AWS Document

: Nikhil Mahesh Shebannavar

What is cloud computing?

Cloud computing is the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing. Instead of buying, owning, and maintaining physical data centers and servers, you can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud provider like Amazon Web Services (AWS).

What is AWS?

AWS (Amazon Web Services) is a comprehensive, evolving cloud computing platform provided by Amazon. It includes a mixture of infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS) and packaged software-as-a-service (SaaS) offerings. AWS offers tools such as compute power, database storage and content delivery services.

What is Virtualization?

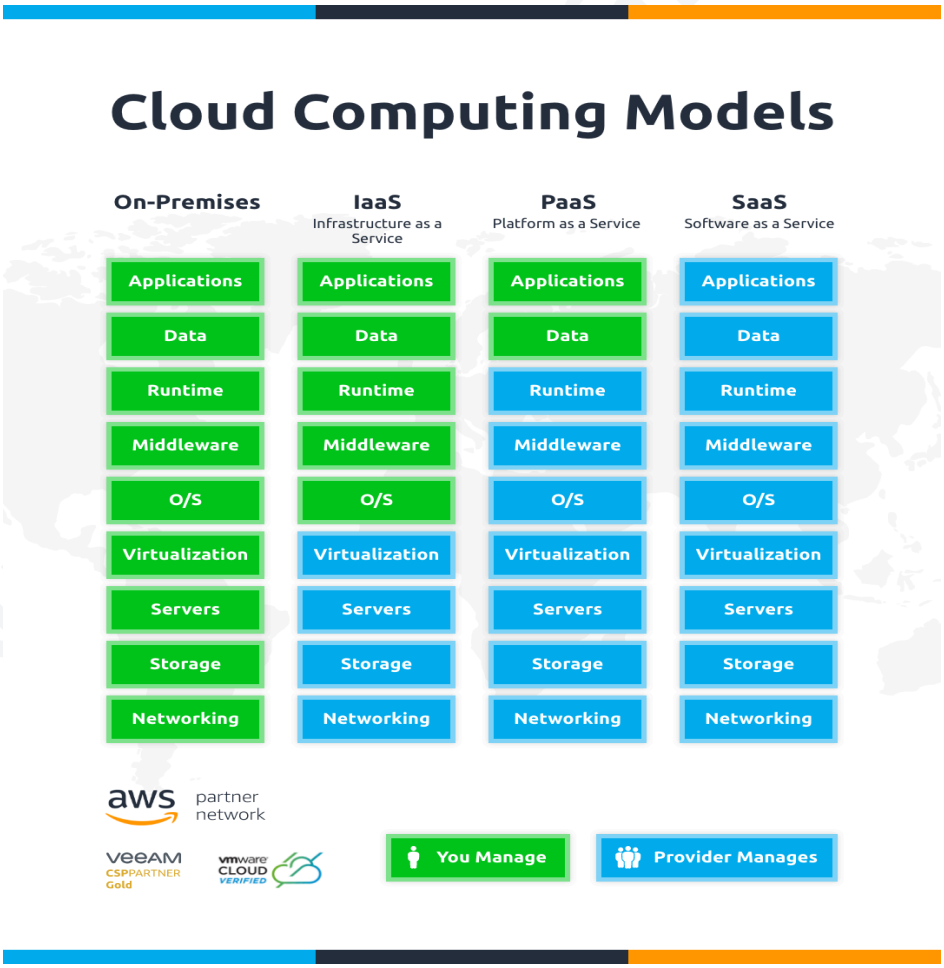
Virtualization is technology that you can use to create virtual representations of servers, storage, networks, and other physical machines. Virtual software mimics the functions of physical hardware to run multiple virtual machines simultaneously on a single physical machine. Businesses use virtualization to use their hardware resources efficiently and get greater returns from their investment. It also powers cloud computing services that help organizations manage infrastructure more efficiently.

Cloud computing models.

IaaS contains the basic building blocks for cloud IT and typically provides access to networking features, computers (virtual or on dedicated hardware), and data storage space. IaaS vendors can help you with the highest level of flexibility and management control over your IT resources and is the type most similar to existing IT resources that many IT departments and developers are familiar with.

PaaS vendors remove the need for organizations to manage the underlying infrastructure (usually hardware and operating systems), and this integration allows you to focus on the deployment and management of your applications. This helps you be more efficient, as you don't need to worry about resource procurement, capacity planning, software maintenance, patching, or any of the other undifferentiated heavy lifting involved in running your application.

SaaS vendors provide you with software applications that are run and managed by the vendor. In most cases, people referring to SaaS are referring to third-party end-user applications. With a SaaS offering you do not have to worry about how the service is maintained or how the underlying infrastructure is managed; you only need to think about how you will use that particular piece of software. A common example of a SaaS application is web-based email where you can send and receive email without having to manage feature additions or maintain the servers and operating systems that the email program is running on.



Region and Availability Zones.

Region is a distinct geographic area, and **Availability Zones (AZs)** are physically isolated data centers within a Region. Regions are independent, while Availability Zones are connected with low-latency links to provide replication and fault tolerance.

EC2 Instance.

What is EC2 Instance?

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud.

It mainly consists in the capability of :

- Renting virtual machines (EC2 - Elastic Compute Cloud)
- Storing data on virtual drives (EBS - Elastic Block Store)
- Distributing load across machines (ELB - Elastic Load Balancing)
- Scaling the services using an auto-scaling group (ASG - Auto Scaling Group)

EC2 Instance Types.

1. General purpose:

General Purpose instances are designed to deliver a balance of compute, memory, and network resources. They are suitable for a wide range of applications, including web servers, small databases, development and test environments, and more.

Series:

- A Series (Medium, Large) : A1
- M Series (Large) : M4, M5, M5a, M5ad, M5d

- T Series : T2, T3, T3a

M6a, M6g, M6gd, M6i, M6id, M6idn, M6in, M7a, M7g, M7gd, M7i, M7i-flex, T4g

2. Compute optimized:

Compute Optimized instances provide a higher ratio of compute power to memory. They excel in workloads that require high-performance processing such as batch processing, scientific modeling, gaming servers, and high-performance web servers.

Series:

- C Series : C4, C5, C5n, C6a, C6g, C6gd, C6gn, C6i, C6id, C6in, C7a, C7g, C7gd, C7gn, C7i

3. Memory optimized:

Memory Optimized instances are designed to handle memory-intensive workloads. They are suitable for applications that require large amounts of memory, such as in-memory databases, real-time big data analytics, and high-performance computing.

Series:

- R Series : R4, R5, R5a, R5d
- X Series : X1, X1e
- Z Series : Z1d

4. Storage optimized:

Storage Optimized instances are optimized for applications that require high, sequential read and write access to large datasets.
They are ideal for tasks like data warehousing, log processing, and distributed file systems.

Series:

- I Series : I3, I3e
- D Series : D2
- H Series : H1

5. Accelerated computing:

Accelerated Computing Instances typically come with one or more types of accelerators, such as Graphics Processing Units (GPUs),
Field Programmable Gate Arrays (FPGAs), or custom Application Specific Integrated Circuits (ASICs). These accelerators offload computationally intensive tasks from the main CPU, enabling faster and more efficient processing for specific workloads.

Series:

- P Series : P2, P3
- G Series : G2, G3
- F Series : F1

6. HPC Optimized (High performance computing) : Hpc6a, Hpc6id, Hpc7a, Hpc7g

- HPC instances on AWS are designed for running high-performance computing workloads efficiently.
- They offer optimized price-performance for scaling HPC tasks.
- Ideal for applications requiring robust processing power, like complex simulations and deep learning tasks.
- Suited for large-scale operations where performance is critical.
- Tailored with high-performance processors to enhance computational capabilities.

