Azure-900

Cloud:
It's a term used to describe a global network of servers, each with a unique function. The cloud is not a physical entity, but instead is a vast network of remote servers around the globe which are hooked together and meant to operate as a single ecosystem. These servers are designed to either store and manage data, run applications, or deliver content or a service such as streaming videos, web mail, office productivity software, or social media. Instead of accessing files and data from a local or personal computer, you are accessing them online from any Internet-capable device—the information will be available anywhere you go and anytime you need it.

Cloud Computing:

Delivering computing services over the internet. Computing services include common IT infrastructure such as virtual machines, storage, databases, and networking. Cloud services also expand the traditional IT offerings to include things like Internet of Things (IoT), machine learning (ML), and artificial intelligence (AI).

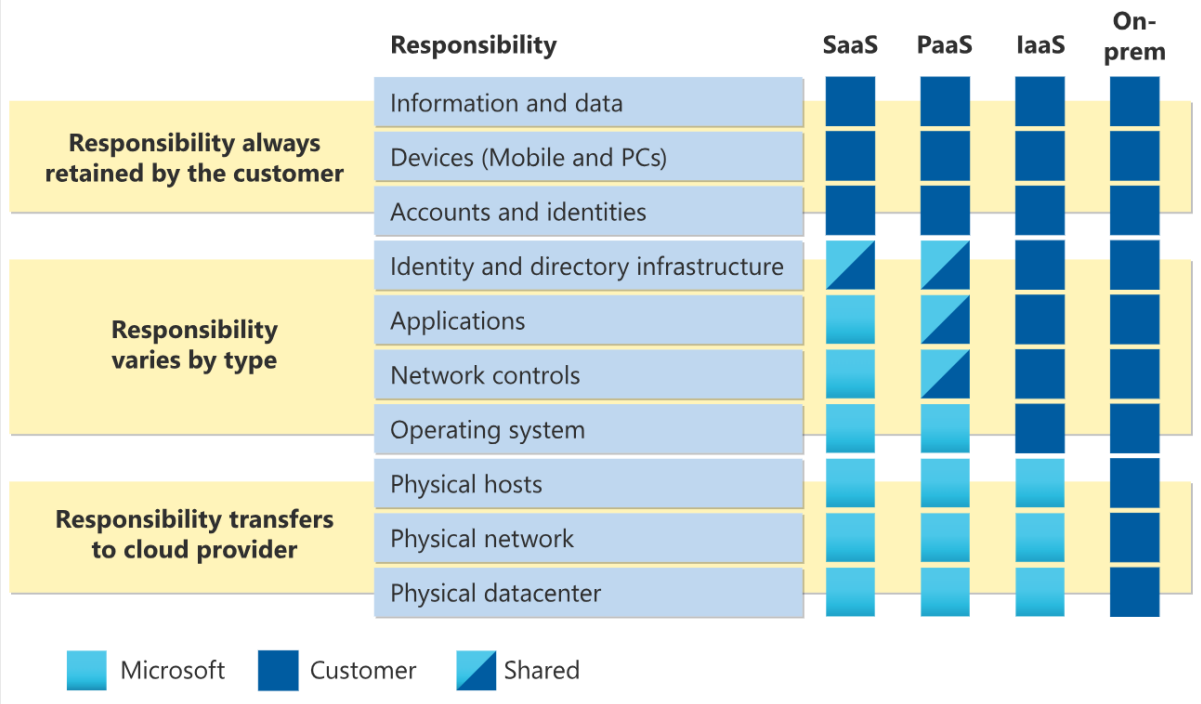
Shared Responsibility Model:

With an on-premises datacenter, you are responsible for everything. With cloud computing, those responsibilities shift.

IaaS (infrastructure as a service) places the most responsibility on the consumer, with the cloud provider being responsible for the basics of physical security, power, and connectivity.

SaaS (software as a service) places most of the responsibility with the cloud provider.

PaaS (platform as a service) evenly distributes responsibility between the cloud provider and the consumer.



☁️ Azure Cloud Service Categories – Summary Notes

1️⃣ IaaS (Infrastructure as a Service)

- **You manage:** OS, runtime, apps, data
- **Azure manages:** Hardware, networking, virtualization
- **Examples:** Azure Virtual Machines, Azure Storage, Azure Virtual Network
- **Use case:** When you want full control of the OS and environment (e.g., installing custom software)

💡 **Scenario:**
👉 *Deploy custom application in virtual machines* → **IaaS**

2️⃣ PaaS (Platform as a Service)

- **You manage:** Applications and data only
- **Azure manages:** OS, runtime, middleware, scaling, availability
- **Examples:** Azure App Service, Azure SQL Database, Azure Functions
- **Use case:** When you just want to focus on your code, not infrastructure

💡 **Scenario:**
👉 *Using Azure App Service to deploy your app* → **PaaS**

3️⃣ SaaS (Software as a Service)

- **You only use the software** — everything else (infra, app, data) is managed by the provider
- **Examples:** Microsoft 365, Gmail, Salesforce
- **Use case:** When you just want to *use* a complete app

💡 **Scenario:**
👉 *Using Gmail* → **SaaS**

☑️ True or False Explanations

Scenario	Answer	Explanation
Customer is responsible for OS updates when using **PaaS**	❌ False	Azure handles OS and runtime updates.
Customer is responsible for **Availability** when using **PaaS**	❌ False	Azure ensures uptime and availability for the platform.

In **PaaS**, customer has access to VM instances	❌ False	You don't manage the VMs — Azure abstracts them.
In **PaaS**, customer can customize OS and install software	❌ False	You can't modify OS — only deploy your app code.
In **PaaS**, customer can configure **auto-scaling** needs	✅ True	You can set scaling rules, and Azure scales automatically.
In **PaaS**, customer can configure **hardware needs** (CPU, memory)	✅ True	You can choose instance size/tier (like Basic, Standard, Premium).
PaaS services only offer compute services	❌ False	PaaS also offers DBs, web apps, integration, analytics, etc.

🧩 In Simple Terms

Feature	IaaS	PaaS	SaaS
OS Control	✅	❌	❌
Runtime Control	✅	❌	❌
App Deployment	✅	✅	❌
User Access	✅	✅	✅
Managed by Provider	Hardware, Network	OS + Runtime	Everything
Example	Azure VM	Azure App Service	Gmail, Microsoft 365

Consumption-based model

When comparing IT infrastructure models, there are two types of expenses to consider. Capital expenditure (CapEx) and operational expenditure (OpEx).

CapEx is typically a one-time, up-front expenditure to purchase or secure tangible resources. A new building, repaving the parking lot, building a datacenter, or buying a company vehicle are examples of CapEx.

In contrast, OpEx is spending money on services or products over time. Renting a convention center, leasing a company vehicle, or signing up for cloud services are all examples of OpEx.

Cloud computing falls under OpEx because cloud computing operates on a consumption-based model. With cloud computing, you do not pay for the physical infrastructure, the electricity, the security, or anything else associated with maintaining a datacenter. Instead, you pay for the IT resources you use. If you do not use any IT resources this month, you do not pay for any IT resources.

High availability in the cloud

When you are deploying an application, a service, or any IT resources, it is important the resources are available when needed. High availability focuses on ensuring maximum availability, regardless of disruptions or events that may occur.

This is important for mission-critical systems that cannot tolerate interruption in service, and any downtime can cause damage or result in financial loss.

Highly available systems guarantee a certain percentage of uptime—for example, a system that has 99.9% uptime will be down only 0.1% of the time—0.365 days or 8.76 hours per year. The number of "nines" is commonly used to indicate the degree of high availability. For example, "five nines" indicates a system that is up 99.999% of the time.

When you are architecting your solution, you will need to account for service availability guarantees. Azure is a highly available cloud environment with uptime guarantees depending on the service. These guarantees are part of the service-level agreements (SLAs).

For more info refer: high-availability-basic-concepts-and-a-checklist

Scalability in the cloud

Scalability refers to the ability to adjust resources to meet demand. If you suddenly experience peak traffic and your systems are overwhelmed, the ability to scale means you can add more resources to better manage the increased demand.

The other benefit of scalability is that you are not overpaying for services. Because the cloud is a consumption-based model, you only pay for what you use. If demand drops off, you can reduce your resources and thereby reduce your costs.

Scaling comes in two varieties: vertical and horizontal. Vertical scaling is focused on increasing or decreasing the capabilities of resources. Horizontal scaling is adding or subtracting the number of resources.

- **Vertical scaling**

  With vertical scaling, if you were developing an app and you needed more processing power, you could vertically scale up to add more CPUs or RAM to the virtual machine. Conversely, if you realized you had over-specified the needs, you could vertically scale down by lowering the CPU or RAM specifications.

- **Horizontal scaling**

  With horizontal scaling, if you suddenly experienced a steep jump in demand, your deployed resources could be scaled out (either automatically or manually). For example, you could add additional virtual machines or containers, scaling out. In the same manner, if there was a significant drop in demand, deployed resources could be scaled in (either automatically or manually), scaling in.

Reliability in the cloud

Reliability is the ability of a system to recover from failures and continue to function. With a decentralized design, the cloud enables you to have resources deployed in regions around the world. With this global scale, even if one region has a catastrophic event other regions are still up and running.

Predictability in the cloud

Predictability can be focused on performance predictability or cost predictability.

- Performance predictability focuses on predicting the resources needed to deliver a positive experience for your customers. Autoscaling, load balancing, and high availability are just some of the cloud concepts that support performance predictability.
- Cost predictability is focused on predicting or forecasting the cost of the cloud spend.

Azure physical infrastructure

The core architectural components of Azure may be broken down into two main groupings: the physical infrastructure, and the management infrastructure.

Physical infrastructure
The physical infrastructure for Azure starts with datacenters. Conceptually, the datacenters are the same as large corporate datacenters. They're facilities with resources arranged in racks, with dedicated power, cooling, and networking infrastructure. As a global cloud provider, Azure has datacenters around the world. However, these individual datacenters aren't directly accessible. Datacenters are grouped into Azure Regions or Azure Availability Zones.

Regions

- A region is a geographical area on the planet that contains at least one, but potentially multiple datacenters that are nearby and networked together with a low-latency network. Azure intelligently assigns and controls the resources within each region to ensure workloads are appropriately balanced.
- When you deploy a resource in Azure, you'll often need to choose the region where you want your resource deployed.
- Some services or virtual machine (VM) features are only available in certain regions, such as specific VM sizes or storage types. There are also some global Azure services that don't require you to select a particular region, such as Microsoft Entra ID, Azure Traffic Manager, and Azure DNS.

Availability Zones

- Availability zones are physically separate datacenters within an Azure region. Each availability zone is made up of one or more datacenters equipped with independent power, cooling, and networking. An availability zone is set up to be an isolation boundary. If one zone goes down, the other continues working. Availability zones are connected through high-speed, private fiber-optic networks.
- To ensure resiliency, a minimum of three separate availability zones are present in all availability zone-enabled regions. However, not all Azure Regions currently support availability zones.
- if you deploy your application or database in **just one AZ**, and that AZ goes down, your service goes down.
  But if you spread (or replicate) your resources across **multiple AZs** in the same region, then:
  - If one AZ fails, traffic can still be served from the other AZ(s).
  - That's what we call **redundancy** (a backup setup that increases reliability).

Use availability zones in your apps

You want to ensure your services and data are redundant so you can protect your information in case of failure. When you host your infrastructure, setting up your own redundancy requires that you create duplicate hardware environments. Azure can help make your app highly available through availability zones.

You can use availability zones to run mission-critical applications and build high-availability into your application architecture by co-locating your compute, storage, networking, and data resources within an availability zone and replicating in other availability zones. Keep in mind that there could be a cost to duplicating your services and transferring data between availability zones.

Availability zones are primarily for VMs, managed disks, load balancers, and SQL databases. Azure services that support availability zones fall into three categories:

- Zonal services: You pin the resource to a specific zone (for example, VMs, managed disks, IP addresses).
- Zone-redundant services: The platform replicates automatically across zones (for example, zone-redundant storage, SQL Database).
- Non-regional services: Services are always available from Azure geographies and are resilient to zone-wide outages as well as region-wide outages.

Even with the additional resiliency that availability zones provide, it's possible that an event could be so large that it impacts multiple availability zones in a single region. To provide even further resilience, Azure has Region Pairs.

Region pairs

- Most Azure regions are paired with another region within the same geography (such as US, Europe, or Asia) at least 300 miles away. This approach allows for the replication of resources across a geography that helps reduce the likelihood of interruptions because of events such as natural disasters, civil unrest, power outages, or physical network outages that affect an entire region. For example, if a region in a pair was affected by a natural disaster, services would automatically fail over to the other region in its region pair.
- Not all Azure services automatically replicate data or automatically fall back from a failed region to cross-replicate to another enabled region. In these scenarios, recovery and replication must be configured by the customer.
- **By default**
    - Many Azure services only replicate data *within the same region* (sometimes across availability zones if you choose a "zone-redundant" SKU).
    - Cross-region replication (e.g. East US → West US) is **not automatic** for most services.
- **When you enable cross-region replication or disaster recovery (DR):**
    - You're essentially using **two sets of resources**:
        - Primary (in your main region).
        - Secondary (in the paired or chosen failover region).
    - This means **extra cost** for storage, compute, and sometimes data transfer.
- Example:
    - **Azure Storage**: Standard "Locally Redundant Storage (LRS)" is cheapest. If you want "Geo-Redundant Storage (GRS)" or "Read-Access GRS (RA-GRS)," you pay **extra per GB**.
    - **Azure SQL Database**: If you configure a geo-replicated secondary database, you're billed separately for both the primary and the replica.
    - **VMs**: If you set up failover with Azure Site Recovery, you'll pay for replication + storage in the target region (and eventually VM compute costs if you fail over).
- **Customer responsibility**
    - Azure provides the *features* (like Site Recovery, Availability Sets, Zone-Redundant SKUs, or GRS storage).
    - But enabling them is up to you, and yes — you'll be billed for them.
- 💡 In short:
  Yes, it costs extra, because you're essentially reserving extra infrastructure in another region for resilience.

- In Azure, **SKU** stands for **Stock Keeping Unit** — but in cloud terms, you can think of it as **a specific "edition" or "configuration" of a service that defines its features, performance, and price**.

