High Performance and Distributed Computing for Big Data

Unit 3:Cloud Systems and Big Data Management

Session 6, 7 & 8 - AWS Networking

Francesc Solsona francesc.solsona@udl.cat

Universitat Rovira i Virgili and Universitat de Lleida

Recap

Networking concepts we already met

Throughout the course, we have already been interacting with different networking concepts. Here are some of them:

- We have been **connecting to** EC2 instances using SSH.
- We have been using the **public IP** address of the EC2 instances.
- We have been using the **DNS** name of the EC2 instances.
- We have been using different **ports** to launch different services

Networking concepts we already met

Although we have been working with these concepts, we haven't really discussed them in detail. Here are some questions that we can ask ourselves:

- Why do we sometimes specify **which port to use** when connecting to a service?
- Why do we use the public IP address of an EC2 instance instead of the private IP address?
- Why can we use the DNS name and the public IP address **interchangeably**?
- What is the **difference between a public and a private** IP address?
- Why is the **IP address changing** every time we restart the lab?

Networking basics

What is an IP address?

An **IP address** is a unique identifier for a device. The same way a street address identifies a house, an IP address identifies a device on a network.

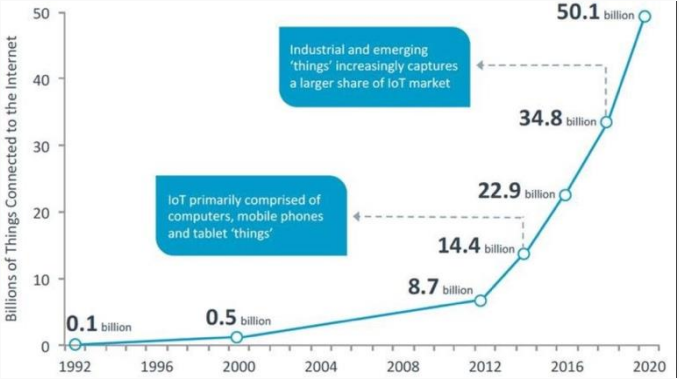
What is an IP address?

An **IP address** is a unique identifier for a device. The same way a street address identifies a house, an IP address identifies a device on a network.

- An IP address is a 32-bit number that is divided into four 8-bit numbers separated by dots. For example, `1.2.3.4` or `52.4.75.36`.
- Every one of these 8-bit numbers is called an **octet** and can have a value between 0 and 255.
- All IP addresses in the world have a value between 0.0.0.0 and 255.255.255.255 which makes up a total of 4,294,967,296 possible IP addresses (2^32).

So we have around four billion IP addresses. This might seem like a lot, but **it is not enough to give an IP address to every device in the world**.

What is an IP address?



How do we solve this? Any idea?

Private vs Public IP addresses

To solve the problem of not having enough IP addresses, we use **private** and **public** IP addresses.

- **Private IP addresses** are used to identify devices within a network. They are not unique and can be reused in different networks.
- **Public IP addresses** are used to identify devices on the internet. They are unique and can't be reused.
- **Address Ranges** are reserved for private and public IP addresses. IPs that fall into `10.x.x.x, 172.16.x.x - 172.31.x.x, 192.168.x.x` are reserved for private IP addresses.

Private vs Public IP addresses

Think of it as a hotel. The room number is the **private IP address** and the hotel street address is the **public IP address**.

- The room number is unique within the hotel, but not unique across all hotels.
- The hotel street address is unique and can't be reused.
- The hotel street address is used to identify the hotel from the outside world, while the room number is used to identify the room within the hotel.

DNS

The **Domain Name System (DNS)** is a system that translates domain names to IP addresses. Remember during class when I said you can use the DNS name or the public IP address to connect to an EC2 instance? A DNS name or a domain name is a human-readable name that references an IP address.

If we go back to the hotel analogy, the hotel name is the **DNS name** and the hotel street address is the **public IP address**.

To know to which IP address a domain name corresponds, we use a **DNS server**. This server is like a phone book that translates domain names to IP addresses.

Ports

Once we know the address of a device, whether we already knew the IP address or the DNS name, we need to know **which service** we want to connect to. So imagine the EC2 machine we configured with jupyter notebook. That machine has at least two services running: the SSH service (port 22) and the Jupyter notebook service (we used port 8888).

When we connect to the EC2 machine through SSH, we don't need to specify the port because the default port for SSH is 22, but we could if we wanted.

```
ssh -i ~/.ssh/aws-keypair ec2-user@<your-ec2-instance-public-ip> -p 22
```

The above command will work the same as if we didn't specify the port. But notice that if we specify a different port:

```
ssh -i ~/.ssh/aws-keypair ec2-user@<your-ec2-instance-public-ip> -p 1234
```

We get an error:

```
ssh: connect to host <your-ec2-instance-public-ip> port 1234: Connection timed out
```

Ports

What did we do to access te jupyter notebook service? We used the public IP address of the EC2 instance (or the DNS name) and the port 8888. But instead of using the SSH command, we typed the following in the browser:

```
http://<your-ec2-instance-public-ip>:8888
```

That is because for SSH, we use the `ssh` command and point to a specific IP address and port. But for jupyter notebooks, we use the browser and point to a specific IP address and port.

In both cases we are accessing a service on a remote device.

Ports

To make this fit on the hotel analogy, imagine that each hotel service had a different window in the room door assigned to it. So every room door in the hotel would have a window for clothes laundry, a window for mail, a window for food delivery, etc. When you want to send mail to a room, you put the mail in the mail window. When you want to send food, you put the food in the food window. And so on.

Ports

Okay so going back to our EC2 example with the jupyter notebook and the SSH service. Imagine the EC2 instance is a hotel room. The room number is the private IP address, the hotel street address is the public IP address, the hotel name is the DNS name, and the windows in the door are the ports.

When we know the precise location of the hotel room (our EC2 instance), **we can access the services** in the room through the different windows in the door (the ports). If we try to put mail into the food window, **we get an error**. If we try to put food into the mail window, we also get an error.

Recap

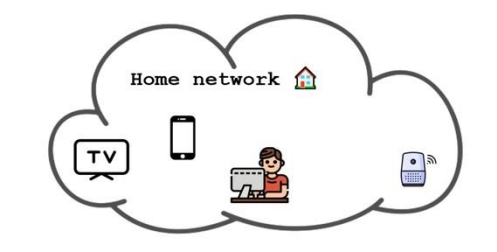
We've seen how IP addresses are used to **identify devices**, and how there's private and public IP addresses that are used to identify devices **within a network** and **on the internet** respectively. Some questions to think about:

- How do multiple devices in a network share a single public IP address?
- What even is a network?
- How do we manage the IP addresses in a network? How do we choose which devices get which IP addresses?

Let's explore these questions further starting with a situation you may be familiar with: your home network.

Our home network

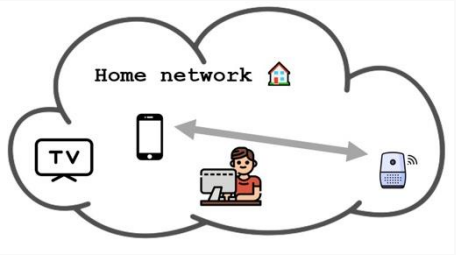
Let’s imagine a simple home network where we have some devices connected. So for example we could have the computer we are using right now, a phone, a smart TV and a smart speaker (like an Amazon Echo).



Our home network

This devices can talk between themselves, for example when I am playing a song on spotify on my phone, I can send it to the smart speaker so my smart speaker is now playing the song.

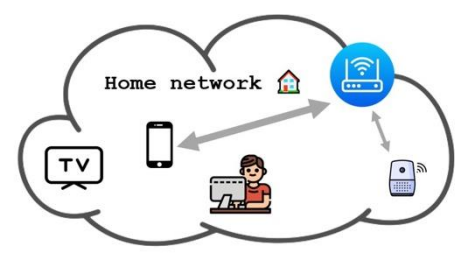
But what is happening? Is my phone directly talking to my smart speaker?



Your router and NAT

No! There is a key player and I am sure you have heard of it: the **router**. Tt is that device we restart when the internet is not working, the place where there is a sticker with the wifi password, that white box with antennas and blinking lights.

The router is the **receptionist** of the network. It is the one responsible for routing traffic between devices in the network and between the network and the internet. So if my phone wants to talk to the smart speaker, it sends the message to the router and the router sends it to the smart speaker.



Your router and NAT

Remember we said there are different private IP ranges reserved for private networks? Well, the router uses these private IP addresses to identify the devices in the network. So when my phone sends a message to the smart speaker, **it uses the private IP address of the smart speaker**. The same way a hotel uses room numbers to identify rooms.

In fact, let's do an experiment. Open a terminal and run the following command if you are on Windows:

```
ipconfig
```

Or if you are on Linux or MacOS:

```
ifconfig
```

Your router and NAT

If you look for your `IPv4 Address`, you will see an address that fits into 192.168.x.x, 10.x.x.x or 172.16.x.x - 172.31.x.x. Most of the times on regular homes it will be 192.168.x.x. This is analogous to how different hotels will use the same room numbers (like room 101) that **are not unique in the world**.

If you instead visit a website like whatismyip.com, you will see a different IP address. **This is the public IP address of your network**. This is the address that identifies your network to the **outside world**.

Your router and NAT

Let's see now how the router uses the **public IP address to communicate with the internet**.

Imagine you are staying in the hotel and want to send a letter to a friend's home (and imagine the hotel is willing to send the letter for you). You would write the address of your friend's home on the envelope, hand it to the receptionist, and the receptionist would send it to your friend's home.

More people on the hotel may be sending letters to their respective friends' homes, so the receptionist **needs to keep track of which letter goes to which home** so when the responses come back, the receptionist knows to whom to give the letter.

Your router and NAT

The router does the same thing. If your phone is streaming a YouTube video, and your smart TV is streaming a Netflix movie, the router is handling traffic between your phone and YouTube and between your smart TV and Netflix **both through the same public IP address** and the **public IPs of YouTube and Netflix**.

