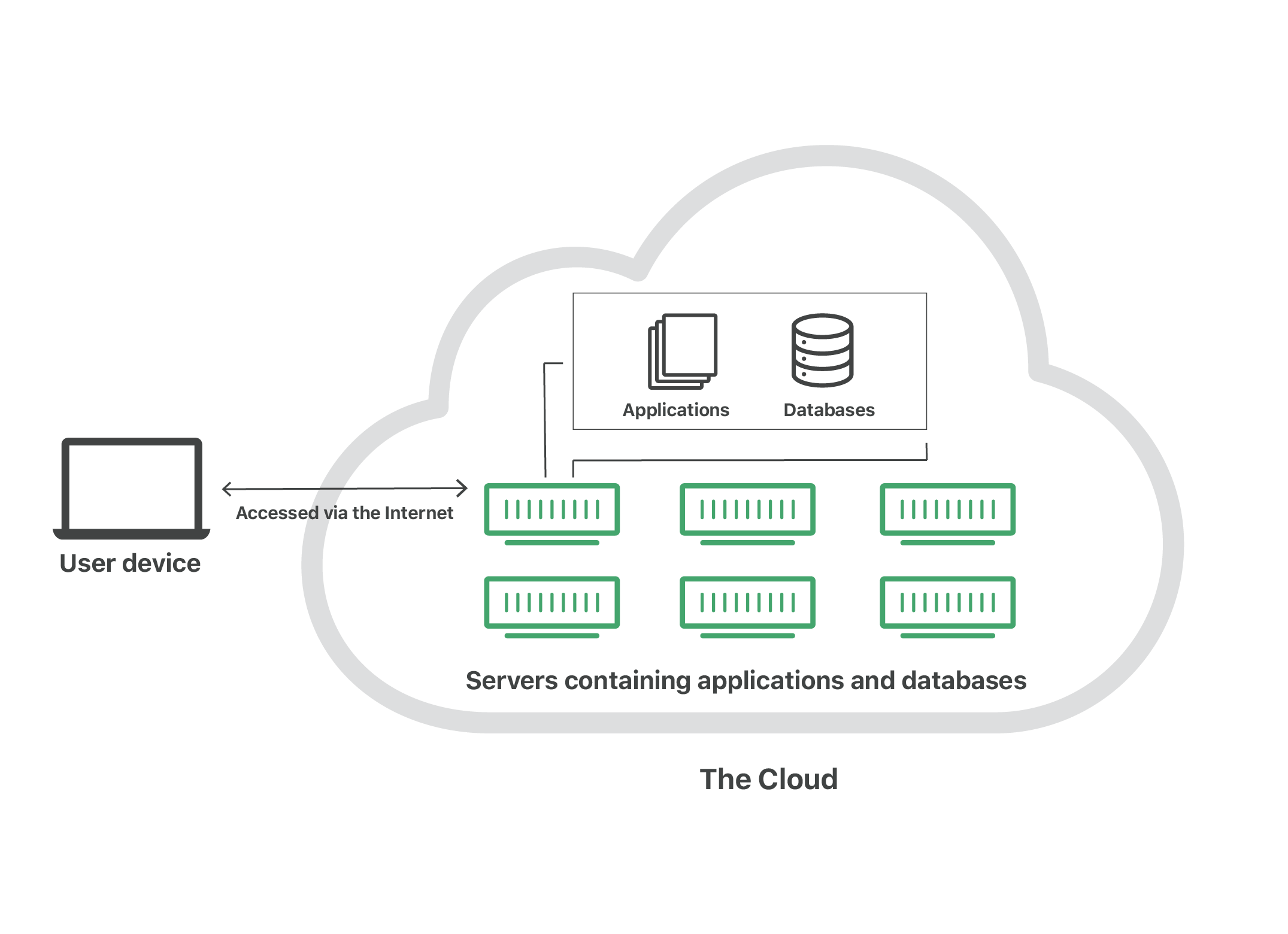
**Azure Notes:**

**Cloud:** The cloud is a distributed collection of servers that hosts software and infrastructure, and it is accessed over the internet.

The cloud is made up of a vast network of computer servers located around the globe along with the data, content, applications, databases, and other computing resources that reside on these servers.

The cloud enables businesses and consumers to access computing resources on demand via the internet, rather than installing their own physical servers, running their own software, and managing their own databases.

The cloud allows users to access data, applications, and computing resources from anywhere in the world, rather than needing to be connected to a computer in an office.



**Cloud Computing:** Cloud computing refers to the on-demand availability of computing resources, such as data storage, servers, networking capabilities, and software, over the internet.

This model allows users to access and utilize these resources without the need for direct active management or ownership of the physical infrastructure

Cloud computing is a way for providing computing resources over internet.

The computing resources include physical or [**virtual servers**](https://www.tutorialspoint.com/cloud_computing/cloud_computing_server_virtualization.htm), data storage, networking capabilities, application development tools, software, analytic platforms and more.

