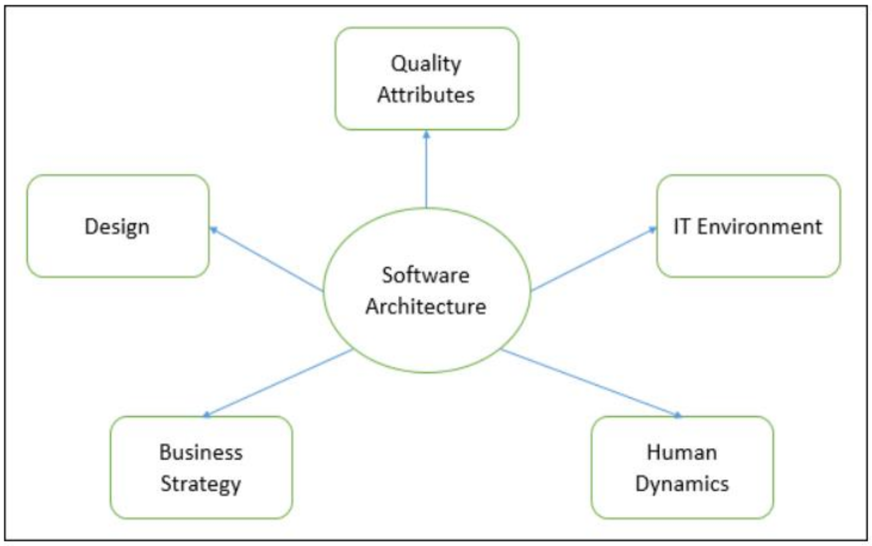
Unit-2

DevOps Influence on Architecture

1. Introducing software architecture

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

Software Architecture

Software architecture is the high-level structure of a software system, focusing on how the system's components are designed and interact with each other. It serves as the blueprint that guides the system's development, ensuring that it meets both functional (what the system does) and non-functional (how the system performs) requirements.

Key Aspects of Software Architecture:

1. **Components**: The major building blocks of the system, such as modules, services, libraries, databases, etc. These components encapsulate specific functionality and often interact with one another to form the complete system.
2. **Patterns and Styles**: Architectural patterns are reusable solutions to common problems. Examples include:
   - **Layered Architecture**: Divides the system into layers such as presentation, business logic, and data access.
   - **Microservices Architecture**: Breaks the system into smaller, independent services that can be developed and deployed separately.
   - **Monolithic Architecture**: A single unified application where all components are interconnected.
3. **Interfaces and Communication**: The way different components of the system communicate. This could involve APIs, messaging queues, or direct calls between services.
4. **Data Management**: How the system stores, retrieves, and manipulates data. This involves databases, file systems, caching mechanisms, and the design of how data is shared or processed.
5. **Non-Functional Requirements**: These are attributes like **performance**, **scalability**, **security**, **availability**, and **maintainability** that the architecture must address. For example, a highly scalable architecture might use microservices to allow different parts of the system to scale independently.

6. **Deployment and Infrastructure**: How the software is deployed and managed in production environments. This includes considerations for cloud infrastructure, containers, orchestration (e.g., Kubernetes), and monitoring.

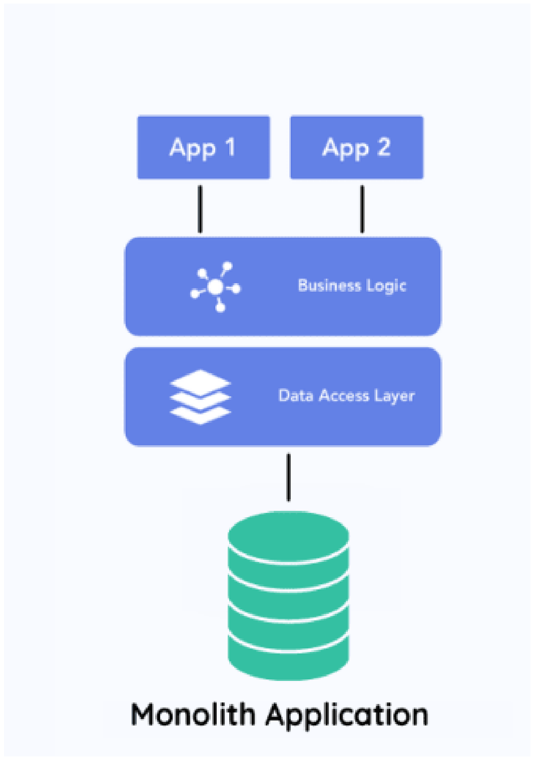
Why is Software Architecture Important?

- **System Quality**: A well-designed architecture ensures that the software is reliable, maintainable, and scalable over time. It helps meet both the expected functionality and performance requirements.
- **Guiding Development**: The architecture provides a blueprint that guides the development team throughout the software lifecycle, ensuring consistency and proper organization.
- **Managing Complexity**: As systems grow in size and functionality, a good architecture helps manage complexity by breaking the system into manageable, loosely coupled components.
- **Risk Mitigation**: By addressing potential problems early in the design phase, such as scalability or security concerns, software architecture can help avoid costly issues later.

