**3-tier architecture**

**Difference between monolithic and micro-services**

**what is monolithic?**

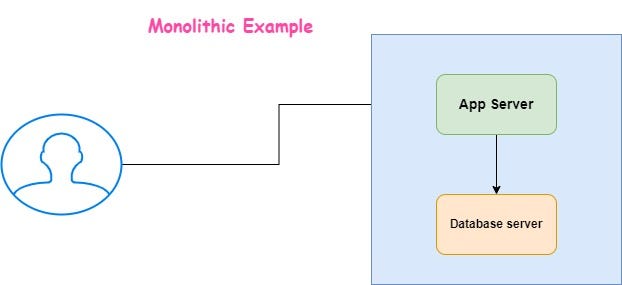
In earlier systems, everything was hosted on a single server. The application was large, encompassing both front-end and back-end components.

* **Deployment Challenges:** Even a simple change required redeploying the entire application, leading to frequent and potentially disruptive deployments.
* **Maintenance Issues:** Maintaining such a monolithic application could be a significant challenge due to its complexity and size.

The development team would typically include:

* **Front-End Developers:** Specializing in technologies such as HTML, CSS, JavaScript, and frameworks like ReactJS.
* **Back-End Developers:** Working with server-side languages and frameworks such as Java, .NET, Node.js, Python, and PHP.

In this setup, front-end technologies handle the user interface and user experience, while back-end technologies manage the server-side logic and database interactions.



**let us take one example**

* If front-end and back-end developers work together to integrate their application into a single server for deployment, they will release the application as a unified system.
* However, if the client requests changes after the application is live, developers must modify the application and redeploy it. This process could result in significant downtime, potentially lasting a whole day. During this downtime, business operations might be severely impacted, leading to a loss of revenue and customer dissatisfaction.

**Disadvantages of MONOLITHIC**

Consider a scenario with 1,000 Java developers working on a monolithic application.

* **Single Language Dependence:** The entire application is built using Java. Migrating to a different programming language, like Python, would be challenging because the entire system is tightly coupled to Java.
* **Single Point of Failure:** An error in any component can cause the entire application to go down, impacting the whole system.
* **Maintenance and Monitoring:** Maintaining and monitoring a monolithic application can be complex due to its tightly integrated nature.
* **Scaling Challenges:** During peak traffic times, such as during festival seasons or special offers (e.g., Amazon’s high-traffic periods), scaling the application to handle increased loads can be difficult.

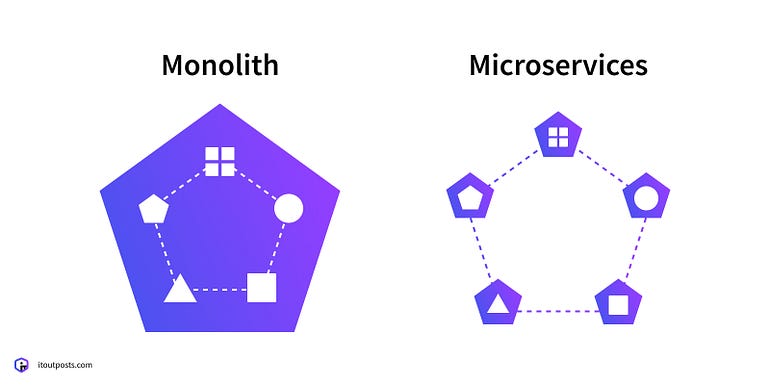
To address these issues, the concept of micro-services was introduced. Micro-services break down the application into smaller, independent services, each of which can be developed, deployed, and scaled separately, making it easier to manage and adapt to changing needs.

**what is micro-services?**

Micro-services are API-based applications that promote a decoupled architecture.

* **Separation of Concerns:** In a micro-services architecture, the dependencies between the front-end and back-end are minimized. This means that the front-end and back-end teams can work independently of each other.
* **Front-End Team:** Focuses solely on the web tier, handling the user interface and user experience. Changes made by this team only affect the web tier.
* **Back-End Team:** Works on the application tier, managing server-side logic and services. Changes in this area are confined to the application tier.
* **Database Team:** Handles the database tier, dealing with data storage and management. Any changes here are isolated to the database tier.

Today, many organizations use micro services combined with a 3-tier architecture to enhance modularity, scalability, and maintainability of their applications.



