**Monolithic Architecture**

Monolithic architecture is the approach to building a self-contained application also called as “single-tier” application. In a monolithic the application is made up of three main components that together create one codebase:

* **The user interface (UI)**

The front-end / presentation layer / client-side

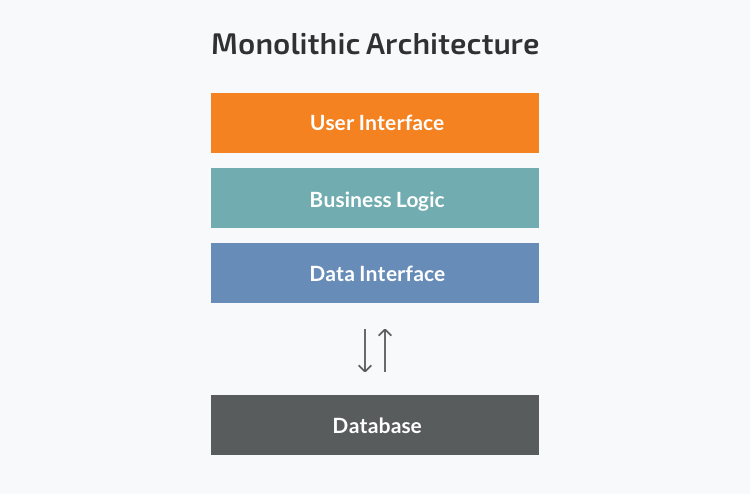
* **The business logic**

The application and server-side programming (part of the back-end)

* **The data interface**

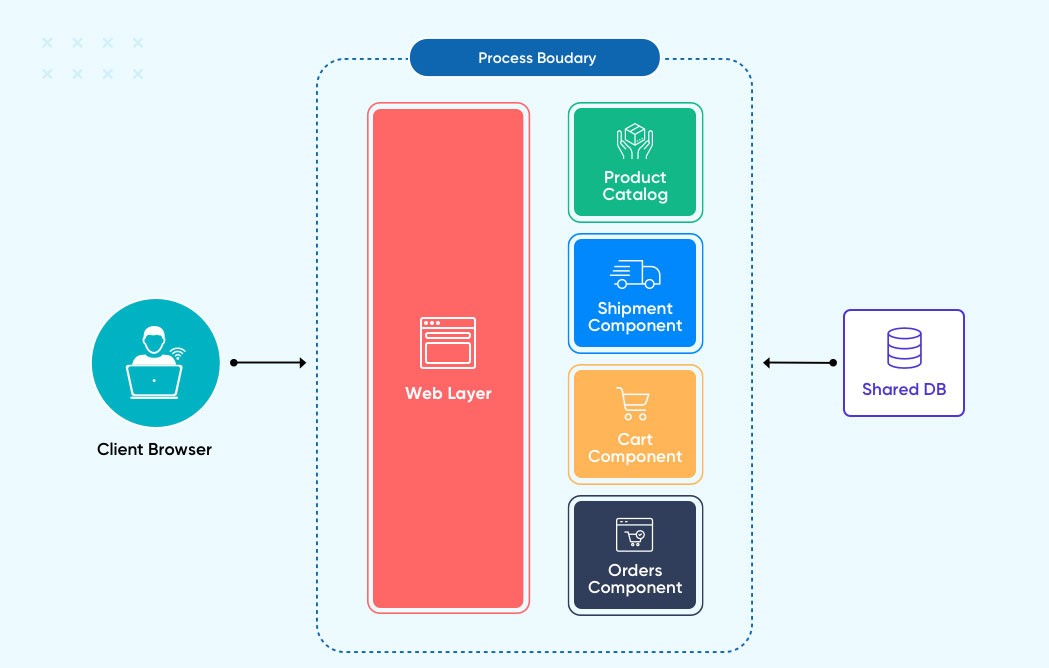
The data access code that communicates the request to the database

These three parts are managed and served from **one large code base** that is then connected to the **database.** (As shown in below image)



Let’s take an example

**Ecommerce Application in monolithic architecture**

****

In this example, the client browser accesses the application, which displays UI to interact with the product catalog and completes the order with in-process communication with one or more databases shared by the entire monolithic application.

