AWS EC2 Traffic Control Lab- Security Groups vs NACL

1. Overview

This lab demonstrates the practical differences between **Security Groups (Stateful)** and **Network ACLs (Stateless)** in AWS. We provisioned a custom VPC and an EC2 instance to host a simple Python HTTP server. We then tested connectivity by manipulating firewall rules at the instance level (Security Group) and the subnet level (NACL) to observe traffic behavior and rule priority logic.

Goal: To prove that Security Groups operate at the instance level (allowing return traffic automatically), while NACLs operate at the subnet level (stateless), and to demonstrate how NACL rule numbering affects traffic evaluation.

2. Environment Details

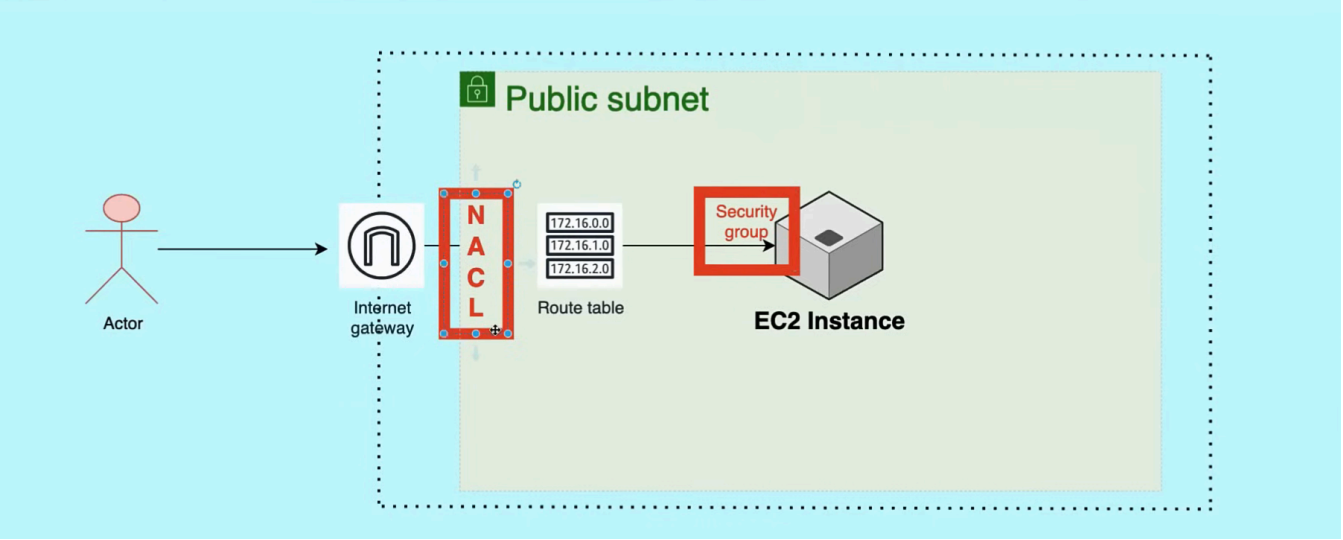
Derived directly from the uploaded evidence.

- **Region:** `ap-south-1` (Mumbai).
- **VPC:** `trafic-control-lab-vpc` (CIDR: `10.0.0.0/16`).
- **Subnets:** `trafic-control-lab-subnet-public1` (IPv4: `10.0.0.0/20`).
- **EC2 Instance:**
- **Instance ID:** `i-07e9a15a76ead02ff`.
- **Private IP:** `10.0.10.205`.
- **Public IP:** `13.232.34.158`.
- **Security Group:** `launch-wizard-3` (`sg-070a840227267d88f`).
- **Network ACL:** `acl-0a5bb1a55ec0b49f3` (Associated with `vpc-0932b3c7f35262c38`).
- **Client IP:** `106.215.180.99` (Detected in server logs).

3. What I Built (Architecture)

We created a custom VPC network and deployed a Linux server running a Python web service on port 9080. We then placed filters (SG and NACL) in the traffic path to control access.

Architecture Diagram:



