**Task on Monitoring- CloudTrail & CloudWatch**

**1) Enable cloudtrail monitoring and store the events in s3 and cloudwatch log events.**

**Task Title :**

**Enable CloudTrail and Store Logs in S3 & CloudWatch Logs**

**Objective :**

**Enable AWS CloudTrail to monitor all API events and store them securely in S3 and CloudWatch Logs for auditing and monitoring.**

**Prerequisites :**

* **IAM Admin / CloudTrail permissions**
* **S3 bucket (CloudTrail can create)**
* **CloudWatch Log Group / IAM role for CloudTrail**

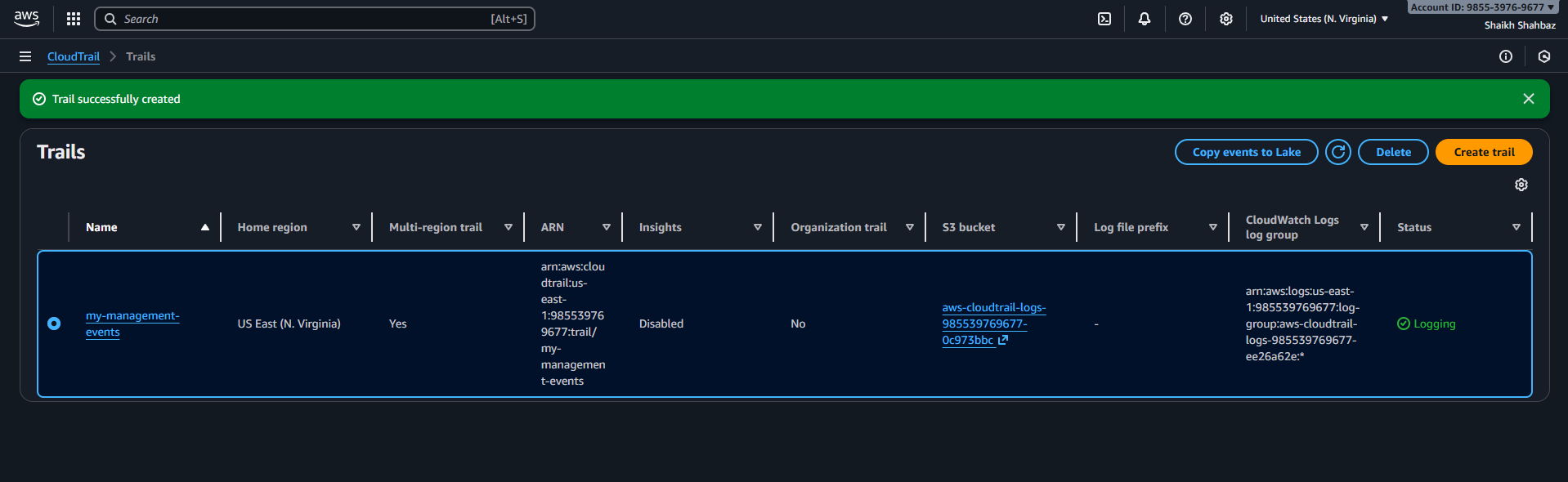
**Step-by-Step Implementation (With Evidence)**

**Step 1: Create a CloudTrail Trail**

**Console Path:  
AWS Console → CloudTrail → Trails → Create Trail**

**Explanation:  
Creates a trail to capture management**

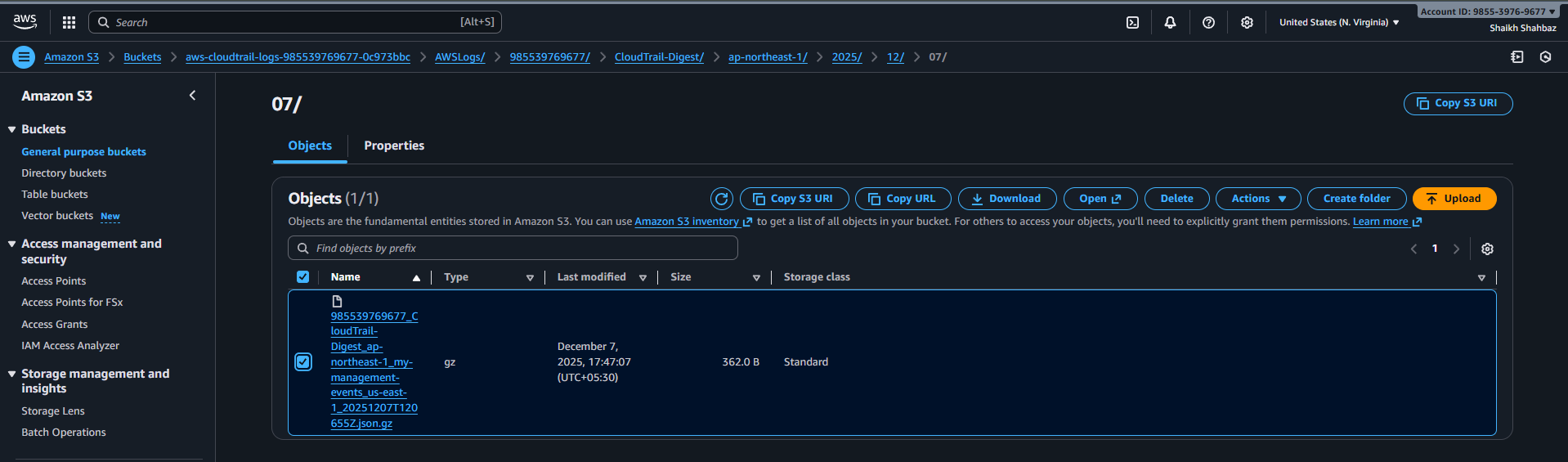
**Evidence:**

******

**Step 2: Configure S3 Bucket for CloudTrail**

**Explanation:  
S3 bucket stores all CloudTrail logs.**

**Evidence:  
*(Screenshot of bucket configuration)***

****

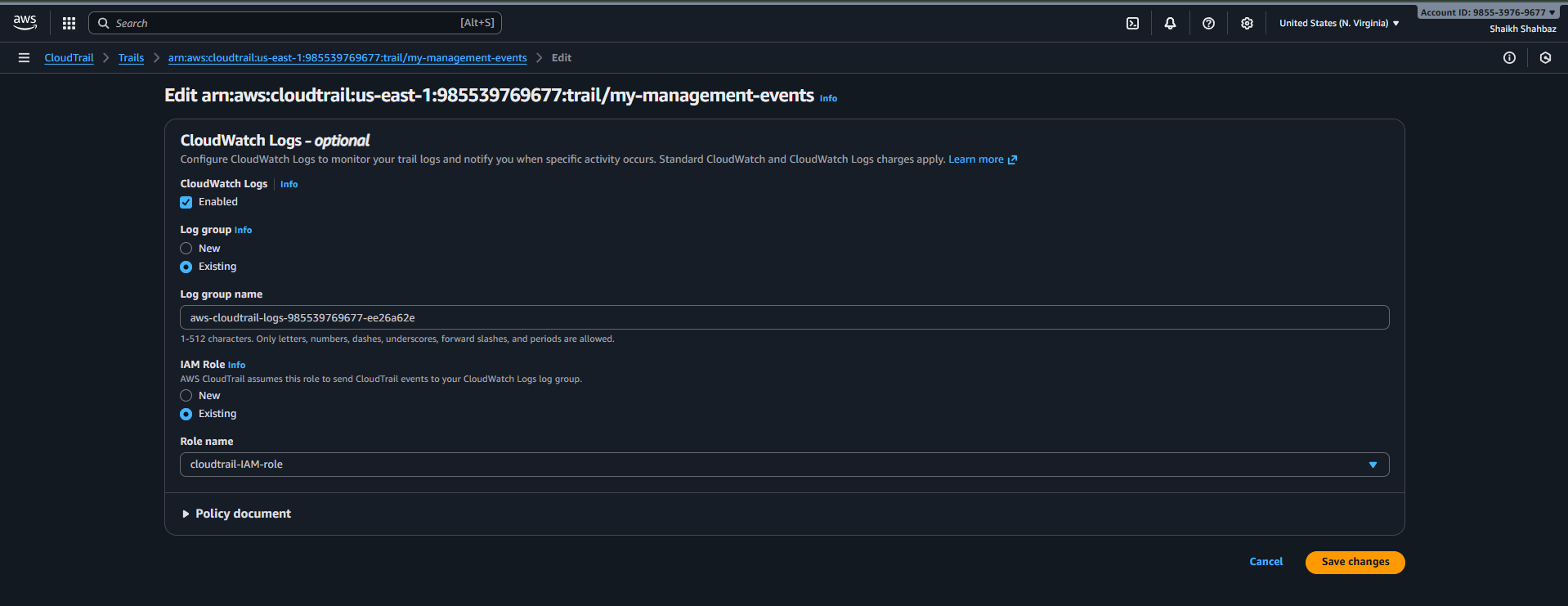
**Step 3: Enable CloudWatch Logs Integration**

**Console Path:  
Trail settings → CloudWatch Logs → Enable**

**CloudTrail auto-creates:**

* **Log group**
* **IAM role**

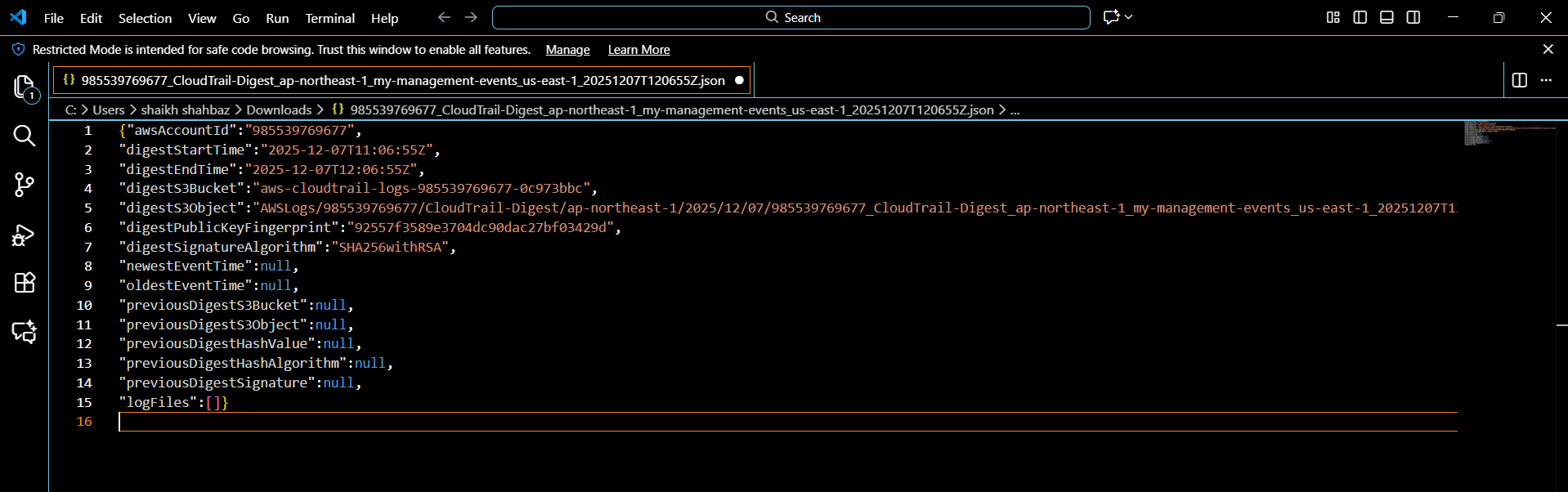
**Evidence:  
*(Screenshot of CloudWatch Logs integration page)***

****

**Step 4: Your trail is been created**

**Download the trail to view all trial logs**

**Evidence:  
*(Screenshot showing trail created & Logging: ON)***

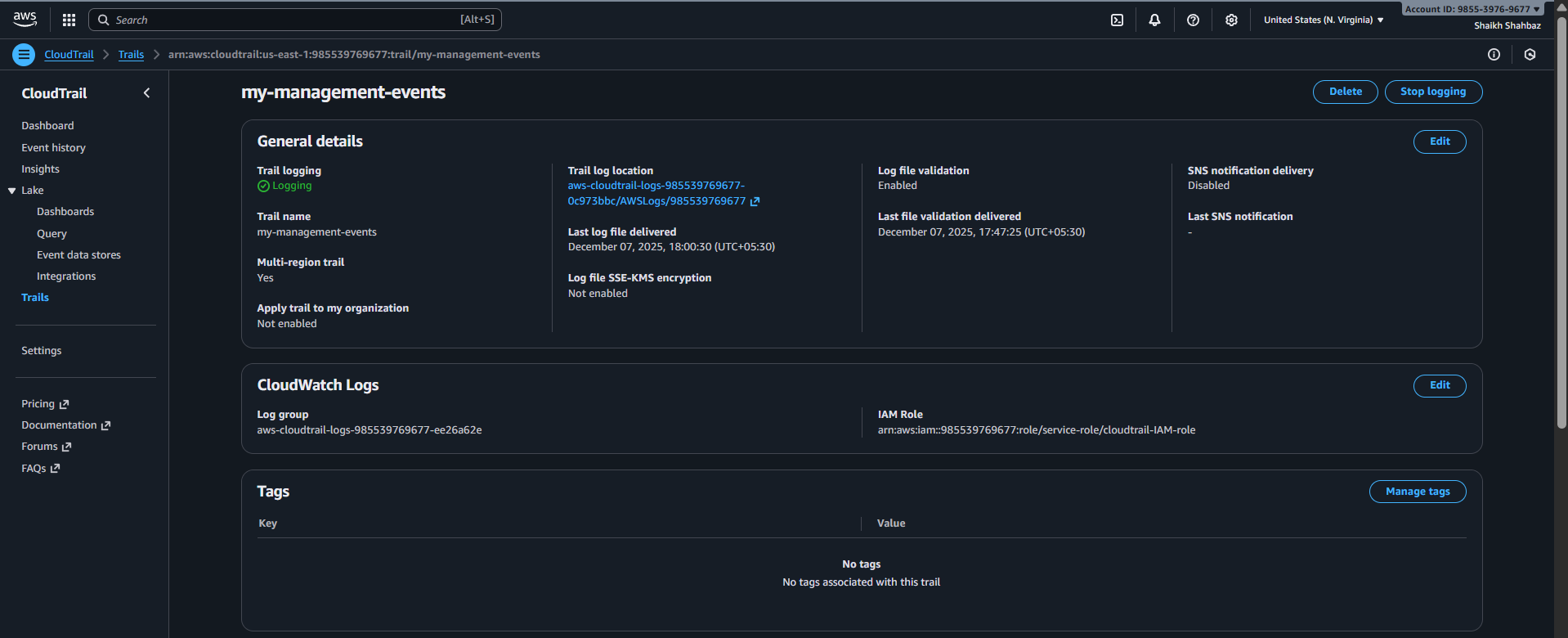
****

**Validation Steps (With Evidence)**

**Validation 1: CloudTrail Logging Status**

**CloudTrail → Trails → Should show Logging: ON**

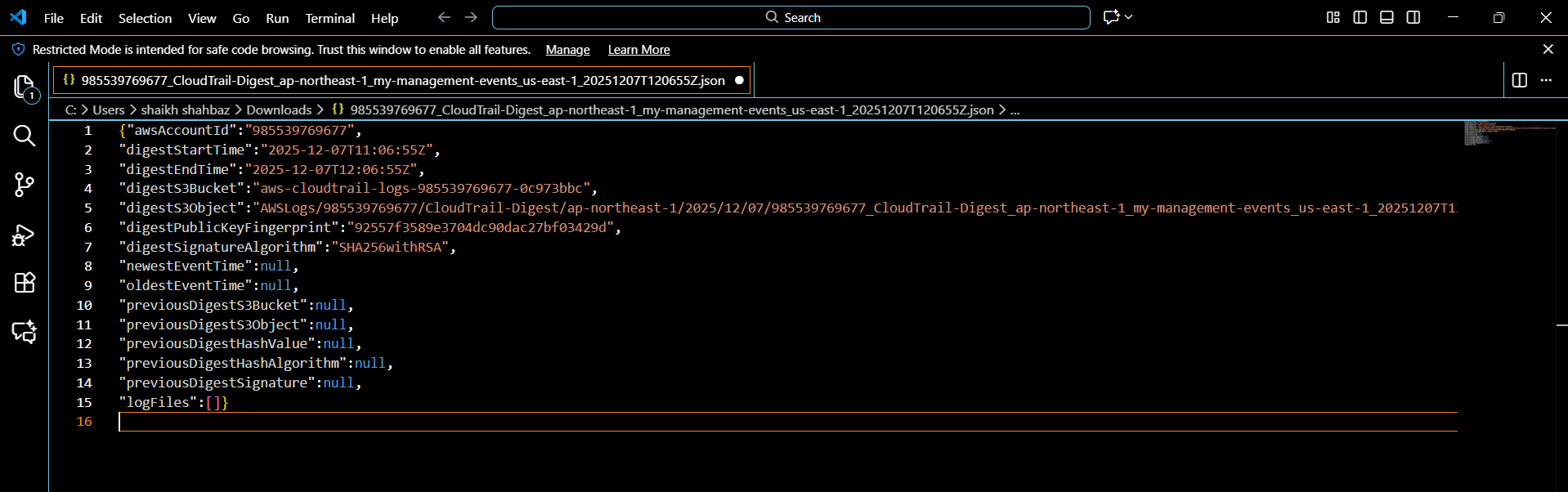
**Evidence:  
*(Screenshot of trail status)***

