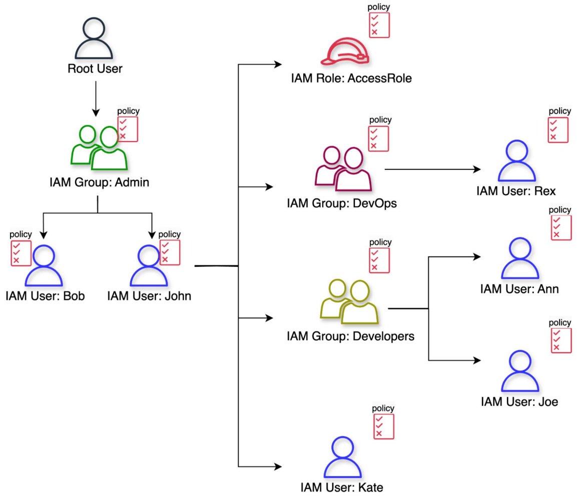
# IAM: Users & Groups

IAM (Identity and Access Management) is a global, free AWS service that centrally controls who can access your AWS account and what they can do. It manages identities (root user, IAM users, groups, roles) and their permissions for all AWS services across all Regions from one place.



**Root Account (Root User)**:
Created with the AWS account.
Has unrestricted access to all resources and settings.
Used only for critical tasks (billing, account security); never shared or used daily.

**IAM Users**:
Represent individual people or applications needing long-term access.
Each user has unique credentials (console password / access keys).
No permissions by default – everything must be granted explicitly with policies.

**IAM Groups**:
Logical collections of users only (no groups inside groups).
Typical examples: Admins, Developers, ReadOnly.
Policies are attached to groups; users inherit permissions via group membership.

**Effective Permissions (Exam Mindset)**:
A user can belong to multiple groups or none.
Final access = union of all permissions from all groups + any policies directly attached to that user.
Design access using groups and roles instead of one-off user-specific permissions.

# IAM - Permissions

IAM permissions define what actions an identity can perform and on which services/resources inside your AWS account. Permissions are controlled using JSON-based policies attached to users, groups, or roles.

Policies as Permission Documents:
JSON documents that specify allowed/denied actions (for example ec2:Describe*) on specific resources (by ARN or *).

Where Policies Attach:
Policies can be attached to users, groups, and roles. The effective permissions for an identity are the combination of all attached policies.

Least Privilege:
Always grant only the minimum permissions required to do the job. Avoid overly broad permissions like full *:* access unless absolutely necessary.

Allow vs Deny (High Level):
Effect: Allow → grants the listed actions.
Effect: Deny → always overrides allows and blocks access; useful to enforce strict guardrails (e.g. block specific services or regions).

Exam / Real-Life Mindset:
Think of policies as rules answering: Who gets this policy (via attachment)? What actions can they do? On which resources? Under which conditions?

# IAM Policies Inheritance + IAM Policies Structure

IAM policies are JSON permission documents that define which actions are allowed or denied. Permissions are inherited from all policies attached to users, groups, and roles, and evaluated together to decide if a request is allowed.

Policies Inheritance:

- A user's permissions = sum of all:
  - Policies attached directly to the user.
  - Policies attached to any groups the user belongs to.
  - Permissions from any role the user assumes.

Policy Types:

Identity-based policies: Attached to users, groups, or roles (most common).
Resource-based policies: Attached to resources (e.g., S3 bucket policy), specify who (Principal) is allowed.
Trust policies: Special resource-based policies for roles, define who can assume the role.

Version – policy language version.

Statement – one or more permission rules.

Sid – optional ID for the statement.

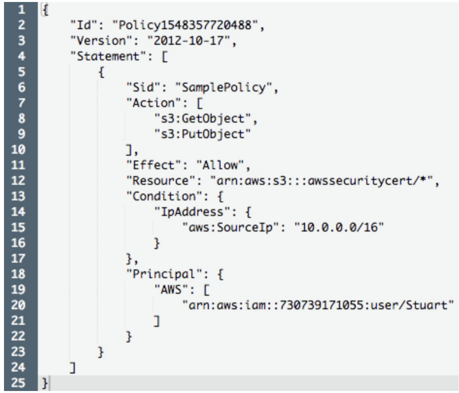
Effect – Allow or Deny.

Principal – who the policy applies to (mainly in resource-based/trust policies).

Action – which API calls are allowed/denied (e.g. s3:GetObject).

Resource – which resources (ARNs or *).

Condition – optional, adds filters (IP, MFA required, time, etc.).

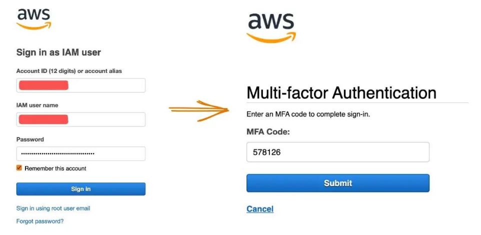


```
{
    "Id": "Policy1548357720488",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SamplePolicy",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::awssecuritycert/*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": "10.0.0.0/16"
                }
            },
            "Principal": {
                "AWS": [
                    "arn:aws:iam::730739171055:user/Stuart"
                ]
            }
        }
    ]
}
```

# Multi Factor Authentication (MFA) + MFA Devices

Multi Factor Authentication (MFA) adds an extra security layer on top of username & password, and is strongly recommended especially for the root user and privileged IAM users.

What is MFA?
Combines something you know (password) + something you have (security device/code).
If the password is stolen, the attacker still can't log in without the second factor.



Sign in as IAM user

Account ID (12 digits) or account alias: [redacted]

IAM user name: [redacted]

Password: ····································

☑ Remember this account

Multi-factor Authentication

Enter an MFA code to complete sign-in.

MFA Code: 578126

When to Use MFA:
Always enable for root user.
Enable for admins and sensitive IAM users (production, billing, security).

