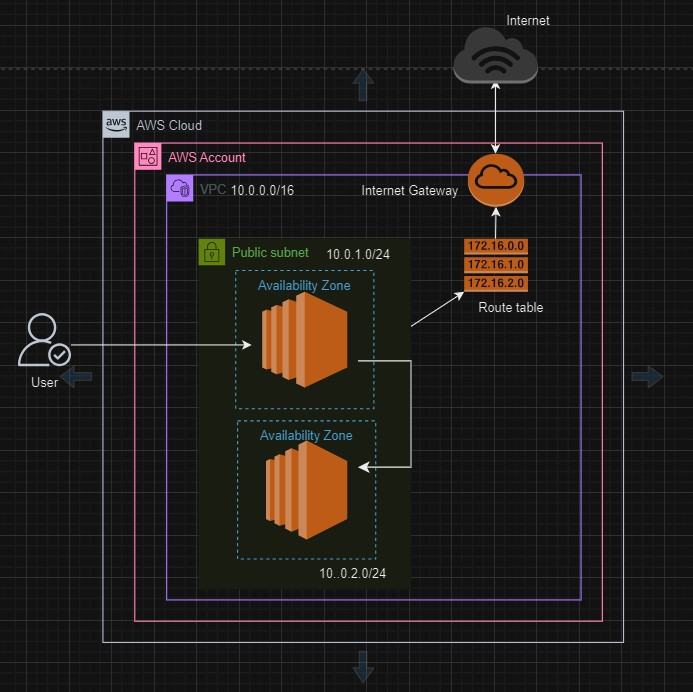
Hands on: Launching an EC2 Instance

Here I am launching an EC2 Instance in own VPC on AWS. And connecting to instance through SSH from windows local host using terminal.

**Steps**:

1. Install AWS Cli v2 in local host (Windows 11).
2. Login to your AWS account.
3. Create your own VPC for instances.
4. Launch two instances in public subnet.
5. Set permissions for keypair.
6. Connecting to first instance from local host using ssh from Windows terminal.
7. Connecting to second instance from first instance using SSH.
8. Host a “hello world” html web page in second instance.

**Reference Architecture Diagram:**

****

1. **Install AWS CLI v2**

Download and run the AWS CLI MSI installer for Windows (64-bit):

<https://awscli.amazonaws.com/AWSCLIV2.msi>

make sure that OpenSSH Client and OpenSSH Server was installed in Windows 11 Optional features (System-Optional features)

For verify the installation:

Open a Command Prompt or PowerShell window.

Run the following command to verify the installation and check the version:

*aws --version*

You should see output similar to

*aws-cli/2.x.x Python/3.x.x Windows/10 botocore/2.x.x.*

For Configure the AWS CLI:

Run the configuration command to set up your AWS credentials and default region:

*aws configure*

You will be prompted to enter your AWS Access Key ID, Secret Access Key, default region name, and default output format.

1. **Login to your AWS Account**

Navigate to the [AWS Management Console](https://aws.amazon.com/console/).

