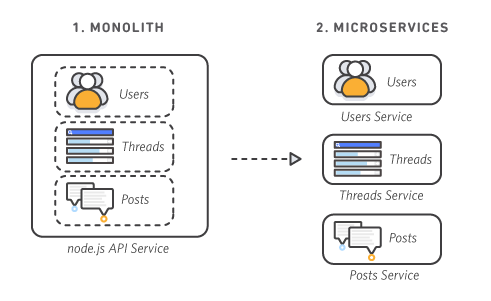
**MONOLITHIC SERVICES :** A monolithic service refers to a software architecture where an application is built as a single, self-contained unit, with all its components tightly coupled and deployed as a single, indivisible entity. In essence, it's a large, single codebase where the user interface, business logic, and data access layers are all part of the same application.

* **Single codebase:**  All components of the application reside in one codebase.
* **Unified deployment:** The entire application is deployed as a single unit, and any updates or changes require redeploying the whole monolith.
* **Tight coupling:** Components within the application are highly interconnected and dependent on each other, meaning changes in one part can impact other areas of the application.
* **Shared memory:** Components typically operate within the same memory space, enabling direct communication between them.
* **Centralized database:** A single database often stores all the application's data.



**MICROSERVICES :** A microservices is an architectural style for building applications as a collection of small, independent services that communicate with each other. These services are loosely coupled and can be developed, deployed, and scaled independently. Think of it as breaking down a large application into smaller, manageable pieces, each with its own specific function.

* **Independent deployment:** Each microservice can be developed, deployed, and updated independently without affecting the functioning of other services. This allows for rapid iteration and deployment of new features or bug fixes.
* **Loosely coupled:** Services are loosely coupled, meaning they are not heavily dependent on each other's internal details. Communication between services happens through well-defined interfaces (APIs), ensuring that changes to one service's design or implementation don't require changes in others.
* **Specialized:** Each microservice is designed for a specific set of capabilities, focusing on solving a particular business problem. If a service becomes too complex, it can be broken down into smaller, more manageable microservices.
* **Decentralized:** Microservices promote decentralized governance, allowing development teams to choose the most suitable technologies and processes for their services. This autonomy boosts productivity and delivery speed.
* **High fault tolerance:** Service independence increases an application's resistance to failure. If one microservice fails, it doesn't necessarily impact the entire application, as the system can degrade gracefully or reroute requests to working services.

