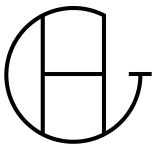

HoGent

Faculteit Bedrijf en Organisatie

Unikernels

Michiel De Wilde

Scriptie voorgedragen tot het bekomen van de graad van
Bachelor in de toegepaste informatica

Promotor:
Bert Van Vreckem
Co-promotor:
Jan Janssen

Instelling: Hogeschool Gent

Academiejaar: 2015-2016

2e examenperiode

Faculteit Bedrijf en Organisatie

Unikernels

Michiel De Wilde

Scriptie voorgedragen tot het bekomen van de graad van
Bachelor in de toegepaste informatica

Promotor:
Bert Van Vreckem
Co-promotor:
Jan Janssen

Instelling: Hogeschool Gent

Academiejaar: 2015-2016

2e examenperiode

Samenvatting

Voorwoord

Toen ik begon met programmeren had ik geen idee wat er gebeurde in de achtergrond van de computer. Ik starte met de simpele todolist-applicaties om alles te leren over programmeren. Na een tijd kwam ik een black box tegen: het besturingssysteem. Vooral om servers sneller te laten werken en de techniek erachter te leren kennen begon ik aan een zoektocht. Linux was de startplek bij uitstek. Package managers en file systems waren de eerste concepten die mij met verstomming lieten staan. Toen ik meer en meer naar infrastructuur keek begon ik termen te leren en alle handige tips om ervoor te zorgen dat je server altijd beschikbaar is. Toen een paar jaar geleden Docker voor het eerst echt vaart maakte met containers was ik verbaasd. Ik dacht eerst dat dit nooit zou werken. Na een tijd heb ik wel het licht gezien en gebruikte ik containers meer en meer. Toen er gevraagd werd om een onderwerp voor mijn thesis dacht ik meteen en wat volgens mij de volgende stap is: unikernels.

Inhoudsopgave

Hoofdstuk 1

Inleiding

De wereld is steeds in verandering. Dit is een zekere waarheid in de wereld van informatica. Kijk maar naar de veranderingen dat er gebeuren elke paar maanden op vlak van javascript frameworks. Best practices van vijf jaar geleden, kunnen bad practices zijn vandaag. Je moet mee met de deze stroom van veranderingen wil je relevant blijven. Dit is voor iedereen in uitdaging. In mijn mening is dit iets goed, blijven leren is een van de beste dingen die je kan doen.

Er zijn grote veranderingen aan het gebeuren op het vlak van infrastructuur. Een paar geleden was de een virtuele machine met een klassieke stack de normale gang van zaken. De verandering heeft hier ook voor veranderingen gezorgd. Een container was al een tijd een droom van vele mensen maar er was nog geen echte goede uitwerking. Tot Docker. Met Docker werden containers eenvoudiger om te gebruiken. Unikernels zitten nu in de situatie van containers vooraleer Docker opzetten kwam. Zal unikernels dezelfde revolutie ontketenen of zal het hand in hand kunnen leven met Docker? Docker heeft unikernel systems overgenomen begin vorige jaar en je kan de invloed al zien in hun releases.

Deze thesis focust niet enkel op unikernels. 1 Van de onderzoeksvragen gaat over de rol van systeembeheerder in de toekomst. Unikernels en Docker hebben een groot gevolg voor hun maar zijn nog andere factoren die een rol spelen zoals orchestration frameworks, registries, CI/CD en veel meer.

Ik zal mijn uiterste best doen om deze omgeving te beschrijven en aan te tonen wat er kan gebeuren in de toekomst maar er kunnen natuurlijk nieuwe technologieen te voorschijn komen. Dit is een antwoord op de onderzoeksvragen vanuit mijn standpunt en de huidige informatie beschikbaar.

1.1 Probleemstelling en Onderzoeksvragen

Unikernels zijn een een nieuwe stroom binnen het landschap van besturingssystemen. We hebben al aangehaald dat containers de nieuwe werkwijze is wanneer men applicaties wilt ontwikkelen en schaalbaar wilt maken. Unikernels gaat nog een stap verder. De systeembeheerders moeten dus mee met containers en de veranderingen ondergaan. De vraag is welke veranderingen er zich zullen voordoen wanneer unikernels op de markt komen. Zullen de competenties van de systeembeheerder veranderen? Wordt het opzetten van applicaties meer en meer eenvoudiger of juist niet? We kunnen wel spreken over de opvolger van containers maar is deze al werkbaar in de toekomst? Wat is de impact op beveiliging, meer bepaald aspecten als beschikbaarheid, autorisatie, integriteit en vertrouwelijkheid van gegevens?

Hoofdstuk 2

Methodologie

Hoofdstuk 3

Virtualisatie

Vroeger was de tijd dat je een computer kon gebruiken beperkt. Vooral bij de eerste computers had men problemen om programma's en concepten uit te werken omdat de computer door meerdere mensen gebruikt werd. Een voorbeeld vanuit een situatie uit die tijd is het ontwikkelen van een programma. De source code van een programma werd manueel ingegeven en als een job in een queue gezet. Pas een paar dagen later kon men de resultaten bekijken. Als je een kleine schrijf- en/of logische denkfout maakte dan kon je opnieuw beginnen. De grootste bijdrage tot de ontwikkelsnelheid van programma's is de lengte van de feedbackcyclus: hoe snel kan je je programma testen laten werken. Dit leidt tot een verlies van tijd en dus geld.

