Cloud Computing (Easy Notes)

1) Welcome & Learning Objectives

Goal: Understand cloud basics, deployment models, service models, virtualization, containers, scaling, and practical IaaS use cases. You'll be able to pick the right model (SaaS/PaaS/IaaS/FaaS), explain private vs. public vs. hybrid/multi-cloud, and reason about costs.

---

2) What is Cloud Computing?

- **Simple idea:** Rent computing over the internet (servers, storage, databases, networks, tools). Pay for what you use.
- **Analogy:** Instead of buying a power generator, you plug into the electricity grid.
- **Why:** Faster setup, elastic capacity, global reach, less upfront cost.

---

3) Deployment Models

Public Cloud

- **Owned/operated by:** A provider (AWS, Azure, GCP). Shared infrastructure, logically isolated.
- **Pros:** Quick start, low upfront cost, global scale, rich managed services.
- **Cons:** Ongoing cost, data egress fees, potential **vendor lock-in**.
- **Use when:** You want speed, elasticity, and minimal maintenance.

Private Cloud

- **Owned by:** Your organization (on-prem or hosted).
- **Pros:** Control, customization, easier compliance for strict data.
- **Cons:** Hardware cost, capacity planning, slower to scale.
- **Use when:** Strong regulatory/security isolation is required.

Hybrid Cloud

- **Mix:** On-prem/private + public. Workloads can move/split.
- **Pros:** Keep sensitive data on-prem, burst to cloud.
- **Cons:** Integration complexity, networking & identity challenges.

Multi-cloud

- **Use multiple public clouds** (e.g., AWS + Azure).
- **Pros:** Reduce lock-in, choose best-of-breed services.

- **Cons:** Skills/tooling duplication, more ops complexity.

---

4) Hyperscale Providers

- **AWS:** Huge service catalog; popular: EC2 (compute), S3 (object storage), RDS (managed DB), Lambda (FaaS).
- **Azure:** Deep enterprise/Windows integration; popular: VM, Blob Storage, Azure SQL, Functions.
- **GCP:** Strong data/AI; popular: Compute Engine, Cloud Storage, BigQuery, Cloud Functions.

---

5) Service Models (Who manages what?)

```
On-Prem: You manage everything (hardware → apps).
IaaS: Provider manages hardware/virtualization; you manage OS → apps.
PaaS: Provider manages OS/runtime; you deploy code + data.
SaaS: Provider delivers full app; you just use it.
FaaS: You write small functions; provider runs them on demand.
```

SaaS — Software as a Service

- **What:** Ready-to-use apps (e.g., Gmail, Salesforce).
- **You manage:** Users, data, configs.
- **Good for:** Fast business capability, no infra.

IaaS — Infrastructure as a Service

- **What:** Virtual machines, networks, storage (you build on top).
- **You manage:** OS, patches, middleware, your app.
- **Good for:** Lift-and-shift, custom stacks.

PaaS — Platform as a Service

- **What:** Managed runtime (e.g., App Engine, Azure App Service, Elastic Beanstalk).
- **You manage:** Your code + data.
- **Good for:** Focus on coding, minimal ops.

FaaS — Function as a Service (Serverless compute)

- **What:** Event-driven functions (e.g., AWS Lambda, Azure Functions, Cloud Functions).
- **Billing:** Per execution/time. Auto-scales.
- **Good for:** APIs, cron jobs, data processing triggers.

---

6) Why Virtualization?

Underutilized Physical Servers

- Traditional model: One app per server ⇒ low CPU usage, wasted money.
- **Goal:** Pack multiple apps per physical machine safely.

Virtual Machines (VMs)

- **Hypervisor** slices a physical server into many VMs, each with its own OS.
- **Type-1 hypervisor:** Bare-metal (e.g., ESXi, Hyper-V).
- **Type-2:** Runs on host OS (e.g., VirtualBox).
- **Pros:** Strong isolation, flexible OS choice.
- **Cons:** Heavier than containers (each VM has full OS).

---

7) Scaling Applications

Vertical Scaling (Scale-Up)

- **Add power** to a single machine (more CPU/RAM).
- **Easy but limited** by maximum machine size.

Horizontal Scaling (Scale-Out)

- **Add more machines** behind a load balancer.
- **Stateless** services scale best; session/data goes to shared stores.