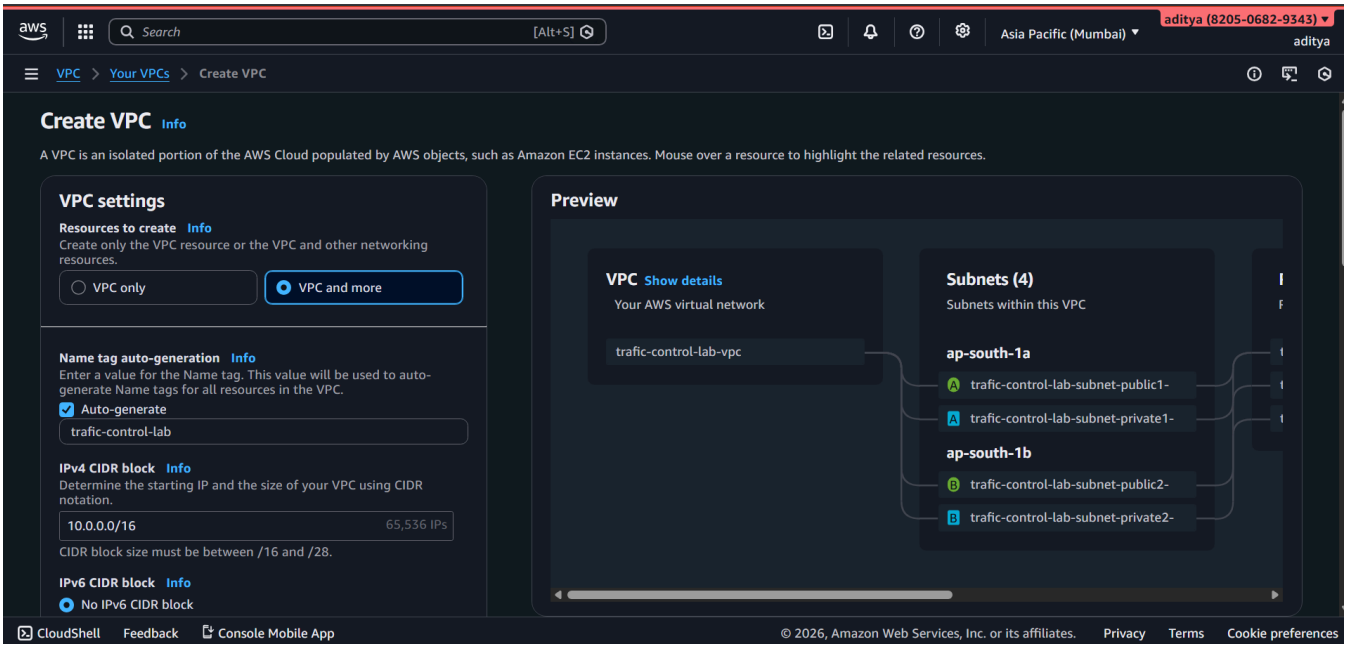
Traffic flow diagram showing the relationship between NACLs and Security Groups.

```
Internet (Client IP: 106.215.180.99)
           |
           v
  [ Internet Gateway ]
           |
+----------|----------------------+
| VPC: trafic-control-lab-vpc     |
|          |                      |
|   [ Network ACL ] <---(Subnet Boundary)
|          |                      |
|   +------v-------+              |
|   | Public Subnet|              |
|   |      |       |              |
|   |  [ Sec Grp ] <---(Instance Boundary)
|   |      |       |              |
|   |   [ EC2 ]    |              |
|   | App: :9080   |              |
|   +--------------+              |
+---------------------------------+
```