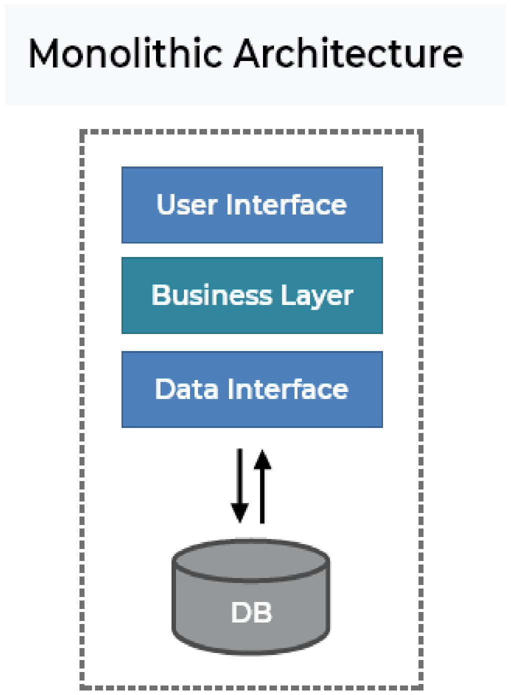
2. The monolithic scenario

Monolithic Service Architecture:-(Single block)

Monolithic Architecture in Devops refers to a traditional Software Design where all components of a application like front end, Back end and database all these are tightly integrated into a single codebase. Which means everything is kept together in a single folder.



(OR)



- In Monolithic Architecture, User Interface, Business Logic and data access layer works as a single unit, so its easier to manage all operations because all components of the software located in one place.
- In a Monolithic Architecture changes made in one layer can affect other layers Because they all are tightly integrated.

Example:-I created a college website in that college website, I want to add a new feature like adding new courses . So, whenever I change the presentation layers, I need to change the business logic layers and data access layer also.

Advantages:-

1. Simple to develop, test and deploy. Since Everything is in one place.
2. Communication between components is faster since everything in one place.
3. Developers have a clear understanding of entire application as all components are tightly integrated.
4. Deploying Monolithic applications is straight forward compared to micro services since there’s only one codebase to manage.

Disadvantages:-

1. It's challenging to update existing ones as the entire application needs to be modified.
2. A fault in one part of the application can bring down the entire system since everything is tightly coupled.
3. As everything is tightly coupled it is difficult for teams to work independently. Leading to coordination challenges.

3. Architecture rules of thumb

Architecture Rules of Thumb in Devops:-

- "Rules of Thumb" are general principles or guidelines widely used in a specific area. They are not strict rules, more like practical suggestions to achieve positive results in System design and implementation.
- In Devops architecture, They represent general best practices that teams can follow to achieve effective and efficient system design.

- Modularity:-Design systems in a modular fashion(i.e., break software into smaller modules that work independently) enabling easier maintenance, updates.
- Scalability:- Ensure that Architecture can handle increased load without sacrificing performance.
- Automation:-Automate process wherever possible, including deployment, testing and monitoring, to increase efficiency and reduce human error.
- Resilience:-Resilience means building systems that can handle failures by including backup plans, automatic switches.
- Infrastructure as Code(Ioc):-Manage Infrastructure like settings to track changes, recreate setups, and setup automatically.
- Continuous Integration/Continuous Deployment(CI/CD):-Automate CI/CD process by creating pipelines
- Monitoring and Logging:-
  Creating monitoring and logging systems to check system performance, and detect issues early.

- Security:- Integrate Security practices into every stage of the development and deployment process, including encryption & access control.
- Containerization:- Utilize Containerization technologies like docker tool to package and manage applications efficiency.
- Feedback loop:- Establish a feedback loop between development, operations and customers to continuously improve process and systems based on real world usage and feedback.

