APPLICATIE CONTAINERS

EEN NIEUWE MANIER VOOR HET BOUWEN, INPAKKEN EN UITROLLEN VAN APPLICATIES

Stefan van Gastel, Ontwikkelaar Innovatie en John Stroosnijder, Innovatie Manager bij JIVC/KIXS

Dit artikel gaat over containers. Nee, we hebben het hier niet over de transportwereld, maar echt over ICT. Sinds een paar jaar is de applicatie container technologie namelijk sterk in opmars, met Docker als de bekendste container software. Met container technologie worden applicaties op een hele slimme manier gebouwd, ingepakt en uitgerold in het netwerk. De techniek komt grotendeels uit de keuken van Google en de basis (control groups[1]) is al 10 jaar geleden in Linux geïntroduceerd. Inmiddels is het ook op Windows servers beschikbaar. Maar wat is het nu precies en hoe kan het ons helpen? ➔

Containers in een notendop

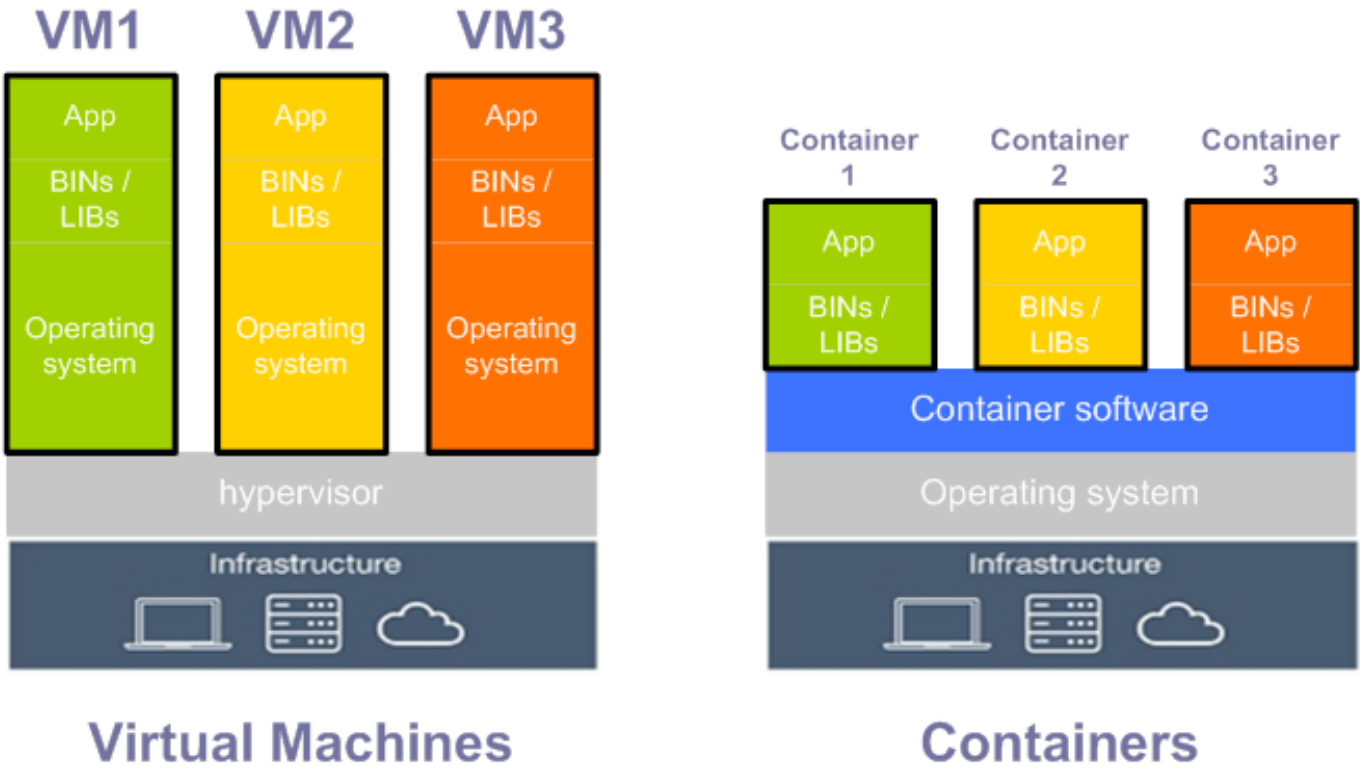
Containers zijn gebaseerd op operating-system-level virtualization[2]. Dit is een techniek waarbij één of meerdere processen geïsoleerd kunnen draaien op een gedeeld onderliggend besturingssysteem. Een container krijgt flexibel en naar behoefte virtuele CPU en memory resources toebedeeld van het onderliggende besturingssysteem op basis van zogenaamde control groups (cgroups) en daarbij draait de container in zijn eigen wereld, de namespace. Een container wordt zo opgebouwd dat alle afhankelijkheden in de vorm van binaries en libraries worden gebundeld. Deze bundel kan op ieder besturingssysteem met ondersteuning voor containers worden gestart.