**Key Operations**

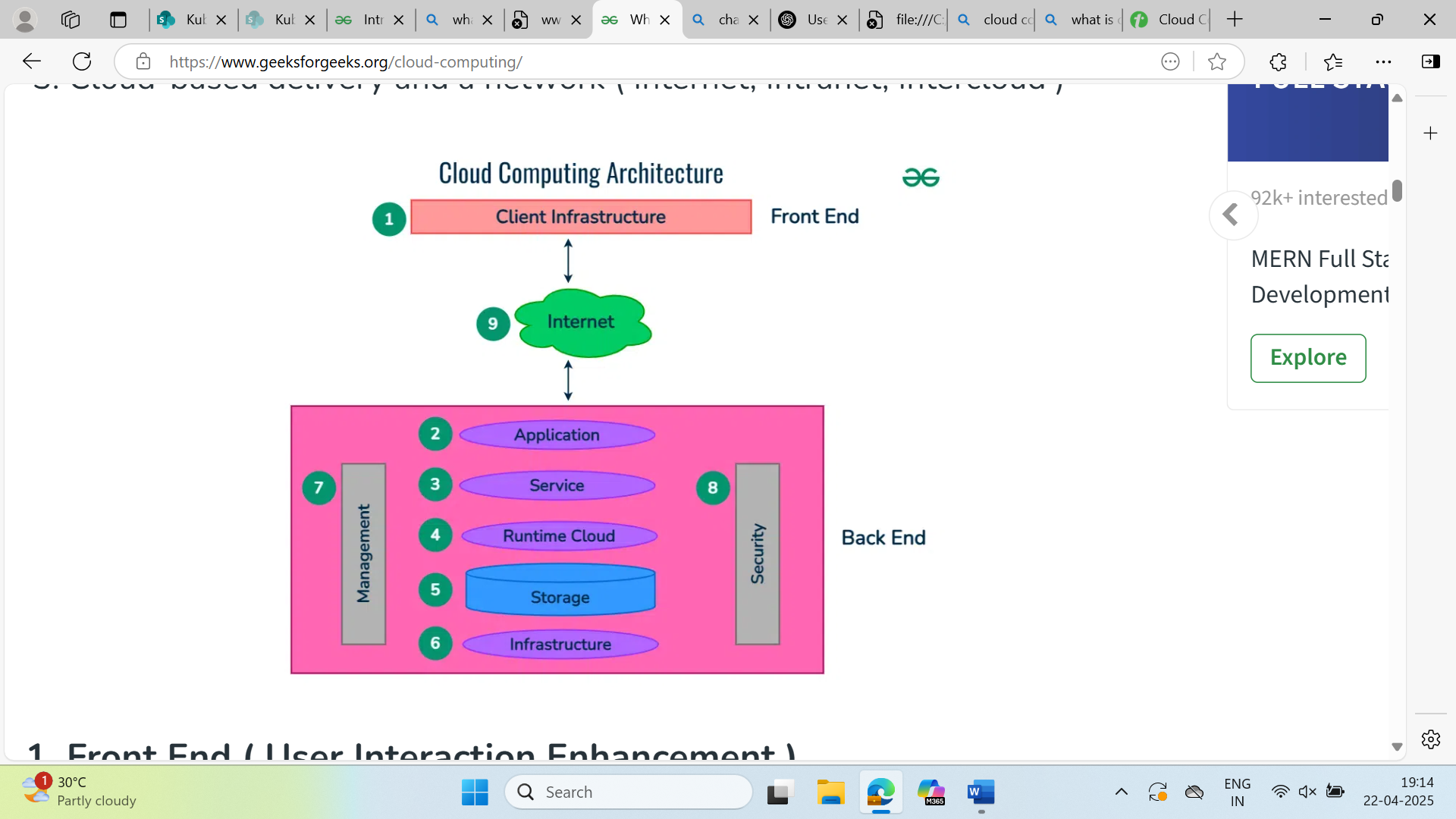
Cloud computing enables various operations such as:

* Developing new applications and services
* Storage, backup, and recovery of data
* Hosting blogs and websites
* Delivery of software on demand
* Analysis of data

**Architecture Of Cloud Computing**

[Cloud computing architecture](https://www.geeksforgeeks.org/architecture-of-cloud-computing/) refers to the components and sub-components required for cloud computing. These components typically refer to:

1. Front end ( Fat client, Thin client)
2. Back-end platforms ( Servers, Storage )
3. Cloud-based delivery and a network ( Internet, Intranet, Intercloud )



**1. Front End ( User Interaction Enhancement )**

The User Interface of Cloud Computing consists of 2 sections of clients. The Thin clients are the ones that use web browsers facilitating portable and lightweight accessibilities and others are known as Fat Clients that use many functionalities for offering a strong user experience.

**2. Back-end Platforms ( Cloud Computing Engine )**

The core of cloud computing is made at back-end platforms with several servers for storage and processing computing. Management of Applications logic is managed through servers and effective data handling is provided by storage. The combination of these platforms at the backend offers the processing power, and capacity to manage and store data behind the cloud.

**3. Cloud-Based Delivery and Network**

On-demand access to the computer and resources is provided over the Internet, Intranet, and Intercloud. The Internet comes with global accessibility, the[Intranet](https://www.geeksforgeeks.org/what-is-intranet/)helps in internal communications of the services within the organization and the [Intercloud](https://www.geeksforgeeks.org/inter-cloud-resource-management/) enables interoperability across various cloud services. This dynamic network connectivity ensures an essential component of cloud computing architecture on guaranteeing easy access and data transfer.

**Benefits of Cloud Computing**

The cloud computing provides several benefits such flexibility and reliability, increased performance and efficiency, and lower IT costs, when compared to on-premises compute resources. In on-premises, the company owns and maintains physical data centers and servers.