4. The separation of concerns

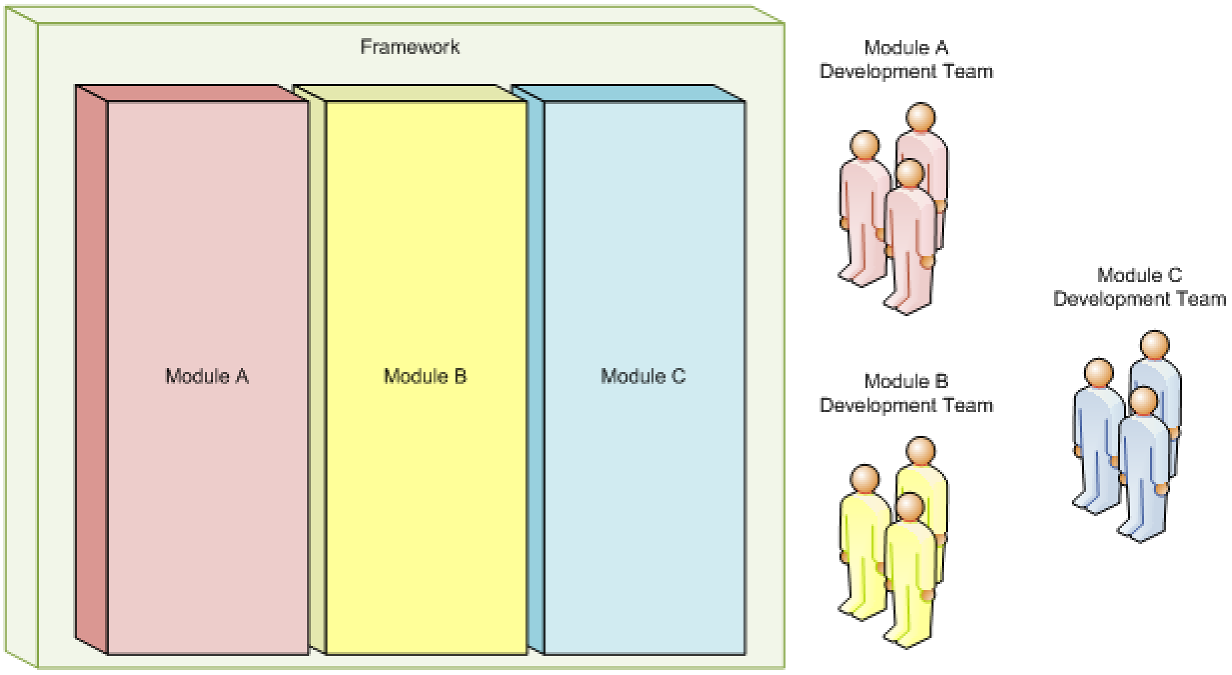
Separation of Concerns in Software Architecture:-

Separation of Concerns(SOC) means dividing entire program into various sections based on their functionalities, so that each section runs independently without effecting other services. Changes made in one section will not affect other sections.

→Separation of Concerns(Soc) will improve

- Maintainability:- Easier to update and fix parts of the code without affecting the whole system.
- Scalability:- Simple to expand the system to handle more load and add simple to add new features
- Readability:-Code is easier to Understand, making it simpler for developers to work with that code.
- Debugging:-Simple to identify and fix bugs

Example:- I am creating a college application. My college application contains four pages, so I will divide it into four modules. The first module contains code related to the student login page, the second module contains code related to the student results page, the third module contains code related to the student attendance page, and fourth module contains code related to faculty details page.Changes made in one module will not affect other modules because they all are separated.

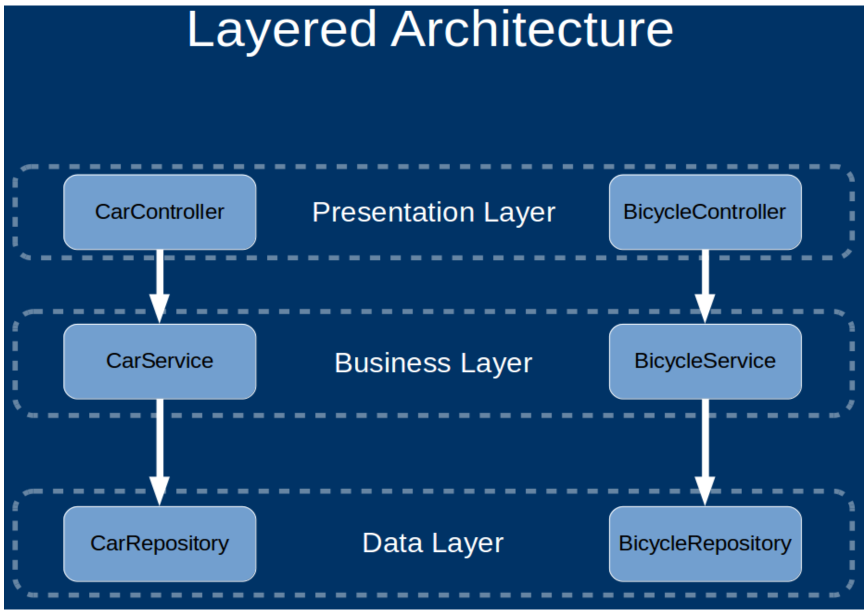

Framework
Module A
Module B
Module C
Module A Development Team
Module B Development Team
Module C Development Team

Real world Examples of SOC implementation:-

1. Model-View-Controller(MVC) Architecture:-
   MVC is a classic example of SOC because it divides on application into three components.
   Model:-Manages data and business logic
   View:- Handles the presentation layer
   Controller:-Interface between model and view process user inputs.
   Example:-Web Application
2. Micro Services Architecture:-
   In a Micro Service Architecture, an application is built as a collection of loosely coupled services. Where each service will perform certain task.
   Ex:-E-commerce Platform
3. Client-Server Architecture:-
   This Architecture Splits an application into two main components. They are client and service
   - Client:-Front end that interacts with users
   - Server:-Backend that process User requests and manages data

   Example:-Social Media Applications

4. Layered Architecture:-It divides an application into layers, each layer has specific role.



Example:-Online Banking System

5. Handling database migrations

Handling Database Migrations in Devops:-

→Database migration means transferring data from one database to another. This process can not only include just moving the data itself, but also transforming it to fit the new database structure(schema), updating databse structure, and ensuring data integrity and consistency throughout the migration.

