

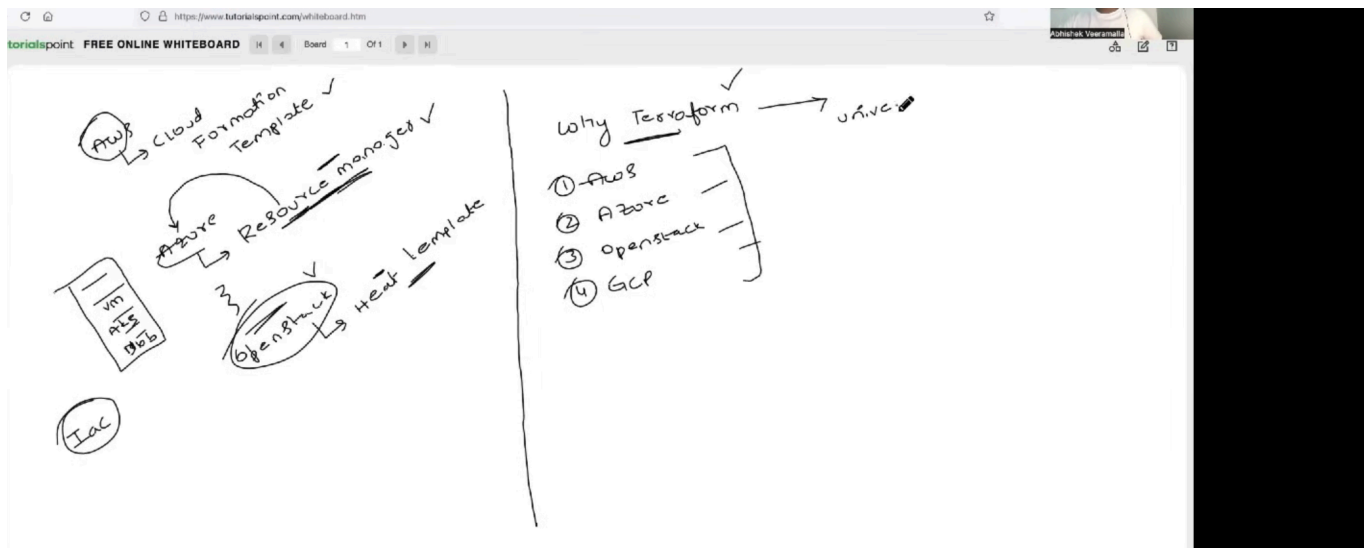
AWS-Terraform writeup (From Abhishek Veramulla Youtube Channel).

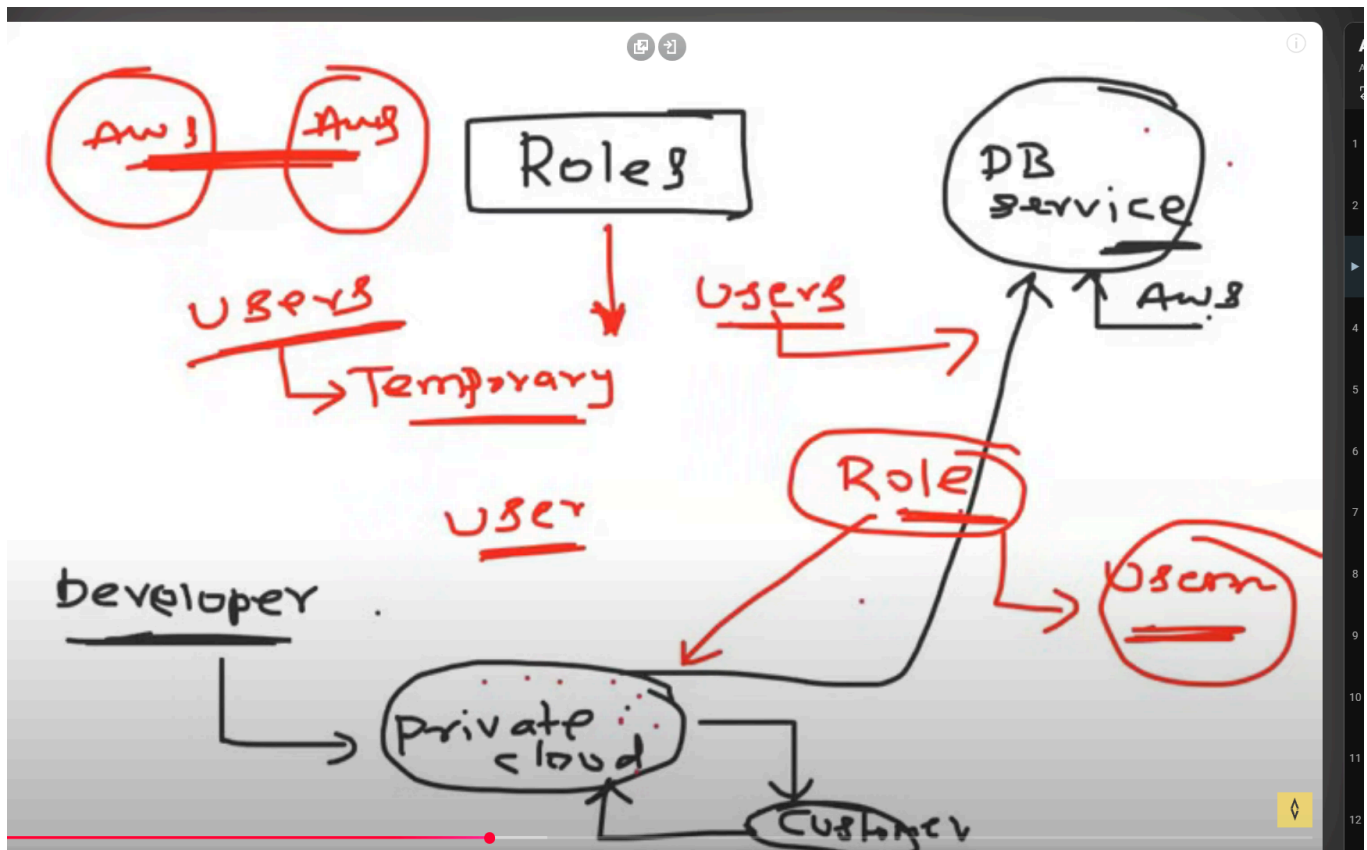
Amaan Abbas

This is my write-up of AWS-Terraform that I was doing over the Summer of 2025. I found cloud security to be very interesting and delved into it . I used resources like Google and also mainly Abhishek Veramulla's YouTube Channel. All the images are from me implementing my learning.

☁ AWS Cloud Types

- **Public Cloud:** Managed by external providers like AWS, Azure, etc. You don't manage hardware.
- **Private Cloud:** You manage your own infrastructure and servers.





Why Use AWS?

- AWS is widely adopted because it **pioneered cloud services**.
- It offers **scalability**, **pay-as-you-go** pricing, and a large ecosystem.



AWS IAM (Identity and Access Management)

- IAM controls **who can access AWS** and **what they can do**.
- **Groups**: Help organize users and assign permissions in bulk.
- **Roles**: Temporary access with defined permissions (e.g., for EC2 instances or cross-account access).
- When you create a user, AWS gives you a `.csv` file with login credentials:

plaintext

Copy code

```
User name Password Console sign-in URL test-user-xxx W!xxxx%
https://xxxxxxx082xx6.signin.aws.amazon.com/console
```

IAM Policy (JSON Example)

json

Copy code

```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": [ "s3:*",  
"s3-object-lambda:*" ], "Resource": "*" } ] }
```

- This policy gives full access to **S3** and **S3 Object Lambda**.

```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Sid": "block_access",  
6       "Effect": "Deny",  
7       "Principal": "*",  
8       "Action": "s3:*",  
9       "Resource": [  
10        "arn:aws:s3:::itzmeamaan-s3-prod-example.com"  
11      ]  
12    }  
13  ]  
14 }
```

EC2 – Elastic Compute Cloud

- Virtual servers in the cloud where you can define **CPU**, **RAM**, **disk**, etc.
- An Amazon EC2 instance is a virtual server within Amazon's Elastic Compute Cloud (EC2) that allows users to run applications and workloads on the AWS cloud.
- Common Types t2.micro (free tier), t3.medium, m5.large, c6g.large
- How to use an EC2 free version while using for a personal project?

