# Cloud Learning Document

## Cloud computing services model

1.Iaas (infrastructure as service): In this, the entire hardware will be its own and it will give you an operating system by installing it, we will see what to do in that operating system.

```
eg: EC2 is eg of iaas,
```

2.Paas(platform as a service):you are db admin or admin u purchased 15-20 server and giving database service application after installation we have to manage data

eg: rds is platform as service

3.Saas(software as service): If you use mail, it is a part of SaaS because we have added service to it and paying the money, everything will be maintained by the end user.

## Deployment computing services model:

1. public cloud :-over the internet we use resources and we create any resources and it is available to public all over the world public:-use multiple hardware to deploy using virtualization undeline same path
2. private cloud: client specifically assign to include all the hardware all data center assign to one organnization company so that to use the resources in company network access.

More secure and not customizable

1. hybrid cloud is both organization take there own public cloud + private cloud

### Difference Between Public Cloud and Private Cloud

| Feature | Public Cloud | Private Cloud |
|---|---|---|
| **Definition** | Cloud services offered over the internet to multiple users (shared infrastructure) | Cloud infrastructure used exclusively by one organization (dedicated infrastructure) |
| **Ownership** | Owned and managed by a cloud provider (e.g., AWS, Azure, GCP) | Owned and managed by the organization or a third-party provider |
| **Cost** | Pay-as-you-go; lower upfront cost | Higher cost due to hardware and maintenance needs |
| **Scalability** | Highly scalable (virtually unlimited resources) | Limited scalability based on on-premises hardware |

| | | |
|---|---|---|
| **Security** | Shared responsibility model; secure but shared | More control over security; used for sensitive data |
| **Maintenance** | Managed by the cloud provider | Managed by the organization (or outsourced) |
| **Access** | Accessible over the public internet | Restricted to internal networks or VPNs |
| **Example Providers** | AWS, Microsoft Azure, Google Cloud | VMware, OpenStack, HPE Private Cloud |

# Aws region:

Region: 36 launched region

**114 Availability Zones**

43 local zone

region in one region there are multiple avaliablity zone

min-2

max-6 hoskte h

wg of webiste want to create and see where is my audienc is so according to its created

sometimes its is particularly present in specific area only

Why we use:

1. to imrpove performace
2. govt ka data in specific region only
3. disaster recovery if something happend to ek replica created in another region and ur data is safe
4. region has there own specific code
5. 200+ services are there in aws
6. ec2:-mai u=infra crreate krskte h s3 mai storage use krskte h

AWZ availablity zone:

eg kai liye: ek region eg mumabai region mai multiple data center h ek jaga

mai h and suddenly elecctricuty cut hogi toh machine bndh hogi application down hoga

netwrok isssue agya

ek region multiple availablity zone h about distance of 60-100 km

har ek avalablity zone different power house sai cionnect rheta h and if ek bndh ho rha toh dusra chal rha so that application run kre

data transfer in encrypted form

eg region code is ap-south-1

availablityzone code is :ap-south-1 a, ap-south-1 b like this

network speed is very high

# Local zone:

**AWS Local Zones** are an extension of an AWS Region that brings **compute, storage, and other services closer** to end-users in specific geographic locations.

---

### Main Reasons to Use Local Zones:

| Purpose | Explanation |
|---|---|
| **Low Latency** | Reduces the time it takes for data to travel, ideal for apps that require **<10ms latency** (e.g., gaming, video editing, real-time streaming). |
| **Proximity to Users** | Helps host workloads closer to large population centers or data-sensitive locations where no full AWS Region exists. |
| **Data Residency** | Helps meet **local compliance and data residency** requirements by keeping data within a certain city or country. |
| **Edge Computing** | Useful for **edge applications** like AR/VR, live video processing, and machine learning inference. |
| **Hybrid Deployments** | You can run **low-latency workloads** locally while still using the full AWS Region for backup, analytics, or heavy computation. |

---

### Example Use Cases:

- **Gaming**: Hosting multiplayer game servers near players for faster response.
- **Media & Entertainment**: Live video production and editing in cities with Local Zones.

- **Healthcare**: Complying with data locality laws by storing and processing medical data nearby.
- **Financial Services**: High-speed trading systems needing ultra-low latency.

