




# Förståelse för Monolitisk arkitektur, SOA och Microservices

En genomgång av olika  
arkitekturmodeller inom  
systemutveckling





# Introduktion

- Bakgrund om arkitekturer och deras användningsområden.
- Syftet med att välja rätt arkitektur baserat på skalbarhet och affärsbehov.
- Fokus på skalbarhet, modularitet och autonomi.

# Gloslista

**Virtuell skalning:**

Processen att tilldela mer resurser (CPU, RAM) till en applikation på en enda maskin för att hantera ökad arbetsbelastning.

**Horisontell skalning:**

Att lägga till fler maskiner eller noder till ett system för att sprida arbetsbelastningen och förbättra prestanda.

**Message Queue:**

En infrastruktur för att hantera och lagra meddelanden som skickas mellan olika komponenter i ett system för att möjliggöra asynkron kommunikation. Exempel: RabbitMQ, Kafka.

**RPC (Remote Procedure Call):**

En teknik för att möjliggöra kommunikation mellan olika system eller processer, vanligtvis på olika datorer i ett nätverk.

**gRPC:**

Ett modernt RPC-ramverk utvecklat av Google, baserat på HTTP/2 och Protobuf för hög prestanda.

**XML-RPC:**

Ett äldre protokoll som använder XML för att serialisera förfrågningar och svar.

**JSON-RPC:**

Ett lättviktigt protokoll som använder JSON för att serialisera förfrågningar och svar.

**SOAP (Simple Object Access Protocol):**

En äldre protokollbaserad RPC-variant som används i företagsapplikationer för strukturerad dataöverföring.

**Distribuerade system:**

System där olika komponenter körs på separata maskiner men samverkar för att uppnå ett gemensamt mål.

**Synkron kommunikation:**

Kommunikationen sker i realtid, där klienten väntar på ett svar från servern innan den fortsätter.

**Enterprise Service Bus (ESB):**

En integrationsplattform som används för att möjliggöra kommunikation mellan olika tjänster och komponenter i ett distribuerat system, ofta inom SOA.

**Glue Code:**

Kod som fungerar som en brygga för att integrera olika system eller komponenter som annars inte skulle kunna kommunicera.

**Event Streaming:**

En teknik för att kontinuerligt fånga och bearbeta händelser i realtid, till exempel med Apache Kafka.

**Asynkron kommunikation:**

Kommunikation där klienten inte behöver vänta på ett svar från servern, utan kan fortsätta med andra uppgifter.

**Load Balancer:**

En komponent som fördelar inkommande trafik mellan flera servrar för att optimera resursanvändning och förbättra prestanda.

**API Gateway:**

En ingångspunkt för att hantera API-anrop, autentisering, routing och loggning i mikrotjänstarkitekturer.

**Circuit Breaker:**

Ett mönster som används för att förhindra överbelastning i distribuerade system genom att avbryta anrop till tjänster som inte fungerar.

**Service Registry:**

En katalog där tjänster i en mikrotjänstarkitektur registreras och upptäcks av andra tjänster.

**Orkestrering:**

Samordning och hantering av arbetsflöden mellan olika tjänster eller system.

**Containerization:**

Tekniken att paketera en applikation och dess beroenden i en isolerad behållare. Exempel: Docker.



# Historisk Kontext

2000

- Monolitisk arkitektur var standard.

2010

- Microservices utvecklades för skalbarhet och flexibilitet.

2000

- SOA (Service-Oriented Architecture) introducerades för att hantera komplexa system.