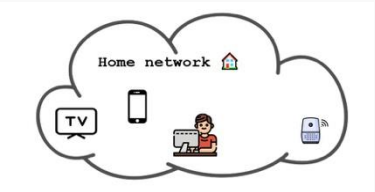
The router keeps track of which traffic is going where so when the responses come back, it knows to whom to send the response. In networking terms, this is called **Network Address Translation (NAT)** and is the protocol routers use to allow multiple devices with private IP addresses to share a single public IP address.Read more about NAT here.

What is a network?

So far we have been talking about networks, using private IPs to communicate between devices within the network and public IPs to communicate with the internet. But **what is a network**? And **what is the internet**?

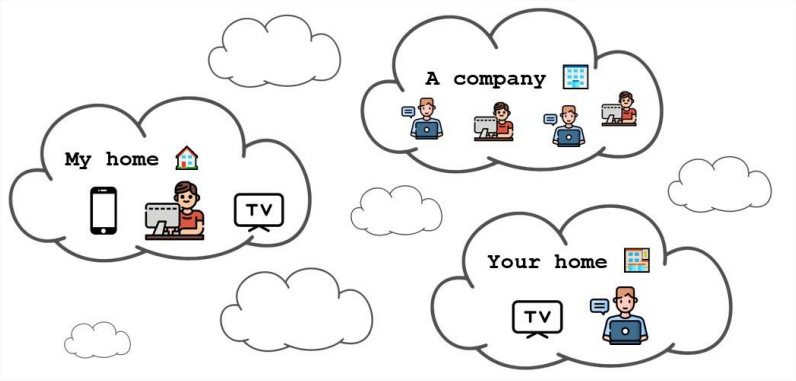
A network is a collection of devices that are connected to each other. So in our home network example, the phone, the smart TV, the smart speaker, and the computer are all part of the network and they are identified by **private IPs**.

In the hotel analogy, the network is the hotel and the different devices are the rooms, where the room number would be the private IP address.



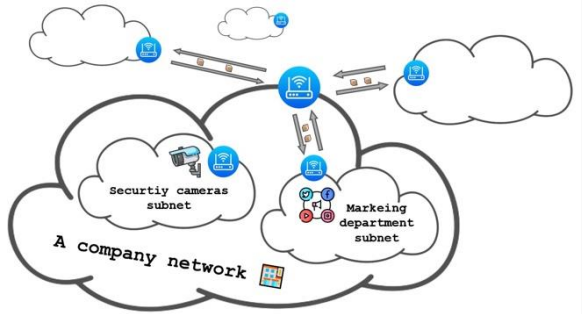
What is a network?

But our home network is not the only network in the world. There are many networks all over the world; your home network, a given company network, the network of a university, etc.



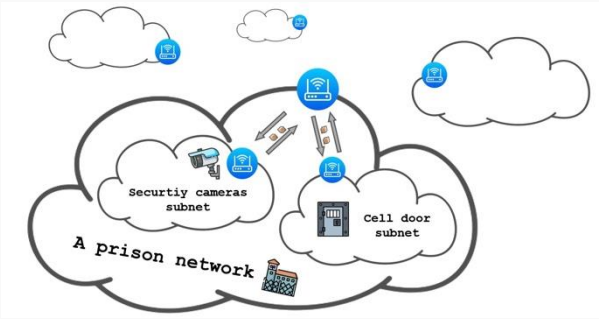
What is the internet?

Networks are **hierarchical**. So the network of a company can be made up of different networks, like the network of the sales department, the network of the marketing department, etc.



What is the internet?

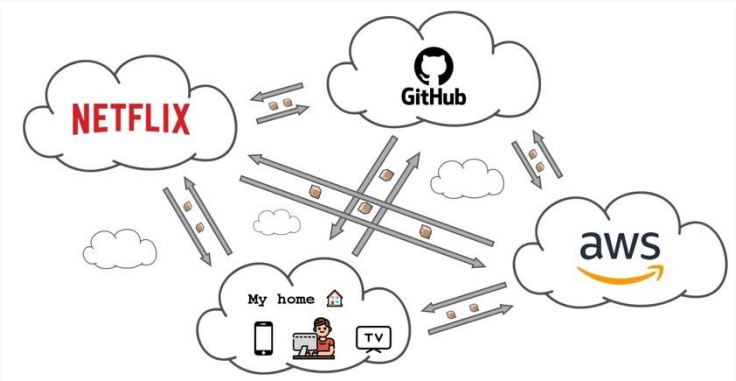
Networks **can be totally isolated** from the rest of the world. That happens in environments like prisons, where there may be multiple networks composing the entire prison network (the network of security cameras, the network of the cell doors, etc). But the bigger prison network itself **is isolated from the rest of the world and does not have access to the internet**.



What is the internet?

Although this private networks exist, the most common scenario is for networks to be connected to the rest of the world. In fact, the **internet** is nothing more than the biggest network of networks in the world.

That is why from our home network we can access anything available on the internet, like YouTube, Netflix, AWS or even this lecture. The whole picture would look like this:



What is the internet?

We constantly use concepts like *the cloud* or *the internet* without really thinking that this is just a colossal amount of devices connected between them.

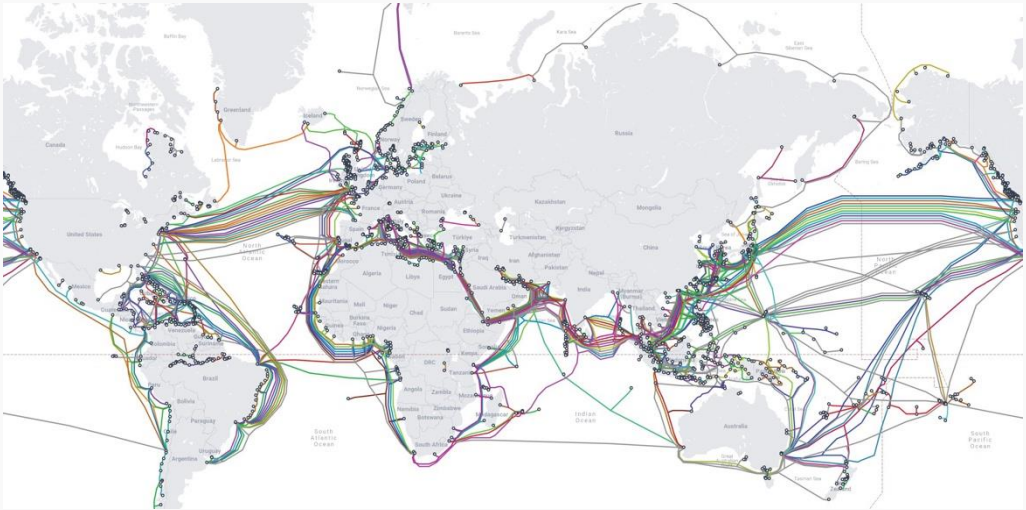
When I want to connect to an EC2 machine in *the cloud*, (assuming it is on the default region `us-east-1`), I am just sending electrical signals to a computer located in a datacenter in the United States.

What is the internet?

Think for a moment the insane trip that the electrical signals have to make to go from your computer to the EC2 machine. They will first go from my computer to my router, from my router at my home to a series of routers managed by Internet Service Providers (ISPs) in Spain, from there it has to **cross the Atlantic Ocean** to reach the United States, and then go through a series of routers managed by ISPs in the United States to finally reach the datacenter where the EC2 machine is located.

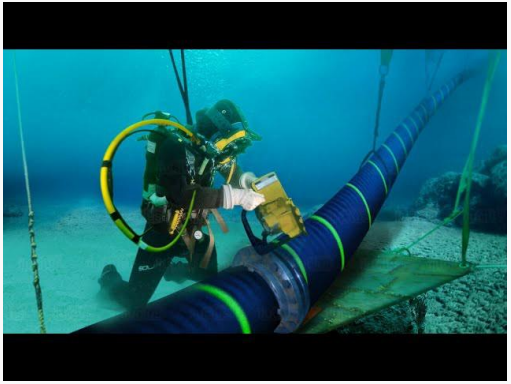
The internet is possible because of an immense infrastructure of cables, routers, switches, and servers that are all connected to each other. In fact, there are a huge amount of cables that go under the ocean to connect the different continents. You can see a map of these cables submarinecablemap.com.

What is the internet?



What is the internet?

Have you ever thought that when you are doing anything on the internet, you may be sending electrical signals that are **crossing the ocean**?



We can do a quick experiment to gauge how far the electrical signals have to travel to reach a given location. We can use the `ping` command to measure the time it takes for a signal to go from our computer to a given location and back.

```
ping google.es
```

The above command will send a signal to google.es and measure the time it takes for the signal to go and come back. The time it takes is called the **round-trip time** and is measured in milliseconds. The round-trip time is a good indicator of how far the location is from your computer. The bigger the round-trip time, the farther the location is.

To see how much time does it take for network traffic to reach different parts of the world, we can use domain mains that we know are located in different countries. For example, we can use the following domains:

```
ping es.pool.ntp.org # Spain
ping us.pool.ntp.org # United States
ping ru.pool.ntp.org # Russia
ping nz.pool.ntp.org # New Zealand
```

Another command we can use to explore the internet is the `tracert` command (**or `traceroute` for MacOS or Linux users**). This command shows the path that the network traffic takes to reach a given location by printing the IP addresses of the routers that the traffic goes through.

```
tracert es.pool.ntp.org # Spain
tracert us.pool.ntp.org # United States
tracert ru.pool.ntp.org # Russia
tracert nz.pool.ntp.org # New Zealand
```

Recap

We've seen how a network is a collection of devices that are connected to each other and how the internet is the biggest network of networks in the world. Some questions to think about:

- How can a single network be made up of multiple networks?
- How does my router choose which IP address to give to the devices that get connected?
- How does a router decide which traffic goes where and which subnet can talk to which other subnet?

Subnetting

What is a subnet?

While explaining what is the internet, we mentioned that a network can be made up of different networks.

This is where **subnets** come into play. A **subnet** is a way of dividing a network into smaller networks.

Think of it as the different floors of a hotel. Imagine there is a restaurant on the ground floor and the remaining floors are for rooms.