---

## Example Cities with Local Zones:

- Los Angeles
- Dallas
- Miami
- New Delhi
- Taipei

(*AWS keeps expanding Local Zones globally*)

EC2:- is elastic compute cloud

aws machine hum aws kai infrastructure mai bnaye gai remotely access through ssh

1. which os u want linux ubantu
2. commpute power ??
3. ram ?
4. storage ?
5. network speed rules set krskte h which netowork in and out'

machine create krne kai key zruri h to access the data

**Firewall (security groups)**

**Info**

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group

Security group means which security is attached to ur server

has 2 rules'

1. inound rule: which means jb hum koi bhui traffic behj rhe server ko jis port pai chle ga usee inbound khete h
2. outbond means server kha sai access krskta h internet sai access kre or wrb server ko access kre

## Aws vpc : virtual private cloud:netwrok setup krte h

it is a networking service of aws

multiple desktop h toh vo apas mai connect kr skee and file sharing mai help ho skee

u have to give private ip bracuse creating private network not creating public network

it is scope of region scope available in every availablity zone and region

min ip address=00.0.0.0.0.0.

max ip address=1.1.1.1.11.111.1

why use cidr classless inter domain routing /24 means first 24 bit static hogi

192.168.0.0 /24

.0 : host id

192.168.0- network id

netwrok issue atte security na hoti

S3 bucket:- when we create a snapshot in aws ami there we store our snap shot in s3 storage

## What is Amazon S3?

Amazon S3 (Simple Storage Service) is a **scalable, durable, and secure** object storage service used for:

- Storing files (objects) like images, videos, logs, backups, etc.
- Hosting static websites
- Big data analytics
- Application data storage
- Backup and recovery

---

# Commands:

aws --version

aws configure

uploading the file:

aws s3 cp "D:\Bridgelabz\Pdf files\Trello_Documentation.docx.pdf" s3://buckets06/Trello_Documentation.docx.pdf

downloading the file:

aws s3 cp s3://buckets06/Trello_Documentation.docx.pdf "C:\Users\kuber\Downloads"

# S3 Key Concepts

| Term | Description |
|---|---|
| **Bucket** | A container for storing objects (files) |
| **Object** | The actual data (file) stored in a bucket |
| **Key** | The unique name assigned to an object within a bucket |
| **Prefix** | A way to simulate folders within a flat namespace using `/` |
| **Region** | Buckets are created in a specific AWS region |
| **Storage Classes** | Controls cost and availability (Standard, Infrequent Access, Glacier, etc.) |

---

# Key Features of S3

| Feature | Description |
|---|---|
| **Unlimited Storage** | Store virtually unlimited data |
| **High Durability** | 99.999999999% durability (11 9s) |
| **Versioning** | Keep multiple versions of objects |
| **Lifecycle Policies** | Automatically move or delete objects based on rules |
| **Access Control** | Fine-grained permissions using IAM, bucket policies, and ACLs |
| **Static Website Hosting** | Host static HTML/CSS/JS websites |
| **Event Notification** | Trigger Lambda, SQS, or SNS on object events (PUT, DELETE, etc.) |
| **Encryption** | Server-side (SSE) and client-side encryption supported |
| **Logging and Analytics** | Enable logging for requests and access |

---

# Useful AWS CLI Commands for S3

## ◆ Bucket Operations

| Task | Command |
|---|---|
| List all buckets | `aws s3 ls` |
| Create a bucket | `aws s3 mb s3://my-bucket-name` |
| Delete a bucket | `aws s3 rb s3://my-bucket-name --force` |
| Enable versioning | `aws s3api put-bucket-versioning --bucket my-bucket-name --versioning-configuration Status=Enabled` |

---

## ◆ Upload/Download Objects

| Task | Command |
|---|---|
| Upload a file | `aws s3 cp myfile.txt s3://my-bucket-name/` |
| Upload a directory | `aws s3 cp myfolder/ s3://my-bucket-name/ --recursive` |
| Download a file | `aws s3 cp s3://my-bucket-name/myfile.txt ./` |
| Sync local and S3 | `aws s3 sync ./local-folder/ s3://my-bucket-name/` |

---

## ◆ Object Operations

| Task | Command |
|---|---|
| List objects | `aws s3 ls s3://my-bucket-name/` |
| List recursively with size | `aws s3 ls s3://my-bucket-name/ --recursive --summarize` |
| Delete an object | `aws s3 rm s3://my-bucket-name/file.txt` |
| Delete all objects in bucket | `aws s3 rm s3://my-bucket-name/ --recursive` |

---

## ◆ Permissions and Policies

| Task | Command |
|---|---|
| Add bucket policy | `aws s3api put-bucket-policy --bucket my-bucket-name --policy file://policy.json` |
| Get bucket policy | `aws s3api get-bucket-policy --bucket my-bucket-name` |

