Sreenivasa Inst. of Technology &
Management Studies

Dr. Visweswaraiah Road,
Bangalore-Tirupathi Bypass Rd,
Murukampattu, Andhra Pradesh 517127

Cloud Computing

Edited by
R. Eswar Reddy

Contents

Chapter 1

Unit 1: Basics of Cloud Computing

1.1 Introduction to Cloud Computing

1.1.1 Introduction to Cloud Computing

Cloud computing is an on-demand delivery of IT resources (compute, storage, databases, networking, etc.) over the Internet with pay-as-you-go pricing. Instead of owning physical data centers, users access services from providers like AWS, Azure, or GCP.

Key benefits include cost savings, scalability, speed, productivity, performance, and reliability.

Cloud computing has embarked a revolution in accessing, provisioning and consumption of the information and computing in the ICT industry. It has emerged as a novel paradigm of high- performance and large-scale computing that actuates relocation of computing and data fromdesktops and personal computers to big data centers. Cloud is a construct (infrastructure) that allows to access application that actually resides at a remote location of another internet connected device, most often, this will be a distant datacenter. Cloud computing takes the technology, services, and applications that are similar to those on the Internet and turns them into a self-service utility (Figure 1). Cloud provides an abstraction based on the notion of pooling physical resources and presenting them as a virtual resource. It is a new model for provisioning resources, for staging applications, and for platform-independent user access to services. "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computer resources (networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" (Figure 2).

Cloud computing refers to manipulating, configuring, and accessing the applications online. It offers online data storage, infrastructure and application and involves both a combination of software and hardware- based computing resources delivered as a network service. Example:Suppose we want to install MS-Word in our organization's computer. We have to bought the CD/DVD of it an install it or can setup a S/W distribution server to automatically install this application on your machine. Every time Microsoft issued a new version, we have to perform the same task. If some other company hosts your application, that is, they handle the cost of servers and manage the software updates. The customers are charged as per their utilization, that is, as per the usage (Figure 3). It reduces the cost of using that software along with the reduction in the cost of installation of heavy servers. Additionally, cloud aids in reducing the cost of electricity bills

Clouds can come in many different types, and services and applications that run on clouds may or may not be delivered by a cloud service provider. The different types and

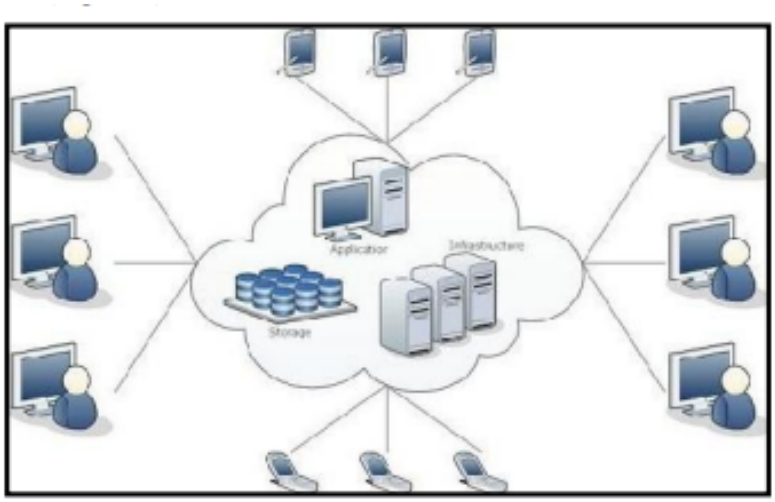


Figure 2: Cloud Scenario

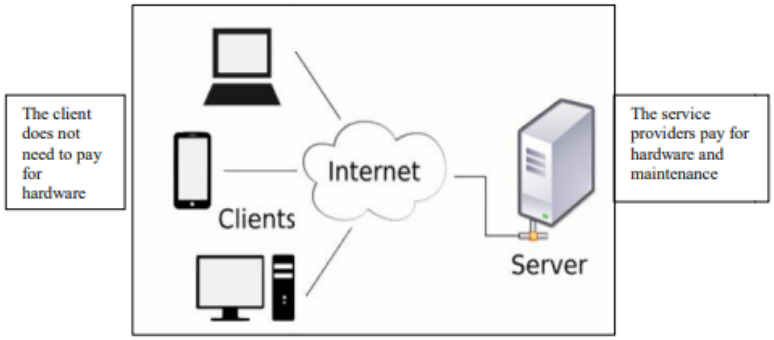


Figure 3: Cloud Client-Server Perspective

levels of cloud services mean that it is important to define what type of cloud computing system you are working with. **Cloud Computing Collaboration** With the growth of the Internet, there was no need to limit group collaboration to a single enterprise's network environment. The users from multiple locations within a corporation, and from multiple organizations, desired to collaborate on projects that crossed company and geographic boundaries. Projects had to be housed in the "cloud" of the Internet, and accessed from any Internet-enabled location. The concept of cloud-based documents and services took wing with the development of large server farms, such as those run by Google and other search companies. Cloud-collaboration is also termed as Internet-based group collaboration.

Who Benefits from Cloud Computing?

The following groups of users particularly benefit from adopting cloud computing:

- **Collaborators:** If you often collaborate with others on group projects, the ability to share and edit documents in real time between multiple users is one of the primary benefits of web-based applications.

  Example: Suppose you're in charge of an upcoming presentation to the senior management of your company. You need to work with the heads of your company's various departments, which happen to be based in a half-dozen locations. Given everyone's busy schedules, it's tough enough to schedule a group conference call. How in the

world can all of you get together to create a cohesive presentation? Just use Google Presentations!

- **Road Warriors:** When you work at one office today, at home the next day, and in another city the next, it's tough to keep track of all your documents and applications. You may end up with one version of a document on your work PC, another on your laptop, and a third on your home PC — and that's if you remember to copy that document and take it with you from one location to the next.

  When you're in the office, you log in to your web-based app and access your stored document. Even if you travel to another city, the same application and document are still available to you.

- **Cost-Conscious Users:** Another group of users who should gravitate to cloud computing are those who are cost conscious. With cloud computing you can save money on both your hardware and software.

  Hardware-wise, there's no need to invest in large hard disks or super-fast CPUs. Because everything is stored and run from the web, you can cut costs by buying a less fully equipped computer.

- **Cost-Conscious IT Departments:** Many corporate IT departments are also becoming enamored of the cloud computing model. Although they might appreciate the software savings we just discussed, for them bigger savings result from having to buy fewer central servers.

- **Users with Increasing Needs:** Hardware-based cost savings also apply to individual computer users. Do you need more hard-disk space to store all your digital photos and MP3 files? You could purchase a new external hard drive, or you could utilize lower-cost (or free) cloud storage instead.

Advantages of Cloud Computing

Cloud computing is an emerging technology that almost every company is switching to from its on-premises technologies. Whether it is public, private, or hybrid, cloud computing has become an essential factor for companies to rise up to the competition. Let us find out why the cloud is so much preferred over on-premises technologies.

- **Cost Efficiency:** The biggest reason behind companies shifting to cloud computing is that it takes considerably less cost than any on-premise technology. Now, companies need not store data in disks anymore as the cloud offers enormous storage space, saving money and resources. CapEx and OpEx costs are reduced because resources are only acquired when needed and are only paid for when used.

- **High Speed:** Cloud computing lets us deploy the service quickly in fewer clicks. This quick deployment lets us get the resources required for our system within minutes.

- **Excellent Accessibility:** Storing information in the cloud allows us to access it anywhere and anytime regardless of the machine, making it a highly accessible and flexible technology of the present times.

- **Back-up and Restore data:** Once data is stored in the cloud, it is easier to get its back-up and recovery, which is quite a time-consuming process in on-premise technology.
- **Manageability:** Cloud computing eliminates the need for IT infrastructure updates and maintenance since the service provider ensures timely, guaranteed, and seamless delivery of our services and also takes care of all the maintenance and management of our IT services according to the service-level agreement (SLA).
- **Sporadic Batch Processing:** Cloud computing lets us add or subtract resources and services according to our needs. So, if the workload is not 24/7, we need not worry about the resources and services getting wasted and we won't end up stuck with unused services.
- **Strategic Edge:** Cloud computing provides a company with a competitive edge over its competitors when it comes to accessing the latest and mission-critical applications that it needs without having to invest its time and money on their installations.

Disadvantages of Cloud Computing

Every technology has both positive and negative aspects that are highly important to be discussed before implementing it.

- **Vulnerability to Attacks:** Storing data in the cloud may pose serious challenges of information theft since in the cloud every data of a company is online. Security breach is something that even the best organizations have suffered from and it's a potential risk in the cloud as well. Although advanced security measures are deployed on the cloud, still storing confidential data in the cloud can be a risky affair.
- **Network Connectivity Dependency:** Cloud computing is entirely dependent on the Internet. This direct tie-up with the Internet means that a company needs to have reliable and consistent Internet service as well as a fast connection and bandwidth to reap the benefits of cloud computing.
- **Downtime:** Downtime is considered as one of the biggest potential downsides of using cloud computing. The cloud providers may sometimes face technical outages that can happen due to various reasons, such as loss of power, low Internet connectivity, data centers going out of service for maintenance, etc. This can lead to a temporary downtime in the cloud service.
- **Vendor Lock-In:** When in need to migrate from one cloud platform to another, a company might face some serious challenges because of the differences between vendor platforms. Hosting and running the applications of the current cloud platform on some other platform may cause support issues, configuration complexities, and additional expenses. The company data might also be left vulnerable to security attacks due to compromises that might have been made during migrations.
- **Limited Control:** Cloud customers may face limited control over their deployments. Cloud services run on remote servers that are completely owned and managed by service providers, which makes it hard for the companies to have the level of control that they would want over their back-end infrastructure.

Characteristics of Cloud Computing (NIST Definition)

- **On-demand self-service**: Users provision resources automatically without human interaction.
- **Broad network access**: Services available over the network (e.g., Internet) via heterogeneous devices.
- **Resource pooling**: The provider's computing resources are pooled to serve multiple users using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Examples of resources include storage, processing, memory, and network bandwidth.
- **Rapid elasticity**: assigned according to consumer demand. Examples of resources include storage, processing, memory, and network bandwidth.
- **Measured service**: Cloud systems automatically control and optimize resource use by leveraging a metering capability appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). The resource usage can be monitored, controlled, and reported, providing transparency for both the provider and user of the service. It follows a "Pay as you grow" modelor for internal IT epartments to provide IT chargeback capabilities. The usage of cloud resources is measured and user is charged based on some metrics such as amount of CPU cycles used, amount of storage space used, number of network I/O requests etc. are used to calculate the usage charges for the cloud resource.
- **Performance:**Dynamic allocation of resources as per the application workloads helps to easily scale up or down and maintain performance.
- **Reduced costs:**Cost benefits for applications as only as much computing and storage resources are required can be provisioned dynamically and upfront investment in purchase of computing assets to cover worst case requirements is avoided.
- **Outsourced Management** Cloud computing allows the users to outsource the IT infrastructure requirements to external cloud providers and save upfront capital investments. This helps in easiness of setting IT infrastructure and pay only for the operational expensesfor the cloud resources used.
- **Multitenancy:** Multitenancy allows multiple users to make use of the same shared resources.Modern applications such as Banking, Financial, Social networking, e-commerce, B2B etc. are deployed in cloud environments that support multi-tenanted applications.
- **Service Oriented Architecture (SOA):** SOAis essentially a collection of services which communicate with each other.SOA provides a loosely-integrated suite of services that can be used within multiple business domains. The approach here is usually implemented by Web service

Cloud Service Models

- **IaaS (Infrastructure as a Service)**: Provides virtualized computing resources (VMs, storage, networks). Examples: AWS EC2, Azure Virtual Machines, Google Compute Engine.
- **PaaS (Platform as a Service)**: Provides runtime environment for applications. Developer focuses on code. Examples: AWS Elastic Beanstalk, Azure App Service, Google App Engine.
- **SaaS (Software as a Service)**: Complete applications delivered over the Internet. Examples: Google Workspace, Microsoft 365, Salesforce.

1.1.2 Cloud Deployment Models

Public Cloud

Cloud Computing mainly has three deployment models: Private, Public, and Hybrid cloud.

A **public cloud** is a cloud deployment model in which cloud resources are offered over the Internet and are open to all users and organizations. The public cloud is one of the most widely adopted deployment models today. Adopting a public cloud is often more cost-effective because third-party providers manage the underlying infrastructure and resources.

A public cloud is a widely used cloud computing model where companies and individuals can access services such as storage, applications, and virtual machines over the Internet. These services are provided by third-party companies and are available to anyone who subscribes or pays for them.

1. **Infrastructure and Ownership**
   - **Provider-Owned Systems:** Major providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) own and manage the entire infrastructure — including data centers, servers, networking equipment, and software.
   - **Shared Resources:** Users share the physical infrastructure (servers, storage, networking) with other customers. This is known as *multi-tenancy*. Data and workloads are securely isolated using virtualization, access controls, and encryption to ensure privacy and security.
2. **Service Delivery** Public cloud services are delivered over the Internet and are classified into three main categories (service models):
   - **Infrastructure as a Service (IaaS):** Provides fundamental computing resources such as virtual machines, storage, and virtual networks. Users have control over operating systems and applications (e.g., AWS EC2, Google Compute Engine, Azure Virtual Machines).
   - **Platform as a Service (PaaS):** Offers a platform allowing customers to develop, run, and manage applications without managing the underlying infrastructure (e.g., Google App Engine, Azure App Service, AWS Elastic Beanstalk).

- **Software as a Service (SaaS):** Delivers fully managed software applications over the Internet on a subscription basis (e.g., Google Workspace, Microsoft 365, Salesforce CRM).

The architecture consists of:

- **Cloud Data Centers** located at multiple geographic regions
- **Organizations** that consume shared cloud resources
- **Individual Users** who also access the same infrastructure

AWS is one of the leading cloud service providers offering a robust public cloud environment with high availability, scalability, and global reach.

- **Global Infrastructure:** AWS operates in multiple geographic **Regions**, each containing multiple **Availability Zones** (isolated data centers).
- **Cost-Effective Pricing:** Pay-as-you-go model with no upfront costs; options include On-Demand, Reserved Instances, and Spot Instances.
- **Nearly Zero Downtime:** Achieved through multi-AZ deployments, load balancing, and auto-scaling.
- **Services Offered:** Compute (EC2, Lambda), Storage (S3, EBS), Database (RDS, DynamoDB), Networking (VPC, Route 53), and many others.
- **Security & Fault Tolerance:** Implements latest encryption standards, IAM, DDoS protection (Shield), and compliance certifications.

Advantages of Public Cloud

- Virtually unlimited scalability — resources can be scaled up/down instantly
- High availability and fault tolerance due to geographically distributed data centers
- Cost efficiency through pay-as-you-go pricing model
- No upfront capital expenditure (CapEx) — infrastructure is maintained by the provider
- Rapid provisioning and deployment of services

Disadvantages of Public Cloud

- Lower level of security and privacy compared to private cloud (shared infrastructure)
- Limited control over the underlying physical infrastructure
- Risk of vendor lock-in when heavily dependent on one provider's proprietary services
- Dependency on a reliable, high-speed Internet connection (or dedicated private links)

Private Cloud

A **Private Cloud** is a dedicated cloud computing environment exclusively used by a single organization. Unlike public clouds where resources are shared among multiple tenants, a private cloud provides a more secure, controlled, and customizable environment. It can be hosted on-premises (in the organization's own data center) or externally by a third-party provider in a dedicated facility.

Private clouds allow organizations to enjoy cloud benefits such as scalability, flexibility, and virtualization while maintaining full control over data, security policies, and compliance requirements.

Private cloud deployments are classified based on location and management responsibility.

1. On-Premises Private Cloud Hosted entirely within the organization's own data center. The organization owns and manages the hardware, software, networking, and security.

Key Characteristics:

- Complete control over infrastructure and data
- Highest level of security and customization
- Ideal for highly regulated industries

2. Externally Hosted (Managed) Private Cloud A third-party provider hosts and manages the infrastructure in their data center, but resources are dedicated exclusively to one organization (single-tenant).

Key Characteristics:

- No need to invest in physical hardware
- Provider handles maintenance, updates, and physical security
- Organization retains control over applications, data, and policies

3. Virtual Private Cloud (VPC) A logically isolated section of a public cloud provider's infrastructure dedicated to a single organization. It combines public cloud scalability with private cloud-like isolation.

Example: AWS Virtual Private Cloud (VPC), Azure Virtual Network, Google Cloud VPC.

Challenges and Considerations in Adopting Private Cloud

- **Cost:** High upfront capital expenditure (CapEx) for hardware, software, and setup. Ongoing maintenance adds to operational costs (OpEx). *Tip:* Perform detailed TCO (Total Cost of Ownership) analysis; consider hybrid models to reduce initial investment.
- **Maintenance:** Requires significant in-house IT expertise or managed service providers for patching, monitoring, and upgrades. *Tip:* Partner with managed service providers to offload routine tasks.
- **Scalability:** Scaling requires purchasing and installing new hardware, which is slower than public cloud auto-scaling. *Tip:* Use hybrid cloud bursting to leverage public cloud during peak demand.

- **Compliance and Security:** While easier to achieve compliance, organizations must implement and audit security measures themselves. *Tip:* Establish strong governance, regular audits, and encryption policies.
- **Skill Requirements:** Needs advanced IT staff or external support.
- **Vendor Lock-In:** Risk if using proprietary technologies.
- **Performance:** Can offer low latency but depends on internal network quality.

Benefits of Private Cloud

- **Enhanced Security:** Exclusive use reduces risk of data exposure from other tenants.
- **Customization:** Full control to tailor hardware, software, networking, and security to exact business needs.
- **Compliance & Regulatory Control:** Easier to implement and prove compliance with GDPR, HIPAA, PCI-DSS, etc.
- **Performance & Reliability:** Dedicated resources ensure predictable performance, low latency, and high reliability.
- **Long-term Cost Efficiency:** Optimized resource usage can reduce costs over time despite higher initial investment.

Hybrid Cloud

A **hybrid cloud** combines public and private cloud services, giving businesses the best of both worlds — scalability from public clouds and security from private clouds. This setup allows companies to store sensitive data privately while using cost-effective public cloud resources for non-sensitive workloads.

For example, in software development, teams often use a public cloud to host projects and a private cloud for secure testing. This approach offers flexibility, cost savings, and better control over data and applications.

In a hybrid cloud, public and private cloud environments are connected, enabling seamless data movement, application portability, and workload orchestration. The public cloud typically handles bursty or scalable workloads, while the private cloud manages sensitive or regulated data.

During events like the COVID-19 pandemic, many organizations adopted hybrid cloud strategies to securely handle sensitive information while leveraging public cloud scalability for remote operations and virtual services.

1.1.3 How to Set Up a Hybrid Cloud

1. Define your hybrid cloud goals
   Identify which workloads stay private (sensitive data, compliance) and which move to public (scalable apps, testing, AI/ML).

2. Choose the right cloud providers
   Public: AWS, Azure, Google Cloud, IBM Cloud, Oracle Cloud.
   Private: On-premises, VMware, OpenStack.
   Ensure compatibility and integration capabilities.

3. Set up secure cloud connectivity
   Use dedicated links: AWS Direct Connect, Azure ExpressRoute, Google Cloud Interconnect.
   Implement VPN or SD-WAN for secure, low-latency data transfer.