How SKUs fit in:

- Every Azure service (VMs, Storage, Databases, Load Balancers, etc.) has **different SKUs**.
- The SKU you pick determines:
    - **Performance** (e.g., number of cores, IOPS, throughput).
    - **Redundancy/availability features** (e.g., LRS vs GRS for Storage).
    - **Cost** (higher redundancy/performance = higher price).
-

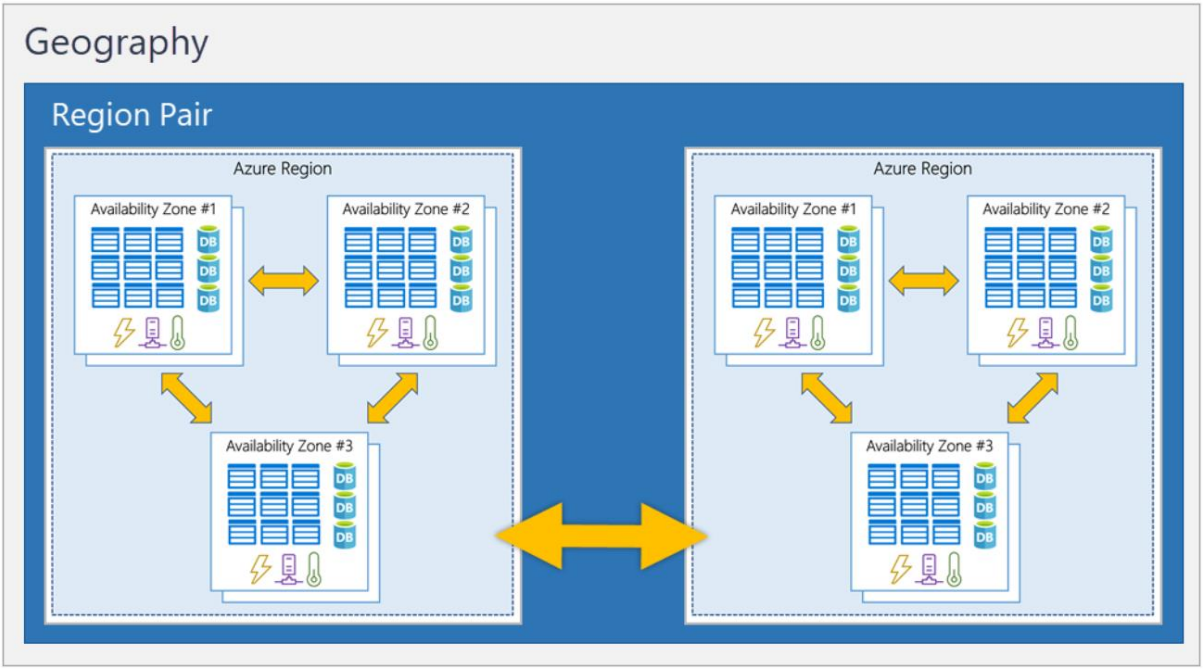
Example 1: Azure Storage SKUs

- **LRS (Locally Redundant Storage)**: cheapest, data kept in one region.
- **ZRS (Zone-Redundant Storage)**: data replicated across 3 availability zones in a region.
- **GRS (Geo-Redundant Storage)**: data replicated to another paired region.
- **RA-GRS (Read-Access GRS)**: same as GRS, but with read access to the replica.
- Each of these is a different **SKU** → different price point.

Example 2: Azure Load Balancer SKUs

- **Basic SKU**: simple, limited features, cheaper.
- **Standard SKU**: zone-redundant, higher scale, more features, higher cost.

- 💡 So when we said earlier *"not all services automatically replicate across regions"*, that's because **only certain SKUs support cross-region redundancy** — and those SKUs cost more.



Sovereign Regions

In addition to regular regions, Azure also has sovereign regions. Sovereign regions are instances of Azure that are isolated from the main instance of Azure. You may need to use a sovereign region for compliance or legal purposes.

Azure sovereign regions include:

- US DoD Central, US Gov Virginia, US Gov Iowa and more: These regions are physical and logical network-isolated instances of Azure for U.S. government agencies and partners. These datacenters are operated by screened U.S. personnel and include additional compliance certifications.
- China East, China North, and more: These regions are available through a unique partnership between Microsoft and 21Vianet, whereby Microsoft doesn't directly maintain the datacenters.

Azure management infrastructure

The management infrastructure includes Azure resources and resource groups, subscriptions, and accounts.

Azure resources and resource groups

A resource is the basic building block of Azure. Anything you create, provision, deploy, etc. is a resource. Virtual Machines (VMs), virtual networks, databases, cognitive services, etc. are all considered resources within Azure.

- Resource groups are simply groupings of resources. When you create a resource, you're required to place it into a resource group.
- While a resource group can contain many resources, a single resource can only be in one resource group at a time.
- Some resources may be moved between resource groups, but when you move a resource to a new group, it will no longer be associated with the former group.

- Additionally, resource groups can't be nested, meaning you can’t put resource group B inside of resource group A.

Azure subscriptions

In Azure, subscriptions are a unit of management, billing, and scale. Similar to how resource groups are a way to logically organize resources, subscriptions allow you to logically organize your resource groups and facilitate billing.

Using Azure requires an Azure subscription. A subscription provides you with authenticated and authorized access to Azure products and services. It also allows you to provision resources. An Azure subscription links to an Azure account, which is an identity in Microsoft Entra ID or in a directory that Microsoft Entra ID trusts.

An account can have multiple subscriptions, but it’s only required to have one. In a multi-subscription account, you can use the subscriptions to configure different billing models and apply different access-management policies. You can use Azure subscriptions to define boundaries around Azure products, services, and resources. There are two types of subscription boundaries that you can use:

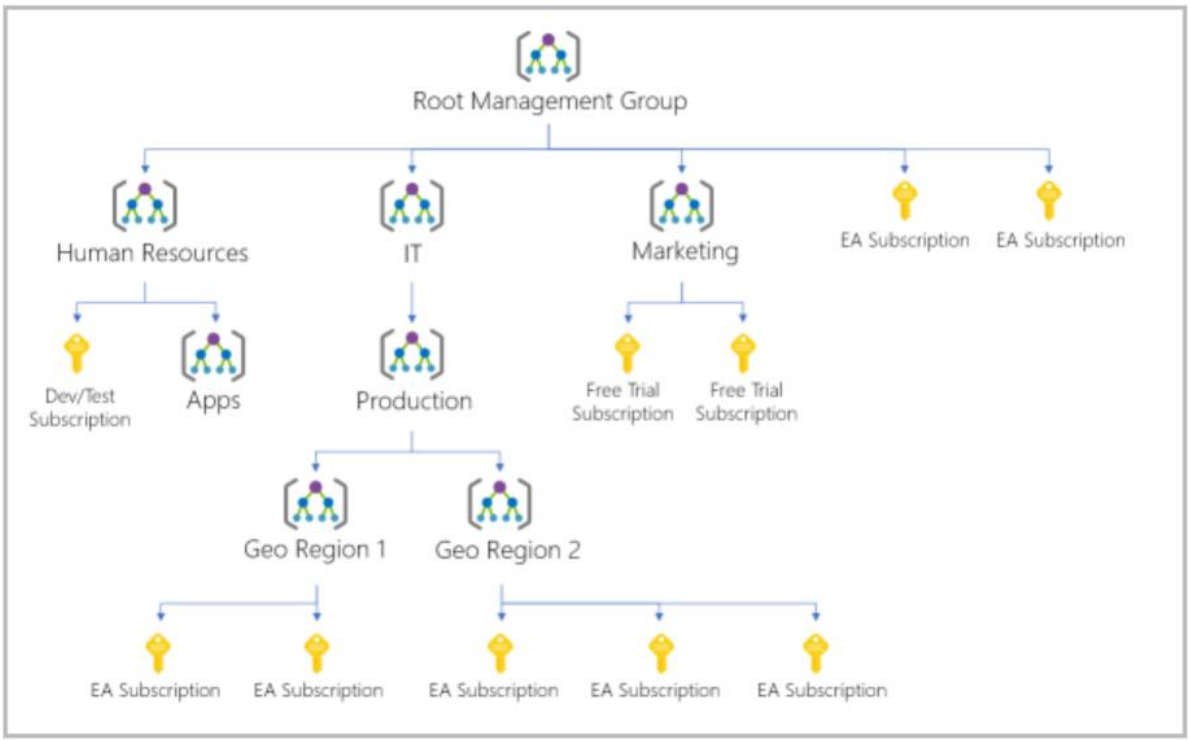
- **Billing boundary**: This subscription type determines how an Azure account is billed for using Azure. You can create multiple subscriptions for different types of billing requirements. Azure generates separate billing reports and invoices for each subscription so that you can organize and manage costs.
- **Access control boundary**: Azure applies access-management policies at the subscription level, and you can create separate subscriptions to reflect different organizational structures. An example is that within a business, you have different departments to which you apply distinct Azure subscription policies. This billing model allows you to manage and control access to the resources that users provision with specific subscriptions.

You can also list your subscriptions and view their IDs programmatically by using Get-AzSubscription (Azure PowerShell) or az account list (Azure CLI).

Azure management groups

The final piece is the management group. Resources are gathered into resource groups, and resource groups are gathered into subscriptions.

If you have many subscriptions, you might need a way to efficiently manage access, policies, and compliance for those subscriptions. Azure management groups provide a level of scope above subscriptions. You organize subscriptions into containers called management groups and apply governance conditions to the management groups. All subscriptions within a management group automatically inherit the conditions applied to the management group, the same way that resource groups inherit settings from subscriptions and resources inherit from resource groups. Management groups give you enterprise-grade management at a large scale, no matter what type of subscriptions you might have. Management groups can be nested.



What is Virtualization?

- Virtualization is technology that you can use to create virtual representations of servers, storage, networks, and other physical machines.
- Virtual software mimics the functions of physical hardware to run multiple virtual machines simultaneously on a single physical machine.
- Virtualization is a process that allows a computer to share its hardware resources with multiple digitally separated environments. Each virtualized environment runs within its allocated resources, such as memory, processing power, and storage.
- With virtualization, organizations can switch between different operating systems on the same server without rebooting.

Virtual machine

- A *virtual machine* is a software-defined computer that runs on a physical computer with a separate operating system and computing resources.
- The physical computer is called the *host machine* and virtual machines are *guest machines*.
- Multiple virtual machines can run on a single physical machine.
- Virtual machines are abstracted from the computer hardware by a hypervisor.

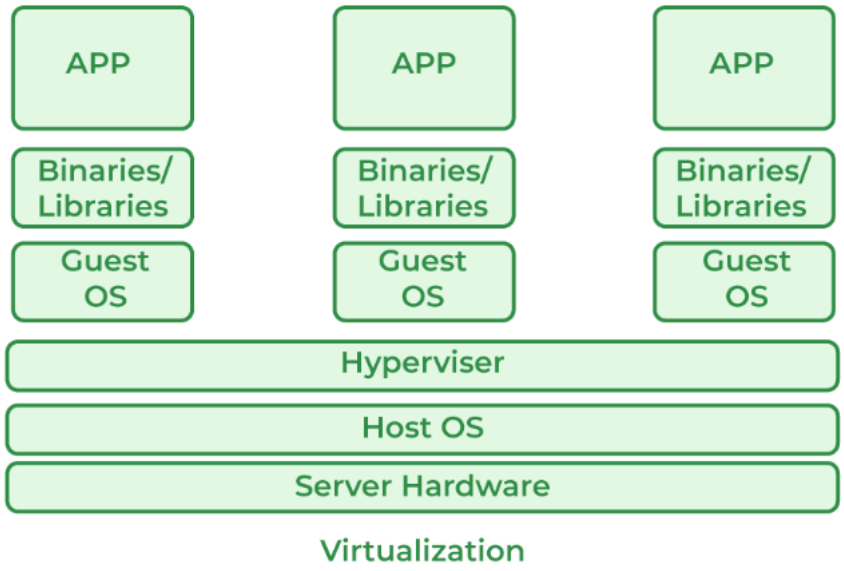
Hypervisor

The *hypervisor* is a software component that manages multiple virtual machines in a computer. It ensures that each virtual machine gets the allocated resources and does not interfere with the operation of other virtual machines.

- The hypervisor is the virtualization software that you install on your physical machine. It is a software layer that acts as an intermediary between the virtual machines and the underlying hardware or host operating system.

- The hypervisor coordinates access to the physical environment so that several virtual machines have access to their own share of physical resources.

For example, if the virtual machine requires computing resources, such as computer processing power, the request first goes to the hypervisor. The hypervisor then passes the request to the underlying hardware, which performs the task.



Types of Virtualization

1. Application Virtualization
2. Network Virtualization
3. Desktop Virtualization
4. Storage Virtualization
5. Server Virtualization
6. Data virtualization

🔑 Features of Azure Virtual Machines

1. **On-Demand Servers**
    a. Run Windows or Linux VMs in the cloud without buying hardware.
    b. Full control over OS, applications, and configurations.
2. **VM Sizes & SKUs**
    a. Wide variety of VM types (general purpose, compute-optimized, memory-optimized, GPU, high storage).
    b. You choose CPU, memory, storage, and network capacity.
3. **Scalability**
    a. **VM Scale Sets**: automatically add/remove VM instances based on demand.
    b. Manual or automatic scaling.
4. **Load Balancing**
    a. Distribute traffic across multiple VMs with **Azure Load Balancer**.
5. **High Availability & Redundancy**
    a. **Availability Sets**: protect against hardware/rack failures.

    b. **Availability Zones**: replicate across datacenter zones in a region.
6. **Storage Integration**
    a. Attach managed disks (SSD/HDD).
    b. Connect to Blob storage or file shares.
    c. Options for redundancy (LRS, ZRS, GRS).
7. **Networking & Security**
    a. Virtual Network (VNet), subnets, private/public IPs.
    b. Network Security Groups (firewalls).
    c. VPN & ExpressRoute for hybrid connectivity.
8. **Backup & Disaster Recovery**
    a. VM snapshots and backup services.
    b. Azure Site Recovery for cross-region disaster recovery.
9. **Monitoring & Management**
    a. Azure Monitor, Log Analytics, Alerts.
    b. Automation for patching and updates.
10. **Cost Flexibility**

- Pay-as-you-go, reserved instances (1–3 years), spot VMs for cheaper rates.

1. Image 🖼️

Think of an **image** as the **template** for your virtual machine.

- It defines the **operating system** (Windows, Linux, etc.) and sometimes comes with **pre-installed software**.
- Examples:
    - **Windows Server 2022**
    - **Ubuntu 22.04 LTS**
    - **Red Hat Enterprise Linux**
    - **SQL Server on Windows** (image includes both OS + SQL already installed)
    - **Custom Image** (you create your own with all your apps, configs, security patches).