Verschillen containers en virtuele machines

Operating-system-level virtualization is anders dan servervirtualisatie[3]. Servervirtualisatie, waarbij wordt gewerkt met virtual machines (VMs), virtualiseert op het niveau van hardware. Hierbij wordt een speciale software, genaamd hypervisor, gebruikt. Figuur 1 geeft de verschillen weer tussen op hypervisor gebaseerde virtualisatie met virtuele servers ten opzichte van operating-system-level virtualization met containers. Zoals figuur 1 weergeeft zit er een verschil in een aantal lagen. Bij virtuele machines draaien er meerdere virtuele servers, inclusief besturingssysteem op een hypervisor.
Dit kan bijvoorbeeld met VMware vSphere, wat nu zowel binnen MULAN als Titaan wordt gebruikt.

De hypervisor zit tussen de virtuele servers en de hardwarelaag in en beheert de onderliggende hardware. Door tussenkomst van de hypervisor denkt iedere VM dat het eigen hardware heeft en iedere VM is geïsoleerd van de andere VMs. Het besturingssysteem in de VM is 100% dedicated voor de VM. Daarom is een VM relatief groot en het besturingssysteem moet onderhouden worden met (security) patches zoals bij iedere reguliere server.

Door disk opslag toe te kennen aan een VM, kan feitelijk alles gedaan worden wat met een fysieke server kan.
Een container wordt op een andere manier gevirtualiseerd. De containers draaien op een normaal besturingssysteem. Alle containers delen dit besturingssysteem, echter de containers zien elkaar niet. Ze draaien namelijk in hun eigen wereld (de namespace) en werken daar in een tijdelijk bestandsysteem waarin ze een kopie van het onderliggende besturingssysteem zien. Een container schrijft updates van data in deze tijdelijke ruimte.
Bij een herstart van de container is de gewijzigde data weg, omdat de tijdelijke ruimte vluchtig is.



Figuur 1

Statefull vs stateless

De verschillen tussen VMs en containers zijn vooral gericht op de ‘state’ van de machines. VMs zijn over het algemeen statefull, ze houden data vast en hebben een eigen besturingssysteem aan boord. Dit maakt VMs geschikt voor allerlei software die data wil opslaan, inclusief databases. Op het gebied van statefull applicaties is een hypervisor zoals VMware vSphere

uitermate geschikt, omdat deze oplossing helemaal is gericht op hoge beschikbaarheid van statefull applicaties en de data.
Containers daarentegen zijn standaard stateless en slaan data niet permanent op. Ze hebben geen eigen besturingssysteem aan boord. Containers zijn uitermate geschikt voor applicaties die zelf geen data opslaan, zoals een webapplicatie of de back-end van een mobiele app die data alleen maar weergeeft of een app die data doorgeeft aan een ander systeem dat wel statefull is. Dit stelt daarom andere (lagere) eisen aan de onderliggende infrastructuur. Hier past een virtualisatie of cloud oplossing zoals Openstack[4] beter bij dan de hypervisors die zijn ontworpen voor statefull applicaties.