Timesharing werd uitgevonden om het verlies van tijd te beperken. Bij timesharing konden de gebruikers inloggen op een console en zo computer gebruiken. Dit was een technische uitdaging. De computer zou van de ene context naar de andere moeten kunnen veranderen. Timesharing en verschillende gebruikers op 1 computer zou de basis vormen voor de moderne besturingsystemen. 1 Van de nadelen van timesharing is de isolatie van twee verschiilende processen. Als er een kans bestaat dat ze elkaar kunnen beinvloeden is dit een heel groot probleem.

Een besturingsysteem moet kunnen werken op verschillende soorten hardware. Om al deze hardware te ondersteunen moet er een standaard zijn waarop een besturingssysteem kan gebouwd worden. Dit zou betekenen dat veel van de complexiteit van het besturingssysteem naar de hardware zou verhuizen. Spijtig genoeg gebeurt dit nog altijd niet. Dit is 1 van de voornaamste redenen waarom de grootte van een OS tussen de 200MB en 1GB kan liggen. Een aantal besturingssystemen doen niet de moeite om de vele soorten hardware apparatuur te ondersteunen. Zij tonen de specificaties die een hardware apparaat moet hebben wanneer je het besturingssysteem wil gebruiken.

Doorheen de tijd werden computers meer krachtig en applicaties konden niet alle middelen van de computer ten volle benutten. Dit zorgde voor de creatie van virtual resources. Deze gingen de fysieke hardware simuleren om zo verschillende applicaties tegelijkertijd te laten werken op dezelfde virtuele machine. Zo worden de middelen van een fysieke machine optimaal gebruikt. De algemene term voor dit concept is virtualisatie. Bij het virtualiseren van een server gaat men een fysieke server opdelen in verschillende kleine en geisoleerde delen. Deze delen kunnen dan gebruikt worden door verschillende gebruikers. De voordelen van het virtualiseren van een server zijn de volgende: financieel voordeel (men kan van 1 taak naar meerdere taken gaan op 1 server), het besparen van energie (minder servers gebruiken want ze worden beter benut), betere beschikbaarheid.

3.1 Hypervisor

Een hypervisor is een stuk software, firmware of hardware waarop een virtuele machine zich bevindt. De host machine zorgt voor de middelen zoals CPU, RAM, ... Elke virtuele machine die zich bevindt op de host machine zal dan gebruik maken van deze middelen. Doordat virtualisatie alomtegenwoordig werd in datacenters heeft het ervoor gezorgd dat de meeste logica ook meer kwam te liggen bij de hypervisor. Cloud computing komt zeer sterk opzetten en dit zorgt ervoor dat de meeste mensen niet in contact komen met hypervisors.

3.1.1 Hosted Hypervisors

Een hosted hypervisor zal zich bevinden op een het besturingssysteem van de host machine en heeft geen directe toegang tot de hardware. Dit heeft als voordeel dat de hardware niet zo belangrijk is maar zorgt voor een extra laag tussen de hardware en de hypervisor. Een goede regel is: "hoe minder lagen we hebben, hoe beter de performantie."Wanneer je een Ubuntu instantie hebt in een virtuele machine dan zal de computer de hypervisor zijn. De computer kan dan de virtuele machine beheren en veranderingen uitvoeren.

Voorbeelden van een hosted hypervisor zijn: Virtualbox en VMware Workstation.

3.1.2 Bare-metal Hypervisors

Een alternatief is een bare-metal hypervisor. Hierbij is er geen extra laag tussen de hardware en de virtuele machine. We hebben geen host besturingssysteem nodig omdat de hypervisor zich rechtstreeks bevindt op de hardware. Dit zorgt voor betere performantie, schaalbaarheid en stabiliteit.

3.2 Hardware Virtualization

De virtuele machine is een toepassing van de virtualisatie van de hardware. Een virtuele machine bevindt zich op een fysieke machine door gebruik te maken van een hypervisor. De nadelen van een virtuele machine als de kleinste eenheid is de schaalbaarheid en de veiligheid. Wanneer je je eigen server opstelde dan moet je een besturingssysteem kiezen. Meestal gaan we kiezen voor een Linux distributie. Daarna moet je de software installeren die ervoor zorgt dat de applicatie kan werken. Daaropvolgend moet je je server beveiligen. Het veranderen van de SSH-poort, zorgen dat de root-user niet te bereiken is. IPtables en firewalls opstellen. Voor dit te vergemakkelijken kunnen we gebruikmaken van een configuratie management tool (Chef, Puppet, Ansible...).

Meest bekende voorbeelden van bare-metal hypervisors zijn VMware ESXi, Microsoft Hyper-V en Xen.

