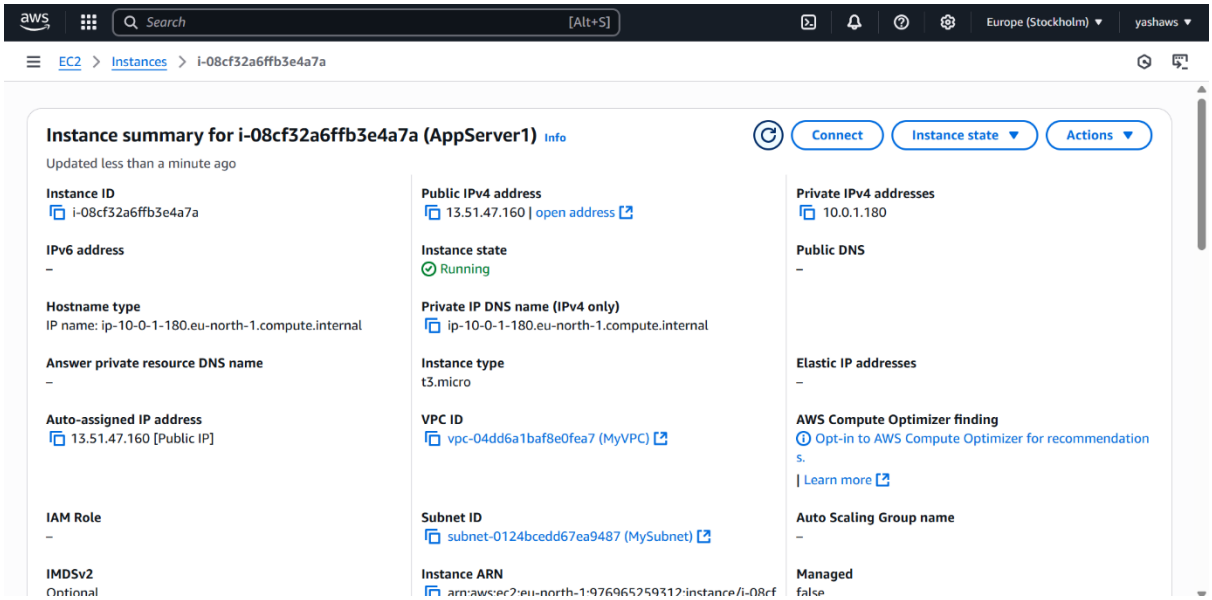


(Created a route table, associated the subnet to it, and selected the Internet Gateway by editing the routes.)

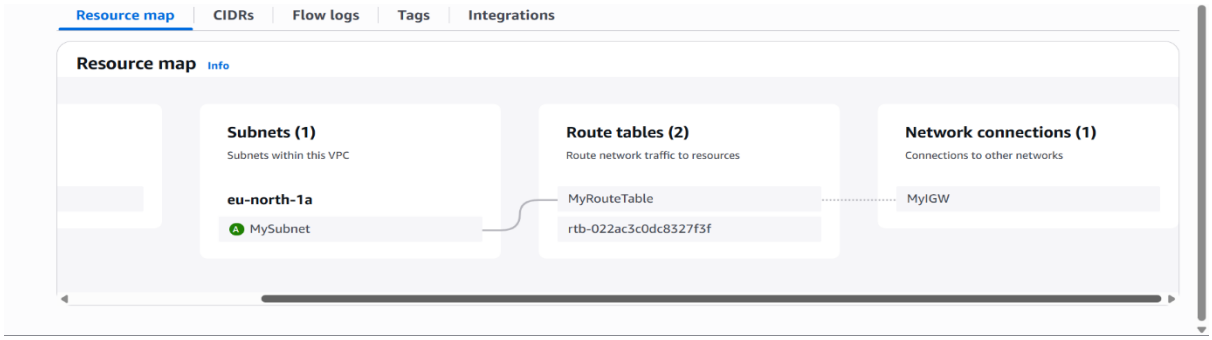


- Launched an EC2 instance named **AppServer** within the VPC **MyVPC** to serve as the target for simulated SSH attacks.

  (An instance named AppServer was created, using the VPC and subnet that we created.



- Generated a resource map of the VPC to visualize the architecture and resource relationships.



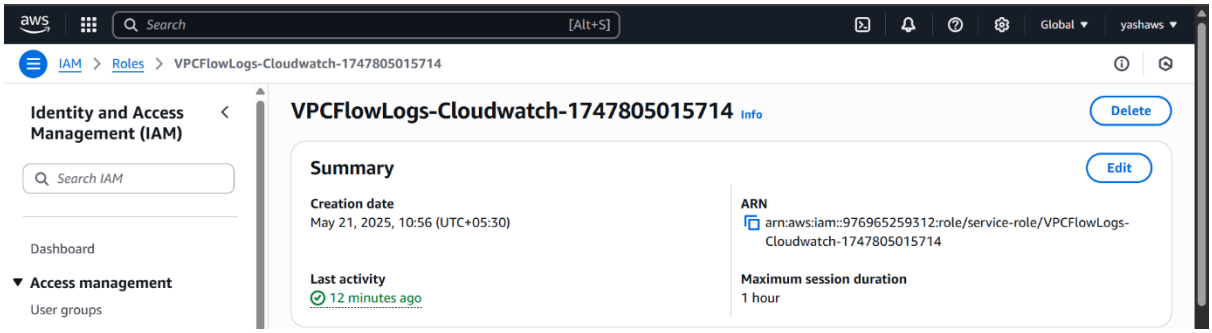
5.2 Logging and IAM Role Setup

Effective monitoring requires detailed traffic logs and appropriate permissions. This phase focuses on enabling VPC Flow Logs to capture network traffic metadata and creating the required IAM roles that allow AWS services to interact securely.

Purpose: Enable detailed network traffic visibility and assign appropriate permissions for automation and monitoring.

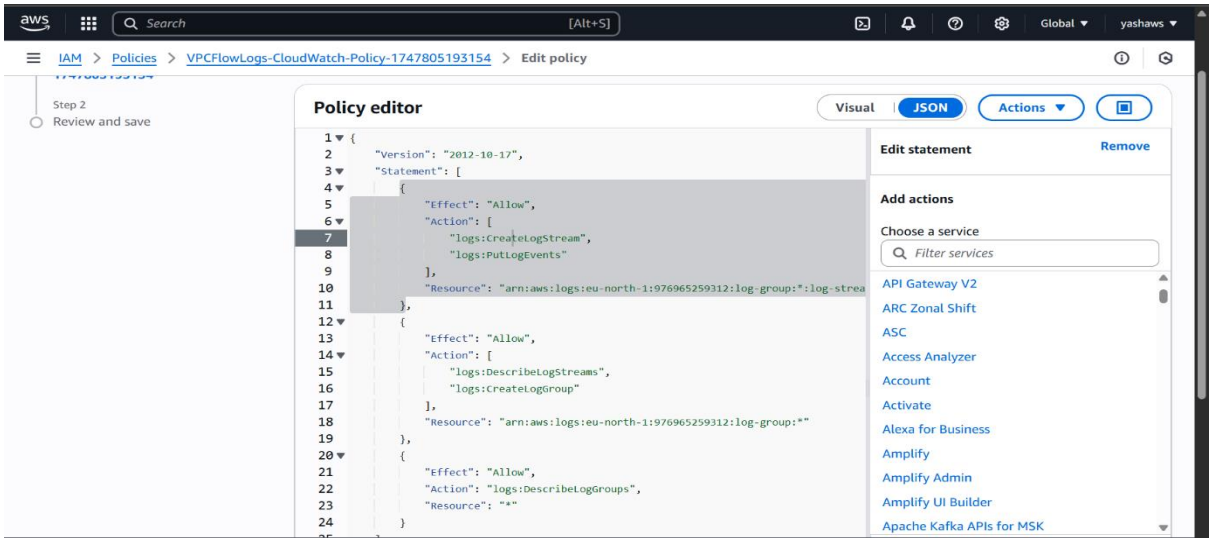
- Created a VPC Flow Log named **MyFlow** to collect network traffic metadata and an IAM role with permissions for publishing these logs.

(I created the IAM role for the Flow Log.)



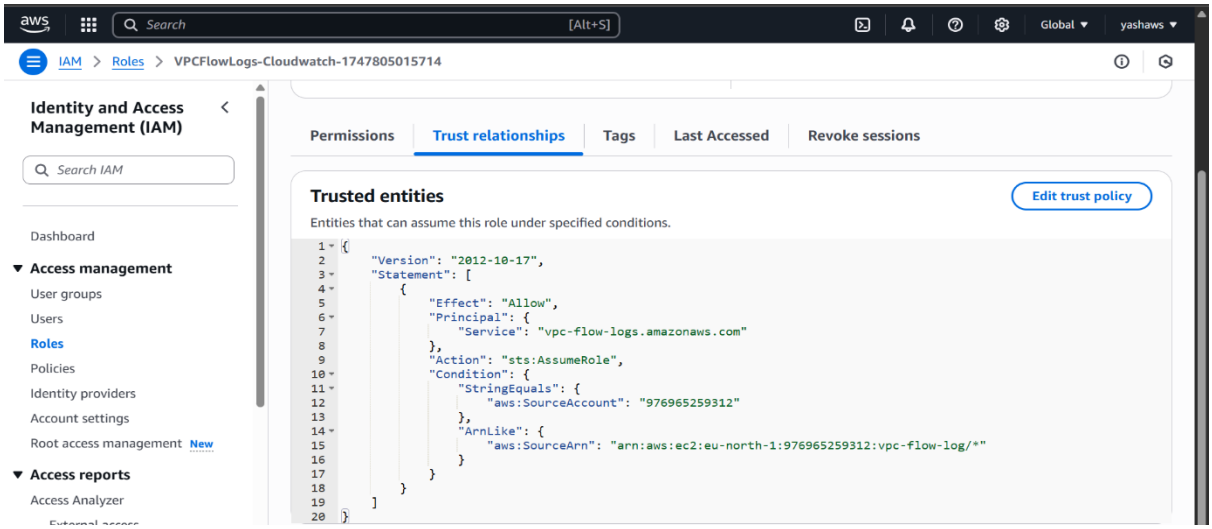
This IAM policy configuration enables the ability to create log streams and publish log events within any log stream associated with any log group under the AWS account 976965259312 in the eu-north-1 region. Additionally, it grants permissions to describe existing log streams and create new log groups within the same account and region, supporting log management and monitoring tasks. Furthermore, it allows listing and describing all log groups globally, across all regions, which is useful for gaining visibility into log group configurations throughout the account.

