**Raja Muhammad Mustansar Javaid**

AWS Solutions Architect Associate

**Notes**



## **Ch # 1**

**Cloud Computing**



## **Cloud Computing**

* Cloud computing is like renting computer resources (like storage, processing power, or software) over the internet
* a shared "library" managed by someone else, instead of owning and maintaining your own computers.
* You access these resources on-demand, pay only for what you use, and can scale them up or down as needed, all without needing to buy physical hardware.

**On-Premises Computing**

* On-premises computing (often called "on-prem") is when a company owns, manages, and stores its own computer resources (servers, storage, software) in its own physical location, like an office or data center.
* Everything is handled internally, including maintenance, upgrades, and security.

### **Differences Between Cloud Computing and On-Premises Computing**

Here’s a clear comparison to highlight the differences:

* **Location of Resources**:
  + **Cloud Computing**: Resources (servers, storage) are hosted in remote data centers managed by a third-party provider (e.g., AWS, Microsoft Azure). You access them over the internet.
  + **On-Premises**: Resources are physically located in your own facility (e.g., your office or data center).
* **Ownership and Maintenance**:
  + **Cloud Computing**: The provider owns and maintains the hardware, software, and infrastructure. You just use the services.
  + **On-Premises**: You own and are responsible for maintaining the hardware, software, updates, and repairs.
* **Cost Structure**:
  + **Cloud Computing**: Pay-as-you-go model—you only pay for the resources you use (like renting books). No upfront costs for hardware.
  + **On-Premises**: High upfront costs to buy servers, software, and infrastructure, plus ongoing costs for maintenance, power, and staff.
* **Scalability**:
  + **Cloud Computing**: Easily scale resources up or down instantly based on demand (like borrowing more books when needed).
  + **On-Premises**: Scaling is harder and slower—you need to buy and install new hardware, which takes time and money.
* **Access and Flexibility**:
  + **Cloud Computing**: Access resources from anywhere with an internet connection, using any device.
  + **On-Premises**: Access is usually limited to your physical location or through specific setups like VPNs.
* **Control and Security**:
  + **Cloud Computing**: The provider handles security, but you share control with them. You rely on their security measures.
  + **On-Premises**: You have full control over security and data, which is great for sensitive information but requires expertise to manage.
* **Maintenance and Upgrades**:
  + **Cloud Computing**: The provider handles updates, patches, and maintenance automatically.
  + **On-Premises**: Your team is responsible for all updates, maintenance, and troubleshooting.

### **Why Use Both (Hybrid Approach)?**

Sometimes, businesses use a **hybrid approach**, combining cloud computing and on-premises computing, to get the best of both worlds. Here’s why:

* **Flexibility and Cost Efficiency**:
  + Use the **cloud** for tasks that need quick scaling, like handling sudden website traffic spikes (e.g., during a sale for an online store).
  + Use **on-premises** for stable, predictable workloads or sensitive data that must stay in-house (e.g., financial records).
* **Data Security and Compliance**:
  + Some industries (like healthcare or finance) have strict rules about where data can be stored. **On-premises** gives full control for sensitive data, while the **cloud** can handle less-sensitive tasks like customer-facing apps.
* **Cost Optimization**:
  + **Cloud** saves money on upfront hardware costs and is ideal for temporary or variable needs.
  + **On-premises** can be cheaper in the long run for stable, high-demand workloads, as you avoid ongoing cloud subscription fees.
* **Transitioning to the Cloud**:
  + Businesses with existing on-premises systems may not want to move everything to the cloud at once. A hybrid approach lets them use both while gradually transitioning to the cloud.
* **Reliability and Redundancy**:
  + Using both ensures that if one system (cloud or on-prem) fails, the other can keep operations running. For example, critical systems can stay on-premises, while the cloud provides backup or extra capacity.

**Example**: A hospital might keep patient records on-premises for security but use the cloud for its public website or staff scheduling app. This balances control, cost, and flexibility.

### **Summary**

* **Cloud Computing**: Rent resources over the internet, managed by a provider. It’s flexible, scalable, and cost-effective but relies on the provider for security and control.
* **On-Premises Computing**: Own and manage your own hardware on-site. It offers control and security but requires high upfront costs and maintenance.
* **Why Use Both**: A hybrid approach combines the scalability and affordability of the cloud with the control and security of on-premises, tailored to a business’s specific needs.

**Cloud Infrastructure**

Globally distributed, interconnected sets of independent data centers providing on-demand IT resources over the internet with pay-as-you-go pricing.

**Key Characteristics**:

* **Globally Distributed**:
  + Over 100 data centers (e.g., AWS, Azure) operational worldwide.
  + Strategically located to serve various regions.
* **Inter-Connected**:
  + Data centers linked via high-speed, redundant network connections forming a network backbone.
  + Ensures low-latency and reliable data transfer across regions.
* **Sets Of**:
  + Data centers operate in pairs or sets for redundancy and backup.
  + Minimum of 3 data centers per set in some regions (e.g., Microsoft standard).
  + Busy regions may have up to 6 data centers for enhanced capacity and resilience.
* **Independent Data Centers**:
  + Each data center operates autonomously with dedicated resources.
  + Equipped with its own power supply, cooling systems, networking, and security measures.
  + No sharing of power, networking, or cooling infrastructure between data centers.

**Core Features**:

* **On-Demand Delivery**:
  + Cloud providers offer resources (compute, storage, etc.) instantly as needed.
  + Eliminates the need for upfront infrastructure investment.
* **IT Resources Over the Internet**:
  + Includes servers, networking, storage, development tools, and applications.
  + Accessible via the internet, enabling scalable and flexible usage.
* **Pay-As-You-Go Pricing**:
  + Pay only for the resources used, with no upfront costs or long-term commitments.
  + Enables cost efficiency and scalability based on demand.

### **Key Features of Cloud Computing**

* **On-Demand Self-Service**: You can walk into the library (cloud) and pick a book (resource like storage or software) whenever you want, without needing to ask a librarian (human interaction). You use a simple interface, like a website or app, to get what you need instantly.
* **Broad Network Access**: You can visit the library from anywhere—home, school, or a café—using your phone, laptop, or tablet. The cloud works the same way: you access its services over the internet from any device, anywhere, using standard methods.
* **Resource Pooling**: The library has a huge collection of books shared by many people, which makes it cheaper than everyone buying their own. In the cloud, resources like storage and computing power are shared among users, lowering costs for everyone.
* **Rapid Elasticity**: Imagine the library magically adds more books when it’s busy and removes extras when it’s quiet. The cloud automatically adjusts resources (like adding more computing power) based on how much you need at any time.
* **Measured Service**: You only pay for the books you borrow from the library, not the entire collection. In the cloud, you’re charged only for the resources you use, like storage or processing, and usage is tracked clearly.

### **Cloud Deployment Models**

* A framework that defines how cloud computing resources (e.g., servers, storage, applications) are provisioned, managed, accessed, and integrated.
* Determines factors like ownership, security, scalability, cost, and data control.
* Based on the NIST (National Institute of Standards and Technology) model, which categorizes deployment based on who owns and manages the infrastructure.

**Types of Cloud Deployment Models**:

* **Public Cloud**:
  + **Definition**: A cloud environment where resources are owned and operated by a third-party provider and shared among multiple users (tenants) over the internet. Examples: AWS, Microsoft Azure, Google Cloud.
  + **Key Concepts**:
    - Multi-tenancy: Resources like servers and storage are shared among users, but logically isolated for security.
    - On-demand access: Users provision resources via self-service portals without human intervention.
    - Pay-as-you-go pricing: Charged based on usage (e.g., per hour, per GB), with no upfront capital expenditure.
    - Global scalability: Providers offer vast, distributed data centers for elasticity and redundancy.
    - Standardization: Uses standard APIs and protocols for broad compatibility.
  + **Why Use It**: Cost-effective for variable workloads; reduces IT management burden as the provider handles maintenance, updates, and security of the infrastructure.
  + **When to Use It**: Ideal for startups, small businesses, or applications with unpredictable demand (e.g., web apps, testing environments). Use when data sensitivity is low and quick scalability is needed without owning hardware.
  + **Pros**: Low cost, high scalability, rapid deployment, automatic updates.
  + **Cons**: Less control over data security; potential for "noisy neighbor" issues in shared environments; compliance challenges for regulated industries.
  + **Analogy**: Like a big public library—shared, accessible to all, managed by others.
* **Private Cloud**:
  + **Definition**: A dedicated cloud environment built and used exclusively by a single organization, either on-premises or hosted by a third-party. It incorporates cloud features like virtualization and automation but remains isolated.
  + **Key Concepts**:
    - Single-tenancy: Resources are not shared with external users, ensuring full isolation.
    - Customization: Tailored to specific needs, with control over hardware, software, and configurations.
    - Self-service and automation: Users within the organization can provision resources via portals, with elasticity for scaling.
    - On-premises or hosted: Can be managed internally or by a provider (e.g., VMware, OpenStack-based setups).
    - Compliance focus: Easier to meet strict regulatory requirements (e.g., GDPR, HIPAA) due to data locality and control.
  + **Why Use It**: Provides enhanced security and privacy for sensitive data; allows legacy system integration while adopting cloud benefits.
  + **When to Use It**: Suitable for large enterprises, government agencies, or industries with high security needs (e.g., finance, healthcare). Use when data must stay within organizational boundaries or when custom hardware is required.
  + **Pros**: High security, full customization, better performance control, predictable costs for stable workloads.
  + **Cons**: Higher upfront costs for setup and maintenance; less elastic than public clouds; requires in-house expertise.
  + **Analogy**: Like your own personal library at home—exclusive, controlled by you.
* **Multi-Cloud**:
  + **Definition**: A strategy involving the use of multiple public cloud providers (e.g., AWS + Google Cloud + Azure) in a single architecture or deployment, often for different services or workloads.
  + **Key Concepts**:
    - Vendor diversity: Avoids dependency on one provider by distributing workloads (e.g., storage on AWS S3, AI on Google Cloud).
    - Interoperability: Requires tools like containers (Kubernetes) or APIs for seamless integration across providers.
    - Redundancy and resilience: Enhances fault tolerance by spreading risks across vendors.
    - Best-of-breed approach: Selects optimal services from each (e.g., Azure for Windows integration, AWS for e-commerce).
    - Management complexity: Involves multi-cloud management platforms (e.g., Terraform, Ansible) for orchestration.
  + **Why Use It**: Mitigates vendor lock-in; leverages competitive pricing and specialized features; improves bargaining power with providers.
  + **When to Use It**: For organizations with diverse needs (e.g., global operations requiring region-specific services) or to avoid downtime from a single provider's outage. Common in mature cloud adopters for optimization.
  + **Pros**: Flexibility, cost optimization through competition, enhanced reliability.
  + **Cons**: Increased complexity in management, security, and data transfer; potential for higher costs due to integration tools.
  + **Analogy**: Like borrowing books from two different public libraries—mix and match for the best options.
* **Hybrid Cloud**:
  + **Definition**: An integrated environment combining private cloud (or on-premises infrastructure) with one or more public clouds, allowing data and applications to move seamlessly between them.
  + **Key Concepts**:
    - Integration: Uses APIs, VPNs, or direct connections (e.g., AWS Direct Connect) for unified management.
    - Bursting: Scale from private to public cloud during peak loads (cloud bursting).
    - Data tiering: Sensitive data stays private; non-sensitive or scalable workloads go public.
    - Orchestration: Tools like Azure Arc or AWS Outposts enable consistent operations across environments.
    - Not just legacy + public: True hybrid requires cloud-native features in both, not merely connecting old hardware to public cloud.
  + **Why Use It**: Combines the security of private cloud with the scalability and cost-efficiency of public cloud; facilitates gradual cloud migration.
  + **When to Use It**: For businesses transitioning from on-premises to cloud, or those with mixed workloads (e.g., core banking in private, customer apps in public). Use in regulated industries needing compliance while leveraging public resources.
  + **Pros**: Balanced security and scalability, cost savings for variable demands, supports legacy integration.
  + **Cons**: Complex setup and management; potential security risks at integration points; higher operational overhead.
  + **Analogy**: Like combining your private home library with a public library—best of both worlds.

**General Key Concepts Across Models**:

* **Security and Compliance**: Public and multi-cloud rely on shared responsibility; private and hybrid offer more control.
* **Cost Management**: Public/multi favor OPEX (operational expenditure); private/hybrid involve CAPEX (capital expenditure).
* **Scalability and Elasticity**: All models support it, but public/multi excel in rapid global scaling.
* **Vendor Lock-In**: Multi-cloud reduces it; hybrid can bridge to avoid full dependency.
* **Data Sovereignty**: Private/hybrid ensure data stays in specific locations; public requires careful region selection.

**Why Choose a Specific Model**:

* Based on business needs: Cost (public), security (private), flexibility (multi/hybrid), or migration stage (hybrid).
* Evolving trends: As of 2025, multi-cloud is rising for resilience post-outages; hybrid for AI/ML workloads needing on-premises data.

**When to Use in Practice**:

* Startups: Public for quick start.
* Enterprises: Hybrid for phased adoption.
* Global apps: Multi-cloud for redundancy.
* Sensitive data: Private for isolation.

**Cloud Computing Service Models:**

* Cloud service models define how cloud computing resources are delivered over the internet.
* Each model (IaaS, PaaS, SaaS) determines the level of user control and responsibility over infrastructure, platform, and software.

**Infrastructure as a Service (IaaS)**:

* **Definition**: Provides core IT resources (virtual machines, storage, networking) for rent, with users managing software but not physical hardware.
* **How It Works**:
  + Rent virtual servers, storage, and networking resources.
  + Users install, configure, and manage operating systems, applications, and data.
  + Pay only for resources used (pay-as-you-go).
* **User Responsibilities**:
  + Operating systems.
  + Software updates and patches.
  + Application deployment.
  + Data backups and management.
* **Why Use IaaS?**:
  + **Scalable**: Easily add servers or capacity as needed.
  + **Cost-Effective**: No upfront hardware costs.
  + **Customizable**: Full control over software stack.
  + **Fast Deployment**: Quickly set up infrastructure for development or testing.
* **Example Scenario**:
  + A startup rents virtual servers (e.g., AWS EC2) to build a web app.
  + They install the OS, configure a database, and deploy the app.
  + As demand grows, they scale by renting additional servers without purchasing hardware.
* **Examples**: AWS EC2, Amazon S3, Azure Virtual Machines, Google Compute Engine.

**Platform as a Service (PaaS)**:

* **Definition**: Provides a ready-made platform for developing and running applications, with the provider managing servers, OS, and hardware.
* **How It Works**:
  + Users focus on writing and deploying application code.
  + Provider handles infrastructure, OS, server management, and scaling.
  + Built-in tools for coding, testing, and deployment.
  + Pay for resources used, typically on a subscription or usage basis.
* **User Responsibilities**:
  + Application code and configuration.
  + Data management within the platform.
* **Why Use PaaS?**:
  + **Faster Development**: Focus on coding, not infrastructure management.
  + **No Server Management**: Provider handles backend operations.
  + **Easy to Scale**: Automatically adjusts resources based on demand.
  + **Cost-Efficient**: Pay only for consumed resources, no hardware or OS maintenance costs.
* **Example Scenario**:
  + A developer builds a mobile app using a PaaS (e.g., AWS Elastic Beanstalk).
  + They use built-in tools for coding, testing, and deployment, store data in a cloud database, and ignore server setup or OS updates.
  + The app launches quickly and scales automatically with user growth.
* **Examples**: AWS Elastic Beanstalk, Google App Engine, Azure App Service, Heroku.

**Software as a Service (SaaS)**:

* **Definition**: Provides fully managed, ready-to-use software over the internet, with the provider handling all infrastructure, updates, and maintenance.
* **How It Works**:
  + Users access software via a web browser or client.
  + No installation, management, or updates required.
  + Provider manages servers, OS, and application.
  + Subscription-based pricing (monthly/yearly).
* **User Responsibilities**:
  + Using the software and managing user accounts/data within the app.
* **Why Use SaaS?**:
  + **Easy Access**: Available on any internet-connected device.
  + **No Setup Needed**: Sign up and start using immediately.
  + **Automatic Updates**: Always access the latest version.
  + **Budget-Friendly**: Predictable subscription costs.
* **Example Scenario**:
  + A company uses Gmail or Google Docs for email and document collaboration.
  + Employees access the tools via browser, share files, and communicate without installing or maintaining software.
  + Updates are applied automatically by the provider.
* **Examples**: Google Workspace, Microsoft 365, Salesforce, Dropbox.

**Shared Responsibility Model**:

* **IaaS**:
  + **Provider**: Manages physical hardware, virtualization, and network infrastructure.
  + **User**: Manages OS, applications, data, security configurations, and backups.
* **PaaS**:
  + **Provider**: Manages hardware, OS, server software, and platform tools.
  + **User**: Manages application code, data, and application-specific configurations.
* **SaaS**:
  + **Provider**: Manages all infrastructure, OS, and application (full stack).
  + **User**: Manages user accounts, data input, and usage within the application.
* **Key Concept**: Responsibility shifts from user (IaaS) to provider (SaaS), with PaaS in between.

**Cost and Ownership**:

* **IaaS**:
  + Pay for resources used (e.g., compute hours, storage GB).
  + Higher user control but requires more management overhead.
* **PaaS**:
  + Pay for platform usage (e.g., runtime, database usage).
  + Balances control and management; less overhead than IaaS.
* **SaaS**:
  + Subscription-based (monthly/yearly).
  + Minimal management; provider handles all infrastructure and updates.
* **Ownership**: No physical hardware ownership across all models; users rent resources, reducing capital expenditure.
* **Comparison Table**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **IaaS** | **PaaS** | **SaaS** |
| **Control Level** | High (OS, apps, data) | Medium (app code, data) | Low (user data, accounts) |
| **Management** | User manages OS, apps, updates | Provider manages OS, servers | Provider manages everything |
| **Scalability** | Manual or scripted scaling | Automatic scaling | Automatic scaling |
| **Cost Model** | Pay-as-you-go (resources) | Pay-as-you-go (platform usage) | Subscription (per user/month) |
| **Use Case** | Custom apps, testing environments | App development, deployment | End-user software (email, CRM) |
| **Examples** | AWS EC2, Azure VMs, S3 | AWS Elastic Beanstalk, Heroku | Google Workspace, Salesforce |

* **Use Case Summary**:
  + **IaaS**: Ideal for startups or enterprises needing full control over infrastructure (e.g., custom web app hosting).
  + **PaaS**: Best for developers focusing on rapid app development without server management (e.g., mobile app deployment).
  + **SaaS**: Suited for businesses or individuals needing ready-to-use software with minimal setup (e.g., email, collaboration tools).

## **Ch # 2**

**AWS Cloud Computing**



**AWS (Amazon Web Services)**

* AWS is a comprehensive, evolving cloud computing platform provided by Amazon.
* Offers a mix of **Infrastructure as a Service (IaaS)**, **Platform as a Service (PaaS)**, and **Software as a Service (SaaS)** solutions.
* Provides scalable, flexible, and cost-effective cloud services for businesses, developers, and enterprises.

**Key Offerings**:

* **IaaS**: Virtual machines (e.g., EC2), storage (e.g., S3), networking (e.g., VPC).
* **PaaS**: Development platforms (e.g., Elastic Beanstalk, AWS Lambda) for building and deploying applications.
* **SaaS**: Ready-to-use software (e.g., Amazon WorkSpaces, Amazon Chime) managed by AWS.

**AWS Cloud Use Cases**:

* **Enterprise IT, Backup & Storage, Big Data Analytics**:
  + Manage and store large-scale data for enterprises.
  + Analyze data for insights using tools like Amazon Redshift, Athena, or EMR.
  + Ensure secure backups with services like S3, Glacier, or AWS Backup.
* **Website Hosting, Mobile & Social Apps**:
  + Host websites with scalable infrastructure (e.g., S3, EC2, Route 53).
  + Support mobile and social apps with services like API Gateway, AppSync, and Cognito.
* **Gaming**:
  + Develop and host video games using AWS GameLift, Lumberyard, or EC2.
  + Scale game servers dynamically to handle player demand.

**Durability & Availability**:

* **Durability**:
  + Probability that data persists over time and can be retrieved from storage (e.g., S3, Glacier).
  + AWS S3 offers **99.999999999% (11 9’s)** durability, ensuring objects are not lost due to hardware failures.
* **Availability**:
  + Probability that data can be accessed immediately when requested.
  + Affected by temporary issues like network or system outages.
  + Example: S3 Standard offers **99.99% availability**, meaning brief downtimes may occur but data remains durable.
* **Key Difference**:
  + Durability ensures data is not lost; availability ensures data is accessible at any given moment.
  + Example: An object in S3 Glacier Deep Archive is highly durable but may not be immediately available (retrieval takes hours).

**Key AWS Services by Use Case**:

* **Enterprise IT & Backup**:
  + **S3**: Object storage for backups and archives.
  + **EBS**: Block storage for persistent data.
  + **AWS Backup**: Centralized backup management.
* **Big Data Analytics**:
  + **Redshift**: Data warehousing for large-scale analytics.
  + **Athena**: Query data in S3 using SQL.
  + **EMR**: Managed Hadoop for big data processing.
* **Website Hosting**:
  + **S3**: Static website hosting.
  + **Route 53**: DNS and domain management.
  + **CloudFront**: CDN for fast content delivery.
* **Mobile & Social Apps**:
  + **API Gateway**: Create and manage APIs.
  + **Cognito**: User authentication and authorization.
  + **AppSync**: GraphQL APIs for mobile apps.
* **Gaming**:
  + **GameLift**: Managed game server hosting.
  + **Lumberyard**: Game development engine.
  + **EC2**: Scalable compute for game backends.

**Benefits of AWS**:

* **Scalability**: Easily scale resources up or down based on demand.
* **Flexibility**: Supports diverse workloads, from startups to enterprises.
* **Cost-Effective**: Pay-as-you-go pricing; no upfront hardware costs.
* **Global Reach**: Data centers in multiple regions for low latency and redundancy.
* **Security**: Robust security features (e.g., IAM, encryption, VPC).

**Best Practices**:

* Use **IAM** to enforce least privilege access for users and resources.
* Leverage **availability zones** and **regions** for high availability and durability.
* Enable **MFA** for enhanced account security.
* Use **CloudWatch** for monitoring and **CloudTrail** for auditing.
* Optimize costs with tools like **AWS Cost Explorer** and **Trusted Advisor**.

**Validate Your Knowledge**:

* **Question**: A company hosts a web application on AWS S3 and uses CloudFront for content delivery. They want to ensure data is not lost and is accessible with minimal downtime. Which AWS feature ensures high durability, and which ensures high availability?
  + **Answer**:
    - **Durability**: S3’s 99.999999999% (11 9’s) durability ensures data is not lost.
    - **Availability**: CloudFront and S3 Standard’s 99.99% availability ensure minimal downtime for access.
  + **Explanation**: S3’s durability protects against data loss, while CloudFront and S3’s high availability ensure the application is accessible with minimal interruptions.

## Scalability vs. High Availability

While both terms relate to a system's ability to handle challenges and remain functional, they address different problems:

### 1. Scalability: Handling Growth (More Load) 📈

Scalability is a measure of a system's ability to **increase its capacity** to accommodate a growing demand, such as more users, more data, or more transactions.

|  |  |  |
| --- | --- | --- |
| Characteristic | Vertical Scalability (Scaling Up) | Horizontal Scalability (Scaling Out) |
| **Method** | Increasing the power of **a single resource**. | Increasing the number of **resources** (servers/VMs). |
| **Action** | Adding more CPU, RAM, or a bigger disk to an existing server. | Adding more identical servers or instances to a cluster (elasticity). |
| **Focus** | Performance enhancement for a single node. | Distribution of load across multiple nodes. |
| **Limitation** | You will eventually hit the hardware or software limits of the single machine. | More complex to manage (requires load balancing and distributed systems). |

### 2. High Availability (HA): Handling Failure (Less Downtime) ⚙️

High Availability is a measure of how resilient a system is to **failures** and how consistently it can remain operational. It's about minimizing service downtime, whether planned or unplanned.

|  |  |
| --- | --- |
| Characteristic | High Availability (HA) |
| **Goal** | To ensure the service remains accessible even if a component fails, aiming for ("five nines") uptime. |
| **Method** | **Redundancy** (having duplicate components) and **Failover** (automatic switching to the backup component). |
| **Examples** | Running two identical database servers in a **Primary/Replica** configuration, or having redundant power supplies (PSUs) in a single server. |
| **Measurement** | **Uptime** (e.g., minutes of downtime per year). |

## Why They Are Linked But Different

The link between the two concepts comes from horizontal scaling, but they are fundamentally different:

|  |  |  |
| --- | --- | --- |
| Scenario | Scalability (Can it handle more?) | High Availability (Will it fail?) |
| **A Single, Powerful Server** | Highly scalable (vertically) up to its limit. | **Low HA.** If the server fails, the entire application goes down (SPOF). |
| **Two Servers in a Cluster with Load Balancer** | Highly scalable (horizontally) by adding more servers. | **High HA.** If one server fails, the load balancer directs traffic to the other, maintaining service. |

**In short:**

* **Scalability** prevents your system from being **slow** under heavy load.
* **High Availability** prevents your system from being **down** due to a failure.

A system can be highly available but not scalable (e.g., two small servers that can handle a failure but can't handle peak holiday traffic). Conversely, a system can be highly scalable (a huge server) but not highly available if it lacks redundancy.

The core difference between **scalability** and **elasticity** lies in the **automation** and **direction** of resource changes in response to workload.1

* **Scalability** is the **ability** of a system to handle a growing workload by adding resources (up or out).2 It is a measure of a system's *potential*.
* **Elasticity** is the **dynamic, automatic** ability to scale **both up and down** quickly to precisely match the current workload, thereby optimizing cost.3

|  |  |  |
| --- | --- | --- |
| Feature | Scalability | Elasticity |
| **Definition** | The ability to handle increasing load by provisioning **more resources**. | The ability to **automatically and rapidly** grow **(scale out)** or shrink **(scale in)** resources. |
| **Mechanism** | Generally **manual** and **planned** (e.g., *we'll add 5 servers next month*). | **Automatic** and **dynamic** (e.g., *CPU load is for 5 minutes, add 2 servers*). |
| **Direction** | Primarily focuses on **scaling up/out** (handling growth). | Focuses on scaling **out AND in** (handling peaks and troughs). |
| **Goal/Focus** | Handling **long-term, predictable** growth and maintaining performance. | **Cost optimization** and matching **short-term, unpredictable** demand spikes. |
| **Analogy** | Buying a **bigger house** to prepare for a growing family. | **Renting a hall** for a party: adding chairs *only* when guests arrive and removing them *immediately* when they leave. |

## 1. Scalability (The Potential for Growth)

Scalability is an architectural property that describes whether a system can be made larger to handle more work.4 It is an overall design goal.

There are two types of scalability:5

1. **Vertical Scaling (Scale Up):** Increasing the capacity of a single resource (e.g., upgrading a single server from 68 GB to 732 GB of RAM).8
2. **Horizontal Scaling (Scale Out):** Adding more identical resources to share the load (e.g., adding 5 more web servers behind a load balancer). Horizontal scaling is the foundation for elasticity.

## 2. Elasticity (The Dynamic Response)

Elasticity is a core concept of **cloud computing** that relies on horizontal scalability.9 It is the implementation of a scaling strategy that is fully automated to prevent both:

* **Under-provisioning:** Not having enough resources to handle a peak load, leading to slowdowns or crashes.10
* **Over-provisioning:** Having too many idle resources during low periods, leading to wasted money.11

Elasticity is what allows an e-commerce site to automatically spin up 50 extra web servers for a Black Friday rush and then automatically shut them down hours later, paying only for the few hours they were needed.12

Vertical scalability and horizontal scalability are the two fundamental ways to increase a system's capacity, differing in **how** they add resources.1

|  |  |  |
| --- | --- | --- |
| Feature | Vertical Scalability (Scaling Up) ⬆️ | Horizontal Scalability (Scaling Out) ➡️ |
| **Action** | Adding more resources (CPU, RAM, storage) to a **single existing machine** (server, database instance). | Adding **more machines** (servers, nodes, instances) to a pool of resources. |
| **Limit** | **Hard physical limit** based on the maximum capacity of the single machine's hardware. | **Theoretically limitless**; only constrained by budget and network architecture. |
| **Fault Tolerance** | **Low.** The single server is a **Single Point of Failure (SPOF)**; if it fails, the system goes down. | **High.** Workload is distributed; if one server fails, the others continue running (redundancy). |
| **Cost** | **Initially lower** and simpler. **Exponentially more expensive** at high-end tiers. | **Higher upfront complexity/cost** for architecture setup (load balancers, clustering). **More cost-effective** long-term with commodity hardware. |
| **Downtime** | **Often required** (reboot) to install hardware upgrades or resize a Virtual Machine (VM). | **Generally zero**; new nodes are added while the existing ones continue to serve traffic. |
| **Complexity** | **Low.** Simple to manage and maintain one powerful system. | **High.** Requires complex distributed systems architecture, load balancing, and data synchronization. |
| **Use Case** | **Monolithic applications**, proprietary databases (like a single primary SQL server), or systems with **predictable, moderate growth**. | **Modern web applications, microservices, cloud-native apps**, and systems with **rapid, unpredictable, or massive scale** (elasticity). |

## Key Differences and Examples

### 1. The Single Point of Failure (SPOF)

Vertical scaling concentrates all work and data on one machine, which is a major risk. Horizontal scaling inherently provides **fault tolerance** and **High Availability** because the failure of a single node in the cluster does not take the entire service down.

### 2. The Scaling Ceiling

Vertical scaling hits a physical wall. For example, a server can only hold a finite amount of RAM (512 GB or 1 TB). Once you purchase the largest available machine, you can no longer scale vertically. Horizontal scaling, by contrast, lets you keep adding new, cheaper commodity servers indefinitely.

### 3. Application Design

* **Vertical Scaling** is easier for applications that are **stateful** (where a user's session data is tied to a specific server) or **tightly coupled** (monolithic), as the code is simpler to write for a single machine.
* **Horizontal Scaling** requires applications to be **stateless** (session data stored externally) and **distributed** so that any incoming request can be served by any machine in the pool. This is the foundational design principle for cloud computing and microservices.

A good example comparing the two:

* **Vertical Scaling Example:** You have a single **SQL Database server**.2 To scale, you upgrade its CPU, add more RAM, and install a faster SSD drive.3
* **Horizontal Scaling Example:** You have a **NoSQL Database** (like Cassandra or MongoDB). To scale, you add 5 new database nodes, and the data is automatically **partitioned (sharded)** and distributed across all 6 nodes, increasing both capacity and redundancy.

This video provides an excellent summary of the differences you should know when designing systems: Vertical Vs Horizontal Scaling: Key Differences You Should Know

# AWS Client-Server Model

## Definition

* **Client-Server Model**: A computing architecture where a client (e.g., web browser, desktop application) sends requests to a server, which provides services or resources in response.