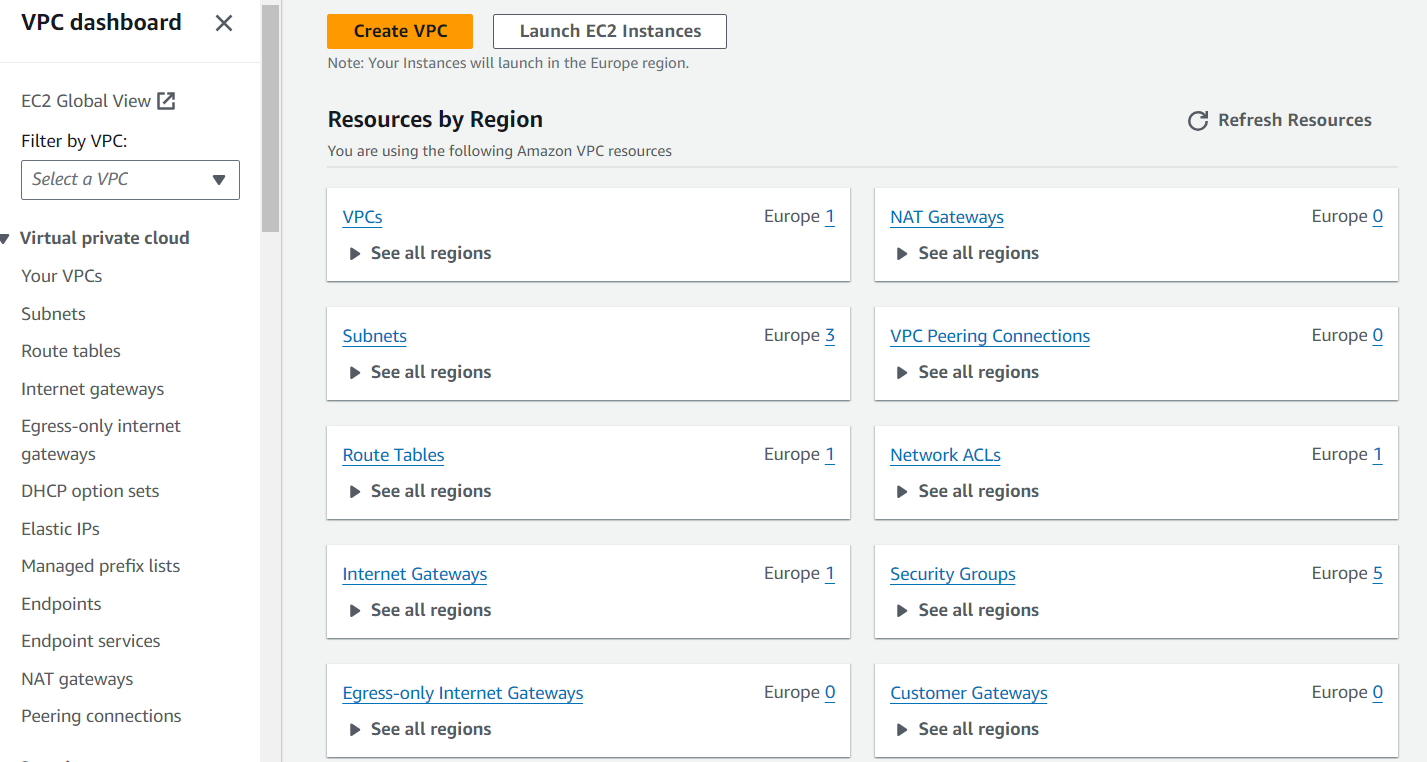
Sign in with your AWS credentials.

1. **Create your own VPC setup**

**Create VPC:**

*AWS Virtual Private Cloud (VPC) enables you to launch AWS resources into a virtual network that you've defined. It provides complete control over your virtual networking environment, including resource placement, connectivity, and security*

1. In the Services menu, under Networking & Content Delivery, select VPC



1. Click on Create VPC

You can use the VPC wizard which provides common network setups, or click on ‘Create VPC only’ to create a custom VPC.

1. **Name tag:** Provide a name for your VPC. Here I use “my-cloud-vpc”
2. **IPv4 CIDR block:** Enter the CIDR block for your VPC

Here we use the CIDR IP block is 10.0.0.0/16

**IPv6 CIDR block:** Optional, you can associate an IPv6 CIDR block. And keep the Tenancy as Default

1. Click on Create VPC

After the VPC is created, you will see a success message.

**Configure VPC:**

**Create an Internet Gateway**

*AWS Internet Gateway (IGW) allows communication between instances in your VPC and the internet*

1. Go to Internet Gateways in the VPC dashboard.
2. Click on Create internet gateway.
3. Provide a name and click Create, here I use “my-cloud-IGW”
4. Attach the Internet Gateway to your VPC.

Attach the ‘my-cloud-IGW’ to ‘my-cloud-vpc’ by using actions – attach to vpc.

**Create Subnets**

*Subnetworks within the VPC. You can define subnets in specific Availability Zones (AZs)*

1. Navigate to the "Subnets" section of the VPC Dashboard.
2. Click on create subnet
3. Give your subnet a name and select the VPC you want to create the subnet in.

Here;

Select VPC as ‘my-cloud-vpc’

We have to create two subnets as public subnet in different AZ’s with IP CIDR 10.0.1.0/24 and 10.0.2.0/24.