4. Manage access & security
   Deploy IAM, RBAC, SSO, MFA.
   Use encryption, threat monitoring, and compliance tools (GDPR, HIPAA, PCI-DSS).

5. Use hybrid cloud management tools
   Containers: Kubernetes, Docker.
   Automation: Terraform, Ansible.
   Monitoring: AWS CloudWatch, Azure Monitor, Google Cloud Operations.

6. Optimize workloads & automate
   Implement auto-scaling, AI-driven resource allocation, and cost optimization.

7. Implement Backup & Disaster Recovery
   Use redundant backups and real-time failover mechanisms.

8. Monitor, Optimize & Improve
   Regularly analyze usage, costs, security, and compliance.

1.1.4 Top Cloud Service Providers for Hybrid Cloud

Cloud Provider	Key Features	Best For	Notable Services
AWS	Largest ecosystem, extensive services, strong support	Enterprises, Startups	EC2, S3, Lambda, RDS, CloudFront
Microsoft Azure	100+ services, multi-language support	SMBs, Enterprises	Virtual Machines, AKS, Cosmos DB, AI/ML
Oracle Cloud	Bare metal servers, strong database focus	Database-heavy, Enterprises	OCI, Autonomous Database
IBM Cloud	AI-driven, DevOps support	AI/ML, Enterprise	Watson, Kubernetes, Blockchain
Google Cloud	Strong data analytics, AI	Data-intensive workloads	BigQuery, Compute Engine, Kubernetes

Table 1.1: Comparison of Major Cloud Providers Supporting Hybrid Deployments

1.1.5 Advantages of Hybrid Cloud

- **Flexibility** — Use both public and private environments based on workload needs
- **Security** — Sensitive data remains in private cloud

- **Cost Effectiveness** — Pay only for public cloud usage; lower overall cost than full private cloud
- **Risk Management** — Better handling of failures, compliance, and security risks
- **Accessibility** — Global resource access with optimized performance

1.1.6 Disadvantages of Hybrid Cloud

- **Complexity** — Managing two environments increases operational complexity
- **Network Dependency** — Requires reliable, high-speed connectivity between clouds
- **Visibility Issues** — Harder to monitor and manage across multiple environments
- **Implementation Time** — Setup and integration take significant time and expertise
- **Security Risks** — Data movement between clouds must be carefully secured
- **Private Cloud**: Dedicated to a single organization (on-premises or hosted).
- **Hybrid Cloud**: Combines public and private for flexibility (e.g., sensitive data private, burst workloads public).
- **Community Cloud**: Shared among organizations with common concerns.

Cloud-Based Services and Applications Examples

E-commerce (Amazon), streaming (Netflix on AWS), collaboration (Google Docs), AI/ML workloads. Cloud Service Providers (CSPs) offer a wide variety of cloud services today. Cloud computing has penetrated almost every sector by providing diverse cloud-based applications. The easy sharing and management of resources has made cloud computing one of the dominant fields in modern computing. Several key properties have established it as an active component across different domains.

1. **Online Data Storage**
   Cloud computing enables storage of files, images, audio, videos, and other data on remote cloud storage servers. Organizations no longer need to invest in expensive physical storage infrastructure for large volumes of business data. As technological growth leads to exponential data generation, traditional storage becomes problematic. Cloud storage provides scalable, on-demand space with anytime, anywhere access.
   Examples: Google Drive, Dropbox, iCloud, OneDrive.

2. **Backup and Recovery**
   Cloud vendors ensure data security and provide automated backup facilities. They offer various recovery applications to retrieve lost or corrupted data easily. In traditional systems, backup processes are complex, time-consuming, and data recovery is often difficult or impossible. Cloud computing simplifies backup and recovery with no risk of running out of backup media or permanent data loss.

3. **Big Data Analysis**
   The massive volume of big data makes it impossible to store and manage using traditional systems. Cloud computing solves this by offering virtually unlimited storage without physical infrastructure concerns. It also provides powerful analytics tools to process raw data and extract valuable insights. High-quality data analytics tools are available on-demand in the cloud.

4. **Anti-Virus Applications**
   Traditionally, organizations and individuals install antivirus software locally to protect against cyber threats. Now, cloud-based antivirus solutions monitor systems remotely from the cloud. These services detect threats, apply fixes automatically, and sometimes allow local software downloads. This approach reduces local resource usage and provides centralized, up-to-date protection.

5. **E-commerce Applications**
   Cloud-based e-commerce platforms enable businesses to respond quickly to market opportunities and challenges. Customer data, product catalogs, inventory, and operational systems are managed in the cloud. This reduces infrastructure costs and setup time, allowing faster business scaling and innovation.

6. **Cloud Computing in Education**
   Cloud computing has transformed education by enabling e-learning platforms, online distance learning, student information systems, and collaborative tools. It creates an attractive, flexible environment for students, teachers, and researchers to connect, access resources, and share knowledge from anywhere.

7. **Technology-enhanced Learning / Education as a Service (EaaS)**
   Cloud platforms offer specialized education services, including:

   - **Google Apps for Education** — Free web-based email, calendar, documents, and collaborative tools widely used in schools and universities.
   - **Chromebooks for Education** — Google-designed devices to enhance classroom innovation and digital learning.
   - **Tablets with Google Play for Education** — Allows educators to quickly deploy modern technology solutions in classrooms.

8. **Testing and Development**
   Setting up development environments, performing various types of testing, and deploying applications traditionally requires significant IT infrastructure. Cloud computing provides scalable, flexible, and cost-effective resources for development, testing, and deployment, reducing expenses and accelerating time-to-market.

9. **E-Governance Applications**
   Governments can leverage cloud services to modernize public administration, improve service delivery, increase scalability, and reduce costs. Cloud helps shift from traditional manual processes to efficient, digital, citizen-centric governance while optimizing budget allocation for public welfare.

10. **Cloud Computing in Medical Fields**
    Healthcare organizations use cloud platforms to store and access patient records, medical images, lab reports, and treatment data securely over the internet. It enables real-time sharing among doctors, specialists, labs, ambulances, and emergency services without physical infrastructure.
    Examples: Telehealth platforms, Medi-Cloud solutions, technology-enhanced healthcare systems.

11. **Entertainment Applications**
    Cloud computing reaches diverse audiences through on-demand entertainment (ODE). Industries adopt multi-cloud strategies to deliver online music, video streaming, online games, and video conferencing across devices (TVs, mobiles, set-top boxes).

    - **Online Games:** Cloud gaming services (Shadow, GeForce Now, Vortex, Project xCloud, PlayStation Now) run games remotely.
    - **Video Conferencing:** Cloud-based apps reduce costs, improve efficiency, and enable seamless communication.

12. **Art Applications**
    Cloud-based design tools help users create attractive cards, booklets, posters, and images easily. Popular platforms include:

    - Moo — Business cards, postcards, mini cards design & printing.
    - Vistaprint — Marketing materials, invitations, business cards.
    - Adobe Creative Cloud — Professional suite (Photoshop, Illustrator, InDesign, etc.) for designers and creatives.

13. **Management Applications**
    Cloud management tools help administrators handle resource deployment, data integration, disaster recovery, and overall platform control. Examples:

    - Toggl — Time tracking for projects.
    - Evernote — Note syncing across devices.
    - Outright — Real-time financial tracking (income, expenses, profit/loss).
    - GoToMeeting — Video conferencing and online meetings.

14. **Social Applications**
    Social networking platforms rely heavily on cloud infrastructure for storage, scalability, and real-time features. Examples:

    - Facebook — Photo/video sharing, status updates, notifications.
    - Twitter (now X) — Microblogging, short posts (tweets), news & following.
    - LinkedIn — Professional networking for students, freshers, and professionals.

In the future, cloud computing is expected to penetrate even more sectors by continuously introducing innovative applications and services.

1.1.7 Cloud Concepts and Technologies

Virtualization

Virtualization creates virtual versions of hardware resources using hypervisors (Type-1 bare-metal or Type-2 hosted). Enables multi-tenancy and efficient resource use.

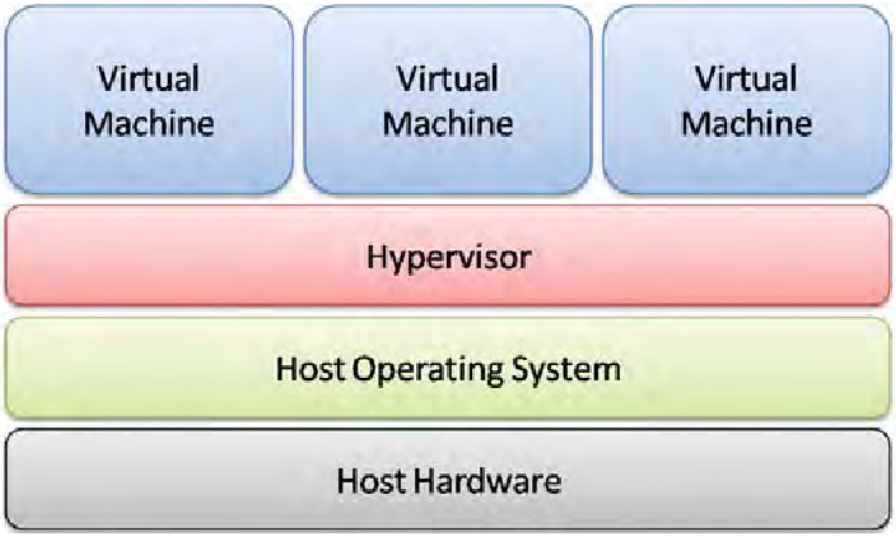
Virtualization in cloud computing helps create virtual versions of hardware, such as desktop computers, with a virtual ecosystem of operating systems, storage, memory, and networking services. The virtualization architecture uses the same hardware to run multiple operating systems on the same machine and optimize their performance.

What is Virtualization?

Virtualization plays an important role and function in cloud computing. It helps in reducing the space or costs associated with the investment. This technology allows end users to run multiple desktop operating systems and applications simultaneously on the same hardware and software.

Virtualization in cloud computing simplifies the creation of virtual machines and makes it easier to run multiple machines. It also helps create a virtual ecosystem of server operating systems, multiple storage facilities, and multiple operating systems.

Cloud computing is an application or service associated with a virtual ecosystem. Such ecosystems can be public or private. Due to virtualization, the need for physical infrastructure can be reduced. The terms cloud computing and virtualization are now used interchangeably and are rapidly converging.



- Hosted Architecture: In this type of configuration, first, the host operating system is installed on the hardware, then the software is installed. The software is a hypervisor or virtual machine (VM) that requires many guest operating systems or VMs to be installed on the hardware to set up the virtualization architecture. Once the hypervisor is in place, applications can be installed and run on the virtual machine as if they were installed on the physical machine.

- Bare Metal Architecture: In this architecture, the hypervisor is installed directly on the hardware, not on top of the operating system. Hypervisors and virtual machines are configured the same way as infrastructure. Bare metal virtualization architecture is designed for applications that provide real-time access or perform some form

of data processing. Virtualization is hypervisor based. A hypervisor separates the operating system and applications from the underlying computer hardware so that the host computer can run multiple virtual machines as guests and share physical resources such as network processor, memory space, and network bandwidth. A hypervisor allocates memory or storage services and distributes some of these services to each virtual machine according to the needs of the virtualization architecture.

More about Hypervisors

Hypervisors are the core software/firmware layer responsible for creating, managing, and running multiple virtual machines (VMs) on a single physical host. They provide virtualization by abstracting and allocating hardware resources (CPU, memory, storage, network) to guest operating systems.

There are two main types of hypervisors:

- **Type 1 Hypervisors (Bare-Metal / Native Hypervisors)**

  Type 1 hypervisors run directly on the host hardware without any underlying host operating system. They are installed directly on the physical server (bare metal), which gives them direct access to hardware resources.

  Key Characteristics:

  - High performance due to no additional OS layer
  - Better resource utilization and efficiency
  - Excellent scalability and stability
  - Greater security (smaller attack surface)
  - Typically used in enterprise data centers, cloud providers, and production environments
  - Require dedicated hardware and specialized management tools

  Examples:

  - VMware ESXi
  - Microsoft Hyper-V (in standalone/server mode)
  - Citrix Hypervisor (formerly XenServer)
  - KVM (Kernel-based Virtual Machine) with appropriate configuration
  - Oracle VM Server for x86 (based on Xen)
  - Proxmox VE (based on KVM)

- **Type 2 Hypervisors (Hosted Hypervisors)**

  Type 2 hypervisors run as an application on top of an existing host operating system (Windows, macOS, Linux, etc.). The host OS manages the hardware, and the hypervisor runs like any other software.

  Key Characteristics:

  - Easier to install and use (runs like a normal application)
  - Ideal for developers, testers, students, and personal use

- Simpler setup and management on desktop/laptop systems
- Lower performance compared to Type 1 (due to extra OS layer)
- Potential security risks (vulnerabilities in host OS can affect VMs)
- Limited scalability and resource efficiency

Examples:

- VMware Workstation Pro
- VMware Fusion (for macOS)
- Oracle VirtualBox
- Parallels Desktop (for macOS)
- QEMU (with graphical frontends)
- Oracle VM VirtualBox

Load Balancing

Load balancing is an essential technique used in cloud computing to optimize resource utilization and ensure that no single resource is overburdened with traffic. It is a process of distributing workloads across multiple computing resources, such as servers, virtual machines, or containers, to achieve better performance, availability, and scalability.

In cloud computing, load balancing can be implemented at various levels, including the network layer, application layer, and database layer. The most common load balancing techniques used in cloud computing are:

- **Network Load Balancing:** This technique is used to balance the network traffic across multiple servers or instances. It is implemented at the network layer and ensures that the incoming traffic is distributed evenly across the available servers.
- **Application Load Balancing:** This technique is used to balance the workload across multiple instances of an application. It is implemented at the application layer and ensures that each instance receives an equal share of the incoming requests.
- **Database Load Balancing:** This technique is used to balance the workload across multiple database servers. It is implemented at the database layer and ensures that the incoming queries are distributed evenly across the available database servers.

Load balancing helps to improve the overall performance and reliability of cloud-based applications by ensuring that resources are used efficiently and that there is no single point of failure. It also helps to scale applications on demand and provides high availability and fault tolerance to handle spikes in traffic or server failures.

Sure, here are some advantages and disadvantages of load balancing in cloud computing:

Advantages:

- **improved Performance:** Load balancing helps to distribute the workload across multiple resources, which reduces the load on each resource and improves the overall performance of the system.

- **High Availability:** Load balancing ensures that there is no single point of failure in the system, which provides high availability and fault tolerance to handle server failures.
- **Scalability:** Load balancing makes it easier to scale resources up or down as needed, which helps to handle spikes in traffic or changes in demand.
- **Efficient Resource Utilization:** Load balancing ensures that resources are used efficiently, which reduces wastage and helps to optimize costs.

Disadvantages:

- **Complexity:** Implementing load balancing in cloud computing can be complex, especially when dealing with large-scale systems. It requires careful planning and configuration to ensure that it works effectively.
- **Cost:** Implementing load balancing can add to the overall cost of cloud computing, especially when using specialized hardware or software.
- **Single Point of Failure:** While load balancing helps to reduce the risk of a single point of failure, it can also become a single point of failure if not implemented correctly.
- **Security:** Load balancing can introduce security risks if not implemented correctly, such as allowing unauthorized access or exposing sensitive data.

Overall, the benefits of load balancing in cloud computing outweigh the disadvantages, as it helps to improve performance, availability, scalability, and resource utilization. However, it is important to carefully plan and implement load balancing to ensure that it works effectively and does not introduce additional risks.

Cloud load balancing is defined as the method of splitting workloads and computing properties in a cloud computing. It enables enterprise to manage workload demands or application demands by distributing resources among numerous computers, networks or servers. Cloud load balancing includes holding the circulation of workload traffic and demands that exist over the Internet. As the traffic on the internet growing rapidly, which is about 100% annually of the present traffic. Hence, the workload on the server growing so fast which leads to the overloading of servers mainly for popular web server. There are two elementary solutions to overcome the problem of overloading on the servers-

First is a single-server solution in which the server is upgraded to a higher performance server. However, the new server may also be overloaded soon, demanding another upgrade. Moreover, the upgrading process is arduous and expensive.Second is a multiple-server solution in which a scalable service system on a cluster of servers is built. That's why it is more cost effective as well as more scalable to build a server cluster system for network services.

Load balancing is beneficial with almost any type of service, like HTTP, SMTP, DNS, FTP, and POP/IMAP. It also rises reliability through redundancy. The balancing service is provided by a dedicated hardware device or program. Cloud-based servers farms can attain more precise scalability and availability using server load balancing. Load balancing solutions can be categorized into two types –

Software-based load balancers: Software-based load balancers run on standard hardware (desktop, PCs) and standard operating systems.

Hardware-based load balancer: Hardware-based load balancers are dedicated boxes which include Application Specific Integrated Circuits (ASICs) adapted for a particular use. ASICs allows high speed promoting of network traffic and are frequently used for transport-level load balancing because hardware-based load balancing is faster in comparison to software solution.

Scalability, and Elasticity

Scalability and elasticity are two fundamental characteristics that distinguish cloud computing from traditional IT infrastructure. They enable cloud systems to handle varying workloads efficiently, cost-effectively, and with minimal manual intervention.

What is Scalability?

Scalability refers to a system's ability to handle increased load (or growth) by adding resources in a way that maintains or improves performance. Cloud scalability is the ability of cloud computing infrastructure to dynamically adjust resources to meet changing demand. This means that as your business grows, your cloud environment can quickly adapt to provide additional resources, such as processing power, storage, or bandwidth, without the need for costly hardware upgrades or infrastructure changes.

Cloud scalability offers a number of key benefits, including improved flexibility, better cost control, increased performance, and enhanced reliability. It allows you to scale your IT resources up or down quickly in response to changing business needs, which can help you optimize your operations, minimize downtime, and improve the overall performance of your IT infrastructure.

Overall, cloud scalability is a crucial aspect of cloud computing and an essential tool for any organization looking to manage its IT resources effectively.

There are three main types of cloud scalability: horizontal scalability, vertical scalability, and hybrid scalability. Each type offers unique benefits and is designed to meet different business needs.

There are two primary types of scalability:

- **Vertical Scalability (Scaling Up/Down)**
  Increasing or decreasing the capacity of an existing resource (e.g., adding more CPU, RAM, or storage to a single server or virtual machine).
  Advantages: Simpler to implement; no need to change application architecture.
  Disadvantages: Limited by the maximum capacity of a single machine; single point of failure; expensive at high scale.

- **Horizontal Scalability (Scaling Out/In)**
  Adding or removing instances (servers, containers, VMs) to distribute the load across multiple resources.
  Advantages: Virtually unlimited scaling; better fault tolerance; cost-effective (use commodity hardware).
  Disadvantages: Requires stateless or distributed application design; more complex to manage.

In cloud computing, horizontal scalability is preferred and is a key enabler for handling massive growth (e.g., Netflix, Amazon during Black Friday sales).

What is Elasticity?

Elasticity is the ability of a cloud system to automatically and dynamically scale resources up or down in real-time based on actual demand, ensuring optimal performance and cost efficiency.

Key features of elasticity:

- **Automatic scaling** — Triggered by metrics such as CPU utilization, request rate, memory usage, or custom business metrics.
- **Rapid provisioning and de-provisioning** — Resources can be added or removed in minutes or seconds.
- **Pay only for what you use** — No need to over-provision for peak loads.