---

### ◈ Advanced Features

| Feature | Command |
|---|---|
| Enable static website hosting | `aws s3 website s3://my-bucket-name/ --index-document index.html` |
| Enable logging | `aws s3api put-bucket-logging --bucket my-bucket --bucket-logging-status file://logging.json` |
| Set lifecycle rule | `aws s3api put-bucket-lifecycle-configuration --bucket my-bucket --lifecycle-configuration file://lifecycle.json` |

---

# Common Security Features

- **Bucket Policies**: JSON policy documents for access control
- **IAM Policies**: Attach to users/roles for permissions
- **ACLs**: Not recommended but available for legacy access
- **Encryption**:
  - SSE-S3 (default server-side)
  - SSE-KMS (with KMS key)
  - Client-side encryption

---

# Monitoring & Notifications

- **Server Access Logs**
- **CloudTrail**: Logs API calls to/from S3
- **S3 Event Notifications**: On PUT/DELETE, trigger Lambda, SQS, SNS

---

# Pro Tips

- Use `sync` to mirror folders between local and S3:

- `aws s3 sync ./local-folder s3://bucket-name`
- Use **presigned URLs** for secure, temporary file access:
- `aws s3 presign s3://bucket-name/myfile.txt --expires-in 3600`
- Use **storage classes** to save cost for infrequent access:
- `aws s3 cp myfile.txt s3://bucket-name/ --storage-class STANDARD_IA`

# AWS IAM (Identity and Access Management): Overview

AWS IAM allows you to **securely control access** to AWS services and resources for users. It is a **centralized access control system** that enables you to create and manage **users, groups, roles**, and **policies**

eg:- ek company h uska aws infra manage krna h in company there are 200-300 emp and they ask for different requests services access toh not able to handle all the things alone so ek person hire krte h

team banu ga 1 person ec2 ko related req aati h validate kro and then need h create krke dai

like this different sevice provide to different person and providing the services to employees

so yeh bhhi sure krna jonsi service jisko mili h usko wahi access kre aur koi service ka access na milai

1. koi bhi resource jo h vo apni company kai account mai createe ho
2. jisko jo service ka accesss h usko dusri koi aur service ka access na ho manage krne kai liye iam atta h
3. problem is user banaya ek region mai toh usko 2usre and 3ersre region ka access nai h toh usko iam sai gloabal service h har ek region milai ga

Groups in iam :- basscially it is used to inherit all the policy from groups to user here group is parent and user is child

and ek user ko multiple groups mai dal skta hu

---

# Key Concepts

### 1. Users

- **Definition**: An individual identity with long-term credentials (username + password or access keys).
- **Used for**: Human or programmatic access to AWS.
- **Authentication**: Proves the identity of the user.
    - **Methods**:

        - **Console access**: Username + Password
        - **Programmatic access**: Access key ID + Secret access key

### 2. Groups

- **Definition**: A collection of IAM users. Policies attached to the group apply to all users in it.
- **Used for**: Managing permissions in bulk (e.g., "developers", "admins").

### 3. Roles

- **Definition**: An IAM identity with permissions that can be assumed by a user or service.
- **Used for**: Temporary access, cross-account access, EC2 instance access, Lambda, etc.

### 4. Policies

- **Definition**: JSON documents that define permissions (who can do what on which resources).
- **Types**:
    - **AWS Managed Policies**: Predefined by AWS.
    - **Customer/User Managed Policies**: Created and maintained by you.
    - **Inline Policies**: Directly attached to a single user, group, or role.
    - **Identity-based policies**: Attached to users, groups, or roles.
    - **Resource-based policies**: Attached directly to a resource (like S3 buckets, Lambda, etc.)

---

# Activity: Create an IAM user and assign a Custom Policy to control EC2 access

### Goal:

- Create IAM user `backend-dev`.
- Allow only limited EC2 actions: `start`, `stop`, `describe`, `terminate` for a specific EC2 instance (your backend instance).

---

# Step-by-Step Instructions:

### Step 1: Identify Your EC2 Instance

- Go to **EC2 Console** → Instances → Note down the **Instance ID**, e.g., `i-0a1234b5c6789def0`.

---

## Step 2: Create IAM User

1. Go to **IAM Dashboard**.
2. Click **Users → Add user**.
3. Enter user name: `backend-dev`.
4. Select **Access type**:
    - **Programmatic access** (for CLI/SDK)
    - **AWS Management Console access** (optional)
5. Set the password or access keys (record secret access key securely).
6. Click **Next: Permissions** → Skip for now.
7. Finish the creation.

