# Week 1

**Introduction to distributed computing systems**.

A distributed system is defined as a **collection of autonomous computing elements (referred to as nodes) that appears to its users as a single coherent system**. These nodes, which can be hardware devices or software processes, operate independently but collaborate to present a unified view.

The emergence of distributed systems was significantly influenced by two technological advancements in the mid-1980s: the development of **powerful microprocessors** and the invention of **high-speed computer networks** (LANs and WANs). This made it feasible to connect previously isolated computers, leading to systems composed of numerous networked machines, often geographically dispersed. The **miniaturisation of computer systems**, exemplified by smartphones, further contributes to this landscape, as these devices are essentially powerful, networked computers.

Key characteristics of distributed systems include:

* **A collection of autonomous computing elements**:
  + Nodes operate independently and are programmed to achieve common goals through message exchange.
  + Each node maintains its own clock, necessitating **node synchronisation** and coordination mechanisms due to the absence of a global clock.
  + **Node coordination** involves managing group membership, including nodes joining and leaving, authorisation, and maintaining lists of communicable nodes, forming open or closed groups with admission control.
* Open group: any node is allowed to join the distributed system
* Closed group: communication & membership is restricted
  + **Overlay networks** are built on top of another network (typically the internet), where nodes (software processes) communicate with a set of neighbours, which can be structured (e.g., tree, ring) or unstructured, and are generally always connected.
* Structured: each node has a well-defined set of neighbors with whom it can communicate (tree, ring).
* Unstructured: each node has references to a random selection of other nodes from the system.
  + The **OSI Model** standardises communication functionalities through abstract layers, with message passing often using TCP/IP or UDP.
  + **Peer-to-peer networks** leverage the collaborative ability of computing systems to aggregate resources across many nodes, offering potential for greater scale and robustness than centralized client-server models, with considerations for fault tolerance.
* **A single coherent system**:
  + The collection of nodes operates consistently, regardless of the location, time, or method of user interaction.
  + A distributed system is considered coherent if it meets user expectations.
  + **Distribution transparency** is a key design goal, aiming to hide the distributed nature of the system from users. This includes various types:
    - **Location transparency:** Users are unaware of the physical location of a resource (e.g., via URLs).
    - **Relocation transparency:** Resources can move without user awareness.
    - **Migration transparency:** Users are unaware if a resource can move.
    - **Replication transparency:** The existence of multiple copies of a resource is hidden.
    - **Concurrency transparency:** Users are unaware of the mechanisms (like locking) managing concurrent resource use.
    - **Failure transparency:** Failures and recovery are masked from users.
  + **Middleware** acts as the operating system for distributed systems, providing inter-node communication, security, reliability, service protocols, and transaction support.

Several important **issues** arise in distributed systems:

* **Scalability:** The ability to grow in size, across locations, and administrative boundaries.
* **Data consistency:** Maintaining uniformity of replicated data.
* **Fault tolerance:** Resilience to partial failures.
* **Distribution transparency:** The degree to which the distribution is hidden.
* **Security:** Protecting distributed processes and data.

The **design goals** of distributed systems include:

* **Accessibility:** Supporting resource sharing.
* **Distribution transparency:** Hiding the system's distributed nature.
* **Openness:** Enabling interoperability, portability, and extensibility to avoid vendor lock-in. This involves separating policy from mechanism and utilising well-defined interfaces.
* **Scalability:** The capacity to handle increased workload by adding resources. This has dimensions of size, geographical distribution, and administration. Approaches to scaling include hiding latencies, partitioning, moving computations, decentralisation, and replication with load balancing. Replication introduces challenges of maintaining consistency.

Developing distributed systems is challenging due to the inherent unreliability and insecurity of networks. It's crucial to avoid false assumptions about network characteristics like reliability, security, homogeneity, static topology, zero latency, infinite bandwidth, zero transport cost, and single administration.

Cloud computing, with infrastructures like AWS, is presented as a modern example of distributed systems, highlighting concepts like provisioning computer resources and scaling computing tasks. Collaboration and data sharing across different cloud providers and on-site infrastructure also demonstrate key aspects of distributed computing.