## Client

* **Role**: Typically a web browser (e.g., Chrome, Firefox) or desktop application used by an individual.
* **Function**: Sends requests to servers to access data, applications, or services.

## Server

* **Role**: Provides services or resources in response to client requests.
* **AWS Example**: Amazon Elastic Compute Cloud (EC2), a virtual server in AWS that hosts applications, processes client requests, or stores data.

## Key Characteristics

* **Request-Response**: Clients initiate requests, and servers respond with the requested data or actions.
* **Scalability**: AWS services like EC2 allow servers to scale dynamically to handle varying client loads.
* **Use Case**: A web browser (client) accessing a web application hosted on an EC2 instance (server) to retrieve a webpage.

## [AWS Design Principles](https://tutorialsdojo.com/aws-well-architected-framework-design-principles/)

**Scalability**:

* **Horizontal Scaling**: Adding more resources (e.g., more servers) to handle increased demand.
* **Vertical Scaling**: Upgrading an existing resource (e.g., increasing a server’s CPU or memory).
* **Horizontal vs. Vertical Scaling**: Horizontal adds more units; vertical enhances a single unit’s capacity.

**Disposable Resources Instead of Fixed Servers**:

* **Instantiating Compute Resources**: Automatically set up new resources with their configuration and code.
* **Infrastructure as Code**: Treat AWS resources like software—programmable, reusable, maintainable, and testable using software development techniques.

**Automation**:

* **Serverless Management and Deployment**: Focus on code; AWS manages servers, automating deployment tasks.
* **Infrastructure Management and Deployment**: AWS automates resource provisioning, load balancing, auto-scaling, and monitoring.
* **Alarms and Events**: AWS monitors resources and triggers actions when specific conditions or metrics are met.

**Loose Coupling**:

* **Well-Defined Interfaces**: Components interact only through specific, technology-neutral interfaces (e.g., RESTful APIs) to reduce dependencies.
* **Service Discovery**: Small services work without needing to know network details, allowing infrastructure changes without disruption.
* **Asynchronous Integration**: Components communicate via a storage layer (e.g., queues) for requests that don’t need immediate responses.

**Distributed Systems Best Practices**:

* Build applications to handle component failures gracefully, ensuring system reliability.

**Services, Not Servers**:

* **Managed Services**: Use AWS-provided building blocks like databases, analytics, or notifications to power applications.
* **Serverless Architectures**: Build event-driven or synchronous apps without managing servers, reducing operational complexity.

**Databases**:

* **Relational Databases**: Offer powerful queries, flexible indexing, and data integrity for structured data.
* **NoSQL Databases**: Trade some query capabilities for flexible, horizontally scalable models (e.g., key-value, graphs, JSON).
* **Data Warehouses**: Specialized relational databases optimized for analyzing large datasets.
* **Graph Databases**: Use graph structures for querying relationships.
* **Search Functionalities**: Enable querying unstructured or structured data with features like ranking, filtering, and synonyms, distinct from formal database queries.

**Managing Increasing Volumes of Data**:

* **Data Lake**: Centralized storage for massive data, ready for processing, analysis, and use by various teams.

**Removing Single Points of Failure**:

* **Introducing Redundancy**:
  + **Standby Redundancy**: Secondary resource takes over after a failure, with brief downtime (used for stateful systems like databases).
  + **Active Redundancy**: Multiple resources handle requests; if one fails, others absorb the load.
* **Detect Failure**: Use health checks and logs to identify issues.
* **Durable Data Storage**:
  + **Synchronous Replication**: Ensures data is stored in primary and replicas before confirming a transaction, protecting data integrity.
  + **Asynchronous Replication**: Primary node updates independently, with replicas lagging slightly.
  + **Quorum-Based Replication**: Combines both, requiring a minimum number of nodes for a successful write.
* **Automated Multi-Data Center Resilience**: Use AWS Regions and Availability Zones for fault tolerance.
* **Fault Isolation and Shuffle Sharding**: Isolate failures and distribute workloads to prevent system-wide issues.

**Optimize for Cost**:

* **Right Sizing**: Choose the appropriate AWS resource types and configurations for your needs.
* **Elasticity**: Save costs by scaling resources up or down based on demand.
* **Purchasing Options**: Use Reserved Instances for predictable workloads or Spot Instances for cost savings on flexible tasks.

**Caching**:

* **Application Data Caching**: Store data in fast, in-memory caches for quick access.
* **Edge Caching**: Deliver content from locations closer to users for lower latency and faster data transfer.

**Security**:

* **Defense in Depth**: Secure multiple layers (network, application, database) using AWS features.
* **Shared Responsibility**: AWS secures the cloud infrastructure; customers secure their data and apps in the cloud.
* **Reduce Privileged Access**: Apply the Principle of Least Privilege to limit access.
* **Security as Code**: Define security settings (e.g., firewall rules, subnets) in templates for consistency.
* **Real-Time Auditing**: Continuously monitor and automate security controls to minimize risks.

**Cloud Architecture Best Practices**:

* **Decouple Components**: Build loosely coupled systems where components operate independently, improving scalability and stability.
* **Think Parallel**: Use parallel processing and automation for efficiency.
* **Implement Elasticity**: Automate scaling to handle demand without manual intervention.
* **Design for Failure**: Assume components will fail and build for high availability and fault tolerance.

**For more Reading:**

<https://d1.awsstatic.com/whitepapers/AWS_Cloud_Best_Practices.pdf>

## [AWS Disaster Recovery](https://tutorialsdojo.com/aws-well-architected-framework-disaster-recovery/)

**Recovery Time Objective (RTO)**:

* The maximum acceptable time to restore a business process to its normal service level after a disruption.

**Recovery Point Objective (RPO)**:

* The maximum acceptable amount of data loss, measured as the time between the last backup and the disaster occurrence.

**Disaster Recovery with AWS Overview**:

* AWS provides strategies to minimize downtime and data loss by leveraging services like backups, replication, and multi-region setups to meet RTO and RPO goals.

**Backup and Restore**:

* Simplest strategy: Store backup data in S3 for quick, reliable recovery.
* RTO: Hours to days; RPO: Based on backup frequency (e.g., daily).
* Cost-effective for non-critical workloads.

**Pilot Light for Quick Recovery**:

* Core system components (e.g., databases) run minimally in AWS and stay updated via replication.
* Faster recovery than backup/restore as infrastructure is pre-provisioned.
* RTO: Minutes to hours; RPO: Near-zero with continuous replication.

**Warm Standby Solution**:

* Scaled-down, fully functional environment runs continuously in the cloud with replicated data.
* Scales up quickly during failover.
* RTO: Minutes; RPO: Minutes to hours.

**Multi-Site Solution**:

* Active-active configuration across multiple sites or regions for redundant operations.
* Traffic routes seamlessly to healthy sites.
* RTO: Seconds to minutes; RPO: Near-zero.

**AWS Production to DR Using Multiple AWS Regions**:

* Leverage AWS Regions and Availability Zones for geo-redundancy.
* Supports automated failover across global infrastructure.

**Key AWS Services for Disaster Recovery**:

* **S3**: Destination for backup data with fast restore capabilities.
* **Import/Export**: Ships physical storage devices to AWS for large datasets.
* **Glacier**: Low-cost, long-term archival storage with hours-long retrieval.
* **Server Migration Service**: Agentless migration of on-premises servers to AWS.
* **Database Migration Service and Schema Conversion Tool**: Migrates databases to AWS and converts SQL schemas.
* **Storage Gateway**: Copies on-premises snapshots to S3 for backup; creates EBS or local volumes.
* **Amazon Machine Images (AMIs)**: Preconfigured server bundles for rapid deployment.
* **Elastic Load Balancing (ELB)**: Distributes traffic across multiple instances.
* **Route 53**: DNS routing to failover sites or applications.
* **Elastic IP**: Static IP addresses for consistent addressing.
* **Amazon VPC**: Isolated, private cloud sections for secure environments.
* **Direct Connect**: Dedicated network from on-premises to AWS.
* **RDS**: Managed relational databases with scalability.
* **DynamoDB**: NoSQL database for high-availability data storage.
* **Redshift**: Petabyte-scale data warehouse for analytics.
* **CloudFormation**: Provisions related resources predictably via templates.
* **Elastic Beanstalk**: Deploys and scales web applications.
* **OpsWorks**: Manages deployment and operations for various app types.

## **AWS Well-Architected Framework**

### **AWS Well-Architected Framework**

* A set of design principles, best practices, and tools to build and run efficient, secure, reliable, and cost-effective workloads on AWS.
* Ensures cloud architectures align with AWS best practices for optimal performance, security, and cost.
* Provides a structured approach to evaluate and improve IT infrastructure across six pillars: Operational Excellence, Security, Reliability, Performance Efficiency, Cost Optimization, and Sustainability.
* Increases likelihood of business success by identifying and addressing architectural weaknesses.

**Purpose and Use**:

* + Acts as a body of knowledge (PDF or online AWS documentation) with best practices, design patterns, and evaluation questions.
  + Helps review and measure cloud architecture quality for existing or new systems.
  + Example: For a financial app, use the Security pillar to verify data protection and compliance.
  + **How to Use**:
    - Answer framework questions (e.g., “How do you protect data at rest?”) to assess architecture.
    - Identify gaps (e.g., lack of encryption) and apply prescriptive guidance to fix deficiencies.
  + **Why Use**: Improves system efficiency, reduces risks, and ensures alignment with industry standards.
  + **When to Use**: During architecture design, before production deployment, or for periodic reviews to optimize systems.

**General Notes**:

* **Evaluation Process**: Use framework questions to assess architecture (e.g., “How do you manage identities?”) and follow guidance to address gaps.
* **Tools**: AWS Well-Architected Tool (available in AWS Console) automates reviews and provides recommendations.
* **Why Use Framework**: Ensures robust, secure, and cost-effective systems; aligns with business goals; reduces risks.
* **When to Use**: During design, before deployment, or for periodic audits of production systems.

Reference:

<https://d1.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf>

**Six Pillars of the AWS Well-Architected Framework**:

**1. Operational Excellence**:

* **Definition**: Ability to run and monitor systems to deliver business value while improving processes and procedures.
* **Why Use**: Ensures smooth operations, quick issue resolution, and continuous improvement.
* **When to Use**: For managing production workloads, automating operations, or aligning IT with business goals.
* **Best Practice Areas**:
  + **Organization**: Align teams and processes (AWS Cloud Compliance, AWS Organizations).
  + **Prepare**: Plan for operations (AWS Config for resource tracking).
  + **Operate**: Monitor and manage systems (Amazon CloudWatch for metrics).
  + **Evolve**: Improve processes iteratively (Amazon Elasticsearch Service for insights).
* **Key AWS Service**: AWS CloudFormation (creates templates for consistent resource provisioning).

**2. Security:**

* **Definition**: Protect information, systems, and assets through risk assessments and mitigation strategies.
* **Why Use**: Safeguards sensitive data and ensures compliance with regulations.
* **When to Use**: For systems handling sensitive data (e.g., financial, healthcare) or requiring compliance (e.g., GDPR, HIPAA).
* **Best Practice Areas**:
  + **Security**: Implement shared responsibility model (AWS Config, AWS Trusted Advisor).
  + **Identity and Access Management**: Secure access (IAM, Multi-Factor Authentication, AWS Organizations).
  + **Detective Controls**: Monitor threats (AWS CloudTrail, Amazon GuardDuty).
  + **Infrastructure Protection**: Secure networks (Amazon VPC, AWS Shield, AWS WAF).
  + **Data Protection**: Encrypt data (EBS, S3, RDS encryption, AWS KMS, Amazon Macie).
  + **Incident Response**: React to issues (IAM, CloudWatch Events).
* **Key AWS Service**: AWS Identity and Access Management (IAM) for access control.

**3. Reliability:**

* **Definition**: Ensure systems recover from disruptions, scale dynamically, and mitigate issues like misconfigurations or network failures.
* **Why Use**: Maintains uptime and resilience for critical applications.
* **When to Use**: For production systems needing high availability or disaster recovery.
* **Best Practice Areas**:
  + **Foundations**: Set up resilient infrastructure (IAM, VPC, AWS Trusted Advisor, AWS Shield).
  + **Change Management**: Track and manage changes (AWS CloudTrail, AWS Config, Auto Scaling).
  + **Failure Management**: Recover from failures (AWS CloudFormation, S3, AWS KMS, Glacier).
  + **Workload Architecture**: Design for reliability (AWS SDK, AWS Lambda).
* **Key AWS Service**: Amazon CloudWatch for monitoring and alerting.

**4. Performance Efficiency:**

* **Definition**: Use resources efficiently to meet system requirements and adapt to changing demands and technologies.
* **Why Use**: Optimizes performance to reduce latency and improve user experience.
* **When to Use**: For applications with variable demand or performance-critical workloads (e.g., streaming, gaming).
* **Best Practice Areas**:
  + **Selection**: Choose optimal resources (Auto Scaling for compute, EBS/S3 for storage, RDS/DynamoDB for databases, Route 53/VPC/Direct Connect for networking).
  + **Review**: Stay updated with AWS innovations (AWS Blog, What’s New section).
  + **Monitoring**: Track performance (Amazon CloudWatch).
  + **Tradeoffs**: Balance performance and cost (Elasticache, CloudFront, Snowball, RDS read replicas).
* **Key AWS Service**: Amazon CloudWatch for performance monitoring.

**5. Cost Optimization:**

* **Definition**: Avoid unnecessary costs by using cost-effective resources and managing spend.
* **Why Use**: Reduces expenses while maintaining performance and functionality.
* **When to Use**: For budget-conscious projects or optimizing existing AWS workloads.
* **Best Practice Areas**:
  + **Cloud Financial Management**: Visualize costs (Amazon QuickSight, AWS Cost and Usage Report).
  + **Cost-Effective Resources**: Select optimal services (Cost Explorer, Trusted Advisor, Amazon Aurora, Direct Connect/CloudFront).
  + **Matching Supply and Demand**: Scale dynamically (Auto Scaling).
  + **Expenditure Awareness**: Monitor spending (AWS Cost Explorer, AWS Budgets).
  + **Optimizing Over Time**: Adapt to changes (AWS News Blog, Trusted Advisor).
* **Key AWS Service**: AWS Cost Explorer for cost analysis and forecasting.

**6. Sustainability:**

* **Definition**: Maximize resource efficiency to reduce environmental impact across workload components.
* **Why Use**: Promotes eco-friendly practices while maintaining performance.
* **When to Use**: For organizations prioritizing green IT or regulatory sustainability goals.
* **Best Practice Areas**:
  + **Region Selection**: Choose efficient regions (AWS Global Infrastructure).
  + **User Behavior Patterns**: Optimize usage (Auto Scaling, Elastic Load Balancing).
  + **Software and Architecture Patterns**: Design efficient systems (AWS Design Principles).
  + **Data Patterns**: Use efficient storage (EBS, EFS, FSx, S3).
  + **Hardware Patterns**: Select efficient compute (EC2, Elastic Beanstalk).
  + **Development and Deployment Process**: Automate for efficiency (AWS CloudFormation).
* **Key AWS Service**: Amazon EC2 Auto Scaling for resource efficiency.

**AWS Pricing**

* AWS uses a pay-as-you-go pricing model: You pay only for the resources you use, with no long-term contracts or upfront commitments for most services. Pricing is similar to utilities (e.g., pay for electricity as you consume it).
* **Three fundamental cost drivers**:
  + **Compute**: Costs for processing power (e.g., EC2 instances, Lambda invocations).
  + **Storage**: Costs for data retention (e.g., S3 buckets, EBS volumes).
  + **Outbound Data Transfer**: Costs for data leaving AWS (e.g., to the internet); inbound is free.

**On-Demand Pricing**:

* Default model: Pay by the second or hour for resources, with no minimums or long-term commitments.
* Flexible for variable workloads but higher cost than committed options.
* Applies to most services (e.g., EC2, S3).

**Reserved Capacity (Reserved Instances)**:

* For services like Amazon EC2, EMR, and RDS: Commit to a term (1 or 3 years) for discounted capacity.
* Savings: Up to 75% compared to On-Demand pricing.
* **Payment Options**:
  + **All Upfront**: Full payment at start for the entire term; highest discount.
  + **Partial Upfront**: Low initial payment + discounted hourly rate for the term.
  + **No Upfront**: No initial payment; discounted hourly rate for the term.
* Why Use: Ideal for predictable, steady-state workloads to reduce costs.
* When to Use: For long-running production environments where capacity needs are stable.

**Volume-Based Discounts**:

* Automatic discounts for high usage volumes (e.g., Amazon S3 tiers: lower rates for >100 GB/month).
* Encourages scaling; applies to storage, data transfer, and some compute services.
* Why Use: Cost optimization for growing workloads.
* When to Use: As usage increases, monitor via AWS Cost Explorer for tiered savings.

**Savings Plans**:

* Flexible alternative to Reserved Instances: Commit to a consistent spend amount (e.g., $10/hour) for 1-3 years, applicable across EC2, Lambda, Fargate, and more.
* Savings: Up to 72% over On-Demand.
* Why Use: More flexible than Reserved Instances (e.g., switch instance types or regions).
* When to Use: For dynamic environments with varying compute needs but predictable overall spend.

**Spot Instances**:

* Bid on unused EC2 capacity for up to 90% savings; interruptions possible with 2-minute notice.
* Why Use: Cost-effective for fault-tolerant, non-critical workloads.
* When to Use: Batch processing, testing, or ML training where interruptions are tolerable.

**AWS Free Tier**:

* Available for new AWS accounts to explore services at no charge.
* **Types of Free Usage**:
  + **Always Free**: 30+ services with perpetual monthly limits (e.g., 750 hours/month of t3.micro EC2, 5 GB S3 storage).
  + **12-Month Free**: Limited usage of select services for the first 12 months (e.g., 750 hours/month of EC2, 20 GB RDS storage).
  + **Short-Term Trials**: Time- or usage-limited trials for specific services (e.g., 1 million Lambda requests/month for 12 months).
  + **New Account Credits**: Up to $200 USD in credits ($100 at sign-up + $100 more while exploring).
* **Duration**: Always Free (ongoing); 12-Month Free (from account creation); Short-Term Trials (starts on first use, up to 12 months).
* **Limitations**: Exceeding limits incurs standard charges; Free Plan limited to select services for up to 6 months; switch to Paid Plan for full access.
* **Important Notes**: No charges under Free Plan unless opting into Paid; credits auto-apply for overages. More details at<https://aws.amazon.com/free/>. As of 2025, no major structural changes noted.
* Why Use: Test AWS without risk; learn services hands-on.
* When to Use: For new users, proofs-of-concept, or development/testing phases.

**AWS Pricing Calculator**:

* Free tool to estimate monthly AWS bills for individual services, workloads, or full architectures.
* **Key Features**:
  + Four main pages: Landing (overview), Add Service (search/filter services), Configure Service (input usage details, region), My Estimate (review/export).
  + Supports templates for common use cases (e.g., web app, data analytics).
  + Now generally available in AWS Console (as of May 2025); generates workload or full-bill estimates.
  + Includes rate configurations: View after discounts, volume savings, and commitments (e.g., Savings Plans, Reserved Instances).
  + Export options: CSV/JSON with resource-level details for analysis or integration with financial tools.
  + No AWS account required; available in all commercial regions (excl. China).
* Why Use: Plan budgets, compare options, optimize for lowest cost; helps estimate migration costs to cloud.
* When to Use: Before launching workloads, during architecture design, or for cost forecasting in 2025 projects (e.g., serverless apps with Lambda/SQS/DynamoDB).

**General Best Practices for AWS Pricing**:

* Monitor with AWS Cost Explorer and Budgets to track spend and set alerts.
* Use tagging for cost allocation by team/project.
* Optimize: Right-size resources, use elasticity, and review commitments annually.
* 2025 Trends: Enhanced calculator support for AI/ML workloads and multi-region estimates for global apps.

## AWS Pricing Resources:

<https://docs.aws.amazon.com/pdfs/whitepapers/latest/how-aws-pricing-works/how-aws-pricing-works.pdf>[https://aws.amazon.com/pricing/](https://aws.amazon.com/pricing/?aws-products-pricing.sort-by=item.additionalFields.productNameLowercase&aws-products-pricing.sort-order=asc&awsf.Free%20Tier%20Type=*all&awsf.tech-category=*all)<https://docs.aws.amazon.com/whitepapers/latest/how-aws-pricing-works/amazon-ec2.html>

## **AWS Support Plans**

* AWS offers support plans to provide tools, resources, and human assistance to customers based on their needs.
* Five plans: Basic, Developer, Business, Enterprise On-Ramp, Enterprise.
* Plans vary by use case, access to support, response times, and proactive services.

**Basic Support Plan**:

* **Definition**: Free, default plan available to all AWS customers.
* **Key Features**:
  + 24/7 access to customer service, documentation, whitepapers, and support forums.
  + AWS Trusted Advisor: Limited to 7 core checks (service quotas, security).
  + AWS Personal Health Dashboard: Alerts and status for AWS service health impacting your resources.
* **Why Use**: Basic troubleshooting and account/billing support for all users.
* **When to Use**: For non-production, small-scale, or new AWS users exploring services.

**Developer Support Plan**:

* **Definition**: Entry-level paid plan for experimenting or testing in AWS.
* **Key Features**:
  + AWS Trusted Advisor: Service quota and security checks.
  + General architectural guidance.
  + Business hours email access to Cloud Support Associates.
  + Unlimited cases, but limited to one primary contact.
  + Prioritized responses on AWS re:Post.
  + **Response Times**:
    - General guidance: < 24 business hours.
    - System impaired: < 12 business hours.
* **Why Use**: Affordable support for early-stage or non-critical workloads.
* **When to Use**: For developers, startups, or proofs-of-concept (POCs) needing basic technical support.

**Business Support Plan**:

* **Definition**: Designed for production workloads in AWS.
* **Key Features**:
  + Full set of AWS Trusted Advisor checks for best practices.
  + Contextual architectural guidance based on use cases.
  + 24/7 phone, email, and chat access to Cloud Support Engineers.
  + Unlimited cases and contacts (IAM-supported).
  + Prioritized responses on AWS re:Post.
  + Third-party software interoperability and configuration guidance.
  + Access to Support Automation Workflows (AWSSupport prefix).
  + Infrastructure Event Management available for an additional fee.
  + **Response Times**:
    - General guidance: < 24 hours.
    - System impaired: < 12 hours.
    - Production system impaired: < 4 hours.
    - Production system down: < 1 hour.
* **Why Use**: Comprehensive support for stable production environments.
* **When to Use**: For businesses running critical applications needing faster response times and broader support.

**Enterprise On-Ramp Support Plan**:

* **Definition**: Intermediate plan for business or mission-critical workloads.
* **Key Features**:
  + Full set of AWS Trusted Advisor checks.
  + Consultative review and guidance tailored to applications.
  + Access to a pool of Technical Account Managers (TAMs) for proactive guidance and program coordination.
  + Access to online self-paced labs for training.
  + Concierge Support Team for account assistance.
  + AWS Support API for programmatic case management.
  + Access to Support Automation Workflows (AWSSupport prefix).
  + One Infrastructure Event Management session per year included.
  + **Response Times**:
    - General guidance: < 24 hours.
    - System impaired: < 12 hours.
    - Production system impaired: < 4 hours.
    - Production system down: < 1 hour.
    - Business-critical system down: < 30 minutes.
* **Why Use**: Bridges gap between Business and Enterprise with proactive support.
* **When to Use**: For organizations scaling up with critical workloads needing TAM support and faster responses.

**Enterprise Support Plan**:

* **Definition**: Premium plan for mission-critical workloads with comprehensive support.
* **Key Features**:
  + Full set of AWS Trusted Advisor checks.
  + Consultative review and guidance tailored to applications.
  + Designated Technical Account Manager (TAM) for proactive monitoring, optimization, and coordination with AWS experts.
  + Access to online self-paced labs for training.
  + Concierge Support Team for account assistance.
  + AWS Support API for programmatic case management.
  + Access to Support Automation Workflows (AWSSupport and AWSPremiumSupport prefixes).
  + Infrastructure Event Management included.
  + Proactive reviews, workshops, and deep dives.
  + White-glove case routing for priority handling.
  + Management business reviews for strategic planning.
  + **Response Times**:
    - General guidance: < 24 hours.
    - System impaired: < 12 hours.
    - Production system impaired: < 4 hours.
    - Production system down: < 1 hour.
    - Business-critical system down: < 15 minutes.
* **Why Use**: Maximum support for complex, mission-critical systems requiring minimal downtime.
* **When to Use**: For large enterprises with business-critical applications needing dedicated support and rapid response.

**General Notes**:

* **Choosing a Plan**:
  + Developer: For experimentation, testing, or POCs.
  + Business: For production workloads with moderate criticality.
  + Enterprise On-Ramp/Enterprise: For business/mission-critical workloads needing proactive support and fast response.
* **Technical Account Manager (TAM)**: Only available in Enterprise (designated) and Enterprise On-Ramp (pooled).
* **Cost Considerations**: Basic is free; Developer is low-cost; Business, Enterprise On-Ramp, and Enterprise scale with price and features. Check<https://aws.amazon.com/premiumsupport/plans/> for details.
* **Use Case Example**: A startup testing an app chooses Developer; an e-commerce platform with steady traffic uses Business; a bank with critical systems opts for Enterprise.

## **AWS Regions, Availability Zones & Edge Locations**

## **1. AWS Region**

### **What is an AWS Region?**

An **AWS Region** is a **geographical location** in the world where Amazon has a group of data centers.  
Example:

* **us-east-1 (N. Virginia)**
* **eu-west-1 (Ireland)**
* **ap-south-1 (Mumbai)**

Each region works **independently**.

### **Why Regions matter?**

1. **Data Residency** — Some countries require data to stay inside the country.
2. **Latency** — Choosing the closest region gives faster performance.
3. **Disaster Isolation** — Each region is separate, so failures don’t affect other regions.
4. **Cost Differences** — Pricing varies across regions.

### **How Regions work?**

* Each region contains **multiple Availability Zones**.
* You select a region when deploying AWS services.
* Some services are **regional** (e.g., EC2, RDS),  
  while others are **global** (e.g., IAM, Route 53).

## 🟩 **2. Availability Zone (AZ)**

### **What is an Availability Zone?**

An **Availability Zone** is:

* A **physically separate data center** inside a region
* It has **independent power, cooling, networking**
* AZs are connected to each other with **low-latency high-speed links**

Example AZs inside Singapore Region (ap-southeast-1):

* ap-southeast-1a
* ap-southeast-1b
* ap-southeast-1c

### **Why AZs matter?**

1. **High Availability**  
   Deploy applications across multiple AZs so if one fails, the other continues.
2. **Fault Isolation**  
   A problem in one AZ does not impact the others.
3. **Scalability**  
   Load balancers can distribute traffic across AZs.

### **How AZs work?**

* You launch resources like EC2 instances **inside AZs**.
* Multi-AZ deployments (like RDS) automatically replicate data.
* Load Balancers route traffic across AZs.

## 🟧 **3. Edge Location**

### **What is an Edge Location?**

An **Edge Location** is a **small data center** placed closer to users for:

* **Content Delivery (CDN)** using CloudFront
* **DNS routing** via Route 53
* **Global Accelerator traffic handling**
* **DDoS protection** using AWS Shield

The goal: **Deliver content fast with low latency**.

Example:  
There may be **only 1 Region** in a country, but **10+ edge locations**.

### **Why Edge Locations matter?**

1. **Faster performance** (videos, images, assets load quickly)
2. **Lower latency** for global users
3. **Improves application response time**
4. **Reduces load on your servers**

### **How Edge Locations work?**

* When a user requests your website:
  + CloudFront checks if the content is cached at the nearest edge location.
  + If cached → delivered instantly
  + If not → content is fetched from your origin (S3, EC2, or on-prem), then cached.

This process is called **Edge Caching**.

# **Summary Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Concept** | **What** | **Why** | **How** |
| **Region** | Geographical area with multiple data centers | Choose best location, cost, compliance | Select when deploying; isolated from other regions |
| **Availability Zone (AZ)** | Separate data center inside a region | High availability and fault tolerance | Deploy multi-AZ apps, load balancers distribute traffic |
| **Edge Location** | Small local caching data centers | Fast delivery, low latency | CloudFront caches content near users |

## **Ch # 3**

**AWS Services**



## **AWS Service Scoping**

## AZ-Scoped Services**:**

## **Operate within a specific Availability Zone (AZ) (e.g., us-east-1a).**

## **Examples: Amazon EC2 (Infrastructure as a Service), RDS databases.**

## **Resources must be explicitly deployed across multiple AZs to achieve high availability.**

## **Requires manual configuration for fault tolerance within a region.**

## Availability Zone (AZ) Services

* **Amazon EC2**: Virtual servers deployed in specific Availability Zones for compute tasks, tied to a single AZ unless configured for multi-AZ.
* **Amazon EBS**: Block storage volumes attached to EC2 instances, scoped to a single Availability Zone.
* **AWS Elastic Load Balancer (ELB)**: Distributes traffic within a specific Availability Zone or across multiple AZs within a Region, depending on configuration.
* **Amazon RDS (Multi-AZ Standby)**: While the primary database instance is AZ-specific, Multi-AZ deployments replicate to a standby instance in a different AZ within the same Region.

## Region-Scoped Services**:**

## **Operate across an entire AWS Region (e.g., us-east-1).**

## **Examples: Amazon S3, DynamoDB, Elastic Beanstalk (Platform as a Service), Lambda (Function as a Service), Rekognition (Software as a Service).**

## **Data remains within the region unless explicitly configured for cross-region replication.**

## **Services are tied to a region’s infrastructure for scalability and fault tolerance.**

## Regional Services

* **Amazon S3**: Object storage service scoped to a Region, with data stored redundantly across multiple AZs within that Region.
* **Amazon DynamoDB**: Managed NoSQL database service operating within a Region, with optional multi-Region replication via Global Tables.
* **AWS Elastic Beanstalk**: Platform for deploying applications, scoped to a Region but leveraging resources across AZs for high availability.
* **Amazon VPC**: Virtual network environment defined within a Region, with subnets spanning specific AZs.
* **AWS Lambda**: Serverless compute service scoped to a Region, executing functions across AZs for redundancy.
* **AWS App Runner**: Managed service for containerized applications, operating within a Region with automatic scaling across AZs.
* **AWS Step Functions**: Coordinates serverless workflows within a Region, integrating with Regional services.

