MICROSERVICES

Microservices are an architectural approach of developing software applications as a collection of small, independent services that communicate with each other over a network.

→ Microservices break down the application into smaller, loosely coupled services.

→ Each microservice is designed to perform a specific business function and can be developed, deployed and scaled independently.

→ Microservices are building blocks of modern applications

→ Each Ms can be written in a variety of languges and frameworks and act as a mini application ~~of~~ on its own.

→ They provide flexibility, Scalability and easier maintainance

How Mss work:

- Modular structure
- Independent functions
- Communication
- Flexibility
- Independence & Updates
- Scalability
- Continuos Improvement

Components of Microservice Architecture:

→ Microservices

→ API Gateway

→ Service Registry & Discovery

→ Load Balancer

→ Containerization

→ Event Bus / Message Broker

→ Centralized Logging & Monitoring

→ Database per microservice

→ Caching

→ Fault tolerence & Resilience Components.

Design Patterns in Microservices

→ Aggregator

→ API Gateway

→ Event sourcing

→ Strangler

→ Decomposition

Moving from Monolithic to Microservices

→ Evaluate Monolith.

→ Define Microservices

→ strangler pattern

→ API Definition

→ CI/CD Implementation

→ Decentralize Data

→ Service Discovery

→ Logging & Monitoring

→ Cross cutting concerns.

→ Iterative improvement.

Role of Microservices in Dev-Ops

→ Continuous Improvement / Continuous Development (CI/CD)

→ Agile Development

→ Continuos Monitoring & Logging.

Benefits of Microservice Architecture:

→ Modularity & Decoupling

* Independent Development

* Isolation of failures

→ Scalability
  * Granular Scaling
  * Elasticity

→ Technology Diversity

→ Autonomous Teams

→ Rapid Deployment & Continuous Delivery
  * Faster Release cycles
  * Continuous Integration & Deployment (CI/CD)

→ Easy Maintainance
  * Isolated codebases
  * Rolling updates.

Challenges of Microservice Architecture:

→ Complexity of Distributed systems

→ Increased development & Operational Overhead

→ Inter-service communication overhead

→ Data consistency & Transaction Management

→ Deployment challenges

→ Monitoring & Debugging Complexity.

API Gateway

API Gateway is a key component in system design particularly in microservice architecture

→ It serves as a centralized entry point for managing and routing requests from clients to the appropriate Mx or backend service within a system.

→ API Gateway is a service that sits between clients & backend services, acting as a reverse proxy to accept incoming requests from clients, perform various operations such as routing, authentication, rate limiting & then forward those requests to appropriate backend services.

How API Gate way Works:

API Gateway acts as a single entry point for clients to access various microservices or backend services. It abstracts the underlying architecture and provides a unified interface for clients to interact with the system.

~~Routing~~:

Step 1: Routing

→ When a client sends a request to the API Gateway, it examines the request to determine which mx or service should handle it. Routing can be based on various criteria such as URL path, HTTP method or headers.

Step-2 : Protocol Translation:

The API Gateway can translate incoming requests from protocol to another. For example, it can accept HTTP requests from client & convert them to gRPC or websocket requests for backend services.

step 3: Request Aggregation:

In some cases, a client may need to fetch data from multiple services to fullfill a single request. The API Gateway can aggregate these requests into a single call to improve efficiency and reduce number of round trips

Step4: Authentication & Authorization:

API Gateway can handle authentication & authorisation for incoming requests. It can verify the identity of the client and check if the client has the necessary permissions to access the requested resources.

Step 5 : Rate Limiting & Throttling

To prevent abuse and ensure fair usage of resources, the API Gateway can enforce rate limiting & throttling policies. It can limit the number of requests a client can make within a certain time period.

~~Load Balancing~~

Step 6: Load Balancing

The API Gateway can distribute incoming requests accoross multiple instances of a service to ensure high availability & scalability.

Step 7: Caching:

To improve performance, the API Gateway can cache responses from backend services & serve them directly to clients for subsequent identical requests

Step 8: Monitoring & Logging:

The API Gateway can collect metrics & logs from incoming requests, providing insights into the usage & performance of the system.