# Monolitisk Arkitektur

## Definition

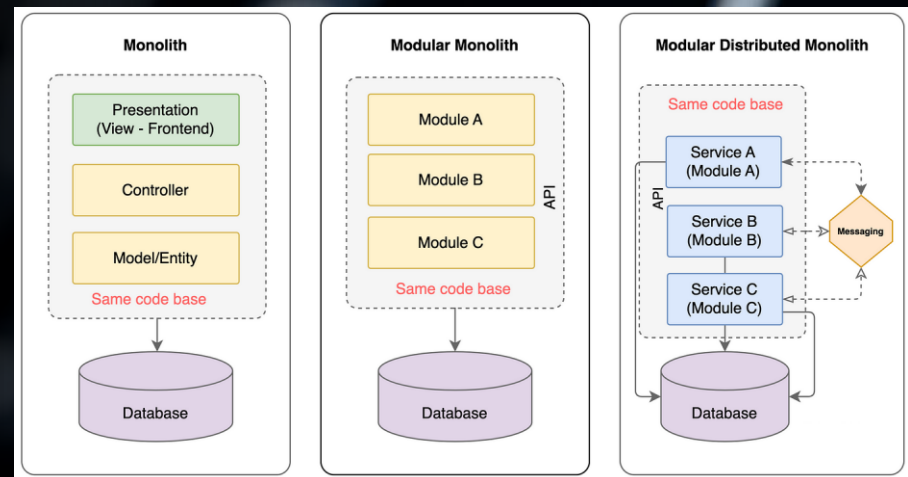
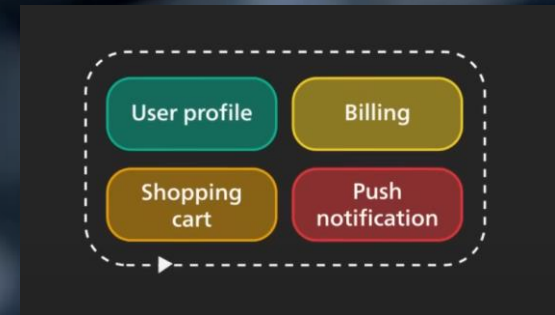
Ett enda stort program där alla delar är sammanfogade. / Alla funktioner är sammanflätade i en enda applikation.

## Fördelar

1. Enkel att utveckla och testa.
2. Mindre komplex distribution.
3. Lägre driftsättningskomplexitet.

## Nackdelar

1. Begränsad skalbarhet.
2. Svårt att skala horisontellt.
3. Svårt att underhålla vid stor tillväxt.
4. Hög kopplingsgrad mellan komponenter.
5. Begränsad modularitet





# SOA (Service-Oriented Architecture)

## Definition

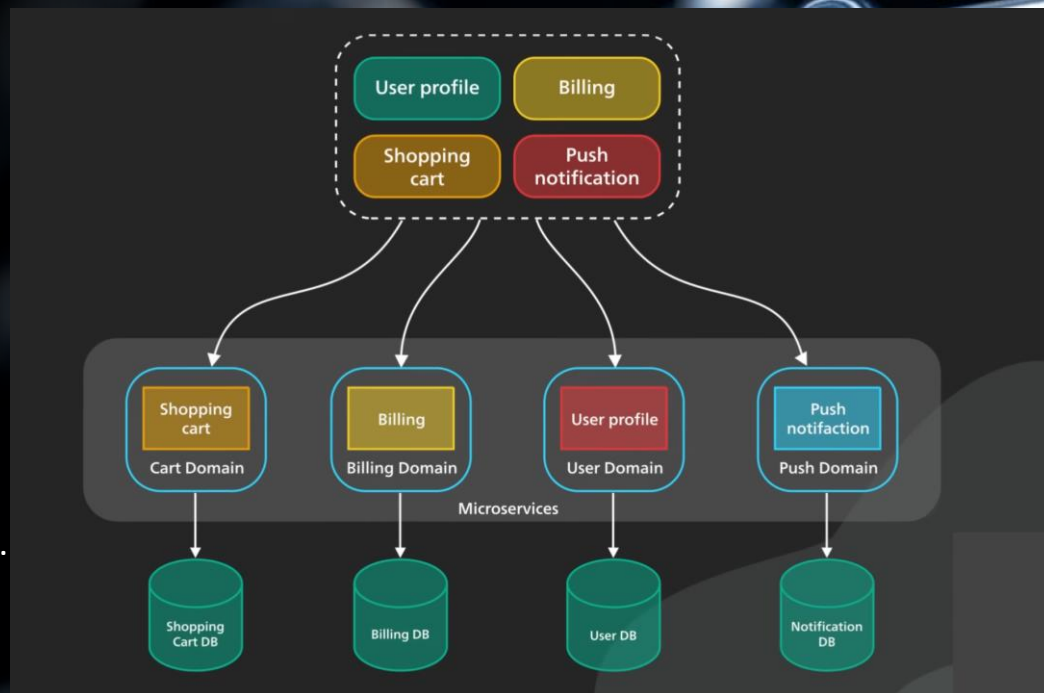
Applikationer delas upp i tjänster som kommunicerar via standardiserade protokoll. Mjukvarusystem uppbyggda av distribuerade tjänster. Komponenterna kommunicerar över nätverk.

