Beginners guide to basic networking in AWS cloud

A step by step guide how to build a standard web-application in AWS with a focus on networking

Matthias Bloemker · Follow
13 min read · Jun 23, 2024

Listen Share More

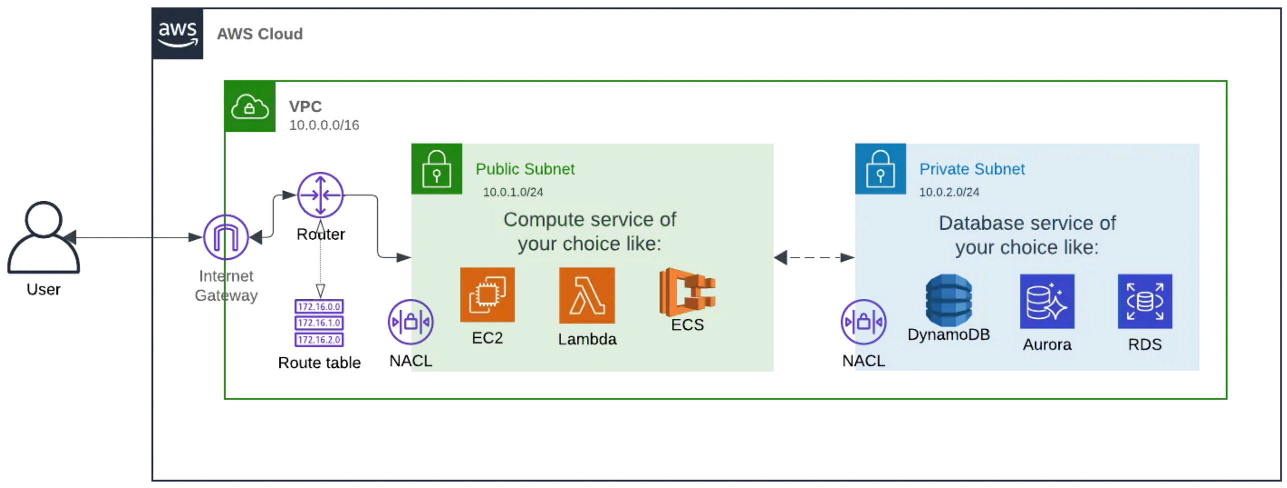
TIME TO TALK ABOUT NETWORKING IN AWS!
BEGINNERS GUIDE TO BASIC AND SECURE NETWORKING IN AWS
VPC
aws

Understanding basic networking principles within AWS are crucial for effectively deploying and managing cloud resources. Networking forms the backbone of how data flows between services, users, and the internet within your AWS Virtual Private Cloud (VPC). Despite its importance, many newcomers to AWS often struggle with networking concepts, finding the array of options and configurations daunting. In this article, we will demystify fundamental AWS networking topics, providing clarity on how to establish connectivity, secure communication channels, and optimize performance within your cloud environment. By mastering these basics, you'll be better equipped to harness the full power of AWS for your applications and infrastructure needs.

What you will find in this article

- We start with an overview of what we are going to build in this article
- Explanation of the AWS network services: What do they do? What's their use case? Are they necessary?
- Bringing the components together: Configuration of the system with in-depth explanation

So let's have a look what we are going to build in this article:



Architecture of a simple but typical AWS system with network components

The picture above shows a simple but also typical AWS architecture with all the network components added, we need to build a secure and working system within the AWS. This architecture can be used for many web services, but let's say it is an online shop, just to give you a real world example.

Let's talk about the network services

In the picture we see a lot of network components, and it might be overwhelming for newbies, but we will go through it step by step so we can get sure that you understand everything :)

Virtual Private Cloud- or in short, *VPC* is your own private part of the AWS infrastructure. Amazon VPC lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways. It is mandatory for building up your system.

Subnets (private/public)- Subnetting is, like the VPC, a basic concept of networking. In your VPC you have defined an area of IP addresses which you can split in several pieces with creating subnets. It is not mandatory to do that. You also could deploy your EC2 instance and your database just in your VPC without creating subnets before. Nevertheless, it is good practice and kind of standard. We differentiate between *private* and *public subnets.* The difference between these two lies in the internet connection. Public subnets are configured in a way that they have internet access, private subnets don't. We will see in the next section how we can configure our subnets.

Network access control list- or in short NACLs provide an additional layer of security for your VPC by controlling traffic at the subnet level. They are stateless, meaning that each rule applies independently to each request and response. I.e. you can define a rule so that only HTTP request are able to go in your public subnet and out, so you can have all your web applications in one subnet.

