VPC Traffic Flow & Security

By Haroon Zaman | November 2025

Introduction

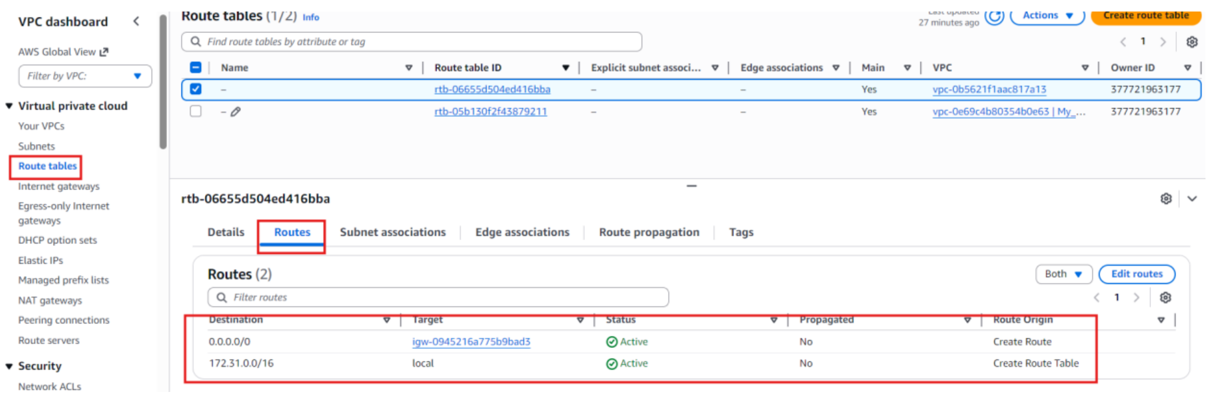
I am doing this project to understand **how traffic flows inside a VPC** and how AWS applies **security at different layers**. This stage is important because just creating a VPC is not enough — it must be **properly routed and secured** to function like a real-world cloud network.

By implementing these components, I aim to learn how AWS handles **routing decisions, instance-level protection, and subnet-level security**, which are essential skills for **cloud engineers and network professionals**.

In this project, I will perform the following tasks:

1. **Create a Route Table** – To control the direction of traffic and define how packets travel inside and outside the VPC.
2. **Create a Security Group** – To protect EC2 instances using a *stateful* firewall that monitors inbound and outbound rules.
3. **Create a Network ACL (NACL)** – To secure the **subnet level** using a *stateless* firewall that filters traffic before it reaches any resource.

Route Tables:
A route table is a set of rules that determines how network traffic is directed within a VPC. It tells resources in subnets where to send packets, whether to other subnets, the internet, or virtual private networks. By configuring route tables, we control the flow of traffic and ensure that resources can communicate securely and efficiently.



Route tables (1/2) Info — 27 minutes ago

Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC	Owner ID
–	rtb-06655d504ed416bba	–	–	Yes	vpc-0b5621f1aac817a13	377721963177
–	rtb-05b130f2f43879211	–	–	Yes	vpc-0e69c4b80354b0e63 | My_...	377721963177

rtb-06655d504ed416bba

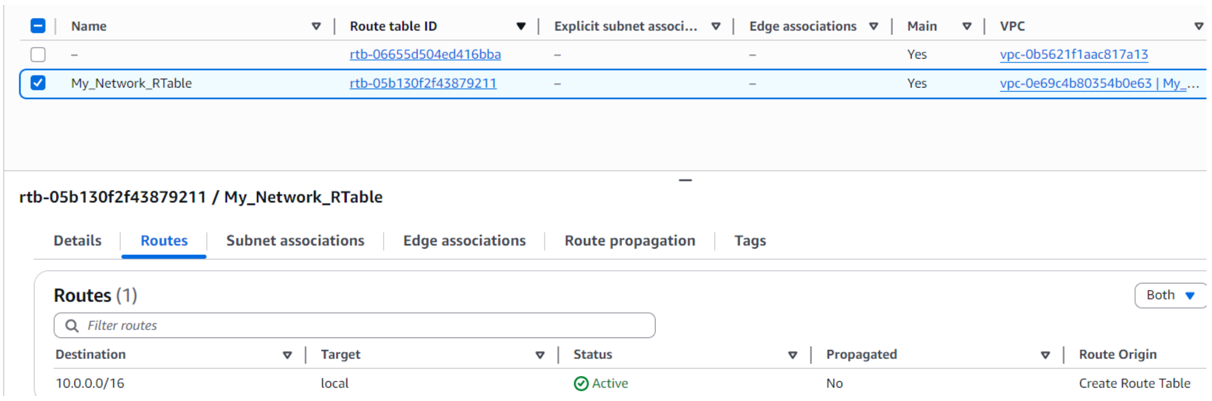
Routes (2)