→In Devops handling database migrations means managing the process of moving data from one databse to another, while ensuring it's done smoothly and without disturbances. This is important because changes to the databse structure or data need to be matched with application updates to avoid errors.

Techniques Used for Database Migration in Devops:-

1. Version Control(Git):-One of the essential techniques for database migration in devops is to use version control for both the application code and the databse schema(Structure). Tools like Liquibase and Flyway can

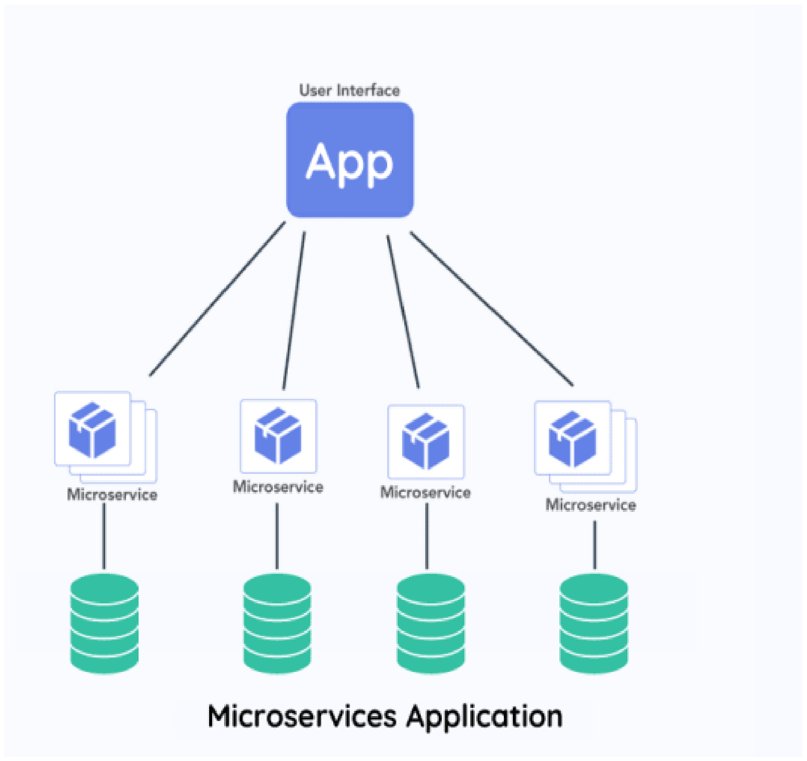
be integrated into CI/CD pipelines to automatically apply database schema changes when corresponding changes are made to the code.

2. Schema Migration Tools:-
   These tools automate making and running SQL Scripts that change your database structure(Schema). They prevent mistakes and reduce downtime. Examples are SQl server Data tools and MySql workbench(Just write SQL Script to Specific Changes instead of writing complete code)
3. Data Migration Tools:-
   Sometimes, you need to move data between databases Data migration tools make this easy and safe they can handle big data, different formats, and keep everything in synchronization.
   Example:- AWS Database Migration Services(DMS) and Azure Data Factory
4. Testing Strategies:-
   Lastly, you need to test the migrated database to make sure it works well. This includes various tests like checking performance and security. Testing ensures the databases meets all requirements and fixes any issues.
   Examples are SQL Test and JMeter.

6. Micro-services

Micro Services Architecture in Devops:-

If you consider Monolithic Architecture entire application is placed in one single folder. So changes made to one component will effect other components. But where as in micro services Architecture you are application is placed in bunch of folders each folder is responsible for a specific function. So changes made in one folder will not effect other folders.



Example:-

In a college website each function like student attendance, faculty Attendance, Student Results , college courses can be considered as separate services in a micro services architecture. Changes made to one service, such as updating the student attendance feature won't effect the other services like faculty attendance or student results.

Benefits:-

1. Scalability:-
   You can modify Specific part of our application independently without effecting other parts.
2. Flexibility:-
   Teams can choose the best tools and technologies for each micro service making it easier to adapt changing requirements.
3. Faster Deployment:-
   Since each micro service is independent you can deploy updates without effecting whole system.
4. Resilience:-

If one micro service fails, it doesn't bring down the entire application. The rest can still function normally.

