

So I know by now you've heard of EC2 instances—as I've been using it to disturb you within these few days. But are you aware that apart from what I demonstrated by going to the console, clicking on 'Launch Instance', selecting the instance type, and so on, there are other ways you can get your instance running—even in just a few minutes? And not just one. If you need to launch ten or a hundred, it's possible to do them all at once.

Come with me—let me take you through. Today, we're going to launch EC2 instances using 6 different methods:

1. AWS Management Console
2. EC2 User Data Script
3. AWS CLI (Command Line Interface)
4. Terraform (Infrastructure as Code)
5. AWS CloudFormation (Declarative Templates)
6. AWS CDK (Cloud Development Kit with Python)

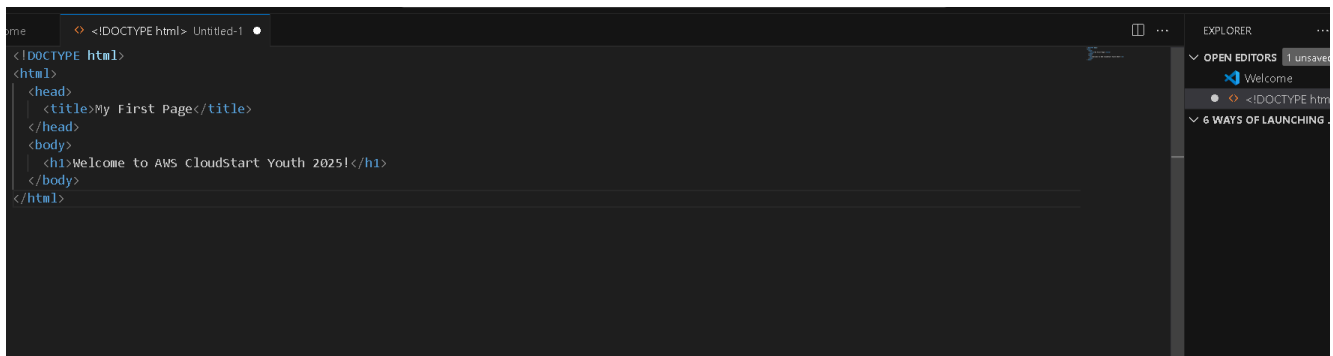
And in each method, we're going to install Apache, which is a web server, and create an HTML file, which is the basic structure of a webpage.

So first, let's break down what Apache and HTML are.

Apache is a software that runs on your server and listens for requests from the internet. When someone types your server's IP address into a browser, Apache responds and shows a web page stored on that server. But for this to work, your EC2 instance must have a public IP address. A public IP allows the server to be accessed from the internet. Without it, even if Apache is properly installed, no one outside your private network can see the page.

It's important to know when to use a public IP and when not to. For example, a hospital storing sensitive patient data wouldn't want to expose its systems to the public internet. In that case, they would use a private IP address and only allow internal access or access through a secure VPN. But for a public website like your personal portfolio or a school's homepage, a public IP is needed so anyone from anywhere can access the content. Think of it like a receptionist who welcomes visitors at your front door and hands them the right paper or brochure—in this case, your webpage.

HTML stands for HyperText Markup Language. It's the most basic building block of the Web. It describes and defines the content of a web page. A simple HTML file can contain text, images, links, and buttons. When you open an HTML file in a browser, the browser reads the tags and displays them as a nice, structured page. For example:



```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Page</title>
  </head>
  <body>
    <h1>Welcome to AWS CloudStart Youth 2025!</h1>
  </body>
</html>
```

Now, let's look at how to get a server on AWS, install Apache, and host this kind of page using 6 different methods.

Method 3: Using AWS CLI (Command Line Interface)

The AWS CLI is a tool that allows you to control your AWS services from your terminal. Instead of clicking through the AWS Console, you can type commands to launch servers, create users, and much more.

If you're using this for the first time, you need to install the AWS CLI on your machine. To do this, visit the official AWS CLI documentation site at <https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html> and download the installer that matches your operating system—Windows, macOS, or Linux. Follow the installation instructions provided there step by step.

Once installed, open your terminal and verify the installation by typing:

```
aws --version
```

You should see the installed version of the CLI printed out.