## Global-Scoped Services**:**

## **Operate across all AWS Regions without needing region-specific configuration.**

## **Examples: Identity and Access Management (IAM), Route 53 (DNS service), CloudFront (Content Delivery Network), Web Application Firewall (WAF).**

## **Provide seamless functionality globally, managing resources or delivering content across regions.**

## Global Services

* **AWS Identity and Access Management (IAM)**: Manages user access and permissions globally across all AWS Regions and accounts.
* **Amazon Route 53**: Global DNS service for routing traffic to AWS resources or external endpoints worldwide.
* **AWS CloudFront**: Content delivery network (CDN) with global edge locations for low-latency content delivery.
* **Amazon Rekognition**: Image and video analysis service accessible globally, with processing tied to Regional endpoints.
* **Amazon Comprehend**: Natural language processing service available globally, with data processing in specified Regions.
* **AWS Chime**: Global communication service for meetings and conferencing, accessible across Regions.
* **AWS Organizations**: Manages multiple AWS accounts centrally with global Service Control Policies (SCPs).
* **Amazon WorkSpaces**: Virtual desktop service with global management, though desktop instances are deployed in specific Regions.

## Key Notes**:**

## Service Availability**: Not all AWS services are available in every region; check the AWS Regional Services List for availability.**

## Cross-Region Replication**: Region-scoped services like S3 or DynamoDB require explicit setup for data replication to other regions for disaster recovery or global access.**

## Global Services Use Case**: IAM manages user access globally, Route 53 routes traffic worldwide, and CloudFront delivers content via edge locations.**

# **AWS Services and Their Service Model**

## Infrastructure as a Service (IaaS)

## Amazon EC2**: Provides scalable virtual servers for compute capacity, allowing full control over the operating system and applications.**

## Amazon VPC**: Enables isolated cloud networks for launching resources with customizable networking configurations.**

## Amazon EBS**: Offers block storage volumes for use with EC2 instances, providing persistent storage.**

## AWS Elastic Load Balancer (ELB)**: Distributes incoming traffic across multiple EC2 instances for scalability and reliability.**

## Platform as a Service (PaaS)

## Amazon S3**: Provides scalable object storage for data storage, backup, and retrieval.**

## Amazon DynamoDB**: A fully managed NoSQL database service for high-performance, scalable applications.**

## AWS Elastic Beanstalk**: Simplifies application deployment and management by handling infrastructure provisioning and scaling.**

## Amazon RDS**: Managed relational database service supporting multiple database engines (e.g., MySQL, PostgreSQL).**

## AWS App Runner**: Fully managed service for deploying containerized web applications with simplified scaling and management.**

## Function as a Service (FaaS)

## AWS Lambda**: Serverless compute service that runs code in response to events without managing servers.**

## AWS Step Functions**: Coordinates multiple AWS services into serverless workflows for building distributed applications.**

## Software as a Service (SaaS)

## Amazon Rekognition**: Provides image and video analysis for facial recognition, object detection, and content moderation.**

## Amazon Comprehend**: A natural language processing service for text analysis, sentiment detection, and entity recognition.**

## Amazon WorkSpaces**: Managed desktop-as-a-service solution for providing virtual desktops to users.**

## AWS Chime**: A communication service for online meetings, video conferencing, and business calling.**

## **Ch # 4**

**AWS Global Infrastructure**



# **AWS Global Infrastructure**

* The AWS Global Infrastructure is a secure, reliable, and scalable network of data centers, regions, and edge locations that supports AWS cloud services, serving over 1 million active customers in 190+ countries.
* Enables high availability, fault tolerance, low-latency content delivery, and compliance with data sovereignty.
* There are 4 aws global infrastructure components regions,  availability zones (AZs), edge networks and AWS local regions.
* Globally distributed, interconnected sets of independent data centers providing on-demand IT resources over the internet with pay-as-you-go pricing.

**Benefits of AWS Global Infrastructure**:

* **High Availability**: Running applications across multiple AZs ensures systems remain operational during outages.
* **Fault Tolerance**: Multi-AZ and multi-region setups prevent single points of failure.
* **Low Latency**: Edge Locations and Regional Edge Caches deliver content faster to global users.
* **Scalability**: Supports massive workloads with flexible resource allocation.
* **Data Sovereignty**: AZs and Regions comply with local data privacy regulations.
* **Content Delivery**: CDN (via CloudFront) caches content globally for faster access.

**Use Case Example**:

* Store high-resolution images in a California server, cache them in edge locations (e.g., Philippines, India, Singapore) for faster access by Asian users, reducing latency compared to fetching from the origin server.

# **AWS IAM (Identity and Access Management) — Complete Explanation**

# **1. WHAT is IAM?**

AWS Identity and Access Management (IAM) is a **core security service** that allows you to securely control **who** (identities) can access **what** (resources) and **how** they can access AWS services.

It provides:

* **Authentication** → Confirming who is trying to access.
* **Authorization** → Deciding what actions they are allowed to perform.

IAM lets you centrally manage **users, groups, roles, and policies** across your AWS environment.

# **2. WHY do we use IAM? (Purpose)**

IAM exists to provide **fine-grained, secure, and scalable** access control:

### ✅ Prevent unauthorized access

IAM ensures only approved identities can access AWS resources.

### ✅ Enforce least privilege

IAM allows granting only the minimum permissions required for a task.

### ✅ Secure workloads

Services like EC2, Lambda, ECS, EKS, etc. use IAM roles instead of credentials.

### ✅ Support multi-user and multi-account environments

IAM makes it easy to manage permissions for teams, automation, and applications.

### ✅ Enable enterprise security

Integrates with SSO, SAML, Active Directory, Organizations, STS, SCPs, and CloudTrail.

# **3. HOW does IAM work?**

IAM controls permissions using:

* **Identities:** Users, Groups, Roles
* **Policies:** JSON documents that define allowed or denied actions
* **Resource-based policies:** Policies attached directly to AWS services (e.g., S3 bucket policies)
* **Evaluation logic:** IAM calculates final permissions based on Allow/Deny rules

IAM checks **WHO** is making the request + **WHAT** policy they have + **WHICH** resource they're accessing.

### **Policy Evaluation Logic**

IAM uses 3 rules:

1. **Explicit Deny → Always wins**
2. **Explicit Allow → Grants access unless denied elsewhere**
3. **No Allow → Access denied**

### **IAM + Resource-based policy = Final Decision**

Examples:

* IAM allows, bucket denies → **DENIED**
* IAM denies, bucket allows → **DENIED**
* IAM allows, bucket allows → **ALLOWED**
* IAM allows, bucket empty → **ALLOWED**

# **4. WHEN do you use IAM?**

IAM is used **whenever human or system access is needed**, including:

### **For People**

* Give developers access to S3 or EC2
* Create admins, power users, read-only auditors
* Manage multi-team environments

### **For AWS Services**

* EC2 instance needs to access S3 or DynamoDB → use **IAM role**
* Lambda function needs CloudWatch logs → use **execution role**
* ECS task needs ECR pull permissions → use **task IAM role**

### **For Applications**

* Mobile apps using Cognito → IAM behind the scenes
* Scripts using AWS CLI → IAM user with access keys (not recommended)

### **For Temporary Access**

* Contractors & short-term users → STS temporary credentials
* Cross-account access → AssumeRole with trust policy

# **5. TYPES of IAM Components**

## **A. IAM Users**

* Represent **human users**
* Auth with **passwords** (console) or **access keys** (CLI)
* Should follow MFA + strong password rules
* Recommended to avoid access keys wherever possible

## **B. IAM Groups**

* Collections of **users**
* Simplify permission management
* Cannot contain roles or other groups

## **C. IAM Roles**

* Identities **assumed temporarily**, not tied to any one person
* Used by:
  + EC2
  + Lambda
  + ECS tasks
  + SSO users
  + Cross-account access
* Have:
  + **Permission policy** → what the role can access
  + **Trust policy** → who/what can assume the role

## **D. IAM Policies**

JSON documents containing:

* **Effect**: Allow / Deny
* **Action**: AWS API operations
* **Resource**: ARN of resource
* **Condition**: Optional filters (IP, MFA, tags, time, VPC)

### **Types of Policies**

| **Type** | **Use Case** | **Notes** |
| --- | --- | --- |
| **AWS Managed Policy** | Quick setup | Maintained by AWS |
| **Customer Managed Policy** | Best practice | Reusable, version-controlled |
| **Inline Policy** | Special one-off permissions | Harder to manage |

## **E. Dynamic Policies (Policy Variables)**

Used for **generic reusable permissions**.

Example:

s3://mybucket/${aws:username}/\*

Allows each user to access only their own folder.

## **F. STS (Security Token Service)**

* Issues **temporary credentials**
* Used for:
  + Cross-account access
  + Federated users
  + Temporary CLI tokens
  + AssumeRole

## **G. IAM Certificate Store**

* Stores 3rd-party SSL/TLS certificates
* Alternative to ACM (AWS Certificate Manager)

# **6. KEY FEATURES of IAM**

### **1. Fine-grained access control**

Control specific actions like:

* s3:ListBucket
* ec2:StartInstances
* lambda:InvokeFunction

### **2. Centralized identity management**

Manage users, groups, roles from one place.

### **3. Federated access**

IAM integrates with:

* Active Directory
* Azure AD
* Google Workspace
* SAML
* OIDC
* Identity Center (AWS SSO)

### **4. Temporary credentials**

Secure access without long-lived passwords or keys.

### **5. Resource-based policies**

Policies attached directly to:

* S3 buckets
* SNS topics
* SQS queues
* Lambda functions
* API Gateway

### **6. PassRole**

Allows a user/application to **pass** a role to a service (e.g., EC2).

Controls **which IAM role** can be attached during instance creation.

### **7. Policy Simulator**

Visual tool to test permissions before deploying.

### **8. Secure access for workloads**

EC2/Lambda roles eliminate the need to store keys or passwords.

### **9. IAM Access Advisor**

Shows:

* Which permissions are unused
* Helps enforce least privilege

### **10. Account Alias**

Replaces:

123456789012.signin.aws.amazon.com

with:

my-company.signin.aws.amazon.com

# **7. IAM BEST PRACTICES**

* Delete root account access keys
* Enable MFA for all users
* Use groups instead of giving users direct policies
* Prefer **customer-managed** policies
* Avoid inline policies
* Rotate keys and passwords
* Use least privilege access
* Audit permissions with IAM Access Advisor
* Use CloudTrail to log all IAM events
* Use IAM roles instead of access keys

# **8. Example Knowledge Validation**

### **Question:**

An EC2 IAM role allows s3:PutObject, but the bucket policy denies it. Can EC2 write to the bucket?

### **Answer:**

❌ **NO — access is denied.**

**Reason:**  
Explicit deny in a bucket policy overrides allow in any IAM policy.

# **AWS IAM Role**

## Definition

* **IAM Role**: An AWS identity that grants temporary permissions to perform specific actions without requiring long-term credentials (e.g., passwords or access keys). It can be assumed by trusted entities such as users, AWS services, or other AWS accounts.

## How IAM Roles Work

* **Step 1: Request to Assume Role**: A trusted entity (e.g., user, AWS service, or another AWS account) requests to assume the IAM role via AWS Security Token Service (STS).
* **Step 2: Temporary Credentials Issuance**: STS provides temporary credentials, including an access key, secret key, and session token.
* **Step 3: Interaction with AWS Services**: The entity uses these temporary credentials to access AWS services as permitted by the role’s attached policies.
* **Step 4: Session Expiry**: Once the session expires, the entity must re-assume the role to obtain new credentials for continued access.

## Duration of Temporary Credentials

* **Default**: 1 hour (3,600 seconds)
* **Minimum**: 15 minutes (900 seconds)
* **Maximum**: 12 hours (43,200 seconds), if the IAM role is configured to allow a longer session duration.

## IAM Role Types

* **Service Roles**: Used by AWS services (e.g., EC2, Lambda, ECS) to access other AWS resources on behalf of the service.
  + **Example**: An EC2 instance assuming a role to access an S3 bucket.
* **User Roles**: Allow IAM users within the same or different AWS accounts to assume roles for temporary access to resources.
  + **Example**: A developer assuming a role to manage resources in a production environment.
* **Cross-Account Roles**: Enable entities in one AWS account to access resources in another AWS account.
  + **Example**: Allowing a third-party AWS account to manage specific resources in your account.
* **Federated Roles**: Used with identity federation to grant access to external identities (e.g., via SAML, OpenID Connect, or AWS SSO).
  + **Example**: Corporate employees using single sign-on (SSO) to access AWS resources.

## Key Concepts

* **Trust Policy**: A JSON policy document that defines which entities (e.g., AWS services, users, or accounts) are trusted to assume the role.
* **Permissions Policy**: A JSON policy document that specifies the actions and resources the role can access once assumed.
* **AssumeRole API**: The AWS STS API operation used to assume a role and obtain temporary credentials.
* **No Long-Term Credentials**: Unlike IAM users, roles do not have permanent credentials, enhancing security by reducing the risk of credential exposure.
* **Role Chaining**: A role assumed by an entity can assume another role, but the session duration is limited to 1 hour and cannot be extended.
* **Use Cases**:
  + Granting AWS services access to other resources (e.g., Lambda accessing DynamoDB).
  + Enabling cross-account access for collaboration or third-party management.
  + Providing temporary access for users or applications without sharing long-term credentials.
* **Security Best Practices**:
  + Follow the principle of least privilege by granting only necessary permissions in the role’s policies.
  + Regularly rotate and audit roles and their trust policies to minimize security risks.
  + Use conditions in policies (e.g., MFA requirements or source IP restrictions) to enhance security.

# **AWS IAM Policy**

## Definition

* **IAM Policy**: A JSON document in AWS that defines permissions, specifying which actions are allowed or denied on specific AWS resources for users, groups, or roles.

## How IAM Policies Work

* **Attachment**: Policies are attached to IAM identities (users, groups, or roles) or resources to control access to AWS services and resources.
* **Evaluation**: When an entity makes a request, AWS evaluates all applicable policies to determine if the action is allowed or denied based on the policy’s statements.
* **Statement Structure**: Each policy contains one or more statements with:
  + **Effect**: Allow or Deny the specified actions.
  + **Action**: The AWS service actions (e.g., s3:GetObject, ec2:StartInstances) allowed or denied.
  + **Resource**: The AWS resources (e.g., S3 buckets, EC2 instances) the actions apply to.
  + **Condition**: Optional conditions (e.g., MFA requirement, IP address) that must be met for the policy to apply.
* **Default Deny**: If no policy explicitly allows an action, AWS denies the request by default.

## IAM Policy Types

* **Identity-Based Policies**: Attached to IAM users, groups, or roles to grant permissions to those identities.
  + **Managed Policies**:
    - **AWS-Managed Policies**: Predefined by AWS, reusable across accounts (e.g., AmazonS3ReadOnlyAccess).
    - **Customer-Managed Policies**: Created and managed by the user for custom permissions, reusable within the account.
  + **Inline Policies**: Embedded directly into a single user, group, or role, not reusable.
* **Resource-Based Policies**: Attached to AWS resources (e.g., S3 buckets, SNS topics) to define who can access the resource and what actions they can perform.
  + **Example**: An S3 bucket policy allowing specific IAM users to read objects.
* **Permissions Boundaries**: Policies that set the maximum permissions an IAM user or role can have, used to limit the scope of permissions granted by other policies.
* **Service Control Policies (SCPs)**: Used in AWS Organizations to set permission guardrails for all accounts within an organization, applied at the account or organizational unit (OU) level.
* **Session Policies**: Temporary policies applied when assuming a role, further restricting permissions for that session.

## Key Concepts

* **Principle of Least Privilege**: Policies should grant only the permissions necessary for a task to minimize security risks.
* **Policy Evaluation Logic**: AWS combines all applicable policies (identity-based, resource-based, SCPs, etc.) and evaluates them as follows:
  + Explicit Deny in any policy overrides any Allow.
  + Explicit Allow is required for access; otherwise, the default is Deny.
  + Conditions in policies must be met for the policy to apply.
* **Policy Variables**: Use placeholders (e.g., ${aws:username}) in policies to make them dynamic and reusable across users or roles.
* **Policy Simulator**: AWS provides a tool to test and validate IAM policies to ensure they grant or deny the intended permissions.
* **Versioning**: Managed policies support versioning, allowing updates while preserving previous versions for rollback or auditing.
* **Use Cases**:
  + Restricting access to specific S3 buckets for a group of users.
  + Allowing an EC2 instance role to access a DynamoDB table.
  + Enforcing organization-wide restrictions using SCPs in AWS Organizations.
* **Security Best Practices**:
  + Regularly review and audit policies to remove unnecessary permissions.
  + Use AWS-managed policies for common tasks to reduce errors.
  + Include conditions (e.g., MFA, time-based restrictions) to enhance security.
  + Use permissions boundaries to limit the scope of roles in sensitive environments.

**AWS IAM Permission Boundary**

* A permission boundary is an IAM policy that sets the **maximum permissions** an IAM user or role can have.
* Acts as a restrictive filter or ceiling on the permissions granted by other IAM policies.

**Key Points**:

* **Does Not Grant Permissions**: Only limits what an IAM policy can allow.
* **Intersection Logic**: Effective permissions = IAM Policy ∩ Permission Boundary.
  + User/role can only perform actions allowed by **both** the IAM policy and the permission boundary.
* **Scope**: Applies to IAM users or roles, not groups or resources.
* **Enforcement**: If a permission boundary is set, it overrides broader permissions in the user/role’s IAM policy.

**Example**:

* **User Policy**: Allows EC2:StartInstances and EC2:TerminateInstances.
* **Permission Boundary**: Allows only EC2:StartInstances.
* **Result**:
  + User can start EC2 instances.
  + User cannot terminate instances (blocked by the boundary).

**Why Use Permission Boundaries?**:

* **Restrict Actions (Extra Control)**:
  + Limits actions even if the IAM policy is overly permissive.
  + Example: Allow stopping EC2 instances only with the tag Environment=Production.
* **Prevent Mistakes or Misuse**:
  + Protects against accidental or intentional risky actions.
  + Example: Deny EC2:TerminateInstances to prevent accidental deletion.
* **Controlled Delegation**:
  + Enables fine-grained access for teams or admins.
  + Example: One admin manages only “Production” servers, another only “Dev” servers.

**How It Works**:

* Attach a permission boundary policy to an IAM user or role.
* The boundary policy defines the maximum allowed actions (e.g., specific services, actions, or resource-based conditions).
* IAM evaluates the user/role’s policy and the boundary together; only actions permitted by both are allowed.
* If no boundary is set, the IAM policy alone determines permissions.

**Use Case Scenarios**:

* **Team-Based Restrictions**: Limit developers to specific S3 buckets or EC2 instances based on project tags.
* **Security Compliance**: Prevent users from performing sensitive actions (e.g., deleting critical resources).
* **Delegated Administration**: Allow an admin to manage IAM roles but restrict actions like creating overly permissive policies.

**Implementation Notes**:

* Permission boundaries are JSON policies, similar to IAM policies, with actions, resources, and conditions.
* Can include ALLOW statements for permitted actions; DENY statements in boundaries are rare but can be used.
* Apply via AWS Management Console, CLI, or SDK when creating/modifying users or roles.

**Limitations**:

* Cannot be applied to IAM groups or AWS-managed policies.
* Does not override explicit DENY in IAM policies (DENY always takes precedence).
* Only one permission boundary per user or role.
* Not retroactive; existing permissions remain until the boundary is applied or updated.

**Best Practices**:

* Use least privilege principles in boundary policies to minimize allowed actions.
* Include conditions (e.g., resource tags, IP ranges) for granular control.
* Regularly audit and update boundaries to align with changing roles or compliance needs.
* Combine with AWS Organizations Service Control Policies (SCPs) for account-level restrictions.
* Test boundaries in a non-production environment to avoid unintended access restrictions.

**Related AWS Services**:

* **IAM**: Manages users, roles, and policies.
* **AWS Organizations**: Use SCPs alongside permission boundaries for broader account-level restrictions.
* **CloudTrail**: Audit permission boundary usage and policy evaluations.
* **AWS Config**: Monitor compliance of permission boundaries.

**Comparison with Other IAM Controls**:

* **IAM Policy**: Grants permissions; defines what a user/role can do.
* **Permission Boundary**: Limits permissions; sets a ceiling on what an IAM policy can allow.
* **Service Control Policy (SCP)**: Applies at the AWS account or OU level in AWS Organizations, not user/role-specific.

**Validate Your Knowledge**:

* **Question**: A developer has an IAM policy allowing s3:PutObject and s3:DeleteObject on all S3 buckets. A permission boundary allows only s3:PutObject for buckets with Environment=Dev. What can the developer do?
  + **Answer**: The developer can only perform s3:PutObject on buckets tagged with Environment=Dev.
  + **Explanation**: The permission boundary restricts the IAM policy, allowing only the intersection of permissions (s3:PutObject on Environment=Dev buckets). s3:DeleteObject is blocked by the boundary.

## **Ch # 5**

**AWS Networking & Content Delivery Services**



### [**AWS Networking & Content Delivery**](https://tutorialsdojo.com/aws-cheat-sheets-networking-and-content-delivery/)

* [Amazon API Gateway](https://tutorialsdojo.com/amazon-api-gateway/)
* [Amazon CloudFront](https://tutorialsdojo.com/amazon-cloudfront/)
* [Amazon Route 53](https://tutorialsdojo.com/amazon-route-53/)
* [Amazon VPC](https://tutorialsdojo.com/amazon-vpc/)
* [AWS Direct Connect](https://tutorialsdojo.com/aws-direct-connect/)
* [AWS Elastic Load Balancing (ELB)](https://tutorialsdojo.com/aws-elastic-load-balancing-elb/)
* [AWS Global Accelerator](https://tutorialsdojo.com/aws-global-accelerator/)
* [AWS Transit Gateway](https://tutorialsdojo.com/aws-transit-gateway/)

**Other Networking-related Notes:**

* [Building a Private React Application Infrastructure with Terraform](https://tutorialsdojo.com/building-a-private-react-application-infrastructure-with-terraform/)
* [Building a Simple Video Hosting Service using Amazon CloudFront, Amazon S3, and AWS Amplify](https://tutorialsdojo.com/building-a-simple-video-hosting-service-using-amazon-cloudfront-amazon-s3-and-aws-amplify/)
* [Increasing MTU for Your EC2 Instance](https://tutorialsdojo.com/increasing-mtu-for-your-ec2-instance/)
* [Lightning-Fast Static Website with Amazon S3 and CloudFront with Origin Access Control (OAC)](https://tutorialsdojo.com/lightning-fast-static-website-with-amazon-s3-and-cloudfront-with-origin-access-control-oac/)
* [Longest Prefix Match: Understanding Advanced Concepts in VPC Peering](https://tutorialsdojo.com/longest-prefix-match-understanding-advanced-concepts-in-vpc-peering/)
* [Navigating DNS Management: Unveiling Amazon Route 53 Inbound and Outbound Resolver Endpoints](https://tutorialsdojo.com/navigating-dns-management-unveiling-amazon-route-53-inbound-and-outbound-resolver-endpoints/)
* [Resolve Route 53 Private Hosted Zones from an On-premises Network](https://tutorialsdojo.com/resolve-route-53-private-hosted-zones-from-an-on-premises-network/)
* [Setting Up Redirects on Amazon S3 for a Custom Domain Registered with Amazon Route 53](https://tutorialsdojo.com/setting-up-redirects-on-amazon-s3-for-a-custom-domain-registered-with-amazon-route-53/)
* [Split-view DNS using Amazon Route 53](https://tutorialsdojo.com/split-view-dns-using-amazon-route-53/)
* [Using Amazon Route 53 Resolver](https://tutorialsdojo.com/using-amazon-route-53-resolver/)
* [VPC Peering](https://tutorialsdojo.com/vpc-peering/)
* [Validate Referrer Headers Using CloudFront Function](https://tutorialsdojo.com/validate-referrer-headers-using-cloudfront-function/)

## [Amazon VPC](https://tutorialsdojo.com/amazon-vpc/)

* **VPC**: Define an IP address range (CIDR block), add subnets, associate security groups, and configure route tables.
* Amazon Virtual Private Cloud (VPC) is a virtual network dedicated to your AWS account, enabling the launch of AWS resources in a logically isolated environment.
* Networking layer for Amazon EC2, spanning all Availability Zones (AZs) in a region.
* Allows specification of IP address ranges, subnets, security groups, and route tables.
* All new AWS accounts have a default VPC.
* New EC2 instances are launched into the default VPC if no subnet is
* specified
* Default VPC has Internet connectivity
* All EC2 instances inside it have public IPv4 addresses
* We also get a public and a private IPv4 DNS names
* You can have multiple VPCs in an AWS region (max. 5 per region – soft limit)
* Max. CIDR per VPC is 5, for each CIDR:
  + Min. size is /28 (16 IP addresses)
  + Max. size is /16 (65536 IP addresses)
* VPC is private, only the Private IPv4 ranges are allowed:
  + 10.0.0.0 – 10.255.255.255 (10.0.0.0/8)
  + 172.16.0.0 – 172.31.255.255 (172.16.0.0/12)
  + 192.168.0.0 – 192.168.255.255 (192.168.0.0/16)
* Your VPC CIDR should NOT overlap with your other networks (e.g.,
* corporate)
* **Subnet**: A range of IP addresses within a VPC; public subnets connect to the internet, private subnets do not.
* **Security**: Use security groups (instance-level) and network access control lists (ACLs) (subnet-level) to protect resources.
* **Expansion**: Add secondary IP ranges to expand the VPC.

* **EC2-VPC vs. EC2-Classic**:
  + **EC2-VPC**: Default platform for modern AWS accounts, supports VPC features like subnets and internet gateways.
  + **EC2-Classic**: Legacy platform, no longer used for new accounts.

* **Default vs. Non-Default VPC**:
  + **Default VPC**:
    - Automatically created with a default subnet in each AZ.
    - Includes an internet gateway; default subnets are public with private and public IPv4 addresses for instances.
  + **Non-Default VPC**:
    - User-created, configurable as needed.
    - Instances have private IPv4 addresses by default; public IPv4 requires explicit assignment or subnet configuration.
    - Requires manual setup of internet gateway and Elastic IP for internet access.

* **Internet Access**:
  + **IPv4**: Use a NAT device (gateway/instance) for private subnet instances to initiate outbound connections while blocking unsolicited inbound traffic.
  + **IPv6**: Use an egress-only internet gateway for outbound-only communication.
  + **Requirements for Internet Access**:
    - Attach an internet gateway to the VPC.
    - Ensure route tables point to the internet gateway (0.0.0.0/0 for IPv4, ::/0 for IPv6).
    - Assign public IPv4, Elastic IP, or IPv6 addresses to instances.
    - Configure security groups and network ACLs to allow relevant traffic.

* **Accessing Corporate/Home Networks**:
  + **AWS Managed VPN**: IPsec VPN connection using a virtual private gateway (VPC side) and customer gateway (on-premises).
  + **AWS Transit Gateway**: Scales connectivity across multiple VPCs, accounts, and on-premises networks; supports moving VPN connections from virtual private gateways.
  + **AWS Direct Connect**: Dedicated private connection, combinable with VPN for IPsec encryption.
  + **AWS PrivateLink**: Privately connects VPC to AWS services, other accounts’ services, or AWS Marketplace services without internet gateway, NAT, VPN, or Direct Connect. Traffic stays within the AWS network.
* **AWS PrivateLink-Supported Services**:
  + Includes Amazon EC2 API, Elastic Load Balancing, CloudWatch, CloudTrail, S3, DynamoDB, SNS, SQS, KMS, and more.
  + Supports endpoint services hosted by other AWS accounts and AWS Marketplace partners.

* **VPC Peering**:
  + Connects two VPCs (same or different accounts) for private traffic routing using private IP addresses.
  + CIDR blocks must not overlap.
  + Supports intra- and inter-region peering; AWS PrivateLink endpoints can be accessed across peering connections.
* **VPC Use Case Scenarios**:
  + VPC with a single public subnet.
  + VPC with public and private subnets (NAT).
  + VPC with public/private subnets and AWS managed VPN.
  + VPC with private subnet only and AWS managed VPN.

* **VPC Networking Components**:
  + **Network Interfaces**:
    - Include primary/secondary private IPv4, Elastic IP, public IPv4, IPv6, security groups, MAC address, source/destination check flag, and description.
    - Primary interface (eth0) cannot be detached; additional interfaces can be attached/detached.
  + **Subnets**:
    - Subset of VPC’s CIDR block, specified per AZ.
    - **Types**:
      * **Public Subnet**: Has an internet gateway.
      * **Private Subnet**: No internet gateway.
      * **VPN-Only Subnet**: Uses a virtual private gateway.
    - **CIDR Rules**:
      * IPv4 CIDR size: /16 (65,536 IPs) to /28 (16 IPs).
      * First four and last IP addresses are reserved (unusable).
      * Example: if CIDR block 10.0.0.0/24, then reserved IP addresses are:
        + 10.0.0.0 – Network Address
        + 10.0.0.1 – reserved by AWS for the VPC router
        + 10.0.0.2 – reserved by AWS for mapping to Amazon-provided DNS
        + 10.0.0.3 – reserved by AWS for future use
        + 10.0.0.255 – Network Broadcast Address. AWS does not support broadcast in a VPC,
        + therefore the address is reserved
      * CIDR blocks cannot overlap with existing VPC CIDR blocks.
      * Cannot resize existing CIDR blocks; primary CIDR cannot be disassociated.
    - **Routing**: Local route added automatically for VPC CIDR block; limits apply to CIDR blocks and route table entries.
    - **Peering Constraints**: No overlapping CIDRs; pending peering requests restrict CIDR additions.
  + **Subnet Routing**:
    - Each subnet is associated with one route table, controlling outbound traffic.
    - Default association with the main route table; can be changed or customized.
    - Uses longest prefix match for routing decisions.
    - NAT gateway/instance or egress-only internet gateway enables outbound-only internet access.
  + **Route Tables**:
    - Define traffic routing rules; one per subnet, multiple subnets per table.
    - Main route table cannot be deleted but can be replaced.
    - Updates required for gateways/connections.
  + **Internet Gateways**:
    - Enable internet communication,
    - no bandwidth/availability constraints.
    - Perform NAT for instances with public IPv4 addresses.
    - Allows resources (e.g., EC2 instances) in a VPC connect to the Internet
    - It scales horizontally and is highly available and redundant
    - Must be created separately from a VPC
    - One VPC can only be attached to one IGW and vice versa
    - Internet Gateways on their own do not allow Internet access…
    - Route tables must also be edited!
  + **Egress-Only Internet Gateways**:
    - Allow IPv6 outbound traffic, block inbound; stateful, no security group association.
  + **DHCP Options Sets**:
    - Provide DNS servers/domain names for instances.
    - Default set includes AmazonProvidedDNS; cannot modify, but can replace or remove.
  + **DNS**:
    - Default VPC: Instances get public/private DNS hostnames.
    - Non-Default VPC: Private DNS hostname; public DNS requires enableDnsHostnames and enableDnsSupport set to true.
    - Custom DNS in Route 53 private hosted zones requires both attributes enabled.
* **VPC Peering**:
  + Enables private communication between VPCs using private IPs.
  + Supports intra- and inter-region connections; no overlapping CIDRs.
  + AWS PrivateLink endpoints accessible across peering.
* **Elastic IP Addresses**:
  + Static public IPv4 addresses, associated with instances or network interfaces.
  + Mask instance failures by remapping; remain with account until released.
  + Charges apply for unused or stopped instances; default limit of five per region.
* **VPC Endpoints**:
  + Privately connect to AWS services or PrivateLink-powered services without internet gateway, NAT, VPN, or Direct Connect.
  + **Interface Endpoints**:
    - Elastic network interface with private IP, supports IPv4 TCP traffic.
    - One subnet per AZ, region-specific, accessible via VPN, Direct Connect, or peering.
    - Supports endpoint policies for access control.
  + **Gateway Endpoints**:
    - Route table target for AWS services (e.g., S3, DynamoDB), IPv4 only, region-specific.
    - Multiple endpoints per VPC/service; modifiable policies and route tables.
  + **Endpoint Services**: Create your own PrivateLink-powered service for other AWS principals.
* **ClassicLink**:
  + Links EC2-Classic instances to a VPC for private communication using VPC security groups (region-specific).
* **VPN Connections**:
  + **AWS Managed VPN**: IPsec VPN with virtual private gateway and customer gateway; two tunnels for redundancy.
  + **AWS VPN CloudHub**: Connects multiple remote networks via virtual private gateway.
  + **Third-Party VPN Appliance**: EC2 instance running partner/open-source VPN software.
  + **Routing**: Supports static or dynamic (BGP) routing; longest prefix match applies.
  + **ASN**: Default 64512, user-specifiable, non-modifiable post-creation.
* **VPC Traffic Mirroring**:
  + Replicates EC2 instance network traffic for security, monitoring, or troubleshooting.
  + Supports Nitro and non-Nitro instances.
* **Amazon VPC Pricing**:
  + Charges for:
    - VPN Connection-hour.
    - NAT Gateway-hour and data processing (per GB).
    - Standard AWS data transfer for NAT gateway traffic.
    - Unused/inactive Elastic IPs.