3.3 Operating System-level Virtualization

Naast hardware virtualisatie kunnen we ook gebruikmaken van de virtualisatie van het besturingssysteem. Bij deze toepassing van virtualisatie gaan we de mogelijkheden van de kernel van het besturingssysteem gebruiken. De kernel van bepaalde besturingssystemen laat ons toe om meerdere geisoleerde user spaces tegelijkertijd te laten werken. Dit zorgt ervoor dat de dat er maar 1 besturingssysteem en een kernel zijn. De verschillende user spaces maken gebruik van CPU, geheugen en netwerk van de host server. Elke user space heeft zijn eigen configuratie omdat de user spaces op zichzelf staan en geisoleerd zijn de andere user spaces. Tegenover hardware virtualisatie zal de besturingssysteem virtualisatie minder gebruik maken van het geheugen en de CPU omdat er maar 1 kernel is en niet meerdere besturingssystemen. Het opstarten neemt maar een fractie van de tijd in beslag tegenover virtuele machines. Deze user spaces worden ook wel containers genoemd.

Hoofdstuk 4

Containers

4.1 Containers versus virtuele machines

Het heeft een tijd geduurd vooraleer containers op de voorgrond treden. Het maken van applicaties werd meestal op de eigen machine gedaan omdat het opzetten van een werkomgeving soms te moeilijk was. Er was natuurlijk vagrant die het stukken gemakkelijker maakte maar dan nog voelde het aan als of er nog iets beter aankwam.

Lange tijd werd vagrant aangeraden omdat het opzetten van een werkomgeving veel tijd in beslag nam en de meeste applicaties eerder een monolitische structuur hadden. Hierbij bedoelen we een grote applicatie die alles doet tegenover verschillende applicaties die elk 1 ding doen.

Containers was het antwoord op veel van de moeilijkheden van de virtuele machine: het beter gebruiken maken van resources, eenduidige ontwikkelingsomgeving en vlugger kunnen ontwikkelen van applicaties. Dus we hebben twee mogelijkheden om applicaties te maken en te laten werken. Ofwel gebruiken we virtuele machines als de kleinste eenheid ofwel containers.

De container gebruikt resources van het besturingssysteem van de host. Dit zorgt ervoor dat een container vlugger kan gestart worden want het besturingssysteem moet niet opgestart worden. Bij virtuele machine moet men wachten tot het besturingssysteem opgestart is.

De omvang van virtuele machines tegenover containers is al besproken hiervoor. Een gevolg hiervan is dat men meerdere containers naast elkaar kan laten werken. De virtuele machine waarop de docker container werken kunnen dan beslissen hoeveel resources er aan wie worden gegeven.

Voor containers was het opzetten van een complexe applicatie met tientallen dependencies een heel moeilijk gegeven. Bij een container moet je gewoon de image downloaden en gebruiken. In deze container is alle configuratie al aanwezig en kan men gewoon starten zonder veel tijd te verspelen aan het configureren en installeren

van de applicatie en het besturingssysteem.

Omgekeerd kan men ook denken dat de developers niet meer moeten nadenken over het ondersteunen van meerdere systemen. 1 Container image kan gebruikt worden voor iedereen en als men veranderingen wil aanbrengen dan kan men verder op de container bouwen.

Als men wilt schalen met virtuele machines dan moet je meerdere besturingssystemen opzetten die worden beheert door een hypervisor. Elk besturingssysteem heeft een applicatie en zijn dependencies.

Bij containers delen ze het besturingssysteem en worden beheert door een container engine die zich bevindt op het niveau van het besturingssysteem. Ze delen ook een de kernel van de host en daarbij kan er ook voor gezorgd worden dat gemeenschappelijke dependencies gedeeld kunnen worden over meerdere applicaties. De container engine vervuld dezelfde functie als de hypervisor bij virtuele machines.

Veiligheid kan een probleem zijn bij containers omdat ze nog niet lang in productie worden gebruikt. Hypervisor zorgen voor extra veiligheid bij virtuele machines omdat ze battle-tested zijn. Ze worden al gebruikt voor lange tijd door grote bedrijven die beveiliging hoog in het vandaag dragen.

4.2 Docker

Onderdelen van het begrip container bestonden al onder diverse vormen. FreeBSD had jail(1998). Google was begonnen met het ontwikkelen van cGroups voor de linux kernel. Het Linux Containers projects bracht veel van deze technologieën samen om te zorgen dat containers een realiteit konden worden.

Docker was het bedrijf dat de al de delen samenbracht onder een mooie interface en uitgebreid ecosysteem. Het werd veel gemakkelijker om containers te maken en distribueren. Docker zorgde ervoor dat veel van moeilijkheden van containers verdwenen of veel kleiner werden. Door veel van de componenten van het ecosysteem open te stellen konden ze rekenen op de steun van vele developers. Deze steun kon zijn in de vorm van het melden van bugs of het uitbreiden of verbeteren van huidige componenten.

Er waren andere formaten dan Docker zoals Rocket die het mogelijk maakte voor containers te maken. Maar door hun ecosysteem en de hulp van de open source gemeenschap hebben zijn ze de standaard geworden voor om containers te maken en te verspreiden.

In 2015 werd het Open Container Initiative opgericht. Veel van de grote spelers op vlak van containers zoals Docker, CoreOS, Microsoft en Google maken hier deel van uit. Ze willen een standaard voor containers vastleggen.

Hoofdstuk 5

Unikernels

5.1 Inleiding

Unikernels bestaan al lang onder verschillende vormen. Je kan dit zeggen van de meeste technologieën die zorgen voor grote veranderingen: eerst waren een paar projecten die uiteenlopende standaarden gebruikten en na verloop van tijd kwamen die samen. Het experimenteren met nieuwe software of in het algemeen met ideeën zorgt voor innovatie.