To use EC2 under the AWS Free Tier, choose the t2.micro or t3.micro instance types, which are free for up to 750 hours per month. Use Free Tier-eligible AMIs like Amazon Linux 2 or Ubuntu 20.04. You get up to 30 GB of SSD (gp2) storage for free. It's best to deploy in less busy regions like us-east-1 or us-west-2 for better performance.

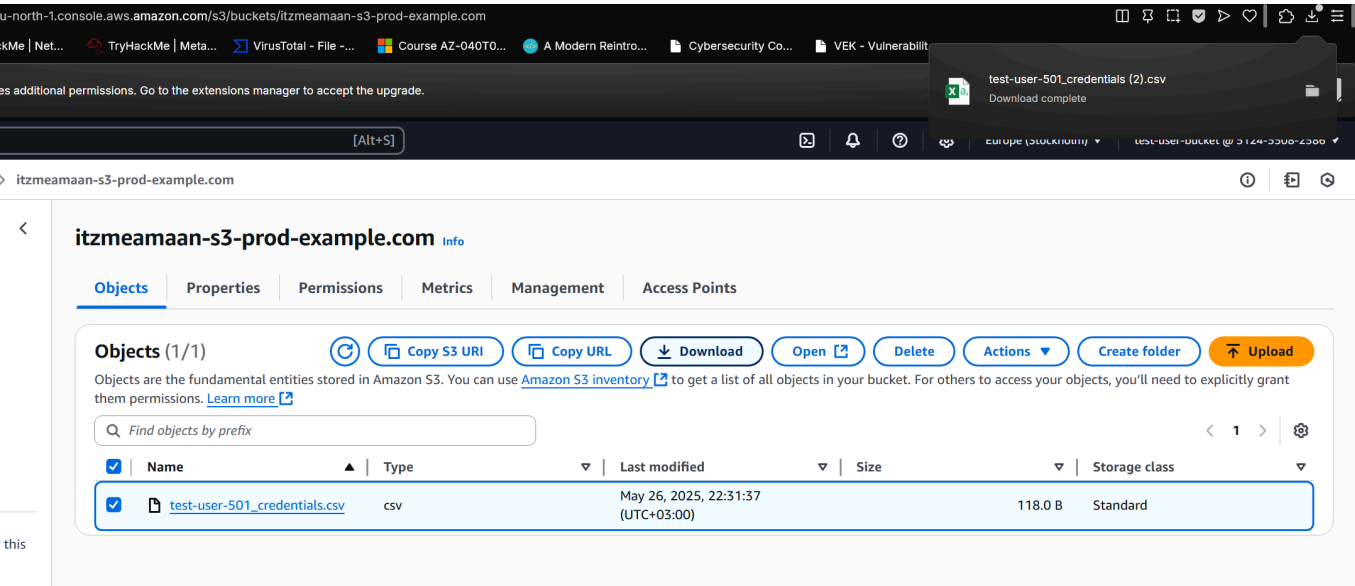
To save hours, stop your instance when it's not in use — but don't terminate it unless you're okay losing data. In your security group, only open necessary ports like 22 (SSH) and 80/443

(web traffic). CloudWatch basic monitoring is free and helps track usage.

Avoid using Elastic IPs unless needed, because you'll be charged if they're not attached to a running instance. Snapshots (backups) aren't free, so use them carefully. Finally, always set up an AWS Budget to stay within the free tier and avoid surprise charges.

AWS S3 (Simple Storage Service)

Feature	Description
Purpose	Object storage for any type of data
Common Uses	Website hosting, backups, static files, logs
Storage Classes	Standard, Intelligent-Tiering, Glacier
Access Control	IAM policies, bucket policies, ACLs
Encryption	SSE-S3, SSE-KMS, SSE-C, client-side
Versioning	Maintains multiple object versions
Upload Methods	Web console, CLI, SDKs, REST API, <code>aws s3 cp</code>
Bucket	A container for storing objects (like folders in cloud)
Pricing	Based on storage, requests (PUT/GET), data transfer



The screenshot shows the AWS S3 console interface. At the top, there's a browser address bar with the URL 's3-north-1.console.aws.amazon.com/s3/buckets/itzmeamaan-s3-prod-example.com'. Below the browser, the console header shows the bucket name 'itzmeamaan-s3-prod-example.com' and tabs for 'Objects', 'Properties', 'Permissions', 'Metrics', 'Management', and 'Access Points'. The 'Objects' tab is active, displaying a list of objects. There is one object: 'test-user-501_credentials.csv', which is a CSV file, 118.0 B in size, and stored in the 'Standard' storage class. The object was last modified on May 26, 2025, at 22:31:37 (UTC+03:00). Above the object list, there are buttons for 'Copy S3 URI', 'Copy URL', 'Download', 'Open', 'Delete', 'Actions', 'Create folder', and 'Upload'.

Diffrence between S3 andEC2:

Feature	S3	EC2
Purpose	Store and retrieve files	Run apps or websites
Type	Storage	Compute (Virtual Machine)

Feature	S3	EC2
Costs	Pay per GB stored	Pay per hour + storage

Using PuTTY with AWS EC2 (SSH Access)

To connect securely to an Amazon EC2 Linux instance from your local Windows machine, you need to use **SSH (Secure Shell)** — a secure network protocol for logging into remote machines. AWS gives you a `.pem` **key file** (Privacy Enhanced Mail format) when you create an EC2 instance. This file is your **private key**, used to authenticate you when you try to log in.

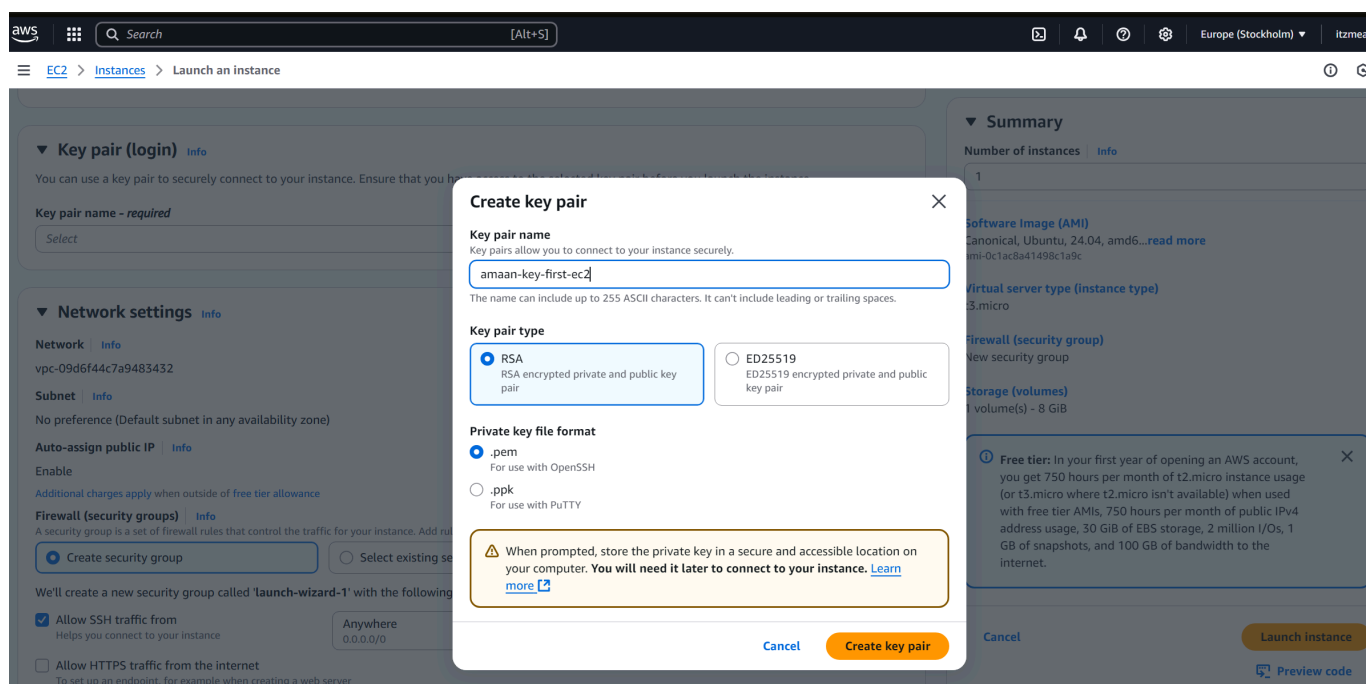
On **Windows**, the most common tool to use SSH is **PuTTY**, a lightweight terminal emulator. But PuTTY doesn't support `.pem` files directly. Instead, it uses `.ppk` (**PuTTY Private Key**) format. That's why you need a tool called **PuTTYgen** — it converts your `.pem` key to a `.ppk` format so you can use it in PuTTY.