👉 So: the **image decides what software environment your VM starts with**.

2. Size 💻

The **VM size** is like choosing the *hardware specs* for your cloud server. It defines **performance and cost**. Each size SKU includes:

- **Type**: category based on workload (General-purpose, Compute-optimized, Memory-optimized, Storage-optimized, GPU, High-performance compute).
- **vCPUs (Virtual CPUs)**: the number of processor cores your VM can use. More vCPUs = more processing power.
- **RAM (GiB)**: how much memory your VM has. Needed for apps/databases that require lots of memory.
- **Data Disks**: max number of additional disks you can attach (beyond the OS disk).

- **Max IOPS (Input/Output Operations per Second)**: storage performance, important for databases and heavy read/write workloads.
- **Local Storage (GiB)**: temporary storage physically attached to the host server. It's **fast but not persistent** (data can be lost if the VM moves or restarts).
- **Premium Disk Support**: tells if the VM supports **Premium SSDs** (low latency, high performance).
- **Cost/Month**: the estimated billing, depends on all the above. Higher size = higher cost.

👉 So: the **size determines how powerful your VM is and how much you pay**.

💡 Quick analogy:

- **Image = what's inside your laptop (Windows/Mac + apps).**
- **Size = the physical laptop itself (CPU, RAM, storage speed, price tag).**

1. Image (the template)

- When you pick an image (e.g., *Ubuntu 22.04 LTS*), it's like a **blueprint**.
- It tells Azure *what operating system and software to install*.

But remember: a blueprint alone can't run — you need a "hard drive" to install it on. That's where the OS disk comes in.

2. OS Disk (the VM's system drive)

- Every VM you create automatically gets **one OS disk**.
- This is where the chosen **image is deployed/installed**.
- It contains the **operating system files**, system settings, and boot loader.
- It acts just like **C: drive** on Windows or **/ (root filesystem)** on Linux.

👉 So the **image → is copied onto the OS disk** when the VM is provisioned.

3. Data Disks (extra storage drives)

- These are **optional additional disks** you can attach for storing:
  - Application data
  - Databases
  - Logs
  - Backups
- They behave like extra hard drives (F:, E: on Windows or mounted paths on Linux).
- You can attach multiple data disks depending on the VM size.

👉 The idea is to **separate OS and data**:

- If the OS disk fails or VM is re-deployed, your data stays safe on the data disks.
- You can also scale performance by adding multiple premium SSDs for databases.

💡 **Analogy:**

- **Image** = the Windows installer DVD.
- **OS Disk** = the laptop's C: drive, where Windows gets installed.
- **Data Disk** = extra F:/E: drives you plug in to store your projects, media, or databases.

When you create an Azure VM:

- You **must** have an **OS disk**, because the VM can't boot without one.
- So Azure automatically **creates and attaches** the OS disk for you — you don't have to manually choose it.

Here's what happens behind the scenes:

1. You pick an **image** (e.g., Windows Server 2022).
2. Azure automatically **creates an OS disk** (usually a managed disk, ~30–128 GB by default).
3. The **image is copied (deployed)** onto that OS disk.
4. That disk becomes the **boot drive** (C: drive or / root partition).

What the specs mean

Spec	Meaning
B1s	The VM size name — a **Burstable** (B-series) general-purpose VM. It's small, low-cost, and earns "credits" to burst CPU when needed.
1 CPU	1 virtual core (vCPU) — so it can handle one thread of processing at a time.
1 GiB RAM	Memory your applications and OS can use (1 GiB ≈ 1.07 GB).
2 data disks (max)	The VM *can attach up to 2 managed data disks*. You don't get them automatically; it's just the limit.

320 Max IOPS	"Input/Output Operations Per Second" — how fast it can read/write to attached disks. 320 IOPS is modest, fine for light workloads.
4 GiB local storage (SCSI)	Small temporary drive (called **D:** on Windows or **/mnt** on Linux). Very fast, but **data is wiped if the VM stops/deallocates**.
Premium disk Supported	You *can* use premium SSDs but not required.
US $7.59/month	Estimated cost if you run it continuously on pay-as-you-go.

- **RAM** → volatile memory used by the OS and apps while running.
- **OS disk** → persistent storage where the operating system is installed.

That 4 GiB "local storage" is **temporary**, not safe for permanent files. It's physically attached to the host machine. If you **stop/deallocate** or **move** the VM, everything on that temporary disk is **deleted**. It's meant for caching or short-term scratch data (like temp files). Data **disks**, in contrast, are **persistent managed disks** stored in Azure Storage. They survive reboots, migrations, and are backed up by Azure automatically.

✅ **Summary**

Component	Purpose	Persistent?
OS Disk (≈30 GB)	Holds the operating system	✅ Yes
Data Disk (optional)	Your app/data files	✅ Yes
Local Storage (≈4 GB)	Temporary scratch space	❌ No

let's see how these **three types of disks** show up **inside a running Azure VM**, depending on whether it's **Windows** or **Linux**.

In a Windows VM

When your Azure VM boots up, open **File Explorer** or **Disk Management**, and you'll typically see this:

Disk Type	Drive Letter	Description	Persistent?

OS Disk	**C:**	This has Windows installed.	✅ Yes
Temporary Disk (Local storage)	**D:**	This is the 4 GiB "local storage" you saw in the specs — super fast but **temporary**.	❌ No
Data Disks (if you attach them)	**E:, F:, ...**	You format and name them yourself; these are for your data, databases, or apps.	✅ Yes

💡 **Important:**
Azure even labels the D: drive with a note —

"Temporary storage – data might be lost after the VM is stopped or deallocated."

🐧 In a Linux VM

If you SSH into your Linux VM and run:

`lsblk`

you'll see something like this:

Disk Type	Typical Mount Point	Description	Persistent?
OS Disk	`/`	The root filesystem — contains Linux OS.	✅ Yes
Temporary Disk (Local storage)	`/mnt` or `/mnt/resource`	The small local storage that gets wiped when VM is stopped.	❌ No
Data Disks (optional)	`/mnt/data`, `/data`, etc.	You create and mount these manually for your files.	✅ Yes

🧠 **Quick recap of behavior:**

Action	OS Disk	Temporary Disk	Data Disk
Restart VM	✅ Retained	✅ Retained	✅ Retained
Stop (deallocate)	✅ Retained	❌ Lost	✅ Retained
Delete VM (default)	❌ Deleted unless you keep it	❌ Lost	❌ Deleted unless you keep it

🧠 RAM cache vs. Temporary disk (local storage)

Feature	RAM cache	Temporary (local) disk
Location	In-memory (inside the VM's RAM)	Physical SSD/HDD on the Azure host
Speed	Extremely fast	Slower than RAM but faster than remote storage
Persistence	Lost when VM shuts down or reboots	Lost when VM is *deallocated or moved*
Purpose	Caching very short-lived data — OS file cache, CPU cache, app cache	Paging file, swap area, temp files, intermediate data
Capacity	Limited to your VM's RAM size	Larger (few GBs to 100+ GBs depending on VM)

◇ So how do they work together?

- The **OS first uses RAM** for caching frequently accessed data — this is instant access.
- If RAM runs out, the OS starts using a **paging/swap file** on disk (which in Azure is usually placed on the temporary disk).
- This gives the illusion of having "extra memory," though it's slower.

💡 Think of it like this:

RAM = your desk space for active work.
Temporary disk = a nearby drawer for things you can't fit on the desk right now.

So yes — **cache lives in RAM**, but when RAM is full or the OS needs extra space for virtual memory, it uses the **temporary disk** as a slower extension.

When i executed command lsblk , i got

```
$ lsblk
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda       8:0    0   30G  0 disk
├─sda1    8:1    0   29G  0 part /
├─sda14   8:14   0    4M  0 part
├─sda15   8:15   0  106M  0 part /boot/efi
└─sda16 259:0    0  913M  0 part /boot
sdb       8:16   0    4G  0 disk
└─sdb1    8:17   0    4G  0 part /mnt
sr0      11:0    1  628K  0 rom
```

Perfect 👏 — that output tells us *exactly* how your Azure VM's storage is laid out. Let's read it line by line so you know what each part means:

🧩 Breaking down your `lsblk` output

Device	Size	Purpose	Mount point	Persistent?
sda	30 GB	This is your **OS disk** — where your Linux operating system is installed.	—	✅ Yes
├─ **sda1**	29 GB	Main root partition (Linux /).	/	✅ Yes
├─ **sda14**, **sda15**, **sda16**	~1 GB total	Small system partitions — EFI boot and bootloader support.	`/boot` and `/boot/efi`	✅ Yes
sdb	4 GB	The **temporary (local) disk** Azure automatically attached.	—	❌ No
└─ **sdb1**	4 GB	Single partition used for temp storage.	`/mnt`	❌ No
sr0	628 KB	Virtual CD-ROM device (not used in most VMs).	—	—

💡 In simple terms:

- **/** (from `sda1`) → your **main operating system and persistent files**.
- **`/mnt`** (from `sdb1`) → your **temporary 4 GB local storage** — fast but wiped when VM is stopped/deallocated.

🔹 What you can do with them:

- Use `/` for everything important — apps, configurations, packages, etc.
- Use `/mnt` only for temporary data, caching, or swap space.

1 Availability Set

An **Availability Set** in Azure is a logical grouping of VMs that ensures **high availability** and **resiliency**.

Purpose:

- Protects your application from **hardware failures** or **planned maintenance**.
- Guarantees that **at least one VM is available** if others are down.

How it works:
When you place VMs in an availability set, Azure distributes them across **Fault Domains** and **Update Domains** (more below).

2 Fault Domain (FD)

- Think of a **fault domain as a rack of servers in a data center**.
- VMs in the **same fault domain share the same power source and network switch**.
- If the rack or power/network fails, all VMs in that FD go down.

Key points:

- Azure automatically spreads VMs in an availability set across multiple fault domains.
- Ensures that **not all VMs fail at the same time due to hardware issues**.
- Usually, you get **2 or 3 fault domains** per availability set.

Analogy: Imagine you have 3 servers in 3 separate racks. If one rack loses power, the other racks keep running.

3 Update Domain (UD)

- Think of an **update domain as a sequence for maintenance updates**.
- Azure **rolls out planned maintenance** one update domain at a time.
- VMs in different UDs **won't be updated at the same time**, so your service remains available.

Key points:

- Default: 5 update domains (can go up to 20).
- Azure updates one UD at a time, leaving VMs in other UDs running.
- Ensures **continuous uptime during patching**.

Analogy: Imagine a team doing server updates floor by floor. Only one floor is updated at a time.

4️⃣ How it all fits together

Concept	Purpose
Availability Set	Group VMs for high availability
Fault Domain (FD)	Protects against hardware failures
Update Domain (UD)	Protects against planned maintenance downtime

Example:

- You have 4 VMs in an availability set.
- Azure places them like this:

VM	Fault Domain	Update Domain
VM1	FD1	UD1
VM2	FD2	UD2
VM3	FD1	UD2
VM4	FD2	UD1

- If FD1 fails, only VM1 and VM3 are affected.
- If maintenance happens on UD1, only VM1 and VM4 are updated at that time.

✅ Result: Your application **stays online**.

1️⃣ What is a Virtual Machine Scale Set?

A **Virtual Machine Scale Set (VMSS)** is an **Azure service that allows you to deploy and manage a group of identical VMs automatically**.

Key idea:

- You don't manually create each VM.
- Azure can **automatically scale the number of VMs up or down** based on demand (CPU, memory, or custom metrics).
- Ensures **high availability** and **elasticity** for your application.

2️⃣ Key Features

Feature	Description
Automatic Scaling	Add or remove VMs automatically based on metrics or schedule.
Load Balancing	VMSS can be integrated with Azure Load Balancer or Application Gateway.
High Availability	VMs are distributed across **fault domains** and **update domains** automatically.

Identical Configuration	All VMs are based on the same **image, OS, and settings**.
Integration with Azure Monitoring	Metrics like CPU, memory, or custom metrics can trigger scaling.

3️⃣ How VMSS is different from Availability Set

Aspect	Availability Set	VM Scale Set
VM Count	You manually create VMs	Azure can automatically scale VM count
Scaling	Manual	Automatic (up or down)
Purpose	High availability	High availability + elasticity
Load Balancer	Optional	Usually integrated to distribute traffic

4️⃣ When to Use VMSS

- Web applications with **variable traffic**
- APIs with **high scalability requirements**
- **Stateless workloads** where you can run multiple identical VMs
- Scenarios where you want **automatic scaling without manual intervention**

Example Scenario

- You have a web app that usually needs 2 VMs.
- Traffic spikes at noon, Azure can **auto-scale to 10 VMs**.
- After traffic drops, it scales back to 2 VMs automatically.

✅ **Summary:**

Virtual Machine Scale Set = Auto-scalable, highly available, and load-balanced group of identical VMs.

1️⃣ What is an IP Address in Azure VM

When you create an Azure Virtual Machine, Azure assigns it **IP addresses** so it can communicate:

- **Private IP address** → Used **inside** your virtual network (VNet).
- **Public IP address** → Used to access the VM **from the internet** (like SSH, RDP, or website).

2️⃣ Dynamic vs Static IP Address

By default, Azure assigns **Dynamic IPs** — which means:

- The IP address can **change** if the VM is **stopped/deallocated** and started again.

If you need a **fixed IP that never changes**, you use a **Static IP**.

3️⃣ Static IP Address – Definition

A **Static IP Address** in Azure means:

The IP address is reserved for your VM permanently and will **not change**, even if the VM is restarted or deallocated.

You can set a static IP for:

- **Private IP** (inside your VNet)
- **Public IP** (for internet access)

4️⃣ Why Use Static IP

Scenario	Why You Need Static IP
Web server	So DNS records or clients always connect to the same IP
Database server	Apps in the network rely on a fixed IP
Firewall or NSG rules	Easier to configure security rules
External API access	Clients can whitelist your public IP

🖥️ 1️⃣ Azure Dedicated Hosts

What it is:

A **Dedicated Host** in Azure is a **physical server** that is **fully dedicated to you** — not shared with any other customer.

Think of it as:

“My own physical machine in the Azure datacenter.”

Key Points

- You get **hardware-level isolation**.
- You can run **one or more VMs** on your dedicated host.
- You can bring your own software licenses (BYOL) for Windows Server, SQL Server, etc.
- Great for **compliance**, **security**, or **licensing** requirements.

