**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.

**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.

**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

**Benefits of Cloud Computing :**

* **Cost Savings** – Pay only for what you use without investing in physical hardware.
* **Scalability** – Instantly scale resources up or down based on demand.
* **Performance** – Reduced latency and improved application performance.
* **Reliability** – Easier and cost-effective backup and disaster recovery.
* **Productivity** – Reduces IT management overhead and setup time.
* **Security** – Offers strong protection for data and applications against threats.

**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.

**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 the internet.

**Advantages of Public Cloud (Short):**

* **Minimal Investment** – Pay-per-use with no large upfront cost.
* **No Setup Cost** – Hardware setup is handled by the provider.
* **No Infrastructure Management** – Managed entirely by the provider.
* **No Maintenance** – Maintenance is the provider's responsibility.
* **Dynamic Scalability** – On-demand resource availability.

**Disadvantages of Public Cloud (Short):**

* **Less Secure** – Shared resources may pose security risks.
* **Low Customization** – Limited personalization options.
* **Limited Control** – Restricted hardware/software configurations.
* **Compliance Issues** – May not meet specific regulatory needs.

**Private Cloud:**

* A Private Cloud is a secure, isolated cloud environment dedicated to a single organization, offering exclusive access to computing resources. It can be hosted on-premises or by a third-party, and is managed internally or externally, providing enhanced control, security, and customization.

**Key Characteristics of Private Cloud:**

* **Exclusive Use:** Resources are not shared with other organizations.
* **Enhanced Security:** Stronger data privacy and security controls.
* **Customization:** High flexibility to configure according to business needs.
* **Compliance-Friendly:** Ideal for industries with strict **regulatory requirements** (e.g., healthcare, finance).

**Advantages:**

* **Better Control** – Full control over IT operations and policies.
* **Data Security & Privacy** – Secure access for authorized users only.
* **Legacy System Support** – Compatible with older systems.
* **Customization** – Tailored solutions for specific business needs.

**Disadvantages:**

* **Less Scalable** – Limited scalability due to dedicated resources.
* **Costly** – Higher cost for personalized infrastructure and management.

**Hybrid Cloud**

* A **Hybrid Cloud** is a combination of both public and private clouds, allowing data and applications to be shared between them. It offers the flexibility to run sensitive workloads in the private cloud while leveraging the scalability and cost-effectiveness of the public cloud.

**Advantages of the Hybrid Cloud Model:**

* **Flexibility and Control:** Combines public and private benefits to meet specific business needs.
* **Cost Efficiency:** Use public cloud resources for non-sensitive operations and scale when needed.
* **Security:** Critical data stays secure in the private cloud.

**Disadvantages of the Hybrid Cloud Model:**

* **Complex Management:** Requires managing both environments.
* **Latency Issues:** Data transfer between clouds may introduce delays.

**Community Cloud**

* A **Community Cloud** is a cloud infrastructure shared by several organizations with similar requirements (e.g., compliance, mission, or security). It can be managed by one or more organizations or a third party.

**Advantages of the Community Cloud Model:**

* **Cost Effective:** Shared costs across multiple organizations.
* **Enhanced Security:** Designed for organizations with common security needs.
* **Resource Sharing:** Enables collaboration and shared infrastructure.

**Disadvantages of the Community Cloud Model:**

* **Limited Scalability:** Shared nature restricts expansion.
* **Less Customization:** Changes are limited due to shared governance.

**Multi Cloud :**

* Multi-Cloudis a cloud strategy where an organization uses services from multiple public cloud providers (like AWS, Azure, GCP) to distribute workloads and avoid dependency on a single vendor.

**Advantages of the Multi-Cloud Model:**

* **Reduced Latency:** Deploy in regions closer to users.
* **High Availability:** Outages in one provider won’t impact all services.

**Disadvantages of the Multi-Cloud Model:**

* **Complex Management:** Multiple platforms increase complexity.
* **Security Risks:** More surfaces for potential security vulnerabilities.

