Mohamed El Laithy

Content Creator

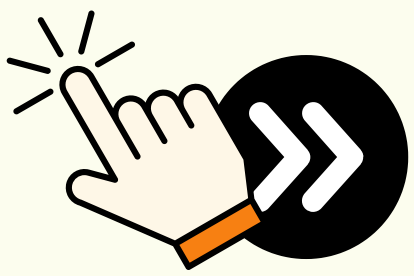
Microservice Architecture Complete Guide

Swipe for more

Mohamed El Laithy Content Creator

Table of Content

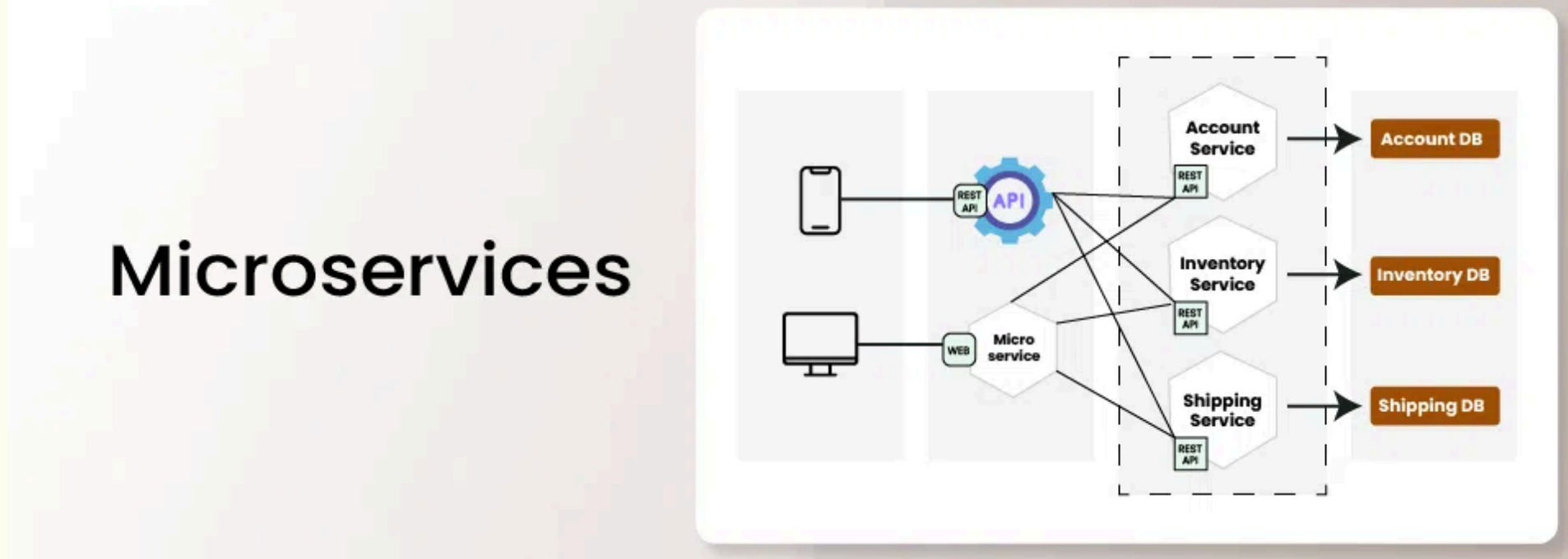
Swipe for more



Mohamed El Laithy Content Creator

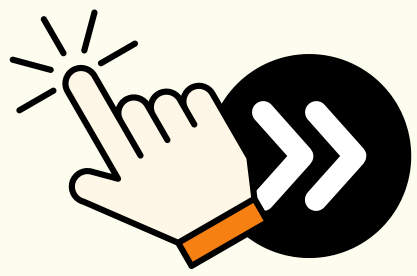
1- What are Microservices?

Microservices are an architectural approach to developing software applications as a collection of small, independent services that communicate with each other over a network. Instead of building a monolithic application where all the functionality is tightly integrated into a single codebase, microservices break down the application into smaller, loosely coupled services.



- This architecture allow you to take a large monolith application and decompose it into small manageable components/services. Also, it is considered as the building block of modern applications.
- Microservices can be written in a variety of programming languages, and frameworks, and each service acts as a mini-application on its own.

Swipe for more



2- How do Microservices work?