Elasticity is often described as "elasticity = scalability + automation."

Scalability vs Elasticity – Key Differences

Aspect	**Scalability**	**Elasticity**
Definition	Ability to handle growth by adding resources	Ability to automatically adjust resources to match demand
Timing	Planned or manual scaling	Real-time, automatic scaling
Resource Adjustment	Can be up/down or out/in	Dynamic up and down (bidirectional)
Automation Level	May require manual intervention	Fully automated (via autoscaling policies)
Cost Efficiency	Requires forecasting and overprovisioning	Pay-per-use with minimal waste
Use Case Example	Adding servers before a product launch	Automatically scaling web servers during a viral marketing campaign

Table 1.2: Comparison: Scalability vs Elasticity in Cloud Computing

How Cloud Providers Enable Scalability and Elasticity

Major cloud platforms provide built-in tools to achieve both:

- **AWS**
  - Auto Scaling Groups
  - Elastic Load Balancing (ELB)
  - AWS Lambda (serverless automatic scaling)
- **Microsoft Azure**
  - Virtual Machine Scale Sets
  - Azure Autoscale

- Azure Functions

- **Google Cloud**
  - Managed Instance Groups
  - Autoscaler
  - Cloud Run (serverless)

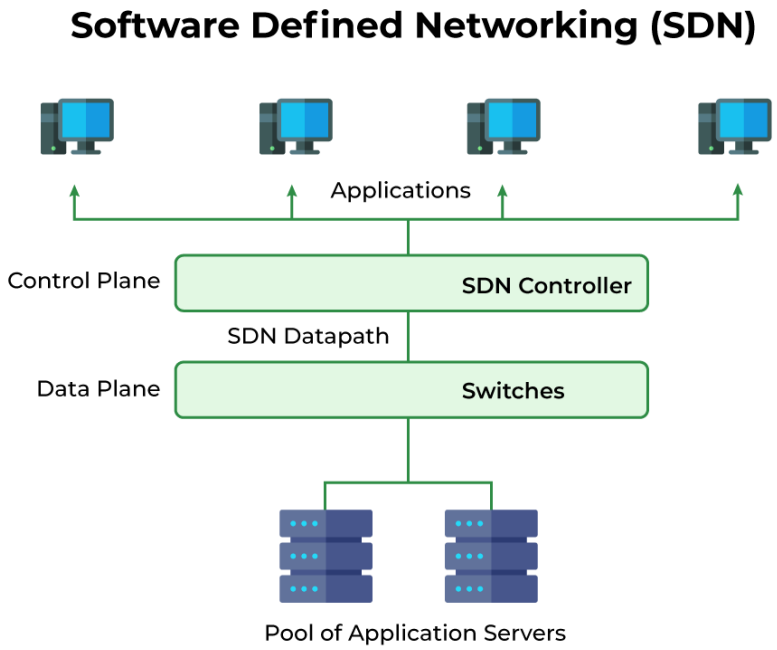
Benefits of Scalability and Elasticity

- Handle unpredictable traffic without downtime
- Reduce costs by avoiding over-provisioning
- Improve performance and user experience
- Enable rapid innovation and business growth
- Support global distribution with low latency

1.1.8 Software Defined Networking (SDN)

Software Defined Networking (SDN) is an approach to network management that enables dynamic, programmatically efficient network configuration in order to improve network performance and monitoring. It represents a new way of managing computer networks that makes them easier and more flexible to control.

In traditional networks, the hardware (routers and switches) makes decisions about how data moves through the network. SDN changes this by separating the **control plane** (which decides where traffic is sent) from the **data plane** (which forwards packets to the selected destination). The control plane is centralized in software, while the data plane remains in the hardware.



In traditional networks, each switch has its own control plane and data plane. Switches exchange topology information to build forwarding tables. In SDN, the control plane is removed from the switches and centralized in an SDN controller. This allows network administrators to manage the entire network from a single console instead of configuring individual devices.

The data plane remains in the switches and forwards packets based on flow tables programmed by the controller. If a packet does not match any flow entry, the switch forwards it to the controller, which installs a new flow rule.

A typical SDN architecture consists of three layers:

- **Application Layer**
  The application layer consists of the SDN Applications, this consists of all the network applications that are used by the organization, which includes the firewalls, load balancers and intrusion detection system. For a traditional network a physical device might be used to manage all of this but the SDN allows us to make use of an application to oversee and manage their working.

- **Control Layer**
  The control layer consists of a SDN controller application. It manages all the policies in the network along with the flow of network traffic.

- **Infrastructure Layer (Data Plane)**
  The infrastructure layer as the name suggests consists of all the physical equipment required by the network such as the switches[2].

Communication occurs via:

- **Northbound APIs** — Between application layer and control layer
- **Southbound APIs** — Between control layer and infrastructure layer (most commonly OpenFlow)

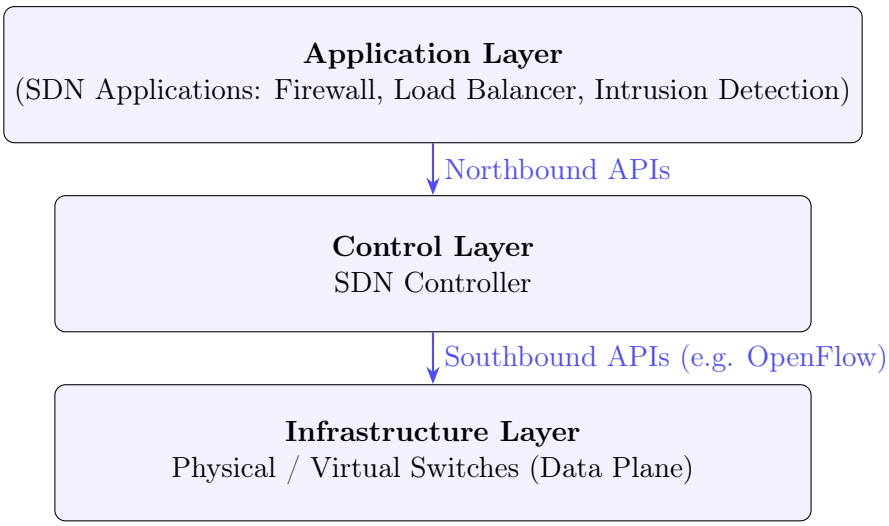


Figure 1.1: Typical Three-Layer SDN Architecture

Data Plane vs Control Plane

- **Data Plane**
  Handles actual user data packets. Activities include:
  - Packet forwarding
  - Segmentation and reassembly
  - Packet replication (multicast)
- **Control Plane**
  The "brain" of the network. Activities include:
  - Building routing tables
  - Setting packet handling policies
  - Topology discovery
  - Path computation

Components of SDN

- **SDN Applications** — This acts as a bridge between the network resources, devices and the SDN controller through a Northbound API.
- **SDN Controller** — The SDN controller forwards the requirements of the SDN Application to the SDN Datapaths. This also contains all the network policies that might be required by the SDN Applications. It also provides them with a view of the network and the network traffic. The controller communicates with the SDN Datapaths/ Dataplane via the Southbound API.
- **SDN Datapaths** — It provides the data packets a path to move on the network by implementing switches.
- **SDN API** — It provides the data packets a path to move on the network by implementing switches.

Where is SDN Used?

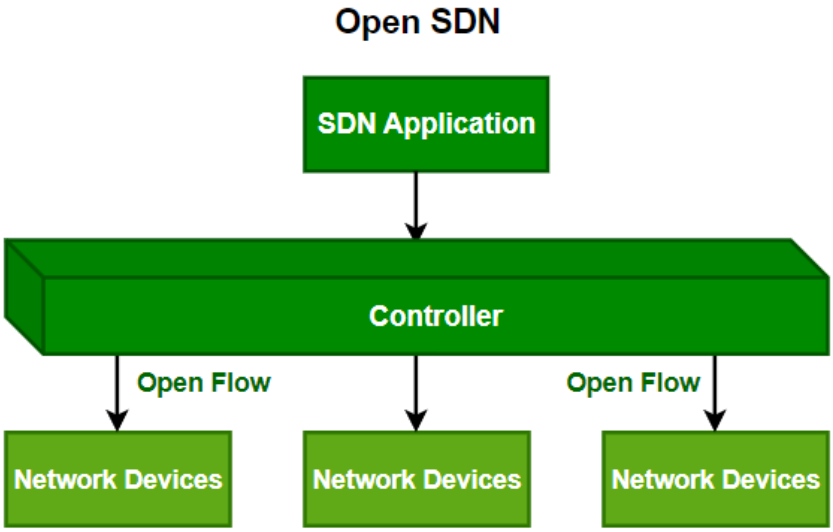
- Enterprise networks — Faster application deployment, centralized management, lower operating costs
- Cloud data centers — White-box switching, reduced CAPEX/OPEX, dynamic provisioning
- Service providers — Traffic engineering, network slicing, 5G core networks
- Research and testbeds — Experimentation with new protocols

Why is SDN Important?

- Better network connectivity and faster data sharing
- Accelerated deployment of new applications and services
- Improved security through better visibility and policy enforcement
- Centralized control with high speed and programmability
- Reduced complexity in large-scale networks

Different Models of SDN

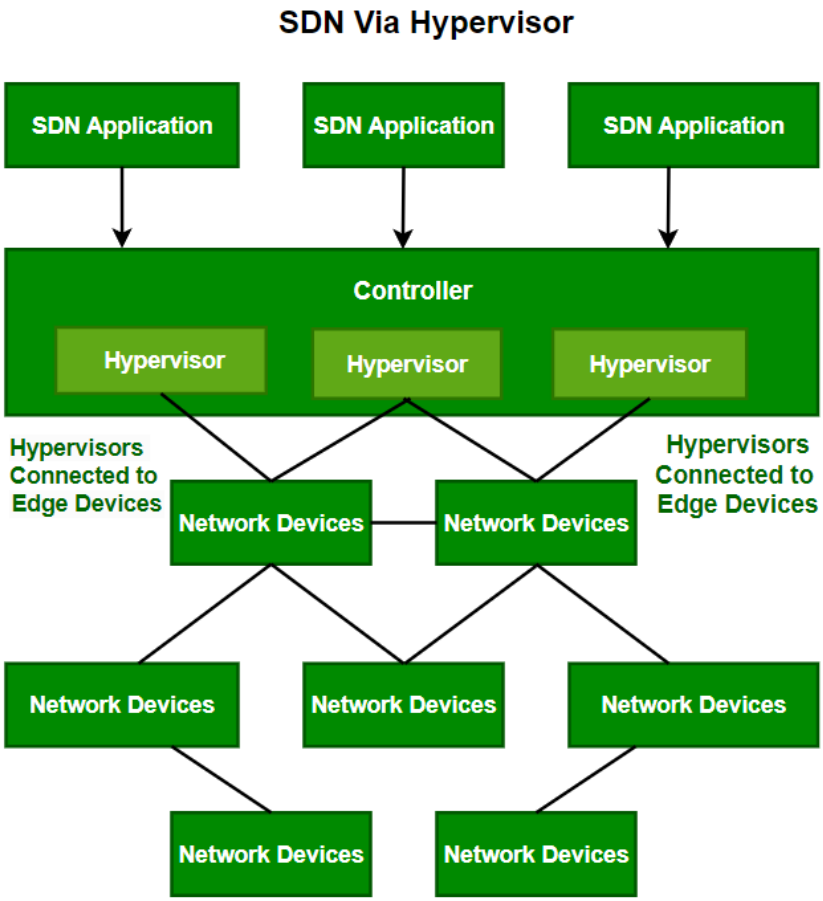
1. **Open SDN**
   Uses OpenFlow protocol for southbound communication. The most straightforward and widely referenced SDN model.



2. **SDN via Hypervisor-based Overlay Network**
   In SDN via the hypervisor, the configuration of physical devices is unchanged. Instead, Hypervisor based overlay networks are created over the physical network. Only the devices at the edge of the physical network are connected to the virtualized networks, thereby concealing the information of other devices in the physical network.

3. **SDN via Hypervisor-based Overlay Networks**
   Creates virtual overlay networks (e.g., VXLAN, NVGRE) on top of existing physical infrastructure. Physical underlay remains unchanged.

4. **Hybrid SDN**
   Hybrid Networking is a combination of Traditional Networking with software-defined networking in one network to support different types of functions on a network.



Advantages of SDN

Software Defined Networking (SDN) offers several significant advantages over traditional networking approaches:

- **Programmability and Centralized Management**
  The network becomes fully programmable. Network behavior can be easily modified, updated, or reconfigured through the centralized SDN controller rather than manually configuring each individual switch or router.

- **Cost Reduction through Simplified Hardware**
  Switch hardware becomes significantly cheaper because switches only need to implement the data plane (forwarding engine). The complex control logic is moved to the centralized controller software, reducing the cost and complexity of network devices.

- **Vendor Independence and Hardware Abstraction**
  The controller abstracts the underlying hardware, allowing network applications and policies to be written independently of specific switch vendors. This promotes multi-vendor environments and reduces vendor lock-in.

- **Improved Security and Threat Response**
  The centralized controller has a global view of the entire network. It can monitor traffic in real time, detect anomalies, and quickly deploy security policies (e.g., reroute suspicious traffic, drop malicious packets, isolate infected segments, or enforce micro-segmentation).

- **Faster Innovation and Service Deployment**
  New network services (firewalls, load balancers, traffic engineering, QoS policies) can be deployed rapidly as software applications on the controller rather than requiring new hardware or firmware upgrades.

- **Better Resource Utilization and Traffic Engineering**
  Dynamic path computation and load balancing become easier, leading to more efficient use of network bandwidth and reduced congestion.

- **Simplified Network Management and Troubleshooting**
  Administrators manage the entire network from a single logical point instead of hundreds or thousands of devices individually.

Disadvantages of SDN

Despite its advantages, SDN also introduces some important challenges and limitations:

- **Single Point of Failure (Controller Dependency)**
  The centralized SDN controller becomes a critical component. If the controller fails, becomes overloaded, or is compromised (e.g., DDoS attack, software bug, corruption), the entire network can be severely affected or completely disrupted.

- **Limited Maturity and Large-Scale Experience**
  While SDN has been successfully deployed in many data centers and service provider networks, large-scale, production-grade deployments across very complex enterprise or carrier networks are still not fully standardized, widely tested, or universally understood.

- **Security Risks of Centralization**
  The controller is an attractive target for attackers. A successful attack on the controller could give an adversary control over the entire network.

- **Performance Overhead**
  The first packet of every new flow must be sent to the controller (packet-in), which can introduce latency in flow setup (especially in very large or high-churn networks).

- **Complexity in Transition and Interoperability**
  Migrating from legacy networks to SDN can be complex. Hybrid deployments (traditional + SDN) require careful planning to avoid inconsistencies.

- **Skill Gap**
  Network engineers need new skills in programming, SDN controllers, OpenFlow, APIs, and automation tools.

1.1.9 Network Functions Virtualization (NFV)

Network Functions Virtualization (NFV) is a technology that transforms traditional network functions—such as routing, load balancing, firewalls, intrusion detection, and VPN gateways—into software-based services (Virtual Network Functions or VNFs) that run on standard, off-the-shelf servers instead of proprietary, specialized hardware appliances.

By decoupling network functions from dedicated hardware, NFV enables network operators and service providers to deploy, scale, update, and manage services much more quickly, flexibly, and cost-effectively.

Key Benefits at a Glance:

- Replaces expensive dedicated hardware with software running on commodity servers
- Enables rapid deployment and scaling of network services (hours instead of months)
- Significantly reduces capital expenditure (CapEx) and operational expenditure (OpEx)
- Improves operational flexibility and simplifies lifecycle management
- Supports automation and orchestration through centralized management systems

NFV Architecture

The architecture of Network Functions Virtualization (NFV) is designed to replace specialized networking hardware with software-based network functions that run on virtualized environments, such as virtual machines (VMs) or containers. This layered architecture enables efficient use of generic computing resources while maintaining the performance, scalability, and manageability required for modern network services. NFV architecture is structured around three main components that work together to deliver, manage, and orchestrate virtual network functions.

1. **Virtualized Infrastructure (NFVI – Network Functions Virtualization Infrastructure)**
   Provides the compute, storage, and networking resources needed to host VNFs. This is abstracted using hypervisors for VMs or container platforms for cloud-native deployments.
   Typically uses hypervisors (e.g., KVM, VMware ESXi) for virtual machines or container platforms (e.g., Docker, Kubernetes) for cloud-native deployments.

2. **Virtual Network Functions (VNFs)**
   Software implementations of network functions such as routing, switching, firewalls, load balancers, and VPNs. VNFs can be deployed, scaled, or updated independently.
   Examples: virtual routers, virtual firewalls, virtual load balancers, virtual WAN optimizers, virtual IDS/IPS, virtual EPC (Evolved Packet Core) components.

3. **Management and Orchestration (MANO)**
   The central framework responsible for lifecycle management of VNFs, resource allocation, automation, and overall coordination of the NFV environment. It includes NFV Orchestrator (NFVO), VNF Manager (VNFM), and Virtualized Infrastructure Manager (VIM).

   Consists of three main functional blocks:

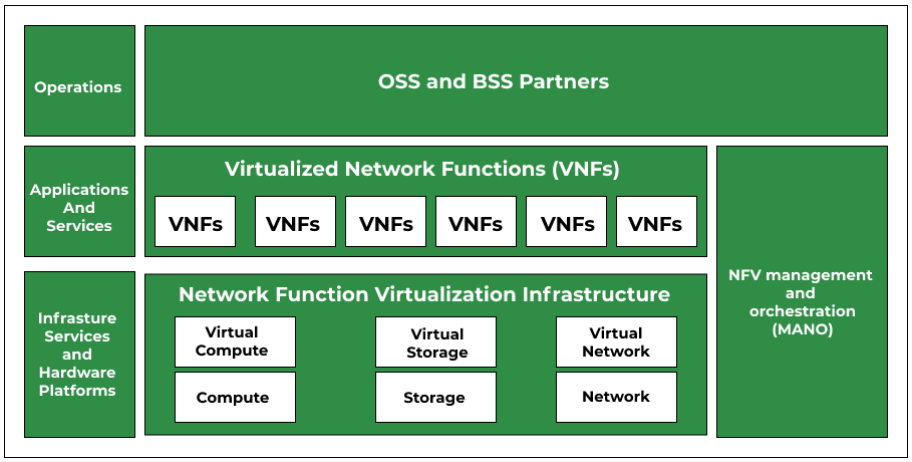
   - **NFV Orchestrator (NFVO)** — Manages network services and end-to-end orchestration
   - **VNF Manager (VNFM)** — Handles lifecycle of individual VNF instances

- **Virtualized Infrastructure Manager (VIM)** — Manages compute, storage, and network resources (e.g., OpenStack, VMware vCenter)

How NFV Works

Network Functions Virtualization (NFV) operates by transforming traditional hardware-based network functions into software-based services that run on virtualized environments, such as virtual machines (VMs) or containers. These virtualized network functions (VNFs) perform essential networking tasks—like routing, load balancing, and firewall protection—while centralized management and orchestration tools handle automation, provisioning, and configuration, allowing networks to be deployed and scaled efficiently.