(Permission Policy)

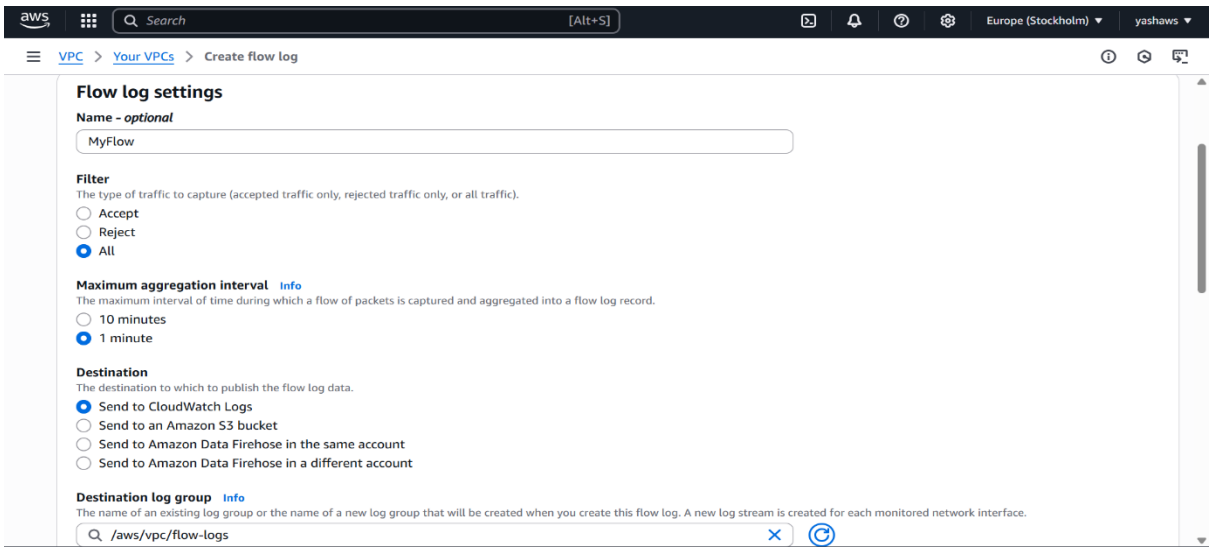


This IAM trust policy allows the AWS VPC Flow Logs service (vpc-flow-logs.amazonaws.com) to assume the role, but only if the request comes from your AWS account (976965259312) and specifically from VPC Flow Logs resources within the eu-north-1 region of that account. This ensures that only legitimate VPC Flow Logs within your account can use this role, adding a layer of security by restricting access based on account and resource ARN.

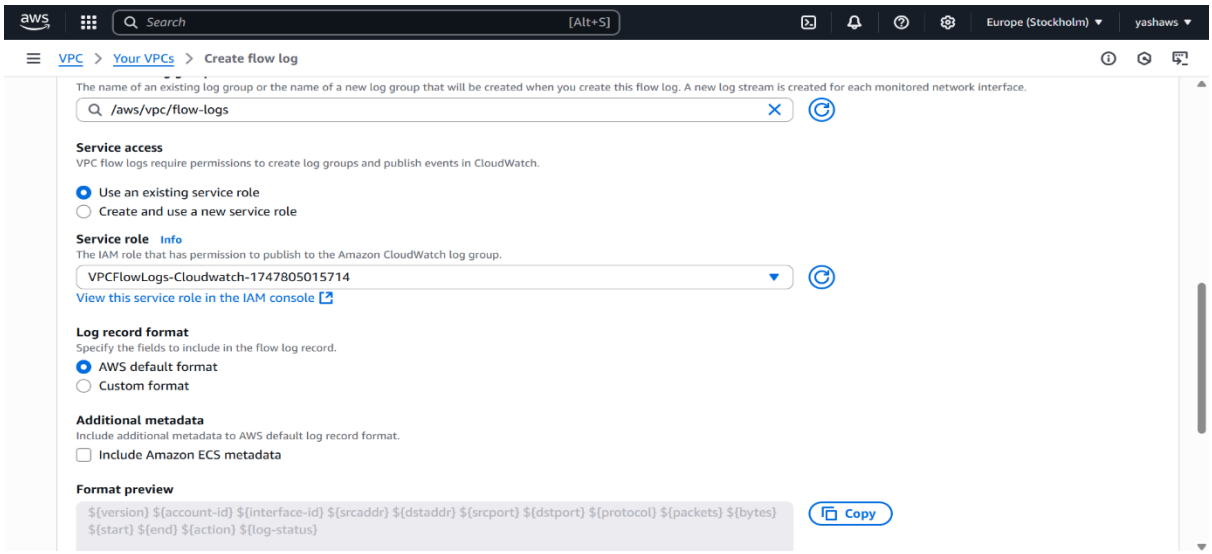
(Trust Relationship.)



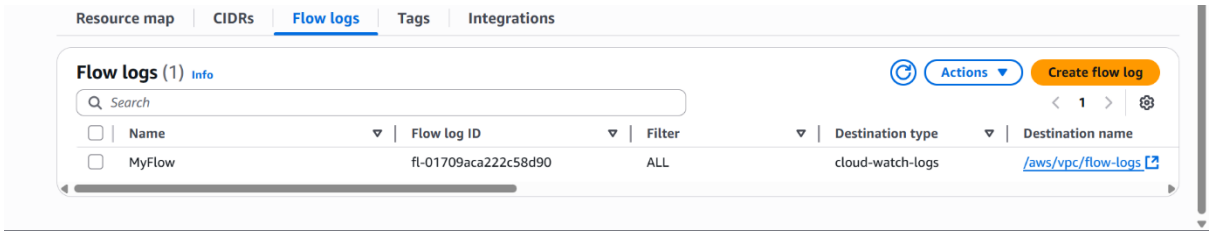
(Created a flow log and set the destination to send data to CloudWatch Logs.)



(Added the previously created service role.

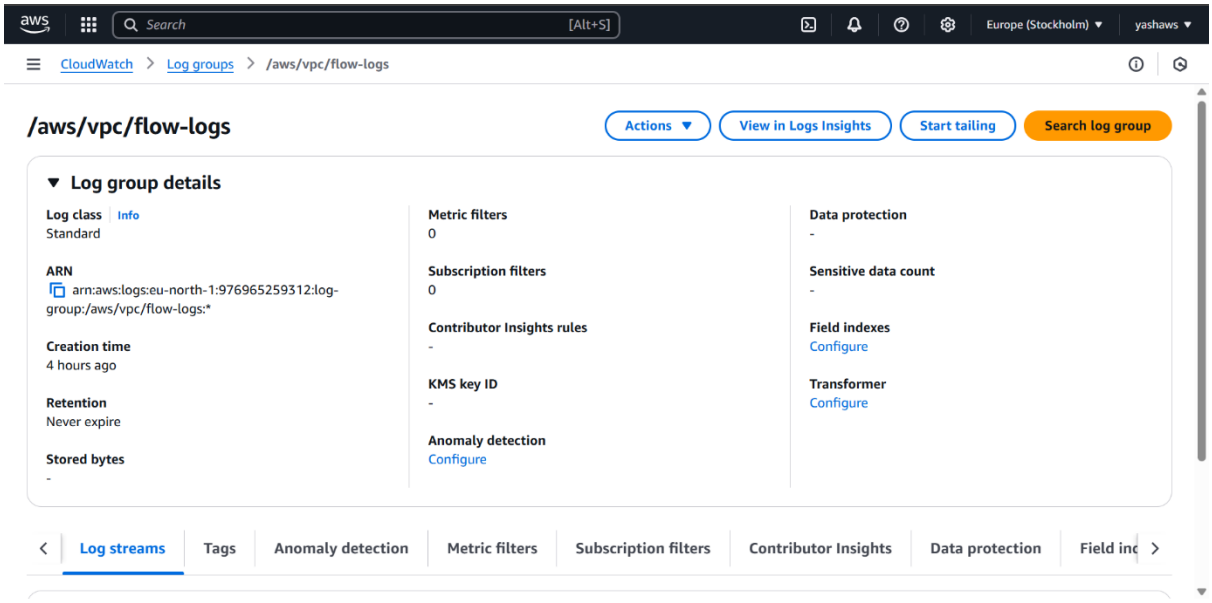


(The flow log was created successfully.)