Microservices break complex applications into smaller, independent services that work together, enhancing scalability, and maintenance. Below is how microservice work:

- Applications are divided into self-contained services, each focused on a specific function, simplifying development and maintenance.
- Each microservice handles a particular business feature, like user authentication or product management, allowing for specialized development.
- Services interact via APIs, facilitating standardized information exchange and integration.
- Different technologies can be used for each service, enabling teams to select the best tools for their needs.
- Microservices can be updated independently, reducing risks during changes and enhancing system resilience.

3- 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.

Swipe for more

- 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.

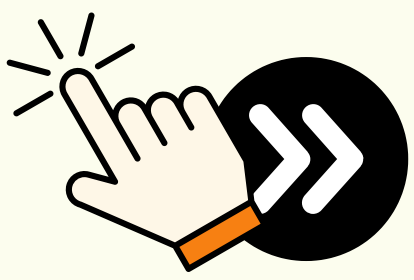
- Event Bus/Message Broker: Facilitates communication between microservices, allowing pub/sub asynchronous interaction of events between components/microservices.

- 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.

Swipe for more



Mohamed El Laithy Content Creator

4- Design Patterns for Microservices Architecture

Below are the main design pattern of microservices:

1. API Gateway Pattern

Think of the API gateway as the front door to your microservices. It acts as a single point of entry for clients, managing requests and directing them to the appropriate service.

This pattern simplifies the client's experience by hiding the complexities of multiple services behind one interface. It can also handle tasks like authentication, logging, and rate limiting, making it a crucial part of microservices architecture.

2. Service Registry Pattern

This pattern is like a phone book for microservices. It maintains a list of all active services and their locations (network addresses). When a service starts, it registers itself with the registry.

Other services can then look up the registry to find and communicate with it. This dynamic discovery enables flexibility and helps services interact without hardcoding their locations.

Swipe for more

3. Circuit Breaker Pattern

Imagine a circuit breaker in your home that stops electricity flow when there's an overload. Similarly, this pattern helps protect your system from cascading failures.

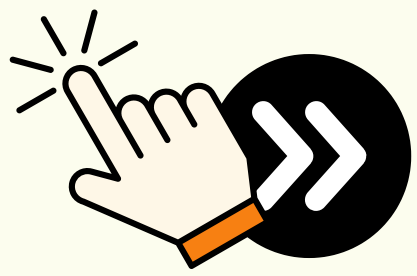
If a service fails repeatedly, the circuit breaker trips, preventing further requests to that service. After a timeout period, it allows limited requests to test if the service is back online. This reduces the load on failing services and enhances system resilience.

4. Saga Pattern

This pattern is useful for managing complex business processes that span multiple services. Instead of treating the process as a single transaction, the saga breaks it down into smaller steps, each handled by different services.

If one step fails, compensating actions are taken to reverse the previous steps. This way, you maintain data consistency across the system, even in the face of failures.

Swipe for more



5. Event Sourcing Pattern

Instead of storing just the current state of an application, this pattern records all changes as a sequence of events.

Each event describes a change that occurred, allowing services to reconstruct the current state by replaying the event history. This provides a clear audit trail and simplifies data recovery in case of errors.

6. Strangler Fig Pattern

This pattern allows for a gradual transition from a monolithic application to microservices. New features are developed as microservices while the old system remains in use.

Over time, as more functionality is moved to microservices, the old system is gradually "strangled" until it can be fully retired. This approach minimizes risk and allows for a smoother migration.

Swipe for more

7. Bulkhead Pattern

Similar to compartments in a ship, the bulkhead pattern isolates different services to prevent failures from affecting the entire system.

If one service encounters an issue, it won't compromise others. By creating boundaries, this pattern enhances the resilience of the system, ensuring that a failure in one area doesn't lead to a total system breakdown.

8. API Composition Pattern

When you need to gather data from multiple microservices, the API composition pattern helps you do so efficiently.

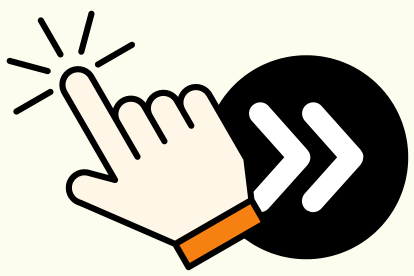
A separate service (the composition service) collects responses from various services and combines them into a single response for the client. This reduces the need for clients to make multiple requests and simplifies their interaction with the system.