1. **Virtualized Network Functions** — Software running on VMs or containers performs tasks traditionally handled by dedicated hardware, including routing, load balancing, firewalls, and VPN services.
2. **Automation and Control:** — Hypervisors, container platforms, or software-defined networking (SDN) controllers manage the provisioning, configuration, and orchestration of VNFs.
3. **Rapid Deployment:** —Network services can be deployed, updated, or scaled automatically without manual hardware setup..
4. **Flexible Management** — Centralized orchestration ensures optimized resource usage, service chaining, and dynamic scaling of network functions.



Advantages of NFV

- **Cost Savings** — Uses standard servers → lower CapEx and OpEx (pay-as-you-go model)
- **Faster Deployment** — New services in hours instead of weeks/months
- **Scalability** — Dynamic horizontal scaling without hardware procurement
- **Simplified Management** — Centralized orchestration and automation reduce manual effort

- **Flexibility** — Easy to update, upgrade, or replace network functions via software
- **Innovation Speed** — Rapid introduction of new services and features

Disadvantages of NFV

- **Weaker Physical Security** — Shared servers increase risk if host is compromised
- **Malware Spread Risk** — Malware can propagate faster between co-located VNFs
- **Reduced Visibility** — East-west traffic between VNFs is harder to monitor
- **Virtualization Overhead** — Potential latency and performance impact from hypervisor/container layers
- **Complex Security Management** — Requires advanced isolation, encryption, and consistent policy enforcement
- **Dependency on Underlying Platform** — Hypervisor/container vulnerabilities can affect multiple VNFs

1.1.10 MapReduce

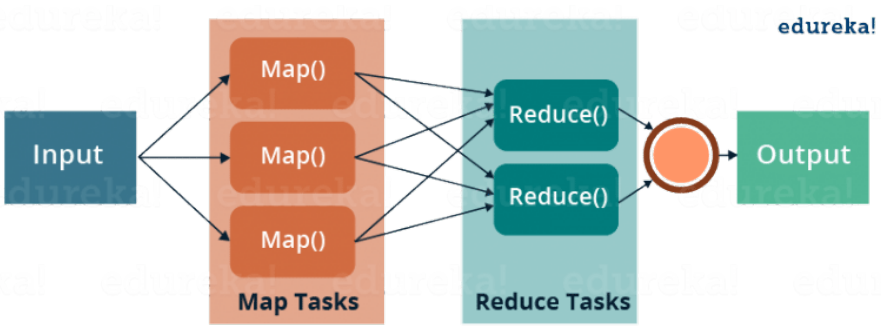
MapReduce is a programming model and processing framework developed by Google for processing and generating large datasets with a parallel, distributed algorithm on a cluster of computers. It is the core processing engine of Apache Hadoop and is widely used for big data analytics in cloud environments.

MapReduce is a software framework and programming model used for processing huge amounts of data. MapReduce program work in two phases, namely, Map and Reduce. Map tasks deal with splitting and mapping of data while Reduce tasks shuffle and reduce the data.

Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. The programs of Map Reduce in cloud computing are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

The input to each phase is key-value pairs. In addition, every programmer needs to specify two functions: map function and reduce function.

MapReduce breaks down large data processing tasks into two main phases: **Map** and **Reduce**, allowing massive datasets to be processed in parallel across many nodes.



Key Concepts of MapReduce

- **Map Phase** — Processes input data in parallel and produces intermediate key-value pairs.
- **Shuffle and Sort** — Groups intermediate data by key and sorts it (handled automatically by the framework).
- **Reduce Phase** — Aggregates the intermediate data to produce the final output.

MapReduce follows the "divide and conquer" strategy and is fault-tolerant, scalable, and suitable for batch processing of large-scale data.

MapReduce Architecture

1. **Input Splitting**
   An input to a MapReduce in Big Data job is divided into fixed-size pieces called input splits Input split is a chunk of the input that is consumed by a single map
2. **Map Phase**
   This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>
3. **Shuffle and Sort Phase**
   This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubed together along with their respective frequency.
4. **Reduce Phase**
   In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

   In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.
5. **Output Writing**
   Final results are written back to HDFS.

Advantages of MapReduce

- Highly scalable — processes petabytes of data across thousands of nodes
- Fault-tolerant — automatic task re-execution on node failure
- Simple programming model — developers only write map and reduce functions
- Data locality — computation moves to data (not data to computation)
- Works well for batch processing of large, unstructured/semi-structured data

Disadvantages of MapReduce

- High latency — not suitable for real-time or low-latency processing
- Disk I/O heavy — intermediate data is written to disk (shuffle phase)
- Complex for iterative algorithms (e.g., machine learning) — requires multiple MapReduce jobs
- Programming is verbose compared to modern frameworks (Spark, Flink)
- No in-memory processing — slower than Spark for iterative workloads

Other Key Technologies

- **MapReduce**: Programming model for processing large datasets in parallel (detailed in Unit-2).
- **Identity and Access Management (IAM)**: Controls user access (e.g., AWS IAM, Azure AD).
- **Service Level Agreements (SLA)**: Contracts defining uptime, performance (e.g., 99.9% availability).
- **Billing**: Pay-per-use models (e.g., per hour, per GB).

Chapter 2

Hadoop and MapReduce

2.0.1 Apache Hadoop

Apache Hadoop is an open-source framework for distributed batch processing of big data. The MapReduce algorithm, proposed as a parallel programming model suitable for the cloud, allows large-scale computations to be parallelized across a large cluster of servers.

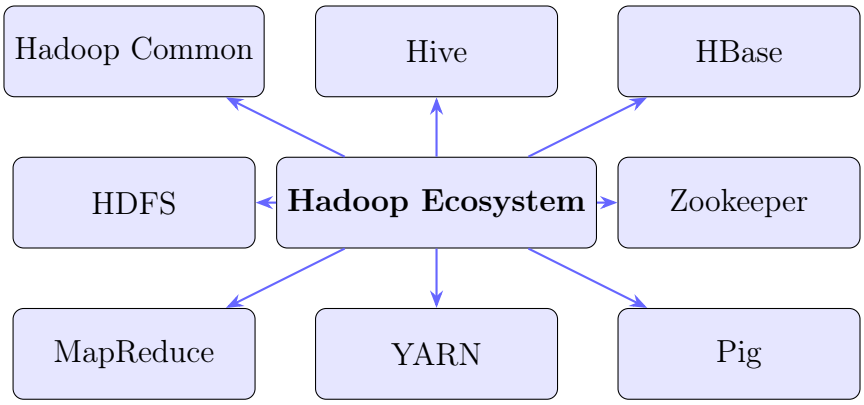


Figure 2.1: Hadoop Ecosystem

The Hadoop ecosystem consists of the following major projects:

- **Hadoop Common**: Common utilities and libraries that support other Hadoop modules.
- **Hadoop Distributed File System (HDFS)**: A distributed file system designed to store very large files across commodity hardware with high throughput and fault tolerance (default replication factor = 3).
- **Hadoop MapReduce**: Parallel programming model and execution environment for large-scale data processing.
- **Hadoop YARN**: Framework for job scheduling and cluster resource management (introduced in Hadoop 2.0).
- **HBase**: Scalable, distributed, column-oriented NoSQL database.
- **ZooKeeper**: Distributed coordination service for configuration, naming, synchronization, and group services.

- **Pig**: Data flow language for analyzing large datasets (compiles to MapReduce jobs).
- **Hive**: Data warehouse infrastructure with HiveQL (SQL-like language) for querying large datasets.
- **Mahout**: Scalable machine learning library.
- **Cassandra, Avro, Oozie, Flume, Sqoop, Chukwa**: Additional tools for database, serialization, workflow, data collection, transfer, etc.

Hadoop MapReduce Job Execution

First Generation Hadoop (MR1)

In MR1, the JobTracker is responsible for both resource management and job scheduling. The cluster consists of:

- **NameNode**: Manages the file system namespace and block locations.
- **Secondary NameNode**: Creates periodic checkpoints of the namespace.
- **JobTracker**: Schedules MapReduce tasks and monitors progress.
- **TaskTracker**: Runs Map and Reduce tasks on slave nodes.
- **DataNode**: Stores data blocks.

Second Generation Hadoop (YARN / MR2)

YARN separates resource management from job scheduling:

- **ResourceManager (RM)**: Global resource manager (Scheduler + Applications Manager).
- **ApplicationMaster (AM)**: Per-application manager that negotiates resources and monitors tasks.
- **NodeManager (NM)**: Per-node agent that launches containers and monitors them.
- **Container**: Logical bundle of resources (CPU, memory, etc.).

2.0.2 Hadoop Schedulers

In Hadoop, we can receive multiple jobs from different clients to perform. The Map-Reduce framework is used to perform multiple tasks in parallel in a typical Hadoop cluster to process large size datasets at a fast rate. This Map-Reduce Framework is responsible for scheduling and monitoring the tasks given by different clients in a Hadoop cluster. But this method of scheduling jobs is used prior to Hadoop 2.

Now in Hadoop 2, we have YARN (Yet Another Resource Negotiator). In YARN we have separate Daemons for performing Job scheduling, Monitoring, and Resource Management as Application Master, Node Manager, and Resource Manager respectively.

Here, Resource Manager is the Master Daemon responsible for tracking or providing the resources required by any application within the cluster, and Node Manager is the slave Daemon which monitors and keeps track of the resources used by an application and sends the feedback to Resource Manager.

Schedulers and Applications Manager are the 2 major components of resource Manager. The Scheduler in YARN is totally dedicated to scheduling the jobs, it can not track the status of the application. On the basis of required resources, the scheduler performs or we can say schedule the Jobs.

These Schedulers are actually a kind of algorithm that we use to schedule tasks in a Hadoop cluster when we receive requests from different-different clients.

A Job queue is nothing but the collection of various tasks that we have received from our various clients. The tasks are available in the queue and we need to schedule this task on the basis of our requirements.

There are mainly 3 types of Schedulers in Hadoop:

FIFO Scheduler

(default): As the name suggests FIFO i.e. First In First Out, so the tasks or application that comes first will be served first. his is the default Scheduler we use in Hadoop. The tasks are placed in a queue and the tasks are performed in their submission order.In this method, once the job is scheduled, no intervention is allowed. So sometimes the high-priority process has to wait for a long time since the priority of the task does not matter in this method.

Advantage:

- No need for configuration
- First Come First Serve
- simple to execute

Disadvantage:

- Priority of task doesn't matter, so high priority jobs need to wait
- Not suitable for shared cluster

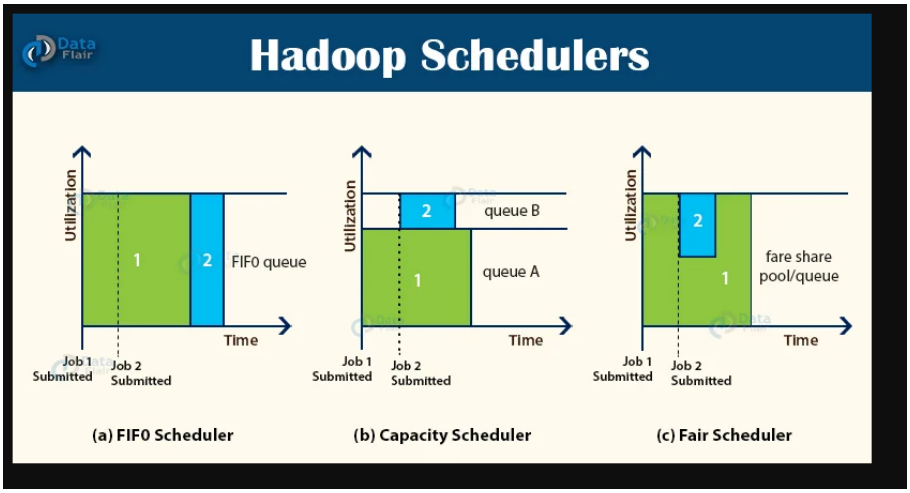
Fair Scheduler

: Assigns resources fairly among jobs and users. Supports pools with guaranteed capacity.

The Fair Scheduler (originally developed by Facebook) aims to allocate resources fairly among all running applications/users over time. It dynamically shares cluster resources without requiring pre-defined capacity reservations.

Key characteristics:

- Resources are shared equally among active applications (fair-share model).
- Scheduling is based on memory (default) but can be configured for CPU or other resources.



- Supports **pools** (similar to queues) to group applications/users.
- When a high-priority job or new application arrives, the scheduler can preempt (kill) tasks from other running jobs to free up resources.
- Can limit the number of concurrent running tasks per pool/user.
- No fixed capacity — resources are redistributed dynamically as jobs come and go.

Advantages of Fair Scheduler:

- Resources assigned to each application depend on its priority and demand
- Excellent fairness across users and jobs
- Can limit concurrent tasks per pool/user
- Good for interactive/multi-user environments

Disadvantages of Fair Scheduler:

- Requires configuration (pools, weights, preemption settings)
- Preemption may cause task restarts (performance impact)
- Less predictable than Capacity Scheduler for production workloads

Capacity Scheduler

: Uses named queues with guaranteed capacity. Supports hierarchical queues and priority.

The Capacity Scheduler allows multiple tenants (users, teams, departments, or applications) to share a large Hadoop cluster in a predictable and guaranteed manner. It organizes jobs into multiple **queues**, each with a pre-assigned portion of cluster resources (capacity).

Key characteristics:

- Each queue is allocated a fixed percentage of cluster resources (slots / capacity).
- Queues can be hierarchical (root → parent → leaf queues).

- Leaf queues are where applications are actually submitted.
- If a queue is not using its allocated capacity, other queues can temporarily borrow unused capacity (elastic sharing).
- When demand increases in an under-utilized queue, running tasks from over-allocated queues may be killed (preemption is optional and configurable).
- Provides visibility into resource usage per queue/user/application.
- Prevents any single user or application from monopolizing the cluster.

Queue Types in Capacity Scheduler:

- **Root Queue** — Top-level queue representing the entire cluster.
- **Parent Queue** — Intermediate queues (e.g., representing organizations or departments).
- **Leaf Queue** — Bottom-level queues where actual applications/jobs are submitted.

Advantages of Capacity Scheduler:

- Best suited for clusters with multiple clients or priority-based jobs
- Guarantees minimum capacity to important queues (production, high-priority)
- Maximizes overall cluster throughput
- Elastic resource sharing improves utilization
- Strong access control (ACLs) per queue

Disadvantages of Capacity Scheduler:

- More complex to configure compared to FIFO
- Requires careful planning of queue capacities
- Not as easy for beginners to set up correctly

Comparison: Capacity Scheduler vs Fair Scheduler

2.1 Hadoop Cluster Setup

A cluster is simply a group of interconnected computers (or nodes) that work together as a single system. These nodes are connected via a Local Area Network (LAN) and share resources and tasks to achieve a common goal. Together, they function as one powerful unit. A Hadoop cluster is a type of computer cluster made up of commodity hardware (inexpensive and widely available devices). These nodes work together to store and process large volumes of data in a distributed environment.

Components of a Hadoop Cluster

- **Master Nodes:**

Feature	Capacity Scheduler	Fair Scheduler
Resource Allocation	Guaranteed minimum capacity per queue	Dynamic fair sharing
Preemption	Optional	Supported (default behavior)
Queue/Pool Model	Hierarchical queues with fixed shares	Pools with dynamic sharing
Best For	Multi-tenant enterprise clusters with predictable workloads	Interactive, research, or multi-user clusters
Guarantees	Strong capacity guarantees	Fairness over time
Complexity	Medium-High	Medium
Default in modern Hadoop	Yes (most distributions)	No

Table 2.1: Comparison of Capacity Scheduler and Fair Scheduler

  - **NameNode:** Handles file system metadata.
  - **ResourceManager:** Manages cluster resources.

- **Slave Nodes:**
  - **DataNodes:** Store actual data.
  - **NodeManagers:** Manage execution of tasks.

The Master nodes coordinate and guide the Slave nodes to efficiently store, process and analyze data.

Types of Data Processed by Hadoop

Hadoop clusters can handle various types of data:

- **Structured Data:** Organized in a fixed schema, e.g., MySQL databases.
- **Semi-Structured Data:** Partially organized data, e.g., XML, JSON.
- **Unstructured Data:** No fixed format, e.g., images, videos, audio files.

Hadoop Cluster Architecture

The Hadoop cluster architecture consists of a Master node that controls and coordinates multiple Slave nodes. This setup enables distributed storage and parallel data processing across inexpensive hardware.

Hadoop Cluster Properties

The following are the core properties of a Hadoop cluster:

- **Scalability:** Hadoop clusters can easily scale up or down by adding or removing nodes. For example, if data grows from 5PB to 7PB, more servers can be added to handle it.
- **Flexibility:** Hadoop can store and process all types of data (structured, semi-structured or unstructured), making it highly adaptable.

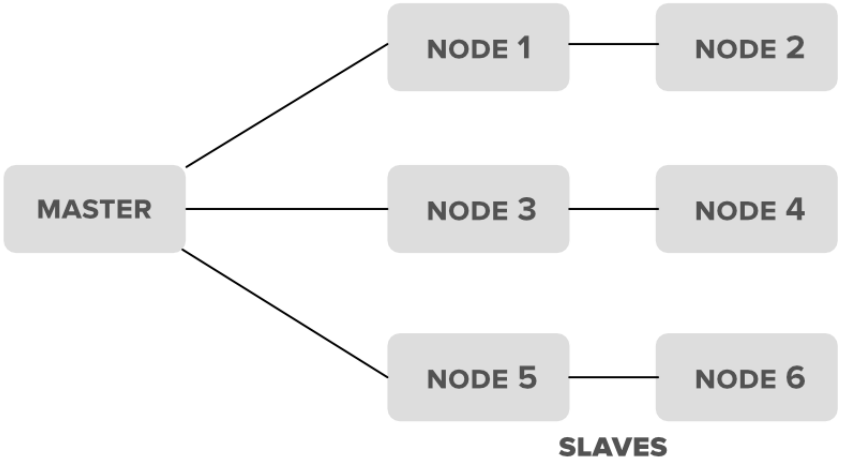


Figure 2.2: Hadoop Cluster Architecture

- **Speed:** Due to its distributed nature and use of MapReduce, Hadoop processes data quickly across multiple nodes.
- **No Data Loss:** Hadoop keeps multiple copies of data across nodes. If one node fails, the data is still safe on another.
- **Economical:** Hadoop runs on low-cost commodity hardware, making it a budget-friendly solution for big data storage and processing.

2.1.1 Types of Hadoop Clusters

Single Node Hadoop Cluster

In a single node cluster, all Hadoop components (NameNode, DataNode, ResourceManager, NodeManager, etc.) run on the same machine...

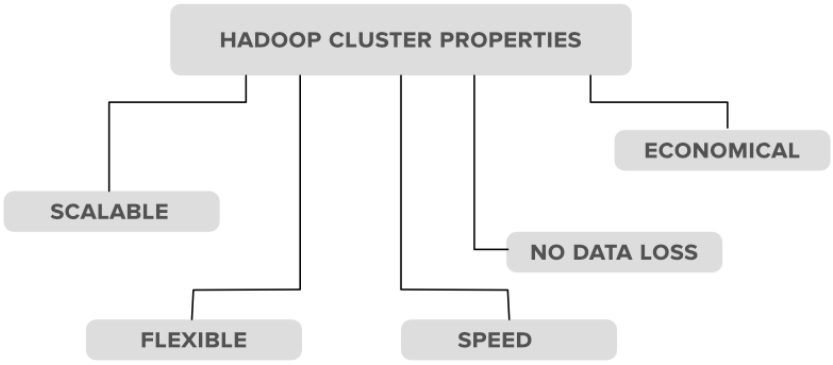


Figure 2.3: Single Node Hadoop Cluster

Multi Node Hadoop Cluster

A multi node cluster has multiple machines...