# Week 2

**Fundamental theoretical concepts and technologies** **of cloud computing**

One of the core enabling technologies of cloud computing is **Virtualization**, which involves **partitioning the resources of a physical system (computing, storage, network, and memory) into multiple virtual resources**. This allows for the **pooling of resources**, which is essential in cloud computing where resources are shared among multiple users using **multi-tenancy**.

The **virtualization layer** is managed by a **hypervisor** or a **virtual machine monitor (VMM)**. The hypervisor **presents a virtual operating platform to a guest operating system (OS)**. There are two main types of hypervisors:

* **Type-1 (Native) Hypervisors**: These run **directly on the host hardware**, controlling the hardware and monitoring guest OSs.
* **Type-2 (Hosted) Hypervisors**: These run **on top of a conventional (main/host) operating system** and monitor guest OSs.

There are different **Types of Virtualization**:

* **Full Virtualization**: The **guest OS is completely decoupled from the underlying hardware and requires no modification**, unaware of being virtualized. It uses **direct execution of user requests and binary translation of OS requests**. Examples include Microsoft and Parallels systems.
* **Para-Virtualization**: The **guest OS is modified to communicate with the hypervisor (via hypercalls) to improve performance and efficiency**. The OS kernel is altered to replace non-virtualizable instructions with **hyper-calls** that directly communicate with the hypervisor. A key feature is **address space virtualization**, giving each virtual machine its own unique address space. Examples include VMware and Xen.
* **Hardware Virtualization (Hardware assisted virtualization)**: Enabled by hardware features like **Intel’s Virtualization Technology (VT-x) and AMD’s AMD-V**. Privileged and sensitive calls **automatically trap to the hypervisor**, eliminating the need for binary translation or para-virtualization.

**Cloud Computing** is defined by the U.S. National Institute of Standards and Technology (NIST) as **“a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”**.

Key **Characteristics of Cloud Computing** include:

* **On-demand self service**: Users can **provision resources automatically** without needing to interact with the cloud service provider.
* **Broad network access**: Resources can be accessed **over the network using standard mechanisms** from various client devices.
* **Resource pooling**: Computing and storage resources are **pooled to serve multiple users using multi-tenancy**.
* **Rapid elasticity**: Resources can be **quickly scaled up or down based on demand**.
* **Measured service**: Users are charged based on their **usage of resources** according to specific metrics (pay-per-use).
* **Performance**: Applications can experience **improved performance** due to the ability to scale resources dynamically.
* **Reduced costs**: Users can **provision only the necessary resources dynamically**, avoiding upfront investments for worst-case scenarios.
* **Outsourced Management**: Users can **outsource IT infrastructure requirements** to cloud providers.
* **Reliability**: Cloud environments generally offer **higher reliability** due to professional management by the service provider.
* **Multi-tenancy**: Shared resources among multiple users, which can be **virtual** (sharing computing and storage) or **organic** (every component shared).

Key **Terminology** in cloud computing includes:

* **Cloud services**: A broad range of resources accessible “as-a-service,” categorised as **Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS)**.
* **Service-level-agreements (SLAs)**: Establish **expectations for uptime, security, availability, reliability, and performance**.

The basic **Cloud Service Models** are:

* **Software as a Service (SaaS)**: Provides **applications, management, and user interfaces over a network**. Users **use the software over the internet without managing any underlying infrastructure**. Characteristics include a complete software application or interface, **outsourced management** of the infrastructure by the provider, **thin client interfaces** (e.g., browser) for platform-independent access, and **ubiquitous access** from anywhere. Examples include Salesforce, Google Workspace, MS Office 365, Dropbox, Zoom, Slack.
* **Platform as a Service (PaaS)**: Offers **application development frameworks, operating systems, and deployment frameworks**. Users **build and deploy applications on a platform provided by the vendor, without managing the underlying infrastructure**. Characteristics include enabling **development and deployment** in the cloud using provider tools and APIs, **provider management of the infrastructure** (servers, network, OS, storage), and **user management of their applications**. Examples include AWS Elastic Beanstalk, Google App Engine, Microsoft Azure App Service, Heroku, OpenShift.
* **Infrastructure as a Service (IaaS)**: Provides **virtual computing, storage, and network resources that can be provisioned on demand**. Users **rent the infrastructure (servers, storage, networking) from the vendor and manage the operating system, applications, and data**. Characteristics include **resource provisioning** (computing and storage), provided as **virtual machine instances and virtual storage** that users can manage, **provider management of the underlying infrastructure**, and a **pay-per-use/pay-as-you-go** billing model. Examples include Amazon Web Services (AWS) EC2, Microsoft Azure Virtual Machines, Google Compute Engine, DigitalOcean, Rackspace.