9. CQRS Design Pattern

The CQRS (Command Query Responsibility Segregation) pattern splits data handling into two distinct operations: commands for modifying data and queries for retrieving it. This separation lets each side be optimized for its specific task.

For example, commands can enforce complex business logic, while queries can be tuned for speed and efficiency. This approach is especially valuable in systems with heavy read and write loads, as it boosts performance and scalability by allowing tailored solutions for each operation type.

Swipe for more



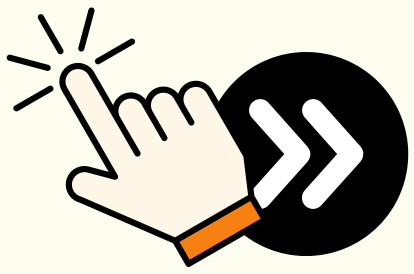
Mohamed El Laithy Content Creator

5- Anti-Patterns for Microservices Architeture

Learning anti-patterns in microservices is crucial for avoiding common mistakes. Below are some anti-patterns in microservices and by understanding these anti-patterns, developers can make informed decisions and implement best practices.

- When microservices share a single centralized database, it can compromise their independence and scalability.
- Microservices that frequently communicate for minor tasks can create excessive network traffic, leading to delays and increased latency.
- Creating too many microservices for small functions can add unnecessary complexity to the system.
- If the boundaries between microservices are not clearly defined, it can cause confusion about their responsibilities.
- Failing to address security issues in microservices can expose the system to vulnerabilities and potential data breaches.