(Define the IPv4 CIDR block for the subnet, block must be a subset of the VPC's CIDR block.)

For identify the subnets in reach region we can use a naming pattern, such as ‘my-cloud-vpc-public-1a’ and ‘my-cloud-vpc-public-1b’

**Create a Route Table**

*Define how traffic is directed within the VPC and to external destinations*

1. Go to the "Route Tables" section in the VPC Dashboard.
2. Click "Create route table".
3. Enter a name tag for the route table and select your VPC.

Here ‘my-cloud-vpc-RT’

1. Click "Create".
2. Select the route table and click on the "Routes" tab, then "Edit routes".
3. Add a route to the internet (0.0.0.0/0) and target the internet gateway you created.
4. Click "Save routes".

Associate the route table with your subnets:

Select the route table.

Click on the "Subnet Associations" tab.

Click "Edit subnet associations".

Select the subnets you want to associate with this route table.

Here we select both the subnets ‘my-cloud-vpc-public-1a’ and ‘my-cloud-vpc-public-1b’ – Click ‘save’.

*Default Route:*

*By default, every VPC has a main route table associated with it, and each subnet that you create in the VPC is automatically associated with this main route table unless you explicitly associate it with another route table.*

*This main route table typically contains routes that direct traffic within the VPC using local routing.*

*Subnet Association to the Internet:*

*For a subnet to have internet access, it must be associated with a route table that has a route directing traffic to an internet gateway.*

*This involves:*

*A public subnet is a subnet that has a route in its route table directing traffic destined for the internet (0.0.0.0/0) to the internet gateway (IGW).*

*Add a route with destination 0.0.0.0/0 (the internet) and target the internet gateway (here ‘my-cloud-IGW’).*

*Associate this route table with the subnet you want to make public.*

**4. Launch Instances in VPC subnets**

Here we have to create instances in each subnet in corresponding AZ’s

1. Navigate to the EC2 Dashboard.
2. Click "Launch Instance".
3. Steps;

**Give Instance’s name**

should be ‘my-cloud-ec2-1a’ (for subnet ‘my-cloud-vpc-public-1a’) and ‘my-cloud-ec2-1b’ (for subnet ‘my-cloud-vpc-public-1b’)

**Select AMI** (Amazon machine image) and Instance type

*An Amazon Machine Image (AMI) provides the information required to launch an instance in AWS. It includes the operating system, application server, and applications. AMIs are fundamental to deploying EC2 instances and can be customized to meet specific needs.*

Here

AMI: Amazon Linux 2023

Type: t3.micro (both are free tier eligible)

**Create a Keypair**

*An AWS key pair is used to securely connect to your Amazon EC2 instances. When you launch an EC2 instance, you can associate it with a key pair, which consists of a public key (stored by AWS) and a private key (downloaded and stored securely by you). You use the private key which downloaded on our local host to securely access your instance via SSH (for Linux instances) or RDP (for Windows instances).*

**Name**: Provide a name for your key pair (here my-cloud-ec2-1a-key and my-cloud-ec2-1b-key respectively for instances)

**Key pair type**: Choose the type of key pair (RSA or ED25519). RSA is commonly used.

**Private key file format**: Choose PEM (for SSH clients) or PPK (for PuTTY). PEM is the most common format.

Click "Create key pair".

**Configure Network**

Click on ‘’Edit’’

* Under VPC, select our VPC - “my-cloud-vpc”
* Under Subnet Preference:

Choose “my-cloud-vpc-public-1a” for “my-cloud-ec2-1a”

And “my-cloud-vpc-public-1b” for “my-cloud-ec2-1b”

* Enable Auto-assign public IP

*Auto-assigning a public IP to an Amazon EC2 instance allows it to be reachable from the internet. This is particularly useful for instances that need to interact with external users or services.*

**Create a Security Group**

* Navigate to the EC2 Dashboard
* Under Network & Security section on the left-hand sidebar
* Click on the Create Security Group button
* Give a name for your SG – here ‘my-cloud-EC2-SG’
* VPC: Select the VPC (Virtual Private Cloud) where you want to create the security group.