Figuur 2

Het bovenstaande is overigens geen wetmatigheid: je kunt een container statefull maken. Dit kan opgelost worden door de container naar externe opslag te laten schrijven, zoals een netwerkschijf, object storage of een externe database. Daarnaast zijn er databases die juist prima in een container kunnen werken, omdat ze data distribueren over verschillende instances van de database, zelfs over meerdere locaties indien gewenst.

Deze ontwikkelingen zijn echter nog vrij nieuw en moeten eerst onderzocht worden. Statefull werken met containers vereist nu nog extra aandacht. Daarom zijn voor de korte termijn ook combinaties van statefull en stateless heel goed denkbaar. Bijvoorbeeld bij een zogenaamde two-tier app, waarbij er sprake is van een stateless webservice als front-end die prima in een container kan draaien, met een statefull database in de back-end die op een VM kan draaien.

Praktisch gebruik van containers

In de afgelopen jaren zijn de applicaties veel meer webbased geworden. De aanpak met cliënt-server oplossing wordt daarbij vanwege de complexiteit op het gebied van upgrades verlaten. De back-end van de moderne webbased applicaties draait in het datacenter in de vorm van een webservice.
Daarnaast is steeds vaker sprake van mobiele apps die veelal onder water met een webservice in de back-end communiceren.
Daarnaast zien we ook steeds meer microservices. In plaats van één groot softwarepakket met tientallen functies, wordt dit opgesplitst in tientallen kleinere microservices met ieder een specifieke taak. Deze services werken samen in een Service Gerichte Architectuur, waarbij de applicaties onderling via API's (Application Programming Interfaces) met elkaar communiceren. Microservices worden vooral ingezet door bedrijven met extreme schaalbaarheidseisen en die snel en vaak (tientallen keren per dag) applicaties willen aanpassen zoals Netflix en Amazon. Door de opkomst van webbased applicaties en microservices in een service gerichte architectuur worden applicaties kleiner, handzamer en makkelijk om bij te werken. Dit sluit aan bij bewegingen als agile en DevOPS, waarbij anders wordt gekeken naar het leveren van nieuwe functionaliteiten. In een agile aanpak heb je geen grote big-bang releases meer die meerdere maanden voorbereiding vereisen, maar regelmatig kleine updates om snel nieuwe functionaliteit te kunnen leveren. Sommige vooruitstrevende ondernemingen leveren op deze manier meerdere updates per dag. We praten dan over Continuous Integration (CI) en Continuous Delivery (CD).

Container voor moderne apps

Er zijn zoals gesteld drie belangrijke ontwikkelingen gaande:

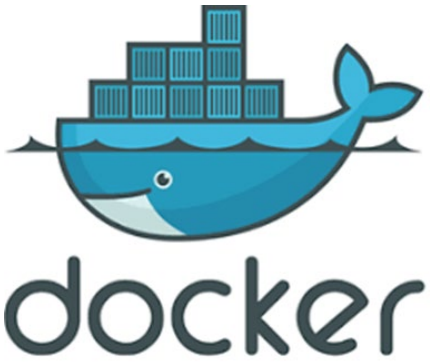
- Webbased apps (HTML5);
- Microservices (en Service Georiënteerde Architecturen);
- Agile ontwikkelmethodes (CI/CD).

De werkwijze met containers sluit hier goed op aan, maar niet out-of-the-box. Er is software nodig om containers op een eenvoudige manier te kunnen bouwen, inpakken en uitrollen. Daarnaast is specifiek voor webbased apps en microservices extra aandacht nodig voor de beschikbaarheid. Een webservice die enkelvoudig is uitgevoerd, is niet voldoende. Er zijn dus meer instances van een webservice nodig voor beschikbaarheid en bij toenemend gebruik moeten de webservices kunnen opschalen met extra instances, die weer opgeruimd moeten worden als het weer rustig is. Een goede webservice is dus:

- Hoog beschikbaar;
- Makkelijk schaalbaar.