Cloud resources can be deployed using different **Cloud Deployment Models**:

* **Public Cloud**: Available for **public use or a large industry group**. Examples include AWS, Microsoft Azure, GCP.
* **Private Cloud**: **Operated for the exclusive use of a single organisation**. Examples include VMware vSphere, Microsoft Azure Stack, a bank’s internal data centre.
* **Community Cloud**: Available for **shared use of several organisations supporting a specific community**. Examples include Government Cloud, Healthcare Cloud, Financial Services Cloud, Education Cloud.
* **Hybrid Cloud**: **Combines multiple clouds (public and private) that remain unique but are bound together** to offer application and data portability. Examples include a company using AWS for compute and storage but keeping sensitive data private, a retailer using Azure for peak traffic while core systems remain private, an organisation using VMware with AWS Outposts, and a development team using private for development and public for production.

**COMPUTE IN THE CLOUD** focuses on **Amazon Elastic Compute Cloud (EC2)**, a service that provides **secure, resizable compute capacity in the cloud as Amazon EC2 instances**.

Traditional on-premises resources require upfront hardware purchases, delivery waiting times, physical installation, and manual configurations. In contrast, with Amazon EC2, users can **provision and launch virtual servers within minutes**, stop using them when finished, and **pay only for the compute time used when an instance is running**. This offers cost savings by paying only for needed server capacity.

**Virtualization Technology enables EC2**. EC2 runs on physical host machines managed by AWS, with a **hypervisor** sharing the underlying resources between multiple virtual machines (instances). This **sharing of hardware is called multitenancy**, coordinated and managed by AWS. The hypervisor also **isolates the virtual machines from each other**, ensuring security.

Using EC2 offers several advantages:

* **Highly flexible, cost-effective, and quick** compared to on-premises servers.
* **Easier to get started** as AWS handles the infrastructure (data centres, servers, networking).
* Provides **flexibility and control** over starting and stopping servers and their configurations, including the choice of operating system (Windows or Linux).
* **Resizable instances** allow for vertical scaling (making instances bigger or smaller) as needed.
* Users also control the **networking aspects** of EC2.

**Amazon EC2 instance types** are grouped into families and **optimized for specific tasks**, offering varying combinations of CPU, memory, storage, and networking capacity. These include:

* **General purpose instances**: Balanced compute, memory, and networking for diverse workloads like web services, application servers, and small/medium databases.
* **Compute optimized instances**: Ideal for compute-intensive tasks like gaming servers, high-performance computing, and batch processing.
* **Memory optimized instances**: Suitable for memory-intensive tasks like high-performance databases and real-time processing of large unstructured data.
* **Accelerated computing instances**: Good for floating-point calculations, graphics processing, and data pattern matching using hardware accelerators, ideal for graphics, game streaming, and application streaming.
* **Storage optimized instances**: For workloads requiring high performance for locally stored data with high sequential read/write access to large datasets, such as distributed file systems, data warehousing, and high-frequency OLTP systems.

**Amazon EC2 pricing** involves paying only for the compute time used. Multiple billing options are available:

* **On-Demand**: Pay only for the duration the instance runs (per hour or per second). No long-term commitments or upfront payments needed. Ideal for short-term, irregular, and non-interruptible workloads, as well as for initial testing and baseline usage assessment.
* **Amazon EC2 Savings Plans**: Reduce compute costs by committing to a consistent hourly spend for 1 or 3 years, resulting in savings up to 66% over On-Demand. Offers flexibility across EC2 instance families, regions, AWS Lambda, and Fargate. There are **Compute Savings Plans** (most flexible) and **EC2 Instance Savings Plans** (lower rates for specific instance families in a region).
* **Reserved Instances**: Offer a billing discount for committing to On-Demand Instances for 1 or 3 years (greater savings with 3-year term). Suited for steady-state workloads with predictable usage, offering up to 75% discount. Require commitment to a specific instance type, region, availability zone, and OS. Payment options include all upfront, partial upfront, and no upfront.
* **Spot Instances**: Allow requesting spare EC2 computing capacity at up to 90% off the On-Demand price. AWS can reclaim the instance with a two-minute warning. Suitable for workloads with flexible start/end times or that can tolerate interruptions, such as batch workloads. Pricing is bid-based and availability is variable based on AWS's unused capacity. Integration with Auto Scaling and EC2 Fleet enables automated scaling and management.
* **Dedicated Hosts**: Physical servers with EC2 instance capacity fully dedicated to your use, for compliance or licensing requirements. More expensive than other EC2 options. Can be purchased On-Demand or as Reservations.