**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.

**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.

**Applications of SaaS:**

* **Email:** Gmail, Outlook
* **Office Suites:** Google Workspace, Microsoft 365
* **Collaboration Tools:** Slack, Zoom, Trello
* **CRM:** Salesforce, HubSpot
* **Accounting:** QuickBooks Online, Zoho Books

**Use Cases:**

* Remote work and collaboration
* Customer relationship management
* Online communication and meetings
* Billing and accounting automation
* Document editing and sharing

**XaaS (Anything as a Service):**

**Definition:** A broad term that refers to delivering anything as a service over the internet, including IaaS, PaaS, SaaS, and more.

**Advantages:**

* **Scalable** to changing needs
* **Flexible** across various IT services
* **Cost-effective**, pay-as-you-go

**Disadvantages:**

* **Vendor dependency**
* **Limited flexibility** for unique workloads
* **Integration issues** with legacy systems

**FaaS (Function as a Service):**

**Definition:** A serverless model where small pieces of code (functions) run in response to events, without managing infrastructure.

**Advantages:**

* Highly scalable & cost-effective
* No server management
* Write functions in any language
* Great for microservices and event-driven apps

**Disadvantages:**

* Cold start latency
* Limited infrastructure control
* Security & scalability concerns

**Example Use Case:**  
Image resizing in a photo app when a file is uploaded.

**Popular Providers:**  
AWS Lambda, Google Cloud Functions, Azure Functions, IBM OpenWhisk

**Security Concerns in IaaS, PaaS, and SaaS**

**IaaS Security Risks:**

IaaS exposes more of the infrastructure to users, which means managing the operating system, applications, and network security is the user's responsibility.

Common Attacks:

Data breaches

Denial-of-service (DoS) attacks

Vulnerabilities in virtual machine configurations

**PaaS Security Risks:**

In PaaS, security is shared between the cloud provider and the customer. While the cloud provider manages the infrastructure, the customer needs to focus on securing their application and data.

Common Attacks:

Misconfiguration vulnerabilities

Insufficient authentication and authorization

API security risks

**SaaS Security Risks:**

SaaS models have the least amount of user control, as the application is fully managed by the provider. However, there are still potential security concerns related to user data and access management.

Common Attacks:

Data leakage

Account hijacking

Service interruptions due to cyberattacks

**Functional Requirements (FR):**

Functional requirements specify what a system should do. They describe the specific behaviors, actions, or functions the system must have to meet the needs of its users. Essentially, functional requirements define the capabilities of the system.

**Characteristics of Functional Requirements:**

Focus on specific actions or operations of the system.

Can be expressed as user stories or use cases.

Directly related to the system's core functionality.

**Non-Functional Requirements (NFR):**

Non-functional requirements specify how a system performs its functions. They define the quality attributes and operational aspects of the system, such as its performance, scalability, security, and availability. These requirements help ensure the system works effectively and meets the users' expectations in terms of system qualities.

Examples of Non-Functional Requirements:

Performance: The system must handle 1000 concurrent users without degradation in speed.

Scalability: The system must be able to scale horizontally to support a growing number of users.

Availability: The system must have 99.9% uptime.

Security: The system must encrypt sensitive user data using AES-256 encryption.

Usability: The system’s user interface must be simple and intuitive for new users.

**Characteristics of Non-Functional Requirements:**

Focus on system quality and performance.

Often more general and can apply across multiple features or parts of the system.

Can be measured or quantified (e.g., performance benchmarks, security standards).

**Compile-Time Caching**

**Definition:**  
Compile-time caching stores pre-compiled assets or intermediate build results (like Java bytecode, bundled JavaScript, or optimized images) during the **build process**.

**Purpose:**

* Speeds up the build process.
* Prevents redoing expensive computations or transformations.
* Common in CI/CD pipelines.

**Examples:**

* Webpack caching when bundling JS/CSS.
* Docker layer caching (e.g., if the base image or dependencies haven’t changed).
* Maven/Gradle cache in Java builds.

**Benefits:**

* **Faster builds.**
* **Efficient resource usage.**
* Reduces build time across environments and teams.

**Runtime Caching**

**Definition:**  
Runtime caching refers to storing data during application execution, so it doesn't need to be fetched or recalculated repeatedly.

**Purpose:**

* Reduces latency for frequently accessed data.
* Minimizes backend/API/database load.