Series:

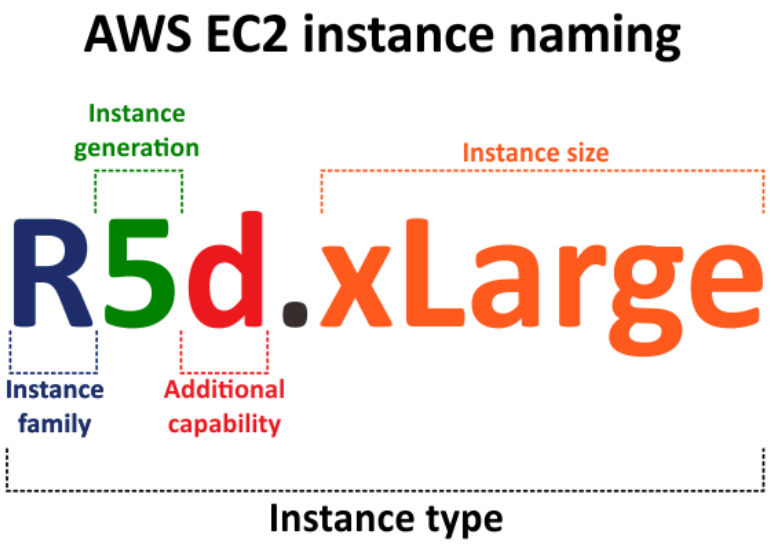
- U Series : U6, U9, U12
- Previous Generation Instance :

T1, M1, C1, CC2, M2, CR1, CG1, i2, HS1, M3,C3, R3

Instance Features

- Scalability: Easily scale your applications up or down based on demand.
- Management Tools: Access a variety of tools to streamline deployment and management processes.
- Monitoring: Monitor the performance of your applications in real-time.
- Flexibility: Choose from a wide range of instance types to suit your specific needs.
- Security: Benefit from built-in security features to protect your applications and data.
- Integration: Seamlessly integrate with other AWS services for enhanced functionality.
- Cost-Effectiveness: Pay only for the resources you use, with cost-effective pricing models.
- Reliability: Rely on Amazon's robust infrastructure for high availability and reliability.
- Burstable Performance Instances
- Multiple Storage Options
- EBS-optimized Instances
- Cluster Networking
- Intel Processor Features

Instance Types



Setup Steps: Launch an EC2 Instance (Linux)

1. Go to EC2 Dashboard
   - Open AWS Console → Search for EC2 → Open Instances section

2. Click “Launch Instance”
   - Click Launch instance → Enter a name for your instance

3. Choose an Amazon Machine Image (AMI)
   - Select an OS (e.g., Amazon Linux 2, Ubuntu, or Windows Server)

4. Choose an Instance Type
   - For free-tier: select t2.micro or t3.micro

5. Create or Select a Key Pair
   - Select an existing key pair or create a new one
   - Download and save the .pem file securely (you’ll need it for SSH)

6. Configure Network Settings
   - Choose a VPC and subnet
   - Enable Auto-assign public IP if you want to access from the internet
   - Create or select a Security Group, and allow:

SSH (port 22) for Linux

HTTP (port 80) for web server (optional)

7. Configure Storage
- Choose the default or add volumes (e.g., 8 GB for root)

8. Launch the Instance
- Click Launch instance
- Wait for the instance state to change to Running

9. Connect to the Instance (Linux)
- Use SSH from your terminal:

chmod 400 your-key.pem
ssh -i your-key.pem ec2-user@your-public-ip

- For Ubuntu, username is ubuntu
- Replace your-public-ip with the instance's IP from the EC2 Console

Advanced Details:

Shutdown behavior
The instance behavior when an OS-level shutdown is performed. Instances can be either terminated or stopped. If no value is specified the value of the source template will still be used. If the template value is not specified then the default API value will be used.

Hibernate
"hibernate" refers to a feature of Amazon EC2 that allows you to pause and resume EC2 instances while preserving their in-memory state. This means you can stop an instance and later resume it from exactly where it left off, preserving data in RAM, leading to faster startup times compared to a regular stop/start cycle.

Termination protection
If enabled, the instance can't be terminated using the console, API, or CLI until termination protection is disabled. If no value is specified the value of the source template will still be used. If the template value is not specified then the default API value will be used.

Stop protection
You can protect instances from being accidentally stopped. Once enabled, you won't be able to stop this instance via the API or the AWS Management Console until stop protection has been disabled.

Tenancy
"Tenancy" refers to how AWS instances (like EC2 instances) are provisioned and how resources are shared between different customers. It essentially dictates who is the "owner" of a resource on the AWS infrastructure. AWS offers two primary tenancy models: Shared Tenancy and Dedicated Tenancy.

User data
Specify user data to provide commands or a command script to run when you launch your instance. Input is base64 encoded when you launch your instance unless you select the User data has already been base64 encoded check box.

Launch template:

A launch template in AWS is a way to define the configuration for launching EC2 instances, including details like AMI, instance type, and network settings. It's essentially a reusable template that captures all the launch parameters within a single resource, simplifying the process of creating new instances.

Setup Steps: Create and Use a Launch Template

1. Go to the EC2 Dashboard
   - Open AWS Console → Navigate to EC2
   - In the left menu, click Launch Templates

2. Click “Create launch template”
   - Enter a name (e.g., web-server-template)
   - Optionally provide a description
   - Leave Auto Scaling guidance enabled if using with ASG

3. Fill in the Launch Template Details

- AMI – Choose an Amazon Machine Image (e.g., Amazon Linux 2, Ubuntu)
- Instance type – e.g., t2.micro
- Key pair – Select a key pair for SSH access
- Network settings – Choose:

VPC and Subnet

Enable or disable public IP

Select a Security Group (allowing SSH, HTTP, etc.)
- IAM instance profile – Choose if EC2 needs access to S3, CloudWatch, etc.
- User data – (Optional) Add a startup script (e.g., install Apache):

```
#!/bin/bash
yum update -y
yum install httpd -y
systemctl start httpd
systemctl enable httpd
```

4. Create the Template
- Review all settings
- Click Create launch template

Amazon EC2 billing and purchasing options

You can use the following options to optimize your costs for Amazon EC2:

- On-Demand Instances – Pay, by the second, for the instances that you launch.

- Savings Plans – Reduce your Amazon EC2 costs by making a commitment to a consistent amount of usage, in USD per hour, for a term of 1 or 3 years.

- Reserved Instances – Reduce your Amazon EC2 costs by making a commitment to a consistent instance configuration, including instance type and Region, for a term of 1 or 3 years.

- Spot Instances – Request unused EC2 instances, which can reduce your Amazon EC2 costs significantly.

- Dedicated Hosts – Pay for a physical host that is fully dedicated to running your instances, and bring your existing per-socket, per-core, or per-VM software licenses to reduce costs.

- Dedicated Instances – Pay, by the hour, for instances that run on single-tenant hardware.

- Capacity Reservations – Reserve capacity for your EC2 instances in a specific Availability Zone.

Elastic IP address:

An Elastic IP address is a static IPv4 address designed for dynamic cloud computing. An Elastic IP address is allocated to your AWS account, and is yours until you release it. By using an Elastic IP address, you can mask the failure of an instance or software by rapidly remapping the address to another instance in your account. Alternatively, you can specify the Elastic IP address in a DNS record for your domain, so that your domain points to your instance. For more information, see the documentation for your domain registrar.

An Elastic IP address is a public IPv4 address, which is reachable from the internet. If you need to connect to an instance that does not have a public IPv4 address, you can associate an Elastic IP address with your instance to enable communication with the internet.

Setup Steps: Allocate and Associate an Elastic IP

1. Go to the EC2 Dashboard
   - Open AWS Console → Search and open EC2

2. Allocate an Elastic IP
   - In the left sidebar, scroll to Network & Security → Click Elastic IPs
   - Click Allocate Elastic IP address
   - Choose:

Scope: Amazon pool

Network: VPC
- Click Allocate

3. Associate Elastic IP with an EC2 Instance
- Select the allocated Elastic IP
- Click Actions → Associate Elastic IP address
- Choose:

Instance: Select your EC2 instance

Private IP: Default or specify
- Click Associate

4. Confirm Static IP Assignment
- Go to EC2 → Instances
- Your instance's Public IPv4 address should now show the Elastic IP

5. (Optional) Use the IP with Route 53
- Go to Route 53 → Hosted Zone for your domain
- Create an A record pointing to the Elastic IP

6. (Optional) Reassign Elastic IP to Another Instance
- Disassociate it from one instance and associate it with another
- Useful during blue-green deployments or instance replacement

Security groups:

Explanation:

- A Security Group in AWS acts like a virtual firewall that controls inbound and outbound traffic to your EC2 instances or other AWS resources.

- You attach a Security Group to EC2, RDS, Load Balancers, and more to define what kind of network traffic is allowed or denied.

- Security Groups are stateful:

If you allow incoming traffic on a port (e.g., SSH on port 22), the response is automatically allowed.