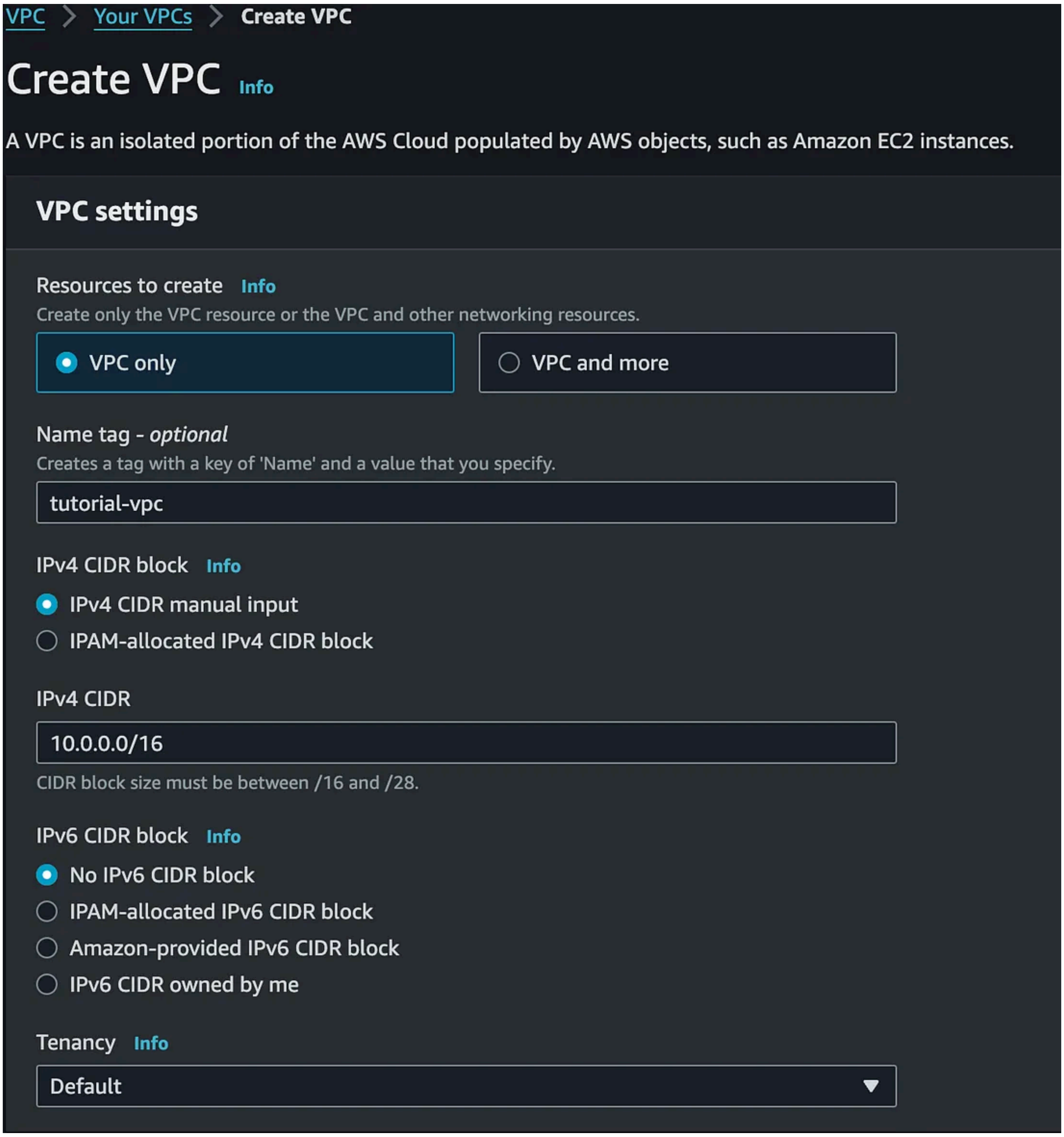
Bring the components together: Configure the network

You know all the components we are using here so we can start bringing these components together and configure it.

Step1: Create a VPC

When you have a fresh AWS account, you have the default VPC. You can of course use that for your system. Nevertheless, it makes sense to know how to create a VPC. Sometimes you come into a situation in that you want to create another VPC to isolate one system from another (i.e. if you create the AWS environment for different teams that have completely different systems)

To create your own VPC, just go type *VPC* in the search bar in the AWS management console and go to *Your VPC* → Create VPC. The CIDR block defines how many IP addresses should be available in your VPC. We do not need so much in our VPC so we can choose the minimum which are 65,536 IPs (if you want to play around with CIDR blocks I can recommend you https://cidr.xyz/). We do not need additional IPv6 CIDR blocks in our VPC. Per default all services within the VPC communicate via IPv4 addresses so will our services, therefore there is no need for running our VPC in dual-stack mode (which means it communicates via IPv4 AND IPv6). In the *tenancy* section, you can choose the *Default* (which means you share the underlying hardware with other users of AWS) and *Dedicated* (which means you have ordered a dedicated host from AWS). Cause this is just a small tutorial, and we are not rich, we are using the default here. Now, you can just add tags if you want to, but this is not mandatory. Now just create your VPC.


VPC > Your VPCs > Create VPC

Create VPC Info

A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Amazon EC2 instances.

VPC settings

Resources to create Info
Create only the VPC resource or the VPC and other networking resources.
[✓ VPC only] VPC and more

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

IPv4 CIDR block Info
[✓ IPv4 CIDR manual input]
IPAM-allocated IPv4 CIDR block

IPv4 CIDR
10.0.0.0/16
CIDR block size must be between /16 and /28.

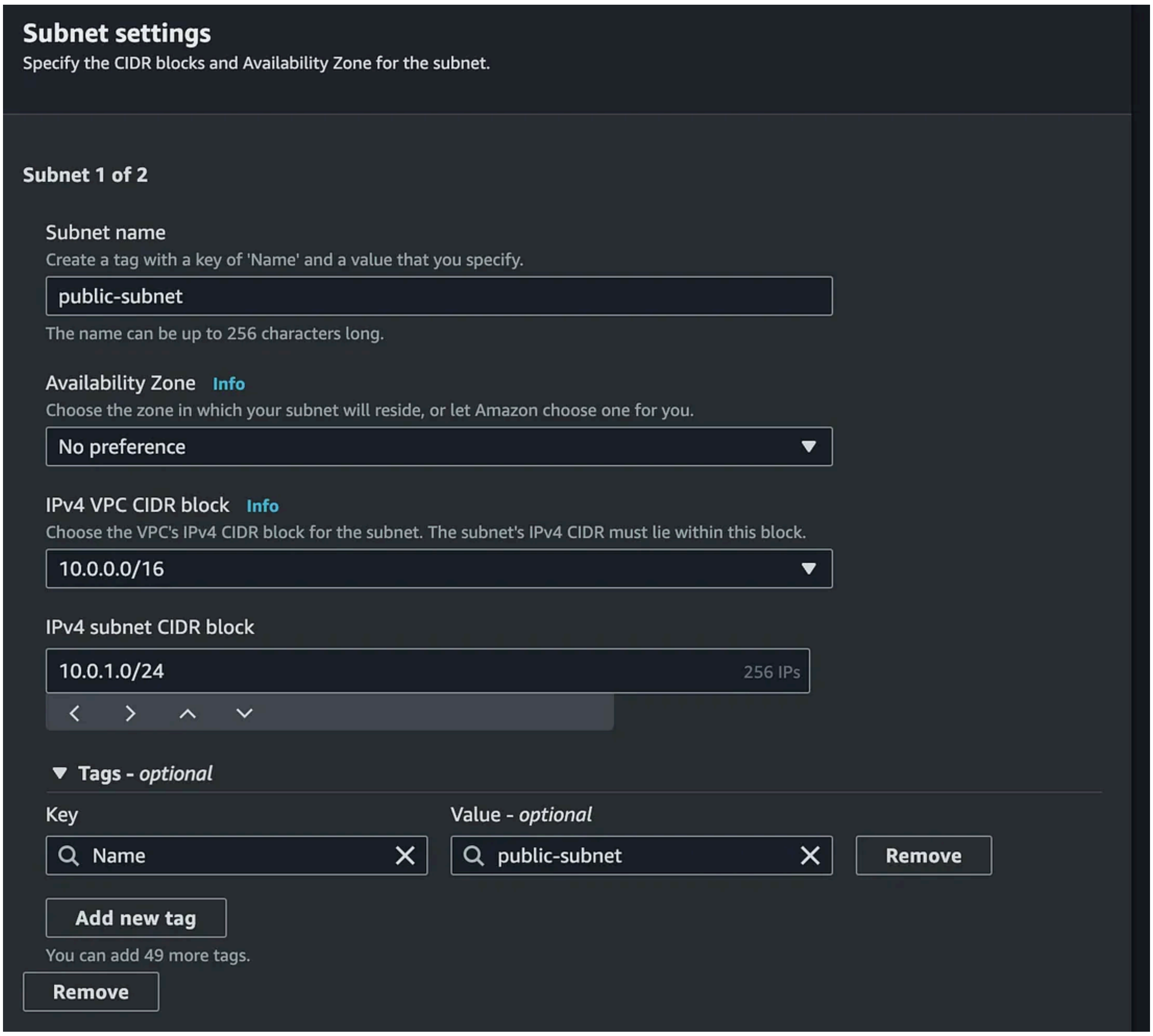
IPv6 CIDR block Info
[✓ No IPv6 CIDR block]
IPAM-allocated IPv6 CIDR block
Amazon-provided IPv6 CIDR block
IPv6 CIDR owned by me

