Planering av skalbara mjukvarusystem

Otto Söderholm

Kandidatavhandling i datateknik

Handledare: Ivan Porres

Institutionen för IT

Åbo Akademi

2019

ÅBO AKADEMI – FAKULTETEN FÖR NATURVETENSKAPER OCH TEKNIK

Abstrakt för kandidatavhandling

Ämne: Datateknik	
Författare: Otto Söderholm	
Arbetets titel: Planering av skalbara mjukvarusystem	
Handledare: Ivan Porres	
Abstrakt: Det blir allt vanligare med globalt tillgängliga tjänster där antalet användare kan variera och systemen som byggs måste kunna skala enligt det. Det är viktigt att förstå sig på den underliggande tekniken för att klara av att hålla systemet säkert och effektivt. Arbetet behandlar vanliga utmaningar som uppstår då man planerar ett skalbart system. Till stor del behandlas utmaningar som uppstår i distribuerade system samt vanliga sätt att upprätthålla prestandan. Definitionen på skalbarhet behandlas också och analyser med tanke på dagens växande nättjänster. Huvudsakligen handlar arbetet om skalbar mjukvara och utmaningarna och lösningarna som behandlas är därmed huvudsakligen mjukvarurelaterade. För att få en bättre bild av helheten diskuteras också skalbar hårdvara, dock kort och ytligt. I slutsatsen analyseras det goda och onda med skalbara och distribuerade system och vi tittar även på några vanliga verktyg som kan användas för att underlätta planeringen och distribueringen.	
Nyckelord: Skalbarhet, mjukvara, molntjänst, säkerhet, distribuering, replikering	
Datum: 3.4.2019	Sidoantal: 20
Abstraktet godkänt som mognadsprov:	

Innehåll

1. Inledning

Då kvalitén av och tillgången till olika tjänster ökar, måste man sträva efter system med allt bättre tillgänglighet. Tjänsterna förväntas vara tillgängliga så gott som hela tiden och med så lite väntande som möjligt för användaren. Det att tjänsten fungerar pålitligt har naturligtvis alltid varit en viktig sak, men med den ökande hastigheten av internet och ibruktagandet av fler och fler molntjänster, förväntas företagen leverera en produkt som alltid fungerar.

Mindre företag har inte alltid råd att investera i enorma datacenter för att kunna säkerställa tjänstens tillgänglighet vid maximal användning. Som lösning erbjuds idag molntjänster och användning av olika verktyg för att skala upp eller ner tjänsten beroende på behov. För att kunna göra detta, måste systemet planeras med skalbarhet i åtanke.

Syftet med denna undersökning är att finna de vanligaste utmaningarna med distribuerade och skalbara system och analysera potentiella lösningar till dessa utmaningar.

1.1. Motivering

Vi startade nyligen ett företag och förväntar oss en stor mängd användare på global nivå. För att bättre kunna planera ett system som inte kostar för mycket i början att upprätthålla men samtidigt är billig att skala upp efter behov, måste vi studera olika tekniker inom skalbar mjukvarudesign.

Många saker är självklara för sådana som tidigare skapat mer komplexa mjukvaruhelheter, men varje område inom ämnet blir snabbt komplicerat då man börjar

pussla ihop de olika delarna. Det går lätt att glömma delar av helheten då man blir och funderar på något specifikt. Många skrivelser på internet innehåller praktiska lösningar på problem och läsaren behöver inte alltid alls förstå hur lösningen fungerar. Då finns risken att lösningen som väljs inte ändå är den bästa möjliga.

1.2. Termer

Med *molnet* menas stora mängder servrar som är sammankopplade via internet och med *molntjänster* menas tjänster som databaser, datalagring, servrar för program, lastbalansering och andra relaterade tjänster som erbjuds av företag eller skapas själv. I dagens läge finns det stora företag som alla erbjuder relativt likadana tjänster då det kommer till molnet. Till de största hör Google, Amazon och Microsoft. De fyra vanligaste formerna av molntjänster är *PaaS* (plattform som nättjänst), *IaaS* (infrastruktur som nättjänst), *SaaS* (programvara som nättjänst) och *DaaS* (lagring som nättjänst). [1]

Serverlösa miljöer har blivit populära under senaste tiden. Ordet är en aning missledande, eftersom det definitivt finns en server som snurrar hela systemet. Tanken med serverlösa miljöer är att utvecklaren inte ska behöva fundera på den underliggande infrastrukturen, utan den sköts av serverlösa miljöns leverantör, molnoperatorn.

Noder är maskiner i ett system. Noderna kan vara fysiska maskiner eller virtuella instanser. Stora sammankopplade helheter som består av en eller flera noder kallas ofta för *kluster*. Man strävar till att hålla noderna på samma område för att reducera avstånden inom klustret, medan klustren ofta är distribuerade.

2. Vad är skalbarhet?

Ett system anses vara skalbart om det fortfarande är effektivt då det sker en signifikant ökning av resurser och användare. [2] Det ska inte heller bli mycket svårare att administrera systemet fast det växer.