Swipe for more

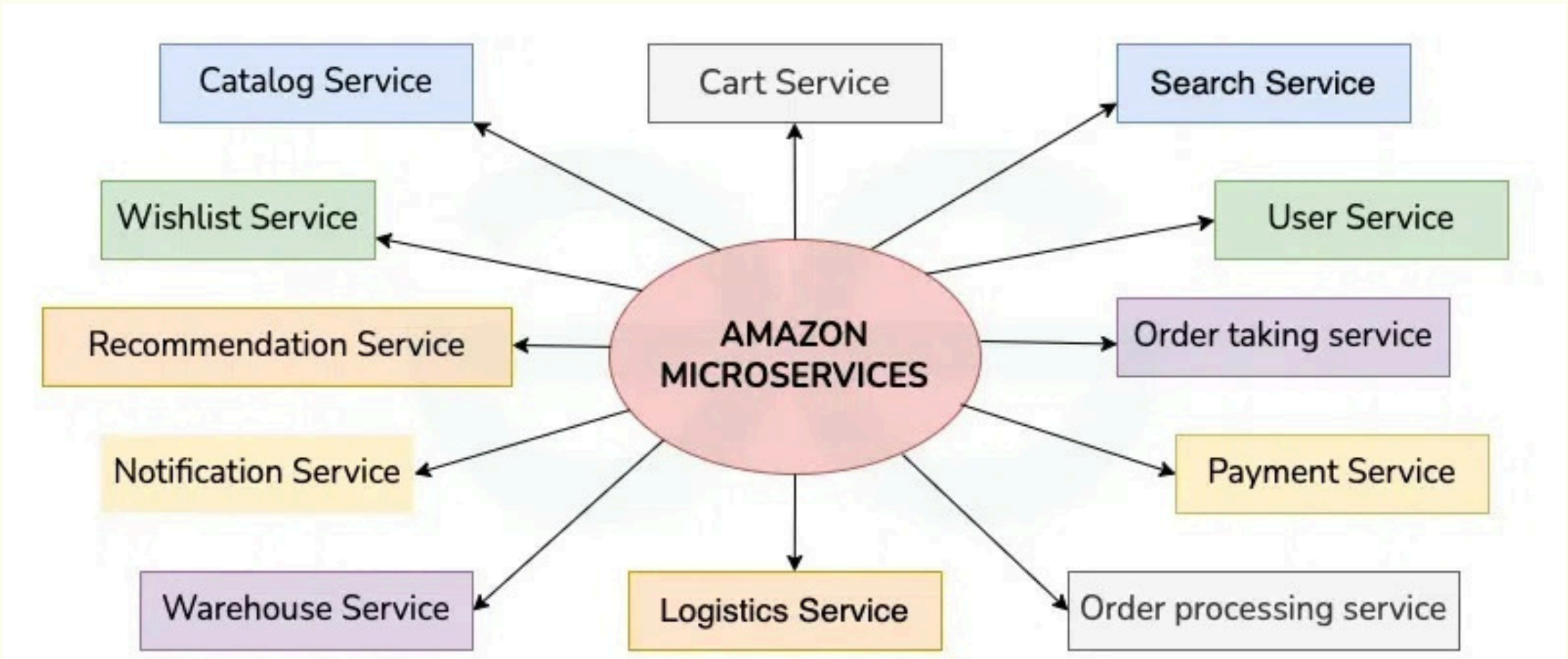


Mohamed El Laithy Content Creator

6- Real-World Example of Microservices

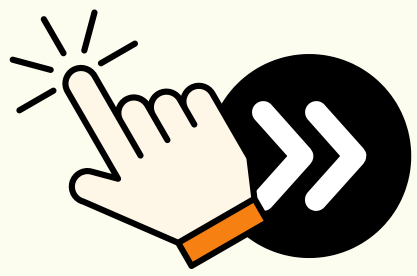
Let's understand the Miscroservices using the real-world example of Amazon E-Commerce Application:

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.



Amazon e-commerce Microservices

Swipe for more



Mohamed El Laithy Content Creator

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

Swipe for more

Mohamed El Laithy Content Creator

7- Microservices vs. Monolithic

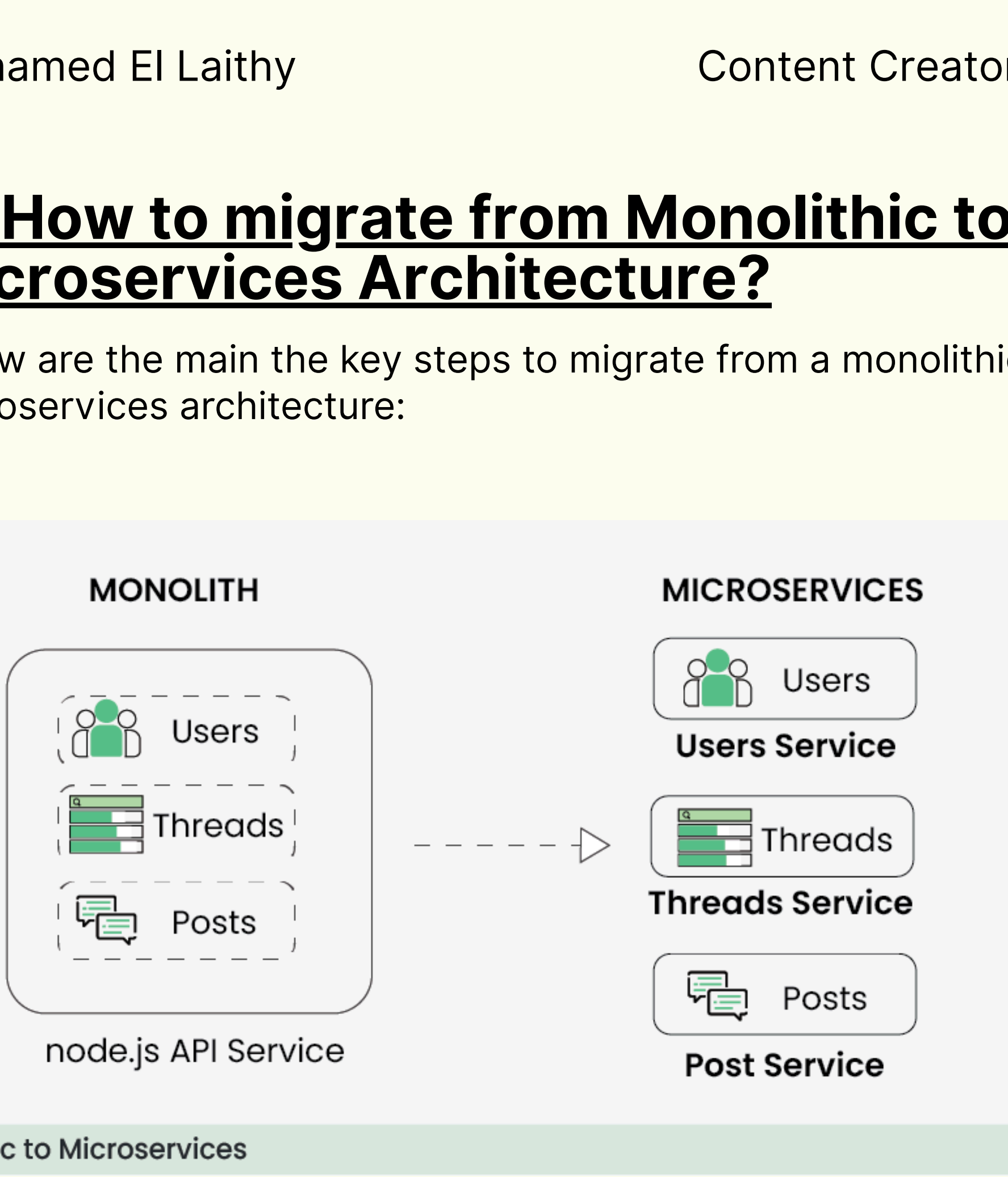
Below is a tabular comparison between microservices and monolithic architecture across various aspects:

Aspect	Microservices Architecture	Monolithic Architecture
Architecture Style	Decomposed into small, independent services.	Single, tightly integrated codebase.
Development Team Structure	Small, cross-functional teams for each microservice.	Larger, centralized development team.
Scalability	Independent scaling of individual services.	Scaling involves replicating the entire application.
Deployment	Independent deployment of services.	Whole application is deployed as a single unit.
Resource Utilization	Efficient use of resources as services can scale independently.	Resources allocated based on the overall application's needs.
Development Speed	Faster development and deployment cycles.	Slower development and deployment due to the entire codebase.
Flexibility	Easier to adopt new technologies for specific services.	Limited flexibility due to a common technology stack.
Maintenance	Easier maintenance of smaller, focused codebases.	Maintenance can be complex for a large, monolithic codebase.

Swipe for more

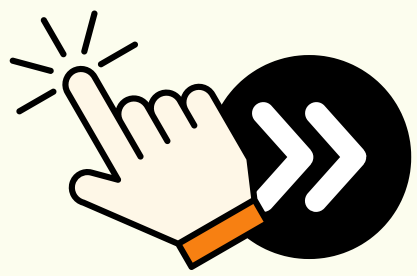
8- How to migrate from Monolithic to Microservices Architecture?

Below are the main the key steps to migrate from a monolithic to microservices architecture:



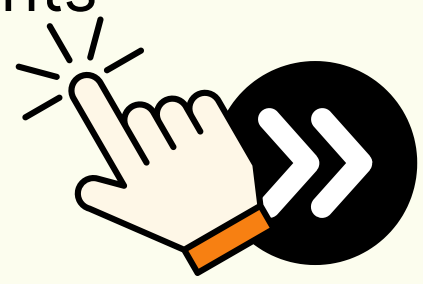
Monolithic to Microservices

Swipe for more



- Step 1: Begin by evaluating your current monolithic application. Identify its components and determine which parts can be transitioned into microservices.
- Step 2: Break down the monolith into specific business functions. Each microservice should represent a distinct capability that aligns with your business needs.
- Step 3: Implement the Strangler Pattern to gradually replace parts of the monolithic application with microservices. This method allows for a smooth migration without a complete overhaul at once.
- Step 4: Establish clear APIs and contracts for your microservices. This ensures they can communicate effectively and interact seamlessly.
- Step 5: Create Continuous Integration and Continuous Deployment (CI/CD) pipelines. This automates testing and deployment, enabling faster and more reliable releases.
- Step 6: Introduce mechanisms for service discovery so that microservices can dynamically locate and communicate with each other, enhancing flexibility.
- Step 7: Set up centralized logging and monitoring tools. This provides insights into the performance of your microservices, helping to identify and resolve issues quickly.
- Step 8: Ensure consistent management of cross-cutting concerns, such as security and authentication, across all microservices to maintain system integrity.
- Step 9: Take an iterative approach to your microservices architecture. Continuously refine and expand your services based on feedback and changing requirements

Swipe for more



Mohamed El Laithy Content Creator

9- Service-Oriented Architecture(SOA) vs. Microservices Architecture

Below is a tabular comparison between Service-Oriented Architecture (SOA) and Microservices across various aspects:

Aspect	Service-Oriented Architecture(SOA)	Microservices Architecture
Scope	Includes a broad set of architectural principles.	Focuses on building small, independent services.
Size of Services	Services tend to be larger and more comprehensive.	Services are small, focused, and single-purpose.
Data Management	Common data model and shared databases are common.	Each service has its own database or data store.
Communication	Typically relies on standardized protocols like SOAP.	Uses lightweight protocols such as REST or messaging.
Technology Diversity	Can have different technologies, but often standardized middleware.	Encourages diverse technologies for each service.
Deployment	Services are often deployed independently.	Promotes independent deployment of microservices.
Scalability	Horizontal scaling of entire services is common.	Enables independent scaling of individual services.
Development Speed	Slower development cycles due to larger services.	Faster development cycles with smaller services.
Flexibility	Can be flexible, but changes may affect multiple services.	Provides flexibility due to independent services.

