Name: Vijaysingh Puwar

Project Overview

This project demonstrates practical skills in configuring and securing cloud infrastructure using Amazon Web Services (AWS), with a focus on launching and managing EC2 instances. It includes core concepts such as setting up Security Groups, Network Access Control Lists (NACLs), and monitoring cloud resources for proactive security. By completing a series of hands-on tasks, I gained experience in configuring access permissions, securing virtual machines, and applying essential security measures to ensure safe communication across a cloud network.

Additionally, I extended the project by implementing **Amazon CloudWatch Monitoring** and creating a **custom CPU Utilization Alarm** using **SNS notifications**, thereby integrating real-time observability into the EC2 instance's performance. This enhancement reinforces proactive cloud security monitoring and showcases operational awareness in identifying abnormal system behaviors.

OBJECTIVE

To configure and secure an Amazon EC2 instance using AWS cloud-native tools, including Security Groups and Network ACLs, and to implement performance monitoring using Amazon CloudWatch. The project aims to build foundational knowledge in cloud security and real-time alerting mechanisms for proactive incident response.

STEP BY STEP

Step1: AWS login

- **Action:** Signed in to the AWS Management Console.
- **Purpose:** To access EC2 services and manage cloud resources.
- **Note:** Used either the AWS Academy Classroom login or standard AWS account.

Once the AWS account is created, we will arrive on the AWS home page which look like as shown in Fig 1.

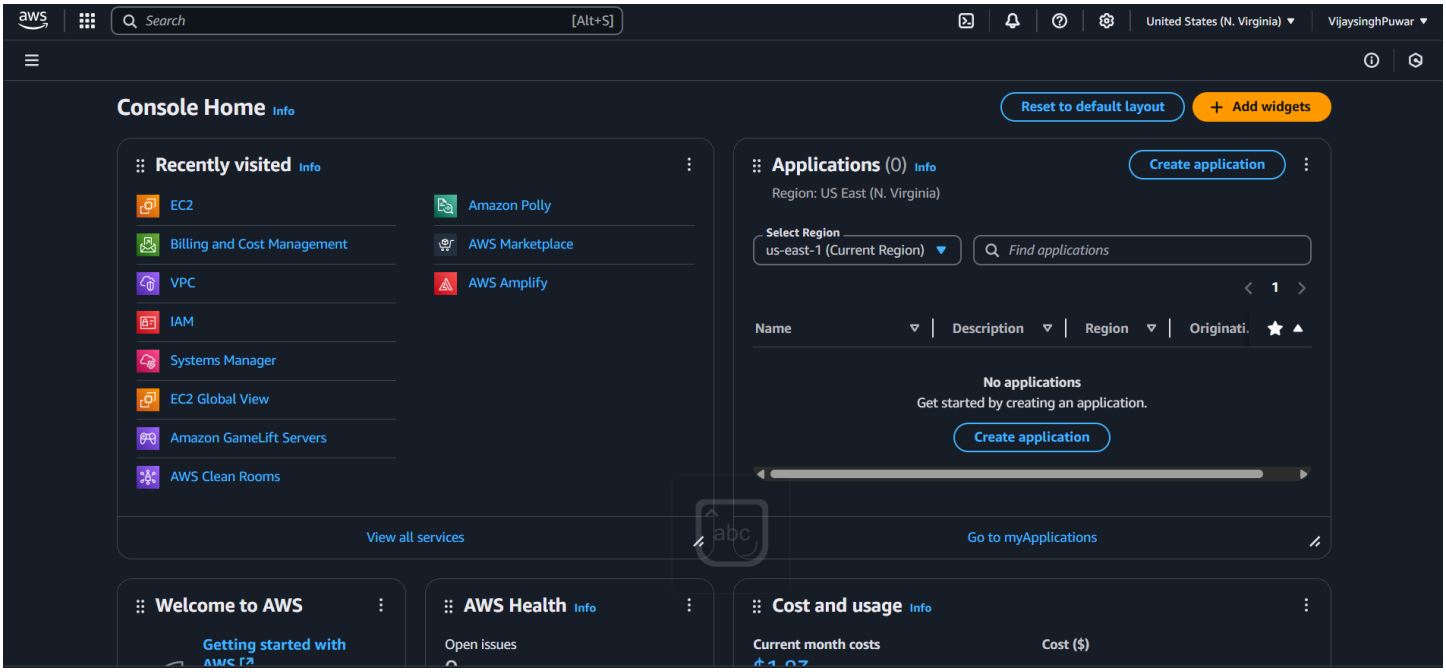


Figure 1: AWS login Page

Step2: Launch EC2 Instance

Action: Navigated to the EC2 Dashboard and selected **Launch Instance**.

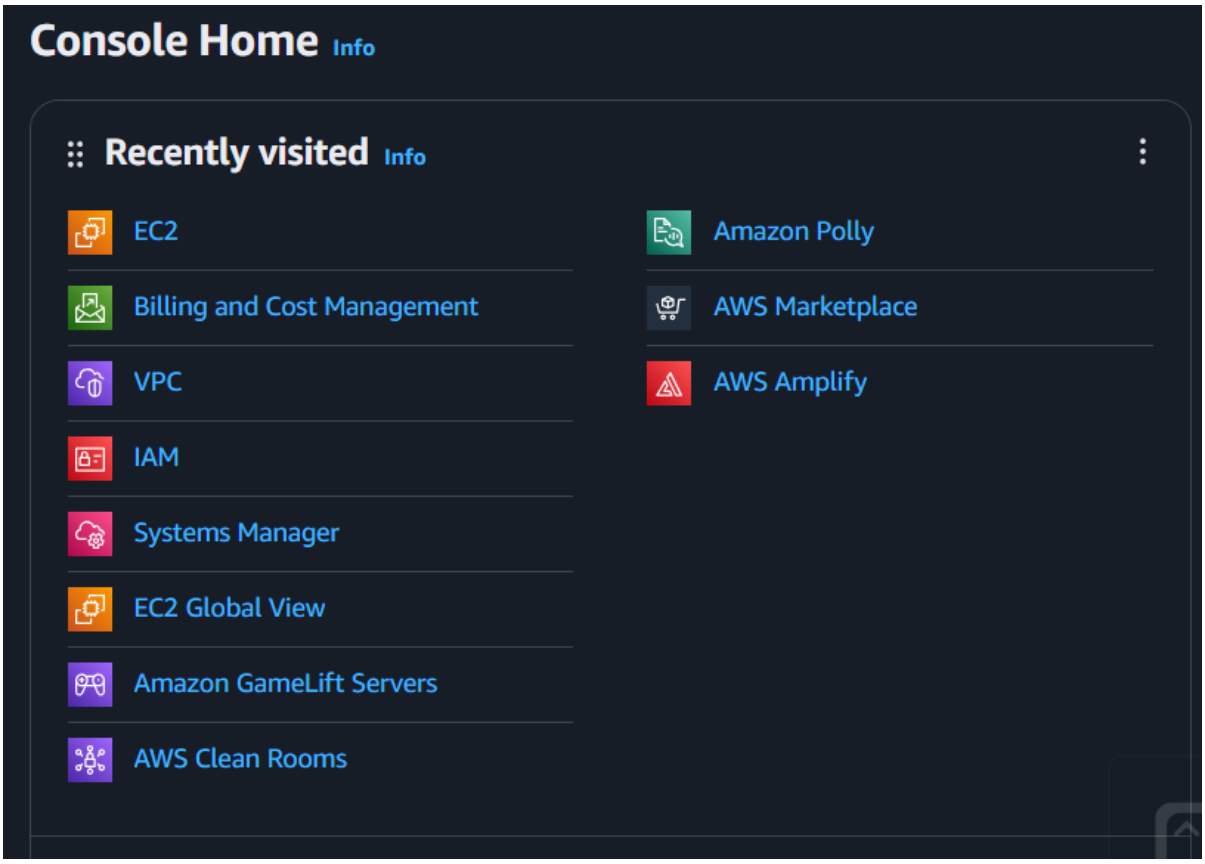


Figure 2:Console Home

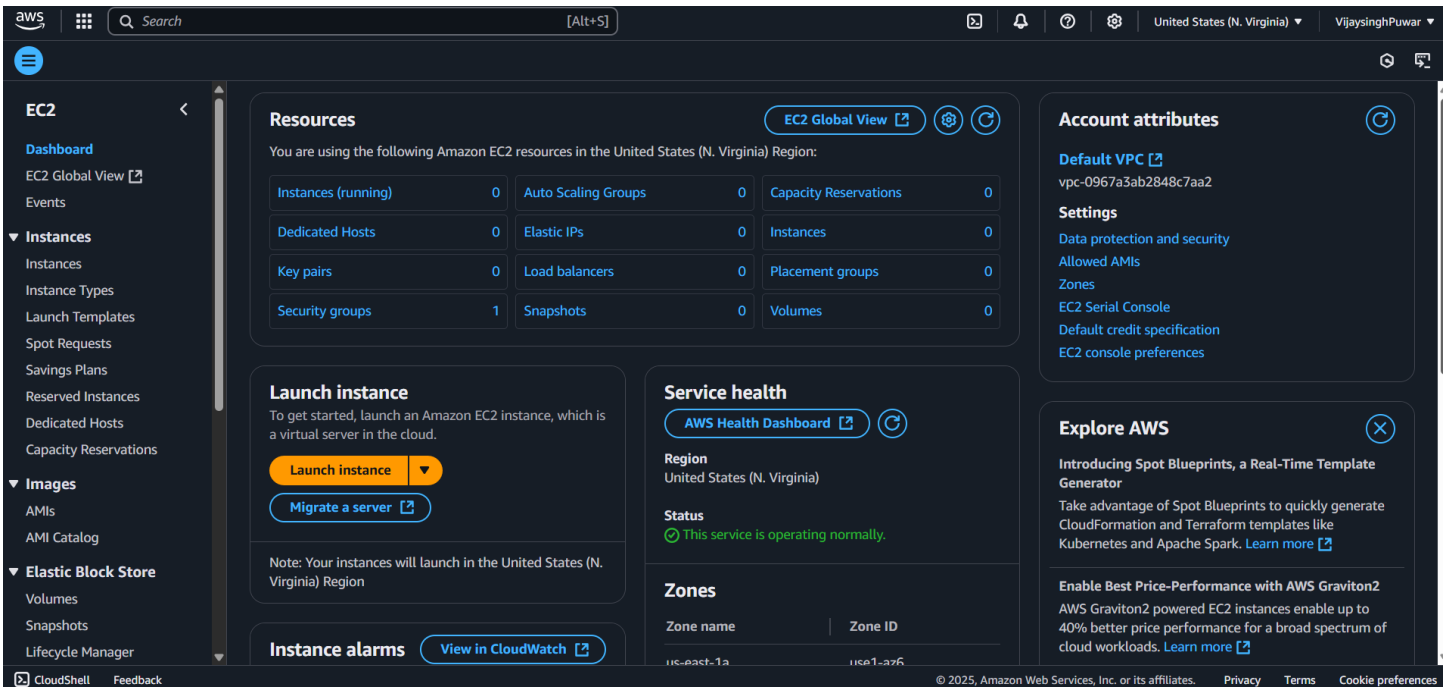


Figure 3: EC2 instance inside

After clicking on the EC2 instance in order to launch it click on the Launch Instance button in the Launch instance section in the EC2 instance page.

Configuration:

- **AMI:** Amazon Linux 2 (Free Tier Eligible)

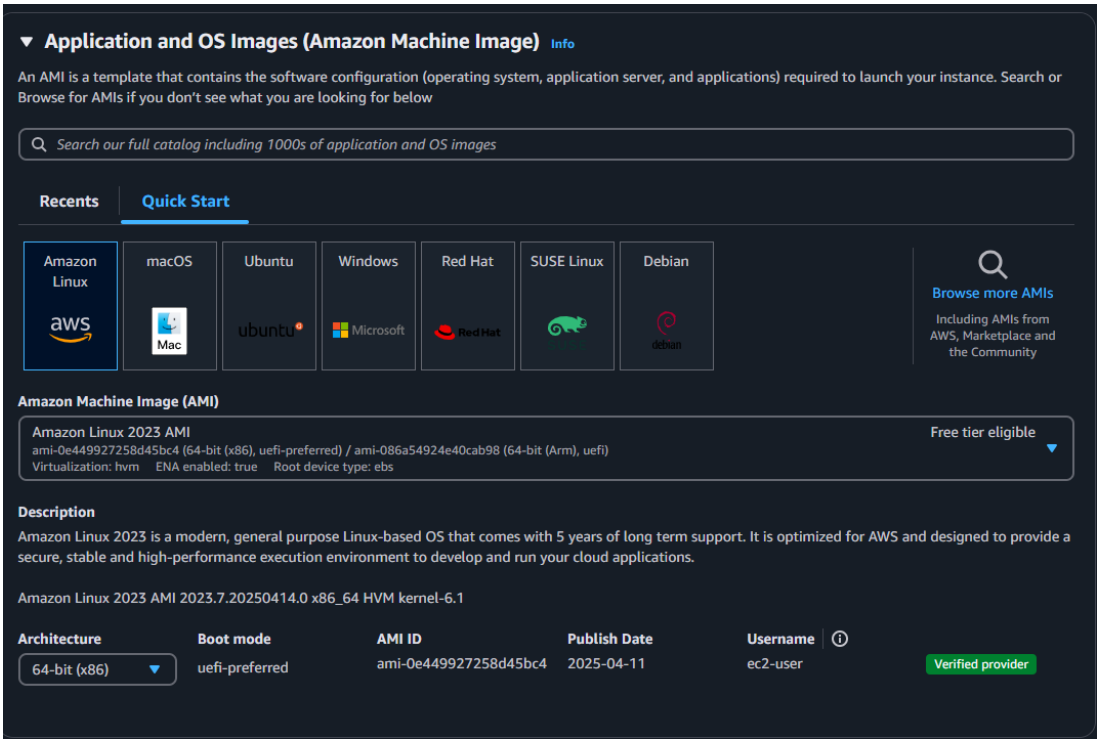


Figure 4: Amazon Linux

- **Instance Type:** t2.micro

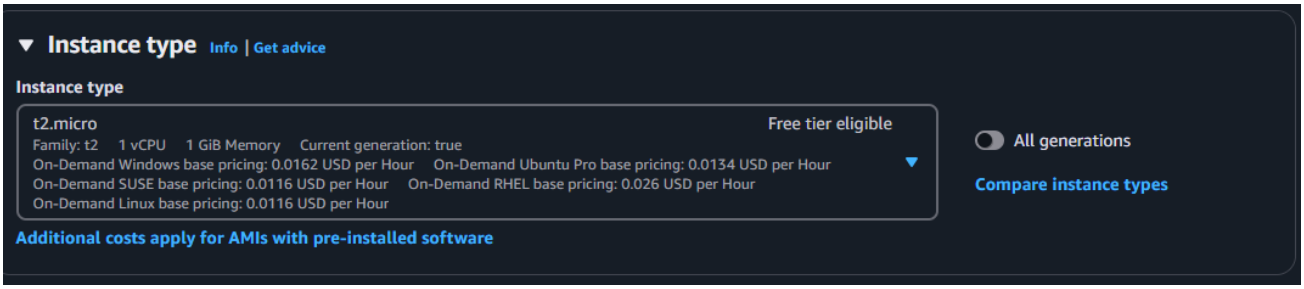


Figure 5: Instance Type

- **Storage & Tags:** Default settings

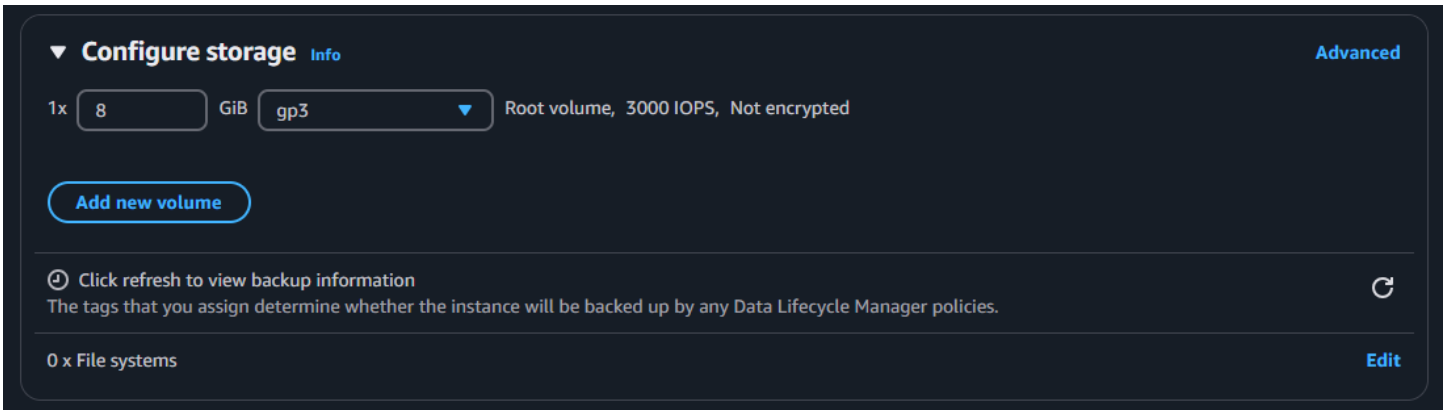


Figure 6: Stprage

- **Security Group/Network Settings:** Allowed SSH (Port 22) from anywhere (0.0.0.0/0)

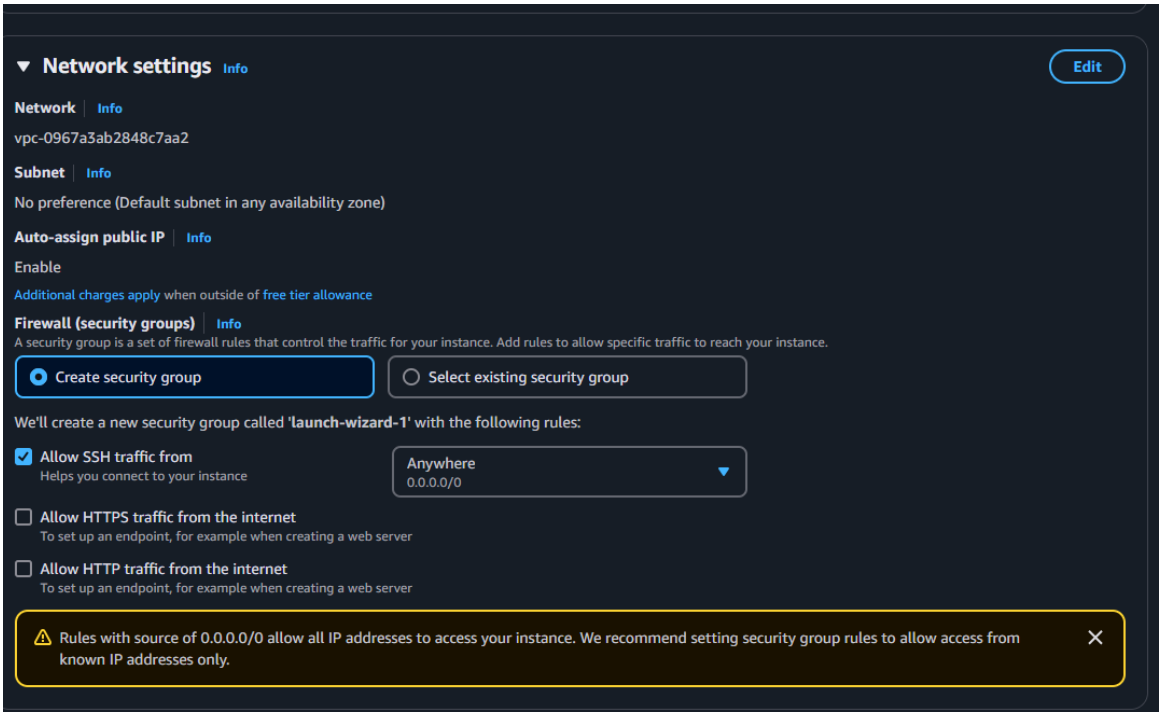


Figure 7: Network Settings

Here the Security groups provide a secure perimeter around each EC2 instance. Each instance can be assigned up to five security groups. A security group blocks all traffic and will only allow traffic that is explicitly allowed. As you can see, EC2 assumes you will want to SSH into your Linux EC2 instance, and it automatically suggests a rule that will allow SSH from any IP address (0.0.0.0/0) into your EC2 instance through port 22.