Challenges:-

1. Complexity:-
   Managing multiple micro services can be more complex then monolithic architecture.
2. Communication:-
   Services need to communicate effectively, which requires good design and coordination.
3. Monitoring and Debugging:-
   With more micro services monitoring and debugging can be challenging.

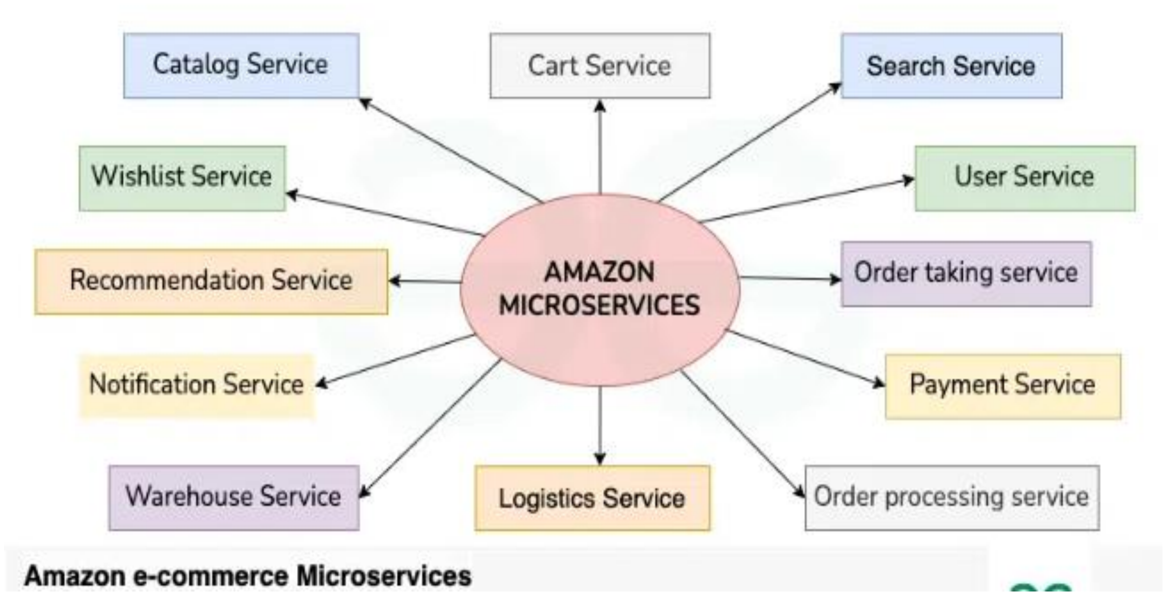
What are the main components of Microservices Architecture?
Main components of microservices architecture include:

- **Microservices:** Small, loosely coupled services that handle specific business functions, each focusing on a distinct capability.
- **API Gateway:** Acts as a central entry point for external clients also they manage requests, authentication and route the requests to the appropriate microservice.
- **Service Registry and Discovery:** Keeps track of the locations and addresses of all microservices, enabling them to locate and communicate with each other dynamically.
- **Load Balancer:** Distributes incoming traffic across multiple service instances and prevent any of the microservice from being overwhelmed.
- **Containerization:** Docker encapsulate microservices and their dependencies and orchestration tools like Kubernetes manage their deployment and scaling.
- **Event Bus/Message Broker:** Facilitates communication between microservices, allowing pub/sub asynchronous interaction of events between components/microservices.
- **Database per Microservice:** Each microservice usually has its own database, promoting data autonomy and allowing for independent management and scaling.
- **Caching:** Cache stores frequently accessed data close to the microservice which improved performance by reducing the repetitive queries.
- **Fault Tolerance and Resilience Components:** Components like circuit breakers and retry mechanisms ensure that the system can handle failures gracefully, maintaining overall functionality.

➔Real-World Example of Microservices

Let's understand the Miscroservices using the real-world example of Amazon E-Commerce Application:

Amazon's online store is like a giant puzzle made of many small, specialized pieces called microservices. Each microservice does a specific job to make sure everything runs smoothly. Together, these microservices work behind the scenes to give you a great shopping experience.



Below are the microservices involved in Amazon E-commerce Application:

- **User Service**: Handles user accounts and preferences, making sure each person has a personalized experience.
- **Search Service**: Helps users find products quickly by organizing and indexing product information.
- **Catalog Service**: Manages the product listings, ensuring all details are accurate and easy to access.
- **Cart Service**: Lets users add, remove, or change items in their shopping cart before checking out.
- **Wishlist Service**: Allows users to save items for later, helping them keep track of products they want.
- **Order Taking Service**: Processes customer orders, checking availability and validating details.
- **Order Processing Service**: Oversees the entire fulfillment process, working with inventory and shipping to get orders delivered.
- **Payment Service**: Manages secure transactions and keeps track of payment details.