**Add Inbound Rules:**

*Inbound rules control the incoming traffic to your instances.*

* Click on the Add Rule button to add a new inbound rule.
* Type: Select the type of traffic (here we use SSH, http and https).
* Protocol: This is usually auto-filled based on the type.
* Port Range: Specify the port range (22 for SSH ,80 for HTTP and 443 for HTTPS).
* Source: Specify the source of the traffic (here select 0.0.0.0 – from all IP).

#### Add Outbound Rules

*Outbound rules control the outgoing traffic from your instances.*

*By default, all outbound traffic is allowed*

Click on the Create security group button.

**Continuation for VPC N/W Config.**

* Under Firewall (security groups)
* Select existing security group - ‘my-cloud-EC2-SG’

**Create EBS Volume**

Under Configure Storage; here we use 8GB EBS as root volume, there is no additional storage we provide.

**Add Userdata**

*EC2 user data is a feature in Amazon Web Services (AWS) that allows you to run scripts and provide instance-specific configuration information during the launch process of an EC2 (Elastic Compute Cloud) instance. This data is commonly used to automate the setup of instances, such as installing software, applying updates, configuring services, or running custom scripts***.**

Here we use below userdata for our EC2 instance (second instance launch without userdata);

**#!/bin/bash**

**# Use this for your user data (script from top to bottom)**

**# install httpd (Linux 2 version)**

**yum update -y**

**yum install -y httpd**

**systemctl start httpd**

**systemctl enable httpd**

**echo "<h1>Hello World from $(hostname -f)</h1>" > /var/www/html/index.html**

**Explanation for each line;**

* **#!/bin/bash**

*This is the shebang line, which tells the system that the script should be run using the Bash shell. It ensures that the script is interpreted correctly.*

* **# Use this for your user data (script from top to bottom)**

*This is a comment line. Comments start with # in Bash scripts and are ignored by the interpreter. This line is meant for human readers to understand the script's purpose.*

* **# install httpd (Linux 2 version)**

*It states that the script will install the httpd package, which is the Apache web server, specifically for Amazon Linux 2.*

* **yum update -y**

*This command updates all the packages on the instance to their latest versions. The -y flag automatically answers "yes" to any prompts that appear during the update process.*

* **yum install -y httpd**

*This command installs the httpd package (Apache web server). The -y flag automatically answers "yes" to any installation prompts.*

* **systemctl start httpd**

*This command configures the Apache web server service to start automatically at boot time.*

* **systemctl enable httpd**

*This command configures the Apache web server service to start automatically at boot time.*

* **echo "<h1>Hello World from $(hostname -f)</h1>" > /var/www/html/index.html**

*This command creates an HTML file at* ***/var/www/html/index.html*** *with the content* ***<h1>Hello World from $(hostname -f)</h1>.***

* ***echo "<h1>Hello World from $(hostname -f)</h1>"*** *prints the string to standard output.*
* ***$(hostname -f)*** *is a command substitution that gets the fully qualified domain name (FQDN) of the instance and inserts it into the string.*
* ***> /var/www/html/index.html redirects*** *the output of the echo command to the file /var/www/html/index.html, creating the file if it doesn't exist or overwriting it if it does.*

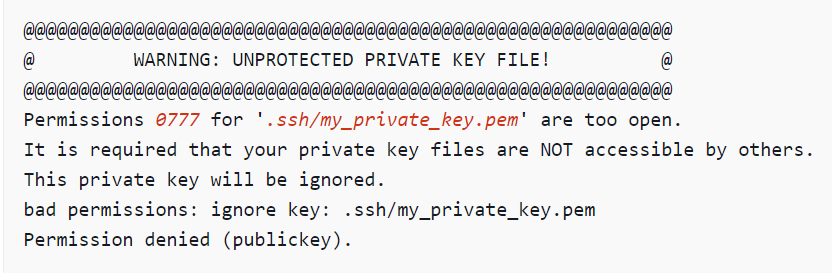
This user data script automates the process of setting up a web server on a new EC2 instance, ensuring that the web server is up and running with a default web page as soon as the instance launches.

Click on Launch Instance.

**5. Set permissions for keypair.**

We have downloaded two key pair at the time of instance creation to our Windows local host.