## Fördelar

1. Återanvändbara tjänster.
2. Lättare att integrera olika system.
3. Modularitet jämfört med monoliter.

## Nackdelar

1. Komplexitet i tjänstekontrakt och styrning.
2. Prestandaproblem vid många tjänsteanrop.
3. Begränsad autonomi.
4. Högre kopplingsgrad än mikrotjänster.



# Microservices Arkitektur

## Definition

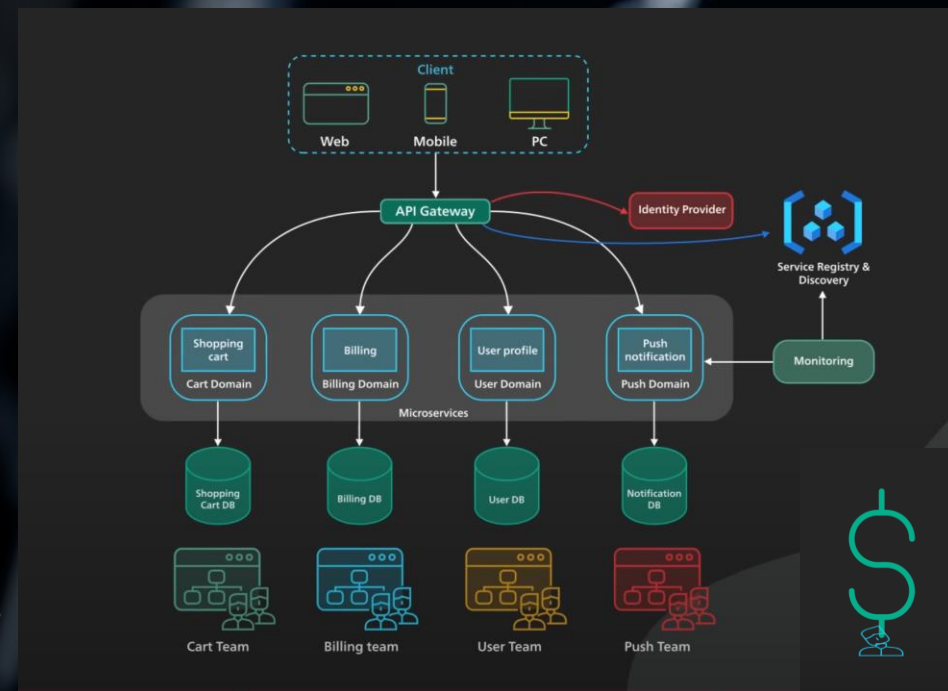
Små, självständiga tjänster som hanterar specifika funktioner. Applikationer byggs som en samling mindre tjänster. Varje tjänst körs som en separat process.