- **Logistics Service**: Coordinates everything related to delivery, including shipping costs and tracking.
- **Warehouse Service**: Keeps an eye on inventory levels and helps with restocking when needed.
- **Notification Service**: Sends updates to users about their orders and any special offers.
- **Recommendation Service**: Suggests products to users based on their browsing and purchase history

7. And the data tier

The Data Tier:-

The Three-Tier Client-Server Architecture is a layered approach to building distributed systems, with each tier serving distinct roles. Below is a detailed explanation of each component:

1. Presentation Tier (Client Tier)

The user interface and user interaction are under the control of the Presentation Tier. It functions as the application’s front end, where users enter information and see the outcomes.

- **Components:**
    - **User Interface (UI):** Includes web browsers, mobile apps, or desktop applications that users interact with. It displays data and collects user inputs.
    - **User Interaction Logic:** Handles how user inputs are processed and communicated to the Application Tier. This can involve form validation, data formatting, and sending requests to the server.
- **Responsibilities:**
    - Display data from the Application Tier to the user.
    - Collect user inputs and forward them to the Application Tier.
    - Provide a user-friendly interface and manage user interactions.

2. Application Tier (Business Logic Tier)

The Application Tier is where the core business logic resides. It processes user requests, performs calculations, enforces business rules, and interacts with the Data Tier to retrieve or store data.

- **Components:**

    - **Business Logic:** Implements the rules and processes specific to the application, such as order processing, authentication, or data validation.
    - **Application Server:** Manages communication between the Presentation and Data Tiers and hosts the business logic. Web servers and application servers such as Apache Tomcat, Microsoft IIS, or JBoss are examples.
- **Responsibilities:**
    - Process and interpret data received from the Presentation Tier.
    - Execute business logic and apply rules.
    - Communicate with the Data Tier to retrieve or store information.
    - Send processed data back to the Presentation Tier for user display.

3. Data Tier (Database Tier)

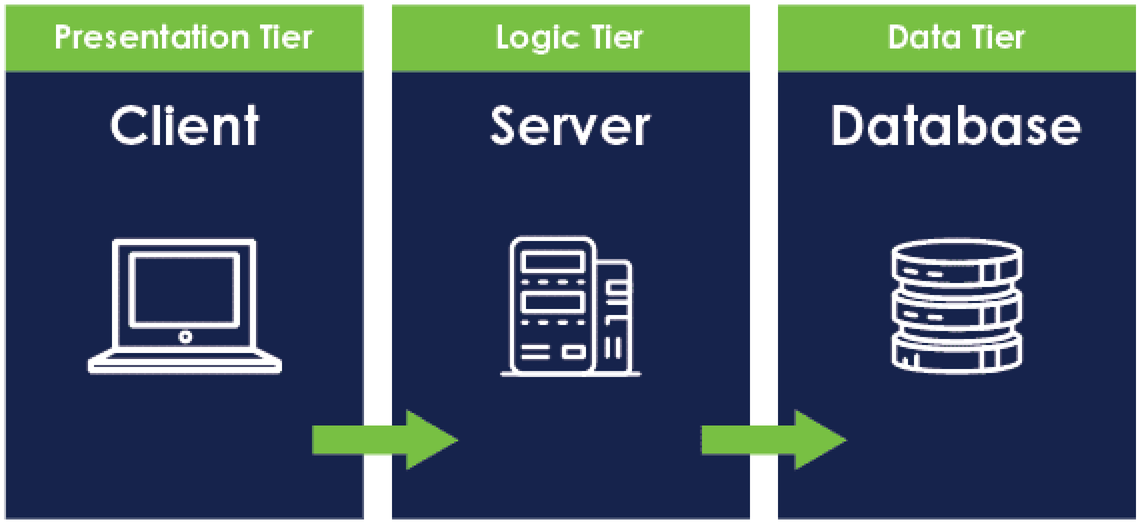
The Data Tier is responsible for data management and storage. It handles all database operations, including data retrieval, updates, and management.

- **Components:**
    - **Database Management System (DBMS):** Software like MySQL, Oracle, or Microsoft SQL Server that manages data storage and retrieval.
    - **Database:** The actual repository where data is stored, organized in tables or other structures.
- **Responsibilities:**
    - Store and manage data securely and efficiently.
    - Handle queries and transactions initiated by the Application Tier.
    - Ensure data integrity, consistency, and availability.
    - Provide backup and recovery mechanisms to protect data.

Interactions Among Tiers

- **Client Request:** The user interacts with the Presentation Tier, which sends a request to the Application Tier.
- **Business Processing:** The Application Tier processes the request using its business logic and may interact with the Data Tier to retrieve or update data.
- **Data Retrieval/Update:** The Data Tier handles data operations and sends the results back to the Application Tier.
- **Response to Client:** The Application Tier processes the results and sends the response back to the Presentation Tier for display to the user.