Docker

Ondanks dat de techniek achter containers (cgroups) al uit 2006 stamt, werd het pas een succes vanaf 2013, toen Docker[5] voor het eerst in beeld kwam. Docker heeft het gebruik van containers dermate toegankelijk gemaakt dat het nu, iets meer dan vier jaar later, niet meer weg te denken is. Docker is de de-facto standaard op het gebied van containers geworden.
Met Docker kun je een applicatie met al zijn afhankelijkheden (dependencies) in een virtuele container inpakken, zodat het op iedere server kan draaien.
Een applicatie wordt verpakt in een image dat met een paar commando's op een willekeurig Linux systeem kan starten. Het maakt daarbij niet uit of het image draait in je eigen datacenter, op een fysieke server, op een virtuele server of buiten het eigen datacenter in de



Figuur 3

cloud. Docker maakt de applicaties zeer flexibel inzetbaar en overdraagbaar (portabiliteit) naar andere versies van Linux.
Daarbij heeft Docker ook het concept van een container registry (de docker hub) geïntroduceerd. Dit is een plek waar containers veilig opgeslagen kunnen worden. Om het probleem van de stateless eigenschap van containers te ondervangen, heeft Docker de Docker volumes geïntroduceerd.

Samen werken aan Docker

Het succes van Docker was al snel heel groot. De techniek werd initieel gepositioneerd als een alternatief voor server-virtualisatie[6], waardoor VMware als marktleider op dit gebied Docker niet kon negeren. In plaats van het te bevechten, heeft VMware zich aangesloten bij de Docker foundation en werkt het nu mee aan de ontwikkeling van Docker. Zo heeft VMware nu native ondersteuning van Docker op de hypervisor middels vSphere Integrated Containers (VIC).
Naast VMware doen meer bekende namen mee, zoals IBM, Cisco en RedHat. Ook Google, feitelijk de grondlegger van de containers, werkt mee aan de ontwikkeling van Docker.
En daarnaast is ook Microsoft aangesloten. Ondanks dat Docker een open-source en Linux gebaseerde techniek is, werkt Microsoft hard mee aan Docker en integreert het Docker in het Windows-server besturingssysteem.

Toepassing van Docker containers

Nu dat de achtergrond van containers bekend zijn, is het goed om te kijken naar de inzetgebieden en de voordelen van deze techniek op basis van Docker.

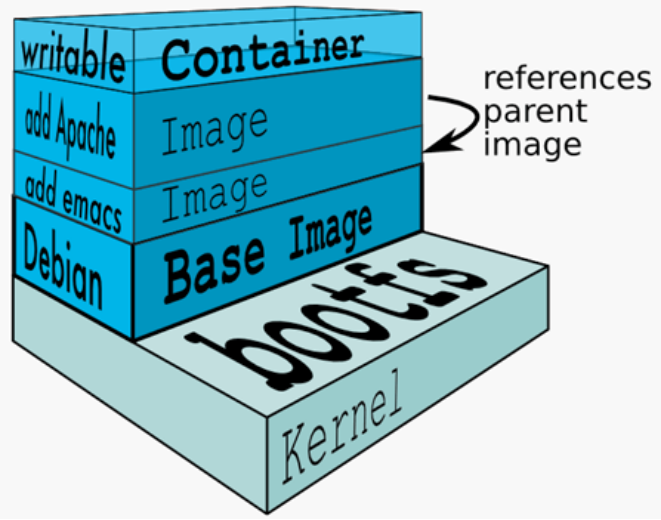
Opslag en transport van software

Containers zijn zoals eerder gesteld een handzaam pakket (image) dat op iedere willekeurige Linux server gestart kan worden. Een image kan vanuit een registry opgehaald worden en op een Docker host gestart worden. Het voordeel hiervan is dat distributie van een applicatie heel eenvoudig wordt. Er zijn geen complexe installatiescripts meer nodig en door het image zo te bouwen, dat ook alle afhankelijkheden aanwezig zijn, is het uitrollen van een applicatie heel eenvoudig geworden.

Afhankelijkheden

Ondanks dat het technisch mogelijk is om een container te maken met alle afhankelijkheden er in, is het ook mogelijk en vaak veel slimmer dit op te delen in meerdere kleine herbruikbare images. Er zijn immers veel programma's die onder water door meerdere applicaties worden aangeroepen. De opbouw is dan als te zien is in figuur 4.

