TETRA OpenCloudEdge tussentijds verslag

Vrije Universiteit Brussel - Industriële Wetenschappen

Mei 2020

Executive summary

Dit verslag voorziet de initiële bevindingen omtrent het voorbereiden, opzetten en consumeren van private cloud infrastructuur in eigen beheer. In afwachting van het beschikbaar worden van installatietools (zoals DevStack) voor de recent verschenen OpenStack *Ussuri* versie, welke compatibel is met de nieuwste generatie Linux-gebaseerde besturingssystemen, hebben we alvast een Kubernetes oplossing uitgerold. We maken hierbij gebruik van containerisatie geleverd door Docker (andere containerisatieomgevingen kunnen ook) in plaats van volwaardige virtualisatie zoals geleverd door OpenStack. Voor het merendeel van de KMO-doeleinden zien we geen problemen bij het gebrek aan virtualisatie, zolang de toegang tot het effectieve cloudbeheer intern blijft. De uitgerolde gecontaineriseerde applicaties kunnen uiteraard wel openbaar toegankelijk zijn. Indien volwaardige virtualisatie toch nodig blijkt, dan is het nog steeds mogelijk om virtualisatie en containerisatie te combineren. Kubernetes en OpenStack kunnen naast elkaar draaien waarbij diensten zoals Keystone kunnen gedeeld worden. Kubernetes kan ook volledig gevirtualiseerd binnen OpenStack uitgerold worden. We merken voorts een shift richting projecten zoals KubeVirt en KataContainers die het mogelijk maken om een vorm van virtualisatie te bereiken vanuit Kubernetes.

We beginnen dit verslag met het voorstellen van onze aangekochte hardware infrastructuur, waarbij de nodige afwegingen zijn gemaakt in functie van het aantonen van hoge beschikbaarheid en voldoende rekenkracht ten opzichte van het vooropgestelde budget. We bespreken de basisconcepten, installatie, gebruik en onze eerste ervaringen voor met Kubernetes in combinatie met Docker. We stellen de tot nu toe uitgerolde applicaties voor die actief zijn binnen onze Kubernetes cluster. We sluiten af met onze eerste bevindingen omtrent het opzetten en consumeren van een private cloud aan de hand van Kubernetes.

Naast onze analyse van Kubernetes zijn we ook gestart met een eerste exploratie van Network Function Virtualization (NFV) aan de hand van het ETSI OSM platform. Dit wordt ook beschreven in het verslag.

1 Voorstelling fysieke cloud infrastructuur

1.1 De kern van onze private cloud

De fysieke infrastructuur van onze private cloud omgeving bevat als kern 3 homogene HPE ProLiant DL325 gen 10 servers. De specificaties zijn als volgt:

- 1U single rack unit
- AMD EPYC Rome 7302P (16 core processor - 32 threads)
- 8 Small Form Factor (2.5") SATA opslag
  - 1 x 1 TB SSD Samsung 860 EVO
  - 3 x 2 TB HDD Toshiba L200
- 4 x 1 Gbps Ethernet

We hebben bewust gekozen om even te wachten en drie homogene exemplaren aan te kopen van het instapmodel van de huidige generatie AMD EPYC servers. Deze generatie AMD-processoren (EPYC Rome) verschaft uitstekende prestaties in functie van de kostprijs. Om de werking en schaalbaarheid van een cloudomgeving te demonstreren is het interessanter om te beschikken over zo veel mogelijk servers als het budget toelaat. Dit is ook gunstig om automatische fail-over orchestratie, gedistribueerde opslag en hoge beschikbaarheid te demonstreren.

Gedistribueerde systemen hebben baat bij een oneven aantal nodes omdat deze typisch werken aan de hand van een quorum met minstens $ceil(N/2)$ nodes, waarbij N het totale aantal nodes is. In het geval van 3 gedistribueerde servers houdt dit in dat er minstens 2 servers operationeel moeten zijn. Er kan dus één server falen zonder functionaliteit te verliezen.

Er is gekozen voor een combinatie van Solid State Disk (SSD) opslag en Hard Disk Drive (HDD) opslag. De tragere HDD-opslag dient vooral voor bulk opslag waar de schrijfprestaties en willekeurige leesprestaties van ondergeschikt belang zijn aan de opslagcapaciteit. We denken hierbij primair aan *object opslag* wat typisch het Write Once, Read Multiple (WORM) principe volgt. De SSD wordt gebruikt voor het host besturingssysteem en daar waar meer performante opslag gevraagd is. We denken hierbij aan databases en meer algemene *block opslag* doeleinden. NVMe gebaseerde opslag geniet de voorkeur wegens de significant hogere prestaties, maar wegens budgetredenen in combinatie met het hebben van 'slechts' 1 Gbps Ethernet interfaces voor de gedistribueerde opslag, is er geopteerd voor de meer toegankelijke 2.5"SATA SSDs die significant goedkoper zijn in serveromgevingen.

