AWS Fundamentals

AWS Regions – The Global Layer

What is a Region?

An **AWS Region** is a **geographically isolated area** that contains multiple **Availability Zones (AZs)**.
Each region is designed to be **independent**, providing **data sovereignty, fault tolerance, and low latency**.

Example Regions:

Region Name	Code	Location
US East (N. Virginia)	`us-east-1`	North America

Each region has its **own set of resources**, like EC2 instances, RDS databases, and S3 buckets

Availability Zones (AZs)

Each **Region** has multiple **Availability Zones** (usually 2–6).
An **AZ** is a **physically separate datacenter (or group of datacenters)** connected via **low-latency links**.

Example:

```
Region: ap-south-1
└── ap-south-1a
└── ap-south-1b
└── ap-south-1c
```

VPC (Virtual Private Cloud) Overview

Inside each Region, you create a **VPC** — an **isolated virtual network** for your AWS resources. You can create multiple VPC in a region.

A VPC lets you define:

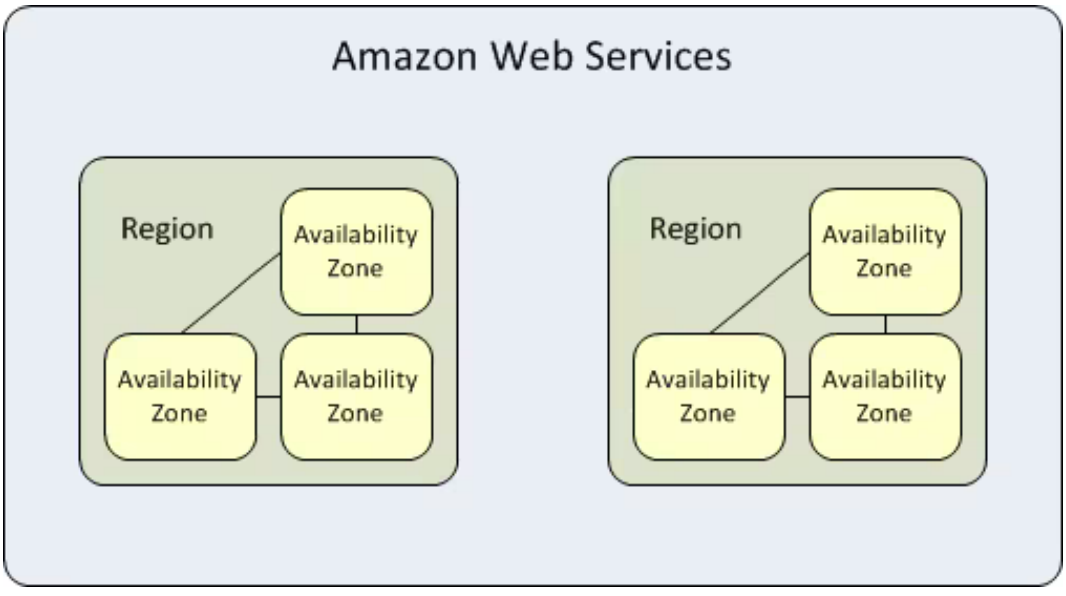
- IP address ranges (CIDR blocks)
- **Subnets**
- **Route tables**
- **Internet Gateways**
- **Security groups / NACLs**

Subnets in AWS

What is a Subnet?

A **Subnet** is a **smaller network segment** inside your VPC.

- Each Subnet **belongs to exactly one Availability Zone**.
- Subnets **divide the VPC** into logical groups (e.g., public vs private).
- You can attach **routing rules** to control internet access.



Subnet Best Practices

Follow these guidelines for real projects:

1. Always use **multiple AZs** for redundancy.
2. Divide subnets into **tiers** (public, private, isolated).
3. Use **NAT Gateways** for private subnet internet access.
4. Reserve **enough CIDR space** for scaling.
5. Tag subnets clearly (`Env=Prod`, `Tier=Public`).
6. Use **Route Tables** and **Security Groups** to isolate traffic.

AWS Fields that appear while creating ec2 instances

1. Name & Tags

- **Name**: A tag where key = `Name` and value = whatever you choose. It helps you identify the instance later. Optional but strongly recommended. AWS Documentation+1
- **Additional Tags**: You can add other key/value tags (for example, `Environment=Production`, `Role=WebServer`, etc.). These help with organization, cost-allocation, automation. AWS Documentation

---

2. Application & OS Images (AMI)

- **AMI**: The Amazon Machine Image you choose. This contains the OS (e.g., Amazon Linux, Ubuntu, Windows) + possibly applications or configurations. AWS Documentation+1
- **Free Tier eligible**: If you are in the Free Tier, you'll want to select an AMI marked as Free Tier eligible. AWS Documentation+1
- **Quick Start / Marketplace / Community**: There are categories: quick-start OS images, Marketplace images (paid or free), community/shared AMIs. AWS Documentation