**Examples:**

* Caching database queries in Redis.
* Browser caching for static assets.
* In-memory cache using tools like Ehcache, Guava, or Spring Cache.

**Benefits:**

* Better performance.
* Reduced server load.
* Improved scalability.

**Alpine Linux**

**Definition:**  
**Alpine Linux** is a **security-oriented, lightweight Linux distribution** designed for power users who need performance and simplicity. It is musl libc- and BusyBox-based, making it extremely small and efficient.

**Benefits of Alpine Linux in Docker:**

| **Feature** | **Benefit** |
| --- | --- |
| **Small Image Size** | Typically ~5 MB, reducing image size and speeding up pulls. |
| **Minimal Packages** | Only the essentials, reducing attack surface. |
| **Security Focused** | Hardened kernel, PaX, grsecurity patches (historically). |
| **Customizability** | Easy to extend and build from. |
| **Community Support** | Well-documented and supported for Docker use cases. |

**Common Use Cases:**

* Microservices in containers (e.g., Go, Node.js, Python apps).
* CI/CD tools and pipelines.
* Serverless functions (custom containers).
* Lightweight VMs in edge computing.

**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, Virtual Network, Azure LoadBalancer for simplicity.
7. **Use Application Security Groups (ASGs)**: Group VMs logically and apply rules to ASGs instead of IPs.

**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.

**Bastion Host:**

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.

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

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

**Use Cases:**

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

**Region:**

A **Region** is a **geographical area** that contains at least one or more **data centers**. Each region is **isolated** from others to ensure fault tolerance and data sovereignty.

**🔹 Key Points:**

* A region can represent a location like **East US**, **West Europe**, **Central India**, etc.
* Cloud providers like AWS, Azure, and GCP offer services across many regions worldwide.
* Organizations choose regions based on:
  + **User proximity** (for low latency)
  + **Compliance or regulatory needs**
  + **Disaster recovery planning**

**Availability Zones (AZs):**

Availability Zones are **physically separate data centers** within a region. They are designed to be **independent** from each other in terms of power, networking, and cooling — but they are **interconnected with high-speed links**.

**🔹 Key Points:**

* Each **AZ** is an **isolated location** inside a region.
* A region typically has **2 or more AZs**.
* If one AZ fails (e.g., due to a power outage or natural disaster), others can **take over**, ensuring **high availability**.
* Applications can be **replicated across multiple AZs** for redundancy and fault tolerance.

**Express Gateways :**

**Express Gateway** is an **open-source API Gateway** built entirely on top of **Node.js** and the popular **Express.js framework**. It’s designed to **manage microservices APIs**, offering features like **authentication, rate-limiting, logging, and monitoring** — all easily configured using a simple YAML file.

**Key Features of Express Gateway:**

* **Microservices Gateway**: Routes requests to multiple backend services (APIs).
* **Built on Express.js**: Leverages the simplicity and flexibility of Express.
* **Authentication & Authorization**: Built-in support for OAuth2, key authentication.
* **Rate Limiting**: Prevents abuse by controlling how often users can access APIs.
* **Plugin System**: Extend functionality with custom plugins.
* **Easy Configuration**: Centralized, YAML-based config — no complex setup.
* **Open Source**: Completely free and community-supported.

**Benefits:**

* Quick to set up and use.
* Ideal for **Node.js and Express-based environments**.
* Flexible enough for small to medium-scale microservice architectures

**Benefits:**

* **Secure communication** between networks.
* **Low latency & high bandwidth** (since traffic doesn’t go through the public internet).
* **No VPN needed** — simpler and cheaper.
* **Resources like VMs, databases, etc., can talk to each other** across networks.
* **Cost Effective:** Without peering, transferring data between VNets requires using public Ips or VPNs which can be costly.

**What is a Subnet?**

A **subnet (subnetwork)** is a smaller, logical section of a larger network (like a VPC in AWS or VNet in Azure). It helps divide a big network into smaller, manageable chunks.

**Why Do We Use Subnets?**

* To **organize** and **isolate** resources (e.g., frontend vs backend).
* To apply **security rules** (e.g., NSGs or firewalls) on specific subnets.
* To control **traffic flow** and improve **performance**.