---

## Step 3: Create a Custom Managed Policy

1. Go to **IAM Dashboard → Policies → Create Policy**.
2. Choose the **JSON tab**.
3. Paste the below policy:

```
json
CopyEdit
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowLimitedEC2AccessToSpecificInstance",
      "Effect": "Allow",
      "Action": [
        "ec2:StartInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances",
        "ec2:DescribeInstances"
      ],
      "Resource": "arn:aws:ec2:region:account-id:instance/i-
0a1234b5c6789def0"
    },
    {
      "Sid": "AllowDescribeInstancesForAll",
      "Effect": "Allow",
      "Action": "ec2:DescribeInstances",
      "Resource": "*"
    }
  ]
}
```

➧ Replace:

- `region` with your EC2 region (e.g., `us-east-1`)
- `account-id` with your 12-digit AWS account ID
- `instance ID` with your EC2 instance ID

1. Click **Next**, name it `BackendEC2AccessPolicy`, and add a description.
2. Create the policy.

---

### Step 4: Attach the Policy to User

1. Go to **Users** → `backend-dev` → **Permissions** tab.
2. Click **Add Permissions** → **Attach existing policies directly**.
3. Search and select `BackendEC2AccessPolicy`.
4. Add the policy.

---

## Summary of Permissions Given

| Action | Allowed | Scope |
|---|---|---|
| `StartInstances` | ✓ | Only on specific instance |
| `StopInstances` | ✓ | Only on specific instance |
| `TerminateInstances` | ✓ | Only on specific instance |
| `DescribeInstances` | ✓ | All instances |

# Commands EC2:

PS C:\Users\kuber\Downloads> ssh -i .\mykey.pem ubuntu@13.235.90.27

for connecting

| Part | Meaning |
|---|---|
| `ssh` | Secure Shell command — used to connect to remote Linux servers securely. |

| | |
|---|---|
| `-i .\\mykey.pem` | `-i` stands for **identity file**. This is your **private key** used to authenticate with the EC2 instance. `.\\` means the file is in the current PowerShell directory. |
| `ubuntu@13.235.90.27` | You're connecting to a remote server (EC2) with: `ubuntu` → the **username** of the remote Linux machine (for Ubuntu EC2 AMIs) `13.235.90.27` → the **public IP address** of your EC2 instance. |

scp -i "C:\Users\kuber\Downloads\mykey.pem" "D:\Bridgelabz\Pdf files\Trello_Documentation.docx.pdf" ubuntu@13.235.90.27:/home/ubuntu/

uploading the file

| Part | Meaning |
|---|---|
| `scp` | The command-line tool used to **securely copy files over SSH**. |
| `-i "C:\\Users\\kuber\\Downloads\\mykey.pem"` | Specifies the **SSH private key file** (your `.pem` file) used for authentication with the EC2 instance. |
| `"D:\\Bridgelabz\\Pdf files\\Trello_Documentation.docx.pdf"` | This is the **local file path** you want to upload. The double quotes are necessary because the folder name has a **space** (`Pdf files`). |
| `ubuntu@13.235.90.27:` | This specifies the **remote user** (`ubuntu`) and the **public IP address** of your EC2 instance. You're saying, "Log in to this remote EC2 server as `ubuntu`." |
| `/home/ubuntu/` | The **destination directory** on the EC2 server where the file should be uploaded. This is typically the home directory for the `ubuntu` user. |

scp -i "C:\Users\kuber\Downloads\mykey.pem" ubuntu@13.235.90.27:/home/ubuntu/Trello_Documentation.docx.pdf "C:\Users\kuber\Downloads"

for downloading

| Part | Meaning |
|---|---|

| | |
|---|---|
| `scp` | Secure Copy — used to transfer files between local and remote systems using SSH. |
| `-i "C:\\Users\\kuber\\Downloads\\mykey.pem"` | Specifies the **SSH private key** to authenticate with the EC2 instance. |
| `ubuntu@13.235.90.27:` | The **remote login** — you're accessing the EC2 instance as the `ubuntu` user. |
| `/home/ubuntu/Trello_Documentation.docx.pdf` | The **file on the EC2 instance** that you want to download. |
| `"C:\\Users\\kuber\\Downloads"` | The **destination folder on your local Windows machine** where the file will be saved. |

# Jenkins:-

# 1. Introduction to CI/CD

### What is CI/CD?

CI/CD stands for **Continuous Integration** and **Continuous Delivery/Deployment**. It is a set of practices that enable development teams to deliver code changes more frequently and reliably.

**Key Concepts:**

- **Continuous Integration (CI):** Developers regularly merge code into a shared repository.
- **Continuous Delivery (CD):** Code is automatically prepared for a release to production.
- **Continuous Deployment:** Every change that passes automated tests is deployed automatically.

