

## Utmaning – Integration mellan molnet och din interna IT

Sven-Håkan Olsson, Definitivus



## Hur du väljer stil för integrering av moln-applikationer med egna applikationer

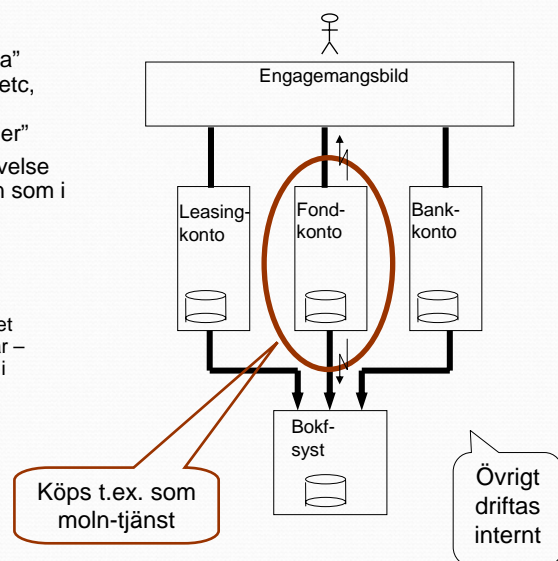
- Online-SOA
- Händelsestyrd SOA
- Replikering
- ...något annat?
  
- One size does NOT fit all!

## Partiell outsourcing (moln etc)

- Partiell outsourcing (eller multisourccing), såsom:
  - Cloud Computing (moln)
  - SaaS (Software as a Service)
  - ASP (Applications Service Provider)
- Kräver nästan alltid integration med
  - Interna applikationer och register
  - Andra outsourcade applikationer och register
- ...för man kan ju inte offra all användarvänlighet på Molnets altare – användarna ska inte behöva hålla ordning på tre separata kundregister till, eller så!

## ETT exempel med moln i kombination med interndrift

- I exemplet tar jag inte "enkla" tjänster som Google maps etc, utan tunga verksamhets-applikationer, "SOA-domäner"
- Användarens helhetsupplevelse ligger i engagemangsbilden som i sin tur beror av tre andra komponenter
- Vanligen helt olika driftsstabilitet hos interna och externa sladdar – inkopplingen till Fondsystemet i exemplet är extern, kanske via Internet!



## Naiv ihopkoppling

- Gör vi som vi gjorde förr när vi skapade monolitiska applikationer i en och samma driftmiljö:  
Synkron online-kommunikation ?!
- + Enkelt att förstå
- + Lätt att programmera
- + Alltid färskt data
- - Allvarlig prestandarisk
- - Allvarlig skalbarhetsrisk
- - Allvarlig tillförlitlighetsrisk
- Duger troligen inte!

## Datafärskhet – några verksamhets-scenarios och optimeringar

- En prisfråga mot en leverantör
  - Frågan körs mot en lokal (replikerad) kopia av prisregistret (10 s – några timmars gammalt data)
- En lagersaldofråga mot en leverantör
  - Frågan körs online mot leverantören (1-2 s)
- En faktura går till en kund
  - Fakturan överförs via batch (eller kö) inom en dag till kunden

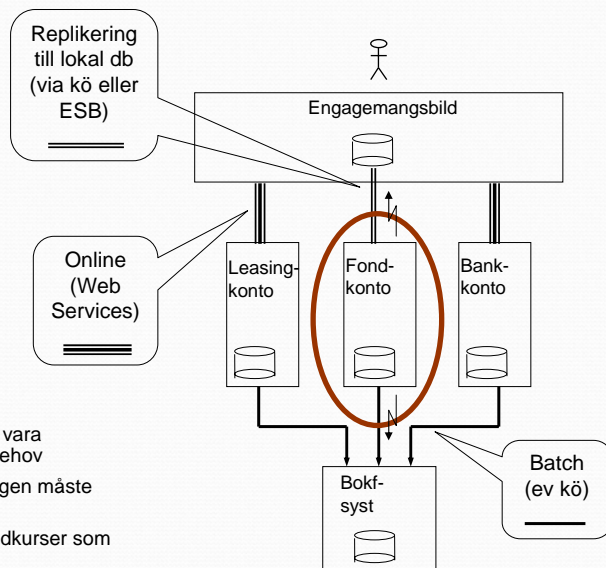
Verksamhetsbehoven ska styra "temporal kvalitet" (dvs färskhet).  
Styr i sin tur optimerad teknislösning.  
Bör modelleras vid grundläggande info-modellering!