**What is Subnetworking (Subnetting) :**

**Subnetworking**, also known as **subnetting**, is the practice of dividing a larger IP network into smaller, manageable **subnets**. This allows better control of traffic flow, more efficient IP address usage, and improved network security.

**Why is Subnetworking Important?**

Imagine you have a company with one big office network. Without subnets, **every computer, server, printer, and device shares the same broadcast space** — this creates unnecessary traffic and security risks.

By subnetting:

* You **organize** the network into logical groups (e.g., HR, IT, Finance).
* You reduce **broadcast traffic**.
* You apply **different access controls** to each subnet.
* You **optimize** how IP addresses are allocated.

**IP Address and CIDR Notation**

Every device on a network has an **IP address**, like 192.168.1.1. Subnets are defined using **CIDR notation** (Classless Inter-Domain Routing), e.g., 192.168.1.0/24.

* 192.168.1.0 is the **network address**
* /24 means the **first 24 bits** are fixed as the network part, leaving **8 bits** for host addresses.

This gives 2^8 = 256 total IPs (including network and broadcast addresses).

**Subnetting Example**

You have a network 192.168.1.0/24 (256 IPs), and you want to split it into 4 subnets:

| **Subnet** | **Range** | **Hosts** |
| --- | --- | --- |
| 192.168.1.0/26 | 192.168.1.0–63 | 62 usable |
| 192.168.1.64/26 | 192.168.1.64–127 | 62 usable |
| 192.168.1.128/26 | 192.168.1.128–191 | 62 usable |
| 192.168.1.192/26 | 192.168.1.192–255 | 62 usable |

**Benefits of Subnetting**

1. **Improved Security** – Restrict access between subnets with firewalls or NSGs.
2. **Better Performance** – Reduced broadcast domains = faster network.
3. **Efficient IP Usage** – No waste of IPs; assign based on exact need.
4. **Scalability** – Makes large networks easier to manage and expand

**Azure Use Case**: Subnets are part of a **VNet (Virtual Network)**.  
Subnetwork Example:

10.0.0.0/24->10.0.0.0-----10.0.0.255 32-24=8 2^8=256  
Network - 1 2 4 8 16 32 64 128 256  
Host - 256 128 64 32 16 8 4 2 1

Subnet mask - /24 /25 /26 /27 /28 /29 /30 /31 /32

IP Range Network ID BroadcastID IP Representation

Team -01- 10.0.0.1-10.0.0.62 10.0.0.0 10.0.0.63 10.0.0.0/26

Team -02 - 10.0.0.65-10.0.0.126 10.0.0.64 10.0.0.127 10.0.0.64/26  
Team -03- 10.0.0.129-10.0.0.190 10.0.0.128 10.0.0.191 10.0.0.128/26

A **Virtual Machine (VM)** is a software-based emulation of a physical computer. It runs an operating system and applications just like a real computer, but it does so inside a host system using virtualization technology.

**Key Points:**

* **Isolated Environment**: Each VM operates independently, with its own OS, resources (CPU, memory, storage), and applications.
* **Runs on a Hypervisor**: A hypervisor (like VMware, Hyper-V, or KVM) manages and runs multiple VMs on a single physical host.
* **Used For**: Development, testing, running legacy apps, hosting services, and scaling cloud environments.
* **Popular in Cloud**: In platforms like Azure or AWS, VMs are on-demand and can be scaled based on need.

**What is a VNet?**

**VNet (Virtual Network)** is **Azure's** way of creating your own **private network** in the cloud.

* It’s like your **own data center network**, but virtual.
* You can define **IP address ranges**, **subnets**, and apply **security rules**.
* You can connect **VMs, databases, web apps**, and other services inside the VNet.
* Think of it like a **custom-made highway** for your cloud resources to talk to each other **securely and privately**.

**What is VNet Peering (or VPC Peering )**

**VNet Peering** is the process of connecting **two virtual networks** so they can **communicate privately**, **just like one network**.

* It lets **resources in different VNets talk to each other** (like a VM in VNet A accessing a database in VNet B).
* The communication is **secure, low-latency**, and **doesn’t go over the public internet**.
* It’s like **building a private tunnel** between two highways (VNets).