Om hoge beschikbaarheid te garanderen wordt gebruik gemaakt van een software gedefinieerde gedistribueerde opslag, beter gekend als *Software Defined Distributed Storage* (SDDS). Deze verspreidt de opslag op een redundante manier over de gedistribueerde servers. Vanwege deze keuze maken we geen gebruik van de ingebouwde hardwarematige RAID voorzieningen; dit wordt immers afgeraden door onze gekozen SDDS oplossing: *ceph* (beschikbaar op `https://ceph.io/`).

Momenteel gebruiken we slechts één van de vier beschikbare 1 Gbps Ethernet interfaces. Om optimaler gebruik te maken van de beschikbare interfaces gaan we nog evalueren of we hiervoor IEEE 802.3ad link aggregation uitrollen of gebruik zullen maken van verschillende Ethernet netwerken voor cloud-intern, gedistribueerde opslag en publiek netwerkverkeer. Alvorens dit getest kan worden moet hiervoor eerst nog een extra Ethernet switch aangekocht worden.

1.2 Uitbreiding naar de Edge

De uitwerking van onze edge is nog niet concreet. Bij initiële testen zijn resource-constrained Linux toestellen (vb. Raspberry Pi 3B+) en standaard Linux workstations geïntegreerd binnen onze Kubernetes cluster. Deze toestellen bevinden zich in hetzelfde interne netwerk. Netwerk delay is hierdoor niet van tel (orde van 100 ns). De netwerk doorvoer, zelfs wanneer de Ethernet links naar de edge toestellen beperkt zijn tot 100 Mbps, volstaan voor het merendeel van de edge toepassingen. Dit soort opstelling resulteert dus eerder in een uitbreiding van onze kleinschalige cloudomgeving met minder krachtige infrastructuur, in plaats van het toevoegen van wat kan beschouwd worden als edge toestellen. Of alleszins toch wanneer men het bekijkt vanaf het netwerk perspectief.

Bij een andere benadering hebben we de thuisserver van een projectmedewerker toegevoegd als edge toestel aan onze cluster. Deze thuisserver bevindt zich, zoals de naam het aangeeft, fysiek bij de projectmedewerker thuis en is via een VPN-verbinding verbonden met het interne VUB-ETRO netwerk waardoor communicatie met onze private cloud mogelijk is. Een latency van ± 10 ms en netwerk doorvoer van een VDSL-verbinding (in de praktijk ± 90/30 Mbps down/up) hebben een zekere invloed. Deze benadering leunt alvast dichter aan bij de typische situatie waarbij edge infrastructuur typisch wordt opgezet om delay en/of netwerkbelasting te verlagen. We gaan er in dit geval vanuit dat de consument van de cloud applicatie, IoT opstelling, of dergelijke, zich nabij de edge locatie bevindt.

1.3 Interactie met de publieke cloud

We zien op dit moment geen technische beperkingen die in de weg kunnen staan voor interactie met een publieke cloud aanbieder zoals AWS, Azure, GKS, publieke OpenStack aanbieders en dergelijke. We hebben de mogelijkheid tot het verkrijgen van tal van publiek routeerbare IPv4 adressen. Hierdoor is het mogelijk om inkomende verbindingen tot binnen onze private cloud omgeving te krijgen. Interactie met een van deze openbare cloud platformen is op dit moment echter nog niet getest.

2 Opzetten van de cloud omgeving

2.1 OpenStack en het belang van het onderliggende besturingssysteem

Een cloudomgeving heeft nood aan een host besturingssysteem in combinatie met een hypervisor en/of container engine. Commerciële besturingssystemen zoals Red Hat Enterprise Linux (RHEL) worden typisch gebruikt in productieomgevingen. Vandaar onze keuze voor CentOS, een community gebaseerde variant gebaseerd op RHEL. Onze eerste testomgevingen in Januari 2020 (vanaf het in ontvangst nemen van de servers) zijn opgezet aan de hand van de recentste iteratie van CentOS: versie 8 (uitgebracht september 2019). Hoewel ten tijde van testen het besturingssysteem al 4 maanden beschikbaar is hebben we vastgesteld dat er nog vele softwarepakketten en bibliotheken nog niet beschikbaar zijn. Zo blijkt ook OpenStack versie *Train*, uitgebracht oktober 2019, niet compatibel te zijn met CentOS 8.

We hebben hierop overwogen om het de OpenStack Train ondersteunde CentOS 7 uit te rollen. Echter dateert deze versie reeds van 2014 met over de jaren slechts kleine upgrades. Bijgevolg dateren veel softwarepakketten en bibliotheken nog van 2014 of daarvoor waardoor geregeld features ontbreken en nieuwere software niet compatibel is. Een bijkomend nadeel aan het werken met verouderde besturingssysteem zoals CentOS 7 op de laatste nieuwe hardware is dat deze de neiging hebben minder goed presteren dan recentere besturingssystemen. (zie `https://www.redhat.com/en/blog/red-hat-enterprise-linux-8-improves-performance-modern-workloads` en `https://www.phoronix.com/scan.php?page=article&item=centos-8-benchmarks`)

