**DAY 1**

**System Software :**

System software serves as the interface between the hardware and user applications. It

manages the hardware resources and provides the necessary environment for application

software to run

**Examples :**

*Operating Systems*

• **Windows 10:** Popular OS for personal computers and business environments.

• **Ubuntu:** A widely-used Linux distribution known for its user-friendliness.

• **Android:** Dominant mobile OS used in smartphones and tablets.

*Device Drivers*

• **NVIDIA GeForce Driver:** Enables high-performance graphics rendering on NVIDIA GPUs.

• **Realtek Network Driver:** Provides network connectivity for Realtek network interface cards.

*Utility Programs*

• **CCleaner:** Cleans unnecessary files and optimizes system performance.

• **Acronis True Image:** Provides comprehensive backup solutions for data protection.

*Firmware*

• **BIOS:** Initializes hardware during the booting process and provides runtime services for the

OS.

• **Router Firmware:** Manages the router's functions and network configurations.

*Shells and Command-Line Interfaces*

• **Bash:** Commonly used in Unix-like operating systems for scripting and command execution.

• **PowerShell:** A task automation framework from Microsoft with a command-line shell and

scripting language.

*System Management Tools*

• **Windows Task Manager:** Provides details about running applications, processes, and system

performance.

• **htop:** An interactive process viewer for Unix systems, offering real-time monitoring of

system resources.

*Virtual Machine Managers (Hypervisors)*

• **VMware ESXi:** Enterprise-level bare-metal hypervisor for running multiple virtual machines. • **Oracle VirtualBox:** A free and open-source hosted hypervisor for running virtual machines

on personal computers.

**Application Software**

Application software is designed to help users perform specific tasks or activities. Unlike

system software, which runs in the background, application software directly interacts with

the user.

**Examples**

*Productivity Software*

• **Microsoft Word:** A widely-used word processor for creating and editing documents.

• **Google Sheets:** A web-based spreadsheet application for creating and managing

spreadsheets online.

*Web Browsers*

• **Google Chrome:** A popular web browser known for its speed and extension support.

• **Safari:** The default web browser for macOS and iOS devices, known for its integration with

Apple's ecosystem.

*Media Players*

• **VLC Media Player:** An open-source media player that supports a wide range of audio and

video formats.

• **iTunes:** A media player, media library, and mobile device management application

developed by Apple.

*Graphics and Design Software*

• **Adobe Photoshop:** A professional image editing software used by photographers and

designers.

• **AutoCAD:** A computer-aided design (CAD) software used by architects, engineers, and

construction professionals.

*Communication Software*

• **Microsoft Teams:** A collaboration platform that combines chat, video meetings, file storage,

and application integration.

• **Zoom:** A video conferencing software that allows for virtual meetings and webinars.

*Web Applications*

• **Gmail:** A free email service provided by Google, accessible through a web browser.

• **YouTube:** A video-sharing platform where users can upload, view, and share videos. *Games*

• **Fortnite:** A popular battle royale game developed by Epic Games.

• **The Sims:** A life simulation game where players create and control virtual people.

*Business Software*

• **SAP:** Enterprise resource planning (ERP) software used to manage business operations and

customer relations.

• **Salesforce:** A cloud-based customer relationship management (CRM) platform for managing

sales, service, and marketing activities.

*Educational Software*

• **Khan Academy:** Provides free online courses, lessons, and practice in various subjects.

• **Duolingo:** A language learning app that offers lessons in multiple languages through

gamified exercises.

*Database Software*

• **MySQL:** An open-source relational database management system.

• **Oracle Database:** A multi-model database management system developed by Oracle

Corporation.

*Email Clients*

• **Microsoft Outlook:** An email client with calendar, task manager, and contact manager

functionality.

• **Mozilla Thunderbird:** A free and open-source email client developed by Mozilla.