---

8) Microservices & Cloud-Native Apps

- **Monolith:** One big deployable.
- **Microservices:** Many small services, each focused on a business capability. Communicate via APIs/queues.
- **Cloud-native principles:** 12-factor apps, automation, containers, CI/CD, observability.
- **Pros:** Independent deploys, better scalability.
- **Cons:** Distributed complexity (networking, tracing, data consistency).

---

9) Containers

- **What:** Lightweight units that package app + dependencies (no full OS per app).
- **How:** Share host OS kernel; isolated user space (namespaces/cgroups).

- **Docker:** Popular tool to build/run containers.
- **Orchestration:** Kubernetes (K8s) schedules, scales, heals containers.

Benefits of Containers

- Fast startup, portable, repeatable builds, efficient resource usage, fits CI/CD, easy rollback with images.

VM vs Container (quick view):

- VM = heavy isolation (full OS), stronger boundary, slower start.
- Container = light isolation (same kernel), very fast, dense packing.

---

10) IaaS Deep Dive

Compute

- **VM sizes:** Choose CPU/RAM/accelerators (GPU).
- **Auto-scaling:** Adds/removes instances based on load.
- **Disks:**
  - *Block storage* (attach to VM; SSD/HDD; good for OS/db disks).

Storage

- **Object storage:** Buckets for files/blobs; cheap, durable; access via API (e.g., S3/Blob/Cloud Storage).
- **Block storage:** Virtual disks for VMs/databases.
- **File storage:** Shared file system (SMB/NFS) for multiple VMs.
- **Tiers:** Hot (frequent), Cool/Cold/Archive (infrequent) with cheaper storage but higher access cost/latency.

Networking

- **VPC/VNet:** Your private network in the cloud.
- **Subnets, route tables, firewalls/NSG, NAT, VPN/Direct Connect/ExpressRoute**, **Load Balancers**, **DNS**, **CDN** for global caching.

---

11) Pricing Models (high-level)

- **Pay-as-you-go (on-demand):** No commitment; highest unit price.
- **Reserved/Committed:** 1–3 year commitment for discounts.
- **Spot/Preemptible:** Use spare capacity; very cheap; can be interrupted.

- **Free tiers/credits:** Limited resources for learning.
  Cost drivers: Compute hours, storage GB-months, requests, **data egress** (leaving the cloud).

---

12) Pros & Cons of IaaS

Advantages:

- Elastic capacity, fast provisioning, global reach, many managed add-ons, improved reliability (multi-AZ/region).
  Disadvantages:
- Lock-in risk, complex pricing, security is **shared responsibility**, compliance/latency concerns, skills required.

---

13) Typical Market Use Cases

- **Lift-and-shift:** Move VMs to cloud with minimal changes.
- **Web/mobile backends:** Autoscale behind load balancer.
- **Data & Analytics:** Data lakes, ETL, warehousing.
- **Disaster Recovery/Backup:** Replicate to another region.
- **Batch/HPC/Rendering:** Short-lived clusters, spot VMs.
- **IoT/Streaming:** Ingest, process, store, analyze events.

---

14) Simple Azure IaaS Example (conceptual)

1. **Create Resource Group** (logical container).
2. **Create VNet/Subnet** and **Network Security Group** (firewall rules).
3. **Provision VM** (size, OS disk), attach to subnet.
4. **Add Public IP/Load Balancer** (if needed).
5. **Open Ports** (22/3389/80/443 via NSG as needed).
6. **Attach Data Disk** or mount **Blob/File Storage**.
7. **Monitor** (logs/metrics) and **configure backups**.

---

15) Quick Recap

- Choose **deployment model** (public/private/hybrid) and **service model** (SaaS/PaaS/IaaS/FaaS) based on control vs. convenience.
- Use **VMs** for full control; **containers** for speed/density; **FaaS** for event-driven bits.
- Mind **costs** (commitments, spot, egress) and **shared responsibility**.

BONUS: Extra Topics (Simple Explanations)

A) Generative AI for Absolute Beginners

- **What:** AI that creates content (text, images, code) from prompts.
- **Key idea:** Models learn patterns from huge datasets; you steer them with prompts + context.
- **Use cases:** Chatbots, content drafts, code assist, summaries.
- **Limits:** Can be wrong/confident ("hallucination"), needs guardrails.