De eerste implementaties komen we tegen op het einde van de jaren 1990. Exokernel werd ontwikkelt door MIT en wou ervoor zorgen dat er zo weinig mogelijk abstractie de developers op te leggen en dat ze zelf over de abstractie moeten beslissen. Nemesis werd dan weer vanuit University of Cambridge ontwikkeld. Zij hadden eerder multimedia als het doel in hun achterhoofd.

5.2 Kernel

De kernel is het programma dat zich centraal bevindt in de computer. Het werkt rechtstreeks met de hardware van de computer. De kernel kan gezien worden als het fundament waar het hele besturingssysteem op steunt. Omdat het zo een belangrijke rol vervult in de computer is veel van het geheugen van de kernel beveiligd zodat andere applicaties geen veranderingen kunnen aanbrengen. Als er iets fout zou gaan met de kernel dan heeft dit rechtstreeks gevolgen op het besturingssysteem. Al de handelingen die de kernel uitvoert bevinden zich in de kernel space. Daartegenover hebben we alles wat de gebruiker uitvoert gebeurd in de user space. Het is van uiterst belang dat de kernel space en user space strikt van elkaar gescheiden zijn. Als dit niet zo zou zijn dan zou een besturingssysteem en tevens de computer onstabiel en niet veilig zijn. De kernel voert nog andere taken uit zoals memory management en system calls.

Als een applicatie wordt uitgevoerd dan bevind die zich binnen de user space. Om het programma in werkelijkheid te kunnen uitvoeren moet men toestemming vragen

aan de kernel om deze instructies van het programma realiseren. Deze instructies moeten worden nagegaan of ze wel veilig zijn. Soms spreken we ook van memory isolation waarbij de user space en kernel space niet rechtstreeks met elkaar kunnen communiceren. Dit is voor nog veiliger te zijn. Om een applicatie uit te voeren moet er veel communicatie gebeuren tussen de onderdelen. Deze onderdelen kunnen zo laag als de hardware gaan tot de applicatie zelf.

5.3 Library besturingssystemen

Het meest gebruikte besturingssysteem waarop een web applicatie zich bevindt is een Linux besturingssysteem. Het meest bekende besturingssysteem voor servers is Ubuntu. Daartegenover hoor je ook dat Ubuntu zich wil begeven naar het grote publiek. Een gratis alternatief voor Windows zowaar. De editie dat je gebruikt voor je web applicaties is niet helemaal dezelfde als het windows-alternatief.

Maar als we er toch bij stilstaan dan zijn er grote delen van een besturingssysteem die we niet nodig hebben voor grote applicaties. Doorheen de geschiedenis zijn er veel verschillende onderdelen nodig geweest om goed gebruik te kunnen maken van een besturingssysteem. Na een tijd zijn sommige onderdelen niet meer nodig. Het verwijderen van deze onderdelen brengt moeilijkheden mee. Sommige gebruikers hebben nog juist dat onderdeel nodig terwijl andere het niet meer nodig hebben. We kunnen ook niet zomaar delen verwijderen. Dit kan ervoor zorgen dat andere delen niet meer werken.

Er is een trend binnen containers dat je een container moet gebruiken die alles heeft wat je nodig hebt maar niets meer. Hoe kleiner de container hoe beter. Alpine is een Linux besturingssysteem dat zeer klein is (5 MB) maar beschikt over een package repository die zeer uitgebreid is. Dit maakt het het ideale besturingssysteem voor containers. Je kan starten met een kleine basis en alle onderdelen toevoegen die je nodig hebt. Dit zorgt voor betere performantie en een kleinere container.

Time-sharing is een onderdeel dat alleen nog maar nodig is in uitzonderlijke gevallen. Time-sharing zorgt ervoor dat de systeem en hardware componenten van de gebruiker worden afgeschermd. Een gebruiker die een programma liet werken zou een andere programma gestart door een andere gebruiker kunnen laten stoppen door een bepaalde instructie uit te voeren. Dit probleem is bijna niet meer voorkomend omdat hardware zo goedkoop is geworden en iedereen zijn eigen computer heeft.

Niet alleen hebben sommige delen geen nut meer maar sommige delen zullen ervoor zorgen dat de performantie van een besturingssysteem gehinderd wordt voor bepaalde taken. Het is dezelfde analogie als het gebruiken van een hamer voor alle klusjes die je moet doen. Het is niet omdat een hamer goed is om spijkers in hout te slaan dat je het moet gebruiken voor een boom om te zagen.

Als we kijken naar het voorbeeld van Alpine binnen containers kunnen we het

dezelfde weg opgaan. Library besturingssystemen neemt dit nog een stap verder. We starten dus van een nog lagere basis en we voegen alleen de delen toe die we nodig hebben. Library duidt op functionaliteit die je kan gebruiken in verschillende programma's of contexten.