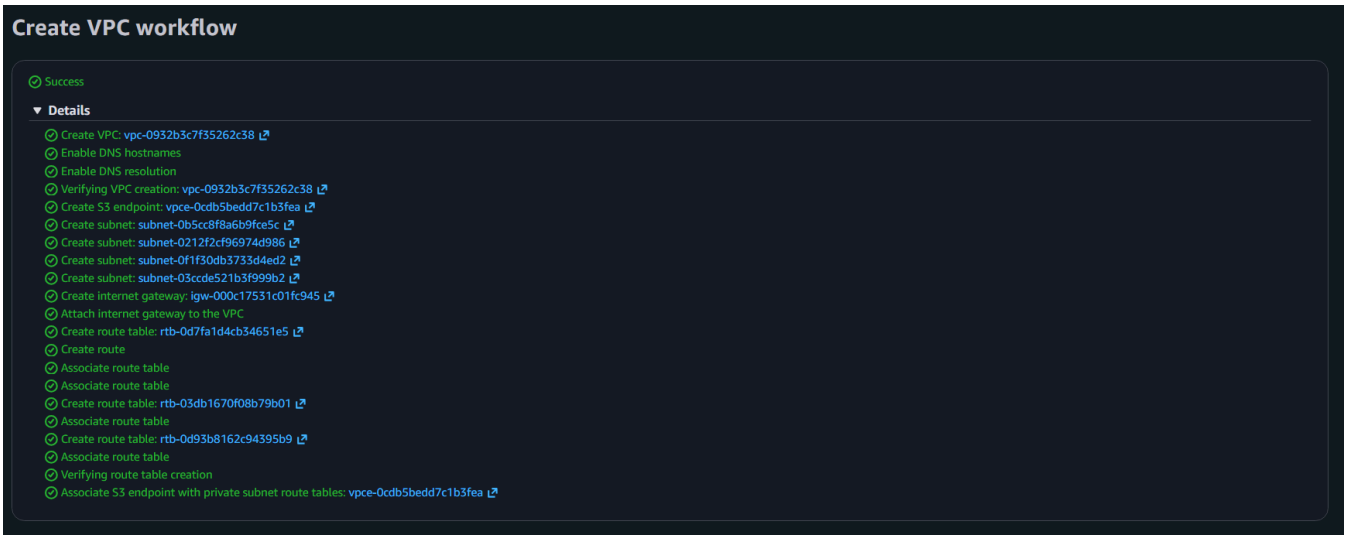
4. Step-by-Step Lab Procedure

Phase 1: Infrastructure Setup

1. **Created VPC:** Used the "VPC and more" wizard to create `trafic-control-lab-vpc` with public/private subnets.

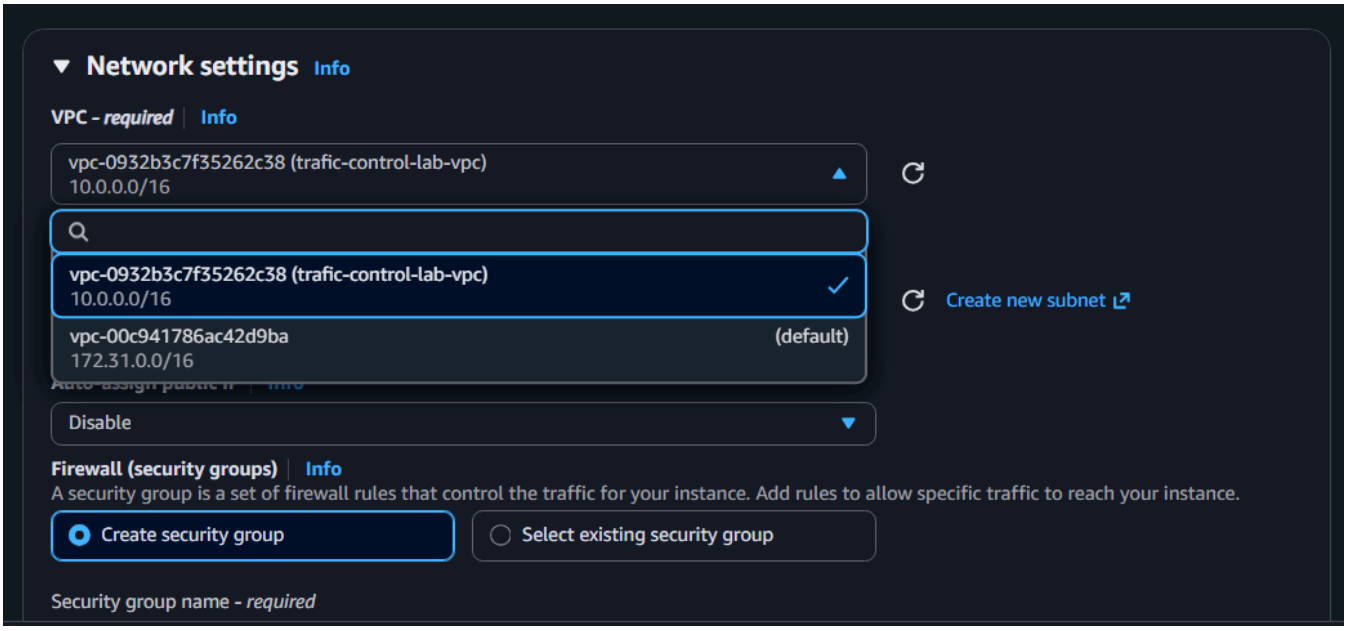


Defining the VPC settings with IPv4 CIDR 10.0.0.0/16.

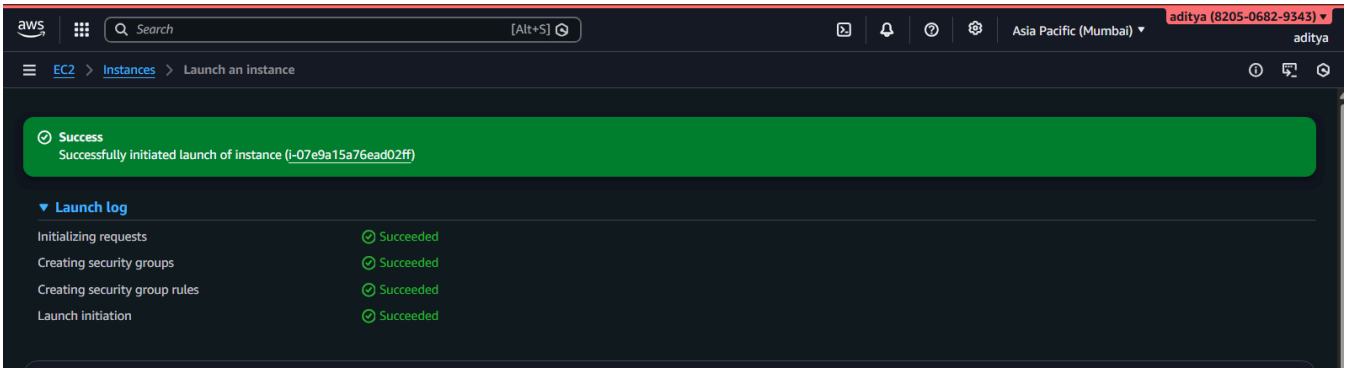


Validation that VPC, Subnets, and IGW were created successfully.