Er is sprake van een zogenaamd boot OS, waarmee de Linux kernel wordt geladen. Dit is feitelijk een mini-OS, zonder extra functionaliteiten. Iedere container bevat vervolgens één of meerdere images en ziet het boot-OS als zijn eigen virtuele filesystem, het bootfs. Veelal wordt hierop als eerste image een zogenaamd base image gemaakt. Een base image bevat alle afhankelijkheden (binaries en libraries) die nodig zijn om de bovenliggende applicaties te gebruiken.
Hiervoor wordt vaak een deel van een Linux distributie gebruikt zoals Ubuntu of Debian.
Iedere container neemt dit base image mee als het opstart. Daarnaast kunnen er meer afhankelijkheden meegenomen worden en als laatste het uiteindelijke applicatie image. De complete stack van images draait in een eigen namespace en heeft via de cgroups functionaliteit eigen resources toegekend gekregen. Dit alles kan met Docker aan de hand van een aantal commando's of API calls gerealiseerd worden.



Figuur 4

Automatiseren van alle stappen, vanaf de bouw tot en met de uitrol zijn daarom heel goed te doen.

Meer dan alleen Docker

Docker zelf heeft de containertechnologie volwassen gemaakt. Maar voor efficiënt beheer van containers, de automatische uitrol en het op- en afschalen is meer nodig. Daarom zijn er diverse softwareoplossingen in het ecosysteem[7] rondom Docker ontstaan. Dit noemt men schedulers en orchestrators. Een scheduler kan gezien worden als een softwarepakket dat zorg draagt voor het starten en stoppen van containers.
De scheduler beheert resources: fysieke servers, virtuele servers of cloud resources. Op deze resources kunnen containers gestart worden. Veelal verzorgt de scheduler ook de load balancing die nodig is voor schaalbare en hoog beschikbare webservices.

Een orchestrator gaat een stapje verder en verzorgt ook interactie met andere bronnen. Bijvoorbeeld storage voor statefull applicaties, of een externe load balancers of DNS dienst.
De grens tussen een scheduler en een orchestrator is bij sommige producten vaag, omdat er veel overlappende functionaliteit in zit. De markt rondom Docker is ook nog heel erg

in beweging. Het lijkt er echter op dat Kubernetes[8] (K8s) de overhand gaat nemen als container orchestrator. Kubernetes verzorgt ook load balancing en heeft self healing eigenschappen en is daarmee een zeer veelzijdige tool.
Het werken met Kubernetes is echter complex, daarom is het gebruik van een container platform naast Kubernetes nog wel handig. Een dergelijk platform levert vaak een managementlaag bovenop een scheduler en/of orchestrator.

Figuur 5

Binnen Defensie en TNO is al ervaring met Rancher[9] als container platform. Rancher gaat in de toekomstige versies volledig over op Kubernetes als onderliggende orchestrator.