**Scalability** involves starting with needed resources and automatically responding to changing demand by scaling out or in, paying only for used resources. **Amazon EC2 Auto Scaling** automatically adjusts the number of EC2 instances in an application based on current conditions to maintain performance, availability, and optimise costs. It uses **dynamic scaling** (responds to changing demand) and **predictive scaling** (schedules instances based on predicted demand), which can be used together. An **Auto Scaling Group** can be configured with a minimum, desired, and maximum number of instances. Benefits include cost efficiency, improved availability, performance optimisation, and simplified management. Use cases include web applications, batch processing, and gaming applications.

**Elastic Load Balancing (ELB)** automatically distributes incoming application traffic across multiple resources, such as EC2 instances. A load balancer acts as a single point of contact for traffic to an Auto Scaling group, distributing the workload across multiple instances. ELB is automatically scalable and highly available. When EC2 instances in an Auto Scaling group come online, ELB is notified and starts routing traffic. When scaling in, ELB stops new traffic and waits for existing requests to complete before the instances are terminated, ensuring no disruption. ELB is a regional construct with a single URL for front-end instances, directing traffic to the back-end with the least outstanding requests, enabling decoupled architecture.

# Week 3

This document provides an overview of various compute services and the global infrastructure offered by AWS. It begins by discussing **messaging and queuing services**, specifically **Amazon Simple Notification Service (Amazon SNS)** and **Amazon Simple Queue Service (Amazon SQS)**.

**Amazon SNS** is a **publish/subscribe service** where publishers send messages to subscribers via topics. Subscribers can include web servers, email addresses, AWS Lambda functions, and others. SNS follows a pub/sub model, allowing a single message to a topic to be distributed to all its subscribers. It can send messages to services and notifications to end users through various methods like mobile push, SMS, and email.

**Amazon SQS** is a **message queuing service** that enables sending, storing, and receiving messages between software components without message loss or requiring constant availability of other services. Applications send messages to a queue, which are then retrieved, processed, and deleted by a user or service. The message data is called a payload and is protected until delivery. SQS queues are managed by AWS, scaling automatically, reliably, and are easy to use. It supports decoupled communication and offers two queue types: Standard (at-least-once delivery) and FIFO (exactly-once processing). SQS integrates well with other AWS services.

The document then contrasts **monolithic applications** with **microservices**. Monolithic applications have tightly coupled components, meaning if one fails, the entire application might fail. In contrast, **microservices** are loosely coupled, so the failure of one component does not necessarily affect others. AWS allows for a microservices approach using its various services.

The document then delves into **additional compute services** beyond traditional servers.

**Serverless computing** is introduced, where code runs on servers, but users do not need to provision or manage them, allowing focus on innovation instead of server maintenance. **AWS Lambda** is a serverless compute option that lets you run code without managing servers. You upload code into a Lambda function, configure a trigger, and AWS runs the code automatically in a managed, scalable, and highly available environment only when triggered. You pay only for the compute time consumed. Lambda can be used for various applications and backend services with zero administration.

**Containers** are presented as an alternative when serverless is not suitable or when access to the underlying environment is needed, offering efficiency and portability. They provide a standard way to package an application's code and dependencies into a single object, ensuring consistent environments across different deployments and reducing debugging time. **Container orchestration services**, such as **Amazon Elastic Container Service (Amazon ECS)** and **Amazon Elastic Kubernetes Service (Amazon EKS)**, help deploy, manage, and scale containerized applications. **Docker** is a widely used platform for delivering software in containers, which run on EC2 instances in isolation. Container orchestration automates tasks like deployment, scaling, load balancing, service discovery, health monitoring, resource management, and configuration management.