---

3. Instance Type

- **Instance type**: Determines the hardware configuration: number of vCPUs, amount of RAM, network performance, etc. Example: `t3.micro`, `m5.large`, etc. Amazon Web

Services, Inc.+1

- You choose the type based on your workload (CPU bound vs memory bound vs general purpose).
- Free Tier often restricts to small instance types like `t2.micro` or `t3.micro` depending on date/account. AWS Documentation+1

---

4. Key Pair (Login)

- **Key pair name**: You can choose an existing key pair or create a new one. This is especially for SSH login (Linux) or RDP (Windows).
- If you proceed without a key pair (not recommended for production), you may have no way to connect securely. AWS Documentation+1

---

5. Network Settings

This section covers how your instance fits into your network (VPC, subnet, security group, public IP, etc).

- **VPC / Subnet**: Choose which VPC (virtual private cloud) and subnet the instance will be launched into.
- **Auto-assign Public IP**: Whether the instance gets a public IPv4 address at launch (makes it reachable from the internet). Default differs for default vs non-default subnet. AWS Documentation
- **Security Group(s)**: Virtual firewall rules for inbound/outbound traffic to/from the instance.
- **Network interfaces / secondary interfaces**: You may optionally attach additional network interfaces (advanced).
- **Subnet type** (public/private) will determine accessibility and routing.

---

6. Configure Storage

- **Root volume + additional volumes**: The AMI defines a base/root volume; you can add more (EBS volumes) or use instance store volumes depending on instance type. AWS Documentation
- **Volume Type**: For EBS you might choose types like `gp3`, `gp2`, `io1`, `io2`, `st1`, etc. These differ in performance, cost, IOPS. AWS Documentation
- **Size (GiB)**: You specify the size of each volume.
- **IOPS / Throughput**: For certain volume types (`io1`,`io2`,`gp3`) you may need to

specify IOPS (for high performance) and throughput. AWS Documentation

- **Delete on termination**: Whether the volume (especially root) is deleted when the instance is terminated. Helps avoid orphan volumes/cost. AWS Documentation
- **Encryption**: Whether the volume is encrypted; if yes, which KMS key to use. AWS Documentation
- **Device name**: Which device the volume will appear as inside the OS (e.g., `/dev/xvda`, `/dev/sdb`).

---

7. Advanced Details

These are optional fields but often very important—especially for production deployments. According to the AWS docs, this section contains many parameters. AWS Documentation Some of the key fields include:

- **IAM instance profile**: The IAM role that will be associated with the instance. Grants permissions to the instance (for example, to read/write to S3, access other AWS services) via the role. AWS Documentation
- **Shutdown behavior**: Choose whether when you shut down the instance (from within the OS) the instance will *stop*or *terminate*.
- **Enable termination protection**: Prevents accidental termination of the instance by users via console/CLI; you must disable it before termination. AWS Documentation
- **Monitoring (detailed CloudWatch monitoring)**: By default, EC2 sends metrics to CloudWatch every 5 minutes; you can enable detailed monitoring for 1-minute intervals (incurs cost). AWS Documentation
- **Placement group**: If you want your instance in a specific placement group (clustered/hpc/partition) for network performance or fault isolation.
- **Tenancy**: Default shared hardware or dedicated host/dedicated instance. You might choose “Dedicated” if you need hardware isolation.
- **User data**: A script or cloud-init content that runs at launch to configure the instance (install packages, bootstrap, etc).
- **Elastic GPU / GPU & FPGA options**: If supported, you may choose GPU/accelerated hardware.
- **Network performance optimization**: Options such as enhanced networking (ENA), SR-IOV support, etc.
- **Boot mode**: UEFI vs legacy BIOS etc (for some OS/hardware combinations).
- **Metadata options / IAM instance metadata service version & restrictions**: You can configure the version and use of Instance Metadata Service (IMDS) to improve security.

- **License options**: For certain OS or software you may specify license types.
- **Tag specifications at launch**: Tags for the instance, volumes, network interfaces, etc.

Linux Fundamentals

This guide covers three essential sections:

1. **Basic Linux Commands** – `mkdir`, `cd`, `touch`, etc.
2. **File Management Commands** – `ls`, `find`, etc.
3. **Process Management Commands** – `ps`, `top`, etc.