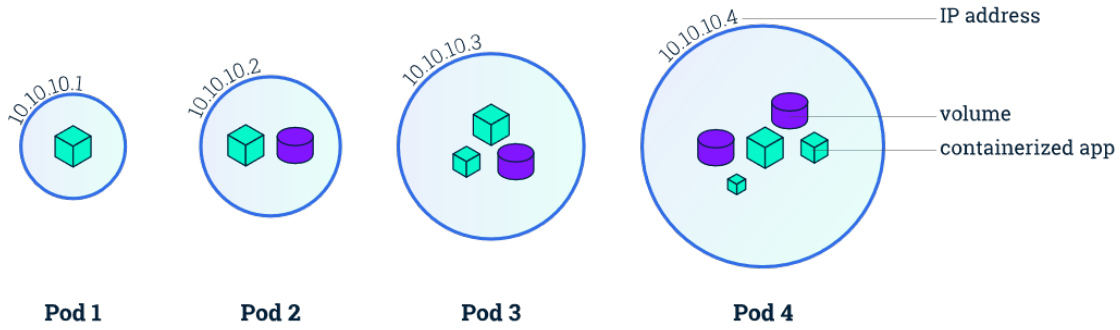
Hieruit volgt onze keuze om te wachten met de uitrol van OpenStack te wachten tot OpenStack *Ussuri* en de bijhorende installatie scripts reeds enige tijd beschikbaar zijn. Deze nieuwe OpenStack versie is halfweg mei 2020 uitgebracht en zou onder meer compatibel zijn met CentOS 8 en Ubuntu 20.04 LTS. In tussentijd maken we kennis met containerisatie aan de hand van Kubernetes aangezien dit een krachtig en steeds populairder wordend alternatief is op volledige virtualisatie oplossingen zoals OpenStack. Beiden kunnen ook gecombineerd worden waarbij OpenStack en Kubernetes naast elkaar kunnen draaien waarbij bepaalde diensten zoals Keystone gedeeld worden. Kubernetes kan ook gevirtualiseerd binnen OpenStack draaien om zo bijvoorbeeld per gebruikersgroep of fase (vb. ontwikkeling, test, productie) aparte clusters te voorzien.

2.2 Een inleiding to Kubernetes en de basisconcepten

Kubernetes, initieel ontwikkeld door Google, is een populaire open source container orchestrator. Dit houdt in dat applicaties draaiende binnen Kubernetes werken op containerniveau in plaats van volledige virtualisatie toe te passen. Voor veel toepassingen is dit zelfs voordelig omdat minder resources nodig zijn en deze sneller opstarten omdat er niet eerst een gastbesturingssysteem moet opstarten; iets wat handig is in geval er horizontaal geschaald wordt.

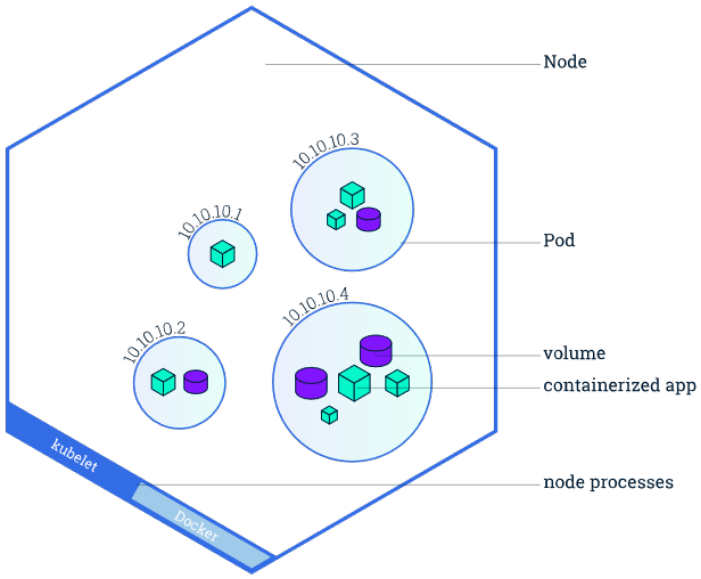
Kubernetes en OpenStack zijn ontwikkeld vanuit een ander principe. OpenStack is enorm krachtig maar de complexiteit vanwege de tientallen modules waaronder virtualisatie, orchestratie, opslag, netwerking, etc. kunnen overweldigend zijn. Kubernetes daarentegen focust zich op containerorchestratie en de interactie met externe technologieën zoals bijvoorbeeld voor opslag. Kubernetes beschikt zelfs niet over een eigen ingebouwde container engine maar maakt gebruik van een externe oplossing, veelal Docker.

Pods zijn de bouwstenen van het uitrollen van applicaties binnen Kubernetes. Ze groeperen één of meerdere gerelateerde **containers** en delen daarbij de netwerkconnectiviteit en optionele opslagvolumes. Communicatie tussen pods en met andere onderdelen gebeurt aan de hand van hun toegewezen, cluster-intern IP-adres.



Figuur 1: Pods als bouwstenen. Bevatten één of meerdere containers, netwerk connectiviteit en optioneel opslagvolumes.