* **Cost Savings** − organizations can rent computing resources on-demand instead of buying expensive physical servers.  You can choose any cloud service model and you only pay for the resources you actually use.
* **Scalability** − organizations can quickly scale up and scale down their compute resources to meet the changing demands. The benefits of cloud computing services include the ability to scale elastically.
* **Performance** − Cloud computing offers greater performance by reducing the network latency for applications.
* **Reliability** − Cloud computing makes data backup, disaster recovery easier and less expensive.
* **Productivity** − On-premises datacenters typically require a lot of hardware and software setup, other time-consuming IT management chores. Cloud computing removes the need for many of these tasks, improving the performance of IT teams.
* **Security** − Cloud computing providers offers security for your data, applications, etc., from potential threats.

**What is a Cloud Deployment Model?**

* A cloud deployment model fundamentally determines where the infrastructure for your deployment exists and who owns and controls that infrastructure. It also figures out the nature and intention of the cloud.
* The cloud deployment model identifies the specific type of cloud environment based on ownership, scale, and access, as well as the cloud’s nature and purpose. The location of the servers you’re utilizing and who controls them are defined by a cloud deployment model.

**Public Cloud:**

In a public cloud, computing resources such as servers, storage, and applications are owned and operated by a third-party cloud service provider and delivered over theinternet.

Public cloud provides cloud infrastructure and services over the internet to the general public or large industry groups. The infrastructure is owned and managed by third-party service providers (e.g., AWS, Azure, GCP), not by theusers.  
Examples: AWS, Microsoft Azure, Google Cloud Platform (GCP).

**Advantages of the Public Cloud Model**

* **Minimal Investment:**Because it is a pay-per-use service, there is no substantial upfront fee, making it excellent for enterprises that require immediate access to resources.
* **No setup cost:** The entire infrastructure is fully subsidized by the cloud service providers, thus there is no need to set up any hardware.
* **Infrastructure Management is not required:**Using the public cloud does not necessitate infrastructure management.
* **No maintenance:**The maintenance work is done by the service provider (not users).
* **Dynamic Scalability:** To fulfill your company’s needs, on-demand resources are accessible.

**Disadvantages of the Public Cloud Model**

* **Less secure:**Public cloud is less secure as resources are public so there is no guarantee of high-level security.
* **Low customization:**It is accessed by many public so it can’t be customized according to personal requirements.

 **Limited Control:** Less control over hardware and software configurations.

 **Compliance Issues:** May not meet specific regulatory or compliance requirements.

**Private Cloud:**

The private cloud is a cloud deployment model dedicated to a single organization, offering exclusive access to systems and services without sharing hardware with others. Also known as the "internal cloud," it operates within a secure environment managed by the organization's IT team, often protected by strong firewalls. This model provides greater control, customization, and security, making it ideal for businesses with strict compliance or data protection requirements.

**Advantages of the Private Cloud Model**

* Better Control: You are the sole owner of the property. You gain complete command over service integration, IT operations, policies, and user behavior.
* Data Security and Privacy: It’s suitable for storing corporate information to which only authorized staff have access. By segmenting resources within the same infrastructure, improved access and security can be achieved.
* Supports Legacy Systems: This approach is designed to work with legacy systems that are unable to access the public cloud.
* Customization: Unlike a public cloud deployment, a private cloud allows a company to tailor its solution to meet its specific needs.

**Disadvantages of the Private Cloud Model**

* Less scalable: Private clouds are scaled within a certain range as there is less number of clients.
* Costly: Private clouds are more costly as they provide personalized facilities.

**Hybrid Cloud:**   
A hybrid cloud combines public and private clouds, allowing data and applications to be shared between them. It allows organizations to securely host critical applications while leveraging the cost-efficiency and scalability of the public cloud. Data and applications can move between environments based on business needs, providing flexibility and optimized resource use.

**Advantages of the Hybrid Cloud Model**

* Flexibility and control: Businesses with more flexibility can design personalized solutions that meet their particular needs.
* Cost: Because public clouds provide scalability, you’ll only be responsible for paying for the extra capacity if you require it.
* Security: Because data is properly separated, the chances of data theft by attackers are considerably reduced.

**Disadvantages of the Hybrid Cloud Model**

* Difficult to manage: Hybrid clouds are difficult to manage as it is a combination of both public and private cloud. So, it is complex.
* Slow data transmission: Data transmission in the hybrid cloud takes place through the public cloud so latency occurs.

**Community Cloud:**

It allows systems and services to be accessible by a group of organizations. It is a distributed system that is created by integrating the services of different clouds to address the specific needs of a community, industry, or business. The infrastructure of the community could be shared between the organization which has shared concerns or tasks. It is generally managed by a third party or by the combination of one or more organizations in the community.

**Advantages of the Community Cloud Model**

* **Cost Effective:**It is cost-effectivebecause the cloud is shared by multiple organizations or communities.
* **Security:** Community cloud provides better security.
* **Shared resources:**It allows you to share resources, infrastructure, etc. with multiple organizations.
* **Collaboration and data sharing:**It is suitable for both collaboration and data sharing.

**Disadvantages of the Community Cloud Model**

* **Limited Scalability:**Community cloud is relatively less scalable as many organizations share the same resources according to their collaborative interests.
* **Rigid in customization:**As the data and resources are shared among different organizations according to their mutual interests if an organization wants some changes according to their needs they cannot do so because it will have an impact on other organizations.

**Multi Cloud :**

Multi-cloud refers to the use of multiple cloud computing services from different providers within a single architecture. Unlike hybrid cloud, which combines public and private clouds, multi-cloud involves using more than one public cloud service (e.g., AWS, Azure, GCP) for different workloads. This approach helps avoid vendor lock-in, increases redundancy, and allows organizations to select the best tools for each task, improving flexibility and risk management.