**Amazon ECS** is a highly scalable container management system supporting Docker containers, allowing API calls to launch and stop Docker-enabled applications. **Amazon EKS** is a fully managed service to run Kubernetes on AWS, enabling deployment and management of containerized applications at scale. **AWS Fargate** is a serverless compute engine for containers that works with both ECS and EKS, eliminating the need to provision or manage EC2 instances for containers.

The document provides guidance on **how to decide on computing options**. **EC2** is suitable for traditional applications needing full OS access. **AWS Lambda** is recommended for short-running, service-oriented, or event-driven applications without managing the underlying environment. For container-based workloads, the choice between **ECS** and **EKS** as orchestration tools and then between running on managed **EC2** instances or serverless **AWS Fargate** needs to be made.

The second part of the document covers **global infrastructure and reliability**. The AWS global infrastructure is designed for high availability and reliability through a network of **Regions** around the world.

A **Region** is a broad geographical area ensuring resource isolation and global scalability. AWS builds Regions close to business traffic demands, and each Region is connected by a high-speed fiber network. Users choose the Region to operate in, and each Region is isolated for security, with no data moving in or out without explicit permission. AWS currently operates across 29 Regions. Deploying resources across multiple Regions with multiple **Availability Zones (AZs)** within each ensures maximum reliability and performance for critical applications.

An **Availability Zone (AZ)** is a single data centre or a group of data centres within a Region, with redundant power, networking, and connectivity, located tens of miles apart for low latency but sufficient distance to reduce the impact of a localized disaster. When an EC2 instance is launched, it runs in a specific AZ. It is recommended to run across at least two AZs for high availability and fault tolerance. Many AWS services operate at the Regional level, running synchronously across multiple AZs.

**Edge locations** are sites used by **Amazon CloudFront**, AWS's Content Delivery Network (CDN), to store cached content closer to customers for faster delivery. CloudFront accelerates the delivery of data, video, applications, and APIs globally with low latency and high transfer speeds. Edge locations are separate from Regions. AWS Edge locations also run **Amazon Route 53**, a scalable and highly available Domain Name System (DNS) web service that directs customers to the correct web locations with low latency. Route 53 translates domain names to IP addresses, supports domain registration, offers various traffic management policies, performs health checks, and integrates with other AWS services.

**AWS Outposts** allows businesses to run AWS services inside their own on-premises data centres. AWS installs and operates a fully functional "mini-Region" within the customer's building, providing 100% AWS functionality in an isolated environment. This is suitable for specific problems requiring local data processing and low latency while leveraging AWS benefits.

The final part of the document discusses **how to provision AWS resources**. Everything in AWS is an **API call**, and there are pre-determined ways to interact with AWS services to provision, configure, and manage resources. Tools for this include:

* **AWS Management Console**: A web-based interface for accessing and managing AWS services. It is user-friendly for learning and testing but not ideal for production automation.
* **AWS Command Line Interface (AWS CLI)**: A tool allowing control of multiple AWS services directly from the command line, enabling scriptable and repeatable actions, reducing human error and allowing for automation.
* **Software Development Kits (SDKs)**: Libraries that make it easier to use AWS services through APIs designed for specific programming languages, enabling developers to integrate AWS with their applications.
* **AWS Elastic Beanstalk**: A Platform as a Service (PaaS) that simplifies the provisioning of EC2-based environments. Users provide application code and configurations, and Elastic Beanstalk automatically builds the environment, managing capacity, load balancing, scaling, and health monitoring.
* **AWS CloudFormation**: An Infrastructure as Code (IaC) tool allowing definition of AWS resources in declarative JSON or YAML templates. It automates the provisioning of resources in a safe and repeatable manner, managing API calls and allowing for consistent environments across different accounts and Regions. Elastic Beanstalk is better for deploying and managing applications, while CloudFormation provides more control over infrastructure as code.

# Week 4

an overview of cloud and global networking concepts within AWS. It explains how to establish boundaries for your AWS resources and how customers can interact with your AWS infrastructure.