Nodes representeren fysieke of virtuele machines die Kubernetes Pods kunnen uitvoeren. Werknodes beschikken over een container engine (vb. Docker) en de kubelet service waardoor de Kubernetes controle-entiteiten hierop pods kunnen uitrollen en beheren.



Figuur 2: Een node stelt een (virtuele) machine voor waarop de Docker en Kubelet processen actief zijn. Kubernetes pods worden op nodes uitgerold.

Deployments specifiëren de gewenste staat van een bepaalde uitrol. Hierbij worden de specificaties van de pods meegegeven, hoeveel pods er uitgerold worden, eventueel kan daarbij ook gespecifieerd worden waarop deze uitgerold worden aan de hand van (anti-) affiniteit en node-specifieke parameters zoals CPU architectuur, bevatten van een GPU, etc. alsook node-specifieke labels die door de gebruiker gezet kunnen worden.

Wanneer er een pod vanuit de deployment niet meer operationeel is vanwege allerlei mogelijkheden (vb. de node waarop de pod draait is onbeschikbaar geworden of de pod is gecrasht) wordt deze automatisch opnieuw uitgerold om het in de deployment meegegeven actieve aantal replica's te evenaren. Bij updates aan een deployment worden de bijhorende pods standaard al rollend gewijzigd naar de nieuwe versie om downtime te vermijden wanneer mogelijk.

Services vormen langdurige aanspreekpunten naar de ingestelde deployments. Dit is essentieel omdat pods binnen een deployment vaak maar kortlevend zijn en bovendien dynamische cluster-interne IP adressen krijgen. Load-balancing functionaliteiten worden tevens voorzien wanneer deployments over meerdere gerepliceerde pods beschikken.

Sommige types *services* maken inkomende communicatie mogelijk van buiten de Kubernetes cluster. Dit kan door een (of meerdere) dynamische poort op alle nodes te binden aan de bijhorende service en achterliggende deployment. Een ander type service maakt het mogelijk een extern IP adres op te zetten die verwijst naar de bijhorende service en achterliggende deployment.

Ingress maakt communicatie van buiten de Kubernetes cluster mogelijk voor webservices aan de hand van een door de cluster voorziene webserver. De aangemaakte Ingress regels voorzien welk (sub)domein naar welke cluster-interne service moet doorverwijzen. Ingress kan TLS terminatie verzorgen voor HTTPS beveiligde communicatie. Een belangrijk voordeel van het werken met Ingress in plaats van gewone extern toegankelijke Kubernetes services is dat er slechts één IP adres nodig is voor de Ingress webserver, ongeacht het aantal aangebrachte Ingress regels die verwijzen naar de binnen de cluster uitgerolde webapplicaties.

2.3 Installatiegids Kubernetes

Het opzetten van een Kubernetes cluster bestaande uit 3 control-plane nodes is uitgevoerd aan de hand van de *kubeadm* tool. Kubernetes is beschikbaar op CentOS 7 en 8 maar we hebben na initiële problemen met deze besturingssystemen tijdelijk gekozen voor Debian 10 omdat we hier meer vertrouwd mee zijn. Merk op dat de procedure, op het afhalen van het initiële softwarepakketten na, gelijkaardig is ongeacht het besturingssysteem.

Bij installatie blijkt de Kubernetes documentatie duidelijk en up-to-date. Het installeren van de benodigde softwarepakketten en uitrollen van een eigen Kubernetes cluster aan de hand van kubeadm is eenvoudig en is probleemvrij. De volledige bootstrapping procedure kan geautomatiseerd of manueel doorlopen worden. Kubeadm faciliteert ook in het eenvoudig toevoegen van extra nodes aan een bestaande cluster. Alvorens het uitrollen van een Kubernetes cluster moet eerst de omgeving opgezet worden:

- Installatie van de container runtime omgeving, in ons geval Docker. Platformafhankelijke instructies zijn beschikbaar op `https://kubernetes.io/docs/setup/production-environment/container-runtimes`. Vergeet niet Docker's daemon.json aan te passen om systemd te gebruiken als cgroup driver.
- Installeren *kubeadm*, *kubelet* en *kubectl* tools. Zie `https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#installing-kubeadm-kubelet-and-kubectl` voor de besturingssysteem afhankelijke instructies aan de hand van hun package manager.
- Kubernetes kan wegens veiligheidsoverwegingen (nog) niet overweg met swap als geheugenuitbreiding. Deze moet permanent gedeactiveerd worden. De huidige situatie omtrent het gebruik van Kubernetes met swap zijn beschikbaar op `https://github.com/kubernetes/kubernetes/issues/53533`.
  - swapoff −a (deactiveert swap tot eerstvolgende heropstart)
  - Verwijderen van swap geralateerde entries in /etc/fstab en /etc/initramfs-tools/conf.d/resume
  - Herbouwen initramfs en grub: update−initramfs −u && update−grub

De kubeadm tool voorziet een elegante manier om TLS beveiligde Kubernetes clusters uit te rollen waarbij het de nodige bootstrap fase op zich neemt. We beginnen met het opzetten van één control-plane node waaraan nadien extra control-plane nodes en worker nodes worden toegevoegd. Informatie omtrent het initialiseren van de cluster aan de hand van kubeadm init is beschikbaar op `https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-init`.

```
kubeadm init --control-plane-endpoint "k8s.opencloudedge
    .be:6443" --upload-certs
```

Het uitvoeren van deze initialisatie functie geeft als output ook de benodigde commando's om extra control-plane nodes en worker nodes toe te voegen met de gegenereerde tokens, hashes en keys.