Tenancy Info
Default

Step 2: Split your VPCs into subnets

For subnet creation, we have to go to the *Subnet* section of the VPC service in the AWS management console and choose *Create subnet*.

Foremost, you select a VPC in that the subnet should get created. Of course, we select the one we created in the step before. After choosing a nice name for your first subnet, you have to select an *Availability zone (AZ).* Unlike VPCs, subnets do not live over several AZs, they are bound to one AZ. We want to deploy our web server in Virginia, so we choose that. The interesting part is the IPv4 subnet CIDR block. The most important points are that you do not want to have too less IP addresses available, and you do not want to have the CIDR blocks of your subnets (in our case the public and the private subnet) overlapping each other. So for the beginning 256 IPs should be enough, and we give that subnet the mask 10.0.1.0/24. Now you can click on *Add new subnet* for your private own and give it the subnet mask 10.0.2.0/24 to get sure that the both subnet do not overlap. Click on *Create subnet* and your basic infrastructure is already created. Just for your information: you can not use i.e. 10.0.0.1/24 for your subnet. You would cut off the first IP 10.0.0.0 which is the network address of your subnet and this is mandatory.



Subnet settings
Specify the CIDR blocks and Availability Zone for the subnet.

Subnet 1 of 2

Subnet name
Create a tag with a key of 'Name' and a value that you specify.
public-subnet
The name can be up to 256 characters long.

Availability Zone Info
Choose the zone in which your subnet will reside, or let Amazon choose one for you.
No preference

IPv4 VPC CIDR block Info
Choose the VPC's IPv4 CIDR block for the subnet. The subnet's IPv4 CIDR must lie within this block.
10.0.0.0/16

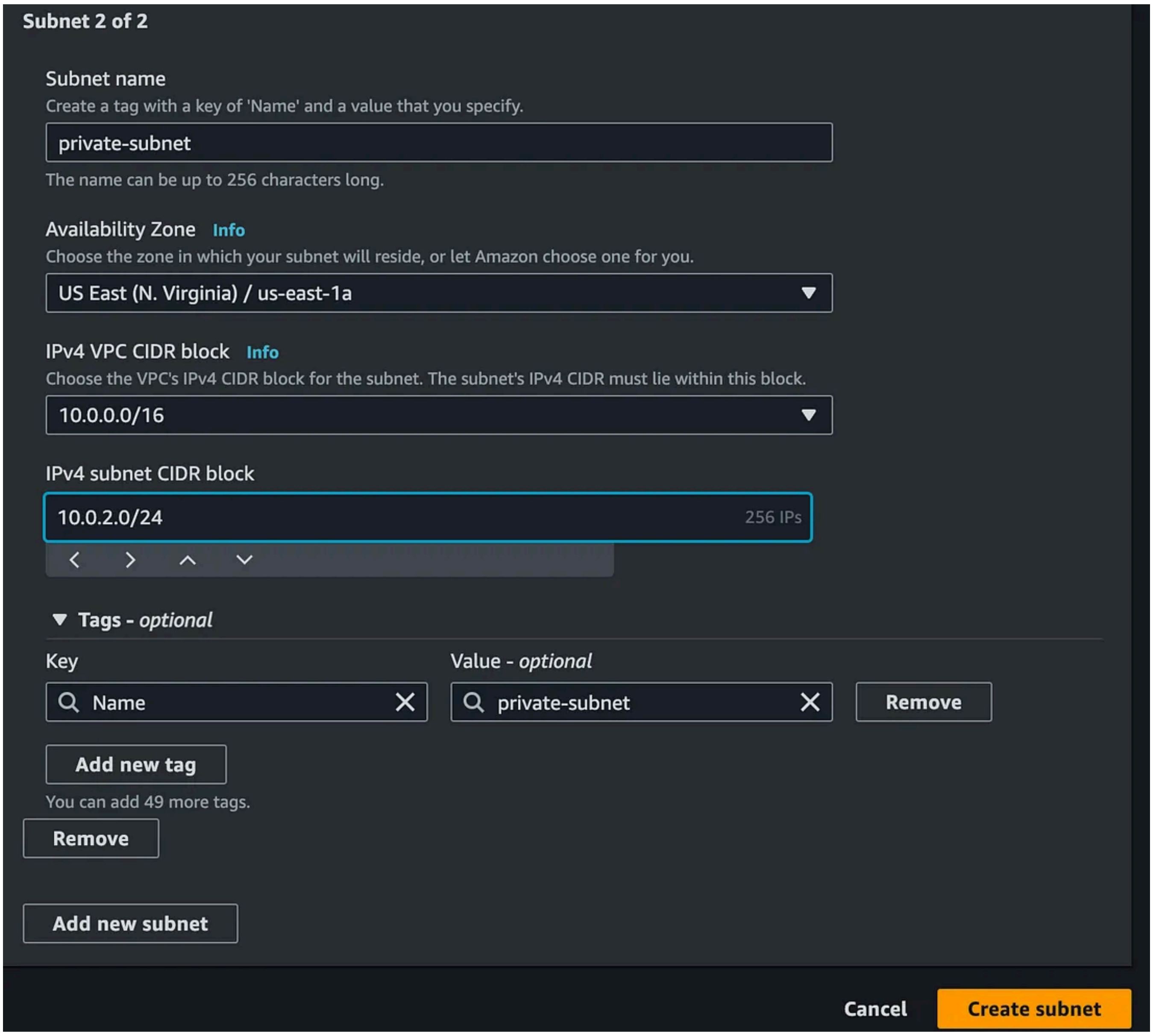
IPv4 subnet CIDR block
10.0.1.0/24 256 IPs