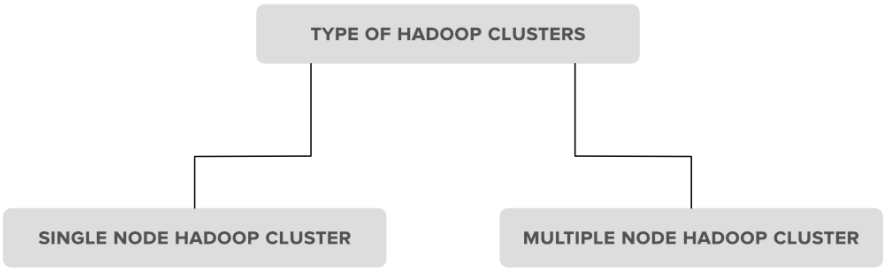


Figure 2.4: Multi Node Hadoop Cluster

Steps to Set Up a Multi-Node Hadoop Cluster

1. Install Java 6 or later on all nodes.
2. Download and unpack Hadoop tarball on all nodes.
3. Configure networking (hostname, /etc/hosts, passwordless SSH).
4. Configure Hadoop files:
   - `core-site.xml`, `hdfs-site.xml`, `mapred-site.xml`
   - `masters`, `slaves`
5. Format NameNode: `bin/hadoop namenode -format`
6. Start daemons: `start-dfs.sh` and `start-mapred.sh`
7. Verify using web UIs:
   - NameNode: `http://master:50070`
   - JobTracker: `http://master:50030`

2.1.2 Cloud Services and Platforms

Cloud computing is a transformative technology that enables users to access computing resources—such as servers, storage, databases, and applications—over the internet on a pay-as-you-go basis. Cloud platforms provide developers and enterprises with scalable infrastructure and frameworks to build, deploy, and manage applications efficiently without managing underlying hardware directly.

Major providers: AWS, Azure, GCP.

Compute Services

Provides resizable virtual computing capacity. Users can select from different hardware configurations, including GPU and cluster instances, to deploy virtual servers. EC2 allows users to create and save customized instance images for future use.

AWS EC2, Azure VMs, Google Compute Engine.

Storage Services

Offers scalable and persistent object storage organized into buckets. Users can store and retrieve any amount of data—from small files to complete disk images—through APIs or the AWS console.

AWS S3 (object), EBS (block); Azure Blob; Google Cloud Storage.

Database Services

AWS RDS, DynamoDB; Azure SQL/Cosmos DB; Google Cloud SQL/Bigtable.

Other Services

Content Delivery (CDN), Analytics, Deployment (e.g., Kubernetes), IAM.

Open Source Private Cloud Software

OpenStack, Eucalyptus, CloudStack.

Chapter 3

UNIT-2: Hadoop and Python

3.0.1 Hadoop MapReduce

Apache Hadoop is an open-source framework for distributed storage (HDFS) and processing (MapReduce/YARN).

Hadoop MapReduce Job Execution

Map phase: Processes input data in parallel → key-value pairs. Shuffle & Sort: Groups by key. Reduce phase: Aggregates results.

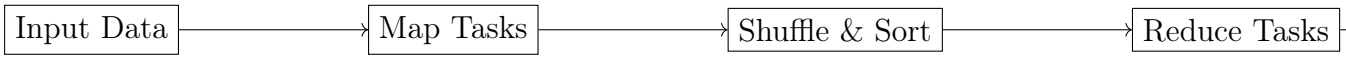


Figure 3.1: MapReduce Data Flow

MapReduce is the processing engine of Hadoop. While **HDFS** is responsible for storing massive amounts of data, MapReduce handles the actual computation and analysis.

It provides a simple yet powerful programming model that allows developers to process large datasets in a distributed and parallel manner.

MapReduce is a **two-phase** data processing model in Hadoop:

- **Map Phase**: Breaks the data into smaller chunks, processes them, and generates intermediate (key, value) pairs.
- **Reduce Phase**: Aggregates and merges the intermediate results to produce the final output.

How MapReduce Works – Step by Step:

Example Input File: `sample.txt`

```
Hello I am GeeksforGeeks
How can I help you
How can I assist you
Are you an engineer
Are you looking for coding
Are you looking for interview questions
what are you doing these days
what are your strengths
```

When stored in HDFS, this file is divided into **input splits** (e.g., `first.txt`, `second.txt`, etc.). Each split is assigned to a Mapper.

Step 1: Input Splitting & Record Reader

- The input file is divided into **splits** (typically 128–256 MB each).
- Each split is converted into (key, value) pairs using a **RecordReader**.
- In `TextInputFormat` (default), the key is the **byte offset** of the line and the value is the line itself.

Example:

```
(0,   "Hello I am GeeksforGeeks")
(26,  "How can I help you")
(48,  "How can I assist you")
...
```

File Formats in Hadoop

Hadoop provides several built-in **InputFormat** classes:

- `TextInputFormat` (Default) → (byte offset, line)
- `KeyValueTextInputFormat` → Splits line using delimiter (e.g. tab)
- `SequenceFileInputFormat` → Binary (key, value) format
- `SequenceFileAsTextInputFormat` → SequenceFile → Text

By default, Hadoop uses `TextInputFormat`.

Step 2: Map Phase

Each Mapper processes its assigned input split and emits intermediate (key, value) pairs.

Example Mapper output (Word Count):

For line `"Hello I am GeeksforGeeks"`:

```
(Hello,       1)
(I,           1)
(am,          1)
(GeeksforGeeks,1)
```

All Mappers run **in parallel**.

Step 3: Shuffling and Sorting

Framework performs (automatically):

- **Partitioning** → decides which reducer gets which key
- **Shuffling** → transfers data across the network
- **Sorting** → groups values for the same key

After shuffle & sort:

```
(How,  [1,1])
(Are,  [1,1,1])
(I,    [1,1])
(what, [1,1])
...
```

Step 4: Reduce Phase

Reducer receives `(key, list of values)` and produces final output.

Example (sum):

```
(How,  [1,1])   $\to$   (How,  2)
(Are,  [1,1,1]) $\to$   (Are,  3)
(I,    [1,1])   $\to$   (I,    2)
```

Final output is written to HDFS (usually in directory `result.output`).

Advantages of MapReduce

- **Scalable** – Handles petabytes of data across thousands of nodes
- **Parallel** – Executes tasks simultaneously on data-local nodes
- **Fault-Tolerant** – Automatically re-executes failed tasks
- **Simple programming model** – Developer only implements map() and reduce(); Hadoop handles distribution, scheduling, fault recovery

Hadoop Schedulers (YARN)

In Hadoop 1.x, MapReduce handled both processing and resource management/scheduling. In Hadoop 2.x and later (YARN architecture), resource management is separated:

- **ResourceManager** (Master) – Tracks cluster resources
- **NodeManager** (per node, Slave) – Manages containers & reports usage
- **ApplicationMaster** (per application) – Manages task lifecycle

The **Scheduler** (part of ResourceManager) is responsible only for **allocating resources** to applications based on policies. It does **not** track application status.

There are three main pluggable schedulers in YARN:

1. FIFO Scheduler (default in early Hadoop)
2. Capacity Scheduler (most commonly used in enterprises)
3. Fair Scheduler

A **job queue** is a collection of pending applications/tasks waiting to be scheduled.

1. FIFO Scheduler (First In First Out)

Jobs are executed in the order they are submitted. Once started, a job gets the entire cluster until completion (no preemption).

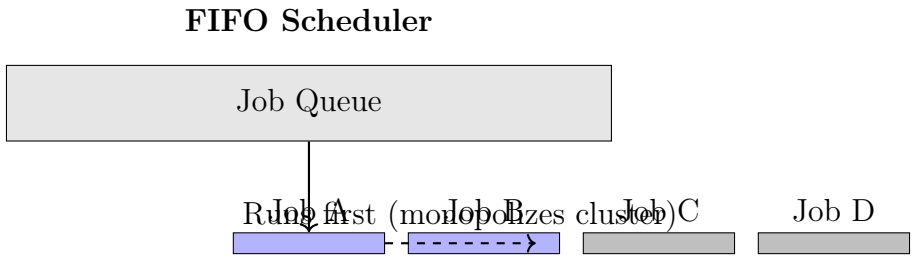


Figure 3.2: FIFO Scheduler: First-come, first-served (monopolizing style)

Advantages:

- No configuration needed
- Simple & predictable
- First-come, first-served

Disadvantages:

- No priority support → long-running/low-priority jobs block high-priority ones
- Poor for multi-tenant/shared clusters

2. Capacity Scheduler

Designed for multi-tenant clusters. Cluster divided into hierarchical queues with guaranteed minimum capacity (percentage of resources). Queues can borrow unused capacity from others (elasticity). Within a queue, usually FIFO ordering.

Advantages:

- Guarantees minimum capacity per organization/team
- Supports priorities & multi-tenancy
- High cluster utilization (borrowing)

Disadvantages:

- More complex configuration
- Requires planning of queue capacities

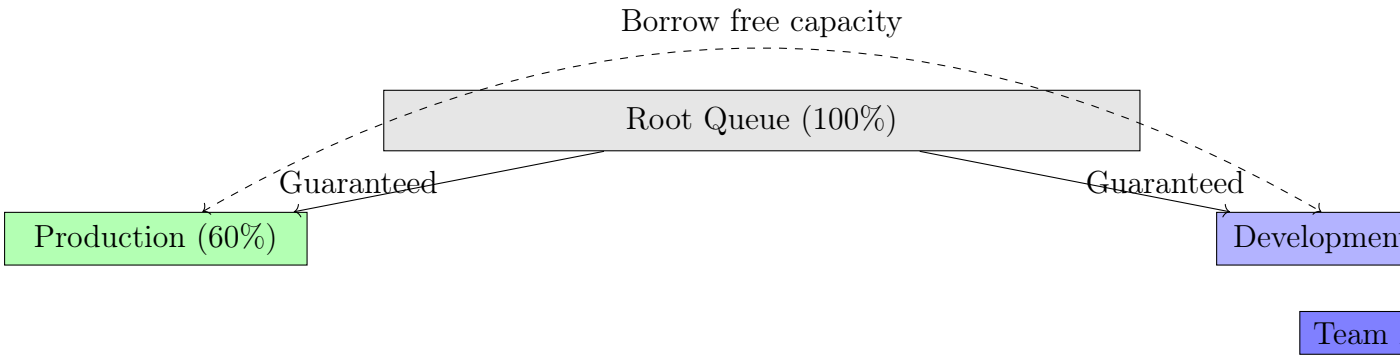


Figure 3.3: Capacity Scheduler: Hierarchical queues with guaranteed + elastic capacity

3. Fair Scheduler

Focuses on fairness over time. All running applications get roughly equal share of resources (dynamic). No fixed capacity reservation — shares adjust dynamically. Supports preemption: kills tasks from over-allocated apps to reclaim for under-served ones. Can be configured per pool/queue with min/max shares.

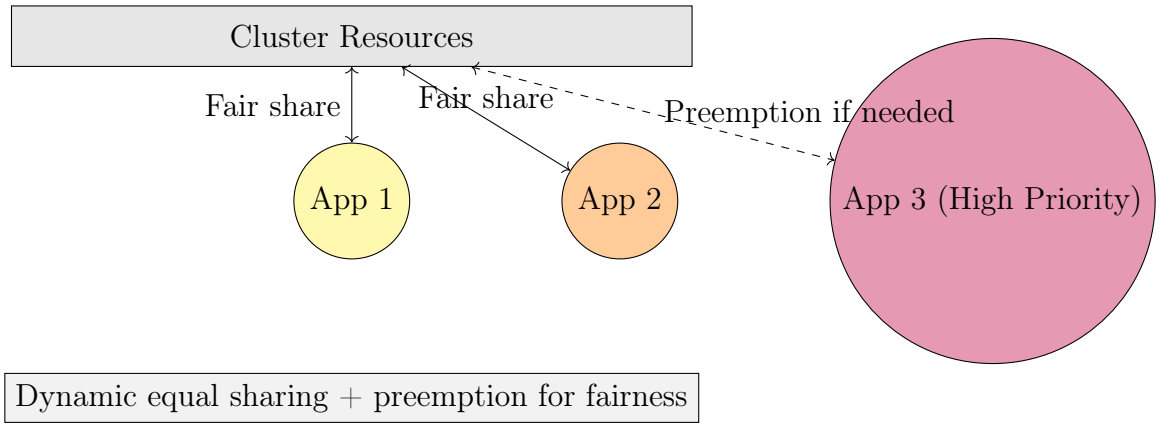


Figure 3.4: Fair Scheduler: Dynamic fair sharing with preemption

Advantages:

- Fair resource distribution over time
- Supports priorities & preemption
- Good response time for short jobs

Disadvantages:

- Requires configuration
- Preemption can kill tasks (may waste work)

Comparison Table

Hadoop Cluster Setup

This document provides a deep, step-by-step explanation of setting up a multi-node Hadoop cluster (Hadoop 3.x) in a non-secure mode, suitable for development and testing.

Feature	FIFO	Capacity	Fair
Default	Yes (early versions)	Common in enterprises	Common in multi-user
Resource Guarantee	None	Yes (minimum per queue)	Dynamic fair share
Multi-tenancy	Poor	Excellent (hierarchical)	Excellent (pools)
Preemption	No	No (or limited)	Yes
Priority Support	No	Yes	Yes
Complexity	Low	Medium	Medium
Best For	Single-user / testing	Predictable multi-tenant	Dynamic / interactive work

Table 3.1: Comparison of YARN Schedulers

The setup involves multiple machines: one or more master nodes (for NameNode and ResourceManager) and several worker nodes (for DataNodes and NodeManagers). We assume a Linux environment (e.g., Ubuntu or CentOS).

Hadoop cluster setup enables distributed storage (HDFS) and processing (YARN/MapReduce). Key benefits include scalability, fault tolerance, and parallel processing. For production, integrate security (Kerberos) and high availability (HA) features like NameNode federation or ZooKeeper-based failover.

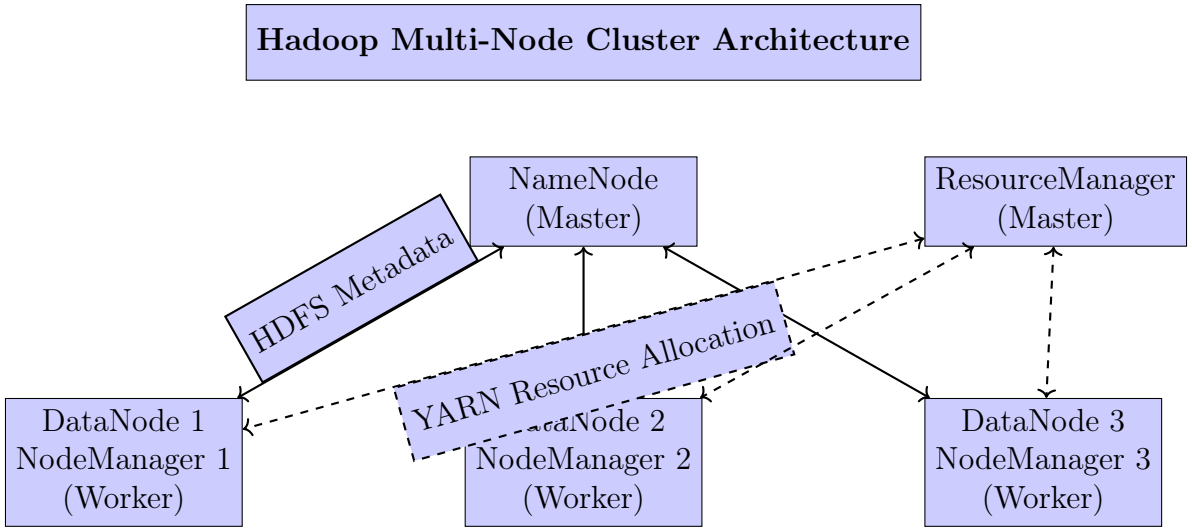


Figure 3.5: Hadoop Cluster: Masters manage metadata/resources; Workers store data and execute tasks. Solid lines: HDFS; Dashed: YARN.

:render type="render$_s earched_i mage$" >< $argumentname =$ "$image_i d$" > 0 < /$argument$ >< $argumentname =$ "$size$" > $LARGE$ < /$argument$ >< /$grok : render$ >

Prerequisites

Before setup:

- **OS**: Linux (e.g., Ubuntu 20.04+).
- **Java**: OpenJDK 11 or later. Set `JAVA_HOME`.
- **Nodes**: 1+ masters, 2+ workers. All networked with hostname resolution (/etc/hosts or DNS).

- **Users**: Create `hdfs`, `yarn`, `mapred` users.
- **Software**: SSH, rsync.

Common pitfalls: Inconsistent Java versions across nodes can cause failures.

Downloading and Installing Hadoop

Download Hadoop 3.x (e.g., 3.3.5) from `https://hadoop.apache.org/releases.html`.
On all nodes:

```
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.5/
   hadoop-3.3.5.tar.gz
tar -xzf hadoop-3.3.5.tar.gz
sudo mv hadoop-3.3.5 /opt/hadoop
sudo chown -R root:root /opt/hadoop
```

Environment Configuration

Edit `~/.bashrc` or `/etc/profile.d/hadoop.sh` on all nodes:

```
export HADOOP_HOME=/opt/hadoop
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

Source the file: `source ~/.bashrc`.
Edit `hadoop-env.sh`:

```
export HADOOP_HEAPSIZE_MAX=4096
export HDFS_NAMENODE_OPTS="-Xmx4g"
```

Create directories (e.g., for PIDs, logs) with proper ownership.

SSH Passwordless Setup

Essential for scripts like `start-dfs.sh`.
On master, as `hdfs` and `yarn`:

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
ssh-copy-id user@worker1
# Repeat for all workers and users
```

Test: `ssh user@worker1` (no password).
Pitfall: Firewall blocks SSH (open port 22).

Configuration Files

Sync configs across nodes. Key files in `$HADOOP_CONF_DIR`.

core-site.xml

Defines default filesystem.

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://master-host:9000</value>
  </property>
</configuration>
```

hdfs-site.xml

HDFS settings.

```
<configuration>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/data/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>/data/hdfs/datanode</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
</configuration>
```

Create dirs: `sudo mkdir -p /data/hdfs/*; sudo chown hdfs:hdfs /data/hdfs`.

yarn-site.xml

YARN settings.