Purpose: Deploy a virtual machine in the cloud.

I created a new key pair named **aws-project8-key** to securely connect to the EC2 instance.

I selected the **RSA** key type and chose the **.ppk** format because it is compatible with **PuTTY**, which I will use later to connect.

The private key was downloaded and saved securely, as it is required for the SSH connection.

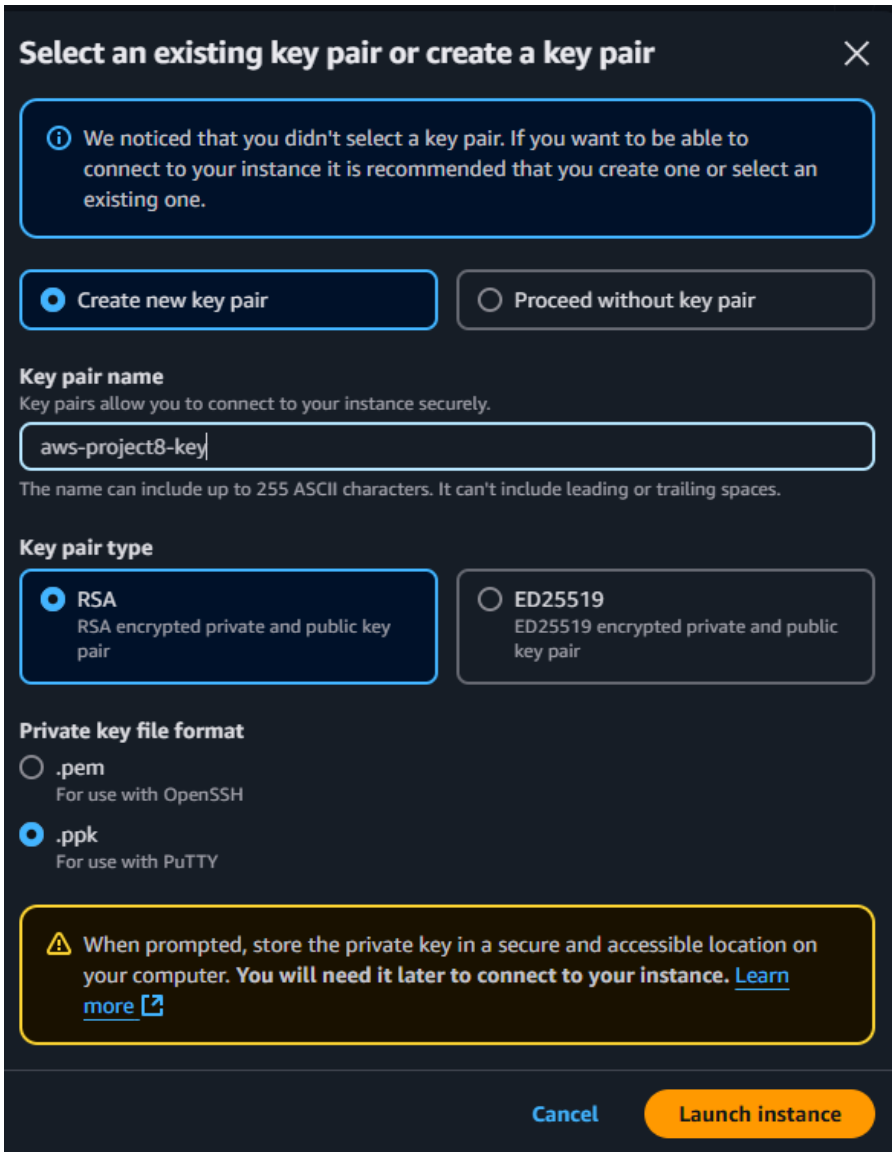


Figure 8: Launch instance

Step3**: EC2 Instance Overview**

After launching the instance, I reached the **Instance Summary** page, which shows all key details about my EC2 instance. Below is a summary of the important components:

- **Instance ID:** i-0c1b1ce977937c0fe
  This is the unique identifier for the EC2 instance.
- **Instance State:** Running
  Confirms that the instance is active and ready for use.

- **Public IPv4 Address:** 34.227.152.67
  This IP address is used to connect to the instance from the internet.
- **Private IPv4 Address:** 172.31.89.225
  Used for internal communication within the VPC (Virtual Private Cloud).
- **Instance Type:** t2.micro
  A free-tier eligible instance type with 1 vCPU and 1 GB RAM.
- **VPC ID:** vpc-0967a3ab2848c7aa2
  Identifies the virtual private cloud in which the instance is deployed.
- **Subnet ID:** subnet-02936abe8bfa8002e
  Shows the specific subnet within the VPC where the instance is located.
- **Public DNS:** ec2-34-227-152-67.compute-1.amazonaws.com
  This DNS address can also be used for SSH access.
- **AMI ID:** ami-0e449927258d45bc4
  This refers to the Amazon Machine Image used to create the instance.
- **Platform:** Linux/UNIX
  Confirms the operating system installed on the instance.
- **Monitoring:** Disabled
  Monitoring is not active by default, but it can be enabled for better visibility.

This information confirms that the instance is correctly set up and accessible. I will use the public IP or DNS name to connect to this instance using PuTTY in the next step.

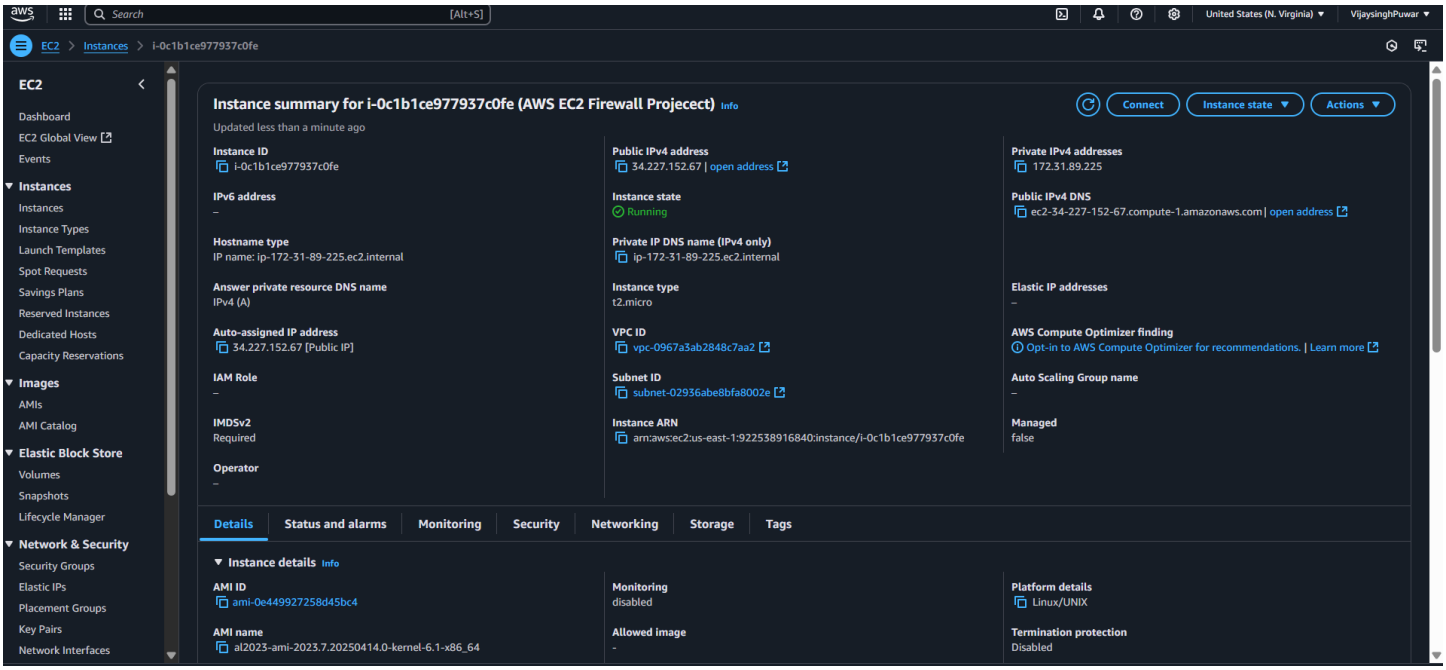


Figure 9: Instance Detail

4. Without changing the rule type, what are three other protocols you could choose for rules in your security group?

Apart from SSH (which uses TCP on port 22), three other commonly used protocols that can be selected in a security group are:

- **HTTP (Port 80)** – Allows web traffic from browsers.
- **HTTPS (Port 443)** – Allows secure web traffic.
- **RDP (Port 3389)** – Used for remote desktop connections, typically for Windows instances.

These protocols can be added depending on the type of access needed for the EC2 instance.

5. What happens when you change the source to My IP?

When you change the source to **My IP**, AWS automatically detects your current public IP address and configures the rule to only allow access from that specific IP.

This enhances security by restricting access so that **only your computer** can connect to the instance. No other IP addresses will be able to establish a connection through that rule.

Step4: Connect to EC2 Instance

In this step, I prepared to connect to my EC2 instance.

From the **"Connect to instance"** page, I chose the connection method **"EC2 Instance Connect"**. This option allows me to connect to the instance directly from the browser without needing to use any additional tools like PuTTY.

- The **instance ID** is: i-0c1b1ce977937c0fe
- The **public IPv4 address** is: 34.227.152.67
- The **default username** is: ec2-user, which is automatically provided because I used an Amazon Linux 2 AMI.

By clicking the **"Connect"** button, I was able to open a terminal session in the browser and access the instance.

Screenshot Reference:

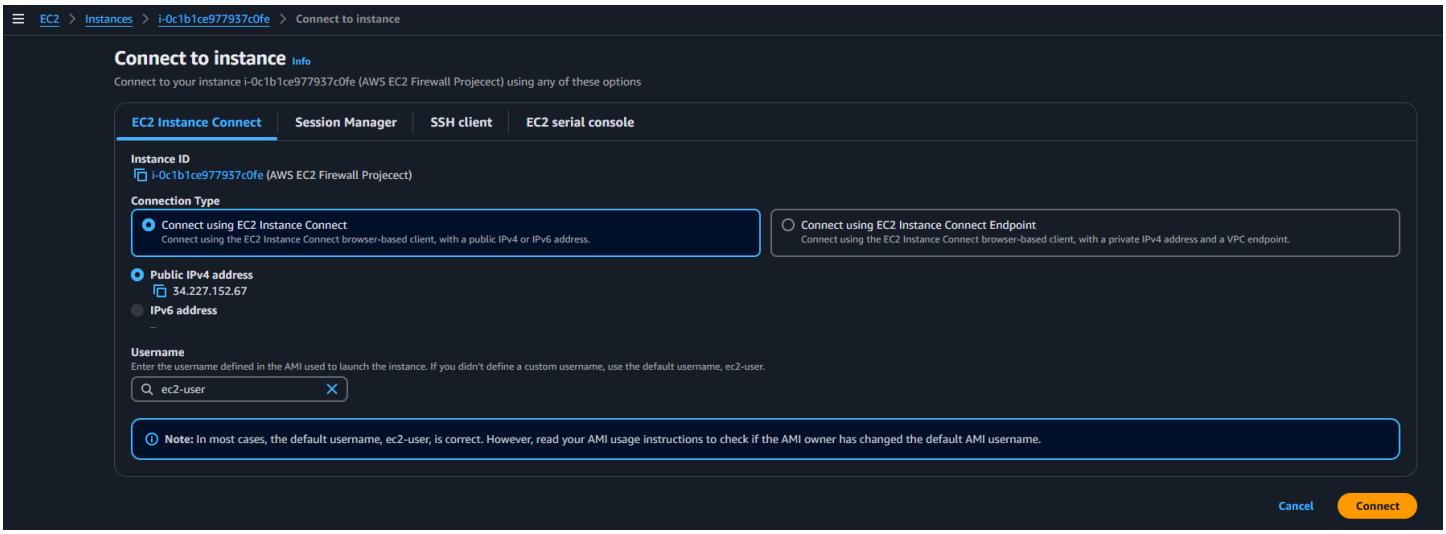


Figure 10: Instance Connect

Step5: Verifying Outbound Connectivity by Pinging 8.8.8.8

After connecting to the EC2 instance using **EC2 Instance Connect** through the browser, I tested the network connectivity by running the following command in the terminal:

ping 8.8.8.8

This command sends ICMP packets to Google's DNS server. As seen in the terminal output, the ping was successful, indicating that:

- The EC2 instance has outbound internet access.
- The **Security Group allows outbound traffic by default**.
- The EC2 instance can reach external servers.

This confirms that the instance is properly configured to communicate with the internet.

Screenshot Reference:

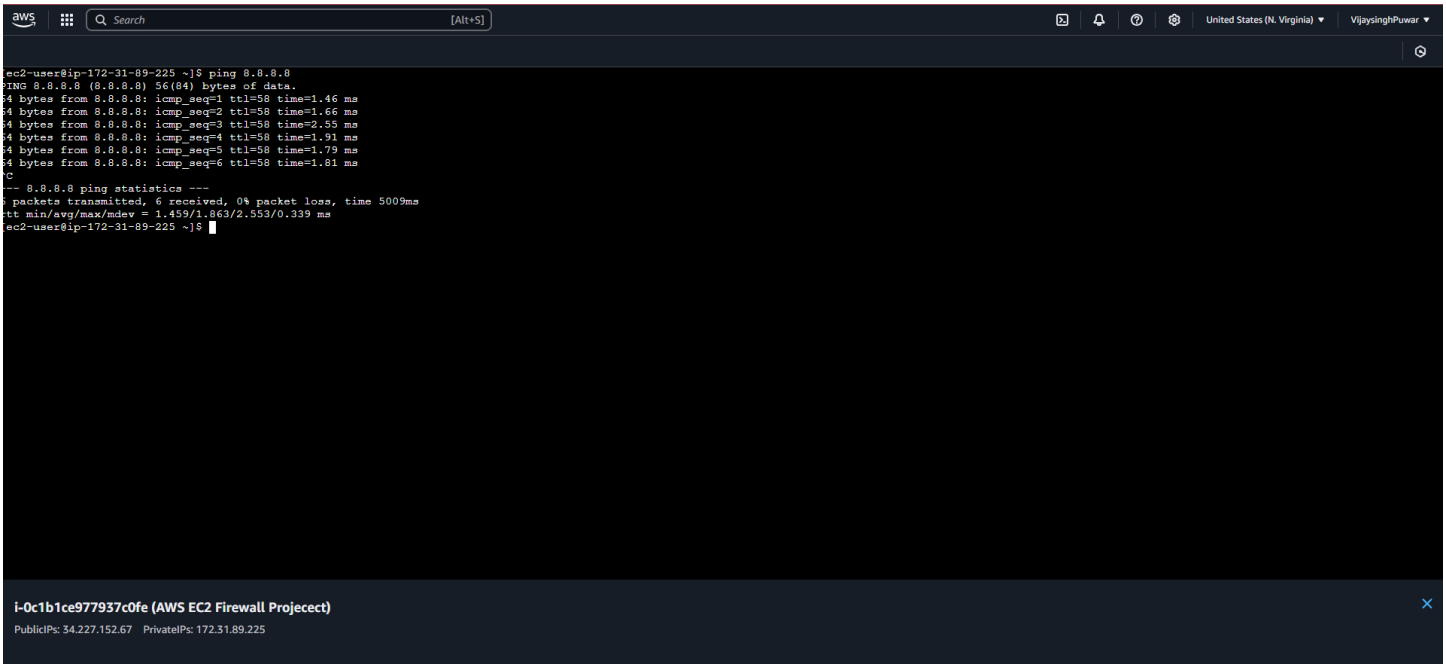


Figure 11: Connect using EC2 Instance Connect

Additional Note on Connection Method

In this project, I used the **“Connect using EC2 Instance Connect”** option. This method allows browser-based SSH access to the EC2 instance using its **public IPv4 address**, without needing to install any SSH client software like PuTTY.

