Microservice Communication Methods

1. HTTP/REST (Representational State Transfer)

How it works: Microservices communicate using standard HTTP methods like GET, POST, PUT, DELETE. RESTful APIs use JSON or XML for data exchange.

Why: Simple, widely supported, and easy to implement. Works well for synchronous requests where the client waits for a response from the service.

Example Situation: A web application that needs to interact with a user service to fetch profile data or update user information in real-time.

2. gRPC

How it works: gRPC is a high-performance, open-source RPC (Remote Procedure Call) framework that uses HTTP/2 for transport and Protocol Buffers (protobufs) for serialization.

Why: Faster than REST for high-throughput systems due to its binary format (protobufs) and ability to support bi-directional streaming.

Example Situation: Financial trading system that requires high performance and low latency.

3. Message Brokers (e.g., Kafka, RabbitMQ, SQS)

How it works: Microservices communicate by publishing and consuming messages via queues or topics. Communication is often asynchronous.

Why: Useful for decoupling services, handling failures, and implementing event-driven architectures.

Example Situation: A recommendation system that processes user interaction events through Kafka.

4. WebSockets

How it works: WebSockets establish a persistent connection between services over TCP, allowing real-time, full-duplex communication.

Why: Best for real-time applications where the server sends updates to the client.

Example Situation: A stock trading platform that pushes real-time price updates.

5. UDP (User Datagram Protocol)

How it works: UDP sends data in packets without establishing a connection, making it faster but less reliable than TCP.

Why: Suitable for systems where speed is critical, and packet loss is acceptable.

Example Situation: Live video streaming where losing a few packets is fine.

6. TCP (Transmission Control Protocol)

How it works: TCP ensures reliable, ordered, and error-checked delivery of data.

Why: Reliable for critical communication, such as payment processing.

Example Situation: A payment processing service where reliable communication is crucial.

7. GraphQL

How it works: API query language that allows clients to request only the specific data they need.

Why: Reduces over-fetching and improves efficiency in complex systems.

Example Situation: A mobile app fetching data from multiple services but requesting minimal data.

8. Service Mesh (e.g., Istio, Linkerd)

How it works: Manages communication between microservices, offering features like load balancing and monitoring.

Why: Provides fine-grained control in large-scale systems.

Example Situation: A SaaS application where reliability and observability are critical.

9. Event Streaming (e.g., Apache Kafka, Pulsar)

How it works: Services publish data streams in real-time, and other services process these events asynchronously.

Why: Ideal for real-time processing and data analytics.

Example Situation: Real-time analytics platform processing clickstream data.

10. Shared Databases

How it works: Services communicate indirectly by reading from and writing to a shared database.

Why: Not recommended in modern microservices but common in legacy systems.

Example Situation: A legacy system where services update and read shared customer records.

In WhatsApp, microservices communicate using:

1. WebSockets for real-time messaging.
2. Message brokers for asynchronous background tasks.
3. REST APIs for non-real-time interactions.
4. gRPC for internal service communication.
5. TCP/UDP for media streaming.
6. End-to-end encryption for data security.
7. Push notifications for notifying users when inactive.