Swipe for more

10- Benefits and Challenges of using Microservices Architecture

Benefits of Using Microservices Architecture:

- Teams can work on different microservices simultaneously.
- Issues in one service do not impact others, enhancing reliability.
- Each service can be scaled based on its specific needs.
- The system can quickly adapt to changing workloads.
- Teams can choose the best tech stack for each microservice.
- Small, cross-functional teams work independently.

Challenges of using Microservices Architecture

- Managing service communication, network latency, and data consistency can be difficult.
- Decomposing an app into microservices adds complexity in development, testing, and deployment.
- Network communication can lead to higher latency and complicates error handling.
- Maintaining consistent data across services is challenging, and distributed transactions can be complex.

Swipe for more

Mohamed El Laithy Content Creator

11- Real-World Examples of Companies using Microservices Architecture

Organizations have undergone significant changes by adopting microservices, moving from monolithic applications. Here are some real-life examples:

- Amazon: Initially a monolithic app, Amazon uses microservices early on, breaking its platform into smaller components. This shift allowed for individual feature updates, greatly enhancing functionality.
- Netflix: After facing service outages while transitioning to a movie-streaming service in 2007, Netflix adopted a microservices architecture. This change improved reliability and performance.
- Uber: By switching from a monolithic structure to microservices, Uber operations were become smoother, resulting in increased webpage views and search efficiency

Swipe for more

Mohamed El Laithy Content Creator

12- Roadmap to understand Microservices

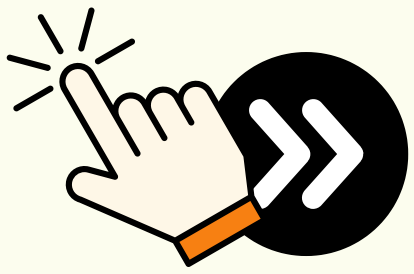
1. Introduction to Microservices
   - What are Microservices?
   - Monolithic vs. Microservices Architecture
   - Upstream and Downstream in Microservices
   - Role of API gateway in Microservices
   - Are Microservices Distributed Systems?
   - Advantages and Disadvantages
2. Building Blocks of Microservices
   - How do Microservices Communicate With Each Other?
   - Microservices Communication Patterns
   - 10 Microservices Design Principles That Every Developer Should Know
   - Service Registry in Microservices
   - Service Discovery in Microservices
   - Service Mesh in Microservices
3. Security in Microservices
   - Authentication and Authorization
   - Session Management in Microservices
   - Security Measures for Microservices Architecture

Swipe for more

4. Designing Microservices

- Circuit Breaker Pattern
- CQRS Design Pattern
- API Gateway Pattern
- Sidecar Design Pattern
- Retry Pattern
- API Composition Pattern
- Edge Pattern
- Service Registry Pattern
- Centralized Logging for Microservices
- Distributed Logging for Microservices
- MVC Architecture vs. Microservices Architecture
- End-to-End Microservices Testing
- Microservices Cross-Cutting Concerns
- Reducing Latency in Microservices

Swipe for more



5. Advanced Topics in Microservices
   - Strangler Fig Pattern
   - Steps to Migrate From Monolithic to Microservices Architecture
   - Circuit Breaker with Bulkhead Isolation in Microservices
   - Websockets in Microservices Architecture
   - Timeout Strategies in Microservices Architecture
   - Saga Pattern
   - How to Design a Microservices Architecture with Docker containers?
   - Orchestration vs. Choreography in Microservices
   - Database Per Service Pattern
   - Long-Tail Latency Problem in Microservices
   - Distributed Tracing in Microservices
   - Stateful vs Stateless Microservices
   - AI and Microservices Architecture
6. Interview Preparation
   - Top 50 Microservices Interview Questions
   - Top Books to Learn Microservices Architecture

Swipe for more

Mohamed El Laithy

Content Creator

If you find this helpful, please like and share it with your friends

Created By:
Mohamed El Laithy

Follow Me:
/in/mohamed-el-laithy-0155b2173/
/dev.to/mohamed_el_laithy