- By default, all inbound traffic is blocked, and all outbound traffic is allowed until you define rules.

Key Features:

- Can be attached to multiple instances

- Easy to modify: changes apply instantly
- Rules are based on protocols, ports, and IP ranges or security groups

- Helps enforce the principle of least privilege

Use Cases:

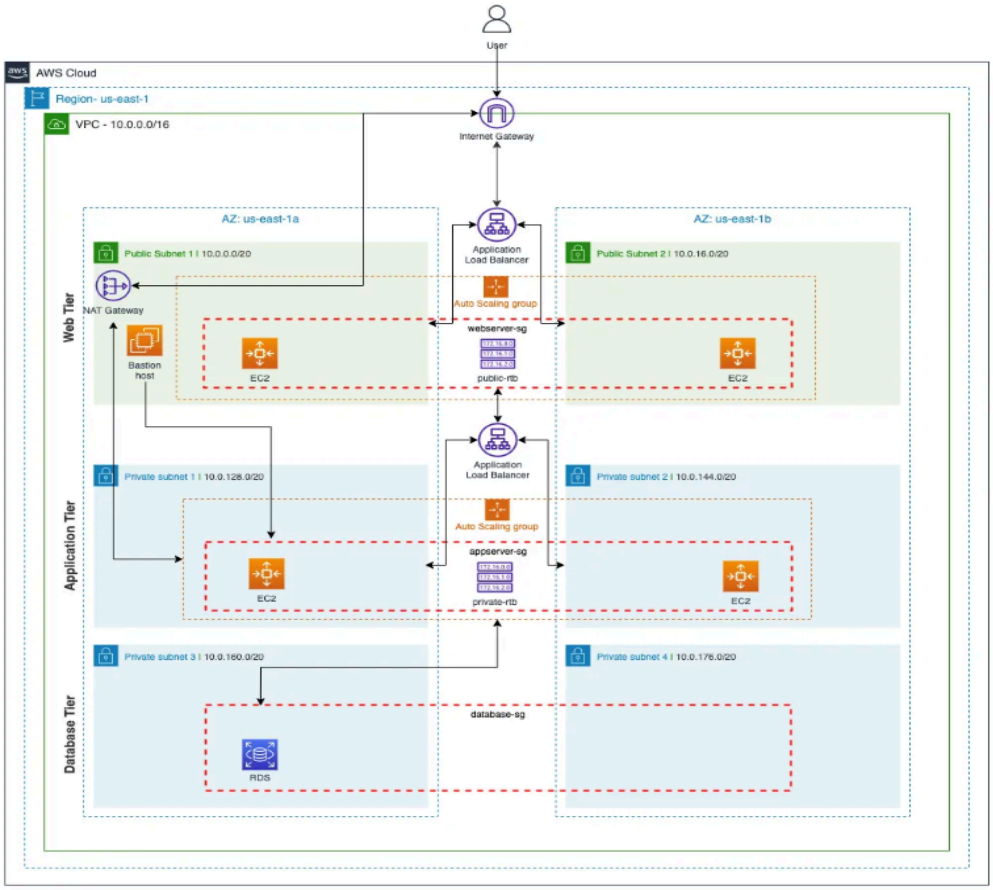
- Allowing SSH access to EC2 only from your IP

- Allowing HTTP/HTTPS for web servers

- Restricting access to a database from app servers only

- Isolating backend services for internal communication only

Cloud Architecture:



Disk Management:

Explanation:

🔹 In AWS, disk storage for EC2 instances is provided by EBS (Elastic Block Store) volumes.

🔹 These volumes act like virtual hard drives that you can attach, format, mount, and expand for your EC2 instances.

🔹 EBS volumes are persistent — they keep data even after the instance stops.

🔹 You can use multiple EBS volumes for separating application files, logs, backups, etc.

Types of EBS Volumes:

- gp3 (General Purpose SSD) – balance of performance and cost (default)
- io2 (Provisioned IOPS SSD) – for I/O-intensive workloads
- st1 (Throughput HDD) – for big data or streaming
- sc1 (Cold HDD) – for infrequent access

Setup Steps: Disk Management with EC2 (Linux)

1. Launch an EC2 Instance (or use existing one)
   - Ensure instance is running in your selected VPC and availability zone

2. Add an EBS Volume
   - Go to EC2 Console → Volumes under Elastic Block Store
   - Click Create Volume
   - Choose:

Volume type (e.g., gp3)

Size (e.g., 8 GiB, 20 GiB, etc.)

Availability Zone (must match the instance)
   - Click Create Volume

3. Attach Volume to an EC2 Instance
   - Select the created volume → Click Actions → Attach Volume
   - Choose the instance
   - Set a device name (e.g., /dev/xvdf)
   - Click Attach

4. Connect to the EC2 Instance
   - SSH into the instance:

```
ssh -i your-key.pem ec2-user@your-instance-ip
```

5. Format the New Volume

🔸 Check available disks:

```
lsblk
```

🔸 Format the volume:

```
sudo mkfs -t xfs /dev/xvdf     # or ext4
```

6. Mount the Volume

🔸 Create a mount point:

```
sudo mkdir /data
```

🔸 Mount the disk:

```
sudo mount /dev/xvdf /data
```

7. Make Mount Persistent (Optional)

🔸 Add to /etc/fstab so it auto-mounts on reboot:

```
echo '/dev/xvdf /data xfs defaults,nofail 0 2' | sudo tee -a /etc/fstab
```

8. Monitor Disk Usage

🔸 Use df -h to check space:

```
df -h
```

🔸 Use lsblk to see all attached volumes

9. (Optional) Resize an EBS Volume

🔸 Go to EC2 → Volumes → Select volume

🔸 Click Modify Volume → Increase size → Save

🔸 On EC2, extend the partition and file system:

```
sudo growpart /dev/xvdf 1
sudo xfs_growfs /data        # For XFS
```

Mount and unmount:

https://www.geeksforgeeks.org/how-to-mount-and-unmount-drives-on-linux/

Formatting disk:

https://phoenixnap.com/kb/linux-format-disk

Snapshot:

Explanation:

🔹 A Snapshot in AWS is a backup of an EBS (Elastic Block Store) volume at a point in time.

🔹 Snapshots are stored in Amazon S3 internally (though not visible in your S3 console).

🔹 They can be used to restore data, create new volumes, or copy to other regions for disaster recovery.

🔹 Snapshots are incremental: only changed blocks since the last snapshot are stored, saving space and cost.

Key Features:

🔹 Point-in-time backup of EBS volumes

🔹 Used for disaster recovery and backup strategies

🔹 Can be copied across AWS regions
🔹 Supports automation via Data Lifecycle Manager

Use Cases:

🔸 Backup critical EC2 data (like app files, databases)

🔸 Create test environments from production data

🔸 Recover from accidental deletions or corruption

🔸 Migrate EC2 volumes across regions or accounts

Setup Steps: Create and Use Snapshots

1. Go to the EC2 Console
🔸 Open AWS Console → Navigate to EC2 → Elastic Block Store → Volumes

2. Create a Snapshot of a Volume

- Select a volume attached to your EC2
- Click Actions → Create snapshot
- Add a description (e.g., Backup before patch)
- Click Create Snapshot

3. View the Snapshot

- Go to EC2 → Snapshots
- Your new snapshot will be listed
- Status will show pending → changes to completed

4. Create a Volume from Snapshot (Restore)

- Go to Snapshots → Select your snapshot
- Click Actions → Create Volume
- Choose:

Availability Zone (must match EC2 to attach)

Volume type and size (can be same or larger)

- Click Create Volume

5. Attach Restored Volume to EC2

- Go to EC2 → Volumes
- Select new volume → Actions → Attach Volume
- Choose the instance and a device name (e.g., /dev/xvdf)

6. (Optional) Copy Snapshot to Another Region

- Select the snapshot → Click Actions → Copy Snapshot
- Choose the target region and description
- Click Copy Snapshot
- Use it later to create volumes in other regions

7. (Optional) Automate Snapshots with Lifecycle Manager

- Go to EC2 → Lifecycle Manager
- Create a policy to automatically take snapshots of tagged volumes
- Set frequency, retention rules, and tags

EBS and EFS:

Explanation:

- EBS is a block storage service designed to be used with EC2 instances.
- It works like a virtual hard disk—you attach it to a single EC2 instance and format it with a file system.
- EBS volumes are persistent, resizable, and support snapshots.