Once you have the `.ppk` file, you open PuTTY, enter your EC2 instance's public IP, set the username (usually `ec2-user` for Amazon Linux or `ubuntu` for Ubuntu), and under **Connection** → **SSH** → **Auth**, you load your `.ppk` file. This authenticates you and opens a secure terminal window into your EC2 instance.

You can skip PuTTY entirely if you're using **Mac or Linux**, since they have built-in SSH support in the terminal and can use `.pem` files directly.

Below i used putty and show u what do:

Step 1: Convert `.pem` to `.ppk`



The screenshot shows the AWS Management Console interface during the 'Launch an instance' process. A modal dialog titled 'Create key pair' is open in the center. The dialog has three main sections: 'Key pair name' with a text input field containing 'amaan-key-first-ec2'; 'Key pair type' with two radio buttons, 'RSA' (selected) and 'ED25519'; and 'Private key file format' with two radio buttons, '.pem' (selected) and '.ppk'. A yellow warning box at the bottom of the dialog states: 'When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance.' The background shows the 'Launch an instance' wizard with sections for 'Key pair (login)', 'Network settings', and 'Firewall (security groups)'. The 'Key pair (login)' section is currently active, showing options for key pair name, type, and format.

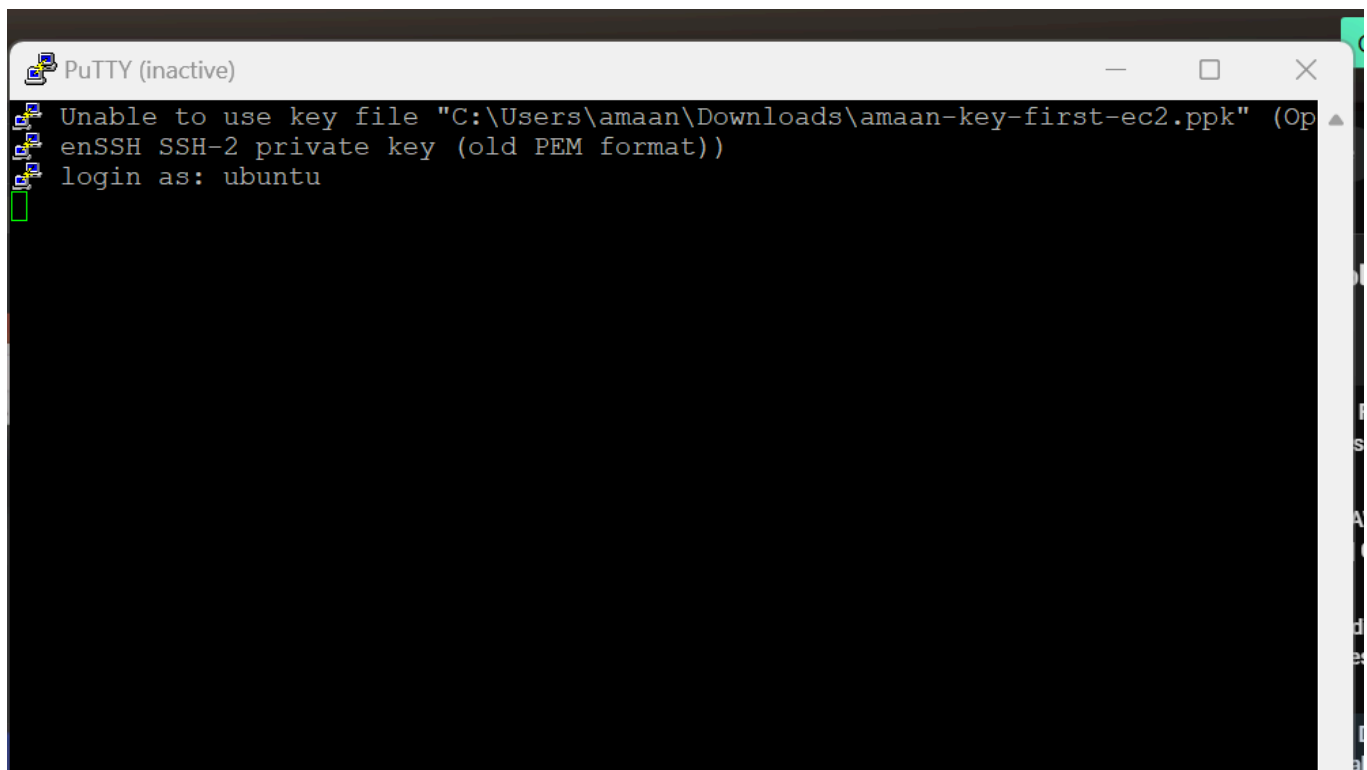
- Use **PuTTYgen**:
 - Open PuTTYgen
 - Click **Load** → select `.pem` (change file type to *All Files*)
 - Click **Save private key** → save as `.ppk`

Step 2: Configure PuTTY Session

- Under **Session**:
 - **Host Name**: `ec2-user@<your-server-ip>`
 - **Port**: 22
 - **Connection type**: SSH

Step 3: Add Private Key

- Go to: `Connection` → `SSH` → `Auth`
- Browse and select your `.ppk` file



Jenkins + AWS

Jenkins in AWS Context (usually covered after basics):

- **What Jenkins Does:**

Automates building, testing, and deploying your code helps implement Continuous Integration / Continuous Delivery (CI/CD).

- **Using Jenkins on AWS:**

- You can run Jenkins on an **EC2 instance** to manage pipelines.
- Jenkins can pull code from repositories like GitHub, build apps, run tests, then deploy to AWS services (e.g., ECS, Lambda, S3).
- Often integrated with AWS CodePipeline or CodeDeploy for smoother automation.

- **Typical Jenkins Setup Steps which we will be doing :**

1. Launch an EC2 instance (Linux recommended).
2. Install Jenkins via command line.
3. Open security group ports (default 8080) to access Jenkins UI.
4. Configure Jenkins jobs/pipelines for your project.
5. Integrate with AWS credentials to deploy resources automatically.
- 6.

I havent delved into the Jenkins part much but here is a video on yotube about this topic