MFA Device Types in AWS:
Virtual MFA Apps – e.g. authenticator apps on mobile; generate 6-digit time-based codes.
U2F / Security Key (FIDO) – physical USB/NFC key (e.g. YubiKey) that you tap to approve login.
Hardware MFA Tokens – physical key fob devices (standard or GovCloud-specific) from supported vendors.

Exam / Real-Life Mindset:
"Protect the keys to the kingdom" → root & admins must have MFA.
MFA is a cheap, high-impact security control you're expected to mention in designs and exam answers.

# How can users access AWS?

Users can access AWS in three main ways: via a web console, command-line tools, or programmatically through code. Each method uses different credentials but is controlled by IAM.

## AWS Management Console

- Web-based UI for humans.
- Sign in with username (or email) + password, strongly recommended with MFA.
- Used for interactive management, viewing resources, configuring services, and admin work.

## AWS Command Line Interface (CLI)

- Access AWS from the terminal using commands.
- Uses Access Keys (Access Key ID + Secret Access Key).
- Great for automation, scripting, DevOps workflows, repeatable tasks.

## AWS SDKs (Programmatic Access)

- Language-specific libraries (Python, Node.js, Java, etc.) inside applications.
- Also use programmatic credentials (Access Keys or temporary credentials via roles).
- Allow your applications to call AWS APIs directly and securely.

## Access Keys – Key Points

- Long-term credentials for programmatic access (CLI/SDK).
- Access Key ID ≈ username, Secret Access Key ≈ password.
- Must be kept secret, rotated regularly, never committed to GitHub or shared.

# IAM Roles

IAM Roles are identities with permissions that do not have long-term credentials like passwords or access keys. Instead, they provide temporary credentials to whoever assumes them (AWS services, applications, or users). Roles are the recommended way for AWS services to securely access other AWS services.

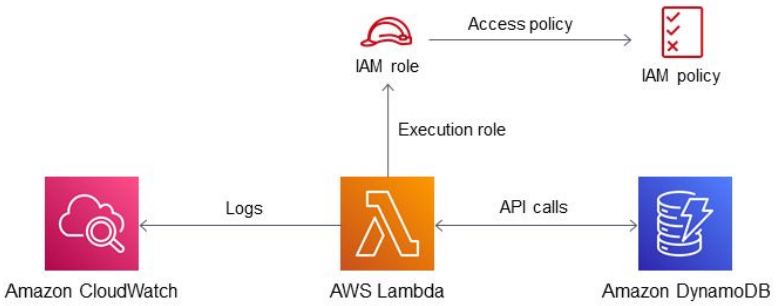
Why Roles (vs Users)?
Users = people/apps with long-term credentials.
Roles = "permission sets" assumed when needed, no static credentials to leak.
Ideal for services (EC2, Lambda, ECS), cross-account access, and temporary elevated access.

Service Roles:
Attach an IAM Role to an AWS service (e.g., EC2 instance role, Lambda execution role).
The service automatically gets temporary credentials to call other AWS services according to the role's policies.

Example use cases:
EC2 reading from S3.
Lambda writing to DynamoDB.

How It Works (High Level):
The service assumes the role.
AWS issues short-lived credentials behind the scenes.
Permissions are defined by the policies attached to the role.

Exam / Real-Life Mindset:
"Never use Access Keys on EC2, always use a Role."
Roles = secure, automatic, temporary, auditable.



# IAM Credentials Report + IAM Access Advisor

These tools help you audit security and clean up permissions in your AWS account.

IAM Credentials Report (Account-Level):

- A downloadable report (CSV) for the entire account.
- Includes all IAM users and their credential status:
- If a password is set / enabled.
- Date of last password change and last password use.
- Access keys active/inactive, creation date, last used, and rotation age.
- Whether MFA is enabled.

IAM Access Advisor:

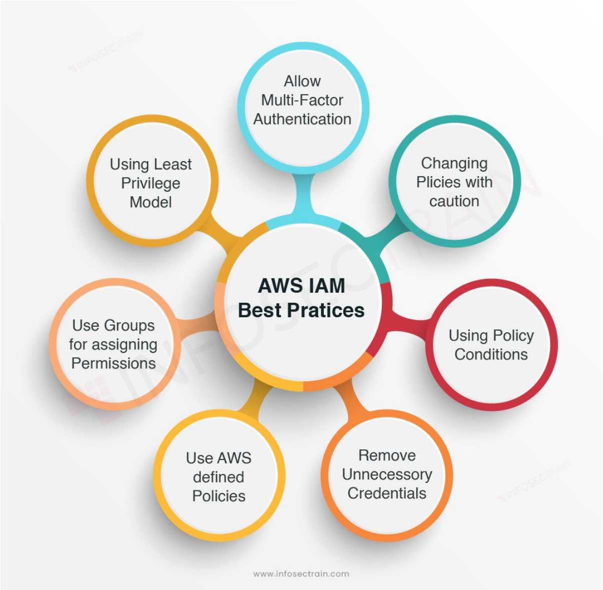
- Works per user or role.
- Shows which AWS services were actually used and when they were last accessed based on granted permissions.

Together:

1. Credentials Report → checks how identities authenticate and if their credentials are safe.
2. Access Advisor → checks what permissions are really used.

# IAM Best Practices

1. Don’t use root; protect with MFA.
2. Create individual IAM users, no shared accounts.
3. Attach permissions to groups, not directly to users.
4. Use roles for AWS services.
5. Enforce MFA.
6. Apply Least Privilege; avoid wildcards unless needed.
7. Rotate & remove unused passwords/keys; never expose them in code.
8. Review access with IAM tools and enforce a strong password policy.


AWS IAM Best Pratices
Allow Multi-Factor Authentication
Changing Plicies with caution
Using Policy Conditions
Remove Unnecessory Credentials
Use AWS defined Policies
Use Groups for assigning Permissions
Using Least Privilege Model

www.infosectrain.com

# IAM Questions

A security review found: root user used daily, no MFA, shared logins, and unused access keys. Which action aligns best with IAM best practices?