****

**Validation 2: Logs in S3**

**Go to S3 bucket → Check logs folder  
AWSLogs/<account-id>/CloudTrail/...**

**Evidence:  
*(Screenshot of S3 log files)***

****

**Issues Faced**

**None encountered.**

**Conclusion**

**CloudTrail successfully enabled and configured to send logs to S3 and CloudWatch.**

**2) Enable SNS for cloudtrial to send alert on email.**

**Task Title :**

**Enable SNS Alerts for CloudTrail Using CloudWatch Metric Filters**

**Objective :**

**Send email notifications for critical CloudTrail events (e.g., unauthorized API calls).**

**Prerequisites :**

* **CloudTrail enabled**
* **CloudWatch Logs enabled**
* **Valid email address**
* **IAM SNS permissions**

**Step-by-Step Implementation (With Evidence)**

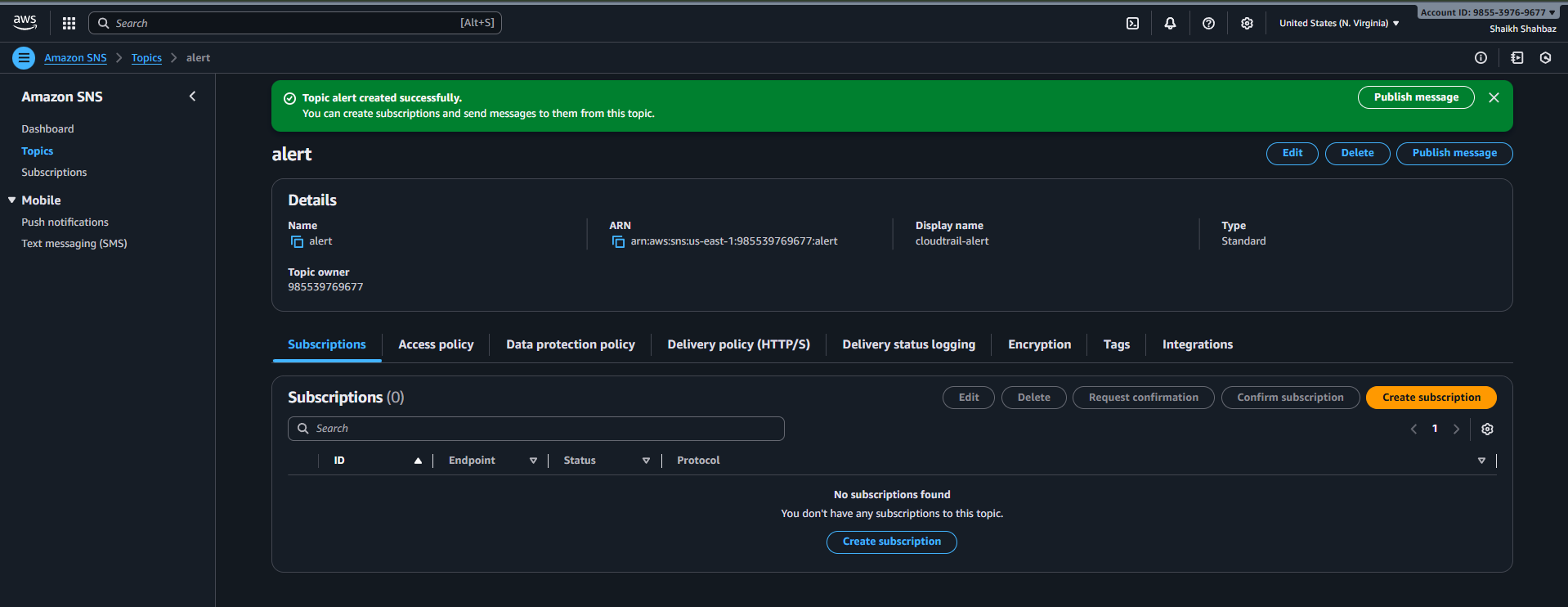
**Step 1: Create an SNS Topic**

**Console Path:  
SNS → Topics → Create Topic**

**Name: cloudtrail-alert**

**Display name – cloudtrail-alert**

**Evidence:  
*(SNS topic screenshot)***

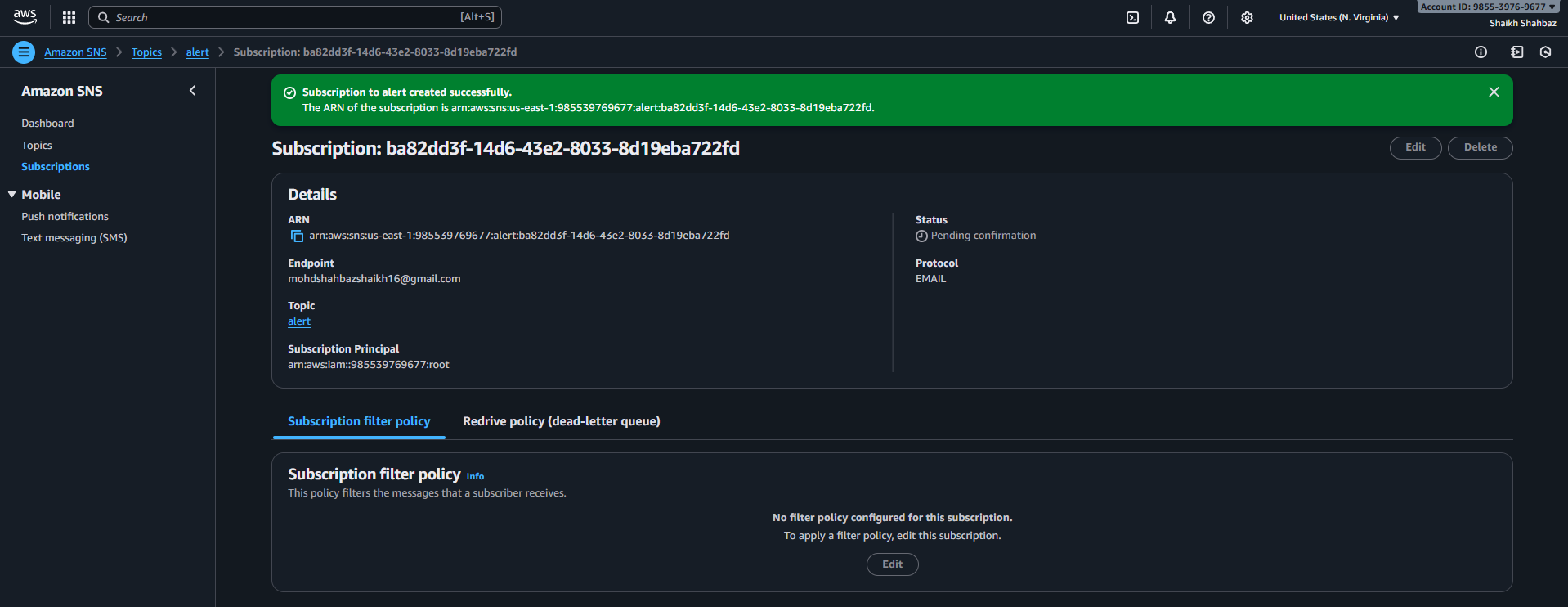
****

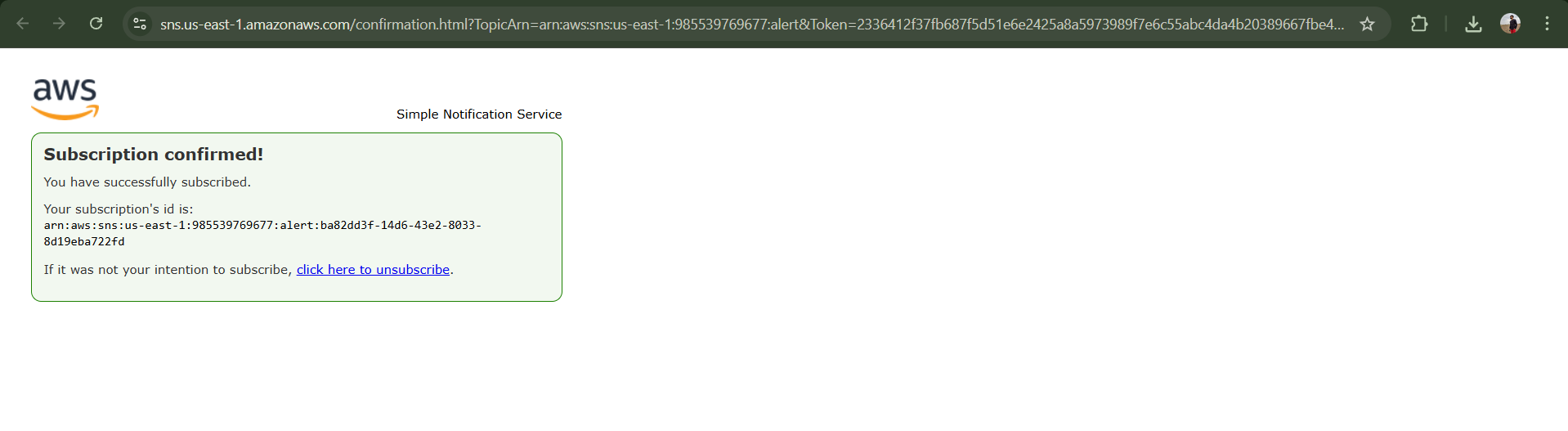
**Step 2: Subscribe Email**

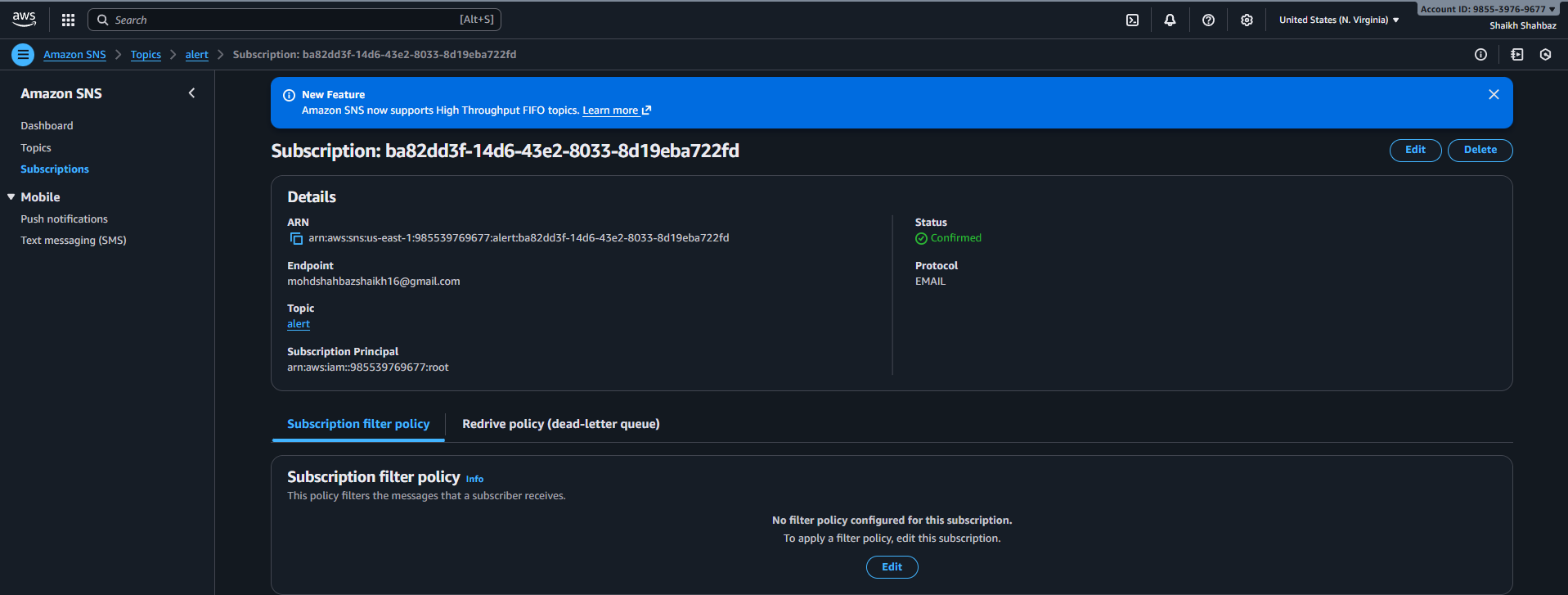
**Protocol: Email  
Enter your email.**

**Confirm subscription from your inbox.**

**Evidence:  
*(Subscription confirmation screenshot)***

****

****

****

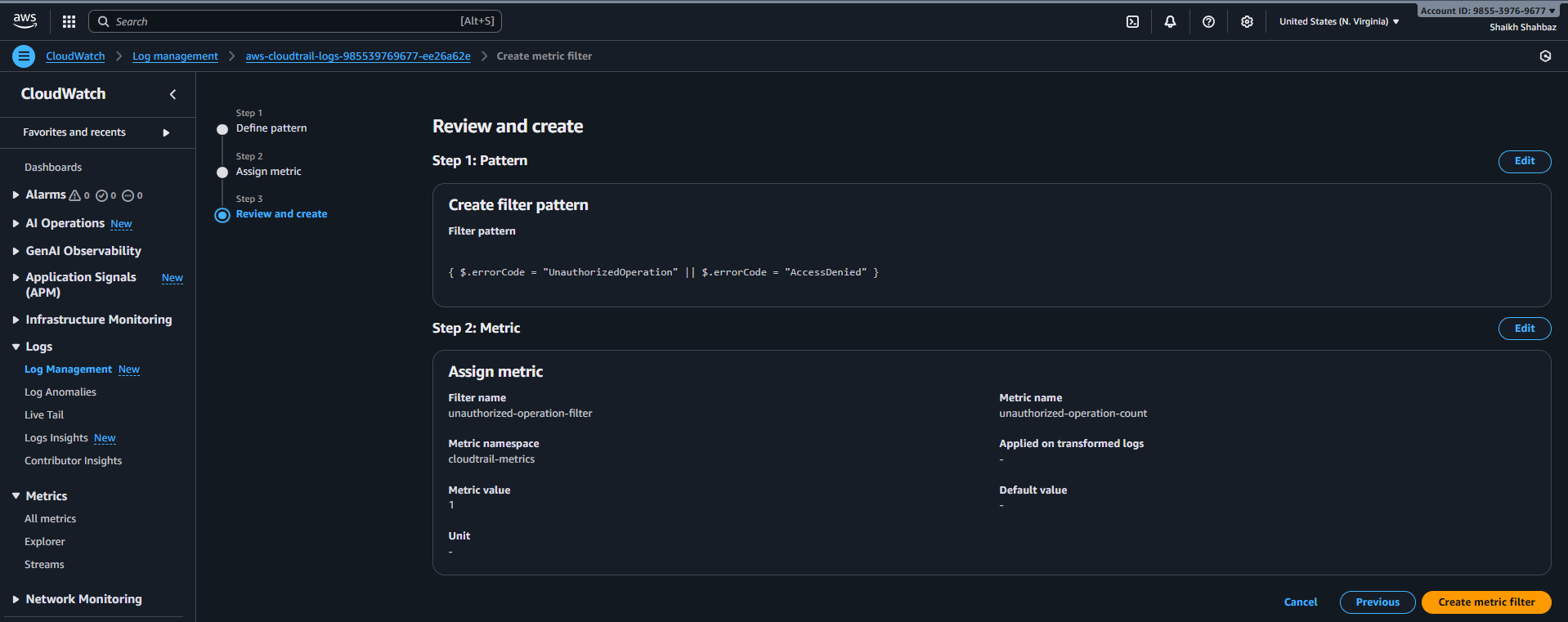
**Step 3: Create CloudWatch Metric Filter**