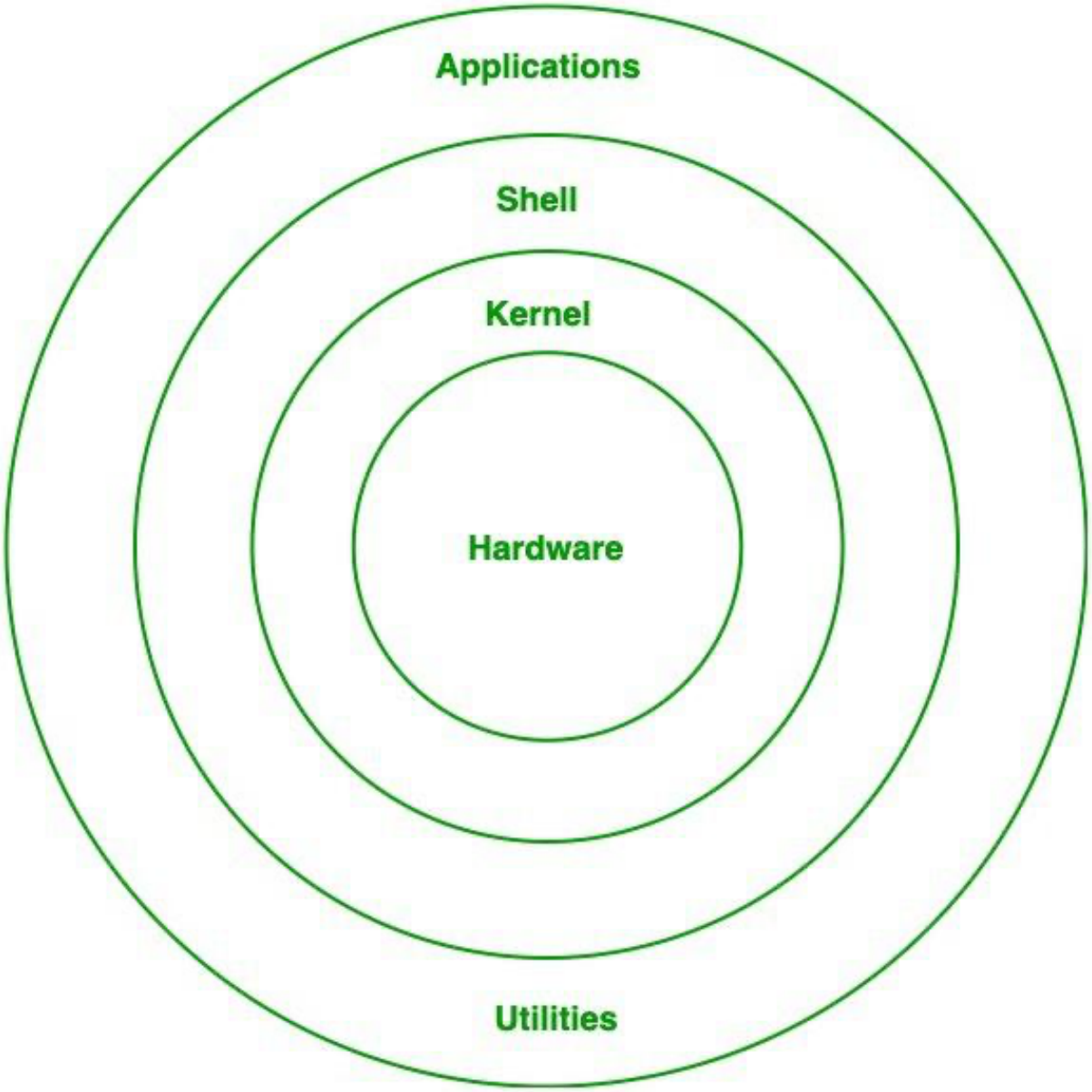
Each section includes:

- Command explanation
- Syntax
- Examples
- Lab exercises
- Expected outcomes

What is Linux?

Linux is an open-source operating system widely used in servers, cloud systems, DevOps pipelines, and automation environments.

Linux Architecture



What is Linux Kernel?

- It manages **CPU, memory, devices, and filesystems**.
- It provides **system calls** so applications can interact with hardware safely.
- It controls **process scheduling, networking, and security**.

Think of it as the **"brain" of Linux**, running in the background and making everything else work.

What is the Shell?

The **shell** is a command-line interpreter that lets you interact with the Linux kernel. Common shells:

- `bash` (default in most systems)
- `zsh`, `sh`, `fish`, etc.

Why Learn Shell Commands?

- Faster automation
- Foundation for DevOps scripting
- Easier debugging and system navigation

---

1. Basic Linux Commands

These are the foundational commands that help users create, navigate, and manage files/directories.