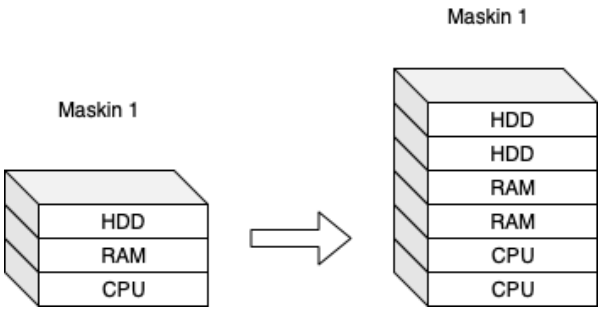
Skalbarhet gäller inte endast programvara, utan även andra tjänster och processer kan vara skalbara. Fabrikernas produktionslinjer brukar planeras skalbart för att minimera tiden som går åt till annat än produktion av varan. I fabriker syns skalbarheten av produktionslinjen direkt i inkomsterna. Man strävar till en balans mellan produktionstid, arbetskraft och efterfrågan för att maximera vinsten.

Om man förväntar sig att systemet kommer växa och det ska vara skalbart, måste systemet planeras och byggas med tanke på det. Om systemet inte klarar av att skala sig enligt kraven, kommer det att resultera i mycket arbete som kunde ha undvikits med förbyggande planering.

Då systemet växer, växer ofta även mängden data. Om systemet inte planerats skalbart, kan oförväntade problem uppstå. Mängden data kan också bli svår för användaren att hantera. Som exempel kan ett problem vara att användarna inte mera hittar material lika lätt som tidigare, vilket gör användningen av tjänsten långsammare.

2.1.1. Vertikal skalbarhet

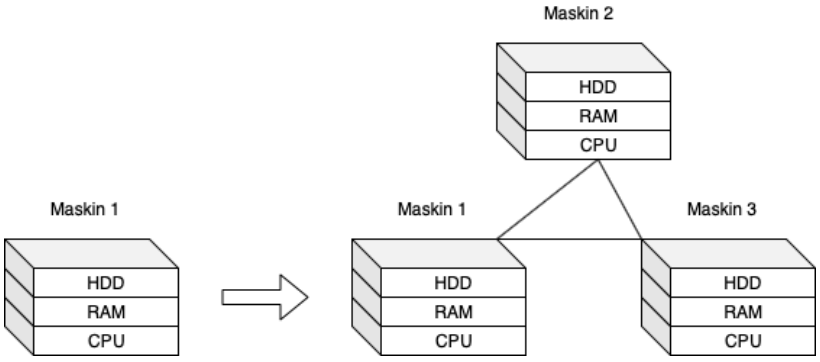
Då man skalar vertikalt, innebär det att man tillsätter resurser till maskinen. Dessa resurser är bland annat RAM (random access memory), processorkärnor eller lagringsutrymme, enligt Figur 1. Man har fortfarande samma mängd maskiner, men de blir snabbare och effektivare.



Figur 1: Vertikal skalning.

2.1.2. Horisontell skalbarhet

Till skillnad från vertikal skalning, där maskinens resurser ökar, ökar istället antalet maskiner i horisontell skalning, enligt Figur 2. Medan det är fysiskt lättare att starta upp flera maskiner, blir mjukvaruplaneringen mer krävande. Även vissa delar av infrastrukturen kan bli svårare planera in i helheten. Exempelvis lagringsutrymme som måste vara tillgänglig för mera än en maskin i taget, kan skapa problem. Dessa är likadana problem som i distribuerade system i allmänhet och kommer därmed att behandlas i detta arbete.



Figur 2: Horisontell skalning.

3. Begränsande faktorer

Då man skapar ett skalbart system, måste man ta i beaktande de faktorer som begränsar skalbarheten och effektiviteten. Dessa faktorer beror på hurdan skalbarhet det handlar

om, men de flesta problemen är saker som finns i de flesta mjukvarusystemen som förväntas bli stora.

Faktorer som begränsar växandet och skalbarheten kan vara fysiska, logiska eller administrativa. Fysiska begränsningar är hårdvaru- eller utrymmesrelaterade och innebär att systemet inte kan växa på grund av begränsningar som man kan se och röra. Till fysiska begränsningar hör också maskinernas hårdvarurelaterade begränsningar, som att det helt enkelt inte finns plats för mera minne.

De logiska begränsningarna är till största delen programvarurelaterade faktorer. Dessa problem är svåra att lösa, eftersom de ofta innebär förändringar i stora delar av systemet. Som ett exempel kunde man ta ett ID-nummer i en databas, som man i hela resten av systemet har räknat med att är en 32 bitars heltal. Detta leder till att systemet inte kan växa till att ha ID-nummer som är större än 32 bitars heltal, utan att ändra alla delar av systemet, där talet har förväntats vara maximalt 32 bitar. Andra mjukvarubaserade faktorer kan vara ett resultat av dålig planering av systemet, som att delar av systemet blir överbelastade eller inte kan nås av alla som behöver dem.