This approach is simple, secure, and recommended for quick testing or configuration from the AWS Management Console.

Step5: Viewing SSH Client Instructions for Connecting to the EC2 Instance

This screen shows the **SSH client tab** within the **"Connect to instance"** section of the EC2 dashboard. It provides the exact steps and command to connect securely to the instance from a local terminal or command-line interface using an SSH key pair.

Explanation of Each Section:

1. Instance ID

- This is the unique identifier of the EC2 instance: i-0c1b1ce977937c0fe
- It confirms the specific virtual machine you're about to connect to.

2. Instructions to Connect

Step 1: Open an SSH Client

- You must use an SSH tool like **PuTTY (Windows)** or **Terminal (Mac/Linux)**.

Step 2: Locate Your Private Key File

- The key used during instance creation is named: aws-project8-key.pem
- This file is essential for authentication and should be stored safely on your computer.

Step 3: Set File Permissions (Linux/macOS only)

- The command:
- chmod 400 aws-project8-key.pem

is used to ensure that the private key file is only readable by you.

- This step is a security requirement before using the .pem key on Linux or Mac.

Step 4: Connect Using SSH

- The final command to initiate the connection is:
- ssh -i "aws-project8-key.pem" ec2-user@ec2-34-227-152-67.compute-1.amazonaws.com
    - -i "aws-project8-key.pem": tells SSH to use this key for authentication.
    - ec2-user@...: connects using the default Amazon Linux 2 username.
    - The DNS address maps to the instance's **public IPv4 address**.

Purpose of This Step

This method provides **secure command-line access** to the EC2 instance. It is especially useful if:

- You prefer working in a terminal or command prompt,
- You need more control than EC2 Instance Connect offers,
- Or you're automating tasks/scripts from your local machine.

Screenshot Reference:

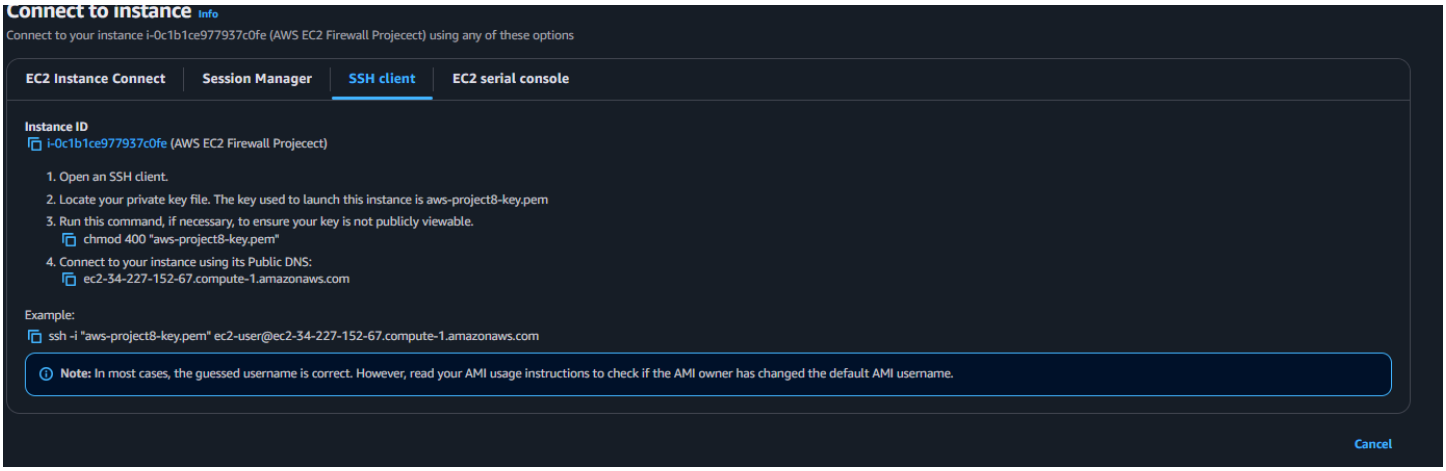


Figure 12: SSH Client instruction.

Step5: Finalizing Connection in PuTTY

This screen shows the final PuTTY configuration under the **Session** category, where the SSH session is prepared to connect to the EC2 instance using the public DNS address and the previously configured .ppk key file.

Explanation of Fields and Settings:

1. Host Name (or IP Address):

- Entered value:
- ec2-34-227-152-67.compute-1.amazonaws.com
- This is the **Public DNS** of the EC2 instance.
- PuTTY uses this address to connect to the instance over the internet.

2. Port:

- Set to **22**, the default port for SSH (Secure Shell) protocol.
- This is required for a secure remote connection to the Linux-based EC2 instance.

3. Connection Type:

- **SSH** is selected.
- This ensures PuTTY establishes a secure, encrypted connection using the SSH protocol.

Optional: Saved Session

- You can type a session name (e.g., EC2-AWS) under **Saved Sessions** and click **Save**.

- This saves your current settings, including hostname and key file path, so you don't have to reconfigure each time.

Final Action:

- Click the **Open** button at the bottom.
- A new terminal window will appear.
- You'll be prompted to **log in as**:
    - Type: ec2-user (default username for Amazon Linux 2)
- Upon successful authentication using your .ppk key, you'll gain command-line access to your instance.

What This Step Achieves:

- This connects your **local machine to your EC2 instance** using a secure, key-based SSH session.
- From here, you can run commands, configure security settings, or test network tools like ping, curl, or traceroute.

Screenshot Reference:

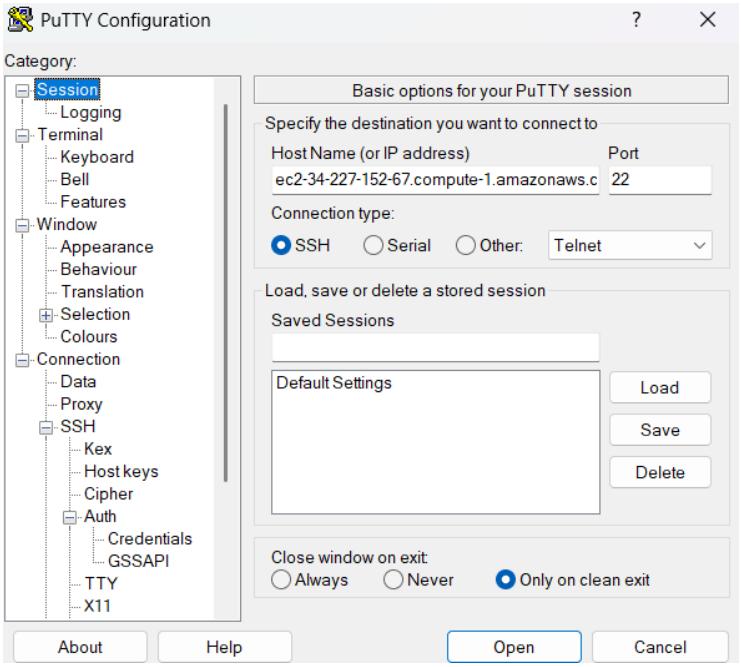


Figure 13: Putty Session

Step6: PuTTY Security Alert

When connecting to the EC2 instance for the first time, PuTTY displays this **Security Alert** because the host key is not yet stored in your local system.

- The message confirms the **server's identity** using a unique SSH key fingerprint.
- This is a normal and expected prompt for first-time connections.

What I did:

I clicked **"Accept"** to trust the server and save its key to PuTTY's cache. This prevents the alert from showing up again in future sessions with the same server.

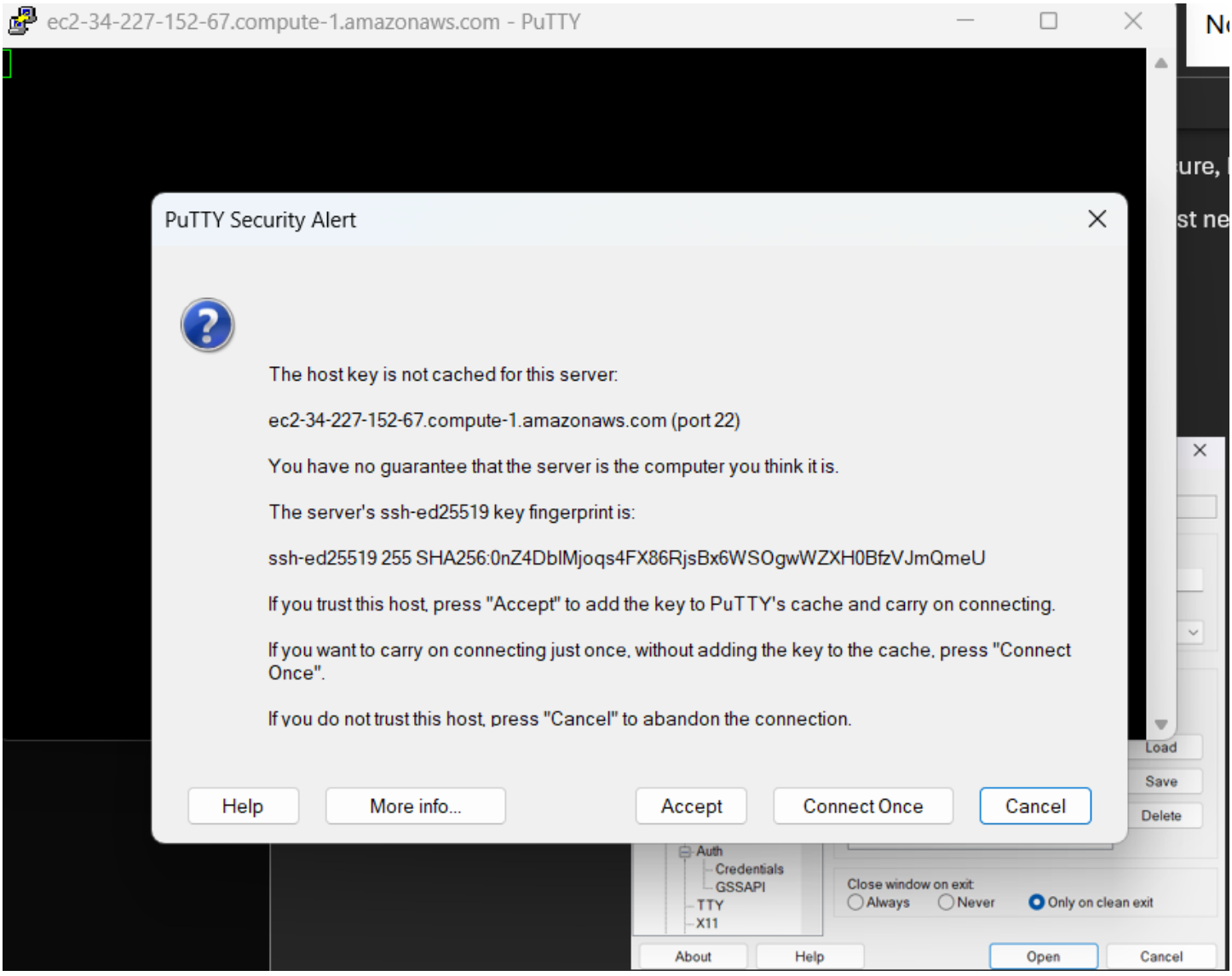


Figure 14: PuTTY Security Alert

Step7: Resolving PuTTY Authentication Error and Establishing Successful Connection

During my initial attempt to connect to the EC2 instance using PuTTY, I encountered an authentication error. The message displayed was:

PuTTY Fatal Error:

No supported authentication methods available (server sent: publickey,gssapi-keyex,gssapi-with-mic)

This error indicated that PuTTY could not find a valid authentication method. Specifically, it failed to locate or apply the required private key for secure login, which is the expected method for EC2 instances.

Root Cause of the Error:

The error was caused due to one or more of the following:

- The private key file (.ppk) was not loaded correctly in PuTTY.
- The correct username (ec2-user) was not specified.
- PuTTY defaulted to unsupported methods (GSSAPI or keyboard-interactive) because it couldn't find a valid key.

Steps Taken to Fix the Issue:

1. I reopened PuTTY and went to:
    - Connection → SSH → Auth
    - Under "Private key file for authentication", I browsed and selected my key: aws-project8-key.ppk
2. To ensure that the correct login username was used, I did one of the following:
    - Entered the full host format in the **Session → Host Name** field: ec2-user@ec2-34-227-152-67.compute-1.amazonaws.com
    - Or, under Connection → Data, specified the **Auto-login username** as ec2-user.

Once I confirmed the key and username were configured properly, I saved the session and attempted to connect again.

Result: Successful SSH Connection

On the second attempt, the connection was successful. The PuTTY terminal opened and displayed the **Amazon Linux 2023** banner, confirming that the login via public key was accepted. This confirmed that

the .ppk file was correctly recognized and used for authentication, and the EC2 instance was now accessible for further commands and configurations.

Screenshots and Placement in Report:

- **Screenshot 1: PuTTY Authentication Error**

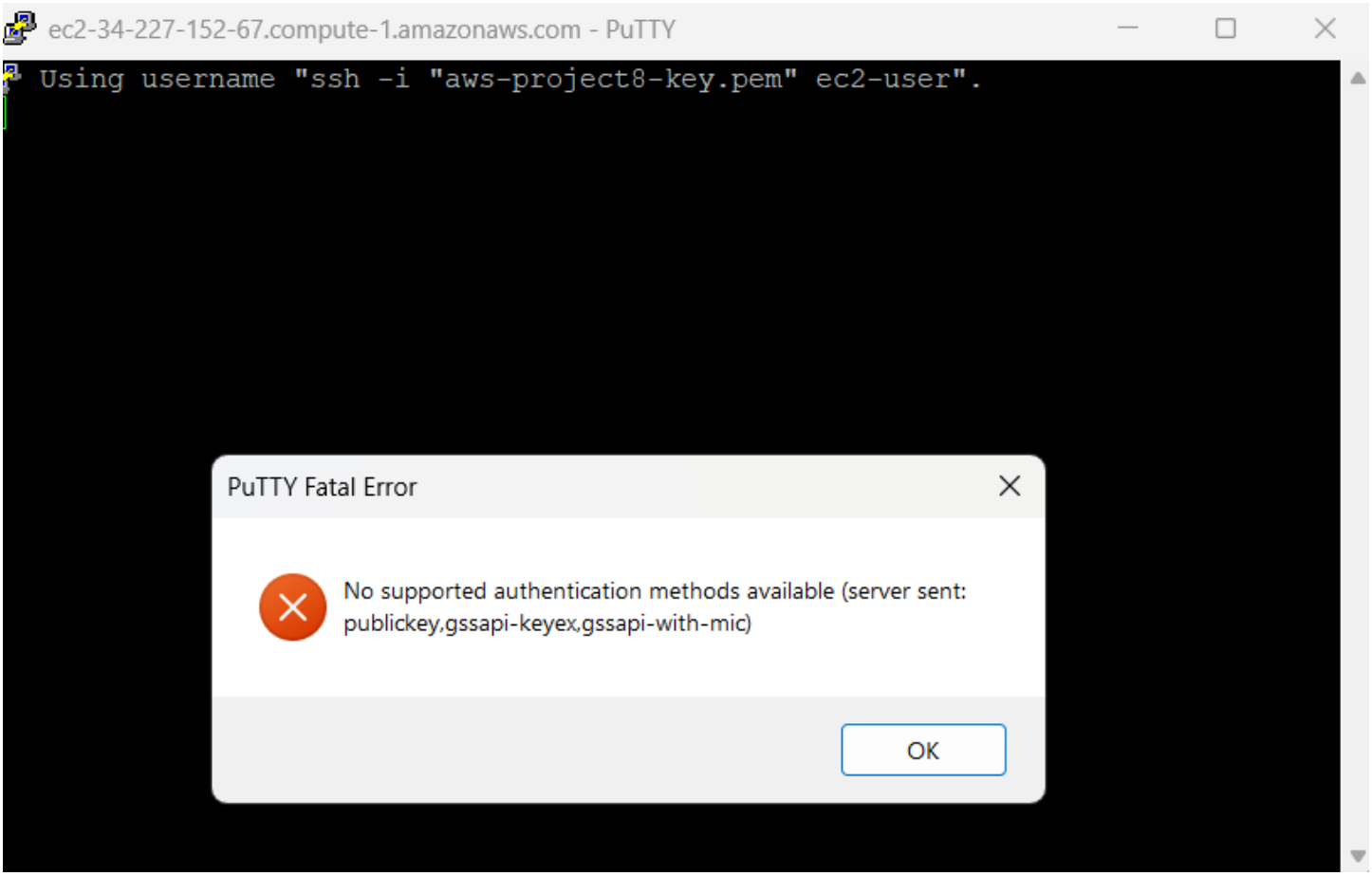


Figure 15: putty-auth-error

  - **Placement:** Immediately after the section titled "Root Cause of the Error"
  - **Purpose:** Shows the error message and demonstrates the failed connection attempt.