## Några stilar för infoutväxling (ibland del av Master Data Management)

- **Replikering**
  - Relationsdatabasers inbyggda replikering
  - EDA, händelseorienterat via enkel kö, ESB eller EAI
- **Online**, synkront
  - Teknik-inlåst online
    - DCOM, Dotnet remoting, Java RMI, LU6.2 ...
  - Interoperabel online
    - Web Services SOAP, dominerar
    - REST, ökar (ger också lättare möjlighet till cache för hybridlösning)
- **Batch**
  - Sekvensiella filer via ftp, http e.dyl.
  - Via enkel kö

ESB (Enterprise Service Bus), EAI (Enterprise Application Integration), EDA (Event Driven Architecture)

## ETT exempel på optimering av färskhet



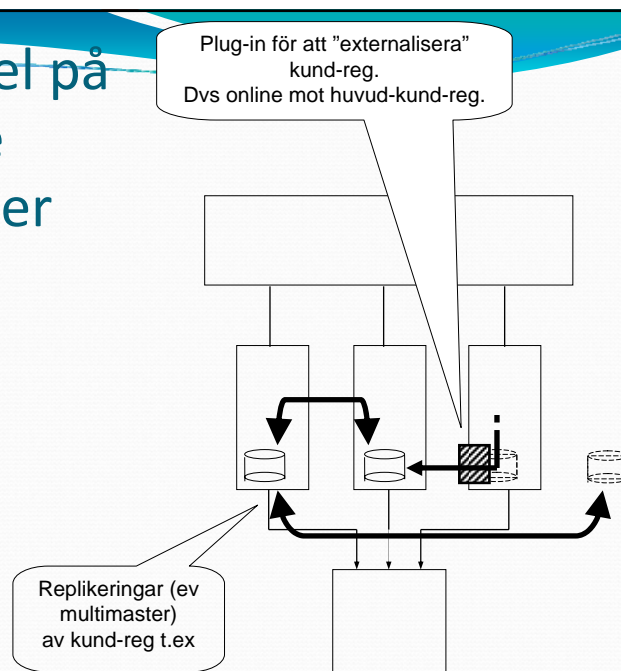
- Så här tänks tekniklösningarna vara optimerade efter verksamhetsbehov
- Online bara där det oundgängligen måste vara det
- Replikering för t.ex. sådana fondkurser som endast uppdateras dagligen
- Bokföringstransar är rättså tidsokänsliga

## Om jag nu vill köpa några nya kontosystem/tjänster?

- Är molntjänsterna du överväger (eller egendriftade applikationer) tillräckligt integrerbara?
  - Har inte vareviga av dem ett eget kundregister t.ex?
  - Har inte vareviga av dem en egen inloggningskatalog?
  - etc etc
- Antingen måste det gå att plugga in online-kommunikation till ditt favoriserade kundregister (men online kan ha nackdelar: prestanda/skalbarhet/tillförlitlighet)
- Eller också måste du kunna replikera data mellan dina olika kundregister (men replikering kan ha nackdelar: icke-färsk info, replikeringskrokar, buggig multimasterreplikering)
- Viktigt vara medveten och lägga detta i vågskålen vid beslut!
- Viktigt ställa tydliga krav på sådan integrerbarhet hos system/tjänst!
- Kolla respektive informationsmodell, är de kompatibla, går det att översätta? Är kund = kund?

## ETT exempel på dubblerade kund-register

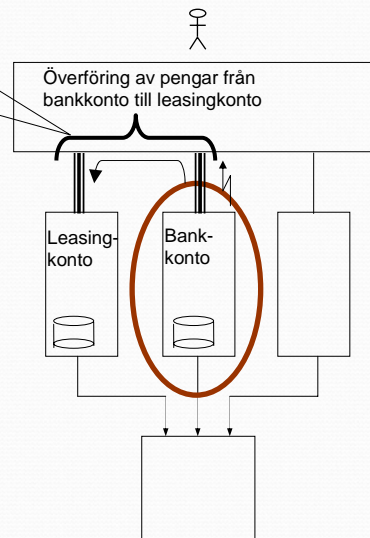
- Båda lösningarna har sina utmaningar
- Ambitiös SOA-domän-design kan göra att plug-in-varianten kan underlättas
- Replikering kan vara snabbt händelseorienterad eller utföras endast periodvis



## Allt-eller-ingen-uppdatering

Skulle behöva commit/rollback, annars kan pengar verkligen försvinna vid systemnedgång mm!

- Kallas även ACID, commit/rollback, atomär transaktion, teknisk transaktion, unit-of-work...
- Självklart för lokala relationsdatabas-uppdateringar men inte för systemintegration, såsom i molnet!
- ACID är en mardröm över långa avstånd, med flera parter, i olika teknikmiljöer
- ACID bryter mot SOA-guidelines som statelessness och loose coupling
- Vanliga Web Services har inte ACID
- Den komplexa standarden WS-Transaction är tveksam