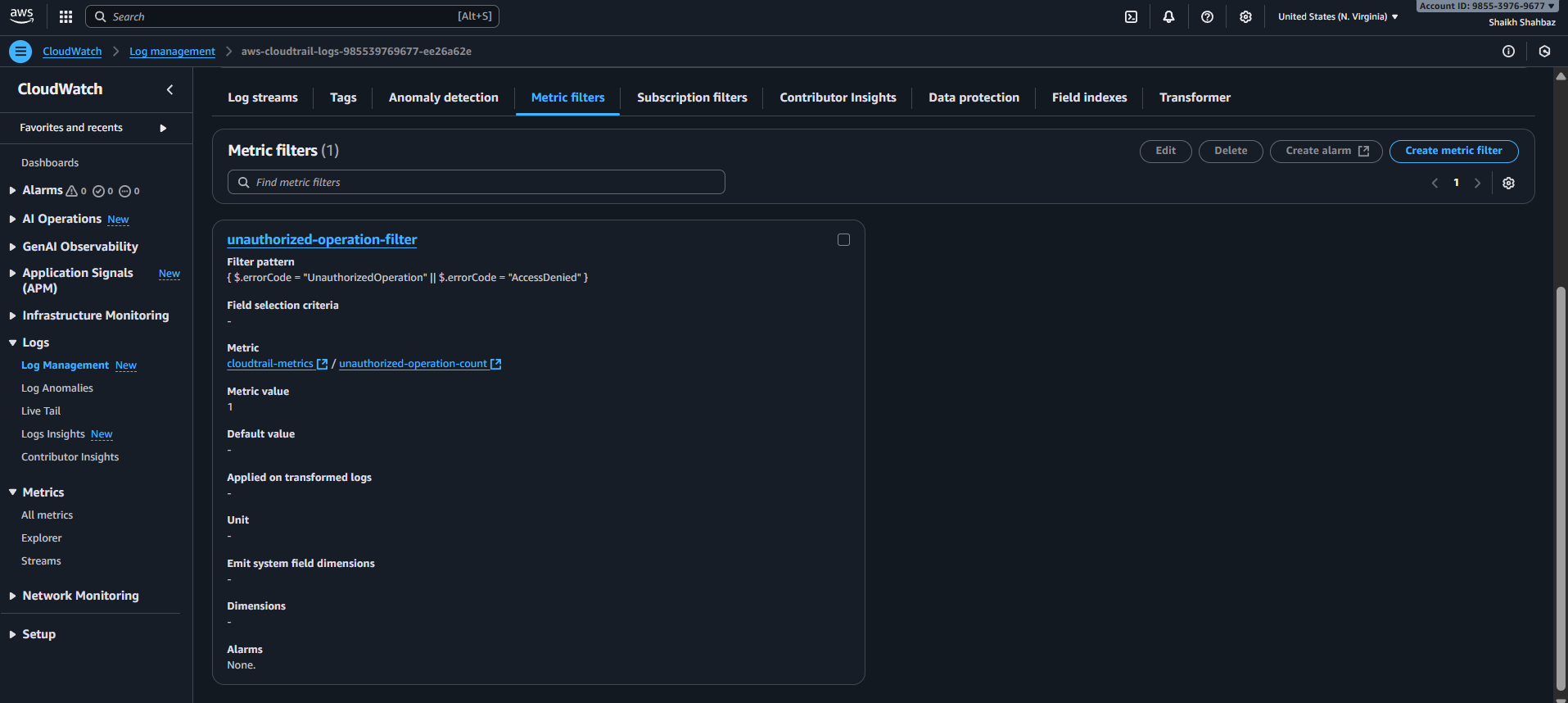
**Console Path:  
CloudWatch → Logs → Log Groups → /aws/cloudtrail/mytrail**

**Filter example:**

**{ $.errorCode = "\*UnauthorizedOperation" }**

**Evidence:  
*(Metric filter screenshot)***

****

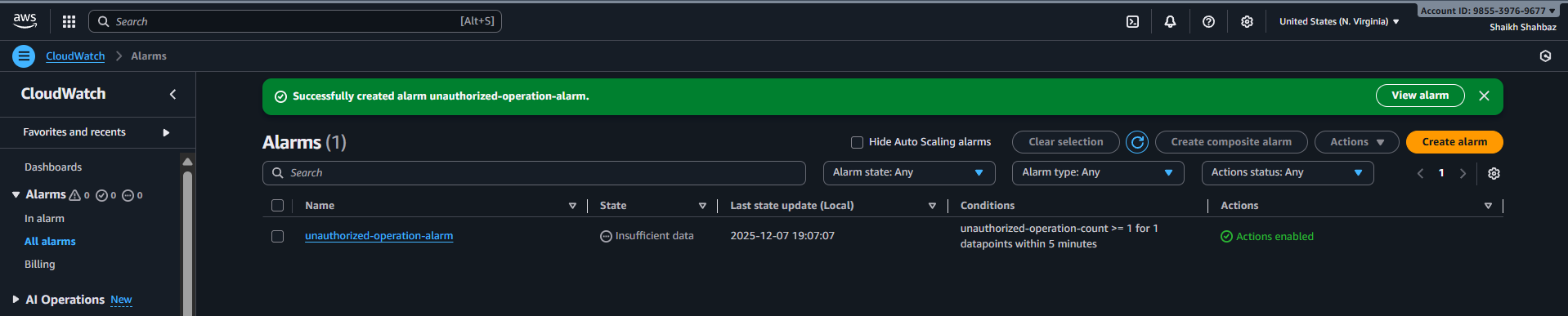
****

**Step 4: Create Alarm for the Filter**

**Console Path:  
CloudWatch → Alarms → Create alarm**

**Notification: cloudtrail-alert-topic**

**Evidence:  
*(Alarm configuration screenshot)***

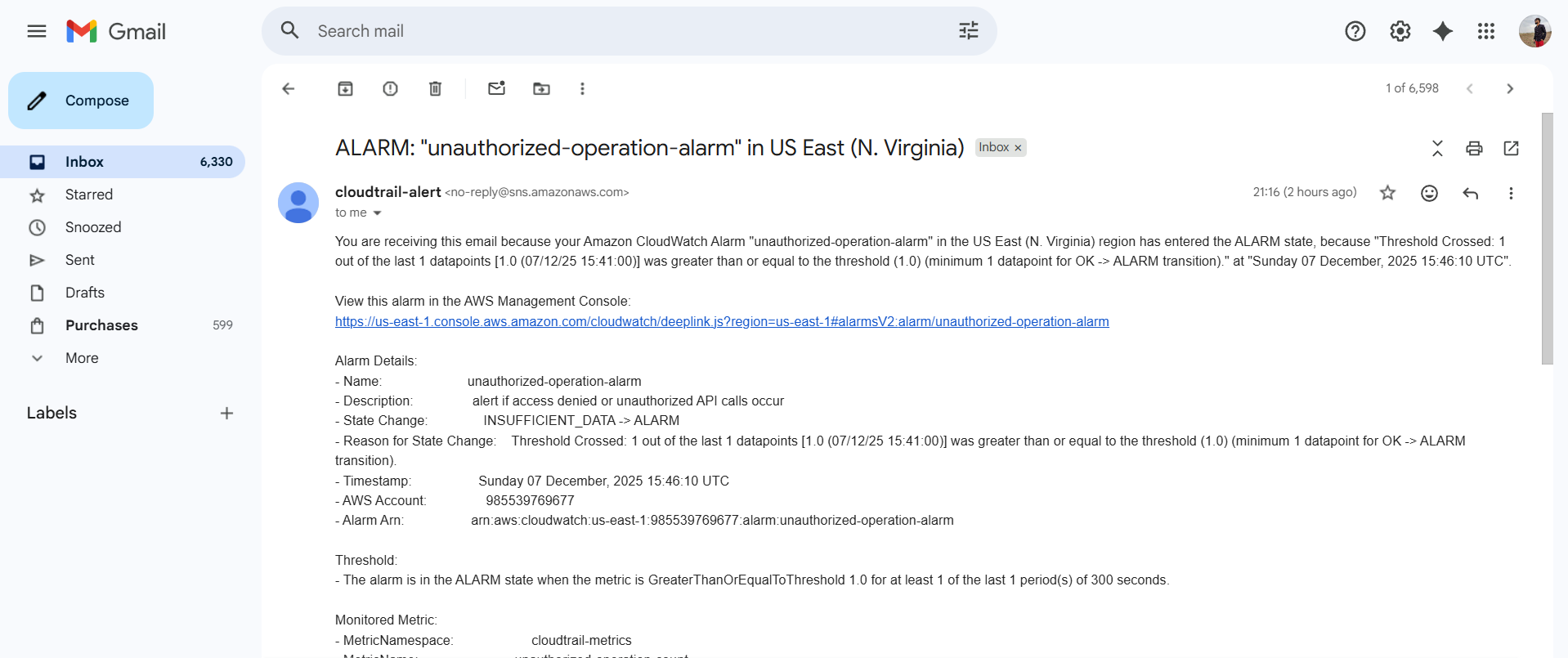
****

**Validation Steps (With Evidence)**

**Validation 1: SNS Email Delivered**

**Trigger a test alert.**

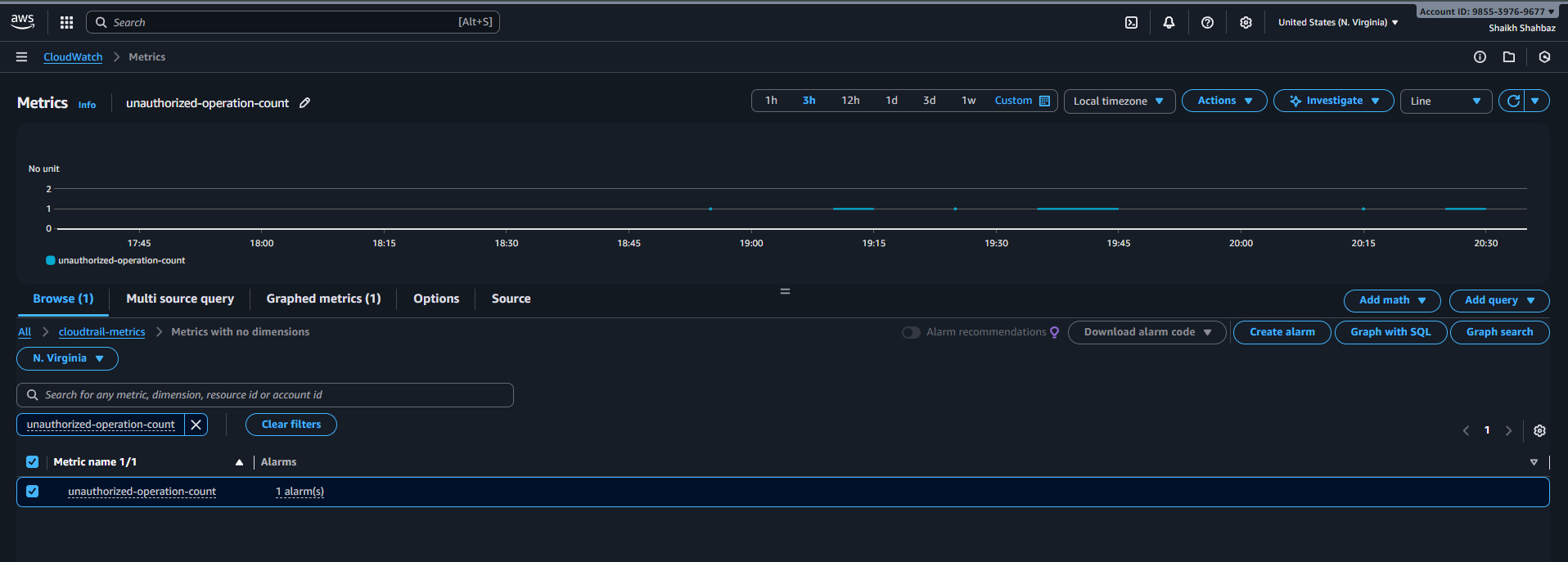
**Evidence:  
*(Email screenshot)***

****

**Validation 2: Metric Filter Counting Logs**

**Metric should show non-zero datapoints after an event.**

**Evidence:  
*(Metric graph screenshot)***

****

**Issues Faced :**

**Email not arriving → Had to confirm subscription.**

**Conclusion :**

**SNS alerting for CloudTrail successfully configured.**

**3) Configure cloud watch monitoring and record the cpu utilization and other metrics of ec2.**

**Task Title :**

**Install & Configure CloudWatch Agent on EC2 for System Monitoring**

**Objective :**

**Collect CPU, Memory, Disk, and EC2 status metrics using CloudWatch Agent.**

**Prerequisites :**

* **EC2 instance**
* **IAM Role with CloudWatch permissions**
* **SSH Access**
* **Internet access (or SSM)**