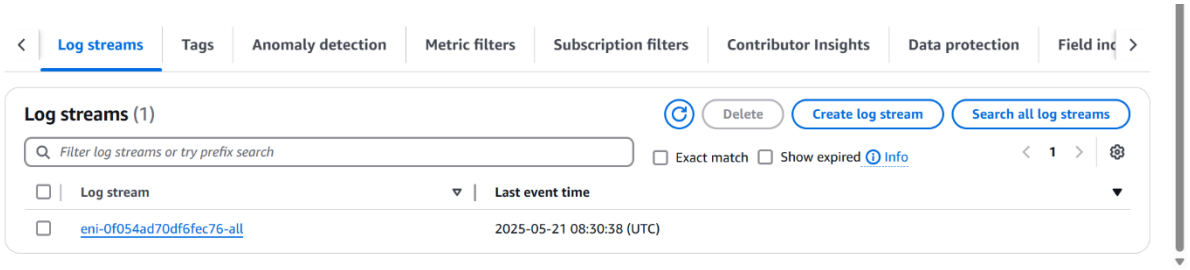


- Verified successful delivery of flow logs to the designated CloudWatch Logs group by inspecting the log stream for incoming entries.

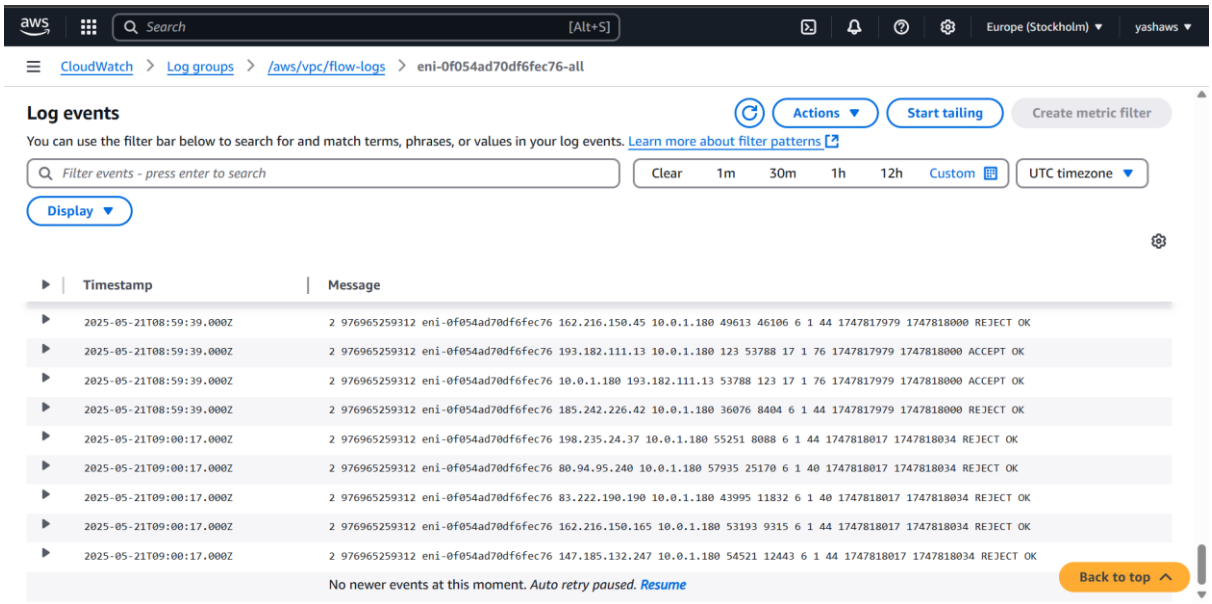
(The log group is created.)



(Creation of the log stream was successful.)

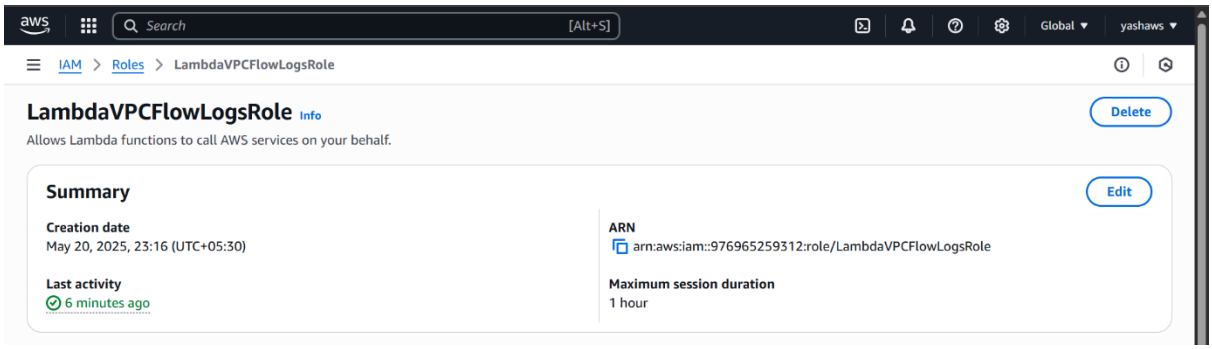


(Verified logs within the log group successfully.)

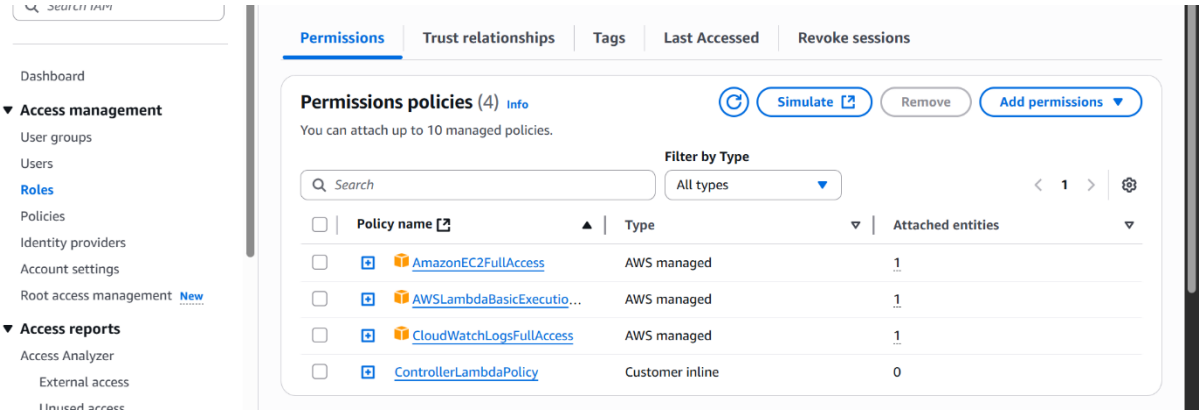


- Created a separate IAM role with policies granting Lambda function necessary permissions to access flow logs and modify network resources.

(The IAM role for the Lambda function was created.)

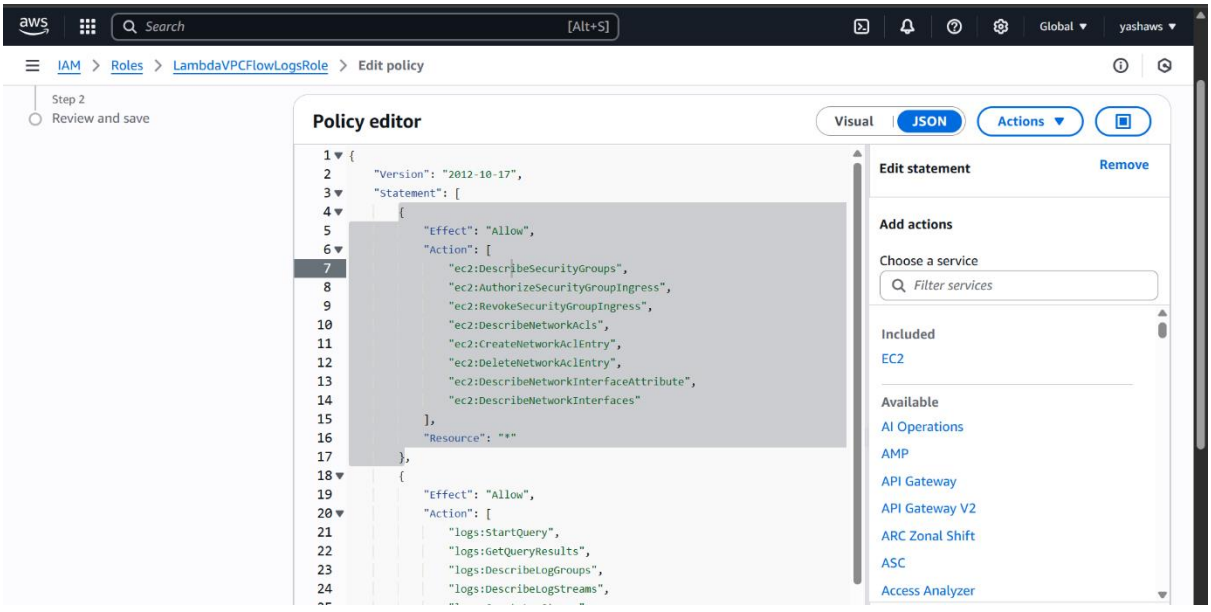


(Permissions granted to the role.)



This policy grants broad permissions to manage and view EC2 security components, including describing security groups and network ACLs, as well as authorizing or revoking security group ingress rules and managing network ACL entries and network interfaces. Additionally, it allows starting and retrieving CloudWatch Logs queries, describing log groups and streams, and creating log streams and putting log events, including full access to all logs and specifically to the Lambda function log group /aws/lambda/ControllerLambda.

(Policy.)