Key Features:

- High performance for transactional workloads (like databases)
- One EC2 instance per volume (unless using special configurations)
- Supports SSD (gp3, io2) and HDD (st1, sc1) types
- Ideal for boot volumes, app data, logs, etc.

Use Cases:

- EC2 instance root or data volume
- Databases (MySQL, PostgreSQL, MongoDB)
- Application or OS-level storage
- Temporary data or caches

EFS (Elastic File System)

Explanation:

🔹 EFS is a managed network file system (NFS) that can be mounted by multiple EC2 instances at once.

🔹 It grows and shrinks automatically as you add/remove files.

🔹 Works best for shared access, scalability, and Linux-based workloads.

Key Features:

🔹 Shared file system accessible from many instances (multi-AZ)

🔹 Scales automatically with usage (no capacity planning)

🔹 Supports POSIX file permissions (Linux-style)

🔹 Works with NFS protocol

Use Cases:

🔸 Shared storage for web apps or microservices

🔸 CMS like WordPress (shared media directory)

🔸 Home directories, dev environments, logs

🔸 Data science & analytics workloads

EBS Setup Steps (Linux EC2)

1. Create Volume → EC2 → Elastic Block Store → Volumes

2. Attach to EC2

3. SSH into EC2 → Format and Mount

sudo mkfs -t xfs /dev/xvdf

sudo mkdir /data

sudo mount /dev/xvdf /data

EFS Setup Steps

1. Go to EFS Console → Click “Create file system”

- Choose VPC and subnets
- Enable automatic backups (optional)

2. Configure Access Points (optional)

- Define user IDs, permissions, and root directories

3. Mount on EC2 Instance (Linux only)

- Install NFS client if not installed:

sudo yum install -y nfs-utils # Amazon Linux

- Mount the EFS:

sudo mkdir /efs

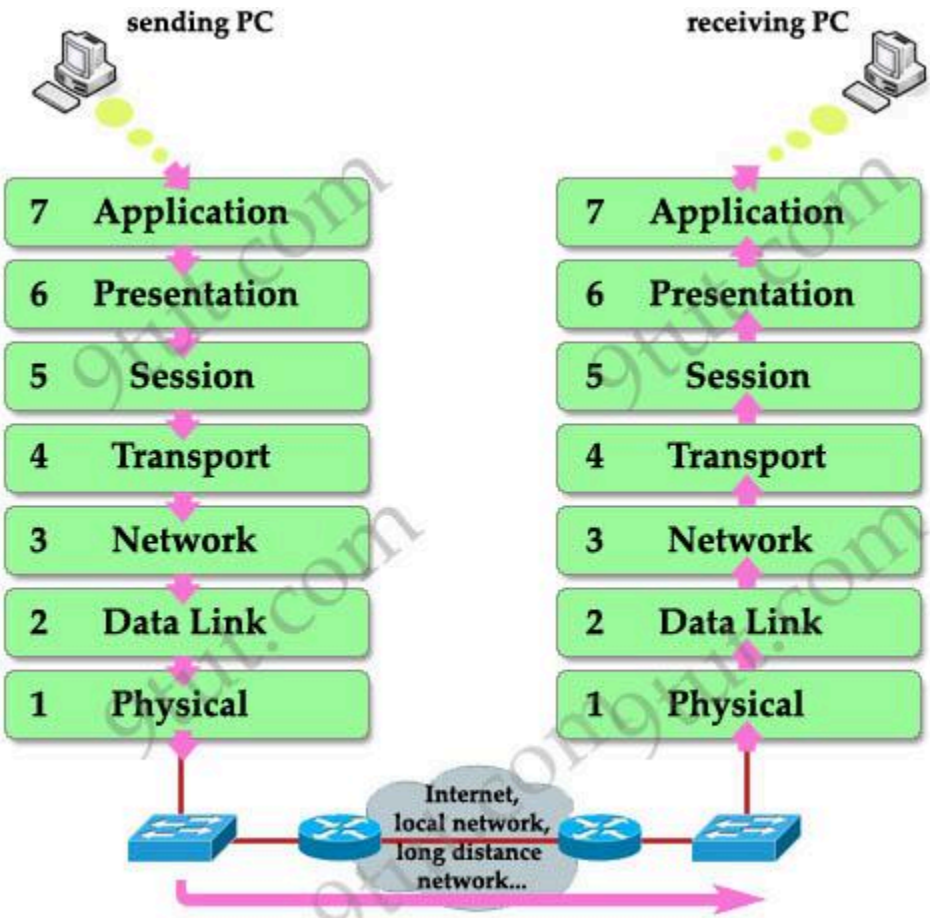
sudo mount -t nfs4 -o nfsvers=4.1 fs-xxxxxxx.efs.region.amazonaws.com:/ /efs

4. (Optional) Add to /etc/fstab for auto-mount on reboot

Networking Basics:

https://shilpathota.medium.com/back-to-basics-networking-dig-deeper-into-ip-addresses-488d4f07e656

OSI Model:



1. Physical Layer: Deals with the physical aspects of the network, like hardware, cables, and transmission of raw bit streams.

2. Data Link Layer: Responsible for error-free transfer of data frames between two directly connected nodes. It also handles physical addressing.

3. Network Layer: Manages logical addressing, routing, and forwarding of data packets between different networks.

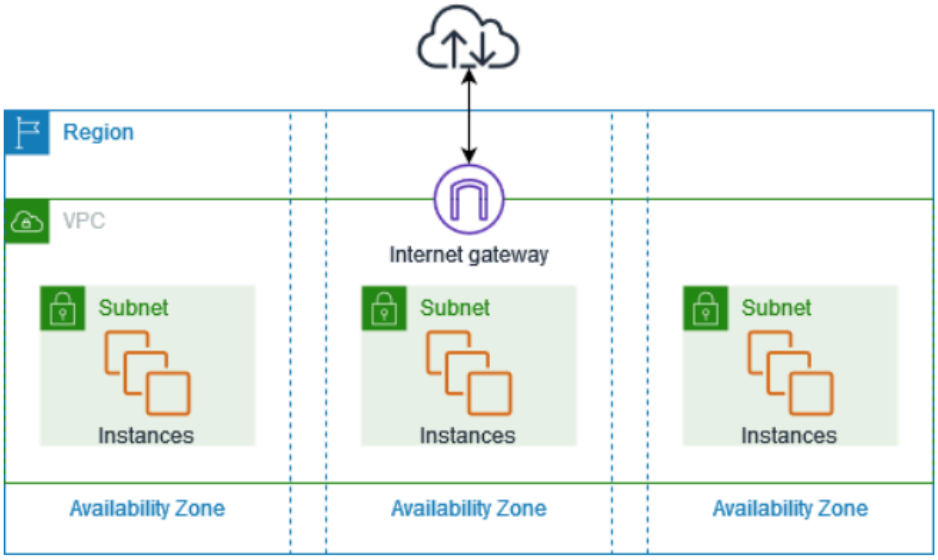
4. Transport Layer: Provides reliable data delivery between end systems, including segmentation, error detection, and flow control.

5. Session Layer: Manages communication sessions between applications, including setting up, maintaining, and terminating connections.

6. Presentation Layer: Handles data formatting, encryption, and compression, ensuring that data is in a usable format for the application layer.

7. Application Layer: Provides the interface for applications to access network services, such as email, file transfer, and web browsing.

VPC:



With Amazon Virtual Private Cloud (Amazon VPC), you can launch AWS resources in a logically isolated virtual network that you've defined. This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS.

The following diagram shows an example VPC. The VPC has one subnet in each of the Availability Zones in the Region, EC2 instances in each subnet, and an internet gateway to allow communication between the resources in your VPC and the internet.

Features:

The following features help you configure a VPC to provide the connectivity that your applications need:

Virtual private clouds (VPC):

A VPC is a virtual network that closely resembles a traditional network that you'd operate in your own data center. After you create a VPC, you can add subnets.

Subnets:

A subnet is a range of IP addresses in your VPC. A subnet must reside in a single Availability Zone. After you add subnets, you can deploy AWS resources in your VPC.

IP addressing:

You can assign IP addresses, both IPv4 and IPv6, to your VPCs and subnets. You can also bring your public IPv4 addresses and IPv6 GUA addresses to AWS and allocate them to resources in your VPC, such as EC2 instances, NAT gateways, and Network Load Balancers.

Routing:

Use route tables to determine where network traffic from your subnet or gateway is directed.