Destination	Target	Status	Propagated	Route Origin
0.0.0.0/0	igw-0945216a775b9bad3	Active	No	Create Route
172.31.0.0/16	local	Active	No	Create Route Table

Since I already deployed a VPC, an Internet Gateway, and a Subnet, I moved on to create a **Route Table**. Without a route table, the subnet and gateway cannot communicate properly, and the VPC would not know how to send traffic to the internet or even within its own network. This step is necessary to **define traffic directions** and enable proper connectivity between all components.

Here’s what I observed and did:

- I clicked on **Route Tables** in the VPC dashboard. I found **two route tables**:
  - One was the **default route table**.
  - The other was **automatically created** when I created the subnet in my VPC.
- The **default route table** already had **two routes defined**:
  - `0.0.0.0/0` → `igw-...` → directs traffic to the internet through the default internet gateway.
  - `172.31.0.0/16` → `local` → manages internal traffic within the VPC.
- The CIDR block may not always be `172.31.0.0/16` — it depends on each unique VPC. What matters is that:
  - The CIDR block (e.g., `172.31.0.0/16`) covers **the entire IP range of the VPC**.
  - The target **local** means all internal traffic should be routed to the resources inside the VPC correctly.
- The **second route table** was automatically created when I set up my **custom VPC**.
  - It only had **one route** that allowed traffic **within the 10.0.0.0/16 CIDR block** to flow inside the network.
  - There was **no route to the internet gateway**, which means traffic **cannot leave the VPC yet**.



Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC
–	rtb-06655d504ed416bba	–	–	Yes	vpc-0b5621f1aac817a13
My_Network_RTable	rtb-05b130f2f43879211	–	–	Yes	vpc-0e69c4b80354b0e63 | My_...

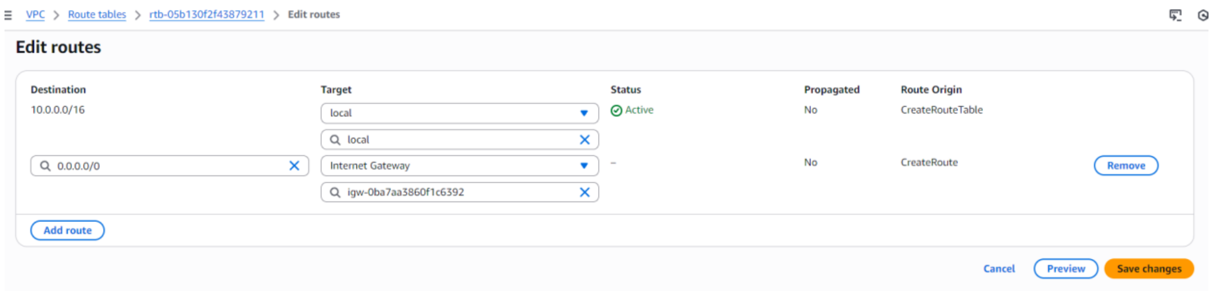
rtb-05b130f2f43879211 / My_Network_RTable

Routes (1)

Destination	Target	Status	Propagated	Route Origin
10.0.0.0/16	local	Active	No	Create Route Table

After reviewing the route tables, the first step I took was to **rename the route table** for better identification.

- I clicked the **edit (pin) icon** in the **Name** field.
- Then I renamed it to **My_Network_Rtable** so that it clearly represents the route table for my custom VPC.