- **Screenshot 2: Successful Login via PuTTY**

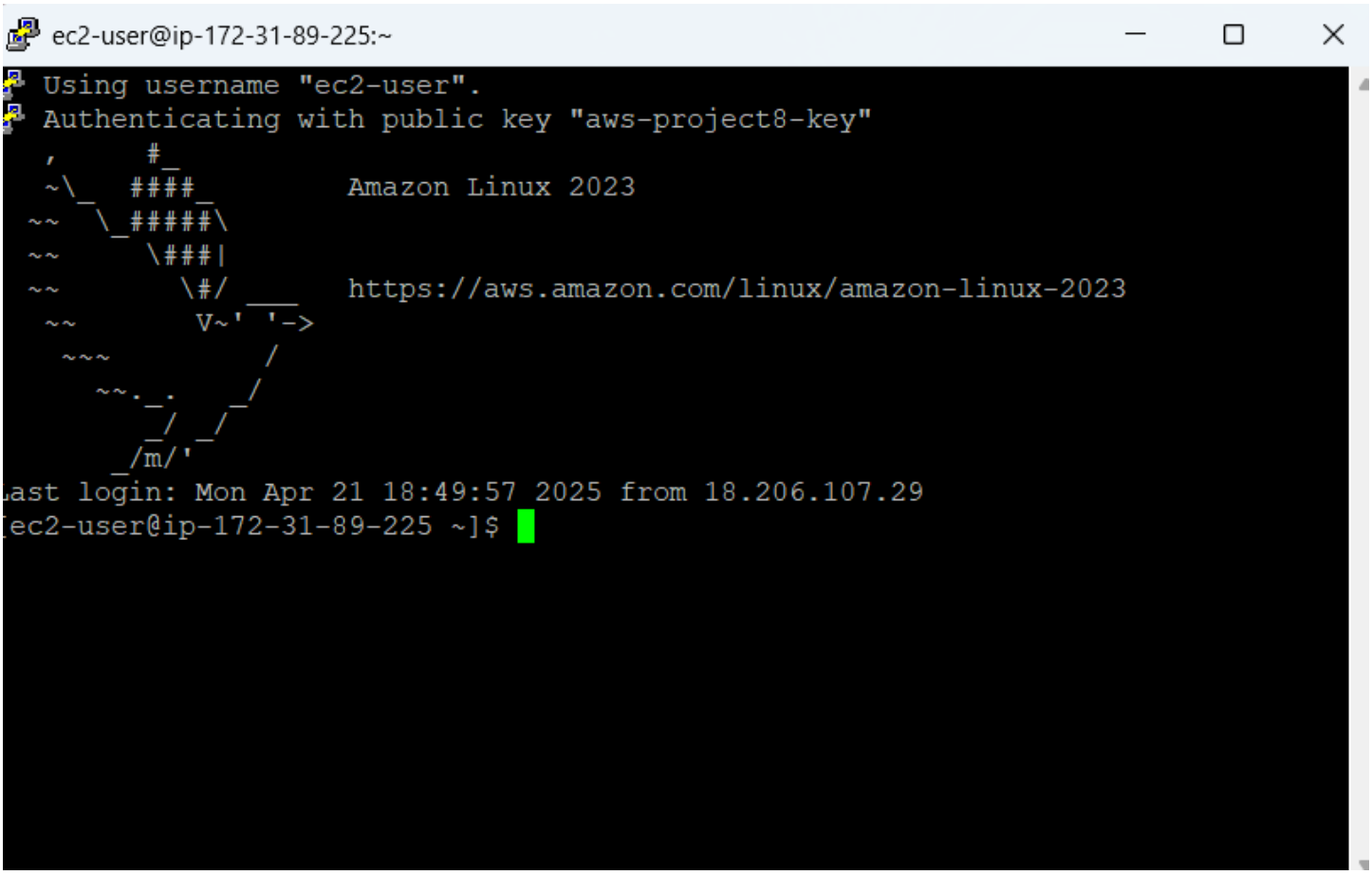


Figure 16: putty-successful-login

  - **Placement:** Directly after the section titled "Result: Successful SSH Connection"
  - **Purpose:** Confirms that the authentication problem was resolved and access to the EC2 instance was granted.

Step6: Testing ICMP Ping from EC2 Instance to Its Own Public IP

In this step, I logged into my EC2 instance via PuTTY and attempted to **ping the instance's own public IP address** to test ICMP connectivity.

Command Used:

ping 34.227.152.67

Here, 34.227.152.67 is the public IPv4 address assigned to the EC2 instance.

Result:

- The terminal output showed that ICMP echo requests were sent:

- 7 packets transmitted, 0 received, 100% packet loss
- This indicates that **no response was received** and **all ping attempts failed**.

Explanation:

- The EC2 instance was able to **send out** the ping requests, which confirms that **outbound ICMP traffic** is allowed.
- However, the instance was **not able to receive replies** because:
    - **AWS does not support loopback traffic to public IPs**, meaning an EC2 instance cannot reach itself using its public IP address.
    - Alternatively, **inbound ICMP Echo Reply** packets may have been blocked by the Security Group or Network ACL settings.

This is an expected result in AWS environments unless additional routing and security rules are configured to specifically allow such traffic.

Conclusion:

This step demonstrated how AWS restricts EC2 instances from communicating with themselves over their public IP addresses. It also emphasized the importance of configuring Security Group and Network ACL settings properly if ICMP-based diagnostics (like ping) are required from external sources.

Screenshot Reference:

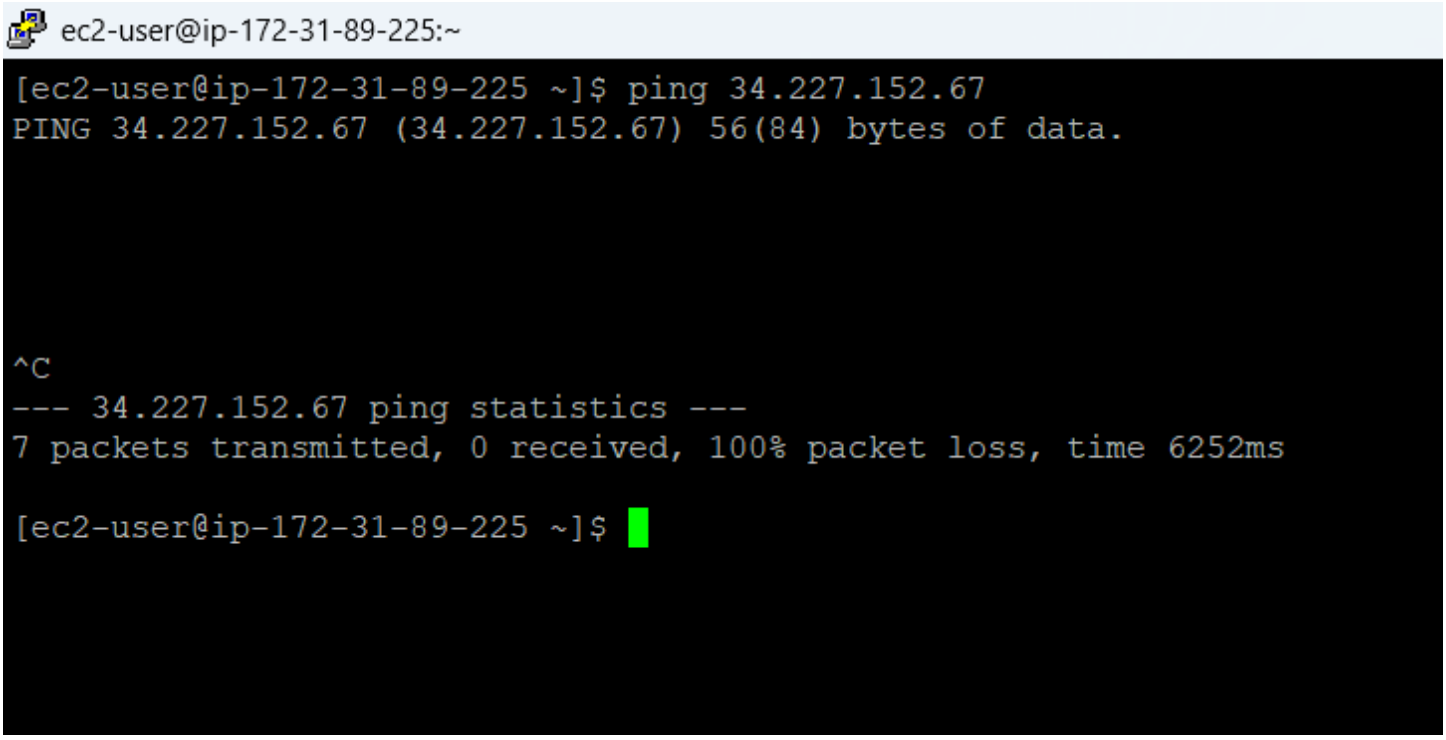


Figure 17: ping-self-public-ip

Step 7: Pinging Google's DNS Server to Test Internet Connectivity

After successfully connecting to the EC2 instance via PuTTY, I entered the following command to verify whether the instance could reach the internet:

ping 8.8.8.8

This IP address is the public DNS server provided by Google, and it's commonly used to test outbound connectivity.

Result:

The command successfully returned multiple responses:

64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=1.69 ms

...

7 packets transmitted, 7 received, 0% packet loss

This indicates that all ping requests were successful and there was **0% packet loss**.

Explanation:

The reason this worked is because:

- By default, **AWS EC2 instances allow outbound traffic** through their Security Groups unless explicitly restricted.
- The **default Security Group configuration** allows all outbound traffic, including ICMP (ping).
- The instance is in a public subnet with internet access, either through an Internet Gateway or NAT.

Therefore, the EC2 instance was able to reach an external public IP (Google DNS), confirming that:

- **Internet connectivity is working**
- **Outbound firewall rules are properly configured**

Conclusion:

This step confirms that the EC2 instance has full outbound internet access. The network is functioning correctly for outgoing traffic, and tools like ping can be used for connectivity diagnostics.

Screenshot Reference:

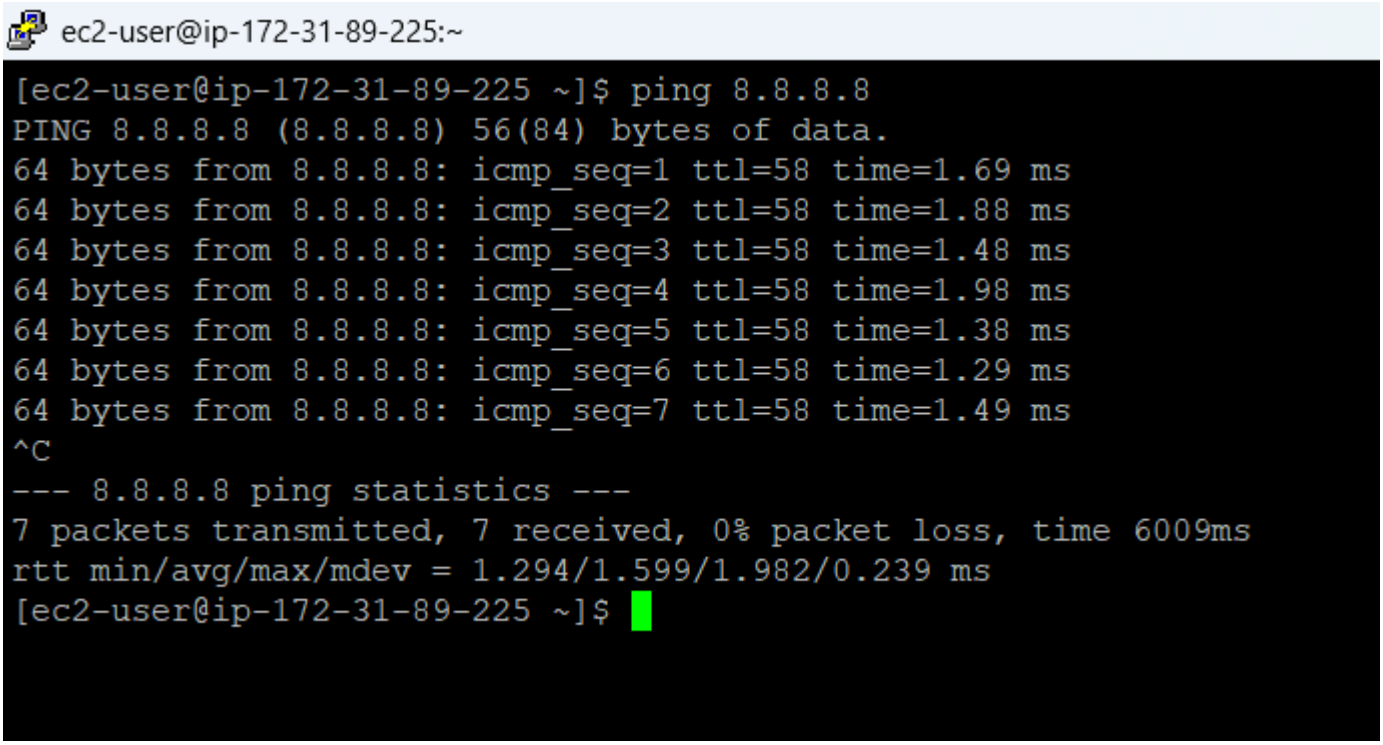


Figure 18: ping-google-dns

Step 8: Identify the EC2 Instance's Public IP Address

In this step, I returned to the **AWS EC2 Management Console** to gather detailed information about the running instance.

From the **Instance Summary** page, I located the **Public IPv4 address** under the instance details.

Result:

Public IP address: 34.227.152.67

This IP address is automatically assigned by AWS and is used to connect to the instance via SSH or other internet-based protocols.

Step 9: Ping the Instance from My Local Computer

To test whether inbound traffic from the internet is allowed, I opened **Command Prompt** on my local Windows machine and typed the following command:

ping 34.227.152.67

Result:

The ping **did not work**. It returned a timeout or 100% packet loss, such as:

Request timed out.

Why This Happened:

This behavior is **expected** and occurs because:

- By default, **AWS Security Groups block all inbound traffic** unless specific rules are added to allow it.
- Although **Security Groups are stateful**, meaning response traffic to an EC2-initiated request is allowed, **unsolicited inbound requests** like pings (ICMP Echo Requests) from external machines are not allowed.
- Since I had not yet added an **inbound rule to allow ICMP**, the instance did not reply to the ping request from my local computer.

Conclusion:

The instance could not be pinged from my local computer because **inbound ICMP traffic is blocked by default** in AWS. To make the ping work, I will need to modify the security group to allow **ICMP Echo Request** from my IP or from the public (0.0.0.0/0) depending on the test requirements.

Step 10: Reviewing and Understanding Security Group Configuration

To allow ping access (ICMP) from my local computer, I first reviewed the current security group settings attached to the EC2 instance.

Actions Taken:

1. **I navigated to the EC2 dashboard, selected my running instance, and clicked the Security tab.**
2. **Under the Security groups section, I clicked the active security group link:**

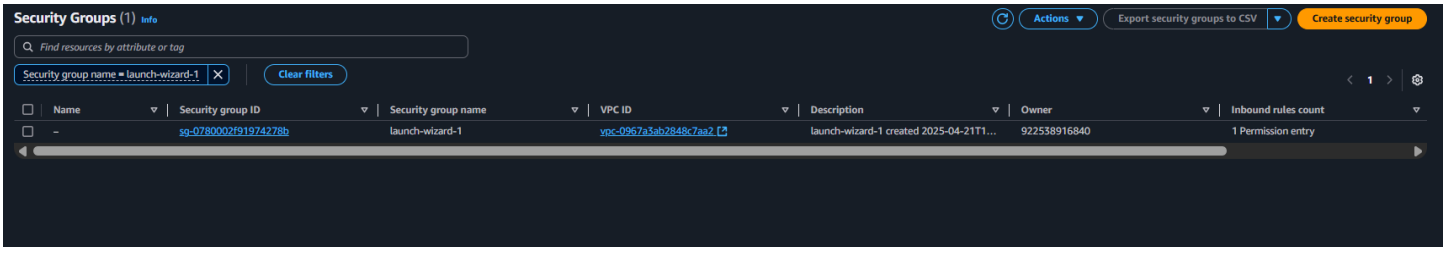


Figure 19: Security Group Overview

This opened the security group details page, where both inbound and outbound rules are listed.

Step 11: Inbound Rule Review

On the Inbound rules tab, I observed that there was only one inbound rule configured:

- **Type: SSH**
- **Protocol: TCP**
- **Port range: 22**
- **Source: 0.0.0.0/0 (Any IP)**

This rule allows SSH access to the instance from any IP address. However, no rule exists to allow inbound ICMP traffic, which is required for ping to work.

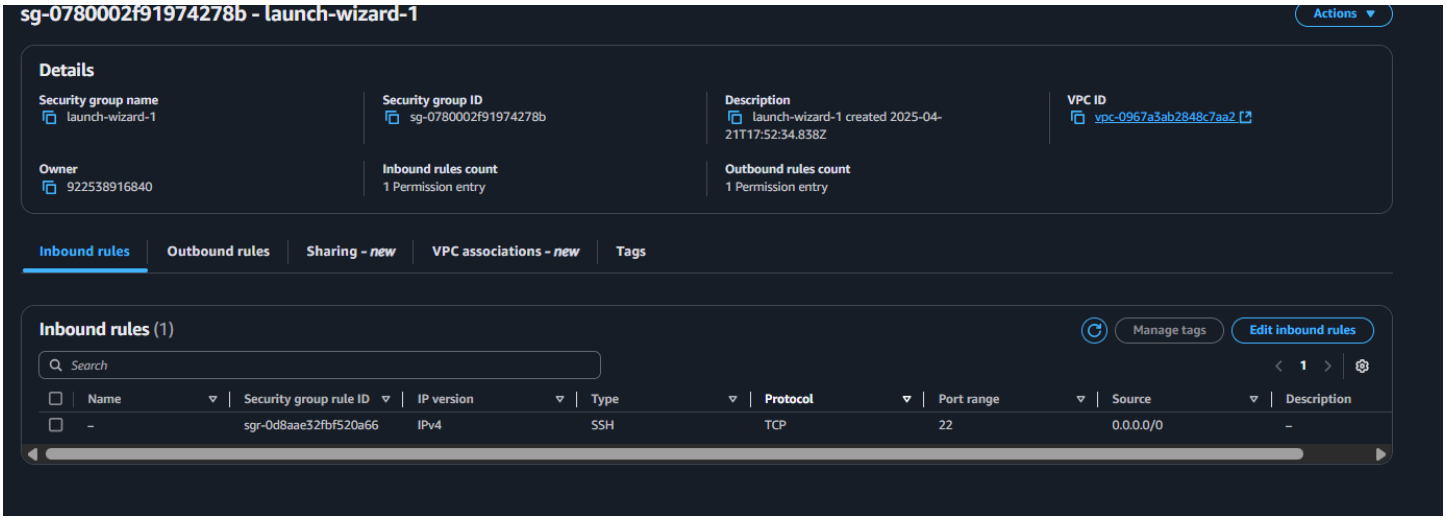


Figure 20: Inbound Rule

Step 11: Outbound Rule Review

In the Outbound rules tab, I found that the default rule allows:

- **Type: All traffic**
- **Destination: 0.0.0.0/0**

This means the instance can send any type of outbound traffic and will receive replies for connections it initiates. This was confirmed in Step 10, where pinging Google's DNS (8.8.8.8) was successful.