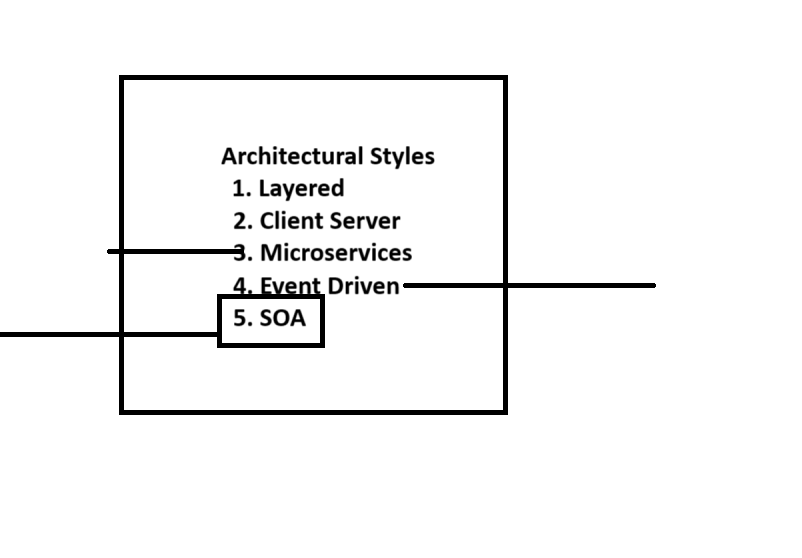
*Antivirus Software*

• **Norton Antivirus:** Provides protection against viruses, malware, and other online threats.

• **Avast:** Offers free and paid antivirus solutions for personal and business use.

These various types of application software cater to different user needs, providing tools for productivity, communication, entertainment and more.

DAY 2

**Architecture** **styles**:

**Design** **patterns**:

1. Singleton
2. Factory
3. Observer
4. Decorator

**Non**-**Functional** **Requirements**:

1. Performances
2. Security
3. Usability
4. Maintainablity
5. Scalability
6. Reliability

**Diagrams**:

1. UML
2. Class Diagrams
3. Sequence Diagrams
4. Component Diagrams
5. Deployment

**SOA**:

Service-Oriented Architecture (SOA) is a design approach where software components, called services, are provided to other components via a network. Each service encapsulates a specific functionality and can be independently deployed and managed. This modularity allows for easier maintenance, scalability, and integration of services.

Key Principles of SOA:

Interoperability: Services can interact with each other regardless of the platform or language they are written in. This is facilitated by standard protocols and interfaces.

Loose Coupling: Services maintain minimal dependencies on each other, allowing changes to be made to one service without significantly affecting others.

Abstraction: The internal workings of a service are hidden from the consumers, who interact with the service through a well-defined interface.

Granularity: Services are designed to be as fine-grained as needed, encapsulating specific business functionalities.

Components of SOA:

Service: The basic building block, which includes the service implementation, service contract, and service interface.

Service Provider: The entity that creates and maintains the services.

Service Consumer: The entity that uses the services.

Service Registry: A directory where services are listed, allowing consumers to find and use them.

Types of SOA:

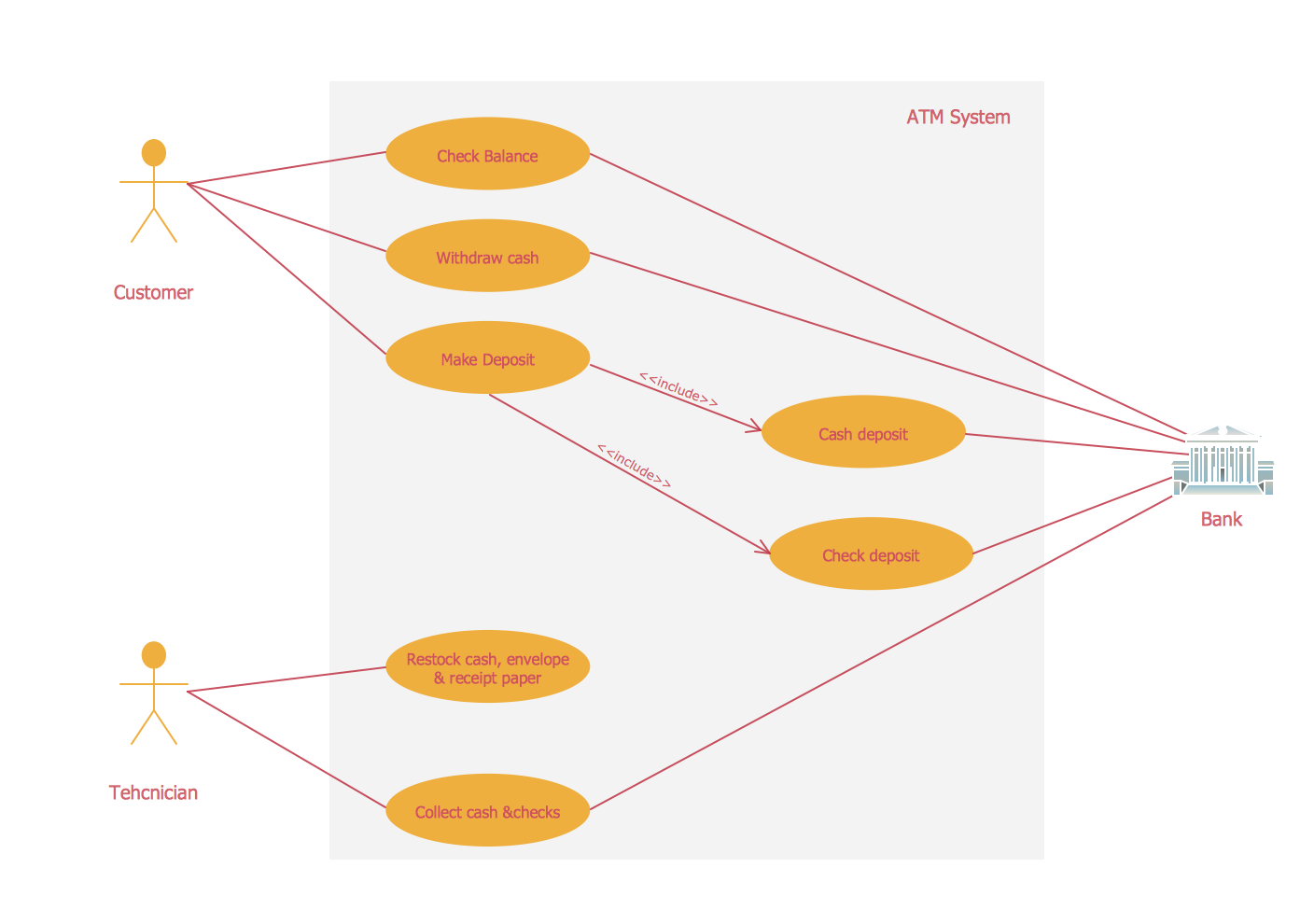
Enterprise Service Bus (ESB): A middleware that helps in the integration and communication between different services. It handles tasks like routing, protocol transformation, and message processing.

Web Services: SOA can be implemented using web services, which use standard web protocols like HTTP and XML to enable communication between applications over the internet.

Examples of SOA

Universal Logins: A system where a single authentication service handles logins for multiple applications within an organization.

Case Study: Service Oriented Architecture in ATM machine



### ATM Machine Case Study:

#### Overview:

An Automated Teller Machine (ATM) is a computerized telecommunications device that provides access to financial transactions in a public space without the need for a human cashier. ATMs are used for a variety of banking transactions, including cash withdrawals, deposits, account transfers, balance inquiries, and bill payments.

#### Key Components of an ATM:

1. **Hardware Components:**
   * **Card Reader:** Reads data from the ATM card's magnetic stripe or chip.
   * **Keypad:** Allows the customer to input their Personal Identification Number (PIN) and select transaction options.
   * **Display Screen:** Provides instructions and feedback to the user.
   * **Cash Dispenser:** Dispenses cash to the customer.
   * **Deposit Slot:** Accepts deposits of cash and checks.
   * **Receipt Printer:** Prints transaction receipts for the user.
   * **Network Connection:** Connects the ATM to the bank's central computer system for transaction processing.
2. **Software Components:**
   * **Operating System:** Manages the hardware and software resources of the ATM.
   * **Application Software:** Manages the user interface and transaction processing.
   * **Security Software:** Ensures the security of transactions and customer data.