Use Cases

- Companies with **strict compliance** (banking, healthcare).
- When you need **visibility into the physical hardware** your VMs run on.
- When you want to **reuse existing on-prem licenses**.

Example:

You host 5 Windows Server VMs on a single **dedicated host**. Only **your** organization uses that hardware — no other Azure customers share it.

⚡ 2️⃣ Azure Spot Instances

What it is:

Azure **Spot VMs** let you use **unused capacity** in Azure's datacenters at **huge discounts** (up to 90% cheaper).
But they can be **evicted (stopped or deleted)** anytime Azure needs that capacity back.

Think of it as:

"Cheaper VMs that can be taken away anytime."

Key Points

- Same performance as normal VMs, but **no availability guarantee**.
- You **set a max price** you're willing to pay.
- If demand increases and Azure needs resources, your Spot VM may be **evicted**.
- You can choose **to stop or delete** when eviction happens.

Use Cases

- **Batch jobs**
- **Rendering**, **testing**, **CI/CD pipelines**
- **Non-critical workloads** that can handle interruptions.

Example:

You run 100 Spot VMs overnight to process data.
If Azure needs the capacity back, some VMs may be evicted — you lose compute temporarily but save huge costs.

💰 3️⃣ Reserved Compute Instances (Reserved VM Instances)

What it is:

Azure **Reserved Instances (RIs)** allow you to **pre-pay** or **commit** to a VM type in a specific region for **1 or 3 years** — and get up to **72% cost savings** compared to pay-as-you-go.

Think of it as:

"I'll use this VM for the next 3 years — give me a big discount."

Key Points

- You reserve a specific **VM size and region**.
- You still pay monthly or upfront, but at a much lower rate.
- Works well for **predictable workloads**.
- You can **exchange or cancel** reservations if needed (with limits).

Use Cases

- Long-running servers (like web servers, databases).
- Always-on enterprise workloads.
- Cost optimization for production systems.

Example:

You reserve a D4s_v3 VM for 3 years in Central India.
Azure gives you ~60–70% discount over pay-as-you-go rates.

🩹 Resilience (in Cloud / Azure context)

Definition:

The **ability of a system to continue operating acceptably — even when one or more components fail.**

In simpler terms:
A **resilient system** doesn't go down when something breaks. It can **recover**, **self-heal**, or **reroute** operations to stay available.

Examples in Azure

Example	Description
Availability Set / Zone	If one datacenter or rack fails, your VM in another zone/domain keeps running.
Load Balancer	If one VM goes down, traffic is automatically sent to a healthy VM.
Auto-scaling	If a VM crashes or demand spikes, Azure automatically adds new VMs.
Geo-redundant Storage (GRS)	If one region fails, data is still accessible from another region.

What are Containers?

A **container** is a standardized, executable package of software that bundles up an application's code with all its dependencies (like specific libraries and configuration files). This ensures the application runs quickly and reliably the same way, regardless of the computing environment it's deployed on.

Virtual Machines vs Containers — Key Difference

Feature	Virtual Machine (VM)	Container
What it virtualizes	The **hardware**	The **operating system**
OS per instance	Each VM runs its **own OS kernel**	All containers **share the host OS kernel**
Startup time	Minutes (boot entire OS)	Seconds (just start app process)
Resource usage	Heavier (GBs of memory and disk)	Lightweight (MBs)
Isolation	Strong (full OS isolation)	Process-level isolation
Use case	Running multiple OSes or legacy apps	Running many lightweight app instances on same host

1. One Container ≠ Many Apps (Usually)

- A **container** is designed to **package and run a single application or service** — for example, a web server (like Nginx) or an API service.
- The idea is **simplicity and isolation**: each container should do **one thing well**.

- You can run **multiple containers**, each with its **own app**, on one host machine.
- Each container typically runs a single application or service. A single host can run multiple containers, each isolated from the others.

✅ Example:

- `container-1`: runs your **frontend (React)**
- `container-2`: runs your **backend API (Node.js)**
- `container-3`: runs your **database (MongoDB)**

All these containers can run on the **same host machine**, but **each one** runs **only one app/process**.

🧩 2. You *Can* Run Multiple Processes in One Container (but discouraged)

Technically, yes — you *can* run multiple apps in one container (for example, using process managers like `supervisord`), but:

⚠️ **It's not a best practice**, because:

- It breaks isolation (harder to scale or update one part).
- Containers become heavier and harder to maintain.
- Restarting one app restarts the whole container.

🐳 Docker — The Most Popular Container Engine

- **Docker** provides the tooling to create, run, and manage containers.
- You define everything (dependencies, environment, app) inside a **Dockerfile**, which builds into a **Docker image**.
- That image runs as a **container instance**.
- **Docker** is the **technology and platform** used to build, run, and manage containers.

Example:

Let's say you have a Node.js app. You write a file called `Dockerfile` that defines:

- Which base image to use (e.g., `node:18`)
- What code to copy
- What command to run

Then you build it:

```
docker build -t myapp .
```

Now you can run it anywhere:

```
docker run myapp
```

✅ **Docker helps you create the container image.**
✅ **Azure runs it in the cloud (ACI, AKS, etc.).**

☸️ 3️⃣ Azure Kubernetes Service (AKS)

Kubernetes is a **container orchestration system** — it manages *many* containers automatically.

Azure Kubernetes Service (AKS) is Azure's **managed Kubernetes offering**, meaning:

- Azure handles the control plane, upgrades, and scaling.
- You focus on deploying and managing your containerized applications.

AKS handles:

- **Scaling** (adds/removes containers based on load)
- **Load balancing**
- **Automatic restarts** if a container crashes
- **Rolling updates** for new versions
- **Networking and secrets management**

In short:

AKS = Tool to run *lots of Docker containers* in a reliable, scalable way.

⚙️ 4️⃣ How They Are Connected

Concept	Role
Docker	Builds and runs individual containers
Container	A lightweight package containing your app
Azure Containers / ACI	Runs a few containers easily, no orchestration
Kubernetes / AKS	Runs and manages *many* containers in production
Azure	Cloud platform providing the infrastructure and management services

Visual Flow:

```
Code → Docker → Container Image → Push to Azure Container Registry (ACR) →
```

Run on ACI or manage at scale with AKS

🧩 Example Flow

1. You **build** a Docker image locally or in CI/CD.
2. You **push** it to **Azure Container Registry (ACR)**.
3. You **deploy** that image to:
    a. **Azure Container Instances (ACI)** → simple, single container
    b. **Azure Kubernetes Service (AKS)** → large-scale, managed cluster

☑️ Summary

Concept	Purpose	Managed by
Docker	Tool to build and run containers	You
Container	Packaged app + dependencies	Runs anywhere
Azure Container Instances (ACI)	Run containers quickly, serverless style	Azure
Azure Kubernetes Service (AKS)	Manage and scale container clusters	Azure
Azure Container Registry (ACR)	Store and manage container images	Azure

***why* containers have become so popular even though Azure App Service already makes deployment easy.**

Let's break it down clearly 👇

🧩 App Service (Traditional Deployment)

- You deploy your **web app code** (e.g., Node.js, .NET, Python) directly to Azure.
- Azure **manages the runtime and environment** for you (OS, framework version, scaling, etc.).
- Great for **simple web apps or APIs**.
- Example: You push your code via GitHub Actions or Zip Deploy → App Service runs it.

🟢 **Pros:**

- Super easy to use (no infrastructure management).
- Built-in scaling, load balancing, and SSL.
- Ideal for straightforward web apps.

🔴 **Cons:**

- Limited control over OS and dependencies.
- If your app needs special libraries, custom runtimes, or multiple services—it can be restrictive.

🐳 Containers (Docker, Azure Container Instances, AKS)

- You **package the entire application + dependencies + OS environment** into a *container image*.
- This image runs identically on any environment (local, dev, test, production).
- You can host these containers on **Azure Container Instances, App Service for Containers, or AKS (Kubernetes)**.

🟢 **Pros:**

- Full control over runtime, dependencies, and environment.
- "Works on my machine" problems disappear.
- Portable between cloud providers and on-prem.
- Scales better for **microservices** or complex architectures.

🔴 **Cons:**

- Slightly more setup and maintenance overhead.
- You need to handle container builds, registry, etc.

🧠 How They Work Together

Actually, **App Service supports containers too** — you can deploy a Docker image *to* App Service. So, you can get the **simplicity of App Service** + **flexibility of containers** at the same time.

🧩 What "OS environment" means in a container

When we say:

"You package the application + dependencies + OS environment into a container image,"

we mean that you include **only the parts of the operating system your app needs to run**, *not an entire OS*.

🧠 Think of it like this:

A **Virtual Machine (VM)** includes:

- Full operating system kernel (e.g., Windows, Linux)
- System libraries
- App dependencies
- Your application

A **Container** includes only:

- Your **application**
- The **libraries and binaries** it depends on
- A **base OS layer** (like `ubuntu`, `alpine`, or `node:18`) — just enough to support the app

👉 It **shares the host OS kernel** with other containers instead of installing its own.

📦 Example

Here's a simple `Dockerfile`:

```
FROM node:18-alpine   # Base OS environment (Alpine Linux + Node runtime)
WORKDIR                                                          /app
COPY                                    .                          .
RUN                                   npm                      install
CMD                                 ["npm",                   "start"]
```

Explanation:

- `node:18-alpine` → This is your **OS environment** (a tiny Linux image with Node.js preinstalled).
- It gives you all the basic system tools and libraries that Node.js needs to run.
- Your **application** and its dependencies get added on top.

So when you run this container, it behaves like a tiny Linux machine with Node installed — but it's lightweight and isolated.

⚙️ In short:

Component	Description
Application	Your actual code (e.g., Node.js app, Python script)
Dependencies	Libraries your app needs (npm modules, pip packages, etc.)

OS Environment	Minimal Linux layer (like Ubuntu or Alpine) + system utilities required by your app

🧩 Summary

OS environment in a container = A lightweight base image that provides the minimal operating system libraries and tools your app needs — but **not the entire OS kernel**, since that's shared with the host.

🧠 What is a Runtime?

A **runtime** is the **environment** in which your application executes. It includes the **language engine, libraries, and configurations** your app needs to run.

Common examples of runtimes:

- Node.js
- .NET Core / .NET 8
- Python
- Java
- PHP
- Go

So when you deploy a web app to **Azure App Service**, Azure provides and manages these runtimes for you.

🧩 What is a Custom Runtime?

A **custom runtime** means:

You need a runtime that **isn't officially supported** by the hosting platform — or you want to **customize** the runtime's behavior, version, or dependencies.

🧱 Examples of Custom Runtime Use Cases

Scenario	Explanation
🧬 **Unsupported language**	Suppose your app is written in **Rust** or **Elixir**, which Azure App Service doesn't natively support. You'll need to bring your own runtime.

⚙️ **Custom version or build**	You need **Node.js 23.5** with special flags, but Azure only supports up to Node.js 22.
🧩 **Custom frameworks**	You use a framework that depends on certain OS libraries or binaries not installed in the default Azure image.
🔐 **System-level dependencies**	Your app needs a native library like `libvips` or `ffmpeg`, which can't be installed in standard App Service runtime.

🚀 How Containers Help

When you use **containers**, you can:

- Build your own runtime environment exactly as you want it.
- Choose your base OS (Ubuntu, Alpine, Debian, etc.).
- Install your own libraries, tools, and runtime versions.

Example:

```
FROM ubuntu:22.04
RUN apt-get update && apt-get install -y python3.12 ffmpeg
COPY app/ /app
CMD ["python3", "/app/main.py"]
```

This Docker container has:

- Ubuntu 22.04 as OS environment
- Python 3.12 (custom runtime)
- FFmpeg (custom dependency)

You can then deploy this container to **Azure App Service for Containers** or **AKS** — and it will run exactly as built.

💡 In short:

A **custom runtime** means bringing your **own execution environment** — when the cloud's built-in options don't meet your needs.

🐳 **Containers** let you do this easily, because you control everything inside the image.

Containers on Azure refers to a family of services (**Container Apps, AKS, ACI**) that give you increasing levels of control over the container runtime and orchestration, which is ideal for microservices and specialized workloads.

Here is a breakdown of the differences, comparing App Service with the container-focused options:

1. Azure App Service (PaaS for Web Apps) 🌐

Characteristic	Azure App Service
Primary Goal	**Simplest way to host web apps/APIs.**
Abstraction	**High Abstraction (Fully Managed).** You only manage your application code. Azure manages the OS, web server (IIS or Kestrel), patching, and container hosting (if you use containers).
Deployment Model	Deploy **code** (e.g., a .NET or Node.js project) *or* a **single container** (Web App for Containers).
Scaling	**Rule-based autoscaling** (based on metrics like CPU, memory). Does **not** scale to zero; an instance must always be running.
Best For	Traditional **monolithic** web applications, internal/external APIs, or simple web apps where **ease of use and fast deployment** are the priority.
Microservices	Suitable for **simple** microservice scenarios, but not optimized for complex orchestration.

2. Container-Specific Services (Flexibility and Microservices) 📦

When people say "containers on Azure," they are usually referring to one of these three services, each offering more control than App Service:

A. Azure Container Apps (ACA) - *The Microservices PaaS* ✨

- **Goal:** Running **serverless microservices** and event-driven applications built on Kubernetes, without the complexity of managing Kubernetes itself.
- **Abstraction:** Medium abstraction. It's a managed Kubernetes environment where **Azure handles the cluster**. You get features like built-in service discovery, traffic splitting, and **Dapr** integration for microservices.
- **Scaling: Event-Driven Autoscaling (KEDA)**. It can **scale to zero** instances, making it highly cost-efficient for applications with bursty traffic.
- **Best For:** Modern microservices architectures, event-driven processing, and applications that need to scale dramatically up and down to zero.

B. Azure Kubernetes Service (AKS) - *Full Orchestration Control* ☸️

- **Goal:** Full, enterprise-grade container **orchestration** using the industry-standard Kubernetes platform.
- **Abstraction: Low Abstraction (High Control).** Azure manages the Kubernetes **control plane**, but **you manage the worker nodes** (updates, scaling, configuration) and the entire container workload.
- **Scaling:** Full, customizable Kubernetes scaling (HPA, VPA, KEDA).

- **Best For:** Complex, large-scale microservice deployments, custom networking, hybrid deployments, or teams with **Kubernetes expertise** that require absolute control and flexibility.