**Why Use VNet/VPC Peering?**

* To connect **different applications** that are split across VNets.
* To keep **environments separate** (like Dev, Test, Prod) but still allow some **controlled communication**.
* To connect **different regions or subscriptions** within your cloud setup.

**Benefits of VNet/VPC Peering:**

* **Private traffic** (no exposure to internet).
* **Low latency and high bandwidth**.
* **No need for VPNs** between VNets.
* **Easy to set up** and manage.

**Example:**

Imagine:

* VNet-A has your **web servers**.
* VNet-B has your **databases**.

With **VNet Peering**, the web servers in VNet-A can securely talk to the databases in VNet-B using **private IP addresses** — no internet involved.

**VM Scale Sets (VMSS) :**

A **Virtual Machine Scale Set (VMSS)** is an **Azure compute service** that lets you **deploy and manage a group of identical, load-balanced virtual machines**.

VM Scale Sets help you automatically **scale out** (add more VMs) or **scale in** (remove VMs) based on demand.

**Key Features of VMSS**

| **Feature** | **Description** |
| --- | --- |
| **Auto-scaling** | Automatically adjusts the number of VM instances based on load (CPU, memory, or custom metrics). |
| **Load balancing** | Works with Azure Load Balancer or Application Gateway to distribute traffic. |
| **Fault domain distribution** | VMs are spread across fault and update domains to ensure high availability. |
| **Integrated with Availability Zones** | Can span across multiple zones in a region for disaster tolerance. |
| **Supports custom images** | You can use your own VM image or standard Azure images. |
| **Built-in monitoring** | Integrates with Azure Monitor and Log Analytics. |
| **Supports Linux and Windows VMs** | You can deploy scale sets using either OS. |

**How VMSS Works**

1. **You define a scale set configuration:**
   * VM image (Ubuntu, Windows Server, etc.)
   * VM size (e.g., Standard\_B2s)
   * Autoscaling rules (CPU > 75%, scale out; CPU < 30%, scale in)
   * Network settings (subnet, NSG, load balancer)
2. **Azure creates the initial number of VMs** (e.g., 2).
3. **Traffic is routed** through a load balancer to the VMs.
4. When demand increases, Azure **adds more VMs** to handle traffic.
5. When demand drops, Azure **removes unnecessary VMs**, saving cost.

**Example Use Case**

**A Web App That Gets Heavy Evening Traffic**

* During the day: 2 VMs handle the traffic.
* At night: Traffic increases → scale set grows to 8 VMs.
* After midnight: Traffic drops → Azure removes extra VMs, back to 2.

This makes your app scalable **without overpaying** for idle resources.

**Scaling Methods in VMSS**

1. **Manual Scaling**  
   You set a fixed number of instances manually.
2. **Custom Auto-scaling**  
   Based on rules (CPU, memory, etc.) or on a schedule.
3. **Application-based Auto-scaling**  
   Trigger scale-out based on custom metrics (queue length, response time).

**When to Use VM Scale Sets**

Use cases:

* High-traffic web applications
* API backends
* Microservices
* Stateless services
* Batch processing

Not ideal for:

* Stateful apps without external storage
* Apps needing persistent local storage

**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?**

A **Fault Domain** is a **logical grouping of hardware resources** (like power supply, network, and physical servers) within a data center that **shares a single point of failure**.

**Purpose:**

It protects your VMs from **hardware failures**. When you deploy multiple VMs, Azure distributes them across fault domains to avoid all your VMs being affected by a single failure.

**Example:**

If you deploy 3 VMs and specify 2 fault domains:

* VM1 goes to Fault Domain 1
* VM2 goes to Fault Domain 2
* VM3 goes back to Fault Domain 1

So, if Fault Domain 1 fails (e.g., power outage), only VM1 and VM3 are affected — **not all** VMs.

**What is an Update Domain?**

An **Update Domain** is a logical group used to **separate VMs for software or OS updates** to **ensure availability** during maintenance.

**Purpose:**

Azure uses Update Domains to **roll out updates one group at a time**, **not all at once**, reducing downtime.