**Step-by-Step Implementation (With Evidence)**

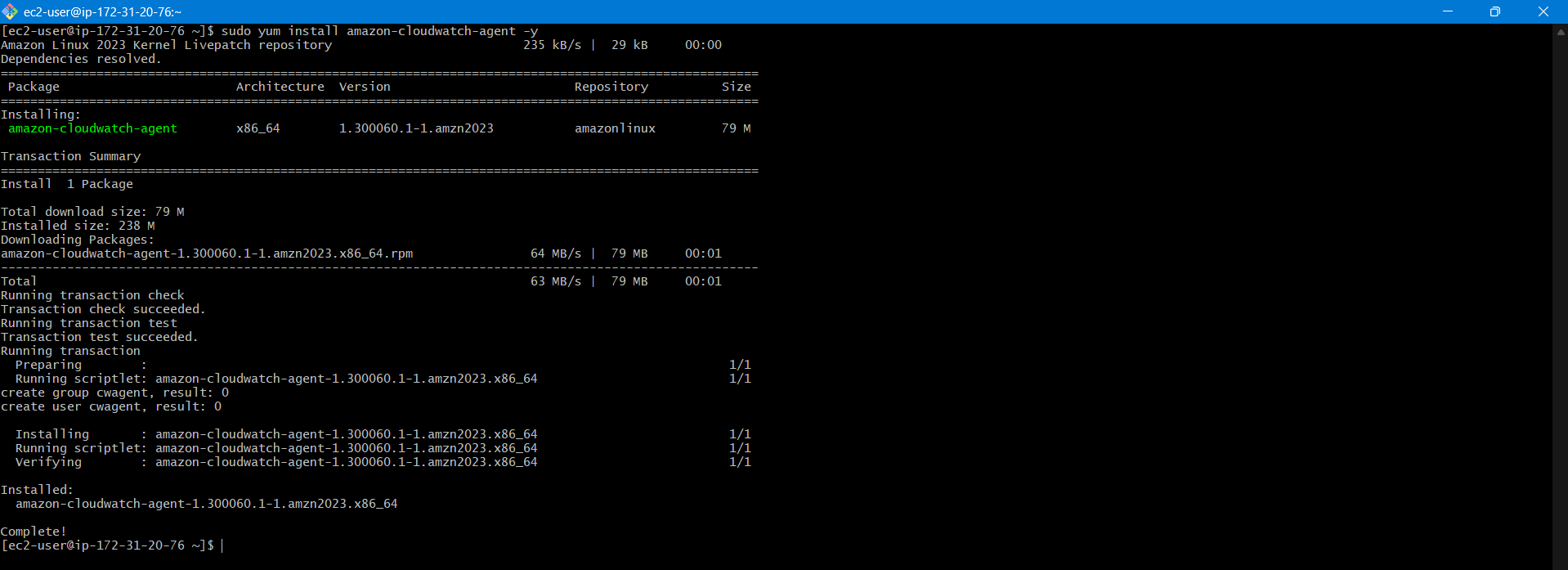
**Step 1: Install CloudWatch Agent**

**SSH into EC2.**

**Amazon Linux:**

**sudo yum install amazon-cloudwatch-agent -y**

**Evidence:  
*(Terminal screenshot)***

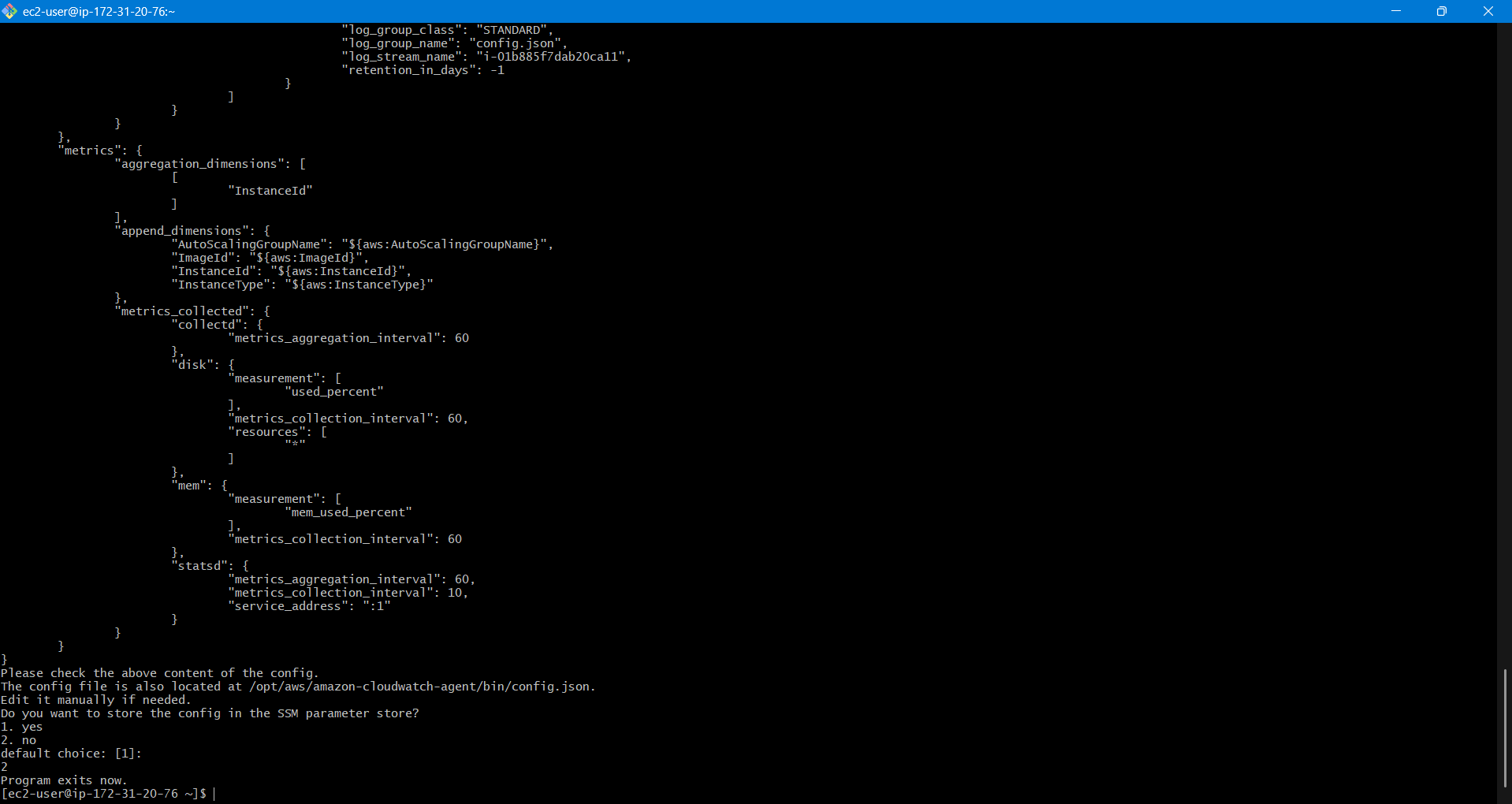
****

**Step 2: Configure the Agent**

**Run config wizard:**

**sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard**

**Evidence:  
*(Wizard terminal output)***

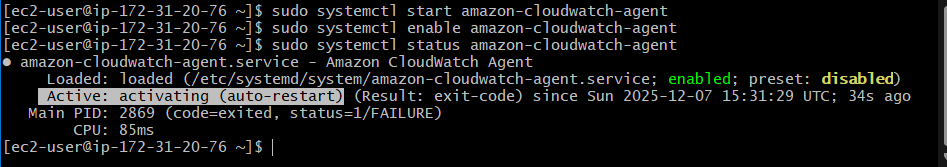
****

**Step 3: Start the Agent**

**sudo systemctl start amazon-cloudwatch-agent**

**sudo systemctl enable amazon-cloudwatch-agent**

**Evidence:  
*(systemctl status agent screenshot)***

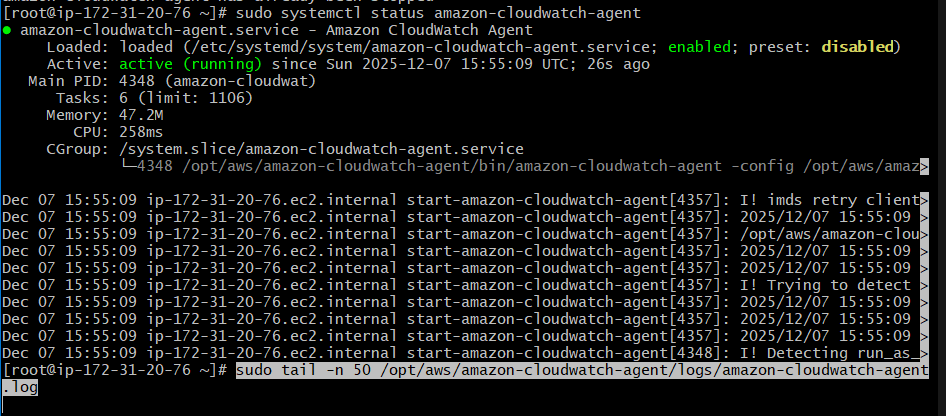
****

**Validation Steps (With Evidence)**

**Validation 1: Agent Running**

**systemctl status amazon-cloudwatch-agent**

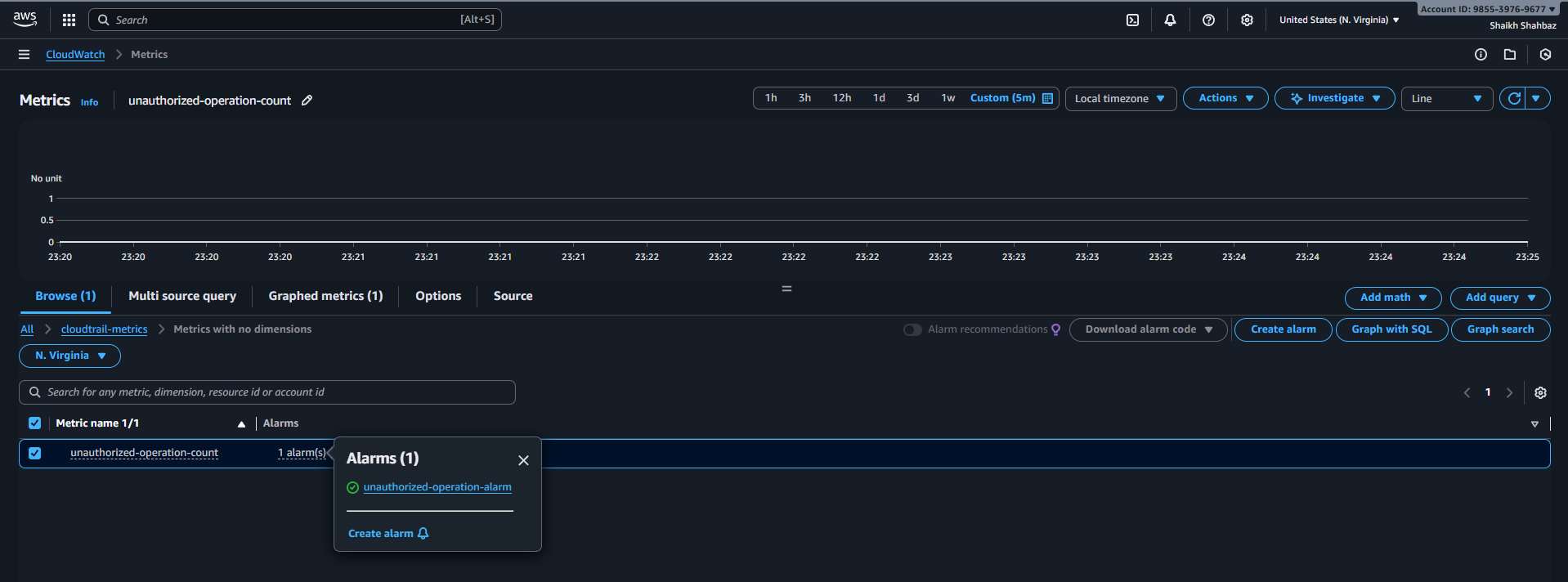
**Evidence: Screenshot**

****

**Validation 2: Metrics Visible**

**CloudWatch → Metrics → CWAgent namespace**

**Evidence: Screenshot**

****

**Issues Faced :**

**Had to wait for email and logs to form.**

**Conclusion :**

**CloudWatch Agent installed and EC2 metrics are available.**

**4) Create one alarm to send alert to email if the cpu utilization is more than 70 percent.**

**Task Title :**

**Configure CloudWatch Alarm for CPU > 70%**

**Objective :**

**Create a CPU alarm that triggers an SNS email notification when load exceeds 70%.**

**Prerequisites :**

* **SNS Topic**
* **CloudWatch metrics**
* **EC2 instance**

**Step-by-Step Implementation (With Evidence)**

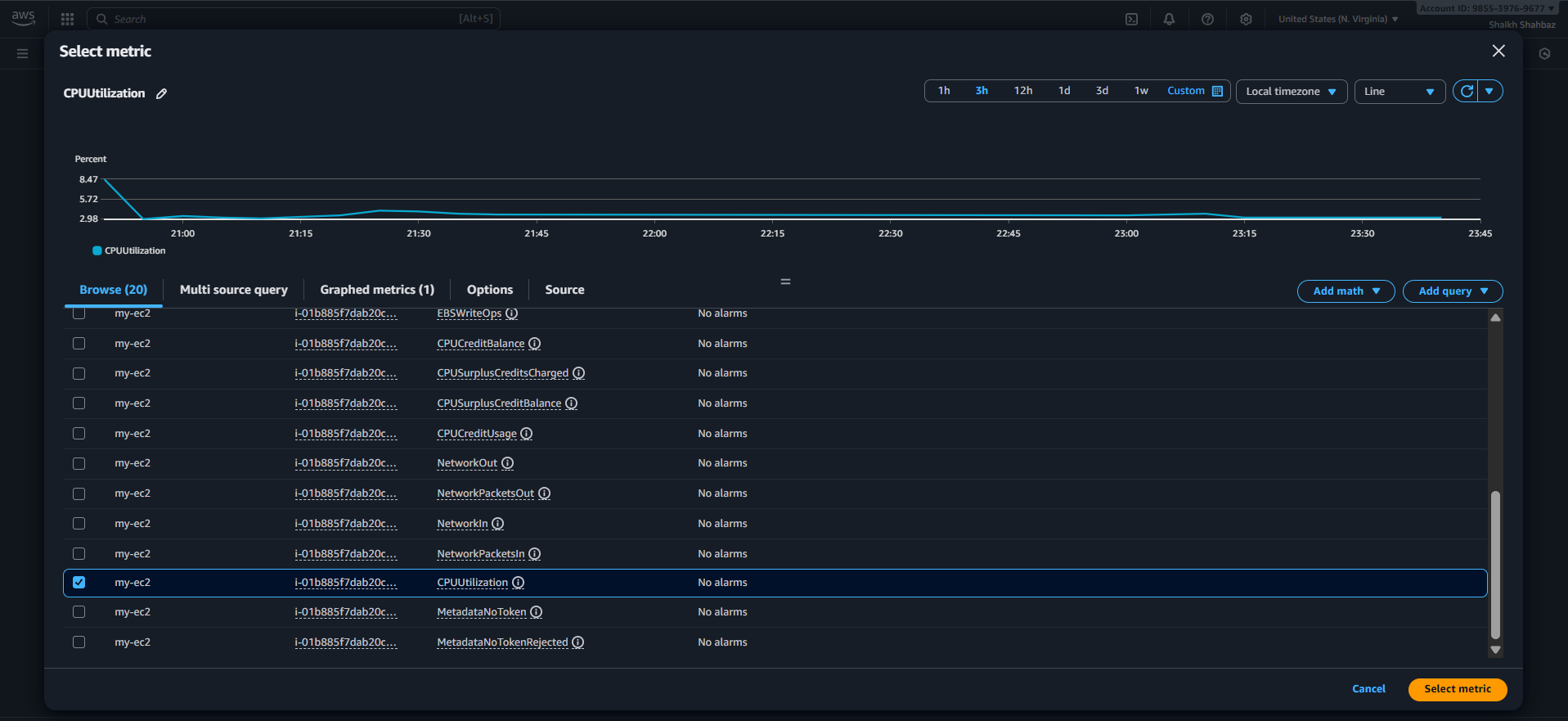
**Step 1: Create Alarm**

**Console Path:  
CloudWatch → Alarms → Create Alarm**

**Metric:**

* **EC2 → Per-instance metrics → CPUUtilization**

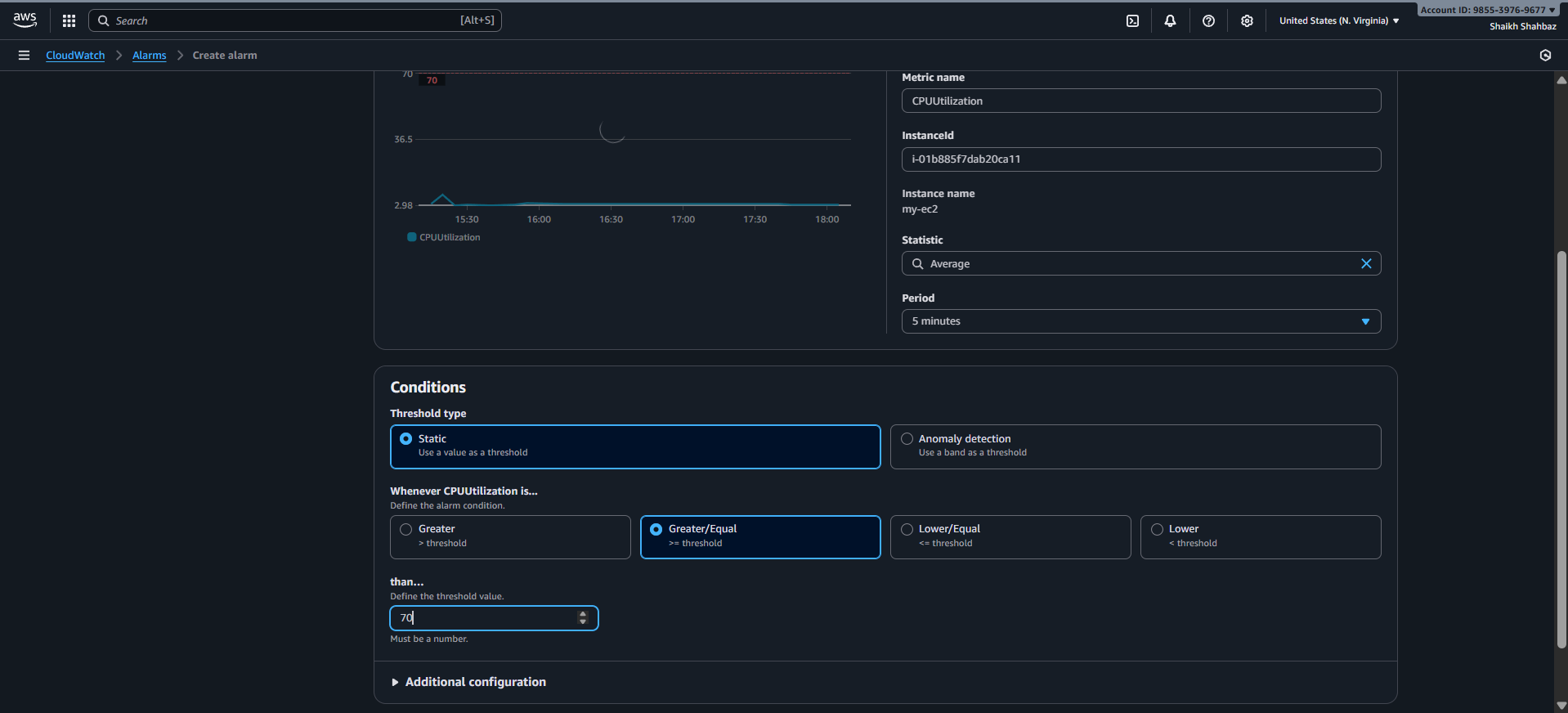
**Evidence:  
*(Metric selection screenshot)***