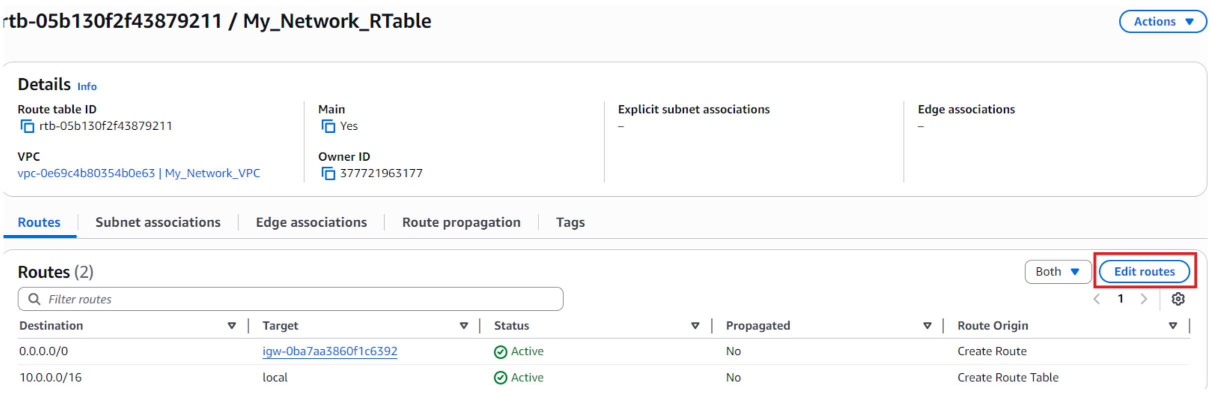
Edit routes

Destination	Target	Status	Propagated	Route Origin	
10.0.0.0/16	local	Active	No	CreateRouteTable	
0.0.0.0/0	Internet Gateway; igw-0ba7aa3860f1c6392	-	No	CreateRoute	Remove

To configure internet access, I clicked on **Edit routes** at the bottom of the selected route table. This opened a new window where I had to define the destination and target:

- For **Destination**, I entered: `0.0.0.0/0` → this means all traffic going outside the VPC.
- For **Target**, I select the option **local**, then I selected **Internet Gateway**.
- From the list, I chose **my internet gateway** that I created earlier.
- Finally, I clicked **Save changes** to apply the route.

By adding this route, I allowed the resources inside my public subnet to send traffic to the internet through the Internet Gateway.


rtb-05b130f2f43879211 / My_Network_RTable

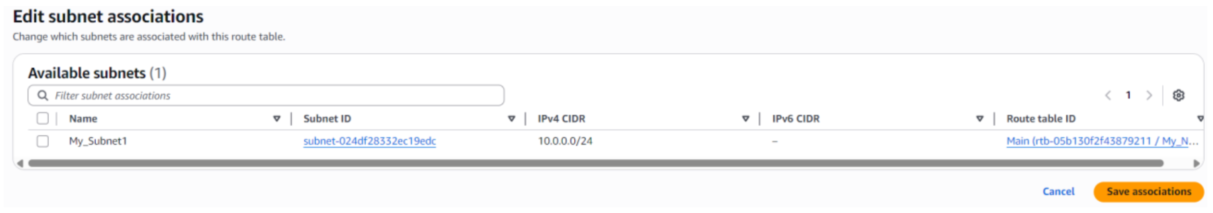
Details Info

Route table ID	Main	Explicit subnet associations	Edge associations
rtb-05b130f2f43879211	Yes	–	–

VPC	Owner ID
vpc-0e69c4b80354b0e63 | My_Network_VPC	377721963177

Routes (2)

Destination	Target	Status	Propagated	Route Origin
0.0.0.0/0	igw-0ba7aa3860f1c6392	Active	No	Create Route
10.0.0.0/16	local	Active	No	Create Route Table


Edit subnet associations
Change which subnets are associated with this route table.

Available subnets (1)

	Name	Subnet ID	IPv4 CIDR	IPv6 CIDR	Route table ID
☐	My_Subnet1	subnet-024df28332ec19edc	10.0.0.0/24	–	Main (rtb-05b130f2f43879211 / My_N...

- Next, I moved to the **Subnet Associations** section of the route table. Under **Explicit subnet associations**, I clicked on the **Edit subnet associations** button on the right side.
- I simply selected **my public subnet** from the list.
- Then I clicked **Save associations** to finalize the process.

This step was important because **even if the route table is correctly configured, it will not work until it is associated with the correct subnet**. Associating the subnet ensures that all traffic coming from that subnet will follow the rules of this route table — including the route to the internet through the Internet Gateway.

Without this association:

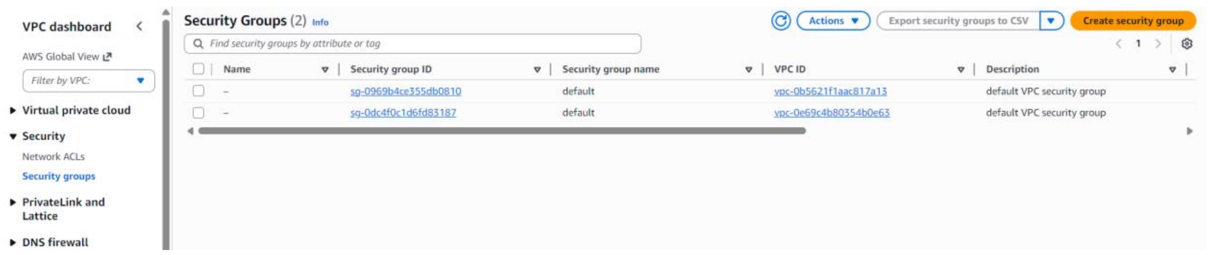
- The subnet would still remain isolated.
- The resources inside the subnet would not be able to reach the internet.
- The routing rules would never apply to the network.