### Cloud Networking with Amazon Virtual Private Cloud (Amazon VPC)

* An Amazon VPC is your **own private network within AWS**, allowing you to define your private IP address ranges and place resources like EC2 instances inside it.
* Resources in a VPC can be **public-facing** with internet access or **private** with no direct internet access, suitable for backend services.
* The grouping of public and private resources occurs within **subnets**, which are ranges of IP addresses within your VPC. Subnets, along with networking rules, determine the public or private availability of resources.
* To allow public internet traffic to enter and exit your VPC, you must attach an **internet gateway (IGW)**. An internet gateway acts as a doorway open to the public, enabling communication between your VPC and the internet. Without an internet gateway, resources within a VPC cannot be accessed from the internet.
* For accessing private resources within a VPC, you can use a **virtual private gateway**. This component allows protected internet traffic to enter your VPC and enables you to establish a **Virtual Private Network (VPN)** connection between your VPC and a private network, such as an on-premises data centre. A virtual private gateway ensures that only traffic from approved networks can enter the VPC. However, VPN connections use the regular internet, which has shared bandwidth.
* **AWS Direct Connect** offers an alternative for connecting your data centre to AWS: a **dedicated, private fibre connection** not shared with anyone else. This provides the lowest possible latency and highest security and can help meet strict regulatory and compliance requirements while avoiding potential bandwidth issues. You work with a Direct Connect partner to establish this physical connection to your AWS VPC. Importantly, a single VPC can have multiple types of gateways attached for different types of resources residing in different subnets within the same VPC.

### Securing VPC Resources

* Having a gateway that controls traffic in and out of the VPC is only one part of a comprehensive IT security strategy. AWS offers various tools covering different security layers.
* **Network hardening** involves controlling what happens inside the VPC, and subnets play a role in this beyond just controlling access to gateways.
* **Network Access Control Lists (ACLs)** are virtual firewalls that control **inbound and outbound traffic at the subnet level**. Every packet crossing subnet boundaries is checked against the network ACL rules to determine if it's allowed to enter or leave based on its origin and communication attempt. You can think of network ACLs as passport control officers checking traffic going into (ingress) and leaving (outgress) a subnet. If traffic isn't on the approved list or is explicitly denied, it gets blocked. Network ACLs perform **stateless packet filtering**, meaning they don't remember previous traffic and check every packet independently in both directions. Each AWS account has a **default network ACL** that allows all inbound and outbound traffic, but you can modify it or create custom network ACLs where all traffic is denied by default until you add specific allow rules. All network ACLs also have an explicit deny rule for packets that don't match any other rule.
* **Security Groups** are virtual firewalls that control **inbound and outbound traffic for individual Amazon EC2 instances**. Unlike network ACLs, security groups operate at the instance level. By default, a security group denies all inbound traffic, but you can modify it to allow specific types of traffic, such as web traffic (HTTPS). Security groups are **stateful**, meaning they remember previous decisions about incoming packets and allow the corresponding response traffic to return, regardless of outbound rules. Think of a security group as a doorman at a building (the EC2 instance), checking who is allowed to enter but not necessarily who is leaving.

### Global Networking

* Customers typically interact with your AWS infrastructure, such as a hosted website, by entering the web address into their browser. This process relies on the **Domain Name System (DNS)** resolution.
* DNS is like the phone book of the internet, translating a **domain name into an IP address**. DNS resolution involves a customer's DNS resolver communicating with a company's DNS server.
* **Amazon Route 53** is a highly available and scalable **DNS web service** that reliably routes end users to internet applications hosted in AWS or elsewhere. It connects user requests to infrastructure like EC2 instances and load balancers. Route 53 also allows you to manage DNS records for domain names, including registering new ones and transferring existing ones.
* Amazon Route 53 can work with **Amazon CloudFront**, a content delivery service, to deliver content to customers efficiently. When a customer requests data from an application, Route 53 resolves the website's domain name to an IP address, and then Amazon CloudFront directs the request to the nearest edge location. CloudFront then retrieves the content from the origin (e.g., EC2 instance via a load balancer) and delivers it to the customer.
* Route 53 offers various **routing policies** to direct traffic to different endpoints based on factors like latency or geographic location (geolocation DNS, geoproximity) and load balancing algorithms such as Round Robin and Weighted Round Robin.

# Week 5

an overview of **cloud storage services** and **database services**, primarily focusing on offerings from Amazon Web Services (AWS).

### Cloud Storage Services

The document begins by highlighting that applications running on Amazon EC2 need access to various components, including storage. It then focuses on **storage access**, particularly **block-level storage**.