# Validate Your Knowledge

1. **An Amazon VPC has EC2 instances in a private subnet connecting to an S3 bucket via a VPC Gateway Endpoint without an Internet Gateway. The security group allows traffic to/from S3 endpoint public IP ranges (from aws ec2 describe-prefix-lists), but requests are timing out. What is the most suitable solution with least effort?**
   * **Answer**: Configure the outbound VPC security group rule to use the AWS-managed prefix-list in connecting to the Gateway VPC endpoint.
     + **Explanation**: Gateway VPC endpoints use private IP addresses within the VPC, not public IPs. Using the AWS-managed prefix-list ensures the security group references the correct private IP ranges for the S3 endpoint, resolving the issue without additional infrastructure.
2. **A company with IPv6-enabled VPCs cannot launch a new EC2 instance due to exhausted IPv4 CIDR blocks. How should the Solutions Architect resolve this while ensuring scalability?**
   * **Answer**: Set up a new IPv6-only subnet with a large CIDR range. Associate the new subnet with the VPC then launch the instance.
     + **Explanation**: Since IPv4 CIDRs are nearing exhaustion, creating an IPv6-only subnet leverages available IPv6 addresses, supports scalability, and avoids the need to modify existing IPv4 configurations or remove IPv4 support, which could disrupt applications.

## Amazon VPC References:

<https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html><https://aws.amazon.com/vpc/details/><https://aws.amazon.com/vpc/pricing/><https://aws.amazon.com/vpc/faqs/>

## **Elastic Network Interfaces - ENI**

* Logical component in a VPC that represents a virtual network card
* An ENI can have the following attributes:
  + Primary private IPv4 address, one or more secondary IPv4 addresses
  + One Elastic IP (IPv4) per private IPv4
  + One Public IPv4
* ENI instances can be created independently from an EC2 instance
* We can attach them on the fly to an EC2 instances or move them from one to another (useful for failover)
* ENIs are bound to a specific available zone
* ENIs can have security group attached to them
* EC2 instances usually have a primary ENI (eth0). In case we attach a secondary ENI, eth1 interface will be available. The primary ENI can not be detached.

## **Public IP**

## Any IP other than Private IP

## **Private IP**

## Private IP does not change if EC2 Instance is restarted (stop and then start)

## Big Network Range 10.0.0.0 - 10.255.255.255 and CIDR (10.0.0.0/8)

## AWS Default Range 172.16.0.0 - 172.31.255.255 and CIDR (172.16.0.0/12)

## Home Network Range 192.168.0.0 - 192.168.255.255 and CIDR (192.168.0.0/16)

## **Elastic IP**

## Does not change if EC2 Instance is restarted (stop and then start) or terminated

## Max 5 Elastic IP can be used in one region

## We can bring our own IP as Elastic IP

## Elastic IP is charge, even if it is not being used

## Elastic IP

* When an EC2 instance is stopped and restarted, it may change its public IP address
* In case there is a need for a fixed IP for the instance, Elastic IP is the solution
* An Elastic IP is a public IP the user owns as long as the IP is not deleted by the owner
* With Elastic IP address, we can mask the failure of an instance by rapidly remapping the address to another instance
* AWS provides a limited number of 5 Elastic IPs (soft limit)
* Overall it is recommended to avoid using Elastic IP, because:
  + They often reflect pool arhcitectural decisions
  + Instead, us e a random public IP and register a DNS name to it

Bastion Hosts (Jump Box)

* A Bastion Host (Jump Box) is a specialized EC2 instance in an Amazon VPC designed to provide secure, controlled access to resources in private subnets.
* Acts as an intermediary between external users (e.g., administrators) and private EC2 instances without public IPs.
* Enhances security by limiting exposure, centralizing access management, and enabling monitoring of administrative traffic.
* We can use a Bastion Host to SSH into our private EC2 instances
* The bastion is in the public subnet which is then connected to all other private subnets
* Bastion Host security group must allow inbound from the internet on por t 22 from restricted CIDR, for example the public CIDR of your corporation
* Security Group of the EC2 Instances must allow the Security Group of the  Bastion Host, or the private IP of the Bastion host
* **Key Concepts**:
  + **Purpose**:
    - Facilitates secure SSH (Linux, port 22) or RDP (Windows, port 3389) connections to private subnet instances.
    - Acts as a gateway for resources not directly reachable from the internet.
    - Reduces attack surface by ensuring private instances lack public IPs.
  + **Placement**:
    - Deployed in a public subnet with a public IP or Elastic IP for external connectivity.
    - Private EC2 instances reside in private subnets, accessible only via the Bastion Host.
  + **Security Groups**:
    - **Bastion Host Security Group**:
      * **Inbound**: Allows SSH (port 22) or RDP (port 3389) only from trusted CIDR ranges (e.g., corporate office IP, not 0.0.0.0/0).
      * **Outbound**: Permits SSH/RDP to private subnet instances.
    - **Private EC2 Instance Security Group**:
      * Blocks direct internet access (no inbound rules from 0.0.0.0/0).
      * Allows inbound SSH/RDP from the Bastion Host’s security group or private IP.
  + **Workflow**:
    - Users connect to the Bastion Host via SSH/RDP using its public IP.
    - From the Bastion Host, users SSH/RDP to private instances using private IPs.
    - Ensures private instances remain isolated from external access.
* **Best Practices**:
  + Restrict Bastion Host inbound access to specific, trusted IP ranges (e.g., corporate or VPN IPs).
  + Use an Elastic IP for consistent addressing, especially for high-availability setups.
  + Enable Multi-Factor Authentication (MFA) for Bastion Host access.
  + Harden the Bastion Host with security patches, minimal OS configurations, and disabled unnecessary services.
  + Consider AWS Systems Manager (SSM) Session Manager as a more secure alternative, avoiding public exposure and using IAM-based access.
  + Enable logging (e.g., CloudTrail, VPC Flow Logs) to monitor activity.
  + Deploy Bastion Hosts in multiple AZs with Auto Scaling for high availability.
  + Use IAM roles instead of storing SSH keys on the Bastion Host.
* **Pros and Cons**:
  + **Pros**:
    - Provides secure access to private instances without internet exposure.
    - Centralizes administrative access for simplified management and auditing.
    - Enables monitoring through a single entry point.
  + **Cons**:
    - Single point of failure; unavailability cuts off access to private instances.
    - Incurs EC2 instance and Elastic IP costs.
    - Public exposure increases risk if not properly secured.
* **Related Concepts**:
  + **AWS Systems Manager Session Manager**:
    - Secure alternative to Bastion Hosts, uses IAM policies, no public IPs or open SSH/RDP ports.
    - Connects via HTTPS (port 443), logs sessions to CloudWatch Logs or S3.
  + **VPC Configuration**:
    - Requires public/private subnets, internet gateway for public subnet, and route tables.
    - Private subnet instances use NAT gateways/instances for outbound internet access (not provided by Bastion Host).
  + **Security Enhancements**:
    - **VPC Flow Logs**: Monitor Bastion Host traffic for security and troubleshooting.
    - **AWS Network Firewall**: Filters traffic to the Bastion Host.
    - **IAM Policies**: Restrict access to specific users/roles.
  + **High Availability**:
    - Deploy across multiple AZs with Auto Scaling.
    - Optionally use Elastic Load Balancer (ELB) for SSH/RDP traffic distribution (less common).
  + **Monitoring and Logging**:
    - Use CloudTrail for API call auditing.
    - Use CloudWatch for instance health/performance metrics.
    - Log SSH/RDP sessions via custom scripts or SSM Session Manager.
* **Use Case Scenarios**:
  + **Secure Remote Administration**: Access private EC2 instances for maintenance or troubleshooting.
  + **DevOps Workflows**: Provide developers secure access to private development/staging environments.
  + **Hybrid Environments**: Connect on-premises administrators to AWS via VPN/Direct Connect through the Bastion Host.
  + **Compliance Requirements**: Centralize access for auditing in regulated industries (e.g., finance, healthcare).
* **Comparison with Alternatives**:
  + **Bastion Host vs. SSM Session Manager**:
    - **Bastion Host**: Requires public IP, open ports, SSH key management; higher maintenance.
    - **SSM Session Manager**: No public IP/ports, IAM-based, browser/CLI sessions, integrated logging, no EC2 costs.
  + **Bastion Host vs. VPN**: VPNs offer broader network access; Bastion Hosts are specific to SSH/RDP.
  + **Bastion Host vs. AWS Client VPN**: Client VPN provides broader endpoint access; Bastion Hosts are simpler for instance-specific access.
* **Implementation Notes**:
  + Use small EC2 instance types (e.g., t3.micro) for cost efficiency.
  + Configure automatic security updates and minimal software to reduce vulnerabilities.
  + Regularly rotate SSH keys and review security group rules.
  + Consider AWS Marketplace AMIs for pre-hardened Bastion Host images.
* **Related AWS Services**:
  + **AWS Systems Manager**: For Session Manager.
  + **Amazon CloudWatch**: For monitoring and logging.
  + **AWS CloudTrail**: For API auditing.
  + **AWS Network Firewall**: For traffic filtering.
  + **Elastic Load Balancer**: For high-availability setups.
  + **AWS Transit Gateway**: For hybrid connectivity with on-premises networks.
* **Pricing**:
  + EC2 instance usage charges (e.g., hourly rate for t3.micro).
  + Elastic IP charges for unattached or stopped instances.
  + Data transfer costs for Bastion Host traffic.
  + No additional costs for SSM Session Manager, though CloudWatch/S3 logging may apply.

# Validate Your Knowledge

* **Question**: A company has a VPC with private EC2 instances requiring secure administrative access from an on-premises network. The Solutions Architect plans to use a Bastion Host. Which configuration ensures the highest security for the Bastion Host setup?
  + **Options**:
    - Deploy the Bastion Host in a private subnet, assign a public IP, and allow SSH from 0.0.0.0/0.
    - Deploy the Bastion Host in a public subnet, assign an Elastic IP, restrict SSH to the corporate office IP range, and allow outbound SSH to private instances.
    - Deploy the Bastion Host in a public subnet, allow SSH from all IPs (0.0.0.0/0), and use IAM roles for access control.
    - Use AWS Systems Manager Session Manager instead of a Bastion Host, with no public IP or open ports.
  + **Answer**: 2. Deploy the Bastion Host in a public subnet, assign an Elastic IP, restrict SSH to the corporate office IP range, and allow outbound SSH to private instances.
  + **Explanation**:
    - Option 2 follows best practices by placing the Bastion Host in a public subnet with an Elastic IP for consistent addressing, restricting inbound SSH to a trusted IP range (e.g., corporate office), and allowing outbound SSH to private instances, minimizing the attack surface.
    - **Option 1**: Incorrect; Bastion Hosts in private subnets cannot receive external SSH without complex NAT setups.
    - **Option 3**: Insecure due to allowing SSH from all IPs (0.0.0.0/0); IAM roles do not replace IP-based restrictions for SSH.
    - **Option 4**: SSM Session Manager is more secure but does not address the Bastion Host setup as requested.

**AWS Security Group vs. Network ACL**

* Both **Security Groups** and **Network ACLs (NACLs)** are firewalls in AWS VPC networking, controlling inbound and outbound traffic.
* Operate at different layers:
  + Security Groups at the instance level,
  + NACLs at the subnet level.
* Both are region-scoped but NACLs have AZ-specific application due to subnet association.
* **Security Groups**:
  + **Definition**: Fundamental network security mechanism acting as a firewall for EC2 instances or services using network interfaces (ENIs).
  + **Layer**: Instance-level (applies to EC2 instances or ENI-based services).
  + **Rules**: Supports **ALLOW rules only**; no DENY rules.
  + **Stateful**: Return traffic is automatically allowed (e.g., inbound port 80 auto-allows outbound responses).
  + **Default Behavior**:
    - **Inbound**: Denies all traffic unless explicitly allowed.
    - **Outbound**: Allows all traffic by default.
  + **Association**:
    - Attached to network interfaces (ENIs).
    - Up to 5 security groups per instance (expandable to 16).
    - Can be attached to multiple instances.
  + **Regulation**:
    - Controls access to **ports**, **IP ranges** (IPv4/IPv6), and **inbound/outbound traffic**.
    - Can reference another Security Group as a source (no need to specify instance IPs).
  + **Limits**:
    - 60 inbound + 60 outbound rules per Security Group (120 total).
    - IPv4 and IPv6 rules counted separately.
  + **Region-Level**:
    - Defined and operate at the **VPC level** within a specific AWS Region.
    - Apply consistently across all Availability Zones (AZs) in the VPC.
    - Example: A Security Group allowing HTTP (port 80) applies to all instances using it in any AZ in the VPC.
  + **Use Case**:
    - Fine-grained control for EC2 instances or ENI-based services.
    - Examples: Allow HTTP/HTTPS for web servers, SSH (port 22) or RDP (port 3389) from specific IPs.
  + **Key Notes**:
    - Cannot blacklist IPs (use NACLs for blocking).
    - Lives outside EC2 instances; blocked traffic is invisible to instances.
    - Maintain a separate Security Group for SSH access for better management.
    - **Timeouts**: Likely a Security Group issue (traffic blocked).
    - **Connection Refused**: Indicates an application error (traffic passed Security Group).
  + **Rule Structure**:
    - **Type**: Protocol type (e.g., TCP, UDP).
    - **Protocol**: Specific protocol (e.g., HTTP, SSH).
    - **Port Range**: Ports allowed (e.g., 80, 22).
    - **Source Address**: IP range, CIDR, or another Security Group.
    - **Description**: Optional rule description.
* **Network ACLs (NACLs)**:
  + **Definition**: Firewall at the subnet level controlling traffic for all resources in a subnet.
  + **Layer**: Subnet-level (applies to all resources in a subnet).
  + **Rules**: Supports both **ALLOW and DENY rules**.
  + **Stateless**: Return traffic must be explicitly allowed (e.g., inbound port 80 requires an outbound rule).
  + **Default Behavior**:
    - **Default NACL**: Allows all inbound and outbound traffic.
    - **Custom NACL**: Denies all traffic until rules are added.
  + **Association**:
    - One NACL per subnet; a single NACL can be shared across multiple subnets.
  + **Rule Evaluation**:
    - Rules numbered (1–32766); lower numbers have higher precedence.
    - First matching rule applies; implicit final rule (\*) denies unmatched traffic.
    - Example: Rule #100 ALLOW 10.0.0.10/32 precedes Rule #200 DENY 10.0.0.10/32 → Traffic allowed.
    - AWS recommends increments of 100 (e.g., 100, 200, 300) for rule flexibility.
  + **Limits**:
    - 200 NACLs per VPC (soft limit, can be increased).
    - 20 inbound + 20 outbound rules per NACL.
    - IPv4 and IPv6 rules counted separately.
  + **Region-Level with AZ-Specific Application**:
    - Defined at the **VPC level** within a specific AWS Region.
    - Applied to subnets, which are tied to specific AZs, making effects AZ-specific.
    - Example: An NACL allowing SSH (port 22) for a subnet in us-east-1a only affects that subnet’s traffic.
  + **Use Case**:
    - Broad subnet-level firewall policies.
    - Ideal for blocking specific malicious IPs or adding an extra defense layer.
  + **Key Notes**:
    - Requires ephemeral port rules (1024–65535) for NAT Gateway, ELB, or Lambda in custom NACLs.
    - Provides broader control compared to Security Groups.
    - One NACL per subnet, new subnets are assigned the Default NACL
    - You define NACL Rules:
      * Rules have a number (1-32766), higher precedence with a lower number
      * First rule match will drive the decision
      * Example: if you define #100 ALLOW 10.0.0.10/32 and #200 DENY 10.0.0.10/32, the IP address will be allowed because 100 has a higher precedence over 200
      * The last rule is an asterisk (\*) and denies a request in case of no rule match
      * AWS recommends adding rules by increment of 100
    - Newly created NACLs will deny everything
    - NACL are a great way of blocking a specific IP address at the subnet level
    - **Default NACL:**
      * Accepts everything inbound/outbound with the subnets it’s associated with
      * Do NOT modify the Default NACL, instead create custom NACLs
* **Flow Logs**:
  + Capture IP traffic to/from network interfaces; published to CloudWatch Logs or S3.
  + Used to diagnose Security Group/NACL rule issues, monitor traffic, or determine traffic direction.
  + No impact on network performance; does not capture real-time logs or link-local/AWS-reserved IPs.

**Comparison Table**:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Security Group** | **Network ACL** |
| **Layer** | Instance-level (ENI) | Subnet-level |
| **Rules** | ALLOW only | ALLOW and DENY |
| **State** | Stateful (return traffic auto-allowed) | Stateless (explicit rules for return traffic) |
| **Association** | Up to 5 per instance (expandable to 16) | One per subnet, shareable across subnets |
| **Default Behavior** | Inbound: Deny all; Outbound: Allow all | Default: Allow all; Custom: Deny all |
| **Rule Limits** | 60 inbound + 60 outbound (120 total) | 20 inbound + 20 outbound per NACL |
| **Evaluation** | All rules evaluated | Numbered rules, first match applies |
| **Use Case** | Fine-grained instance control | Broad subnet policies, IP blacklisting |
| **Ephemeral Ports** | Not required | Required (1024–65535) for NAT/ELB/Lambda |
| **Scope** | Region-level (VPC-wide) | Region-level, AZ-specific via subnets |

* **Key Differences**:
  + Security Groups are instance-specific, stateful, and simpler for allowing traffic; NACLs are subnet-wide, stateless, and support DENY rules.
  + Security Groups cannot blacklist IPs; NACLs are ideal for this.
  + NACLs require explicit return traffic rules, unlike Security Groups’ automatic handling.
  + Both are region-scoped (tied to VPC), but NACLs apply to AZ-specific subnets.

# Stateful vs. Stateless in AWS

## Overview

In AWS, the terms **stateful** and **stateless** describe how applications or services manage and store data about their state (e.g., user sessions, configurations, or transaction details). The distinction is critical for designing scalable, resilient, and fault-tolerant architectures.

## Stateful

* **Definition**: A stateful application or service retains information (state) about previous interactions or sessions, storing it for future use. The state is typically tied to a specific instance or resource.
* **Characteristics**:
  + **Persistent Data**: Maintains data (e.g., user session details, database records) across requests or interactions.
  + **Instance Dependency**: The same instance or resource must handle subsequent requests to access the stored state.
  + **Examples in AWS**:
    - **Amazon RDS**: Stores database state (tables, records) persistently, requiring the same database instance for consistent access.
    - **Amazon EC2 with Session Persistence**: An EC2 instance storing session data locally (e.g., in-memory cache or local disk) is stateful.
    - **Amazon ElastiCache**: In-memory data stores like Redis or Memcached that maintain session or application state.
  + **Use Cases**:
    - Applications requiring persistent user sessions (e.g., shopping carts in e-commerce).
    - Databases where data consistency and history are critical.
  + **Challenges**:
    - Scaling is complex as state must be preserved or synchronized across instances.
    - Fault tolerance requires mechanisms like replication or backups to avoid data loss.

## Stateless

* **Definition**: A stateless application or service does not retain information about previous interactions. Each request is independent, with no dependency on prior states stored on the server.
* **Characteristics**:
  + **No Persistent State**: Each request contains all necessary information, or state is stored externally (e.g., in a database or client-side).
  + **Instance Independence**: Any instance can handle any request, enabling easy scaling and load balancing.
  + **Examples in AWS**:
    - **AWS Lambda**: Serverless functions that process requests independently without retaining state between invocations.
    - **Amazon API Gateway**: Handles HTTP requests statelessly, routing them to backend services without storing session data.
    - **Amazon S3**: Stores data as objects but processes requests (e.g., GET, PUT) without maintaining state on the server.
    - **Elastic Load Balancer (without sticky sessions)**: Distributes requests across instances without tracking client state.
  + **Use Cases**:
    - Web applications with stateless APIs (e.g., RESTful services).
    - Microservices architectures where each service handles requests independently.
  + **Benefits**:
    - Easier to scale horizontally by adding more instances.
    - Improved fault tolerance as any instance can handle requests without data loss.

## Key Differences

* **State Storage**:
  + **Stateful**: Stores state locally or in a persistent store tied to the application or instance (e.g., EC2 instance memory, RDS database).
  + **Stateless**: Stores state externally (e.g., in a client, external database like DynamoDB, or not at all).
* **Scaling**:
  + **Stateful**: Scaling requires state synchronization (e.g., database replication, session sharing via ElastiCache).
  + **Stateless**: Scales easily by adding instances, as no state is tied to a specific instance.
* **Fault Tolerance**:
  + **Stateful**: Requires backup or replication to prevent data loss if an instance fails.
  + **Stateless**: More resilient since no state is lost if an instance fails; another instance can handle the next request.
* **Complexity**:
  + **Stateful**: More complex to manage due to state persistence and synchronization needs.
  + **Stateless**: Simpler to design and manage, especially in distributed systems.
* **AWS Example Scenario**:
  + **Stateful**: An EC2-based web server storing user session data in memory or a local file, requiring sticky sessions via ELB to route requests to the same instance.
  + **Stateless**: A web application using AWS Lambda and API Gateway, with session data stored in Amazon DynamoDB or client-side cookies, allowing any Lambda instance to process requests.

## Best Practices in AWS

* **Stateful**:
  + Use services like Amazon RDS or ElastiCache for persistent state management.
  + Implement Multi-AZ deployments or replication for high availability.
  + Use Elastic Load Balancer with sticky sessions if state must be maintained on specific instances.
* **Stateless**:
  + Design applications to store state in external services (e.g., DynamoDB, S3) or client-side (e.g., JWT tokens).
  + Leverage serverless services like AWS Lambda and API Gateway for stateless architectures.
  + Use Auto Scaling groups with stateless EC2 instances to handle variable loads efficiently.

**Note**: Many AWS architectures aim for stateless designs to maximize scalability and resilience, offloading state to services like DynamoDB, Redis, or client-side mechanisms when possible.

# AWS VPC Peering

## Definition

* **VPC Peering**: A direct networking connection between two Virtual Private Clouds (VPCs) that enables routing of traffic using private IPv4 or IPv6 addresses, managed entirely within AWS infrastructure without requiring physical hardware or VPN.

## Communication

* **Two-Way Traffic**: Facilitates bidirectional communication between VPCs, allowing them to function as if they are part of the same network.
* **Private Communication**: Traffic remains within AWS’s global private network, avoiding the public internet for enhanced security and performance.

## Use Cases

* **Secure Communication**: Enables secure data exchange between:
  + Different departments
  + Separate AWS accounts
  + Different applications or environments (e.g., Development ↔ Production)

## Traffic Control

* **Route Tables**: Must be updated in each VPC’s subnets to enable routing between peered VPCs.
* **Security Groups and NACLs**: Control traffic at the instance (Security Groups) and subnet (Network ACLs) levels for fine-grained access management.
* **Security Group Referencing**: Security group rules in each VPC can reference resources in the peered VPC, including across accounts in the same Region.

## Cross-Region and Cross-Account Peering

* **Inter-Region Peering**: Supports peering between VPCs in different AWS Regions, known as Inter-Region VPC Peering, allowing secure communication across Regions.
* **Cross-Account Peering**: Enables peering between VPCs in different AWS accounts, supporting collaboration across organizations.

## Non-Transitive Nature

* **No Transitive Peering**: Peering connections are not transitive. For example:
  + If VPC A is peered with VPC B, and VPC B is peered with VPC C, VPC A cannot communicate directly with VPC C without a separate peering connection.

## Configuration Requirements

* **Non-Overlapping CIDR Blocks**: Peered VPCs must have non-overlapping IP address ranges (CIDR blocks) to avoid routing conflicts.
* **Peering Scope**: Can be established between VPCs in:
  + The same or different AWS accounts
  + The same or different AWS Regions
* **IAM Permissions**: Proper IAM permissions are required for cross-account peering setups.
* **Route Table Updates**: Necessary to ensure EC2 instances in peered VPCs can communicate.
* **DNS Resolution**: Optional support allows resources in one VPC to resolve private DNS hostnames in the peered VPC.

## Types of Peering

* **Intra-Region Peering**: Peering between VPCs within the same AWS Region.
* **Inter-Region Peering**: Peering between VPCs in different AWS Regions.

## Performance and Cost

* **Bandwidth and Latency**: Offers high bandwidth and low latency, ideal for distributed applications.
* **Cost**:
  + **Intra-Region**: No data transfer charges for traffic within the same AWS Region.
  + **Inter-Region**: Data transfer charges apply, based on the volume of data transferred (GB) and varying by Region.

**Note**: This response consolidates and removes duplicate concepts from the provided inputs to present a clear and comprehensive overview of AWS VPC Peering.

AWS NAT (Network Address Translation)

* Network Address Translation (NAT) enables instances in a private subnet to initiate outbound internet traffic while preventing unsolicited inbound traffic from the internet.
* **Purpose**: Facilitates secure internet access for resources in private subnets without exposing them directly to the public internet.

**Key Concepts**:

* Translates private IP addresses to public IP addresses for outbound traffic.
* Used to conserve public IP addresses and enhance security by hiding internal network structures.
* Supports protocols like TCP, UDP, and ICMP.

**How NAT Works**:

* Private subnet instances send outbound traffic to the NAT device (Gateway or Instance).
* The NAT device translates the private IP to a public IP (Elastic IP) and forwards the traffic to the internet.
* Responses return to the NAT device, which translates the public IP back to the private IP and routes the traffic to the instance.
* Inbound traffic initiated from the internet is blocked unless explicitly allowed.

**Why Use NAT**:

* Enables private subnet instances to access the internet for updates, patches, or external services without direct exposure.
* Enhances security by preventing external access to private resources.
* Conserves public IP addresses by allowing multiple private instances to share a single public IP.

**Types of NAT in AWS**:

* **NAT Gateway**:
  + Managed service residing in a public subnet, requiring an Elastic IP for operation.
  + Specific to an Availability Zone (AZ) with built-in redundancy for high availability.
  + Supports up to 55,000 simultaneous connections per destination.
  + Uses ports 1024–65535 for communication.
  + Does not support traffic over VPC endpoints, VPN, AWS Direct Connect, or VPC peering.
  + Preferred for scalability and low maintenance, with AWS-managed high bandwidth and no administration required.
  + Pay per hour for usage and bandwidth.
  + Cannot be used by EC2 instances in the same subnet (only from other subnets).
  + Requires an Internet Gateway (IGW) for private subnet access (Private Subnet => NATGW => IGW).
  + Provides 5 Gbps of bandwidth with automatic scaling up to 100 Gbps.
  + No Security Groups to manage or require.
  + NAT Gateway is resilient within a single Availability Zone
  + Must create multiple NAT Gateways in multiple AZs for fault-tolerance
  + There is no cross-AZ failover needed because if an AZ goes down it doesn't need NAT

* **NAT Instance**:
  + Less managed alternative using an EC2 instance configured for NAT.
  + Launched in a public subnet with an Elastic IP attached.
  + Requires disabling the EC2 "Source/Destination Check" setting.
  + Route tables must be configured to direct private subnet traffic to the NAT instance.
  + Offers more customization but requires manual management (e.g., patching, scaling).

**Use Cases**:

* Enabling private subnet instances to download software updates or access APIs without public exposure.
* Supporting secure outbound connections for applications in private subnets.
* Facilitating hybrid cloud setups where private resources need controlled internet access.
* Ensuring compliance with security policies by restricting inbound traffic.

**Key Considerations**:

* NAT Gateway is generally preferred for its scalability, high availability, and minimal management overhead.
* NAT Instance is suitable for scenarios requiring customization or cost optimization but demands more administrative effort.
* Proper route table configuration is critical for both NAT Gateway and NAT Instance to function correctly.
* NAT devices do not support inbound connections initiated from the internet, ensuring private subnet security.