This tiered approach helps manage complexity by separating responsibilities, allowing for easier maintenance, scalability, and flexibility in distributed systems



The Power of Separation:
The beauty of the three-tier architecture lies in its **separation of concerns**. Each tier is independent, with its own set of responsibilities and technologies. This brings a wealth of benefits:

- **Flexibility and Scalability:** Imagine expanding the palace! Each tier can be scaled independently, allowing you to beef up the kitchen (application tier) to handle increased orders without affecting the ballroom (presentation tier).

- **Maintainability and Reusability:** Need to update the palace gardens (presentation tier)? No need to tear down the entire structure! Changes to one tier are isolated, making maintenance a breeze and code reusable across projects.

- **Teamwork and Expertise:** Different teams can focus on their areas of expertise, like frontend developers beautifying the gardens while backend engineers strengthen the foundations (application and data tiers).

Examples in Action:
To truly understand the three-tier architecture, let's take a trip to familiar territory:

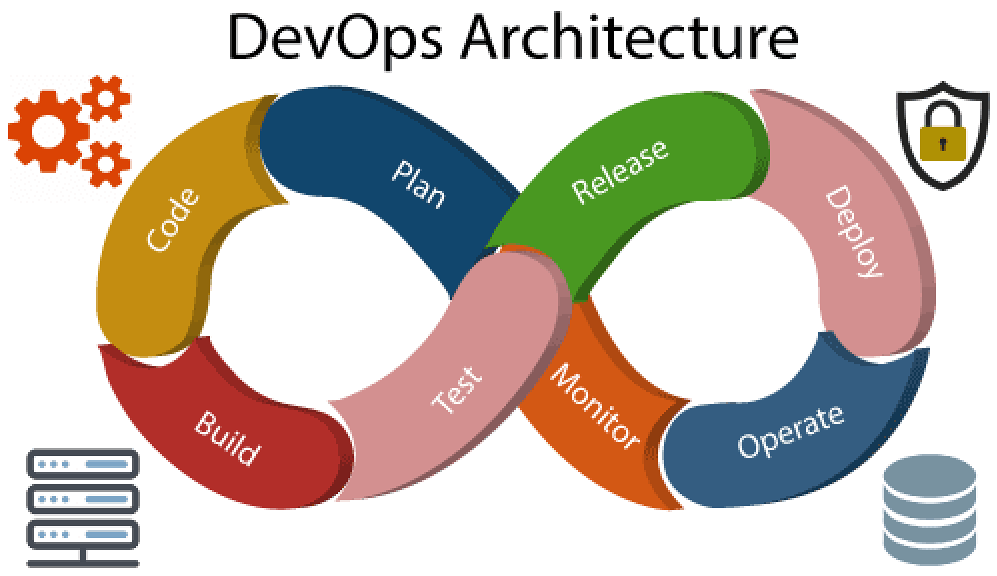
- **E-commerce website:** The product pages are the presentation tier, the shopping cart and checkout process are the application tier, and the product database is the data tier.

- **Online banking app:** The account overview and transaction history are the presentation tier, the payment processing and security checks are the application tier, and your financial data resides securely in the data tier.

A DevOps Champion:
As a DevOps champion, I see the three-tier architecture as a boon for continuous delivery and deployment. With independent tiers, updates and bug fixes can be rolled out to specific layers without disrupting the entire system. Imagine renovating the palace kitchen without disturbing the ongoing royal banquet in the grand hall!