Gateways and endpoints:

A gateway connects your VPC to another network. For example, use an internet gateway to connect your VPC to the internet. Use a VPC endpoint to connect to AWS services privately, without the use of an internet gateway or NAT device.

Peering connections:

Use a VPC peering connection to route traffic between the resources in two VPCs.

Traffic Mirroring:

Copy network traffic from network interfaces and send it to security and monitoring appliances for deep packet inspection.

Transit gateways:

Use a transit gateway, which acts as a central hub, to route traffic between your VPCs, VPN connections, and AWS Direct Connect connections.

VPC Flow Logs:

A flow log captures information about the IP traffic going to and from network interfaces in your VPC.

VPN connections:

Connect your VPCs to your on-premises networks using AWS Virtual Private Network (AWS VPN).

Steps to create:

1. Log in to the AWS Management Console.

2. Navigate to the VPC Dashboard by typing "VPC" in the search bar and selecting VPC under

Services.

3. Click "Create VPC".

4. Choose "VPC and more" to create the VPC along with subnets, route tables, and gateways.

5. In the VPC settings:

- Enter a Name tag (e.g., MyVPC)

- Specify an IPv4 CIDR block (e.g., 10.0.0.0/16)

- Optionally assign an IPv6 CIDR block

- Keep Tenancy as Default

6. In the Subnet settings:

- Add at least one Public Subnet (e.g., 10.0.1.0/24) in an Availability Zone (e.g., us-east-1a)

- Optionally add a Private Subnet (e.g., 10.0.2.0/24) in the same or a different AZ

7. (Optional) Create a NAT Gateway:

- Select a Public Subnet to host the NAT Gateway

- Allocate a new Elastic IP address

8. In the Route Table settings:

- AWS automatically creates and associates route tables

- Public subnets will have routes to the Internet Gateway

- Private subnets (if NAT is used) will route through the NAT Gateway

9. Click "Next", review all settings, and click "Create VPC".

10. AWS will automatically create:

- The VPC

- Public and Private Subnets

- Internet Gateway

- (Optional) NAT Gateway

- Route Tables with correct associations

Subnet:

A subnet is a segment within a VPC where you can place AWS resources like EC2 instances, RDS databases, etc.

There are two types of subnets:

Public Subnet:

Has access to the internet via an Internet Gateway (IGW).

Used for resources that need to be reachable from outside (e.g., web servers).

Private Subnet:

No direct access to the internet.

Used for internal resources (e.g., databases, backend services).

Example:
You create a VPC with CIDR block 10.0.0.0/16.
Then create:

A public subnet 10.0.1.0/24

A private subnet 10.0.2.0/24

Setup Steps: Create Subnets in AWS

1. Go to the VPC Dashboard
   - Open AWS Console → Search and open VPC

2. Click on “Subnets”
   - In the left sidebar under Virtual Private Cloud, click Subnets
   - Click Create subnet

3. Choose VPC and Availability Zones
   - Select an existing VPC
   - Choose one or more Availability Zones (AZs)
   - Name your subnets (e.g., Public-Subnet-A, Private-Subnet-B)

4. Assign CIDR Blocks
   - Example:

VPC CIDR: 10.0.0.0/16

Public Subnet: 10.0.1.0/24

Private Subnet: 10.0.2.0/24

5. Create Internet Gateway (for Public Subnet)
   - Go to Internet Gateways → Create one
   - Attach it to your VPC

6. Modify Route Tables
   - Go to Route Tables
   - Create one for public subnet, and add a route:

Destination: 0.0.0.0/0
Target: Internet Gateway
   - Associate this route table with your public subnet

7. Enable Auto-Assign Public IP (for Public Subnet)
- Go to the Subnet settings → Click Edit subnet settings
- Enable Auto-assign public IPv4 address

8. (Optional) Set up NAT Gateway (for Private Subnet)
- Allows private subnet instances to access the internet for updates
- Create Elastic IP, then NAT Gateway in public subnet
- Update private subnet's route table:

Destination: 0.0.0.0/0
Target: NAT Gateway

Internet Gateways:

An Internet Gateway (IGW) in AWS is a horizontally scaled, redundant, and highly available VPC component that allows communication between instances in your Virtual Private Cloud (VPC) and the internet.

Steps:
1.Go to the VPC Dashboard.

2.Click "Internet Gateways" on the left menu.

3.Click "Create internet gateway" → give it a name → Create.

4.Select the IGW → click "Actions" → Attach to VPC → select your VPC.

5.Update your Route Table (associated with the public subnet):

6.Add route: Destination: 0.0.0.0/0 → Target: your Internet Gateway.

NAT:

A NAT Gateway (Network Address Translation Gateway) in AWS allows instances in a private subnet to access the internet, while preventing the internet from initiating a connection back to those instances.

Create a NAT Gateway:

1.Go to VPC Dashboard > NAT Gateways.

->Click Create NAT Gateway.
->Choose a public subnet (must have a route to an Internet Gateway).

->Assign an Elastic IP (EIP).

2.Update Route Table for Private Subnet:

->Go to Route Tables → select the one for your private subnet.

->Add route:
Destination: 0.0.0.0/0 → Target: NAT Gateway.

3.Verify security group/NACL rules allow outbound traffic.

NACL (Network Access Control List):

A NACL (Network Access Control List) in AWS is a stateless firewall that controls inbound and outbound traffic at the subnet level within a VPC.

Setup Steps: Create and Associate a NACL

1. Go to the VPC Dashboard
   - Open AWS Console → Search and open VPC

2. Click “Network ACLs” in the Sidebar
   - Under Security, click Network ACLs

3. Create a NACL
   - Click Create network ACL
   - Choose the VPC
   - Give it a name tag

- Click Create network ACL

4. Add Inbound and Outbound Rules

- Example Inbound Rules:

Rule #	Type	Protocol	Port Range	Source	Allow/Deny
100	HTTP	TCP	80	0.0.0.0/0	ALLOW
110	HTTPS	TCP	443	0.0.0.0/0	ALLOW
120	SSH	TCP	22	Your IP (e.g., 203.0.113.5/32)	ALLOW
*	All Traffic	All	All	0.0.0.0/0	DENY

- Example Outbound Rules:

Rule #	Type	Protocol	Port Range	Destination	Allow/Deny
100	All Traffic	All	All	0.0.0.0/0	ALLOW

- Click Edit inbound rules and Edit outbound rules to add these.

5. Associate NACL with Subnets
- Click the NACL → Subnet associations → Edit subnet associations
- Select one or more subnets to apply the NACL to
- Save

6. Test the Rules
- Launch an EC2 instance in a subnet associated with the NACL
- Try to access it via allowed and denied ports to validate rules

Difference between Security groups and NACL:

Security Groups are stateful, operate at the instance level, and only allow "allow" rules.
NACLs are stateless, operate at the subnet level, and support both "allow" and "deny" rules.
Security Groups evaluate all rules together, while NACLs evaluate rules in number order.

Security Groups automatically allow return traffic, but NACLs require explicit rules for return traffic.
Use Security Groups for fine-grained control on EC2 access, and NACLs for broad subnet-level filtering.

VPC Peering:

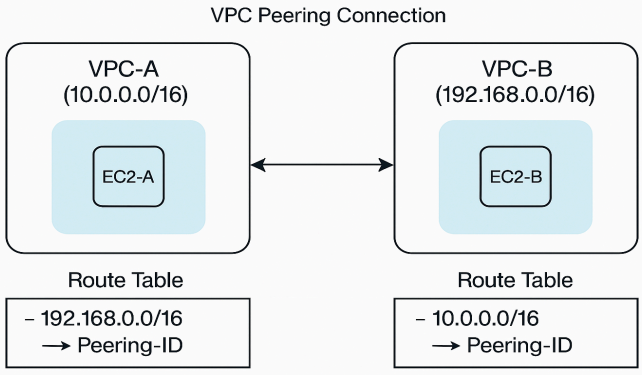
VPC Peering allows you to connect two VPCs privately so they can communicate using private IP addresses, as if they were in the same network.

It works across the same or different AWS regions and accounts, but the VPCs must not have overlapping CIDR blocks.

You must create a peering connection, then accept it, and update route tables in both VPCs to allow traffic.

VPC Peering is non-transitive — if VPC A is peered with B, and B with C, A cannot talk to C unless A–C are also peered directly.