**Benefits of CI/CD in Software Development:**

- Faster delivery of features.
- Early detection of bugs.
- Reduced integration issues.
- More frequent and reliable deployments.
- Improved collaboration between development and operations teams.

---

### Continuous Integration (CI)

**Definition and Purpose:**

CI is the practice of merging all developer working copies to a shared mainline several times a day, followed by automated testing.

**Key Practices:**

- **Frequent code commits** to version control (e.g., Git).
- **Automated builds** and **unit testing**.
- Use of CI tools like Jenkins, Travis CI, CircleCI.
- **Static code analysis** for code quality checks.

---

### Continuous Delivery (CD)

**Definition and Purpose:**

CD ensures that code is **always in a deployable state**. After CI, the code is automatically prepared for production deployment.

**Continuous Delivery vs. Continuous Deployment:**

| Feature | Continuous Delivery | Continuous Deployment |
|---|---|---|
| Deployment | Manual trigger | Automatic after tests pass |
| Risk Level | Controlled | Higher automation risk |
| Use Case | Environments needing approval | Rapid production environments |

---

# 2. Introduction to Jenkins

### What is Jenkins?

Jenkins is an **open-source automation server** used to implement CI/CD pipelines.

**Overview and History:**

- Initially developed as **Hudson** by Sun Microsystems.
- Renamed Jenkins in 2011 after a fork.
- Written in Java and highly extensible.

**Key Features and Benefits:**

- Automates builds, tests, and deployments.
- Supports plugins for integration with other tools (Git, Docker, Maven, etc.).

- Scalable with a distributed architecture (master/agent).

---

### Jenkins Architecture

**Master and Agent Nodes:**

- **Master (Controller):** Schedules jobs, monitors agents, and manages Jenkins UI.
- **Agents (Slaves):** Run the actual build/test/deploy tasks.
- Enables **distributed builds** across multiple machines.

**Jenkins Plugins:**

- Over **1800 plugins** available.
- Add functionality like Git integration, Slack notifications, Docker support, etc.
- Central to Jenkins' flexibility and extensibility.

---

# 3. Installing Jenkins

## Installation Methods:

- **Linux (Debian/Ubuntu):**

  ```
  bash
  ```

  CopyEdit

  ```
  sudo apt update sudo apt install openjdk-11-jdk wget -q -O -
  https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add - sudo
  sh -c 'echo deb https://pkg.jenkins.io/debian binary/ >
  /etc/apt/sources.list.d/jenkins.list' sudo apt update sudo apt install
  jenkins
  ```

## Initial Setup and Configuration:

- Access Jenkins at `http://localhost:8080`.
- Unlock Jenkins using the admin password found in `/var/lib/jenkins/secrets/initialAdminPassword`.
- Install suggested plugins.
- Create first admin user.

---

# 4. Jenkins Basics

### Jenkins User Interface:

- **Dashboard:** Overview of jobs.
- **Manage Jenkins:** Configure system settings, plugins, and nodes.
- **Job Configuration:** Set job details like triggers, build steps, and post-build actions.

### Creating Your First Jenkins Job:

#### Types of Jenkins Jobs:

- **Freestyle Project:** Basic job configuration using GUI.
- **Pipeline Project:** Defines build logic as code (Jenkinsfile).

#### Configuring a Simple Freestyle Job:

- Click **"New Item"**, name it, select **"Freestyle project"**.
- Configure **source code management** (e.g., Git repo).
- Add **build steps** (e.g., shell script).
- Add **post-build actions** (e.g., email notification).

---

# 5. Jenkins Plugins

### Introduction to Jenkins Plugins:

- Plugins extend Jenkins' functionality.
- Installed via **Manage Jenkins → Manage Plugins**.
- Examples:
    - **Git Plugin:** Integrate Git.
    - **Pipeline Plugin:** Enable pipeline jobs.
    - **Docker Plugin:** Interact with Docker containers.

---

# 6. Jenkins Pipeline

### Introduction to Jenkins Pipelines:

#### Definition and Benefits:

- Pipeline is a **suite of plugins** that support integration and implementation of continuous delivery pipelines.
- Defined as code using **Jenkinsfile**.

**Declarative vs. Scripted Pipelines:**

| Type | Declarative | Scripted |
|---|---|---|
| Syntax | Structured and user-friendly | Groovy-based, more flexible |
| Readability | Easier to read and maintain | More complex |
| Example Start | `pipeline { ... }` | `node { ... }` |