****

**Step 2: Configure Threshold**

* **Static threshold**
* **CPU > 70%**
* **Period: 5 minutes**

**Evidence:  
*(Threshold screenshot)***

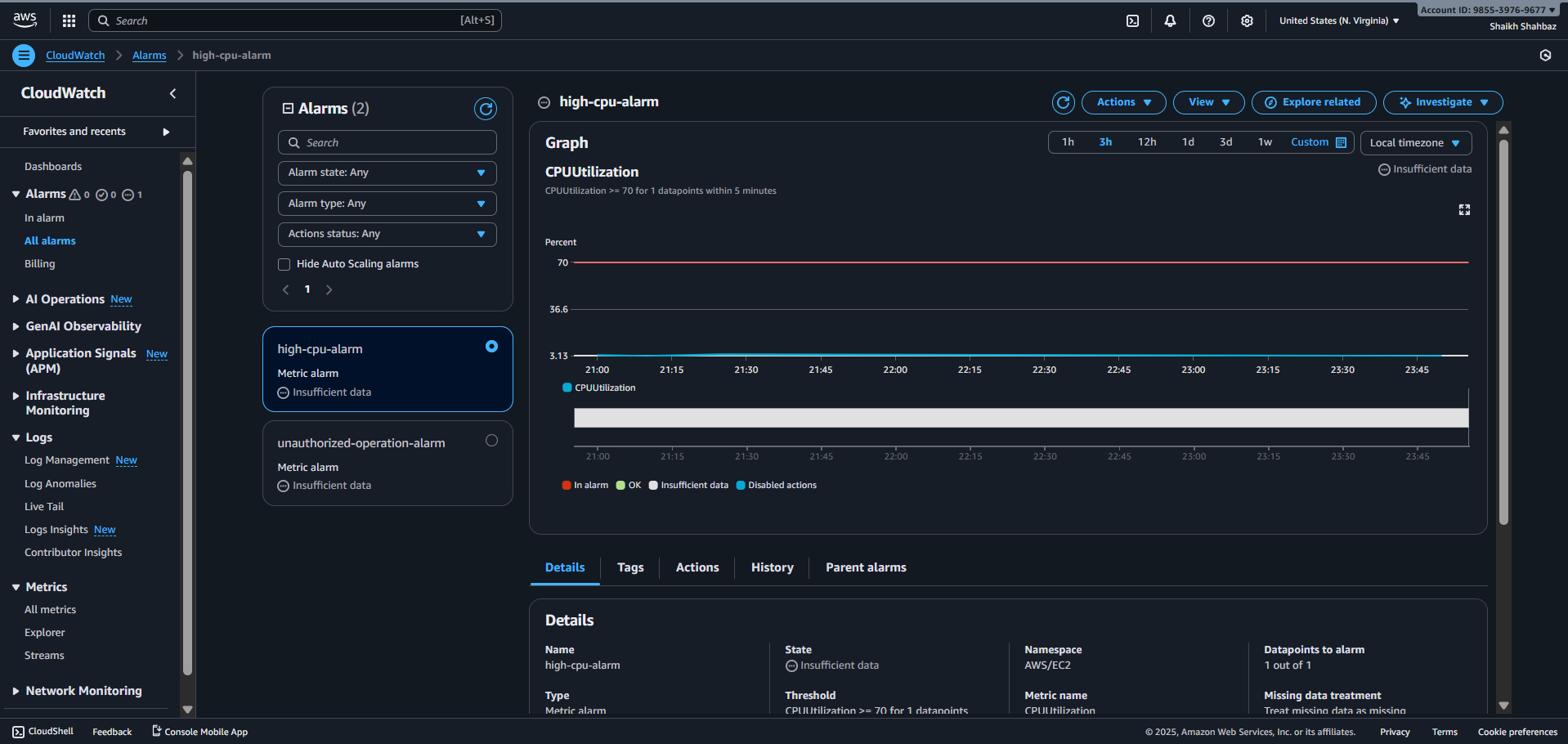
****

**Step 3: Configure Alert Action**

**Notification target: high-cpu-alert**

**Evidence:  
*(SNS action screenshot)***

****

****

**Validation Steps (With Evidence)**

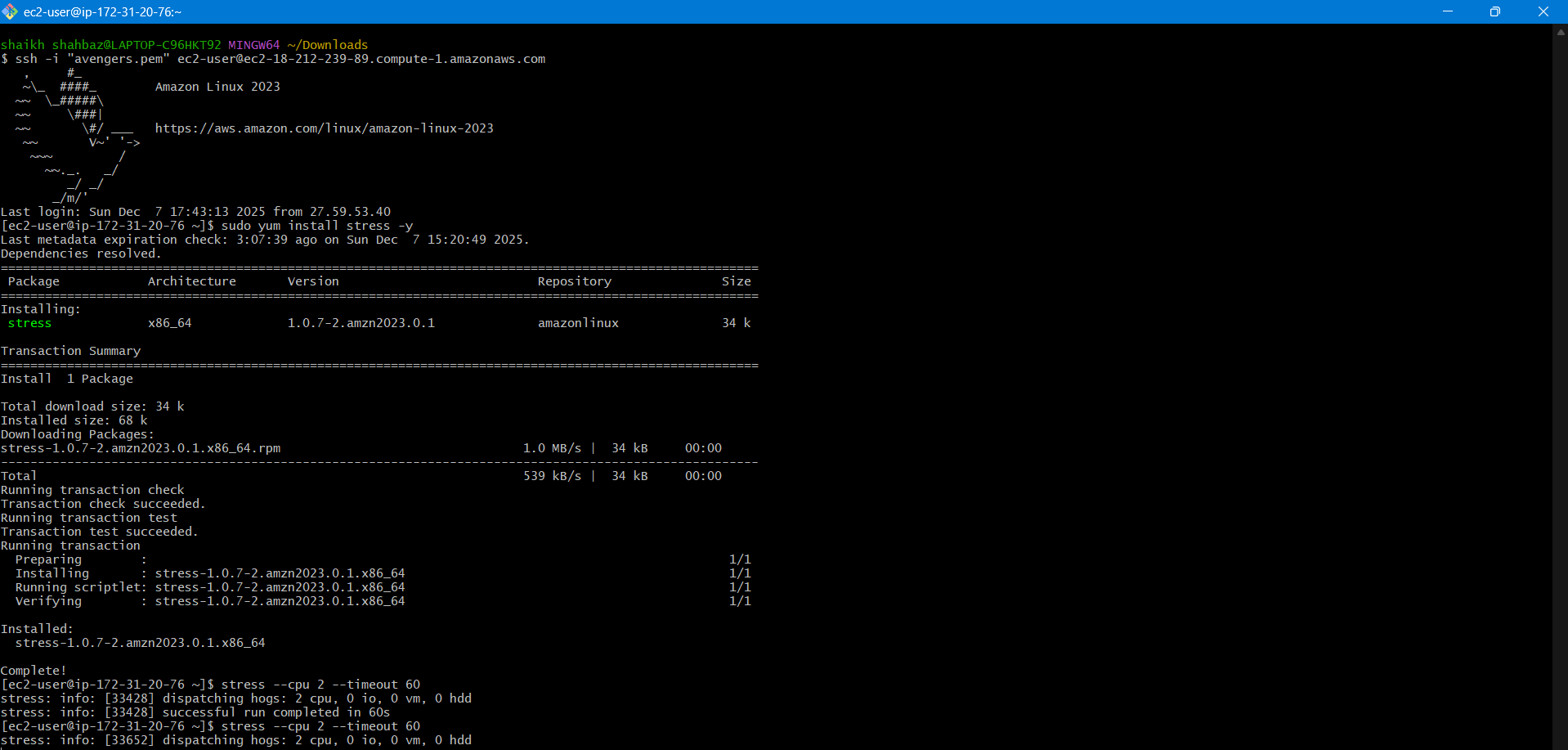
**Validation 1: CPU Stress Test**

**Run:**

**sudo yum install stress -y**

**stress --cpu 2 --timeout 60**

**Evidence: CPU spike graph screenshot**

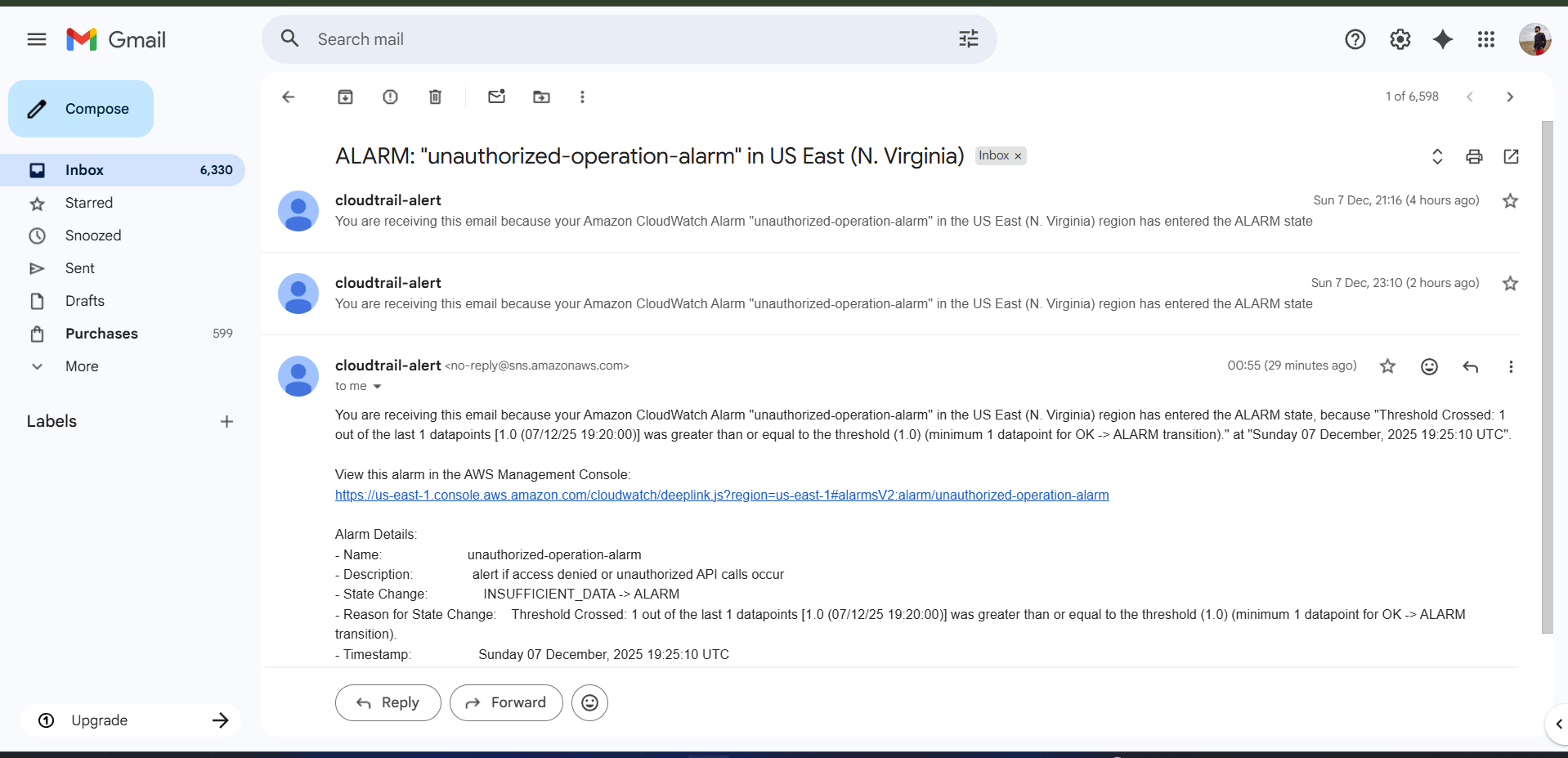
****

****

****

**Validation 2: SNS Email Received**

**Evidence: Email screenshot**

****

**Issues Faced:**

**Had to wait for CPU utilization to cross 70%.**

**For that I have applied stress multiple time and then checked for cpu utilization using top command multiple times.**

**Conclusion:**

**CPU alarm configured successfully.**

**5) Create Dashboard and monitor tomcat service wether it is running or not and send the alert.**

**Task Title :**

**Monitor Tomcat Service Using Custom CloudWatch Metrics**

**Objective :**

**Track Tomcat service uptime and trigger alerts if service stops.**

**Prerequisites :**

* **EC2 instance with Tomcat installed**
* **CloudWatch Agent or AWS CLI installed**
* **IAM Role with CloudWatch permissions**

