MICROSERVICES:

- Monolithic
- Distributed
  - SOA
  - Microservices

— Monolithic
  - Pros:
    -
  - Cons:
    - Flexibility
    - Responsiveness (adaptable to the change is difficult)
    - Availability (if any issue in one point, it affects everything depending on that)
    - Heterogeneity (change is hard and needs rewrite)

— Distributed Architecture
  - SOA(Service Oriented Architecture)
    -Process oriented, adaptable to change, incrementally built % deployed, loosely coupled, implementation abstraction
    - Cohesion and Coupling
    -cohesion: class or app contains all the required dependencies present (any software must be highly cohesive)
    - Pros:
      - Components can be designed developed independently
      - loosely coupling
      - heterogeneity is possible to integrate and adopt
      - reuse of services
    - Cons:
      - since they depend on one another, tight coupling still remains
      - SOA started to focus on technology and less on business

— Microservices Architecture
  - they are small , autonomous services that work together
  - they make organization agile and respond faster to the changes
  - DDD(domain driven design), continuous delivery, small autonomous teams
  - highly maintainable
  - loosely coupled
  - owned by small team
  - each service can have its own technology in implementation
  - Reduce Dependency
  - Better troubleshooting

- characteristics
    - componentization via services
    - products not projects (u build u run it)
    - services are organized around business capabilities
    - smart Endpoints and dumb pipes
    - decentralized governance
    - decentralized data management
    - infrastructure automation
    - design for failure
    - evolutionary design

- principles
    - model around ur business domain
    - build a culture of automation
    - hide implementation details
    - embrace decentralization
    - deploy independently
    - focus on consumers first
    - isolate failure
    - make them highly observable

— Limitations of Microservices
    - distributed system complexity
    - more services equals more resources

— Reactive System
    - Responsive
    - Resilient
    - Elastic
    - message driven

— Design Patterns. https://microservices.io/patterns/
- The patterns are organized into
    - Context
    - problem
    - forces
    - solution
    - resulting context
    - issues

— Microservices Design

—Saga pattern:

— Service Registry
A *service registry* is a database of services used to keep track of the available instances of each microservice in an application.
The service registry needs to be updated each time a new service comes online and whenever a service is taken offline or becomes unavailable.
This can be achieved with either self-registration or third-party registration.

https://konghq.com/learning-center/microservices/service-discovery-in-a-microservices-architecture#:~:text=Microservices%20Service%20Registry&text=A%20service%20registry%20is%20a,taken%20offline%20or%20becomes%20unavailable.

— Messaging (Apache Kafka)
Kafka is primarily used **to build real-time streaming data pipelines and applications that adapt to the data streams**.
It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data
Asynchronous messaging for inter-service communication. Services communicate by exchanging messages over messaging channels.

Topic : messages sent through topics
Partitions: topics are partitioned for scalability (to send to the instances in the consumer)
Kafka broker: kafka server which maintains the partitions / replicas
Kafka cluster: collection of kafka broker servers
replicas
Offset: logical marker of until where the messages are consumed by the client
Zookeeper: monitors the health of the Kafka broker servers. If any leader server comes down, it elects the kafka broker among the remaining brokers

How to apply the patterns

Application architecture patterns

- Monolithic architecture
- Microservice architecture

Decomposition

- Decompose by business capability
- Decompose by subdomain
- Self-contained Servicenew
- Service per teamnew

Refactoring to microservicesnew

- Strangler Application
- Anti-corruption layer

Data management

- Database per Service
- Shared database
- Saga
- https://microservices.io/patterns/data/saga.html
- https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga
- API Composition
- CQRS
- Domain event
- Event sourcing

Transactional messaging

- Transactional outbox
- Transaction log tailing
- Polling publisher

Testing

- Service Component Test
- Consumer-driven contract test
- Consumer-side contract test

Deployment patterns

- Multiple service instances per host
- Service instance per host
- Service instance per VM
- Service instance per Container
- Serverless deployment
- Service deployment platform

Cross cutting concerns

- Microservice chassis
- Service Template
- Externalized configuration

Communication style

- Remote Procedure Invocation
- Messaging
- Domain-specific protocol
- Idempotent Consumer

External API

- API gateway
- Backend for front-end

Service discovery

- Client-side discovery
- Server-side discovery
- Service registry
- Self registration
- 3rd party registration

Reliability

- Circuit Breaker

Security

- Access Token

Observability

- Log aggregation
- Application metrics
- Audit logging
- Distributed tracing
- Exception tracking
- Health check API
- Log deployments and changes

UI patterns

- Server-side page fragment composition
- Client-side UI composition