Unikernel voor Go

Henri Verroken

3 januari 2016

Ingediend door Henri Verroken (Computerwetenschappen), David Vercauteren (Computerwetenschappen), Maxim Bonnaerens (Computerwetenschappen) en Evert Heirbaut (Informatica).

1 Situering

1.1 De programmeertaal Go

Go[1] is een vrij jonge statically typed programmeertaal die ongeveer zes jaar geleden werd ontwikkeld door Google. Sindsdien heeft de taal ongelooflijk aan populariteit gewonnen en heeft ze zich in vele bedrijven genesteld als vervanger van Python, Ruby, PHP, C++, Java, etc. Ze laat haar sterktes zien bij het ontwikkelen van onder andere REST API's bij web development en grote complexe softwareprojecten zoals Docker[2], Gogs[3], etcd[4], Caddy[5] en nog vele andere[6]. De redenen hiervoor zijn onder andere

- Statically typed en gecompileerd naar native code.
- Go code wordt gecompileerd tot één statisch gecompileerde binary zonder dependencies. Dit wil zeggen dat er niet gelinkt wordt naar externe libraries zoals `libssl` of zelfs `libc`. Dit zorgt ervoor dat binaries onmiddellijk op eender welke Linux-distributie kunnen worden gedeployed door het installeren van slechts één bestand.
- Go code compileren is heel gemakkelijk. In de meeste gevallen volstaat het om het commando `go build` uit te voeren om de binary te compileren.

[1] `https://golang.org`

[2] Wrapper rond de containerizatiefeatures (cgroups) van de Linux kernel, `https://github.com/docker/docker`

[3] GitHub clone, `https://github.com/gogits/gogs`

[4] Key-value store als onderdeel van CoreOS, een besturingssysteem voor cloud-based computing, `https://github.com/coreos/etcd`

[5] Zeer snelle HTTP/2 enablede statische web server, `https://github.com/mholt/caddy`

[6] `https://github.com/search?q=language%3Ago&type=Repositories`

- Go code compileren voor een ander platform is zeer gemakkellijk. Het volstaat om het target platform en de architectuur te specifieren, zoals bijvoorbeeld `GOOS=linux GOARCH=arm GOARM=5 go build`
- Go wordt compileerd door een compiler die volledig in Go geschreven is.[7] Dit wil zeggen dat men met een werkende installatie van Go op bijvoorbeeld Linux aan de hand van het commmando `GOOS=windows GOARCH=amd64 go build` een Go compiler kan compileren voor Windows.
- Go kent een zeer grote en actieve community. Er zijn immens veel open source bibliotheken beschikbaar. De source code wordt via het commando `go get` gedownload, waarna ze onmiddellijk in de eigen code kan gebruikt worden. Er is geen package manager aanwezig wat opnieuw de zaken vereenvoudigt.
- Go kent een zeer grote standaardbibliotheek. Netwerkprimitieven zijn geïmplementeerd in de platformonafhankelijke standaardlibrary, alsook de meeste cryptografische primitieven en protocollen, zoals AES, SHA1, SHA2, ECDSA, RSA, TLS en vele andere.

1.2 Virtualisatie

De voorbije jaren is er heel wat vooruitgang geboekt inzake virtualisatie. In het huidige landschap zijn er verschillende mogelijkheden om een applicatie te deployen. De belangrijkste worden opgesomd.

Op hardware Applicaties draaien op een besturingssysteem dat rechtstreeks op de hardware draait. Er is geen virtualisatie of isolatie.

In een volwaardige VM Per applicatie, zoals bijvoorbeeld een databank of netwerkapplicatie, heeft men een VM die deze applicatie draait en isoleert van de andere applicaties. Per applicatie is er een volledig besturingssysteem of kernel aanwezig die draait op een hypervisor zoals Xen, VMware, KVM, Hyper-V, etc. De kernel zorgt onder andere voor de network stack, het bestandssysteem, etc.