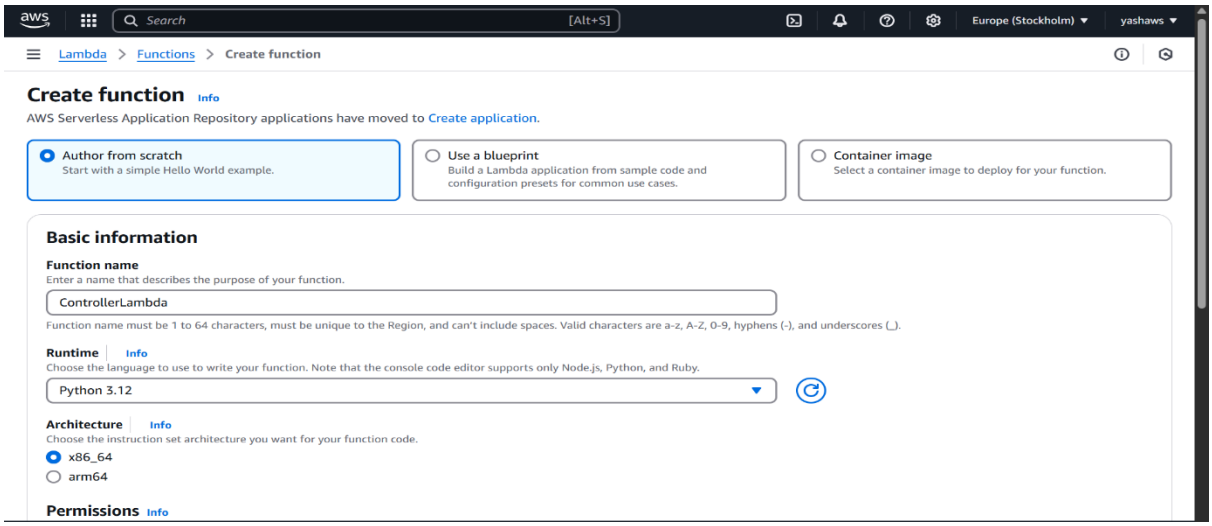
5.3 Lambda Function and Scheduling

At the heart of automation lies the Lambda function, which processes logs and applies security rules. This phase develops the function and sets up scheduled execution, ensuring continuous vigilance over network traffic.

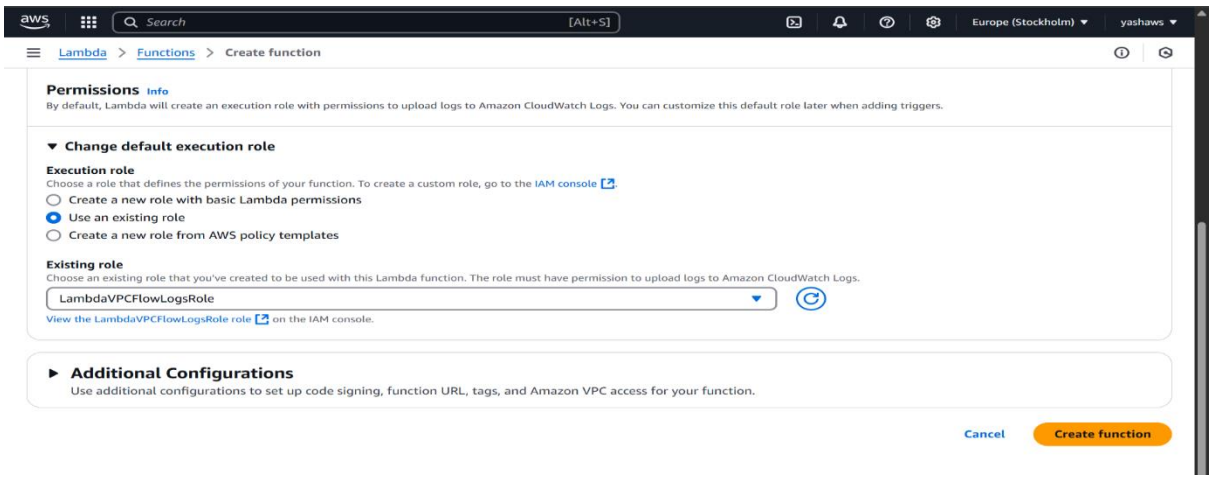
Purpose: Automate the analysis of network logs and the enforcement of dynamic security policies in near real-time.

- Developed the Lambda function named **ControllerLambda** that analyzes incoming VPC Flow Logs to detect unauthorized SSH attempts and programmatically modifies security groups and NACLs.

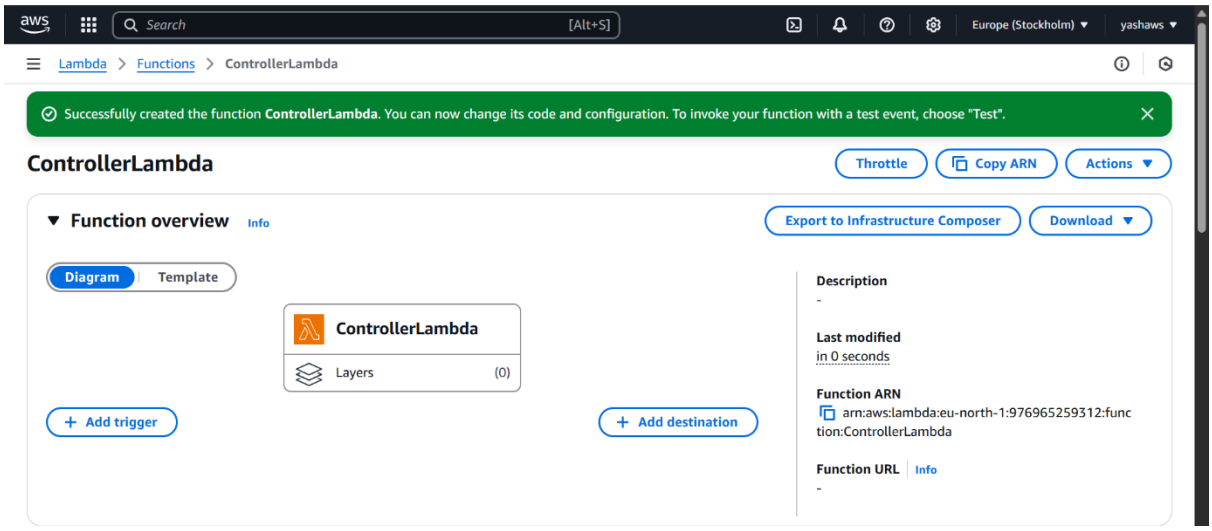
(A Lambda function was created.)



(Utilized the existing role created earlier.)



(Lambda function created successfully.)



The Lambda function code is provided below.

```
import boto3
import time
import ipaddress
import logging

ec2 = boto3.client('ec2')
logs = boto3.client('logs')

LOG_GROUP = '/aws/vpc/flow-logs'
SECURITY_GROUP_ID = 'sg-025b0819a4b5799ab'
NACL_ID = 'acl-0f1471843d20dfa98'
VPC_CIDR = '10.0.0.0/16'

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    logger.info("Lambda execution started")
```

```
end_time = int(time.time())
start_time = end_time - 5 * 60

query_string = """
fields @timestamp, srcAddr, dstAddr, dstPort, action
| filter dstPort = 22 and action = "ACCEPT"
| sort @timestamp desc
| limit 50
"""

logger.info(f"Starting Logs Insights query: {query_string.strip()}")
start_query_response = logs.start_query(
    logGroupName=LOG_GROUP,
    startTime=start_time,
    endTime=end_time,
    queryString=query_string,
    limit=50
)
query_id = start_query_response['queryId']
logger.info(f"Query started with ID: {query_id}")

timeout = 30
elapsed = 0
while True:
    response = logs.get_query_results(queryId=query_id)
    status = response['status']
    logger.info(f"Query status: {status}")
    if status in ['Complete', 'Failed', 'Cancelled']:
        break
    time.sleep(1)
    elapsed += 1
    if elapsed >= timeout:
        logger.error("Logs query timed out")
        return {'statusCode': 500, 'body': 'Query timeout'}
```

```
if status != 'Complete':
    logger.error(f"Logs query failed or cancelled: {status}")
    return {'statusCode': 500, 'body': 'Query not completed'}

blocked_ips = []
logger.info(f"Query returned {len(response['results'])} results")

for result in response['results']:
    src_ip = None
    dst_ip = None
    for field in result:
        if field['field'] == 'srcAddr':
            src_ip = field['value']
        elif field['field'] == 'dstAddr':
            dst_ip = field['value']

    logger.info(f"Found src IP: {src_ip}, dst IP: {dst_ip}")

    for ip in [src_ip, dst_ip]:
        if ip:
            if is_unknown_ip(ip):
                logger.info(f"IP {ip} is outside VPC CIDR, attempting to block")
                removed = remove_sg_rule(ip)
                added = add_nacl_deny(ip)
                if removed or added:
                    blocked_ips.append(ip)
            else:
                logger.info(f"IP {ip} is inside VPC CIDR, skipping")
        else:
            logger.warning("IP field missing")

logger.info(f"Blocked IPs in this run: {blocked_ips}")
```

```
    return {
        'statusCode': 200,
        'body': f'Blocked IPs: {blocked_ips}'
    }


def is_unknown_ip(ip):
    try:
        vpc_network = ipaddress.ip_network(VPC_CIDR)
        ip_obj = ipaddress.ip_address(ip)
        result = ip_obj not in vpc_network
        logger.info(f"is_unknown_ip({ip}) -> {result}")
        return result
    except ValueError:
        logger.error(f"Invalid IP address: {ip}")
        return False


def remove_sg_rule(ip):
    try:
        sg = ec2.describe_security_groups(GroupIds=[SECURITY_GROUP_ID])['SecurityGroups'][0]
        perms = sg.get('IpPermissions', [])
        to_revoke = []

        for perm in perms:
            if perm.get('FromPort') == 22 and perm.get('ToPort') == 22 and perm.get('IpProtocol') in ['tcp', '-1']:
                for ip_range in perm.get('IpRanges', []):
                    if ip_range.get('CidrIp') == f"{ip}/32":
                        to_revoke.append({
                            'IpProtocol': perm['IpProtocol'],
                            'FromPort': perm['FromPort'],
                            'ToPort': perm['ToPort'],
```

```
                    'IpRanges': [{'CidrIp': f"{ip}/32"}]
                })
        if to_revoke:
            ec2.revoke_security_group_ingress(GroupId=SECURITY_GROUP_ID,
IpPermissions=to_revoke)
            logger.info(f"Removed SG rule for IP {ip}")
            return True
        else:
            logger.info(f"No SG rule found to remove for IP {ip}")
        return False
    except Exception as e:
        logger.error(f"Failed to remove SG rule for {ip}: {e}")
        return False


def find_free_rule_number(nacl, starting_point=100):
    used = {entry['RuleNumber'] for entry in nacl['Entries']}
    for num in range(starting_point, 0, -1):
        if num not in used:
            return num
    for num in range(starting_point + 1, 32766):
        if num not in used:
            return num
    return None


def add_nacl_deny(ip):
    try:
        nacl = ec2.describe_network_acls(NetworkAclIds=[NACL_ID])['NetworkAcls'][0]

        for entry in nacl['Entries']:
            if (entry['RuleAction'] == 'deny' and
                entry.get('CidrBlock') == f"{ip}/32" and
                entry.get('Protocol') == '6' and
```

```
            entry.get('PortRange', {}).get('From') == 22 and
            not entry.get('Egress', True)):
            logger.info(f"NACL deny rule already exists for IP {ip}")
            return False

    allow_rule_numbers = [entry['RuleNumber'] for entry in nacl['Entries']
                          if entry['RuleAction'] == 'allow' and not entry.get('Egress', True)]

    if allow_rule_numbers:
        min_allow = min(allow_rule_numbers)
        rule_number = find_free_rule_number(nacl, starting_point=min_allow - 1)
    else:
        rule_number = find_free_rule_number(nacl, starting_point=100)

    if rule_number is None:
        logger.error("No available rule numbers to create deny rule")
        return False

    ec2.create_network_acl_entry(
        NetworkAclId=NACL_ID,
        RuleNumber=rule_number,
        Protocol='6',
        RuleAction='deny',
        Egress=False,
        CidrBlock=f'{ip}/32',
        PortRange={'From': 22, 'To': 22}
    )
    logger.info(f"Added NACL deny rule for IP {ip} with rule number {rule_number}")
    return True
except Exception as e:
    logger.error(f"Failed to add NACL deny rule for {ip}: {e}")
    return False
```