2. **Launched EC2:** Deployed an Ubuntu instance into the new VPC.



Mapping the network settings to our custom VPC.



Successfully initiated launch of instance i-07e9a15a76ead02ff.

3. **Access:** Successfully SSH'd into the instance using the key file.

```
C:\Users\Nitro\Downloads>ssh -i "key.pem" ubuntu@ec2-13-232-34-158.ap-south-1.compute.amazonaws.com
The authenticity of host 'ec2-13-232-34-158.ap-south-1.compute.amazonaws.com (13.232.34.158)' can't be established.
ED25519 key fingerprint is SHA256:fZiEPrTr67chC8tHq1DtWGAWz5JT+b5trowUs4vQgG8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-13-232-34-158.ap-south-1.compute.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1018-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri Jan 23 11:56:31 UTC 2026

  System load:  0.14              Temperature:           -273.1 C
  Usage of /:   25.9% of 6.71GB   Processes:             116
  Memory usage: 25%               Users logged in:       0
  Swap usage:   0%                IPv4 address for ens5: 10.0.10.205

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
```

Terminal showing successful login to ubuntu@10.0.10.205.

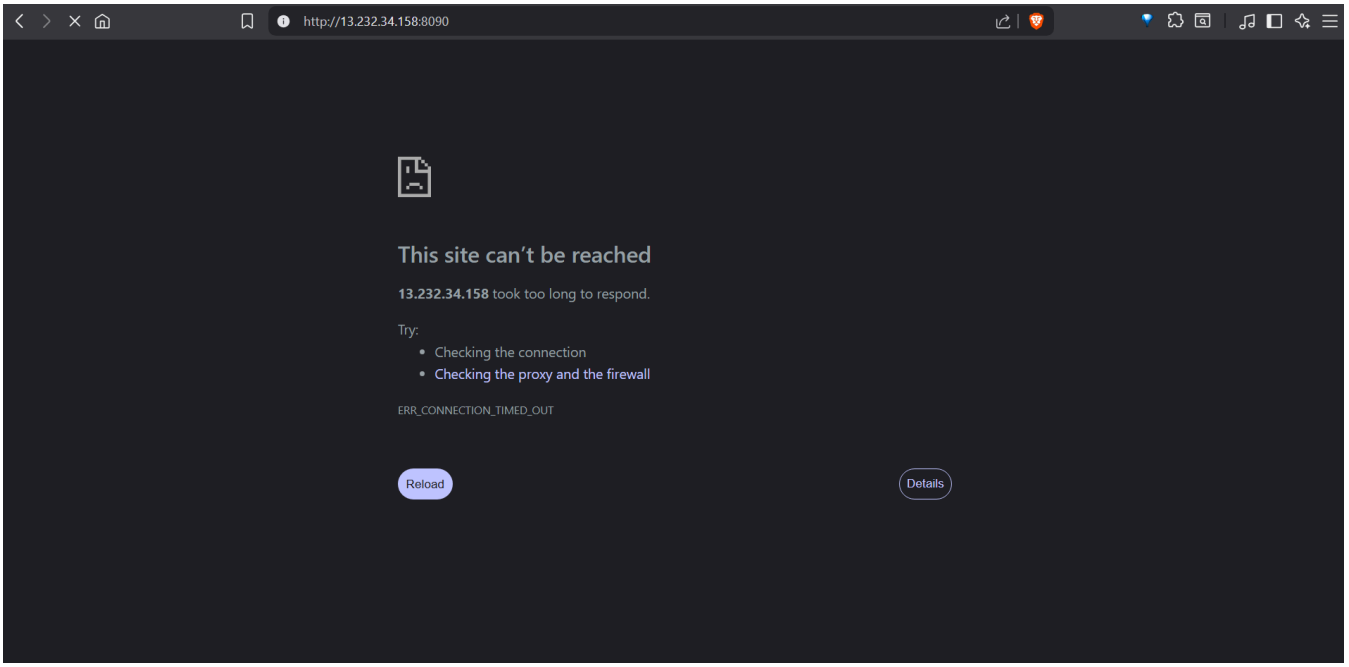
Phase 2: Application Setup & Initial Tests

4. **Started Application:** Ran a simple Python HTTP server on port 9080.

```
ubuntu@ip-10-0-10-205:~$ python3
Python 3.12.3 (main, Jan  8 2026, 11:30:50) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
ubuntu@ip-10-0-10-205:~$ python3 -m http.server 9080
Serving HTTP on 0.0.0.0 port 9080 (http://0.0.0.0:9080/) ...
_
```

Command `python3 -m http.server 9080` executing.