```
<configuration>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>master-host</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

mapred-site.xml

MapReduce on YARN.

```
<configuration>
  <property>
```

```
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

workers File

List workers:

```
worker1
worker2
```

Formatting and Starting the Cluster

As `hdfs` on master:

```
hdfs namenode -format
```

Start HDFS:

```
start-dfs.sh
```

Start YARN (as `yarn`):

```
start-yarn.sh
```

Start HistoryServer (as `mapred`):

```
mapred --daemon start historyserver
```

Verification

- Web UIs: NameNode[](http://master:9870), RM[](http://master:8088). - Commands: `hdfs dfs -ls /`, `yarn node -list`.

:render type="render$_{s}earched_{i}mage$" >< $argumentname = "image_{i}d" > 1 < /argument ><$ $argumentname = "size" > SMALL < /argument >< /grok : render >$

Example: Running a Python MapReduce Job via Hadoop Streaming

Hadoop supports Python via Streaming. Here's a word count example.

Python Mapper Code (mapper.py)

```
import sys
for line in sys.stdin:
    words = line.strip().split()
    for word in words:
        print(f"{word}\t1")
```

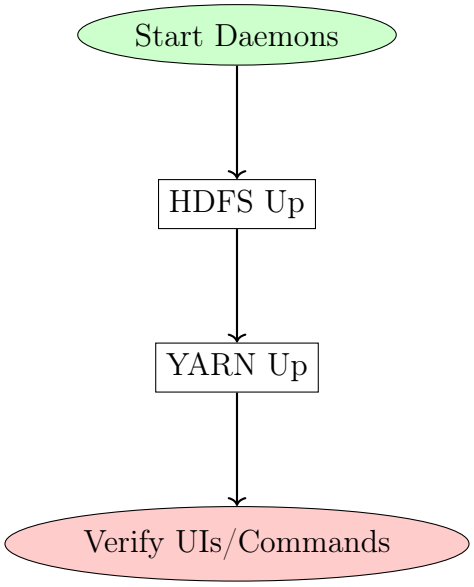


Figure 3.6: Cluster Startup Flow

Python Reducer Code (reducer.py)

```
import sys
current_word = None
current_count = 0
for line in sys.stdin:
    word, count = line.strip().split('\t', 1)
    count = int(count)
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count
if current_word:
    print(f"{current_word}\t{current_count}")
```

Running the Job

Upload input to HDFS:

```
hdfs dfs -put input.txt /
```

Run:

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.
   jar \
  -input /input.txt \
  -output /output \
  -mapper "python3 mapper.py" \
  -reducer "python3 reducer.py" \
  -file mapper.py \
```

```
  -file reducer.py
```

Sample Input (input.txt)

```
Hello world
Hello Hadoop
Hadoop cluster setup
```

Expected Output

After running: `hdfs dfs -cat /output/part-00000`

```
Hello   2
Hadoop  2
cluster 1
setup   1
world   1
```

This demonstrates Python integration. In a real cluster, monitor via RM UI.

:render type="render$_{s}earched_{i}mage$" >< $argumentname =$ "$image_{i}d$" > 2 < /$argument$ >< $argumentname =$ "$size$" > $LARGE$ < /$argument$ >< /$grok : render$ >

*Security and Common Pitfalls - Use Kerberos for production. - Pitfalls: SSH issues, dir permissions, hostname mismatches. - Deep tip: Tune heap sizes based on RAM; enable log aggregation for debugging.

3.0.2 Cloud Application Design

Reference Architecture for Cloud Applications

The architecture of cloud computing combines **SOA (Service-Oriented Architecture)** and **EDA (Event-Driven Architecture)**.

It is generally divided into two main parts:

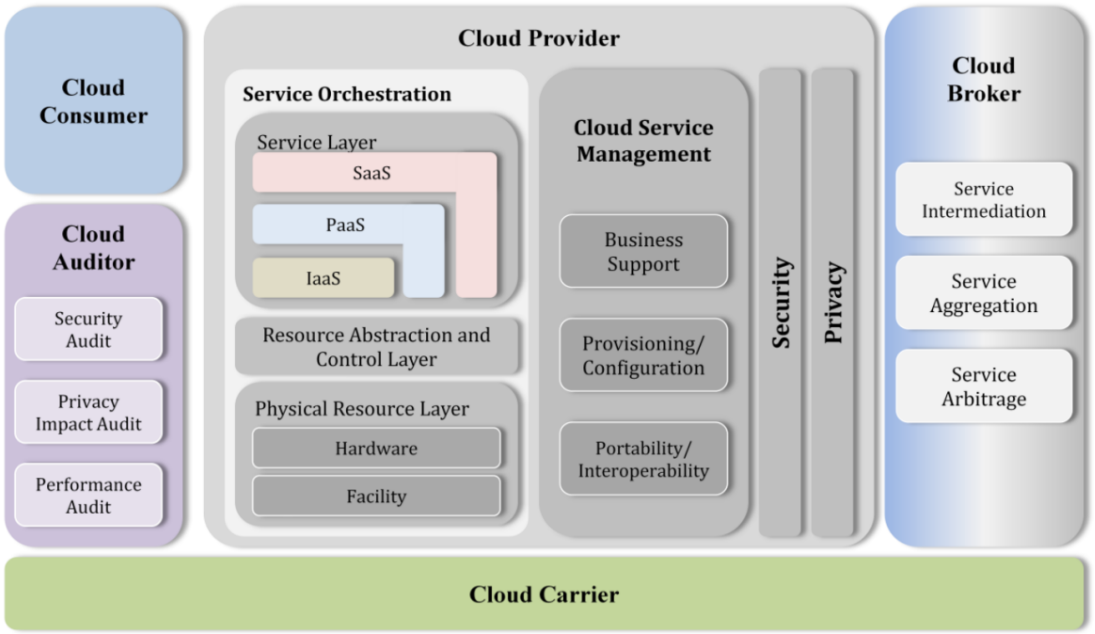
- **Frontend** (client side)
- **Backend** (cloud provider side)

The following figure shows a reference model of cloud computing architecture (NIST-inspired with roles and layers):

[above=0.4cm of provider] **Cloud Computing Reference Architecture**;

Frontend vs Backend

- **Frontend** — Client side
  - User interfaces and applications
  - Web browsers, mobile apps, thick clients
  - Goal: Interact with cloud services
- **Backend** — Cloud provider side



- Manages resources, storage, VMs, security
- Includes virtualization, orchestration, databases, networking

Main Components of Cloud Computing Architecture

1. **Client Infrastructure** (Frontend)
   GUI and applications used to access cloud services
2. **Application** (Backend)
   Software/platform accessed by the client
3. **Service** (Backend)
   SaaS, PaaS, IaaS — determines type of service delivered
4. **Runtime Cloud** (Backend)
   Execution environment for applications/VMs
5. **Storage** (Backend)
   Scalable object, block, file storage (e.g. S3, GCS)
6. **Infrastructure** (Backend)
   Servers, storage devices, network hardware, hypervisors
7. **Management** (Backend)
   Orchestration, provisioning, monitoring, auto-scaling
8. **Security** (Backend)
   IAM, encryption, firewalls, DDoS protection, compliance
9. **Internet**
   Communication bridge between frontend and backend

10. **Database** (Backend)
    RDS, DynamoDB, Cloud SQL, BigQuery, Cosmos DB...
11. **Networking** (Backend)
    VPC, load balancers, DNS, CDN, VPN
12. **Analytics** (Backend)
    Data warehouse, BI tools, ML platforms

Real-World Use Cases

1. **Online Learning Platform (e.g. GeeksforGeeks Classroom)**
    - Video storage → Amazon S3
    - Serverless backend tasks → AWS Lambda
    - User authentication → AWS IAM / Cognito
    - Scalable delivery anywhere
2. **E-Commerce Website (Amazon, Flipkart)**
    - Website hosting → Elastic Beanstalk / EC2
    - Product images → S3
    - Order & inventory → RDS / DynamoDB
    - Real-time monitoring & security
3. **Food Delivery Mobile App (Zomato, Swiggy)**
    - App frontend (mobile)
    - Backend → Firebase / Google Cloud / AWS
    - Notifications, location tracking, payments
    - Secure user data handling

Cloud Application Design Methodologies

Designing applications for the cloud differs significantly from traditional on-premises or monolithic design. Modern cloud applications aim to be:

- **Scalable** horizontally (out/in)
- **Resilient** to failures
- **Observable**
- **Deployable** frequently & safely
- **Cost-efficient**

The most influential cloud application design methodologies / patterns today are:

1. **12-Factor App** (baseline for cloud-native apps)

2. **Microservices Architecture**
3. **Cloud-Native Patterns** (CNCF / Kubernetes style)
4. **Serverless-first Design**
5. **Event-Driven Architecture (EDA)**
6. **Strangler Fig / Strangler Pattern** (legacy modernization)

1. The 12-Factor App

Introduced by Heroku in 2011–2012 — still considered foundational in 2026.

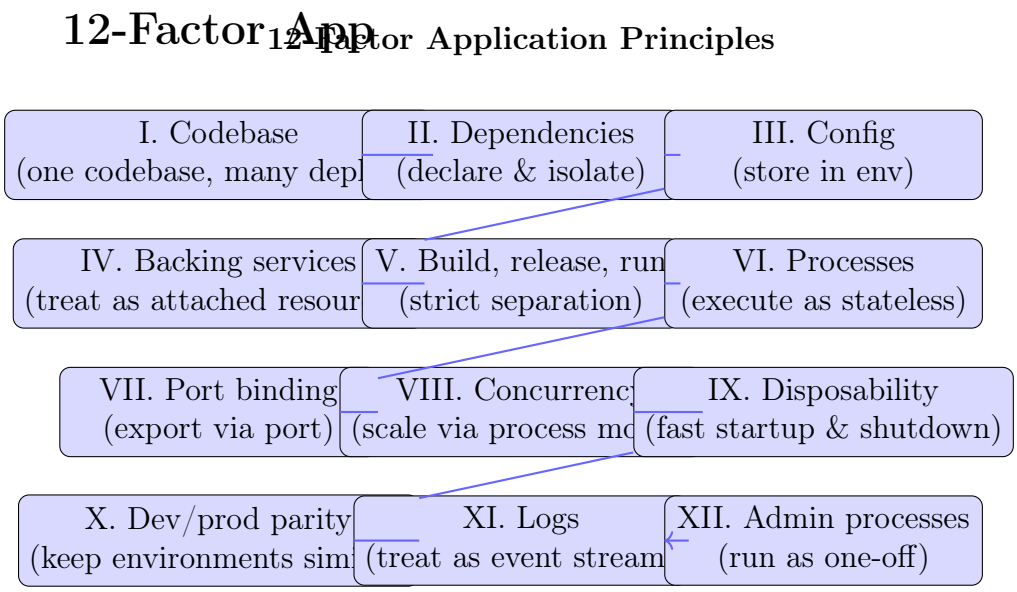


Figure 3.7: 12-Factor App – Core Principles Flow

2. Microservices vs Monolith vs Modular Monolith

3. Cloud-Native Design Patterns (CNCF / Kubernetes Era)

Most popular patterns in 2026:

Pattern	Goal	Key Characteristics / Tools
Sidecar	Extend / augment workload	Istio, Linkerd, Envoy proxy
Service Mesh	Observability, security, traffic mgmt	Istio, Linkerd, Consul
API Gateway	Single entry point, routing, auth	Kong, Ambassador, AWS API Gatew
Circuit Breaker	Prevent cascading failures	Resilience4j, Polly, Istio
Bulkhead	Isolate failures	Thread pools, Kubernetes namespace
Strangler Fig	Incremental legacy replacement	Gradually replace parts with services
Backends for Frontends (BFF)	Tailored APIs per client	GraphQL federation, custom BFFs
Saga	Manage distributed transactions	Choreography / Orchestration
CQRS + Event Sourcing	High write/read separation	Kafka, EventStoreDB, Axon

Table 3.2: Selected Cloud-Native Design Patterns (2026 relevance)

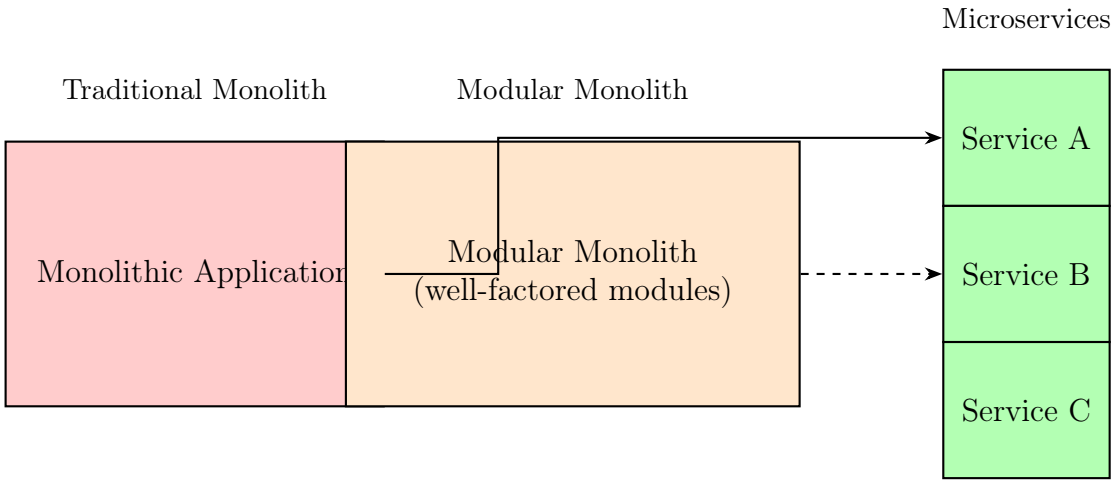


Figure 3.8: Evolution: Monolith → Modular Monolith → Microservices

4. Serverless-first vs Containers-first Decision

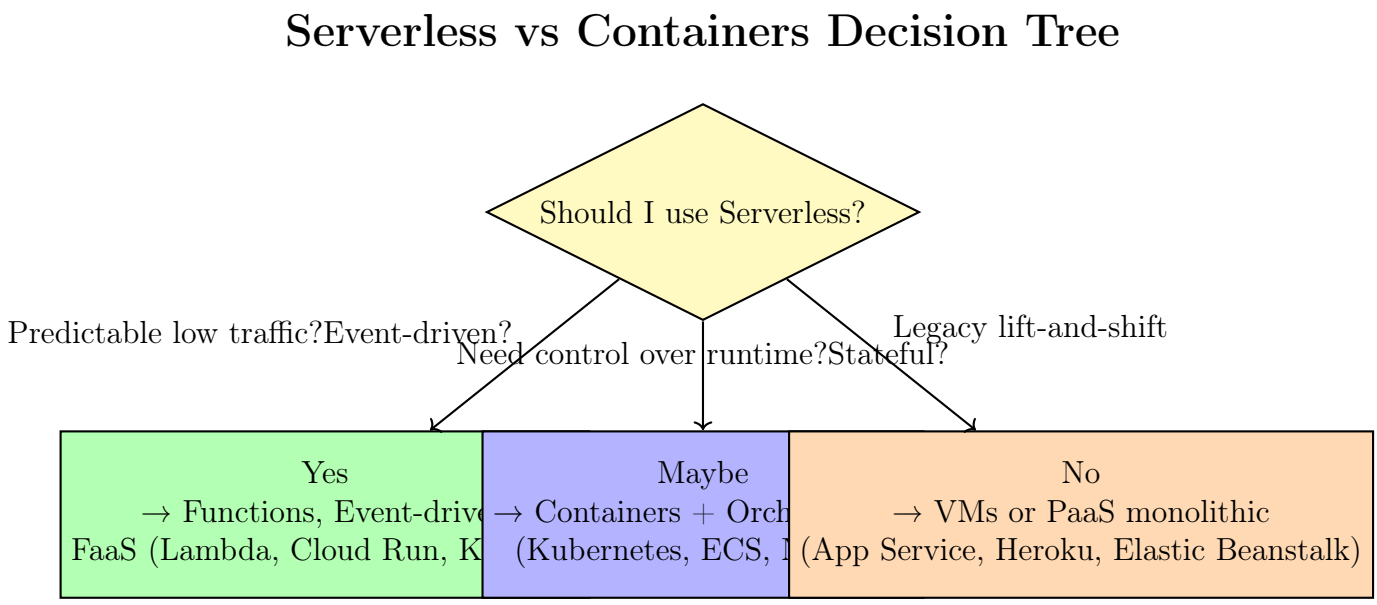


Figure 3.9: Serverless-first vs Containers-first decision heuristic

5. Strangler Fig Pattern – Modernization Path

- **New greenfield project** → 12-Factor + Serverless-first or Kubernetes-native microservices
- **Interactive web/mobile backend** → API Gateway + BFF + Event-driven
- **High-scale data/event processing** → EDA + Kafka + Serverless functions
- **Legacy modernization** → Strangler Fig → Modular Monolith → Microservices
- **Enterprise regulated** → Hybrid: Microservices + Service Mesh + strong observability

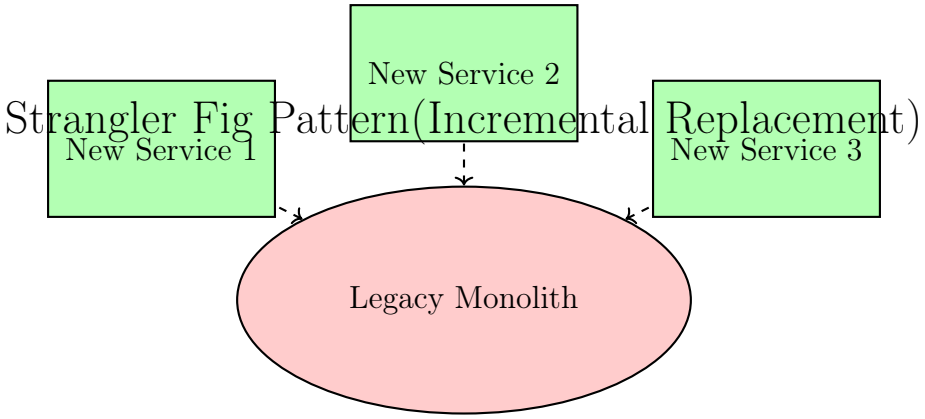


Figure 3.10: Strangler Fig – Gradually replace legacy system with microservices

These methodologies are **not mutually exclusive** — most real-world cloud applications combine several of them.

Data Storage Approaches

Cloud storage enables scalable, durable, and accessible data management without owning physical hardware. Data is stored on remote servers (often distributed globally), with access via internet or private links.

Key advantages (2026 view):

- Elastic scaling (pay for what you use)
- High durability (11–9s or better)
- Built-in redundancy & geo-replication
- Integration with compute, AI/ML, analytics

Main approaches fall into two orthogonal dimensions:

1. **Storage format / access model**: Block, File, Object
2. **Deployment model**: Public, Private, Hybrid, Multi-cloud

3.0.3 Data Storage Approaches in Cloud Computing

Cloud storage is a core pillar of cloud computing, enabling scalable, durable, and highly available data management without the need to own or maintain physical hardware. Data resides on remote, often globally distributed servers and is accessed via the Internet or dedicated private connections.

Key advantages of cloud storage (as emphasized in [**?**, **?**, **?**]):

- Elastic scaling — pay only for actual usage
- High durability (typically 11–9s or better)
- Built-in geo-redundancy and replication
- Seamless integration with compute, analytics, AI/ML services

Cloud storage approaches are generally classified along two main dimensions:

1. **Storage access model / format**: Block, File, Object
2. **Deployment / ownership model**: Public, Private, Hybrid, Multi-cloud

Core Storage Access Models

Modern cloud platforms primarily offer three fundamental storage types:

Feature	Block Storage	File Storage	
Access Protocol	iSCSI / NVMe-oF / virtio	NFS / SMB / CIFS	H
Data Organization	Raw fixed-size blocks	Hierarchical (folders/files)	
Performance (IOPS / latency)	Highest (databases, VMs)	Medium–High	
Scalability	Limited per volume	Good (file system constraints)	
Typical Use Cases	Databases, boot volumes, VMs	Shared drives, CMS, home dirs	E
Durability (2025–2026 norm)	99.999%–99.9999999%	99.999%	
Representative Services	AWS EBS, Azure Disk, GCP PD	AWS EFS, Azure Files, Filestore	AWS

Table 3.3: Comparison of Block, File, and Object Storage

- **Block storage** — behaves like a raw hard disk; ideal for high-performance databases and operating system volumes .
- **File storage** — provides a traditional hierarchical file system; suitable for shared access and legacy applications.
- **Object storage** — flat, HTTP-based, metadata-rich; dominates for massive scale, unstructured data, backups, and data lakes.