Step 12: Reviewing Outbound Rules

Next, I clicked on the **Outbound rules** tab to verify the traffic allowed out of the EC2 instance.

There was one rule:

- **Type:** All traffic

- **Protocol:** All
- **Port range:** All
- **Destination:** 0.0.0.0/0 (Any IP)

This means the EC2 instance is allowed to send **any type of traffic to the internet**, which includes outbound ICMP, as successfully tested in Step 10 when I pinged Google's DNS (8.8.8.8).

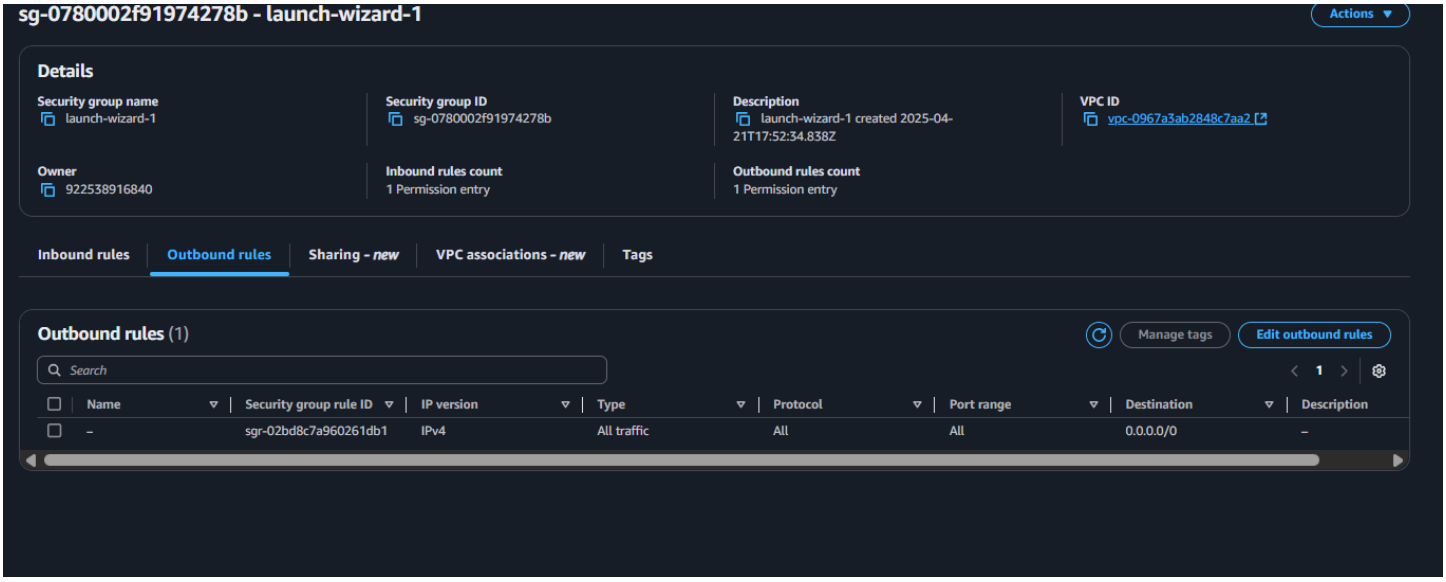


Figure 21: Output

Step 13: Editing the Security Group to Allow ICMP Traffic

To enable ping (ICMP Echo Request) to the EC2 instance, I edited the security group attached to the instance and added a rule that allows inbound ICMP traffic.

Actions Taken:

1. Opened the EC2 instance's **Security Group** configuration.
2. Clicked on **Edit inbound rules**.
3. Added a new rule with the following parameters:
    - **Type:** All ICMP – IPv4
    - **Protocol:** ICMP
    - **Port Range:** All
    - **Source:** 0.0.0.0/0 (Anywhere)

This rule allows any external IP address to send ICMP traffic to the instance, including ping requests. The SSH rule (port 22) remained unchanged to preserve remote access.

Screenshot –

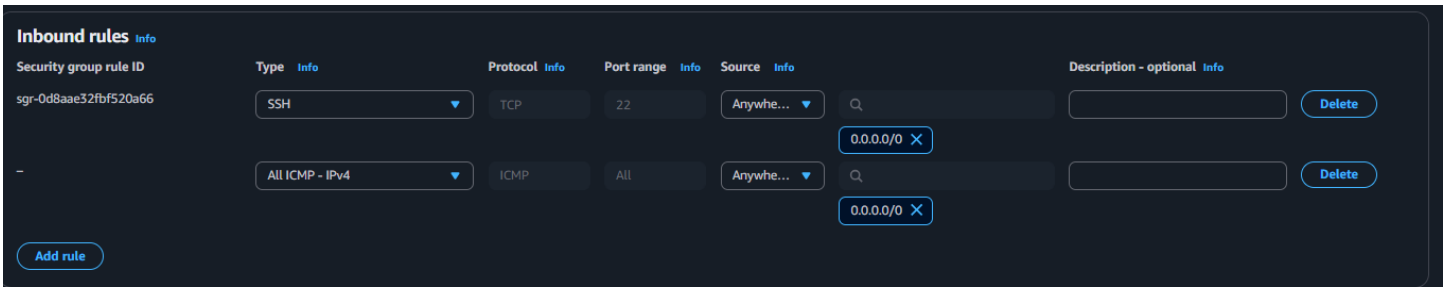


Figure 22: Inbound Rule Added in Edit Mode

After clicking **Save rules**, the system displayed a success message confirming that the inbound rule was applied successfully.

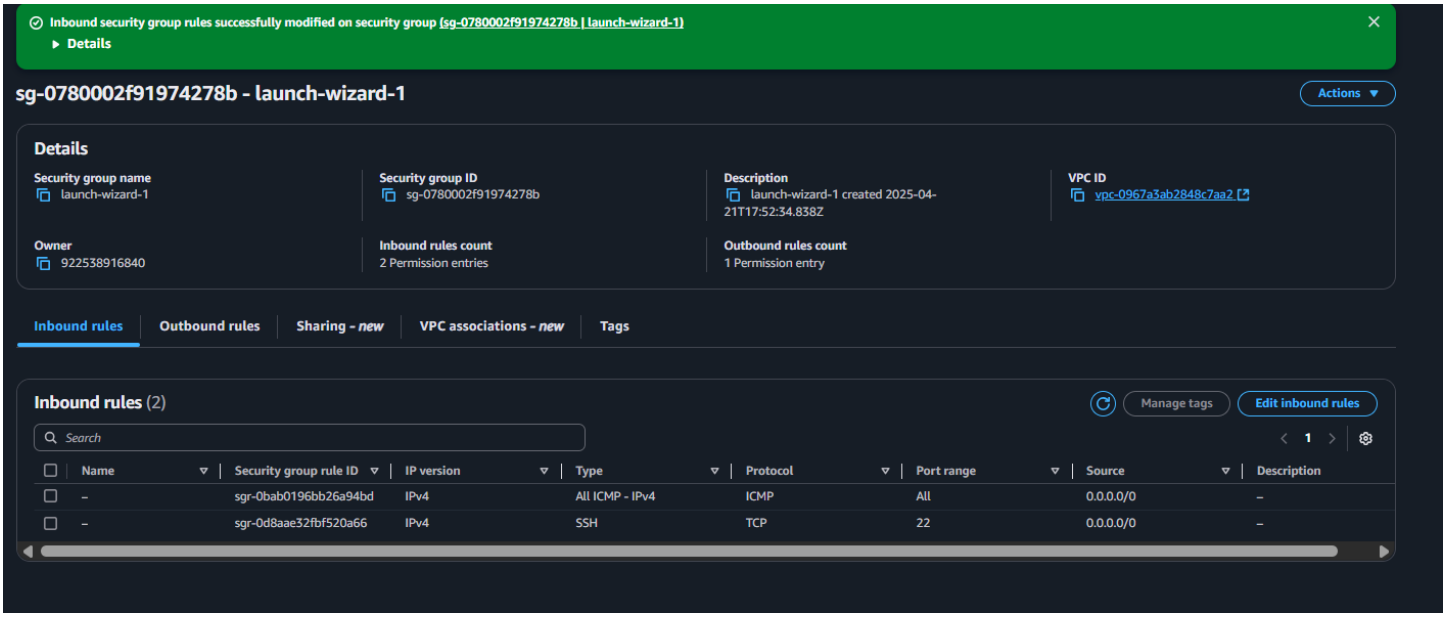


Figure 23: All inbound Rules

At this point, the security group now includes:

- SSH (TCP port 22) from any IP
- ICMP (All types) from any IP

Step 14: Verifying ICMP Access via Ping Test

To test whether the new rule was effective, I used the EC2 terminal to ping the instance's own **public IP address** (34.227.152.67).

Command Executed:

ping 34.227.152.67

Output:

The ping command returned a 0% packet loss with multiple replies, confirming that ICMP packets were received and responded to successfully.

Screenshot – Ping Test Result:

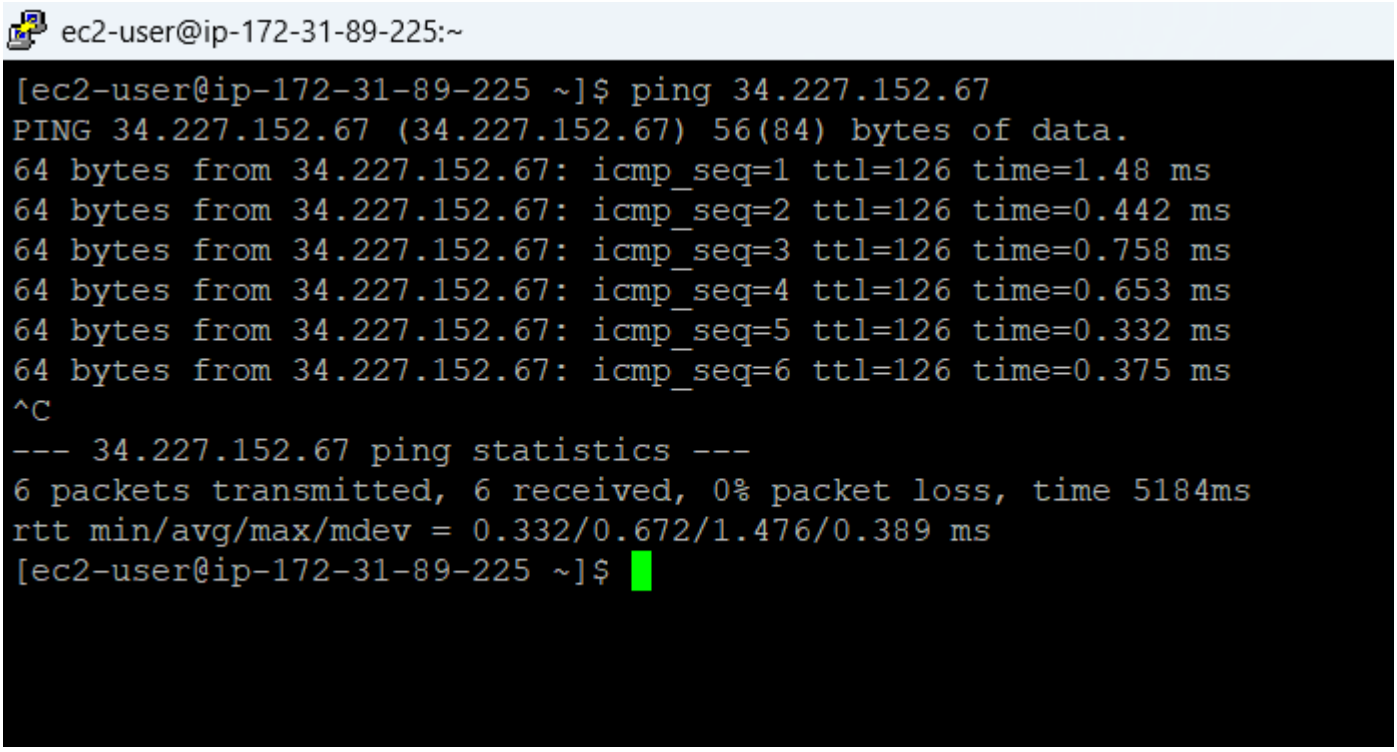


Figure 24: Ping Test Result

Step 15: VPC Overview and Architecture

Screenshot: VPC_Architecture_Overview

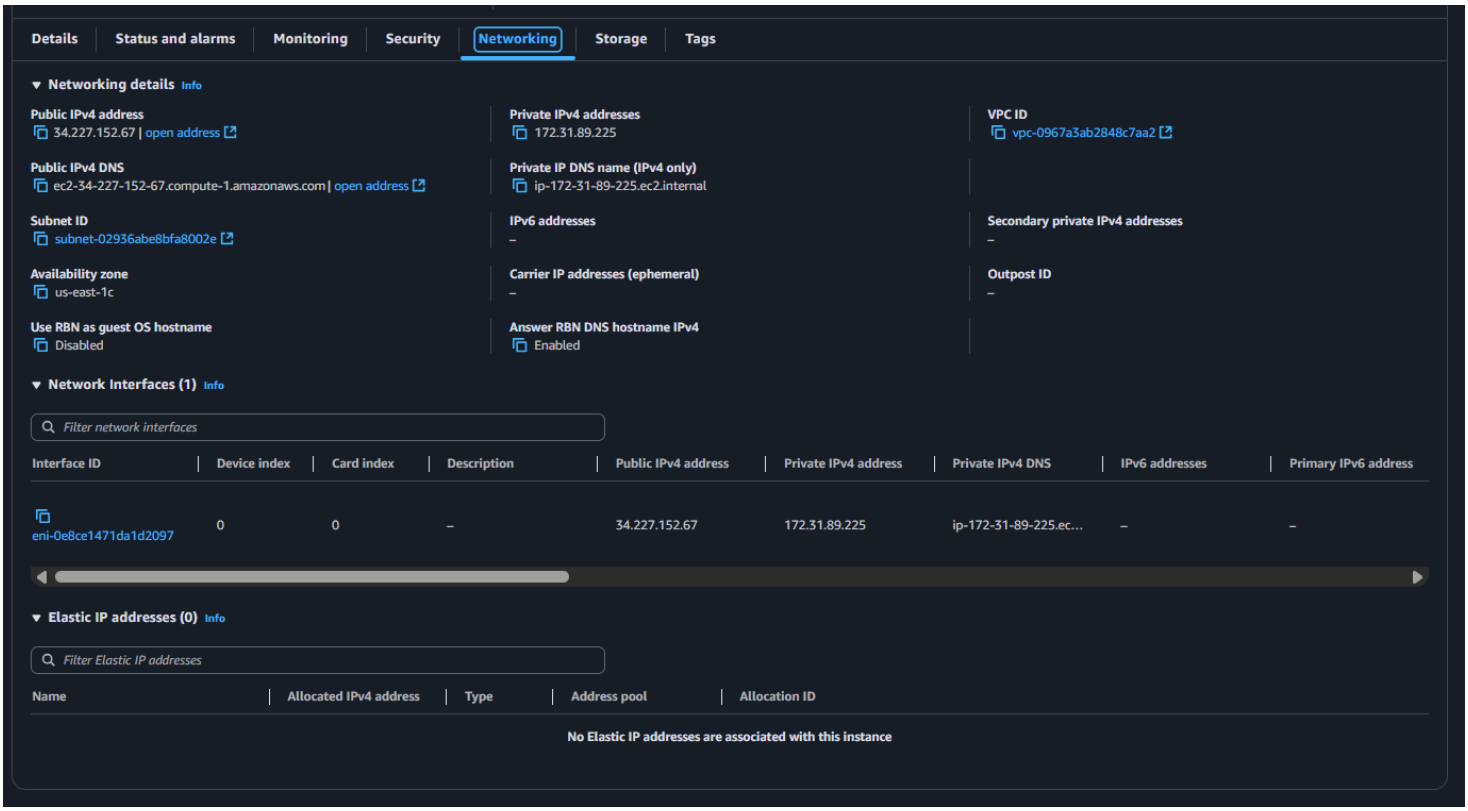


Figure 25: VPC_Architecture_Overview

This screen shows the detailed configuration of the Virtual Private Cloud (VPC) with ID vpc-0967a3ab2848c7aa2. Key details include:

- **VPC State**: Available
- **CIDR Block**: 172.31.0.0/16 (default AWS-assigned range)
- **DNS Resolution and Hostnames**: Enabled, which ensures EC2 instances within this VPC can resolve internal hostnames.
- **Internet Gateway**: igw-0be0651f8a932af9d is attached, which is essential for external communication from public subnets.
- **Route Table**: rtb-00fc4c940bd480904 connects all six subnets to the Internet Gateway.

This confirms that the VPC is correctly set up with internet accessibility.

Step 16: VPC Summary in Dashboard

Screenshot: VPC_List_Details

Figure 26: VPC_List_Details

This is the "Your VPCs" view, filtered by the selected VPC ID. It shows:

- **VPC ID**: vpc-0967a3ab2848c7aa2
- **CIDR Block**: 172.31.0.0/16
- **DHCP Option Set** and **Main Route Table** linked
- **Network ACL**: acl-0f1a070ea2737043a is listed as the default network-level firewall.

This overview confirms the VPC is configured with appropriate network controls and components.

