MATTER	Advanced Web Programming	**NRC**	8011
RACE	Software Engineering	**Teacher**	Dr. Edison Lascano
THEME	Microservices		
Name	Andres Espin		

1. Introduction:

Microservices represent a modern architectural paradigm that decomposes applications into modular, self-contained services. Each service operates independently, focusing on a discrete business capability, and can be developed, deployed, and scaled autonomously. This approach enhances scalability, agility, and maintainability in enterprise systems, aligning with the demands of cloud-native and distributed computing.

2. Architectural Foundations and Patterns

Core Principles

Service Autonomy: Each microservice owns its logic, data storage (via the Database per Service pattern), and lifecycle.

Decentralized Governance: Teams independently manage services, promoting technology diversity (e.g., Java with Spring Boot, Node.js for lightweight tasks).

API-Driven Communication: Services interact via well-defined REST APIs (synchronous) or messaging systems (asynchronous).

Design Patterns for Resilience

Circuit Breaker (Netflix [5]): Mitigates cascading failures by isolating faulty dependencies.

Event Sourcing + CQRS: Maintains an immutable audit log of state changes (Event Sourcing) while separating read/write operations (CQRS) for scalability (Richardson, 2022).

DevOps Integration: CI/CD pipelines automate testing and deployment, enabling rapid, reliable updates.

Technology Ecosystem

Infrastructure Tools

- **Containerization**: Docker packages services into portable units.
- **Orchestration**: Kubernetes manages scaling, load balancing, and service discovery.
- **API Gateway**: Centralizes request routing (e.g., to Authentication, Users, Products, Orders services), security, and rate limiting.

Communication Channels

- **Synchronous:** REST/HTTP for real-time interactions (e.g., user authentication).
- **Asynchronous**: Event Bus (e.g., RabbitMQ, Kafka) for decoupled workflows (e.g., order processing triggering inventory updates).

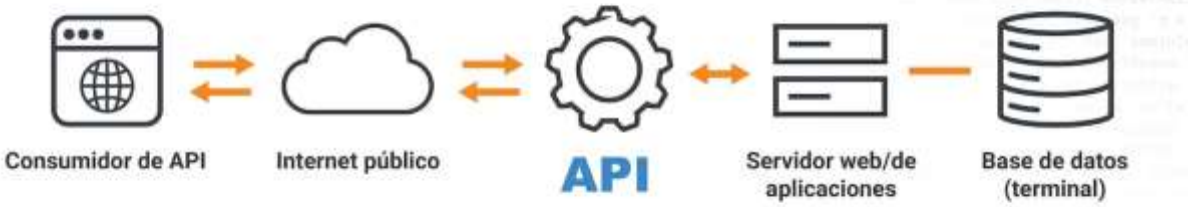
4. Architectural Workflow

A typical implementation follows this flow:

1. **Client Request**: Web/mobile clients send requests to the API Gateway.
2. **Routing**: The Gateway directs traffic to relevant microservices (e.g., Authentication

validates credentials).

3. **Service Execution**:
   - *Users*: Manages user profiles (dedicated database).
   - *Products*: Handles inventory (isolated data store).
   - *Orders*: Processes transactions, emitting events to update product stock or user history.
4. **Data Consistency**: Event-driven messaging ensures eventual consistency across services (e.g., order placement reduces product availability).



Cómo funciona una API web

Akamai

3. Conclusion

Microservices architecture has established itself as an essential model for modern application development, offering significant advantages in scalability, flexibility, and resilience. By breaking down monolithic systems into independent, specialized services, organizations can accelerate development cycles, adopt heterogeneous technologies, and scale critical components in a granular manner. Patterns such as Circuit Breaker and Event Sourcing solve challenges inherent to distributed systems, while tools such as Docker, Kubernetes, and API Gateway provide the infrastructure needed to efficiently manage the lifecycle of services.

However, this architecture is not without its complexities. Decentralizing databases, coordinating distributed transactions, and monitoring services require rigorous planning and the adoption of robust DevOps practices. Moreover, its successful implementation depends on clear governance and a balance between autonomy and standardization.

In scenarios where agility, fault tolerance and adaptability are a priority (such as in e-commerce platforms or cloud-native systems), microservices represent a strategic solution. However, their adoption must be critically evaluated, considering the technical maturity of the team, the complexity of the domain and the consistency requirements, to avoid over-engineering. In essence, microservices are not a panacea, but they are a powerful option when aligned with specific business needs and operational capacity.

4. References

Newman, Sam. "Building Microservices" (O'Reilly Media) Evans, Eric. "Domain-Driven Design" (Addison-Wesley)