Administrativa begränsningar innebär ökade svårigheter i hantering av systemet. Detta kan till exempel bero på diversifiering av hårdvara i sådan utsträckning, att systemet inte mera fungerar som förväntat överallt och därmed måste administreras mera. Administrativa problem kan leda till mera arbete på organisationsnivå, ifall man måste anställa mera personal och därmed omstrukturera organisationen.

Det är viktigt att adressera alla möjliga problem innan de uppstår. Då ett system väl är i användning och har vuxit, är det svårt att ändra på. Om systemets ursprungliga delar måste ändras, måste man även beakta eventuella delar som fortfarande är beroende av den gamla logiken.

4. Skalbar mjukvara

Med dagens orkestreringsvärktyg och kontinuerliga implementeringsvärktyg är det lätt att glömma bort de underläggande teknikerna som är i användning. Stora internetföretag har använt mycket resurser för att försöka skapa enkla men robusta program för att hjälpa till med det växande behovet av processeringskraft. Många molntjänst leverantörer använder specialiserad programvara för att kombinera datorer runt om på världen och får det att verka som om allting vore på en och samma maskin.

Alltid fungerar inte dessa allmänt fungerande verktygen, utan ett behov för att förbättra de existerande verktygen kan uppstå eller så kan ens system kräva helt skräddarsydda lösningar. I dagens läge finns det programvara för nästan vilket behov som helst, men till exempel ny hårdvara kan kräva grundlig undersökning och optimering och då kan det vara lättare att göra allting själv.

Lyckligtvis finns det även en hel mängd undersökning och teori om hur man ska gå till väga.

4.1. Distribuering

För att ett system ska kunna växa, måste man se till att alla delar hänger med i tillväxten. Ibland kan distribuering hjälpa till att fördela på arbetsmängden. Istället för att köra allting på en stor maskin, kan man ha delar av projektet på flera, mindre maskiner. Alternativt kan man ha samma process på olika maskiner med hjälp av replikering.

En annan fördel med distribuering är att man med hjälp av replikering kan sprida ut processerna till andra länder. Då systemet och antalet användare växer kan man skära ner tiden det tar för en begäran att bli besvarad, genom att rikta begäran till närmaste maskin där processen körs. Ifall samma innehåll inte måste nås från alla länder, behöver man inte replikera data eller vänta på att den hämtas från ett centralt ställe. Detta är en

vanlig teknik i stora, globala tjänster med stora mängder data, så som videoströmnings tjänster. [3]

Inom distribuerade system uppstår dock andra problem. En del av problemen kan även uppstå i centraliserade system, men förekommer med större sannolikhet då programvaran blir uppspjälkat i olika delar. Delat minne är en av de vanligaste problemkällorna, eftersom minnet inte mera hanteras av endast ett operativsystem.

4.2. Replikering

Replikering innebär att man skapar flera instanser av samma process för att fördela arbetsmängden. Detta går ofta ihop med distribuering av systemet, eftersom det tillåter och ofta kräver startandet av flera likadana tjänster. Det kan till exempel vara effektivare att ha flera olika globalt utspridda servrar och låta användarna kontakta den närmaste. [4]

Replikering är effektivt då det handlar om processer som kan jobba individuellt, men det blir svårare då processerna måste dela data. Data måste kommas åt från alla processer, men om den är lagrad på endast ett ställe, kommer belastningen på datalagret att öka. En lösning till detta är replikering av data. Då samma data finns lokalt i alla processer, kommer man snabbt åt det och systemet hålls effektivt.

För att replikera data används ofta olika bakgrundsprocesser som kallas för replikeringshanterare. Dessa granskar ifall data har ändrat med jämna mellanrum eller då det behövs. Det innebär att data inte nödvändigtvis är korrekt då man begär den, eftersom den kan ha ändrat i ett annan process och inte ännu uppdaterats i alla andra processer. För att minska på väntandet kan man ha bakgrundsprocesser som påbörjar uppdateringen då de får en signal från process om att data ändrats.

Läsandet av data är lätt, men då flera system kan skriva samtidigt, uppstår andra problem som att vilken uppdatering är den gällande. Om det bara finns ett värde som

gäller och det värdet inte bygger på det tidigare, kan man anta att den senaste uppdateringen är den gällande. Om det nya värdet är beroende av det gamla, kan det bli bortfall då flera processer försöker skriva och läsa värdet. För att dela upp arbetet kan man använda sig av sharding, som beskrivs mera under databaser under nästa rubrik.

4.3. Datalagring i skalbara system

Datalagring är en viktig del av de flesta system och måste därför planeras noga för att förebygga eventuella framtida problem. Lagringsystemen hör till de mest belastade delarna av distribuerade system. [2] Medan en del data kan kastas efter att den använts, finns det också krav för beständig lagring. Med beständig lagring menas lagring som håller sitt innehåll även efter användning och som kan hämtas senare på nytt.