Step 17: EC2 Instance Networking Tab

Screenshot: EC2_Networking_Info

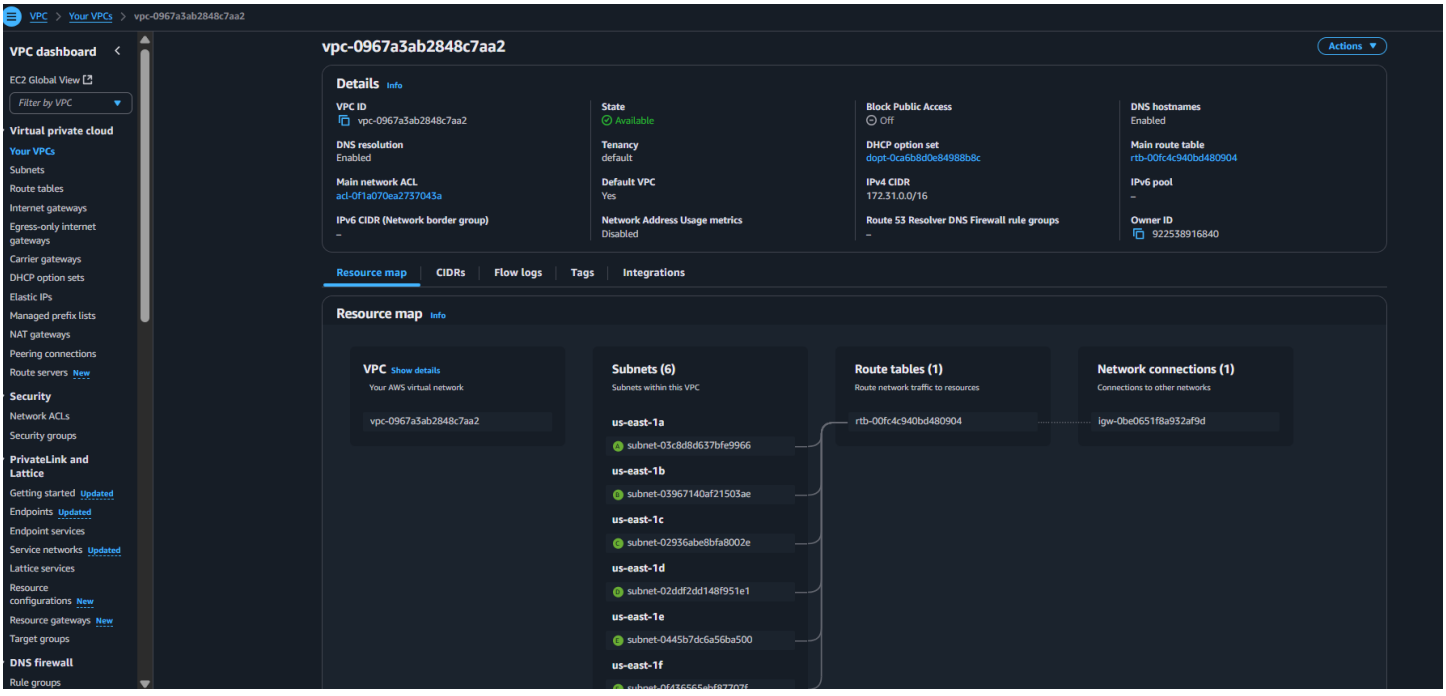


Figure 27: EC2_Networking_Info

This section displays the networking details of the EC2 instance:

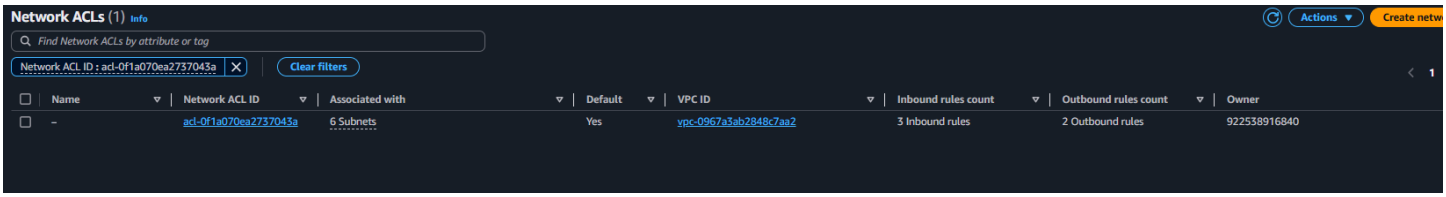
- **Public IPv4 Address**: 34.227.152.67
- **Private IPv4 Address**: 172.31.89.225

- **Public DNS**: ec2-34-227-152-67.compute-1.amazonaws.com
- **Subnet ID**: subnet-02936abe8bfa8002e
- **Availability Zone**: us-east-1c
- **Network Interface ID**: eni-0e8ce1471da1d2097

This confirms that the EC2 instance is publicly accessible and placed in a public subnet.

Step 18: Network ACL Inbound Rules

Screenshot: Network_ACL_Inbound_Rules



This screenshot shows the inbound rules for the Network ACL (acl-0f1a070ea2737043a) associated with your VPC:

- **Rule 1 and 100**: Allow all traffic from any IP address (0.0.0.0/0)
- **Rule * (default)**: Deny all (this only applies if no other rule matches)

The allow rules are evaluated before the deny rule, so traffic is permitted if it matches earlier rules.

Step 19: Network ACLs Overview

Screenshot: Network_ACL_List

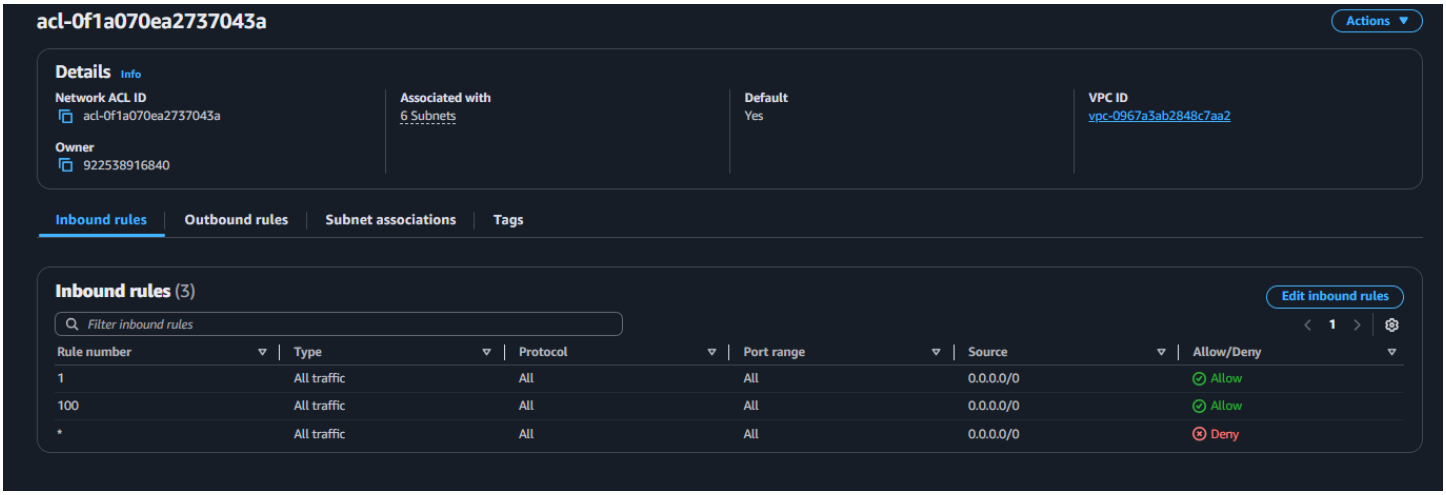


Figure 28: Network_ACL_List

This view shows a filtered list of Network ACLs:

- **ACL ID**: acl-0f1a070ea2737043a
- **Associated with**: 6 subnets
- **Inbound and Outbound Rules**: 3 and 2, respectively
- **Default**: Yes (this ACL applies to all subnets unless custom ACLs are applied)

This confirms the ACL applied to the VPC is configured to handle traffic properly.

Step 20: Editing the Network ACL to Block ICMP Traffic

(Referenced Screenshot: Screenshot 2025-04-21 170249.png)

1. Navigate to the **VPC Dashboard** and open the **Network ACL** section.
2. Select your NACL (e.g., acl-0f1a070ea2737043a) and click **Edit inbound rules**.
3. Click **Add new rule**.
4. Create a rule with a rule number **lower than 100** (e.g., 51) so it takes **priority** over the existing rule #100.
5. Set the **type** to All ICMP - IPv4, protocol to ICMP, and source to 0.0.0.0/0.
6. Under **Allow/Deny**, select Deny.
7. Click **Save changes**.

This rule ensures that any ICMP traffic from any IPv4 address is denied **before** the more general allow rule (Rule #100) is processed.

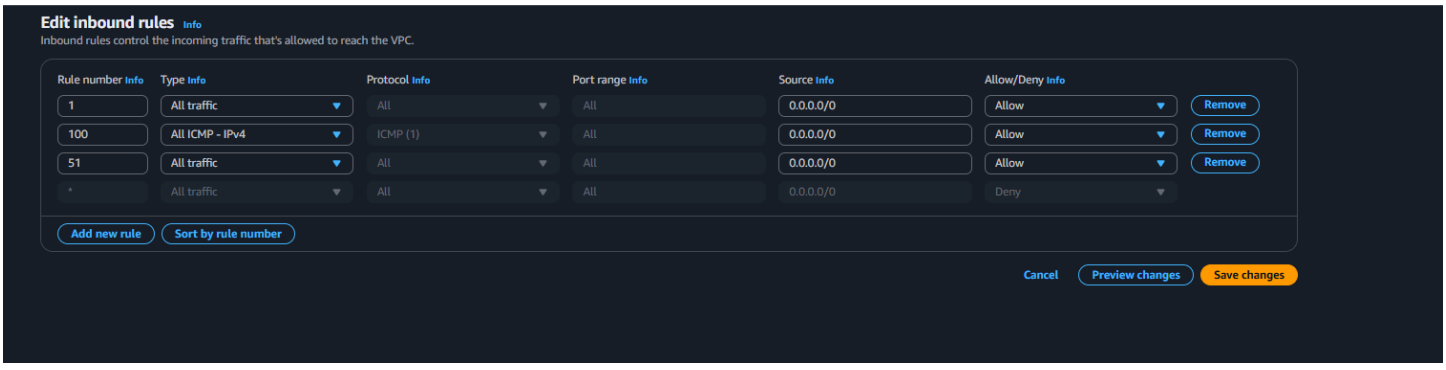


Figure 29: NACL_ICMP_Block_Rule

Step 21: Verifying Ping Test After Rule Change

After applying the NACL changes, you returned to your EC2 instance terminal and ran a **ping command** to test connectivity to another instance or endpoint.

The output shows:

- All packets were received (0% packet loss)
- Average round trip time was normal

This indicates that **outbound ping requests are still working** even after blocking inbound ICMP from the outside world, because the rule you applied **only affects traffic coming into the subnet from outside**, not the instance's own outbound traffic.

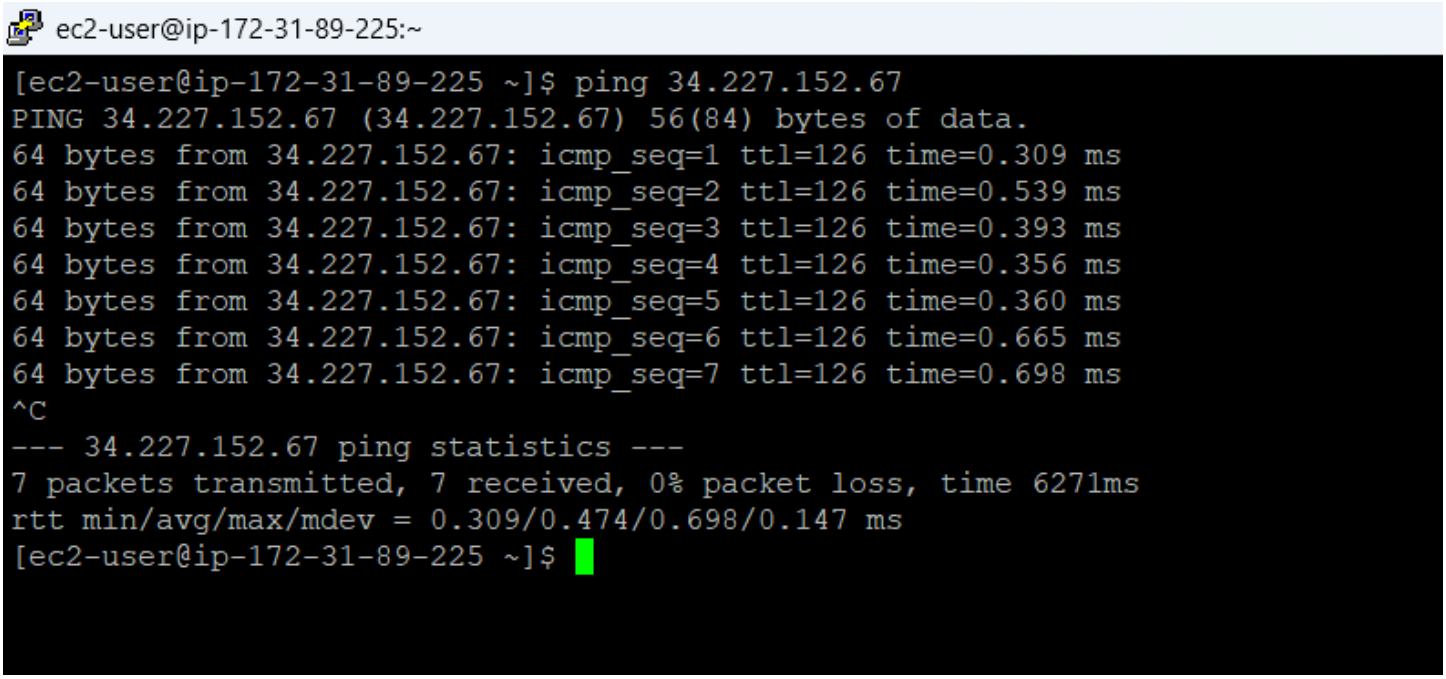


Figure 30: Ping_Success_From_EC2.

Answers to Questions

21. Task: Create an inbound NACL rule to block ICMP – IPv4 with a rule number lower than 100.

✅ **Completed.**

We created a rule with **rule number 51**, set it to block All ICMP - IPv4 from 0.0.0.0/0, and saved the changes. This rule will take precedence over the default allow rule at position 100.

22.Question: Is the NACL inbound rule alone enough to block ping requests from external devices?

Answer:

Yes, **the inbound NACL rule is sufficient** to block incoming ping requests from external sources.

NACLs are **stateless**, which means you would only need a corresponding **outbound rule** if your goal was to stop your EC2 instance from **sending** ping responses.
But since the goal is just to **block incoming pings**, **only the inbound ICMP block rule** is required.

23. Question: After applying the block rule, try to ping your EC2 instance from your local machine. Does it work?

Answer:
If configured correctly, the ping **from your local computer to the EC2 instance should fail**.
This is because the inbound ICMP traffic from your public IP is now **denied** at the NACL level **before** it reaches the instance.
Even if the security group allows ICMP, the NACL (which is evaluated first for network-level filtering) will drop the packet.

Additional Implementation: Real-Time Monitoring with Amazon CloudWatch

After completing the initial cloud security configuration project, I implemented an additional layer of operational visibility by configuring **Amazon CloudWatch monitoring with a custom alarm**.

This enhancement involved enabling **detailed monitoring on the EC2 instance** and creating a **CloudWatch alarm** that triggers when **CPU utilization exceeds a predefined threshold**. When triggered, the alarm sends a notification through **Amazon SNS (Simple Notification Service)** to alert administrators, helping ensure immediate awareness of any abnormal system behavior.

This addition enhances the project by enabling real-time monitoring and proactive incident response, aligning with cloud security best practices.

Step-by-Step Guide: Implementing CloudWatch Alarm for EC2 Monitoring

Step 1: Enabled EC2 Instance Monitoring via CloudWatch

To begin proactive resource monitoring and security tracking, I enabled CloudWatch Monitoring on my EC2 instance. This allows me to observe critical metrics such as CPU usage, network traffic, and packet flow in near real time.