A. Keep root for daily ops but enable MFA only for IAM users

B. Disable all IAM users and use just root with a strong password

C. Enable MFA for root, create individual IAM users, use groups/roles, remove unused keys

D. Use a single IAM user for the whole team with AdministratorAccess

our company wants centralized permission management where developers are granted access based on their job function (e.g. “Developers”, “Admins”) instead of per-user policies. What is the best way to implement this?

A. Attach inline policies directly to each IAM user

B. Share the root user credentials with all admins

C. Use only resource-based policies on all services

D. Use IAM groups with managed policies attached to the groups

# IAM Questions

A security review found: root user used daily, no MFA, shared logins, and unused access keys. Which action aligns best with IAM best practices?

A. Keep root for daily ops but enable MFA only for IAM users
Wrong: root must not be used daily and must have MFA enabled.

B. Disable all IAM users and use just root with a strong password
Wrong: increases risk and breaks least privilege and accountability.

C. Enable MFA for root, create individual IAM users, use groups/roles, remove unused keys
Correct: fixes MFA, removes shared accounts, applies least privilege, and cleans old credentials.

D. Use a single IAM user for the whole team with AdministratorAccess
Wrong: still shared identity, over-privileged, no traceability.

Your company wants centralized permission management where developers are granted access based on their job function (e.g. “Developers”, “Admins”) instead of per-user policies. What is the best way to implement this?

A. Attach inline policies directly to each IAM user
Hard to manage and scale; no central role-based structure.

B. Share the root user credentials with all admins
Critical security violation; root should never be shared or used daily.

C. Use only resource-based policies on all services
Not practical as a sole method; doesn’t cleanly model job-function access for all identities.

D. Use IAM groups with managed policies attached to the groups
Correct: groups represent roles; assigning users to groups centralizes and simplifies permission management.

# Amazon EC2

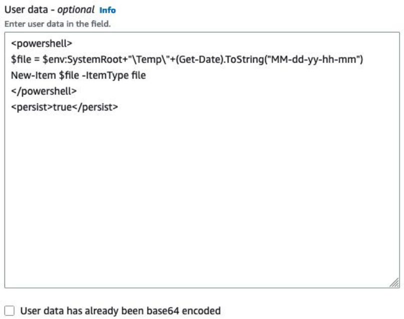
Amazon EC2 (Elastic Compute Cloud) is a core AWS service that provides resizable virtual servers in the cloud, so you can run applications without managing physical hardware.

- Launch instances in different sizes, CPU/RAM combos, and OS types (Linux/Windows).
- Integrates with EBS/EFS storage, VPC networking, Security Groups, and IAM Roles.
- Supports many use cases: web apps, APIs, backend services, databases (self-managed), batch processing, CI/CD agents, and more.
- Can scale up or down quickly to handle changing traffic.
- Pricing based on instance type, size, region, and usage model (On-Demand, Reserved, Spot, etc.).

# EC2 User Data

EC2 User Data lets you automate instance bootstrap by running scripts at launch so instances come up pre-configured and consistent.

- Runs automatically on first boot by default (can be scripted to run on every boot).
- Used to install software, apply updates, pull code, configure services without manual steps.
- On Linux: usually a bash script starting with #! /bin/bash, running as root.
- On Windows: can run PowerShell or batch commands.
- Commonly defined in Launch Templates / Auto Scaling Groups for repeatable infrastructure.
- Reduces configuration drift and supports immutable infrastructure patterns.
- SAA exam / use case: Choose EC2 User Data when the question talks about auto-configuring instances at launch (e.g. install web server on start) without manual SSH.



User data - *optional* Info

Enter user data in the field.

```
<powershell>
$file = $env:SystemRoot+"\Temp\"+(Get-Date).ToString("MM-dd-yy-hh-mm")
New-Item $file -ItemType file
</powershell>
<persist>true</persist>
```

☐ User data has already been base64 encoded

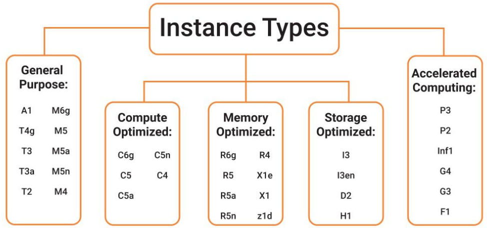
# EC2 Instance Types

EC2 instance types are families of virtual machines optimized for different workloads. Choosing the right type is about matching CPU, memory, storage, and network to your use case.

1. General Purpose (e.g. t, m) – Balanced CPU/RAM. Use for web servers, small DBs, dev/test.
2. Compute Optimized (e.g. c) – More CPU vs RAM. Use for high-performance compute, batch processing.
3. Memory Optimized (e.g. r, x) – More RAM vs CPU. Use for in-memory DBs, caching, big analytics.
4. Storage Optimized (e.g. i, d) – High IOPS/local SSD. Use for NoSQL, big data, heavy read/write.
5. Accelerated Computing (e.g. p, g, f) – GPUs / special hardware. Use for ML, GPU rendering, HPC.

SAA exam / use case:

- Pick the family that matches the workload: **web app** → General Purpose, **CPU-heavy** → Compute, **RAM-heavy** → Memory, **high I/O** → Storage, **ML/graphics** → GPU.z



Instance Types

General Purpose:
A1 M6g
T4g M5
T3 M5a
T3a M5n
T2 M4

Compute Optimized:
C6g C5n
C5 C4
C5a

Memory Optimized:
R6g R4
R5 X1e
R5a X1
R5n z1d

Storage Optimized:
I3
I3en
D2
H1

Accelerated Computing:
P3
P2
Inf1
G4
G3
F1

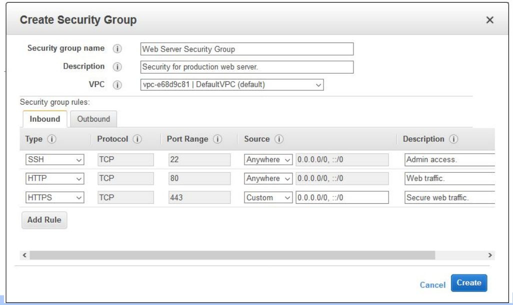
# Security Groups

Security Groups act as virtual firewalls for your EC2 instances, controlling inbound and outbound traffic at the instance (ENI) level.