It's commonly used for private communication between microservices, shared services (e.g., logging), or VPCs in different environments (e.g., dev ↔ prod).



Steps:

1. Go to the VPC Dashboard

- Open the AWS Management Console
- Navigate to VPC

2. Create a VPC Peering Connection

- In the left-hand menu, click Peering Connections
- Click Create Peering Connection

✅ Name tag (optional)

✅ VPC Requester: Choose the initiating VPC

✅ VPC Accepter:

- If same account: Select from your list of VPCs
- If different account: Enter the Account ID and VPC ID

✅ Region: Choose appropriate region for accepter (same or cross-region)

- Click Create Peering Connection

3. Accept the Peering Request

- Go to Peering Connections
- Select the connection
- Click Actions → Accept Request

4. Update Route Tables in Both VPCs

- Go to Route Tables
- Select the route table associated with the VPC's subnets

- Click Edit Routes → Add Route
- Destination: CIDR of the other VPC (e.g., 10.1.0.0/16)
- Target: Select the peering connection ID
- Save the changes
- Repeat this process for both VPCs

5. Update Security Groups (Recommended)

- Navigate to Security Groups
- Modify inbound and outbound rules to allow traffic from the other VPC's CIDR block
- Example: Allow TCP/ICMP from 10.1.0.0/16

6. Test the Connection

- Launch EC2 instances in both VPCs
- Connect using private IPs (e.g., SSH or ping)
- Confirm the connectivity works as expected

Note: Both VPC should have different CIDR range.

Load Balancer:

Explanation:

- A Load Balancer in AWS automatically distributes incoming traffic across multiple targets (like EC2 instances, containers, IP addresses) in one or more Availability Zones.

- It improves availability, fault tolerance, and scalability of applications.
- AWS offers three types of Elastic Load Balancers (ELB):
- Application Load Balancer (ALB) – for HTTP/HTTPS (Layer 7)
- Network Load Balancer (NLB) – for TCP/UDP (Layer 4)
- Gateway Load Balancer (GWLB) – for third-party appliances like firewalls

Key Features:

🔹 Distributes incoming traffic evenly

🔹 Automatically performs health checks

🔹 Supports auto scaling and high availability

🔹 Can handle millions of requests per second

🔹 Works across multiple AZs

Use Cases:

🔸 Host a scalable web application

🔸 Route traffic based on URL paths or hostnames (ALB)

🔸 Serve high-performance or low-latency applications (NLB)

🔸 Deploy inspection appliances (GWLB)

Setup Steps: Application Load Balancer (ALB)

1. Go to the EC2 Dashboard

- Open the AWS Management Console
- Navigate to EC2
- In the left-hand menu, click Load Balancers under Load Balancing

2. Create an Application Load Balancer

- Click Create Load Balancer → Choose Application Load Balancer
- Enter name (e.g., my-alb)
- Scheme: Choose Internet-facing or Internal
- IP address type: IPv4 or Dualstack
- Select VPC and at least 2 Availability Zones

3. Configure Listeners and Routing

- Listener: Default is HTTP (port 80); optionally add HTTPS
- Under Default Action, choose:
    - Forward to Target Group
    - Click Create Target Group if not already created

4. Configure Target Group

- Choose target type: Instance, IP, or Lambda
- Protocol and port (e.g., HTTP, port 80)
- Health checks: Leave default or customize
- Register your EC2 instances as targets

- Click Next and finish configuration

5. Add Security Groups

- Select or create a security group that allows HTTP/HTTPS traffic
- Example: allow port 80 and/or 443

6. Review and Create

- Review your configuration
- Click Create Load Balancer

7. Test the Load Balancer

- Copy the DNS name of the load balancer
- Paste it in a browser — it should route traffic to your EC2 instances

Auto Scaling:

Explanation:

- Auto Scaling in AWS automatically adjusts the number of EC2 instances based on demand.
- It ensures high availability and optimizes cost by launching or terminating instances dynamically.
- Auto Scaling works with predefined policies or real-time metrics (like CPU usage) to decide when to scale.
- It integrates with Elastic Load Balancers (ELB) to distribute traffic across healthy instances.

Key Features:

- Automatically replaces unhealthy instances
- Maintains a consistent number of running instances
- Scales based on demand or schedules
- Cost-effective — removes idle resources
- Highly available — supports multi-AZ scaling

Use Cases:

- Web applications with variable traffic
- Disaster recovery or fault-tolerant architectures
- Batch processing jobs
- E-commerce apps during seasonal spikes

Setup Steps: Auto Scaling Group (ASG)

1. Go to the EC2 Dashboard
   - Open the AWS Management Console
   - Navigate to EC2
   - In the left menu, click Auto Scaling Groups

2. Create a Launch Template
   - Click Launch Templates → Create launch template
   - Enter a name and description
   - Choose AMI, instance type, and key pair
   - Configure network, storage, and security group
   - Click Create launch template

3. Create an Auto Scaling Group
   - Go to Auto Scaling Groups → Click Create Auto Scaling group
   - Select the launch template you just created

- Name your Auto Scaling group

4. Configure Network and Load Balancer
- Choose a VPC and Availability Zones
- (Optional) Attach the group to an Application Load Balancer

5. Set Group Size and Scaling Policies
- Define Minimum, Desired, and Maximum number of instances
- Select a scaling policy type:
    - Target tracking (e.g., maintain 50% CPU)
    - Step scaling (based on CloudWatch alarms)
    - Simple scaling (fixed instance count)

6. Configure Notifications (Optional)
- Use SNS to send scale-in or scale-out alerts

7. Review and Create
- Double-check all settings
- Click Create Auto Scaling group

8. Test the Auto Scaling Group
- Simulate high CPU usage or terminate an instance
- Auto Scaling should automatically launch new instances

AWS CLI:

Setup Steps: Install and Configure AWS CLI on Ubuntu

1. Update Your Package Index
- Open the terminal and run:

```
sudo apt update
```

2. Install Required Dependencies
- Install curl and unzip if not already present:

sudo apt install curl unzip -y

3. Download the AWS CLI Installer (v2)
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

4. Unzip the Installer
unzip awscliv2.zip

5. Run the Installer
sudo ./aws/install

6. Verify Installation
- Run this command to verify:

aws --version
- Output should look like: aws-cli/2.x.x Python/3.x.x Linux/x86_64

7. Configure AWS CLI with Your Credentials
- Run the configuration command:

aws configure
- Enter the following when prompted:

AWS Access Key ID

AWS Secret Access Key

Default region name (e.g., us-east-1)

Default output format (json, yaml, or text)

IAM:

Explanation:

- IAM is a security service in AWS that helps you control access to AWS resources.

🔹 It allows you to create users, groups, roles, and policies to manage permissions securely.

🔹 IAM is a global service, meaning it does not require region-specific configuration.

🔹 With IAM, you can follow the principle of least privilege — giving only the permissions needed, and nothing more.

Key Components:

🔸 IAM Users – Represent individual people or applications

🔸 IAM Groups – Collection of users with the same permissions

🔸 IAM Roles – Used for temporary access by AWS services or federated users

🔸 IAM Policies – JSON documents that define what actions are allowed or denied

Use Cases:

🔹 Securely manage employee or system access

🔹 Grant temporary credentials to services like EC2, Lambda, or external apps

🔹 Restrict access to specific AWS services or regions

🔹 Implement multi-factor authentication (MFA) and password policies

Setup Steps: IAM User, Group, Policy, Role, and Permissions

1. Create an IAM Group
🔸 Go to the IAM Dashboard in AWS Console
🔸 In the left menu, click User groups → Create group
🔸 Enter a group name (e.g., Developers)

- Under Attach permissions policies, select one or more policies (e.g., AmazonEC2ReadOnlyAccess)
- Click Create group

2. Create an IAM User
- In the IAM Dashboard, click Users → Add users
- Enter a username (e.g., john.doe)
- Select:

Programmatic access (for CLI/API access)

AWS Management Console access (for web console login)
- For Console access, set a custom password or auto-generate one
- Click Next: Permissions

3. Assign User to Group (for Permission)
- On the Set permissions page, choose:

Add user to group

Select the group created earlier (e.g., Developers)
- Click Next: Tags (optional)
- Click Next: Review
- Click Create user
- Download or copy the access credentials (Access Key ID and Secret Key)