**Advantages of the Monolithic Architecture**

A monolithic architecture is an “all in one” solution: a single code base deployed as a single unit. In general, this offers the following benefits:

* Low cost to develop for simple apps
* The architecture is easy to design, test and deploy
* Supports vertical scaling to share the workload and improve performance
* Eliminates cross-cutting concerns that exist when an application is spread across different modules (e.g. logging, performance monitoring)

### Disadvantages of the Monolithic Architecture

* Updates to any part of the code base impact the entire code base, requiring full recompile of the app
* Any errors or server issues impact the entire application, impacting its overall reliability
* Code reuse is limited, often only supported with shared libraries (leading to coupling issues)
* Changes to any part of the app code become expensive due to dependencies in the code
* Code base can become large over time, marking it difficult to maintain and for new developers to jump in to contribute
* Does not support horizontal scaling (vertical scaling requires that the whole application be loaded onto multiple servers), so everything scales when only a part of the application may be experiencing a large load
* Tied to a single technology stack for everything

Monolithic apps are simple to develop, but over time tend to become large, difficult to manage, and complex to update.

## **Microservices Architecture**

Microservices architecture decouples the front and back-ends of architecture, linking various independent services in the back-end (Microservices) to the front-end via API. The Microservices approach supports flexibility to choose and scale services as needed.

**Best Practices for Microservices Architecture**

* **Single Responsibility**

The single responsibility principle (SRP) in programming states that every module should have responsibility over only one part of the app, narrowly defined and encapsulated.

* **Built Around Business Capabilities**

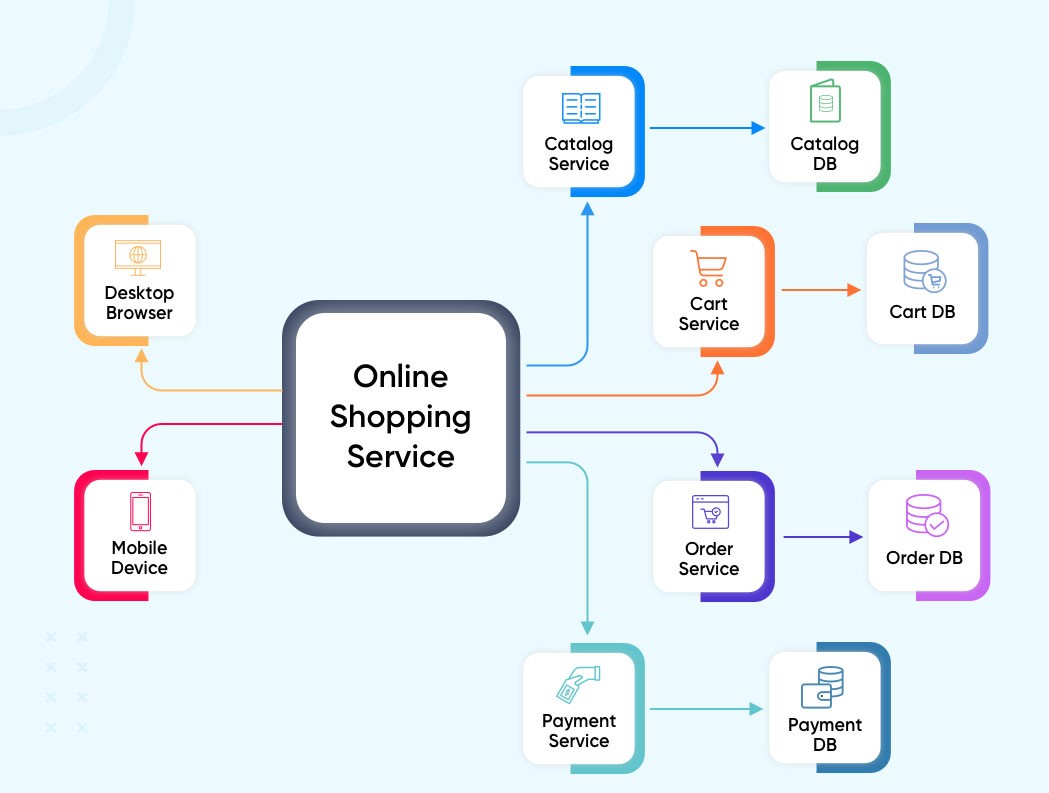
Business logic is the core of microservices, with each service built around a *business capability* rather than a *technical capability*. Each business problem can be solved in a microservice that leverages its own tech stack.