**Example:**

If you have 5 VMs and 3 update domains:

* VM1 → Update Domain 1
* VM2 → Update Domain 2
* VM3 → Update Domain 3
* VM4 → Update Domain 1
* VM5 → Update Domain 2

Azure updates one domain at a time, waits for success, then proceeds. So, your app **stays online** during updates.

**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: 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.**

**Azure Web App :**

**Azure Web App** (part of **Azure App Service**) is a **PaaS (Platform as a Service)** offering that allows developers to build and host **web apps**, **APIs**, and **mobile backends** in a **fully managed environment**.

It takes care of **infrastructure management**, such as server maintenance, patching, load balancing, scaling, and security — so you can focus on writing code.

**Key Features of Azure Web App**

**1. Multiple Language Support**

* Run applications written in:
  + **.NET / .NET Core**
  + **Java**
  + **Node.js**
  + **Python**
  + **PHP**
  + **Ruby**
  + You can also run **custom containers** using Docker.

**2. Built-in CI/CD Integration**

* Azure Web App integrates easily with:
  + **GitHub**
  + **Azure Repos**
  + **Bitbucket**
  + **GitLab**
* Supports automatic deployments with each code push.
* Works with **GitHub Actions** and **Azure DevOps Pipelines**.

**3. Scaling and Load Balancing**

* Supports **auto-scaling** based on:
  + CPU/Memory usage
  + HTTP queue length
  + Custom rules
* **Load balancing** is handled automatically behind the scenes.
* Supports **horizontal and vertical scaling**.

**4. Global Availability**

* Host your app in **any Azure region**.
* Use **Traffic Manager** for geo-distributed routing and high availability.

**5. Custom Domains & SSL**

* Easily bind **custom domains**.
* Free **SSL certificate** via Azure or upload your own.

**6. Security & Authentication**

* Built-in authentication support with:
  + **Azure AD**
  + **Google**
  + **Facebook**
  + **Twitter**
  + **Microsoft Accounts**
* Supports **managed identities**, **VNet integration**, and **private endpoints**.

**7. Monitoring & Diagnostics**

* Integrated with **Azure Monitor**, **Application Insights**, and **Log Analytics**.
* Provides real-time monitoring, logging, and performance analytics.

**8. Backup and Restore**

* Automated **daily backups**.
* One-click **restore** support.

**9. Deployment Slots**

* Test in **staging slots** before pushing to production.
* Supports **zero-downtime deployments** by swapping slots.

**Use Cases**

* Hosting dynamic websites or web applications
* Running backend APIs for mobile or web apps
* Creating portals or dashboards
* Hosting e-commerce or SaaS platforms
* Integrating with Logic Apps, Functions, and databases

**Benefits :**

* No infrastructure management
* Quick deployments
* Highly scalable
* Cost-effective (pay-as-you-go model)
* Enterprise-grade security

**Azure Subscription:**

An Azure Subscription is essentially a logical container for your Azure resources. It is the boundary within which Azure resources (such as virtual machines, databases, web apps, etc.) are created and managed. A subscription allows users to manage and organize their resources based on different factors such as project, department, or environment (production, development, etc.).

**Types of Azure Subscriptions:**

1. **Free**: Provides a limited number of resources for free (good for testing or learning).
2. **Pay-As-You-Go**: Charges based on the consumption of resources.
3. **Enterprise Agreement**: Designed for larger organizations with complex billing needs.
4. **Visual Studio Subscription**: For developers, offering discounts or credits.

**Azure Resource Groups:**

An **Azure Resource Group** is a container that holds related Azure resources. Resources within the same resource group share a lifecycle, meaning you can manage, update, and delete them together.

Resource groups are useful for organizing and managing your Azure resources in a logical, structured way.

**Key Points about Azure Resource Groups**:

* Resource Organization
* Management and Deployment
* Access Control
* Resource Lifecycle
* Geographical Location
* Tagging

**Best Practices:**

* Group resources logically by **application, team, or environment**.
* Keep resource groups **manageable and clean**.
* Apply **minimum necessary permissions** using RBAC.
* Keep resources in the **same region** when possible to reduce latency.

**Virtual Machine Setup and Network Configuration**

**Step 1: Creating a New User in Microsoft Entra ID**

Microsoft Entra ID (formerly Azure Active Directory) allows you to manage identities and access in Azure. Creating a user is essential for assigning roles and securing access.