## [AWS Transit Gateway](https://tutorialsdojo.com/aws-transit-gateway/)

* AWS Transit Gateway is a managed networking service that acts **as a regional virtual router using a hub-and-spoke model.**
* Connects Amazon Virtual Private Clouds (VPCs), on-premises data centers, remote offices, and other networks through a single central gateway.
* Simplifies management by requiring only one connection from the gateway to each VPC or on-premises network.
* Automatically connects new VPCs to the Transit Gateway, making them accessible to all other connected networks.
* Supports thousands of VPC attachments and scales elastically based on traffic volume.
* Available in multiple AWS Regions, including US East (N. Virginia), US West (Oregon), EU (Ireland), Asia Pacific (Tokyo), and others.
* **Key Benefits**:
  + **Simplified Network Architecture**: Consolidates connections, reducing complexity compared to multiple VPC peering or VPN setups.
  + **Scalability**: Handles high traffic volumes, supports up to 5 Gbps per attachment, and scales beyond single VPN limits via ECMP.
  + **Centralized Control**: Uses route tables for traffic segmentation, propagation, and policy enforcement.
  + **Improved Security**: Integrates with AWS Network Firewall, Gateway Load Balancer, and supports encryption via AWS backbone.
  + **Cost-Effective**: Reduces operational overhead; charged per attachment-hour and data processed.
  + **Redundancy**: Highly available, supports failover with multiple VPN tunnels or Direct Connect links.
* **Features**:
  + **Inter-Region Peering**:
    - Connects Transit Gateways across AWS Regions using the AWS global network for low-latency routing.
    - Ideal for geographic redundancy, data replication, or resource sharing between Regions.
    - Traffic stays on the AWS backbone (encrypted), avoiding the public internet.
    - Supports static routing only for peering attachments.
  + **Multicast**:
    - Enables fine-grained control over multicast traffic production and consumption.
    - Supports Internet Group Management Protocol (IGMPv2) for dynamic group joining/leaving.
    - Scalable for distributing content streams to multiple subscribers.
    - Uses multicast domains; shareable via AWS Resource Access Manager (RAM) across accounts or AWS Organizations.
    - Static members can be added/removed via console, CLI, or SDK.
  + **Automated Provisioning**:
    - Automatically discovers Site-to-Site VPN connections and associated on-premises resources.
    - Manually define on-premises networks using Transit Gateway Network Manager.
    - Integrates with AWS RAM for sharing gateways across accounts.
  + **Equal Cost Multi-Path (ECMP) Routing**:
    - Aggregates multiple VPN or Direct Connect paths for higher bandwidth (up to 1.25 Gbps per VPN tunnel; scalable with multiples).
    - Requires dynamic BGP routing; uses 5-tuple hash for load balancing.
    - Supported for VPN and Direct Connect attachments; not for static routing.
  + **Transit Gateway Connect**:
    - Native integration for SD-WAN appliances using GRE and BGP protocols.
    - Improves bandwidth, dynamic routing, and route limits without multiple IPsec VPNs.
    - Supports higher MTU (up to 8500 bytes) for VPC-to-VPC, Direct Connect, and peering traffic.
  + **Appliance Mode**:
    - Ensures symmetric routing for stateful appliances (e.g., firewalls) by enforcing AZ stickiness.
    - Drops packets if no attachment exists in the source AZ, preventing asymmetric flows.
    - Enabled on VPC attachments for centralized inspection across AZs.
    - Integrates with AWS Gateway Load Balancer and Network Firewall.
  + **Monitoring and Visibility**:
    - Publishes metrics to Amazon CloudWatch (e.g., bandwidth, packet drops) and flow logs to CloudWatch Logs or S3.
    - Transit Gateway Network Manager provides global topology views, events, and performance metrics.
    - Route Analyzer for troubleshooting routing paths.
  + **Routing and Attachments**:
    - Supports IPv4 and IPv6; attachments include VPCs, VPNs, Direct Connect, peering, and Connect peers.
    - Route tables control associations and propagation; default route table included.
    - No CIDR overlap allowed between attachments; longest prefix match for route selection.
  + **Integration**:
    - Works with AWS Direct Connect Gateway for hybrid connectivity across Regions/accounts.
    - Supports public DNS resolution to private IPs for VPC-attached resources.
    - Compatible with AWS Network Firewall, Gateway Load Balancer Endpoints (GWLBE), and third-party appliances.
* **Transit Gateway Components**:
  + **Attachments**: Logical connections to VPCs, VPNs, Direct Connect, or peering; each supports up to 1.25 Gbps (scalable with ECMP).
  + **Route Tables**: Define routing policies; associate attachments and propagate routes.
  + **Multicast Domains**: Group for multicast sources/receivers; supports IGMP and static members.
  + **Peering Attachments**: For inter/intra-Region connections; encrypted over AWS backbone.
* **Use Cases**:
  + **Multi-VPC Connectivity**: Central hub for VPC-to-VPC routing in single or multi-account setups.
  + **Hybrid Cloud**: Connect on-premises via VPN/Direct Connect to multiple VPCs.
  + **Global Networks**: Inter-Region peering for disaster recovery or low-latency access.
  + **Network Inspection**: Centralized firewalls with Appliance Mode and GWLB.
  + **SD-WAN Extension**: Seamless branch connectivity with Transit Gateway Connect.
  + **Multicast Applications**: Video streaming, financial data distribution.
* **Limitations**:
  + Regional resource; peering required for cross-Region.
  + VPN ECMP requires dynamic routing and ECMP-capable customer gateways.
  + Appliance Mode drops asymmetric traffic; plan AZ coverage.
  + Maximum 5,000 attachments per gateway; route limits apply.
* **Pricing**:
  + Hourly attachment fees (e.g., $0.05/hour per VPC attachment).
  + Data processing: $0.02/GB for intra-Region, $0.05/GB inter-Region.
  + No upfront costs; pay for attachments and processed data.
* **Related Resources**:
  + **AWS Certified Advanced Networking Specialty**: Recommended practice exams and study guide for deeper networking topics.

# Validate Your Knowledge

1. **A multinational bank has two data centers 60 miles apart, a single transit gateway with multiple VPC and VPN attachments, two AWS Direct Connect connections via a partner to a Direct Connect location, and a Direct Connect Gateway connected via transit virtual interface. What other network connections can be implemented? (Select TWO.)**
   * **Answers**:
     + **Connect multiple VPCs in the same or different AWS account using the Direct Connect connection.**
       - **Explanation**: A Direct Connect Gateway associated with the Transit Gateway allows private connectivity to multiple VPCs attached to the Transit Gateway, across the same or different AWS accounts, using the dedicated Direct Connect links.
     + **Associate multiple transit gateways in different AWS Regions to the Direct Connect Gateway and use the same ASNs for each transit gateway. Enable the Appliance mode for all transit gateways.**
       - **Explanation**: Direct Connect Gateways support associations with multiple Transit Gateways across Regions, enabling global hybrid access; same ASNs can be used, and Appliance Mode can be enabled on each for symmetric routing in inspection scenarios.

**AWS Transit Gateway  Reference:**

<https://aws.amazon.com/transit-gateway/>

**Hub and Spoke model**

The Hub and Spoke model in AWS is a network architecture where a central hub (e.g., a Transit Gateway or a shared VPC) connects multiple spokes (other VPCs or on-premises networks) in a star-like topology. The hub routes traffic between spokes, simplifying management and enabling secure, scalable communication.

Why Use It: Centralizes network control, reduces complexity for connecting multiple VPCs, and supports secure, private communication without direct peering between every spoke.

### Analogy

Imagine a city with a central bus station (the hub) connecting various neighborhoods (the spokes). Each neighborhood doesn’t need direct roads to every other neighborhood; instead, they all connect to the bus station, which routes travelers efficiently between them.

### Technical Explanation

* Hub: Typically an AWS Transit Gateway (or a shared VPC in simpler setups) that acts as the central point for routing traffic.
* Spokes: Individual VPCs, on-premises networks, or other AWS resources that connect to the hub.
* **How It Works:**
  + The Transit Gateway (hub) connects to each spoke (VPC or on-premises network) via attachments.
  + Route tables in the Transit Gateway control traffic flow between spokes, allowing transitive routing (e.g., VPC A can reach VPC C via the hub without direct peering).
  + Security groups and network ACLs manage access control.
* **Key Benefits:**
  + Scalability: Easily connect many VPCs without needing multiple VPC Peering connections (e.g., 10 VPCs would need 45 peering connections, but only 10 attachments with a Transit Gateway).
  + Centralized Management: Simplifies routing, monitoring, and security policies at the hub.
  + Hybrid Connectivity: Connects VPCs to on-premises networks via VPN or AWS Direct Connect.
* Example Use Case: A company has 5 VPCs (e.g., for dev, prod, and testing) and an on-premises data center. A Transit Gateway (hub) connects all VPCs and the data center, allowing secure communication without complex peering setups.
* Comparison to VPC Peering: Unlike VPC Peering (point-to-point connections), the hub-and-spoke model avoids the complexity of managing multiple direct connections and supports transitive routing.

Analogy Recap: The bus station (Transit Gateway) connects all neighborhoods (VPCs) efficiently, so they don’t need direct roads (peering) to each other, and the station manages all traffic flow.

# Differences Between NAT Gateway, Transit Gateway, and Internet Gateway in AWS

* **NAT Gateway**:
  + **Purpose**: Enables instances in private subnets to initiate outbound internet traffic while preventing unsolicited inbound traffic.
  + **Key Features**:
    - Managed service residing in a public subnet, requiring an Elastic IP.
    - Translates private IP addresses to public IPs for outbound traffic.
    - Supports TCP, UDP, and ICMP; handles up to 55,000 simultaneous connections per destination.
    - Specific to an Availability Zone (AZ) with built-in redundancy.
    - Does not allow traffic over VPC endpoints, VPN, Direct Connect, or VPC peering.
  + **Use Case**: Allows private subnet instances to access the internet (e.g., for software updates) securely without exposing them to inbound internet traffic.
  + **Scope**: Operates within a single VPC and is focused on internet-bound traffic for private subnets.
  + **Inbound/Outbound**: Only supports outbound-initiated traffic; blocks unsolicited inbound connections.
* **Transit Gateway**:
  + **Purpose**: Simplifies network connectivity by acting as a hub to connect multiple VPCs, on-premises networks, and other AWS services.
  + **Key Features**:
    - Managed service that centralizes network routing across VPCs, Direct Connect, VPNs, and peering connections.
    - Supports transitive routing, allowing resources in different VPCs or networks to communicate without direct peering.
    - Scales to handle thousands of connections and high traffic volumes.
    - Supports attachments like VPCs, VPNs, Direct Connect Gateways, and Transit Gateway Connect for SD-WAN.
    - Configurable via route tables for fine-grained traffic control.
  + **Use Case**: Connects multiple VPCs and on-premises networks in a hub-and-spoke model for centralized management (e.g., enterprise network architectures).
  + **Scope**: Operates across multiple VPCs, regions (via inter-region peering), and hybrid environments.
  + **Inbound/Outbound**: Facilitates both inbound and outbound traffic based on routing rules, not inherently tied to internet access.
* **Internet Gateway**:
  + **Purpose**: Provides a direct connection between a VPC and the public internet, allowing resources to be publicly accessible.
  + **Key Features**:
    - Managed service attached to a VPC, enabling bidirectional internet traffic.
    - Requires association with a VPC and configuration of route tables to direct traffic to the internet.
    - Allows instances with public IPs or Elastic IPs in public subnets to communicate directly with the internet.
    - Does not perform address translation; instances need public IPs for direct internet access.
    - Supports all protocols allowed by security groups and network ACLs.
  + **Use Case**: Hosts public-facing applications (e.g., web servers) in public subnets that need direct internet access.
  + **Scope**: Operates within a single VPC and is focused on enabling direct internet connectivity.
  + **Inbound/Outbound**: Supports both inbound and outbound traffic for instances with public IPs.
* **Key Differences**:
  + **Functionality**:
    - NAT Gateway: Translates private IPs for outbound internet access, protecting private subnets.
    - Transit Gateway: Centralizes routing across VPCs and hybrid networks, not focused on internet access.
    - Internet Gateway: Enables direct, bidirectional internet access for public subnets.
  + **Scope**:
    - NAT Gateway: Single VPC, private subnet internet access.
    - Transit Gateway: Multiple VPCs, regions, and on-premises networks.
    - Internet Gateway: Single VPC, public subnet internet access.
  + **Traffic Direction**:
    - NAT Gateway: Outbound-only for private subnets.
    - Transit Gateway: Bidirectional, based on routing configuration.
    - Internet Gateway: Bidirectional for public subnets.
  + **Management**:
    - NAT Gateway: Fully managed, minimal configuration.
    - Transit Gateway: Managed but requires route table configuration for complex networks.
    - Internet Gateway: Fully managed, simple attachment to VPC.
  + **Use Case Focus**:
    - NAT Gateway: Secure internet access for private resources.
    - Transit Gateway: Network scalability and interconnectivity.
    - Internet Gateway: Public internet exposure for resources.

### **AWS Endpoint**

* **Definition**: An AWS Endpoint is a specific URL or network address used to access an AWS service or resource within a particular region.
* **Purpose**: Enables secure and direct communication between clients (applications, users, or services) and AWS services like S3, EC2, or DynamoDB.

## Types of Endpoints

* **Public Endpoints**: Accessible over the internet for general access to AWS services (e.g., s3.us-east-1.amazonaws.com, ec2.us-west-2.amazonaws.com).
* **VPC Endpoints**: Private endpoints within a Virtual Private Cloud (VPC) for secure access to AWS services without traversing the public internet, powered by AWS PrivateLink.
* **Regional Endpoints**: Specific to AWS regions (e.g., us-east-1, eu-west-1) to reduce latency and ensure data residency.

## Benefits

* **Enhanced Security**: Limits exposure to the public internet, improving security for sensitive operations.
* **Reduced Latency**: Optimized routing within AWS infrastructure for faster communication.
* **Simplified Network Management**: Streamlines connectivity within VPCs without requiring additional gateways.

## Configuration and Use Cases

* **Configuration**: Configurable via AWS Management Console, CLI, or SDKs to route traffic to specific services or regions.
* **Use Cases**:
  + Accessing S3 buckets privately from a VPC.
  + Connecting to AWS APIs securely for automation or application integration.
  + Ensuring compliance with data residency requirements using regional endpoints.

## VPC Endpoints (AWS PrivateLink)

* **Functionality**: Allows connection to AWS services using a private network instead of the public internet, despite services having public URLs.
* **Characteristics**: Redundant and horizontally scalable, eliminating the need for Internet Gateway (IGW), NAT Gateway (NATGW), or similar setups.
* **Troubleshooting**:
  + Verify DNS resolution settings in the VPC.
  + Check route tables for proper configuration.

## Types of VPC Endpoints

* **Interface Endpoints** (Powered by AWS PrivateLink):
  + Provisions an Elastic Network Interface (ENI) with a private IP address as an entry point.
  + Requires attachment of a Security Group.
  + Supports most AWS services.
  + **Cost**: Charged per hour and per GB of data processed.
* **Gateway Endpoints**:
  + Provisions a gateway that must be added as a target in a VPC route table.
  + Does not use security groups.
  + Supports S3 and DynamoDB only.
  + **Cost**: Free.

## Choosing Between Gateway and Interface Endpoints for S3

* **Gateway Endpoints**:
  + Preferred in most scenarios, especially for cost efficiency (free).
  + Ideal for standard S3 access within a VPC.
* **Interface Endpoints**:
  + Preferred when access is required from on-premises (via Site-to-Site VPN or Direct Connect), a different VPC, or a different region.
  + Incur costs based on usage.

**AWS VPN**

# VPN

* A Virtual Private Network (VPN) is a technology that creates a secure, encrypted connection over a less secure network, such as the public internet, allowing users to send and receive data as if their devices were directly connected to a private network.
* It extends a private network across public networks, enabling remote access to resources while maintaining privacy and security.
* VPNs mask your IP address, making your online actions harder to trace, and encrypt your internet traffic to protect against eavesdropping.
* They are commonly used for secure remote work, accessing geo-restricted content, and enhancing privacy on public Wi-Fi.

# AWS VPN

* AWS VPN is a managed service provided by Amazon Web Services that establishes secure and private tunnels between your network or device and the AWS Cloud.
* It consists of two primary services: AWS Site-to-Site VPN and AWS Client VPN, which enable connectivity to AWS resources like VPCs or on-premises networks.
* AWS VPN allows for scalable, highly available connections, integrating with other AWS services for hybrid cloud setups.
* It provides encrypted communication over the public internet, ensuring data security for remote access or network integration.

# What is Difference between VPN and AWS VPN

* A general VPN is any virtual private network service or software that can be provided by various vendors, used for personal privacy, bypassing restrictions, or secure connections anywhere, often running on consumer devices or servers.
* AWS VPN is specifically AWS's cloud-managed VPN solution, designed for integrating on-premises networks or remote users with AWS infrastructure, offering enterprise-grade features like high availability and scalability within the AWS ecosystem.
* General VPNs may use various protocols and can be self-hosted or third-party, while AWS VPN uses managed IPsec for Site-to-Site and OpenVPN for Client VPN, with AWS handling the infrastructure.
* AWS VPN focuses on cloud connectivity with features like automatic failover and integration with VPCs, whereas general VPNs are more versatile for everyday internet use but may lack cloud-specific optimizations.

# How VPN Work and How AWS VPN

* **How a General VPN Works**: A VPN routes your device's internet traffic through a remote VPN server, encrypting the data in a secure tunnel using protocols like IPsec, OpenVPN, or WireGuard; this hides your IP address, prevents ISP monitoring, and allows access as if you're at the server's location.
* The encryption scrambles data so only authorized parties can read it, and the tunnel protects against interception on public networks.
* **How AWS VPN Works**: AWS VPN operates similarly by creating encrypted tunnels but is managed by AWS; for Site-to-Site, it uses IPsec to connect networks with dual tunnels for redundancy, while Client VPN uses OpenVPN for user sessions to access AWS resources.
* It integrates with AWS VPCs, using gateways or endpoints to handle traffic routing, authorization, and logging, providing seamless hybrid connectivity.

# AWS VPN Type

## AWS Site-to-Site VPN

* **How It Works**: Establishes an IPsec-encrypted tunnel between a VPC (via a virtual private gateway or transit gateway) and a remote network, using two redundant tunnels for automatic failover. The customer configures their on-premises gateway.
* **When to Use**: Suitable for connecting entire networks, such as on-premises data centers to AWS VPCs, for secure and reliable integration.
* **Use Cases**: Hybrid cloud setups, disaster recovery, or extending corporate networks to AWS for data transfer and application hosting.

## AWS Client VPN

* **How It Works**: Users connect to a managed endpoint using an OpenVPN client, establishing a TLS-encrypted session. Traffic is routed through associated VPC subnets with authorization rules and logging.
* **When to Use**: Ideal for remote individual users needing secure access to AWS or on-premises resources from anywhere.
* **Use Cases**: Remote workforce accessing cloud applications, developers connecting to VPC resources, or secure access for contractors.

## AWS VPN CloudHub

* **How It Works**: Builds on Site-to-Site VPN by connecting multiple remote networks through a single virtual private gateway, enabling communication between them via AWS as a hub.
* **When to Use**: Appropriate when multiple branch offices or remote sites need to interconnect securely through AWS without direct links.
* **Use Cases**: Distributed organizations with multiple locations requiring shared access to central resources or inter-site communication.

## AWS Direct Connect

* **How It Works**: Provides a dedicated physical network connection between an on-premises data center and AWS, bypassing the public internet for higher bandwidth, lower latency, and consistent performance. It uses a virtual interface to connect to VPCs or other AWS services.
* **When to Use**: Ideal for scenarios requiring high-throughput, low-latency, or consistent network performance between on-premises infrastructure and AWS.
* **Use Cases**: Large-scale data transfers, real-time application hosting, or hybrid cloud workloads requiring stable, high-bandwidth connections.

## Key Differences

* **Site-to-Site VPN vs. Client VPN**: Site-to-Site VPN connects entire networks using IPsec with failover, ideal for infrastructure integration, while Client VPN is user-focused, using OpenVPN for individual, mobile access with per-user authorization.
* **Site-to-Site VPN vs. CloudHub**: CloudHub extends Site-to-Site VPN for a hub-and-spoke model, enabling inter-network traffic between multiple sites, unlike the point-to-point nature of standard Site-to-Site VPN.
* **Site-to-Site/CloudHub vs. Direct Connect**: Site-to-Site VPN and CloudHub use encrypted tunnels over the internet for flexibility, while Direct Connect provides a dedicated, private physical connection for higher throughput and consistent performance.
* **Client VPN vs. Others**: Client VPN emphasizes individual user access with mobility, whereas Site-to-Site VPN, CloudHub, and Direct Connect focus on network-level connectivity with higher throughput needs.

**AWS Elastic Load Balancing**

* Distribute network traffic to improve application scalability
* In AWS, a Load Balancer distributes incoming network traffic across multiple targets (like EC2 instances, containers, or IP addresses).
* **It ensures:**
  + Smooth performance
  + No single server is overloaded
  + High availability

## Benefits:

### Security: **Secure your applications with SSL/TLS termination, integrated certificate management, and client certificate authentication.**

### Automatic scaling: **Deliver applications with high availability and automatic scaling.**

### Monitor in real time:**Monitor the health and performance of your applications in real time, uncover bottlenecks, and maintain SLA compliance.**

## How it works:

* + Elastic Load Balancing (ELB) automatically distributes incoming application traffic across multiple targets and virtual appliances in one or more Availability Zones (AZs).

## Why Do We Use a Load Balancer:

* To handle heavy traffic efficiently
* To prevent server crashes when one instance is overloaded
* To achieve high availability (if one server fails, traffic goes to others)
* To improve scalability (easily add/remove servers)
* To provide fault tolerance and better performance

## Use cases:

### Modernize application with serverless and containers: **Scale modern applications to meet demand without complex configuration or API gateways** [Learn more](https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html)

### Improve hybrid cloud network scalability : **Load balance across AWS and on-premises resources using single load balancer** [Learn more](https://docs.aws.amazon.com/vpc/latest/privatelink/privatelink-share-your-services.html)

### Retain your existing network appliances: **Deploy network appliances from your preferred vendor while taking advantage of the scale and flexibility of the cloud** [Learn more](https://aws.amazon.com/elasticloadbalancing/gateway-load-balancer/)

* **Types of AWS Load Balancers**
  + AWS offers Elastic Load Balancing (ELB) with three main types:

* Classic Load Balancer (CLB)
  + **Layer:** 4 & 7
  + **Purpose:** Basic load balancing for HTTP/HTTPS and TCP traffic
  + **Key Features:**
* Distributes traffic across multiple EC2 instances
* Supports both **HTTP/HTTPS** and **TCP**
* Provides **basic health checks**
* Can use **sticky sessions** (same user goes to same instance)
* Supports **SSL termination**
* **Limitations**
  + No advanced routing (can’t route by URL, host, or header)
  + Limited metrics and flexibility
  + Old — AWS recommends ALB or NLB for new setups
* **Use Case**
  + Legacy EC2-based applications created before 2016

* **Application Load Balancer (ALB):**
  + **Layer:** 7 (Application Layer)
  + **Best for:** Web applications (HTTP/HTTPS traffic)
  + **Key Features:**
    - Routes requests based on:
      * **URL path** (/api, /images)
      * **Hostname** ([app.example.com](http://app.example.com))
      * **Headers / Query strings**
    - Works perfectly with **microservices & containers (ECS, EKS)**
    - Supports **WebSockets, HTTP/2, and WAF (Web Application Firewall)**
    - Can perform **redirects** (HTTP → HTTPS)
  + **Use Case Example:**
    - A website where:
      * /api/\* → backend servers
      * /images/\* → image storage servers

* **Network Load Balancer (NLB)**
  + **Layer: 4 (Transport Layer)**
  + **Best for:** High-performance, low-latency applications
  + **Key Features:**
* Handles millions of requests per second
* Works with TCP, UDP, TLS traffic
* Provides static IP addresses
* Very fast and efficient for real-time apps
* **Use Case Example:**
  + Real-time gaming servers, financial apps, or IoT services needing low latency

* **Gateway Load Balancer (GWLB)**
  + **Layer: Combination of Layer 3 (Network) & Layer 4 (Transport)**
  + **Best for:** Deploying virtual appliances (firewalls, security inspection, etc.)
  + **Key Features:**
    - Integrates with **third-party security tools**
    - Routes traffic through **virtual firewalls, intrusion detection systems, etc.**
    - Ensures **scalable and fault-tolerant** network inspection
  + **Use Case Example:**
    - When you want all inbound/outbound traffic to pass through a security appliance (e.g., firewall, antivirus proxy)

* **Common Features Across All ELBs**
  + Automatic Scaling with EC2 Auto Scaling groups
  + Health Checks — send traffic only to healthy instances
  + Cross-Zone Load Balancing — even distribution across AZs
  + Monitoring & Logging — integrated with CloudWatch
  + SSL/TLS Termination — for secure HTTPS

**AWS Target Group**

* A **Target Group (TG)** in AWS is a **collection of targets (servers, containers, IPs, Lambda functions, etc.)** that a **Load Balancer** sends traffic to.
* **Purpose:** Define where the Load Balancer sends requests
* **Used By:** ALB, NLB, GWLB
* **Target Types:** Instance, IP, Lambda, ALB
* **Main Job:** Route, Health Check, Stickiness, Scaling
* **Key Attributes:** Deregistration delay, Stickiness, Slow start
* **Health Check:** Keeps only healthy targets active
* **Supports Weighted Routing:** Yes, for gradual deployment
* **Does CLB use it?:** No (legacy type)

* **Why Target Groups Exist (Purpose)**
  + To organize backend resources the Load Balancer routes to.
  + To allow dynamic scaling — targets can be added or removed automatically.
  + To perform health checks on each target.
  + To enable different routing rules for different parts of an app.
  + **Example:**
    - **/api/\* → API Target Group**
    - **/images/\* → Image Target Group**
  + Each group has its own servers, ports, and health rules.

* **How Target Groups Work (Step-by-Step)**
  + User sends a request to the Load Balancer (e.g., myapp.com).
  + Load Balancer rule decides *which target group* to use based on path, host, port, etc.
  + Target Group forwards the request to one of its healthy targets (e.g., EC2 instance, container).
  + Response is sent back through the Load Balancer to the user.
  + Health checks keep running periodically to ensure only healthy targets receive traffic.

* **Key Features of Target Groups**
  + **Dynamic Registration:** Automatically adds/removes targets via Auto Scaling
  + **Health Checks:** Monitors and routes only to healthy targets
  + **Stickiness (Session Affinity):** Sends the same user to the same target for session consistency
  + **Weighted Target Groups:** Distribute traffic by percentage (used for blue/green or canary deployments)
  + **Cross-Zone Load Balancing:** Distribute traffic across all Availability Zones
  + **Target Attributes:** Configure advanced options like deregistration delay, slow-start mode, etc.

* **Types of Targets You Can Register**
  + **Instance**
    - Directly register EC2 instances
    - Simple web app on EC2
  + **IP Address**
    - Register IP addresses (private or public)
    - Apps outside VPC or on-prem
  + **Lambda Function**
    - Register AWS Lambda as target
    - Serverless backend APIs
  + **Application Load Balancer**
    - Route traffic from one ALB to another
    - Complex app architectures (multi-tier)

* **Types of Target Groups (Based on Load Balancer Type)**
  + **Classic Load Balancer (CLB)**
    - Doesn’t use target groups (older style)
    - Legacy apps only
  + **Application Load Balancer (ALB)**
    - Instance, IP, Lambda
    - /api → EC2, /upload → Lambda
  + **Network Load Balancer (NLB)**
    - Instance, IP, ALB
    - Real-time financial app
  + **Gateway Load Balancer (GWLB)**
    - IP
    - Security appliances, Firewalls

* **Target Group Protocols and Ports**
  + You define protocol (HTTP, HTTPS, TCP, UDP, TLS) and port(e.g., 80, 443) for communication.
  + **Example:**
    - ALB → Target Group → EC2 instances on port 8080
    - NLB → Target Group → Database cluster on port 3306

* **Health Checks in Target Groups**
  + Each target group has health check settings to monitor targets’ status.
  + **Purpose:** To ensure Load Balancer only sends traffic to healthy targets.
  + **Configurable Parameters:**
    - Protocol (HTTP, HTTPS, TCP)
    - Health check path (/health, /status)
    - Interval & timeout
    - Success codes (e.g., 200–299)
    - Unhealthy threshold (how many failed checks before marking target unhealthy)
  + **Example:**
    - ALB health checks /health every 30 seconds.
    - If 3 checks fail → target is removed from service.

* **Target Group Attributes**
  + **Deregistration Delay:** Time to wait before stopping traffic to a target being removed — Default: 300 seconds
  + **Slow Start Mode:** Gradually increases traffic to new targets — Prevents overload on startup
  + **Stickiness (Session Affinity):** Keeps user connected to the same target — Used for shopping carts or login sessions
  + **Proxy Protocol v2:** Passes client IP info to backend (NLB) — Useful for logging client IPs

* **Weighted Target Groups (Advanced Use Case)**
  + Used when multiple target groups are attached to one listener rule.
  + **Example:**
    - TG1 = old version of app (80% traffic)
    - TG2 = new version (20% traffic)

* **Example Setup**
  + **You run a website with these components:**
    - /api/\* — API backend (EC2)
    - /images/\* — Image service (S3 or ECS)
    - /lambda/\* — Serverless logic (Lambda)
  + **Architecture Flow:**
    - User → Application Load Balancer
    - **ALB listener rule:**
      * /api/\* → Target Group A (EC2)
      * /images/\* → Target Group B (ECS)
      * /lambda/\* → Target Group C (Lambda)
    - Each TG does health checks and sends requests to healthy targets.

# **AWS Load Balancer — Complete Guide**

### (What, Why, How, Types, Features, When to Use, Use Cases)

# 💠 **What is an AWS Load Balancer?**

An **AWS Load Balancer** is a managed service that **automatically distributes incoming traffic** across multiple targets such as:

* EC2 instances
* Containers (ECS, EKS)
* Lambda functions
* IP addresses
* On-prem servers (via Direct Connect / VPN)

Its main purpose is to give you:

* **High availability**
* **Fault tolerance**
* **Scalability**
* **Zero downtime**

Load Balancers are part of **Elastic Load Balancing (ELB)** service.

# 💠 **Why Load Balancers Matter**

1. **Avoid server overload**  
   Traffic is automatically balanced across many servers.
2. **High availability (HA)**  
   If one server fails, traffic moves to healthy ones.
3. **Zero-downtime deployment**  
   Update servers behind the LB without affecting customers.
4. **Security**  
   Supports TLS termination, WAF integration, Shield protection.
5. **Automatic scaling**  
   Works with Auto Scaling Groups to handle traffic spikes smoothly.

# 💠 **How Load Balancers Work**

1. User sends a request (HTTP/HTTPS/TCP/UDP).
2. The request reaches the Load Balancer.
3. LB checks **target health** using health checks.
4. Traffic is routed to the **healthy targets** based on rules.
5. LB ensures **even distribution** and **session consistency** if required.

# 🔵 **Types of AWS Load Balancers**

AWS currently offers **four** types:

1. **Application Load Balancer (ALB)**
2. **Network Load Balancer (NLB)**
3. **Gateway Load Balancer (GWLB)**
4. **Classic Load Balancer (Legacy – not recommended for new systems)**

Below is a full breakdown.