**Advantages of the Multi-Cloud Model**

* **Reduced Latency:** To reduce latency and improve user experience, you can choose cloud regions and zones that are close to your clients.
* **High availability of service:** It’s quite rare that two distinct clouds would have an incident at the same moment. So, the multi-cloud deployment improves the high availability of your services.

**Disadvantages of the Multi-Cloud Model**

* **Complex:**The combination of many clouds makes the system complex and bottlenecks may occur.
* **Security issue:**Due to the complex structure, there may be loopholes to which a hacker can take advantage hence, makes the data insecure.

**Cloud Computing Services:**

**Infrastructure as a Service (IaaS)** is a cloud computing model that delivers virtualized computing resources—like servers, storage, and networking—over the internet. It eliminates the need for organizations to own physical hardware.

**🧩 Key Characteristics**

* **On-Demand Access**: Instantly provision computing resources.
* **Self-Service**: Users manage resources via web portals or APIs.
* **Scalability**: Easily scale resources up/down based on demand.
* **Pay-as-You-Go**: Cost based on actual usage.
* **Multi-Tenant Model**: Resources shared among multiple users securely.

**✅ Benefits / Advantages**

* **Flexibility** in configuring infrastructure.
* **Scalability** to handle varying workloads.
* **Cost-Efficiency** (no upfront hardware costs).
* **Fast Provisioning** of VMs and services.
* **Global Reach** with data centers worldwide.
* **High Availability & Resilience** (backups, failover).
* **Built-in Security** measures and compliance options.

**⚠️ Disadvantages / Challenges**

* **Complex Management** for resource optimization.
* **Internet Dependency** for access and performance.
* **Security Risks** if not properly configured.
* **Vendor Lock-In** due to proprietary tools/services.
* **Cost Overruns** from poor resource monitoring.

**🛠️ Common Use Cases**

* Hosting websites and applications.
* Running virtual machines for development/testing.
* Backup and disaster recovery.
* High-performance computing (HPC).
* Scalable database hosting.
* Big data analytics and IoT.

**Platform as a Service:**

**Platform as a Service (PaaS)** provides a cloud-based runtime environment where developers can **build, test, deploy, and manage applications** without handling the underlying infrastructure. The platform handles hardware, OS, networking, storage, and scalability.

**🧩 Key Characteristics**

* **Managed Infrastructure**: Servers, storage, networking handled by the provider.
* **Integrated Development Tools**: IDEs, SDKs, build tools.
* **Built-in Middleware**: For databases, app hosting, and messaging.
* **Auto-Scalability**: Automatically adjusts resources based on demand.
* **Multi-language Support**: Java, Python, Node.js, PHP, .NET, Ruby, etc.
* **Web-based Access**: Development from anywhere via browser.

**🛠️ Core Services & Tools**

* **Programming Languages**: Java, Python, PHP, Ruby, .NET, Node.js.
* **Application Frameworks**: Django, Spring, Laravel, Rails.
* **Managed Databases**: MySQL, PostgreSQL, SQL Server, MongoDB, Redis.
* **Development Tools**: IDEs, version control, testing, deployment tools.
* **Collaboration Tools**: Shared repos, chat, project management.
* **Monitoring & Analytics**: Performance tracking, user behavior insights.
* **Security Services**: Identity management, access control, encryption.
* **Integration Tools**: APIs for third-party services (e.g., payments, analytics).
* **Load Balancing & Scaling**: Automatically distributes traffic.

**🌟 Advantages**

1. **Simplified Development**: Focus on coding, not infrastructure.
2. **Lower Upfront Cost**: No need to invest in hardware/software.
3. **Built-in Functionality**: Reusable components speed up development.
4. **Community Support**: Developer forums and help centers.
5. **Scalability**: Apps grow seamlessly from one user to millions.
6. **Faster Time-to-Market**: Rapid prototyping and deployment.
7. **Cross-Platform Development**: Build apps for web, mobile, or API services.

**⚠️ Disadvantages / Challenges**

1. **Vendor Lock-in**: Migration to another platform can be difficult.
2. **Data Privacy Concerns**: Sensitive data outside company walls.
3. **Limited Customization**: Control over infrastructure is restricted.
4. **Integration Complexity**: Issues integrating with on-prem systems.
5. **Downtime & Dependency**: Dependent on provider’s uptime and SLA.

**💡 Common Use Cases**

* Web & Mobile App Development
* API & Microservices Hosting
* Dev/Test Environments
* IoT and Real-Time Applications
* Business Process Automation
* Machine Learning Model Deployment

**🔧 Popular PaaS Examples**

* Google App Engine
* Microsoft Azure App Services
* Heroku
* Red Hat OpenShift
* Salesforce Force.com

**Software as a Service (SaaS)**

SaaS is also known as **"On-Demand Software"**. It is a **software distribution model** in which applications are hosted by a cloud service provider and made available to users over the internet. SaaS eliminates the need for organizations to install, maintain, or manage software locally.

**🧩 Characteristics of SaaS**

* **Web-Based Delivery**: Access via web browsers from anywhere with an internet connection.
* **Multi-Tenancy**: A single instance of the application serves multiple users.
* **Automatic Updates**: Providers manage all updates and patches.
* **Scalable**: Easily scalable to accommodate growing user demands.
* **Subscription-Based Pricing**: Pay-per-month/year model for predictable costs.
* **Data Security**: Handled by the provider, including encryption and access controls.

**🌐 Access to Software Applications Over the Internet**

* Hosted and maintained on the provider’s infrastructure.
* Accessed via browser or mobile app using login credentials.
* No local installation required.
* Always up to date with the latest features and security.
* Offers global accessibility on multiple devices.