We may want people to be able to go from any floor to the restaurant and back, but **we don't want people from one of the room floors to be able to go to another room floor**. We need to have some **control over who can go where**.

What is a subnet?

This is exactly what subnets do. They allow us to divide a network into smaller networks and control which subnet can talk to which other subnet.

In the case of a company we may want the subnet of sales department and the subnet of the marketing department to be able to talk to the subnet of IT administration, but we don't want the subnet of sales to be able to talk to the subnet of marketing.

We could also have a security camera subnet that we don't want to communicate with any other subnet **nor with the internet**.

What is a subnet?

Remember the private IP address ranges we mentioned earlier?
`10.x.x.x, 172.16.x.x - 172.31.x.x, 192.168.x.x`.

Imagine a company network that has a public IP address allowing it to communicate with the internet. Devices inside this network will be assigned private IP addresses. If we just wanted to have one big network, we could assign all devices the same private IP address range.

How do we define address ranges for subnets?

What is a subnet?

You may have noticed that IP addresses often come with a trailing slash and a number. For example, `192.168.1.0/24`.

This `/24` is called the **subnet mask** and it means that we are not in front of an IP address but an **IP address range**, in this case, the range of IP addresses between `192.168.1.0` and `192.168.1.255`.

What is a subnet?

Remember we earlier said that an IP address is a 32-bit number divided into four 8-bit numbers **separated by dots**. Which in simple terms just means that an IP address is built from 4 numbers between 0 and 255 separated by dots.

The subnet mask is a way of telling **which part of the IP address is fixed and which part can change**. In the case of `/24`, the first 24 bits are fixed and the last 8 bits can change, which means that the last number in the IP address can be any number between 0 and 255.

Other common subnet masks are `/16` and `/8`. `/16` means the first 16 bits are fixed and the last 16 bits can change. `/8` means the first 8 bits are fixed and the last 24 bits can change.

What is a subnet?

Depending on how many octets of the IP address are fixed, we can have a different amount of private IPs available.

Subnet Mask	Number of IPs
/8	16,777,216 (2^24)
/16	65,536 (2^16)
/24	256 (2^8)

What is a subnet?

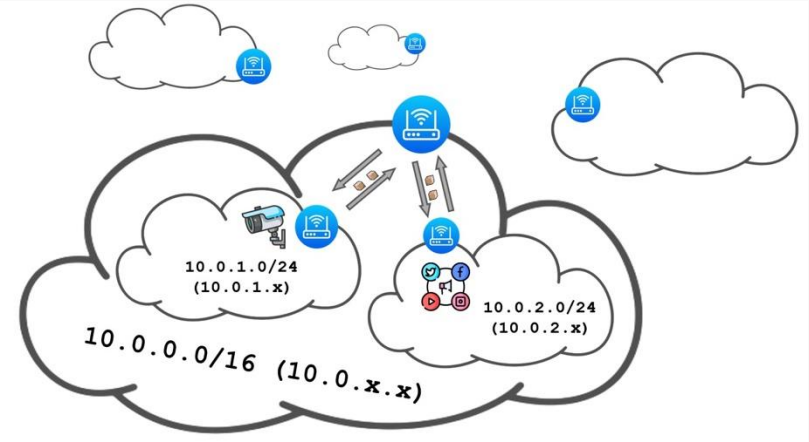
Let's see an example of how we can divide a network into subnets. Imagine we have a network with the IP address range `10.0.0.0/16`. This means **in total we have 65,536 IP addresses** available ranging from `10.0.0.0` to `10.0.255.255`.

If we wanted to divide this network into two subnets, we could use the following IP address ranges:

- Subnet 1: `10.0.1.0/24` (IP addresses from `10.0.1.0` to `10.0.1.255`)
- Subnet 2: `10.0.2.0/24` (IP addresses from `10.0.2.0` to `10.0.2.255`)
- Subnet 3: `10.0.3.0/24` (IP addresses from `10.0.3.0` to `10.0.3.255`)
- etc.

What is a subnet?

So if we had a company where we want to divide our network into the security cameras subnet and the marketing department subnet, we could have the following setup:



What is a subnet?

And how do routers apply different rules to different subnets? They use **route tables**. It consists of a list of rules that specify which traffic can go where.

We could build a route table that says that traffic from the marketing department and security cameras can go to the internet (so we may simplify and say that traffic originating from **any** subnet can go to the internet), but traffic from the marketing department can't go to the security cameras subnet and vice versa.

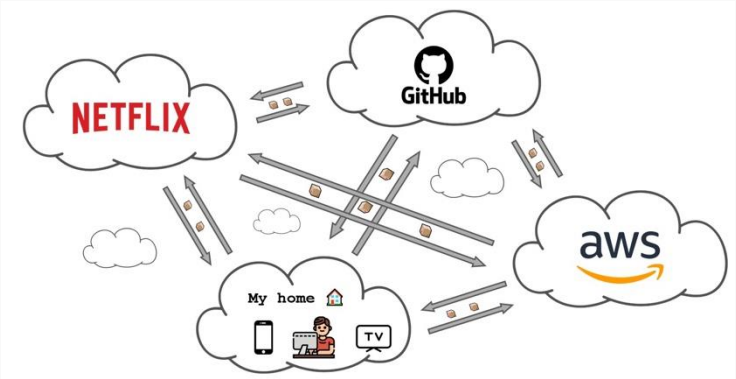
What is a subnet?

Don’t get overwhelmed by the amount of information, I know subnets and route tables can be confusing and they indeed are complex topics. But just stick with the idea that **subnets are a way of dividing a network into smaller networks so we can control how does traffic flow between them and to the internet**.

Networking in AWS

How do AWS networks work?

So far we have been talking about networks in general, but how do networks work in AWS? Just as a reminder, remember we are in this **network of networks called the internet**.



How do AWS networks work?

And of course, AWS is part of the internet.

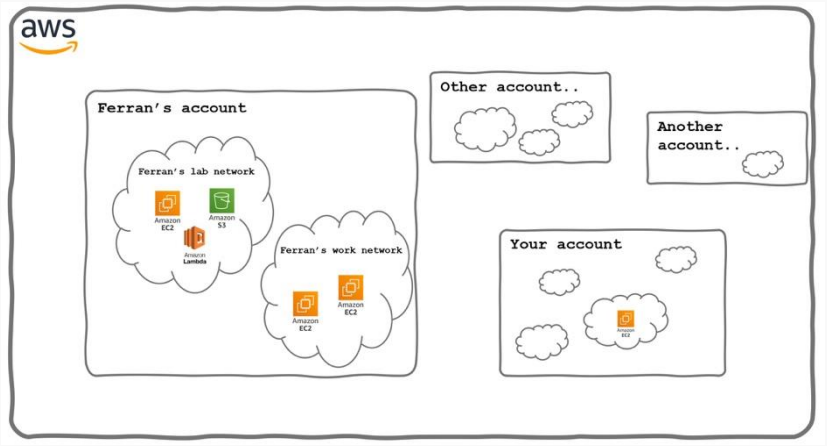
How do AWS networks work?

But what happens inside the AWS network? How do **different users** have its own individual space?

Each user in AWS has its own set of networks. These networks are called **Virtual Private Clouds (VPCs)** which is just a fancy name for the AWS service that lets us create and manage our own networks inside AWS.

By default, everything we launch in AWS lives **inside the same VPC**, which is one that AWS creates and configures for us. But we can create other VPCs and configure them as we want.

How do AWS networks work?

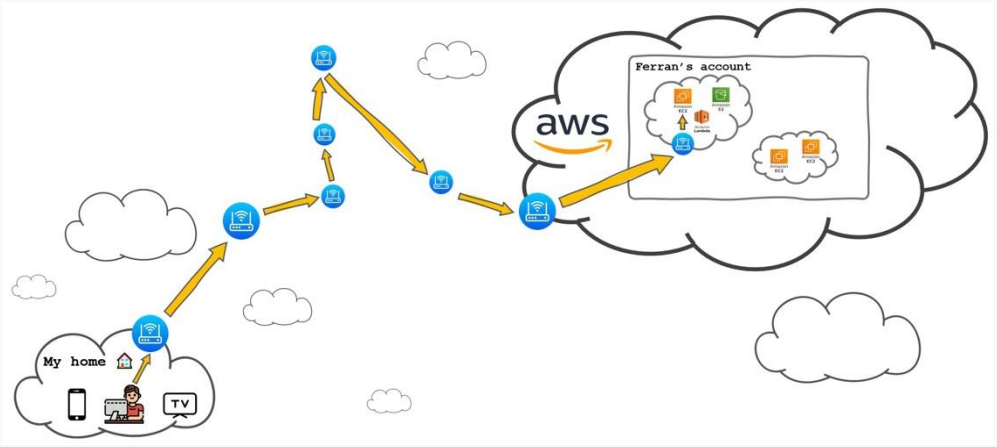


How do AWS networks work?

And what happens when I connect to an EC2 instance in AWS? How does the traffic flow?

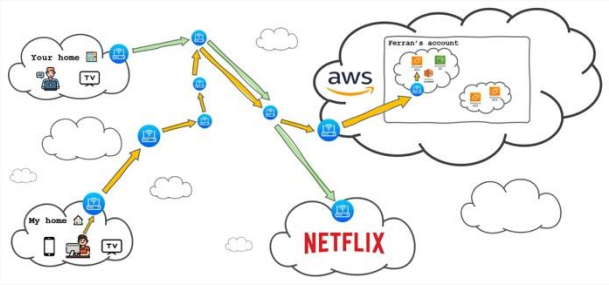
When we connect to an EC2 instance, the traffic goes **through the internet** to the AWS datacenter where the EC2 instance is located. But it doesn't go directly to the EC2 instance. Once it has gotten to *our account territory*, it goes to the router of the VPC where the EC2 instance is located. This router is the one that finally sends the traffic to the EC2 instance. In AWS terms, this router is called an **Internet Gateway**.

How do AWS networks work?



How do AWS networks work?

Just as a reminder, when I say **through the internet**, this means that my network traffic, at some point, could be going through the same cables that your network traffic is going through. So it is possible that my network traffic and your network traffic **cross paths** at some point. The same way someone going from Spain to Germany and someone going from France to Italy could cross paths at some point.