#### Block-level storage

Block-level storage is described as a place to store files, where a file is a series of bytes stored in blocks on a disc. When a file is updated, only the changed pieces (delta updates) are overwritten, making it efficient for databases, enterprise software, or file systems. Personal computers use block-level storage (the hard drive), and EC2 instances also have hard drives of different types.

#### Instance stores

**Instance stores** provide temporary block-level storage directly attached to the host computer of an EC2 instance, meaning their lifespan is tied to the instance. Data in an instance store is lost when the instance is terminated (ephemeral storage) or even stopped, as the instance might restart on a different host without that attached volume. Due to this temporary nature, instance stores are suitable for temporary files and data that can be easily recreated. It is advised not to store important, persistent data on instance store volumes.

#### Amazon Elastic Block Store (Amazon EBS)

**Amazon EBS** offers persistent block-level storage volumes for use with EC2 instances. Data on an attached EBS volume remains available even if the EC2 instance is stopped or terminated. To use EBS, you configure and provision a volume and then attach it to an EC2 instance. EBS volumes are on separate drives and not directly tied to the host, ensuring data persistence across instance stops and starts. **Amazon EBS snapshots** provide incremental backups of EBS volumes, where the first backup copies all data, and subsequent backups save only the changed blocks. Regular EBS snapshots are crucial for data backup and recovery in case of drive corruption.

#### Amazon Simple Storage Service (Amazon S3)

**Amazon S3** is a data store that allows you to store and retrieve an unlimited amount of data at any scale. It is an **object storage** service where data is stored as **objects** within **buckets**, analogous to files in directories. Each object consists of data, **metadata** (information about the data), and a unique **key** (identifier). When an object in S3 is modified, the entire object is updated. S3 can store any type of file, with a maximum object size of 5 TB. Permissions can be set to control access to objects, and **versioning** allows tracking changes and protecting against accidental deletion by retaining previous versions. You only pay for the storage you use and can choose from various **S3 storage classes** based on data access frequency and required availability.

**S3 Storage Classes:**

* **S3 Standard:** High availability, frequently accessed data, suitable for websites, content distribution, data analytics, stored in a minimum of three Availability Zones (AZs), 99.999999999% durability, higher cost. Useful for static website hosting.
* **S3 Standard-Infrequent Access (S3 Standard-IA):** Infrequently accessed data requiring high availability (rapid access), lower storage price but higher retrieval price, stored in a minimum of three AZs, ideal for backups and disaster recovery.
* **S3 One Zone-Infrequent Access (S3 One Zone-IA):** Infrequently accessed data, stores data in a single AZ, lower storage price than S3 Standard-IA, suitable if data can be easily reproduced in case of an AZ failure.
* **S3 Intelligent-Tiering:** Automatically moves data between frequent access (S3 Standard) and infrequent access (S3 Standard-IA) tiers based on access patterns, ideal for data with unknown or changing access, includes a small monitoring fee.
* **S3 Glacier:** Low-cost storage for data archiving, retrieval within minutes to hours, suitable for audit data requiring long-term retention, vaults can be created with **Glacier vault lock policies** for compliance (e.g., WORM - write once/read many).
* **S3 Glacier Deep Archive:** Lowest-cost object storage for archiving, retrieval within 12 hours.

**Lifecycle policies** can be created to automatically move data between different S3 storage tiers based on defined schedules, without requiring application code changes.

Comparisons between EBS and S3 highlight that S3 is web-enabled, regionally distributed with high durability, cost-effective for large amounts of data, and serverless, making it ideal for scenarios like a photo analysis website. However, for tasks involving frequent small changes to large files, like video editing, **EBS** is more efficient due to block-level updates compared to S3's requirement to re-upload the entire object. Ultimately, the best storage class depends on the specific workload.

#### Amazon Elastic File System (Amazon EFS)

**Amazon EFS** is a scalable file system for use with AWS Cloud services and on-premises resources. It automatically grows and shrinks as files are added or removed, scaling on demand to petabytes without application disruption. EFS is designed for scenarios where multiple clients need to access the same data simultaneously, such as shared file systems for analytics. Unlike EBS volumes, which attach to a single EC2 instance within an Availability Zone, EFS is a **regional resource**, allowing multiple instances across different AZs within the same region to access it concurrently. EFS is a true file system for Linux and automatically scales as data is written, eliminating the need for manual provisioning.