**Step-by-Step Implementation (With Evidence)**

**Step 1: Create Tomcat Status Script**

**sudo vi /opt/tomcat-status.sh**

**Add:**

**#!/bin/bash**

**if systemctl is-active --quiet tomcat; then**

**status=1 (Service-active)**

**else**

**status=0 (Service-inactive)**

**fi**

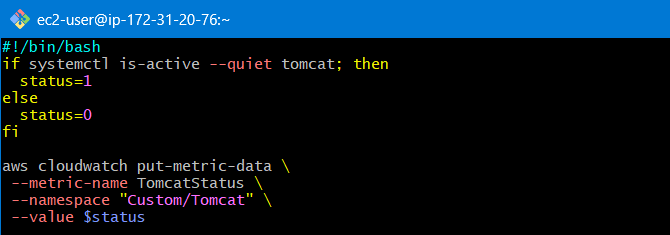
**aws cloudwatch put-metric-data \**

**--metric-name TomcatStatus \**

**--namespace "Custom/Tomcat" \**

**--value $status**

**Evidence:  
Screenshot of script**

****

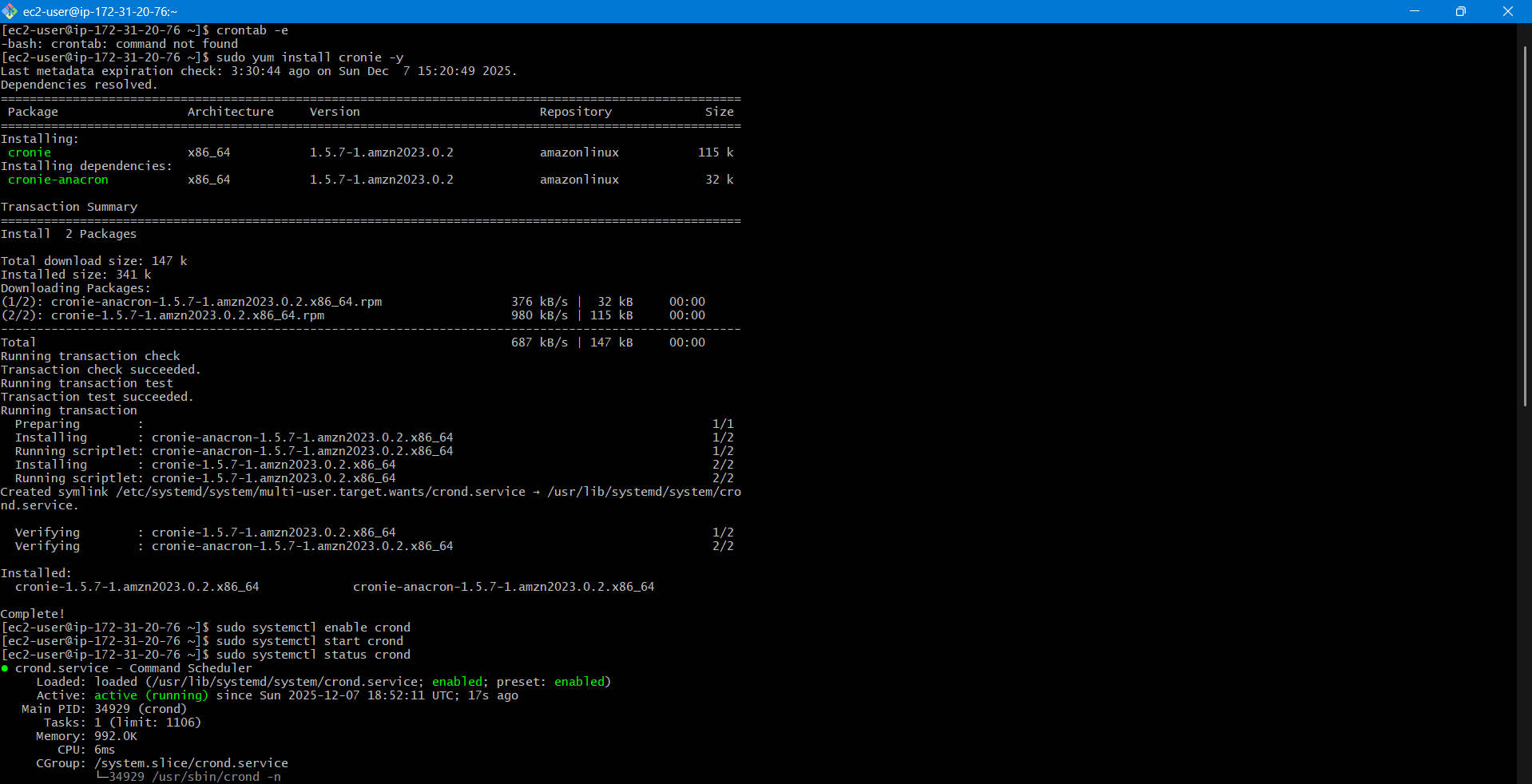
**Step 2: Enable Script via Cron**

**crontab -e**

**Add:**

**\* \* \* \* \* /opt/tomcat-status.sh**

**Evidence:  
Cron screenshot**

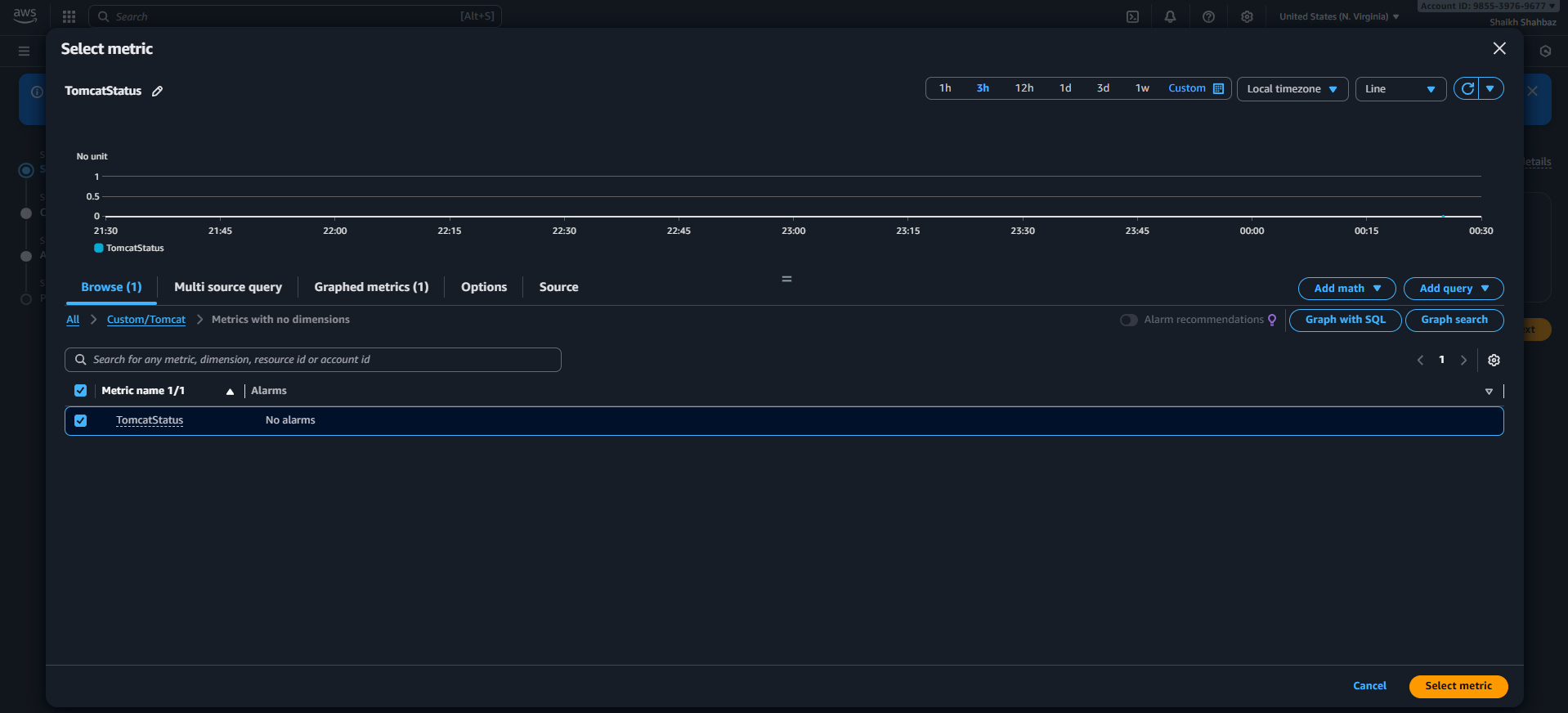
****

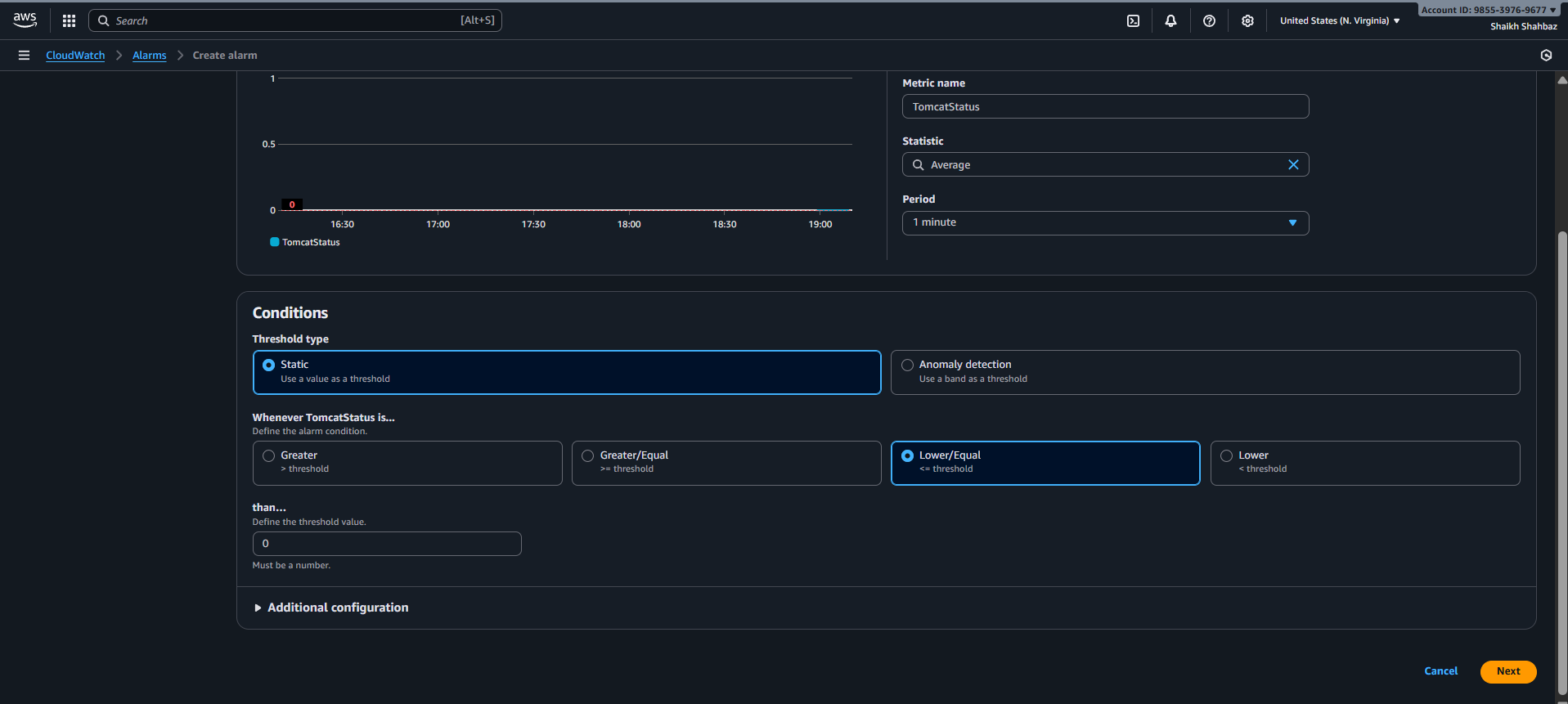
****

**Step 3: Create Alarm**

**Metric: Custom/Tomcat → TomcatStatus  
Alarm when value = 0**

**Evidence: Alarm screenshot**

****

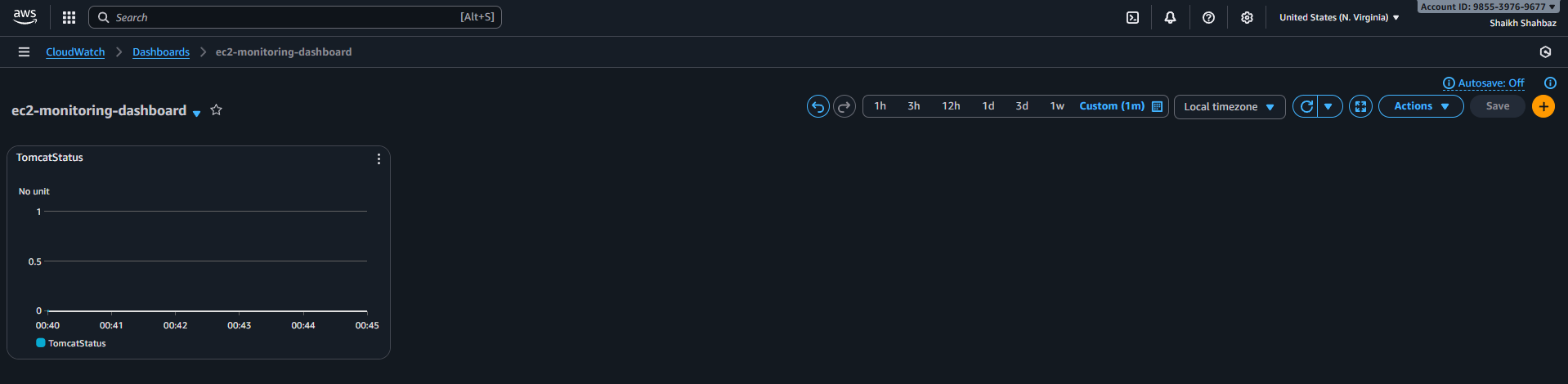
****

****

**Step 4: Add Dashboard Widget**

**CloudWatch → Dashboard → Add Widget → Line Chart  
Select metric: TomcatStatus**

**Evidence: Dashboard screenshot**

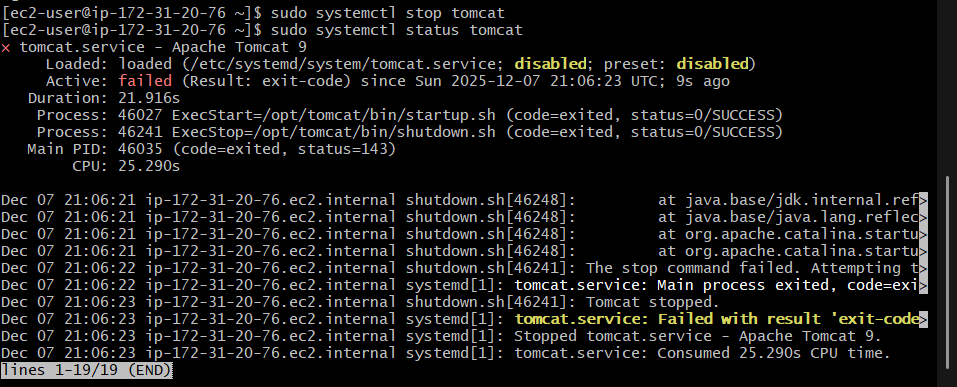
****

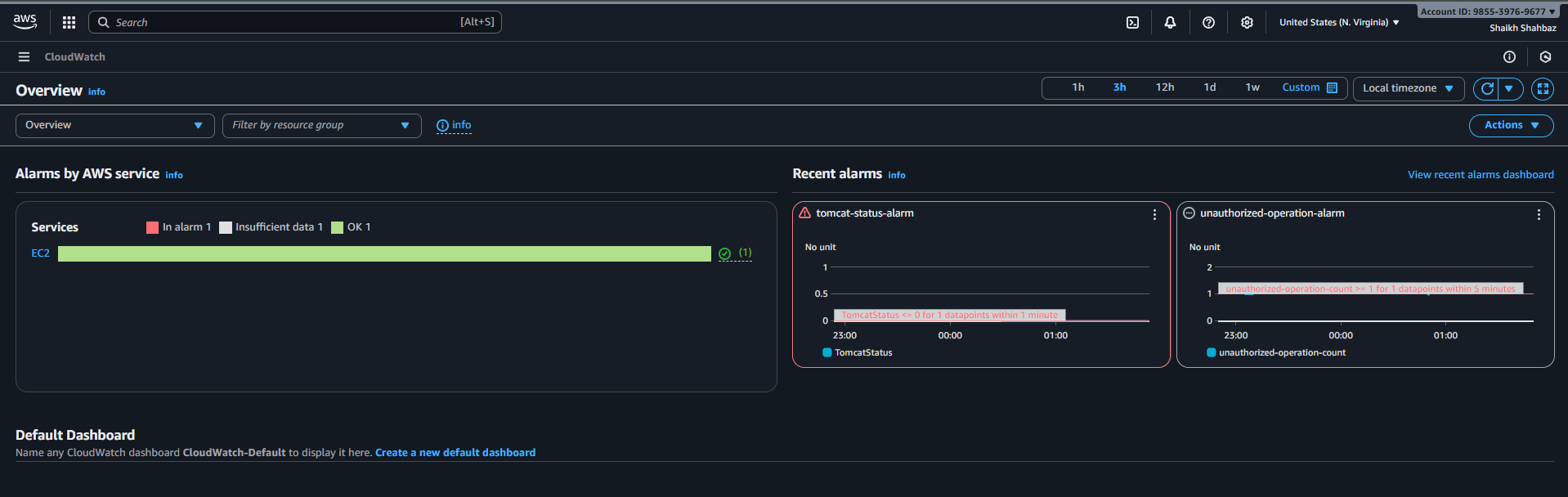
**Validation Steps (With Evidence)**

**Validation 1: Stop Tomcat**

**sudo systemctl stop tomcat**

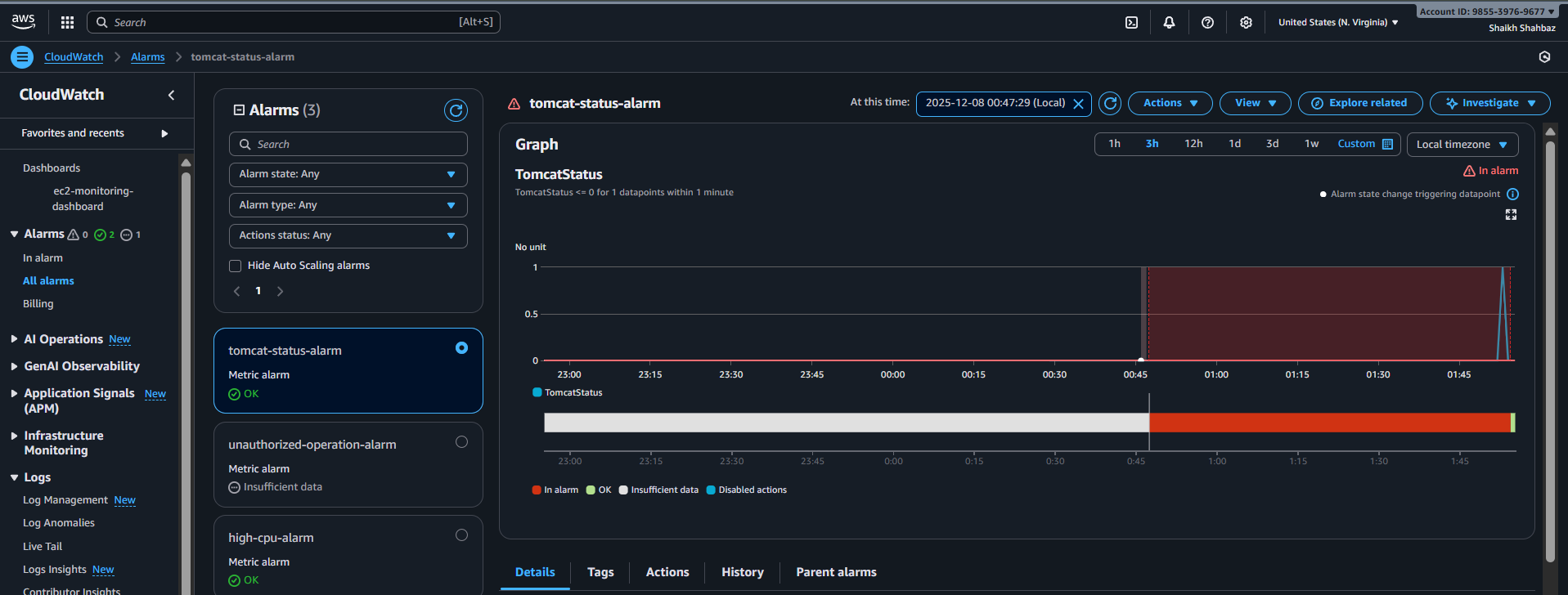
**Evidence: Service inactive screenshot**