#### Functional Requirements:

1. **Authentication:**
   * Validate the user by reading the card information and PIN.
   * Implement multi-factor authentication for enhanced security.
2. **Transaction Processing:**
   * Cash withdrawals, balance inquiries, deposits, and fund transfers.
   * Update the bank database in real-time to reflect transactions.
3. **User Interface:**
   * Provide clear instructions and feedback to the user through the display screen.
   * Offer multiple language options.
4. **Security:**
   * Encrypt data transmission between the ATM and the bank's server.
   * Secure the hardware to prevent tampering.
   * Implement fraud detection mechanisms.

#### Non-Functional Requirements

1. **Reliability:**
   * Ensure high availability and uptime.
   * Regular maintenance and monitoring.
2. **Performance:**
   * Fast transaction processing times.
   * Efficient handling of concurrent transactions.
3. **Usability:**
   * User-friendly interface.
   * Accessible design for users with disabilities.
4. **Scalability:**
   * Support for an increasing number of transactions and users.
   * Ability to add new features and services.
5. **Compliance:**
   * Adherence to banking regulations and standards.
   * Regular audits and security assessments.

#### Use Case Scenarios:

1. **Cash Withdrawal:**
   * User inserts the ATM card.
   * The system prompts the user to enter their PIN.
   * User selects the withdrawal option and enters the amount.
   * The system verifies the user's balance and dispenses cash.
   * The system prints a receipt and updates the user's account balance.
2. **Balance Inquiry:**
   * User inserts the ATM card and enters the PIN.
   * User selects the balance inquiry option.
   * The system retrieves and displays the account balance.
   * The system prints a receipt if requested.
3. **Deposit:**
   * User inserts the ATM card and enters the PIN.
   * User selects the deposit option.
   * The system prompts the user to insert cash or checks.
   * The system credits the user's account and prints a receipt.
4. **Fund Transfer:**
   * User inserts the ATM card and enters the PIN.
   * User selects the fund transfer option.
   * The system prompts the user to enter the recipient's account number and the transfer amount.
   * The system debits the user's account and credits the recipient's account.
   * The system prints a receipt.

#### Security Considerations:

1. **Physical Security:**
   * Install ATMs in secure locations with surveillance cameras.
   * Use tamper-evident seals and alarms.
2. **Data Security:**
   * Encrypt all data transmissions.
   * Use secure PIN entry devices.
   * Regularly update software to protect against vulnerabilities.
3. **User Security:**
   * Educate users about PIN safety and fraud prevention.
   * Implement transaction monitoring and alert systems.

#### Challenges and Solutions:

1. **Fraud and Theft:**
   * Challenge: Skimming devices and card cloning.
   * Solution: Use EMV chip technology and install anti-skimming devices.
2. **Technical Failures:**
   * Challenge: Hardware malfunctions and software bugs.
   * Solution: Regular maintenance, updates, and redundancy systems.
3. **User Error:**
   * Challenge: Incorrect PIN entries and transaction mistakes.
   * Solution: Provide clear instructions and user-friendly interfaces.
4. **Network Issues:**
   * Challenge: Connectivity problems and server downtime.
   * Solution: Use reliable network infrastructure and backup connections.

### Conclusion:

ATMs are an essential part of modern banking, offering convenient and secure access to financial services. The design and operation of ATMs require a combination of reliable hardware, robust software, and stringent security measures to ensure a seamless user experience and protect against various threats. Continuous improvements in technology and security are necessary to meet the evolving needs of users and maintain trust in the ATM system.

Event Driven Architecture:

Event-driven architecture (EDA) is a design paradigm where actions within a system are triggered by events. It allows for decoupled, scalable, and reactive systems. Below are types of event-driven architectures with examples:

Types of Event-Driven Architectures

Simple Event Processing

Description: Events are processed as they arrive without complex rules or conditions.