Your private key file must be protected from read and write operations from any other users. If your private key can be read or written to by anyone but you, then SSH ignores your key and you see the following warning message below.



The above example uses the private key .ssh/my\_private\_key.pem with file permissions of 0777, which allow anyone to read or write to this file. This permission level is very insecure, and so SSH ignores this key.

**If you are connecting from macOS or Linux**, run the following command to fix this error, substituting the path for your private key file.

**chmod 0400 .ssh/my\_private\_key.pem**

Using chmod 400 on a file:

* **Owner**: Can read the file.
* **Group**: No access.
* **Others**: No access.

This command is useful for securing sensitive files, ensuring that only the owner can read them, and preventing any modifications or executions by any user, including the owner.

**Here we are connecting from Windows** using my-cloud-ec2-1a-key.pem to our first EC2 instance, perform the following steps on your local computer.

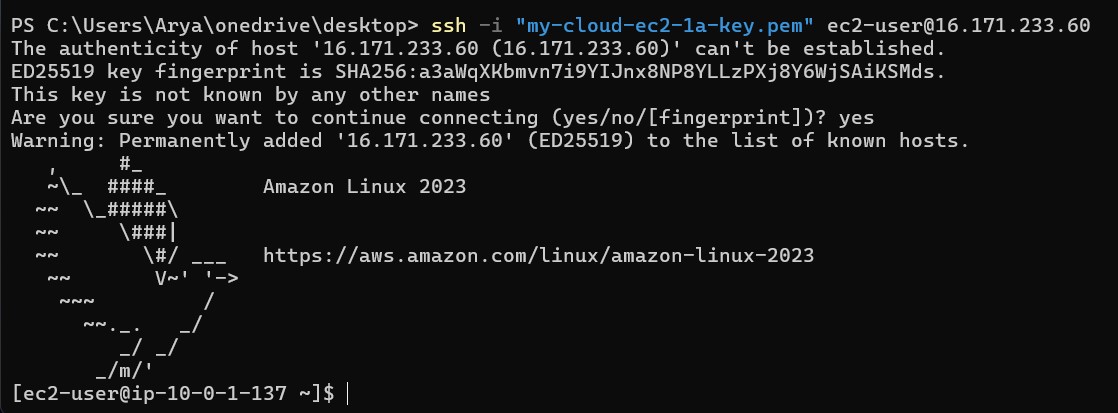
1. Navigate to your .pem file.
2. Right-click on the .pem file and select **Properties**.
3. Choose the **Security** tab.
4. Select **Advanced**.
5. Verify that you are the owner of the file. If not, change the owner to your username.
6. Select **Disable inheritance** and **Remove all inherited permissions from this object**.
7. Select **Add**, **Select a principal**, enter your username, and select **OK**.
8. From the **Permission Entry** window, grant **Read** permissions and select **OK**.
9. Click **Apply** to ensure all settings are saved.
10. Select **OK** to close the **Advanced Security Settings** window.
11. Select **OK** to close the **Properties** window.
12. You should be able to connect to your Linux instance from Windows via SSH.

**6. Connecting to first instance**

* Open Windows Terminal
* Change directory to the keypair path using cd command
* Use the SSH command to connect

ssh -i my-cloud-ec2-1a-key.pem ec2-user@ your-instance-public-ip/dns

* It will log into EC2 instance page, here you can verify the public IP in hostname as same as from console.



Here we can see the private IP is taken from the given CIDR range for the specific subnet.

**7.Connecting to second instance from first instance using SSH.**

**Copy the second instances key to first instance**

* 1. We should create a **my-cloud-ec2-1b-key.pem** file in the second instance’s home directory.
  2. Then we need to copy the private key from the local host to this file.
  3. Use below command for above mentioned steps;

**nano my-cloud-ec2-1b-key.pem**

this will create a file named ‘nano my-cloud-ec2-1b-key.pem, and open the file for editing. Copy the contents from second instance key from the local host and save and quit from the editor.

**cat my-cloud-ec2-1b-key.pem**

Above command shows the contents of that file, so er can verify from the ordinal file.