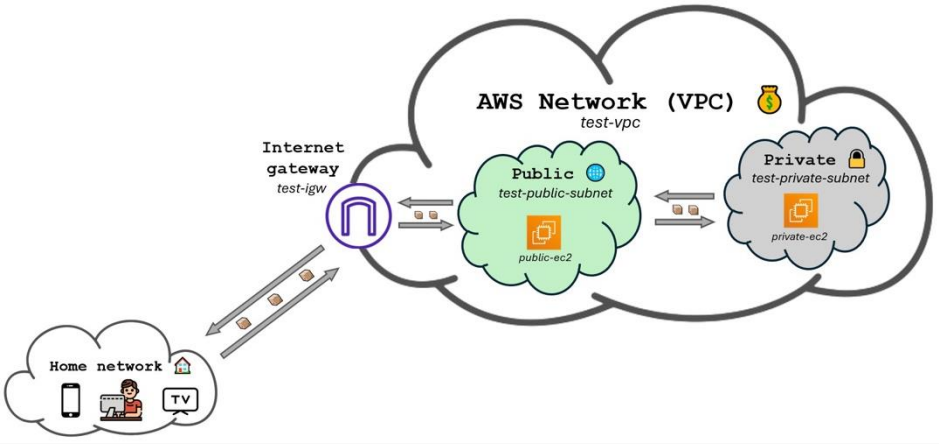
Lab: Experimenting with VPCs

Goal of the lab

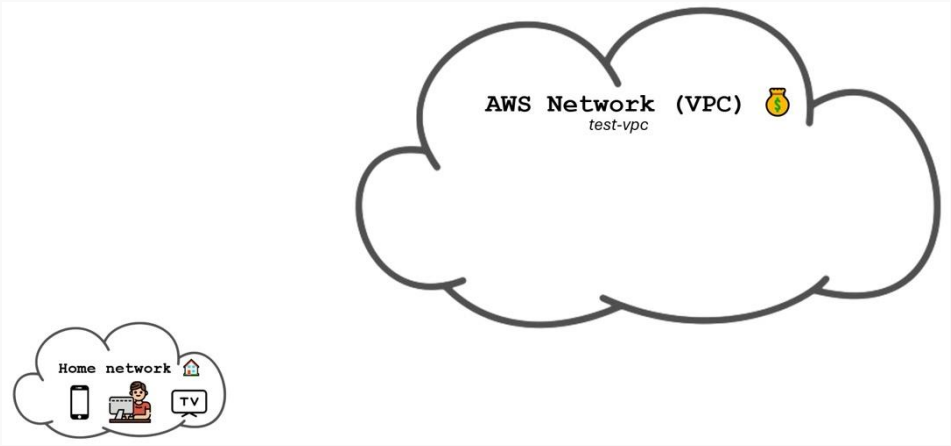
During this lab we will:

- Create a **VPC**.
- Create two **subnets** in the VPC.
- Configure one subnet as **public** and the other as **private** with an **Internet Gateway** and **Route Tables**.
- Deploy an **EC2 instance** in both subnets.
- Access the private EC2 instance from the public EC2 instance.

Goal of the lab



Creating a VPC

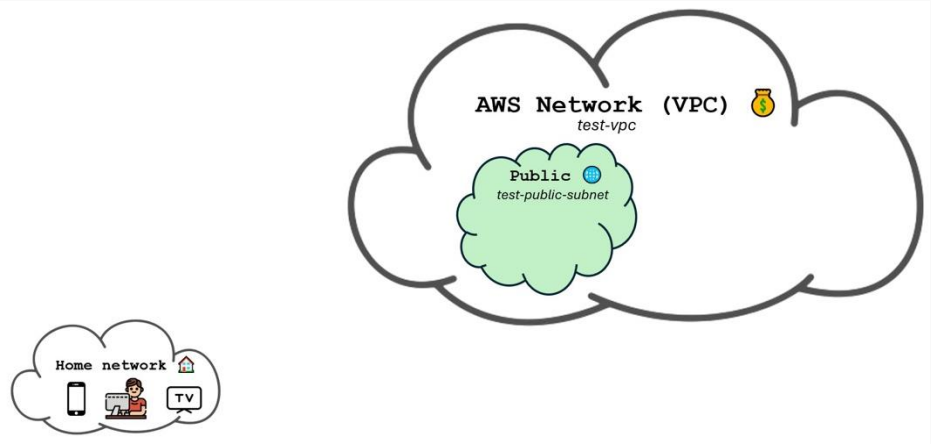


Creating a VPC

1. Go to the VPC service.
2. Click on Create VPC.
3. Choose `VPC only`.
4. Fill the form with the following settings:
   - **Name tag**: test-vpc
   - **IPv4 CIDR block**: 10.0.0.0/16
5. Leave the rest as default and click on Create VPC.

Creating a public subnet in a VPC

We are now going to add a subnet to the VPC we just created.

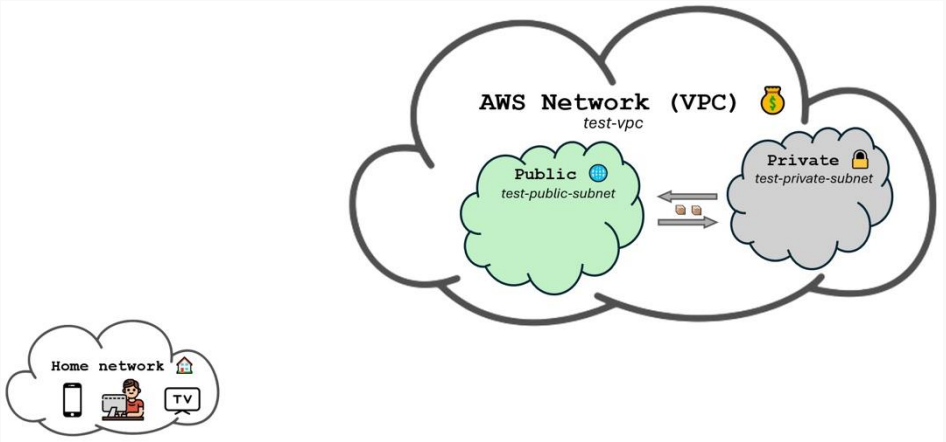


Creating a public subnet in a VPC

1. Navigate to the *VPC service* in the *AWS Management Console*.
2. Select **Subnets** on the left navigation pane.
3. Click on *Create subnet*.
4. Select the VPC you created, which is the one ending with (test-vpc).
5. Fill in the form with the following settings:
   - **Name**: test-public-subnet
   - **Availability Zone**: Select us-east-1a zone (or any other zone but be consistent)
   - **IPv4 VPC CIDR block**: 10.0.0.0/16 (leave as default)
   - **IPv4 subnet CIDR block**: 10.0.1.0/24
6. Click on *Create subnet*.

Creating a private subnet in a VPC

And we add a second one.

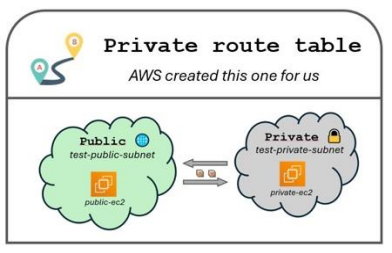


Creating a private subnet in a VPC

1. Navigate to the *VPC service* in the *AWS Management Console*.
2. Select **Subnets** on the left navigation pane.
3. Click on *Create subnet*.
4. Select the VPC you created, which is the one ending with (test-vpc).
5. Fill in the form with the following settings:
   - **Name**: test-private-subnet
   - **Availability Zone**: Select us-east-1a zone (the same as the public subnet)
   - **IPv4 VPC CIDR block**: 10.0.0.0/16 (leave as default)
   - **IPv4 subnet CIDR block**: 10.0.2.0/24
6. Click on *Create subnet*.

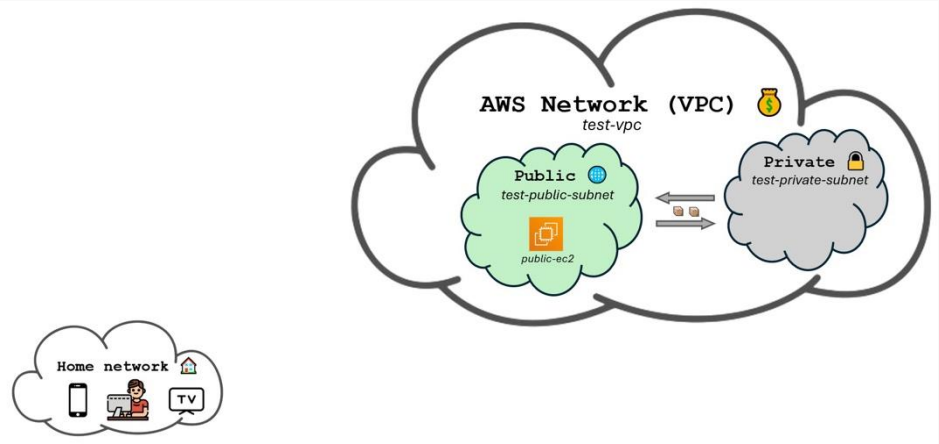
Default route table

Have you noticed the two subnets are connected? This means that the EC2 instances in the public subnet can talk to the EC2 instances in the private subnet. This is because they are part of the same VPC and the default route table allows traffic to flow between them.



Deploying an EC2 instance in the public subnet

Now we are going to launch an EC2 instance inside the public subnet.



Deploying an EC2 instance in the public subnet

1. Go to the *EC2 service*.
2. Click on *Launch instance*.
3. Fill the form with the following settings:
   - **Name**: public-ec2
   - **Key pair (login)**: Choose `aws-keypair` (the one we have already been using)
4. In the network settings tab clik on *Edit* and select the following settings:
   - **VPC - required**: The one ending with (test-vpc)
   - **Subnet**: test-public-subnet
   - **Auto-assign Public IP**: Enable
5. Leave the rest as default and click on *Launch instance*.