In een container Alle applicaties delen eenzelfde kernel maar worden geïsoleerd door chroot-mechanismen[8] en de cgroups-functionaliteit in Linux.[9] Appli-

[7] Dit vanaf Go 1.5

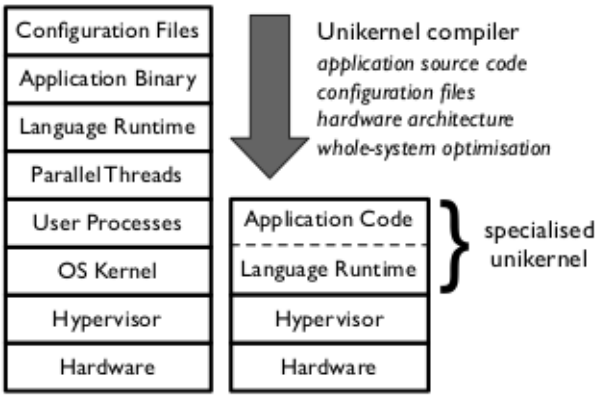
[8] Bij een chroot wordt de wortel van het bestandssysteem geremapped op een andere directory. Zo zorgt het commando `chroot /var/db` ervoor dat de kernel vanaf dan de directory `/var/db` zal gebruiken als `/`. Applicaties die uitgevoerd worden kunnen dan nooit buiten `/var/db` kijken.

[9] cgroups of control groups is een feature van de Linux kernel die het toelaat applicaties volledig te isoleren en virtuele root-gebruikers te creëren. Processen in een veschillende control group kunnen elkaar niet beïnvloeden of zien.

caties die deze functionaliteit aanbieden, zijn onder andere het verouderde OpenVZ[10] en meer recent Docker en LXC/LXD[11].

We zien dat de virtualisatietechnologie evolueert naar zeer lichte virtualisatievormen waar er toch nog afdoende isolatie kan gegarandeerd worden. Het delen van een kernel is eenvoudiger, sneller en vereist minder systeemmiddelen dan het opstarten van een VM voor iedere applicatie. Toch wordt er nog voldoende isolatie gegarandeerd. Waar het opstarten van een volwaardige VM tot enkele minuten kan duren, duurt het opstarten van een container veelal slechts enkele seconden.

Een volgende stap in het virtualisatieproces is de ontwikkeling van de zogenaamde unikernels[12]. Bij unikernels wordt het volledige besturingssysteem vervangen door een library OS. De applicatie wordt gelinkt met dit library OS waardoor deze rechtstreeks op gevirtualiseerde hardware kan draaien zonder tussenkosmt van een besturingssysteem. Het library OS maakt gebruik van hypercalls om te communiceren met hardware. Dit is te zien in Figuur 1.



Figuur 1: Vergelijking tussen een volledige VM-oplossing en een unikernel-oplossing (`https://mirage.io/wiki/technical-background`).

Dit zorgt ervoor dat alle onnodige en ongebruikte code aanwezig in een besturingssysteem niet vervat zit in de unikernel. We krijgen zeer kleine binaries, zowel in grootte als in geheugenvereisten, die geoptimaliseerd zijn voor de virtuele hardware waarop ze draaien. We krijgen ook zeer snelle opstarttijden in de orde van milliseconden.

[10] `https://openvz.org/`

[11] `https://linuxcontainers.org/`

[12] Een zeer goede introductie tot unikernels is te vinden in `https://mirage.io/wiki/overview-of-mirage`

Bovendien is het ook belangrijk te vermelden dat unikernels een verhoogde veiligheid met zich meebrengen. Er is niet alleen een veel kleinere attackvector, maar wanneer een applicatie alsnog gehackt wordt is het onmogelijk conventionele technieken te gebruiken, zoals het opstarten van een shell, omdat deze gewoonweg niet aanwezig zijn in de unikernel.

We zien dus dat het gebruik van unikernels vele voordelen met zich meebrengt. Vooral met het oog op het alsmaar verhoogde niveau van computersysteemvirtualisering en on-demand computing, ook wel de cloud genoemd, is het zeer belangrijk de densiteit op hardware te verhogen.

2 Mogelijke toepassingen

Zoals reeds gezegd situeren de mogelijke toepassingen van unikernels zich vooral bij de verdere ontwikkeling van cloud computing, en in het bijzonder elastic cloud computing. Dit wil zeggen dat een cloudsysteem zich autonoom kan schalen bij een verhoogde laadfactor.