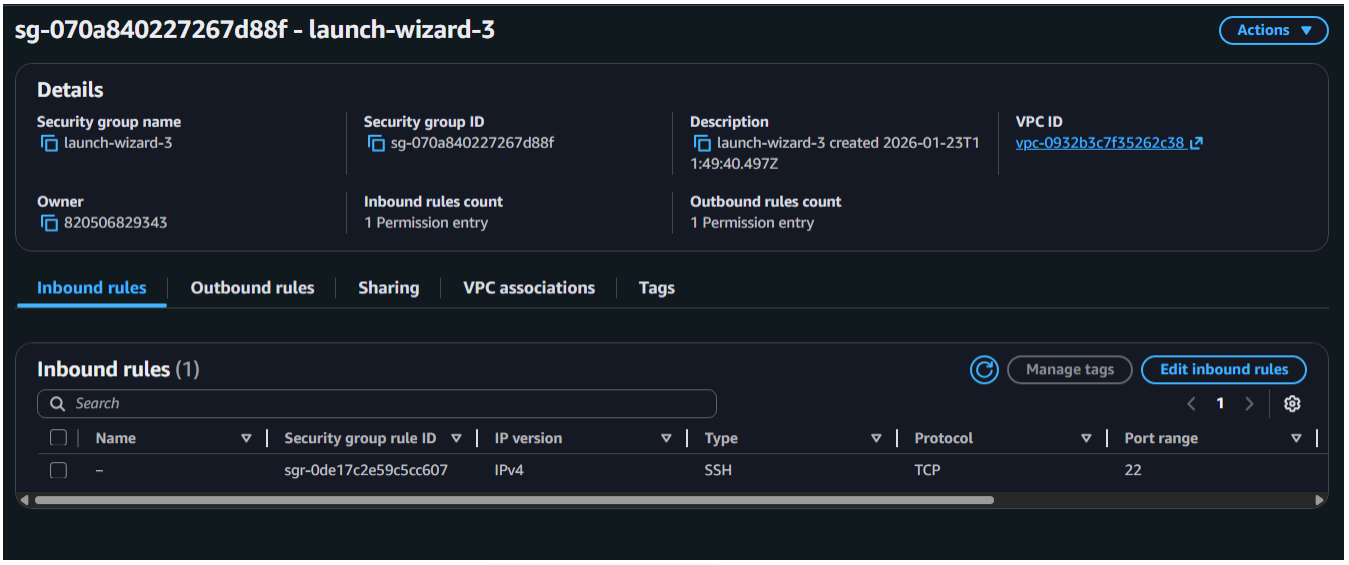
5. **Failed Test (Implicit Deny):** Attempted to access the application via browser on port **9080** before configuring rules.



Result: "This site can't be reached". Reason: The app is on 9080, but the Security Group default is to block everything.