- Work at the instance level, not subnet level.
- Stateful: if inbound is allowed, the response is automatically allowed back out.
- Use allow rules only (no explicit deny).
- Default: all inbound denied, all outbound allowed (can be changed).
- Rules can allow traffic by protocol, port, and source/destination (IP or another Security Group).
- An instance can have multiple Security Groups, and their rules combine.
- Common pattern: allow SSH/HTTP/HTTPS from specific IPs or load balancers only.

SAA exam / use case:

1. If they mention “instance-level firewall”, “stateful rules”, or “only allow, no deny” → Security Group.
2. Use Security Groups for controlling app-tier/web-tier access between instances.



Create Security Group

Security group name: Web Server Security Group

Description: Security for production web server.

VPC: vpc-e68d9c81 | DefaultVPC (default)

Security group rules:

| Type | Protocol | Port Range | Source | | Description |
|---|---|---|---|---|---|
| SSH | TCP | 22 | Anywhere | 0.0.0.0/0, ::/0 | Admin access. |
| HTTP | TCP | 80 | Anywhere | 0.0.0.0/0, ::/0 | Web traffic. |
| HTTPS | TCP | 443 | Custom | 0.0.0.0/0, ::/0 | Secure web traffic. |

# Classic Ports to Know

22 – SSH (secure remote login to Linux/Unix).

3389 – RDP (remote desktop for Windows).

80 – HTTP (unencrypted web traffic).

443 – HTTPS (encrypted web traffic).

21 – FTP (file transfer – insecure).

22 (SFTP over SSH) – secure file transfer.

25 – SMTP (sending email).

53 – DNS (name resolution).

1433 – Microsoft SQL Server.

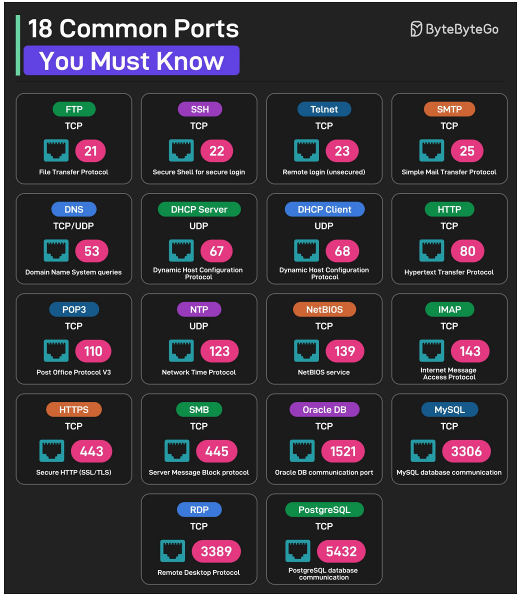
3306 – MySQL / MariaDB.

5432 – PostgreSQL.

SAA exam / use case:

Questions about which port to open in a Security Group for SSH, RDP, web apps, or DB access.

Recognize wrong ports/too-open rules (e.g. SSH/RDP open to 0.0.0.0/0).



18 Common Ports

You Must Know

ByteByteGo

| | | | |
|---|---|---|---|
| FTP | TCP | 21 | File Transfer Protocol |
| SSH | TCP | 22 | Secure Shell for secure login |
| Telnet | TCP | 23 | Remote login (unsecured) |
| SMTP | TCP | 25 | Simple Mail Transfer Protocol |
| DNS | TCP/UDP | 53 | Domain Name System queries |
| DHCP Server | UDP | 67 | Dynamic Host Configuration Protocol |
| DHCP Client | UDP | 68 | Dynamic Host Configuration Protocol |
| HTTP | TCP | 80 | Hypertext Transfer Protocol |
| POP3 | TCP | 110 | Post Office Protocol V3 |
| NTP | UDP | 123 | Network Time Protocol |
| NetBIOS | TCP | 139 | NetBIOS service |
| IMAP | TCP | 143 | Internet Message Access Protocol |
| HTTPS | TCP | 443 | Secure HTTP (SSL/TLS) |
| SMB | TCP | 445 | Server Message Block protocol |
| Oracle DB | TCP | 1521 | Oracle DB communication port |
| MySQL | TCP | 3306 | MySQL database communication |
| RDP | TCP | 3389 | Remote Desktop Protocol |
| PostgreSQL | TCP | 5432 | PostgreSQL database communication |

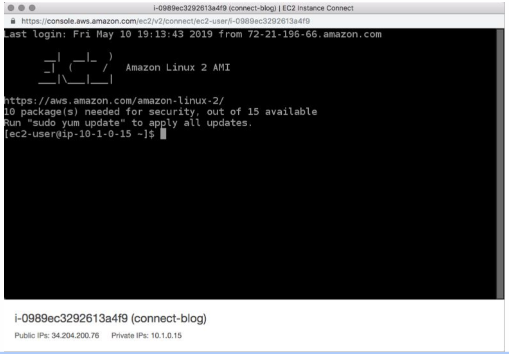
# EC2 Instance Connect

EC2 Instance Connect provides a secure, browser-based and CLI-based way to SSH into EC2 instances without managing permanent SSH keys.

- Uses temporary SSH keys generated and sent via the AWS environment at connection time.
- Works with supported Amazon Linux / Ubuntu instances that have the EC2 Instance Connect package installed.
- Security Group must allow port 22 from the EC2 Instance Connect IP ranges or your corporate IP.
- Reduces need to distribute or store long-term private keys across teams.
- Can be integrated with IAM so only authorized users can initiate connections.