**Step-by-Step Guide to Create a New User**

**1. Sign in to Azure Portal**

* Go to <https://portal.azure.com>
* Sign in with your administrator credentials.

**2. Navigate to Microsoft Entra ID**

* In the left-hand menu or search bar, type **“Microsoft Entra ID”** or **“Azure Active Directory”**.
* Click on **Microsoft Entra ID** to open the directory.

**3. Go to Users Section**

* In the Microsoft Entra ID pane, click on **“Users”** from the left sidebar.
* Then click on **“+ New user”** at the top.

**4. Choose User Type**

* Select **“Create new user”** (default).
* You can also choose to invite an external user if needed (for B2B collaboration).

**5. Fill in User Details**

* **User name**: Enter a unique username (e.g., devuser@yourdomain.onmicrosoft.com)
* **Name**: Enter the full name of the user.
* Optionally, you can add additional details such as:
  + **Job Title**
  + **Department**
  + **Usage location**

**6. Set Password**

* Azure will auto-generate a temporary password.
* You can also set a custom password.
* Make sure to **note down the password**, as the user will be prompted to change it at first login.

**7. Assign Groups or Roles (Optional)**

* You can assign the user to specific **groups** or **roles**:
  + Example: Assign as **Contributor**, **Reader**, or **Virtual Machine Operator**.
* To assign roles:
  + Click on **“Roles”** > **“+ Add assignment”** > choose a role.

**8. Review and Create**

* Review all entered information.
* Click **“Create”**.

**Result**

The user is now created and can:

* Log in via the Azure Portal
* Access only the assigned resources
* Be used for managing permissions on virtual machines, networks, storage, etc

**Creating the Virtual Network and Virtual Machines**

**Virtual Network (VNet)**

* Create a **Virtual Network** to logically group and connect all virtual machines (VMs).
* This allows communication between VMs in the same VNet.

**Virtual Machines**

* Create **three VMs**:
  + **Frontend VM** (Public)
  + **Backend VM** (Intermediate)
  + **Private VM** (Internal-only)
* Assign each VM to the **same virtual network** but place them in appropriate subnets (e.g., public, private).

**Network Security Groups (NSGs)**

**Creation**

* Create **two NSGs**:
  + **Public NSG** (for Frontend and Backend)
  + **Private NSG** (for Private VM)

**Association and Disassociation**

* **Stop each VM** to safely change its NIC (Network Interface Card) settings.
* **Disassociate** the NICs from default NSGs if any.
* **Re-associate** the correct NSG:
  + Public NSG → Frontend and Backend
  + Private NSG → Private VM

**Configuring NSG Rules**

**Private NSG – Inbound Rules**

* **Priority 120** → **Deny All** traffic (to ensure security).
* **Priority 110** → **Allow SSH** or specific internal traffic (to allow internal VM communication).

**Public NSG – Inbound Rules**

* **Priority 105** → **Allow SSH (port 22)** to access the Frontend VM from your local system.

**Key Insertion and SSH Connections**

**Step-by-Step SSH Access**

1. **From Local to Frontend**:
   * Use SSH key to connect to the Frontend VM from your local machine:

* ssh -i keyname.pem username@frontend-ip

1. **Copy Key from Frontend to Backend**:
   * Use SCP to copy the SSH key:

* scp -i keyname.pem keyname.pem username@backend-ip:/home/username/
  + Then SSH into the Backend from the Frontend:
* ssh -i keyname.pem username@backend-ip

1. **From Backend to Private VM**:
   * Use SSH again (use the Private VM’s **private IP** if in same VNet):

* ssh -i keyname.pem username@private-ip

**Note:** If VMs are in the same VNet, you don’t need a public IP to SSH internally.

**Microsoft Entra ID (Formerly Azure Active Directory)**

Microsoft Entra ID (formerly Azure Active Directory, or Azure AD) is a cloud-based identity and access management service. It helps organizations securely manage and control access to resources.

**Key Features:**

* **Identity Management:** Manages user identities and access permissions.
* **Single Sign-On (SSO):** Users can access multiple applications with one login.
* **Multi-Factor Authentication (MFA):** Adds an extra layer of security.
* **Conditional Access:** Restricts or allows access to applications based on user conditions (location, device, etc.).
* **Integration:** Integrates with various third-party applications and services.