Aangezien unikernels weinig onnodige systeemmiddelen gebruiken wordt er heel zuinig met hardware omgesprongen. Bovendien zorgt de lage opstarttijd, die wordt gemeten in milliseconden, ervoor dat er heel snel nieuwe instanties van applicaties kunnen opgestart worden wanneer dat nodig is.[13].

3 Bestaande ontwikkelingen

Het meest ontwikkelde unikernelproject is momenteel MirageOS[14]. MirageOS laat toe om applicaties te schrijven in OCaml op een UNIX systeem en deze dan te compileren als unikernel voor Xen. OCaml is echter een functionele taal die een veel kleinere community heeft dan Go. Er is dan ook een veel minder uitgebreide standaardbibliotheek en er zijn veel minder bibliotheken beschikbaar van derde partijen.

Hierdoor moest er nog veel software van de grond af geïmplementeerd worden. Voorbeeld hiervan zijn bibliotheken voor cryptografische primitieven[15], TLS[16] en HTTP[17]. Bovendien blijft het twijfelachtig of het project ooit de grote menigte zal kunnen bereken, gezien de functionele taal die wordt gebruikt.

[13] Een voorbeeld hiervan is jitsu of just-in-time summoning of unikernels. Deze applicatie zal wanneer er een DNS-aanvraag voor een web server binnenkomt een unikernel opstarten en tegelijk de DNS-aanvraag beantwoorden. Wanner het eerste TCP SYN packet toekomt, is de unikernel opgestart en kan deze de TCP connectie behandelen. `https://www.usenix.org/system/files/conference/nsdi15/nsdi15-paper-madhavapeddy.pdf`

[14] `https://mirage.io/`, `https://github.com/mirage`

[15] `https://github.com/mirleft/ocaml-nocrypto`

[16] `https://github.com/mirleft/ocaml-tls`

[17] `https://github.com/mirage/ocaml-cohttp`

Andere projecten zijn onder meer HalVM[18], LING[19] en Runtime.js[20].

4 Doelstellingen

Doelstellingen van dit project is een port te maken van de Go compiler die een applicatie rechtstreeks compileert naar een bestand dat onmiddellijk uitgevoerd kan worden door de Xen hypervisor[21]. Hypercalls worden gebruikt om te communiceren met de hypervisor.

De uiteindelijke doelstelling van het project is een Go applicatie te kunnen compileren aan de hand van het commando `GOOS=xen GOARCH=amd64 go build`. Dit commando moet ons een binary geven die we onmiddellijk kunnen uitvoeren als een Xen-guest.

Hiervoor onderscheiden we verschillende stappen.

1. Vertrouwd geraken met de werking van Xen en zijn hypercalls. Hoe moeten we iets draaiende krijgen op Xen?
2. Vertrouwd geraken met het compilatieproces in Go. Hoe voegen we het Xen platform toe aan Go als ondersteund platform?
3. Ervoor zorgen dat de Go runtime[22] draait op Xen. In het bijzonder moet er gekeken worden hoe geheugen beheerd wordt in Go.
4. Een werkende IP stack die ons toelaat ICMP echo requests te beantwoorden.
5. Een minimaal read-only bestandssysteem dat toegevoegd wordt aan de binary. Dit laat ons toe statische bestanden te lezen. (extra)
6. Een minimale TCP stack die ons toelaat TCP connecties te gebruiken. (extra)

Merk op dat we vanaf stap 4 reeds applicaties kunnen maken met enig nut, zoals bijvoorbeeld een UDP gebaseerde in-memory key-value store (cfr. memcached[23]). In het hoogst onwaarschijnlijke geval dat we na stap 4 nog tijd over hebben kunnen we een read-only bestandssysteem en het TCP protocol implementeren. Dit zou betekenen dat we alle soorten stateless netwerk applicaties kunnen maken, in het bijzonder database gebaseerde web applicaties of statische web pagina's.

[18] Een unikernel voor Haskell, `https://github.com/GaloisInc/HaLVM`
[19] ErlangOnXen, `http://erlangonxen.org/`
[20] JavaScript VM die rechtstreeks op KVM draait, `https://github.com/runtimejs/runtime`
[21] `http://www.xenproject.org/`
[22] https://github.com/golang/go/tree/master/src/runtime
[23] `https://github.com/memcached/memcached`