Skalabilitet med hjälp av virtuella maskiner och containrar

Heino Sirviö
Kandidatavhandling
Datavetenskap
Fakulteten för naturvetenskaper och teknik
Åbo Akademi
Handledare: Dragos Truscan

Referat

Virtualisering används i datacenter och i molntjänster för att frikoppla applikationer från hårdvaran de körs på. De två ledande metoderna för att åstadkomma detta är virtualisering på hårdvarunivå och virtualisering på operativsystemnivå. På senare tid har vi bevittnat en markant tillväxt i popularitet hos containrar som använder sig av virtualisering på operativsystemnivå.

I denna avhandling jämförs för- och nackdelar med virtuella maskiner och containrar, och hur dessa två kan kombineras för att uppnå önskad funktionalitet och prestanda.

Jämförelse av dessa teknologier visar att samlokaliserade applikationer som körs på samma maskin kan orsaka konflikter då applikationerna försöker använda sig av gemensamma resurser. Containrar har i allmänhet bättre dataflöde, men drabbas mera av konflikter medan virtuella maskiner i sin tur lider av ”klumpighet”, bland annat på grund av det stora lagringsutrymme de kräver. Genom att kombinera virtuella maskiner och containrar kan man dra nytta av de starka sidorna hos båda, medan man minimerar svagheterna.

Innehållsförteckning

Inledning

Bara några år tillbaka innebar hanteringen av storskaliga webbapplikationer mycket med manuellt arbete. Man var tvungen att bland annat sköta om avbilder av virtuella maskiner och belastningsutjämnare. Det blev snabbt komplicerat att ta hand om allt samtidigt. Sedan kom orkestreringsverktyg som Chef, Puppet, Ansible och Salt ikapp problemet för att göra det lättare att automatisera skalningen av applikationer som körs ovanpå virtuella maskiner.

Även då orkestreringen av virtuella maskiner underlättades, kvarstod några problematiska begränsningar. Under de bästa förhållanden kunde det fortfarande ta minuter att starta upp en virtuell maskin för att sedan få den att kommunicera med belastningsutjämnaren och hantera trafik [1]. Några minuter är betydligt snabbare än att hantera skalabiliteten på maskinnivå, men det var fortfarande långt ifrån det optimala.

Containrar som Docker och LXC siktar på att lösa dilemmat. Containrar är lättviktiga och kan startas upp på mindre än en sekund [2]. De är smidiga och behändiga, men som sådana erbjuder de inte all funktionalitet som en fullständig virtuell maskin gör. Containrarna kräver fortfarande isolering och orkestrering för att köra pålitligt och effektivt. Det handlar inte om triviala problem att lösa.

Moderna företag förlitar sig på it-system för sina affärsbehov. Företagens it-system lagras och körs i datacenter som tar hand om beräkning, lagring och nätverksresurser för applikationerna. Datacenter är allt mera virtualiserade så att applikationer körs på en eller flera virtuella maskiner som sedan mappas till hårdvaran. Virtualisering för med sig ett antal fördelar. Det tillåter justerbarhet i allokering av fysiska resurser till applikationer vars behov av resurser kan variera dynamiskt. En annan egenskap som möjliggörs genom virtualisering är multitenans. Detta innebär att flera instanser av en virtualiserad applikation delar samma fysiska server. Med hjälp av multitenans kan datacenter konsolidera (förtäta) en grupp av applikationer till en mindre mängd av servrar, för att minska

på driftskostnader. Virtualisering förenklar även replikering och skalning av applikationer.

I dagens läge finns två primära tekniker för att hantera virtualisering i datacenter – virtualisering på hårdvaru- och operativsystemnivå. I virtualisering på hårdvarunivå körs ett program som kallas en *hypervisor* [3], som virtualiserar och allokerar serverns resurser till de virtuella maskiner som körs på servern. I OS-virtualisering tar man hand om resurser på OS-nivå, vilket innebär att OS-virtualiseringen inkapslar de vanligaste OS-processerna och deras beroenden för att skapa det som kallas containrar, som sköts av den underliggande operativsystemskärnan. Som tidigare nämnts finns det verktyg för att automatisera och orkestrera virtualiseringen i datacenter, både på hårdvarunivå och OS-nivå.

I denna avhandling kommer skillnaderna mellan virtualisering på hårdvarunivå och OS-nivå att tas fram:

- Hur presterar virtualiseringsmetoderna då flera instanser av en applikation körs på samma server?
- Hur presterar dessa två metoder då en applikation körs i ett kluster?
- Kan dessa metoder kombineras för att utnyttja fördelarna med båda metoderna och för att uppnå önskad funktionalitet?

Bakgrund

I denna sektion ges bakgrundsinformation om de två typerna av virtualisering som diskuteras i denna avhandling.

Hårdvaruvirtualisering

Hårdvaruvirtualisering sker genom att skapa virtuella maskiner som emulerar en fysisk maskin. För att det ska vara möjligt för flera virtuella maskiner att samtidigt köra på en värd krävs det ett skikt som kontrollerar distribueringen av de resurser som den fysiska maskinen erbjuder. Dessa resurser inkluderar bland annat arbetsminnet och centralprocessorn. Virtualiseringen och distribueringen av resurser utförs av hypervisorn (kallas även virtual machine monitor, VMM), som är ett program som introducerar ett abstraktionslager mellan hårdvaran och gästoperativsystemen. Denna abstraktion tillåter virtuella maskiner att vara oberoende av värdens hårdvara, vilket gör det smidigt att flytta dem mellan värdmaskiner enligt behov.