Voor een worker node:

```
kubeadm join k8s.opencloudedge.be:6443 --token [token]
    --discovery-token-ca-cert-hash [sha256:hash]
```

Voor een control-plane node:

```
kubeadm join k8s.opencloudedge.be:6443 --token [token]
    --discovery-token-ca-cert-hash [sha256:hash] --
     control-plane --certificate-key [certificate key]
```

Het is uiteraard ook mogelijk om na de initiële uitrol periode nieuwe nodes toe te voegen. Hiervoor is het allicht noodzakelijk een nieuw token te genereren aangezien deze na enige tijd vervallen.

```
kubeadm token create --print-join-command
```

Indien het een control-plane node betreft dient er tevens een nieuwe certificate key gegenereerd te worden.

```
kubeadm init phase upload-certs --upload-certs
```

Bij het initieel opzetten van de cluster is er een configuratiebestand met admin account aangemaakt in /etc/kubernetes/admin.conf. Dit bestand kan gekopieerd worden naar $HOME/.kube/config zodat deze configuratie standaard gebruikt wordt door *kubectl* voor het beheer van de Kubernetes cluster. Het kan ook overgezet worden naar machines buiten de Kubernetes cluster die beschikken over kubectl. Neem hierbij de nodige veiligheidsoverwegingen wel in achting omdat dit absolute toegang tot de Kubernetes cluster biedt.

2.4 Container Network Interfaces (CNI)

Communicatie tussen pods onderling, en met andere componenten, is essentieel binnen een Kubernetes cluster. Dit is triviaal wanneer alle pods op eenzelfde node draaien maar wordt complex wanneer deze verspreid zijn over meerdere nodes. Er

zijn verschillende Container Network Interfaces (CNI) beschikbaar: *Calico*, *Canal*, *Cilium*, *Flannel* en *WeaveNet* zijn slechts enkele van de mogelijkheden.

Wij hebben initieel gekozen voor WeaveNet omdat deze onder de community populaire CNI encryptie mogelijk maakt, eenvoudig is om op te zetten en werkt met quasi elke mogelijke netwerkarchitectuur. WeaveNet gebruikt VXLAN encapsulatie en kan communicatie mogelijk maken met toestellen achter NAT en bij gedeeltelijk verbonden netwerken zonder volledige mesh connectiviteit. WeaveNet kan uitgerold worden met volgend commando:

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-
    version=$(kubectl version | base64 | tr -d '\n')"
```

WeaveNet beschikt (nog) niet over IPv6 ondersteuning waardoor we voor toekomstige interactie met onze IPv6-native IoT infrastructuur noodgedwongen moeten uitwijken naar een andere CNI. Onze keuze gaat in de eerste plaats naar Calico vanwege het gebruik binnen commerciële Kubernetes cloud omgevingen zoals Google's GKE, al beschikken Cilium en Kube-Router ook over IPv6 ondersteuning. Calico voorziet meerdere communicatiemogelijkheden: encapsulatie aan de hand van IP-in-IP of VXLAN, of indien de netwerkinfrastructuur dit toelaat, zuivere IP-routing, eventueel in combinatie met BGP tussen verschillende subnets. Calico kan uitgerold worden met volgend commando:

```
kubectl apply -f https://docs.projectcalico.org/
    manifests/calico.yaml
```

CNIs kunnen op een bestaande Kubernetes cluster uitgewisseld worden door de oude CNI volledig te verwijderen en de nieuwe CNI te installeren. Dit gaat gepaard met enige downtime. Na volledige verwijdering van de oude CNI en installatie van de nieuwe CNI moeten alle pods opnieuw uitgerold worden om deze te laten werken bovenop de nieuwe CNI. Onze initiële ervaring leert ons dat vooral het volledig verwijderen van de oude CNI op alle nodes, inclusief de daarbij aangemaakte virtuele interfaces en iptables regels, het merendeel van het werk omvat.

2.5 GPU-detectie en inplannen van GPU-applicaties

Eén van onze Kubernetes werknodes buiten de 3 hoofdservers is een workstation uitgerust met een Nvidia RTX 2070 grafische kaart. Deze kan gebruikt worden voor CUDA en andere general-purpose GPU-applicaties. Het is mogelijk om hiervan ook binnen Kubernetes gebruik van te maken. Essentieel hiervoor is dat dit soort applicaties enkel uitgerold worden op de geschikte nodes. Hiervoor moet er op de pod of deployment bij *spec.resources.limits* een verwijzing gemaakt worden naar *nvidia.com/gpu* of *amd.com/gpu*.

Allereerst moeten de correcte opstelling omtrent drivers en tools aanwezig zijn om GPU-geaccelereerde taken binnen containeromgevingen uit te voeren. Op `https://github.com/NVIDIA/nvidia-docker` vindt men de nodige informatie hoe dit

in zijn werk gaat. Op `https://github.com/NVIDIA/k8s-device-plugin` is de Kubernetes- specifieke informatie beschikbaar. Merk op dat deze nog gebruik maakt van de verouderde nvidia-docker2 implementatie maar dat dit succesvol werkt.

Aparte Kubernetes *GPU device plugins* worden gebruikt om te detecteren of er een geschikte GPU aanwezig is. Er zijn plugins beschikbaar voor Nvidia en AMD. Deze plugins worden uitgerold als *DaemonSet*: deze zorgt ervoor dat de bijhorende pods automatisch op alle geschikte (vb. alle 64-bit Linux machines) nodes worden uitgerold. Meer informatie omtrent het uitrollen van de *GPU device plugins* en het schedulen van GPU werktaken binnen Kubernetes is beschikbaar op `https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/`.

2.6 Metering en autoschaling

Het is vaak belangrijk op de hoogte te zijn hoeveel resources pods in beslag nemen. Dit maakt het bijvoorbeeld ook mogelijk om automatische horizontale schaling toe te passen om het aantal pods van een bepaalde deployment aan te passen aan de hand van de belasting. Extra pods worden hierbij opgestart wanneer er enige tijd veel belasting is, en het aantal pods wordt terug afgebouwd wanneer de vraag afneemt.

Metrics server is een eenvoudige tool die aan de hand van een deployment en *DaemonSet* (draait op elke node) de nodige metrieken aggregeert met als primair doel het mogelijk maken van autoschaling. Metrics server is beschikbaar op `https://github.com/kubernetes-sigs/metrics-server`. Het is ook mogelijk om externe monitoring tools te gebruiken zoals *Prometheus*.

2.7 Storage classes voor langdurige opslag

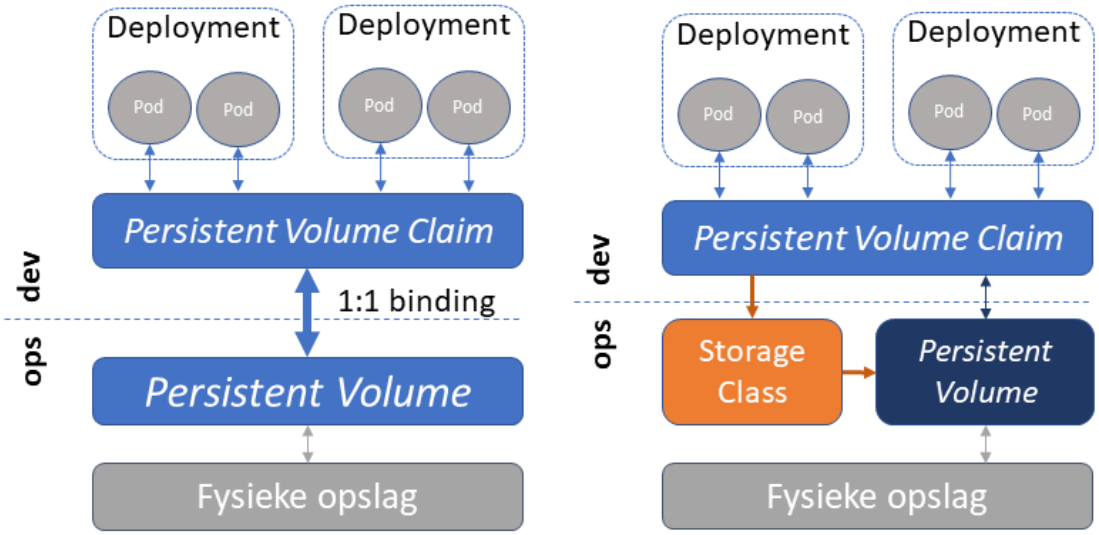
Kubernetes heeft een uitgebreid abstractiemodel om opslagvolumes aan pods te koppelen. Het verzorgen van langdurige opslag, ook gekend als persistent storage, is opgesplitst in de volgende entiteiten:

- De effectieve (netwerkgebaseerde) opslag zoals nfs, rbd, cephfs, etc.
- *PersistentVolume* **PV**: een verwijzing naar de effectieve opslag binnen Kubernetes. (ops kant)
- *PersistentVolumeClaim* **PVC**: een vraag van een Kubernetes applicatie naar een bepaald type opslag (dev kant)
- *Pods* waarin de door PVC gealloceerde volumes gebruikt worden

Elke PV kan maximaal aan één PVC gebonden zijn. Dit heeft als gevolg dat voor elke PVC die aan de developer / applicatie kant wordt aangemaakt er telkens een PV moet aangemaakt worden aan de operations / infrastructuur kant. Merk op dat het, afhankelijk van de schrijfparameters van de volume en onderliggende fysieke opslagtechniek, mogelijk is om meerdere pods en zelfs meerdere deployments