**💰 Pricing Models: Subscription or Usage-Based**

* **Subscription-Based**: Flat monthly/yearly fee depending on users or features.
* **Usage-Based**: Pricing depends on how much of the service is used (e.g., storage, API calls).
* Businesses choose based on usage patterns and cost-effectiveness.

**🛠 Services Provided by SaaS**

**✅ Business Services**

* CRM, ERP, billing, and sales platforms.
* Example: **Salesforce CRM**, **Zoho CRM**.

**✅ Document Management**

* Tools to create, track, and manage digital documents.
* Example: **Box**, **Zoho Forms**, **Samepage**.

**✅ Social Networks**

* Platforms using SaaS to handle high-volume public data.
* Example: **LinkedIn**, **Facebook Business Tools**.

**✅ Mail Services**

* Email platforms hosted via SaaS.
* Example: **Gmail**, **Outlook.com**.

**✅ Collaboration Tools**

* For team communication, project management, and file sharing.
* Example: **Slack**, **Google Workspace**, **Microsoft 365**.

**✅ HR Management**

* Manage payroll, hiring, employee records, etc.
* Example: **Workday**, **BambooHR**, **ADP**.

**✅ Customer Support/Help Desk**

* Platforms for ticketing and customer service.
* Example: **Zendesk**, **Freshdesk**, **Salesforce Service Cloud**.

**✅ Marketing & Sales Automation**

* Tools for lead generation, campaign management.
* Example: **HubSpot**, **Pardot**, **Marketo**.

**✅ E-commerce Platforms**

* Full e-commerce functionality with hosting and payment handling.
* Example: **Shopify**, **BigCommerce**, **WooCommerce**.

**✅ Advantages of SaaS (Cloud Computing Layer)**

1. **Easy to Buy**  
   Subscription-based pricing makes it affordable with no large upfront cost.
2. **One-to-Many Model**  
   A single instance serves multiple users, increasing efficiency.
3. **Reduced Hardware Needs**  
   Software is hosted remotely, requiring no local infrastructure.
4. **Low Maintenance**  
   No installations or updates needed by the user. Everything is managed by the provider.
5. **Uniform Software Versions**  
   Everyone uses the same, latest version via the web.
6. **Multidevice Support**  
   Accessible from desktops, tablets, smartphones, etc.
7. **API Integration**  
   Easily integrates with other services and platforms via APIs.
8. **No Client-Side Installation**  
   Everything runs from the browser — no software to install locally.

**❌ Disadvantages of SaaS (Cloud Computing Layer)**

1. **Security Concerns**  
   Data is stored in the cloud, which may worry organizations with strict security needs.
2. **Latency Issues**  
   Slower response times than local installations for apps requiring real-time processing.
3. **Dependency on Internet**  
   SaaS apps typically require a stable internet connection to work.
4. **Vendor Lock-In**  
   Switching providers can be complex due to data migration and compatibility issues.

**Anything as a Service**

It is also known as Everything as a Service. Most of the cloud service providers nowadays offer anything as a service that is a compilation of all of the above services including some additional services.

**Advantages of XaaS:**

1. **Scalability:** [XaaS](https://www.geeksforgeeks.org/overview-of-everything-as-a-service-xaas/) solutions can be easily scaled up or down to meet the changing needs of an organization.
2. **Flexibility:** XaaS solutions can be used to provide a wide range of services, such as storage, databases, networking, and software, which can be customized to meet the specific needs of an organization.
3. **Cost-effectiveness**: XaaS solutions can be more cost-effective than traditional on-premises solutions, as organizations only pay for the services.

**Disadvantages of XaaS:**

1. **Dependence on the provider:**Users are dependent on the XaaS provider for the availability, scalability, and reliability of the service, which can be a risk if the provider experiences outages or other issues.
2. **Limited flexibility**: XaaS solutions may not be able to accommodate certain types of workloads or applications, which can limit the value of the solution for certain organizations.
3. **Limited integration:** XaaS solutions may not be able to integrate with existing systems and data sources, which can limit the value of the solution for certain organizations.

**Function as a Service**

[FaaS](https://www.geeksforgeeks.org/function-as-a-service-faas-system-design/)is a cloud service that lets you run small pieces of code – called functions- without managing any servers. You just write your code, upload it, and it runs **only when triggered by an event,** like a button click or a file upload.

FaaS is event-driven, meaning the code runs only when something specific happens. You don’t need to keep a sever running in the background – it starts automatically when needed and stops when the job is done. That’s why it is also called serverless (even though servers are still used, they’ re managed entirely by the provider).

**Example:**

Imagine an online photo app that resizes images whenever a user uploads a photo. With FaaS, you write a small functions to resize the image. The Function only runs when a photo is uploaded- and you only pay for the execution.

**Advantages of FaaS**

* **Highly Scalable:**Auto scaling is done by the provider depending upon the demand.
* **Cost-Effective:**Pay only for the number of events executed.
* **Code Simplification:**FaaS allows the users to upload the entire application all at once. It allows you to write code for independent functions or similar to those functions.
* Maintenance of code is enough and no need to worry about the servers.
* Functions can be written in any programming language.
* Less control over the system.

The various companies providing Function as a Service are Amazon Web Services – Firecracker, Google – Kubernetes, Oracle – Fn, Apache OpenWhisk – IBM, OpenFaaS,

**Disadvantages of FaaS**

1. **Cold start latency**: Since FaaS functions are event-triggered, the first request to a new function may experience increased latency as the function container is created and initialized.
2. **Limited control over infrastructure:** FaaS providers typically manage the underlying infrastructure and take care of maintenance and updates, but this can also mean that users have less control over the environment and may not be able to make certain customizations.
3. **Security concerns:**Users are responsible for securing their own data and applications, which can be a significant undertaking.
4. **Limited scalability**: FaaS functions may not be able to handle high traffic or large number of requests.