Accessing the public EC2 instance

Try to access the public EC2 instance using the following command:

```
ssh -i ~/.ssh/aws-keypair ec2-user@<your-public-ec2-instance-public-ip>
```

You'll notice that the command hangs for a while and finally fails with a timeout error. Which means the EC2 instance didn't respond.

Accessing the public EC2 instance

Observations

- The EC2 instance is not accessible from the internet.
- The reason is because the VPC and the subnet are private by default.
- Although we named one subnet as public, it is not public yet.

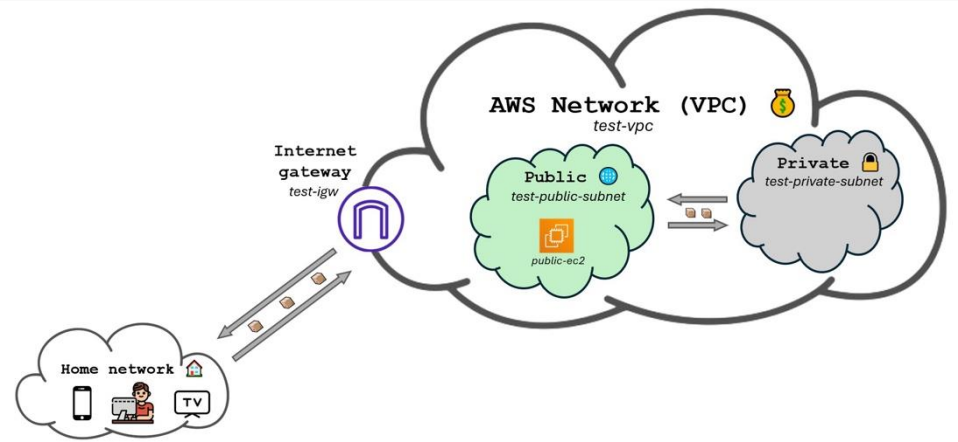
Solution

We need to create an **internet gateway** and a **routing table**.

- The internet gateway will be the entrypoint for the internet traffic to the VPC.
- The routing table will make internet traffic able to go from the internet gateway to a given subnet.
- Once the traffic can reach a subnet, it can reach the EC2 instances on that subnet.
- Doing this with an internet gateway means that the traffic is bidirectional. The EC2 instances can also reach the internet.

Accessing the public EC2 instance

To access the public instance, we'll first need to allow traffic to get to the VPC. We'll do this by creating an **Internet Gateway**.

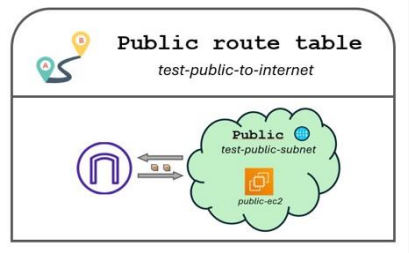


Creating an Internet Gateway

1. Go to the *VPC service*.
2. Click on *Internet Gateways* on the left navigation pane.
3. Click on *Create internet gateway*.
4. Fill the form with the following settings:
   - **Name**: test-igw
5. Click on *Create internet gateway*.
6. Now click on *Actions* on the top right corner and select *Attach to VPC*.
7. Choose the one ending with `test-vpc` and click on *Attach internet gateway*.

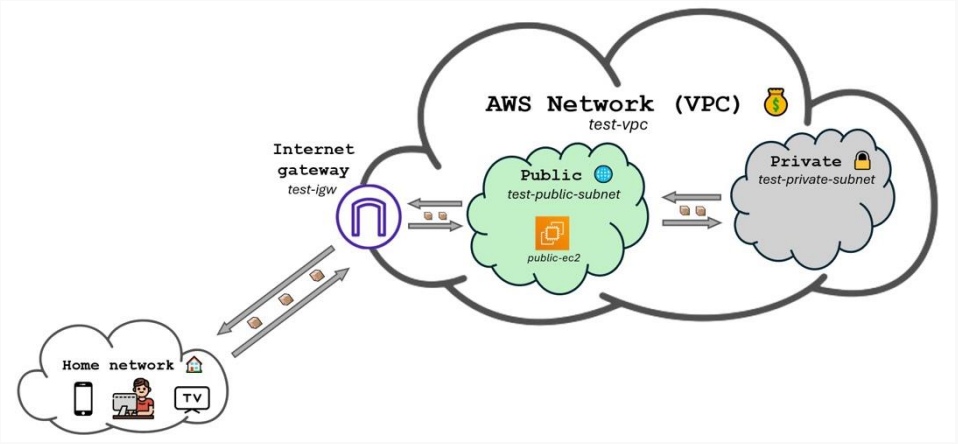
Accessing the public EC2 instance

Now traffic is getting to the VPC, but it still can't reach devices inside the public subnet. That is because despite its name, the public subnet is still private. We need to make it public by creating a **Route Table**.



Accessing the public EC2 instance

Once the table is created, traffic will be able to reach the public subnet.



Creating a Route Table

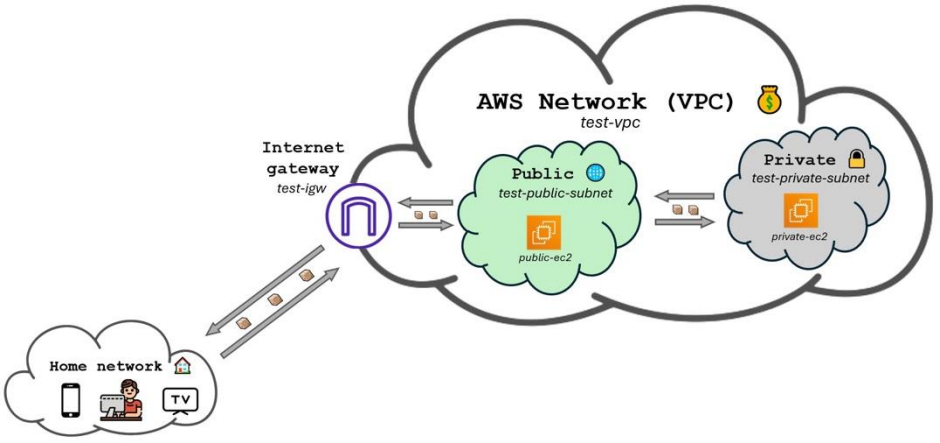
1. Go to the *VPC service*.
2. Click on *Route Tables* on the left navigation pane.
3. Click on *Create route table*.
4. Fill the form with the following settings:
   - **Name**: test-public-to-internet
   - **VPC**: The one ending with (test-vpc)
5. Click on *Create route table*.
6. Click on *Actions* on the top right corner and select *Edit subnet associations*.
7. Choose the public subnet and click on *Save associations*.
8. Click on *Actions* on the top right corner and select *Edit routes*.
9. Click on *Add route* and fill the form with the following settings:
   - **Destination**: 0.0.0.0/0
   - **Target**: Choose *Internet Gateway* and select the one ending with (test-igw).
10. Click on *Save changes*.

Accessing the public EC2 instance

- Now the EC2 instance is accessible from the internet.
- The public subnet is indeed public now.
- The private subnet is still private.

Deploying an EC2 instance in the private subnet

We are now going to launch an EC2 instance inside the private subnet.



Deploying an EC2 instance in the private subnet

1. Go to the *EC2 service*.
2. Click on *Launch instance*.
3. Fill the form with the following settings:
   - **Name**: private-ec2
   - **Key pair (login)**: Choose `aws-keypair` (the one we have already been using)
4. In the network settings tab clik on *Edit* and select the following settings:
   - **VPC - required**: The one ending with (test-vpc)
   - **Subnet**: test-private-subnet
   - **Auto-assign Public IP**: Enable
5. Leave the rest as default and click on *Launch instance*.

Accessing the private EC2 instance

We know we are not going to be able to SSH into the private EC2 instance since it resides in a private subnet. What we aim to do is to access the private EC2 instance from the public EC2 instance.

The first step is going to be to copy our private key onto the public EC2 instance so we can SSH into the private EC2 instance from there. Open a terminal in your **local machine** and run the following command:

```
scp -i ~/.ssh/aws-keypair ~/.ssh/aws-keypair ec2-user@<your-public-ec2-instance-public-ip>:/home/
```

This is just copying the private key to the public EC2 instance. Now we can SSH into the public EC2 instance and from there try to SSH into the private EC2 instance.

Accessing the private EC2 instance

SSH into the public EC2 instance using the following command:

```
ssh -i ~/.ssh/aws-keypair ec2-user@<your-public-ec2-instance-public-ip>
```

And run the following command to change the permissions of the private key:

```
chmod 600 ~/.ssh/aws-keypair
```

From the public EC2 instance, try SSH into the private EC2 instance using the following command:

```
ssh -i ~/.ssh/aws-keypair ec2-user@<your-private-ec2-instance-public-ip>
```

Accessing the private EC2 instance

You'll notice that the command hangs for a while and finally fails with a timeout error just as id did earlier when the public EC2 instance was not accessible from the internet. This is because we are trying to access the private EC2 instance using its **public** IP address. We need to use the **private** IP address instead.

Go to the *EC2 service* in the *AWS Management Console* and copy the **private** IP address of the private EC2 instance. Then SSH into the public EC2 instance and try to SSH into the private EC2 instance using the **private** IP address.

```
ssh -i ~/.ssh/aws-keypair ec2-user@<your-private-ec2-instance-private-ip>
```

Test internet access from the private EC2 instance by running the following command:

```
ping google.com
```

It should error out, signaling that there is no internet access.