Deployment Models of Cloud Storage

- **Public Cloud Storage** — multi-tenant, shared infrastructure (AWS S3, Google Cloud Storage, Azure Blob Storage) — lowest cost, highest elasticity
- **Private Cloud Storage** — dedicated infrastructure (on-premises Ceph, VMware vSAN, or managed dedicated VPC storage) — maximum control and compliance
- **Hybrid Cloud Storage** — sensitive/critical data stays private, burst/less-sensitive workloads use public cloud
- **Multi-Cloud Storage** — uses S3-compatible APIs (MinIO, Ceph) across multiple providers to avoid lock-in

Modern / Specialized Storage Approaches

As described in recent editions and supplementary literature:

- **Data Lakes & Lakehouses** — object storage + open table formats (Delta Lake, Apache Iceberg, Hudi) for analytics and ML workloads
- **Intelligent / Auto-tiered Storage** — automatic movement between Hot → Cool → Cold → Archive classes (S3 Intelligent-Tiering, GCS storage classes)

- **Serverless Storage** — fully managed file/block (EFS IA, Filestore Enterprise, Azure Files Premium).
- **Edge / Distributed Object Storage** — low-latency global access (Cloudflare R2, Fastly, Akamai).
- **AI/ML Optimized Storage** — high-throughput parallel file systems (AWS FSx for Lustre, Google Hyperdisk ML).

Quick Decision Guide

Requirement / Workload	**Recommended Storage Type**
High IOPS databases / VMs	Block (EBS gp3/io2, Azure Ultra Disk, GCP Hyperdisk)
Shared file access (teams, CMS)	File (EFS, Azure Files Premium, Filestore)
Unstructured data, backups, media, archives	Object (S3, GCS, Azure Blob)
Big data lakes / analytics / ML	Object + Lakehouse (S3 + Delta Lake / Iceberg / Hudi)
Lowest-cost long-term archival	Object Archive / Glacier Deep Archive / Coldline
Regulatory / sensitive data	Private / Hybrid with customer-managed keys
Avoid vendor lock-in	S3-compatible object storage (MinIO, Ceph)

Table 3.4: Quick selection guide for cloud storage approaches [**?**, **?**, **?**]

In current practice (2026), most new cloud-native applications begin with **object storage** due to its extreme scalability, cost-efficiency, and rich ecosystem integration. Performance-critical workloads still rely on block storage, while legacy/shared-file use cases continue to use managed file storage [**?**, **?**].

Always combine chosen storage with:

- Lifecycle policies and intelligent tiering
- Server-side encryption (SSE-KMS / customer-managed keys)
- Multi-AZ / multi-region replication
- Observability (CloudWatch, Stackdriver, Azure Monitor)

latex

3.0.4 Python Basics

Python is a high-level, interpreted, general-purpose programming language known for its readability, simplicity, and versatility. It supports multiple programming paradigms (procedural, object-oriented, functional) and is widely used in cloud computing, data science, web development, automation, AI/ML, DevOps, and scripting.

Created by Guido van Rossum and first released in 1991, Python emphasizes code readability with significant whitespace and a philosophy summarized in "The Zen of Python" (`import this`).

Key features:

- Simple and easy-to-learn syntax

- Dynamically typed (no need to declare variable types)
- Extensive standard library ("batteries included")
- Cross-platform (Windows, macOS, Linux)
- Large ecosystem of third-party packages via PyPI
- Strong community support and excellent documentation

Installing Python

There are several ways to install Python depending on your operating system and use case.

Recommended methods (2026):

1. **Official installer** — from `https://www.python.org/downloads/`
   - Download the latest stable version (Python 3.12 or 3.13 recommended)
   - Check `Add Python to PATH` during installation (Windows)
   - Verify: `python -version` or `python3 -version`
2. **Anaconda / Miniconda** — best for data science, machine learning, scientific computing
   - Includes Python + popular packages (NumPy, pandas, Jupyter, matplotlib, scikit-learn, etc.)
   - Download from `https://www.anaconda.com/download`
3. **Operating system package managers**
   - Ubuntu/Debian: `sudo apt update && sudo apt install python3 python3-pip`
   - macOS (Homebrew): `brew install python`
   - Windows (winget): `winget install -e -id Python.Python.3`

Verify installation:

```
python --version    # or python3 --version
pip --version       # or pip3 --version
```

Python Data Types and Data Structures

Python has several built-in data types and powerful collection types (data structures).

Basic / Primitive Data Types:

- `int` — arbitrary precision integers: `42`, `-10`, `10**100`
- `float` — floating-point numbers: `3.14`, `2.5e-4`
- `bool` — `True`, `False`
- `str` — immutable text: `"Hello"`, `'Python'`, triple quotes for multiline

- `NoneType` — `None` (absence of value)

Collection / Data Structure Types:

- **list** — ordered, mutable, allows duplicates

```
fruits = ["apple", "banana", "cherry", "apple"]
fruits.append("orange")
print(fruits[1])      # banana
```

- **tuple** — ordered, immutable

```
point = (10, 20)
# point[0] = 15   # TypeError - cannot modify
```

- **dictionary** (dict) — unordered (insertion order preserved since 3.7), mutable, key-value pairs

```
student = {"name": "Eswar", "age": 30, "city": "Chennai"}
print(student["name"])          # Eswar
student["grade"] = "A"
```

- **set** — unordered, mutable, no duplicates

```
unique_nums = {1, 2, 2, 3, 4}   # {1, 2, 3, 4}
unique_nums.add(5)
```

Control Flow

Conditional statements:

```
age = 20
if age >= 18:
    print("Adult")
elif age >= 13:
    print("Teen")
else:
    print("Child")
```

Loops:

```
# for loop
for i in range(5):          # 0 to 4
    print(i)

# while loop
count = 0
while count < 3:
    print("Hello")
    count += 1

# break, continue, pass
for num in range(10):
```

```
    if num == 7:
        break
    if num % 2 == 0:
        continue
    print(num)
```

Functions

Functions are defined using the `def` keyword.

```
def greet(name="Guest", greeting="Hello"):
    """This function greets a person."""
    return f"{greeting}, {name}!"

print(greet("Eswar"))           # Hello, Eswar!
print(greet(greeting="Hi"))     # Hi, Guest!

# Lambda (anonymous) function
square = lambda x: x * x
print(square(5))                # 25
```

Modules and Packages

Module — single `.py` file containing functions, classes, variables.
Package — directory containing modules + `__init__.py`.

```
# mymodule.py
def add(a, b):
    return a + b

# main.py
import mymodule
print(mymodule.add(10, 20))     # 30

# or
from mymodule import add
print(add(5, 7))                # 12

# Standard library examples
import math
import datetime
import random
import os
```

Installing external packages:

```
pip install requests pandas numpy matplotlib fastapi uvicorn boto3
```

File Handling

```python
# Reading a file
with open("data.txt", "r", encoding="utf-8") as file:
    content = file.read()           # entire content
    lines = file.readlines()        # list of lines

# Writing a file
with open("output.txt", "w", encoding="utf-8") as file:
    file.write("Hello, Cloud Computing!\n")
    file.writelines(["Line 1\n", "Line 2\n"])

# Appending
with open("log.txt", "a") as file:
    file.write("New log entry\n")

# Working with CSV
import csv
with open("students.csv", "r") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

Date and Time Operations

```python
from datetime import datetime, timedelta, date

# Current date and time
now = datetime.now()
print(now)                          # 2026-02-27 21:58:45.123456

# Formatting
print(now.strftime("%Y-%m-%d %H:%M:%S"))    # 2026-02-27 21:58:45

# Date arithmetic
tomorrow = now + timedelta(days=1)
one_week_ago = now - timedelta(weeks=1)

# Date only
today = date.today()
print(today)                        # 2026-02-27
```

Classes (Object-Oriented Programming)

Python supports full object-oriented programming with classes, inheritance, polymorphism, and encapsulation.

```python
class CloudService:
    """Base class for cloud services"""

    def __init__(self, name, provider):
```

```
        self.name = name
        self.provider = provider
        self.status = "Running"

    def stop(self):
        self.status = "Stopped"
        print(f"{self.name} stopped.")

    def info(self):
        return f"{self.name} ({self.provider}) - {self.status}"

class StorageService(CloudService):
    """Derived class for storage services"""

    def __init__(self, name, provider, capacity_gb):
        super().__init__(name, provider)
        self.capacity_gb = capacity_gb

    def info(self):  # Method overriding
        return f"{super().info()} | Capacity: {self.capacity_gb}
           GB"

# Usage
s3 = StorageService("MyBucket", "AWS", 1000)
print(s3.info())
s3.stop()
```

This covers the core Python basics needed for cloud computing tasks (scripting, automation, SDK usage, data processing, REST APIs, etc.).

Chapter 4

UNIT-3: Python for Cloud Computing

Python has become one of the most popular languages for cloud development due to its simplicity, readability, vast ecosystem, and excellent official SDK support from all major cloud providers.

4.1 Python for Major Cloud Platforms

4.1.1 Python for Amazon Web Services (AWS)

AWS provides **Boto3** — the official Python SDK for interacting with almost all AWS services.

Key features:

- Supports EC2, S3, Lambda, DynamoDB, SQS, SNS, RDS, IAM, CloudWatch, etc.
- Resource-based and client-based APIs
- Session & credential management (including IAM roles, SSO, MFA)

Example: List S3 buckets and upload a file

```
import boto3
from botocore.exceptions import ClientError

# Using default credentials (environment variables / IAM role /
   ~/.aws/credentials)
s3 = boto3.client('s3')

# List all buckets
try:
    response = s3.list_buckets()
    print("Existing␣buckets:")
    for bucket in response['Buckets']:
        print(f"␣␣{bucket['Name']}")
except ClientError as e:
    print(f"Error:␣{e}")

# Upload a file to S3
try:
    s3.upload_file("local-report.pdf", "my-company-docs", "reports
       /2026/Q1-report.pdf")
```

```
    print("Upload successful")
except ClientError as e:
    print(f"Upload failed: {e}")
```

4.1.2 Python for Google Cloud Platform (GCP)

GCP provides modular client libraries under the **google-cloud-*** namespace.

Important libraries:

- google-cloud-storage
- google-cloud-pubsub
- google-cloud-bigquery
- google-cloud-functions, google-cloud-compute

Example: Upload file to Cloud Storage

```
from google.cloud import storage

def upload_blob(bucket_name, source_file_name,
   destination_blob_name):
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    blob.upload_from_filename(source_file_name)
    print(f"File {source_file_name} uploaded to {
       destination_blob_name}.")

# Usage
upload_blob("my-analytics-bucket", "sales-2026.csv", "data/sales-
   q1-2026.csv")
```

4.1.3 Python for Microsoft Azure (Windows Azure)

Azure provides the **azure-*** SDK family (most popular: azure-storage-blob, azure-identity).

Example: Upload file to Azure Blob Storage

```
from azure.storage.blob import BlobServiceClient
import os
from dotenv import load_dotenv

load_dotenv()  # Load from .env file

connect_str = os.getenv("AZURE_STORAGE_CONNECTION_STRING")
blob_service_client = BlobServiceClient.from_connection_string(
   connect_str)

container_name = "documents"
file_path = "project-proposal.pdf"
```

```
blob_name = "2026/proposals/project-proposal-v2.pdf"

container_client = blob_service_client.get_container_client(
   container_name)
with open(file_path, "rb") as data:
    blob_client = container_client.upload_blob(name=blob_name,
       data=data, overwrite=True)
    print(f"Uploaded␣{file_path}␣to␣{blob_client.url}")
```

4.1.4 Python for MapReduce

Python is widely used with Hadoop via **Hadoop Streaming** and the popular **mrjob** library.

mrjob example — Word Frequency Count

```
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"[\w']+")

class MRWordFrequency(MRJob):
    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

    def combiner(self, word, counts):
        yield word, sum(counts)

    def reducer(self, word, counts):
        yield word, sum(counts)

if __name__ == '__main__':
    MRWordFrequency.run()
```

Run locally:

```
python wordfreq.py input.txt > output.txt
```

Run on Hadoop cluster:

```
python wordfreq.py -r hadoop hdfs:///input/*.txt -o hdfs:///output/wordfreq
```

4.1.5 Python Packages of Interest for Cloud

4.1.6 Python Web Application Frameworks

Two most popular frameworks for cloud-native APIs and applications:

- **FastAPI** — modern, async-first, automatic OpenAPI docs, type hints, very fast
- **Flask** — lightweight, flexible, minimalistic
- **Django** — full-featured (ORM, admin panel, authentication), batteries-included

FastAPI is currently (2026) the most recommended for cloud microservices and REST APIs.

Package	Purpose / Common Use in Cloud
boto3	Official AWS SDK
google-cloud-*	Official Google Cloud client libraries
azure-*	Official Microsoft Azure SDK family
mrjob	Simplified MapReduce jobs in Python
apache-airflow	Workflow orchestration & scheduling
fastapi + uvicorn	Modern, high-performance REST API framework
flask	Lightweight web framework
requests / httpx	HTTP client for calling cloud APIs
pandas + numpy	Data manipulation & analysis
paramiko / fabric	SSH automation for cloud servers
python-dotenv	Load environment variables (.env files)
pydantic	Data validation & settings management (used in FastAPI)

Table 4.1: Popular Python packages for cloud development

4.1.7 Designing a RESTful Web API

RESTful principles:

- Use standard HTTP methods: GET (read), POST (create), PUT/PATCH (update), DELETE (delete)
- Stateless — each request contains all necessary information
- Resource-based URIs: `/users`, `/users/123`, `/orders/456/items`
- Proper status codes: 200, 201, 400, 401, 403, 404, 500, etc.
- JSON as standard response format
- Versioning (optional): `/v1/users`, header-based, etc.

FastAPI minimal REST API example

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List

app = FastAPI(title="Cloud Items API")

class Item(BaseModel):
    id: int
    name: str
    price: float
    is_available: bool = True

items_db = []

@app.get("/items/", response_model=List[Item])
def get_items():
    return items_db

@app.post("/items/", response_model=Item, status_code=201)
```

```
def create_item(item: Item):
    if any(i.id == item.id for i in items_db):
        raise HTTPException(status_code=400, detail="Item already
           exists")
    items_db.append(item)
    return item

@app.get("/items/{item_id}", response_model=Item)
def get_item(item_id: int):
    for item in items_db:
        if item.id == item_id:
            return item
    raise HTTPException(status_code=404, detail="Item not found")
```

Run: `uvicorn main:app -reload`
Auto-generated docs appear at: `http://127.0.0.1:8000/docs`

4.2 Cloud Application Development in Python

4.2.1 Design Approaches

Common modern approaches for building cloud applications in Python:

- Microservices architecture (FastAPI services + Docker + Kubernetes)
- Serverless functions (AWS Lambda, Google Cloud Functions, Azure Functions)
- Containerized apps (Docker + Kubernetes / ECS / Cloud Run)
- Event-driven systems (Pub/Sub, SQS, Kafka + Python consumers)
- 12-Factor App principles

4.2.2 Image Processing App (Serverless Example – AWS Lambda)

Goal: Automatically resize & compress images uploaded to S3

```
from PIL import Image
import boto3
import io
import urllib.parse

s3 = boto3.client('s3')

def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['
       object']['key'])

    # Download image
    obj = s3.get_object(Bucket=bucket, Key=key)
    img_data = obj['Body'].read()
```

```
    # Resize & compress
    img = Image.open(io.BytesIO(img_data))
    img.thumbnail((800, 800))

    output = io.BytesIO()
    img.save(output, format='JPEG', quality=85, optimize=True)
    output.seek(0)

    # Save resized version
    new_key = f"resized/{key}"
    s3.put_object(
        Bucket=bucket,
        Key=new_key,
        Body=output,
        ContentType='image/jpeg'
    )

    return {
        'statusCode': 200,
        'body': f"Resized image saved as {new_key}"
    }
```

Trigger: S3 Event Notification → Lambda on object creation.

4.2.3 Document Storage App (Azure Blob Example)

Goal: Upload and generate shareable URL for PDF documents

```
from azure.storage.blob import BlobServiceClient,
   generate_blob_sas, BlobSasPermissions
from datetime import datetime, timedelta
import os

connect_str = os.getenv("AZURE_STORAGE_CONNECTION_STRING")
blob_service_client = BlobServiceClient.from_connection_string(
   connect_str)

def upload_document(file_path, container="documents"):
    blob_name = os.path.basename(file_path)
    container_client = blob_service_client.get_container_client(
       container)

    with open(file_path, "rb") as data:
        blob_client = container_client.upload_blob(name=blob_name,
            data=data, overwrite=True)

    # Generate SAS URL (valid for 7 days)
    sas_token = generate_blob_sas(
        account_name=blob_client.account_name,
        container_name=container,
        blob_name=blob_name,
        account_key=blob_service_client.credential.account_key,
```

```
        permission=BlobSasPermissions(read=True),
        expiry=datetime.utcnow() + timedelta(days=7)
    )

    sas_url = f"{blob_client.url}?{sas_token}"
    return sas_url
```

4.2.4 MapReduce App (mrjob – Word Frequency on Cloud Storage)

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import re

WORD_RE = re.compile(r"[\w']+")

class MRCloudWordCount(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_words,
                   combiner=self.combiner_count_words,
                   reducer=self.reducer_count_words)
        ]

    def mapper_get_words(self, _, line):
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def combiner_count_words(self, word, counts):
        yield (word, sum(counts))

    def reducer_count_words(self, word, counts):
        yield (word, sum(counts))

if __name__ == '__main__':
    MRCloudWordCount.run()
```

Run on cloud storage (e.g. S3):

```
python wordcount.py s3://my-bucket/logs/*.log -r emr --no-output
```

4.2.5 Social Media Analytics App (Twitter/X Sentiment Example)

Goal: Fetch recent tweets and perform basic sentiment analysis

```
import tweepy
from textblob import TextBlob
import os
from dotenv import load_dotenv

load_dotenv()
```

```
auth = tweepy.OAuth1UserHandler(
    consumer_key=os.getenv("TWITTER_API_KEY"),
    consumer_secret=os.getenv("TWITTER_API_SECRET"),
    access_token=os.getenv("TWITTER_ACCESS_TOKEN"),
    access_token_secret=os.getenv("TWITTER_ACCESS_SECRET")
)

api = tweepy.API(auth)

def analyze_tweets(query, count=100):
    tweets = api.search_tweets(q=query, lang="en", count=count,
       tweet_mode="extended")

    for tweet in tweets:
        text = tweet.full_text
        analysis = TextBlob(text)
        polarity = analysis.sentiment.polarity

        sentiment = "Positive" if polarity > 0 else "Negative" if
           polarity < 0 else "Neutral"
        print(f"Tweet: {text[:80]}... | Sentiment: {sentiment} ({
           polarity:.2f})")

# Usage
analyze_tweets("cloud computing", count=50)
```

Note: For production use, prefer Twitter API v2 with `tweepy.Client` and Bearer Token.

This completes a comprehensive coverage of Unit-3 topics with practical, cloud-relevant examples.

Chapter 5

UNIT-4: Big Data, Multimedia and Tuning

5.1 Big Data Analytics

5.1.1 Introduction to Big Data Analytics

Big Data Analytics refers to the process of examining large and varied data sets (Big Data) to uncover hidden patterns, unknown correlations, market trends, customer preferences, and other useful business information.

5 Vs of Big Data (core characteristics):

- **Volume** — terabytes to petabytes of data
- **Velocity** — high speed of data generation and processing (streaming)
- **Variety** — structured, semi-structured, unstructured (logs, images, video, text)
- **Veracity** — uncertainty, noise, inconsistency in data
- **Value** — extracting meaningful insights for decision making

Types of Analytics:

- Descriptive — What happened? (reports, dashboards)
- Diagnostic — Why did it happen? (root cause analysis)
- Predictive — What will happen? (machine learning models)
- Prescriptive — What should we do? (optimization, recommendation)

Tools & Frameworks: Hadoop, Spark, Flink, Kafka, Hive, Pig, HBase, NoSQL databases, ML libraries (scikit-learn, TensorFlow, PyTorch).

5.1.2 Clustering Big Data

Clustering is an unsupervised machine learning technique that groups similar data points into clusters based on features.

Challenges in Big Data Clustering:

- Massive volume → cannot fit in memory
- High dimensionality → curse of dimensionality

- Velocity → need incremental / streaming clustering
- Variety → mixed data types

Popular scalable clustering algorithms:

- **K-Means** — partitioning (centroid-based), parallelized in Spark MLlib
- **BIRCH** — hierarchical, balanced iterative reducing and clustering using hierarchies
- **DBSCAN** — density-based (good for arbitrary shapes), parallel versions exist
- **CURE** — hierarchical with random sampling + shrinking
- **Streaming K-Means / CluStream** — for real-time data streams

Example in PySpark (K-Means):

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=["feature1", "feature2"],
   outputCol="features")
data = assembler.transform(df)

kmeans = KMeans().setK(5).setSeed(1)
model = kmeans.fit(data)
predictions = model.transform(data)
```

5.1.3 Classification of Big Data

Classification is a supervised learning task that predicts categorical labels for new data points.