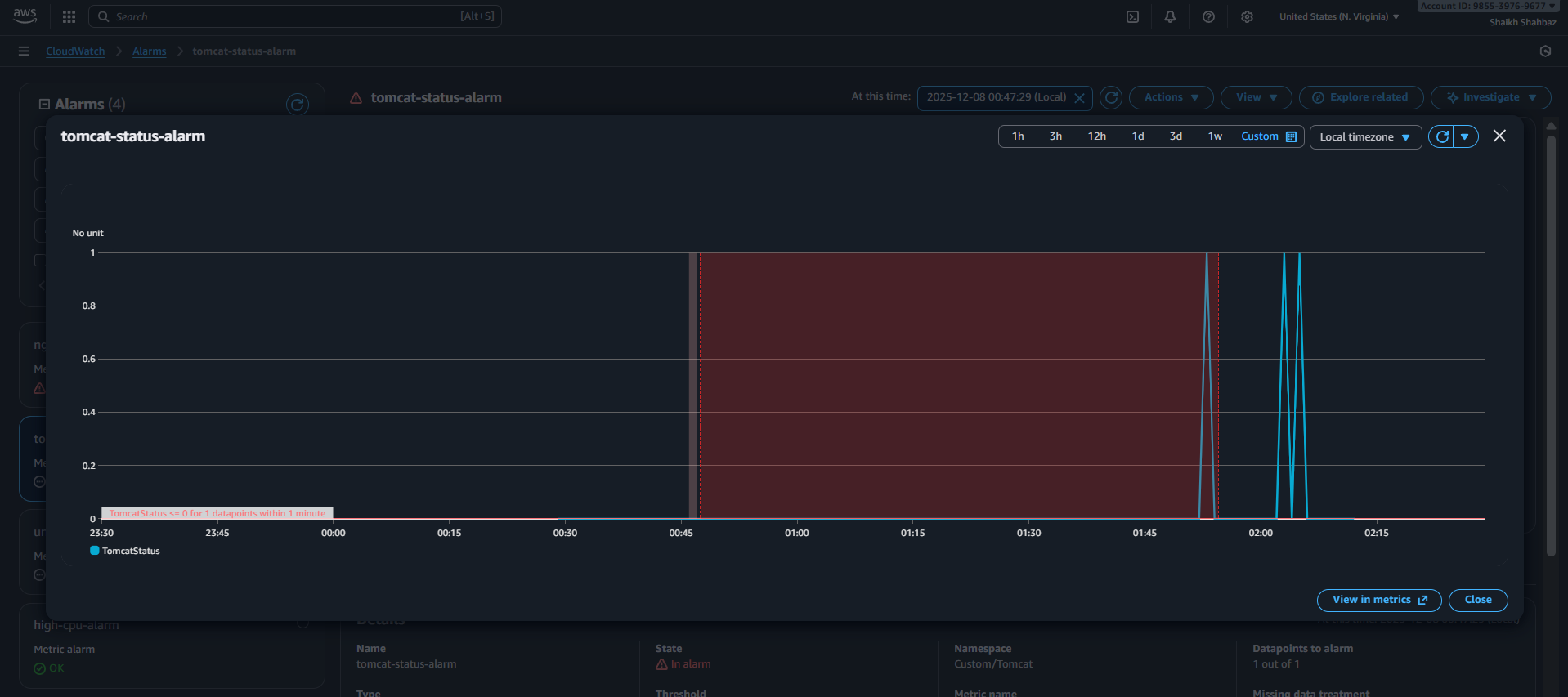
****

****

**Validation 2: Alarm Trigger**

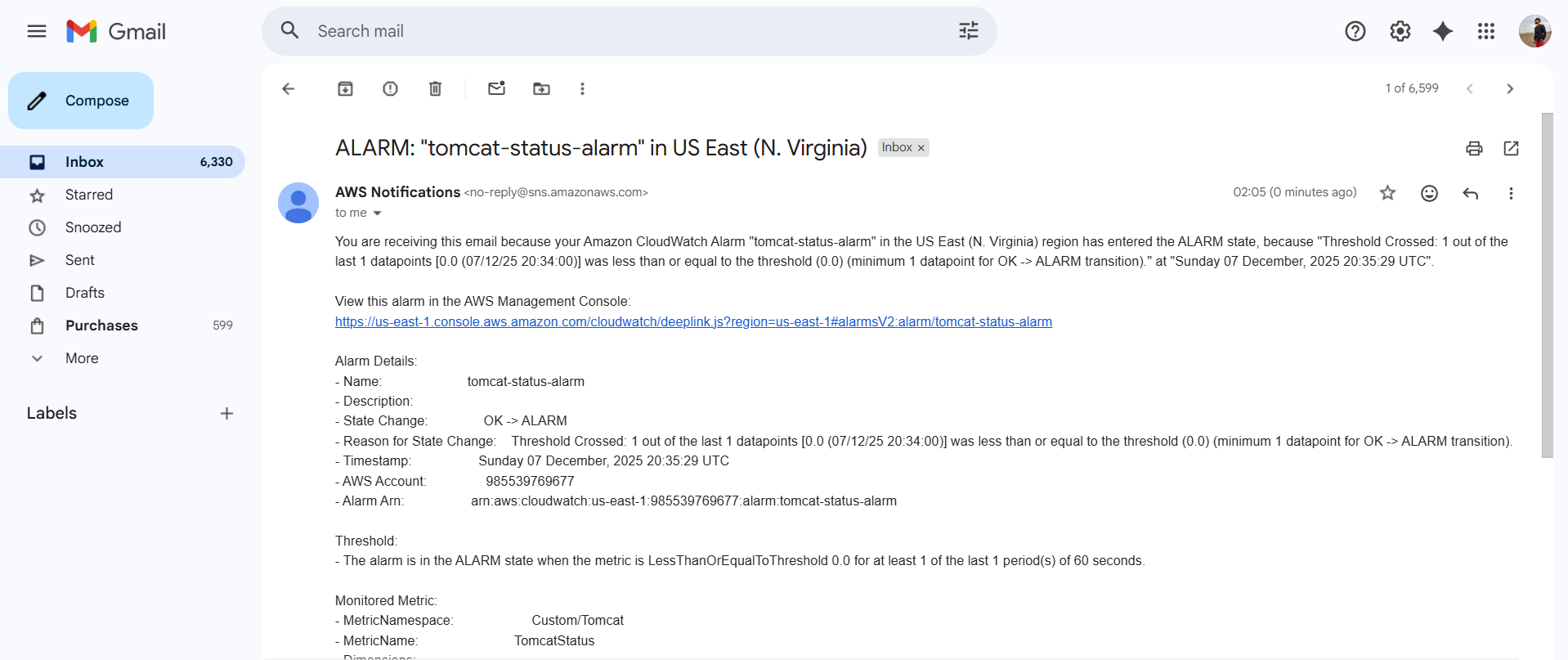
**Evidence: Alarm in ALARM state**

****

****

**Validation 3: SNS Email Received**

**Evidence: Email screenshot**

****

**Issues Faced :**

**None.**

**Conclusion :**

**Tomcat monitoring & alerts configured successfully.**

**6) Create Dashboard and monitor nginx service to send the alert if nginx is not running.**

**Task Title :**

**Monitor Nginx Service Using Custom CloudWatch Metrics**

**Objective :**

**Monitor Nginx service health and receive alerts when it stops.**

**Prerequisites :**

* **EC2 with Nginx installed**
* **AWS CLI**
* **CloudWatch write permissions**

**Step-by-Step Implementation (With Evidence)**

**Step 1: Create Nginx Status Script**

**sudo vi /opt/nginx-status.sh**

**Add:**

**#!/bin/bash**

**if systemctl is-active --quiet nginx; then**

**status=1 (Service-active)**

**else**

**status=0 (Service-inactive)**

**fi**

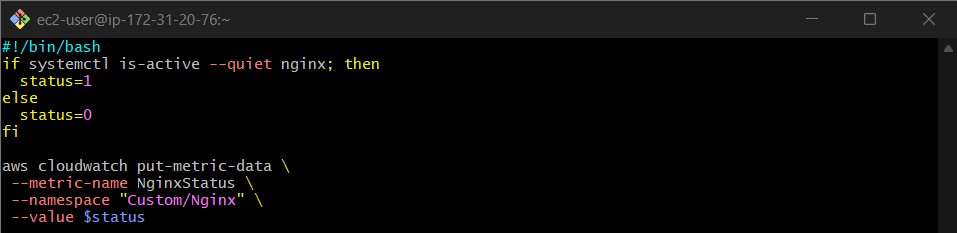