The Lambda function is the core component of the dynamic SDN controller. It automates the process of detecting unauthorized SSH login attempts and enforces network policies by modifying AWS Security Groups and Network ACLs based on real-time log analysis.

Function Overview

The Lambda function performs the following tasks:

1. Queries recent VPC Flow Logs for accepted SSH (port 22) connections using CloudWatch Logs Insights.
2. Extracts source and destination IPs from these logs.
3. Evaluates whether the IPs are external (i.e., not part of the internal VPC CIDR).
4. If an IP is suspicious:
   - It removes any existing Security Group rules allowing SSH from that IP.
   - Adds a new deny rule for the IP in the Network ACL.
5. Logs every step and decision made for auditing and debugging.

Core Code Logic – IP Evaluation Snippet

```
def is_unknown_ip(ip):
    vpc_network = ipaddress.ip_network(VPC_CIDR)
    ip_obj = ipaddress.ip_address(ip)
    return ip_obj not in vpc_network
```

This helper function checks whether a given IP address is **outside the private VPC range**, treating it as suspicious if so. Only these IPs are subject to blocking.

Triggering Mechanism

The Lambda function is triggered every **5 minutes** using **Amazon EventBridge (formerly CloudWatch Events)**. This ensures continuous monitoring and rapid response to threats.

Security Operations Performed

Security Group Cleanup: The function scans for rules allowing SSH from suspicious IPs and revokes them using:

```
ec2.revoke_security_group_ingress()
```

NACL Rule Insertion:

A deny rule is added for port 22 using:

ec2.create_network_acl_entry()

Rule numbers are dynamically chosen to avoid conflict with existing entries.

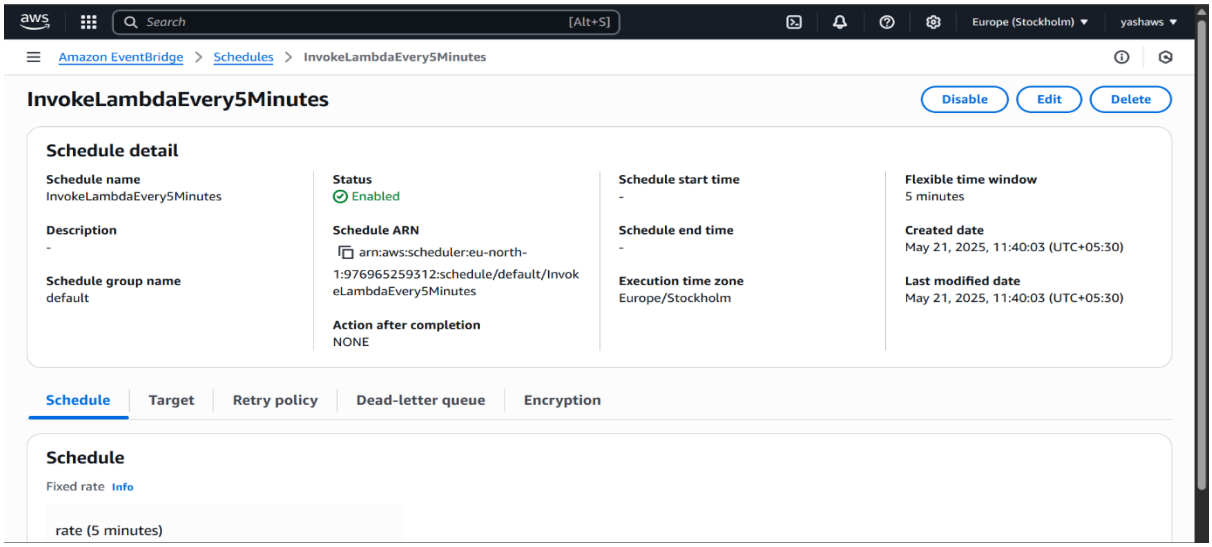
Logging and Observability

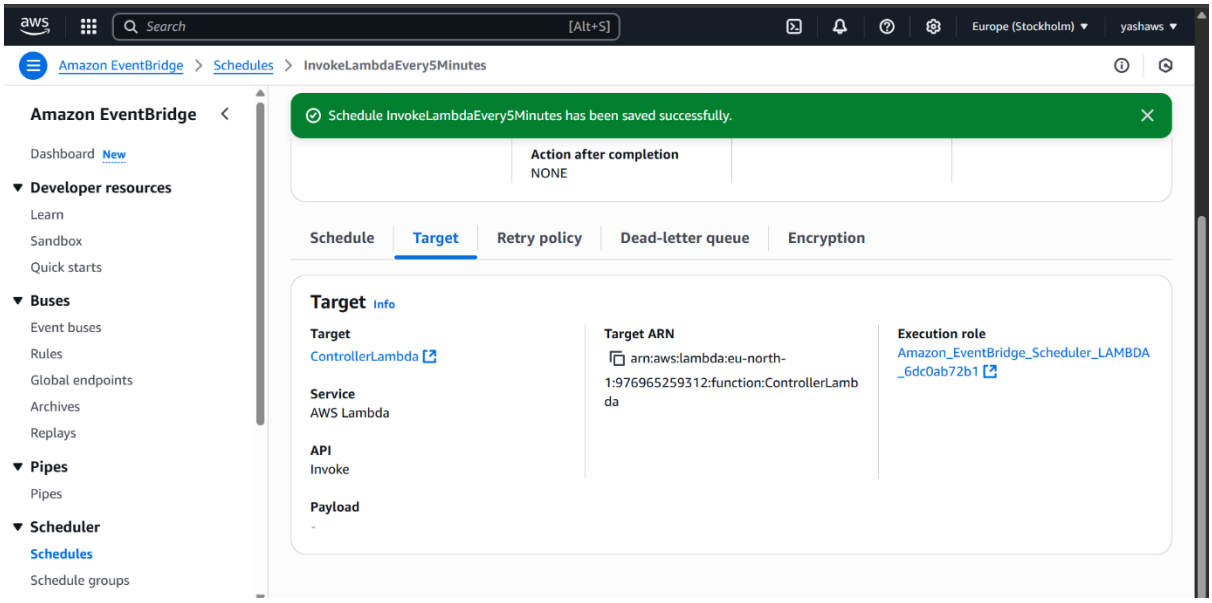
The function logs all events via AWS CloudWatch, including:

- Queries executed
- IPs processed
- Rules added or removed
- Any errors or timeouts

- Configured Amazon EventBridge to trigger the Lambda function every 5 minutes, ensuring near real-time processing and response.

(Scheduled the Lambda function to run at 5-minute intervals.)