## Fördelar

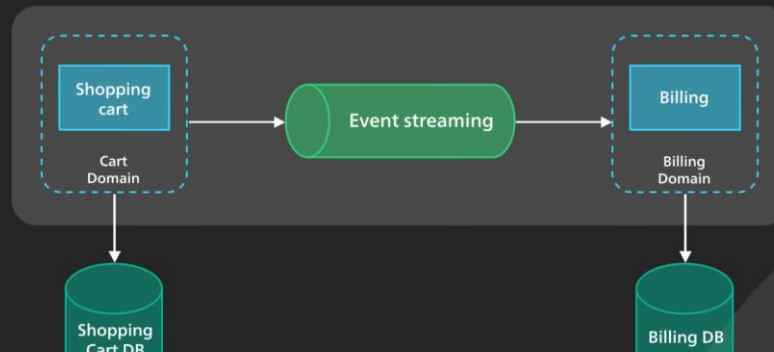
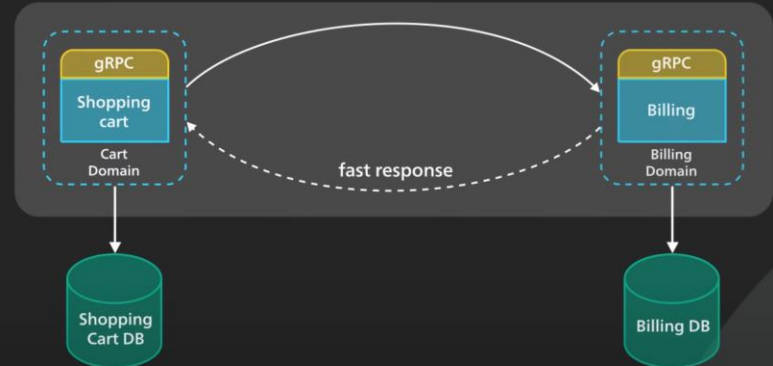
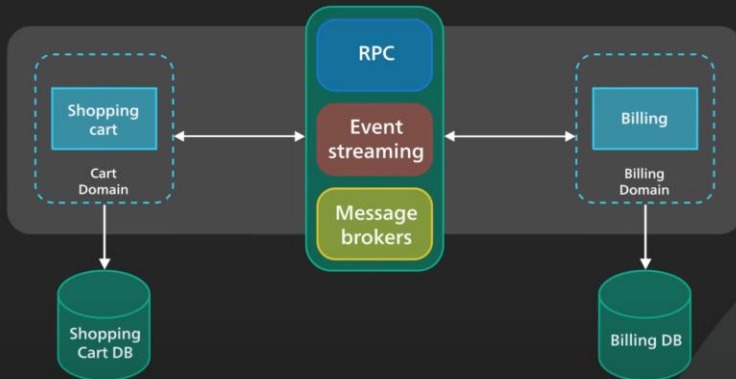
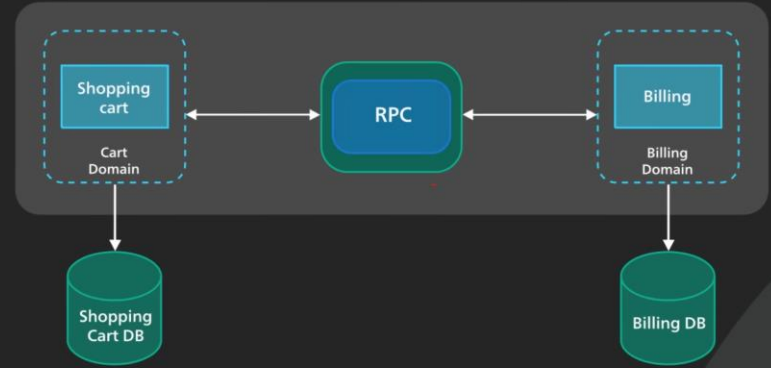
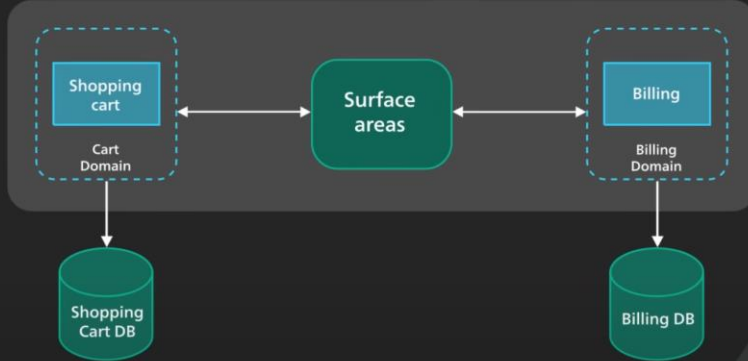
1. Hög flexibilitet och skalbarhet.
2. Granulär skalbarhet.
3. Teknologisk frihet.
4. Felisolering.
5. Oberoende distribution och utveckling.