SAA exam / use case:

- Choose EC2 Instance Connect when the question wants secure SSH access without sharing static keys, or centralized IAM-controlled access to instances.



i-0989ec3292613a4f9 (connect-blog) | EC2 Instance Connect

https://console.aws.amazon.com/ec2/v2/connect/ec2-user/i-0989ec3292613a4f9

```
Last login: Fri May 10 19:13:43 2019 from 72-21-196-66.amazon.com

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
10 package(s) needed for security, out of 15 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-1-0-15 ~]$ 
```

i-0989ec3292613a4f9 (connect-blog)

Public IPs: 34.204.200.76    Private IPs: 10.1.0.15

# EC2 Instances Purchasing Options (1)

On-Demand Instances

- Pay by the hour/second with no long-term commitment.
- Flexible, easy to start/stop.
- Best for short-term, spiky, or unknown workloads; dev/test or POCs.
- SAA use case: Choose On-Demand when usage is unpredictable or commitment is not possible.

Reserved Instances (Standard & Convertible)

- Commit to 1-year or 3-year usage for significant discount vs On-Demand.
- Standard RI: largest discount, less flexible (specific family/region).
- Convertible RI: lower discount, but can change instance family/OS/tenancy.
- Best for steady-state workloads (e.g., always-on production).
- SAA use case: Pick RIs when traffic is predictable and long-term.

Savings Plans (Compute & EC2)

- Commit to a $ amount per hour for 1 or 3 years instead of specific instance.
- More flexibility than RIs (can apply across instance families, regions, Fargate, Lambda – depending on type).
- Same idea: commitment → discount.
- SAA use case: When scenario mentions flexibility across services/regions but still long-term commitment.

# EC2 Instances Purchasing Options (2)

Spot Instances

Use unused EC2 capacity with up to ~90% discount vs On-Demand.
Can be interrupted by AWS with short notice when capacity is needed back.
Best for fault-tolerant, stateless, flexible workloads (batch jobs, containers, big data, CI).
SAA use case: Choose Spot when cost is key and interruptions are acceptable.

Dedicated Hosts

Physical servers dedicated to a single customer.
Full control over socket/core/host-level for licensing and compliance.
Good for BYOL licenses, strict compliance or hardware isolation.
SAA use case: When scenario mentions per-socket licensing, compliance, or tenant isolation.

Dedicated Instances

Run on hardware dedicated to one customer, but without host visibility.
Less control than Dedicated Hosts, still provides physical isolation.
SAA use case: When isolation from other customers is required, but no specific licensing needs.

Capacity Reservations / Zonal Reservations

Reserve capacity in a specific AZ to ensure you can launch when needed.
Can be combined with RIs or Savings Plans for discounts.
SAA use case: When guaranteed capacity in a certain AZ is required (e.g., critical prod).

# Spot Fleets

A Spot Fleet is a collection of instances launched using multiple Spot (and optionally On-Demand) instances to get the lowest cost while meeting capacity goals.

- You define target capacity, allowed instance types, and max price or strategy.
- Fleet automatically chooses cheapest capacity pools across types/AZs.
- Can mix Spot + On-Demand for more reliability.
- Automatically replaces interrupted Spot Instances to maintain capacity.
- Great for batch processing, big data, containers, CI/CD, image/video processing.

SAA exam / use case:

- Choose Spot Fleet when they want cost-optimized, fault-tolerant scaling using many Spot instances with automatic replacement.

# Elastic IPs

Elastic IP (EIP) is a static public IPv4 address that you can attach to an EC2 instance so it keeps the same public IP, even if you stop/start or replace the instance.

- Useful for fixed endpoints (DNS records, whitelisting, legacy systems).
- You can remap an Elastic IP from one instance to another in the same region.
- Charged when allocated but not attached or attached to a stopped instance.
- Should be used sparingly; prefer Load Balancers or DNS for scalability.

SAA exam / use case:

- Choose Elastic IP when a scenario needs a static public IP that can move between instances.

# Placement Groups

Placement Groups control how EC2 instances are placed on underlying hardware to optimize performance, availability, or both.

## Cluster Placement Group

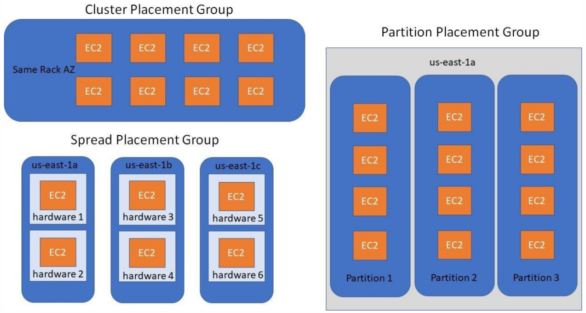
- Packs instances close together in a single AZ.
- Very low latency, high throughput (HPC, tightly-coupled apps).
- Lower fault tolerance (same rack/hardware).
- SAA use case: Choose Cluster for HPC / high-performance workloads needing fast network.

## Spread Placement Group

- Spreads instances across distinct hardware (up to 7 per AZ).
- Maximizes high availability; failures are isolated.
- Good for small number of critical instances.
- SAA use case: Choose Spread for critical instances that must not fail together.

## Partition Placement Group

- Divides instances into partitions, each on separate racks.
- Limits impact of hardware failure to a single partition.
- Designed for large distributed systems (Hadoop, HDFS, Cassandra).
- SAA use case: Choose Partition for big data / distributed workloads needing scale + failure isolation.



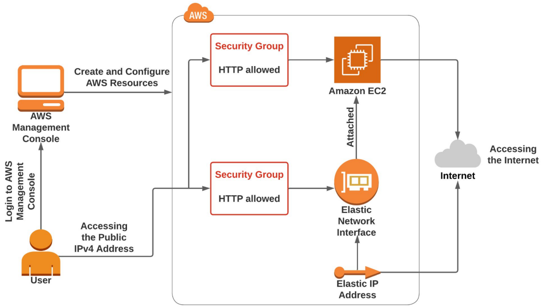
# Elastic Network Interfaces (ENI)

An **Elastic Network Interface (ENI)** is a virtual network card that you can attach to an EC2 instance in a VPC.