So, associating the subnet is what **actually activates the route table** and enables traffic flow based on the rules I defined.

Security Groups

A security group acts like a **virtual firewall** for individual AWS resources such as EC2 instances. It controls **who can enter and who can leave**, using rules based on **IP addresses, protocols, and port numbers**. Unlike route tables or network ACLs, security groups do not attach to the whole VPC or subnet — they are applied **directly to each resource** inside the subnet.

Every resource must be associated with a security group. If no security group is manually assigned during launch, AWS automatically applies the **default security group** created when the VPC was set up.



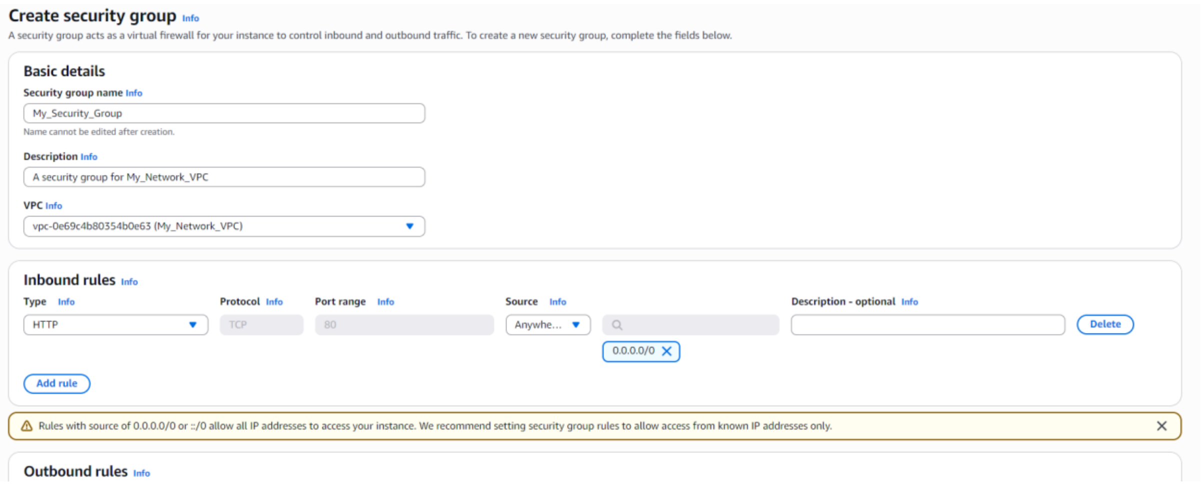
Security Groups (2) Info

Name	Security group ID	Security group name	VPC ID	Description
–	sg-0969b4ce355db0810	default	vpc-0b5621f1aac817a13	default VPC security group
–	sg-0dc4f0c1d6fd83187	default	vpc-0e69c4b80354b0e63	default VPC security group

To create my own security group, I first went to the **VPC console**. In the left side panel, under the **Security** section, I clicked on **Security Groups**.

- I noticed that there were already **two security groups** present:
  - One was the **default security group** created automatically by AWS for the default VPC.
  - The second one was **automatically generated for my custom VPC** when I created it.

This helped me understand that **AWS always ensures at least one security group exists per VPC**, even if I don't create one manually.



Create security group

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name: My_Security_Group
Name cannot be edited after creation.

Description: A security group for My_Network_VPC

VPC: vpc-0e69c4b80354b0e63 (My_Network_VPC)

Inbound rules

Type	Protocol	Port range	Source		Description - optional
HTTP	TCP	80	Anywhe...	0.0.0.0/0	

Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Outbound rules

Even though AWS automatically created a security group for my VPC, I decided to **create my own custom security group** for practice and better understanding.