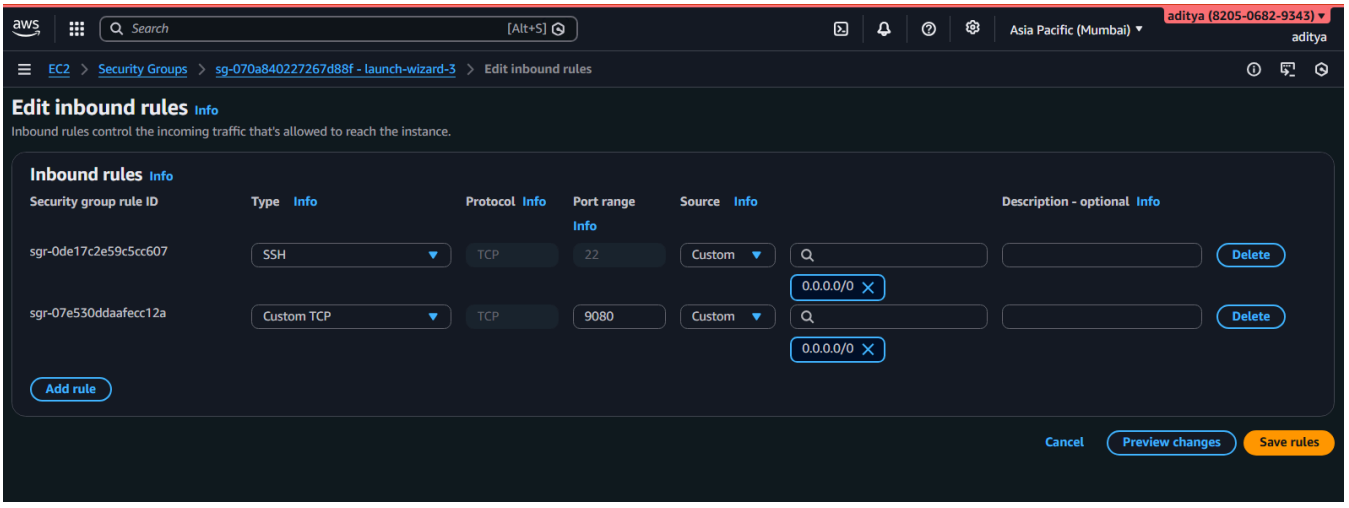
Phase 3: Configuring Security Groups

6. **Initial State:** Checked the Security Group `launch-wizard-3`. It initially only allowed SSH.



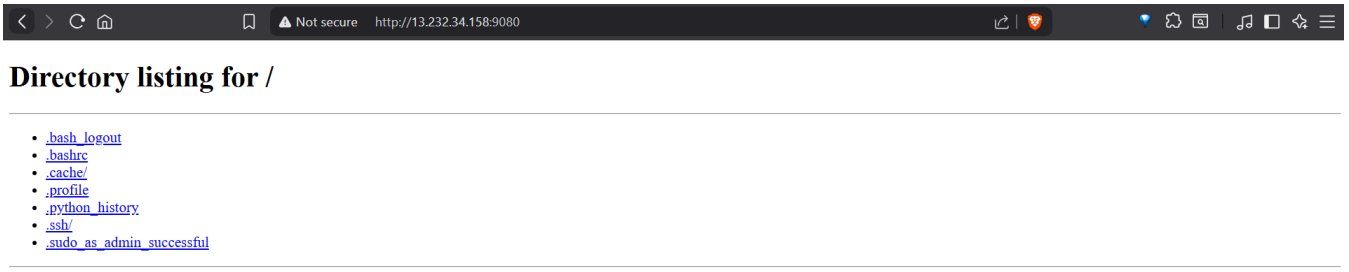
Details of the created Security Group `launch-wizard-3`.

7. **Modified Security Group:** Edited inbound rules to **Allow Custom TCP 9080** from `0.0.0.0/0`.

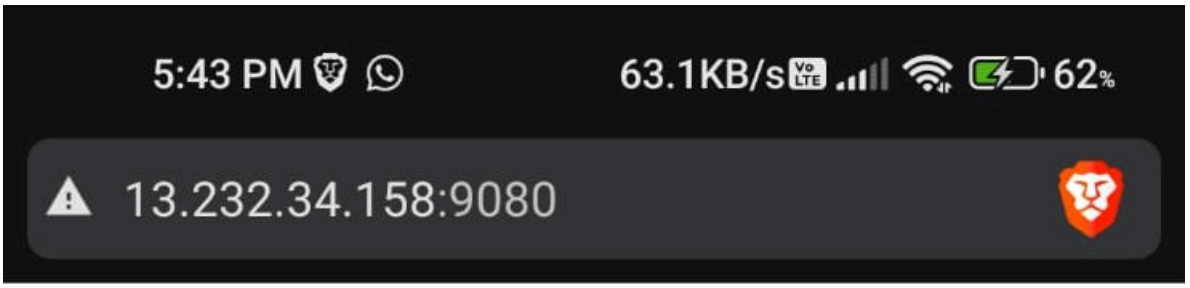


Added rule: Custom TCP, Port 9080, Source 0.0.0.0/0.

8. **Successful Test:** Accessed `http://13.232.34.158:9080` in the browser.

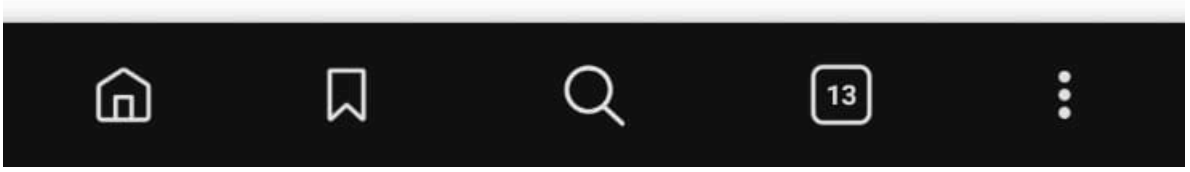


The "Directory listing" page loads, proving the SG now allows traffic.



Directory listing for /

- .bash_logout
- .bashrc
- .cache/
- .profile
- .python_history
- .ssh/
- .sudo_as_admin_successful



Verification of access via mobile device.