**3-tier architecture**

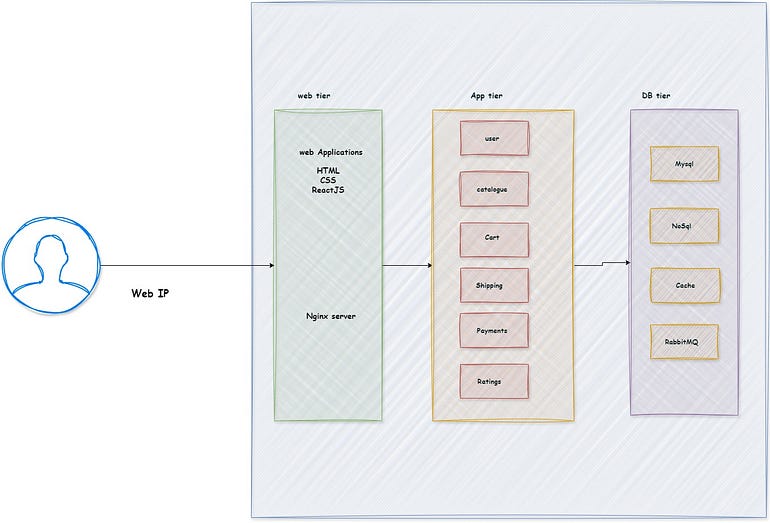
1. Web-tier
2. APP-tier
3. Database-tier

* In earlier days, there is only desktop applications,now a days everybody is using web-based Applications.
* Because in web based Applications installations are done automatically, we no need to install and we no need to do upgrades.
* It will not consume memory and CPU(storage)
* When working on a desktop, if your laptop crashes, you might lose all your data. However, if you store your data on the web, it remains safe and secure.
* web can be accessible from anywhere and it acts as a version control as well

**Database-tier**The DB-tier stores any kind of databases in Raw format like MySQL, cache, NoSQL etc

**APP-tier**  
It takes Raw data from DB tier and it makes some format changes to understand easily by using back-end programming languages like Java, .net, Python etc., and it is not exposed to user.

**web-tier**  
It takes data from App-tier and exposed to user by adding some more modifications which can be easily understood by user and also it acts as a front-end,by using HTML, CSS, JavaScript etc.,



***let us take one practical example for 3-tier architecture(micro-services)***  
  
**Example** : ***Restaurant***



**1. Raw Vegetables — Database Tier**

* **Raw Vegetables:** Imagine raw vegetables as the unprocessed data in a database. Just like raw vegetables are fundamental ingredients that need preparation, raw data stored in databases like MySQL, NoSQL, or cache systems is not directly usable by end-users. It’s in a raw, un-formatted state and needs to be transformed to be useful.

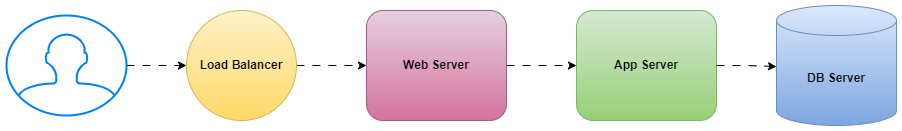
**2. Chef — Application Tier**

* **Chef:** The chef represents the application tier, where raw vegetables (data) are turned into a delicious dish (processed information). Just as a chef uses skills and recipes to prepare food, the application tier uses programming languages like Java or Python to process and transform raw data into meaningful and structured information. This layer handles the logic and processing needed to make data accessible and usable.

**3. Waiter — Web Tier**

* **Waiter:** The waiter symbolizes the web tier. After the chef has prepared the dish, the waiter adds the final touches, such as garnishing and presentation, and then serves it to the customer. Similarly, the web tier uses technologies like HTML, CSS, and JavaScript to present the processed data in a user-friendly format. It ensures that the information is well-organized, visually appealing, and accessible to users.

**4. Load Balancer (Captain):** The load balancer ensures high availability and performance, scaling resources as needed and managing user requests efficiently. This helps maintain a seamless experience for users, even during peak traffic times.



**Conclusion:**

Switching from a monolithic to a microservices architecture with a 3-tier design changes how applications are managed. It turns large, unwieldy systems into smaller, more flexible parts that can be scaled easily. Just like a well-organized restaurant, this setup allows each part of the application to work on its own and fit together smoothly, making it easier to adjust and perform well in a fast-changing world.