Example: Logging systems where each log entry is processed and stored as soon as it occurs.

Complex Event Processing (CEP)

Description: Multiple events are combined or analyzed to detect patterns, trends, or anomalies.

Example: Fraud detection systems in banking, where multiple transactions are analyzed to identify suspicious behavior.

Event Streaming

Description: Continuous flow of events is processed in real-time.

Example: Real-time analytics in social media platforms, where user interactions like likes, comments, and shares are processed to generate trends.

Event Sourcing

Description: State changes are captured as a series of events. The current state can be derived by replaying these events.

Example: Financial systems where each transaction is an event, and the account balance is derived by replaying all transactions.

Examples

IoT Devices

Description: Sensors generate events when certain conditions are met (e.g., temperature exceeds a threshold).

Example: Smart home systems where events from different sensors control lighting, heating, or security systems.

Microservices

Description: Services communicate by emitting and listening to events.

Example: An e-commerce platform where an order service emits an event when an order is placed, triggering inventory and shipping services to process the order.

Stock Market Systems

Description: Real-time processing of stock trades and price updates.

Example: Trading platforms where buy/sell orders are processed as events, and price changes are broadcasted to all subscribed systems and users.

Event-driven architectures enable systems to be more responsive and resilient by decoupling components and allowing them to react to changes asynchronously.

Case Study of Event-Driven Architecture in Networking

Event-driven architecture (EDA) is a design paradigm where system components communicate and respond to events asynchronously. This approach is particularly beneficial in networking and large-scale systems due to its scalability, responsiveness, and ability to decouple components. Here are two notable case studies that demonstrate the application of EDA in networking:

Immutable's Use of Amazon EventBridge

Immutable, an Australian gaming company, transitioned to an event-driven architecture using Amazon EventBridge to manage its rapid growth and enhance its infrastructure for Web3 games. Prior to this transition, Immutable faced challenges with scaling their Kubernetes-based infrastructure as the company expanded from 50 to nearly 300 employees within a year.

Implementation Details:

Amazon EventBridge: Used to handle events such as customer transactions and internal system notifications.

AWS Lambda: For serverless computing to run event-handling code without managing servers.

Microservices: Enabled different engineering teams to work independently and deploy services autonomously.

Benefits:

Scalability: Supported a 1,500% increase in monthly customer transactions.

Reliability: Improved system reliability by isolating issues to individual features, reducing the impact of failures.

Innovation: Increased the speed of feature releases and innovation by allowing teams to work independently.

This transition allowed Immutable to efficiently manage its growing customer base and maintain high performance and reliability standards for its gaming solutions​ (Amazon Web Services, Inc.)​​ (Amazon Web Services, Inc.)​.

Serverless Espresso at AWS re

The Serverless Espresso project at AWS re

is a practical demonstration of EDA using various AWS services to handle real-time orders for coffee.

Implementation Details:

EventBridge: Central to managing events such as new coffee orders.

AWS Step Functions: Orchestrates the workflow from order placement to fulfillment.

AWS Lambda: Executes business logic in response to events.

Other AWS Services: Includes API Gateway, S3, DynamoDB, and Cognito for managing different aspects of the application.

Benefits:

Cost-Effective: Running the workshop and processing 1000 requests costs as little as $1.

Scalability and Real-Time Processing: Demonstrates the ability to handle real-time transactions efficiently.

Educational Value: Provides a hands-on lab for understanding the practical implementation of EDA and serverless architectures.

Serverless Espresso serves as an educational tool to showcase the ease and efficiency of deploying event-driven architectures using serverless technologies, making it accessible and cost-effective for developers​ (IT Revolution)​.

Conclusion

Event-driven architectures in networking offer significant advantages in scalability, reliability, and independent deployment. Companies like Immutable and projects like Serverless Espresso highlight how EDA can be effectively utilized to manage large-scale operations and real-time processing efficiently. These case studies illustrate the practical benefits and implementation strategies for adopting EDA in complex networked system.