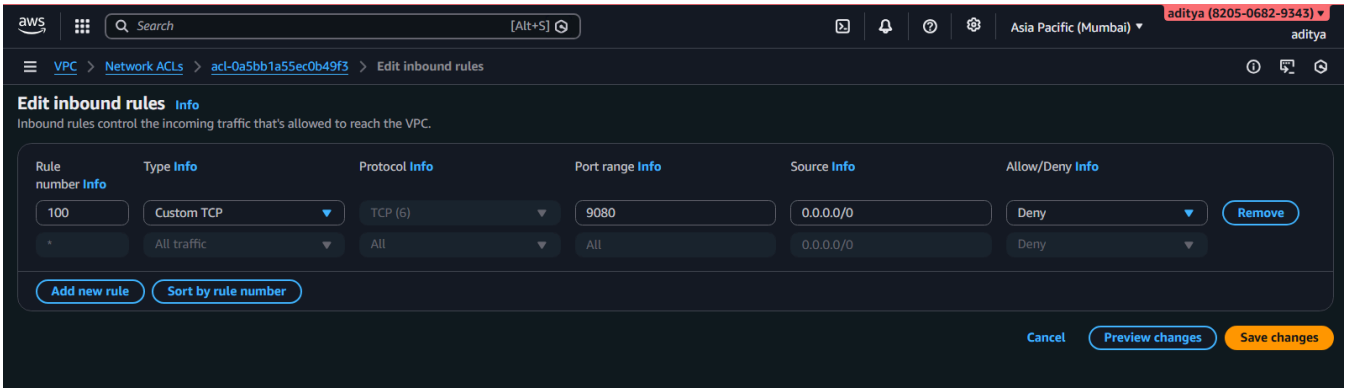
```
ubuntu@ip-10-0-10-205:~$ python3 -m http.server 9080
Serving HTTP on 0.0.0.0 port 9080 (http://0.0.0.0:9080/) ...
106.215.180.99 - - [23/Jan/2026 12:11:13] "GET / HTTP/1.1" 200 -
106.215.180.99 - - [23/Jan/2026 12:11:14] code 404, message File not found
106.215.180.99 - - [23/Jan/2026 12:11:14] "GET /favicon.ico HTTP/1.1" 404 -
-
```

Terminal logs showing the Client IP (106.215.180.99) connecting with status 200 OK.

Phase 4: Testing Network ACLs (NACL)

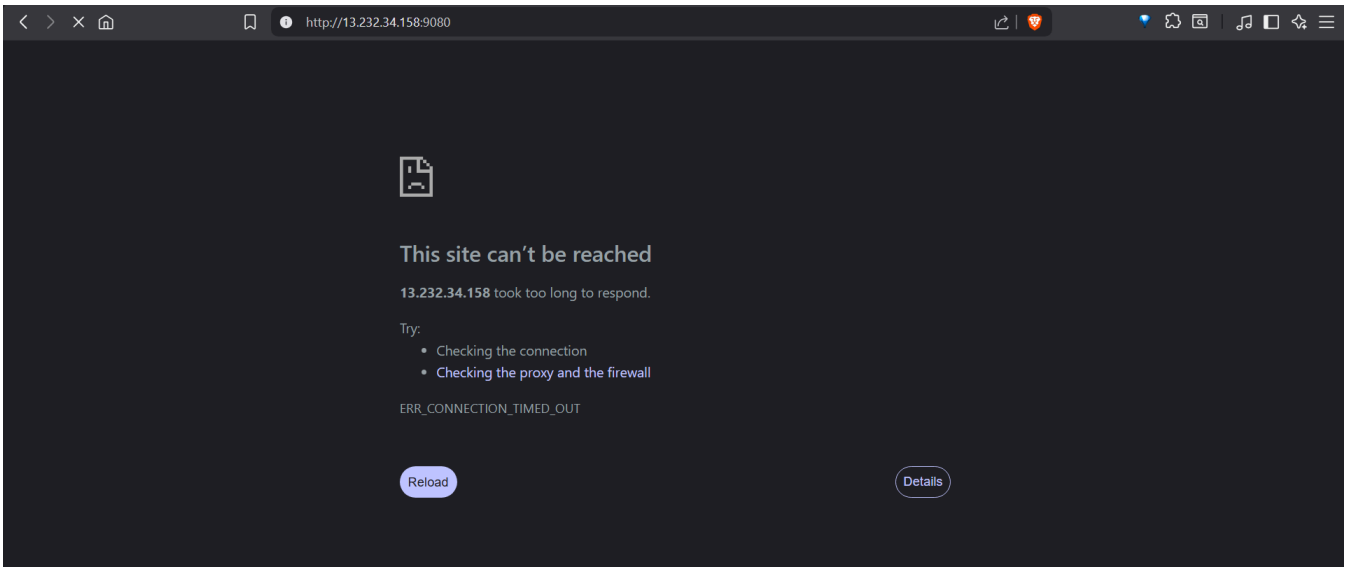
9. **Blocked Traffic via NACL:** Edited the Subnet's NACL Inbound Rules.