te koppelen aan dezelfde PVC. Deze standaard opstelling met manueel beheer van de PVs is voorgesteld in Figuur 3(a).

Het telkens opnieuw aanmaken van een nieuwe PV voor elke PVC kan geautomatiseerd worden aan de hand van een *StorageClass*. In deze situatie roept de PVC een bepaald type StorageClass aan die dan automatisch een dynamische PV creëert gebaseerd op diens parameters (zoals het type van opslag, welke opslagserver, het mountpad, login gegevens, etc.). Deze situatie is voorgesteld in Figuur 3(b).



(a) Manueel PV beheer (b) Dynamisch PV beheer via *StorageClass*

Figuur 3: Persistente opslag aan de hand van *Persistent Volumes* (PV) en *Persistent Volume Claims* als Kubernetes abstractielagen

We hebben op onze fysieke cloud omgeving Ceph uitgerold als gedistribueerde opslagtechnologie. De CephFS (filesystem storage) en RBD (block storage) StorageClass zijn ondersteund binnen Kubernetes maar de hiervoor benodigde *provisioners* dienen nog apart toegevoegd te worden vanaf `https://github.com/kubernetes-incubator/external-storage/tree/master/ceph`. De StorageClass parameters dienen geconfigureerd te worden aan de hand van de onderliggende ceph opstelling.

2.8 Load Balancer service - MetalLB

Kubernetes voorziet langdurige toegang tot pods via services. De *ClusterIP* en *NodePort* service zijn standaard beschikbaar als aanspreekpunten. ClusterIP voorziet een cluster-intern IP-adres als aanspreekpunt. Indien toegang van buiten de cluster gewenst is kan gekozen worden voor een NodePort service. Hierbij wordt een service extern toegankelijk gemaakt op elke node binnen de cluster op eenzelfde dynamische poort boven poortnummer 30000.

Men kan ook opteren om een IP-adres te voorzien als aanspreekpunt tot een service. Dit is mogelijk binnen Kubernetes via de *LoadBalancer* service. Deze

vereist echter nog een externe, cloud-specifieke implementatie dewelke voorzien is bij de IaaS aanbieders zoals AWS, Google Cloud, Azure, etc. maar ontbreekt bij het uitrollen van een eigen bare-metal Kubernetes omgeving. Men kan gebruik maken van *MetalLB*, een externe plugin die de elementaire LoadBalancer service taken op zich neemt. MetalLB kan werken in BGP-mode en Layer2 mode aan de hand van de ARP (IPv4) en NDP (IPv6) protocollen. Binnen MetalLB's configuratie kan men pools van IP adressen specifiëren die voor de LoadBalancer services kunnen toegewezen worden. MetalLB's project pagina is toegankelijk op `https://metallb.universe.tf/`, de projectcode zelf is beschikbaar op `https://github.com/metallb/metallb`

2.9 Inkomende HTTP(S) verbindingen nginx ingress

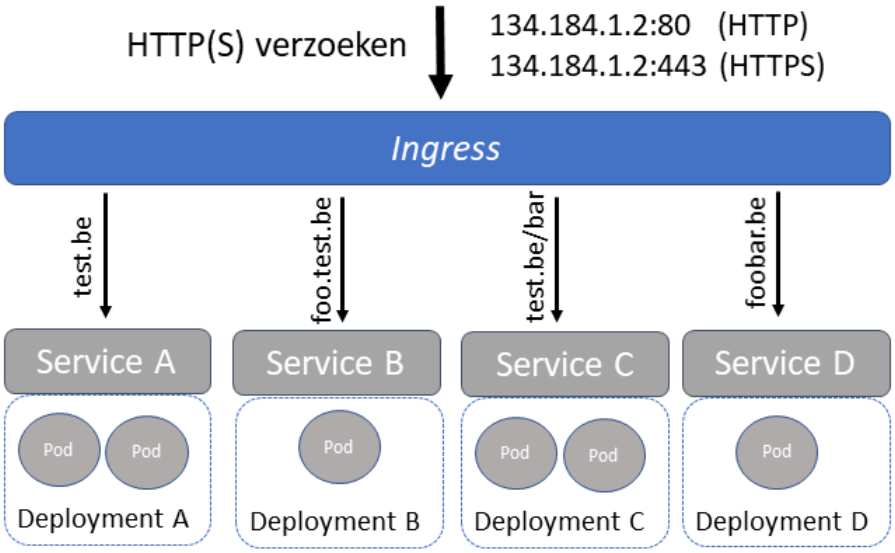
Naast de *NodePort* en *LoadBalancer* services die externe toegang naar de cluster mogelijk maken is er voor webapplicaties nog een ander soort API beschikbaar voor inkomende verbindingen: *Ingress*.

Nginx (uitgesproken als engine-X) is een populaire webservertechnologie en is ook beschikbaar als Kubernetes ingress type. De nginx ingress laat toe om aan de hand van het opgevraagde (sub)domein of directory een andere Kubernetes service door te verbinden, op voorwaarde dat al deze (sub)domeinen naar het IP-adres van de nginx ingress server (vb. 134.184.1.2) leiden. Merk op dat de nginx-ingress op zich ook uitgerold is binnen de Kubernetes cluster en toegankelijk is gemaakt via een IP-adres voorzien via de MetalLB LoadBalancer service.

De manier van werken met ingress regels is gelijkaardig aan hoe *server blocks* (binnen een fysieke nginx webserver) en *virtual hosts* (binnen een apache webserver) het mogelijk maken meerdere webapplicaties te hosten op eenzelfde machine. Hierbij wordt gekeken naar het opgevraagde (sub)domein en pad binnen de HTTP(S) verzoeken en worden aan de hand daarvan doorverbonden met de daarbij geconfigureerde applicatieservices. Het groot voordeel van werken op deze manier is dat er meerdere onafhankelijke webapplicaties kunnen voorzien worden op de gestandaardiseerde HTTP (TCP 80) en HTTPS (TCP 443) poorten op eenzelfde IP-adres. De elementaire werking van een ingress is voorgesteld in Figuur 4.