### Database-As-A-Service

The document then shifts to **database services**, discussing the need to maintain relationships between data, which often necessitates a **relational database management system (RDBMS)**. Relational databases store data in tables with related attributes, and data is typically queried using **SQL (Structured Query Language)**.

#### Amazon Relational Database Service (Amazon RDS)

**Amazon RDS** enables running relational databases in the AWS Cloud and supports major database engines like Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and Microsoft SQL Server. RDS is a **managed service** that automates tasks like hardware provisioning, database setup, patching, and backups, as well as redundancy, failover, and disaster recovery. This allows users to focus on application innovation rather than administrative tasks. RDS offers security features like encryption at rest and in transit.

**Amazon Aurora** is a cloud-native, enterprise-class relational database compatible with MySQL and PostgreSQL. It claims to be significantly faster and more cost-effective than standard commercial-grade databases. Aurora provides high availability by replicating six copies of data across three AZs, supports up to 15 read replicas for offloading read operations, has continuous backups to S3, and offers point-in-time recovery.

#### Nonrelational databases

**Nonrelational databases**, also known as **NoSQL databases**, use structures other than rows and columns (like tables) to organise data. One common structure is **key-value pairs**, where data is organised into items (keys) with attributes (values). Nonrelational databases offer flexibility as attributes can be added or removed at any time, and not all items need the same attributes.

#### Amazon DynamoDB

**Amazon DynamoDB** is a **key-value** and **document database service** that offers single-digit millisecond performance at any scale. It is a **serverless database**, meaning no underlying infrastructure needs to be managed. Data in DynamoDB is organised into tables, which contain items with attributes. DynamoDB automatically manages storage and scaling and replicates data redundantly across Availability Zones. It is highly performant with millisecond response times and massively scalable, as demonstrated by the volume of requests during Amazon Prime Day. DynamoDB has a flexible schema, unlike rigid SQL database schemas, but does not support complex SQL queries; instead, queries are based on designated key attributes. It's purpose-built for specific use cases requiring high speed and scalability with simpler queries.

Comparisons between RDS and DynamoDB indicate that RDS is suitable for systems requiring complex analysis of data across multiple tables (complex relational joins), such as a supply chain management system. DynamoDB, with its speed and lack of overhead, is better for applications that don't need complex joins and can store data in a single table with flexible attributes, like an employee contact list app. The choice depends on the specific workload.

#### Amazon Redshift

**Amazon Redshift** is a **data warehousing service** for **big data analytics**. It is designed for historical analytics (OLAP) rather than real-time operational analysis (OLTP). Redshift can collect data from various sources to identify relationships and trends. It is massively scalable, handling petabyte-scale data, and can even query exabytes of unstructured data in data lakes using Amazon Redshift Spectrum. Redshift is optimised for business intelligence workloads, offering significantly higher performance for these tasks compared to traditional databases.

#### AWS Database Migration Service (AWS DMS)

**AWS DMS** enables the migration of relational and nonrelational databases and other data stores between source and target databases, which can be of the same or different types. The source database remains operational during migration, minimising downtime. **Homogeneous migration** involves databases of the same type (e.g., MySQL to Amazon RDS for MySQL) and is generally straightforward. **Heterogeneous migration**, involving different database types (e.g., Oracle to PostgreSQL), is a two-step process requiring the use of the **AWS Schema Conversion Tool** to convert the schema and code before migrating the data with DMS. DMS also supports use cases like development and test migration, database consolidation, and continuous replication for disaster recovery.

#### Other database services

The document briefly mentions other specialised database services offered by AWS:

* **Amazon DocumentDB:** A document database service supporting MongoDB workloads, suitable for content management and user profiles.
* **Amazon Neptune:** A graph database service for applications with highly connected datasets, like recommendation engines and fraud detection.
* **Amazon Quantum Ledger Database (Amazon QLDB):** A ledger database service that provides a complete history of all changes to application data.
* **Amazon Managed Blockchain:** A service for creating and managing blockchain networks.
* **Amazon ElastiCache:** A caching service for improving read times by adding caching layers on top of databases, supporting Redis and Memcached.
* **Amazon DynamoDB Accelerator (DAX):** An in-memory cache for DynamoDB to further improve response times.