5.2. MICROSERVICE DEPENDENCY TREE

Definition

A **Microservice Dependency Tree** is a hierarchical representation that shows the dependencies and communication relationships among various microservices in an application.

It helps developers understand:

- Which services depend on others.
- Request and response flow.
- Service interactions.
- System complexity.

Structure of Dependency Tree

Customer Service
|
Order Service
|
Payment Service
|
Notification Service

In the above example:

- Customer Service depends on Order Service.
- Order Service depends on Payment Service.
- Payment Service depends on Notification Service.

Components

Parent Service

Initiates requests to other services.

Child Service

Provides functionality to parent services.

Service Registry

Maintains service information.

API Communication

Used for interaction among services.

Advantages

- Better visibility of service relationships.
- Easy fault identification.
- Simplified debugging.
- Improved monitoring.

Disadvantages

- Complex management in large systems.
- Increased communication overhead.
- Difficult dependency tracking.

---

2. Challenges with Microservices

Although microservices offer many benefits, they also introduce several challenges.

---

1. Distributed System Complexity

Microservices run on different servers and communicate through networks.

Problems:

- Network failures.
- Communication delays.
- Message loss.

---

2. Data Management

Each service has its own database.

Challenges:

- Data consistency.
- Data synchronization.
- Transaction management.

---

3. Service Communication

Services communicate through APIs.

Issues:

- API failures.
- Version conflicts.
- Latency problems.

---

4. Monitoring and Logging

Multiple services generate huge amounts of logs.

Challenges:

- Tracking failures.
- Monitoring performance.
- Log aggregation.

---

5. Security

Each service becomes an attack point.

Security requirements:

- Authentication.
- Authorization.
- Encryption.
- API security.

---

6. Testing Complexity

Testing becomes difficult because multiple services interact together.

Types:

- Unit Testing
- Integration Testing
- End-to-End Testing

---

7. Deployment Complexity

Managing hundreds of services is difficult.

Problems:

- Configuration management.
- Service discovery.
- Load balancing.

---

3. SOA vs Microservices

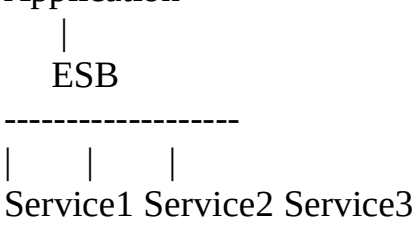
Definition of SOA

Service-Oriented Architecture (SOA) is an architectural style where applications are built using reusable services connected through an Enterprise Service Bus (ESB).

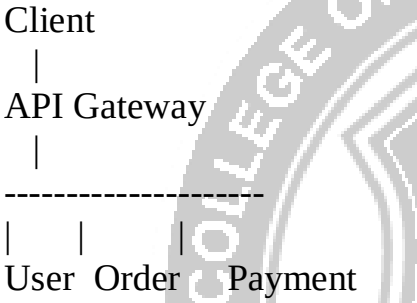
Comparison between SOA and Microservices

SOA	Microservices
Large services	Small services
Enterprise-wide architecture	Application-specific architecture
Uses ESB	Uses APIs
Shared database	Separate databases
Complex deployment	Independent deployment
Less scalable	Highly scalable
Slower development	Faster development
Heavy communication	Lightweight communication

SOA Architecture



Microservice Architecture



Advantages of Microservices over SOA

- Better scalability.
- Faster deployment.
- Independent services.
- Easier maintenance.
- Cloud-native support.

---

4. Microservice and API

Definition of API

API (**Application Programming Interface**) is a set of rules that allows software applications and microservices to communicate with each other.

APIs act as communication channels between services.

Need for APIs

Without APIs:

- Services cannot communicate.
- Data sharing becomes difficult.

With APIs:

- Faster communication.
- Standardized interaction.
- Better interoperability.

Types of APIs

REST API

Most popular API type.

Features:

- Uses HTTP protocol.
- Lightweight.
- Easy to use.

Methods:

- GET
- POST
- PUT
- DELETE

SOAP API

Features:

- XML-based communication.
- High security.
- Enterprise applications.

GraphQL API

Features:

- Flexible data retrieval.
- Efficient communication.

- Reduces data transfer.

API Communication Flow

Client
|
REST API
|
User Service
|
Database

Benefits of APIs in Microservices

- Loose coupling.
- Reusability.
- Easy integration.
- Platform independence.
- Better scalability.

---

5. Deploying and Maintaining Microservices

Deployment of Microservices

Deployment means making microservices available for users.

Steps in Deployment

Step 1: Develop Service

Create business logic.

Step 2: Build Service

Compile source code.

Step 3: Containerization

Package service using Docker.

Step 4: Testing

Perform automated testing.

Step 5: Deployment

Deploy to cloud or servers.

Docker for Deployment

Definition

Docker is a containerization platform used to package applications and their dependencies.

Advantages:

- Lightweight.
- Portable.
- Fast deployment.

Kubernetes for Deployment

Definition

Kubernetes is a container orchestration platform.

Functions:

- Scaling.
- Load balancing.
- Auto-healing.
- Resource management.

Deployment Architecture

Developer
|
Git Repository
|
Jenkins CI/CD
|
Docker Container

|
Kubernetes Cluster
|
Production Server

Maintaining Microservices

Maintenance ensures continuous availability and performance.

---

1. Monitoring

Tracks:

- CPU usage.
- Memory usage.
- Network traffic.
- Service health.

Tools:

- Prometheus
- Grafana

---

2. Logging

Stores system events and errors.

Benefits:

- Troubleshooting.
- Performance analysis.
- Security auditing.

Tools:

- ELK Stack
- Splunk

---

3. Security Maintenance

Includes:

- Authentication.
- Authorization.
- API security.
- Data encryption.

---

4. Backup and Recovery

Protects against:

- Data loss.
- Service failures.
- System crashes.

---

5. Scaling

Horizontal Scaling

Adding more service instances.

Vertical Scaling

Increasing server resources.