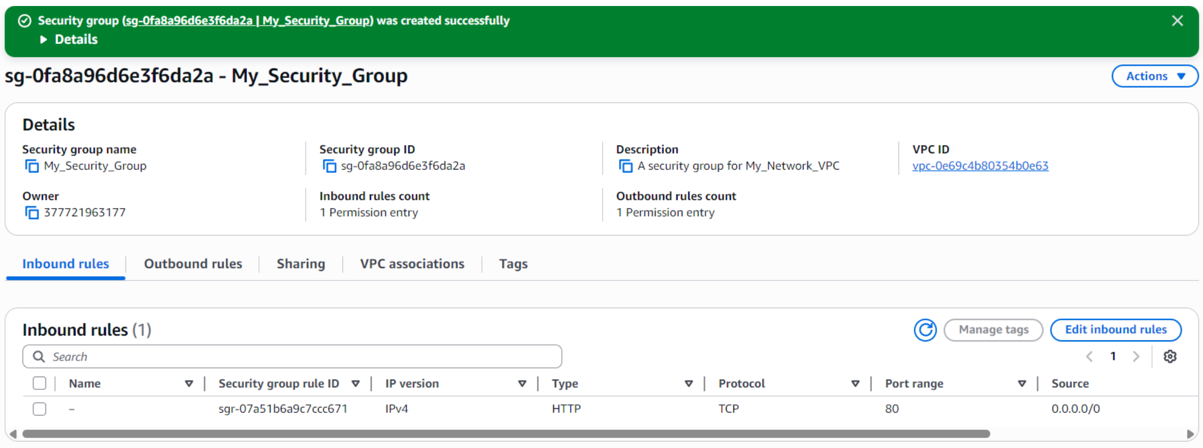
- I clicked on **Create security group**.

- I named it **My_Security_Group** and added a short description.
- Then I selected the **VPC that I created earlier** as the network for this security group.

Next, I configured the rules:

- I added **one inbound rule** allowing **HTTP traffic (port 80)** from anywhere.
    - This was just for practice, because allowing unrestricted access is considered **risky in real production environments**.
- For outbound rules, AWS had already added a **default rule** allowing **all traffic to go outside**, which means any resource can reach the internet if needed.

After reviewing everything, I clicked on **Create Security Group** to finish the setup.



Security group (sg-0fa8a96d6e3f6da2a | My_Security_Group) was created successfully

Details

sg-0fa8a96d6e3f6da2a - My_Security_Group

Details

Security group name: My_Security_Group	Security group ID: sg-0fa8a96d6e3f6da2a	Description: A security group for My_Network_VPC	VPC ID: vpc-0e69c4b80354b0e63
Owner: 377721963177	Inbound rules count: 1 Permission entry	Outbound rules count: 1 Permission entry	

Inbound rules (1)

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
–	sgr-07a51b6a9c7ccc671	IPv4	HTTP	TCP	80	0.0.0.0/0

With all settings applied, my **security group was successfully created** and is now ready to protect any resource I launch inside my VPC.

Network ACLs (NACLs)

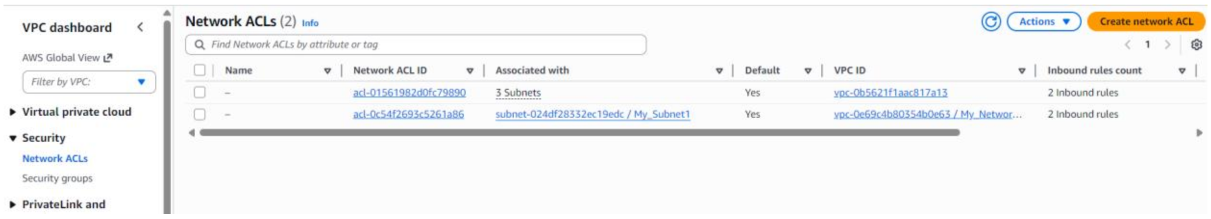
Network ACLs act as **traffic filters at the subnet level**. They work like traffic police officers standing at every entry and exit point of the subnet, inspecting every incoming and outgoing data packet. Each packet is checked against a list of rules in the ACL table before it is allowed through.

Unlike security groups, which apply to **individual resources**, Network ACLs apply to **entire subnets** — making them the **first layer of defense** before traffic reaches any resource inside the network.

Network ACLs are:

- **Stateless** — they do not remember past traffic.
- **Subnet-level firewalls** — protecting everything inside the subnet.
- Based on **numbered rule evaluation** — lowest rule number is checked first.
- Able to **allow or deny** traffic explicitly.

They work together with security groups to form a **strong layered security model** inside AWS networks.