Hypervisorn emulerar virtuell hårdvara som CPU, arbetsminne, I/O-enheter och nätverksenheter för var och en virtuell maskin som körs på servern. Den sköter även om multiplexering (uppdelning av ett till flera) av de fysiska resurserna till alla virtuella maskiner som körs. Moderna hypervisorer stöder olika sätt att allokera och dela fysiska resurser. Man kan bland annat välja att allokera dedikerade resurser för var och en virtuell maskin. En alternativ metod är använda sig av *best effort*-beteende (för mera information om *best effort* se [4]).

En viktig uppgift som tillhör hypervisorn är isolering, vilket innebär att hypervisorn ansvarar för att de virtuella maskinerna som körs påverkar varandra så lite som möjligt. Den åstadkommer isolering genom att begränsa gästoperativsystemets direkta hårdvarutillgång för att köra operationerna genom hypervisorn istället.

Några populära alternativ för hårdvaruvirtualisering är bland annat VMware ESXi, Xen, Linux KVM och Microsoft Hyper-V.

Hypervisor

Hypervisorn är ett virtualiseringsverktyg som låter flera operativsystem dela på gemensam hårdvara. För varje operativsystem ser det ut som att den har tillgång

till alla resurser som värddatorn har att erbjuda, även om det är flera operativsystem som delar på hårdvaran.

Det finns två huvudtyper av hypervisorer. Typ 1 hypervisorer kallas *bare metal*. Bare metal-hypervisorn låter operativsystemen bestämma över datorns hårdvara direkt genom en virtuell maskin. Typ 2 hypervisorer kallas *hosted* och till skillnad från bare metal-hypervisorer har den ett värdoperativsystem och kör gästoperativsystemen i virtuella maskinerna mot värdsystemet.

Enligt IDG:s ordlista är en *hypervisor* ett program som låter flera operativsystem dela på en dator, medan en *hyperviser* döljer den underliggande hårdvaran för operativsystemen genom emulering. Dessa två ord är så lika att de ständigt blandas ihop.

Operativsystemnivå-virtualisering

Operativsystemsnivå-virtualisering innebär att man till skillnad från hårdvaruvirtualisering virtualiserar OS-kärnan istället för den fysiska hårdvaran. De virtuella maskinerna som sedan inkapslar OS-processerna kallas containrar. Var och en container är med andra ord isolerad från de andra containrarna, vilket betyder att de processer som körs inom containrarna är fullständigt omedvetna om andra processer som eventuellt körs på samma värddator. Det är OS-kärnan som ansvarar för containerabstraktionen. Den står för allokeringen av de fysiska resurserna (CPU, arbetsminne, nätverksenheter) för var och en container. OS-kärnan sköter även om isoleringen av filsystemet.

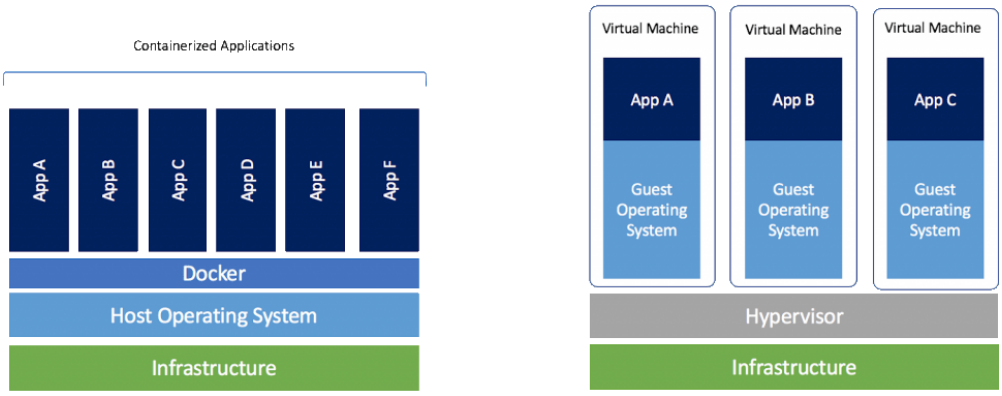
Liksom i hårdvaruvirtualisering kan man välja att använda sig av olika metoder då man allokerar resurser till containrarna. Några av de vanligaste sätten att allokera resurser är att använda sig av *delad*-beteende (kallas *shared* på engelska), *dedikerad*-beteende (*dedicated* på engelska), eller *best effort* (för mera information om *shared och dedicated* se [5]).

Containrar är lättviktiga jämfört med virtuella maskiner eftersom de inte kör egna operativsystem, utan använder sig av det underliggande operativsystemet och de tjänster som operativsystemet erbjuder.

Det är möjligt för OS-kärnan att emulera olika versioner av OS-kärnor för de processer som körs i en container för att stöda bakåt-kompabilitet hos applikationer. Denna egenskap, att kunna emulera extra funktionalitet är något som bland annat används av Windows för att kunna köra Linux-applikationer [6].

