Micro services

* Monolith architecture

- all software components are executed in a single process.
- No distribution of any kind.
- strong coupling both all layers.
- usually implemented as s?b.

* Pros

- Easier to design
- Performance → fast

* first coined in 1998.
- ~~Apps~~ usually implemented using SOAP & WSDL.
- implemented with ESB

Enterprises Service Bussiness.

- communicate with the help of service → handle request & response.

SOAP boys

- sharing Data & functionality.
- Polyglot between services → avoid platform dependencies.

Problems with — Monolith

- technology
- deployment
- cost

Monolith → all components developed using the same development platform.

- can't use specific platform for specific features.
- future upgraded is a program → need to upgrade the whole app.

Inflexible Deployment

- new deployment is always for whole app.
- No way to deploy only part of the app.
- Even when updating only one component — the whole codebase is deployed.
- forces regress testing for every deployment
- long development cycles.

Cons of Monolith

- Codebase is large & complex.
- Every little change can affect other components.
- Testing not always detects all the bugs.
- very difficult to maintain,
- Might make the system isolate.

Microservices Architecture

→ first appeared in 2011.

→ 2014 → Martin Fowler & James Lewis
↓
published article on "Microservices".

Features – Independent deployment.
- well defined interface.
- Quick development
- loosely coupled

→ With Microservices each ~~device~~ service has its own database.

→ short deployment cycles.
→ Automation is essential.

* Design for failure –

↳ There are a lot of processes and a lot of network traffic.

↳ Motivation :– Increase system reliability.

* Impl. attributes –

> Componentization
> Organized around business capabilities
> Decentralized governance.
> Infrastructure automation.

* The Architecture Process –

- Understand the System's Requirement
- Understand the non-functional Requirement.
- Map the Component
- Select the Technology stack
- Design the Architecture
- Write Architecture Document
- Support the Team.

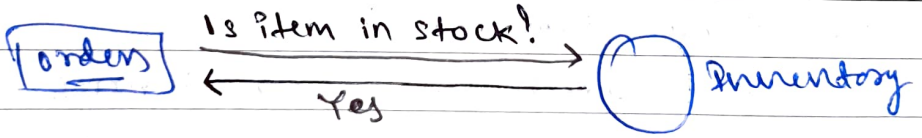
* Mapping based on –

- Bussiness requirements.
- functional autonomy
- Data entities → orders, item (related to ID)
- Data autonomy. → atomic unit.

* Business requirement → eg. Orders management
  ↳ Add, remove, update calculate amount.

* Communication Pattern → It is crucial
  - 1-to-1 Sync
  - 1 to 1 Async
  - Pub-Sub / Event Driven
  } Pattern

① 1-to-1 Sync — A service calls another service and waits for the response
  - used mainly when the first service needs the response to continue processing.



Pros- Immediate response
Error handling
Easy to implement

Cons - Performance
↳ low.

② 1-to-1 Async —
  - A service calls another service and continues working.
  - Doesn't wait for response — fire & forget.
  - Used — 1st service wants to pass a message to the other service.

Handle Payment

- Pub-Sub – A service wants to notify other services about something.
  - no idea how many services listen
  - Doesn't wait for response → fire & forget.
  - used → when 1st service wants to notify about an impt. event in the system.

- Development Platforms

  .Net , .NET Core , Java
  node.js , PHP , Python

- Data Store
  - Relational Database
  - No SQL Database → schema less → store in (JSON)
  - Cache → stores in memory data
  - Object Store

NoSQL Database → Example – mongo DB, amazon DynamoDB, Couchbase, Azure Cosmos DB.

Cache – Redis

Object store – Stores un-structural, large data
  - Documents, Photos, files.
    - eg. → Microsoft Azure
    - → amazon S3
    - → MINIO

23 Sunday

FEBRUARY '20

M	T	W	T	F	S	S	M	T	W	T	F	S	S
					1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29								

Testing Microservices

-> Types—

- Unit Test
- Integration Test
- End-to-End Tests

1) Unit Test –

- Test individual code units
  - Method, interface
- Usually automated
- Developed by the developers

In Microservices → Test only in-process code
→ use same framework & methodologies.

2) Integration Testing

- Test the service's functionality
- Cover all codes path in service
  - Database, other services

↳ Three types –

- > Fake – stores data in-memory
- > Stub – replaces data stored in DB
- > MOCK – Holds no data

– Use Service's API

3) End-to-End Tests

- Test the whole flow of the system
- Test for end state
- Extremly fragile
- Reluise code

* Service Mesh
  - Manage all service-to-service communication
  - Provides additional services

Provide services

- Protocol conversion
- Communication security
- Authentication
- Reliability (timeouts, retries, health checks, Circuit breaking)
- Monitoring
- Service Discovery.
- Testing (A/B testing, traffic splitting)
- Load balancing.

Types of Mesh service -

1) In-Process → Performance
2) sidecar → Platform & code agnostic

Logging

- Recording the system's activity
- Audit
- Documenting errors

Monitoring

- Based on system's metrices
- Alerting when needed.

Conway's Law

- Introduced in ~~1968~~ 1967 by Melvin Conway
- describe relationship betn the organization and software structure / architecture.

Traditional Team v/s Ideal Team

↓ responsible

responsible for all aspects

+

front-end

Back-end

DBA.

Breaking Monolith to Microservices—

Strategies — 1) ~~Breaking M~~ New Modules as Services
2) Separate Existing Modules to Services
3) Complete Rewrite

Pros

- End result is pure Microservice architecture
- Opportunity for Modernization.

Cons:

- Takes Time
- Rigorous Testing Required.