- Contains primary private IP, optional secondary private IPs.
- Can be associated with Elastic IPs and Security Groups.
- Can have one or more ENIs attached to the same instance (depending on type).
- You can detach and reattach ENIs between instances in the same AZ (for failover).
- Useful for separating traffic (management vs app), high availability, or multi-homed architectures.

SAA exam / use case:

- Choose ENI when scenario mentions multiple IPs, multiple Security Groups, or moving a network interface (and its IPs) between instances for fast recovery.



FlipTheScript

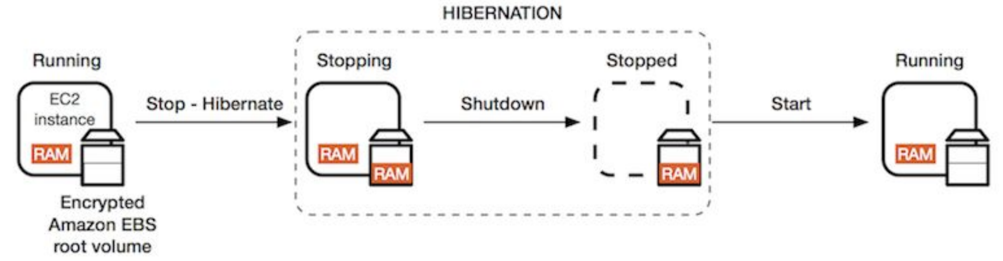
# EC2 Hibernate

EC2 Hibernate lets you pause and resume an instance by saving the contents of RAM to the root EBS volume when you stop it, so it restarts faster with the same in-memory state.

- When hibernating, RAM state is saved to the root EBS volume; on start, the instance restores memory and resumes where it left off.
- Keeps instance attributes: instance ID, private IPs, ENIs, and EBS volumes stay the same (like Stop).
- Best for long-running apps that take time to initialize (in-memory caches, agents, sessions).
- Requires EBS root volume, enough EBS space, and supported instance types/OS; mainly for On-Demand.
- Billing: pay for EBS storage, not for instance compute while hibernated.

SAA exam / use case:

- Choose Hibernate when you need fast restart with preserved RAM state, not just normal Stop/Start.



# EC2 Questions

You must run a CPU-intensive batch job for a few hours with unpredictable schedule, and interruptions are acceptable to reduce cost. Which option is most cost-effective?

A. On-Demand Instances in a single AZ

B. Reserved Instances for 3 years

C. Spot Instances for the batch job

D. Dedicated Hosts for better performance

An application server needs two network interfaces: one for public web traffic and one for internal admin traffic, each with different security groups and IP ranges. What should you use?

A. Two Security Groups on the same single ENI

B. Two Elastic IPs on the same single ENI

C. Multiple ENIs attached to the same EC2 instance

D. Two different key pairs on the instance

# EC2 Questions

You must run a CPU-intensive batch job for a few hours with unpredictable schedule, and interruptions are acceptable to reduce cost. Which option is most cost-effective?

A. On-Demand Instances in a single AZ
Works but more expensive; no use of interruption tolerance.

B. Reserved Instances for 3 years
Wrong match: long commitment for short, unpredictable jobs.

C. Spot Instances for the batch job
Correct: cheapest option when workload is interruption-tolerant and flexible.

D. Dedicated Hosts for better performance
Overkill and expensive; aimed at licensing/compliance, not this use case.

An application server needs two network interfaces: one for public web traffic and one for internal admin traffic, each with different security groups and IP ranges. What should you use?

A. Two Security Groups on the same single ENI
Doesn’t separate networks/IPs; both flows share the same interface.

B. Two Elastic IPs on the same single ENI
Still a single interface; no clean separation of internal vs external.

C. Multiple ENIs attached to the same EC2 instance
Correct: each ENI has its own IPs and security groups, ideal for traffic separation.

D. Two different key pairs on the instance
Only affects SSH authentication, not networking or segmentation.

# EBS (Elastic Block Store)

Amazon EBS provides persistent block storage volumes for EC2 instances, like virtual hard disks that keep data even when the instance stops.

- Volumes are AZ-specific and attached over the network to EC2 instances.
- Can be used as root volumes or data volumes.
- Data is persistent across reboots and Stop/Start of the instance.
- You can detach and reattach volumes between instances in the same AZ.
- Designed for low-latency, high-performance workloads (databases, applications, logs).
- Supports snapshots for backup, restore, and migration to new volumes.

SAA exam / use case:

- Choose EBS when you need durable block storage for a single instance in one AZ (e.g., DB, OS disk).

# EBS Snapshots

EBS Snapshots are point-in-time backups of EBS volumes stored in durable object storage, used for backup, restore, and migration.

- Capture the state of a volume at a specific moment.
- Incremental: after the first snapshot, only changed blocks are saved → faster and cheaper over time.
- Can be used to create new volumes in the same AZ or different AZ/Region (for migration or DR).
- Useful for backups before changes, restoring corrupted volumes, and cloning environments.
- Snapshots can be automated with policies (e.g., scheduled backups).

SAA exam / use case:

- Choose Snapshots for backup/restore, DR, or copying EBS data across AZs/Regions.

# AMI Overview

An Amazon Machine Image (AMI) is a pre-configured template used to launch EC2 instances with a specific OS, software, and configuration.

- Defines: OS, root volume type/size, installed packages, config files, and launch permissions.
- Ensures consistent, repeatable deployments across many instances (no manual setup each time).

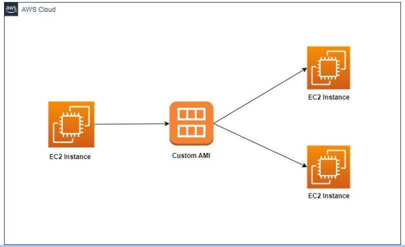
Types:

- AWS provided – base Linux/Windows images maintained by AWS.
- AWS Marketplace – vendor/partner images (security tools, DBs, appliances).
- Custom AMI – created from your configured instance (the “golden image”).