**NSG(Network Security Group):**

NSG (Network Security Group) is a security feature in Microsoft Azure that acts as a virtual firewall to control inbound and outbound traffic to Azure resources at the network level.

NSGs allow or deny traffic based on rules like source/destination IP addresses, ports, and protocols.

**Where NSGs are Applied**

NSGs can be associated with:

* **Network Interfaces (NICs)** – for controlling traffic to individual virtual machines.
* **Subnets** – for controlling traffic to all resources in the subnet.

Each NSG contains a list of **security rules**. Each rule has:

* Name: Rule identifier
* Priority: Integer (lower number = higher priority)
* Direction: Inbound or Outbound
* Access: Allow or Deny
* Protocol: TCP, UDP, or \* (any)
* Source: IP, CIDR, Service Tag, or Application Security Group
* Source Port Range: Specific port(s) or \*
* Destination: IP, CIDR, or ASG
* Destination Port Range: Specific port(s)

**Best Practices for Using NSGs**

1. **Least Privilege**: Allow only necessary traffic, deny all else.
2. **Use NSGs at Subnet Level**: For broad controls.
3. **Use NSGs at NIC Level**: For VM-specific rules.
4. **Avoid Conflicting Rules**: Ensure rules at different levels don’t contradict.
5. **Log and Monitor**: Use **Azure Network Watcher** to monitor and diagnose traffic.
6. **Tag Traffic Sources**: Use **Service Tags** like Internet, VirtualNetwork, AzureLoadBalancer for simplicity.
7. **Use Application Security Groups (ASGs)**: Group VMs logically and apply rules to ASGs instead of IPs.

**Non-Functional Requirements (NFRs)?**

Non-Functional Requirements define **how** a system performs its functions, rather than **what** it does.

**Why Are NFRs Important?**

* Ensure a **smooth user experience**
* Help with **system resilience under load**
* Guide the **design of cloud infrastructure**

**Categories of NFRs with Examples**

**1. Performance**

**Definition**: How fast the system responds to user or system requests.

**Examples**:

* "System should respond within 2 seconds for 95% of all requests."
* "Data retrieval from the database must not exceed 1.5 seconds."

**2. Scalability**

**Definition**: System’s ability to handle increased load.

**Examples**:

* "The application must support 10,000 concurrent users."
* "The system should auto-scale based on CPU usage exceeding 70%."

**Relevance**:

* Achieved through **Horizontal Pod Autoscaling** in Kubernetes
* Tested via **load simulation** in staging environments

**3. Availability**

**Definition**: The system’s readiness for correct service.

**Examples**:

* "The service must be available 99.99% of the time."
* "System downtime should not exceed 1 hour/month."

**Cloud-Specific**:

* Architected using **Availability Zones** and **load balancers**
* Backed by **SLAs from cloud providers (like AWS, Azure, GCP)**

**4. Reliability**

**Definition**: System’s ability to operate without failure.

**Examples**:

* "System must recover from any crash within 5 minutes."
* "Must complete 99.999% of transactions successfully."

**DevOps Strategy**:

* Use of **circuit breakers**, **retries**, and **monitoring tools**
* Chaos Engineering tools like **Gremlin** are used to test this

**5. Maintainability**

**Definition**: Ease of maintaining the system post-deployment.

**Examples**:

* "Fixing a critical bug should take less than 4 hours."
* "Codebase should follow clean coding principles."

**Best Practices**:

* Use **microservices**, **modular architecture**
* Maintain **CI/CD pipelines with tests, linting, and formatting**

**6. Security**

**Definition**: Protection from unauthorized access and attacks.

**Examples**:

* "Data should be encrypted at rest and in transit using AES-256."
* "Users should be authenticated via OAuth 2.0."

**Implementation**:

* **RBAC in Kubernetes**
* Integration with **SSO providers**, **firewalls**, and **IDPs**

**7. Usability**

**Definition**: How intuitive and user-friendly the system is.

**Examples**:

* "Users should be able to complete the signup process within 3 steps."
* "UI should be accessible to screen readers."

**Tools**:

* Use **UX research**, **usability testing**
* Follow **WCAG (Web Content Accessibility Guidelines)**

**8. Portability**

**Definition**: How easily the system can be moved between environments.

**Examples**:

* "The system must run on both AWS and GCP."
* "Should be containerized to allow migration without config change."

**Related Technologies**:

* Docker, Kubernetes
* Terraform (cloud-agnostic IaC)

**9. Interoperability**

**Definition**: Ability to interact with other systems and services.

**Examples**:

* "The app must integrate with third-party payment APIs."
* "Support JSON and XML data formats."

**Tools**:

* API Gateways, Webhooks, REST, GraphQL

**10. Compliance**

**Definition**: Adherence to legal and regulatory requirements.

**Examples**:

* "System must comply with GDPR, HIPAA."
* "All data must be logged and stored for 5 years."

**Audit & Compliance Tools**:

* AWS Artifact, Azure Compliance Center, HashiCorp Sentinel

**Types of Kubernetes Scaling**

Scaling in Kubernetes ensures your application runs smoothly under varying workloads by adjusting **pods** or **nodes** automatically or manually.