4. Create a Custom IAM Policy (Optional)
- In IAM Dashboard, click Policies → Create policy
- Choose the JSON tab
- Paste your custom policy document, e.g.:

json
Copy
Edit
```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetObject"
      ],
      "Resource": "*"
    }
  ]
}
```

- Click Next: Tags → Next: Review
- Give your policy a name (e.g., S3ReadOnlyCustomPolicy)
- Click Create policy

5. Attach Policy to Group or User
   - Go to Groups or Users, depending on where you want to attach
   - Click the group or user name → Go to the Permissions tab
   - Click Add permissions → Attach policies directly
   - Search and select your custom policy or AWS managed policy
   - Click Next: Review → Add permissions

6. Create an IAM Role
   - Go to IAM → Roles → Create role
   - Select AWS service (e.g., EC2 or Lambda) as the trusted entity
   - Click Next: Permissions
   - Choose a policy to attach (e.g., AmazonS3FullAccess)
   - Click Next: Tags → Next: Review
   - Enter a role name (e.g., EC2-S3AccessRole)
   - Click Create role

7. Assign IAM Role to EC2 Instance (Example)
   - Go to EC2 Dashboard
   - Click Instances → Select an instance
   - Click Actions → Security → Modify IAM Role
   - Choose the IAM role you created and click Update IAM role

S3 Bucket:

Explanation:

🔹 Amazon S3 (Simple Storage Service) is a highly scalable and durable object storage service used to store and retrieve any amount of data.

🔹 An S3 Bucket is a container used to store objects (files, images, backups, etc.).

🔹 Each bucket has a globally unique name, and objects are stored in a flat structure, not folders (though folder-like prefixes can be used).

🔹 Buckets can be public or private, support versioning, encryption, and lifecycle rules.

Key Features:

🔹 Unlimited storage capacity

🔹 99.999999999% (11 9's) durability

🔹 Fine-grained access control using IAM and bucket policies

🔹 Support for static website hosting, logging, and analytics

🔹 Integration with AWS services like Lambda, CloudFront, and Glacier

Use Cases:

🔸 Backup and restore solutions

🔸 Hosting static websites

🔸 Storing logs and event data

🔸 Serving images, videos, or downloadable content

- Big data analytics and archiving

Setup Steps: Create and Access an S3 Bucket

1. Go to the S3 Dashboard
   - Open AWS Management Console
   - Search for and select S3

2. Create a New Bucket
   - Click Create bucket
   - Enter a globally unique bucket name (e.g., my-website-bucket)
   - Choose an AWS Region
   - Leave other settings default (or customize: block public access, enable versioning, encryption, etc.)
   - Click Create bucket

3. Upload Objects (Files)
   - Click on the bucket name
   - Click Upload → Add files or Add folder
   - Select files and click Upload

4. Set Bucket/Object Permissions (Optional)
   - By default, buckets and objects are private
   - Use Bucket Policy or Object Permissions to allow read/write access (e.g., for public hosting or cross-account use)

5. Enable Static Website Hosting (Optional)
   - Go to the Properties tab
   - Enable Static website hosting
   - Set index document (e.g., index.html) and error document (e.g., error.html)
   - Use the provided S3 website endpoint to access the hosted site

6. Attach Access Policy to EC2 via IAM Role (for programmatic S3 access)
   - Go to IAM → Roles → Create Role
   - Choose Trusted Entity: AWS service → EC2
   - Click Next: Permissions

🔸 Attach policy: choose a predefined policy like AmazonS3FullAccess or your custom one
🔸 Click Next → Name the role (e.g., EC2S3AccessRole) → Create role

🔸 Now, attach this role to your EC2 instance:

Go to EC2 Dashboard → Instances
Select the instance → Click Actions → Security → Modify IAM Role
Choose the IAM Role you created and click Update IAM Role

Cloud Watch:

Explanation:

🔹 Amazon CloudWatch is a monitoring and observability service by AWS.

🔹 It collects and tracks metrics, logs, and events from AWS resources and applications.

🔹 You can set alarms, create dashboards, and automate actions (like auto scaling or sending SNS alerts) based on monitored data.

🔹 CloudWatch helps detect problems, optimize performance, and respond to system-wide changes.

Key Features:

🔹 Real-time monitoring of EC2, RDS, Lambda, API Gateway, etc.

🔹 Collect custom metrics and logs from applications

🔹 Create CloudWatch Alarms to trigger alerts or actions

🔹 Build visual dashboards

🔹 Automatically respond with Auto Scaling or Lambda

Use Cases:

- Monitor EC2 CPU usage or memory (custom metric)
- Get alerts when a resource crosses a threshold
- Store, search, and analyze application logs
- Trigger Auto Scaling based on usage
- Visualize health status of services in a dashboard

Setup Steps: Using CloudWatch with EC2

1. Go to CloudWatch Dashboard
   - Open the AWS Management Console
   - Search for and go to CloudWatch

2. View Default Metrics for EC2
   - In the left menu, click Metrics
   - Choose EC2 → Per-Instance Metrics
   - Select your instance to view metrics like:

CPUUtilization

NetworkIn / NetworkOut

DiskReadBytes / DiskWriteBytes

3. Create a CloudWatch Alarm for EC2
   - Click Alarms → Create alarm
   - Click Select metric
   - Choose:

EC2 → Per-Instance Metrics → CPUUtilization
   - Select your instance → Click Select metric
   - Set conditions:

Threshold type: Static

Whenever CPU is greater than 80% for 5 minutes

- Click Next

4. Configure Alarm Actions
- Choose Send notification
- Create or select an SNS topic
- Enter an email to receive alerts
- Click Create topic

5. Name and Create the Alarm
- Give the alarm a name (e.g., HighCPUAlert)
- Review and click Create alarm

6. (Optional) Monitor Logs with CloudWatch Logs
- In EC2, install the CloudWatch Agent to monitor memory, disk, and custom logs
- Example for Ubuntu EC2:

```
sudo apt install amazon-cloudwatch-agent
sudo
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
```

- Configure it to push metrics and logs to CloudWatch

7. (Optional) Create a Dashboard
- In CloudWatch → Dashboards → Create dashboard
- Add widgets for EC2, RDS, S3, Lambda, etc.
- Customize views using graphs, numbers, or text

RDS:

Explanation:

- Amazon RDS is a managed database service that makes it easy to set up, operate, and scale relational databases in the cloud.

🔹 It supports engines like MySQL, PostgreSQL, MariaDB, Oracle, SQL Server, and Amazon Aurora.

🔹 AWS handles tasks like backups, patching, monitoring, and high availability (Multi-AZ).

🔹 You can connect RDS to applications running on EC2, Lambda, or outside AWS.

Key Features:

🔹 Supports automatic backups, snapshots, and replication

🔹 Multi-AZ for high availability and failover

🔹 Read replicas for scaling read-heavy workloads

🔹 Built-in monitoring via CloudWatch

🔹 Supports IAM authentication and VPC security

Use Cases:

🔸 Web applications needing a relational database

🔸 Storing structured business data

🔸 Hosting WordPress or eCommerce backends

🔸 Data warehousing or analytics workloads

Setup Steps: Create and Connect to an RDS Instance

1. Go to the RDS Console
   🔸 Open the AWS Console → Search and select RDS

2. Click “Create Database”
- Choose a database creation method: Standard Create
- Select a database engine (e.g., MySQL, PostgreSQL)

3. Choose Database Version and Templates
- Select the DB version
- Choose a template (e.g., Free Tier, Dev/Test, or Production)

4. Set DB Instance Settings
- DB instance identifier: (e.g., mydbinstance)
- Master username: (e.g., admin)
- Master password: Set and confirm the password

5. Configure Instance Specifications
- DB instance class (e.g., db.t3.micro for Free Tier)
- Storage type: General Purpose (SSD)
- Enable storage autoscaling if needed

6. Configure Connectivity
- Select a VPC (existing or default)
- Set public access: Yes (if connecting from your PC)
- Choose availability zone and subnet group
- Create or select a VPC security group to allow inbound traffic on the database port (default: 3306 for MySQL)

7. Additional Settings
- Optionally configure database name, backup retention, encryption, maintenance settings
- Leave defaults if unsure

8. Click “Create Database”
- Wait a few minutes for the instance to become available

9. Allow Access in Security Group
- Go to EC2 → Security Groups → Edit inbound rules
- Add a rule:

Type: MySQL/Aurora

Protocol: TCP

Port Range: 3306

Source: Your IP or Anywhere (0.0.0.0/0) (for testing only; restrict in production)

10. Connect to the Database
- Get the endpoint from the RDS console
- Use any database client like MySQL Workbench, DBeaver, or terminal:

mysql -h your-endpoint.rds.amazonaws.com -u admin -p

Route 53:

Explanation:

- Amazon Route 53 is a scalable Domain Name System (DNS) service that routes user requests to AWS and external resources based on domain names.