Containern har haft en lång historia inom datavetenskapen, men den har inte varit särskilt populär innan Docker uppstod. Docker erbjuder funktioner som underlättar mjukvaruutvecklingsprocessen. De två viktigaste egenskaperna som Linux containrar använder sig av är *Cgroup* [7] och *Namnrymd* (kallas *namespace* på engelska) [8]. Cgroup är en mekanism som OS-kärnan använder sig av för att kontrollera resurser som är allokerade för processgrupper. Cgroup existerar för alla de viktigaste typerna av resurser som, bland annat inkluderar CPU och arbetsminne. Med hjälp av namnrymden skapas en abstraktion av OS-kärnans resurser för containrarna. Med andra ord får namnrymden det att verka som att var och en container har en egen isolerad instans av den resursen som finns i namnrymden. Den skapar isoleringen genom att tilldela containerns ID till processerna.

I figur 1 (lånad från [9]) visas hur de två ovannämnda typerna av virtualisering är strukturerade. I figuren används Docker som exempel, men samma struktur gäller för alla varianter av containrar. Till vänster ser man att containrarna som i det här fallet körs av Docker Engine, ligger ovanpå operativsystemet till skillnad från virtuella maskinerna som körs på hypervisorn. Var och en virtuell maskin innehåller en egen avbild av ett operativsystem, vilket innebär att mera lagringsutrymme krävs.



Figur 1: Visualisering av uppbyggnadsstrukturen av hårdvaruvirtualisering och operativsystemnivå-virtualisering.

Virtualisering i datorkluster

Hittills har främst virtualisering av resurserna hos en enda server behandlats i avhandlingen. De två ovannämnda typerna av virtualisering fungerar på nivån av individuella maskiner, men i verkligheten delas arbetsbördan i datacenter av stora kluster av servrar, där alla maskiner är virtualiserade. Följaktligen förlitar sig datacentralen på ramverk som effektivt tar hand om en serverklusters resurser.

Dessa ramverk används som verktyg för att underlätta mappningen av virtuella maskiner till fysiska maskiner. Ramverket tar även hand om lastbalansering då arbetsbördan stiger eller sjunker, genom att flytta eller kopiera virtuella maskiner från en server till en annan. Ramverk för hanteringen av virtualisering i datacenter, har utvecklats och optimerats under tiotals år, för att effektivt orkestrera och automatisera styrningen av de virtualiserade serverklusterna. Några populära ramverk för hårdvaruvirtualisering är OpenStack och VMware vCenter. Kubernetes och Docker Swarm är däremot de mest använda ramverken för OS-virtualisering.

Prestandajämförelse av virtualiseringsmetoderna

Det finns många aspekter att ta i beaktande då man mäter prestanda, men i den här avhandlingen fokuseras det främst på pålägg (overhead). Genom att mäta dataflödet och latensen (väntetid, fördröjning) under virtualisering i förhållandet till icke-virtualiserad exekvering av samma applikationer får vi reda på hur mycket pålägg som orsakas av virtualiseringsmetoderna. En annan sak som kan vara av intresse, är skalabiliteten av applikationer. Om en virtuell maskin kraschar, kommer den vertikala skalabiliteten av virtuella maskiner påverka prestandan av applikationen som en helhet. Skalabiliteten av virtuella maskiner beror mycket på applikationens arkitektur. Docker har gjort containrar lättare att ta i bruk och flytta enligt behov, vilket underlättar skalningen av applikationer.

I detta kapitel tas två olika prestandaundersökningar upp. Orsaken till att ta upp två olika forskningar är att få en klarare bild av den allmänna prestandaskillnaden mellan hårdvaruvirtualisering och OS-virtualisering.

Forskning 1

I det här kapitlet tas det en titt på undersökningen *Performance Comparison between Linux Containers and Virtual Machines* [10], genomförd av Ann Mary Joy, professor i datavetenskap. Professor Joy var intresserad av riktmärkning av prestandan hos virtuella maskiner och containrar då båda körde samma applikation. Hon mätte skalabilitet på basis av tiden det tog för virtuella maskiner och containrar att skala upp eller skala ner om de skulle uppleva ett misslyckande (en krasch).

Upplägget av testsystemet inkluderade AWS ec2 molntjänsten och två fysiska servrar med serverversionen av Ubuntu 14.04 installerat på dem. 2GB av RAM allokerades för ec2-maskinerna och för att köra Docker på de fysiska servrarna.

Prestanda

Upplägget för prestandajämförelsen bestod av en front-end server som var värd för en Joomla php-applikation, och en back-end server som var värd för en postgreSQL databas som kopplades till front-end applikationen. Joy valde att använda JMeter för att skapa ett stresstest för servrarna. JMeter är ett program under Apache 2.0-lisens som hon använde för att skicka meddelanden till Joomlaapplikationen tills servrarna upplevde störningar i prestandan.

Testet utfördes genom att skicka så många meddelanden som möjligt inom 10 minuter. Programmet började med att skicka ett meddelande, och väntade sedan på svar från servern innan det skickade nästa meddelande. Antalet parallella meddelanden som JMeter skickade ut ökade linjärt tills programmet, efter 520 sekunder nådde 80 samtidiga meddelanden.