Tags - *optional*

Key	Value - *optional*	
Name	public-subnet	Remove

Add new tag
You can add 49 more tags.
Remove

Create public subnet


Subnet 2 of 2

Subnet name
Create a tag with a key of 'Name' and a value that you specify.
private-subnet
The name can be up to 256 characters long.

Availability Zone Info
Choose the zone in which your subnet will reside, or let Amazon choose one for you.
US East (N. Virginia) / us-east-1a

IPv4 VPC CIDR block Info
Choose the VPC's IPv4 CIDR block for the subnet. The subnet's IPv4 CIDR must lie within this block.
10.0.0.0/16

IPv4 subnet CIDR block
10.0.2.0/24 256 IPs

Tags - *optional*

Key	Value - *optional*
Name	private-subnet

You can add 49 more tags.

Create private subnet

Step 3: Make your public subnet public

Just because you named your one subnet public, it is not public at all. You could now deploy your EC2 instance in this subnet, but it would not have internet access.

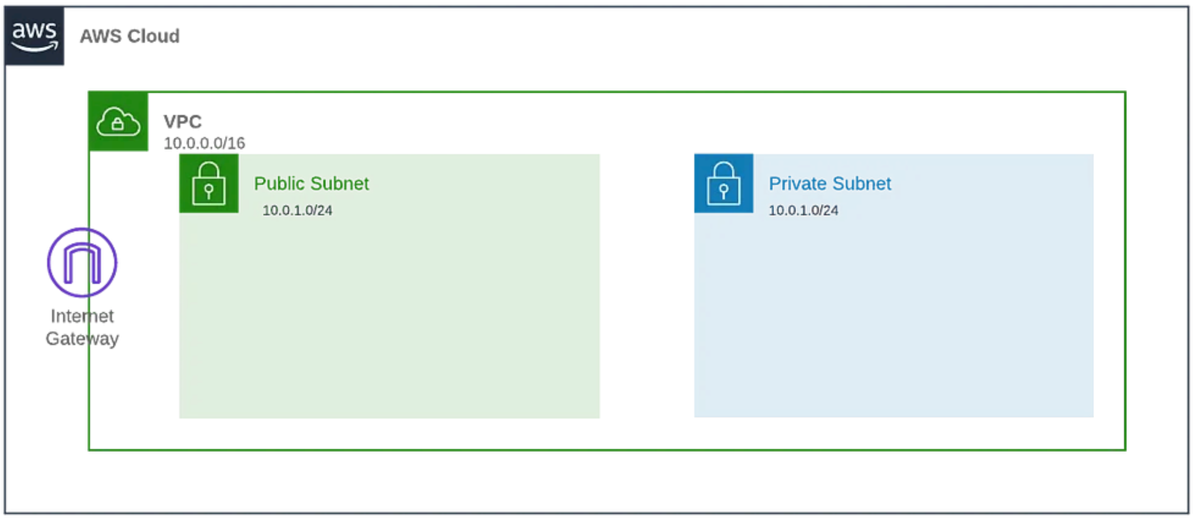
Enable auto IPv4 assignment

First, the services within the subnet need a public IP address, otherwise it would not be possible to reach these services. To do that automatically, you need to enable auto assignment of public IPv4 addresses within your subnet. To do that go to your subnet overview in the AWS console, mark the public subnet, click on actions, edit the subnet settings and then enable the auto IPv4 assignment. Create the internet gateway

Now it is time to create the *internet gateway.* From the section before, you already know what this resource is good for. You can create it from your AWS management console:*VPC → Internet gateways → Create internet gateway.* Just create the internet gateway (you can not do something wrong here). Nevertheless, the internet gateway is just flying around in your account. To use it with your VPC, you need to attach it to that. To do that, go to your internet gateway, go to Actions and then *Attach to VPC.* Choose your VPC and you're done.