Dessa krav varierar beroende på systemet och därför finns det flera alternativ för lagringstekniker i distribuerade miljöer. Beroende på exempelvis kraven på versionshantering och datas storlek, finns det olika lösningar på lagringsbehoven.

4.3.1. Namngivning och struktur

Planeringen av skalbara namn på objekt, klasser och filer sträcker sig genom hela systemet. Objekt och funktioner inom programvaran måste vara beskrivande, men får inte bli besvärliga att använda. Struktureringen kan hjälpa till med förståendet av programvarans uppbyggnad. Olika system kräver olika typers strukturer.

Många programmeringsspråk har vägledande bestämmelser gällande namngivning och det lönar sig ofta att följa dem. Exempelvis programmeringsspråket Go (eller Golang), som har relativt starka instruktioner gällande formatering av kod, bygger långt på att namngivningen av paket är den samma som namnet på katalogen eller mappen där de

olika funktionerna finns. [5] På detta sätt kan delar av funktions- och variabelnamn lämnas bort, eftersom de redan finns i paketets namn.

Datalagring kan också följa kategorisering med hjälp av kataloger och underkataloger. Sökning av data hålls lätt, så länge man vet vilken sorts data man är ute efter, då allting är kategoriserat. Det blir dock svårare att hitta rätt om man inte vet var man bör söka eller om datamängden helt enkelt är så stor att kategoriseringen inte mera hjälper. En lösning kunde vara att alltid använda ett unikt namn för varje entitet.

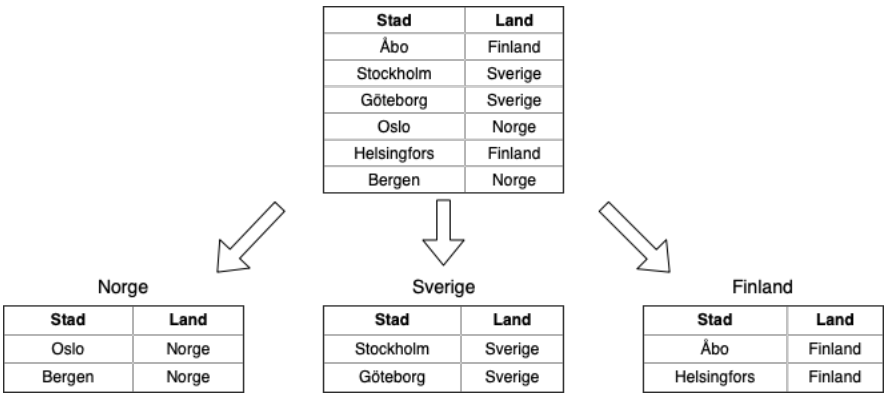
Unika namn eller ID:n måste kunna växa med systemet. Genom att strukturera data rätt, kan man ha flera mera med samma ID eftersom de är lagrade på olika ställen. I stora system finns det en risk att systemet inte klarar av så stora tal eller så långa strängar. Nyare system som stöder 64-bitars tal är inte ännu i samma riskzon eftersom de klarar av exponentiellt större tal än 32-bitars system.

4.3.2. Databaser

Databaser är bra för att hålla reda på data och för att organisera data i rätta kategorier. I en databas kan man inte lagra alla former av data, men de kan istället användas för att lagra adressen för mer komplexa data, som bilder och andra filer. Då man planerar tabellerna i databasen, bör det göras så att minimala förändringar och migrationer krävs efter att systemet redan är i användning. Förändringar i modellerna kan kräva strukturella förändringar i databasen och kan leda till onödiga fält eller till att information försvinner.

Normalt lagrar databaser sin data på ett ställe. Denna data måste alltså antingen replikeras till olika instanser av systemet eller så måste man ha ett tillräckligt effektivt system för databasen för att prestandan ska hållas med i tillväxten av resten av systemet. Ett sätt att minska tiden det tar att söka igenom databasen är *sharding*. Sharding innebär att man delar upp databasen i mindre *shards* (*skärvor*) så att varje shard innehåller hela rader, se Figur 3. Systemet där databasen körs kan på detta sätt

skalas horisontellt istället för vertikalt. Varje shard har en process som hanterar den lokala databasen, vilket tillåter flera simultana sökningar.