8. DevOps Architecture

DevOps Architecture:-

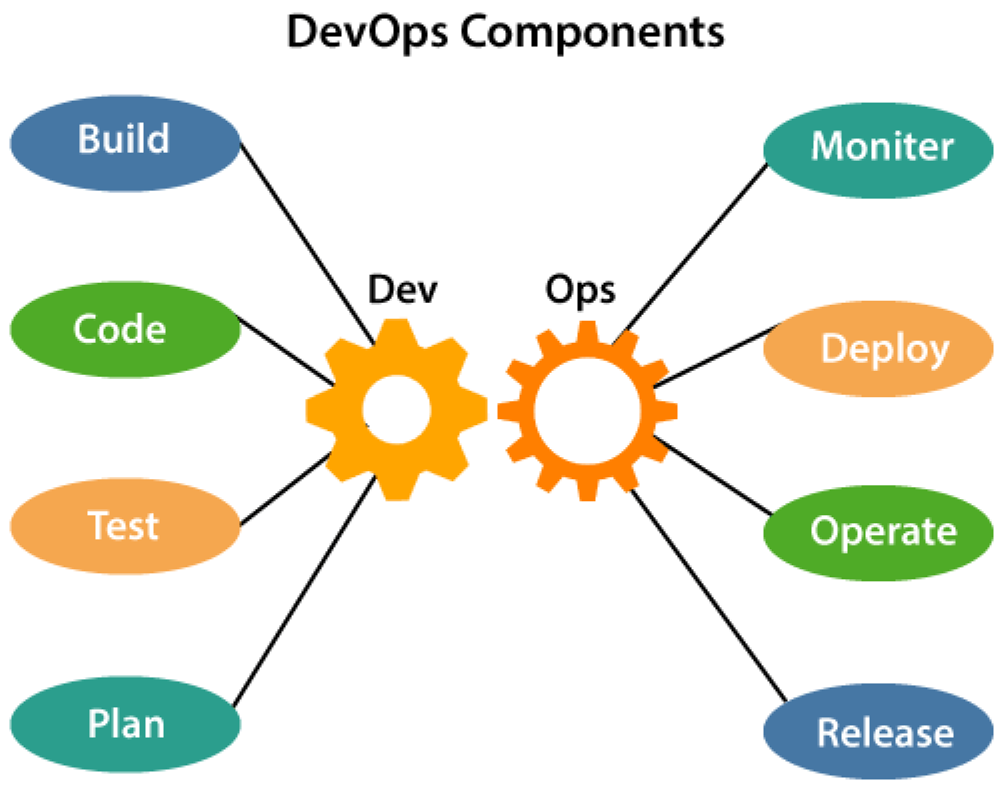


Development and operations both play essential roles in order to deliver applications. The deployment comprises analyzing the **requirements, designing, developing**, and **testing** of the software components or frameworks.

The operation consists of the administrative processes, services, and support for the software. When both the development and operations are combined with collaborating, then the DevOps architecture is the solution to fix the gap between deployment and operation terms; therefore, delivery can be faster.

DevOps architecture is used for the applications hosted on the cloud platform and large distributed applications. Agile Development is used in the DevOps architecture so that integration and delivery can be contiguous. When the development and operations team works separately from each other, then it is time-consuming to **design, test**, and **deploy**. And if the terms are not in sync with each other, then it may cause a delay in the delivery. So DevOps enables the teams to change their shortcomings and increases productivity.

Below are the various components that are used in the DevOps architecture:



1) Build

Without DevOps, the cost of the consumption of the resources was evaluated based on the pre-defined individual usage with fixed hardware allocation. And with DevOps, the usage of cloud, sharing of resources comes into the picture, and the build is dependent upon the user's need, which is a mechanism to control the usage of resources or capacity.

2) Code

Many good practices such as Git enables the code to be used, which ensures writing the code for business, helps to track changes, getting notified about the reason behind the difference in the actual and the expected output, and if necessary reverting to the original code developed. The code can be appropriately arranged in **files, folders**, etc. And they can be reused.

3) Test

The application will be ready for production after testing. In the case of manual testing, it consumes more time in testing and moving the code to the output. The testing can be automated, which decreases the time for testing so that the time to deploy the code to production can be reduced as automating the running of the scripts will remove many manual steps.

4) Plan

DevOps use Agile methodology to plan the development. With the operations and development team in sync, it helps in organizing the work to plan accordingly to increase productivity.

5) Monitor

Continuous monitoring is used to identify any risk of failure. Also, it helps in tracking the system accurately so that the health of the application can be checked. The monitoring becomes more comfortable with services where the log data may get monitored through many third-party tools such as **Splunk**.

6) Deploy

Many systems can support the scheduler for automated deployment. The cloud management platform enables users to capture accurate insights and view the optimization scenario, analytics on trends by the deployment of dashboards.

7) Operate

DevOps changes the way traditional approach of developing and testing separately. The teams operate in a collaborative way where both the teams actively participate throughout the service lifecycle. The operation team interacts with developers, and they come up with a monitoring plan which serves the IT and business requirements.

8) Release

Deployment to an environment can be done by automation. But when the deployment is made to the production environment, it is done by manual triggering. Many processes involved in release management commonly used to do the deployment in the production environment manually to lessen the impact on the customers.

9. Resilience

Resilience in Devops:-

Resilience in Devops means the ability of Systems to recover quickly from problems and continue working smoothly. It ensures that services remain available even when there are issues like failures or high traffic.

How Organizations Achieve Resilience Using Devops:-

1. Automation:-
   Automate tasks like deployment and testing. Reduces human errors and speeds up recovery
2. Continuous Integration/Continuous Deployment(CI/CD):-
   Regularly integrate and deploy small changes. Makes it easier to find and fix issues quickly.
3. Monitoring and Logging:-
   Logging(It Means Recording detailed information about the operations of software)
   Constantly check systems for problems. Use logs to understand issues and respond faster.
4. Infrastructure as code(Ias):-
   Infrastructure(It means hardware, software, networks, servers, database and other components)
   Manage Infrastructure with code. Easily rebuild or update systems using scripts.
5. Micro Services Architecture:-
   Break applications into smaller, independent services. If one service fails, others continue to work.

6. Regular Backups and Disaster Recovery Plans:-
   Keep backups of data and systems. Have Plans to recover from major failures.
7. Scalability:-
   Design Systems to handle more load automatically.