C. Azure Container Instances (ACI) - *Quick and Simple* ⚡

- **Goal:** Quickly running **individual containers** on-demand without any cluster management overhead.
- **Abstraction:** High abstraction, but **no orchestration**. It's just a single, isolated container or a small, related container group.
- **Scaling: None/Manual**. You deploy a single instance. To scale, you must manually launch more ACI instances and manage load balancing yourself.
- **Best For:** Short-lived tasks, batch processing, CI/CD agents, and simple proof-of-concept deployments.

Summary: How to Choose

Scenario	Recommended Azure Service	Rationale
Traditional Web App	**Azure App Service**	Easiest setup, fully managed, built-in features for HTTP traffic.
Serverless Microservices	**Azure Container Apps (ACA)**	Optimized for microservices, scales to zero, provides orchestration without complexity.
Short-Lived Job/Task	**Azure Container Instances (ACI)**	Fastest way to run a single container for a quick, one-off task.
Complex Microservices	**Azure Kubernetes Service (AKS)**	Requires Kubernetes expertise but provides the most flexibility, customizability, and control for large-scale, complex environments.

What is Container Orchestration?

When you have **one or two containers**, you can manage them manually — start, stop, update, or monitor them.

But in real-world applications, you might have:

- **Hundreds** of containers
- Spread across **multiple servers**
- Handling **auto-scaling**, **load balancing**, **updates**, **failures**, and **networking**

👉 Managing all that manually is impossible.

That's where **container orchestration** comes in.

Container orchestration = automated management of containerized applications — including deployment, scaling, networking, and health monitoring.

⚙️ What Container Orchestration Does

Function	Description
🧱 **Deployment & Scheduling**	Decides where and when to start containers across multiple machines.
🔁 **Scaling**	Automatically adds or removes containers based on demand (CPU, traffic, etc.).
🔄 **Self-healing**	Restarts containers that fail, reschedules them if a node goes down.
🌐 **Networking**	Manages communication between containers and exposes apps to the internet.
🔐 **Configuration & Secrets**	Securely manages environment variables, secrets, and configs.
🚀 **Rolling Updates**	Deploys new versions of your app with zero downtime.

☁️ Container Orchestration in Azure

Azure provides multiple options, but the main one is **Azure Kubernetes Service (AKS)**.

🛰️ Azure Kubernetes Service (AKS)

Kubernetes is the most popular open-source container orchestration platform. Azure offers it as a **managed service**, so you don't have to install or maintain Kubernetes yourself.

AKS handles:

- Container scheduling and scaling
- Load balancing between containers
- Monitoring and logging
- Rolling updates and rollbacks
- High availability and fault tolerance

You only manage:

- Your **container images**

- Your **Kubernetes configurations (YAML files)**

How AKS Fits in Azure

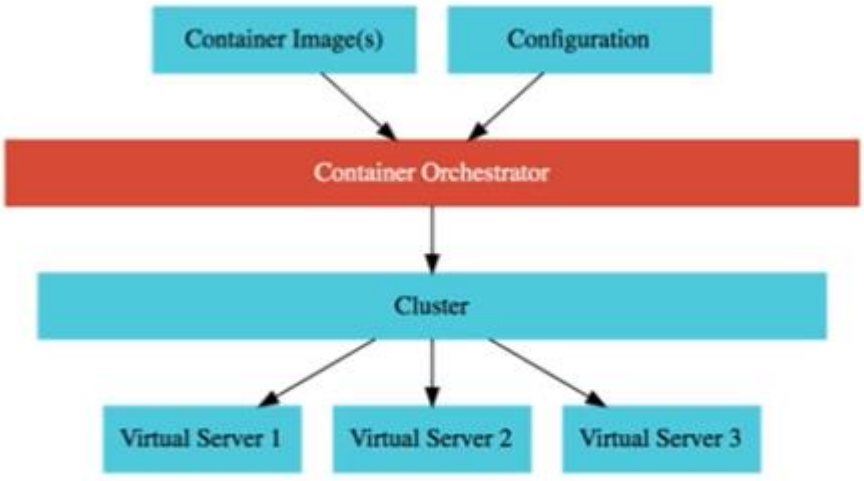
1. You build a **Docker image** of your app.
2. Push it to **Azure Container Registry (ACR)**.
3. Create a **Kubernetes deployment** in AKS that pulls and runs those images.
4. AKS automatically:
    a. Runs containers on multiple VMs (nodes)
    b. Balances traffic
    c. Restarts failed containers
    d. Scales up/down as needed

Other Azure Orchestration Options (lighter use cases)

Service	Use Case
Azure Container Instances (ACI)	Simple, single-container workloads — no orchestration needed.
App Service for Containers	Deploy a containerized web app easily — Azure manages it for you.
AKS (Kubernetes)	Full orchestration — ideal for production-scale microservices.

In short:

Container orchestration in Azure = automated management of containers for scaling, health, and networking — handled by **Azure Kubernetes Service (AKS)**.



⚙️ Diagram Explanation

1️⃣ Container Image(s) & Configuration

- A **container image** is your packaged application (e.g., a Docker image).
- A **configuration** defines:
  - How many replicas to run
  - Resource limits (CPU/RAM)
  - Networking rules
  - Environment variables

These are the inputs the orchestrator uses to know *what* and *how* to deploy.

2️⃣ Container Orchestrator (the brain)

This is the **central controller** — the software that automates container management across multiple servers.
Examples:

- **Kubernetes** (used in **Azure Kubernetes Service**)
- **Docker Swarm**
- **OpenShift**

It takes your **images and configuration** and decides:

- Where each container should run
- How to balance the load
- How to restart failed containers
- When to scale up or down

3️⃣ Cluster

- A **cluster** is a group of **virtual servers (nodes)** that work together to run containers.
- The orchestrator manages the cluster and decides which container runs on which node.

In the diagram:

```
Cluster
 ├──                Virtual                Server                1
 ├──                Virtual                Server                2
 └──                Virtual                Server                3
```

Each **virtual server** (or **node**) runs multiple containers as assigned by the orchestrator.

🚀 Typical Features (on the left side of the slide)

Feature	What it means
Auto Scaling	Automatically increases or decreases the number of container instances based on CPU usage, memory, or traffic.
Service Discovery	Helps containers (like microservices A & B) find and talk to each other inside the cluster without knowing IP addresses.
Load Balancer	Distributes incoming traffic evenly among multiple container instances to prevent overload.
Self Healing	Automatically replaces or restarts failed containers to keep services running.
Zero Downtime Deployments	Updates your app to a new version without shutting it down — users don't experience downtime.

🧠 Example in Azure

In **Azure**, this orchestration is done by **Azure Kubernetes Service (AKS)**:

- You push your **container images** to **Azure Container Registry (ACR)**.
- AKS (the **container orchestrator**) uses those images + configuration files (YAML manifests).
- AKS deploys containers across a **cluster of virtual machines (nodes)**.
- AKS handles **auto-scaling, load balancing, service discovery, and self-healing** automatically.

🧩 1️⃣ What is a Virtual Server (Node)?

- A **virtual server** (also called a **node**) is just a **virtual machine (VM)**.
- It's a **computer** provided by Azure (or any cloud), with its own CPU, memory, and OS.
- On that VM, you can run **containers**.
- Think of it as a *host machine* where containers actually live.

Example:
You create 3 VMs in Azure — those become **Virtual Server 1, 2, and 3** (as shown in your image).

🧩 2️⃣ What is a Cluster?

- A **cluster** is a **group of virtual servers (nodes)** managed together.
- The cluster acts like **one big computer** to the outside world.
- The **container orchestrator** (like Kubernetes / AKS) controls the whole cluster.

Why a cluster?

- You rarely run all containers on one VM — it's risky.
- Instead, you distribute them across multiple VMs (nodes) for:
    - **High availability** (if one node fails, others still work)
    - **Load balancing** (traffic is spread)
    - **Scalability** (you can add more nodes)

🧩 3️⃣ What is a Container?

- A **container** is a lightweight, isolated environment that runs your **application code**.
- Each container runs inside a node (VM).
- The orchestrator places containers across nodes depending on resources and load.

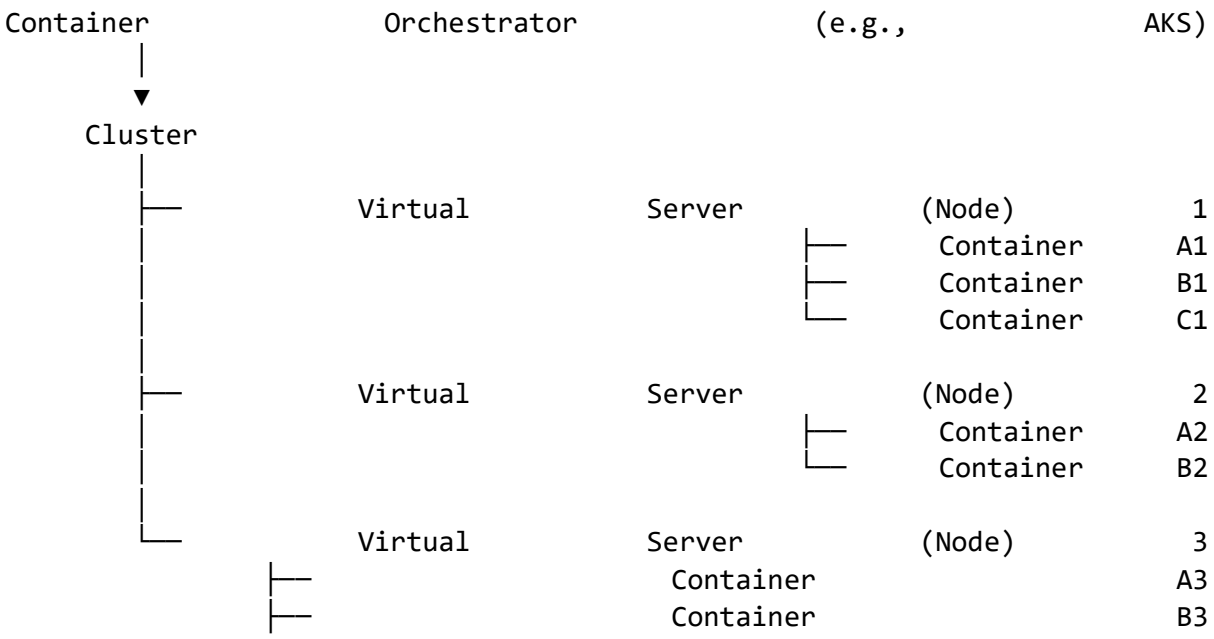
Example:
You have 15 containers total:

- Node 1 runs 5 containers
- Node 2 runs 5 containers
- Node 3 runs 5 containers

If Node 2 fails, the orchestrator moves its containers to Node 1 and Node 3 automatically — that's **self-healing**.

🧠 4️⃣ How They Are Related

Here's the hierarchy 👇



- **Containers** run your apps.
- **Virtual servers (nodes)** host those containers.
- **Cluster** groups multiple nodes together.
- **Orchestrator** (like Kubernetes) manages everything — scheduling, scaling, healing, etc.

◇ In Azure:

- Azure **creates and manages the nodes (VMs)**.
- Azure **Kubernetes Service (AKS)** acts as the **orchestrator**.
- Your **containers** (from Azure Container Registry) run on those nodes inside the **AKS cluster**.

🐳 1️⃣ Azure Container Instances (ACI) — *"Run one or few containers quickly"*

- **Purpose** → Run a single or small number of containers *without managing servers*.
- **Type** → PaaS (Platform as a Service)
- **Management effort** → ✅ Minimal — you just provide the container image.
- **Orchestration** → ❌ No orchestration (no auto-scaling, no load balancing, no service discovery).
- **When to use** →
  - Quick test or demo
  - Background jobs / scripts
  - Event-driven short tasks (e.g., image processing, batch jobs)
  - Lightweight APIs or microservices without complex scaling needs

🧠 *Think of it like*: "Run this container for me and stop when it's done."

🚀 2️⃣ Azure Container Apps (ACA) — *"Smart auto-scaling containers without managing Kubernetes"*

- **Purpose** → Build microservice or event-driven apps that automatically scale.
- **Type** → PaaS (Managed serverless container platform)
- **Management effort** → Moderate — Azure handles infrastructure; you focus on app logic.
- **Features** →
  - Built-in **scaling** (via KEDA: event-driven auto-scaler)
  - **Load balancing**
  - **Revisions and zero-downtime deploys**
  - Built-in **ingress/egress**, **Dapr** support for microservices communication
- **When to use** →

- Microservices architecture
- API-based apps
- Event-driven or background workers
- You want Kubernetes-like benefits, but without managing clusters

🧠 *Think of it like*: "Kubernetes power, but serverless simplicity."

🧩 3️⃣ Azure Kubernetes Service (AKS) — *"Full-scale container orchestration"*

- **Purpose** → Manage and orchestrate *large fleets* of containers across many nodes.
- **Type** → Managed Kubernetes (IaaS + orchestration)
- **Management effort** → High — you manage clusters, nodes, and networking.
- **Features** →
  - Full Kubernetes ecosystem
  - Advanced orchestration (auto-scaling, load balancing, self-healing)
  - Complex networking, service meshes, CI/CD
  - Works best for enterprise-grade microservices
- **When to use** →
  - Large-scale apps
  - Multi-container, multi-environment deployments
  - Custom control over deployment, networking, or security
  - Teams with DevOps / Kubernetes expertise

🧠 *Think of it like*: "I want full control of my containerized environment."

⚖️ Comparison Table

Feature	ACI	Container Apps	AKS
Type	PaaS	PaaS (Serverless Kubernetes)	Managed Kubernetes
Scaling	Manual	Auto (event-based)	Manual or auto (K8s-based)
Load Balancing	❌	✅	✅
Service Discovery	❌	✅	✅
State Management	Basic	Dapr support	Kubernetes Persistent Volumes
Ideal For	Simple jobs, testing	Microservices / APIs	Large-scale production apps
Management Effort	Very low	Medium	High
Cost	Low	Moderate	Higher (depends on cluster size)

What is a Server?

A **server** is basically a **computer** that provides **services, resources, or data** to other computers (clients) over a network.

In the cloud (like Azure), a **server** isn't a physical box you can touch — it's a **virtualized machine** running on Azure's massive datacenters.

Servers in Azure are of Two Main Types:

1 Physical Servers (Azure manages these)

- These are **actual computers** (hardware) in Microsoft's data centers all over the world.
- You never access or configure these directly.
- They're used to **host** your virtual resources.