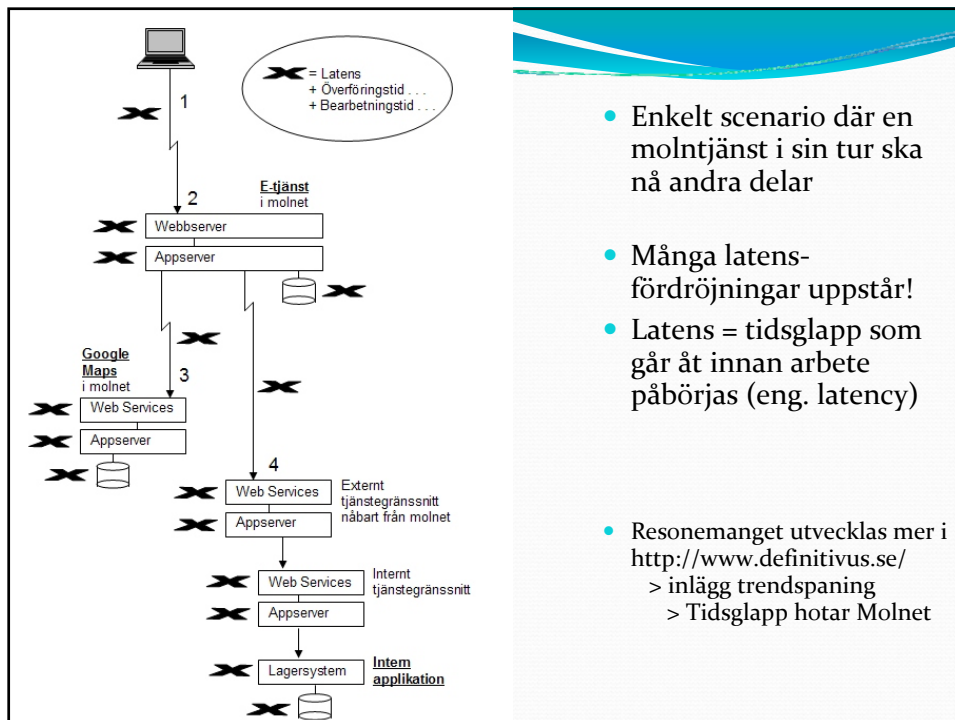
ACID (Atomicity, Consistency, Independency, Durability)

## Hur gör man då, utan ACID?

- Finns ett antal tekniska knep
- Men det som det talas mycket om är långa verksamhets-transaktioner (eng. long running transactions) som kan pågå flera dagar:
  - Hitta på sätt att upptäcka att uppdateringsfel skett (avstämningar etc)
  - Fixa felet med med s.k. kompensering.
    - T.ex. skapa en ny bokföringstransaktion med omvänt belopp som tar ut den felaktiga
  - Ofta manuell rutin. Ibland maskinell.
- OBS! Att datat är fel under tiden, fram till felupptäckt och kompensering!
- ACID-problemet är en allvarlig nackdel med SOA och med mången moln-kommunikation, men det finns så många stora fördelar också!

# Leveransgaranti mm

- Har de asynkrona lösningarna (kö, ESB, EAI, EDA etc):
  - Garanti för att meddelandena verkligen kommer fram?
  - En och endast en gång, eller kan det bli dubletter (behovet av s.k. idempotency, dubletteliminering)?
  - Kommer de fram i avsändningsordning?
  - Säkras kön persistent eller ligger den bara i primärminne som försvinner vid systemnedgång (t.ex kräver standarden WS-ReliableMessaging inte persistent kö)?
  - ACID tillsammans med lokala databasuppdateringar?
- Gör noggrann utvärdering!
- OBS! Vanliga Web Services med SOAP är inte i sig asynkrona, har ingen kö inbyggt, sådan måste tillföras.



- Enkelt scenario där en molntjänst i sin tur ska nå andra delar
- Många latensfördröjningar uppstår!
- Latens = tidslapp som går åt innan arbete påbörjas (eng. latency)
- Resonemanget utvecklas mer i <http://www.definitivus.se/>
  - > inlägg trendspaning
  - > Tidslapp hotar Molnet

## Trendspaningar, artiklar

- Kolla gärna in nya [www.trendspaning.se](http://www.trendspaning.se) där jag regelbundet deltar
- På min enkla sajt [www.definitivus.se](http://www.definitivus.se) finns också ett artikelarkiv på undersidan för trendspaning som successivt fylls på
- Där finns också material om mitt föredrag om Composability på SOA Symposium i Amsterdam hösten 2008
- Samt andra föredrag i olika sammanhang, inklusive detta.

## **Sven-Håkan Olsson**, oberoende IT-konsult.

- Erfarenhet sen 70-talet. Alltid intresserad av att förstå nya IT-företeelser och se ifall de kan ge verksamhetsnytta.
- Kursledare inom Dataföreningen, bl.a. för Certifierad IT-arkitekt. Medgrundare av Know IT.
- Gör gärna: IT-arkitektuppdrag, reviewer av applikationer – arkitektur - projekt, systemutveckling, kursledning, är föredragshållare.

*sven-hakan.olsson[hos]definitivus.se,  
www.definitivus.se 0708-84 01 34*