Screenshot Reference

Below is the monitoring dashboard for my EC2 instance i-0cb1ce977937c0fe:

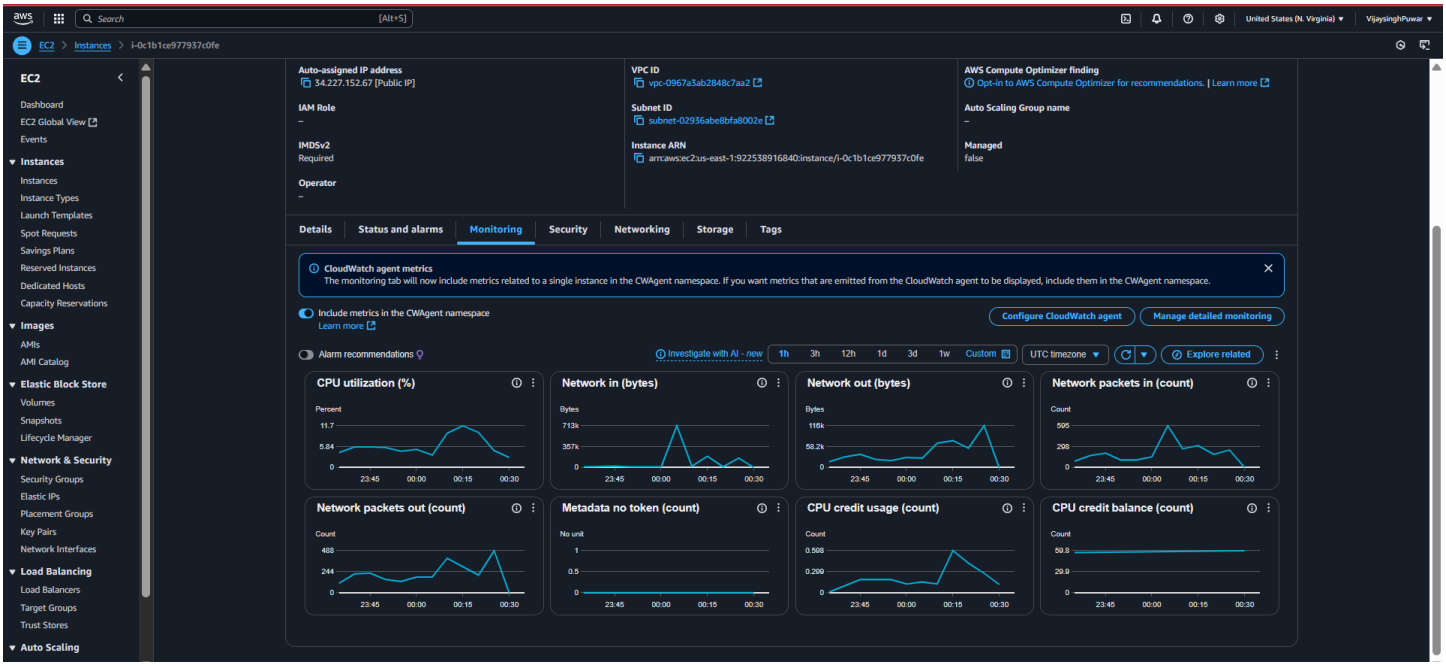


Figure 31: Monitoring Screenshot

What This Shows

This screenshot demonstrates that the Monitoring tab now displays real-time metric graphs due to CloudWatch Agent integration. Key visible elements include:

- CPU utilization (%): Real-time CPU load monitoring
- Network In/Out (bytes): Incoming and outgoing network traffic
- Network Packets In/Out (count): Packets processed by the instance
- CPU Credit Usage & Balance: For burstable performance (T-type instances)
- Metadata no token (count): Monitoring security requests via IMDSv2

Additionally, the blue banner confirms that metrics are being collected using the CloudWatch agent namespace, which enables richer and more granular insights.

Step 2: Open CloudWatch Console

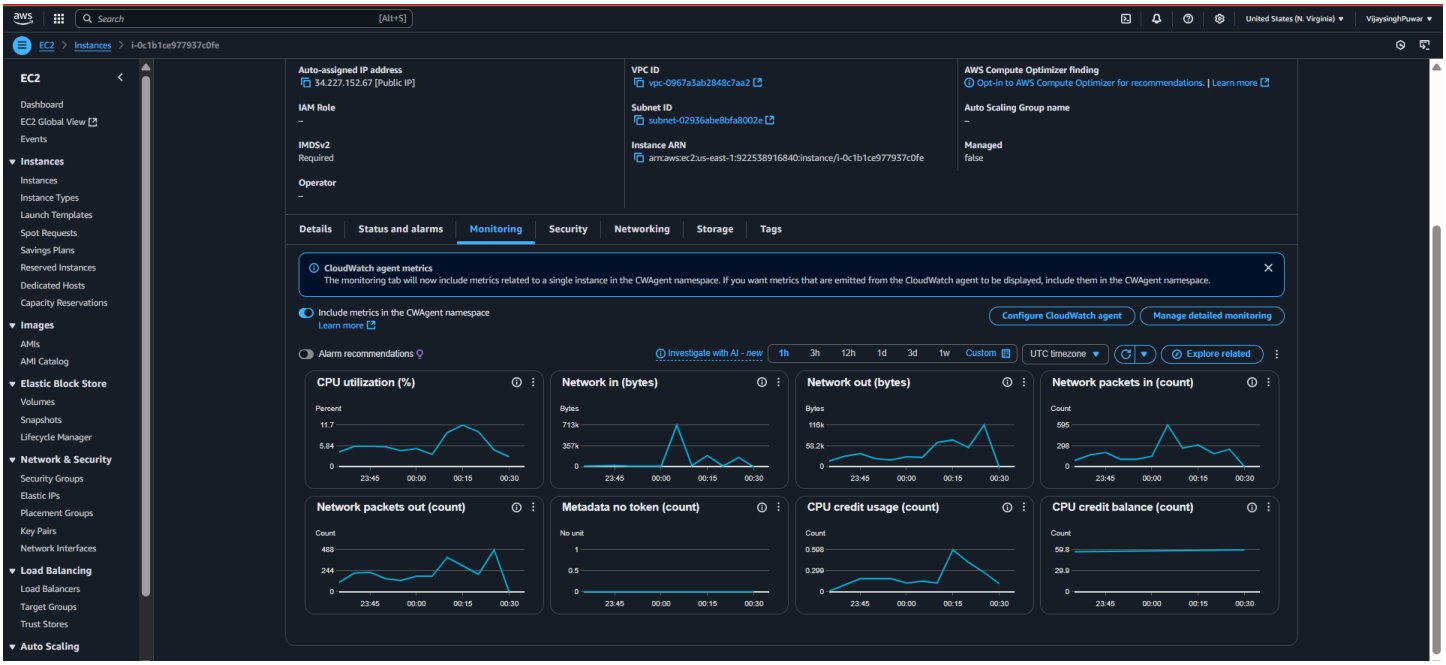


Figure shows the AWS CloudWatch dashboard.
This is where you manage monitoring features such as logs, alarms, metrics, and events.

Step 2: Confirm No Existing Alarms

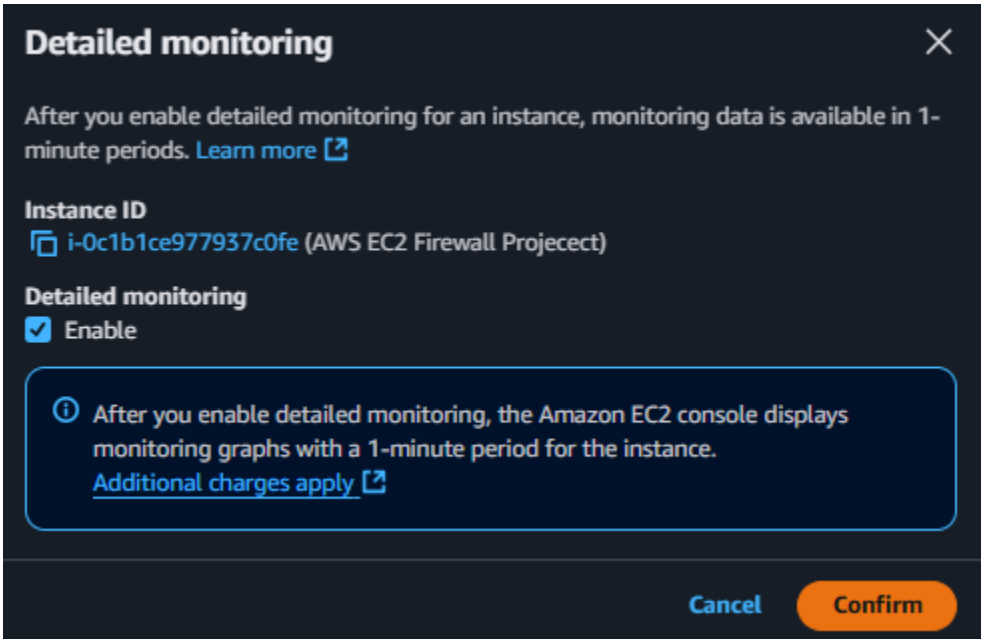


Figure displays the Alarms panel before any alarms were created.
It confirms a clean starting point for creating a new CloudWatch alarm.

Step 3: Select the Monitoring Metric

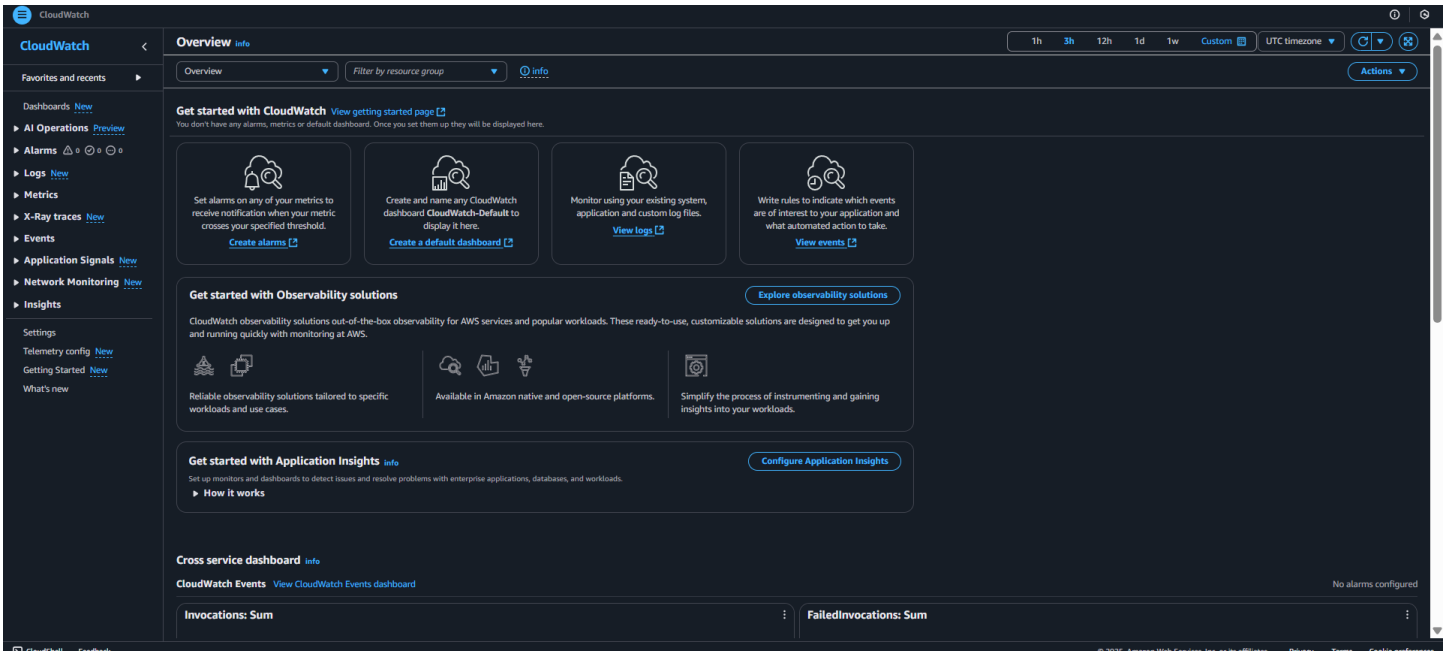


Figure illustrates the selection of the CPUUtilization metric from the list of available EC2 metrics.
This metric tracks the percentage of CPU usage for an instance, and it's crucial for detecting performance issues.

Step 4: Define the Alarm Metric and Threshold

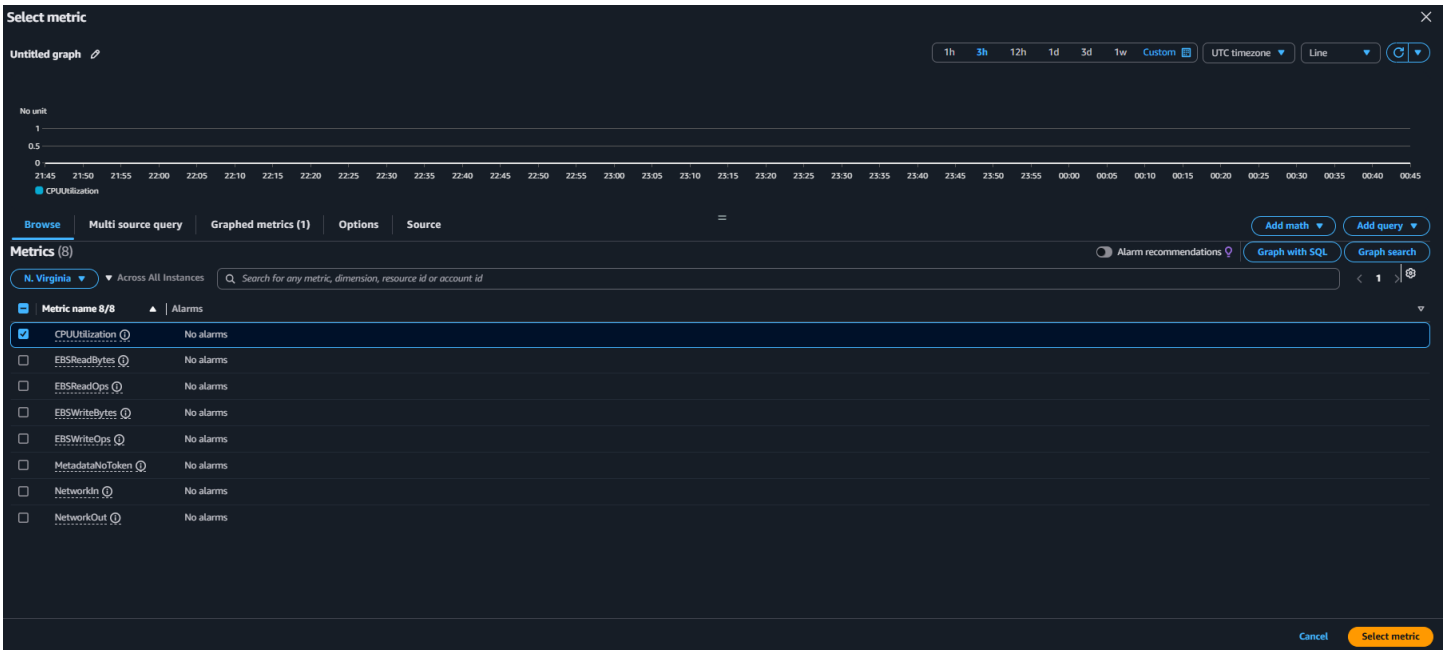


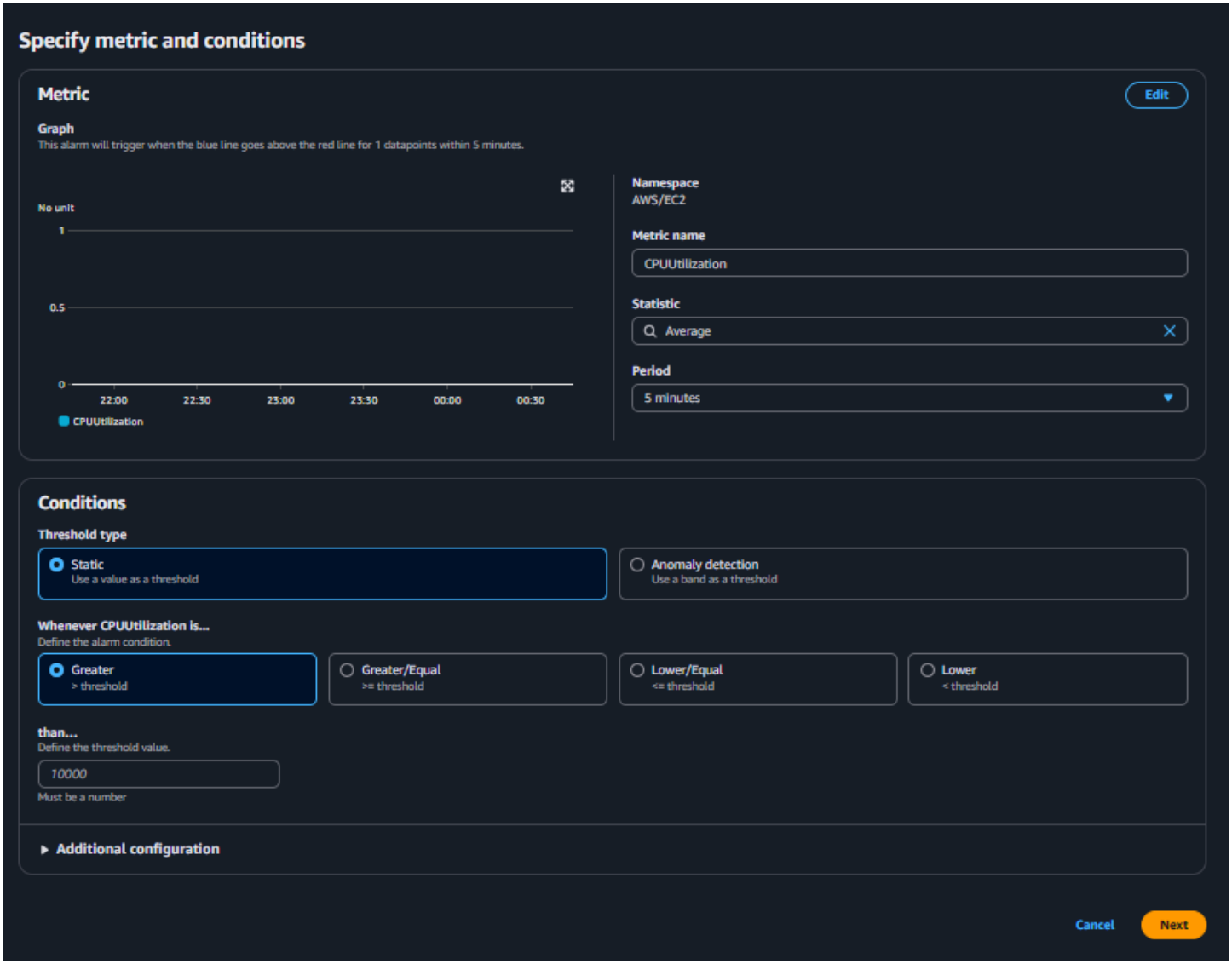
Figure shows the configuration of metric conditions:

- **Metric name:** CPUUtilization
- **Statistic:** Average
- **Period:** 5 minutes
- **Threshold type:** Static
- **Condition:** Greater than 10000 (test value to simulate alarm behavior)

This threshold is set high for testing purposes. In real-world scenarios, you would set a threshold like 70%.

Note: **Figure 5** contains a similar view and may be skipped for redundancy.