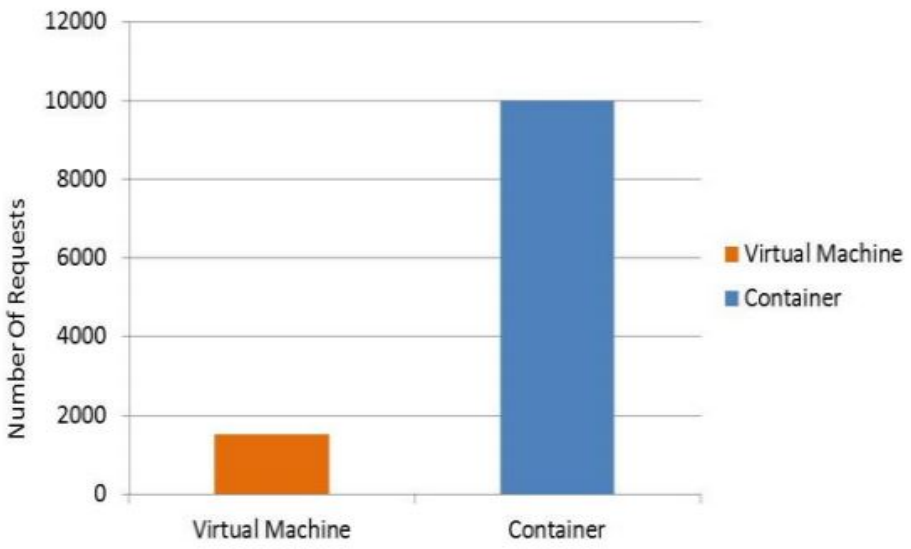
I figur 2 (lånat från sida 345 av [10]) syns antalet meddelanden som behandlades framgångsrikt under 600 sekunder. Joy rapporterade att Docker behandlade över tre gånger mera anrop än virtuella maskinerna under de första 400 sekunderna. Joys forskning visar även att det tog längre tid för den virtuella maskinen att behandla individuella meddelanden. Enligt henne är inte den virtuella maskinen lika konsekvent som Docker. Hon menar att detta beror på virtuella maskinens minneshanteringsprocedurer.

Skalabilitet

För skalabilitetstestet valde professor Joy att använda AWS ec2 automatiska skalningsverktyg. Verktyget skalar upp virtuella maskin-instanser då de når den maximala CPU-kapaciteten. För att skala upp Linuxcontainrarna använde hon Kubernetes, ett verktyg som tog hand om skalningen av Dockercontainrarna. Med hjälp av Kubernetes, jobbade båda fysiska servrarna i samma kluster.

I hennes experiment körde båda virtualiseringsomgivningarna en WordPress-applikation som var lastbalanserad. Hon använde återigen JMeter för att öka

arbetsbördan. Hon rapporterar att det tog mycket längre tid för virtuella maskinen att skala upp för att hantera den ökade arbetsbördan jämfört med containern. Det tog tre minuter för virtuella maskinen att skala upp, medan det bara tog 8 sekunder för containern. Det handlar alltså om en markant skillnad mellan dessa två metoder.



Figur 2: Antalet behandlade meddelanden under ett 600 sekunder långt stresstest.

Forskning 2

I föregående kapitel sågs en förvånansvärt stor skillnad mellan virtuella maskiner och containrar, både i prestanda och i skalabilitet. För att undersöka om det generellt är så stor skillnad mellan dem kan det löna sig att se om andra forskningsresultat upprepar samma mönster. I det här kapitlet analyseras resultat från *Containers and Virtual Machines at Scale: A Comparative Study*, av P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay [11].

Författarna av publikationen hade en mera fördjupad tillvägagångssätt då de undersökte prestandaskillnader mellan hårdvaruvirtualisering och OS-virtualisering.

Sharma och hans kollegor var intresserade av att ta reda på hur virtualiseringen beter sig då den virtualiserade servern utsätts för olika typer av stresstest. Två av de vanligaste utmaningarna som måste beaktas i samband med virtualisering är multitenans och övertilldelning av resurser. Multitenans uppstår då flera applikationer tas i bruk, och delar på samma fysiska resurser på värddatorn. Övertilldelning sker då man valt att köra applikationer som kräver mera resurser än vad som fysiskt finns tillgängligt på servern, eller servrarna. Både multitenans och övertilldelning används för att driva ner kostnaderna i stora datacenter.
I alla experiment som Sharma och kans kollegor utförde, valde de att använda sig av KVM för att köra virtuella maskiner, och LXC som bas för containrar. De poängterade att både KVM och LXC använder sig av samma Linux-kärna, vilket gav virtualiseringsmetoderna en så rättvis utgångspunkt som möjligt.

Då endast en instans av en applikation körde normalt på servern syntes inga märkvärda skillnader i prestanda mellan KVM och LXC. Hårdvaruvirtualisering drabbas inte mycket av pålägg då applikationen inte behöver gå igenom hypervisorn. Då läsning och skrivning testades blev nackdelen med hårdvaruvirtualisering synligare. All åtkomst till hårddisken går genom hypervisorn vilket kan leda till problem då disken behöver utföra en större mängd av små läsningar och skrivningar.