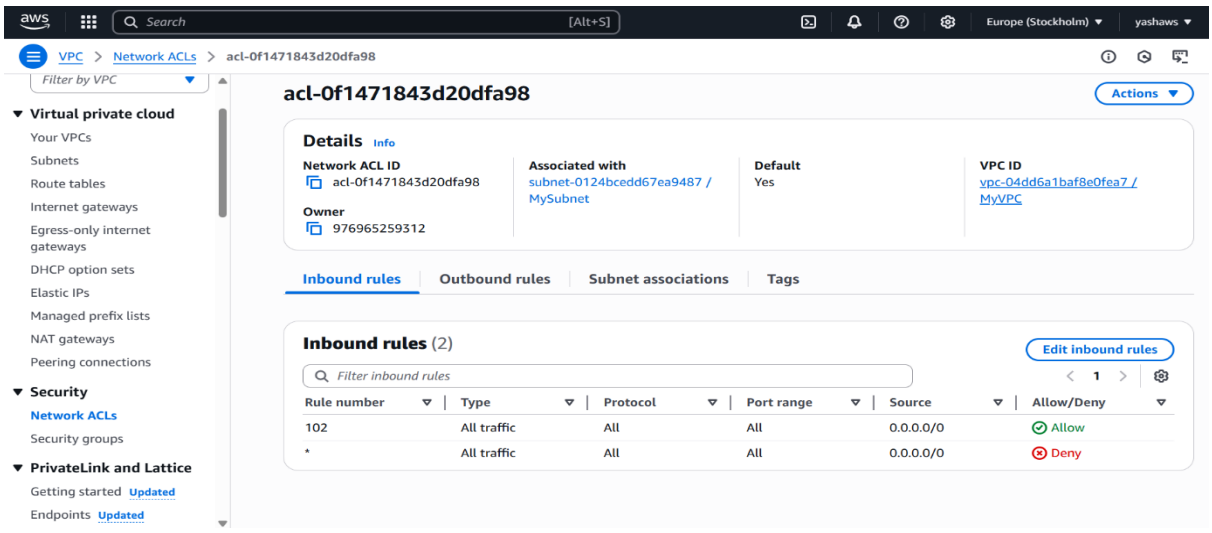
5.4 Security Policy Verification and Attack Simulation

To validate the system's responsiveness, this phase involves testing the baseline security posture and simulating attack scenarios. It confirms the environment's readiness and the controller's detection capability.

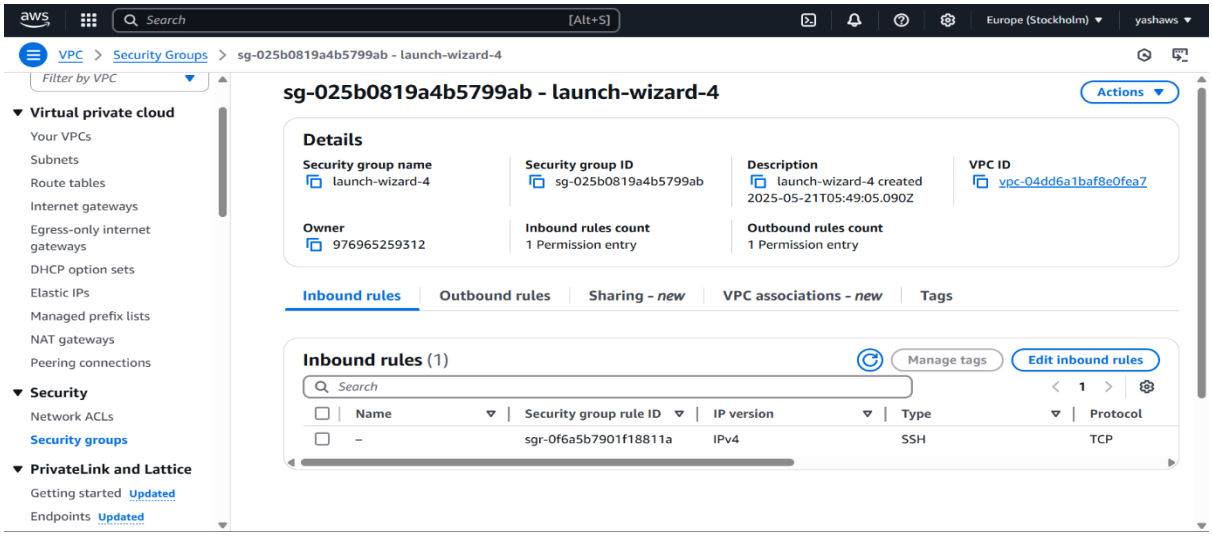
Purpose: Test the baseline security posture and validate the detection and response capabilities of the system.

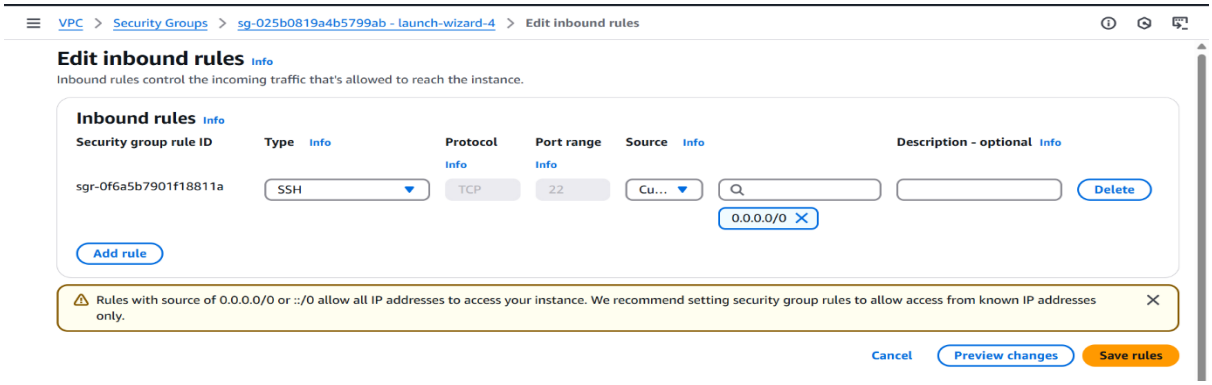
- Inspected existing NACL and security group rules, confirming they initially allowed broad access (0.0.0.0/0) to port 22 (SSH).

(All inbound traffic is allowed.)



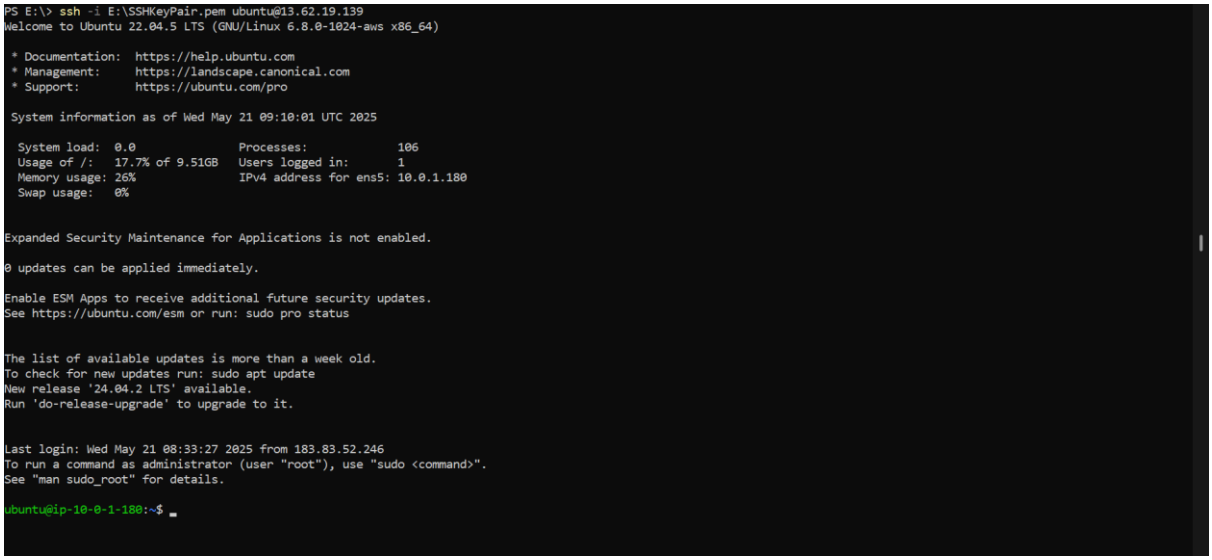
(The security group allows all incoming TCP traffic.)





- Simulated an attack by attempting SSH logins from unauthorized IP addresses to trigger the Lambda function's detection logic.

  (Use SSH on the attacker machine to connect to our instance.)