Next, you need to configure the CLI so it can communicate with your AWS account. Run:

```
aws configure
```

It will prompt you to enter four things:

1. Your AWS Access Key ID
2. Your AWS Secret Access Key
3. Your default AWS region (for example, us-east-1)
4. Your default output format (you can type json)

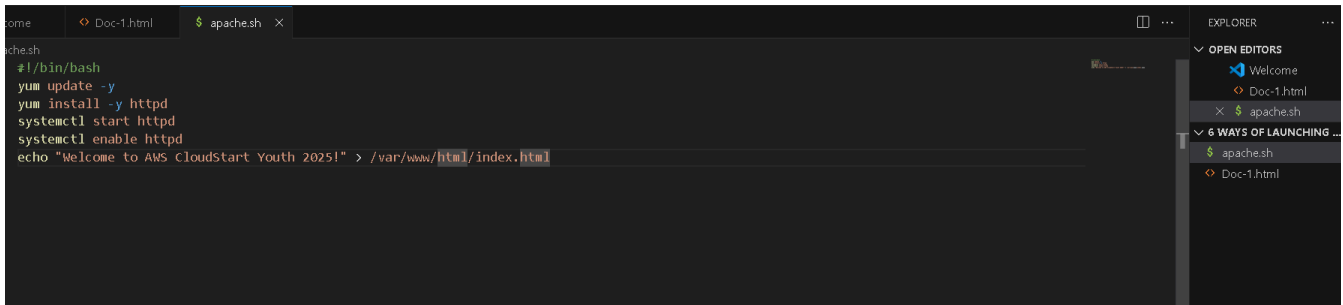
You can get your access key and secret key from the AWS Console. Go to the top-right corner of the AWS Console, click on your account name, and choose 'My Security Credentials'. Under the 'Access keys' section, create a new key if you don't have one. Download it and store it securely—you will not be able to see the secret key again after you close that window.

It is very important to store your credentials securely. Never share them or upload them publicly. Use environment variables or encrypted credential managers when possible. After installation, configure your CLI by running:

aws configure

This will ask you for your AWS access key, secret key, region, and output format. Once this is set up, you can begin to use the CLI to manage your AWS infrastructure.

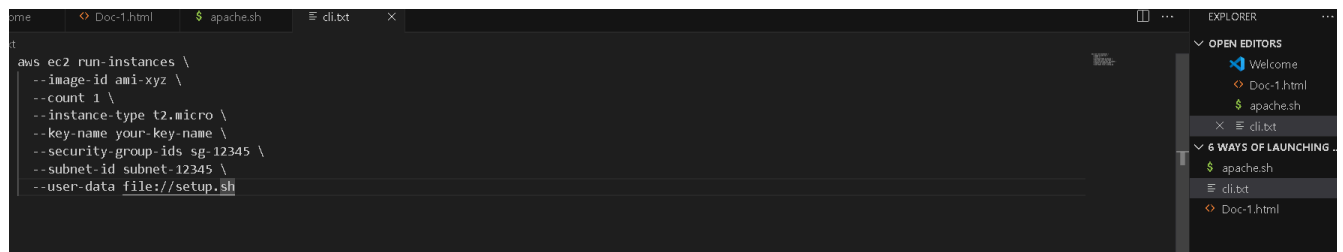
Let's create a script that will automatically install Apache and create our HTML page. Save the following as setup.sh:



```
#!/bin/bash
yum update -y
yum install -y httpd
systemctl start httpd
systemctl enable httpd
echo "Welcome to AWS CloudStart Youth 2025!" > /var/www/html/index.html
```

This script tells the EC2 instance to first update its software packages, then install Apache (httpd is the name of the Apache package), start the web server, enable it to start on reboot, and finally create a basic HTML page.

Now let's use the AWS CLI to launch our instance:



```
aws ec2 run-instances \
  --image-id ami-xyz \
  --count 1 \
  --instance-type t2.micro \
  --key-name your-key-name \
  --security-group-ids sg-12345 \
  --subnet-id subnet-12345 \
  --user-data file://setup.sh
```

Let's break this down:

- --image-id specifies the AMI (Amazon Machine Image), which is the template of the OS we are launching.
- --count 1 tells AWS to launch one instance.
- --instance-type defines the size of the server. We use t2.micro because it's free-tier eligible.
- --key-name is the name of your SSH key, needed to connect to the instance.
- --security-group-ids allows traffic rules, like HTTP and SSH access.
- --subnet-id places the instance inside a Virtual Private Cloud (VPC) subnet.
- --user-data is the script that installs Apache and creates the HTML page automatically.

