UTT
UNIVERSIDAD TECNOLÓGICA DE TIJUANA
GOBIERNO DE BAJA CALIFORNIA

TEMA:

Architecture specification

PRESENTADO POR:

Valenzuela Esparza Javier Ivan

GRUPO:

10B

MATERIA:

Desarrollo Movil Integral

PROFESOR:

Ray Brunett Parra Galaviz

FECHA:

07/01/2025

MICROSERVICES ARCHITECTURE

The microservices architecture is a modern approach to designing software systems, where applications are structured as a collection of small, autonomous services. Each service is responsible for a specific functionality and operates independently, allowing for greater flexibility, scalability, and maintainability.

Key Characteristics of Microservices Architecture

1. **Service Independence**
   Each service is developed, deployed, and maintained independently. This autonomy enables teams to work on different services simultaneously, reducing development time.
2. **Decentralized Data Management**
   Services typically have their own databases, tailored to their specific needs. This decentralization enhances scalability and reduces coupling between services.
3. **Communication via APIs**
   Services communicate using lightweight protocols such as REST, GraphQL, or gRPC. These APIs enable seamless interaction between services, even if they are developed in different languages.
4. **Scalability**
   Microservices allow independent scaling of services based on demand. For example, a service handling user authentication can scale independently from a service managing reports.
5. **Resilience**
   Failure in one service does not necessarily affect the entire system. This fault isolation improves the reliability and availability of the application.
6. **Technology Diversity**
   Teams can choose the most suitable technology for each service, as long as it adheres to the communication standards defined for the system.
7. **Continuous Delivery and Deployment**
   Microservices align well with DevOps practices, as independent services can be tested and deployed without affecting others.

Advantages of Microservices Architecture

- **Flexibility**: Enables easier updates, enhancements, and experimentation without impacting the entire system.
- **Team Autonomy**: Teams can work independently on different services, enhancing productivity.
- **Enhanced System Resilience**: Faults in one service do not propagate across the system.

- **Optimized Resource Usage**: Independent scaling ensures resources are used efficiently.

Challenges and Best Practices

1. **Complexity in Management**
   Managing multiple services requires robust tools and frameworks for orchestration and monitoring, such as Kubernetes and Prometheus.
2. **Communication Overhead**
   Since services communicate over the network, latency and reliability issues must be addressed. Protocols like gRPC can minimize overhead.
3. **Data Consistency**
   Distributed data systems can lead to challenges in maintaining consistency. Tools like Apache Kafka or RabbitMQ are used for event-driven communication to handle these issues.
4. **Security Concerns**
   Each service exposed via APIs increases the attack surface. Security measures like API gateways and OAuth are essential to mitigate risks.

Bibliographical Sources

Production-grade container orchestration. (s/f). Kubernetes. Recuperado el 8 de enero de 2025, de https://kubernetes.io/

(S/f-c). Amazon.com. Recuperado el 8 de enero de 2025, de https://aws.amazon.com/es/microservices/