Think of them as **the foundation** — the metal boxes with CPU, RAM, storage, and network cards.

2 Virtual Servers (You manage these)

- A **Virtual Machine (VM)** in Azure is your **virtual server**.
- It acts like a full computer with its own OS (Windows/Linux), IP address, storage, etc.
- You can install software, host apps, or run workloads on it.

Example:
When you create a VM in Azure (say Ubuntu Server or Windows Server), Azure runs it **on top of** one of those physical servers in the data center.

So your "Azure VM" = your **own virtual server** hosted in Microsoft's cloud

Server Hierarchy in Azure

Layer	Who Manages It	Description
Physical Hardware (Data Centers)	Azure	Real servers with CPUs, RAM, disks
Hypervisor Layer (Virtualization)	Azure	Divides physical servers into multiple virtual servers (VMs)
Virtual Machine (Your Server)	You	A virtualized computer you can log into
Applications	You	Software, APIs, or websites running on the VM

☁️ Types of Servers You Can Create in Azure

Type	Example Service	What It Provides
🖥️ **Virtual Machine (Compute)**	Azure Virtual Machines	Full OS-level control
🗄️ **Database Server**	Azure SQL Database, MySQL, PostgreSQL	Database engine hosting
🐳 **Container Server**	Azure Container Instances, AKS	Runs Docker containers
⚙️ **App Server (Serverless)**	Azure App Service, Functions	Runs your app without you managing the VM
🧱 **File/Storage Server**	Azure Storage, Azure Files	Stores files or data
🌐 **Web Server**	Azure App Service, Nginx on VM	Hosts websites or APIs

🧠 1️⃣ Meaning of Serverless

Despite the name, **"serverless" does not mean there are no servers**. It means:

You **don't manage the servers** — Azure manages them **automatically**.

You focus on your **application code** or **container**, and Azure automatically handles:

- Server provisioning (creating/allocating compute)
- Scaling up/down based on demand
- Fault tolerance and patching
- Billing only for what you use

⚙️ 2️⃣ Traditional vs. Serverless in Azure

Feature	Traditional VM/App Service	Serverless
Server setup	You choose OS, size, region	Azure handles all that
Scaling	Manual or preconfigured	Automatic (even scale to 0)
Billing	Pay for uptime	Pay only when code runs
Maintenance	You patch/update OS	Azure handles everything
Control	Full control of environment	Focus only on app logic

🧠 In short:

Serverless = no infrastructure management + event-driven + pay-per-use

💡 3️⃣ How Serverless Works (Simplified Flow)

1. An **event** happens — for example:
    a. A user uploads a file
    b. A message arrives in a queue
    c. An HTTP request hits your API
2. Azure automatically allocates compute power
3. Your **function/code/container** runs
4. Once finished, resources are released

You don't keep a server running 24/7 — Azure runs it only when needed.

🔷 4️⃣ Serverless Services in Azure

Category	Service	Description
🧠 **Compute**	**Azure Functions**	Run small code snippets in response to events (like a mini program)
🧩 **Containers**	**Azure Container Apps**	Run containers with auto-scaling and no Kubernetes management
🔁 **Workflows**	**Azure Logic Apps**	Automate workflows (connect apps/services visually)
🗃️ **Databases**	**Azure Cosmos DB (Serverless mode)**	Pay only for queries made
📣 **Messaging / Events**	**Event Grid, Event Hubs, Service Bus**	Trigger serverless apps based on events
🌐 **APIs**	**API Management (Consumption tier)**	Serverless API gateway

🚀 5️⃣ Example: Serverless Image Processing App

Imagine you build an image-processing system.

Step	Service	Role
1️⃣ User uploads an image	**Azure Blob Storage**	Stores image
2️⃣ Event triggers	**Event Grid**	Detects new upload
3️⃣ Function runs	**Azure Function**	Resizes image
4️⃣ Result stored	**Azure Storage**	Saves processed image

→ No servers, no scaling logic — all automatic. You only pay for the few seconds the function executes.

🧱 6️⃣ Advantages of Serverless in Azure

✅ **No infrastructure management** — focus on logic, not servers
✅ **Auto scaling** — scales up and down instantly
✅ **Cost efficiency** — pay only per execution
✅ **High availability** — Azure ensures uptime
✅ **Event-driven** — great for reactive systems

⚠️ 7️⃣ When NOT to Use Serverless

Scenario	Why Not Ideal
Long-running processes	Functions have time limits
Heavy CPU-bound apps	Costs can get high
Complex dependencies or OS-level control	Serverless doesn't let you customize OS deeply

In those cases → use **VMs**, **Containers**, or **AKS**.

🧩 8️⃣ In Simple Words

Serverless in Azure means you can run your **apps, code, or containers** without worrying about servers, scaling, or maintenance.

You just say **"run this when X happens"**, and Azure takes care of the rest.

⚙️ 1️⃣ What is Azure Functions?

Azure Functions is a **serverless compute service** that lets you run **small pieces of code** (called *functions*) **without managing any servers**.

You just write your code → Azure runs it automatically **when triggered by an event**.

🧠 2️⃣ The Idea

Normally, if you want to run code, you'd need to:

- Set up a server
- Deploy your app

- Keep it running all the time

With **Azure Functions**, you only write the function logic — Azure automatically:

- Starts it when needed
- Allocates compute resources
- Scales up/down
- Stops it when finished

And you pay **only for the execution time** (milliseconds).

🔔 3️⃣ Triggers (How Functions Run)

Functions are **event-driven** — they run when a specific *trigger* happens.

Common types of triggers:

Trigger Type	Example Event	Typical Use
HTTP Trigger	HTTP request (like an API call)	Create serverless APIs
Timer Trigger	Runs on schedule	Cron jobs, daily cleanups
Blob Trigger	File uploaded to Azure Blob Storage	Image processing, file validation
Queue Trigger	New message in a queue	Process background jobs
Event Grid / Service Bus Trigger	System event	React to Azure events

🧠 Example:
If you create a *Blob Triggered Function*, Azure automatically runs it whenever someone uploads a file to a specific Blob Storage container.

🧩 4️⃣ Bindings (Input / Output Connections)

Azure Functions can automatically connect to other Azure services — these are called **bindings**.

Example:

A function triggered by a new file upload (input binding) and writes data to a database (output binding).

```
def main(myblob: func.InputStream, output: func.Out[bytes]):
    data = myblob.read()
    output.set(data.upper())
```

Here:

- Input: Blob Storage file
- Output: Another Blob or Database

Bindings save you from writing complex connection code.

🧱 5️⃣ Hosting Options

Azure provides three ways to run Functions:

Plan	Description	Billing
Consumption Plan (default)	Fully serverless, scales automatically	Pay per execution (best for most cases)
Premium Plan	Pre-warmed instances for faster startup	Pay per instance running
Dedicated (App Service Plan)	Runs on dedicated VMs	Pay for uptime

🧰 6️⃣ Supported Languages

You can write Azure Functions in:

- C#
- JavaScript / TypeScript
- Python
- Java
- PowerShell
- Go (via custom handlers)

🌐 7️⃣ Example: HTTP Triggered Function

```
module.exports = async function (context, req) {
    const name = req.query.name || (req.body && req.body.name);
    context.res = {
        status: 200,
        body: `Hello ${name || "World"}!`
    };
};
```

📦 Deploy this to Azure
🌐 Call it at https://<your-function-app>.azurewebsites.net/api/hello?name=Anand
💬 Response → "Hello Anand!"

🚀 8️⃣ Real-World Use Cases

Use Case	Example
🧾 **Serverless API**	Build backend APIs that scale automatically
📦 **Event Processing**	Run code when a message/file/event arrives
⏰ **Scheduled Jobs**	Run cleanup or report generation daily
🔁 **Automation**	Move or transform data between services
💬 **Notification Services**	Send emails, push notifications, or alerts

💡 9️⃣ Key Benefits

✅ No infrastructure management
✅ Scales automatically
✅ Pay-per-execution
✅ Easy integration with Azure services
✅ Fast to develop and deploy

⚠️ 1️⃣0️⃣ Limitations

Limitation	Description
Cold start	Functions may take a moment to start if inactive
Short execution time	Default max ~5–10 minutes (configurable)
Stateless	Each run is isolated (no in-memory persistence)
Limited environment control	Can't install OS-level packages (use Containers for that)

🧩 1️⃣1️⃣ Relationship to Other Azure Compute Models

Model	You manage servers?	Scales automatically?	Typical use
VM	✅ Yes	Manual	Full control
App Service	❌ Partial	✅	Web apps

Container Apps	❌ No	✅	Microservices
Azure Functions	❌ No	✅	Event-driven, small tasks

◇ In Summary

- 🔸 **Azure Functions** = Small pieces of code that run in response to events.
- 🔸 **Serverless** = You don't manage infrastructure.
- 🔸 **Pay only for what runs**, and scale automatically.

🧠 Think of it like:

"A lightweight function that wakes up, does its job, and sleeps — and you only pay for that moment."

⚡ Azure Functions – Explained

◇ What is Azure Functions?

Azure Functions is a **serverless compute service** — meaning you focus only on your **code logic**, while Azure automatically handles:

- Server provisioning
- Scaling
- Patching
- Infrastructure management

You simply write small pieces of code called **functions**, and they run **on demand** in response to **events or triggers**.

🧠 When to Use Azure Functions

Use Azure Functions when:

- You want to run code **only when triggered** by an event (like HTTP request, message, file upload, timer, etc.)
- You don't want to manage servers or runtime environments.
- The work can be completed quickly (seconds or less).
- Demand can **fluctuate** — sometimes high, sometimes none.

💡 **Example Scenarios:**

- Automatically resize images when uploaded to Azure Blob Storage.
- Send email notifications when a new record is added to a database.

- Run cleanup or scheduled tasks every midnight.

🚀 Key Benefits of Azure Functions

1️⃣ No Infrastructure Management

- You don't worry about VMs, scaling, or patching — Azure handles it all.
- Just write the function and define the trigger.

2️⃣ Event-Driven Execution

- Functions respond to **events** such as:
  - HTTP requests (API calls)
  - Queue messages (from Azure Queue or Service Bus)
  - File uploads (to Blob Storage)
  - Timers (scheduled jobs)
- You can integrate Functions with many Azure services.

3️⃣ Automatic Scaling

- Azure automatically scales **up or down** based on demand.
- If 1,000 events happen at once, Azure can spin up multiple instances of your function automatically.
- When demand drops, instances shut down — saving cost.

4️⃣ Pay-Per-Use (Consumption Model)

- You pay **only for the compute time** your code actually runs.
- If your function doesn't run → **cost = ₹0**.
- Great for workloads that are **sporadic or unpredictable**.

💡 *You're charged only for CPU and memory used during execution.*

5️⃣ Stateless and Stateful Options

- **Stateless (default):**
  - Each function runs independently with no memory of past runs.

    - Ideal for quick, one-time operations.
- **Stateful (Durable Functions):**
    - Keeps track of previous runs using a **context object**.
    - Useful for workflows or chained processes (e.g., order → payment → shipment).

6️⃣ Flexible Deployment

- You can start serverless, then move to a more controlled environment (like App Service or Kubernetes) if you need:
    - Advanced networking (VNETs)
    - Custom scaling
    - Isolation or compliance controls

⚙️ Quick Example Flow

Let's say you have a **function** that runs every time a file is uploaded to Blob Storage:

1️⃣ File uploaded → Blob trigger activates
2️⃣ Azure spins up your function automatically
3️⃣ Function resizes the image
4️⃣ Function completes → resources are released
5️⃣ You only pay for those few seconds of processing

☁️ What is Azure App Service?

Azure App Service is a **fully managed platform** for building, deploying, and scaling **web apps, REST APIs, mobile back ends**, and **background jobs** — **without managing servers or infrastructure**.

Think of it as **"Web Hosting + Auto Scaling + Deployment Integration + Security + Monitoring"** — all built into Azure.

🧩 In Simple Terms

You upload your web app code to Azure App Service. Azure automatically provides servers, OS, load balancing, security, scaling, and continuous deployment — so you focus only on **your app**, not the **infrastructure**.

⚙️ Key Features of Azure App Service

Feature	Description
Fully Managed Platform (PaaS)	No need to manage servers, OS updates, or runtime — Azure does it all.
Multi-language Support	Supports .NET, .NET Core, Java, Python, Node.js, PHP, Ruby.
Cross-Platform Hosting	Choose **Windows** or **Linux** for your app environment.
Auto Scaling & Load Balancing	Automatically adjusts instances to handle traffic spikes; includes built-in load balancer.
High Availability (HA)	Built-in redundancy and integration with Azure Traffic Manager for global failover.
Continuous Deployment	Connect directly to GitHub, Azure DevOps, or Bitbucket for automatic CI/CD.
Integrated Security	Easily enable HTTPS, identity providers (Microsoft, Google, Facebook, etc.), and Azure AD authentication.
Custom Domains & SSL	Bind your domain name and manage SSL/TLS certificates.
Monitoring & Diagnostics	Azure Monitor and Application Insights provide performance metrics and logging.

🧱 App Service Types

Azure App Service includes **four main sub-services**, all hosted under the same managed environment 👇

1️⃣ Web Apps

- Used to **host dynamic websites or web applications**.
- Supports frameworks like ASP.NET, Node.js, Java, PHP, Python, or Ruby.
- Choose **Windows or Linux** as the runtime OS.

💡 *Example:* Host your company's main website or an internal web portal.

2️⃣ API Apps

- Used to **host RESTful APIs** that can be consumed by web, mobile, or IoT clients.
- Supports **Swagger/OpenAPI** for easy documentation and testing.
- APIs can be published to **Azure API Management** or even listed in the **Azure Marketplace**.

💡 *Example:* Host an API that returns customer or order data for a frontend app.

3️⃣ WebJobs

- Used to run **background processes or scripts** alongside your web or API app.
- Supports `.exe`, `.cmd`, `.bat`, PowerShell, Python, or Node.js scripts.
- Can run **continuously**, **on a schedule**, or **triggered** by events (like a queue message).

💡 *Example:* Run a job every night to clean up old log files or send summary emails.

4️⃣ Mobile Apps

- Provides a **back-end for mobile applications**.
- Quickly connect your mobile apps (iOS, Android, React Native, Xamarin) to Azure.
- Includes:
  - **Data storage** (in SQL or Cosmos DB)
  - **Authentication** (Google, Facebook, Microsoft, etc.)
  - **Push notifications**
  - **Offline sync**

💡 *Example:* Build a back-end for a food delivery mobile app.