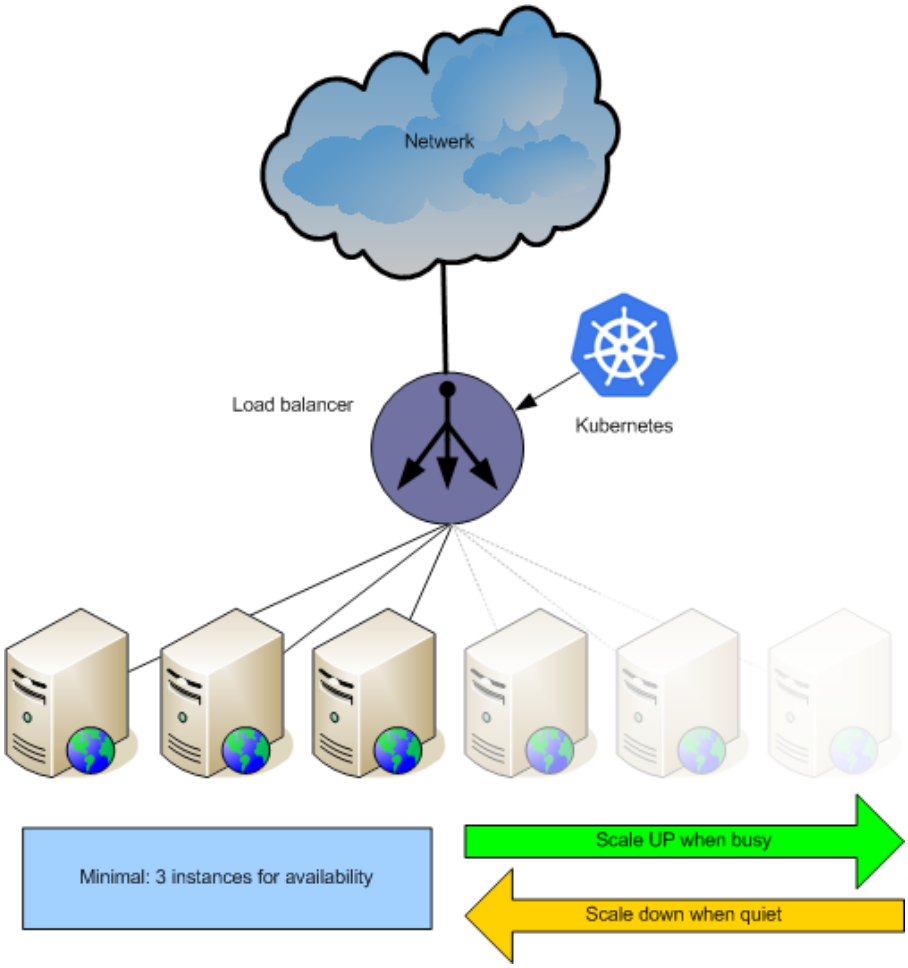
Figuur 6

Load balancing als basis

Load balancing van webservices is essentieel. Zowel voor beschikbaarheid als schaalbaarheid is een geautomatiseerde load balancing een vereiste.
Orchestrators als Kubernetes vullen dit in, door het op en afschalen van de containers automatisch bij te werken in de configuratie van de load balancers.

Containers en VMs

Het stateless karakter van containers sluit niet aan bij de meeste van de gangbare werklasten. Daarnaast bleek het gebruik van fysieke servers als basis voor containers veel lastiger te beheren dan een goed opgezet virtualisatiecluster.



Figuur 7

Omdat containers op ieder systeem kunnen draaien, maakt het voor containers dus niet uit of er gebruik wordt gemaakt van fysieke of virtuele servers, daarom wordt tegenwoordig ook veel gebruik gemaakt van virtuele servers voor containers. In sommige gevallen wordt dit juist expliciet aanbevolen.
De voordelen en hoge mate van automatisering van virtuele servers worden dan gecombineerd met de flexibiliteit en schaalbaarheid van containers.
In principe is iedere hypervisor of cloud oplossing bruikbaar, maar een oplossing die voorzien is van een goed gedocumenteerde open API en daarbij is gericht op stateless applicaties heeft daarbij wel de voorkeur. Deze combinatie is heel flexibel en kan gezien worden als een win-win situatie.

Een werkbare combinatie kan dan zijn:

- Openstack als private cloud platform voor de VMs;
- Docker als container engine;
- Kubernetes als container orchestrator;
- Rancher als container manager.

Alle genoemde oplossingen kunnen gezien worden als programmeerbare infrastructuur, waarbij het werken met API's onderling heel goed is ontwikkeld.

Ontwikkel tools

De eerder genoemde tools zoals Openstack, Docker, Kubernetes en Rancher zijn erg gericht op de infrastructuur. Wat nog ontbreekt is software voor ontwikkeling en dus CI/CD ten behoeve van een agile aanpak.
Hiervoor is door KIXS onderzoek gedaan, waarbij gebruik is gemaakt van Gitlab en Gitlab runner met een CI feature. Hierbij kan Gitlab automatisch containers bouwen en testen.