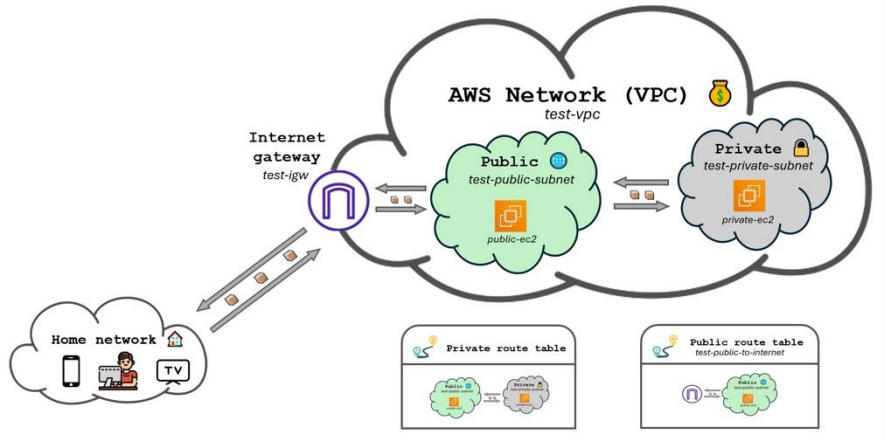
Accessing the private EC2 instance

Observations

- The EC2 instance is not accessible from the internet and will never be.
- We can access the private EC2 instance from the public EC2 instance.
- The private instance doesn't have access to the internet.

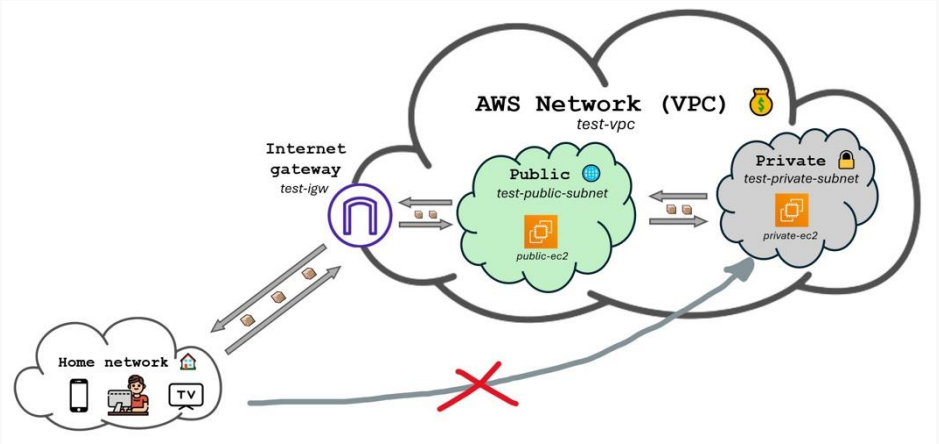
Recap

We ended up with this setup.



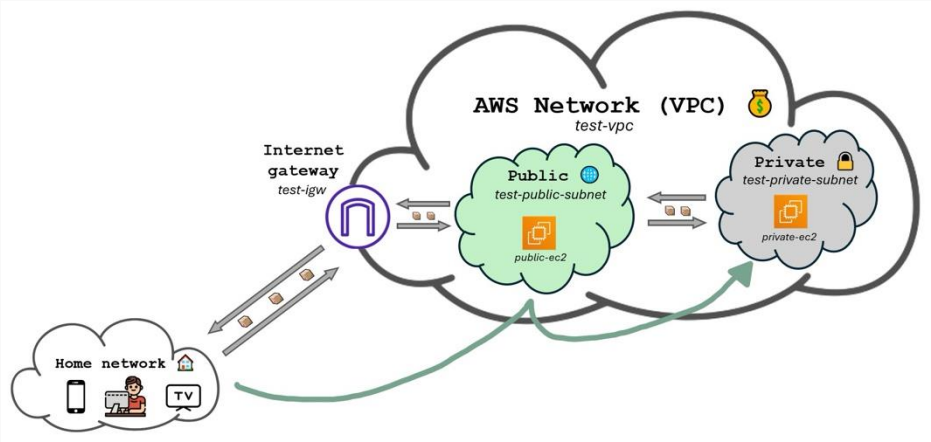
Recap

Where we can't directly access the private EC2 instance through SSH.



Recap

But we can access the private EC2 instance from the public EC2 instance.



VPNs

What is a VPN?

A Virtual Private Network (VPN) is a secure connection between two networks over the internet. They allow devices to **behave like they are on a network they are not physically connected to**.

What is a VPN?

A Virtual Private Network (VPN) is a secure connection between two networks over the internet. They allow devices to **behave like they are on a network they are not physically connected to**.

VPNs encrypt traffic, making communication secure.

They **enable private resources to be accessed** remotely as if directly connected to the private network.

What is a VPN?

We use VPNs mainly for:

- **Privacy**: Mask the source IP address.
- **Spoofing location**: Make it look like we are in a different location (to access geo-restricted content).
- **Remote Access**: Connect securely to resources inside a private network from anywhere.

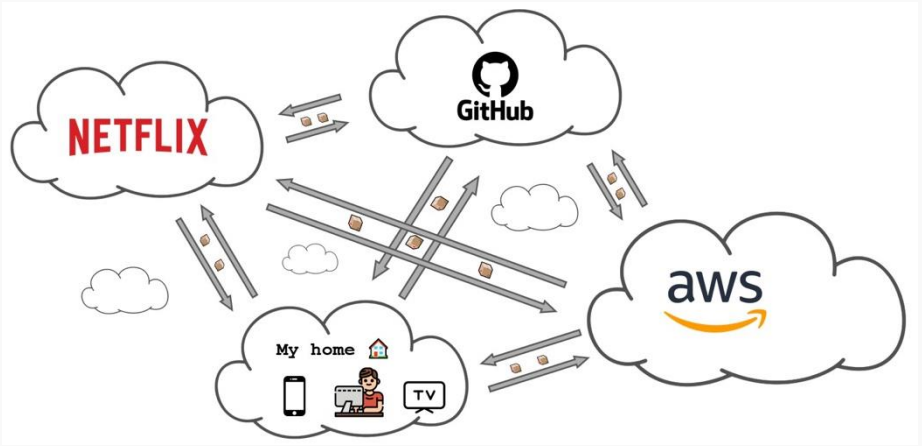
What is a VPN?

Think of a VPN as **a tunnel connecting two buildings**. The tunnel would allow me to access private resources on the other building as if I were inside it.

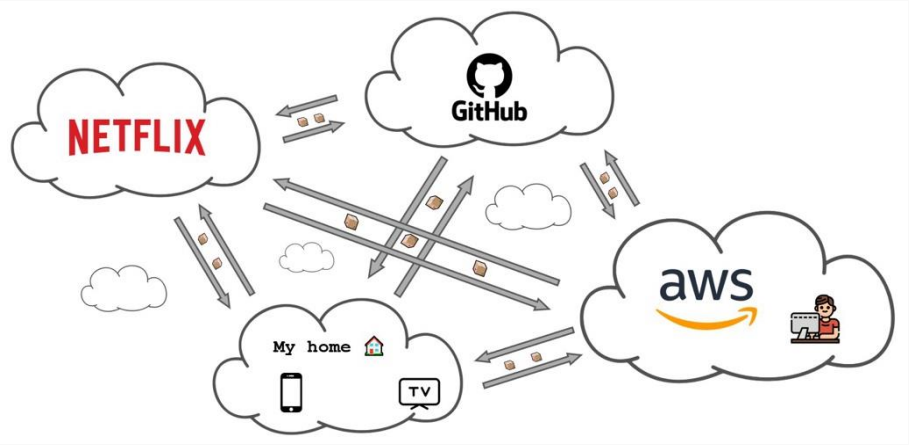
If I wanted to send a package from my home but no one to know it was sent from my home, I could send through the tunnel first and then from the other building to the destination. **The package would be sent from the other building, not from my home.**

Users inside the tunnel communicate **as if they were in the same location**.

What is a VPN?



What is a VPN?



What a VPN is not

Due to misleading marketing, many people think that VPNs are a way to be **anonymous** on the internet. This is not true.

How many of you have seen advertisements of companies like NordVPN or ExpressVPN saying that **they will make you anonymous on the internet**?

VPNs hide the traffic from your ISP, but **the VPN provider can see your traffic**. We're just moving the trust from the ISP to the VPN provider.

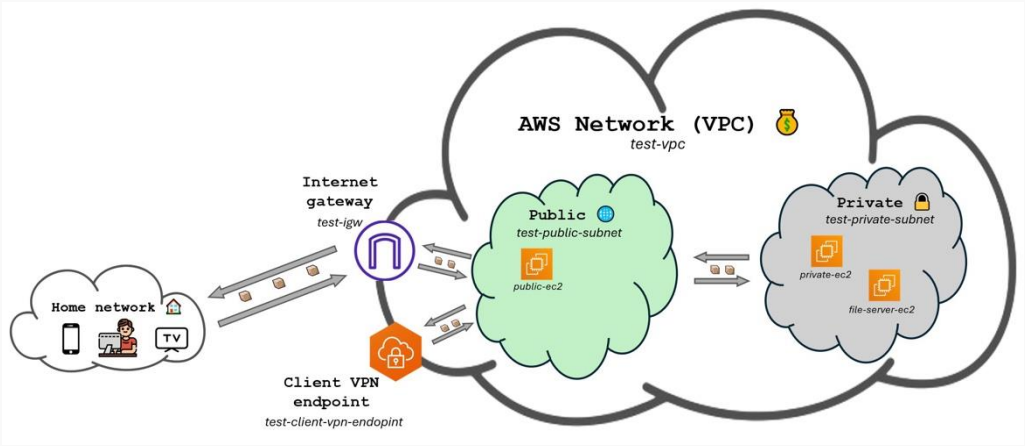
Lab: Building our own VPN

Goal of the lab

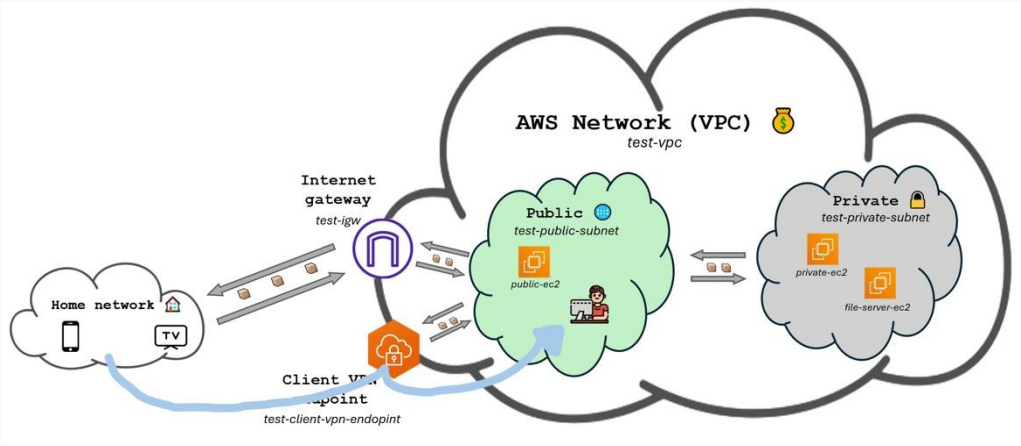
During this lab we will:

- Create a **VPN** using the **AWS Client VPN** service.
- Connect to the VPN from our computer.
- Deploy a file server on a private EC2 instance.
- Access the file server from our computer through the VPN.