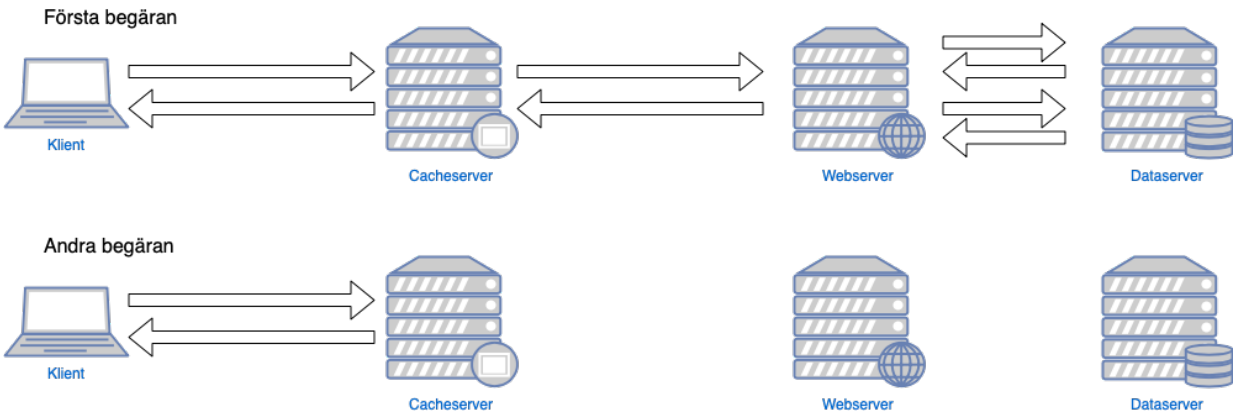


Figur 3: Exempel på sharding av en databas enligt land.

Sharding är effektivare i större system och kan istället vara till mera besvär än nytta i mindre helheter eftersom varje shard egentligen är en egen databas. I stora system har shards den fördelen att de kan distribueras till flera maskiner. Dessutom minskar risken för att hela databasen blir oåtkomlig, försvinner eller är tillgänglig för obehörig tillgång. Sharding används inte bara för databaser, utan kan möjliggöra flyttandet av annan data också så att den är närmare dem som ofta använder sig av den.

4.3.3. Cache

Cacheminne används för att reducera tiden det tar att hitta data. Meningen är att hålla reda på data som man tror att kommer behövas snart igen, det vill säga data som ofta används. En av de vanligaste teknikerna för distribuerade system är att ha en skild server för cache. Bland med de vanligare systemen i användning är Redis, en cache- och databas-server som är baserad på öppen källkod.



Figur 4: Cacheserver som serverar tidigare begärda data.

De flesta cacheservrar fungerar som ett av de första ställen där klientens begäran behandlas. Ifall en motsvarande begäran gjorts tidigare och data som returnerats är möjlig att lagra i cacheminne, behöver begäran inte gå vidare från cacheservern, utan rätt svar finns redan där och kan returneras till klienten. Detta demonstreras simpelt i Figur 4. Statiska data är lättare att cacha eftersom det inte är lika användarspecifikt; t.ex. HTML, CSS, bilder och andra statiska filer.

Om cache används på rätt sätt, behöver inte resten av systemet lika mycket resurser och detta höjer i sin tur tröskeln för behovet att skala upp systemet.

4.4. Säkerhet

Systemets säkerhet är en essentiell del av helheten. Då systemet växer ökar även säkerhetsriskerna eftersom det finns flera mellanrum för attackeraren att tränga sig in i. [6] Ifall systemet blir distribuerat, måste data som rör sig mellan noderna och mellan processer hållas krypterad, annars ökar risken att bli utsatt för en MITM (man-in-the-middle) attack.

Det anses finnas fyra olika typers säkerhetsrisk i mjukvara: avlyssning, avbrytning, modifiering och fabrikation. [7] Avlyssning innebär att obehöriga parter kan ta upp information som rör sig antingen inom systemet eller ut och in från systemet. Även om

dessa två avlyssningsmetoder ofta fungerar olika, kan båda förhindras eller åtminstone försvåras genom att kryptera trafik och data.

Med avbrytning menas i allmänhet DoS (Denial-of-Service) attacker av olika slag, då attackerare förhindrar systemet att fungera på förväntat sätt. Ett system kan bli förhindrat både inifrån och utifrån. Attacker från insidan utförs ofta på operativsystemsnivå eller genom att utnyttja fel i programvara. För att hindra obehöriga personer från att komma in i systemet kan man använda sig av autentiseringsmetoder och olika form av behörighetskontrol. Attacker från utsidan är mycket vanliga men kräver ofta mera resurser. Distribuerade DOS-attacker (DDOS) utförs genom att med ett stort antal maskiner utföra begäran mot systemet. Detta resulterar i att systemet måste allokera resurser till behandlandet av dessa begäran, vilket minskar antalet resurser systemet har för riktiga användare. Om resurserna tar slut, kan systemet krascha. Genom att förhindra klienter av att nå det egentliga systemet, kan attackerare även skapa egna icke legitima kopior av systemet och därmed avlyssna klienters information.

Fabrikation och modifiering av data kan skapa många problem. En obehörig part kan till exempel skapa eller ändra kontoinformation för att få åtkomst till systemet. Vanligtvis beror dessa attacker på dålig behörighetskontroll, men kan även utföras på grund av buggar i källkoden. Ibland är skillnaden mellan missar i behörighetskontroll och buggar liten och ibland är de beroende av varandra. Injektioner av olika slag är de vanligaste sårbarheterna i dagens webapplikationer. [8] Detta innebär att man injekterar data som sedan exekveras på tjänsten. På detta sätt kan en attackerare åstadkomma mycket skada eftersom injekterade koden ofta körs med samma rättigheter som systemet i sig.