After this runs, your instance should be up and running. Make sure the security group allows HTTP traffic and that the instance has a public IP address. Without a public IP, it cannot be accessed from the internet. Open the IP in a browser and your page should appear.

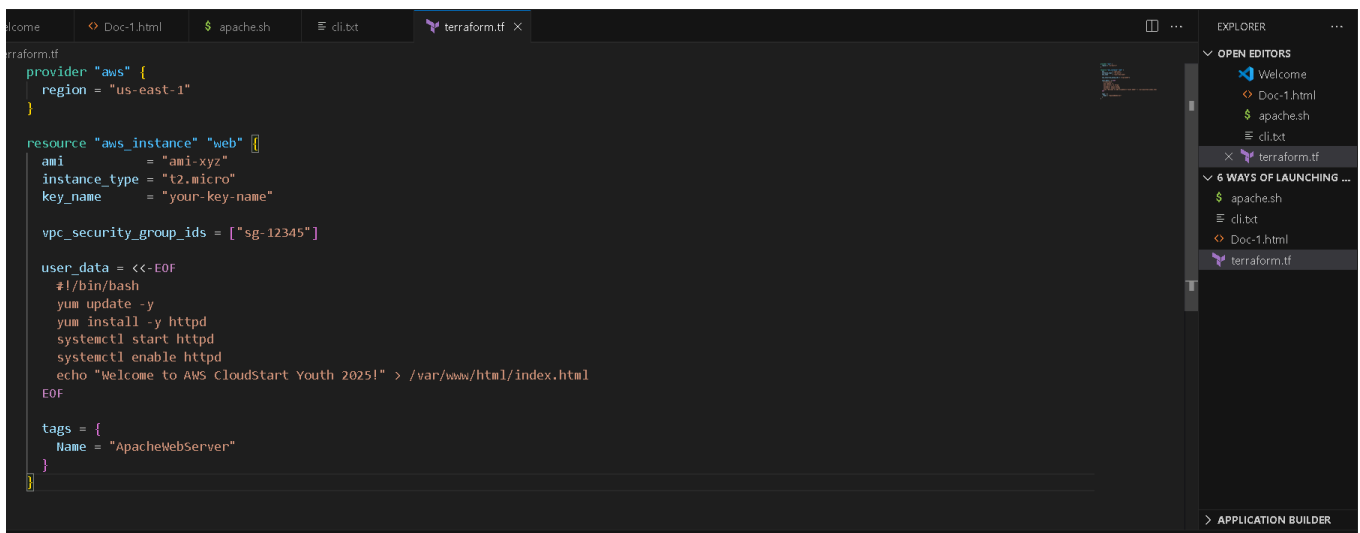
Method 4: Using Terraform

Terraform is an Infrastructure as Code (IaC) tool that allows you to define your cloud resources in text files and deploy them automatically. Instead of clicking in the console or typing CLI commands, you write a file that describes your infrastructure.

Why is this important? Imagine you work at Netflix or Google, where hundreds of servers are launched and destroyed every day. Manually clicking or typing would be slow and prone to errors. Terraform lets you manage all this as reusable code.

Let's set it up:

First, install Terraform from terraform.io. Then create a new folder and inside it create a file called `main.tf`. This is your configuration file.



```
terraform.tf
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "web" {
  ami           = "ami-xyz"
  instance_type = "t2.micro"
  key_name      = "your-key-name"

  vpc_security_group_ids = ["sg-12345"]

  user_data = <<-EOF
  #!/bin/bash
  yum update -y
  yum install -y httpd
  systemctl start httpd
  systemctl enable httpd
  echo "Welcome to AWS CloudStart Youth 2025!" > /var/www/html/index.html
  EOF

  tags = {
    Name = "ApacheWebServer"
  }
}
```

Explanation:

- provider tells Terraform we're working with AWS and which region to deploy to.
- aws_instance defines the virtual machine we want to launch. This is like telling Terraform: "Please create a new EC2 instance for me."
- ami is the ID of the operating system image that the server will run. Think of it as choosing what kind of computer you're launching—Amazon Linux, Ubuntu, Windows, etc.
- instance_type defines the size and capacity of the server. t2.micro is a small instance and fits within the free tier.
- key_name allows you to securely connect to your instance through SSH.
- vpc_security_group_ids defines which security group the instance will use. A security group acts like a virtual firewall—it controls what kind of traffic is allowed to reach your instance. Here, it must allow HTTP (port 80) and SSH (port 22) so we can access the webpage and connect via terminal.