DAY 3

PPT Presentation: Service Oriented Architecture with ATM Machine

<https://docs.google.com/presentation/d/1cTNwW_PlxHn-pD-drCvAGLqCDc0xd7-k9JzESwxQjzc/edit#slide=id.g2ecaf48611f_0_5>

DAY 4

MVC (Model-View-Controller) and its variants

MVC (Model-View-Controller) is a software architectural pattern commonly used for developing user interfaces. It divides an application into three interconnected components: the Model, the View, and the Controller. This separation helps manage complexity, makes the code more modular, and improves the maintainability of the application.

**Core Components of MVC:**

1. **Model**:
   * Represents the application's data and business logic.
   * Directly manages the data, logic, and rules of the application.
   * Responds to requests for information and updates the state of the data.
2. **View**:
   * Represents the UI of the application.
   * Displays data from the Model to the user.
   * Sends user commands to the Controller.
3. **Controller**:
   * Acts as an intermediary between Model and View.
   * Receives input from the View, processes it (often by updating the Model), and returns the result to the View.

**Variants of MVC:**

1. **MVVM (Model-View-ViewModel)**:
   * Commonly used in frameworks like Angular and Knockout.
   * **Model**: Same as in MVC.
   * **View**: Same as in MVC.
   * **ViewModel**: An abstraction of the View, which contains the state of the View and binds to the Model. The ViewModel handles most of the UI logic.
2. **MVP (Model-View-Presenter)**:
   * Often used in desktop applications.
   * **Model**: Same as in MVC.
   * **View**: Interface that displays data and sends user interactions to the Presenter.
   * **Presenter**: Similar to the Controller but more closely tied to the View. It updates the View directly after processing data from the Model.
3. **MVW (Model-View-Whatever)**:
   * A flexible variant where the third component can vary depending on the framework or specific needs.
   * Examples include Model-View-Adapter (MVA), Model-View-ViewState (MVVS), etc.

**Key Differences and Usage:**

* **MVC**: Used in traditional web applications and some desktop applications.
* **MVVM**: Favored in data-binding scenarios, such as with Angular or WPF applications.
* **MVP**: Preferred in scenarios where the UI needs to be very testable or decoupled from the business logic.

.

**Categories of Design Patterns:**

1. **Creational Patterns**:
   * Deal with object creation mechanisms.
   * Aim to create objects in a manner suitable to the situation.
   * **Examples**:
     + **Singleton**: Ensures a class has only one instance and provides a global point of access to it.
     + **Factory Method**: Defines an interface for creating an object but lets subclasses alter the type of objects that will be created.
     + **Abstract Factory**: Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
     + **Builder**: Separates the construction of a complex object from its representation, allowing the same construction process to create different representations.
     + **Prototype**: Creates new objects by copying an existing object, known as the prototype.
2. **Structural Patterns**:
   * Deal with object composition.
   * Focus on how classes and objects can be composed to form larger structures.
   * **Examples**:
     + **Adapter**: Allows incompatible interfaces to work together.
     + **Decorator**: Adds new functionality to an object dynamically without altering its structure.
     + **Facade**: Provides a simplified interface to a complex subsystem.
     + **Composite**: Composes objects into tree structures to represent part-whole hierarchies.
     + **Proxy**: Provides a surrogate or placeholder for another object to control access to it.

Cloud Computing and Services

Cloud computing is a model for delivering computing services over the internet, allowing users to access and use shared resources, such as servers, storage, databases, networking, software, and analytics, without having to manage them directly. It offers scalable resources and economies of scale.

**Key Characteristics of Cloud Computing:**

1. **On-Demand Self-Service**: Users can provision resources as needed automatically, without requiring human intervention from the service provider.
2. **Broad Network Access**: Resources are accessible over the network and can be accessed through standard mechanisms by heterogeneous client platforms.
3. **Resource Pooling**: The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model.
4. **Rapid Elasticity**: Resources can be rapidly and elastically provisioned to scale out or in, commensurate with demand.
5. **Measured Service**: Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer.