* **Design for Failure**

To design for failure means to build resiliency into the core of the app. By defining boundaries around independent and encapsulated microservices, issues in one service should not affect other services. However, downtime is inevitable in every system, so designing for failure introduces how to degrade service if one or more components fails. For example, when Instagram is down (not fetching new content), it still supports browsing of cached content.

Let’s take an example

**Ecommerce application in Microservices architecture**



In this example, the user accesses the user interface, which then connects to each microservice independently when, and only when, they are needed. If the website suddenly sees a lot of traffic, the catalog service can be scaled up, with other microservices being scaled as much or as little as needed.

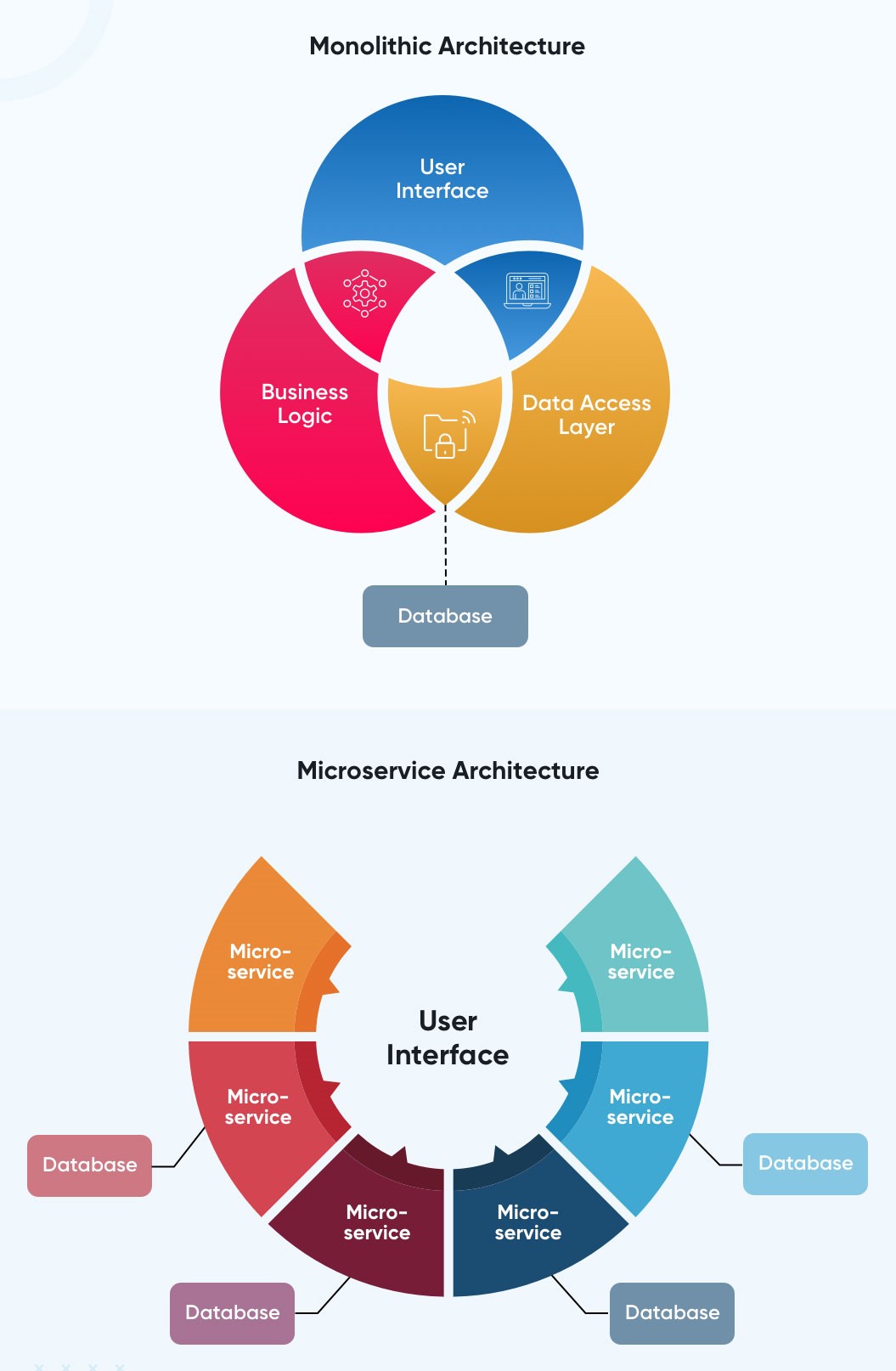
**Advantages of the Microservices Architecture**