4.4.1. Kryptering

En essentiell metod för att motverka de möjliga säkerhetsproblemen är bland annat kryptering. Kryptering är ett steg mot ett säkrare system som bör tas direkt från början. Utmaningen är att hitta en krypteringsmetod som både är tillräckligt säker och effektiv för att fungera i det långa loppet. När datorer blir effektivare, minskar också tiden för

brytandet av krypteringsalgoritmer. I februari 2017 meddelade Google Research och CWI Amsterdam att de lyckats skapa en *hash*-kollision med hjälp av den populära och tidigare som säker ansedda SHA-1 algoritmen. [9] Även om metoden i fråga skulle kräva ett år av uträkning ifall man hade 110 grafikkort till sitt förfogande, anses algoritmen vara ändå vara osäker och det rekommenderas därför att man byter till en säkrare krypteringsalgoritm.

Eftersom de flesta krypteringsalgoritmer producerar hashar är så kallade *one way hashes*, man kan inte dekryptera dem, kan man inte bara övergå till en starkare kryptering ifall det finns existerande, krypterade data. Om det till exempel handlar om användares lösenord, måste man vänta på att användaren loggar in för att man skall kunna skapa en ny hash av lösenordet då det är i klartext.

4.4.2. Autentisering och behörighetskontroll

Det är viktigt att kunna begränsa åtkomsten till data och att kunna övervaka vem som har behandlat data och hur. Olika metoder existerar för att begränsa åtkomst till olika delar av systemet. Ett vanligt och relativt enkelt sätt att skydda systemet från utomstående är att använda en brandvägg. Brandväggen begränsar trafiken på basis av adress, port eller gränssnitt. Även om säkerheten är viktig och bör vara en central del av systemet, kan det bli just säkerheten som begränsar systemets skalbarhet. För att minska belastningen på ett ställe, kan man distribuera slutpunkterna och tillåta flera simultana autentiseringar. Om man distribuerar slutpunkterna måste man se till att varje slutpunkt har samma, uppdaterade klientinformation, som i sin tur leder till att exekveringstiden blir längre.

Olika former av behörighetskontroll (eng. *access control*) existerar och klassas enligt vilka värden åtkomst begränsas efter. Några exempel på olika former är *roll baserad behörighetskontroll* (RBAC), *attribut baserad behörighetskontroll* (ABAC), *kontext behörighetskontroll* (CBAC) och *åtkomstlistor* (eng. *Access Control List*, kort. ACL). Dessa varierar i både användningsändamål och hur rättigheterna ges.

RBAC, som är en av de vanligare behörighetskontrollmodellerna, fungerar genom att användarna har roller, och rollerna har rättigheter. En roll kan ha flera rättigheter och en flera användare kan höra till en roll. Det har saknats officiell definition på RBAC och därför implementerar många nästan omedvetet någon form av behörighetsmodellen. [10]

4.4.3. Programvaruberoenden

Många programvaruramverk innehåller idag metoder för att lätt utveckla och använda olika moduler för att man inte alltid skulle måsta bygga saker från grunden upp. Detta har lett till en ökning av beroenden i programvaran, vilket har både för- och nackdelar. Ju flera beroenden programvaran har, desto större risk är det för att något av paketen som programvaran är beroende av, har någon slags sårbarhet. Å andra sidan gör ramverkens verktyg det ofta lättare att märka sårbarheterna och uppdatera beroenden.

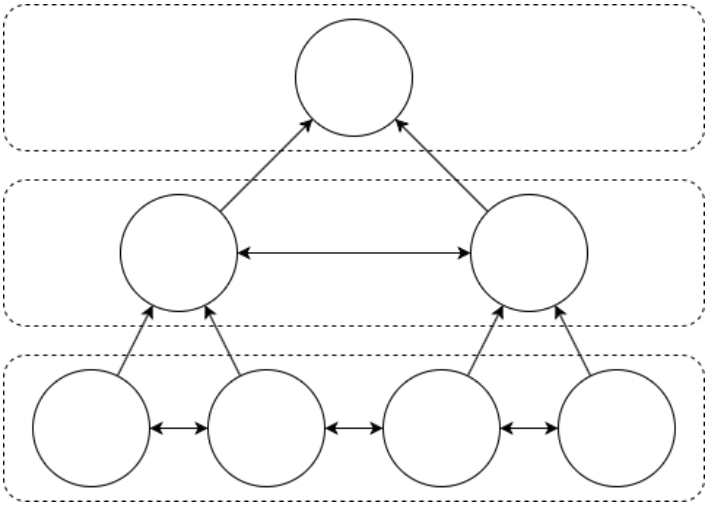
Många moderna ramverk har dessutom flera moduler som är baserade på öppen källkod. Öppen källkod gör det lättare för vem som helst att bidra till projekten. På detta sätt finns det flera ögonpar som kan lägga märke till eventuella buggar och sårbarheter. Detta möjliggör öppenhet och ansvarighet, som är en viktig faktor i ett säkert system. Alla projekt är dock inte säkra fastän de är öppen källkod. Ibland måste man betala för programvara och då förväntas det ofta att programvaran upprätthålls av ett företag.