**Types of Cloud Services:**

Cloud services can be broadly classified into three categories:

1. **Infrastructure as a Service (IaaS)**:
   * Provides virtualized computing resources over the internet.
   * Examples: Amazon Web Services (AWS) EC2, Google Compute Engine (GCE), Microsoft Azure Virtual Machines.
   * Users: System administrators and developers needing raw computing resources.
2. **Platform as a Service (PaaS)**:
   * Provides a platform allowing customers to develop, run, and manage applications without dealing with the infrastructure.
   * Examples: Google App Engine, Microsoft Azure App Services, Heroku.
   * Users: Developers who want to focus on coding and application logic without worrying about underlying infrastructure.
3. **Software as a Service (SaaS)**:
   * Delivers software applications over the internet, on a subscription basis.
   * Examples: Google Workspace (formerly G Suite), Microsoft Office 365, Salesforce.
   * Users: End-users who access applications via web browsers.

**Deployment Models:**

1. **Public Cloud**:
   * Services are delivered over the public internet and shared across multiple organizations.
   * Examples: AWS, Google Cloud Platform (GCP), Microsoft Azure.
2. **Private Cloud**:
   * Cloud infrastructure is dedicated to a single organization, offering greater control and security.
   * Examples: VMware Cloud, Open Stack.
3. **Hybrid Cloud**:
   * Combines public and private clouds, allowing data and applications to be shared between them.
   * Examples: AWS Outposts, Azure Stack.
4. **Community Cloud**:
   * Infrastructure shared by several organizations with common concerns (e.g., security, compliance).
   * Examples: Government cloud services, research institutions.

**Benefits of Cloud Computing:**

1. **Cost Efficiency**: Reduces the capital expense of buying hardware and software.
2. **Scalability**: Easily scales up or down based on demand.
3. **Accessibility**: Access services from anywhere with an internet connection.
4. **Disaster Recovery**: Provides robust backup and recovery solutions.
5. **Automatic Updates**: Providers manage software updates and security patches.

**Popular Cloud Service Providers:**

1. **Amazon Web Services (AWS)**
2. **Microsoft Azure**
3. **Google Cloud Platform (GCP)**
4. **IBM Cloud**
5. **Oracle Cloud**

**Emerging Trends in Cloud Computing:**

1. **Serverless Computing**: Abstracts the infrastructure management, allowing developers to run code in response to events without provisioning servers.
   * Examples: AWS Lambda, Azure Functions, Google Cloud Functions.
2. **Edge Computing**: Extends cloud computing to the edge of the network, enabling data processing closer to the source of data.
   * Examples: AWS Greengrass, Azure IoT Edge.
3. **Multi-Cloud Strategies**: Using multiple cloud services from different providers to avoid vendor lock-in and increase resilience.
4. **AI and Machine Learning Services**: Offering pre-built AI and ML models and tools as cloud services.

**Service Models: SaaS, PaaS, IaaS**

1. **Infrastructure as a Service (IaaS)**:
   * **Definition**: Provides virtualized computing resources over the internet.
   * **Components**: Virtual machines, storage, networks, and operating systems.
   * **Users**: System administrators and developers who need to manage hardware resources directly.
   * **Examples**: Amazon Web Services (AWS) EC2, Google Compute Engine (GCE), Microsoft Azure Virtual Machines.
   * **Use Cases**: Hosting websites, storing data, running virtual machines, and managing databases.
2. **Platform as a Service (PaaS)**:
   * **Definition**: Provides a platform allowing customers to develop, run, and manage applications without dealing with the underlying infrastructure.
   * **Components**: Operating systems, development frameworks, database management systems, and middleware.
   * **Users**: Developers who focus on creating and deploying applications without worrying about hardware and software infrastructure.
   * **Examples**: Google App Engine, Microsoft Azure App Services, Heroku.
   * **Use Cases**: Developing web applications, building APIs, and running development and testing environments.
3. **Software as a Service (SaaS)**:
   * **Definition**: Delivers software applications over the internet on a subscription basis.
   * **Components**: Fully functional applications hosted and managed by a cloud service provider.
   * **Users**: End-users who access software applications via web browsers.
   * **Examples**: Google Workspace (formerly G Suite), Microsoft Office 365, Salesforce.
   * **Use Cases**: Email, customer relationship management (CRM), collaboration tools, and enterprise resource planning (ERP).