<https://www.youtube.com/watch?v=6YZvp2GwT0A&t=666s>

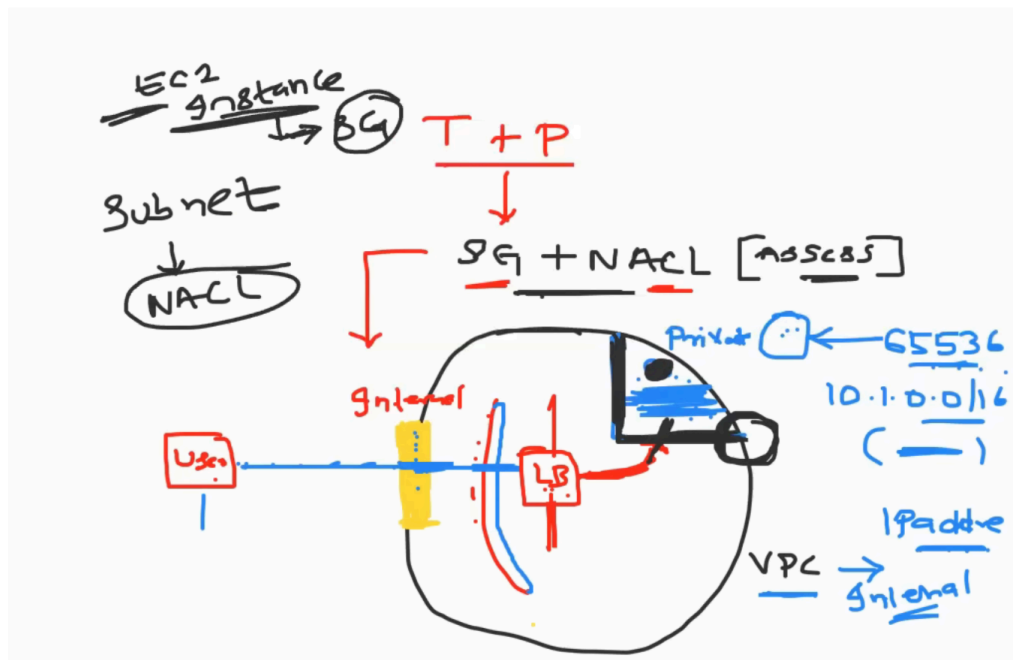
```
root@ip-172-31-47-168:~# sudo systemctl status jenkins
* jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
   Active: failed (Result: exit-code) since Sun 2025-05-25 14:32:56 UTC; 7s ago
   Process: 7043 ExecStart=/usr/bin/jenkins (code=exited, status=1/FAILURE)
   Main PID: 7043 (code=exited, status=1/FAILURE)
   CPU: 637ms

May 25 14:32:56 ip-172-31-47-168 systemd[1]: jenkins.service: Scheduled restart job, res
May 25 14:32:56 ip-172-31-47-168 systemd[1]: jenkins.service: Start request repeated too
May 25 14:32:56 ip-172-31-47-168 systemd[1]: jenkins.service: Failed with result 'exit-c
May 25 14:32:56 ip-172-31-47-168 systemd[1]: Failed to start jenkins.service - Jenkins C
lines 1-11/11 (END)
* jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
   Active: failed (Result: exit-code) since Sun 2025-05-25 14:32:56 UTC; 7s ago
   Process: 7043 ExecStart=/usr/bin/jenkins (code=exited, status=1/FAILURE)
   Main PID: 7043 (code=exited, status=1/FAILURE)
   CPU: 637ms

May 25 14:32:56 ip-172-31-47-168 systemd[1]: jenkins.service: Scheduled restart job, restart counter is at 5.
May 25 14:32:56 ip-172-31-47-168 systemd[1]: jenkins.service: Start request repeated too quickly.
May 25 14:32:56 ip-172-31-47-168 systemd[1]: jenkins.service: Failed with result 'exit-code'.
May 25 14:32:56 ip-172-31-47-168 systemd[1]: Failed to start jenkins.service - Jenkins Continuous Integration Server.
~
~
```



VPS (Virtual Private Server)

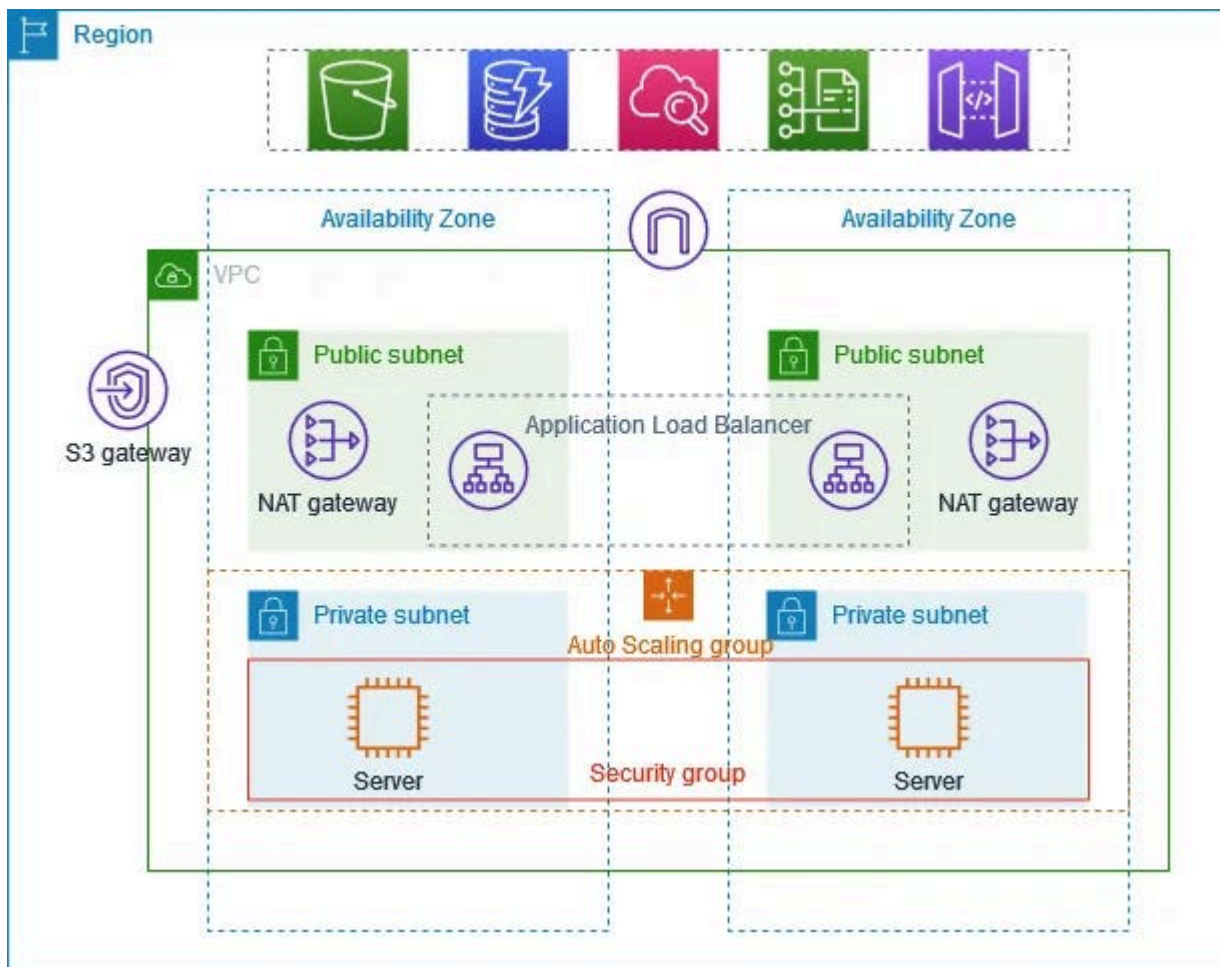


Group and NACL | Theory + Practical | AWS FREE COURSE by Abhishek | #devops #aws

- It lets you **define your own network space** within AWS.
- You control **IP addresses, subnets, route tables, and network gateways**.
- It isolates your resources (like EC2 instances) from others in the AWS cloud. This helps in security since only people we want to have access to something gets it.
- You can create **public subnets** (accessible from the internet) and **private subnets** (isolated from internet).
- VPC allows you to configure **security controls** such as security groups and network ACLs.
- It provides **networking flexibility** for connecting to your on-premises data center via VPN or Direct Connect."

In real-world use, a **VPC environment** is used to isolate and secure resources like EC2 instances, databases, and internal services. For example, a company might set up a **public subnet** for a web server that hosts their website and a **private subnet** for a database that should not be accessible from the internet. Within this VPC, they can configure **S3 buckets** to store app assets, backups, or user uploads. S3 can be made **public-facing** (like hosting a static website or serving frontend files) by setting proper **bucket policies** and linking it to a **custom domain using Route 53**. At the same time, private buckets can be used for internal development builds, with **IAM roles or pre-signed URLs** controlling access. This setup ensures a secure, scalable environment tailored to both **development** and **production** needs.

Below is an image from aws of the components needed for it:



VPC

↓

Subnets (Public & Private)

↓

Route Tables

↓

Internet Gateway (for public subnets) ↔ NAT Gateway (for private subnets)

↓

Security Groups & Network ACLs (apply to instances/subnets)

↓

EC2 Instances (inside subnets)

Below is an example of making an VPC in aws:

VPC

>

Your VPCs

>

Create VPC

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 auto-generation

Info

Enter a value for the Name tag. This value will be used to auto-generate Name tags for all resources in the VPC.