# 🟦 1. **Application Load Balancer (ALB)**

### (Layer 7 Load Balancer)

## **What is ALB?**

A **Layer 7 (Application Layer)** HTTP/HTTPS load balancer designed for:

* Web apps
* Microservices
* Modern architectures (REST, GraphQL)

## **Why use ALB?**

* Understands **HTTP/HTTPS**
* Makes routing decisions based on URL, headers, cookies
* Supports **microservices**, multiple apps on same LB
* Best for websites and web APIs

## **How ALB Works**

* You define **Listeners** (HTTP/HTTPS)
* You create **Rules** (path-based, host-based)
* Traffic is sent to **Target Groups** (EC2, ECS, Lambda)

## **ALB Features**

✔ Path-based routing (/api/\*, /user/\*)  
✔ Host-based routing (app1.example.com, app2.example.com)  
✔ Supports **WebSockets**, **HTTP/2**, **gRPC**  
✔ Sticky sessions (Session Affinity)  
✔ Integrated with **WAF**  
✔ Can route to **Lambda functions**  
✔ Load balancing for containers (ECS, EKS)

## **When to Use ALB**

* Websites + Web applications
* Microservices
* REST APIs / GraphQL APIs
* Host/path-based routing
* Serverless backend using Lambda
* Docker/Kubernetes workloads

## **Use Cases for ALB**

* E-commerce sites
* Multi-tenant applications
* Route /login to one service and /payment to another
* API Gateway alternative for simple APIs
* Balancing traffic across ECS tasks

# 🟩 2. **Network Load Balancer (NLB)**

### (Layer 4 Load Balancer — TCP/UDP)

## **What is NLB?**

A **Layer 4** load balancer designed for:

* **High performance**
* **Low latency**
* **Millions of requests per second**
* TCP, UDP, TLS traffic

## **Why use NLB?**

* Handles **extreme traffic load**
* Uses **static IP addresses**
* One of the fastest LB types
* Suitable for financial, gaming, telecom apps

## **How NLB Works**

* Receives TCP/UDP traffic
* Performs **connection-level load balancing**
* Health checks determine healthy targets
* Sends raw traffic directly to backend servers

## **NLB Features**

✔ Ultra-low latency  
✔ Supports **Static IPs** per AZ  
✔ TLS termination  
✔ Load balancing between VPCs  
✔ Cross-zone load balancing  
✔ Source IP preservation

## **When to Use NLB**

* High-performance apps
* Real-time systems
* Gaming servers
* IoT traffic
* Low latency workloads
* Need static IP load balancer

## **Use Cases**

* Payment processing systems
* High-frequency trading apps
* Gaming servers
* VPN or proxy servers
* Microservices requiring TCP/UDP load balancing

# 🟧 3. **Gateway Load Balancer (GWLB)**

### (Layer 3 — Network Firewall Load Balancing)

## **What is GWLB?**

A load balancer designed to host and distribute traffic to:

* **Firewalls**
* **Intrusion Detection Systems (IDS)**
* **Intrusion Prevention Systems (IPS)**
* **Deep packet inspection tools**

It acts as a **single-entry gateway** for traffic inspection.

## **Why GWLB?**

* You can insert **security appliances** in your architecture
* Scaling network firewalls automatically
* Unified gateway for traffic inspection
* PAYG model replaces expensive physical appliances

## **How GWLB Works**

* Uses **GENEVE protocol** for tunneling
* Traffic flows through GWLB → firewall appliances → VPC

## **GWLB Features**

✔ Auto scale firewalls  
✔ Transparent deployment  
✔ Traffic inspection  
✔ VPC-to-VPC firewalling  
✔ High availability for network security

## **When to Use GWLB**

* If you need advanced security inspection
* Multi-VPC firewall connectivity
* Centralized security layer

## **Use Cases**

* Central security VPC
* Enterprise-grade firewall clusters
* High-security environments
* Traffic filtering before reaching apps

# 🟥 4. **Classic Load Balancer (CLB)**

### (Legacy — Not Recommended)

## **What is CLB?**

Old generation load balancer supporting:

* HTTP/HTTPS (Layer 7)
* TCP (Layer 4)

## **Why it’s not recommended?**

* Limited features
* Not cost-effective
* No path/host routing
* No advanced routing
* ALB/NLB provide better performance

Use only if you maintain old environments.

# 🟦 Quick Comparison Table

| **Feature** | **ALB** | **NLB** | **GWLB** | **CLB** |
| --- | --- | --- | --- | --- |
| Layer | 7 | 4 | 3 | 4/7 |
| Protocols | HTTP/HTTPS | TCP/UDP/TLS | GENEVE | HTTP/HTTPS/TCP |
| Best for | Web apps | High performance | Security appliances | Legacy apps |
| Supports WebSockets | ✔ | ❌ | ❌ | ✔ |
| Path/Host Routing | ✔ | ❌ | ❌ | ❌ |
| Static IP | ❌ | ✔ | ❌ | ❌ |
| WAF Integration | ✔ | ✔ (via ALB/NLB) | ❌ | ❌ |

# 🟢 Final Summary in Simple Words

* **ALB = Websites, APIs, Microservices**
* **NLB = High-speed, low-latency TCP/UDP**
* **GWLB = Firewalls, Security Appliances**
* **CLB = Old systems, avoid unless required**

**Amazon ElastiCache** is a fully managed in-memory caching service offered by Amazon Web Services (AWS).1 It significantly **improves application performance** by retrieving frequently accessed data from fast, managed, memory-based systems instead of relying entirely on slower, disk-based databases.2

It handles all the heavy lifting of cache administration, such as hardware provisioning, software patching, configuration, monitoring, and failure recovery.3

## Supported Engines

ElastiCache is compatible with two popular open-source in-memory data store engines:4

1. **Redis (Redis OSS/Valkey):5**
   * **Features:** Supports a rich set of data structures (strings, lists, sets, sorted sets, hashes, bitmaps, etc.). It can be used as a cache, a message broker (Pub/Sub), and a high-performance in-memory database due to its replication and persistence features.6
   * **Availability:** Supports **Multi-AZ deployments** with automatic failover for high availability.7
2. **Memcached:**
   * **Features:** A simpler, more lightweight key-value store.8 It's ideal for pure caching and utilizing large nodes with multiple cores.9
   * **Availability:** Does **not** support replication or automatic failover like Redis, so fault tolerance must be managed at the application level.10

## How It Works (Caching)

ElastiCache acts as a fast, temporary storage layer that sits between your application and your persistent database (like Amazon RDS or DynamoDB).11

1. **Application Request:** A user request comes in to view data (e.g., a product catalog).12
2. **Cache Check (Cache Hit):** The application first checks ElastiCache for the requested data.13
   * If the data is in the cache (**cache hit**), ElastiCache returns it immediately with **microsecond latency**.14
3. **Database Check (Cache Miss):** If the data is not in the cache (**cache miss**), the application queries the backend database.15
4. **Cache Write:** Once the database returns the result, the application serves the user and simultaneously **writes the data to ElastiCache** for all future requests to benefit from the faster response time.16

## Primary Use Cases

|  |  |
| --- | --- |
| Use Case | Description |
| **Database Caching** | Reduces the load on backend databases by caching frequently read query results, improving database and application response times. |
| **Session Store** | Stores user session information (like login status, shopping cart contents) in-memory, enabling faster user experience and easy horizontal scaling of the application servers. |
| **Real-Time Leaderboards** | Uses Redis's **Sorted Sets** data structure to create and update gaming or competition leaderboards in real-time. |
| **Messaging & Queues** | Utilizes Redis's Pub/Sub functionality to act as a fast, in-memory message broker for distributing events and messages between application components. |

## Serverless Option

AWS offers **ElastiCache Serverless**, which removes the need for capacity planning.17 It automatically and instantly scales the cache both vertically (node size) and horizontally (adding/removing shards) to meet the application's demand, and you pay only for the data stored and the compute utilized.18

The video below gives an overview of how ElastiCache for Redis helps improve performance and reduce database costs.

AWS Databases in 15 - Getting Started with ElastiCache for Redis Performance & Cost Optimization

## **Ch # 5**

**AWS Compute Services**



### [**AWS Compute Services**](https://tutorialsdojo.com/aws-cheat-sheets-compute-services/)

* [Amazon EKS](https://tutorialsdojo.com/amazon-elastic-kubernetes-service-eks/)
* [Amazon Elastic Compute Cloud (EC2)](https://tutorialsdojo.com/amazon-elastic-compute-cloud-amazon-ec2/)
* [Amazon Elastic Container Registry (ECR)](https://tutorialsdojo.com/amazon-elastic-container-registry-amazon-ecr/)
* [Amazon Elastic Container Service (ECS)](https://tutorialsdojo.com/amazon-elastic-container-service-amazon-ecs/)
* [Amazon Lightsail](https://tutorialsdojo.com/amazon-lightsail/)
* [AWS Batch](https://tutorialsdojo.com/aws-batch/)
* [AWS Elastic Beanstalk](https://tutorialsdojo.com/aws-elastic-beanstalk/)
* [AWS Fargate](https://tutorialsdojo.com/aws-fargate/)
* [AWS Lambda](https://tutorialsdojo.com/aws-lambda/)
* [AWS ParallelCluster](https://tutorialsdojo.com/aws-parallelcluster/)
* [AWS Serverless Application Model (SAM)](https://tutorialsdojo.com/aws-serverless-application-model-sam/)
* [AWS Serverless Application Repository](https://tutorialsdojo.com/aws-serverless-application-repository/)
* [AWS Savings Plan](https://tutorialsdojo.com/aws-savings-plan/)
* [AWS Wavelength](https://tutorialsdojo.com/aws-wavelength/)

**Other Compute-related Notes:**

* [Amazon EC2 Auto Scaling Lifecycle Hooks](https://tutorialsdojo.com/amazon-ec2-auto-scaling-lifecycle-hooks/)
* [Building Serverless APIs with AWS API Gateway and AWS Lambda](https://tutorialsdojo.com/building-serverless-apis-with-aws-api-gateway-and-aws-lambda/)
* [Configuring Notifications for Amazon EC2 Auto Scaling Lifecycle Hooks](https://tutorialsdojo.com/configuring-notifications-for-amazon-ec2-auto-scaling-lifecycle-hooks/)
* [ECS Network Modes Comparison](https://tutorialsdojo.com/ecs-network-modes-comparison/)
* [ECS Task Placement Strategies](https://tutorialsdojo.com/ecs-task-placement-strategies/)
* [Real-time Monitoring of 5XX Errors using AWS Lambda, CloudWatch Logs and Slack](https://tutorialsdojo.com/real-time-monitoring-of-5xx-errors-using-aws-lambda-cloudwatch-logs-slack/)
* [Reduce The Cost Of Your Lambda functions With Event Filters(Part 1 of 2)](https://tutorialsdojo.com/reduce-the-cost-of-your-lambda-functions-with-event-filters-part-1-of-2/)
* [Reduce The Cost Of Your Lambda functions With Event Filters(Part 2 of 2)](https://tutorialsdojo.com/reduce-the-cost-of-your-lambda-functions-with-event-filters-part-2-of-2/)
* [SFTP on Your Ubuntu EC2 Instance – Quick and Simple Setup](https://tutorialsdojo.com/sftp-on-your-ubuntu-ec2-instance-quick-and-simple-setup/)
* [Understanding and Managing AWS Lambda Recursive Loop Invocations](https://tutorialsdojo.com/understanding-and-managing-aws-lambda-recursive-loop-invocations/)
* [Utilizing Lambda Functions to Control Amazon EC2 Instances via Slack](https://tutorialsdojo.com/utilizing-lambda-functions-to-control-amazon-ec2-instances-via-slack/)

**AWS EC2 Elastic Compute Cloud**

* Amazon EC2 (Elastic Compute Cloud) is a Linux-based, Windows-based, or Mac-based virtual server (instance) provisioned in the cloud.
* Provides scalable, secure, and resizable compute capacity integrated with AWS services like S3, RDS, etc.
* Operates as an Infrastructure as a Service (IaaS) with automatic scaling capabilities.
* Reliable, pay-as-you-go pricing model.
* Instances can be launched across multiple regions and Availability Zones (AZs).
* Supports various operating systems and works with AWS VPC for secure networking.
* AWS offers [over 300 EC2 instance types](https://www.densify.com/resources/ec2-instance-types) across 5 instance families (general purpose family, memory-optimized, storage-optimized, compute-optimized, and accelerated computing), each with varying resource and performance focuses

**Core Capabilities**:

* **EC2 Instance**: Renting a virtual machine.
* **EBS**: Storing data on a virtual drive (Elastic Block Store).
* **ELB**: Distributing load across machines (Elastic Load Balancer).
* **ASG**: Scaling services using Auto Scaling Groups.

**Sizing and Configuration Options**:

* **Operating System**: Linux, Windows, or Mac.
* **CPU**: Compute power or cores.
* **RAM**: Memory allocation.
* **Storage**:
  + Network-attached storage (EBS, EFS).
  + Hardware-based EC2 instance store.
* **Network Card**:
  + Speed of the card.
  + Public or private IP assignment.
  + Firewall rules via security groups.
* **EC2 User Data**: Bootstrap scripts to configure instances at first launch.

**Instance Limits**:

* Limited to On-Demand Instances per vCPU-based limits, 20 Reserved Instances, and Spot Instances per dynamic Spot limit per region.

**Amazon EC2 Features**

* **AWS Nitro System**:
  + Next-generation platform for EC2 instances, offloading functions to dedicated hardware/software for enhanced performance and cost efficiency.
  + Outperforms the Xen Hypervisor, delivering near bare-metal performance.
* **Instance Types**:
  + **General Purpose**: t-type, m-type.
  + **Compute Optimized**: c-type.
  + **Memory Optimized**: r-type, x-type, z-type.
  + **Storage Optimized**: d-type, h-type, i-type.
  + **Accelerated Computing**: f-type, g-type, p-type.
* **Amazon Machine Images (AMIs)**:
  + Reusable templates packaging OS and additional software.
  + Includes root volume template, launch permissions, and block device mappings.
  + Supports UEFI Secure Boot and Instance Metadata Service Version 2 (IMDSv2).
  + AMIs can be copied across regions, restored via Recycle Bin, and have a 2-year deprecation for public AMIs.
* **Storage**:
  + **Instance Store Volumes**: Temporary storage, deleted on stop/terminate.
  + **EBS Volumes**: Persistent storage, supports encryption, can be stopped/restarted.
  + **Root Device Volumes**: Contains the boot image; EBS-backed by default, deleted on termination unless configured otherwise.
* **Networking and Security**:
  + **Elastic IP Addresses**: Static IPv4 addresses for dynamic cloud computing, limited to 5 per region by default.
  + **Security Groups**: Virtual firewalls controlling traffic, stateful, permissive rules only.
  + **VPCs**: Logically isolated virtual networks, optionally connectable to on-premises networks.
  + **Key Pairs**: Secure login, supports .pem, .ppk, and ED25519 keys for EC2 Instance Connect/Serial Console.
* **Instance Management**:
  + **User Data**: Shell scripts or cloud-init directives (16 KB limit) for instance configuration.
  + **Instance States**:
    - **Start**: Normal operation, billed continuously.
    - **Stop**: Shuts down, preserves EBS data, no usage charges.
    - **Hibernate**: Saves in-memory state to encrypted EBS root volume, no hourly charges.
    - **Terminate**: Deletes instance, preserves EBS volumes by default.
  + **Protection**: Enable termination or stop protection to prevent accidental changes.
  + **Host Recovery**: Automatically restarts instances on a new Dedicated Host after hardware failure.
  + **Hibernation**: Supported for specific instance types (e.g., M3, M4, M5, C3, C4, C5, R3, R4, R5) with encrypted EBS volumes.
* **Integration**:
  + Automatically connect EC2 instances to RDS databases.
  + Network traffic replication for security and monitoring.

# **Amazon EC2 Image Builder**

* Fully managed service to automate AMI creation, management, and deployment.
* Supports AWS Management Console, CLI, or APIs for custom image creation.
* Configures pipelines for automated updates and system patching.
* Provides a standalone command for AMI creation based on defined configurations.

# Amazon EC2 Pricing

* **On-Demand**: Pay per second, no long-term commitments.
* **Reserved Instances**:
  + One-time payment for 1- or 3-year terms with discounted hourly rates.
  + **Standard**: Higher discount (40% for 1-year, 60% for 3-year), limited modifications, sellable in Reserved Instance Marketplace.
  + **Convertible**: Lower discount (31% for 1-year, 54% for 3-year), exchangeable for different attributes, not sellable.
  + **Scope**: Regional (flexible AZ, instance family) or Zonal (specific AZ, instance type).
* **Spot Instances**:
  + Up to 90% discount, uses unused EC2 capacity.
  + Spot Fleets combine Spot and On-Demand Instances to meet target capacity.
  + Allocation strategies: LowestPrice, Diversified, CapacityOptimized, InstancePoolsToUseCount.
  + Can be stopped/started for EBS-backed instances, interrupted if capacity is unavailable or price exceeds bid.
* **Dedicated Hosts**: Fully dedicated physical hosts for existing software licenses.
* **Dedicated Instances**: Single-tenant hardware, billed hourly.
* **On-Demand Capacity Reservations**:
  + Reserve capacity in a specific AZ without term commitments.
  + Apply Savings Plans or Regional Reserved Instances for discounts.
  + Cannot be used with Dedicated Hosts, supports placement groups.
* **Additional Charges**:
  + Data transfer between regions, unattached Elastic IPs, or additional Elastic IPs.
  + EBS pricing is separate from instance pricing.

# Amazon EC2 Security

* **IAM**: Control instance access via policies and roles.
* **Security Groups**:
  + Virtual firewalls, permissive rules only, automatically applied.
  + Default allows all outbound traffic and inbound traffic from instances in the same group.
* **Network Traffic**: Replicate for inspection, monitoring, or troubleshooting.
* **Password Security**: Disable password-based logins for AMIs to enhance security.

# Amazon EC2 Networking

* **Elastic IP Addresses**:
  + Static IPv4 for dynamic cloud computing, region-specific, transferable between accounts.
  + Required for internet communication if auto-assign public IP is disabled.
* **Network Interfaces**:
  + Primary interface (eth0) cannot be detached; additional interfaces can be created.
  + Maximum interfaces vary by instance type, can attach across subnets in the same AZ.
* **Enhanced Networking**: Uses SR-IOV for high bandwidth, low latency in placement groups.
* **Elastic Fabric Adapter (EFA)**: Accelerates HPC and machine learning with low latency, high throughput.
* **Bastion Hosts**: EC2 instances for secure SSH/RDP access to VPC instances.

# Amazon EC2 Monitoring

* **Metrics**:
  + EC2 Metrics: CPU, network, disk performance, reads/writes.
  + Monitoring Agent/CloudWatch Logs: Memory, disk swap, space, page file, logs.
* **Tools**:
  + **System Status Checks**: Monitor AWS systems for instance operation.
  + **Instance Status Checks**: Monitor instance software/network configuration.
  + **CloudWatch Alarms**: Trigger actions based on metric thresholds.
  + **CloudWatch Events**: Automate responses to system events.
  + **CloudWatch Logs**: Store and access logs from EC2, CloudTrail, etc.
* **Data Collection**: Default 5-minute intervals, optional 1-minute detailed monitoring.

# Amazon EC2 Instance Metadata and User Data

* **Instance Metadata**: Data for configuring/managing instances, accessible at http://169.254.169.254/latest/meta-data/.
* **User Data**: Shell scripts or cloud-init directives (16 KB limit), not executed after instance restart.
* **Access**: Retrieve user data at http://169.254.169.254/latest/user-data.
* **Tags**: Instance tags accessible via metadata, includes Auto Scaling lifecycle state.

# Amazon EC2 Placement Groups

* **Cluster**: Low-latency group in a single AZ for high network throughput applications.
* **Spread**: Spreads instances across hardware, supports up to 7 instances per AZ, spans multiple AZs.
* **Partition**: Spreads instances across logical partitions to avoid correlated failures, ideal for large distributed workloads (e.g., HDFS, HBase, Cassandra).

# Amazon EC2 Rules

* Placement group names must be unique per AWS account/region.
* Cannot merge placement groups or launch host-tenancy instances in them.
* Instances are limited to one placement group at a time.

# Amazon EC2 Storage

* **EBS**: Durable block storage, supports snapshots to S3 for backups.
* **Instance Store**: Temporary storage, data lost on stop/terminate.
* **EFS**: Scalable file storage for multiple instances.
* **FSx**:
  + FSx for Windows File Server: Managed Windows-based storage.
  + FSx for Lustre: High-performance file system for HPC.
  + FSx for NetApp ONTAP: Managed ONTAP-based storage.
  + FSx for OpenZFS: Managed OpenZFS-based storage.
* **S3**: Stores EBS snapshots and instance store-backed AMIs.
* **Torn Write Prevention**: Enhances performance for I/O-intensive database workloads.

# Amazon EC2 Resources and Tagging

* **Resources**: Images, instances, volumes, snapshots with unique regional/AZ-specific IDs.
* **Global Resources**: AWS account, key pairs (can be uploaded to regions).
* **Regional Resources**: AMIs, Elastic IPs, security groups, EBS snapshots.
* **AZ-Specific Resources**: EBS volumes, instances.
* **Tags**: User-defined key-value pairs for resource metadata.

# Validate Your Knowledge

1. **Which Amazon EC2 instance purchasing option can help you address compliance requirements and reduce costs by allowing you to use your existing server-bound software licenses?**
   * **Answer**: Dedicated Host
     + Dedicated Hosts allow you to use existing per-socket, per-core, or per-VM software licenses on a fully dedicated physical host, addressing compliance and cost-saving needs.
2. **A company deployed a high-performance computing (HPC) cluster across multiple Amazon EC2 instances and Availability Zones, experiencing slowdown due to latency issues. Which is the MOST suitable solution for low-latency network performance for tightly-coupled node-to-node communication?**
   * **Answer**: Set up a cluster placement group within a single Availability Zone in the same AWS Region.
     + Cluster placement groups place instances close together in a single AZ, minimizing latency for HPC workloads requiring tight node-to-node communication.

## Amazon EC2 References:

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/><https://aws.amazon.com/ec2/features/><https://aws.amazon.com/ec2/pricing/><https://aws.amazon.com/ec2/faqs/>

<https://tutorialsdojo.com/amazon-elastic-compute-cloud-amazon-ec2/>

**EC2 Purchasing and Pricing Models Or Instance Types**

* Amazon offers different purchasing and pricing models to help you choose the best way to pay for these virtual computers based on your needs.
* Each model fits different needs
* **Flexibility:** On-Demand and Spot let you use EC2 without long-term commitments, great for testing or irregular tasks.
* **Cost Savings**: Savings Plans and Reserved Instances save money if you know you’ll use EC2 a lot.
* **Control**: Dedicated Hosts/Instances give you full control for special requirements, like licensing or security.
* **Match Your Workload**: Some apps need constant power (use Reserved or Savings Plans), while others can handle interruptions (use Spot).
* **On-Demand Instances:**
  + Pay as you go, flexible, expensive.
  + pay for the computing power by the hour or second, with no long-term commitment.
  + You can start and stop using the instance anytime.
  + Great for unpredictable workloads or short-term projects.
  + No upfront payment or contract, so it’s flexible.
  + Most expensive per hour, but no long-term obligation

* **Savings Plans**
  + Commit to usage, save money, flexible.
  + You commit to using a certain amount of computing power
  + measured in dollars per hour for 1 or 3 years
  + Amazon gives you a discount
  + You can use this across different EC2 instance types or even other AWS services.
  + Saves money if you know you’ll use EC2 consistently.
  + Flexible because you can change instance types or regions.
  + Lower than On-Demand (up to 72% savings)
  + you commit to a minimum spend.

* **Reserved Instances**
  + Commit to a specific server, bigger savings, less flexible.
  + You commit to using a specific type of EC2 instance (e.g., size, region, operating system) for 1 or 3 years
  + you get a big discount
  + You can pay upfront, partially upfront, or monthly.
  + Best for predictable, steady workloads that won’t change much.
  + Cheaper than On-Demand (up to 75% savings)
  + less flexible than Savings Plans.

* **Spot Instances:**
  + Super cheap, but can be interrupted.
  + You bid on spare EC2 capacity that Amazon isn’t using.
  + It’s super cheap
  + Amazon can take the instance back if someone else needs it.
  + Perfect for tasks that can handle interruptions, like data analysis or batch processing.
  + Up to 90% cheaper than On-Demand
  + you risk losing the instance.
  + **Spot Fleet**
    - Spot Fleet is a set of spot instances and optional on-demand instances
    - The spot fleet will try to meet the target capacity with price constraints
    - AWS will launch instances from a launch pool, meaning we have to define the instance type, OS, AZ for a launch pool
    - We can have multiple launch pools from within the best one is chosen
    - If a spot a fleet reaches capacity or max cost, no more new instances are launched
    - **Strategies to allocate spot instances in a spot fleet:**
      * lowerPrice: the instances will be launched from the pool with the lowest price
      * diversified: launched instances will be distributed from all the defined pools
      * capacityOptimized: launch with the optimal capacity based on the number of instances

* **Dedicated Hosts/Instances:**
  + Exclusive server, costly, for special needs.
  + You get a physical server just for your use, not shared with other customers
  + You can choose Dedicated Instances (pay per instance) or Dedicated Hosts (pay for the whole server).
  + Needed for specific software licenses (e.g., Windows Server) or compliance requirements.
  + Most expensive option due to exclusive use.

### **Launch Type**

* On Demand
* Pay as you go
* Reserved
* 3 types in terms of duration
  + Regular Reserved Instance
  + Min 1 to max 3 Years
  + Convertible Reserved Instance
  + Can be convert the types like more cpu optimized, more memory optimized
  + Scheduled Reserved Instance
  + Will be up and running for certain times in regular basis
  + Recommended when the time frame is at least 1 year usage
* 2 types in terms of capacity reservation
  + Zonal - Allow capacity reservation, applicable for availability zones reserve instances
  + Regional - Allow capacity reservation, applicable for regional reserve instances
* Spot Instance
* can loose instance
* very low price
* Define max spot price
* if the current spot price goes high of the defined spot price, we loose the instance in 2 minutes
* Using Spot Block we can extend the termination delay till -6 hours
* Two type of request
  + One time request
  + Once max price < current price, all instances are removed
  + Persistent Request
  + If spot instances stop and then things are good, these instances launched automatically
    - To stop persistent request
    - First delete the spot request
    - Then removed the spot instances
* Spot Fleets
  + Set of
  + Spot Instances and On-Demand Instances
  + We can define
  + Possible launch pools
    - Multiple AZ
    - Various type of instance
    - Various OS
  + Automatically stops when meed the capacity
  + It offers
  + lowestPrice
  + diversified (distribute across az and workloads)
  + capacityOptimized
* Dedicated Instance
* Does not share hardware
* Dedicated Host
* Does not share server, entire placement is booked

### **Dedicated Instance Vs Dedicated Host**

* In Dedicated Instance
* The billing is is done per instance
* The other instance of same account, may share hardware
* No control over placement group
* In Dedicated Host, billing is on whole Dedicated Host
* Control over placement group

## EC2 Instance Types

* R: applications that needs a lot of RAM - in-memory cache
* C: applications that need good CPU - compute/database
* M: applications that are balanced - general / web app
* I: applications that need good local I/O - databases
* G: applications that need GPU - video rendering / ML
* T2/T3 - burstable instances
* T2/T3 unlimited: unlimited burst

### Bustable Instances (T2/T3)

* Overall the performance of the instance is OK
* When the machine needs to process something unexpected (a spike load), it can burst and CPU can be very performant
* If the machine bursts, it utilizes "burst credits"
* If all the credits are gone, the CPU becomes bad
* If the machine stops bursting, credits are accumulated over time
* Credit usage / credit balance of a burstable instance can be seen in CloudWatch
* CPU credits: bigger the instance the faster credit is earned
* T2/T3 Unlimited: extra money can be payed in case the burst credits are used. There wont be any performance loss

### **Instance Type**

* R
* Instance with lots of memory/RAM
* Used when in memory caching is required
* C
* Instance with good Computation Power
* Used for DB Server
* M
* Middle between RAM and Computation
* Used for Application Server / General Application
* Has 10-25 GB/s networking
* Has ENA enabled
* Low latency network with NVME ebs
* I
* For heave I/O Application
* Used for DB
* When good Instance Storage is required
* G
* GPU optimized instance
* Used for video rendering or machine learning
* T2/T3
* Burstble instance
* Provide good performance according to the capacity
* There is a criteria, where unlimited burst is provided

### **Terminating a EC2 Instance**

* For production EC2 Instance
* Need a tag
* In resource level there should be explicit deny for production tag for not to terminates

## EC2 for Solution Architects

* EC2 instances are billed by the second, t2.micro is free tier
* On Linux/Mac we can use SSH, on Windows Putty or SSH
* SSH is using port 22, the security group must allow our IP to be able to connect
* In cas of a timeout, it is most likely a security group issue
* Permission for SSH key => chmod 0400
* Security groups can reference other security groups instead of IP addresses
* EC2 instance can be customized at boot using EC2 User Data
* 4 EC2 launch modes:
  + On-demand
  + Reserved
  + Spot
  + Dedicated hosts
* We can create AMIs to pre-install software
* An AMI can be copied through accounts and regions
* EC2 instances can be started in placement groups:
  + Cluster
  + Spread
  + Partition

**AWS EC2 Placement Group:**

* Amazon EC2 Placement Groups are a way to control how your ec2 instances are placed on Amazon’s physical servers to improve performance, reduce delays, or ensure separation for safety.
* We use Placement Groups in AWS to organize instances for faster performance, better reliability, or to keep them separate so one failure doesn’t break everything, depending on what your app needs.
* **Cluster Placement Group**
  + Cluster ec2 instance into a low latency group in a same rack and same AZ
  + groups instances closely for high-speed communication
  + 10 Gbps bandwidth b/w instance
  + High risk of failing
  + The disadvantage is that if the rack fails, all instances on that rack fail at the same time.
  + Use in big data job that needs to complete fast
  + App that needs extremly low latency and high network throughput
* **Partition Placement Group**
  + Ec2 instance across many different partitions
  + These different partitions rely on different sets of racks within an AZ
  + spreads instances across separate hardware partitions for large-scale app reliability
  + Up to 7 partitions per AZ
  + Up to 100s of EC2 instances
  + Scales to 100s of EC2 instances per group ( hadoop, cassandra, kafka)
  + The instances in a partition do not share racks with the instances in the other partitions
  + A partitions failure can affect many EC2 but won't affect other partitions
  + Ec2 instances get access to the partitions information as metadata
* **Spread Placement Group** 
  + ec2 instance across underlying different physcial hardware
  + Can span across different AZs
  + Max 7 instance per AZ per placement group
  + Use for Critical applicions where each instance must be isolated from failure from each other
  + Each instance places on distinct servers for maximum independence and fault tolerance.
  + Reduced risk is simultaneous failure because if hardware one fail, the other hardware still works
  + Applications that needs to maximize high availability

## EC2 User Data

* It is possible to bootstrap (run commands for setup) an EC2 instance using EC2 User data script
* The user data script is only run once at the first start of the instance
* EC2 user data is used to automate boot tasks such as:
  + Installing update
  + Installing software
  + Downloading common files from the internet
  + Any other start-up task
* THe EC2 user data scripts run with root user privileges

### **EC2 Meta Data**

* Information about the instance

## EC2 Hibernate

* We can stop or terminate EC2 instances:
  + If an instance is stopped: the data on the disk (EBS) is kept intact
  + If an instance is terminated: any root EBS volume will also gets destroyed
* On start, the following happens in case of an EC2 instance:
  + Fist start: the OS boots and EC2 User data script is executed
  + Following starts: the OS boots
  + After the OS boot the applications start, cache gets warmed up, etc. which may take some time
* EC2 Hibernate:
  + All the data from RAM is preserved on shut-down
  + The instance boot is faster
  + Under the hood: the RAM state is written to a file in the root EBS volume
  + The root EBS volume must be encrypted
* Supported instance types for hibernate: C3, C4, C5, M3, M4, M5, R3, R4, R5
* Supported OS types: Amazon Linux 1 and 2, Windows
* Instance RAM size: must be less then 150 GB
* Bare metal instances do not support hibernate
* Root volume: must be EBS, encrypted, not instance store. And it must be large enough
* Hibernate is available for on-demand and reserved instances
* An instance can not hibernate for more than 60 days

## AMI

* AWS comes with lots of base images
* Images can be customized ar runtime with EC2 User data
* In case of more granular customization AWS allows creating own images - this is called an AMI
* Advantages of a custom AMI:
  + Pre-install packages
  + Faster boot time (on need for the instance to execute the scripts from the user data)
  + Machine configured with monitoring/enterprise software
  + Security concerns - control over the machines in the network
  + Control over maintenance
  + Active Directory out of the box
* An AMI is built for a specific region (NOT GLOBAL!)