Als we een web applicatie gaan bouwen dan moeten we kunnen communiceren met internet. Hiervoor bestaan verschillende netwerkprotocollen die je kan gebruiken. Wij hebben TCP nodig om te communiceren met internet. Bij alledaagse besturingssystemen zoals Ubuntu is dit al aanwezig. Omdat we starten vanaf nul moeten we deze zelf implementeren. Gelukkig zijn er libraries die we hiervoor kunnen gebruiken. Het verschil zit hem in de grote van het uiteindelijke besturingssysteem en zijn grotere varianten. De libraries en de applicatie worden dan gecompiled met de configuratie. Als resultaat heb je dan 1 binary die rechtstreeks op een hypervisor of de hardware kan werken. Je compiled de applicatie voor de omgeving waar hij zal werken en enkel daar zal hij werken. Dit is een verschil tegenover een container die minder afhankelijk is van de omgeving.

Doordat we zelf een applicatie opbouwen met enkel wat de applicatie nodig heeft zorgt dit voor een kleinere attack surface. De code dit uitenindelijk terecht komt op de server is veel minder groot.

5.4 Single Address Space

Wat een unikernel nog uniek maakt is dat de kernel geen concept heeft van een user space en kernel space. Alle processen bevinden zich dus in dezelfde omgeving. Dit zou problemen geven bij traditionele besturingssystemen. Maar we compileren de applicatie en zijn libraries en controleren of dat er zich geen problemen kunnen voordoen. Een voordeel dat we kunnen aanhalen is dat er geen communicatie moet gebeuren tussen de user space en de kernel space omdat ze niet bestaan. We zullen sowieso in problemen lopen wanneer we meerdere applicaties naast elkaar laten lopen of applicaties met veel functionaliteit. Dit is ook niet de manier waarop men unikernels moet gebruiken. In de laatste secties van dit hoofdstuk zullen we dit concept nog verder aanhalen.

5.5 Veiligheid

Het grootste verkooppunt van unikernels is veiligheid. Vulnerabilities en veiligheidsrisico's kunnen zorgen voor grote schade. De gebruikers vertrouwen hun informatie aan ons en wanneer dit vertrouwen geschaad word is dit niet goed. Web applicaties moeten veilig zijn maar andere sectoren zoals de banksector maken een obsessie van veligheid.

Doordat cloud computing zo hard op de voorgrond treedt, komen er veel meer applicaties op het internet. Meestal zijn deze appliacties niet zeer goed beveiligd. Een

bijdrage daarbij is zeker ook dat de drempel veel lager wordt.

Veiligheid is soms zeer moeilijk te vinden wanneer we grote besturingssystemen gebruiken als basis voor onze applicaties. Deze besturingssystemen hebben zodanig veel onderdelen dat het heel moeilijk wordt om deze allemaal te controleren. Daar komen ook nog de combinaties van deze onderdelen bij.

Containers gebruiken een besturingssystemen als basis en door dit gebruiken ze ook de Linux kernel. De Linux kernel is niet resistent tegen bepaalde onveiligheden. Daartegenover starten unikernels met een heel minimaal gegeven en voegen toe wat de applicatie nodig heeft. Het schrijven van een exploit is ook veel moeilijker voor een unikernel omdat elke unikernel anders in zit.

Wanneer je bijvoorbeeld toegang hebt tot een container kan je gemakkelijk toegang krijgen tot de shell van de container en vanalle kattenkwaad uithalen.

Wanneer een hacker binnendringt op een virtuele machine die een traditionele stack heeft dan kan hij tools installeren door de package manager te gebruiken en andere commando's om meer informatie te verzamelen. De server kan misschien geinfecteerd worden met een virus. Dit scenario is veel moeilijker bij unikernels omdat veel van commando's weggenomen zijn en er geen package manager aanwezig is. De hacker zou al heel veel werk moeten leveren om de unikernel te infecteren omdat hij zelf de vulnerability zou moeten schrijven. Dan blijft er enkel nog de hypervisor over om te misbruiken. De onveiligheden van hypervisors zijn veel moeilijker uit te buiten omdat ze al een paar decennia in gebruik zijn van veel grote bedrijven.

Uitbreiden over TLS?

5.6 Just in Time

Network latency is een groot probleem wanneer de meeste services die u aanbied zich in de cloud bevinden. Dit maakt de applicatie zeer afhankelijk is van de server. Wanneer de connectie met de server wegvalt kan de applicatie in uitzonderlijke gevallen nog werken of helemaal niet meer. De applicatie is rechtstreeks afhankelijk van de connectiviteit met de server. Zelf een slechte connectie kan zorgen voor problemen. De oorzaak van een slechte connectie kan een oorzaak van de server of de applicatie. We gaan er vanuit dat de applicatie en server goed geschreven zijn en dat er geen grote problemen liggen bij de infrastructuur.

De locatie van je servers tegenover de locatie van je gebruiker kan zorgen voor problemen. Je infrastructuur verspreiden over verschillende datacenters en nadenken over hun locatie kan hierbij een goede actie zijn. Maar wanneer je resources verbruikt in een datacenter dat weinig gebruikt wordt kan dit duur uitkomen. Het verbruik van de resources aanpassen aan de load van de server is een stap die al genomen wordt door vele cloud providers. Dan nog moeten er nieuwe instanties van jouw applicaties of servers worden opgestart. Dit alles kan meer dan een halve minuut duren. Sommige

soorten applicaties, zoals IOT en anderen, hebben constante connectiviteit nodig. Dit kan van zeer groot belang zijn. Unikernels kunnen hierbij helpen.