Efter att författarna till publikationen etablerade en grund för undersökningen fortsatte de med att komparera hårdvaru- och OS-virtualisering i de fallen då:

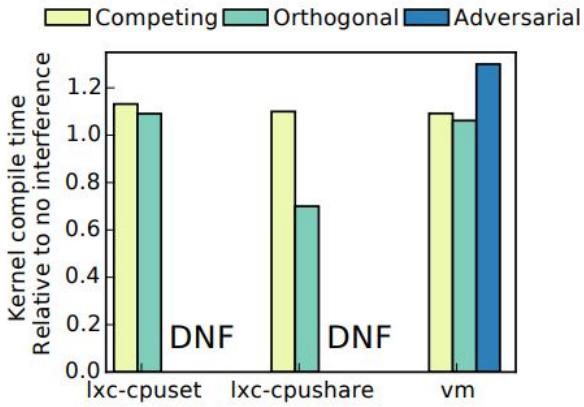
- Närliggande applikationer försökte använda sig av samma resurser (t.ex. då applikationer utförde CPU-intensiva beräkningar samtidigt).
- Närliggande applikationer använde sig av olika resurser (t.ex. då en applikation använde CPU, medan den andra använde nätverksenheten).
- En eller flera närliggande applikationer försökte hindra användningen av en resurs genom att ockupera den fullständigt.

De testade de ovannämnda scenariona med fokus på isolering av CPU, arbetsminne, lagringsenhet och nätverksenhet skilt för sig.

Resultat

För att få reda på hur mycket närliggande containrar och virtuella maskiner stör varandras CPU-prestanda jämfördes körtiden för kompilering av Linux-kärnan i förhållande till när de kördes individuellt (utan störningar). För att stressa processorn användes en fork-bomb, som förgrenade en process om och om igen.

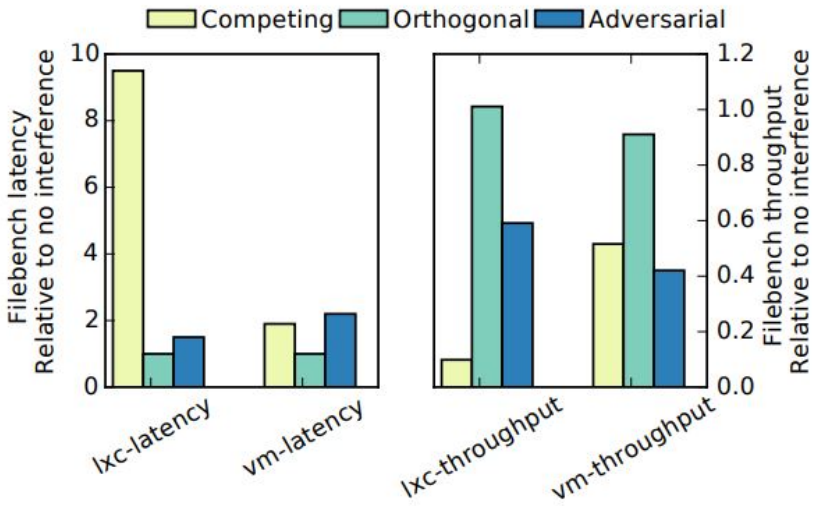
Det finns två inställningar att beakta då man allokerar CPU-resurser för LXC. Man kan välja att tilldela containrar till CPU-kärnor (cpu-sets), eller att multiplexera containers över alla CPU-kärnorna (cpu-shares). I denna undersökning allokerades båda varianterna samma mängd resurser (50% CPU, 2 av 4 kärnor). Det visade sig att CPU-shares upplevde betydligt mera störningar - upp till 60% mera, jämfört med den individuella, störningsfria prestandan. När LXC-containrarna var inställda för att försöka ockupera resurserna från närliggande containrar, slutfördes inte kompileringen inom sund tidsram. Virtuella maskinerna klarade av att slutföra kompileringen med en prestandaförlust på 30%. Se figur 3 (lånad från [11]) för tydligare resultat.



Figur 3: Resultat av störning under CPU-intensiv arbetsbörda. Containrarna upplevde mera störning än virtuella maskinerna. Då containrarna var inställda för att ockupera alla resurser, slutfördes inte testet inom vettig tidsram.

Testet för arbetsminne utfördes på samma vis som för CPU, men istället för att köra en fork-bomb, användes nu en malloc-bomb, som inkrementellt allokerade minne tills minnet tog slut. Containrar ser ut att prestera bättre då applikationerna använder sig av olika resurser. I alla andra fall var virtuella maskinerna mera konsistenta. Störningarna som containrarna upplevde är inom acceptabla ramar, beroende på användningsfall.

När Sharmas forskningsgrupp testade hur containrar och virtuella maskiner hanterar skrivningar till, och läsningar från lagringsenheten, bemöttes de av ett intressant resultat. Latensen för LXC ökade med en faktor av 8, medan latensen för virtuella maskinerna endast ökade med en faktor av 2. De hänvisar i artikeln till att lagringsprestandan för enskilda virtuella maskiner ursprungligen var mycket sämre än för containrar. Hursomhelst är skillnaden signifikant, och tyder på dålig isolering av I/O-enheter hos containrar. Figur 4 (lånat från [11]), visar langringstestets slutresultat.