Goal of the lab



Goal of the lab



Generate client and server certificates

First of all we need to generate the certificates for the VPN. We will use **EasyRSA** to generate them.

For Windows users:

1. Visit https://github.com/OpenVPN/easy-rsa/releases and look for a file in `Assets` ending with `win64.zip`.
2. Download the file and extract it.
3. Open a terminal and navigate to the folder where you extracted the files.
4. Run the following command: `.\EasyRSA-Start.bat`

For MacOS and Linux users:

```
git clone https://github.com/OpenVPN/easy-rsa.git
cd easy-rsa/easyrsa3
```

Generate client and server certificates

Now that we have EasyRSA running, we can generate the certificates.

```
./easyrsa init-pki
./easyrsa build-ca nopass
```

When the following is shown in the terminal just press enter:

```
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:
```

Generate client and server certificates

Now run the commands below:

```
./easyrsa --san=DNS:server build-server-full server nopass
./easyrsa build-client-full client1.domain.tld nopass
```

When you see the text below, just type `yes` and hit enter:

```
Type the word 'yes' to continue, or any other input to abort.
  Confirm requested details: yes
```

Generate client and server certificates

Okay so we've now successfully generated the certificates. We just need to move them to another folder to make it easier to work with them.

Windows users type `exit` and then run the following commands (instructions for MacOS/Linux are on the next slide):

```
mkdir C:\custom_folder
copy pki\ca.crt C:\custom_folder
copy pki\issued\server.crt C:\custom_folder
copy pki\private\server.key C:\custom_folder
copy pki\issued\client1.domain.tld.crt C:\custom_folder
copy pki\private\client1.domain.tld.key C:\custom_folder
cd C:\custom_folder
```

Generate client and server certificates

For those on MacOS and Linux (you **don't** have to type `exit`):

```
mkdir ~/custom_folder/
cp pki/ca.crt ~/custom_folder/
cp pki/issued/server.crt ~/custom_folder/
cp pki/private/server.key ~/custom_folder/
cp pki/issued/client1.domain.tld.crt ~/custom_folder
cp pki/private/client1.domain.tld.key ~/custom_folder/
cd ~/custom_folder/
```

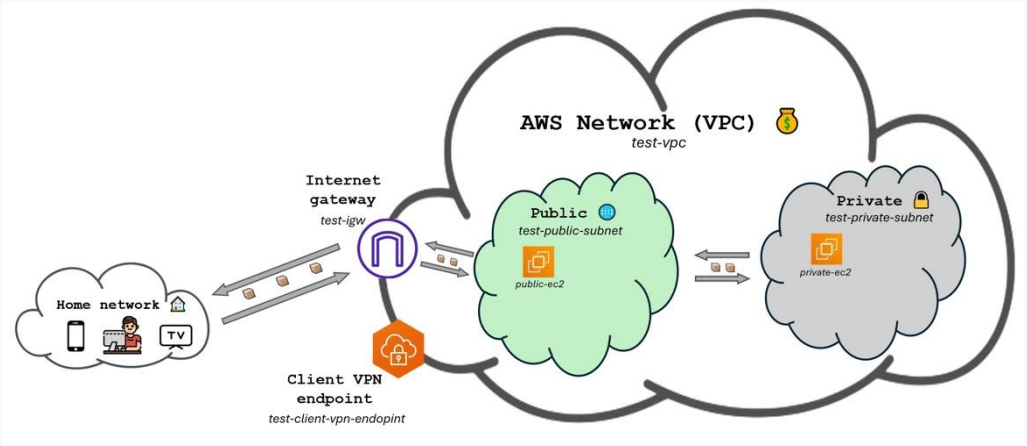
Import the certificates to AWS

Now we just need to import the certificates to AWS. We'll do that with the AWS CLI, so **remember to update your .aws/credentials file with your access key and secret key**.

```
aws acm import-certificate --certificate fileb://server.crt \
    --private-key fileb://server.key \
    --certificate-chain fileb://ca.crt --region us-east-1
```

```
aws acm import-certificate --certificate fileb://client1.domain.tld.crt  \
    --private-key fileb://client1.domain.tld.key \
    --certificate-chain fileb://ca.crt --region us-east-1
```

Create client VPN endpoint



Create client VPN endpoint

Now we are going to create the client VPN endpoint.

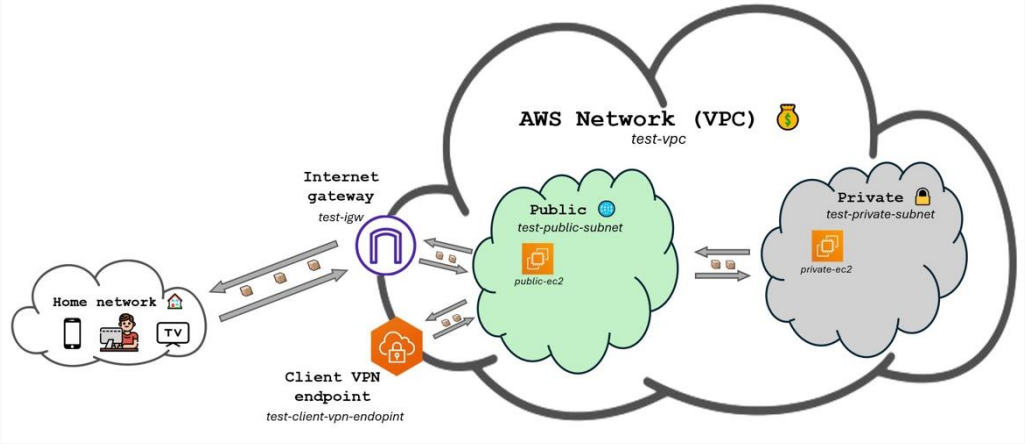
1. Search for **Client VPN endpoints** in the AWS Management Console and click on the first result.
2. Click on **Create Client VPN endpoint**.
3. Fill the form with the following
   - **Name tag**: test-client-vpn-endpoint
   - **Client IPv4 CIDR**: 10.83.0.0/16
   - **Server certificate ARN**: Look for the one called `server`.
   - **Authentication options**: Enable *Mutual authentication* and select the client certificate called `client1.domain.tld`.
   - **DNS server 1 IP address**: 1.1.1.1
   - **DNS server 2 IP address**: 8.8.8.8
   - **VPC ID**: The one ending with (test-vpc)
1. Leave the rest as default and click on **Create Client VPN endpoint**.

Associate the client VPN endpoint with the VPC public subnet

Now we need to associate the client VPN endpoint with the public subnet.

1. Click on the client VPN endpoint we just created or search for *Client VPN endpoints* in the AWS Management Console if you are not there.
2. Click on the **Target network associations** tab down below.
3. Click on **Associate target network**.
4. Choose `test-vpc` for the VPC and `test-public-subnet` for the subnet.
5. Click on **Associate target network**.

Add an authorization rule for the VPC



Add an authorization rule for the VPC

Now we are going to add an authorization rule to allow traffic to flow from the VPN to the VPC.

1. Click on the **Authorization rules** tab.
2. Click on **Add authorization rule**.
3. Enter `10.0.0.0/16` for the destination network and click on **Add authorization rule**.

Provide access to the internet for devices connected to the VPN

We would now be able to connect to the VPN, but once connected to the VPN we would not have acces to the internet. That is because we have to add a route to the route table and an authorization rule allow traffic to the internet.

1. Click on the **Route table** tab.
2. Click on **Create route**.
3. Fill the form with the following
   - **Route destination**: 0.0.0.0/0
   - **Subnet ID for target network association**: Choose the onte starting with *subnet*
4. Click on **Create route**.
5. Click on the **Authorization rules** tab.
6. Click on **Add authorization rule**.
7. Choose `0.0.0.0/0` for **Destination network to enable access** and click on **Add authorization rule**.

Setup the Client VPN endpoint configuration file

Now we are going to setup the configuration file for the VPN. This file is the one we are going to use on our computer to connect to the VPN.

1. Search for **Client VPN endpoints** in the AWS Management Console if you are not there and click on the client VPN endpoint we created.
2. You'll now be able to click on **Download client configuration**.
3. This will download a file called `downloaded-client-config.ovpn`.
4. Open the file with a text editor since we're going to modify it.

Setup the Client VPN endpoint configuration file

We are going to use those certificate files we created earlier, so we would need a terminal on the folder we moved the certificates to.Follow the steps below if you closed the terminal we used before.

Windows users:

```
cd C:\custom_folder
```

MacOS/Linux users:

```
cd ~/custom_folder/
```

Setup the Client VPN endpoint configuration file

With the `downloaded-client-config.ovpn` file open in a text editor, add the following lines at the beginning of the file:

```
<cert>
Contents of client certificate (.crt) file
</cert>

<key>
Contents of private key (.key) file
</key>
```

Setup the Client VPN endpoint configuration file

Now replace the `Contents of client certificate (.crt) file` with the content of the `client1.domain.tld.crt` file and the `Contents of private key (.key) file` with the content of the `client1.domain.tld.key` file.

To obtain the contents of those files you can run the following commands on the terminal we have opened:

```
cat client1.domain.tld.crt
cat client1.domain.tld.key
```

Once you are done, save the `downloaded-client-config.ovpn` file.