Het opstarten van een virtuele machine en alles in orde brengen om te werken kan 1 minuut of langer duren. Bij containers wordt dit veel minder omdat er geen besturingssysteem van nul moet worden opgestart. Unikernels neemt dit nog een paar stappen verder omdat men gewoon een binary moet uitvoeren op de hypervisor of hardware. De opstarttijd zal meestal minder dan een seconde innemen. Dit geeft ons een betere manier om de capaciteit aan te passen aan de load. Het opstarten van een unikernel kan sneller gebeuren dan de request naar de server toe. Kosten zullen hierdoor lager liggen.

5.7 Andere voordelen

De binary die gemaakt wordt wanneer de unikernel gecompileerd word zal zeer klein zijn. Omdat verschillende drivers zelf geïmplementeerd worden. Er moet ook niet in dezelfde mate als het besturingssysteem rekening gehouden worden met andere drivers. De hypervisor zorgt ook voor een stabiele interface en deze zorgt ervoor dat we geen uitgebreide drivers moeten schrijven om te zorgen voor compatibiliteit voor de verschillende hardware apparaten.

5.8 Productie

Veel van de commentaar op unikernels komt van de onmogelijkheid om in productie te bekijken wat er fout aan het gaan is bij een unikernel. Voor de developers is het soms gemakkelijker om een applicatie aan te passen in productie om te zien of een bepaalde bug wordt verholpen. Sowieso is dit geen best practice. Er moet uitgebreid getest worden in development cycle om fouten/bugs tegen te gaan. Wanneer er dan toch iets fout gaat in productie dan zal men de situatie proberen na te bootsen en dan applicaties aan te passen om zo de fout/bug op te lossen. Het terugzetten van een oudere versie van de applicatie kan helpen om de gebruikers geen ongemak te voorzien en tevens meer tijd te hebben om bepaalde bugs op te lossen.

5.9 Implementaties van moderne unikernels

5.9.1 ClickOS

Deze implementatie van een unikernel komt van Cloud Networking Performance Lab. Ze zijn al langer bekend omdat ze een minimalistisch besturingssysteem hebben ge-

maakt voor de Xen hypervisor: MiniOS. Vanuit de dezelfde denkwijze als unikernels starten ze met het minimale en kan je de onderdelen die je nodig hebt.

Elke implementatie van een unikernel zal een bepaalde omgeving kiezen waar het op zal werken. We hadden al eerder gezien dat we een binary zullen compileren voor 1 omgeving omdat dit het proces veel gemakkelijker maakt. De omgeving waar ClickOS op focust is de Xen hypervisor die we al eerder hebben aangehaald.

De applicaties waarvoor ClickOS wordt voor gebruikt zijn middleboxes. Een middlebox is een netwerk apparaat dat netwerktrafiek kan omzetten, filteren, inspecteren of manipuleren. Voorbeelden hiervan zijn firewalls en load balanceres.

ClickOS is gebaseerd op MiniOS. We starten met een modulaire software router waarop je onderdelen kan toevoegen. Vroeger zat de laag die alles regelde met de netwerk trafiek eerder bij de hardware, maar je kan dus ook je eigen implementatie schrijven om veel van functionaliteit van de hardware over te nemen. Dit zorgt voor een implementatie die veel efficiënter is. De resulteerde unikernel is rond de 6MB groot, starten op rond de 30 milliseconden en kunnen snel gebruikt worden 45 microseconden. Bij snel gebruikt bedoel ik dat ze worden toegevoegd om netwerk trafiek te verwerken.

De taal waarin de unikernels die gebruik maken van ClickOS is C++. Bijna alle unikernels gebruiken 1 programmeertaal om hun unikernel in te schrijven.

Hun documentatie over ClickOS, MiniOS en toolchain om deze unikernels te maken is zeer uitgebreid.

5.9.2 Clive

Clive is een besturingssysteem gemaakt voor te werken in gedistribueerde en cloud computing omgevingen. Het heeft veel geleend op vlak van ontwerp van Plan 9.

Plan 9 wordt ontwikkeld door Bell Labs en dit is gestart vanaf de jaren 80. Het nut van Plan 9 is vooral te vinden in gedistribueerde systemen. Het is een besturingssysteem kernel in de eerste plaats maar ook een collectie van libraries. Elk proces bevindt zich in zijn eigen name space en staat zo los van de andere processen. Deze namespace heeft ook de mogelijk om een connectie te maken met een andere Plan 9 server en zo data uit te wisselen. Dit zorgt ervoor dat het maken van gedistribueerde systemen veel logischer word.

Clive is een besturingssysteem gemaakt door Lsub. Ze gebruiken Go als programmeertaal voor de applicaties. Golang maakt het gemakkelijk voor meerdere instanties van een applicatie naast elkaar te laten werken terwijl ze met elkaar kunnen communiceren.

Om Go te laten werken werken in een unikernel moeten er een paar aanpassingen worden gedaan. De Go compiler en runtime werden aangepast voor een betere interface open te stellen voor netwerk applicaties. Hiervoor zou de interface van channels moeten worden aangepast om bepaalde moeilijkheden tegen te gaan wanneer men

communiceert over het netwerk. Ook moet de applicatie wanneer ze gecompileerd is, zijn eigen kernel zijn.

Als resultaat krijgen we een unikernel die we kunnen laten werken rechtstreeks op de hardware en andere besturingssysteem.