- user_data contains a script that runs automatically when the server starts. It updates the system, installs Apache, starts it, and creates a sample HTML page.
- tags are metadata that help you label and organize your AWS resources.
- ami, instance_type, and key_name are used the same way as with the CLI.
- vpc_security_group_ids allows access (like SSH and HTTP).
- user_data contains the script to set up Apache.
- tags help you identify the instance in the AWS Console.

Now open your terminal in this folder and run:

```
terraform init
```

This sets up Terraform.

Then:

```
terraform apply
```

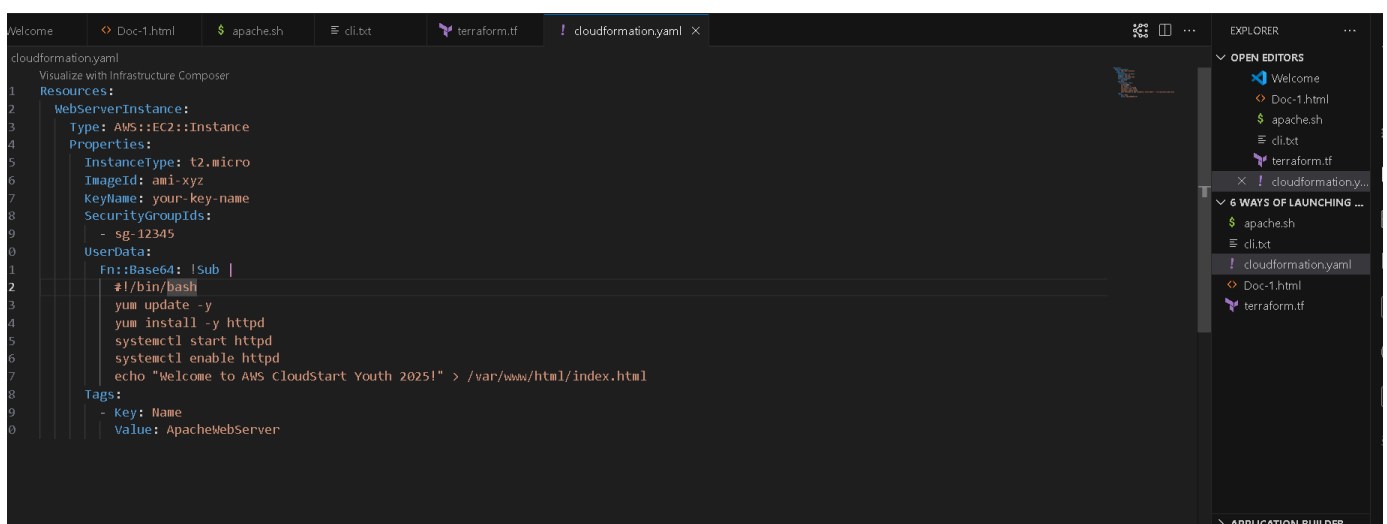
Type "yes" when prompted. Terraform will launch the EC2 instance and handle everything.

When done, go to your EC2 Dashboard, find the public IP of the instance, and visit it in your browser. The webpage will display your message.

Method 5: Using AWS CloudFormation

CloudFormation is a service that lets you model and set up your AWS resources using templates. It's like writing down everything you want in a YAML or JSON file and letting AWS handle the setup. This method is used in large enterprises to ensure environments are deployed the same way every time—safely and predictably.

Let's take a look at a simple CloudFormation YAML template that launches an EC2 instance, installs Apache, and sets up an HTML page.



Explanation:

- Resources is the section where you define all the infrastructure you want CloudFormation to create.
- WebServerInstance is a name we give to our EC2 instance.
- Type: AWS::EC2::Instance tells CloudFormation that we want to create an EC2 virtual machine.
- InstanceType, ImageId, and KeyName work just like before.
- SecurityGroupIds connects our instance to a security group so it can receive web traffic and SSH.
- UserData is used to run the Apache setup script. We use Fn::Base64 because AWS requires the script to be encoded.
- Tags are used to label the server in your AWS Console.

To deploy this, go to the AWS Console → CloudFormation → Create Stack → Upload this template file → Follow the prompts → Click “Create Stack.”

CloudFormation will automatically create the instance and apply all the configurations.

(Next, we will explain CDK with clear step-by-step breakdowns.)