🔄 How Deployment Works

You can deploy your code to Azure App Service in multiple ways:

- From **GitHub**, **Azure DevOps**, or any **Git repo** (continuous deployment)
- Using **VS Code / Visual Studio** (direct publish)
- Using **FTP**, **Zip deploy**, or **Azure CLI**
- Using **Container images** (App Service can host Docker containers too!)

📈 Scaling in App Service

App Service offers **two types of scaling:**

1. **Vertical Scaling (Scale Up):**
   Increase compute resources — like upgrading from Basic → Standard → Premium tier.
2. **Horizontal Scaling (Scale Out):**
   Increase the number of instances (e.g., 1 to 5 servers).
   This can be **manual or automatic** based on CPU/memory usage or schedule.

☁️ Azure Storage Types Overview

Azure provides three main ways to store data, depending on *how* you want to access and manage it:

Type	Best For	Azure Service Example
Block Storage	Disks for VMs	Azure Managed Disks
File Storage	Shared file systems	Azure Files
Object Storage	Unstructured data like images, backups	Azure Blob Storage

🧱 1️⃣ Block Storage

🔹 Concept:

- Stores data in **fixed-size blocks**.
- Each block has a unique ID; the system reassembles them when read.
- Acts like a **hard disk drive (HDD/SSD)** for a virtual machine.
- Provides **low-latency and high-performance** access.

🔹 Use Cases:

- OS disk and data disk for **Virtual Machines**.
- Databases and transaction-heavy workloads.

🔹 Azure Service:

- **Azure Managed Disks** (Standard HDD, Standard SSD, Premium SSD, Ultra Disk)

🔹 Example:

When you create a VM in Azure, the OS disk and data disks are stored using **block storage**.

📁 2️⃣ File Storage

🔹 Concept:

- Provides a **shared file system** accessible over the network.
- Uses the **SMB (Server Message Block)** protocol — same as traditional Windows file shares.
- Acts like a **shared drive** that multiple VMs or users can mount.

◇ Use Cases:

- Shared access for multiple VMs or apps.
- Lift-and-shift of on-premises apps that use file shares.
- Replacing or extending on-prem file servers.

◇ Azure Service:

- **Azure Files**

◇ Example:

Several Azure VMs need to access the same shared folder containing log files — use **Azure Files**.

🗃️ 3️⃣ Object Storage

◇ Concept:

- Stores data as **objects** (each with metadata + unique ID).
- Unlike block or file storage, there's **no hierarchy or folders** — it's flat.
- Highly **scalable, durable, and cost-effective**.
- Ideal for **unstructured data** like media files, backups, or logs.

◇ Use Cases:

- Storing images, videos, documents, logs.
- Backups and disaster recovery.
- Hosting static website files.

◇ Azure Service:

- **Azure Blob Storage**

◇ Example:

You upload profile images or application logs — they're stored as **objects in Azure Blob Storage**.

⚖️ Comparison Table

Feature	Block Storage	File Storage	Object Storage

Azure Service	Azure Managed Disks	Azure Files	Azure Blob Storage
Structure	Fixed-size blocks	Hierarchical (folders/files)	Flat (objects + metadata)
Access Type	Attached to VM	SMB/NFS network share	REST API / HTTPS
Use Case	VM OS/Data disks	Shared access between apps	Backups, images, media
Performance	High	Medium	Depends on tier
Scalability	Limited (per disk)	Moderate	Extremely high

🧠 Quick Analogy

Storage Type	Think of it as...
Block Storage	A hard disk attached to your PC (fast, structured)
File Storage	A shared drive on your office network
Object Storage	A giant online bucket (like Google Drive or Dropbox backend)

☁️ Azure Storage Overview

🔹 What It Is

Azure Storage is a **managed cloud storage solution** provided by Microsoft. That means Microsoft handles:

- Hardware maintenance
- Redundancy (backups, fault tolerance)
- Scalability (you can store **petabytes** of data)
- Security and access control

It's designed to be:

- **Highly available** → always accessible
- **Durable** → your data is protected against hardware failure
- **Massively scalable** → grows automatically as your data increases

📦 Core Storage Services in Azure

Azure provides **five main storage types**, each serving a different purpose 👇

🧱 1️⃣ Azure Disks

- **Type:** Block Storage (like virtual hard disks)
- **Purpose:** To store OS and data for **Azure Virtual Machines**
- **Usage:** When you create a VM, it uses an **Azure Managed Disk** for the operating system and additional disks for data.

💡 *Think of it like the C: and D: drives on your computer.*

📁 2️⃣ Azure Files

- **Type:** File Storage (shared files)
- **Purpose:** Provide **shared file storage** accessible via **SMB protocol** (like Windows shared drives)
- **Usage:** Multiple Azure VMs or even on-prem servers can access the same files.

💡 *Think of it like a shared network folder that all your servers can use.*

🗃️ 3️⃣ Azure Blobs

- **Type:** Object Storage (for unstructured data)
- **Purpose:** Store text, images, videos, backups, logs, etc.
- **Usage:** Ideal for large-scale storage, media files, or static website hosting.

💡 *Think of it like an online "bucket" where you drop files — fast, cheap, scalable.*

📬 4️⃣ Azure Queues

- **Type:** Messaging Service
- **Purpose:** Helps **decouple applications** using **message-based communication**
- **Usage:** One component adds a message to the queue, another component reads and processes it asynchronously.

💡 *Example:* When a web app receives a user request, it sends a message to a queue. A background worker later processes it.
→ This improves performance and reliability.

🗄️ 5️⃣ Azure Tables

- **Type:** NoSQL Key-Value Store (basic)

- **Purpose:** Store structured data (like JSON objects) in a simple, scalable, and inexpensive way.
- **Usage:** For quick storage of semi-structured data without needing a full database.

💡 *Note:* For more advanced NoSQL needs, Microsoft recommends **Azure Cosmos DB**.

⚙️ Storage Account – The Prerequisite

Before you use **Azure Files**, **Blobs**, **Queues**, or **Tables**, you must create a **Storage Account**.

◇ What is a Storage Account?

A **Storage Account** is like a **container** or **root account** that holds all your storage services. It defines:

- Security (access keys, shared access signatures)
- Redundancy (LRS, GRS, ZRS, etc.)
- Performance tier (Standard or Premium)
- Region (where your data is stored)

💡 *Think of it like a main folder that contains all your Azure storage services.*

🧠 Summary Table

Service	Type	Best For	Example Use Case
Azure Disks	Block	VM OS/Data	Running a Virtual Machine
Azure Files	File	Shared file access	Shared drive for multiple VMs
Azure Blobs	Object	Unstructured data	Images, backups, videos
Azure Queues	Messaging	Decoupling apps	Async message processing
Azure Tables	NoSQL	Semi-structured data	Storing metadata or logs

☁️ Azure Storage Data Redundancy – In Depth

Azure ensures your data is **durable, available, and recoverable** even in case of hardware failure, data center outage, or regional disaster. This is achieved through **replication**, i.e., **storing multiple copies of your data** in different physical locations.

Azure gives **four main redundancy options**, each with different **cost, availability, and durability**.

⚙️ 1️⃣ Locally Redundant Storage (LRS)

- **Replication type:** 3 copies (synchronous) within a **single data center**.
- **Durability:** 99.999999999% (11 nines) within the same data center.
- **Availability:** If that **data center fails completely**, data might be lost.
- **Cost:** 💰 Lowest cost.
- **Use Case:** Best for **non-critical**, **re-creatable**, or **development/test data**.

💡 *Example:*
Storing temporary log files or cache data for an app running in the same region.

📍 **Visual Concept:**

```
Region                                                                  A
 └──    Data    Center    1:    [    Copy1    |    Copy2    |    Copy3    ]
```

⚙️ 2️⃣ Zone-Redundant Storage (ZRS)

- **Replication type:** 3 copies (synchronous) across **three separate availability zones** within the **same region**.
- **Durability:** 99.9999999999% (12 nines).
- **Availability:** Survives a **data center** or **zone-level failure**.
- **Cost:** 💰💰 Moderate.
- **Use Case:** Suitable for **business-critical** data requiring **high availability** but staying **within one region** (for compliance or latency).

💡 *Example:*
Hosting data for an e-commerce app that must stay online even if one zone fails.

📍 **Visual Concept:**

```
Region                                                  A
 ├──                Zone                1:          Copy1
 ├──                Zone                2:          Copy2
 └──                Zone                3:          Copy3
```

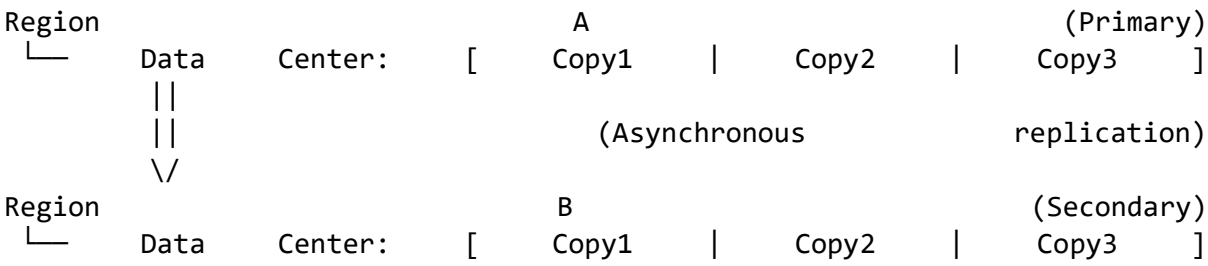
⚙️ 3️⃣ Geo-Redundant Storage (GRS)

- **Replication type:**
  - 3 copies in **primary region** (LRS)
  -
    - 3 **asynchronous** copies in a **paired secondary region** (hundreds of km away).
- **Durability:** 99.99999999999999% (16 nines) over a year.
- **Availability:** Survives **entire region failure**.

- **Failover:** Manual – you need to **initiate account failover** to secondary region.
- **Cost:** 💰💰💰 Higher than ZRS.
- **Use Case:** For **disaster recovery (DR)** and **data residency** compliance.

💡 *Example:*
Data from `East US` region replicated to `West US` for recovery if East US goes down.

📍 **Visual Concept:**

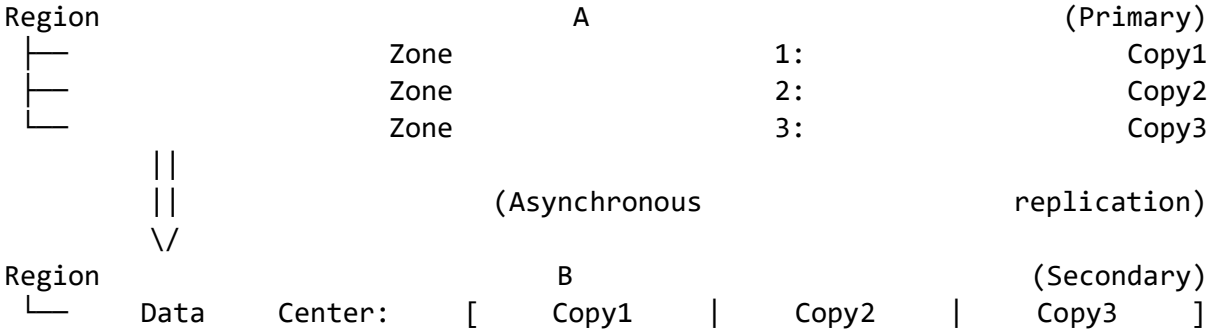


☸️ 4️⃣ Geo-Zone-Redundant Storage (GZRS)

- **Replication type:**
  - 3 **synchronous** copies across zones in **primary region** (ZRS)
  -
    - 3 **asynchronous** copies in a **secondary region** (LRS).
- **Durability:** 99.99999999999999% (16 nines) — same as GRS but more resilient.
- **Availability:** Survives **zone failure + region failure**.
- **Failover:** Manual.
- **Cost:** 💰💰💰💰 Most expensive.
- **Use Case:** For **mission-critical apps** that must remain available and recoverable even during **major disasters**.

💡 *Example:*
Banking or healthcare systems where **zero data loss** and **continuous uptime** are mandatory.

📍 **Visual Concept:**



🔁 RA-GRS and RA-GZRS (Read-Access Versions)

Both **GRS** and **GZRS** have read-access variants:

- **RA-GRS** → Read-Access Geo-Redundant Storage
- **RA-GZRS** → Read-Access Geo-Zone-Redundant Storage

✅ These allow **read access to the secondary region** even before a failover. Useful for **geo-distributed apps** that want to **read data globally** while still writing to the primary region.

💡 *Example:*
A global news website that writes data to `East US` but allows read-only access from `West Europe`.

📊 Summary Comparison Table

Option	Copies	Replication Type	Region Coverage	Zone Tolerance	Region Tolerance	Read Access Secondary	Cost	Best Use Case
LRS	3	Synchronous	Single DC	❌	❌	❌	💰	Dev/test, low-cost workloads
ZRS	3	Synchronous	Single Region (multi-zone)	✅	❌	❌	💰💰	High availability within a region
GRS	6	Sync + Async	Two Regions	❌	✅	❌	💰💰💰	Disaster recovery
GZRS	6	Sync + Async	Two Regions (multi-zone)	✅	✅	❌	💰💰💰💰	Mission-critical data
RA-GRS	6	Sync + Async	Two Regions	❌	✅	✅	💰💰💰	DR + read access from secondary
RA-GZRS	6	Sync + Async	Two Regions (multi-zone)	✅	✅	✅	💰💰💰💰	Max durability + read access

☁️ Azure Storage Account Types – Standard vs Premium

◇ What is a Storage Account?

A **Storage Account** in Azure is a logical container that holds your **data objects** — like blobs, files, queues, and tables.
Every data service (Azure Blobs, Files, Queues, Tables, etc.) must exist inside a **storage account**.

⚙️ 1️⃣ Standard Storage Accounts

Default and most commonly used option.

◇ Features:

- Uses **magnetic hard disk drives (HDDs)** for storage.
- Designed for **cost efficiency** and **general workloads**.
- Ideal for **data that doesn't require very high IOPS or ultra-low latency**.

◇ Supported Redundancy Options:

✅ LRS
✅ ZRS
✅ GRS
✅ GZRS
✅ RA-GRS
✅ RA-GZRS

So — **Standard accounts support all redundancy options**, giving **maximum flexibility**.