Step 5: Configure Notification Action



As seen in **Figure** , the alarm is set to trigger **in alarm** state and notify an **SNS topic** named Default_CloudWatch_Alarms_Topic.
An email endpoint vpuwar199@gmail.com is added to this topic, which will receive alarm notifications upon confirmation.

Step 6: Add Alarm Name and Description

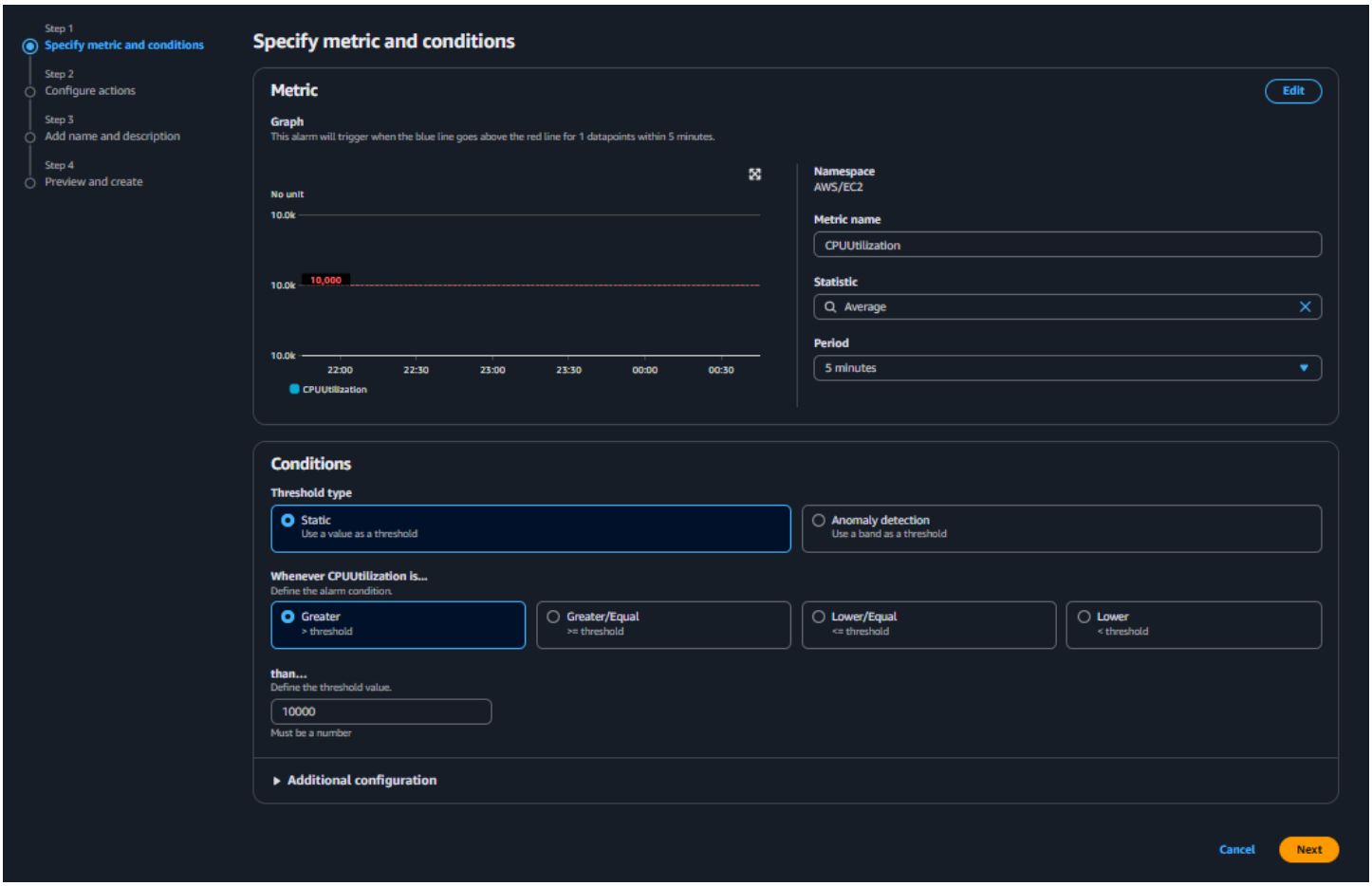


Figure shows the naming and description step.

- **Alarm name:** OverUsed
- **Description:**

Triggers an alert when EC2 instance CPU utilization exceeds 70% for more than 3 data points in 5 minutes. Helps detect performance bottlenecks or potential attack-related spikes.

This step provides documentation and clarity on what the alarm is configured to do.

Step 7: Final Confirmation

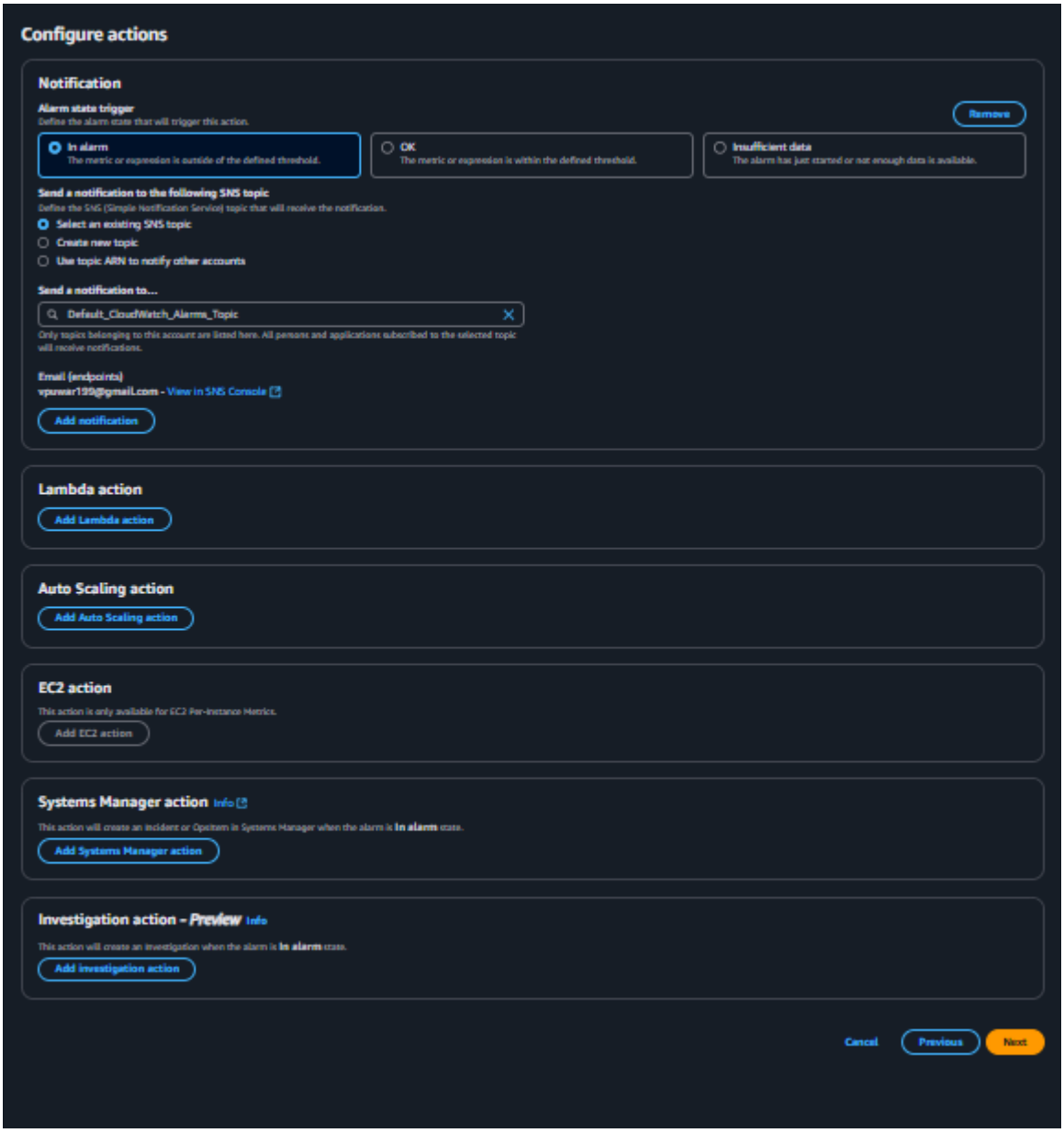


Figure confirms that the alarm OverUsed has been successfully created.
Note: A blue banner shows that the SNS subscription is pending confirmation, which must be completed via email.

Step 8: Enable Detailed Monitoring for EC2

Figure captures the confirmation dialog where **detailed monitoring** is enabled.
This increases the metric data frequency from 5 minutes to 1 minute, allowing faster and more accurate detection of resource spikes.

Step 9: EC2 Monitoring Dashboard

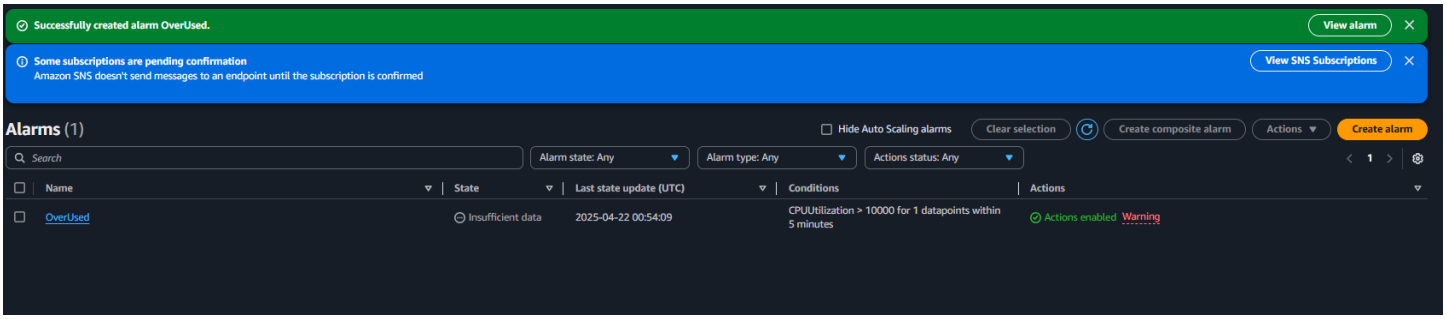
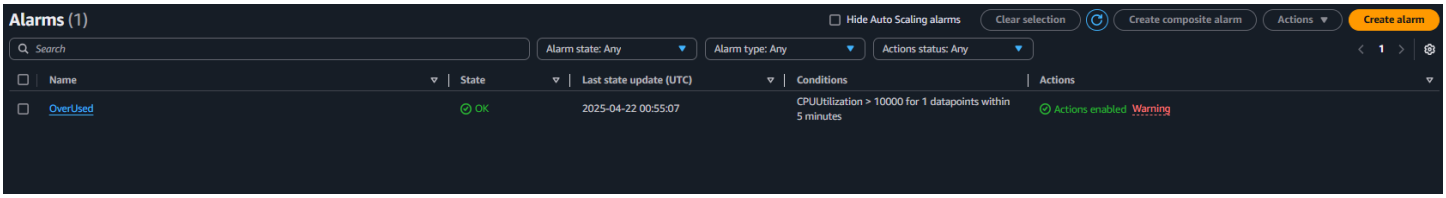


Figure displays the EC2 instance's monitoring tab.
It shows real-time graphs for:

- CPU Utilization
- Network In/Out
- Metadata
- CPU credit usage

This proves that monitoring is active and detailed metrics are being collected.

Step 10: Alarm in Active State



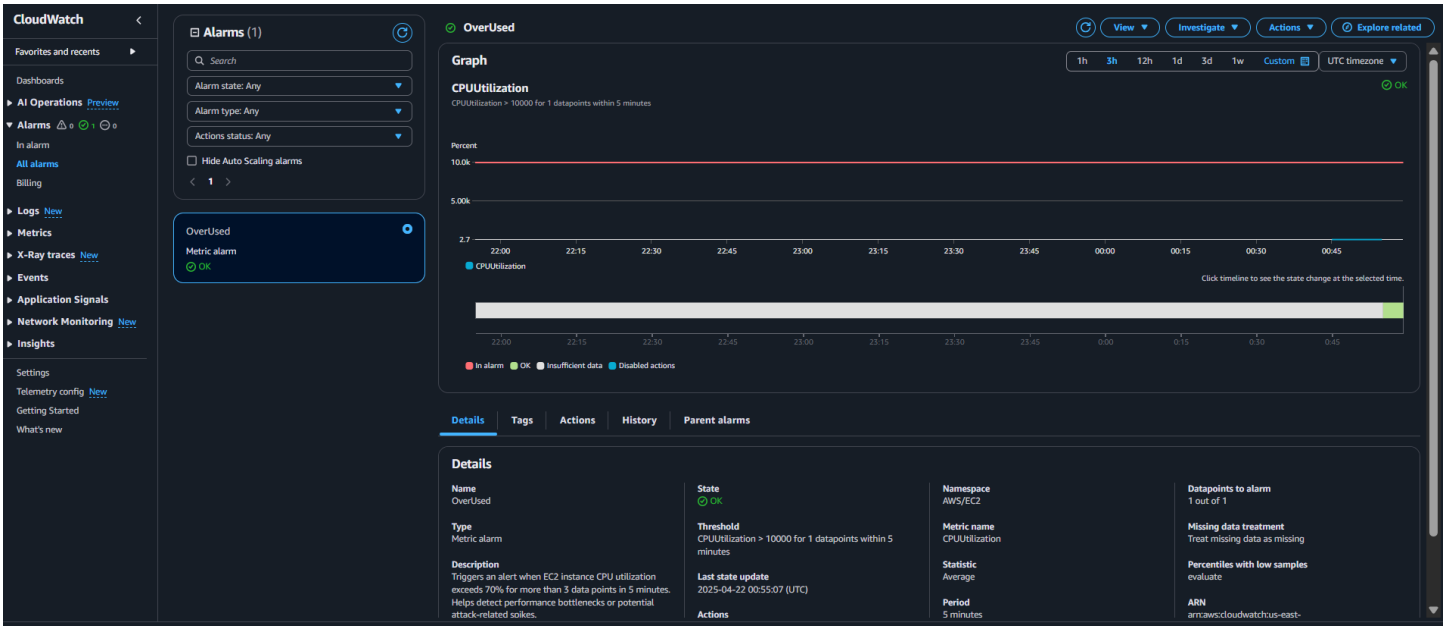
In **Figure**, the created alarm OverUsed appears under the CloudWatch alarms panel with status OK, meaning no threshold breach has occurred yet.
This shows the alarm is properly configured and actively monitoring.

Step 11: Alarm Details and History

Figure 12 provides an in-depth look at the alarm's:

- Graph timeline
- State transition (OK, In Alarm, Insufficient Data)
- Description
- Conditions
- Last update timestamp
- Namespace and metric settings

This confirms that the setup is complete, accurate, and the monitoring system is functioning.



Summary: Configuring and Securing AWS Cloud Infrastructure with Real-Time Monitoring

This project showcases a detailed and methodical approach to configuring secure and observable infrastructure in Amazon Web Services (AWS), using a combination of EC2 instances, Security Groups, Network ACLs, and Amazon CloudWatch Monitoring. It is designed to demonstrate the fundamental principles of cloud security, access control, and proactive performance monitoring.

Core Project Achievements:

1. **AWS EC2 Instance Deployment**
    - Launched a t2.micro Amazon Linux 2 EC2 instance.
    - Configured secure SSH access using a custom key pair (.ppk).
    - Gained console and PuTTY-based terminal access for hands-on operations.
2. **Security Group Configuration**
    - Created custom inbound/outbound rules.
    - Allowed essential traffic such as SSH (port 22) and later enabled ICMP (ping) for diagnostics.
    - Tested connectivity and validated firewall behavior with ping tests.
3. **Network ACL Implementation**
    - Explored the stateless nature of NACLs by creating inbound deny rules for ICMP.
    - Observed that despite Security Groups allowing traffic, NACLs could effectively block inbound ping.
    - Highlighted how layered security improves protection at both instance and network levels.
4. **Connectivity Validation & Troubleshooting**
    - Verified internet connectivity via ping to external servers (e.g., Google DNS 8.8.8.8).
    - Diagnosed why self-ping using public IP failed and explained AWS behavior regarding loopback traffic.
    - Addressed PuTTY authentication errors by properly configuring the key path and username.

Additional Enhancement: CloudWatch Monitoring and Alarm Setup

To strengthen the security observability aspect of the project, an additional feature was implemented:

1. **Enabled Detailed Monitoring for the EC2 Instance**

- Collected granular performance data every 1 minute using the CloudWatch agent.

2. **Created a CloudWatch Metric Alarm**
    - Configured an alarm for high CPU usage (CPUUtilization > 70%).
    - Linked the alarm to an SNS topic for real-time email notifications.
    - Ensured proactive monitoring and early warning for potential performance anomalies or attacks.
3. **Tested Alarm Behavior and Verified Setup**
    - Reviewed the CloudWatch console for real-time metric graphs.
    - Confirmed alarm was active and functioning, with pending SNS email confirmation in place.
    - This reflects production-grade readiness and alerting.

Key Takeaways:

- **Security Groups and NACLs** work together to form a defense-in-depth model in AWS. While Security Groups are stateful and instance-level, NACLs are stateless and subnet-wide, allowing granular control.
- **SSH configuration** and **key-based access** were successfully applied, addressing real-world issues like authentication errors.
- **ICMP traffic behavior** in AWS clarified internal limitations and required configurations to allow ping from external machines.
- **CloudWatch monitoring** and **SNS-based alerting** added a professional layer of observability and incident response, ensuring not just secure, but smart cloud architecture.

Professional Impact:

This project demonstrates practical experience in:

- Cloud security controls
- Network diagnostics
- Real-time monitoring and alerting
- Resolving connection errors
- Configuring access permissions in cloud environments

These skills are directly aligned with the responsibilities of a Cloud Security Engineer, SOC Analyst, or AWS Practitioner role.