Connect to the Client VPN endpoint

We have everything ready to connect to the VPN, we just need to download the AWS VPN Client which can be obtained here https://aws.amazon.com/vpn/client-vpn-download/.

Once you have installed the client, open it, click on `File` > `Manage Profiles` > `Add Profile`, type whichever name you want for **Display name** and for **VPN Configuration file** click on the browse icon and select the `downloaded-client-config.ovpn` file we just modified.

Test the VPN

Now that we are connected to the VPN, we can test do a couple tests.

First of all, we should now be able to SSH into the private EC2 instance **directly from our computer**. To do this, open a terminal and run the following command:

```
ssh -i ~/.ssh/aws-keypair ec2-user@<your-private-ec2-instance-private-ip>
```

Another cool thing to do is to use an IP geolocation website like https://iplocation.io/ to see that your IP address is now the one of the VPN and that **your location has changed**.

You could also try visiting websites that are only available to people in the US. For example https://www.radio.com.

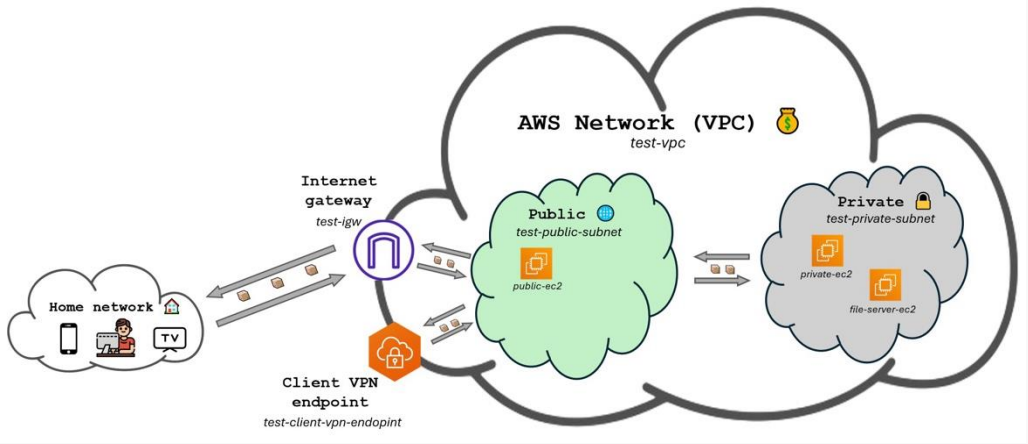
Deploying a file server on the private subnet

We're now going to add another machine **on the private** subnet where we are going to deploy **a file server** (like a Google Drive).

This service is called **FileGator**, read more about it here https://filegator.io/. It's basically one of many open-source locally deployable file servers.

This time **we're not** going to use the Amazon Linux base image and instead we'll use Ubuntu since it will make things easier.

Deploying a file server on the private subnet



Create an EC2 instance on the private subnet

1. Go to the *EC2 service*.
2. Click on *Launch instance*.
3. Fill the form with the following settings:
   - **Name**: file-server-ec2
   - **Application and OS Images**: Click on *Ubuntu*
   - **Key pair (login)**: Choose `aws-keypair` (the one we have already been using)
4. In the network settings tab clik on *Edit* and select the following settings:
   - **VPC - required**: The one ending with (test-vpc)
   - **Subnet**: test-private-subnet
   - **Auto-assign Public IP**: Enable
5. Still inside the network settings, on the **Inbound Security Group Rules** section, click on the button **Add security group rule** and add the following rule:
   - **Type**: Custom TCP
   - **Source type**: Anywhere
   - **Port range**: 8080
6. Leave the rest as default and click on *Launch instance*.

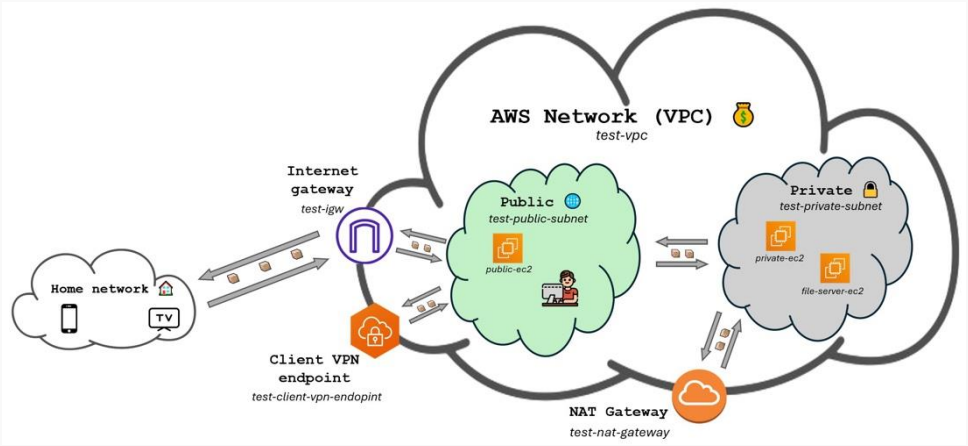
Give it access to internet

If you remember, machines deployed on the private subnet **dit not have access to the internet**.

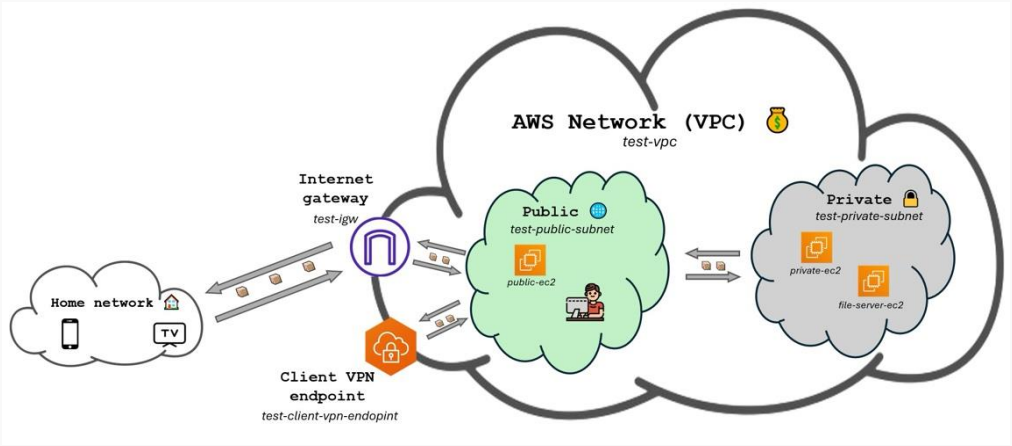
We're going to temporarily enable internet access (which **is not the same as making it public**) so we can install the necessary software. Once we're finished we are going to remove the internet access.

To do so, we're going need a **NAT Gateway** and a new **Route table** for the private subnet.

Give it access to internet



Give it access to internet



Give it access to internet

Follow the steps below to create the NAT Gateway:

1. Search for `NAT Gateways` in the AWS Management Console and click on the first result.
2. Click on **Create NAT Gateway**.
3. Fill the form with the following settings:
   - **Name**: test-nat-gateway
   - **Subnet**: Choose the one ending with (test-public-subnet)
   - **Elastic IP allocation ID**: Click on the button on the right called **Allocate Elastic IP**.
4. Click on **Create NAT Gateway**.

Give it access to internet

Next, we'll have to **modify the route table of the private subnet** to allow traffic to flow to the internet. Remember we said AWS already created one default route table for our subnets? That is the one we are going to modify.

1. Search for `VPC` service and click on the first result.
2. Click on **Route Tables** on the left navigation pane.
3. Click on the default route table that is associated with `test-vpc`. It is **the one with no name** and that ends with `| test-vpc` on the **VPC** column (you may have to scroll to the right to see it).
4. Click on the **Routes** tab down below.
5. Click on **Edit routes**.
6. Click on **Add route** and type `0.0.0.0/0` for **Destination** and select the NAT Gateway we just created for **Target** (`NAT Gateway` > `test-nat-gateway`).

Deploy the file server

Now that our EC2 machine has internet access, let's connect to it and deploy the file server.

First of all, SSH into the machine **using its private IP** and **while being connected to the VPN**:

```
ssh -i ~/.ssh/aws-keypair ubuntu@<file-server-ec2-instance-private-ip>
```

Note that we are now using `ubuntu` for the user instead of `ec2-user`.

Deploy the file server

Now we just have to run a couple commands to install and deploy the file server.

```
sudo apt update -y && sudo apt install podman-docker -y
mkdir data
docker run -p 8080:8080 --restart unless-stopped -d \
    -v data:/var/www/filegator/repository docker.io/filegator/filegator
```

That's it! You can now access the file server by going to `http://<file-server-ec2-instance-private-ip>:8080`.

Remove access to the internet

Since we didn't want machines on the private subnet to have internet access, now that the file server is configured and **we won't need to download any more software**, we can remove the access to the internet.

This would be as easy as removing the route we added to the route table. But for the sake of reducing costs we're **also going to delete the NAT Gateway**.

Remove access to the internet

To delete the route we added to the route table:

1. Search for `VPC` service and click on the first result.
2. Click on **Route Tables** on the left navigation pane.
3. Click on the default route table that is associated with `test-vpc` (the one with no name we modified before).
4. Click on the **Routes** tab down below.
5. Click on **Edit routes**.
6. Click on **Remove** on the right of the route we added.
7. Click on **Save changes**.

Remove access to the internet

To delete the NAT Gateway:

1. Search for `NAT Gateways` in the AWS Management Console and click on the first result.
2. Click on the NAT Gateway we created.
3. Click on **Actions** and then on **Delete NAT Gateway**.
4. Type `delete` and click on **Delete**.

Recap

During this lab we have:

- Created a VPN using the AWS Client VPN service.
- Connected to the VPN from our computer.
- Deployed a file server on a private EC2 instance by temporarily giving it access to the internet.
- Accessed the file server from our computer through the VPN.

Lots of companies use VPNs **to allow their employees to access resources inside the company network from anywhere in the world**. A practice which was accelerated by pandemic and the remote work it brought.

Try experimenting with being and not being connected to the VPN. Make sure the scenario we've setup makes sense.