### Public AMI

* We can leverage AMIs from other people
* We can also pay for other people's AMI by the hour, basically renting the AMI form the AWS Marketplace
* Warning: do not use AMI which is not trustworthy!

### AMI Storage

* An AMI takes space and they are stored in S3
* AMIs by default are private and locker for account/region
* We can make our AMIs public and share them with other people or sell them on the Marketplace

### Cross Account AMI Sharing

* It is possible the share AMI with another AWS account
* Sharing an AMI does not affect the ownership of the AMI
* If a shared AMI is copied, than the account who did the copy becomes the owner
* To copy an AMI that was shared from another account, the owner of the source AMI must grant read permissions for the storage that backs the AMI, either the associated EBS snapshot or an associated S3 bucket
* Limits:
  + An encrypted AMI can not be copied. Instead, if the underlying snapshot and encryption key where shared, we can copy the snapshot while re-encrypting it with a key of our own. The copied snapshot can be registered as a new AMI
  + We cant copy an AMI with an associated billingProduct code that was shared with us from another account. This includes Windows AMIs and AMIs from the AWS Marketplace. To copy a shared AMI with billingProduct code, we have to launch an EC2 instance from our account using the shared AMI and then create an AMI from source

## **Ch # 6**

**AWS Storage Services**



### [AWS Storage Services](https://tutorialsdojo.com/aws-cheat-sheets-storage-services/)

* [Amazon EBS](https://tutorialsdojo.com/amazon-ebs/)
* [Amazon EFS](https://tutorialsdojo.com/amazon-efs/)
* [Amazon FSx](https://tutorialsdojo.com/amazon-fsx/)
* [Amazon S3 Glacier](https://tutorialsdojo.com/amazon-glacier/)
* [Amazon S3](https://tutorialsdojo.com/amazon-s3/)
* [AWS Backup](https://tutorialsdojo.com/aws-backup/)
* [AWS Snowball Edge](https://tutorialsdojo.com/aws-snowball-edge/)
* [AWS Snowmobile](https://tutorialsdojo.com/aws-snowmobile/)
* [AWS Storage Gateway](https://tutorialsdojo.com/aws-storage-gateway/)
* [AWS Transfer Family](https://tutorialsdojo.com/aws-transfer-family/)

**Other Storage-related Notes:**

* [Amazon EBS Multi-Attach](https://tutorialsdojo.com/amazon-ebs-multi-attach/)
* [Amazon S3 Access Points](https://tutorialsdojo.com/amazon-s3-access-points/)
* [Amazon S3 Bucket Policies for VPC Endpoints](https://tutorialsdojo.com/amazon-s3-bucket-policies-for-vpc-endpoints/)
* [Amazon S3 Event Notifications](https://tutorialsdojo.com/amazon-s3-event-notifications/)
* [Cost Allocation Tags on Amazon S3 Buckets and Using AWS Cost Explorer](https://tutorialsdojo.com/cost-allocation-tags-on-amazon-s3-buckets-and-using-aws-cost-explorer/)
* [Enhancing S3 Bucket Security by Prohibiting Uploads of Unencrypted Objects](https://tutorialsdojo.com/enhancing-s3-bucket-security-by-prohibiting-uploads-of-unencrypted-objects/)
* [Locking your Glacier Vault using the Amazon S3 Glacier API](https://tutorialsdojo.com/locking-your-glacier-vault-using-the-amazon-s3-glacier-api/)
* [Redacting PIIs Using S3 Object Lambda](https://tutorialsdojo.com/redacting-piis-using-s3-object-lambda/)
* [Streamlining EBS Snapshot Management with Amazon Data Lifecycle Manager Automation](https://tutorialsdojo.com/streamlining-ebs-snapshot-management-with-amazon-data-lifecycle-manager-automation/)
* [Using Origin Access Control (OAC) to Secure S3 Object Access to CloudFront](https://tutorialsdojo.com/using-origin-access-control-to-secure-s3-object-access-to-cloudfront/)

# **Amazon S3**

* Amazon S3 stores data as objects within buckets.
* An **object** consists of a file, a unique key (identifier), and optional metadata.
* **Buckets** are containers for objects with virtually unlimited storage capacity.
* **Key**: Unique identifier for an object within a bucket.
* **Buckets**:
  + Control access (create, delete, list objects).
  + View access logs for buckets and objects.
  + Choose the geographical region for storage (cannot change after creation).
  + Bucket names must be globally unique, DNS-compliant, and visible in object URLs.
  + Default limit: 100 buckets per AWS account.
  + Cannot delete buckets with 100,000+ objects via S3 console or if versioning is enabled via AWS CLI.
  + Supports static website hosting.
* **Data Consistency Model**:
  + **Read-after-write consistency**: For PUTS of new objects, HEAD/GET requests, overwrite PUTS, and DELETES in all regions.
  + **Strong consistency**: For all storage requests.
  + **Eventual consistency**: For listing buckets after deletion or enabling versioning for the first time.
* **Storage Classes**:
  + **Frequently Accessed**:
    - **S3 Standard**: General-purpose, frequent access, stored across ≥3 AZs, 99.99% availability, 99.999999999% (11 9’s) durability.
    - **S3 Express One Zone**: High-performance, single AZ, single-digit millisecond latency, 10x faster access, 50% lower request costs, 99.95% availability, 512 KB minimum capacity charge, 1-hour minimum storage duration.
  + **Infrequently Accessed**:
    - **S3 Standard-IA**: Long-lived, less frequent access, ≥3 AZs, 99.99% availability, 128 KB minimum capacity, 30-day minimum storage duration, per-GB retrieval fee.
    - **S3 One Zone-IA**: Single AZ, 99.5% availability, 128 KB minimum capacity, 30-day minimum storage duration, per-GB retrieval fee, less resilient to AZ loss.
  + **S3 Intelligent-Tiering**:
    - Automatically moves objects between frequent and infrequent access tiers based on access patterns.
    - Moves objects to archive access tier after 90 days and deep archive after 180 days of no access.
    - No retrieval fees, 30-day minimum storage duration, 99.99% availability.
  + **S3 Glacier**:
    - **Instant Retrieval**: Rarely accessed, millisecond retrieval, 128 KB minimum capacity, 90-day minimum storage duration, per-GB retrieval fee, resilient to AZ loss.
    - **Flexible Retrieval**: Archive data accessed 1-2 times/year, retrieval in minutes to hours, 40 KB minimum capacity, 90-day minimum storage duration, per-GB retrieval fee.
      * Retrieval options: Expedited (1-5 minutes), Standard (3-5 hours), Bulk (5-12 hours).
    - **Deep Archive**: Rarely accessed (once/year), lowest cost, retrieval within 12-48 hours, 40 KB minimum capacity, 180-day minimum storage duration, per-GB retrieval fee.
  + Objects in Glacier classes are not available for real-time access; must be restored first.
* **S3 on Outposts**:
  + Delivers object storage to on-premises AWS Outposts using S3 APIs.
  + Data encrypted with SSE-C/SSE-S3, stored redundantly across Outposts servers.
  + Supports AWS DataSync for automated data transfer, access points, and lifecycle rules.
* **S3 API**:
  + **REST**: Uses HTTP requests for bucket/object operations; supports virtual hosting via HTTP Host header.
  + **SOAP**: Deprecated for new features, available over HTTPS only.
  + **Operations**: Supports HeadObject, ListObjects, ListObjectsV2 via S3 Object Lambda.
* **Bucket Configurations**:
  + **Location**: Specify AWS Region for bucket creation.
  + **Policy and ACL**: Manage bucket-level permissions (private by default).
  + **CORS**: Allows cross-origin requests for web applications.
  + **Website**: Configures static website hosting.
  + **Logging**: Tracks access requests (requester, bucket, action, etc.).
  + **Event Notification**: Notifies on events (e.g., object creation/removal) to SNS, SQS, or Lambda.
  + **Versioning**: Keeps multiple object versions, protects against overwrites/deletions, enabled explicitly.
  + **Lifecycle**: Defines rules for transitioning objects to other storage classes or expiration.
  + **Cross-Region Replication (CRR)**: Asynchronously copies objects across regions.
  + **Tagging**: Manages tags for cost allocation (up to 20 KB bucket policy size).
  + **RequestPayment**: Charges requesters for downloads instead of bucket owner.
  + **Transfer Acceleration**: Uses CloudFront edge locations for faster, secure file transfers.
* **S3 Objects**:
  + Private by default; permissions granted via policies/ACLs.
  + Comprise data, key, and metadata (system or user-defined).
  + Metadata cannot be modified post-upload.
  + **System Metadata** (examples):
    - **Date**, **Content-Length**, **Last-Modified**, **Content-MD5**: Non-modifiable.
    - **x-amz-server-side-encryption**, **x-amz-storage-class**, **x-amz-website-redirect-location**: Modifiable.
    - **x-amz-version-id**, **x-amz-delete-marker**: Non-modifiable, versioning-related.
  + Supports uploads up to 5 GB in a single operation; 5 TB via multipart upload API.
  + **Tagging**: Up to 10 tags per object, unique keys, 128-character key length, 256-character value length, case-sensitive.
  + **Object Delete**:
    - Non-versioned: Specify key only.
    - Versioned: Specify key and version ID; delete marker added for non-versioned DELETE.
    - MFA Delete: Requires MFA for versioning state changes or permanent version deletion.
  + **Object Lock**: Prevents deletion/overwrites for a retention period or legal hold (versioned buckets only).
  + **Object Ownership**: Bucket-owner-full-control ACL assumes ownership of uploaded objects.
  + **S3 Select**: Retrieves specific data from CSV, JSON, Parquet, or BZIP2-compressed objects, monitored via CloudWatch.
* **Lifecycle Management**:
  + Defines rules for transitioning objects to other storage classes or expiring them.
  + Transition actions: Move to S3-IA/One Zone-IA after 30 days.
  + Expiration actions: Delete expired objects automatically.
* **Data Integrity Protections**:
  + Uses CRC-based checksums during uploads, stored in metadata.
  + Verifies integrity during access/downloads, auto-detects/repairs corruption.
  + Applies to all storage classes at no extra cost.
  + Provides verification reports for data health monitoring.
* **S3 Metadata (Preview)**:
  + Enhances object metadata management for discovery and AI/ML workflows.
  + Supports system, custom, and event metadata.
  + Stores metadata in read-only Apache Iceberg tables in S3 table buckets.
  + Queryable via Athena, EMR, Redshift, Spark, Trino; integrates with AWS Glue Data Catalog and QuickSight.
* **S3 Tables**:
  + Stores tabular data (e.g., transactions, sensor data) in Apache Iceberg format.
  + **Table Buckets**: Optimized for high TPS and query throughput.
  + **Features**:
    - Built-in Apache Iceberg support with schema/partition evolution.
    - Automated table optimization (compaction, snapshot management).
    - IAM-based access control, public access always enabled.
    - Integrates with AWS analytics services (Athena, Redshift, QuickSight).
  + Currently in preview.
* **Storage Browser for Amazon S3**:
  + Open-source React component for web/intranet apps to browse, upload, download, copy, delete S3 data.
  + Supports LIST, GET, PUT, COPY, UPLOAD, DELETE operations.
  + Ensures high throughput, checksum-validated uploads, displays only authorized data.
  + Compatible with all S3 storage classes except Glacier Flexible Retrieval, Deep Archive, and certain Intelligent-Tiering tiers.
* **S3 Pricing**:
  + Pay-as-you-go for storage, requests, retrievals, early deletes, storage management, and bandwidth.
  + **No Charges**: Bucket creation, data ingress, egress to EC2/CloudFront in same region.
  + **Charges**:
    - Storage: Based on size, duration, and storage class.
    - Requests: For GET, lifecycle, etc., based on request type.
    - Retrievals: For S3 Standard-IA, One Zone-IA, Glacier classes.
    - Early Deletes: For objects deleted before minimum storage duration.
    - Bandwidth: For data transfers (except exempted cases).
    - Transfer Acceleration: Additional fee for CloudFront-based transfers.
* **S3 Networking**:
  + **Hosted-Style Access**:
    - Default endpoint: s3.amazonaws.com (routes to US East if not region-specific).
    - Format: http://bucket.s3.amazonaws.com or http://bucket.s3-aws-region.amazonaws.com.
  + **Path-Style Access**:
    - Endpoint must match bucket’s region: http://s3.amazonaws.com/bucket or http://s3-aws-region.amazonaws.com/bucket.
  + **Custom URLs**: Use CNAMEs matching bucket name.
  + **Transfer Acceleration**: Uses bucket.s3-accelerate.amazonaws.com, cannot be disabled but can be suspended.
* **S3 Security**:
  + **Policies**:
    - **Resource-Based**:
      * **Bucket Policies**: Centralized access control, supports conditions (e.g., IP address), 20 KB size limit.
      * **ACLs**: Grant permissions to other AWS accounts, limited to 100 grants per ACL, recommended only for S3 Log Delivery group write access.
    - **User Policies**: Managed via IAM, supports access keys and temporary credentials.
  + **Encryption**:
    - **Server-Side**: SSE-S3, SSE-KMS, SSE-C.
    - **Client-Side**: AWS KMS customer-managed key or client-side master key.
  + **MFA Delete**: Requires MFA for versioning state changes or permanent version deletion.
  + **Cross-Account Access**:
    - Methods: Resource-based policies, ACLs, IAM roles.
    - Supports access points and Multi-Region access point failover.
  + **Requester Pays**: Requesters pay for request and download costs; bucket owner pays storage costs.
* **S3 Monitoring**:
  + **Tools**:
    - **CloudWatch Alarms**: Monitor metrics over time, trigger actions.
    - **CloudTrail**: Logs API calls, supports real-time monitoring via CloudWatch Logs.
  + **Metrics**:
    - **Daily Storage Metrics**: Bucket storage data, readable via CloudWatch.
    - **Request Metrics**: Monitor requests at 1-minute intervals, up to 1000 configurations per bucket.
    - **Event History**: Records S3 events in CloudTrail.
* **Website Hosting**:
  + Enabled via bucket properties, uses region-specific website endpoint.
  + Requires objects to be publicly readable via s3:GetObject permission.
  + **Key Differences**:
    - REST API: Supports public/private content, all operations, XML error responses, SSL.
    - Website Endpoint: Public content only, GET/HEAD requests, HTML error responses, no SSL, supports redirects.
    - Root GET/HEAD: REST returns object key list; website returns index document.
* **S3 Event Notifications**:
  + Configurable for events like object creation, removal, or RRS object loss.
  + Destinations: SNS, SQS, Lambda.
* **Cross-Region Replication (CRR)**:
  + Asynchronously copies objects across regions for compliance, latency reduction, or efficiency.
  + **Requirements**:
    - Versioning enabled on source/destination buckets.
    - Different regions for source/destination.
    - S3 permissions to replicate objects.
    - Object owner must grant READ/READ\_ACP permissions if not bucket owner.
  + **Replicated**:
    - Objects created post-replication configuration.
    - Unencrypted objects, SSE-S3, SSE-KMS (if enabled).
    - Metadata, tags, ACL updates (if ownership unchanged).
  + **Not Replicated**:
    - Pre-existing objects, SSE-C objects, replicas, lifecycle actions, unauthorized objects.
  + **Delete Operations**:
    - Non-versioned DELETE adds delete marker, replicated.
    - Versioned DELETE (with version ID) deletes source object, not replicated.
* **S3 Batch Operations**:
  + Manages billions of objects for tasks like copying, tagging, ACL updates, or restoring Glacier objects.
* **S3 Storage Lens**:
  + Provides analytics to visualize trends, flag outliers, and offer recommendations for S3 data.

Amazon S3-related Cheat Sheets:

* [Amazon S3 vs EBS vs EFS](https://tutorialsdojo.com/amazon-s3-vs-ebs-vs-efs/)
* [Amazon S3 vs Glacier](https://tutorialsdojo.com/amazon-s3-vs-glacier/)
* [S3 Standard vs S3 Standard-IA vs S3 One Zone-IA](https://tutorialsdojo.com/s3-standard-vs-s3-standard-ia-vs-s3-one-zone-ia/)
* [S3 Pre-signed URLs vs CloudFront Signed URLs vs Origin Access Identity (OAI)](https://tutorialsdojo.com/s3-pre-signed-urls-vs-cloudfront-signed-urls-vs-origin-access-identity-oai/)
* [S3 Transfer Acceleration vs Direct Connect vs VPN vs Snowball vs Snowmobile](https://tutorialsdojo.com/s3-transfer-acceleration-vs-direct-connect-vs-vpn-vs-snowball-vs-snowmobile/)

## Amazon S3 Cheat Sheet References:

<https://aws.amazon.com/blogs/aws/introducing-default-data-integrity-protections-for-new-objects-in-amazon-s3/><https://aws.amazon.com/about-aws/whats-new/2024/12/amazon-s3-metadata-preview/><https://docs.aws.amazon.com/AmazonS3/latest/userguide/metadata-tables-overview.html><https://docs.aws.amazon.com/AmazonS3/latest/userguide/storage-browser.html>[https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html  
https://aws.amazon.com/s3/faqs/](https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html)

**AWS block storage**

* The AWS block storage portfolio consists of two types of block storage services—instance stores and Amazon Elastic Block Store (Amazon EBS).

### AWS Instance Storage

#### What is AWS Instance Storage?

* Instance storage, also known as Amazon EC2 instance stores, provides temporary (ephemeral) block-level storage for EC2 instances.
* It consists of disks physically attached to the host computer where the EC2 instance runs, resembling direct-attached disk drives.
* Data is stored locally on the instance's hardware, exposed as block devices, and is not replicated or spread across multiple devices for durability.
* Unlike Amazon EBS (Elastic Block Store), instance storage is non-persistent and tied to the instance's lifecycle.

#### Why Use AWS Instance Storage?

* To achieve high-performance, low-latency storage for temporary data that doesn't require long-term persistence.
* Ideal when speed and throughput are prioritized over durability, as it leverages local hardware for faster access compared to network-attached storage like EBS.
* Helps in scenarios where data can be easily regenerated or replicated, reducing costs and complexity for non-critical workloads.
* Supports specific use cases in AWS environments where ephemeral storage aligns with application needs, such as in scalable fleets.

#### Types of AWS Instance Storage

* NVMe-based Instance Stores: Provide sub-millisecond latency and high IOPS, suitable for latency-sensitive applications.
* HDD-based Instance Stores: Optimized for throughput-based performance, ideal for sequential workloads requiring high data transfer rates.
* Availability depends on EC2 instance type (e.g., i3, m5d, r5d families support NVMe; d2 supports HDD).
* Instance stores can consist of one or more volumes, with size and number varying by instance type (e.g., up to 48 TiB on certain instances).

#### Features of AWS Instance Storage

* Low Latency: Sub-millisecond access times, especially with NVMe, outperforming EBS in speed due to local attachment.
* High IOPS and Throughput: Delivers very high input/output operations per second and bandwidth for demanding workloads.
* Ephemeral Nature: Data persists during reboots but is lost on instance stop, termination, hibernation, or underlying disk failure.
* TRIM Support: Allows efficient space management on SSD-based stores.
* No Additional Cost: Included with the instance price, unlike EBS which incurs separate volume charges.
* Integration: Configured at launch, supports RAID configurations for improved performance or redundancy (user-managed).

#### Use Cases for AWS Instance Storage

* Temporary Storage: For buffers, caches, scratch data, or intermediate computation results in big data processing (e.g., Hadoop clusters).
* Replicated Data: In load-balanced web server pools where data is mirrored across instances, such as session data or temporary logs.
* High-Performance Computing: For applications like genomics analysis or machine learning training that need fast local I/O but can tolerate data loss.
* NoSQL Databases: As temporary storage in databases like Cassandra or MongoDB for non-persistent nodes in a cluster.
* Media Processing: Handling transient files in video rendering or transcoding workflows where data is regenerated if lost.

#### Benefits of AWS Instance Storage

* Performance Boost: Faster than EBS due to local hardware attachment, enabling sub-millisecond latency and high IOPS (e.g., up to millions of IOPS on large instances).
* Cost-Effective: No extra charges beyond the instance cost; ideal for short-lived, high-throughput needs without paying for persistent storage.
* Simplicity: Easy to provision at instance launch; no need for separate volume management like snapshots or attachments.
* Scalability: Supports large volumes on compatible instances, allowing for massive temporary storage (e.g., petabyte-scale on certain types).
* Reliability for Temporary Data: Persists through reboots, providing some stability for short-term operations.

#### Disadvantages of AWS Instance Storage

* Data Loss Risk: Non-persistent; data is deleted on instance stop, termination, hibernation, or hardware failure, making it unsuitable for critical data.
* Limited Availability: Only supported on specific EC2 instance types; cannot be added post-launch or detached/remounted like EBS.
* No Built-in Redundancy: Not replicated, so single point of failure if the host hardware fails; users must handle replication manually.
* Size and Configuration Constraints: Volume size and count are fixed by instance type, limiting flexibility compared to EBS.
* Not for Long-Term Storage: Requires alternative solutions like EBS or S3 for durable data, increasing architectural complexity.

#### Examples of AWS Instance Storage

* Web Application Caching: Using instance store on an m5d instance for Redis caching layer, where cache data is temporary and can be rebuilt from a database.
* Big Data Processing: In an EMR cluster on i3 instances, storing intermediate MapReduce results on NVMe instance stores for fast access during computations.
* Load Balancer Fleet: A pool of r5d instances for an Auto Scaling group, replicating session data across nodes to handle web traffic without persistent storage.
* AI Model Training: Temporary scratch space on p3dn instances for holding datasets during GPU-accelerated training, where data is reloaded from S3 if lost.
* Throughput-Intensive Tasks: Using d2 instances with HDD instance stores for sequential data processing in log analysis pipelines.

#### Other Key Notes

* Comparison to EBS: EBS offers persistent, network-attached storage with snapshots and replication; instance store is local, faster but ephemeral—use EBS for databases or valuable data.
* Lifetime Details: Data survives intentional/unintentional reboots but is lost on stop/terminate/hibernate/disk failure; always back up important data to EBS/S3.
* Best Practices: Combine with EBS for hybrid setups (e.g., OS on EBS, temp data on instance store); use RAID 0 for striping to maximize performance, or RAID 1 for mirroring (though still ephemeral).
* Certification/Interview Tips: Remember instance store is "ephemeral" and "instance-type dependent"; key differentiator from EBS is no data persistence post-stop; often tested in scenarios involving performance vs. durability trade-offs.
* Practical Tasks: When launching an EC2 instance, select types like i3en for NVMe stores; monitor with CloudWatch for IOPS; script data replication using tools like rsync for fault tolerance.
* Limitations: Cannot be resized, snapshotted, or encrypted by default (use OS-level encryption); not available on all regions or instance families—check AWS documentation for compatibility.

### What is AWS EBS?

* **Definition**: Amazon Elastic Block Store (EBS) is a high-performance block storage service designed for use with Amazon EC2 instances, providing persistent block-level storage volumes that behave like raw, unformatted block devices.
* **Core Functionality**: EBS volumes can be mounted as devices on EC2 instances, allowing creation of file systems or use in any block device manner; they persist independently from the instance's lifecycle.
* **Comparison to Other Storage**: Similar to a network-attached HDD or SAN (Storage Area Network), but replicated within an Availability Zone (AZ) for durability; not physically on the host but network-attached.

### Why Use AWS EBS?

* **Purpose**: To provide persistent data storage for EC2 instances, ensuring data survives instance stops, terminations, or hibernations (except default root volumes on termination).
* **Necessity for Workloads**: Supports transaction-intensive (e.g., databases) and throughput-intensive (e.g., big data) workloads; ideal when data must persist beyond instance life, unlike ephemeral instance store.
* **Scalability and Flexibility**: Allows dynamic changes to volume size, type, and performance without downtime, making it suitable for mission-critical systems requiring high availability.

### Types of EBS Volumes

* **SSD-Based Volumes (For Transactional Workloads)**:
  + **General Purpose SSD (gp3 and gp2)**: Balanced price/performance for general applications; gp3 offers provisioned IOPS/throughput, gp2 uses burst credits.
  + **Provisioned IOPS SSD (io1, io2, io2 Block Express)**: High-performance for demanding apps; io2 provides higher durability (99.999%), io2 Block Express scales to 64 TiB and 256,000 IOPS.
* **HDD-Based Volumes (For Throughput-Intensive Workloads)**:
  + **Throughput Optimized HDD (st1)**: For frequently accessed data like big data or log processing; up to 500 MB/s throughput.
  + **Cold HDD (sc1)**: Lowest-cost for infrequently accessed data; up to 192 MB/s throughput.
* **Legacy Type**:
  + **Magnetic (standard)**: Older, lower-performance option; not recommended for new workloads.

### Features of AWS EBS

* **Persistence and Management**: Volumes persist independently; can be detached from one EC2 instance and reattached to another in the same AZ.
* **Encryption**: Built-in at-rest and in-transit encryption using AWS KMS keys; protects data volumes, boot volumes, and snapshots.
* **High Availability and Durability**: Replicated within an AZ; durability ranges from 99.8-99.9% (most types) to 99.999% (io2/io2 Block Express); AFR of 0.001-0.2%.
* **Elastic Volumes**: Dynamically increase size, change type, or tune performance (IOPS/throughput) without downtime.
* **Multi-Attach**: Allows one io1/io2 volume to attach to up to 16 Nitro-based instances in the same AZ; requires app-level I/O fencing for consistency.
* **Snapshots**: Point-in-time backups to S3; incremental (only changed blocks saved); features include sharing, copying across Regions, resizing, and Fast Snapshot Restore (FSR) for instant access.
* **Monitoring**: Integrates with CloudWatch for metrics like bandwidth, throughput, latency, and queue length.
* **Backups**: Supported via AWS Backup for centralized, policy-based protection across services.
* **Limits and Defaults**: Default root volume size 8 GiB; volumes up to 64 TiB; AZ-specific (cannot attach across AZs without snapshot/migration).

### Use Cases

* **Boot Volumes**: Default root volume for EC2 OS; automatically created on instance launch.
* **Databases and Transactional Apps**: Provisioned IOPS SSD for SAP HANA, SQL Server, or DB2 requiring consistent high IOPS.
* **Virtual Desktops and Development**: General Purpose SSD for balanced performance in test environments or interactive apps.
* **Big Data and Log Processing**: Throughput Optimized HDD for MapReduce or streaming workloads needing high throughput.
* **Cold Storage**: Cold HDD for archival data with infrequent access.
* **Multi-Instance Shared Storage**: Multi-Attach for clustered apps managing shared volumes (e.g., high-availability databases).
* **Data Migration**: Snapshots for moving volumes across AZs or Regions (create snapshot, then new volume in target AZ).

### Benefits

* **Durability and Reliability**: Automatic replication prevents single-point failures; low AFR compared to commodity drives (0.001-0.2% vs. 2-4%).
* **Performance Optimization**: Choose volume types for specific needs; provision IOPS (up to 256,000) and throughput (up to 4,000 MB/s) independently of size.
* **Cost-Effectiveness**: Pay only for provisioned storage; no upfront costs; options like gp3 for baseline 3,000 IOPS/125 MiB/s at low cost.
* **Scalability**: Scale to petabytes using multiple volumes; dynamic modifications without impact.
* **Security**: Seamless encryption; integration with KMS for key management.
* **Backup and Recovery**: Incremental snapshots to S3 for geographic protection; quick restores with FSR.
* **Integration**: Works seamlessly with EC2, CloudWatch, and AWS Backup for monitoring and automation.

### Disadvantages

* **AZ Limitation**: Volumes are AZ-bound; cannot directly attach across AZs (requires snapshot and recreation in new AZ).
* **Single Attachment Default**: Only one instance per volume unless Multi-Attach enabled (limited to io1/io2 and Nitro instances).
* **Performance Dependencies**: IOPS/throughput tied to volume size in some types (e.g., gp2 max 16,000 IOPS requires larger sizes); burst limits in gp2/st1/sc1.
* **Costs for High Performance**: Provisioned IOPS types (io1/io2) are more expensive; additional fees for FSR or multi-attach (though multi-attach itself is free).
* **No Built-in Multi-Writer Consistency**: Multi-Attach requires application-level handling for I/O fencing.
* **Limits on Accounts**: Per-account limits on volumes and storage; may need requests for increases.
* **Not for Ephemeral Needs**: Overkill for temporary data (use instance store instead); root volumes delete on termination by default.

### Examples

* **Root Volume Creation**: Launching an EC2 instance automatically creates an 8 GiB gp2/gp3 root EBS volume for the OS.
* **Volume Attachment**: Attach a 100 GiB io2 volume (provisioned 10,000 IOPS) to an EC2 instance for a database; detach and reattach to another instance.
* **Migration Across AZs**: Take snapshot of EBS in AZ1, create new volume from snapshot in AZ2, then attach to instance in AZ2.
* **Performance Scaling**: Start with gp3 at 3,000 IOPS/125 MiB/s; dynamically increase to 16,000 IOPS/1,000 MB/s as workload grows.
* **Multi-Attach Scenario**: Attach one io2 volume to 8 Nitro instances for a shared file system in a cluster, with app handling consistency.
* **Snapshot Backup**: Create incremental snapshot of a 1 TiB volume; only changed blocks since last snapshot are stored in S3.
* **Encryption Example**: Enable KMS encryption on a boot volume to secure OS data at-rest and in-transit.