- Custom AMIs are ideal after hardening, installing agents, app dependencies, monitoring, and security baselines.
- AMIs can be shared with other accounts and copied across Regions, supporting multi-account and DR strategies.
- Often backed by EBS volumes, enabling fast launch and snapshot-based management.
-

SAA exam / use case:

- Choose AMI when you need to launch many identical instances quickly, enforce standardized images, or migrate a known-good setup between Regions/accounts.



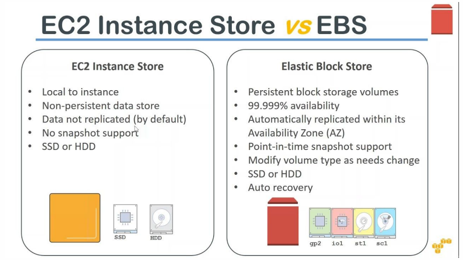
# EC2 Instance Store

EC2 Instance Store provides temporary block storage that is physically attached to the host running your instance, offering very high I/O performance but no durability if the instance is stopped or terminated.

- Storage is ephemeral: data is lost on Stop, Terminate, hardware failure, or instance migration.
- Included with specific instance types; must be chosen at launch (you cannot add Instance Store later).
- Ideal for temporary data: caches, buffers, scratch space, replicated data, short-lived processing.
- Not for important data: do not store critical logs, databases, or single-source data without replication.
- Unlike EBS, you cannot detach an Instance Store volume or independently snapshot it; backup must be done by copying data to EBS/S3.
- Very high throughput and low latency, good for performance-heavy workloads that can tolerate data loss.

SAA exam / use case:

- Choose Instance Store when the question mentions high-performance temporary storage that can be regenerated.
  If they need persistence across Stop/Start or easy backup → answer is EBS, not Instance Store.



## EC2 Instance Store *vs* EBS

**EC2 Instance Store**

- Local to instance
- Non-persistent data store
- Data not replicated (by default)
- No snapshot support
- SSD or HDD

**Elastic Block Store**

- Persistent block storage volumes
- 99.999% availability
- Automatically replicated within its Availability Zone (AZ)
- Point-in-time snapshot support
- Modify volume type as needs change
- SSD or HDD
- Auto recovery

# EBS Volume Types

General Purpose SSD – gp3 (default)
Balanced price/performance for most workloads.
Configure IOPS and throughput independently of size.
Ideal for boot volumes, app servers, small–medium DBs.
SAA: default choice unless there's a special need.

General Purpose SSD – gp2
Legacy type; performance scales with size.
Still seen in questions; similar use cases as gp3.

Provisioned IOPS SSD – io1 / io2
Designed for high IOPS, low latency, consistent performance.
You can provision high IOPS explicitly.
Ideal for large relational DBs, NoSQL, critical transactional systems.
Some support Multi-Attach for attaching one volume to multiple instances.
SAA: choose io1/io2 when DB needs thousands of consistent IOPS.

Throughput Optimized HDD – st1
HDD, optimized for high throughput, sequential reads/writes.
Good for big data, log processing, streaming workloads.
Lower cost, not for random small I/O, typically not for boot volumes.
SAA: choose st1 for large, streaming, frequently accessed data.

Cold HDD – sc1
Lowest-cost HDD for infrequently accessed data.
Lower performance; not for boot or production DBs.
SAA: choose sc1 for cheap, cold, large volumes accessed rarely.

Magnetic (Standard) – legacy, mostly for older questions; generally replaced by SSD types.

**gp3/gp2** → default / general workloads & boot. - **io1/io2** → critical DB, high IOPS.

**st1** → big, frequently scanned data. - **sc1** → cheapest cold data, rare access.

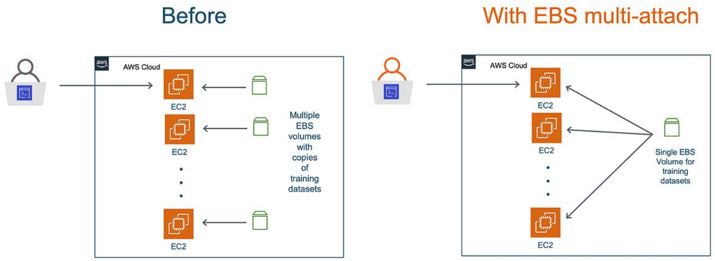
# EBS Multi-Attach

EBS Multi-Attach allows a single Provisioned IOPS SSD volume (io1/io2) to be attached to multiple EC2 instances in the same AZ at the same time.

- Supports up to multiple instances (same AZ) sharing one volume.
- Designed for clustered applications that are built to handle concurrent writes (e.g., clustered file systems, HA databases).
- Not safe for standard file systems (like ext4/NTFS) without a clustering layer → risk of corruption.
- All instances see the same data and can read/write simultaneously with proper coordination.
- Useful for high availability, failover, and shared storage between nodes.

SAA exam / use case:

- Choose EBS Multi-Attach when scenario mentions shared block storage for multiple instances in same AZ with an app that manages concurrency.



Before

With EBS multi-attach

AWS Cloud
EC2
EC2
EC2
Multiple EBS volumes with copies of training datasets
AWS Cloud
EC2
EC2
EC2
Single EBS Volume for training datasets

# EBS Encryption

EBS Encryption protects data at rest, in transit between instance and volume, and in snapshots using keys managed by AWS KMS.

- Encrypts: data on the volume, snapshots, and all volumes created from encrypted snapshots.
- Uses industry-standard AES-256 with keys from AWS KMS (AWS-managed or customer-managed).
- No action needed in the OS – encryption is handled transparently by EBS.
- No significant performance impact for most workloads.
- You can enable default EBS encryption at the account/Region level so all new volumes are encrypted automatically.
- Unencrypted → encrypted: To encrypt an existing unencrypted EBS volume, create a snapshot of it, copy that snapshot as encrypted, then create a new encrypted volume from that snapshot and attach it instead of the old one.