B) Vector Databases (for beginners)

- **Problem:** Keyword search can't capture meaning well.
- **Solution:** Convert data to **embeddings** (vectors of numbers) so "similar meaning" ≈ "nearby vectors."
- **Vector DB:** Stores vectors and finds nearest neighbors fast.
- **Use:** Semantic search, recommendations, **RAG** (retrieve context for AI).

C) Prompt Engineering (for GenAI)

- **Goal:** Get reliable answers from AI.
- **Tips:**
  - Be explicit about role, task, format.
  - Provide examples ("few-shot").
  - Add constraints (tone, length, steps).
  - Iterate: test → refine → test.

D) SQL Bootcamp (Data Analysis)

Level 1: SELECT, WHERE, ORDER BY, LIMIT; JOINS (INNER/LEFT/RIGHT); GROUP BY + aggregates; basic data types.
Level 2: Subqueries, **CTEs**, **Window functions** (ROW_NUMBER, RANK, SUM OVER), CASE, indexes basics, query plans.

- **Why learn:** Turn raw tables into insights; works across tools (Postgres, MySQL, BigQuery, etc.).

E) Cloud for Beginners — IaaS (extra view)

- Plan networks, provision VMs/storage, secure with firewalls/identity, automate with IaC (Terraform, ARM/Bicep, CloudFormation).

F) Cloud for Beginners — Database Technologies

- **Relational (SQL):** Tables/rows; strong consistency; ACID (e.g., PostgreSQL, MySQL, SQL Server).
- **NoSQL:**
  - *Key-value* (Redis), *Document* (MongoDB/Firestore), *Wide-column* (Bigtable/Cassandra), *Graph* (Neo4j/Cosmos DB Graph).
- **Managed services:** RDS/Azure SQL/Cloud SQL; backups, patching handled for you.
- **Choose by access pattern:** Joins/transactions → SQL; flexible schema/high scale → NoSQL.

G) Microservices Architecture (beginner)

- **Split by business capability** (Payments, Orders, Users).
- **APIs & Gateway, Service Discovery, Config, Circuit Breakers**, **Queues/Events**, **Observability** (logs, metrics, traces).
- **Data:** Often "database per service" to reduce coupling.

H) Machine Learning (Absolute Beginners)

Level 1: Concepts—features, labels; **supervised** (regression/classification) vs **unsupervised** (clustering); train/test split.
Level 2: Pipelines—clean data, pick model, train, validate; metrics (accuracy, precision/recall, ROC-AUC, RMSE); avoid overfitting.
Level 3: Deployment—batch vs real-time; MLOps (versioning data/models, monitoring drift), responsible AI (bias, privacy).

I) Electronics Theory (beginner)

- **Voltage (V):** Push; **Current (I):** Flow; **Resistance (R):** Opposition. **Ohm's Law:** $V = I \times R$.
- **Series vs Parallel** circuits; **Capacitors/Inductors** store energy; **Diodes** one-way; **Transistors** switch/amplify.
- **Digital logic:** AND/OR/NOT; build to CPUs/microcontrollers.

Mini-Glossary (1-liners)

- **Egress cost:** Charge for data leaving the cloud.
- **SLA:** Provider's uptime promise (e.g., 99.9%).
- **Stateful vs Stateless:** Keeps session/data in memory vs none.
- **IaC:** Infrastructure as Code (Terraform, etc.).
- **RPO/RTO:** Backup/DR objectives (data loss/time to restore).

A Tiny Practice Plan (optional)

- **Day 1–2:** Draw a diagram of public vs private vs hybrid; list pros/cons.
- **Day 3–4:** Create a free-tier object storage bucket; upload/list/download a file.
- **Day 5–6:** Package a tiny app in Docker; run locally.
- **Day 7:** Write a small “hello” Function (FaaS) and trigger it via HTTP.

Android Developer Angle (quick)

- **SaaS/BaaS:** Firebase Auth/Firestore for quick backends.
- **FaaS:** Cloud Functions for push notifications/webhooks.
- **IaaS:** Host custom APIs your app calls.
- **Containers:** Deploy backend microservices; use CDNs for media.

Tip: Start simple with managed services; add IaaS only when you need custom control.