Figur 4: Både containrar och virtuella maskiner upplevde en hög nivå av störningar under läsning från och skrivning till lagringsenheten.

Slutligen testades isoleringen av nätverksenheterna. För att mäta störningar i nätverket användes en UDP-bomb på servern, som försöker skicka ut så många

små paket som möjligt. Sharma och hans kollegor bemöttes av nästan identiska resultat för både virtuella maskinerna och containrarna. Då applikationerna försökte överbelasta samma resurser sjönk prestandan betydligt för båda virtualiseringsmetoderna.

Summering av kapitlet: Forskningen som publicerats i *Containers and Virtual Machines at Scale: A Comparative Study*, visar att containrar presterar bättre än virtuella maskiner i många fall, men även att virtuella maskiner är mera konsistenta på grund av den extra isoleringen de erbjuder.

Virtualisering i kluster

Det finns några viktiga saker att tänka på då man väljer att bygga ett skalbart it-system. Klusteroperatorer försöker fylla alla resurskrav som applikationerna ställer, samtidigt som de försöker öka konsolideringen så mycket som möjligt. Konsolidering innebär att man försöker utnyttja servrarnas kapacitet så bra som möjligt för att minska på driftskostnader. I det här kapitlet gås det igenom hur egenskaperna hos containrar och virtuella maskiner påverkar hanteringen och provisioneringen av resurser i ett kluster.

En av de viktigaste aspekterna är naturligtvis allokering av hårdvaruresurser. Det diskuterades tidigare i avhandlingen att containrar erbjuder möjligheten att använda sig av mjuka allokeringsbegränsningar av hårdvaruresurser. Virtuella maskinernas resurser är bestämda då de startas upp, vilket begränsar möjligheterna för optimering av resursallokering. Det är möjligt att dynamiskt allokera mera resurser till en virtuell maskin, förutsatt att operativsystemet stöder det. Tyvärr stöds inte dynamisk allokering av alla ramverk för hantering av virtuella maskin-kluster.

En annan aspekt som hittills har förbisetts i avhandlingen är migrering av virtuella maskiner och containrar. Migrering av virtualiserade applikationer används i datacenter för lastbalansering, konsolidering och för att hantera feltolerans.

Virtuella maskiner stöder live-migrering (migrering av en uppstartad virtuell maskin) genom att tidvis kopiera från sitt minne till destinationen för den nya maskinen. Eftersom virtuella maskiner har ett stort fotavtryck innebär detta överföring av en stor mängd data. Live-migrering av virtuella maskiner har länge använts i datacenter, och det stöds av de flesta ramverken.

Tyvärr erbjuder inte containrar samma live-migreringsmöjligheter. *CRIU* (Checkpont Restart In Userspace) [12], är ett projekt som har som mål att erbjuda live-migrering av containrar, men funktionaliteten av deras tjänst är begränsad. CRIU stöder bara en mindre mängd applikationer som använder sig av specifika OS-tjänster. Lösningen är att helt enkelt starta en ny instans av containern, vilket oftast sker inom en sekund.

Som avslutning till kapitlet behandlas brukstagning av nya instanser av applikationer. Då man jobbar med skalning av applikationer är förmågan att starta nya virtuella maskiner och containrar väldigt viktig. Naturligtvis ska det även ske fort. Man har under många år gjort en märkvärdig satsning på att optimera provisioneringspolicyn, som ser till att de nya instanserna som startas upp, uppfyller de krav som ställs på datacentret (bland annat tids- och konsolideringskrav). Orkestreringsverktyg som Kubernetes tar hand om skalning av applikationer. Om applikationen behöver skalas upp horisontellt, startar Kubernetes en ny kopia av containern. Verktyget övervakar även de containrar som redan är igång, ifall någon av dem skulle krascha, i vilket fall Kubernetes ersätter den kraschade containern med en ny.

Kombinera containrar med virtuella maskiner

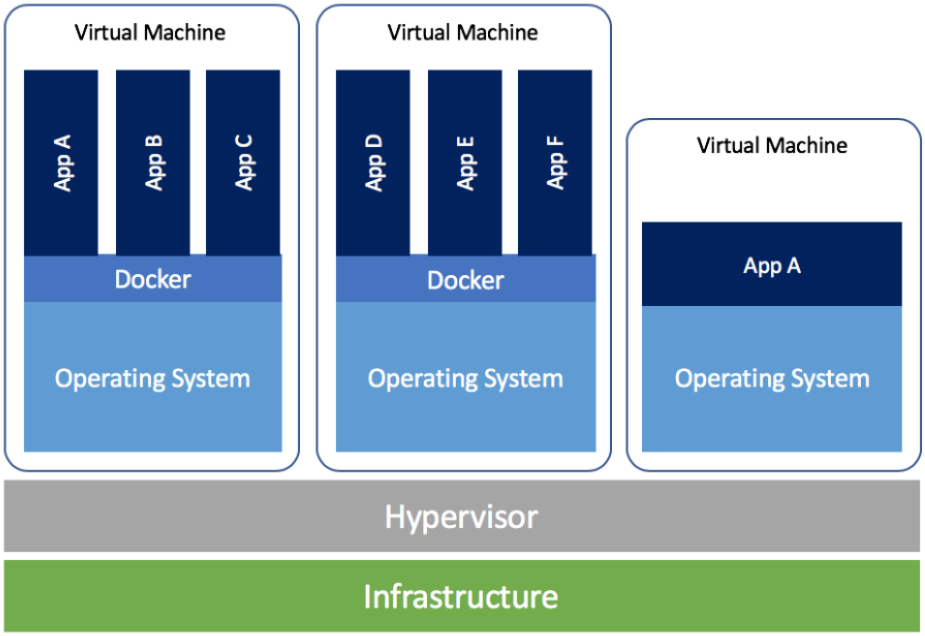
I avhandlingen har det hittills tagits upp skillnaderna mellan containrar och virtuella maskiner, och vilka kompromisser de för med sig. Idealt skulle man vilja låta bli att kompromissa i varken prestanda eller säkerhet. Genom att kombinera containrar och virtuella maskiner hoppas man på att utnyttja de goda egenskaperna hos båda, medan man önskar att minimera nackdelarna. I detta