- Added **Rule 100: Deny** traffic on port 9080.



Explicitly denying port 9080 at the subnet level (Rule #100).

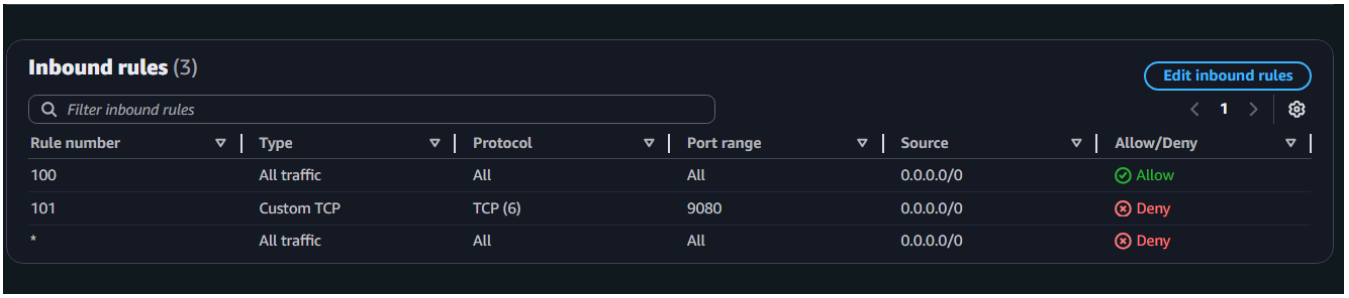
- **Result:** Browser access timed out immediately.



"This site can't be reached" - proving NACL acts before the Security Group.

10. **Tested NACL Rule Priority:** Modified NACL Inbound Rules again to test order of operations.

- **Rule 100:** Allow All Traffic (`0.0.0.0/0`).
- **Rule 101:** Deny Port 9080.



Setting up a conflict: Rule 100 (Allow) vs Rule 101 (Deny).

- **Result:** Access was **Successful**. Since Rule 100 (Allow) is evaluated *before* Rule 101 (Deny), the traffic was allowed. (Referencing previous successful connection images).

5. Testing + Results Summary

Test #	Changed Resource	Configuration	Expected Result	Actual Result	Why it happened
1	Security Group (Initial)	Default Rules (SSH only)	Timeout	**Timeout**	Port 9080 was not explicitly allowed (Implicit Deny).
2	Security Group	Added Inbound Rule: Allow `9080`	Allow	**Success**	SG allowed traffic to reach the instance.
3	NACL	Rule 100: **Deny** `9080`	Block	**Blocked**	NACL is the first line of defense; it dropped packets before they reached the SG.
4	NACL	Rule 100: **Allow All** Rule 101: **Deny** `9080`	Allow	**Success**	**Rule Priority:** AWS evaluates rules in ascending order. Rule 100 matched first, so Rule 101 was ignored.

6. Traffic Flow Explanation

1. **Client Request:** The User sends a packet to `13.232.34.158:9080`.
2. **Internet Gateway:** Routes traffic into the VPC `trafic-control-lab-vpc`.
3. **NACL (Subnet Boundary):**
   - AWS checks the Inbound Rules of `acl-0a5bb1a55ec0b49f3`.
   - It checks rules in numerical order (100, then 101, etc.). The first match determines the action (Allow/Deny).
4. **Security Group (Instance Boundary):**
   - If NACL allows, the packet reaches the EC2 network interface.
   - AWS checks `sg-070a840227267d88f`. If Port 9080 is open (as seen in `(10).png`), traffic passes.
5. **EC2 Application:** The Python server receives the request and sends a response (Log: `"GET / HTTP/1.1" 200`).
6. **Return Traffic (The "Stateful" Difference):**
   - **SG:** The response is allowed out automatically (Stateful).
   - **NACL:** The response is checked against Outbound Rules (Stateless).

7. Key Learnings / Conclusion

- **Security Groups are Stateful:** We only opened *Inbound* port 9080 in `launch-wizard-3`. We did not need to configure Outbound rules for the server to reply to the browser.
- **NACLs are Stateless:** NACLs act as a firewall for the subnet and require explicit rules for traffic in both directions.
- **NACL Rule Priority:** The test proved that **Rule Number matters**. Lower numbers are processed first. A "Deny" rule (101) is useless if an "Allow" rule (100) covers the same traffic with a higher priority.

8. Cleanup

- Terminated EC2 Instance `i-07e9a15a76ead02ff`.
- Deleted Security Group `launch-wizard-3`.
- Deleted VPC `trafic-control-lab-vpc` and associated subnets.