◇ Common Use Cases:

- Backups, archives, logs
- Web content storage
- Big data analytics
- Regular file shares or blob storage
- Disaster recovery setups

💡 *Example:* Hosting website images, or storing app logs that don't need instant high-speed access.

⚙ 2 Premium Storage Accounts

Designed for performance-critical, low-latency workloads.

◇ Key Difference:

- Uses **Solid-State Drives (SSDs)** instead of HDDs.
- Provides **very low latency**, **high throughput**, and **high IOPS**.
- Best suited for **transaction-intensive workloads**.

⚡ Performance Benefits:

Metric	Premium Storage	Standard Storage
Latency	< 1 ms typical	5–15 ms typical
Throughput	High (GB/s level)	Moderate
IOPS	20,000+ IOPS	Few thousand IOPS
Medium	SSD	HDD

◇ Constraint

Premium accounts **only support LRS and ZRS redundancy** — ❌ No geo-redundant options (GRS or GZRS). This is because **replicating SSD data asynchronously across regions** adds latency and cost, which defeats the goal of ultra-fast performance.

◇ Supported Premium Account Types

Premium Account Type	Description	Use Case
Premium Block Blob Storage	Stores unstructured data (text, images, video) using SSDs	Real-time analytics, ML input data, hot blob storage
Premium File Shares	High-performance **Azure Files** over **SMB or NFS protocols**	Enterprise file sharing, lift-and-shift file servers
Premium Page Blobs	Optimized for **random read/write** (VM disks)	Azure Virtual Machine disks, database storage (SQL, Oracle)

◇ Premium Blob Storage

- Type of Azure Blob optimized for frequent read/write operations.
- Good for apps like:
  - Real-time streaming
  - IoT data ingestion

    - High-performance computing workloads
- Available as **Block Blob** or **Append Blob**.

◇ Premium File Shares

- Used when apps need **shared file systems** with **high throughput and low latency**.
- Supports:
    - **SMB (Server Message Block)** – for Windows-based workloads.
    - **NFS (Network File System)** – for Linux-based workloads.
- Perfect for:
    - File-based business apps
    - Lift-and-shift Windows file servers
    - Dev/test environments with fast I/O

◇ Premium Page Blobs

- Backing storage for **Azure Managed Disks** (OS and Data disks for VMs).
- Optimized for **random read/write**, not sequential streaming.
- Perfect for:
    - Databases (SQL, Oracle)
    - Virtual Machines requiring fast disk I/O
    - Transactional systems

❁ When to Use Premium vs Standard

Scenario	Recommended Type
General-purpose workloads	**Standard**
Backup or archival	**Standard (Cool or Archive tier)**
Big data analytics	**Standard or Premium (depending on throughput)**
High-performance databases (SQL/Oracle)	**Premium (Page Blobs)**
Shared file systems with high IOPS	**Premium File Shares**
Real-time or ML workloads	**Premium Block Blobs**
Disaster recovery (geo-redundant)	**Standard (GRS/GZRS)**

💡 Extra Tips

1. **Mix and Match:**

a. You can use **Premium Storage** for your performance-critical workloads (e.g., databases)
b. And **Standard Storage** for backups or infrequently accessed data.

2. **Tiering (for Blob Storage):**
   a. You can choose **Hot**, **Cool**, or **Archive** tiers in **Standard accounts** to optimize cost based on access frequency.
   b. **Premium** storage has only **Hot** tier (because it's built for speed).
3. **Billing:**
   a. Premium accounts charge based on **provisioned storage**, not just usage. (You pay for the capacity you reserve.)
   b. Standard accounts charge for **actual data stored** and **transactions**.

Azure Disk Storage Explained

What is Azure Disk Storage?

Azure **Disk Storage** provides **block-level storage volumes** that can be attached to **Azure Virtual Machines (VMs)**.
They act just like physical hard drives but are **virtual**, **scalable**, and **managed by Azure**.

Each VM OS, app, and data disk in Azure uses **Azure Disk Storage**.

Disk Types in Azure

Azure offers **four types of disks**, each optimized for specific workload and performance requirements.

Disk Type	Backing Storage	Performance	Use Case	Redundancy
Standard HDD	Traditional Hard Disk Drives	Low	Backup, infrequent access	LRS, ZRS
Standard SSD	Solid State Drives (SSD)	Medium	Web servers, test environments	LRS, ZRS
Premium SSD	High-Performance SSD	High	Production workloads, databases	LRS, ZRS
Ultra SSD	Advanced SSD (NVMe)	Very High	IO-intensive apps like SAP, SQL	LRS only

1 Standard HDD Disks

- Backed by **magnetic hard drives (HDDs)**.
- **Lowest cost** option.
- Best for workloads with **low IOPS** and **high latency tolerance**.
- **Throughput:** ~60 MB/s per disk.
- **IOPS:** Up to 500 IOPS.

- **Use Cases:**
  - Backup
  - Archive storage
  - Non-critical workloads
  - Test environments (non-performance)

💡 *Think of it like a basic mechanical hard disk — cheap, good for storing data, not for speed.*

⚙️ 2️⃣ Standard SSD Disks

- Backed by **Solid-State Drives**, providing **faster performance** than HDD.
- Offers **better reliability and consistency**.
- **IOPS:** Up to 6,000.
- **Latency:** Typically <10 ms.
- **Use Cases:**
  - Web servers
  - Light enterprise apps
  - Development and testing
  - Small databases

💡 *Balanced between cost and performance.*

⚡ 3️⃣ Premium SSD Disks

- Designed for **high-performance production workloads**.
- Uses **enterprise-grade SSDs**.
- **IOPS:** Up to 20,000+ per disk.
- **Throughput:** Up to 900 MB/s per disk.
- **Latency:** 1–2 ms.
- **Redundancy:** LRS, ZRS supported.
- **Use Cases:**
  - Production-grade web servers
  - High-traffic applications
  - Databases like SQL, Oracle, MySQL
  - Application servers
  - Virtual desktops (Azure Virtual Desktop)

💡 *Perfect for apps needing fast response and consistent performance.*

🚀 Ultra SSD Disks

- The **highest-performing** disk type in Azure.
- Based on **NVMe (Non-Volatile Memory Express)** technology.
- **IOPS:** Up to 160,000 IOPS per disk!
- **Throughput:** Up to 4,000 MB/s.
- **Latency:** <1 ms.
- You can **dynamically adjust** IOPS and throughput **without detaching the disk**.
- **Use Cases:**
  - Mission-critical databases (SQL, Oracle, SAP HANA)
  - Real-time transaction-heavy apps
  - Financial systems

💡 *Think of Ultra SSD as "super SSDs" — built for ultra-low latency and massive throughput.*

🔁 Availability and Durability

- All Azure Managed Disks automatically store **three replicas** of your data within the same region.
- Premium and Ultra SSDs offer **very high availability (99.9%+)**.
- For disaster recovery, you can use **Azure Backup** or **Azure Site Recovery** to replicate disks across regions.

🧭 Managed vs Unmanaged Disks

🔷 Managed Disks (Modern Approach ☑)

- Introduced to **simplify disk management**.
- Azure **automatically handles storage**, replication, and performance.
- You **don't need to create or manage storage accounts**.
- Automatically distributed across **fault and update domains** → improves reliability.
- Supports **encryption**, **snapshots**, and **scaling**.
- Supports **up to 50,000 managed disks per subscription** per region.

💡 *You focus on your VM — Azure handles the rest.*

Advantages:

- High availability (replicated data)
- Auto-placement for reliability
- Easy backup & restore (via snapshots)
- Simpler scaling and management

◇ Unmanaged Disks (Legacy Approach ⚠)

- Older model — **you manage the storage account** that holds the VHD files.
- Each disk is stored as a **VHD blob** in an Azure Storage Account.
- You must ensure that the storage account does not exceed its **IOPS or capacity limit (20,000 IOPS max per account)**.
- **More administrative overhead** and prone to human error.

💡 *Now mostly deprecated — Azure recommends Managed Disks for all new workloads.*

How Azure Uses Disks

- Each Azure VM typically has:
  - **OS Disk** → Boot disk (C drive)
  - **Data Disks** → For app/data storage
  - **Temporary Disk (D drive)** → Local SSD used for caching/swap; data is *not persistent*.
- You can **attach/detach disks** easily and resize disks without downtime.

Extra Features of Azure Managed Disks

Feature	Description
Snapshots	Point-in-time backup of a disk.
Encryption	Data encrypted at rest using AES 256-bit or your own key (Customer-Managed Key).
Shared Disks	Attach a single disk to multiple VMs (useful for clustered apps).
Bursting	Some SSDs can temporarily exceed their performance limits for short bursts.
Disk Pooling	Combine multiple disks for higher throughput (like RAID).
Scalability	Up to 32 TB per disk.

Azure Shared Disks

Definition:
Azure Shared Disks allow you to **attach a single managed disk to multiple virtual machines simultaneously**.
This is useful for **clustered applications** that require **shared block-level storage**, such as:

- **Windows Server Failover Clustering (WSFC)**

- **SQL Server Failover Cluster Instances (FCI)**
- **SAP ASCS/SCS clusters**
- **Oracle RAC (Real Application Cluster)**

⚙️ Why It's Needed

Traditionally, in on-premises environments, cluster nodes connect to the same **shared SAN (Storage Area Network)** or **shared disk**.
In Azure, each VM usually has its **own dedicated disk**, but **Shared Disks** make similar architectures possible in the cloud.

So, multiple VMs can:

- Read/write to the same disk (coordinated via clustering software).
- Detect disk-level failures.
- Manage failover and resource sharing.

🧱 Supported Disk Types

Disk Type	Shared Disk Support	Notes
Premium SSD	✅ Yes	Recommended for production clusters
Ultra SSD	✅ Yes	Highest performance
Standard SSD / HDD	❌ No	Not supported

⚙️ Key Features and Rules

1. **Max Shares Setting**
    a. Each shared disk has a setting called `maxShares` (1–5).
    b. This defines **how many VMs** can attach the disk at once.
    c. Example: If `maxShares = 2`, up to **two VMs** can attach it.
2. **Attachment Mode**
    a. Disks must be attached in **"Shared" mode**, not "Exclusive".
    b. This mode allows multiple VMs to mount the same disk.
3. **Operating System Requirements**
    a. Cluster-aware OS (e.g., **Windows Server 2016+**, **RHEL**, **SLES**)
    b. You must use **cluster file systems** (e.g., NTFS with WSFC, or GFS2 for Linux).
4. **Availability Sets or Zones**
    a. The VMs sharing a disk must be in the **same availability set** or **availability zone** (for latency and consistency).
5. **Supported Disk Sizes**
    a. Shared Disks are available only for **Premium SSDs ≥ P15 (256 GB)** or **Ultra Disks**.

Benefits

- Enables **lift-and-shift** of traditional clustered applications to Azure.
- No need to refactor your app for distributed storage.
- High availability and automatic failover supported.
- Simplifies **shared data access** across multiple VMs.

⚠ Limitations

Limitation	Description
Not supported on Standard SSD/HDD	Only Premium & Ultra SSDs support shared access.
No write coordination by Azure	App or OS cluster software must manage access.
Same region / zone	All attached VMs must reside in the same region and zone.
Snapshot and Backup restrictions	Some operations (e.g., snapshotting) require detaching from one or more VMs.
Managed Disks only	Shared disks are only supported for **Managed Disks**, not Unmanaged.

Example Use Case: SQL Server Failover Cluster

Imagine you have **two VMs (Node1 and Node2)** running **SQL Server FCI**:

- Both share the same **Premium SSD (Shared Disk)** that stores the database files.
- If **Node1** fails, **Node2** takes ownership of the shared disk and continues serving data.
- This setup ensures **high availability** for the SQL instance.

Quick Summary

Feature	Shared Disk
Purpose	Allow multiple VMs to access same disk
Disk Types	Premium SSD, Ultra SSD
Typical Use	Clustering, high-availability setups
Max Shared VMs	Up to 5
Region Scope	Same region/zone
OS Requirement	Cluster-aware OS
Management	Only Managed Disks supported

✅ **In short:**
Yes, you can attach a single disk to multiple VMs in Azure — using **Azure Shared Disks** on **Premium SSD** or **Ultra SSD**.
This enables **traditional clustering scenarios** in a cloud-native way.

Azure Files

Azure Files is a fully managed **file share service in the cloud** provided by Microsoft Azure. It allows organizations to store and share files in a way similar to traditional on-premises file servers, but with cloud scalability, security, and accessibility.

Key Features:

1. **Managed File Shares**
    a. Fully managed by Azure – no need to worry about hardware, patching, or maintenance.
    b. Supports **persistent storage** accessible from multiple clients simultaneously.
2. **Multi-Device Connectivity**
    a. Connect from **cloud-based VMs** or **on-premises servers**.
    b. Works across multiple operating systems: **Windows, Linux, macOS**.
3. **Protocol Support**
    a. **SMB (Server Message Block):** Common for Windows-based applications.
    b. **NFS (Network File System):** Common for Linux/UNIX workloads.
4. **Concurrent Access**
    a. Multiple VMs or devices can read/write to the same file share at the same time.
5. **Use Cases**
    a. **Shared configuration files** for multiple VMs.
    b. **Media workflows** – video editing, large file sharing.
    c. **Lift-and-shift applications** that rely on file shares.
    d. **Backup and archival storage** for on-premises data.

Additional Important Points:

6. **Storage Tiers**
    a. Azure Files supports **standard (HDD) and premium (SSD) tiers** depending on performance needs.
7. **Integration with Azure AD**
    a. Secure file shares using **Azure Active Directory (AD) authentication**.
    b. Supports **role-based access control (RBAC)** for fine-grained access management.
8. **Snapshot Support**
    a. Can take **snapshots of file shares** for backup, recovery, or versioning.
9. **Scalability**
    a. Can scale **up to hundreds of TiB** per share.
    b. Handles **high IOPS workloads** when using premium tier.
10. **Hybrid Access**
    a. Using **Azure File Sync**, on-premises Windows Servers can **cache frequently accessed files locally** while keeping the master copy in the cloud.
11. **Security**
    a. Data is **encrypted at rest** and **in transit**.
    b. Can integrate with **firewalls and private endpoints** to control network access.

12. **Pricing Model**
    a. Pay for storage **per GB** used.
    b. Premium tier charges based on **provisioned size**.

✅ **Summary:**

Azure Files acts like a cloud-based file server that allows multiple users and VMs to securely share files across platforms. It's ideal for enterprises needing **scalable, managed, and secure file storage** accessible from anywhere.