There are **three main types of scaling**:

1. **Horizontal Pod Autoscaler (HPA)**
2. **Vertical Pod Autoscaler (VPA)**
3. **Cluster Autoscaler**

**Horizontal Pod Autoscaler (HPA)**

**What it does:**

* Automatically **increases or decreases** the number of **pod replicas** based on **resource utilization** or **custom metrics**.

**Metrics Used:**

* CPU utilization (default)
* Memory (via custom metrics adapter)
* Custom metrics (e.g., queue length, request rate)

**Components Required:**

* **Metrics Server** (must be installed in the cluster)
* Deployment or StatefulSet with resources.requests.cpu and resources.limits.cpu defined

**How it works:**

* HPA controller checks metrics every **15 seconds** (configurable).
* Compares current usage with the **target average utilization**.
* Adjusts the number of replicas in the deployment.

**Vertical Pod Autoscaler (VPA)**

**What it does:**

* Automatically **adjusts the CPU and memory requests/limits** for a pod.

**Why Use It:**

* When app performance depends more on pod **capacity per instance** rather than **number of pods**.
* Ideal for **batch jobs**, **stateful apps**, or **low request-volume services**.

**How it works:**

* Monitors usage patterns over time
* Recommends or applies updated resource requests
* Can **evict and restart** pods with the new values

**Cluster Autoscaler**

**What it does:**

* Automatically adds/removes **nodes** to/from the Kubernetes cluster based on **unschedulable pods**.

**Why Use It:**

* Helps save costs in **cloud environments** (e.g., AWS, GCP, Azure) by adjusting the size of the cluster dynamically.

**Requirements:**

* Must be deployed separately in the cluster
* Needs **cloud provider support** (e.g., auto-scaling groups on AWS EC2)

**How it works:**

* Periodically checks if any pods are in a Pending state due to lack of resources.
* If yes, adds more nodes (up to a max limit).
* If nodes are underutilized for a long time, it scales them down.

**Example in Cloud:**

In **AWS EKS**, you would:

* Use Auto Scaling Groups (ASGs)
* Tag ASGs with k8s.io/cluster-autoscaler/enabled and the cluster name
* Deploy Cluster Autoscaler as a pod in your cluster.

**Bastion Host**

**Definition:**

A **Bastion Host** is a special-purpose instance (typically a Linux or Windows VM) that sits in a **public subnet** and provides **secure access (SSH/RDP)** to resources inside a **private subnet**.

It acts as a **jump server** or **gateway**, limiting the exposure of internal resources.

**Key Characteristics:**

* Has a **public IP** and is reachable from the internet.
* Allows access **only via SSH/RDP**.
* Internal resources **do not have public IPs**.
* Often protected with **firewall/NACL/NSG rules**, MFA, and hardened OS configurations.

**Use Cases:**

* Secure admin access to **EC2 instances (AWS)**, **Azure VMs**, **GCP Compute Engines** in **private subnets**.
* Acts as a **jump box** for DevOps/administrators.
* Ideal for **highly secure environments** like banking, healthcare, and enterprise systems.

**Best Practices:**

* Use **Just-In-Time (JIT)** access and **MFA**.
* Harden the OS with tools like **CIS benchmarks**.
* Restrict access to specific IP ranges.
* Use **Bastion Services** provided by cloud providers (e.g., **Azure Bastion** or **AWS Systems Manager Session Manager**) to avoid managing your own VM.

**VNet (Virtual Network) & VPC (Virtual Private Cloud)**

**Definition:**

* A **VNet (Azure)** or **VPC (AWS)** is a **logically isolated network** in the cloud.
* It allows you to define your own **IP address range**, **subnets**, **route tables**, and **security rules**.

**Key Features:**

* **Custom CIDR blocks** (e.g., 10.0.0.0/16)
* **Subnets** to divide and organize resources
* **Security control** via NSGs (Azure) or NACLs/Security Groups (AWS)
* **Internet Gateway**, NAT Gateways, VPNs
* **Peering**, **Private Link**, and **Service Endpoints** for secure communication

**VNet = VPC = VPC Network (GCP)**

| **Cloud** | **Network Name** | **Notes** |
| --- | --- | --- |
| AWS | VPC | Uses Route Tables, IGW |
| Azure | VNet | Uses NSGs, Route Tables |
| GCP | VPC Network | Global scope by default |

**Use Cases:**

* Host web apps, DBs, containers in isolated environments
* Set up hybrid connectivity (e.g., VPN to on-prem)
* Deploy microservices securely

**Region**

**Definition:**

A **region** is a **geographical area** that contains one or more **availability zones (AZs)** (data centers).

**Examples:**

* **AWS**: us-east-1, ap-south-1
* **Azure**: East US, West Europe
* **GCP**: asia-southeast1, europe-west4

**Importance:**

* **Latency**: Choose region close to your users for performance.
* **Compliance**: Some regulations require data to reside in specific countries (e.g., GDPR).
* **Redundancy**: Deploy in multiple regions for disaster recovery.

**Use Cases:**

* Multiregion deployments for failover
* CDN and edge caching
* Legal compliance (India, EU, etc.)

**Availability Zones (AZ)**

**Definition:**

**Availability Zones** are **physically separated data centers** within a single region.

Each AZ has **independent power, cooling, and networking**, but they are connected via low-latency links.

**Benefits:**

* **High availability**: Deploy replicas across zones.
* **Resilience**: If one AZ goes down, others still operate.
* **Disaster isolation** within region.