### Other Key Notes

* **IOPS and Throughput Details**: Baseline IOPS = 3 per GiB (gp2/gp3 min 3,000); e.g., 100 GiB gp2 = up to 300 IOPS baseline, burst to 3,000; io2 max 256,000 IOPS on Block Express.
* **Size and Limits**: Min 1 GiB (SSD) or 125 GiB (HDD); max 16 TiB (most) or 64 TiB (io2 Block Express); throughput limits vary (e.g., gp3 max 1,000 MB/s).
* **Burst Credits**: gp2/st1/sc1 use credits for temporary high performance; credits accrue when below baseline.
* **Nitro Dependency**: Higher IOPS/throughput and Multi-Attach require Nitro-based instances.
* **Billing**: Charged for provisioned storage, IOPS, throughput, and snapshots; no charge for Multi-Attach enabling.
* **Best Practices**: Monitor with CloudWatch to optimize; use AWS Backup for policies; change root volume termination behavior if persistence needed.
* **Comparison to Instance Store**: EBS for persistence; instance store for high-speed temporary storage (data lost on stop/terminate).
* **Exam/Interview Tips**: Remember AZ scope, snapshot for cross-AZ moves, volume types' IOPS/throughput formulas (e.g., 1 GiB = ~3 IOPS baseline for gp2), and durability percentages.

### AWS Elastic File System (EFS)

* **What is EFS?**
  + Amazon EFS is a fully managed, scalable, and elastic file storage service provided by AWS, designed for cloud-native applications and workloads.
  + It provides a simple, serverless Network File System (NFS) interface that allows multiple Amazon EC2 instances, containers (via ECS/EKS), and on-premises servers to access shared file storage concurrently.
  + EFS is built on Linux-based file systems and supports POSIX-compliant semantics, making it suitable for applications requiring shared access to files.
  + It operates as a regional service, automatically replicating data across multiple Availability Zones (AZs) in an AWS Region for durability.
* **Why Use EFS?**
  + To enable shared file access across multiple compute instances without managing underlying storage infrastructure, ideal for distributed applications.
  + For workloads needing high throughput, low latency, and automatic scaling as data grows, without provisioning fixed capacity.
  + To achieve high availability and durability in multi-AZ setups, reducing the need for custom replication solutions.
  + As a pay-per-use alternative to traditional on-premises NAS (Network Attached Storage) systems, simplifying migration to the cloud.
* **Types of EFS**
  + **Storage Classes**:
    - Standard: General-purpose storage for frequently accessed data, offering low-latency access and high durability (default option).
    - Infrequent Access (IA): Cost-optimized for infrequently accessed files, with automatic lifecycle management to move data between Standard and IA tiers.
  + **Throughput Modes**:
    - Bursting: Default mode; uses burst credits for high throughput during peaks, suitable for workloads with variable I/O patterns (scales with file system size).
    - Provisioned: Allows specifying a fixed throughput (in MiB/s) independent of storage size, ideal for consistent high-performance needs.
  + **Performance Modes**:
    - General Purpose: Optimized for latency-sensitive applications like web servers (default, up to 7,000 IOPS baseline).
    - Max I/O: Designed for high-throughput workloads with many concurrent connections, supporting higher IOPS (up to millions) but with slightly higher latency.
* **Features**
  + **Shared Access**: Supports concurrent read/write access from thousands of EC2 instances, ECS tasks, or EKS pods across multiple AZs.
  + **POSIX-Compliant**: Ensures compatibility with standard Linux file permissions, semantics, and commands (e.g., ls, cp, mv).
  + **Auto-Scaling**: Automatically grows and shrinks storage capacity as files are added or removed, with no downtime.
  + **Pay-as-You-Go Pricing**: Charges only for storage used, throughput provisioned, and data transfer; no upfront costs.
  + **Secure Access**: Integrates with IAM for access control, VPC security groups, and supports encryption at rest (using AWS KMS) and in transit (TLS).
  + **NFS Protocol Support**: Uses NFSv4.0 and NFSv4.1 protocols for mounting; compatible with on-premises via Direct Connect or VPN.
  + **Lifecycle Management**: Automatically transitions files to IA storage class after a specified period (e.g., 30 days) to optimize costs.
  + **Backup and Replication**: Integrates with AWS Backup for automated snapshots; supports cross-region replication for disaster recovery.
  + **Monitoring and Metrics**: CloudWatch integration for performance metrics like IOPS, throughput, and burst credits.
* **Use Cases**
  + **Content Management Systems (CMS)**: Shared storage for media files in applications like WordPress or Drupal, allowing multiple web servers to access the same content.
  + **Big Data Analytics**: Storing and sharing large datasets across compute clusters (e.g., with EMR or EC2 for Hadoop/Spark workloads).
  + **Web Serving and Development**: Hosting static assets or code repositories for load-balanced web fleets needing consistent file access.
  + **Enterprise Applications**: Shared file systems for databases (e.g., Oracle RAC) or SaaS applications requiring multi-instance access.
  + **Machine Learning Workloads**: Storing training datasets accessible by multiple GPU instances or EKS pods.
  + **Log and Data Sharing**: Centralizing logs from multiple servers for monitoring tools like ELK Stack.
* **Benefits**
  + High availability (99.99% SLA) with automatic multi-AZ replication, eliminating single points of failure.
  + Elastic scalability up to petabytes of data and millions of IOPS, handling growth without manual intervention.
  + Simplified management as a serverless service—no need to provision servers or manage RAID configurations.
  + Cost efficiency for variable workloads via bursting mode and lifecycle policies, reducing over-provisioning.
  + Seamless integration with AWS services like EC2, Lambda (via mount targets), ECS, EKS, and SageMaker.
  + Enhanced security with fine-grained access controls and compliance certifications (e.g., HIPAA, PCI DSS).
* **Disadvantages**
  + Higher cost compared to EBS (about 3x the price of gp2/gp3 volumes) or S3 for certain workloads; not ideal for cost-sensitive, low-I/O applications.
  + Limited to Linux/Unix-based systems; no native support for Windows EC2 instances (requires workarounds like NFS clients).
  + Potential latency issues in Max I/O mode or for very small files due to NFS overhead.
  + Mount targets required per VPC subnet (one per AZ), adding minor setup complexity and costs ($0.10/hour per mount target).
  + Not suitable for high-performance block storage needs (use EBS instead); throughput can burst but may throttle if credits deplete.
  + Data transfer fees apply for cross-AZ or internet access, which can accumulate in large-scale setups.
* **Examples**
  + **WordPress Hosting**: Mount EFS on multiple EC2 instances in an Auto Scaling group behind an ALB; all instances share the /wp-content directory for themes, plugins, and uploads.
  + **Shared Data Processing**: In a media transcoding pipeline, EFS stores input videos accessible by Lambda functions or EC2 workers for parallel processing.
  + **DevOps Workflow**: Use EFS as a shared volume for Jenkins servers, allowing build artifacts to be accessed across CI/CD pipelines in different AZs.
  + **Disaster Recovery**: Configure cross-region replication to a secondary EFS in another region; in failover, remount EC2 instances to the replica.
* **Other Key Notes**
  + **Lifecycle and Management**: EFS file systems have no fixed limits on size (grows to petabytes); up to 1,000 file systems per account, and each EC2 can mount up to 1,000 file systems (not 5 as sometimes misstated). Attach via mount targets in VPC subnets; use AWS CLI, SDK, or console for creation/mounting.
  + **Mounting Commands**: On Linux EC2, use sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-xxxx.efs.us-east-1.amazonaws.com:/ /mnt/efs. Check with df -h for usage, lsblk for block devices (though EFS is network-mounted, not block).
  + **Scalability and High Availability**: Supports horizontal scaling (add more clients) and vertical (increase throughput mode); tightly coupled with HA via multi-AZ design. Bursting mode provides 50 MiB/s per TiB baseline, bursting up to 100 MiB/s per TiB.
  + **Performance Tips for Practice**: For exams/interviews, remember IOPS limits: General Purpose caps at 35,000 read/7,000 write IOPS per file system; Max I/O scales to millions. Test mounting in labs: Create EFS, add security group ingress for NFS (port 2049), mount on EC2.
  + **Common Exam/Interview Questions**: Differences vs. EBS (block vs. file storage), vs. FSx (for Windows/Lustre), vs. S3 (object vs. file); how to optimize costs (use IA class); encryption requirements; troubleshooting mount failures (VPC/DNS issues).
  + **Limitations and Best Practices**: Maximum file size 47.9 TiB; supports up to 35,000 concurrent NFS connections. Always use DNS names for mounts to ensure AZ failover; enable access points for multi-tenant isolation in shared file systems.

### Amazon EBS (Elastic Block Store)

#### What

* Amazon EBS is a high-performance block storage service designed for use with Amazon EC2 instances, providing persistent, low-latency storage volumes that act like raw hard drives.

#### Why

* Offers scalable, durable block-level storage for demanding workloads, ensuring data persistence across instance stops/restarts.
* Provides high availability with automatic replication within an Availability Zone and 99.999% durability for certain volume types.
* Enables encryption, snapshot-based backups, and integration with AWS services for disaster recovery and compliance.

#### Use Cases

* Running databases (e.g., MySQL, Oracle, SAP HANA) requiring consistent IOPS and low latency.
* Boot volumes for EC2 instances or high-performance computing workloads like big data analytics (Hadoop, Spark).
* Migrating on-premises SAN storage to the cloud for mission-critical applications.

#### Features

* Supports snapshots for point-in-time backups stored in S3, with incremental updates for efficiency.
* Encryption at rest using AWS KMS keys and in-transit encryption.
* Integration with Amazon Data Lifecycle Manager for automated snapshot lifecycle policies.
* Multi-Attach support for limited shared access (up to 16 instances for io1/io2 volumes).

#### Pricing

* Pay per GB-month of provisioned storage, plus IOPS and throughput charges where applicable.
* Example rates (US East - N. Virginia, as of 2025): gp3 volumes ~$0.08/GB-month; io2 ~$0.125/GB-month; provisioned IOPS ~$0.065 per IOPS-month (io2).
* Snapshots: ~$0.05/GB-month; free inbound data transfer, outbound varies.
* No upfront costs; scales with usage, but check AWS Pricing Calculator for region-specific details.

#### Types

* **SSD-based (General Purpose)**: gp2/gp3 – Balanced for most workloads; gp3 allows independent IOPS/throughput provisioning.
* **SSD-based (Provisioned IOPS)**: io1/io2 – For high-performance needs; io2 Block Express offers ultra-high durability.
* **HDD-based (Throughput-Optimized)**: st1 – For big data and logs with sequential access.
* **HDD-based (Cold)**: sc1 – Cost-effective for infrequently accessed data.

#### Throughput Mode

* Bursting: For gp2/gp3, credits allow temporary bursts up to 3,000 IOPS/250 MB/s.
* Provisioned: For io1/io2, specify baseline throughput up to 1,000 MB/s per volume.

#### IOPS

* Baseline: gp3 up to 3,000 (included), burstable to 16,000; io2 up to 256,000 (provisioned).
* Measured in read/write operations per second; SSD types excel in random access.

#### Provisioned IOPS

* Allows specifying exact IOPS (up to 256,000 for io2) for predictable performance in latency-sensitive apps like databases.
* Charged per provisioned IOPS-month; ratio limit of 50:1 IOPS to GiB for io2.

### Amazon EFS (Elastic File System)

#### What

* Amazon EFS is a fully managed, scalable file storage service that provides a simple, shared file system (NFS-based) for thousands of EC2 instances and on-premises resources.

#### Why

* Enables seamless scaling to petabytes without provisioning, ideal for shared access across multiple instances.
* Offers 99.999999999% (11 9s) durability and up to 99.99% availability for reliable, multi-AZ access.
* Reduces management overhead with automatic scaling and lifecycle policies to optimize costs.

#### Use Cases

* Shared storage for DevOps workflows, like code repositories or configuration files across microservices.
* Big data analytics (e.g., Spark) or ML training where multiple instances need concurrent file access.
* Content management systems (CMS) or web serving for dynamic scaling without downtime.

#### Features

* Automatic scaling of throughput and storage; supports POSIX-compliant file systems.
* Encryption at rest/transit, VPC integration, and access control via IAM/policies.
* Lifecycle management to transition files to cheaper classes automatically.
* Regional access with multi-AZ redundancy.

#### Pricing

* Pay per GB-month used; no minimum or provisioning fees.
* Standard class: ~$0.30/GB-month; Infrequent Access (IA): ~$0.025/GB-month + $0.01/GB retrieval.
* Throughput: ~$0.06 per MB/s-month for provisioned; bursting is free up to baseline.
* Monitor via AWS Cost Explorer; lower TCO with IA/Archive classes (up to 97% savings).

#### Types

* **Standard**: For frequently accessed data with high throughput.
* **Infrequent Access (IA)**: For less-accessed files, with automatic transition.
* **Archive**: For long-term, rarely accessed data (up to 12 months minimum).

#### Throughput Mode

* **Bursting**: Automatically scales based on workload (up to 10x baseline, free).
* **Provisioned**: Specify fixed throughput (e.g., 100-10 GB/s) for predictable performance; charged per MB/s-month.

#### IOPS

* Scales automatically with file size (up to 35,000 per file) and throughput; ~3 IOPS per KiB for small files.
* Supports up to 500,000 IOPS per file system in bursting mode.

#### Provisioned IOPS

* Not directly provisioned like EBS; IOPS tied to throughput provisioning (e.g., 1 MB/s provides ~500 IOPS).
* Use for consistent performance in high-concurrency scenarios.

### Amazon EC2 Instance Store

#### What

* Instance Store is ephemeral, high-performance block storage physically attached to the host machine of an EC2 instance, providing temporary, low-latency disk space.

#### Why

* Delivers the lowest latency and highest IOPS/throughput for data-intensive tasks without network overhead.
* Cost-effective as it's included in instance pricing; ideal for transient data that doesn't require persistence.
* Supports replication across instance fleets for temporary caching or buffering.

#### Use Cases

* Scratch space for high-performance computing, like temporary data in ML training or big data processing.
* Caches, buffers, or logs in distributed systems (e.g., load-balanced web servers).
* Temporary storage in analytics engines where data is regenerated on failure.

#### Features

* Exposed as block devices (e.g., /dev/xvdb); varies by instance type (e.g., i3, d3 types offer NVMe SSDs).
* No snapshots or backups; data lost on instance stop/termination.
* High durability within the instance lifecycle but not across failures.

#### Pricing

* No additional charge; included in the EC2 instance hourly rate.
* Varies by instance type (e.g., m5d.large with 75 GB NVMe ~$0.096/hour total).
* Free for the duration of instance runtime; no separate storage fees.

#### Types

* **SSD-based**: NVMe or instance-store SSDs on types like i3 (up to 3.3 TB per instance), c5d.
* **HDD-based**: Magnetic disks on older types like h1, but largely replaced by SSDs.
* Varies by family: e.g., d3 for dense HDD, i4 for high-IOPS NVMe.

#### Throughput Mode

* Not configurable; hardware-dependent (e.g., up to 2,080 MB/s on i3en instances).
* Operates at full hardware speed without modes like bursting.

#### IOPS

* Extremely high: Up to 3.3 million read IOPS on i3en.3xlarge (hardware-limited).
* Optimized for random 4K reads/writes; no bursting needed.

#### Provisioned IOPS

* Not supported; performance is fixed by instance hardware, no provisioning options.
* For provisioned needs, use EBS instead.

### Differences Between EBS, EFS, and Instance Store

|  |  |  |  |
| --- | --- | --- | --- |
| Aspect | EBS | EFS | Instance Store |
| What | Persistent block storage for single/multi EC2. | Scalable shared file storage (NFS) for multiple EC2/on-prem. | Ephemeral block storage attached to host. |
| Why | Persistence, low latency, snapshots for DR. | Shared access, auto-scaling, zero management. | Lowest latency, temp data, no network. |
| Use Cases | Databases, boot volumes, single-instance apps. | Shared files, ML/big data, CMS. | Caches, scratch data, HPC temp processing. |
| Features | Snapshots, encryption, Multi-Attach (limited). | Auto-scale, lifecycle mgmt, POSIX. | High raw performance, no backups. |
| Pricing | Per GB + IOPS/throughput (~$0.08/GB gp3). | Per GB used (~$0.30/GB standard). | Included in instance (~$0.096/hour). |
| Types | gp3/io2 (SSD), st1/sc1 (HDD). | Standard/IA/Archive. | NVMe SSD/HDD by instance family (i3/d3). |
| Throughput Mode | Burst/provisioned (up to 1,000 MB/s). | Burst/provisioned (up to 10 GB/s). | Hardware-fixed (up to 2 GB/s). |
| IOPS | Up to 256,000 (provisioned). | Up to 500,000 (auto-scaled). | Up to 3.3M (hardware). |
| Provisioned IOPS | Yes (up to 256K for io2). | No (tied to throughput). | No (fixed by hardware). |

# Amazon EC2 Instance Store

## What is EC2 Instance Store?

* Ephemeral block storage physically attached to EC2 host machine (HDD/SSD).
* Local, not network-attached like EBS; part of instance’s core (CPU, RAM).
* Availability and size depend on instance type (e.g., i3, c5d); cannot add if not included.

## Why Use It?

* Lowest latency, high IOPS/throughput for data-intensive tasks.
* Cost-effective; included in instance price, no extra storage fees.
* Supports replication for caching/buffering in distributed systems.

## Use Cases

* Scratch space for HPC (e.g., ML training, big data).
* Caches, buffers, logs for web servers or distributed apps.
* Temporary storage for analytics with regenerable data.

## Example

* **Scenario**: Web app on i3.large caches session data on Instance Store (/dev/nvme1n1).
* **Benefit**: High IOPS (~206K), low latency, no extra cost; data loss on stop is okay as it’s transient.

## Features

* **Access**: Block devices (e.g., /dev/xvdb, /dev/nvme1n1); varies by instance.
* **Ephemeral**: Data lost on stop/termination; persists on reboot.
* **No Backups**: Not included in AMIs; no snapshots.
* **Durability**: Reliable during instance lifecycle.
* **Types**:
  + SSD (NVMe, e.g., i3, c5d, up to 3.3 TB).
  + HDD (older, e.g., h1; mostly replaced by SSD).
* **Performance**:
  + IOPS: Up to 3.3M (i3en.3xlarge); fixed, no bursting.
  + Throughput: Fixed (e.g., 2,080 MB/s on i3en); no configurable modes.
  + No provisioned IOPS (use EBS for that).
* **Pricing**: Included in instance rate (e.g., m5d.large, 75 GB NVMe, ~$0.096/hour).
* **Instance Variations**: Differs by family (e.g., d3 for HDD, i4 for high-IOPS NVMe).

# Amazon Elastic Block Store (EBS)

## What is Amazon EBS?

* Persistent block storage for EC2, acting as a virtual hard drive.
* Placed in a specific Availability Zone (AZ) with automatic replication for durability.
* Typically attached to one EC2 instance (Multi-Attach for io1/io2 supports up to 16 instances).

## Why Use EBS?

* **Persistence**: Data persists independently of EC2 instance lifecycle (survives stops/terminations).
* **High Durability**: Replicated within AZ; io2 Block Express offers up to 99.999% durability.
* **Performance/Cost Optimization**: SSD (IOPS-focused) and HDD (throughput-focused) types for varied workloads.
* **Security**: Supports encryption at rest and in transit via AWS KMS with minimal performance impact.

## Use Cases

* Primary storage for EC2 boot volumes or applications.
* Databases (e.g., MySQL, Oracle, PostgreSQL) needing consistent IOPS/low latency.
* Mission-critical apps requiring high-performance, durable storage.
* Big data/analytics (e.g., Hadoop, Spark) with resizable volumes.
* Testing/development using EBS Snapshots for production data.

## Example

* **Scenario**: A PostgreSQL database on an EC2 instance uses a 100 GB gp3 volume.
* **Implementation**: Configured with 3,000 IOPS and 125 MB/s throughput for low-latency queries. Snapshots back up data to S3 nightly.
* **Outcome**: Persistent, high-performance storage with ~$8/month cost (US East, 2025, ~$0.08/GB-month).

## Features

* **Volume Types**:
  + **SSD**:
    - General Purpose (gp2/gp3): Balanced cost/performance; gp3 allows independent IOPS/throughput (up to 16,000 IOPS, 1,000 MB/s).
    - Provisioned IOPS (io1/io2): High-performance for critical apps; io2 Block Express up to 256,000 IOPS, 99.999% durability.
  + **HDD**:
    - Throughput Optimized (st1): Low-cost for big data, sequential access.
    - Cold HDD (sc1): Cheapest for infrequent access (e.g., archival).
* **Snapshots**: Incremental, point-in-time backups stored in S3 for recovery/migration.
* **Elastic Volumes**: Modify capacity, IOPS, or type without downtime.
* **Multi-Attach**: io1/io2 volumes shareable across up to 16 instances.
* **Encryption**: Seamless at rest (KMS) and in-transit encryption.
* **Automation**: Amazon Data Lifecycle Manager for snapshot scheduling.
* **Performance**:
  + **IOPS**: gp3: 3,000 baseline, burst to 16,000; io2: up to 256,000 provisioned.
  + **Throughput**: gp2/gp3 burst to 250 MB/s; io1/io2 up to 1,000 MB/s.
  + **Bursting**: gp2/gp3 use credits for temporary performance boosts.
* **Pricing** (US East, 2025):
  + gp3: ~$0.08/GB-month; io2: ~$0.125/GB-month.
  + Snapshots: ~$0.05/GB-month.
  + Provisioned IOPS charged per IOPS-month; no upfront costs.
  + Use AWS Pricing Calculator for region-specific rates.

# Amazon Elastic File System (EFS)

## What is Amazon EFS?

* Fully managed, scalable Network File System (NFS) using NFSv4 protocol.
* File-level storage for multiple EC2 instances and on-premises resources via AWS Direct Connect/VPN.
* Data redundantly stored across multiple Availability Zones (AZs) for high durability.

## Why Use EFS?

* **Shared Access**: Supports thousands of concurrent connections from EC2 instances or on-premises servers.
* **Elastic Scaling**: Automatically grows/shrinks as files are added/removed, no provisioning needed.
* **High Durability**: Offers 99.999999999% durability and up to 99.99% availability with multi-AZ redundancy.
* **Simplicity**: Fully managed; AWS handles maintenance, patching, and backups.

## Use Cases

* Web serving/content management for shared codebases or media files.
* Big data/analytics (e.g., Hadoop, Spark) with centralized, petabyte-scale storage.
* Media processing (e.g., video editing, rendering) for concurrent access to large files.
* Home directories for resilient, shared user storage.
* Lift-and-shift migrations of POSIX-compliant enterprise apps to AWS.

## Example

* **Scenario**: A web app on multiple EC2 instances uses EFS to store shared media files.
* **Implementation**: EFS Standard mounts provide low-latency access; lifecycle policies move older files to EFS IA for cost savings.
* **Outcome**: Scales to handle traffic spikes, costs ~$0.30/GB-month (Standard) or ~$0.025/GB-month (IA), with no provisioning overhead.

## Features

* **Storage Classes**:
  + Standard: High-throughput for frequent access (~$0.30/GB-month).
  + Infrequent Access (IA): Cost-optimized for less-accessed files (~$0.025/GB-month + $0.01/GB retrieval).
  + Archive: For rarely accessed data (up to 12 months).
* **Performance Modes**:
  + General Purpose: Low-latency for most workloads.
  + Max I/O: High parallelization for big data, with slightly higher latency.
* **Throughput**:
  + Bursting: Scales up to 10x baseline (free, tied to file system size).
  + Provisioned: Fixed 100 MB/s–10 GB/s (~$0.06/MB/s-month).
* **IOPS**: Auto-scales with size (up to 35,000/file, 500,000/system); ~3 IOPS/KiB for small files; tied to throughput (1 MB/s ~500 IOPS).
* **Security**: Encryption at rest/transit, IAM, and VPC Security Groups for access control.
* **Management**: Auto-scaling, lifecycle policies for cost optimization, and EFS Automatic Backups.
* **Pricing** (US East, 2025): Pay-as-you-go; no minimum fees; use AWS Pricing Calculator for region-specific rates.

## **Ch # 7**

**AWS Database Services**



### [**AWS Database Services**](https://tutorialsdojo.com/aws-cheat-sheets-database-services/)

* [Amazon Aurora](https://tutorialsdojo.com/amazon-aurora/)
* [Amazon DocumentDB](https://tutorialsdojo.com/amazon-documentdb/)
* [Amazon DynamoDB](https://tutorialsdojo.com/amazon-dynamodb/)
* [Amazon ElastiCache](https://tutorialsdojo.com/amazon-elasticache/)
* [Amazon MemoryDB for Redis](https://tutorialsdojo.com/amazon-memorydb-for-redis/)
* [Amazon Neptune](https://tutorialsdojo.com/amazon-neptune/)
* [Amazon Quantum Ledger Database (QLDB)](https://tutorialsdojo.com/amazon-quantum-ledger-database-qldb/)
* [Amazon RDS](https://tutorialsdojo.com/amazon-relational-database-service-amazon-rds/)
* [Amazon Redshift](https://tutorialsdojo.com/amazon-redshift/)
* [Amazon Redshift Serverless](https://tutorialsdojo.com/amazon-redshift-serverless/)

**Other Database-Related Notes:**

* [Amazon Aurora Machine Learning](https://tutorialsdojo.com/amazon-aurora-machine-learning/)
* [Aurora Serverless Tutorial – Part 1](https://tutorialsdojo.com/aurora-serverless-tutorial-part-1/)
* [Aurora Server Tutorial – Part 2](https://tutorialsdojo.com/aurora-serverless-tutorial-part-2/)
* [Calculating Required RCU And WCU For Your DynamoDB Table](https://tutorialsdojo.com/calculating-the-required-read-and-write-capacity-unit-for-your-dynamodb-table/)
* [DynamoDB Scan vs Query](https://tutorialsdojo.com/dynamodb-scan-vs-query/)
* [Global Secondary Index vs Local Secondary Index](https://tutorialsdojo.com/global-secondary-index-vs-local-secondary-index/)
* [Harnessing Real-Time Slack Notifications for Amazon RDS Event Monitoring](about:blank)
* [Lambda Integration With Amazon DynamoDB Streams](https://tutorialsdojo.com/aws-lambda-integration-with-amazon-dynamodb-streams/)
* [Redis (Cluster Mode Enabled vs Disabled) vs Memcached](https://tutorialsdojo.com/redis-cluster-mode-enabled-vs-disabled-vs-memcached/)
* [Redis Append-Only Files vs Redis Replication](https://tutorialsdojo.com/redis-append-only-files-vs-redis-replication/)

## **Ch # 7**

**AWS Monitoring Services**



## **Cloudwatch**

### **Cloudwatch Metrics**

* Cloudwatch provide metrics for every AWS Service
* Metrics is a variable to monitor, such as
* CPU Utilization
* Networking data
* Metric belong to Namespace
* Namespace are similar to Group
* Metric dimensions are Attribute, like
* Instance ID
* Environment Name
* Each Metric can have up to 10 Dimensions
* Metrics have Timestamps
* Using metric, the Cloudwatch Dashboard is generated

### **Cloudwatch Detailed Monitoring**

* By default, EC2 have metrics each 5 minutes
* With Detailed Monitoring
* Metric generate every 1 Minute
* Good for ASG
* Free Tier allows 10 Detail Monitoring
* For EC2 Memory Usage, there is no default metric. Need to use Custom Metric

### **Cloudwatch Custom Metric**

* Can send custom metrics to Cloudwatch
* Ability to send Dimension
* instance.id
* environment.name
* Metric Resolution
* Standard 1 Minute
* High Resolution, up to 1 Sec
* To send custom metric use PutMetricData

### **Cloudwatch Dashboard**

* Dashboard are Global (cross region, cross account)
* Dashboard Graph includes different Region
* Dashboard Graph includes different Account
* Can setup Auto Refresh
* Pricing
* 3 Dashboard (Up to 50 Metrics) free
* After free tier, 3 dollar/dashboard/per month

### **Cloudwatch Logs**

* Logs can be send to Cloudwatch through SDK
* Cloudwatch collect log from
* Elastic Beanstalk
* ECS
* AWS Lambda
* VPC Flow Logs
* API Gateway
* Cloudtrail
* Cloudwatch Log Agent (From EC2 Instance)
* Route 53 (DNS Query)
* Logs go to
* S3 to store or archive
* Stream to Elastic Search for analytics
* Log Storage Architecture
* Groups: Log is grouped under name
* Each group has streams of logs
* Can define expiration period (After the expiration period, the logs will be deleted)
* Encryption
* KMS can be used to encrypt the logs
* Encryption is done in log group level
* Using encryption key, both new (create-log-group) and existing (associate) log group can be encrypted
* Encryption operation can be done only by CLI or SDK
* To send logs, make sure the Permission to write logs are set
* To follow/tail logs, we can use AWS CLI
* Possible to filter by expression
* Helpful to find logs or specific IP
* Can use to trigger alarm
* Cloudwatch Logs Insights
* Can be used to query logs
* To use, need to install the Unified Cloudwatch Logs Event

### **Cloudwatch Logs Agent Vs Unified Agent**

Logs Agent

* Old Version

Unified Agent

* Newer version
* Get Additional system level metrics
* Can use SSM Parameter Store to centralized configuration

### **Cloudwatch Alarms**

* Alarms are used to trigger notification for any metric
* Alarms can go to
* ASG
* EC2 Actions
* SNS Notifications
* Alarm can raise
* Sampling value
* Percentage value
* Max or Min value
* Alarm States
* OK (When everything is alright)
* INSUFFICIENT\_DATA (When not enough data to measure it its OK or ALARM state)
* ALARM (When metrics reached the Threshold)
* Period
* Time length to evaluate the metric
* In case of High Resolution Metric, period can be 10 sec

### **Creating Cloudwatch Event**

While creating a cloudwatch event, we can set

1. Period: Define evaluation time in seconds.
2. Evaluation Period / Number of Data Point: Known as Data Point. Number of recent Period to consider to generate a alarm state
3. DataPoints To Alarm: Determine to go to ALARM state. We can define how many period can be reached within a evaluation period to go to ALARM

### **Cloudwatch Event**

* Can schedule CRON Jobs
* Event Pattern
* Rules on react a service doing something
* Example: Code Pipeline state change
* Can trigger
* Lambda Function
* SQS
* SNS
* Kinesis
* Cloudwatch Event create a sample document to give information about the change
* Use case in S3 and Code Pipeline
* Code can be uploaded to S3
* Cloudwatch Event trigger the Code Pipeline
* Code will be deployed to the Elastic Beanstalk
* Can be used to change the number of Fargate Cluster should run according to the events
* Allow monitor jobs in the batch jobs

### **Cloudwatch Agent**

* Collect system info and log files
* Can track memory, swap and disk space

## **Cloudtrail**

* All the AWS history and events are stored here, including
* Console
* SDK
* CLI
* AWS Service
* Cloudtrail is enabled by default
* Can put the logs to CloudWatch Logs
* Example, if need to trace, who change the resource, need to go to Cloudtrail
* The logs are encrypted by default
* By default Cloudtrail Logs are encrypted by S3 Server Side Encryption
* Also we can use KMS for encryption
* Enabling Cloudtrail Log File Integrity ensure
* Non compliance log
* Generate public and private key of the logs
* Put the digest in separate folder
* Cloudtrail Global Event Logs can only done by AWS CLI, not Console
* To monitor API calls in the Redshift Cluster, need to use Cloudtrail