kapitel behandlas två tillvägagångssätt för att kombinera virtualiseringsmetoderna – placering av containrar inuti en virtuell maskin, och lättvikts-virtuella maskiner (lightweight VM).

Det förstnämnda alternativet innebär att alla containrar är inkapslade i en virtuell maskin, och att applikationerna i sin tur körs i containrarna. Det kan alltså finnas en eller flera containrar inuti en virtuell maskin. Det finns ett flertal fördelar med att kombinera containrar och virtuella maskiner på det viset. Applikationerna kan dra fördel av den ökade säkerheten till följd av isoleringen som virtuella maskiner för med sig. Samtidigt utnyttjar man provisionerings- och implementeringsegenskaperna som containrar har att ge. Kombinationen av hårdvaruvirtualisering och OS-virtualisering är rätt populär, och den har använts av molntjänster som Amazon AWS och Google Cloud Platform. Dessa molntjänster erbjuder i sin tur stöd för driftsättning av Docker containrar inuti dem. I figur 5 (lånad från [9]) illustreras den strukturella skillnaden mellan containrar som körs i virtuella maskiner i jämförelse med enbart en virtuell maskin.

Om det underliggande operativsystemet saknar stöd för containrar, men fortfarande har stöd för virtuella maskiner, kan man välja att starta upp en virtuell maskin som har stöd för de containrar man önskar använda. Den här typen av användning av containrar gör det lättare för programvaruutvecklare att arbeta med containrar som inte stöds av det operativsystem de jobbar i.

Då man begränsar antal användare av en virtuell maskin kan man säkra sig av att de containrar som körs i en virtuell maskin inte försöker ockupera alla resurser för att skada sina närliggande containrar. Eftersom man vet att de närliggande containrarna är pålitliga, är det mera attrktivt att använda mjuka allokeringsbegränsningar av resurser. Som tidigare diskuterats, innebär det att containrar har möjligheten att utnyttja resurser som förblivit oanvända av närliggande containrar i samma virtuella maskin.



Figur 5: Visualisering av Dockercontainrar som körs inuti virtuella maskiner jämfört med enbart en enkel virtuell maskin.

Det finns en alternativ lösning för att uppnå liknande funktionalitet, utan att behöva köra containrar inuti virtuella maskiner. Lättvikts-virtuella maskiner som Clear Linux [13] och Firecracker [14], lovar samma säkerhet och isolering som traditionella virtuella maskiner erbjuder, och samma prestanda och utvecklingsegenskaper som containrar har. Lättvikts-virtuella maskiner är paravirtualiserade, vilket innebär att gästoperativsystemet som körs i en lättvikts-virtuell maskin är optimerat för virtualisering, till skillnad från operativsystemet i en traditionell virtuell maskin som agerar likadant som de gör då det inte är virtualiserat. På så vis kan Clear Linux, Firecracker och andra hybrider erbjuda hårdvaruvirtualisering med ett minskat fotavtryck.

En av de största nackdelarna med traditionella virtuella maskiner är fotavtrycket, något som dessa hybrider försöker åtgärda genom att radera överflödig funktionalitet som vanliga hypervisorer fortfarande innehåller. Det handlar bland annat om funktionalitet som bootloaders och stöd för äldre hårdvaruenheter som

t.ex. diskettenheten. Genom nedskärning av onödig funktionalitet och genom optimering av OS-kärnan för snabbstart, kan nu lättvikts-virtuella maskiner startas upp på under en sekund, jämfört med traditionella virtuella maskiner som behöver tiotals sekunder för att komma igång. Det handlar dock om kallstart i detta fall. Traditionella virtuella maskiner kan återställas från en ögonblicksbild (snapshot på engelska), eller klonas från en annan virtuell maskin betydligt snabbare än vad de kallstartas.

En viktig egenskap hos lättvikts-virtuella maskiner är att de har direkt tillgång till filsystemet. Det eliminerar behovet för att skapa en virtuell diskavbild för varje applikation. Traditionella virtuella maskiner kräver fortfarande abstraktionen av en virtuell disk.