SAA exam / use case:

- Choose EBS Encryption when requirement mentions data at rest encryption, compliance, encrypted backups, or keys managed in KMS.

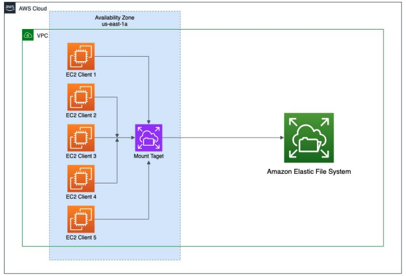
# Amazon EFS

Amazon EFS (Elastic File System) is a fully managed, scalable, shared file system for Linux instances and services.

- Provides a standard NFS (Network File System) that multiple EC2 instances can mount at the same time.
- Scales automatically up and down as files are added/removed (no capacity planning).
- Regional / multi-AZ: data stored redundantly across multiple AZs for high availability and durability.
- Ideal for shared content, home directories, microservices, web/app servers needing the same files, analytics, and container workloads.
- Can be mounted from EC2, ECS, EKS, and on-prem via Direct Connect/VPN (hybrid).
- Pay for storage used, supports performance & lifecycle tiers for cost optimization.

SAA exam / use case:

- Choose EFS for shared, POSIX-compliant, scalable storage across many Linux instances or AZs (vs EBS = single instance, single AZ).



# EFS – Performance & Storage Classes

Amazon EFS offers performance modes, throughput modes, and storage classes to balance cost and performance.

Performance Modes

- General Purpose – default; low latency; best for web, app servers, home dirs.
- Max I/O – higher aggregate throughput and IOPS, but more latency; for large, highly parallel workloads.

Throughput Modes

- Bursting – default; scales with data size; good for most workloads.
- Provisioned Throughput – set fixed throughput regardless of storage size; for workloads needing consistent, high throughput.

Storage Classes

1. EFS Standard – multi-AZ, high durability; for frequently accessed data.
2. EFS Standard-IA – multi-AZ, lower cost for infrequently accessed data (per-GB + per-access cost).
3. EFS One Zone – single AZ, cheaper; for less critical, frequently accessed data.
4. EFS One Zone-IA – single AZ, low-cost for infrequently accessed data.

- Lifecycle policies can automatically move cold files to IA classes to reduce cost.

SAA exam / use case:

- Choose correct combo: Standard + General Purpose + Bursting for typical shared storage; IA for cost savings on cold data; Max I/O/Provisioned for heavy, parallel or analytics workloads; One Zone when cost-sensitive and AZ-level durability is acceptable.

# EBS vs EFS

## Amazon EBS

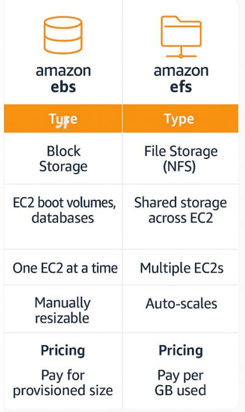
- Block storage attached to a single EC2 instance (per volume) in one AZ.
- Great for OS disks, single-instance databases, application data.
- Persistent, low-latency, supports different performance types (gp3, io2, etc.).
- Can detach/attach between instances in the same AZ; uses snapshots for backup/migration.
- SAA use case: choose EBS for durable disk for one server/DB in a specific AZ.

## Amazon EFS

- Managed shared file system (NFS) for multiple EC2 instances across multiple AZs.
- Auto-scales capacity; pay per GB used.
- Ideal for shared content, home directories, web/app farms, containers, analytics.
- Offers performance modes and Standard / IA / One Zone classes for cost/perf tuning.
- SAA use case: choose EFS for shared, scalable, POSIX-compliant storage across many Linux instances / AZs.

Key comparison (exam quick-pick)

1. Need single-instance disk, high IOPS, DB/OS → EBS.
2. Need multi-instance shared filesystem, auto-scale, multi-AZ → EFS.



| amazon ebs | amazon efs |
|---|---|
| Type | Type |
| Block Storage | File Storage (NFS) |
| EC2 boot volumes, databases | Shared storage across EC2 |
| One EC2 at a time | Multiple EC2s |
| Manually resizable | Auto-scales |
| **Pricing**<br>Pay for provisioned size | **Pricing**<br>Pay per GB used |

# Instance Storage Questions

Several Linux EC2 instances across multiple AZs must share the same files, with automatic growth and POSIX file system support. What should you choose?

A. Multiple EBS volumes attached to each instance

B. Amazon EFS file system mounted on all instances

C. One EBS volume with Multi-Attach in different Regions

D. Instance Store volumes replicated manually between instances

A single EC2 instance in one AZ runs MySQL and needs durable, low-latency block storage that persists across reboots. Which storage option is MOST appropriate?

A. Instance Store

B. Amazon EFS

C. Amazon EBS

D. S3 Standard

# Instance Storage Questions

Several Linux EC2 instances across multiple AZs must share the same files, with automatic growth and POSIX file system support. What should you choose?

A. Multiple EBS volumes attached to each instance
Not shared; EBS is per-AZ/per-instance without extra layers.

B. Amazon EFS file system mounted on all instances
Correct: shared NFS, multi-AZ, POSIX-compliant, auto-scaling.

C. One EBS volume with Multi-Attach in different Regions
Multi-Attach is same AZ only and for special clustered apps.

D. Instance Store volumes replicated manually between instances
Complex, fragile, ephemeral; not a managed shared solution.

A single EC2 instance in one AZ runs MySQL and needs durable, low-latency block storage that persists across reboots. Which storage option is MOST appropriate?

A. Instance Store
Ephemeral; data lost on stop/terminate; not suitable for DB durability.

B. Amazon EFS
Shared file system; not ideal as primary block storage for a single-instance DB.

C. Amazon EBS volume
Correct: persistent, low-latency block storage designed for this scenario.

D. S3 Standard
Object storage; cannot be used directly as a DB disk.