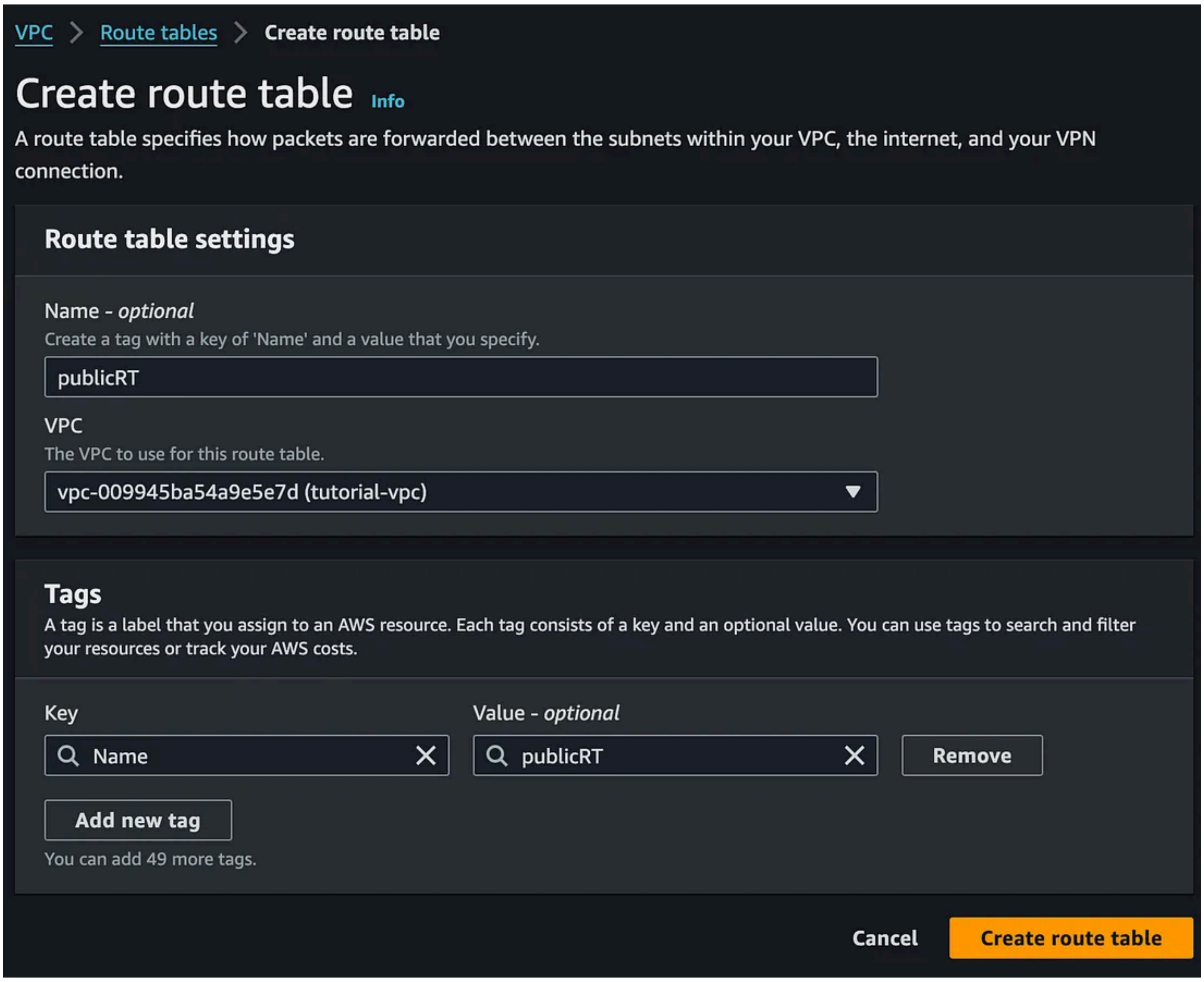
Let your public subnet send and receive traffic via the internet gateway

Let's have a look what we have created so far. We have created our own VPC and defined 2 subnets within this VPC. We enabled the automatic public IPv4 assignment for our public subnet. After that, we created the internet gateway and attached it to the VPC so it is usable for the VPC. So the interim status looks like this:



Interim status of our system configuration

Nevertheless, there is one point missing. In the AWS world and in networking in general, the routing table is responsible to coordinate the communication inside a network. In the AWS world, our network is the VPC. So the routing table lives on the VPC level and coordinates all the traffic between the components. We want our public subnet to use the internet gateway to communicate to the outer world, therefore we have to configure a rule for that. To do that, you can go in your AWS management console to VPC → Route tables. You may find already the default route table there. The default route table gets created when you create your VPC and is in general valid for all subnets, therefore it is good practice to create a new route table for your use case and leave the default table untouched to minimize side effects. Now click on *Create route table* and attach it to your VPC:



VPC > Route tables > Create route table

Create route table Info

A route table specifies how packets are forwarded between the subnets within your VPC, the internet, and your VPN connection.

Route table settings

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

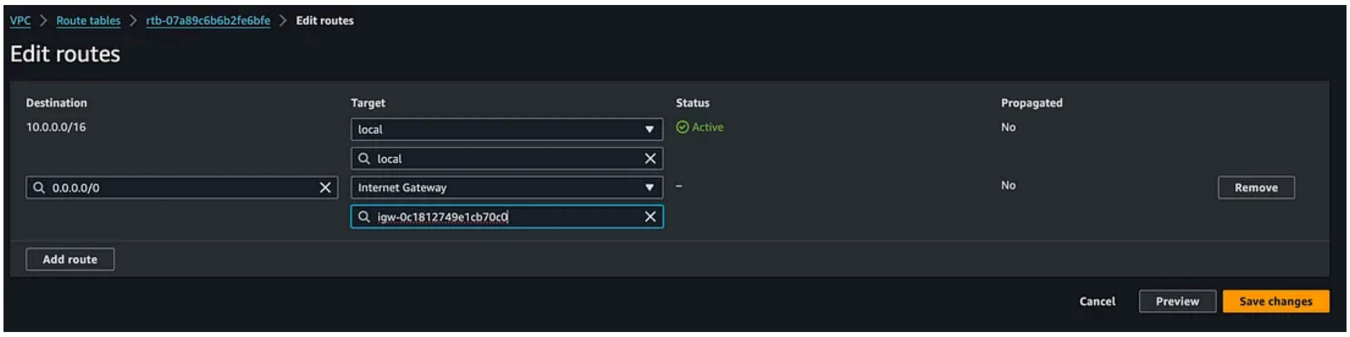
VPC
The VPC to use for this route table.
vpc-009945ba54a9e5e7d (tutorial-vpc)

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	publicRT

You can add 49 more tags.

After creating the route table for your public subnet, you can add a route to that. For that, click on *edit routes* and add the following route:



VPC > Route tables > rtb-07a89c6b6b2fe6bfe > Edit routes

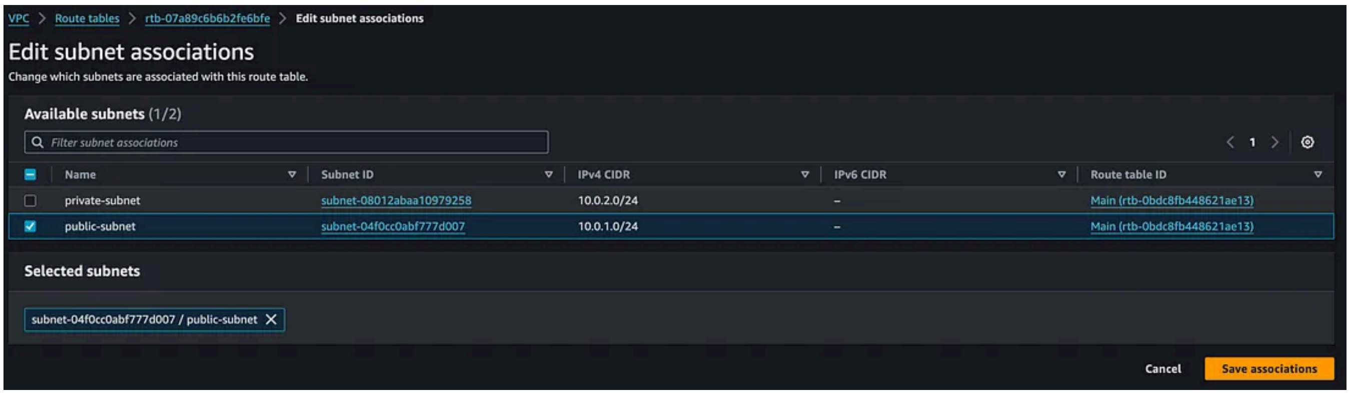
Edit routes

Destination	Target	Status	Propagated
10.0.0.0/16	local	Active	No
0.0.0.0/0	Internet Gateway igw-0c1812749e1cb70c0	-	No

Route table configuration for the public subnet route table

Maybe you ask yourself what this means, but no worry, we will go through it. Let's start with the first line, which is the default configuration of every new created route table. The destination is 10.0.0.0/16 and the target is local, correct? Maybe you can remember after this quiet long article that we created our VPC with the IP range of 10.0.0.0/16. Which basically means that every request that gets fired inside our VPC to one of these addresses inside the range will get routed locally (inside the VPC) and does not go out to the internet or to any other VPC. This route makes it possible that our public subnet (IP range: 10.0.1.0/24) can communicate with our private subnet (IP range: 10.0.2.0/24) and with every other subnet inside our VPC if we created more. On the other hand, 0.0.0.0/0 defines the whole IP spectrum, so every data transfer which does not have a destination address within 10.0.0.0/16 will get routed to our internet gateway which means to the outside world.

After editing the route table, you can now attach the route table to our public subnet (Go to your public route table detail view and click on *Subnet association → Edit subnet associations* and add your public subnet)



VPC > Route tables > rtb-07a89c6b6b2fe6bfe > Edit subnet associations

Edit subnet associations

Change which subnets are associated with this route table.

Available subnets (1/2)

Filter subnet associations

	Name	Subnet ID	IPv4 CIDR	IPv6 CIDR	Route table ID
☐	private-subnet	subnet-08012abaa10979258	10.0.2.0/24	–	Main (rtb-0bdc8fb448621ae13)
☑	public-subnet	subnet-04f0cc0abf777d007	10.0.1.0/24	–	Main (rtb-0bdc8fb448621ae13)

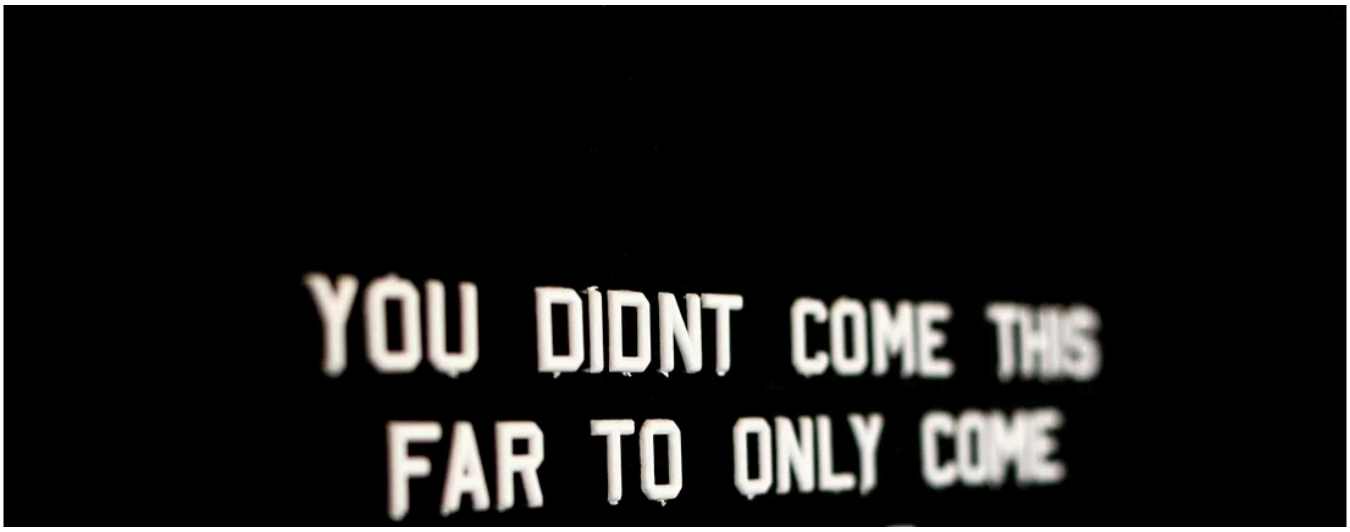
Selected subnets

subnet-04f0cc0abf777d007 / public-subnet

Associate the route table to your public subnet

Your public subnet is now able to receive any requests and send responses to the World Wide Web. Congratulations! You could now deploy your services in your subnets and everything would work like charm 😎

But:



So let’s make it secure…

Step 4: Let’s care about our private subnet

Configure the routing for the private subnet

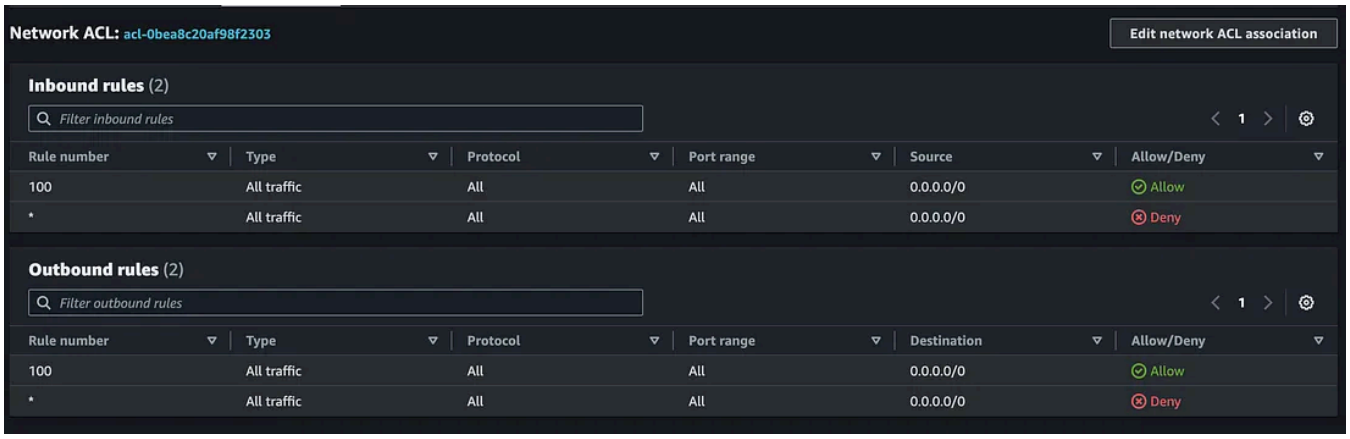
We are still missing the configuration of our private subnet. If you now go to your private subnet and take a look at the route table configuration, you will see that this subnet is already associated with one of the main route tables living in your VPC. This is your default route table (we already talked about in the previous section) and has just one route that routes every traffic going to IP addresses within your VPC to local, which basically means, that your private subnet is not able to communicate to the outside world. You may think right now: Well the subnet does not have a route to the outside world, this is actually what we want. The private subnet has a route to local, so the traffic to the public subnet will get routed correctly, this is actually what we want. So, do we actually need to do something here? No. At least not in the routing configuration.

Flashback: *Can you remember the best practice mentioned before that we should not edit the default routing table because it could resolve in side effects? This is the reason. The default routing table is automatically associated to every subnet, so I want to be sure that* *there is just this one route in the routing table and no routes, i.e. to the internet gateway which could end up in serious security risks.*Restrict the access to the private subnet

We do not need to edit the route configuration. Nevertheless, we want to limit the access to this private subnet. We want to deploy a database in it, that should be able to communicate with a computing service inside our public subnet, but we do not want any other service from any other subnet to access our private subnet for security reasons.

Remember*: best practices concerning security in AWS is the* ***least privilege principle****. Referring to subnet security, this means: Give the subnet all rights it needs for doing its work, block everything else.*

We can meet this requirement with *Network Access Control Lists,* or in short: *NACL*. You can find your NACL rules if you navigate to your private subnet and then under *Network ACL*. Your subnet should have the default NACL rules that should look like the following. Every subnet must have at least one NACL.



Network ACL: acl-0bea8c20af98f2303

Edit network ACL association

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

Outbound rules (2)

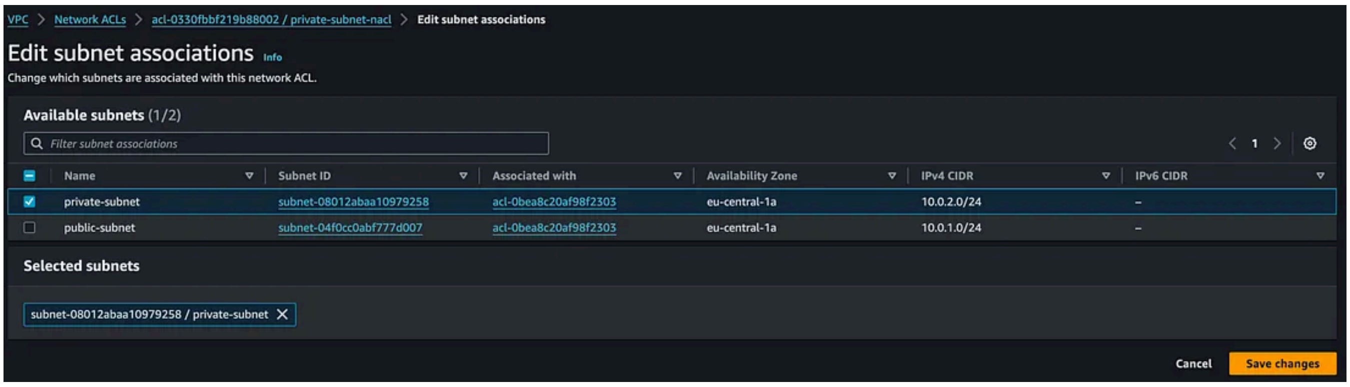
Rule number	Type	Protocol	Port range	Destination	Allow/Deny
100	All traffic	All	All	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

Default NACL rules: Allow everything

With NACL rules, you can either explicitly allow or explicitly deny some sort of traffic. The rules will be evaluated in ascending order. So the rule with number 100 has a higher prioritization than the rule with number 200. What you see in the default NACL rules is basically allowing everything (rule with number 100). The *-rule is a mandatory rule that will hit if there is no other rule before that matches. Because we have rule number 100 that allows all traffic, the *-rule will never get used in this default set up. NACLs are stateless, which means, you have to adjust the inbound rules and the outbound rules. If, i.e., you allow incoming HTTP traffic but do not add a corresponding outbound rule, the HTTP request will reach the service inside your subnet, but it won't be able to answer via HTTP.

But now we want to restrict this access. There is no need to leave our subnet open for all traffic and take the risk for unauthorized access. So let’s go over to work…

Foremost, we should create an own NACL for this subnet. To do that, go to VPC → security → Network ACLs. Create a new NACL for your private subnet. After creation, we can associate the NACL with our subnet (you can do that under *Edit subnet association*)


VPC > Network ACLs > acl-0330fbbf219b88002 / private-subnet-nacl > Edit subnet associations

Edit subnet associations Info

Change which subnets are associated with this network ACL.