Big Data Classification Challenges: Same as clustering + labeled data scarcity, imbalance, distributed training.

Scalable classification algorithms:

- Logistic Regression, SVM (linear), Naive Bayes — naturally parallelizable
- Decision Trees / Random Forest — distributed in Spark ML, H2O
- Gradient Boosted Trees — XGBoost, LightGBM, CatBoost (distributed versions)
- Deep Learning — distributed training with TensorFlow, PyTorch + Horovod / Spark

Example: Spark ML Random Forest Classifier

```
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(featuresCol="features", labelCol="
   label", numTrees=100)
model = rf.fit(train_data)
predictions = model.transform(test_data)
```

5.1.4 Recommendation Systems

Recommendation systems suggest items (products, movies, music, articles) that users may like.

Types:

- **Content-based** — recommend similar items based on item features
- **Collaborative Filtering** — user-item interactions (user-based / item-based)
- **Hybrid** — combine content + collaborative
- **Context-aware** — include time, location, device

Big Data scale approaches:

- Matrix Factorization (ALS – Alternating Least Squares) — Spark MLlib standard
- Deep learning-based (Neural Collaborative Filtering, Wide & Deep, DeepFM)
- Graph-based (GraphSAGE, PinSage for Pinterest-scale)

Example: ALS in Spark

```
from pyspark.ml.recommendation import ALS

als = ALS(maxIter=10, regParam=0.1, userCol="userId", itemCol="
    movieId", ratingCol="rating")
model = als.fit(training)
recommendations = model.recommendForAllUsers(10)
```

5.2 Multimedia Cloud

5.2.1 Introduction

Multimedia Cloud combines cloud computing with multimedia services (audio, video, images, animation) to provide scalable, on-demand storage, processing, streaming, transcoding, and delivery of rich media content.

Key advantages:

- Elastic compute/storage for transcoding & streaming
- Global CDN integration for low-latency delivery
- Pay-per-use cost model
- Support for live & on-demand content

5.2.2 Case Study: Live Video Streaming App

Architecture (typical modern live streaming):

- Capture → Encoder (OBS, FFmpeg) → Ingest server (RTMP / WebRTC)
- Cloud: AWS MediaLive / Elemental, Azure Media Services, Google Cloud Live Streaming
- Transcoding → adaptive bitrate (HLS / DASH)
- Distribution → CDN (CloudFront, Akamai, Azure CDN)
- Playback → HTML5 video player (Video.js, Shaka Player)

Technologies: WebRTC (ultra-low latency), HLS (Apple), MPEG-DASH (standard), CMAF.

5.2.3 Streaming Protocols

Protocol	Latency	Use Case / Notes
RTMP	5–10 sec	Legacy ingest protocol (still widely used for live push)
RTSP	Low	Used in surveillance cameras
HLS (HTTP Live Streaming)	10–30 sec	Apple ecosystem, adaptive bitrate, widely supported
MPEG-DASH	5–30 sec	Open standard, codec-agnostic, adaptive
WebRTC	<1–2 sec	Ultra-low latency (interactive live, gaming)
SRT	<2 sec	Secure Reliable Transport – low-latency over unreliable networks
LL-HLS	2–5 sec	Low-Latency HLS (Apple 2020+)
WebRTC + CMAF	sub-second	Emerging for ultra-low latency adaptive streaming

Table 5.1: Comparison of Popular Streaming Protocols

5.2.4 Case Study: Video Transcoding App

Goal: Convert uploaded videos to multiple resolutions/formats for adaptive streaming.

Cloud-native architecture (AWS example):

- Upload to S3 → Trigger AWS Lambda / Step Functions
- Use AWS Elemental MediaConvert for transcoding
- Output: multiple bitrates/resolutions to S3
- Create HLS/DASH manifests
- Distribute via CloudFront CDN

Python + boto3 snippet (trigger MediaConvert job):

```
import boto3

client = boto3.client('mediaconvert', region_name='ap-south-1')

response = client.create_job(
    Role='arn:aws:iam::123456789012:role/MediaConvertRole',
    Settings={
        'Inputs': [{'FileInput': 's3://input-bucket/video.mp4'}],
        'OutputGroups': [{
            'Name': 'File Group',
            'Outputs': [{'ContainerSettings': {'Container': 'MP4'
               }}]
        }]
    }
)
```

5.3 Cloud Application Benchmarking and Tuning

5.3.1 Introduction

Benchmarking measures performance of cloud applications under different workloads to identify bottlenecks, validate scalability, and optimize cost/performance.

Tuning involves adjusting configurations, architecture, and resources to meet SLAs (latency, throughput, availability).

5.3.2 Workload Characteristics

Typical cloud workload types:

- **Transactional** — high concurrency, low latency (e-commerce, banking)
- **Analytical** — large scans, batch (data warehouse, ETL)
- **Streaming** — continuous ingestion/processing (IoT, logs)
- **Mixed** — combination (social media feeds)

Key parameters: request rate, data size, read/write ratio, burst vs steady.

5.3.3 Application Performance Metrics

- **Latency** — p50, p95, p99 (response time percentiles)
- **Throughput** — requests/sec, ops/sec, MB/sec
- **Error rate** — 4xx, 5xx, timeouts
- **Resource utilization** — CPU, memory, I/O, network
- **Cost efficiency** — $ per 1000 requests / TB processed
- **Availability / uptime** — SLA compliance

5.3.4 Design Considerations for Benchmarking Methodology

- Realistic workload — mimic production (YCSB, TPC, OpenMessaging)
- Repeatability — same conditions, seed values
- Warm-up period — avoid cold-start effects
- Multiple runs — statistical significance
- Scale-out testing — test horizontal scaling limits
- Cost monitoring — include cloud pricing

5.3.5 Benchmarking Tools

- **YCSB** — Yahoo! Cloud Serving Benchmark (NoSQL, key-value)
- **TPC-DS / TPC-H** — analytical / decision support
- **Apache JMeter** — load testing for web apps/APIs
- **Locust** — Python-based, easy scripting
- **wrk / wrk2** — high-performance HTTP benchmarking
- **Apache Bench (ab)** — simple HTTP
- **Hadoop/Spark Bench** — big data specific

5.3.6 Deployment Prototyping, Load Testing & Bottleneck Detection

Steps:

1. Prototype small-scale deployment (e.g., single AZ)
2. Run baseline load test (JMeter / Locust)
3. Monitor metrics (CloudWatch, Prometheus, Grafana)
4. Identify bottlenecks: CPU, memory, DB I/O, network, GC pauses
5. Scale resources / optimize code / change architecture
6. Re-test → iterate

Case Study: Hadoop Benchmarking

- Tools: HiBench, BigDataBench, TPC-DS on Spark/Hadoop
- Metrics: job completion time, throughput (MB/s), scalability
- Common bottlenecks: shuffle phase I/O, disk contention, network saturation
- Tuning: increase containers, adjust YARN memory, enable compression, use SSDs

This unit integrates Big Data processing, multimedia delivery, and performance engineering — all critical for modern cloud-native applications.

Chapter 6

UNIT-V: Applications and Issues in Cloud

6.1 Cloud Security

6.1.1 Introduction

Cloud security encompasses the policies, technologies, and controls that protect cloud computing environments, data, applications, and infrastructure from threats. Security responsibilities are shared between the cloud provider and customer (shared responsibility model).

Key differences from traditional security:

- Multi-tenancy → isolation is critical
- Dynamic scaling → security must be elastic
- Shared infrastructure → provider controls physical layer

6.1.2 CSA Cloud Security Architecture

The Cloud Security Alliance (CSA) defines a reference architecture with key domains:

- **Governance & Enterprise Risk Management** — policies, compliance, risk assessment
- **Legal & Electronic Discovery** — contracts, e-discovery, data sovereignty
- **Compliance & Audit** — standards (ISO 27001, SOC 2, GDPR, HIPAA)
- **Information Management & Data Security** — classification, encryption, DLP
- **Interoperability & Portability** — avoiding vendor lock-in
- **Traditional Security, Business Continuity & Disaster Recovery** — adapted to cloud
- **Data Center Operations** — provider-managed
- **Incident Response, Notification & Remediation** — shared responsibilities
- **Application Security** — secure SDLC in cloud
- **Encryption & Key Management** — critical control

- **Identity, Entitlement & Access Management (IAM)** — core focus
- **Virtualization & Container Security** — hypervisor, container runtime
- **Security as a Service** — cloud-native security offerings

6.1.3 Authentication, Authorization, and Identity Access Management (IAM)

Authentication — verifying identity (who you are)

- Methods: Passwords, MFA, SSO (SAML, OAuth 2.0, OpenID Connect), certificates, biometrics
- Cloud-native: AWS IAM, Azure AD, Google Identity Platform

Authorization — determining permissions (what you can do)

- Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC), Policy-Based
- Least privilege principle

IAM in Cloud (examples):

- **AWS IAM**: Users, Groups, Roles, Policies (JSON), MFA, temporary credentials (STS)
- **Azure AD**: Users, Groups, Enterprise Apps, Conditional Access, Privileged Identity Management (PIM)
- **Google Cloud IAM**: Primitive roles, Predefined roles, Custom roles, Service Accounts

6.1.4 Data Security and Key Management

Data Security Controls:

- Data classification (public, internal, confidential, restricted)
- Encryption at rest (SSE-S3, SSE-KMS, client-side encryption)
- Encryption in transit (TLS 1.2/1.3)
- Tokenization / masking
- Data Loss Prevention (DLP)

Key Management:

- Customer-managed keys (CMK) vs provider-managed
- Hardware Security Modules (HSM) — AWS CloudHSM, Azure Dedicated HSM
- Key rotation, versioning, revocation
- Bring Your Own Key (BYOK), Hold Your Own Key (HYOK)

6.1.5 Auditing

Cloud auditing ensures accountability, compliance, and forensics.

- **Logs**: API calls (AWS CloudTrail, Azure Monitor Logs, Google Cloud Audit Logs)
- **Monitoring**: Real-time alerts (CloudWatch Alarms, Azure Sentinel, Security Command Center)
- **Compliance reports**: PCI DSS, SOC, ISO 27001 reports
- **Immutable logs**: Write-once-read-many (WORM) storage

6.2 Cloud for Industry, Healthcare & Education

6.2.1 Cloud Computing for Healthcare

Applications:

- Electronic Health Records (EHR) storage & sharing
- Telemedicine & remote patient monitoring
- Medical image storage & analysis (PACS + AI)
- Genomics & personalized medicine (big data processing)
- Drug discovery & clinical trials (HPC in cloud)

Benefits: Scalability, disaster recovery, cost reduction, real-time collaboration **Challenges:** HIPAA/GDPR compliance, data privacy, latency for real-time apps **Examples:** AWS for Health, Azure Health Data Services, Google Cloud Healthcare API

6.2.2 Cloud Computing for Energy Systems

Applications:

- Smart grid management & demand-response
- Predictive maintenance of equipment (IoT + ML)
- Renewable energy forecasting (solar/wind)
- Energy trading platforms
- Carbon footprint tracking & reporting

Examples: AWS IoT for smart meters, Azure Digital Twins for grid simulation

6.2.3 Cloud Computing for Transportation Systems

Applications:

- Connected vehicles & V2X communication
- Traffic prediction & optimization
- Fleet management & logistics (route optimization)
- Autonomous driving data processing
- Real-time passenger information systems

Examples: Google Maps + Cloud, AWS for automotive (simulation & OTA updates)

6.2.4 Cloud Computing for Manufacturing Industry (Industry 4.0)

Applications:

- Digital twins of factories/machines
- Predictive maintenance (IIoT + ML)
- Supply chain visibility & optimization
- Quality control with computer vision
- Cloud-based MES (Manufacturing Execution Systems)

Examples: Siemens MindSphere, GE Predix, AWS RoboMaker

6.2.5 Cloud Computing for Education

Applications:

- E-learning platforms (LMS: Moodle, Canvas on cloud)
- Virtual classrooms & collaboration (Google Workspace, Microsoft Teams)
- Online labs & simulations (cloud HPC)
- Student data analytics & personalized learning
- Storage of educational content (videos, e-books)

Examples: Google Classroom, Microsoft Education, AWS Educate

6.3 Migrating into a Cloud

6.3.1 Introduction

Cloud migration is the process of moving applications, data, and workloads from on-premises or legacy environments to cloud infrastructure.

6.3.2 Broad Approaches to Migrating into the Cloud

- **Rehost** (Lift & Shift) — minimal changes, move as-is
- **Replatform** — minor optimizations (e.g., containerize, use managed DB)
- **Refactor / Re-architect** — redesign for cloud-native (microservices, serverless)
- **Repurchase** — switch to SaaS (e.g., CRM → Salesforce)
- **Retire** — decommission unused apps
- **Retain** — keep some workloads on-prem (hybrid)

6.3.3 The Seven-Step Model of Migration into a Cloud

1. **Assess** — inventory applications, dependencies, TCO analysis
2. **Isolate** — group applications by migration wave / priority
3. **Mapping** — choose migration strategy per app (6 Rs)
4. **Re-architect** — redesign selected apps (if refactor)
5. **Implement** — execute migration (tools: AWS Migration Hub, Azure Migrate, Google Migrate)
6. **Integrate & Test** — validate functionality, performance, security
7. **Iterate & Optimize** — monitor, cost optimize, modernize further

6.4 Organizational Readiness and Change Management in the Cloud Age

6.4.1 Introduction

Successful cloud adoption requires not only technology but also organizational readiness and effective change management.

6.4.2 Basic Concepts of Organizational Readiness

- Leadership support
- Skills & training (cloud certifications)
- Cultural shift (DevOps, agile mindset)
- Governance framework
- Financial model (CapEx → OpEx)

6.4.3 Drivers for Change: A Framework to Comprehend the Competitive Environment

PESTLE + Porter's Five Forces + SWOT adapted to cloud:

- Political — data sovereignty laws
- Economic — cost pressure, pay-per-use
- Social — remote work, digital transformation
- Technological — AI/ML, serverless, edge
- Legal — compliance (GDPR, HIPAA)
- Environmental — sustainability goals

6.4.4 Common Change Management Models

- **Kotter's 8-Step Model** — create urgency, build coalition, vision, communicate, empower, short-term wins, consolidate, anchor
- **ADKAR** — Awareness, Desire, Knowledge, Ability, Reinforcement
- **Lewin's Change Model** — Unfreeze → Change → Refreeze

6.4.5 Change Management Maturity Models

- Level 1: Ad-hoc / chaotic
- Level 2: Repeatable (some processes)
- Level 3: Defined (standardized)
- Level 4: Managed (measured)
- Level 5: Optimized (continuous improvement)

6.4.6 Organizational Readiness Self-Assessment

Key questions:

- Does leadership understand cloud benefits/risks?
- Are staff trained/certified?
- Is there a cloud governance board?
- Can finance handle OpEx billing?
- Are security & compliance teams involved early?

6.5 Legal Issues in Cloud Computing

6.5.1 Introduction

Legal challenges arise due to the distributed, multi-tenant, and cross-border nature of cloud services.

6.5.2 Data Privacy and Security Issues

- Compliance with GDPR, CCPA, HIPAA, DPDP Act (India), etc.
- Data residency / sovereignty requirements
- Breach notification obligations
- Right to be forgotten / data deletion
- Sub-processor risks (provider's subcontractors)

6.5.3 Cloud Contracting Models

- Standard Terms (non-negotiable — AWS, Azure, GCP)
- Negotiated Enterprise Agreements
- Key clauses to review:
  - Liability limitation
  - Data ownership & usage rights
  - Termination & data export
  - SLAs & penalties
  - Indemnification
  - Governing law & jurisdiction

6.5.4 Jurisdictional Issues Raised

- Which country's law applies? (governing law clause)
- Where is data physically stored? (regions)
- Cross-border data transfers (Schrems II, adequacy decisions)
- Government access requests (CLOUD Act – US, other national laws)
- Dispute resolution — arbitration vs courts

This unit covers critical non-technical aspects of cloud adoption — security, vertical applications, migration strategy, organizational change, and legal considerations — essential for real-world cloud deployment.

References

Textbooks

1. Arshdeep Bahga and Vijay Madisetti, *Cloud Computing: A Hands-on Approach*, Universities Press, 2016.

2. Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski, *Cloud Computing: Principles and Paradigms*, Wiley, 2011.

Reference Books

1. Rajkumar Buyya, Christian Vecchiola, and S. Thamarai Selvi, *Mastering Cloud Computing*, Tata McGraw Hill (TMH), 2013.

2. Arshdeep Bahga and Vijay Madisetti, *Cloud Computing: A Hands-on Approach*, (Reprint / Edition as per university recommendation), 2016.

3. Anthony T. Velte, Toby J. Velte, and Robert Elsenpeter, *Cloud Computing: A Practical Approach*, Tata McGraw Hill, Reprinted 2011.

4. Gautam Shroff, *Enterprise Cloud Computing*, Cambridge University Press, 2010.

5. George Reese, *Cloud Application Architectures: Building Applications and Infrastructure in the Cloud*, O'Reilly / SPD, Reprinted 2011.

6. K. Chandrasekaran, *Essentials of Cloud Computing*, CRC Press, 2014.

Note:

- All references are listed as per the details provided.
- Use the most recent editions available in your university library or bookstore.
- For assignments and exams, prefer primary citations from the textbooks.