## Nackdelar

1. Ökad komplexitet.
2. Utmaningar med övervakning och integration.
3. Svårare att hantera kommunikation mellan tjänster.
4. Kräver en mogen DevOps-process.



# Kommunikationsytor





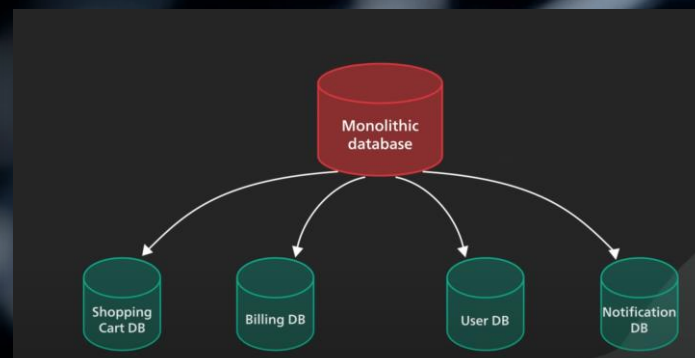
# Jämförelse mellan arkitekturer

## Monolitisk arkitektur

Hög kopplingsgrad och beroenden.

Mycket begränsad skalbarhet.

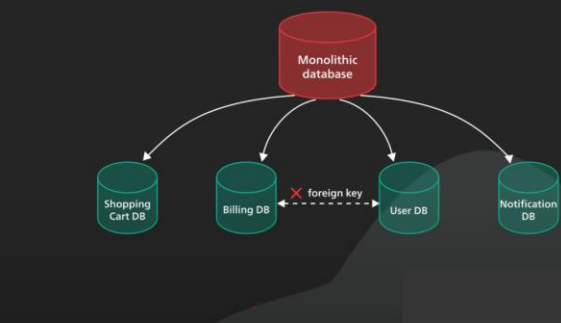
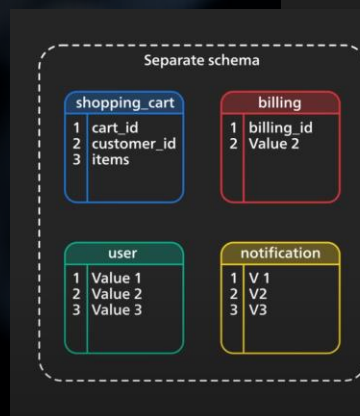
Hög prestanda för isolerade tjänster.



## SOA

Modularitet och återanvändning.

Högre autonomi än monoliter.



## Mikrotjänster

Låg kopplingsgrad.

Optimal horisontell skalbarhet.