Network ACLs (2) Info

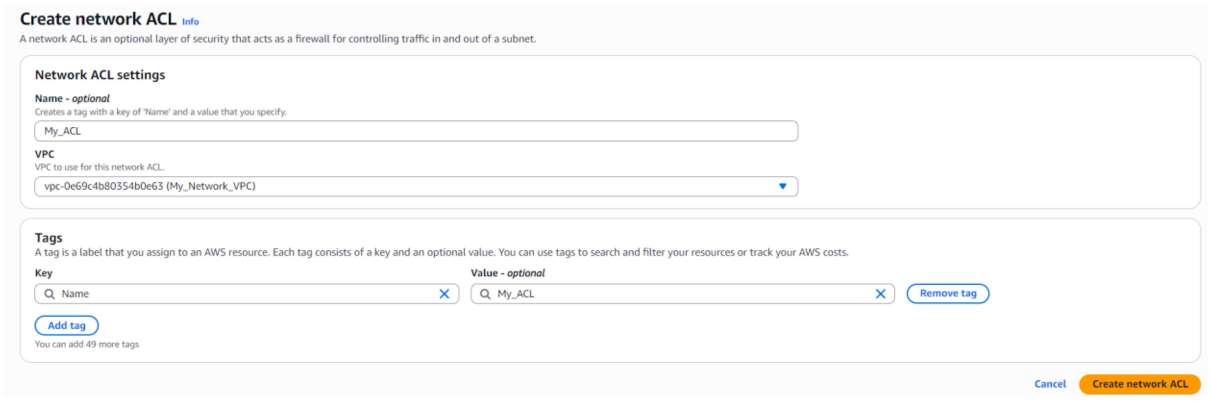
Find Network ACLs by attribute or tag

Name	Network ACL ID	Associated with	Default	VPC ID	Inbound rules count
–	acl-01561982d0fc79890	3 Subnets	Yes	vpc-0b5621f1aac817a13	2 Inbound rules
–	acl-0c54f2693c5261a86	subnet-024df28332ec19edc / My_Subnet1	Yes	vpc-0e69c4b80354b0e63 / My_Networ...	2 Inbound rules

In the same **Security** section of the VPC console, I found **Network ACLs** and clicked on it.

- I noticed that there were already **two ACLs present**:
    - One was automatically created for the **AWS default VPC**.
    - The other was automatically created when I set up **my custom VPC and subnet**.

This showed me that **AWS always provides default ACLs** for every VPC and subnet, even if I don’t create one manually.



Create network ACL Info

A network ACL is an optional layer of security that acts as a firewall for controlling traffic in and out of a subnet.

Network ACL settings

Name - *optional*
Creates a tag with a key of 'Name' and a value that you specify.
My_ACL

VPC
VPC to use for this network ACL.
vpc-0e69c4b80354b0e63 (My_Network_VPC)

Tags
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

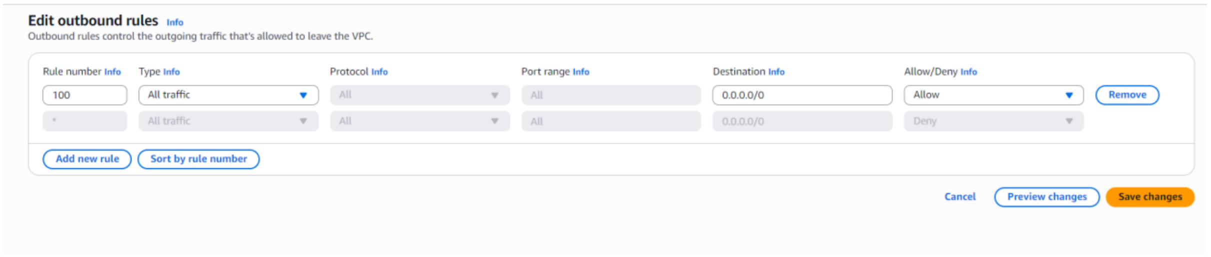
Key	Value - *optional*
Name	My_ACL

You can add 49 more tags

Even though AWS had already created a Network ACL for my VPC, I decided to **create my own custom ACL** to practice.

- I clicked on **Create ACL**.
- I gave it the name **My_ACL**.
- I selected **my custom VPC** from the list.

This created a new Network ACL that I could later configure with **custom inbound and outbound rules** to control traffic at the subnet level.



Edit outbound rules Info

Outbound rules control the outgoing traffic that's allowed to leave the VPC.