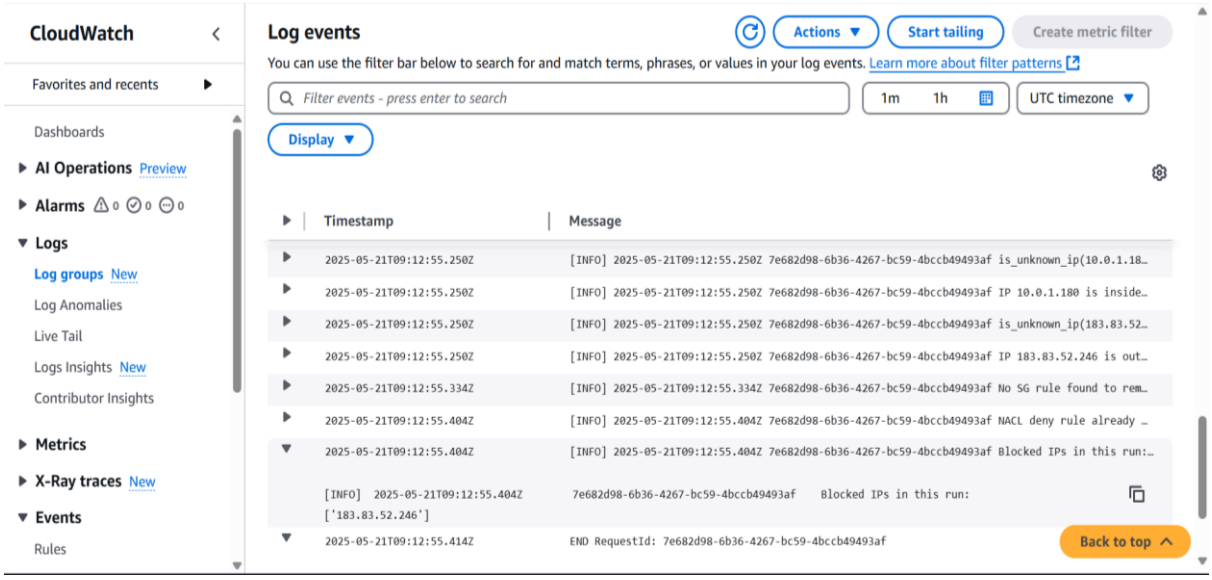
5.5 Monitoring, Detection, and Enforcement Validation

This final phase verifies the end-to-end functionality of the SDN controller. It includes analyzing logs for threat detection, confirming the application of blocking rules, and ensuring that unauthorized access attempts are effectively thwarted.

- Monitored Lambda function logs in **/aws/lambda/ControllerLambda** for entries indicating detection of suspicious SSH login attempts.

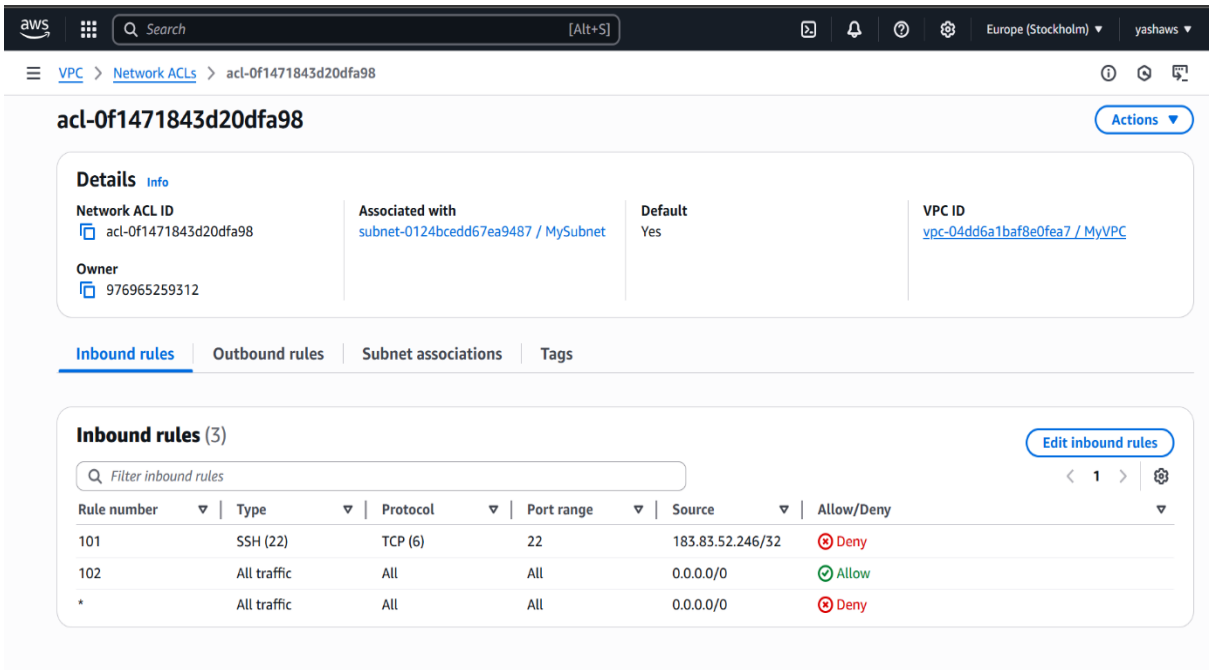
  Purpose: Confirm that the SDN controller correctly identifies threats and enforces security policies dynamically.

  (Network traffic was successfully captured.)



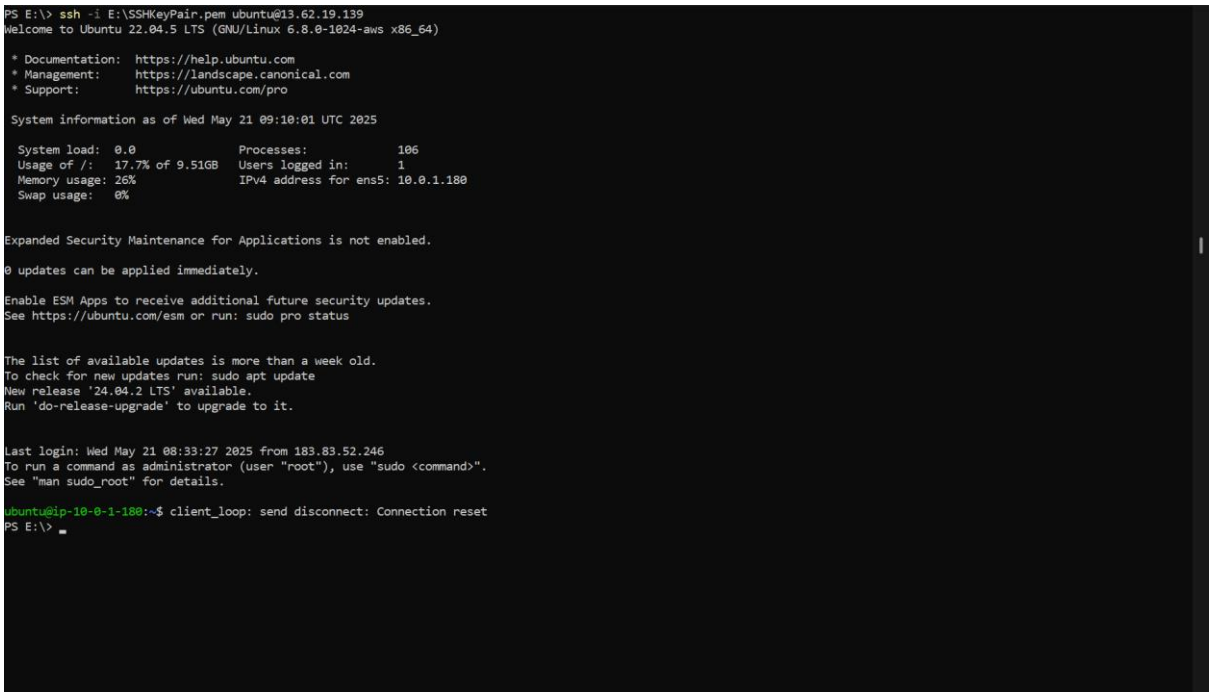
- Verified that the Lambda function successfully updated NACLs by blocking the malicious IP addresses.

(The suspicious IP address has been blocked in the NACL rules.)

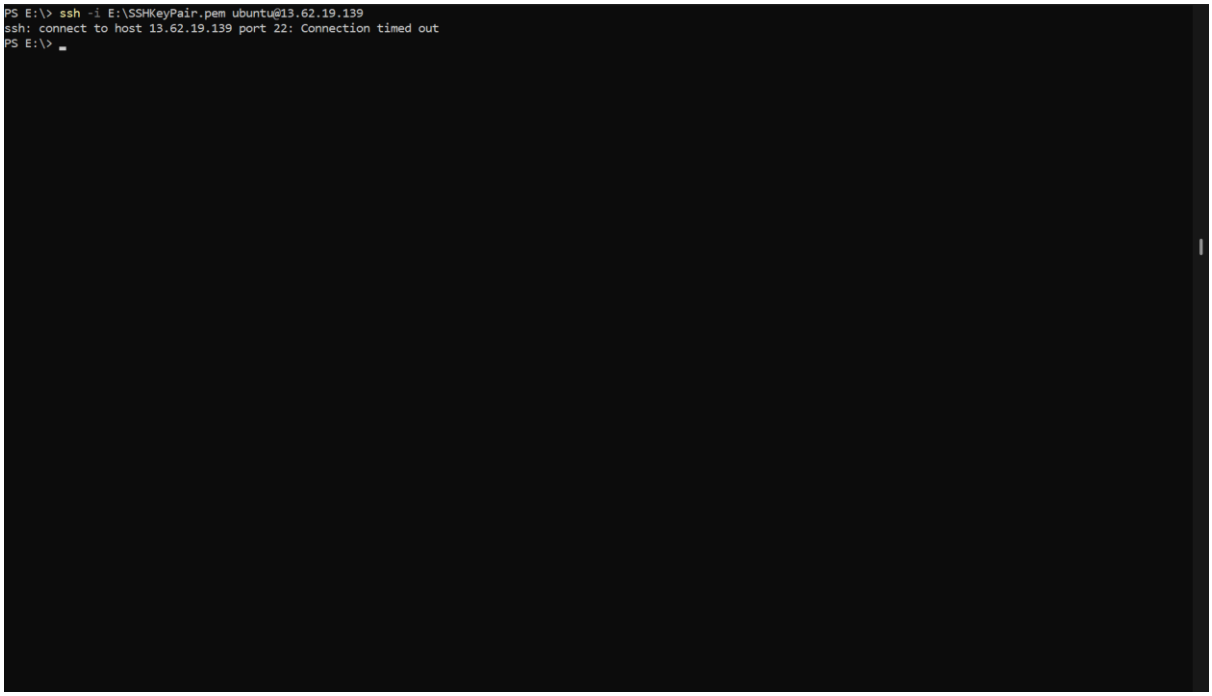


- Observed connection reset behavior on SSH login attempts from blocked IPs, confirming enforcement effectiveness.