4.5. Administrering

Som tidigare nämnt, måste administreringen av systemet hållas hanterlig fastän systemet växer. Administrering innefattar alla de tidigare nämnda områdena och är därför viktig att hålla smidig. Ett system som är klumpigt att administrera leder till ökad arbetsmängd och högre kostnader.

Att upprätthålla rättigheter, programvara och infrastruktur kan bli utmanande då systemet växer. Då den underliggande infrastrukturen uppdateras eller skalas upp, måste även mjukvaran uppdateras. Ifall nya noder läggs till i ett distribuerat system, måste varje nod installeras för att kunna köra den egna mjukvaran. Genom att använda molntjänster kan man undvika en del arbete.

Organisationer måste administreras och människan har utvecklat olika modeller för att hantera den enskilda individens arbetsbörda. Man kan exempelvis dela upp organisationen i olika ansvarsområden och avdelningar, där var och en har en specifik uppgift. En del av dessa modeller används även inom planeringen av infrastruktur och programvara. Programvara brukar delas upp i olika nivåer. Högre upp finns ofta det yttersta, till exempel det som levererar det slutliga resultatet till användaren. Ju lägre ner man går i nivåer, desto mer specifika blir uppgifterna. Figur 5 visar hur programvara eller organisationer kan uppdelas i olika administrativa nivåer, där den underliggande nivån förmedlar resultat uppåt mellan nivåerna. Kommunikation kan också ske inom de olika nivåerna.



Figur 5: Uppdelning av en organisation eller programvaruhelhet i nivåer med kommunikation representerad som pilar.

Genom att fördela programvaruhelheten i olika nivåer, är det även lättare att senare fördela de administrativa uppgifterna. För att behålla integriteten i systemet, kan man

implementera gränssnitt i de olika delarna av systemet. Gränssnitt tillåter förändringar i enskilda delar av systemet utan att ha desto större inverkan på de övriga delarna.

4.5.1. Mjukvaruleverans

För att underlätta leveransen och lanseringen av programvara, kan man utnyttja metoder av *kontinuerlig integrering* (eng. *continous integration*) och *kontinuerlig leverans* (eng. *continous delivery*), oftast som en helhet som kallas för CI/CD. Ordet kontinuerlig innebär att mjukvaran konstant testas, verifieras och levereras till olika miljöer, helst automatiskt.

Många tjänster som erbjuder versionkontroll, som GitHub och GitLab, erbjuder idag serverlösa lösningar för kontinuerlig integration och leverans. Man kan också köra motsvarande miljöer på sina egna servrar.

4.5.2. Orkestreringsverkyg

För att lättare skapa och hantera skalbara system, finns det i dagens läge en handfull så kallade orkestreringsverktyg. Exempel på sådana är *Kubernetes*, *Docker Swarm* och *OpenShift*. Dessa verktyg har som huvudsaklig uppgift att underlätta skapandet av kluster och hanteringen av program som körs i klustret samt hantering av de olika resurser som programmen behöver. Med resurser menas lagring, DNS-tjänster och lastbalansering.

Oftast associeras orkestreringsverktyg med molntjänster. De flesta orkestreringsverktygen kör programvaran i behållare (eng. *containers*) för att lättare erbjuda rätt omgivning för varje program att köra i. Eftersom alla behållare innehåller allt som varje program behöver, behöver inte värd systemet innehålla dessa saker och man kan även lättare migrera äldre programvara vid sidan om nyare programvara.

Även om många orkestreringsverktyg är öppen källkod, kan de vara integrerade i molntjänsternas verktyg eller så kan företag ha egna versioner av dessa orkestreringsverktyg för att erbjuda bättre integrering med deras arkitektur och infrastruktur. Dokumentationen för dessa modifierade versioner kan variera från den ursprungliga och kan därför också göra användningen svårare. Det gäller därför att förstå sig på vad orkestreringsverktyget försöker åstadkomma med de olika kommandona.

5. Slutsats

Många utmaningar existerar inom planering och byggande av skalbar mjukvara, men då planeringen är gjort ordentligt, kan stora system fungera smärtfritt. Stora internetföretag är beviset på att skalbarhet är en grundsten i leveransen av pålitlig och tillgänglig programvara. Många delar även med sig av sina erfarenheter och undersökningar.