Available subnets (1/2)

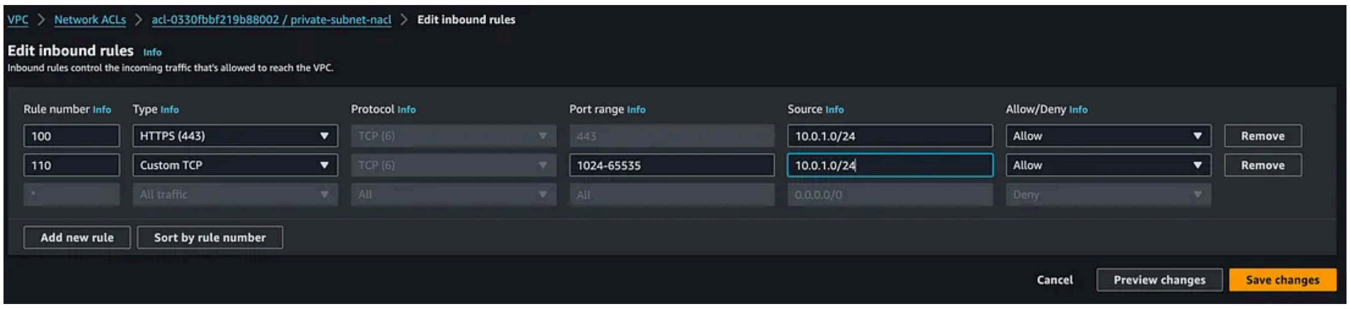
	Name	Subnet ID	Associated with	Availability Zone	IPv4 CIDR	IPv6 CIDR
✓	private-subnet	subnet-08012abaa10979258	acl-0bea8c20af98f2303	eu-central-1a	10.0.2.0/24	–
☐	public-subnet	subnet-04f0cc0abf777d007	acl-0bea8c20af98f2303	eu-central-1a	10.0.1.0/24	–

Selected subnets

subnet-08012abaa10979258 / private-subnet

Associate the NACL with the private subnet

Now we can add rules to our list. It makes sense to limit the access to the subnet based on our needs. In general, we want to access our database from our public subnet via HTTPS:


VPC > Network ACLs > acl-0330fbbf219b88002 / private-subnet-nacl > Edit inbound rules

Edit inbound rules Info

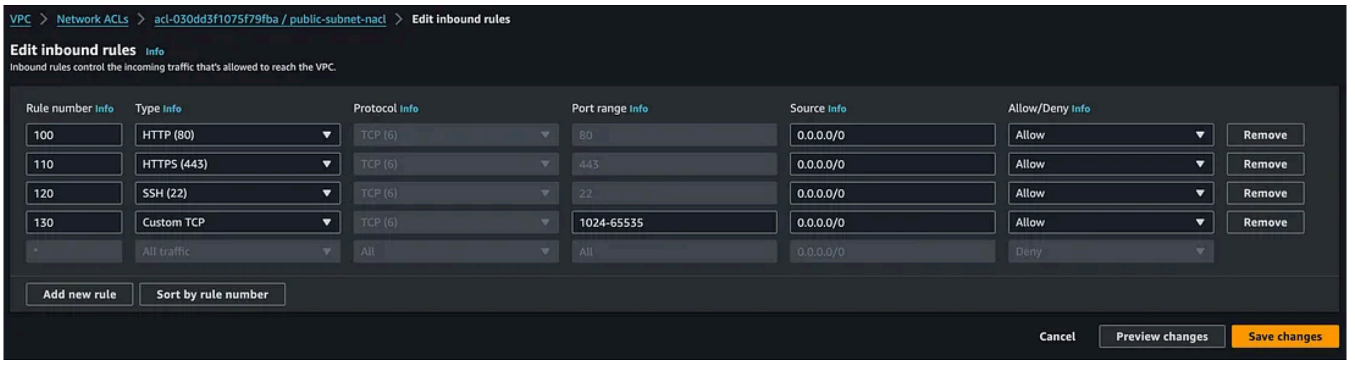
Inbound rules control the incoming traffic that's allowed to reach the VPC.

Rule number	Type	Protocol	Port range	Source	Allow/Deny
100	HTTPS (443)	TCP (6)	443	10.0.1.0/24	Allow
110	Custom TCP	TCP (6)	1024-65535	10.0.1.0/24	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

With this rule, we allow HTTPS traffic (TCP over port 443) to our private subnet from our public subnet (source: 10.0.1.0/24). The rule number 110 allows the ephemeral ports, which are needed to establish a connection with services (for further information: https://www.coursera.org/articles/ephemeral-ports). With the *-rule, every other traffic will get denied. Add a corresponding rule to the outbound rules as well, and the NACL configuration for our private subnet is done.

Step 5: Secure our public subnet

Yep, our public subnet could need a small restriction as well. It is public so we should not restrict the IP range, but we definitely should not leave it wide open. Let’s assume we would like to deploy an EC2 instance in our public subnet, then our configuration should look like this:


VPC > Network ACLs > acl-030dd3f1075f79fba / public-subnet-nacl > Edit inbound rules

Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the VPC.

Rule number Info	Type Info	Protocol Info	Port range Info	Source Info	Allow/Deny Info	
100	HTTP (80)	TCP (6)	80	0.0.0.0/0	Allow	Remove
110	HTTPS (443)	TCP (6)	443	0.0.0.0/0	Allow	Remove
120	SSH (22)	TCP (6)	22	0.0.0.0/0	Allow	Remove
130	Custom TCP	TCP (6)	1024-65535	0.0.0.0/0	Allow	Remove
*	All traffic	All	All	0.0.0.0/0	Deny	

NACL rules for inbound traffic for our public subnet

We should accept HTTP and HTTPS for our web server, of course the ephemeral ports and, if we want to ssh into our EC2 instance, we should also allow ssh. Adjust the outbound traffic table corresponding and your public subnet is good to go.

Conclusion

So what we have achieved in this tutorial? We created our own VPC, split it in subnets, configured the routing and after that we secured our subnets with NACLs. This is how a modern and secure network infrastructure looks like.

The next, not networking related steps, would be to fill this VPC with life. Deploy your web server in your public subnet, your database in your private subnet, configure Route53 so your server is reachable from a URL and maybe use a load balancer for scalability of your EC2 instances. Or maybe you want to go a serverless way with AWS Lambda. It is up to you.

I know, mastering networking is quite hard when you come from plain software development, but especially for you guys I made this tutorial. I hope you enjoyed it, and you feel way more comfortable building VPCs and network structures in AWS. Check out my profile for more helpful AWS articles.

AWS Aws Networking Aws Vpc Cloud Computing Cloud Security