(On the attacking machine, the connection has been reset.)

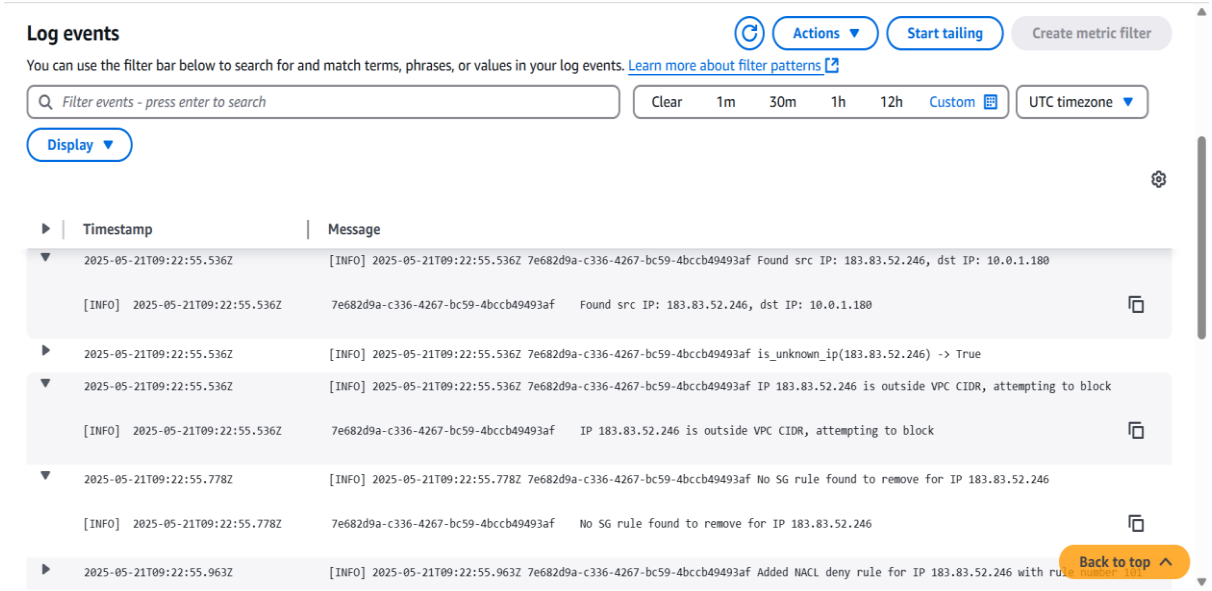


(Tried to connect again, but the connection was denied.)

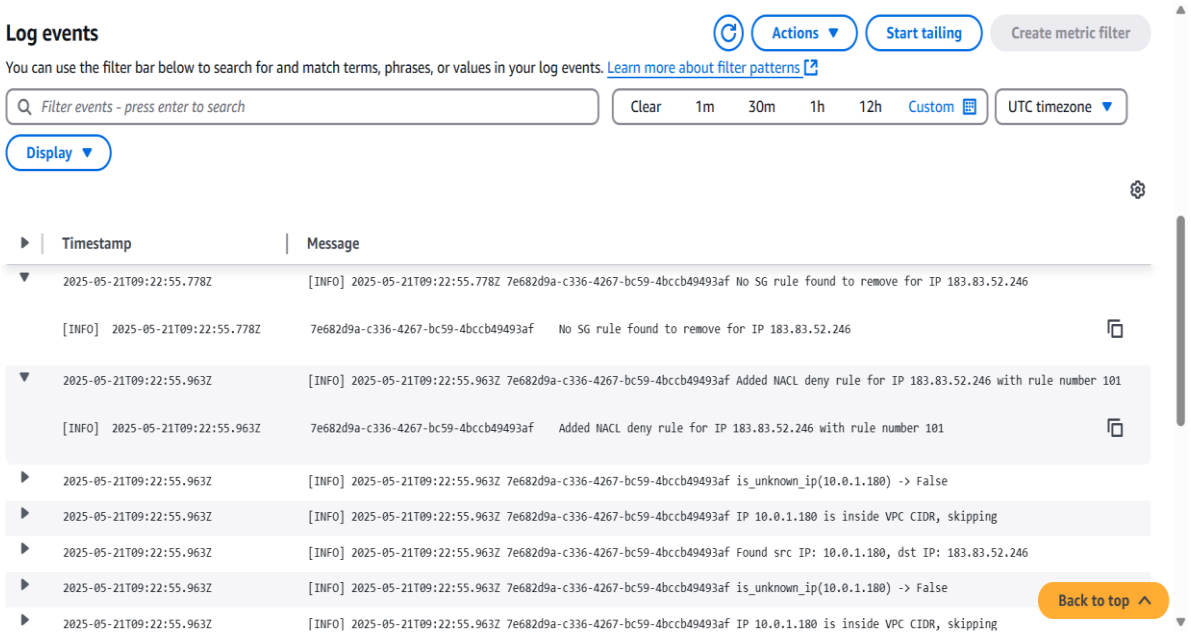


- Reviewed detailed audit logs to ensure comprehensive recording of detection and mitigation actions.

(The logs show that the IP is blocked.)



(Detailed log entries)



6. Results And Findings

This section presents the observed outcomes of the implemented SDN-style network controller using AWS Lambda. Each finding highlights a critical aspect of system performance, automation accuracy, and threat response capability.

6.1 Successful Flow Log Capture

The VPC Flow Logs were effectively configured to monitor network traffic to and from the EC2 instance. The logs were successfully streamed to CloudWatch, ensuring full visibility into all connection attempts. This foundational step proved critical for detecting unauthorized access patterns and feeding data to the Lambda function for real-time analysis.

6.2 Lambda Detection Logic Worked as Expected

The custom Lambda function, written in Python, accurately parsed flow logs and identified unauthorized SSH login attempts. It correctly filtered logs for port 22, isolated unknown or suspicious IP addresses, and triggered the appropriate response. This validated the effectiveness of the custom detection logic embedded in the function.

6.3 Automated Policy Enforcement

Upon detecting malicious traffic, the Lambda function dynamically updated the Network ACL (NACL) to block the source IP address. This confirmed that real-time policy enforcement was functioning correctly, without requiring manual intervention, thereby achieving the goal of an autonomous SDN controller.

6.4 EventBridge Scheduling Triggered Lambda Reliably

The AWS EventBridge schedule rule was configured to trigger the Lambda function every 5 minutes. During testing, the trigger executed consistently and reliably, allowing for continuous monitoring. This regular execution cycle ensures timely detection and response to any future suspicious traffic.

6.5 Confirmed Blockage and SSH Denial

Following the IP block by the Lambda, SSH access attempts from the attacker IP were met with connection resets, confirming that the mitigation mechanism worked in real-time.

7. Recommendations

Based on the implementation and observations, the following recommendations can enhance the security, scalability, and robustness of the system.

7.1 Expand Policy Coverage Beyond Port 22

Currently, the controller targets port 22 (SSH). However, attackers may exploit other common ports such as HTTP (80), HTTPS (443), RDP (3389), or MySQL (3306). Expanding the Lambda logic to analyze a broader range of ports would enhance threat detection and protect a wider surface of potential attack vectors.

7.2 Integrate with SNS for Alerting

To increase visibility for administrators, integrating the Lambda function with Amazon SNS would provide real-time alerts via email or SMS whenever an attack is detected or a policy is enforced. This ensures rapid awareness and facilitates human oversight for potentially critical actions.

7.3 Implement IP Whitelisting

Introducing a whitelist feature in the Lambda code can prevent trusted IPs (e.g., internal users, management systems) from being mistakenly blocked. This is especially important in environments with shared or fluctuating IP usage, reducing the risk of service disruptions due to false positives.

7.4 Log Archival and Retention Strategy

While CloudWatch provides good short-term log storage, a long-term retention and archival strategy using Amazon S3 is advisable. Archiving flow logs ensures that historical network data is preserved for forensic analysis, regulatory compliance, or machine learning applications in the future.

7.5 Add Rate Limiting and Throttling Controls

Instead of instantly blocking an IP after one connection attempt, implementing rate-limiting logic (e.g., allow 3 failed attempts within 5 minutes) adds nuance to the detection mechanism. This helps avoid false positives from legitimate users making occasional mistakes, while still preventing brute-force attempts.

8. Conclusion

The implementation of a Lambda-based SDN-style network controller within the AWS environment was both technically effective and strategically valuable. Through structured phases—starting from the creation of VPC components and EC2 deployment, to the configuration of flow logs and Lambda automation—the system was designed to monitor and respond to suspicious network activity in real-time. The core mechanism relied on analyzing VPC Flow Logs, detecting unauthorized SSH login attempts, and dynamically modifying Network ACLs to block malicious IP addresses. The use of AWS Lambda, IAM roles, and EventBridge scheduling ensured that the entire process remained automated, scalable, and secure.

Moreover, the test simulation using SSH login attempts confirmed the accuracy of the detection logic and the reliability of the enforcement mechanism. The solution also upheld core cloud security principles such as least privilege access, real-time monitoring, and automated mitigation. The success of this project demonstrates a practical, lightweight alternative to traditional SDN

controllers—well-suited for dynamic cloud environments. Going forward, integrating alerting systems, expanding protocol coverage, and incorporating analytics dashboards could further enhance visibility, control, and response capabilities.