☒ Auto-generate

project

IPv4 CIDR block

Info

Determine the starting IP and the size of your VPC using CIDR notation.

10.0.0.0/16

65,536 IPs

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

IPv6 CIDR block

Info

☒ No IPv6 CIDR block

☐ Amazon-provided IPv6 CIDR block

Tenancy

Info

Default

Preview

VPC

Show details

Your AWS virtual network

project-vpc

Subnets (4)

Subnets within this VPC

eu-north-1a

A

project-subnet-public1-eu-north-1a

A

project-subnet-private1-eu-north-1a

eu-north-1b

B

project-subnet-public2-eu-north-1b

B

project-subnet-private2-eu-north-1b

Route tables (3)

Route network traffic to resources

project-rtb-public

project-rtb-private1-eu-north-1a

project-rtb-private2-eu-north-1b

Resource map

Info

VPC

Show details

Your AWS virtual network

project-vpc

Subnets (4)

Subnets within this VPC

us-east-1a

A

project-subnet-public1-us-east-1a

A

project-subnet-private1-us-east-1a

us-east-1b

B

project-subnet-public2-us-east-1b

B

project-subnet-private2-us-east-1b

Route tables (4)

Route network traffic to resources

project-rtb-private1-us-east-1a

rtb-013c3366b5336e0b5

project-rtb-private2-us-east-1b

project-rtb-public

Network connections (2)

Connections to other networks

project-igw

project-vpc-s3

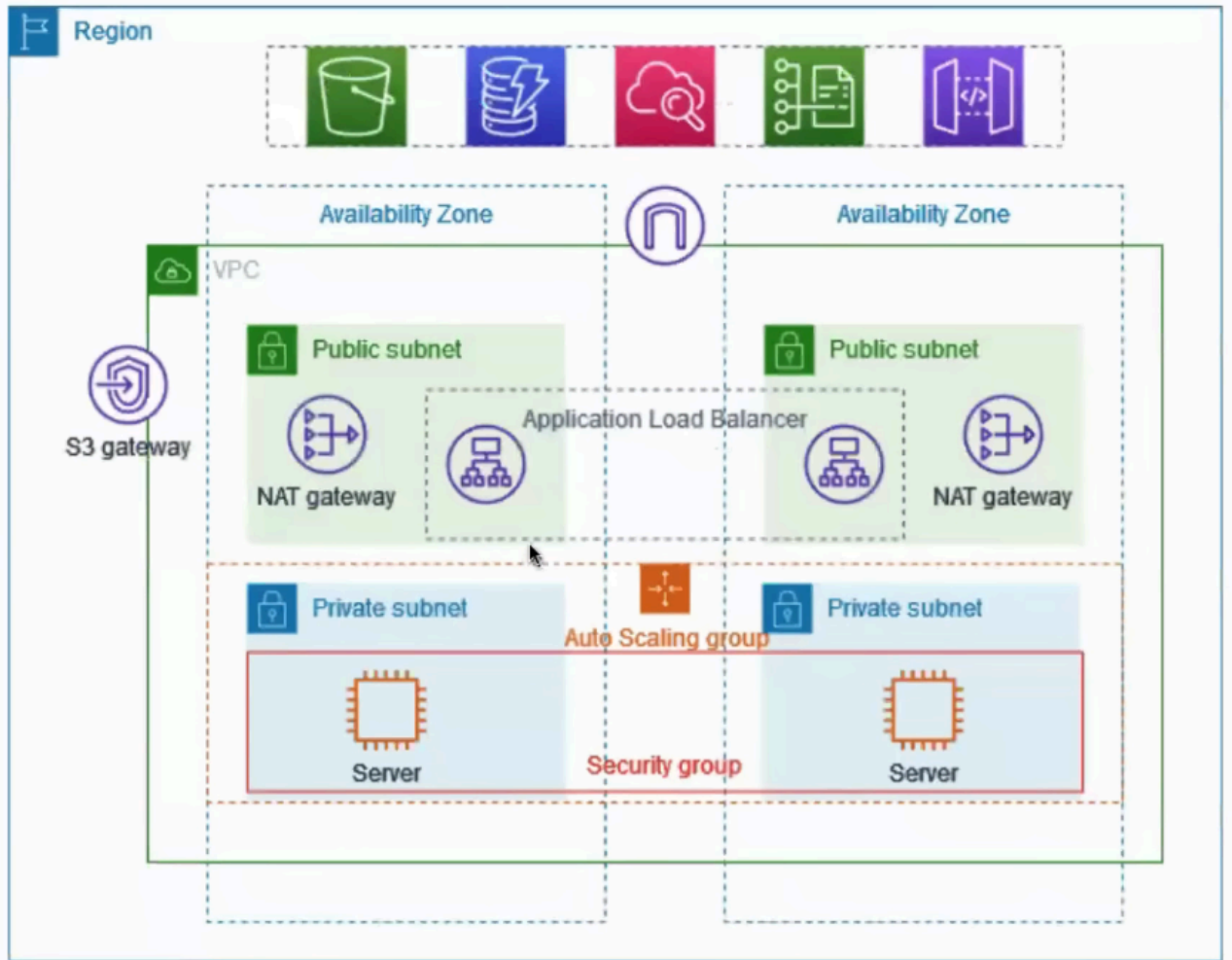
Security Group (SG) in AWS

Feature	Detail
Type	Virtual firewall for EC2
Scope	Instance-level, not subnet-level
Inbound	Deny all by default, allow specific ports
Outbound	Allow all by default
Stateful	Return traffic is automatically allowed
Protocols	Supports TCP, UDP, ICMP, etc.

AWS VPC Resource Breakdown

1. VPC – Virtual Private Cloud

- Isolated private network within AWS
- You define subnets, route tables, gateways



2. Subnets:

A **subnet** (short for sub-network) is a smaller chunk of a network inside your **VPC**. It divides your IP address range into segments, so you can separate resources based on function or security. Think of it as splitting your building into rooms where each room (subnet) has its own purpose and access level.

- Subdivisions within the VPC
- **Public Subnets** (Internet access via IGW)
 - `project-subnet-public1-us-east-1a`
 - `project-subnet-public2-us-east-1b`
- **Private Subnets** (No direct internet)
 - `project-subnet-private1-us-east-1a`
 - `project-subnet-private2-us-east-1b`

Type	Public Subnet	Private Subnet
Internet Access	Has direct access to/from the internet via Internet Gateway	No direct access to the internet
Usage	Hosts web servers, bastion hosts, public apps	Hosts databases, internal services , backend apps
Routing	Route table points to an Internet Gateway (IGW)	Route table uses NAT Gateway or no IGW
Example	Frontend website server	MySQL/Postgres database

How to Create a Subnet from a Given IP

Example:

You're given a **VPC CIDR block**: 192.168.0.0/24

This gives 256 IP addresses (192.168.0.0 to 192.168.0.255)

Split the /24 Block into Smaller Subnets

Subnet CIDR	IP Range	Typical Use
192.168.0.0/25	192.168.0.0 – 192.168.0.127	Public Subnet
192.168.0.128/25	192.168.0.128 – 192.168.0.255	Private Subnet

Each /25 gives 128 IPs (AWS reserves 5, so ~123 usable)

How to Make a Subnet Public

1. Attach an **Internet Gateway (IGW)** to your VPC
2. Create a **route table** that sends traffic to the IGW
3. Associate that route table with your **public subnet**
4. Enable **Auto-assign Public IP** when launching instances
5. Set correct **Security Group** and **Network ACL** rules