- It converts human-readable domain names (like example.com) into IP addresses (like 192.0.2.1) that computers use to connect.

- Route 53 also supports domain registration, health checks, and routing policies (e.g., failover, geolocation, latency-based).

- It is fully integrated with other AWS services like EC2, S3, CloudFront, and ELB.

Key Features:

- Domain registration and DNS hosting

- Supports different record types (A, CNAME, MX, TXT, etc.)

- Highly available and globally distributed DNS
- Health checks and DNS failover
- Routing policies: Simple, Latency, Failover, Weighted, Geolocation, Geoproximity, Multivalue

Use Cases:

- Point a domain to an EC2 instance, S3 static website, or Load Balancer
- Create subdomains like app.example.com or api.example.com
- Enable failover between two endpoints
- Route traffic based on lowest latency or location

Setup Steps: Route 53 – Hosting a Domain and Pointing to EC2

1. Register a Domain (or Use an Existing One)
   - Go to Route 53 → Domains
   - Click Register Domain to buy one (or skip if using an external domain)
   - If using an external domain, update its name servers to Route 53 ones later

2. Create a Hosted Zone
   - Go to Route 53 → Hosted zones
   - Click Create hosted zone
   - Enter your domain name (e.g., example.com)
   - Choose type: Public Hosted Zone
   - Click Create hosted zone

3. Add DNS Records (e.g., Point to EC2)
   - Click into the hosted zone you just created
   - Click Create Record → Choose Record type: A – IPv4 address
   - For Value, enter the Elastic IP address of your EC2 instance
   - Click Create records

4. (Optional) Add CNAME or Other Records

- Example: To point www.example.com to example.com, add a CNAME record
- For www, enter:

Type: CNAME

Value: example.com.

5. Update Domain Name Servers (if domain is external)

- Go to the Hosted zone → Copy the list of NS (Name Server) records
- In your domain registrar's DNS settings, replace the existing NS records with Route 53's

6. Wait for DNS Propagation

- DNS changes may take a few minutes to a few hours to fully propagate
- Use tools like nslookup or dig to verify the domain is resolving correctly

7. (Optional) Set Up Health Checks and Failover

- Go to Health Checks → Create health check
- Monitor an endpoint (like an EC2 IP or URL)
- Use it in a failover routing policy to redirect traffic if the primary endpoint is down

CloudFront:

Explanation:

- Amazon CloudFront is a Content Delivery Network (CDN) service that securely delivers your content (web pages, images, videos, APIs) with low latency and high speed.

- It uses a global network of edge locations to cache and serve content closer to the users.

- CloudFront integrates with S3, EC2, Load Balancers, and even non-AWS origins (like external websites).

🔹 It improves performance, reduces load on your servers, and adds security features like DDoS protection and HTTPS support.

Key Features:

- 🔹 Global distribution using AWS edge locations
- 🔹 Caching for faster access and reduced origin load
- 🔹 HTTPS support with custom or default SSL certificates
- 🔹 Integration with WAF and Shield for security
- 🔹 Supports static and dynamic content delivery

Use Cases:

- 🔸 Accelerating websites and APIs
- 🔸 Delivering static websites (S3 + CloudFront)
- 🔸 Streaming videos or media globally
- 🔸 Securing content with signed URLs or geo-restrictions

Setup Steps: CloudFront with S3 (Static Website Hosting Example)

1. Upload Your Content to S3

- 🔸 Go to the **S3 Console**
- 🔸 Create a bucket (e.g., my-website-bucket)
- 🔸 Upload your static files (HTML, CSS, JS, images)

2. Enable Static Website Hosting on the S3 Bucket

- 🔸 In the bucket, go to **Properties**
- 🔸 Scroll to **Static website hosting** → Click **Edit**
- 🔸 Enable it, and specify:

- Index document: index.html

- Error document: error.html
  - Save changes

3. Make Content Public (Optional for Testing)

- Go to **Permissions** → **Bucket policy**
- Add a policy to allow public read access (only for testing):

json
CopyEdit

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "PublicRead",
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::my-website-bucket/*"
  }]
}
```

4. Go to the CloudFront Console

- Open the AWS Console → Search and open **CloudFront**
- Click **Create Distribution**

5. Choose Web as the Delivery Method (Default)

- Under **Origin Settings**:

  - Origin domain: Choose your S3 bucket

  - Origin path: Leave blank

  - Viewer protocol policy: Redirect HTTP to HTTPS

  - Cache policy: Use the default unless customizing

6. Configure Default Behavior Settings

- Leave most settings as default

- Enable **Compress objects automatically**
- You can add restrictions like geo-blocking or signed URLs (optional)

7. Add an Alternate Domain Name (Optional)

- If using your own domain (e.g., www.example.com), add it here
- Attach an **SSL certificate** via AWS Certificate Manager (ACM)

8. Create the Distribution

- Click **Create distribution**
- Wait for deployment (can take a few minutes)

9. Access Your Site

- Copy the **CloudFront domain name** (e.g., d123abcd.cloudfront.net)
- Visit the URL in your browser — it should display your site via CloudFront

Lambda;

Explanation:

- AWS Lambda is a serverless compute service that runs your code in response to events—automatically provisioning and managing infrastructure.

- You don't need to manage servers; just upload your code and define a trigger (like S3 upload, API Gateway request, CloudWatch event, etc.).

- Lambda supports multiple languages: Python, Node.js, Java, Go, Ruby, .NET, etc.

- It automatically scales, and you only pay for the compute time your code actually uses.

Key Features:

- Serverless — no server provisioning or management

- Event-driven execution (S3, DynamoDB, API Gateway, etc.)

- Automatic scaling and high availability

- Granular billing (per millisecond)
- Supports environment variables, VPC access, and concurrency controls

Use Cases:

- Image or file processing when uploaded to S3
- Real-time data transformation (e.g., stream processing)
- Backend logic for web/mobile apps via API Gateway
- Automation and scheduled tasks (e.g., nightly backups)
- Chatbot or Alexa skills

Setup Steps: Create and Test a Lambda Function

1. Go to the Lambda Console
- Open AWS Console → Search and select **Lambda**

2. Click "Create Function"
- Choose **Author from scratch**
- Enter:
  - Function name (e.g., MyTestFunction)
  - Runtime (e.g., **Python 3.10**, **Node.js 18.x**, etc.)

3. Set Execution Role
- Choose:
  - **Create a new role with basic Lambda permissions**
    - This allows CloudWatch logging
    - (Optional) Create a custom role if accessing S3, DynamoDB, etc.

4. Write or Upload Code

- In the **Function code** section, you can:
  - Write inline using the code editor
  - Or upload a .zip file or use **S3** or **container image**

- Example code (Python):

python
CopyEdit

```
def lambda_handler(event, context):
    return {
        'statusCode': 200,
        'body': 'Hello from Lambda!'
    }
```

5. Deploy the Function

- Click **Deploy** to save the code

6. Test the Function

- Click **Test** → Create a new test event (use default JSON)
- Click **Test** again
- The result will appear in the **Execution results** section

7. Set Up a Trigger (Optional)

- Go to the **Configuration** tab → **Triggers** → **Add trigger**
- Choose source like:
  - **S3** → Trigger on object upload
  - **API Gateway** → Expose as REST API
  - **EventBridge (CloudWatch Events)** → Trigger on a schedule

8. Monitor with CloudWatch Logs

- Logs are auto-sent to **CloudWatch**
- Go to **Monitor** tab or open CloudWatch to view logs and metrics

Three Tier architecture:

Explanation:

🔹 A Three-Tier Architecture is a well-organized and scalable structure that separates your application into three layers:

Presentation Layer (Web Tier) – UI that users interact with (e.g., web browsers)

Application Layer (Logic Tier) – Business logic, APIs, processing

Data Layer (Database Tier) – Database that stores persistent data

🔹 This separation allows for better scalability, security, and maintainability.

🔹 On AWS, it is typically implemented using services like Elastic Load Balancer, EC2, Auto Scaling, and RDS.