**Use Cases:**

* Providing secure access to cloud and on-premises resources.
* Managing employee and contractor access to enterprise applications.
* Securing APIs and services in the cloud.

**2. Virtual Machines (VMs) in Azure**

Azure Virtual Machines (VMs) allow you to run a wide variety of operating systems in the cloud. You can deploy, manage, and scale VM instances with Azure.

**Key Features:**

* **Customizable OS:** Supports Windows, Linux, and other operating systems.
* **Scalable:** VMs can be scaled up or down based on demand.
* **Secure:** Includes built-in security with Azure Security Center.
* **Integration:** VMs can be integrated with other Azure services like networking, storage, and monitoring.

**Use Cases:**

* Running applications that require full control of the operating system.
* Hosting legacy applications.
* Testing or developing software in a virtualized environment.

**3. Azure Resource Groups**

A resource group in Azure is a container that holds related resources for an Azure solution. It allows you to manage and organize Azure resources in a structured way.

**Key Features:**

* **Logical Grouping:** Organizes resources like virtual machines, storage accounts, etc., into a single group.
* **Unified Management:** Allows easy management, monitoring, and deployment of resources as a unit.
* **Access Control:** Resources within a group can have Role-Based Access Control (RBAC) policies applied.

**Use Cases:**

* Organizing resources by environment (dev, test, production).
* Grouping resources by project or business unit.
* Managing access and policies at the resource group level.

**4. Azure App Services**

Azure App Services is a fully managed platform for building, deploying, and scaling web apps, mobile backends, and APIs.

**Key Features:**

* **Auto-Scaling:** Automatically scales web apps based on demand.
* **Integration:** Supports integration with databases, caching, and other Azure services.
* **Multi-language Support:** Supports .NET, Node.js, Python, Java, and more.
* **Security Features:** Provides built-in authentication, authorization, and SSL certificates.

**Use Cases:**

* Hosting web applications (e-commerce sites, content management systems).
* Running REST APIs for mobile or web applications.
* Hosting serverless backends.

**5. Azure Storage Accounts**

Azure Storage Accounts provide scalable, durable, and secure cloud storage for various types of data. It supports data such as blobs, files, queues, and tables.

**Key Features:**

* **Blob Storage:** For storing large amounts of unstructured data, like images, videos, backups.
* **File Storage:** Offers shared storage for legacy applications.
* **Queue and Table Storage:** For message queues and NoSQL-like database solutions.
* **Durability and Availability:** Multiple replication options to ensure data durability and availability.

**Use Cases:**

* Storing application data, backups, and media files.
* Providing file share services to on-premises applications.
* Storing large datasets for Big Data and analytics.

**6. Azure SQL Databases**

Azure SQL Database is a fully managed relational database service based on SQL Server. It offers high availability, automatic scaling, and built-in security.

**Key Features:**

* **Automated Backups:** SQL Database automatically handles backups and restores.
* **Scalable Performance:** Can scale resources dynamically based on workload demands.
* **Security:** Includes built-in encryption and firewall protection.
* **High Availability:** Built-in features like auto-failover groups for high availability.

**Use Cases:**

* Hosting applications that require a relational database.
* Storing business-critical transactional data.
* Running analytics on structured data.

**7. Cost Management in Azure**

Azure Cost Management is a tool to monitor and control your Azure spending. It provides insights into where and how costs are incurred, and helps you optimize your spending.

**Key Features:**

* **Cost Analysis:** Helps visualize and analyze costs by resource, resource group, subscription, etc.
* **Budgets:** Set up budgets for different teams, departments, or projects to avoid overspending.
* **Cost Recommendations:** Provides insights on how to optimize costs, such as resizing underused resources.
* **Forecasting:** Predict future costs based on historical data.
* **Alerts:** Set up cost alerts to notify you when spending approaches a defined limit.

**Use Cases:**

* Monitoring Azure resource usage and expenditures.
* Optimizing cloud resources to reduce unnecessary costs.
* Setting and tracking budgets for departments or project.