Den nya tekniken som tillåter hårdvaruvirtualisering direkt åtkomst till värdens filsystem gör de möjligt att kombinera hårdvaru- och OS-virtualisering. Clear Linux erbjuder möjligheten att köra existerande Dockercontainrar som lättvikts-virtuella maskiner. För användarna verkar det som om containrarna fungerar som vanligt, men de har via Clear Linux fått ökad isolering. Tyvärr är Firecracker inte Docker-kompatibelt i skrivande stund, men enligt Amazon AWS är det i sikte i snar framtid [15].

Slutsats

Containrar och virtuella maskiner skiljer sig fundamentalt i hur de är uppbyggda, och det visas även i deras prestanda. Varken den ena eller den andra täcker alla användningsfall. Forskningarna som hänvisas till i avhandlingen visar att containrar presterar minst lika bra, om inte bättre (framförallt när det handlar om dataflöde) än virtuella maskiner, men de kan fortfarande drabbas av minskad prestanda som resultat av störningar på grund av multitenans. En tydlig nackdel för OS-virtualisering är att den använder sig av den underliggande OS-kärnan och saknar därför samma isoleringsmöjligheter som hypervisorerna har att erbjuda för virtuella maskiner. Däremot erbjuder OS-virtualisering möjligheten att använda

sig av mjuka allokeringsbegränsningar av resurser, till skillnad från virtuella maskiner vars allokering av resurser är strikta. Denna egenskap är i synnerhet användbar då man vill utnyttja oanvända resurser som allokerats för närliggande containrar, och genom att göra så, öka prestanda.

Bortsett från möjlig prestandaökning så har populariteten av OS-virtualisering ökat som följd av de fördelar den för med sig till programvaruutvecklingsprocessen. Detta gäller i synnerhet för Docker som erbjuder *kopiering vid ändring* (*copy-on-write,* förkortat *cow* på engelska) och versionshantering, vilket underlättar programvaruutvecklingsprocessen från ändpunkt till ändpunkt.

För att dra nytta av den ökade prestandan som containrar erbjuder utan att kompromissa med säkerhet genom isolering, rekommenderas det att slå ihop båda teknikerna. Genom att köra en mängd containrar inuti en virtuell maskin, garanterar man att containrarna är isolerade från eventuella processer som körs utanför virtuella maskinen.

Intresset för att utveckla en variant av virtualisering som utnyttjar fördelarna hos både containrar och virtuella maskiner har ökat betydligt. Clear Linux och Firecracker är lättvikts-virtuella maskiner som erbjuder isolering av en virtuell maskin tillsammans med de utvecklingsegenskaper som containrar för med sig. Forskningsområdet för hybrider som Clear Linux och Firecracker har en lovande framtid.

Källförteckning

[1] D. K. Rensin, *Kubernetes - Scheduling the Future at Cloud Scale*. 2015.

[2] J. Turnbull, *The Docker Book: Containerization is the new virtualization*. 2014.

[3] WMware, “Hypervisor.” [Online]. Available: https://www.vmware.com/topics/glossary/content/hypervisor. [Accessed: 14-Mar-2019].

[4] T. L. I. P. (LINFO), “Best Effort Definition by.” [Online]. Available: http://www.linfo.org/best_effort.html. [Accessed: 29-Mar-2019].

[5] DreamHost, “Shared Hosting vs Dedicated Hosting.” [Online]. Available: https://www.dreamhost.com/hosting/shared-vs-dedicated/. [Accessed: 07-Apr-2019].

[6] Microsoft, “Windows Subsystem for Linux Documentation,” 2016. [Online]. Available: https://docs.microsoft.com/en-us/windows/wsl/about. [Accessed: 07-Apr-2019].

[7] P. Menage, “Linux cgroups,” 2004. [Online]. Available: https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt. [Accessed: 07-Apr-2019].

[8] “Linux namespaces.” [Online]. Available: http://man7.org/linux/man-pages/man7/namespaces.7.html. [Accessed: 07-Apr-2019].

[9] J. Fong, “Are containers replacing virtual machines?,” 2018. [Online]. Available: https://blog.docker.com/2018/08/containers-replacing-virtual-machines/. [Accessed: 12-Mar-2019].

[10] A. M. Joy, “Performance comparison between Linux containers and virtual machines,” *Conf. Proceeding - 2015 Int. Conf. Adv. Comput. Eng. Appl. ICACEA 2015*, no. Lxc, pp. 342–346, 2015.

[11] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, “Containers and Virtual Machines at Scale,” *Proc. 17th Int. Middlew. Conf. - Middlew. ’16*, pp. 1–13, 2016.

[12] “CRIU.” [Online]. Available: https://criu.org/. [Accessed: 01-Apr-2019].

[13] “Clear Linux* Project.” [Online]. Available: https://clearlinux.org/. [Accessed: 10-Apr-2019].

[14] Amazon, “Introducing Firecracker,” 2018. [Online]. Available: https://aws.amazon.com/about-aws/whats-new/2018/11/firecracker-lightweight-virtualization-for-serverless-computing/. [Accessed: 08-Apr-2019].

[15] “Firecracker.” [Online]. Available: https://firecracker-microvm.github.io/. [Accessed: 03-Apr-2019].