Rule number	Type	Protocol	Port range	Destination	Allow/Deny
100	All traffic	All	All	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

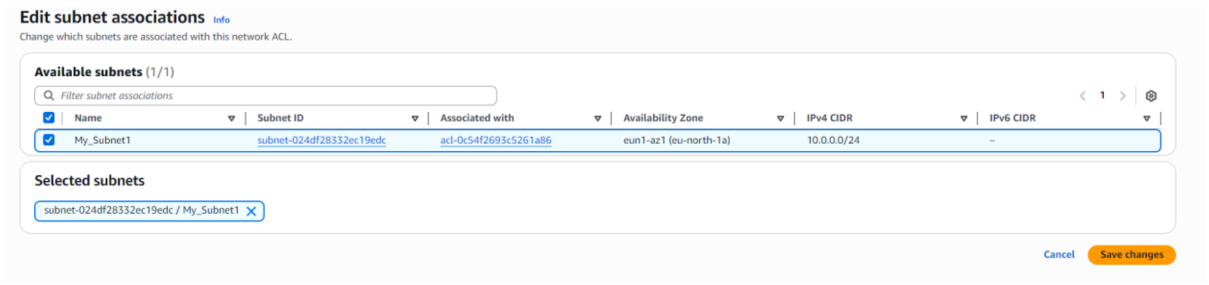
When the ACL was created, I noticed that its **default rule was to deny all traffic**. This means no data could flow in or out until I added specific rules.

I then proceeded to **add custom rules for both outbound and inbound traffic**:

- For **outbound traffic**:
    - I chose **rule number 100** — I selected a higher number to give it **lower priority**, since ACLs evaluate rules in order from lowest to highest.
    - I set the **Type** to **All Traffic**.
    - **Destination** was set to `0.0.0.0/0` to allow traffic to any IP outside the VPC.
- For **inbound traffic**:
    - Similar to outbound, I set the **Type** to **All Traffic**.
    - Here, instead of destination, I used **Source** as `0.0.0.0/0` to allow incoming traffic from anywhere.

Finally, I clicked **Save changes**.

By doing this, I ensured that my subnet could send and receive traffic while still being controlled by the ACL rules — creating a secure, functional network environment.



Edit subnet associations Info

Change which subnets are associated with this network ACL.

Available subnets (1/1)

☑	Name	Subnet ID	Associated with	Availability Zone	IPv4 CIDR	IPv6 CIDR
☑	My_Subnet1	subnet-024df28332ec19edc	acl-0c54f2693c5261a86	eun1-az1 (eu-north-1a)	10.0.0.0/24	–

Selected subnets

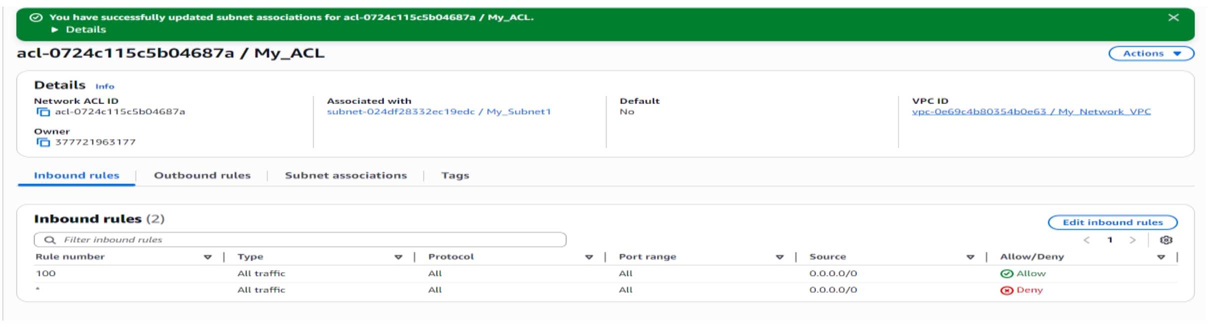
subnet-024df28332ec19edc / My_Subnet1

- On the selected ACL, I went to the **Subnet Associations** section in the bottom panel and clicked on **Edit Associations**.
- I simply selected **my subnet** to associate it with the ACL and clicked on **Save**.

This step is important because **Network ACLs only take effect when they are associated with a subnet**. Without this association:

- The ACL rules would not apply to any traffic.
- The subnet would remain unprotected at the network level.
- Inbound and outbound traffic would bypass the ACL entirely.

By associating the ACL with the subnet, I ensured that **all traffic entering or leaving the subnet** would be evaluated against the rules I configured, providing **an extra layer of security** beyond the security groups.