* Components are only loosely coupled by API, with fewer interdependencies, making it easier to update or replace a module
* Microservices can be reused by other modules
* Module code base makes it easier for developers to learn each module, to find bugs, or to make changes
* Scaling is faster and cheaper since each microservice can scale independently and it’s easier to spot bottlenecks
* It’s designed for failure – one module going down will not impact the entire application
* Each microservice can use the technology stack best for its own operations

**Disadvantages of the Microservices Architecture**

* Microservices require more oversight on monitoring and logging each microservice
* Require more oversight to address security
* Moving code across services can be difficult if technologies differ

**Microservices vs Monolithic**

****

**Deployment Strategy**

While it is more obvious that monolithic architecture (one code base) is deployed in a traditional format to standard web servers, the same cannot be said for microservices architecture. There are, in fact, a variety of approaches that can support microservices deployment including:

* **One service-One host**  
  Each service is deployed to just one virtual machine (host). This is the lowest cost and most straightforward option.
* **One Service-One Container**  
  Docker containers help to isolate microservices (one of the goals of good strategy) but to allow those containers to share resources such as operating servers, libraries, or frameworks.
* **Serverless Deployment**  
  Serverless deployments abstract and outsource infrastructure. While the program being created will run on a server, the servers are hosted and managed completely by a third party (cloud) who worries about all the patching, scaling, and load tasks.

**Architecture Comparison**

|  |  |  |
| --- | --- | --- |
|  | **Monolithic Microservice** | |
| **Deployment** | Simple and fast deployment of the entire system | Requires distinct resource, making orchestrating the deployment complicated |
| **Scalability** | It is hard to maintain and handle new changes; the whole system needs to be redeployed | Each element can be scaled independently without downtime |
| **Agility** | Not flexible and impossible to adopt new tech, languages, or frameworks | Integrate with new technologies to solve business purposes |
| **Resiliency** | One bug or issue can affect the whole system | A failure is one microservice does not affect other services |
| **Testing** | End-to-end testing | Independent components need to be tested individually |
| **Security** | Communication within a single unit makes data processing secure | Inter-process communication requires API gateways raising security issues |
| **Development** | Impossible to disturb the team’s effort due to the huge indivisible database | A team of developers can work independently on each component |

### Choosing a Monolithic Architecture

Choose monolithic architecture if the organization or project involves:

* **Small Team**  
  Monolithic architecture is ideally suited to the small business or startup. With a lean IT team, you can have deep experience in one technology stack to use for the entire application and not worry about skill shortages for different IT stacks, knowledge silos, or the ongoing management of a more complex microservice architecture.
* **Simple Application**  
  There is no need to reinvent the wheel for a simple application that is very straightforward and not likely to scale at an unprecedented pace.
* **No Microservices Expertise**  
  Effective microservices requires multiple people with expertise in specific services, technologies or frameworks as well as those who have experience in how to bring it all together to work well.
* **Quick Launch**  
  A simple application can be developed to prototype rapidly with a monolithic approach.

### Choosing a Microservices Architecture

On the flip side, monolithic architecture is not ideally suited where organizations need to remain nimble against competition and where resources are at play to develop a large or complex app.

* **A Complex and Scalable Application**  
  Microservices will help the organization develop more complex software or app that involve a lot of business logic (operated by many different modules), either offering personalization or lots of features or heavy use of interactivity. Microservices is also ideally suited for those “breakthrough” startup apps or apps that reach a large audience and need to be able to scale rapidly.
* **Microservices Expertise**  
  If the team has access to, or can hire out, the right skills and knowledge, it’s possible to aim for a microservices approach to build and maintain an app built on microservices architecture. IT teams need experience in microservices, DevOps, containers and Cloud.
* **Enough Engineering Skills**  
  As above, microservices teams require enough in-house or outsourced skills in enough areas to develop each module / tech stack or to work with SaaS options for different modules.
* **Excellent Cloud-based Infrastructure**  
  Microservices relies on Cloud Infrastructure to be effective and to scale appropriately (at the right speed, cost) as necessary. Make sure the cloud infrastructure chosen is flexible on pay and on technology to provide the greatest flexibility in architecture.