Op het gebied van CI en CD kan Gitlab iedere nacht de containers bijwerken met nieuwe patches en deze vervolgens digitaal ondertekenen en plaatsen in de interne registry. Zo staan iedere dag de verse images klaar. Als de applicaties achter een load balancer werken, kan vervolgens automatisch een rolling upgrade gestart worden, waarbij de actieve instances van een webservice één voor één worden bijgewerkt.
Hierbij wordt een term geïntroduceerd die bij veel IT'ers de wenkbrauwen doet fronsen: reverse uptime. Het idee achter reverse uptime is dat een lange uptime van een systeem NIET goed is. Als de instance lang in de lucht is, is deze hoogst waarschijnlijk niet recent bijgewerkt en lopen functionaliteiten en security achter. Bij containers hoort dus het regelmatig vervangen van het systeem en een lange uptime van een individuele instance is dus niet gebruikelijk. Dit alles is ook weer in lijn met het stateless concept van containers.
Deze aanpak staat in schril contrast tot de aanpak in de datacenters tot nu toe, waar uptime heel belangrijk is. Updates worden op basis van een change proces ingepland en maar al te vaak worden updates lang uitgesteld om downtime te voorkomen. Daarnaast is bij de oude aanpak de overgang naar een nieuwe versie van een besturingssysteem ook erg complex, getuige de vaak lastige en langdurige migratietrajecten om van oude besturingssystemen (Windows XP, Windows 2003, enz.) af te komen. Met container technologie kan je het onderliggende besturingssysteem zonder een traan te laten met pensioen sturen en een nieuw besturingssysteem daarvoor in de plaats zetten.

Samengevat

De manier waarop moderne applicaties in de tijd van het internet en de mobiele devices worden ontwikkeld is sterk aan het veranderen. De oude aanpak van client-server oplossingen wordt langzaamaan verlaten omdat het gewoonweg niet meer past bij de moderne wereld waarin smartphones, tablets, macbooks en chromebooks steeds meer gemeengoed worden.
De moderne applicaties worden anders ontwikkeld. We zien geen grote softwarepakketten meer met tientallen functies, maar een samenstel van samenwerkende microservices die ieder een specifieke taak uitvoeren. Wijzigingen in deze services zijn eenvoudig en snel door te voeren, omdat de gebruikers met een browser werken of een mobiele app gebruiken die automatisch bijgewerkt wordt via de Apple appstore of Google Playstore. Bij deze ontwikkelingen passen containers perfect. Containers zijn zeer goed bruikbaar in zowel het agile ontwikkelproces waar het vooral gaat om Continuous Integration en Continuos Delivery (CI/CD) als in vraagstukken rondom schaalbaarheid en hoge beschikbaarheid van services in de back-end. De ontwikkeling gaat dermate snel, dat het te verwachten is dat huidige uitdagingen op het gebied van statefull werken met containers ook opgelost gaan worden, waardoor ook veel legacy applicaties in containers verpakt kunnen worden. Verschillende initiatieven spelen hier al op in door hoog beschikbare databases te leveren die specifiek geschikt zijn voor het draaien in containers.

Er komt dus nog veel moois aan op dit gebied en daar kunnen we bij Defensie voordeel uit halen. Containers zijn daarbij misschien wel het belangrijkste stuk in de puzzel rondom agile werken en bieden tevens mogelijkheden voor een eenduidige werkwijze voor deployment van applicaties over alle gebruiksomstandigheden heen.

---

1 Zie https://en.m.wikipedia.org/wiki/Cgroups
2 Zie https://en.m.wikipedia.org/wiki/Operating-system-level_virtualization
3 Zie https://www.slashroot.in/difference-between-hypervisor-virtualization-and-container-virtualization
4 Zie https://www.openstack.org/software/
5 Zie https://www.docker.com/what-docker
6 Zie https://www.linux.com/news/containers-vs-hypervisors-choosing-best-virtualization-technology en https://www.linux.com/news/containers-vs-hypervisors-battle-has-just-begun
7 Zie https://github.com/cncf/landscape en https://www.cncf.io/
8 Zie https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
9 Zie http://rancher.com/rancher/