I dag räknar man med att systemen fungerar utan problem 99,9% av tiden. Detta är fallet eftersom vissa internetjättar erbjuder just det och många använder sig av de stora företagens tjänster. I teorin är det dock möjligt för vem som helst att skapa skalbara system och tack vare de stora molntjänst operatörerna är det lätt att starta ett globalt distribuerat system. Olika former av molntjänster kan köpas enligt eget behov. Många mindre och nya företag övergår från IaaS till PaaS för att snabbare få ut sina produkter och inte måsta fundera på den underliggande infrastrukturen. Ännu mer sällsynt är det att nya företag har sina egna fysiska servrar, ifall det inte handlar om mycket känsliga data.

Säkerheten hålls som en av de viktigaste aspekterna av olika system och i och med statliga krav och lagar ser till att det hålls så. Hur dessa lagar påverkar tjänsten är fast i landet, men i de flesta fallen vill företagen ändå betjäna kunden eller användaren på det

bästa möjliga sättet. Användarna blir också mer måna om sin information och vill gärna veta var den finns och vad som görs med den.

Distribuering blir vanligare i och med molntjänster och snabbare förbindelser mellan datacenter. En del som använder sig av tjänster i form av PaaS, vet nödvändigtvis inte ens om att deras programvara körs på en distribuerad plattform. Klientmaskinernas prestanda ökar också, vilket möjliggör distribuering av uträkningar till klientmaskinerna, som i sin tur minskar belastningen på den egna tjänsten.

5.1. Diskussion

Om systemet inte förväntas växa är det viktigare att fokusera på säkerhet och övrig pålitlighet. Systemet kan ofta byggas och levereras snabbare då och det går inte onödig tid till planerandet. Det är också lättare att komma ihåg de viktiga detaljerna som kanske annars faller bort.

Vissa delar borde tas i bruk på bredare plan, även i mindre system. Exempelvis verktyg för kontinuerlig integrering och leverans kunde vara bra för de flesta systemen. I dagen läge erbjuds flera alternativ och många av dem har blivit billiga och lätta att sätta upp. Automatisering av tester och leverans sparar tid och säkerställer att programvaran fungerar innan den levereras.

Överlag anser jag att automatisering och delegering av vissa uppgifter kan vara gynnsamt för vilken verksamhet som helst. Det finns bra modeller för hantering av organisationer och för andra system. Alla modeller fungerar inte för alla ändamål, det är därför viktigt att komma ihåg att många av dem bygger på filosofier och bör kunna tolkas på ett sätt som fungerar för de egna kraven.

Tekniken, infrastrukturen och programvaran ändrar konstant, men teorin bakom det mesta hålls relativt likadan. De verktyg som används för att hantera skalbara miljöer utvecklas konstant för att hållas med i utvecklingen. Skalbarhetens och distribueringens

viktighet förstås av företag och utvecklargemenskapen och kommer även i framtiden att vara en av de mest användbara egenskaperna inom modern teknik.

6. Referenser

[1] T. Dillon, C. Wu och E. Chang, ”Cloud Computing: Issues and Challenges,” *2010 24th IEEE International Conference on Advanced Information Networking and Applications,* pp. 27-33, 2010.

[2] G. Coulouris, J. Dollimore, T. Kindberg och G. Blair, Distributed Systems, Concepts and Design, Fifth edition, Boston: Pearson Education, Inc., 2012.

[3] Netflix Technology Blog, ”Global Cloud — Active-Active and Beyond,” Netflix, 30 Mars 2016. [Online]. Available: https://medium.com/netflix-techblog/global-cloud-active-active-and-beyond-a0fdfa2c3a45. [Använd 28 Mars 2019].

[4] D.-W. Sun, G.-R. Chang, S. Gao, L.-Z. Jin och X.-W. Wang, ”Modeling a Dynamic Data Replication Strategy to Increase System Availability in Cloud Computing Environments,” *Journal of Computer Science and Technology,* vol. 27, nr 1860-4749, pp. 256-272, 2012.

[5] The Go Authors, ”Effective Go - The Go Programming Language,” [Online]. Available: https://golang.org/doc/effective_go.html#names. [Använd 28 Mars 2019].

[6] B. C. Neuman, ”Scale in Distributed Systems”.

[7] A. S. Tanenbaum och M. V. Steen, Distributed Systems Principles and Paradigms, New Jersey: Pearson Education. Inc., 2007.

[8] The Open Web Application Security Project (OWASP), ”OWASP Top Ten Project,” 2017. [Online]. Available: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf. [Använd 25 March 2019].

[9] M. Stevens, E. Bursztein, P. Karpman, A. Albertini och Y. Markov, ”The first collision for full SHA-1,” 2017.

[10] R. Sandhu, D. Ferraiolo och R. Kuhn, ”The NIST Mo del for Role-Based Access Control: Towards A Unifed Standard,” [Online]. Available: https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=916402. [Använd 28 Mars 2019].

[11] T. Laszewski, K. Arora, E. Farr och P. Zonooz, Cloud Native Architectures, Birmingham: Packt Publishing, 2018.