**Advantages of Cloud Computing:**

1. **Cost Efficiency**: Reduces the capital expense of buying hardware and software. You pay only for the services you use.
2. **Scalability**: Easily scale up or down based on demand.
3. **Accessibility**: Access services from anywhere with an internet connection.
4. **Disaster Recovery**: Provides robust backup and recovery solutions.
5. **Automatic Updates**: Providers manage software updates and security patches.

**Popular Cloud Service Providers:**

1. **Amazon Web Services (AWS)**
2. **Microsoft Azure**
3. **Google Cloud Platform (GCP)**
4. **IBM Cloud**
5. **Oracle Cloud**

### Docker

Docker is a platform that allows developers to automate the deployment of applications inside lightweight, portable containers. It simplifies and accelerates the development workflow by providing a consistent environment from development to production.

#### Key Concepts of Docker:

1. Containerization:
   * Containers: Encapsulate an application and its dependencies into a single package, ensuring consistency across different environments.
   * Benefits: Portability, consistency, and isolation. Containers run the same regardless of where they are deployed.
2. Docker Images:
   * Images: Read-only templates used to create containers. They include everything needed to run an application, such as the code, runtime, libraries, and configuration files.
   * Building Images: Images are created using Dockerfiles, which are scripts containing a series of instructions on how to build the image.
3. Docker Hub and Registries:
   * Docker Hub: A cloud-based registry service where Docker users can create, manage, and distribute Docker images. It provides a centralized resource for container image discovery.
   * Private Registries: Organizations can set up their own registries to store and manage Docker images securely.

#### Basic Docker Commands:

* docker run: Creates and starts a new container from a specified image.
* docker build: Builds an image from a Dockerfile.
* docker pull: Downloads an image from a registry.
* docker push: Uploads an image to a registry.
* docker ps: Lists running containers.
* docker stop: Stops a running container.

### Kubernetes:

Kubernetes (often abbreviated as K8s) is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.

#### Key Components of Kubernetes:

1. Nodes:
   * Master Node: Manages the cluster, schedules workloads, and handles scaling.
   * Worker Nodes: Run the actual application workloads.
2. Pods:
   * The smallest deployable units in Kubernetes, consisting of one or more containers that share storage and network resources.
   * Each pod is assigned a unique IP address and can be scaled independently.
3. Services:
   * An abstraction that defines a logical set of pods and a policy by which to access them.
   * Services provide stable IP addresses and DNS names for the set of pods.
4. Deployments:
   * Manage the desired state of pods. They ensure that a specified number of replicas of a pod are running at any given time.
   * Allow rolling updates and rollbacks of applications.
5. ConfigMaps and Secrets:
   * ConfigMaps: Store configuration data as key-value pairs.
   * Secrets: Similar to ConfigMaps but used for sensitive data such as passwords, tokens, and keys.

#### Basic Kubernetes Commands (using kubectl):

* kubectl apply -f <file.yaml>: Applies a configuration from a YAML file to create or update resources.
* kubectl get pods: Lists all pods in the current namespace.
* kubectl describe pod <pod-name>: Provides detailed information about a specific pod.
* kubectl logs <pod-name>: Retrieves the logs of a specific pod.
* kubectl scale --replicas=<number> deployment/<deployment-name>: Scales the number of replicas for a deployment.
* kubectl delete pod <pod-name>: Deletes a specific pod.

### Integration of Docker and Kubernetes:

* Docker: Provides the container runtime for Kubernetes, meaning Kubernetes can orchestrate and manage Docker containers.
* Kubernetes: Extends the capabilities of Docker by providing tools for automated deployment, scaling, and management of containerized applications.