Example with Real Public IP Block

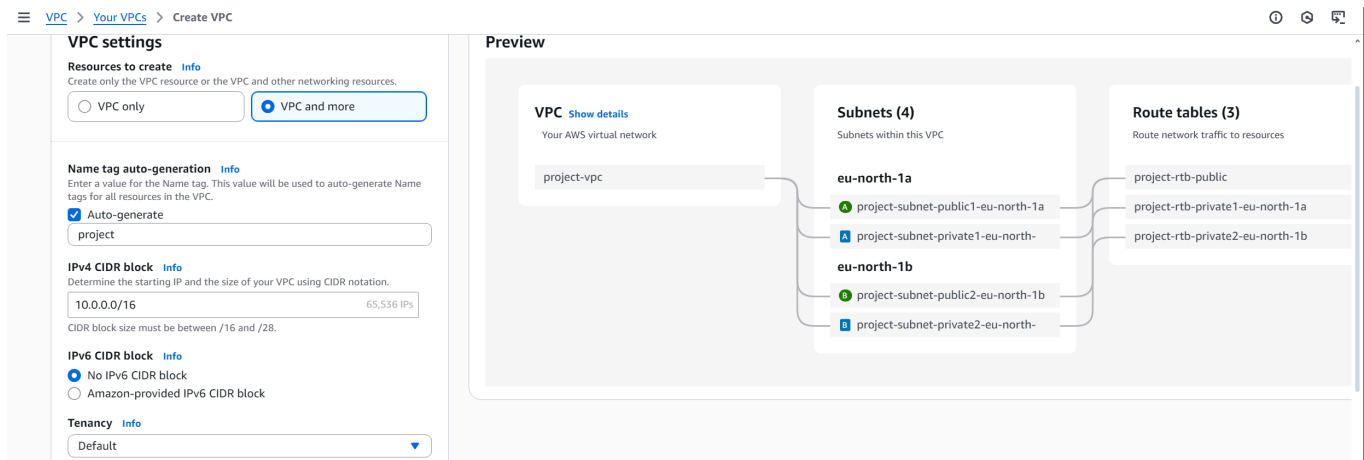
If you're assigned 13.57.23.0/24 (a public IP range), you can:

- Use 13.57.23.0/25 as your **public subnet**
- Use 13.57.23.128/25 as your **private subnet]**

Subnetting concepts get tricky especially when you take host id and net id etc.

<https://www.youtube.com/watch?v=nFYilGQ-p-8>, <https://www.youtube.com/watch?v=B1vqKQIPxr0>

A simple video on this topic



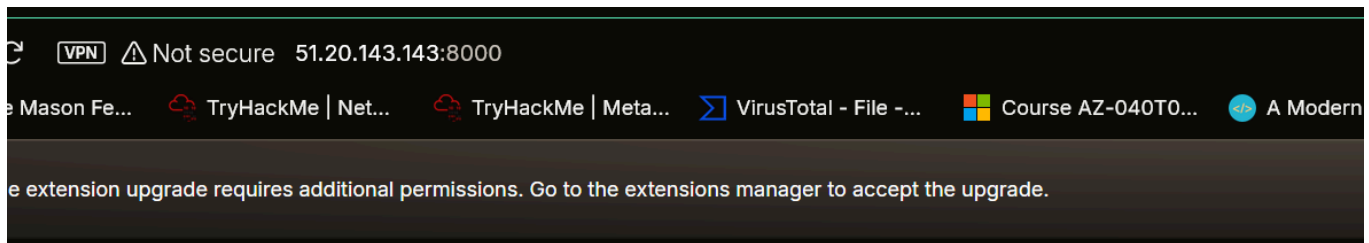
3. Route Tables

AWS route tables are used to control where traffic is directed within a **VPC (Virtual Private Cloud)**. Each subnet in a VPC is associated with a route table that decides how traffic is routed (for example, to the internet, other subnets, or a VPN). Below is the route tables we use

Name	Purpose
project-rtb-public	For public subnets
project-rtb-private1-us-east-1a	Private subnet 1a
project-rtb-private2-us-east-1b	Private subnet 1b
rtb-013c3366b5336e0b5	Possibly default

4. Network Connections

Name	Type	Purpose
project-igw	Internet Gateway	Enables public internet access
project-vpce-s3	VPC Endpoint (S3)	Private access to S3 without internet



ctory listing for /

[shrc](#)
[ofile](#)
[thon_history](#)
[v/](#)
[p/](#)

Application

Active

[vpc-0ee74bd8dae0f9b71](#)

IPv4

Scheme
Internet-facing

Hosted zone
Z23TAZ6LKFMNIO

Availability Zones
[subnet-09dfa35297a2e8a3c](#) eu-north-1a (eun1-az1)
[subnet-0516f4abd41283b44](#) eu-north-1b (eun1-az2)

Date created
May 26, 2025, 17:27 (UTC+03:00)

Load balancer ARN
[arn:aws:elasticloadbalancing:eu-north-1:512455082586:loadbalancer/app/load-balancer-3/de6b5166877fc23f](#)

DNS name
[load-balancer-3-1809626336.eu-north-1.elb.amazonaws.com](#) (A Record)

Listeners and rules

Network mapping

Resource map

Security

Monitoring

Integrations

Attributes

Capacity

Tags

Resource map

How, explore, and troubleshoot your load balancer's architecture.

Overview

Unhealthy target map

Show resource details

ad-balancer-3

Last fetched seconds ago

Export

Listeners (1)
HTTP:8000

Rules (1)
Priority default
Forward to target group

Target groups (1)
Instance
aws-prod-example

Targets (2)
i-054add7905fddfedea Port 8000
i-06cac0acbba1de8e1 Port 8000

Bastion Host

Why use a Bastion Host?

- **Secure access point:** It acts as a controlled entry point into a private network.
- **Reduces attack surface:** Instead of opening SSH/RDP access to many private instances, you only open it on the bastion host.

- **Monitored gateway:** All admin access to internal servers goes through the bastion, making it easier to log, audit, and control.

AWS Load Balancer :

The screenshot displays the AWS Management Console interface for an AWS Load Balancer. The top section, titled 'load-balancer-3', contains a 'Details' tab with the following information:

- Load balancer type:** Application
- Status:** Active (indicated by a green checkmark)
- VPC:** vpc-0ee74bd8dae0f9b71
- Load balancer IP address type:** IPv4
- Scheme:** Internet-facing
- Hosted zone:** Z23TAZ6LKFMNIO
- Availability Zones:**
 - subnet-09dfa35297a2e8a3c (eu-north-1a)
 - subnet-0516f4abd41283b44 (eu-north-1b)
- Date created:** May 26, 2025, 17:27 (UTC+03:00)
- Load balancer ARN:** arn:aws:elasticloadbalancing:eu-north-1:512455082586:loadbalancer/app/load-balancer-3/de6b5166877fc23f
- DNS name:** load-balancer-3-1809626336.eu-north-1.elb.amazonaws.com (A Record)

Below the details, a navigation bar includes tabs for 'Listeners and rules', 'Network mapping', 'Resource map', 'Security', 'Monitoring', 'Integrations', 'Attributes', 'Capacity', and 'Tags'. The 'Listeners and rules' tab is active, showing a table with one listener:

Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate
HTTP:80	Forward to target group <ul style="list-style-type: none"> aws-prod-example (1 (100%)) Target group stickiness: Off 	1 rule	ARN	Not applicable	Not applicable

- Use Cases

A **Load Balancer** in AWS is used to **distribute incoming traffic** across multiple targets (EC2 instances, containers, etc.) You can also add an Auto Scaling Group to your Load Balancer, which helps with **automatically adjusting the number of EC2 instances based on traffic demand**, ensuring high availability, cost-efficiency, and performance during varying load conditions.

Key Benefits

- Distributes traffic evenly
- Increases availability
- Adds fault tolerance
- Supports auto-scaling
- Can terminate SSL/HTTPS
- Acts as a central entry point (DNS)

CloudFormation vs Terraform

Feature	CloudFormation (CFT)	Terraform
Tool Type	AWS-native IaC	Open-source, multi-cloud IaC
Cloud Support	Only AWS	AWS, Azure, GCP, Oracle, etc.
Language	YAML / JSON	HCL (HashiCorp Configuration Language)
State Management	Managed by AWS	Manages own state file
Flexibility	Less flexible	Highly customizable
Rollback Support	Built-in	Manual rollback
Modules	Limited nested stacks	Powerful reusable modules
Permissions	IAM Roles	Provider blocks with credentials
Learning Curve	Easier for AWS-only users	More powerful but steeper learning curve
Ecosystem	Integrated in AWS Console	CLI-based + rich plugin/tool ecosystem

We have used aws command line below we will look at Terraform later on:

To configure the AWS CLI from VS Code, first open your terminal inside VS Code. Make sure you have the AWS CLI installed. Then type `aws configure` and press Enter. It will prompt you to enter your AWS Access Key ID, Secret Access Key, default region name, and output format. These credentials can be generated from your AWS Management Console under IAM by creating a user with programmatic access. Once you input the details, your credentials will be saved in a hidden folder called `.aws` inside your home directory. You can now use AWS CLI commands directly from your VS Code terminal to interact with AWS services like S3, EC2, or Lambda.


```
$ % aws ->
$ %
$ %
$ % aws configure

[Kv]:

$ % API -> Access keys, API tokens

AWS Secret Access Key [None]: NYEDJOWNPGZZR01CXUVOZB
Default region name [None]: eu-north-1
Default output format [None]: json

C:\Program Files>aws s3 ls
2025-05-26 23:07:48 itzmeamaan-s3-prod-example.com
2025-05-25 16:09:37 itzmeamaan-testbucket1
```

Terraform:

<https://developer.hashicorp.com/terraform/docs>

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

Terraform, the **state file** (`terraform.tfstate`) is a critical file that **tracks the infrastructure** Terraform manages.