You have successfully updated subnet associations for acl-0724c115c5b04687a / My_ACL.

▸ Details

acl-0724c115c5b04687a / My_ACL

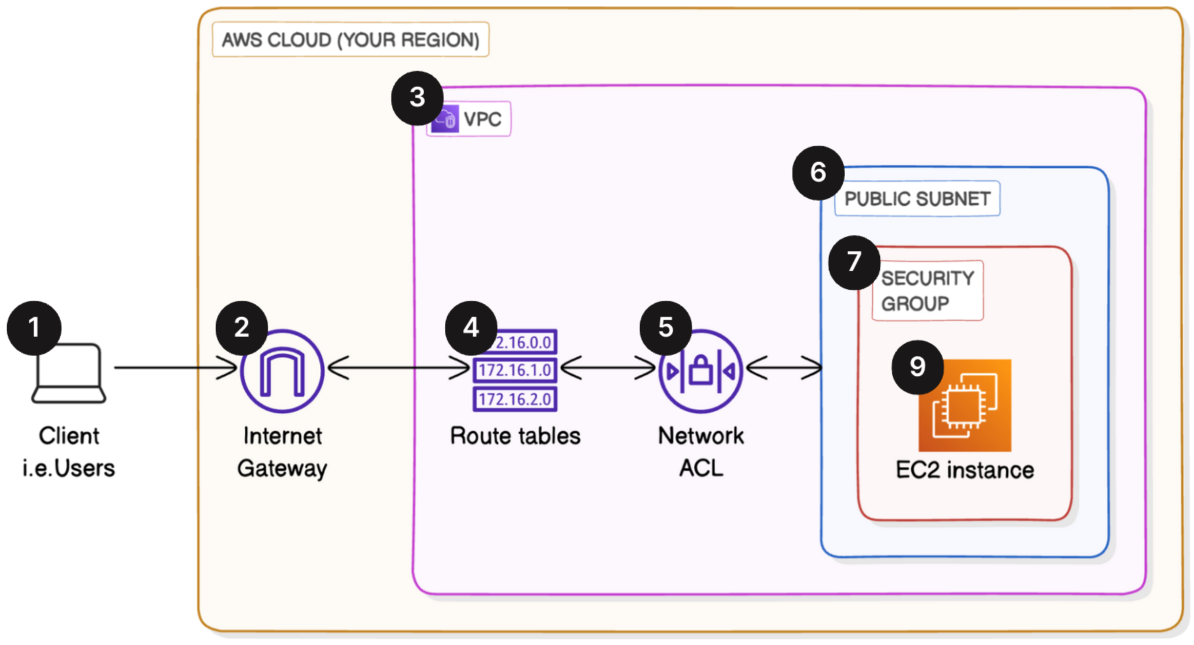
Details Info

Network ACL ID	Associated with	Default	VPC ID
acl-0724c115c5b04687a	subnet-024df28332ec19edc / My_Subnet1	No	vpc-0e69c4b80354b0e63 / My_Network_VPC

Owner: 377721963177

Inbound rules (2)

Rule number	Type	Protocol	Port range	Source	Allow/Deny
100	All traffic	All	All	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny



After completing all the steps, everything was fully set up. My VPC now had:

- A **route table** directing traffic to the internet,
- A **security group** controlling inbound and outbound traffic at the resource level, and
- A **Network ACL** providing subnet-level traffic filtering.

With these components in place, the VPC was **secure, organized, and ready for resources to communicate internally and externally**.

Conclusion

This project allowed me to gain hands-on experience with **VPC traffic flow and security in AWS**. By building on the foundation of my VPC, public subnet, and internet gateway, I was able to implement essential networking and security components to create a **functional and secure cloud environment**.

Key takeaways from the project include:

- **Route Tables:** I learned how to define routing rules to direct traffic both within the VPC and to the internet, ensuring proper connectivity for resources.
- **Security Groups:** I understood how to protect individual resources by allowing or restricting traffic based on IP, protocol, and port numbers.
- **Network ACLs:** I gained insight into subnet-level security, controlling traffic at the entry and exit points of the subnet with stateless rules.

- **Subnet Associations:** I realized the importance of linking route tables and ACLs to subnets to make the rules effective.

Overall, this project helped me **think like a cloud/network engineer**, understanding not just how to create resources, but how to **secure and manage traffic flow** in a real-world AWS environment. It reinforced the importance of layered security and proper traffic management in designing scalable, safe cloud architectures.