5.9.3 HaLVM

HaLVM wordt ontwikkeld door Galios. Zij zijn een Amerikaans software development agency dat unikernels al een tijd in productie gebruikt in bepaalde use cases.

De programmeertaal waarin de unikernel van HaLVM wordt geschreven is haskell. Haskell is een functionele programmeertaal met een zeer uitgebreid type system.

Het werd ontwikkeld met als doel voor besturingssysteem componenten te testen. Maar na een tijd werd het mogelijk om het te gebruiken voor meer use cases.

Bij HalVM gaan we terug Xen als omgeving gebruiken. Er is een integratie met de Xen hypervisor waarop de core library van HaLVM op rust. Er bestaat ook een communications library die bestaat uit Haskell File System en Haskell Network Stack. Deze library kan gebruikt in de meeste gevallen als je een netwerklaag nodig hebt. Als we meer mogelijkheden nodig hebben voor de applicatie dan gaan we modules toevoegen. Ze hebben verder een ecosysteem uitgebouwd om het gemakkelijker te maken voor developers om hun eigen modules te bouwen.

Zoals in de meeste gevallen moet de compiler van Haskell worden aangepast om rechtstreeks te kunnen werken op de Xen hypervisor. Het is ook geen probleem om andere Haskell libraries in de code te gebruiken. Er kunnen soms wel problemen optreden wanneer de compiler zelf wordt aangepast.

HalVM heeft zelf uiterst goede documentatie met een groot aantal voorbeelden. Die voorbeelden zijheel gemakkelijke applicaties tot beginnende complexe applicaties.

Het wordt gebruikt door Galios in productie en dit maakt het gemakkelijk om vragen te stellen. De GitHub repository waar de applicatie zich op bevindt is redelijk actief.

5.9.4 Erlang on Xen

Xen is 1 van de meest gebruikte hypervisors en daarom gaan veel unikernels zich richten om die omgeving. Erlang on Xen is daarbij geen uitzondering.

Erlang on Xen is niet alleen een unikernel. De grote visie is dat ze Erlang infrastructuur willen ontwikkelen die een unikernel gaat opstarten als een request binnenkomt. Zo kunnen ze pieken in gebruik opvangen. Dit was het Just in Time gedeelte dat we al hebben behandelt in 1 van de vorige secties.

De meeste bedrijven of werkgroepen die aan unikernels werken gaan zich enkel werpen op de unikernel niet op de architectuur er rond. Erlang on Xen wil een platform zijn dat unikernels en JIT mogelijk maakt zonder veel problemen.

LING is het besturingssysteem en tegelijk het platform. LING is een erlang VM die herschreven is met unikernels als hoofdgedachte. LING zal dus werken op Xen en een unikernel opstarten wanneer er een request binnenkomt. Wanneer de content dat de unikernel moet verzenden is verzonden zal de unikernel worden vernietigd.

De focus van Erlang on Xen was xen in het begin. Met het uitbrengen van LING is het mogelijk geworden om ports te maken voor andere omgevingen. Dat heeft het mogelijk gemaakt om het te laten werken op ARM. De omgeving ARM is vooral te vinden in IOT en mobiele applicaties. Unikernel kunnen handig zijn op deze servers omdat ze soms te klein zijn voor een groot besturingssysteem. De ruimte die je wint bij een unikernel kan hierbij helpen. Verder opent dit ook de mogelijkheid voor de unikernels van LING op bare-metal te laten werken.

5.9.5 Rumprun

Rumprun gebruikt rump kernels om een unikernel te maken. Deze rump kernel wordt samengesteld uit componenten afkomstig van NetBSD. NetBSD is een groter besturingssysteem maar zit modulair in elkaar dus dit kunnen we gebruiken om een rump kernel te maken. De rump kernel wordt dan samen met de applicatie verpakt om gebruikt te kunnen worden in verschillende omgevingen.

De toolchain rond de rumprun kernel maakt het gemakkelijker om je applicatie te maken of om te zetten naar een unikernel.

Er zijn verschillende soorten unikernels. Sommige unikernels specialiseren op basis van programmeertaal en andere op basis van omgeving. Sommige doen zelf beide. Rumprun is hierbij flexibel. Het kan werken op de meeste besturingssystemen die een POSIX applicatie interface openstellen en bijna alle besturingssystemen hebben dit soort interface.

Er wordt ook aangehaald in hun wiki, dat wanneer je een applicatie hebt die specifiek geschreven is om op een programmeertaal-gebaseerde unikernel en jouw omgeving ondersteunt, dat je best die oplossing gebruikt. Als dit niet het geval is dan kan een rumprun kernel een goede oplossing zijn.

De rump-run packages zijn implementaties van drivers en technologieën die je kan toevoegen aan rumprun kernel. Er zijn een groot aantal packages die je kan gebruiken en de meest bekende zijn zeker aanwezig. Het spijtige is wel dat er nog geen packaging systeem aanwezig is. Dit zou er wel voor zorgen dat er gewerkt kan worden met dependencies en versies van packages. Een packaging systeem maken is natuurlijk geen simpele opdracht en moet eerst goed uitgedacht worden voor het uitgebracht word.

De programmeertalen die ondersteund zijn, zijn de volgende: C, C++, Erlang, Go, Javascript(node.js), Python, Ruby en Rust. De keuze tussen de programmeertalen zorgt voor een groot voordeel tegenover andere unikernels.