To connect Terraform to AWS, you first install Terraform and set up your AWS credentials using the AWS CLI (`aws configure`) or by exporting them as environment variables. Then, in your Terraform project folder, you create a `provider` block in your `.tf` file specifying `provider "aws"` along with the region. Once your config is ready, run `terraform init` to initialize the project and download the AWS provider plugin. Then use `terraform plan` to preview changes and `terraform apply` to deploy the resources to AWS.

```
Plan: 7 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_vpc.my_aws_vpc: Creating...
aws_vpc.my_aws_vpc: Creation complete after 3s [id=vpc-068a5210aa2e29304]
aws_internet_gateway.igw: Creating...
aws_subnet.sub1: Creating...
aws_subnet.sub2: Creating...
aws_internet_gateway.igw: Creation complete after 1s [id=igw-0538272b2e12237a2]
aws_route_table.RT: Creating...
aws_route_table.RT: Creation complete after 1s [id=rtb-0d8fd63bf3805c7d3]
aws_subnet.sub2: Still creating... [00m10s elapsed]
aws_subnet.sub1: Still creating... [00m10s elapsed]
aws_subnet.sub2: Creation complete after 11s [id=subnet-0c8aff60dae5c0bff]
aws_route_table_association.rta2: Creating...
aws_subnet.sub1: Creation complete after 11s [id=subnet-01acfc11f547cf533]
aws_route_table_association.rta1: Creating...
aws_route_table_association.rta2: Creation complete after 1s [id=rtbassoc-06b810bf362a16e45]
aws_route_table_association.rta1: Creation complete after 1s [id=rtbassoc-0e27b4ae0f9406c]

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
```



Terraform State File: Overview

Feature	Description
Purpose	Stores metadata and the current state of your infrastructure (like what resources exist, their values, dependencies, etc.).
Default Location	terraform.tfstate in your working directory.
Usage	Used during plan , apply , and destroy to compare actual vs. desired infrastructure.



Why It Matters

- Terraform doesn't query cloud resources every time; it uses this state file as the source of truth.
- It's used to track resource IDs, dependency graphs, and outputs.



Example Content (Simplified)

json

```
{ "version": 4, "resources": [ { "type": "aws_instance", "name": "web",  
"instances": [ { "attributes": { "id": "i-1234567890abcdef0", "instance_type":  
"t2.micro", ... } } ] } ] }
```

Best Practices

Tip	Reason
Use Remote State (e.g., S3)	For team access, safety, and backups.
Lock State	Prevent concurrent modifications (use DynamoDB with S3).
Never edit manually	Can break your setup. Use <code>terraform state</code> commands instead.
Backup regularly	It's your infrastructure truth.

What's a Remote Backend?

A **remote backend** in Terraform **stores the** `terraform.tfstate` **file remotely** (instead of locally) and optionally **locks it** to avoid concurrent changes.

Why Use It?

Benefit	Why it matters
✅ Team Collaboration	Everyone uses the same source of truth.
🔒 State Locking	Prevents multiple users from corrupting state.
💾 Durability	Stored safely in S3, Azure, GCS, etc.
🏠 Separation of Concerns	Keeps state off local machines.

in a security pov state file access should be limited

aws_instance.webServer (remote-exec): error: externally-managed-environment

aws_instance.webServer (remote-exec): × This environment is externally managed

aws_instance.webServer (remote-exec): ↪ To install Python packages system-wide, try apt

install

aws_instance.webServer (remote-exec): python3-xyz, where xyz is the package you are trying to

aws_instance.webServer (remote-exec): install.

aws_instance.webServer (remote-exec):

aws_instance.webServer (remote-exec): If you wish to install a non-Debian-packaged Python package,

aws_instance.webServer (remote-exec): create a virtual environment using python3 -m venv path/to/venv.

aws_instance.webServer (remote-exec): Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make

aws_instance.webServer (remote-exec): sure you have python3-full installed.

aws_instance.webServer (remote-exec):

aws_instance.webServer (remote-exec): If you wish to install a non-Debian packaged Python application,

aws_instance.webServer (remote-exec): it may be easiest to use pipx install xyz, which will manage a

aws_instance.webServer (remote-exec): virtual environment for you. Make sure you have pipx installed.

aws_instance.webServer (remote-exec):

aws_instance.webServer (remote-exec): See /usr/share/doc/python3.12/README.venv for more information.

aws_instance.webServer (remote-exec): note: If you believe this is a mistake, please contact your Python installation or OS distribution provider. You can override this, at the risk of breaking your Python installation or OS, by passing --break-system-packages.

aws_instance.webServer (remote-exec): hint: See PEP 668 for the detailed specification.

aws_instance.webServer: Creation complete after 57s [id=i-0d2eedcde51613163]

dynamodnb

```
resource "aws_dynamodb_table" "dynamodb_table" {
  name           = "amaan-dynamodb-table"
  billing_mode    = "PAY_PER_REQUEST"
  hash_key       = "LockId" # Example hash key, replace with your desired key

  attribute {
    name = "id"
    type = "S"
  }

  tags = {
    Name           = "Amaan-DynamoDB-Table"
    Environment    = "Dev"
  }
}
```

Below is an example terraform code where i breakdown everything:

```
provider "aws" {

    region = "eu-north-1"

}          # we do this choose a provider and also which region our aws is
based on

variable "cidr" {

    description = "The CIDR block for the VPC"

    default     = "10.0.0.0/16"

    type        = string

}

# this is used to make a priv subnet for the VPC we make as a variable so we
can call it later on...we can save this in a variable.tf if we want also

resource "aws_vpc" "myvpc" {

    cidr_block = var.cidr      # here we call a the resorce aws_vpc with the cidr
block defined with the variable

}

resource "aws_key_pair" "name" {

    key_name     = "amaan-key"

    public_key   = file("C:\\Users\\amaan\\xxxxxxx\\yes.pub")

} # we define an aws_key_pair for authentication
```

```

resource "aws_subnet" "sub1" {

    vpc_id          = aws_vpc.myvpc.id

    cidr_block      = "10.0.0.0/24"

    availability_zone = "eu-north-1a"

    map_public_ip_on_launch = true

}

# we create an aws subnet at eu_north_1a...this is done if we need to have
multiple of them..useful incase one goes down


resource "aws_internet_gateway" "igw" {

    vpc_id = aws_vpc.myvpc.id

} # we make a way for the vpc to connect to the internent


resource "aws_route_table" "RT" {

    vpc_id = aws_vpc.myvpc.id # these are route tables for the vpc to knbow what
to do.


    route {

        cidr_block = "0.0.0.0/0"

        gateway_id = aws_internet_gateway.igw.id

        # `gateway_id = aws_internet_gateway.igw.id` tells AWS: **send that
traffic through the internet gateway** (i.e., give the instance internet
access).

    }

}