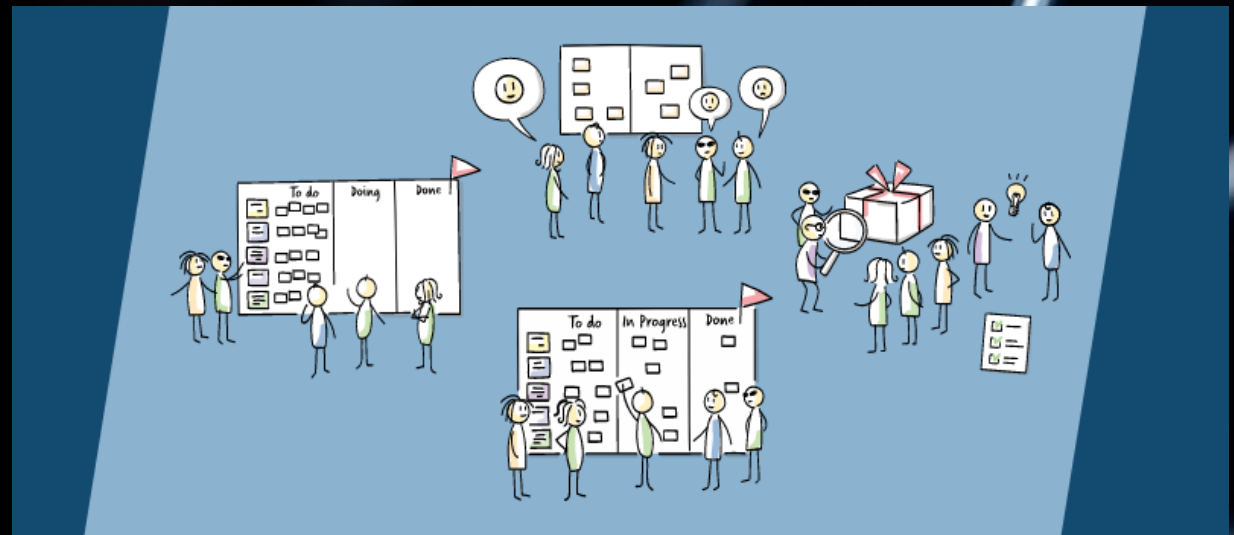


# Jämförelse mellan arkitekturer

Egenskap	Monolith	SOA	Microservices
-----	-----	-----	-----
<b>Skalbarhet</b>	Begränsad	Medel	Hög
<b>Flexibilitet</b>	Låg	Medel	Hög
<b>Prestanda</b>	Hög	Beror på ESB	Kan påverkas av nätverk
<b>Komplexitet</b>	Låg	Medel	Hög
<b>Kostnad</b>	Låg till med	Hög (initialt)	Hög

# Strategier för transformation

1. Identifiera funktionalitet med svag koppling och bryt ut som mikrotjänster.
2. Använd inkrementella och iterativa metoder för transformation.
3. Designa API-kontrakt för kommunikation mellan tjänster.
4. Använd klisterkod "glue code" för att integrera gamla och nya system.





# Skalbarhet | Konsekvenser

## **Monolitisk arkitektur**

1. Virtuellt skalning begränsad till hela applikationen > Kräver kraftfullare servrar.
2. Begränsad horisontell skalning (hela applikationen måste skalas).
3. Hög prestanda för mindre applikationer.
4. Svårt att förändra eller underhålla vid växande system.
5. Uppdateringar innebär att hela systemet påverkas.

## **Mikrotjänstarkitektur**

1. Horisontell skalning möjlig med Docker och lastbalansering.
2. Lätt att skala specifika tjänster horisontellt.
3. Möjlighet att använda olika teknologier för olika tjänster.
4. Bättre anpassning för stora och komplexa system.
5. Hög flexibilitet och möjlighet till oberoende uppdateringar.
6. Kräver mer resurser för hantering av kommunikation, övervakning och felhantering.



# Hur väljer man rätt arkitektur?

## **Monolith**

1. Passar för mindre och enklare projekt med låg förändringsfrekvens.
2. Snabb MVP (Minimum Viable Product).
3. Lägre budget för drift och utveckling.

## **SOA**

1. Stora organisationer med flera system.
2. Integration över flera plattformar.

## **Microservices**

1. Höga krav på skalbarhet och flexibilitet.
2. Perfekt för stora, komplexa och dynamiska system.
3. Möjlighet att använda olika teknologier.





# Sammanfattning

Arkitekturer utvecklas för att möta förändrade behov:

1. Monolith: Enkelhet och stabilitet.
2. SOA: Integration och återanvändbarhet.
3. Microservices: Skalbarhet och flexibilitet.
4. Framåt > Serverless och event-driven arkitekturer blir alltmer vanliga och enklare att använda.