**aws cloudwatch put-metric-data \**

**--metric-name NginxStatus \**

**--namespace "Custom/Nginx" \**

**--value $status**

**Evidence: Script screenshot**

****

**Step 2: Schedule with Cron**

**\* \* \* \* \* /opt/nginx-status.sh**

**Evidence: Cron screenshot**

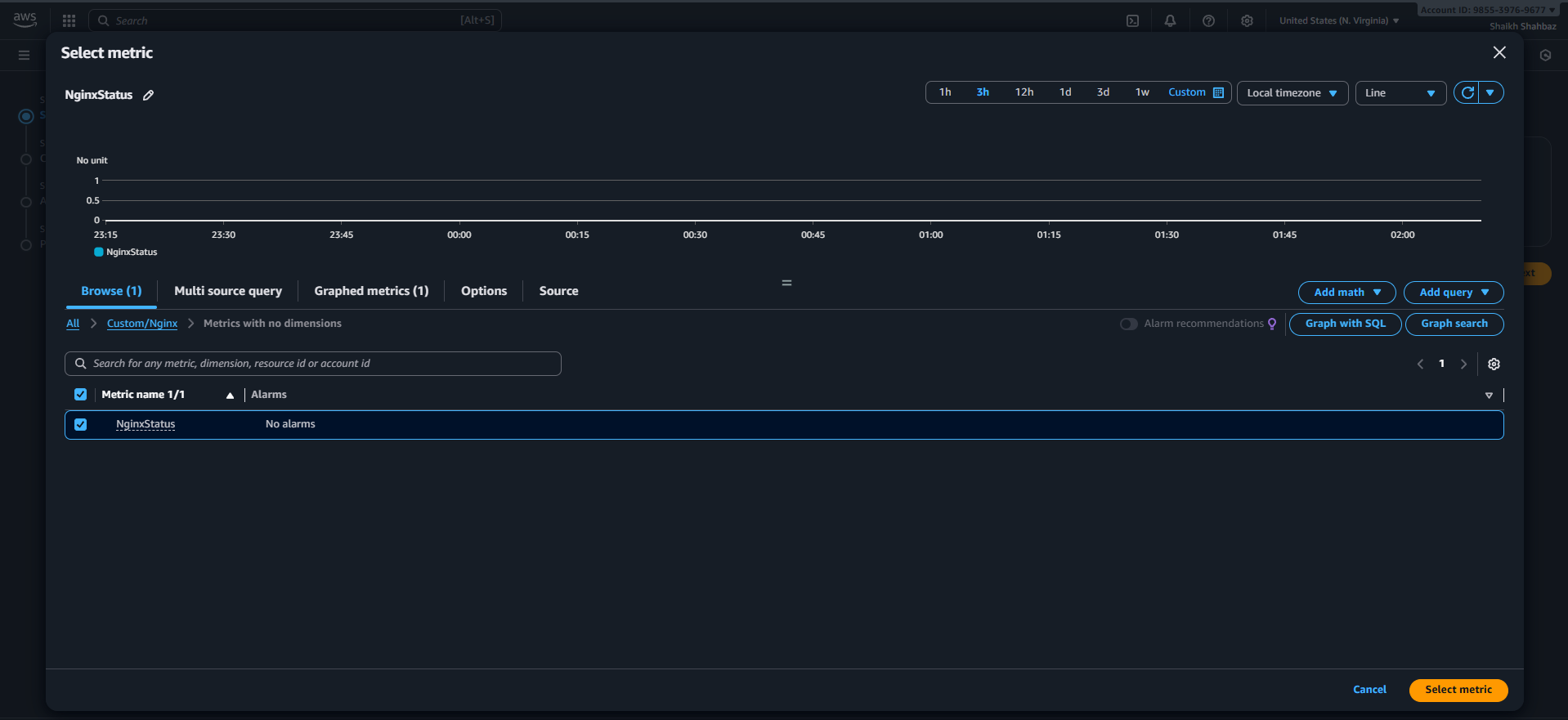
****

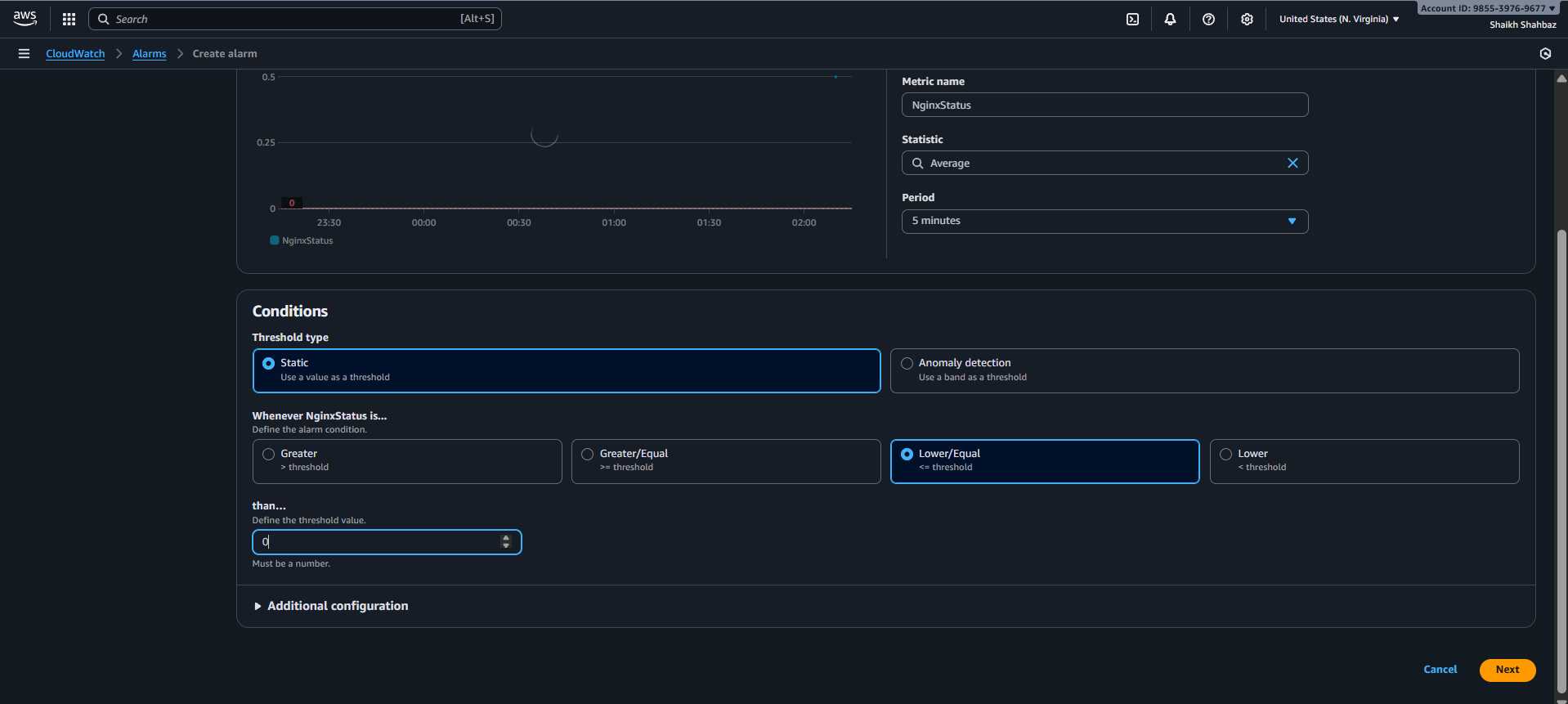
**Step 3: Create Alarm for Nginx**

**Alert when:**

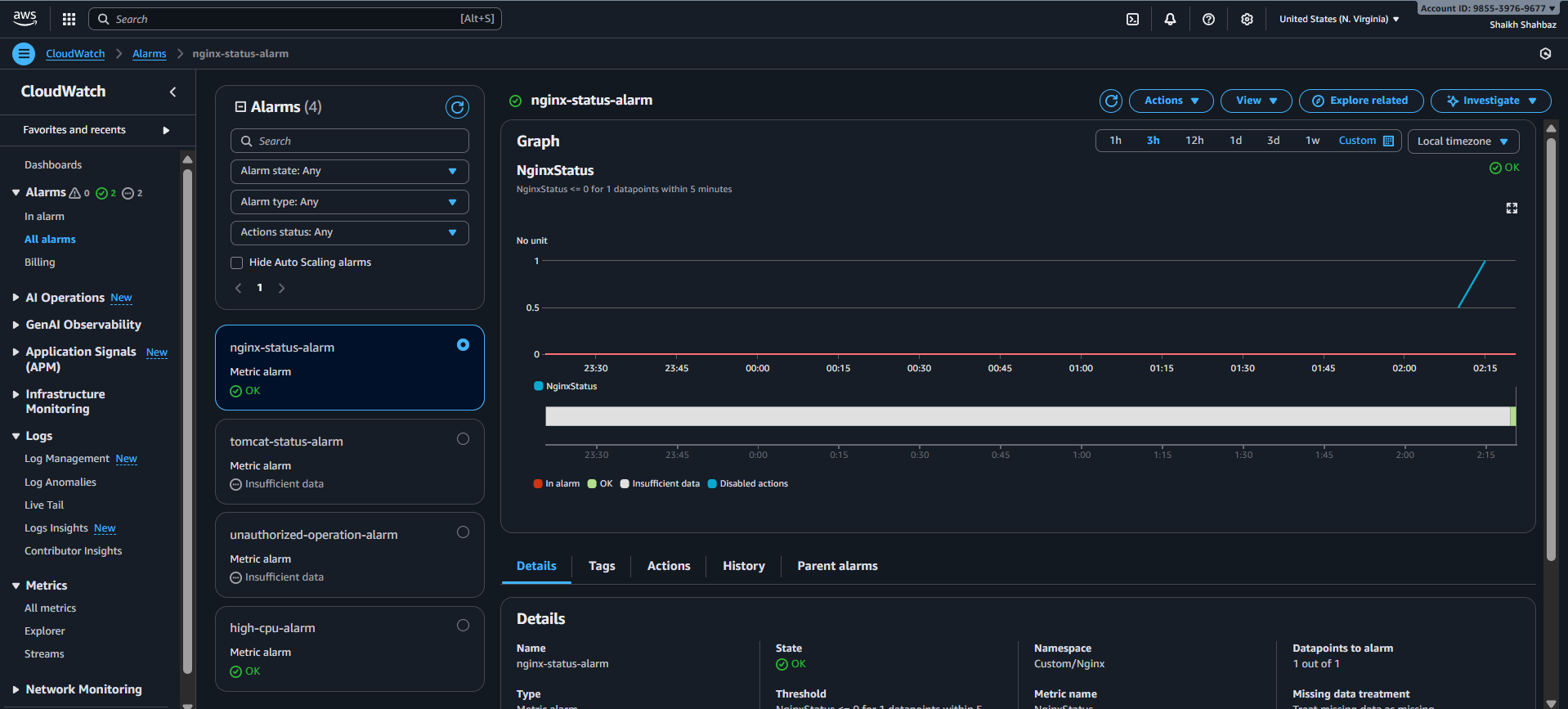
**NginxStatus = 0 (Service-inactive)**

**Evidence: Alarm screenshot**

****

****

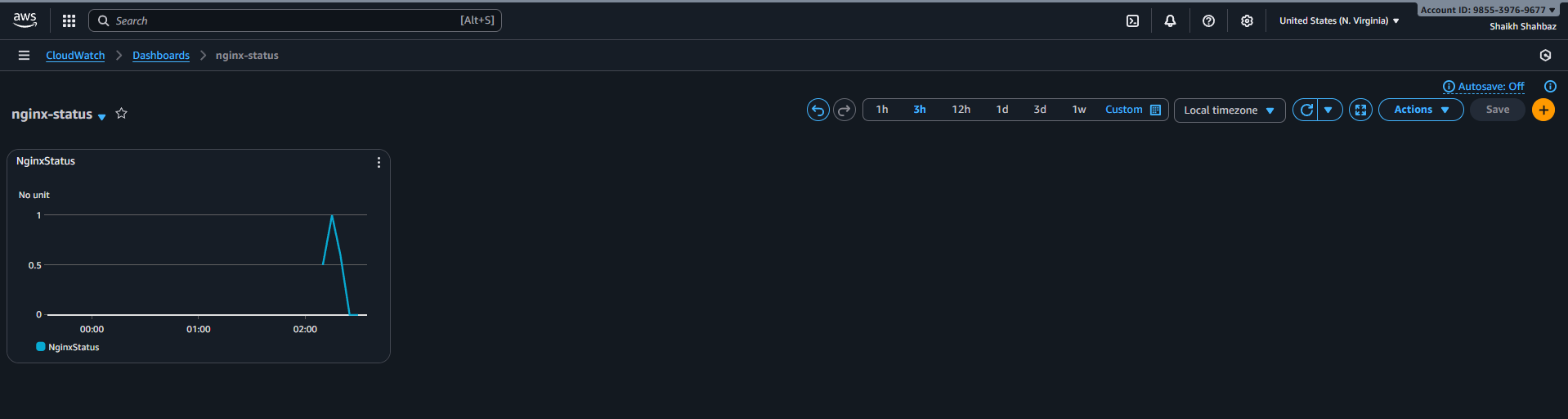
****

****

**Step 4: Add Dashboard Widget**

**Widget: Line  
Metric: Custom/Nginx → NginxStatus**

**Evidence: Dashboard screenshot**

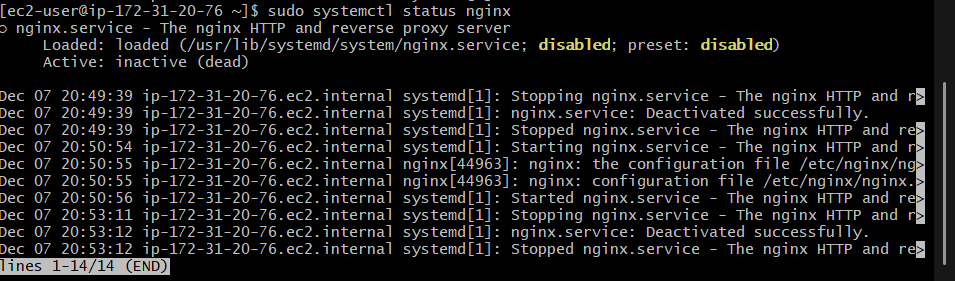
****

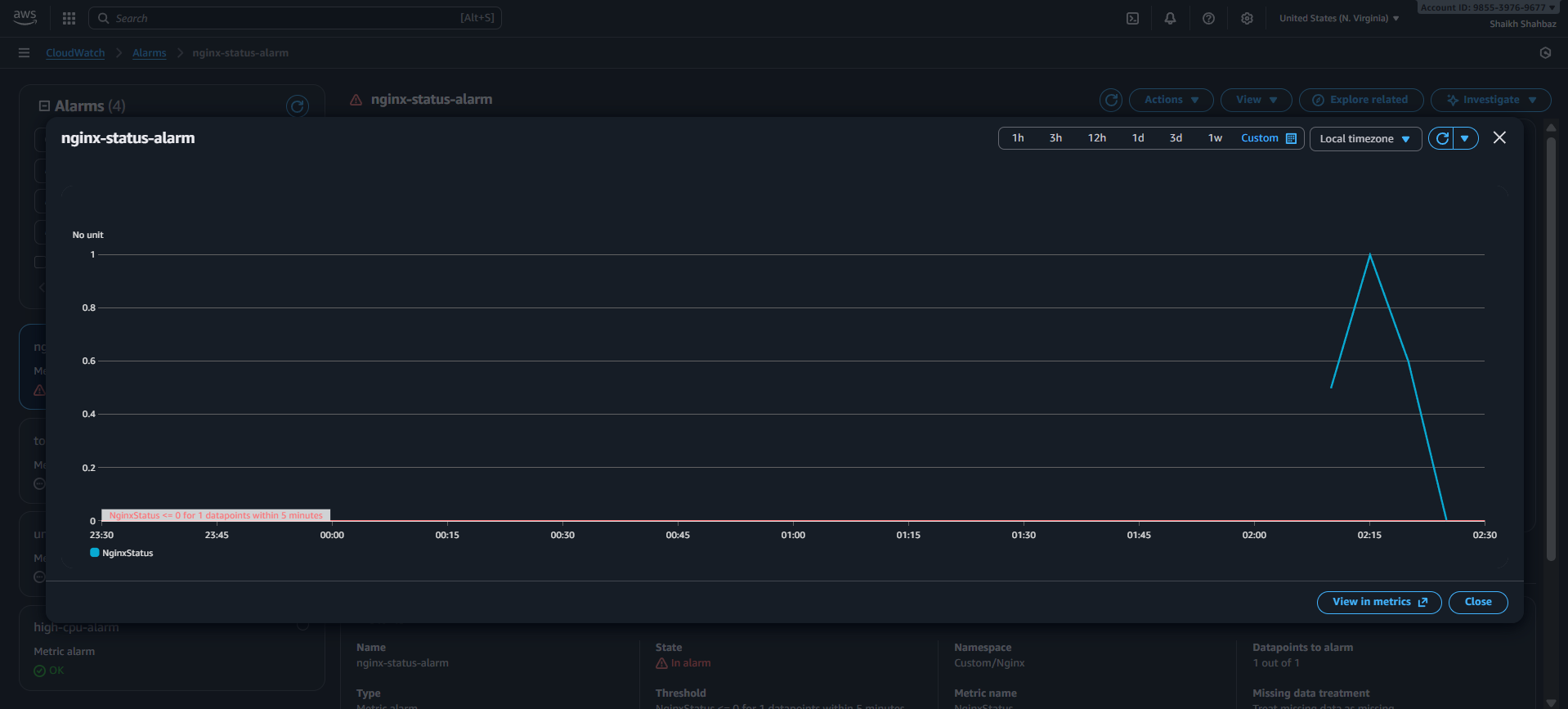
**Validation Steps (With Evidence)**

**Validation 1: Stop Nginx**

**sudo systemctl stop nginx**

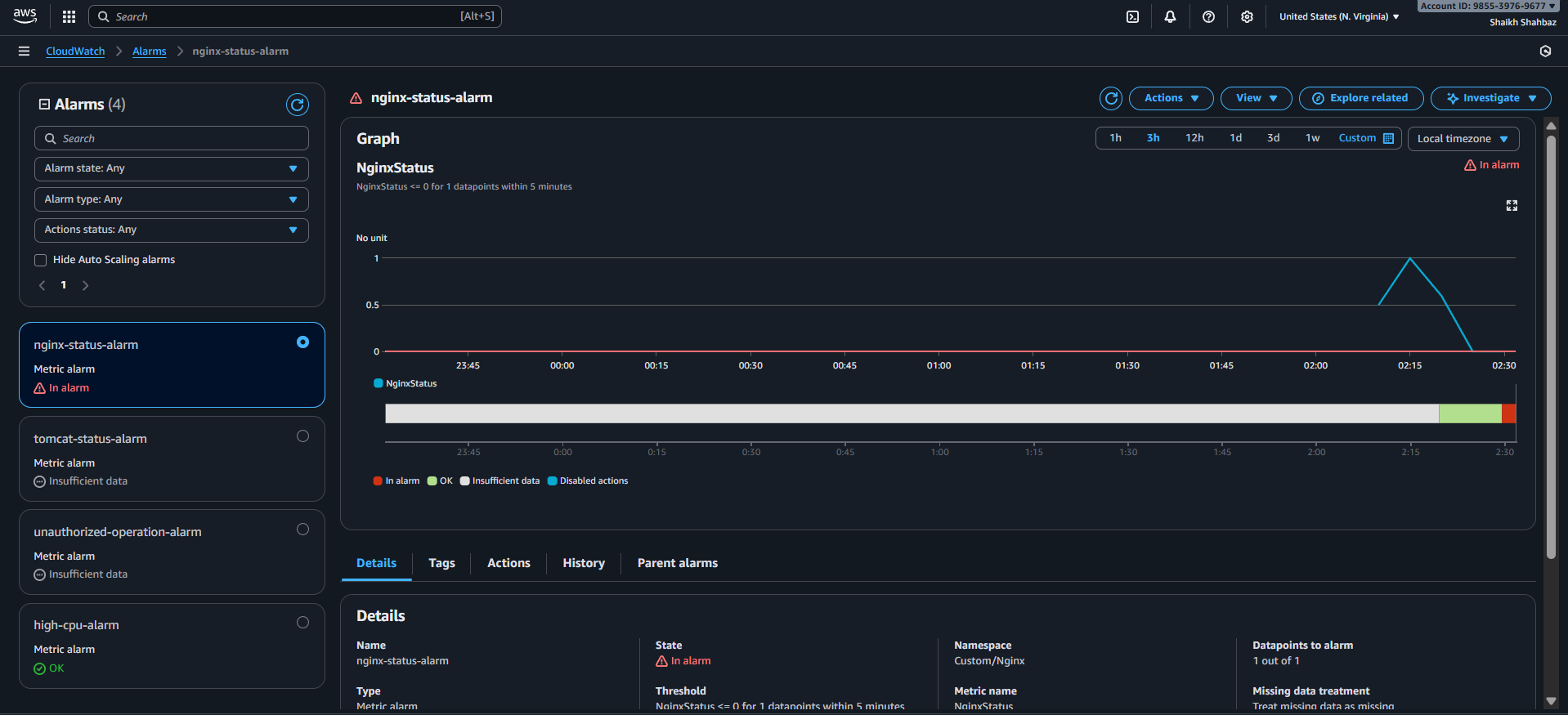
**Evidence: Service inactive screenshot**

****

****

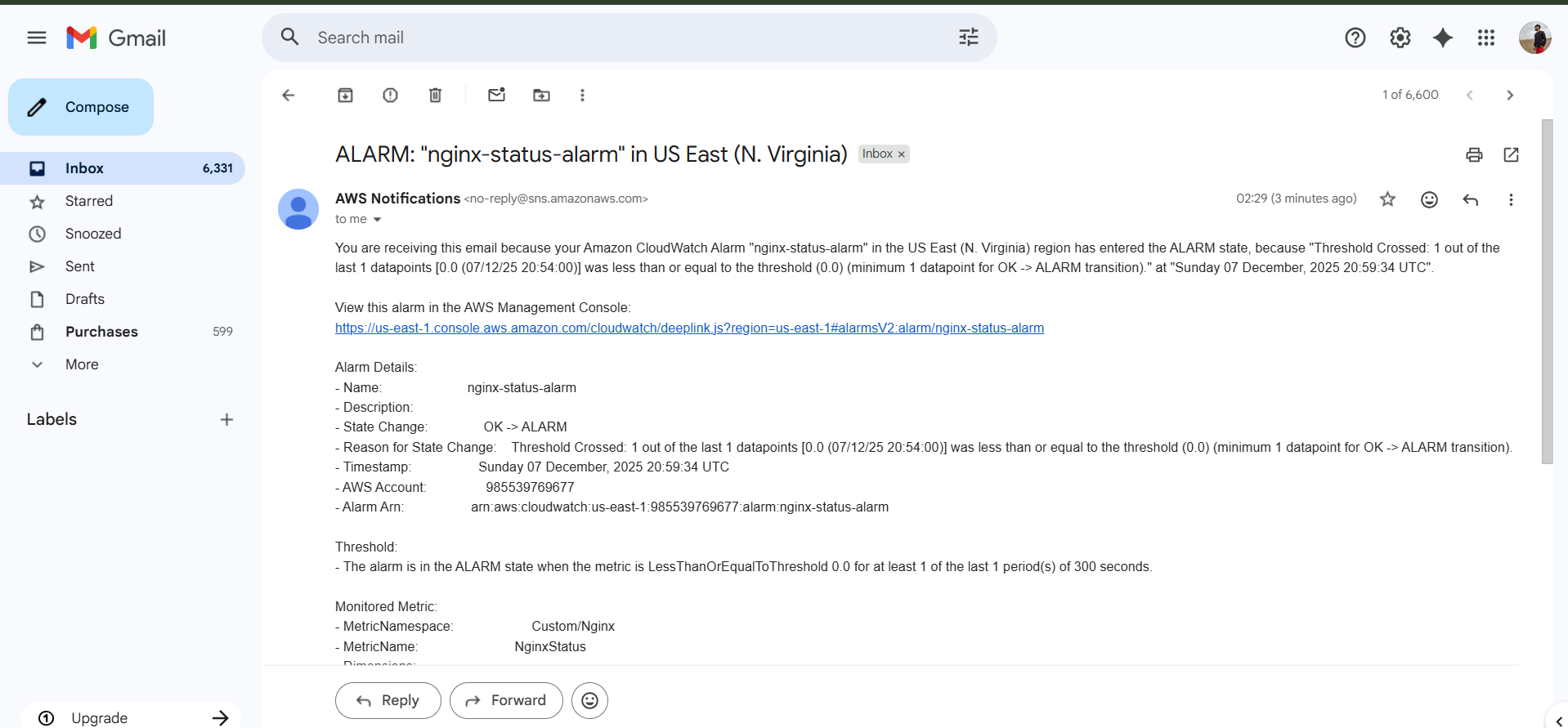
**Validation 2: Alarm Fired**

**Evidence: Alarm in ALARM state**

****

**Validation 3: SNS Email Alert**

**Evidence: Email screenshot**

****

**Issues Faced :**

**None.**

**Conclusion :**

**Nginx monitoring and alerting dashboard successfully created.**