**Examples:**

* **AWS**: us-east-1a, us-east-1b, us-east-1c
* **Azure**: East US has 3 AZs
* **GCP**: Each region has multiple zones, like us-central1-a, us-central1-b

**Use Cases:**

* Deploying across AZs with **load balancers**
* Active-active architecture
* Managed databases with AZ redundancy

**Peering**

**Definition:**

**Peering** enables **private network connectivity** between two **VNets (Azure)** or **VPCs (AWS)** without using public internet.

**Types of Peering:**

* **VNet Peering (Azure)**
* **VPC Peering (AWS)**
* **Cross-region Peering**
* **Cross-subscription Peering**

**Features:**

* Low-latency, high-bandwidth connection
* No public IPs or VPNs required
* Security-controlled with NSG/Security Groups

**Use Cases:**

* Microservices spread across multiple VNets/VPCs
* Centralized logging system accessed by multiple teams
* Shared databases or services across environments (Dev, Test, Prod)

**Limitations:**

* No **transitive routing** (A ↔ B, B ↔ C doesn’t mean A ↔ C)
* CIDR ranges must not overlap

**Subnets**

**Definition:**

A **subnet** is a **subdivision of a VNet or VPC** and is used to organize and isolate resources.

**Types of Subnets:**

| **Type** | **Description** |
| --- | --- |
| **Public** | Has route to Internet via Internet Gateway/NAT Gateway |
| **Private** | No direct internet access, used for internal resources |
| **Isolated** | No internet, no peering, fully internal |

**Why Subnets Matter:**

* Segment workloads by **tier** (web, app, DB)
* Apply **security rules** (e.g., NSGs) per subnet
* Control **routing** between subnets

**Use Cases:**

* Public subnet for frontend (load balancer, web server)
* Private subnet for backend (DB, app servers)
* Separate subnets for **multi-tier applications**

**Scale Set (Azure Virtual Machine Scale Set - VMSS)**

**Definition:**

A **Virtual Machine Scale Set (VMSS)** is an **Azure service** that allows you to **deploy and manage a group of identical VMs**. It automatically scales the number of VM instances based on **demand** or a **schedule**.

**Features:**

* Automatic scaling (horizontal scaling)
* Load balancing support
* Custom images or marketplace images
* Integration with **Azure Load Balancer** and **Azure App Gateway**
* Works well for **stateless workloads** (e.g., web servers)

**Use Case:**

Let’s say you run a web app that experiences high traffic at 8 PM daily:

* VMSS can automatically **add VMs** to handle the load.
* At midnight, when traffic drops, it can **remove excess VMs**.
* Great for **cost optimization** + **scalability**.

**Auto Scaling:**

You can configure:

* **CPU-based** autoscaling (e.g., scale out if CPU > 75% for 5 minutes)
* **Custom metrics** (like queue length, memory)
* **Schedule-based** (e.g., scale at 6 PM every day)

**Load Balancer in Front of Scale Set**

**Purpose:**

Distributes traffic evenly across VMs in a scale set. Think of it as the “**traffic cop**” that ensures **no single VM is overwhelmed**.

**How it Works:**

1. **Public Load Balancer** has a frontend IP (say 20.10.20.30)
2. It forwards incoming traffic to a backend pool — the **scale set VMs**
3. Health probes ensure that traffic goes only to **healthy VMs**
4. When the scale set **scales out**, new VMs are added automatically to the backend pool

**Real Example:**

* Your app runs on 3 VMs in VMSS.
* Traffic increases → scale set adds 2 more VMs.
* Load Balancer **automatically starts routing** traffic to the new VMs.

**Fault Domain**

**Definition:**

A **Fault Domain** is a **physical separation of compute resources** to protect against **hardware failure** (like power, network, or disk failure in a rack).

**In Azure:**

* Each region has **multiple fault domains**.
* When you deploy VMs (or scale sets), Azure **distributes** them across **different fault domains** by default.
* This ensures that **not all your VMs go down** in case of a hardware issue.

**Use Case:**

* Deploy 3 VMs → Azure places each in a **different fault domain**.
* If one rack fails, the other two are **still up**, ensuring high availability.

**SSD (Solid State Drive)**

**Definition:**

SSD is a type of **high-speed storage** that uses **flash memory**. In cloud computing, SSDs are used for **performance-intensive applications**.

**Pros:**

* Very fast (low latency, high IOPS)
* Great for:
  + Databases (SQL, MongoDB)
  + High-performance computing
  + Application boot disks

**Cons:**

* **More expensive** than HDD
* Not ideal for archival storage

**HDD (Hard Disk Drive)**

**🔍 Definition:**

HDDs are **mechanical drives** that store data magnetically. They're **slower** than SSDs but much **cheaper**.

**Use Case:**

* Backup data
* Archive files
* Large, infrequent access workloads

**Comparison:**

| **Feature** | **SSD** | **HDD** |
| --- | --- | --- |
| Speed | Very fast | Slower |
| Cost | High | Low |
| Use Case | Databases, VMs | Backups, cold storage |

**Ephemeral Storage**

**Definition:**

**Ephemeral storage** is **temporary disk space** attached to a virtual machine. It is **non-persistent** — meaning the data is **lost when the VM is stopped or deallocated**.

**Characteristics:**

* Local to the VM → very **fast IOPS**
* Ideal for:
  + OS swap files
  + Temp files
  + Caching
* Not meant for storing **important data**

**Example:**

* You start a VM with 50GB ephemeral storage.
* You write logs/temp files.
* When you stop and restart the VM — **that data is gone**.

**Real-World Use:**

* **Kubernetes** pods often use ephemeral storage for **temp data or logs**
* **CI/CD build agents** might use it for **scratch build**