5.10 Hedendaags gebruik

Unikernels kunnen gebruikt worden in uiterst uitzonderlijke use cases momenteel. Hetzelfde zal je vinden bij containers vooraleer zij op de voorgrond treden. Elke technologie zal dit ondergaan omdat iets gemakkelijker maken en veralgemenen tijd kost. Maar we hebben gezien uit de geschiedenis van besturingssystemen dat we sommige concepten niet moeten blijven gebruiken. Als unikernels een groter publiek kunnen dienen zonder hun voordelen te verliezen dan zou het een groot succes kunnen worden.

Hoofdstuk 6

Architecture in een wereld van unikernels

6.1 Microservices

Er is ook een andere evolutie aan de gang maar dan eerder binnen de architectuur van applicaties. De laatste 20 jaar werd er meestal 1 grote applicatie gemaakt die alle functionaliteit op zich nam. Deze applicatie is gemakkelijk te maken omdat men steeds verder bouwt op vorige functionaliteit. Het probleem bij deze manier van werken is de schaalbaarheid van een applicatie. Wanneer een bepaald deel van een applicatie een bottleneck wordt dan bestaat de mogelijkheid om deze te herschrijven en te optimaliseren. Maar soms moet een deel van een applicatie gewoon veel kunnen verwerken dan een ander deel. Microservices is een concept dat al langer bestaat maar dit is de uitgesproken oplossing voor dit fenomeen. De applicatie opsplitsen in componenten met 1 verantwoordelijke zorgt ervoor dat het veel beter te schalen is. Als je een applicatie hebt met al de functionaliteit in dan zou je een nieuwe virtuele machine moeten opzetten. Dit is niet schaalbaar. Bij microservices moet je gewoon een paar nieuwe instanties van een component starten. Dit vraagt natuurlijk wel om een totaal nieuwe manier van werken. Microservices zijn ook meer op de voorgrond gekomen door het gebruik van containers want zij maken het simpeler om een architectuur op te zetten die gebaseerd is op microservices.

Dit zal ook wel een moeilijkheid meebrengen dat het een puzzel is die in elkaar moet passen. Je moet goed doordacht te werk gaan en je visie naar de toekomst toe ook in het achterhoofd houden. Het geeft het ontwikkelingsteam wel de vrijheid om nieuwe technologieën die uitkomen veel vlugger te gebruiken. Wanneer je iets nieuw wilt gebruiken in een monolithische applicatie dan moet je al een groot stuk herschrijven wat grote kosten met zich meebrengt. Dit zorgt voor wendbaarheid in een omgeving die zeer competitief is.

Met containers kan je ook een microservice architectuur uitbouwen. Dit zorgt ervoor dat er meer kennis zal zijn over de moeilijkheden en de mogelijkheden die deze manier van werken met zich meebrengt.

6.2 Immutable Infrastructure

Unikernels vraagt ook een verschuiving van de manier dat we over infrastructuur denken. Immutable infrastructure is een opkomende gedachtengang. We gaan geen enkele verandering aanbrengen aan applicaties die zich in productie bevinden. Als er een probleem is dan gaan we een nieuwe unikernel maken en deze in productie laten treden terwijl de andere wordt neergehaald. Je kan dit al zien bij sommige cloud providers die werken door middel van een git push om de voormalige versie te vervangen. Dit zorgt voor een betere veiligheid en de hele cyclus van development naar productie wordt veel kleiner.

We gaan dan ook meer naar het model van rolling updates. Rolling updates wordt al gebruikt bij sommige Linux distributies. Hierbij gaan we niet spreken van grote updates waarbij er een lijst van functionaliteit wordt uitgebracht op 1 moment. We gaan updates op gelijk welke tijd uitbrengen zonder de distributie te breken. Er gaan dus meer kleinere updates zijn en veiligheidsmaatregelingen kunnen veel vlugger getroffen worden.

We moeten opletten dat we niet zonder nadenken nieuwe systemen opzetten dit kan leiden tot een hoop die heel moeilijk op te ruimen.

Een nieuw gegeven is ook dat we infrastructuur meer zullen benaderen zoals we programmeren. Dit doen we door de infrastructure uitermate te testen en herhaalbare patronen te gebruiken. Ook zullen we de configuratie van servers bijhouden in version control. Door dit te doen kunnen we een beeld krijgen wat er gebeurd met de infrastructuur door de tijd heen. Dit geeft de devOps de mogelijkheid om problemen terug te leiden naar 1 verandering. Het is belangrijk dat er steeds kleine veranderingen gebeuren zodat we problemen gemakkelijker kunnen herleiden naar 1 oorzaak. Gebruik maken van version control zal niets uitmaken wanneer je tientallen veranderingen bundelt in 1 verandering.

Een trend dat we zeker zien en dat zich zal blijven doorzetten is het lenen van programmeer principes in de wereld van infrastructuur. Dit komt doordat het programmeren het meest geëvolueerd is door de jaren en ook met grotere problemen meestal heeft moeten werken.

Hoofdstuk 7

Orchestration tools

Hoofdstuk 8

Experimenten

Hoofdstuk 9

Conclusie

Bibliografie

Lijst van figuren

Lijst van tabellen