```

```
resource "aws_route_table_association" "rta1" {  
  
    subnet_id      = aws_subnet.sub1.id  
  
    route_table_id = aws_route_table.RT.id  
  
}
```

resource "aws_security_group" "webSg" { # these the security groups are used to define access inside our vpc and outside..one of the main use cases is like if we make an ec2 instance with a website we can shoot this rules on it

```
    name      = "webSg"  
  
    description = "Security group for web server"  
  
    vpc_id      = aws_vpc.myvpc.id
```

```
    ingress {  
  
        from_port    = 80  
  
        to_port      = 80  
  
        protocol      = "tcp"  
  
        cidr_blocks = ["0.0.0.0/0"]  
  
    }
```

```
    ingress {  
  
        from_port    = 22  
  
        to_port      = 22  
  
        protocol      = "tcp"
```

```

        cidr_blocks = ["0.0.0.0/0"]
    }

    egress {

        from_port    = 0

        to_port      = 0

        protocol     = "-1"

        cidr_blocks = ["0.0.0.0/0"]
    }
}

```

resource "aws_instance" "webServer" { # here we launce an ec2 instance with our subnet detailes

```

    ami                = "ami-0c1ac8a41498c1a9c"

    instance_type      = "t3.micro"

    subnet_id          = aws_subnet.sub1.id

    key_name            = aws_key_pair.name.key_name

    vpc_security_group_ids = [aws_security_group.webSg.id]

    tags = {

        Name = "MyWebServer"
    }
}

```



```

connection { # this is done to conect to the instance to add our app.py

    type      = "ssh"

    user      = "ubuntu"

    private_key = file("C:\\Users\\amaan\\Terraform-practice\\DAY-5\\yes")

    host      = self.public_ip
}

provisioner "file" {

    source      = "C:\\Users\\amaan\\Terraform-practice\\DAY-5\\app.py" #
Replace with the path to your local file

    destination = "app.py" # Replace with the path on the remote instance
}

provisioner "remote-exec" {

    inline = [

        "echo 'Hello, World!'",

        "sudo apt update -y",

        "sudo apt-get install -y python3-pip",

        "cd /home/ubuntu",

        "sudo pip install flask",

        "sudo python3 app.py &"

    ]

}

}

```

also for userdata.sh it should be something like:

```
#!/bin/bash
```

```
apt update
```

```
apt install -y apache2
```

```
# Get the instance ID using the instance metadata
```

```
INSTANCE_ID=$(curl -s http://169.254.169.254/latest/meta-data/instance-id)
```

```
# Install the AWS CLI
```

```
apt install -y awscli
```

```
# Download the images from S3 bucket
```

```
#aws s3 cp s3://myterraformprojectbucket2023/project.webp  
/var/www/html/project.png --acl public-read
```

```
# Create a simple HTML file with the portfolio content and display the images
```

```
cat <<EOF > /var/www/html/index.html
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>My Portfolio</title>
```

```
  <style>
```

```
    /* Add animation and styling for the text */
```

```
    @keyframes colorChange {
```

```
      0% { color: red; }
```

```
        50% { color: green; }

        100% { color: blue; }

    }

    h1 {

        animation: colorChange 2s infinite;

    }

</style>

</head>

<body>

    <h1>Terraform Project Server 1</h1>

    <h2>Instance ID: <span style="color:green">${INSTANCE_ID}</span></h2>

    <p>Welcome to server 2</p>

</body>

</html>

EOF

# Start Apache and enable it on boot

systemctl start apache2

systemctl enable apache2
```

`variables.tf` — Defining Input Variables in Terraform

 **Purpose:**

`variables.tf` is where you **declare reusable input variables** that your Terraform config depends on.

✖ Example:

hcl

Copy code

```
variable "region" { description = "AWS region to deploy into" type = string
default = "us-east-1" } variable "instance_type" { description = "EC2 instance
type" type = string default = "t2.micro" }
```

🏠 Common usage:

- `main.tf` uses the values
 - `terraform.tfvars` (or CLI input) **sets** the values
-

🌲 Directory Structure (`tree` command)

Use `tree` to visualize your Terraform project layout:

bash

Copy code

```
tree
```

Example Output:

bash

Copy code

```
. ├── main.tf # Main config (resources, providers) ├── variables.tf # Variable
declarations ├── terraform.tfvars # Variable values (input) ├── outputs.tf #
Outputs from the deployment └── provider.tf # Provider block (e.g., AWS)
```

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS CODEWHISPERER REFERENCE LOG COMMENTS





@iam-veeramalla → /workspaces/terraform-zero-to-hero/Day-6 (main) $ tree
.
├── main.tf
├── modules
│   └── ec2_instance
│       └── main.tf
├── stage.tfvars
├── terraform.tfstate
└── terraform.tfvars
```

Useful Terraform Commands

Command	What it does
<code>terraform init</code>	Initializes the project (downloads provider plugins)
<code>terraform plan</code>	Shows what Terraform will do (dry-run)
<code>terraform apply</code>	Applies the changes (creates resources)
<code>terraform destroy</code>	Deletes all managed infrastructure
<code>terraform fmt</code>	Auto-formats your code
<code>terraform validate</code>	Checks if config is syntactically valid
<code>terraform output</code>	Displays output values defined in <code>outputs.tf</code>
<code>terraform state list</code>	Lists all resources Terraform is managing

What is Vault?

HashiCorp Vault is a tool designed to **securely store, access, and manage secrets**, such as:

-  API keys
-  Credentials (e.g., usernames/passwords)
-  Certificates
-  Tokens

Vault ensures secrets are **centrally managed, auditable, and access-controlled**.

Terraform + Vault Integration: Use Cases

1. Provision Vault Resources with Terraform

Use Terraform to **manage Vault configuration**, such as:

- Enabling secret engines
- Creating policies
- Configuring auth methods (e.g., AppRole, GitHub)

2. Use Vault Secrets in Terraform

Securely **inject secrets into your Terraform deployments**, like:

- AWS access keys
- DB passwords
- TLS certs

Vault becomes a **backend for secure variable injection** during `terraform apply`.

Secure Terraform with AppRole Auth

Configure AppRole for Terraform

```
vault write auth/approle/role/terraform \  
  secret_id_ttl=10m \  
  token_num_uses=10 \  
  token_ttl=20m \  
  token_max_ttl=30m \  
  secret_id_num_uses=40 \  
  token_policies=terraform
```

This creates an AppRole called `terraform` with:

- Limited-use secret ID and token
- A TTL (time-to-live) on tokens
- Bound to a `terraform` policy

Retrieve the `role_id` for Terraform

```
vault read auth/approle/role/terraform/role-id
```

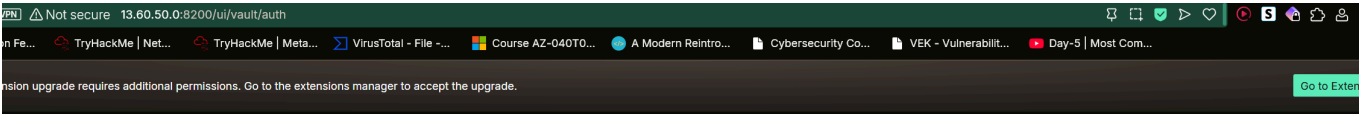
Example Output:

```
role_id = 0dc3d718-ed70-adad-d4a2-24ea7056678b
```

🧠 Conceptual Mapping

AWS IAM	Vault Equivalent
IAM Role attached to EC2/Lambda	AppRole
IAM Policy	Vault Policy
Temporary Security Token	Vault Token (via AppRole)

💡 AppRole is ideal for **non-human machine authentication** like Terraform, CI/CD pipelines, etc.



Sign in to Vault

Method

Token

Token

Sign in

Contact your administrator for login credentials.

1. Terraform dev , staging , prod

These are **environments** representing stages of deployment:

Environment	Purpose	Example Use Case
dev	For developers to test infrastructure	Launch EC2 with dummy app for testing

Environment	Purpose	Example Use Case
staging	Mirror of production for QA/testing	Run full app stack before production push
prod	Live production environment	Real user-facing app, secure and stable

Each environment can use its own:

- `backend` (state storage)
- `variables.tf` (with different values)
- AWS accounts or regions

✅ Benefits of Environment Separation

- Avoid accidental changes to production
- Easier collaboration in teams
- Isolate testing and deployments
- Can apply policies and restrictions per env

And that's wraps for my AWS-Terraform learning...I have also done the Cloud Practitioner which is a free aws beginner course offered by aws. I aim to take the Cloud Architect.