**Change the permission of the key**

Navigate to the key directory – (home directory)

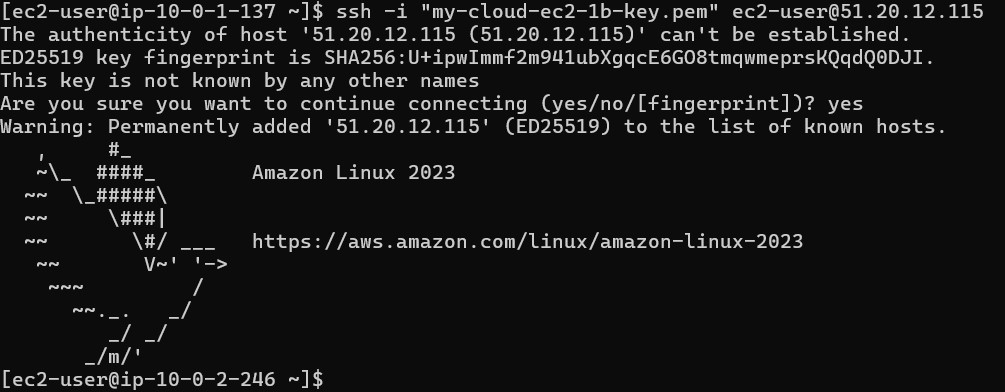
**chmod 400 my-cloud-ec2-1b-key.pem**

*This command ensures that only the owner can read the file, and no one else can read, write, or execute it. This is necessary to securely use the key with SSH.*

**Connecting to the second Instance**

**ssh -i my-cloud-ec2-1a-key.pem ec2-user@ your-instance-public-ip/dns**

this command make connect to second Instance.



Here we can see the private IP 10.0.2.246 taken from the specific CIDR range.

1. **Host a “hello world” html web page in second instance.**

Run the below commands;

1. **sudo yum update -y**

YUM will:

Refresh the list of available packages and their versions.

Check for any updates to the installed packages.

Download and install the latest versions of the packages.

1. **sudo yum install -y httpd**

This command installs the Apache web server package.

1. **sudo systemctl start httpd**

This command starts the Apache web server**.**

1. **sudo systemctl enable httpd**

This command ensures that the Apache service starts automatically when the system boots.

1. **echo "<h1>Hello World from $(hostname -f)</h1>" | sudo tee /var/www/html/index.html**

This command creates an HTML file with a "Hello World" message that includes the fully qualified domain name (FQDN) of the server. The $(hostname -f) command is executed to get the server's FQDN, and the tee command writes the output to /var/www/html/index.html.

Here I have faced an error;

First, I have user the command **echo "<h1>Hello World from $(hostname -f)</h1>" >/var/www/html/index.html**

Without necessary permissions to write to the /var/www/html directory, then I got the below error.

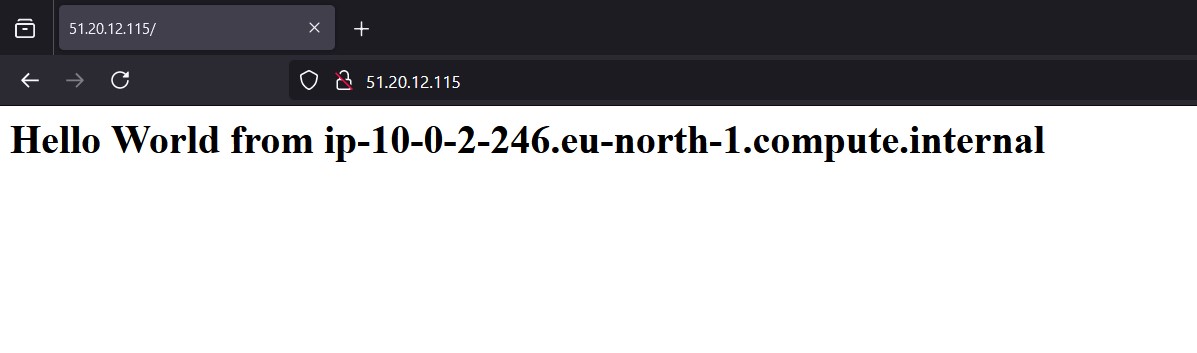


Then I changed the command with sudo privilege and then it works.

Correct command;



“After we can run public IP of second instance in web browser it will show the below page”.

****

**--End--**