Commands Covered:

`pwd`, `cd`, `mkdir`, `rmdir`, `touch`, `cat`, `echo`, `clear`, `history`

Command: `pwd`

Purpose: Displays the current working directory.
Syntax:

```
pwd
```

Example:

```
[user@linux ~]$ pwd
/home/user
```

Command: `cd`

Purpose: Change the current directory.
Syntax:

```
cd [directory_path]
```

Examples:

```
cd /home/user/Documents        # Move to Documents folder
```

```
or
cd /tmp/

cd ..                          # Move to parent directory
cd ~                           # Move to home directory
```

Command: `mkdir`

Purpose: Create new directories.
Syntax:

```
mkdir [directory_name]
```

Examples:

```
mkdir projects
mkdir -p /home/user/code/python  # Create nested directories
```

Command: `rmdir`

Purpose: Remove empty directories.
Example:

```
rmdir old_folder
```

Command: `touch`

Purpose: Create empty files or update timestamps.
Example:

```
touch file1.txt
```

Command: `cat`

Purpose: Display or concatenate file content.
Examples:

```
cat file1.txt                  # View file contents
cat file1.txt file2.txt        # view both files
```

Command: `echo`

Purpose: Print text or write text into files.
Examples:

```
echo "Hello, Linux!"           # Print text
echo "Linux Lab" > lab.txt     # Write to file
```

Command: `history`

Purpose: Show list of previously executed commands.
Example:

```
history
```

Lab Exercise 1 – Basic Commands

1. Create a directory named `starting_point`.
2. Navigate inside it.
3. Create a file called `secret_lair.txt`.
4. Display the absolute path of your current location.
5. View the history of commands executed.

Expected Outcome:

- Students can create, navigate, and manipulate files/directories.

---

2. File Management Commands

These commands let you list, find, copy, move, or remove files efficiently.

Commands Covered:

`ls`, `cp`, `mv`, `rm`, `find`, `locate`, `du`, `df`

Command: `ls`

Purpose: List directory contents.
Examples:

```
ls                          # Basic list
ls -l                       # Long list (permissions, owners, etc.)
ls -a                       # Include hidden files
ls -lh                      # Human-readable format
```

Command: `cp`

Purpose: Copy files or directories.
Examples:

```
cp file1.txt /backup/
cp -r folder1/ /backup/     # Copy directory recursively
```

Command: `mv`

Purpose: Move or rename files.
Examples:

```
mv file1.txt /tmp/
mv oldname.txt newname.txt
```

Command: `rm`

Purpose: Remove files or directories.
Examples:

```
rm file1.txt
rm -r temp_folder
```

Use with caution — no recycle bin!

Command: `find`

Purpose: Search for files and directories.
Examples:

```
find /home/user -name "*.log"       # Find all .log files
find . -type f -size +50M           # Find files > 50MB
```

Command: `du` and `df`

Purpose: Display disk usage and free space.
Examples:

```
du -sh /var/log/      # Total size of directory
df -h                 # Disk space usage overview
```

Lab Exercise 2 – File Management

1. Check if `delete_directory` exists and remove it.
2. Delete `delete.txt`.
3. Use `find` to locate all `.txt` files under `/home/user`.
4. Display disk usage of `/var/log/`.

Expected Outcome:
Students understand file management operations, safe deletion, and disk space checking.

---

3. Process Management Commands

Processes represent running programs. Understanding how to list, monitor, and control them is vital for DevOps and system admins.

Commands Covered:

`ps`, `top`, `htop`, `kill`, `nice`, `renice`, `jobs`, `bg`, `fg`

Command: `ps`

Purpose: Display currently running processes.
Examples:

```
ps                          # Current user's processes
ps -ef                      # All processes in system
ps aux | grep nginx          # Find specific process
```

Command: `top`

Purpose: Real-time system and process monitoring.
Usage:

```
top
```

Press `q` to quit, `k` to kill, `r` to renice.

Command: `htop` *(if installed)*

Purpose: Enhanced version of `top` with color and interactivity.
Usage:

```
htop
apt install htop -y
```

Command: `kill`

Purpose: Terminate a process using its PID.
Examples:

```
sleep 100 &

kill 1234                # Send SIGTERM
kill -9 1234             # Forcefully terminate (SIGKILL)
```

Command: `jobs`, `bg`, `fg`

Purpose: Manage background/foreground jobs.
Examples:

```
sleep 100 &
jobs                        # List background jobs
fg %1                       # Bring job 1 to foreground
```

Lab Exercise 3 – Process Management

1. List all running processes.
2. Identify the process ID of `sshd`.
3. Start a background job using `sleep 60`.
4. Bring it to foreground using `fg`.
5. Kill a specific process using its PID.

Expected Outcome:
Students understand process listing, management, and termination safely.

Extended Concept Practice: Permissions and Ownership

Key Commands:

`chmod`, `chown`, `chgrp`

Examples:

```
chmod 755 file.sh
chown user:group file.sh
```

Lab Exercise 4 – Advanced Challenge

Work with the hidden directory `secret_lair`:

- Count subdirectories (“rooms”)
- Identify which one is permission-restricted
- Find command to change ownership to `user`
- Write all answers to `/home/user/answer.txt`