De nginx-gebaseerde ingress ondersteunt TLS-terminatie waardoor deze HTTPS communicatie op eenvoudige wijze mogelijk maakt. De hiervoor noodzakelijke certificaten en private sleutels worden mee bij de ingress regels geconfigureerd.

Documentatie voor het opzetten en gebruiken van de nginx ingress is beschikbaar op `https://kubernetes.github.io/ingress-nginx/`. Wij hebben binnen onze Kubernetes cluster twee aparte nginx-gebaseerde ingress types opgezet: de ene met slechts een intern IP-adres voor Intranetverkeer binnen ons departement, de andere verwijst naar een publiek routeerbaar IP-adres voor Internetverkeer. Bij het opzetten van een nieuwe ingress regel wordt de keuze tussen deze 2 ingress types gemaakt aan de hand van een annotatie veld, waarbij de Intranet ingress standaard is.



Figuur 4: Ingress kan de inkomende HTTP verzoeken opsplitsen naar de verschillende services in functie van het (sub)domein en pad.

2.10 HTTPS/TLS certificaatbeheer – cert-manager

Verder bouwend op de nginx-gebaseerde ingress die TLS-terminatie verzorgt is het interessant om automatisch certificaatbeheer te hebben. Geldige certificaten uitgegeven door een erkende certificaat autoriteit (CA) zijn immers essentieel. Er bestaan talloze commerciële CAs maar bijvoorbeeld ook de gratis beschikbare LetsEncrypt (`https://letsencrypt.org/`) waarbij men op verschillende methodes kan aantonen 'beheerder' te zijn van een bepaald (sub)domein door een specifiek opgedragen waarde tijdelijk ter beschikking te stellen. Deze methodes zijn gestandaardiseerd (RFC8555) als het Automatic Certificate Management Environment (ACME) protocol.

Cert-manager, beschikbaar op `https://cert-manager.io/docs/`, kan deze ACME challenges, en andere manieren, automatiseren zodat Kubernetes ingress regels automatisch voorzien worden van geldige TLS certificaten. We hebben dit geverifieerd met LetsEncrypt gebruik makende van de ACME HTTP01 methode op specifieke subdomeinen.

LetsEncrypt laat ook toe wildcard subdomein certificaten te genereren aan de hand van de ACME DNS01 methode. Deze methode maakt gebruik van het verifiëren van DNS TXT entries wat dus (tijdelijke) DNS aanpassingen vereist. Dit vraagt manuele interactie met de nameservers van het specifieke domein of interactie aan de hand van een API als deze beschikbaar is. We hebben de ACME DNS01 methode succesvol geautomatiseerd binnen cert-manager door een testdomein over te zetten naar de gratis beschikbare CloudFlare nameservers, dewelke een API aanbiedt die DNS veranderingen toelaat.

De combinatie nginx-ingress met cert-manager binnen de Kubernetes cluster biedt een érg krachtige omgeving waarbij TLS beveiligde webapplicaties snel en gro-

tendeels geautomatiseerd uitgerold worden. De cluster kan daarbij ook horizontale en/of verticale schaling van resources verzorgen en voorziet hoge beschikbaarheid voor het geval een van de nodes onbeschikbaar wordt.

2.11 Geautomatiseerd extern DNS beheer – external DNS

Het automatisch HTTPS certificaatbeheer dat mogelijk is met cert-manager is handig voor het geautomatiseerd uitrollen en onderhouden van publiek toegankelijke webapplicaties. Dit kan nog verder uitgebreid worden naar het geautomatiseerd uitrollen en beheren van publiek toegankelijke (sub)domeinen die naar de gewenste webapplicatie binnen de Kubernetes cluster verwijzen.

Dit is mogelijk door gebruik te maken van Kubernetes' *external-DNS* plugin, dewelke beschikbaar is op `https://github.com/kubernetes-sigs/external-dns`. Hierbij worden DNS-records aangemaakt die verwijzen naar het IP-adres van de ingestelde ingress of LoadBalancer service. De noodzakelijke voorwaarde voor het automatiseren van extern DNS-beheer is dat de DNS-records in kwestie op de gebruikte nameservers in te stellen zijn via een ondersteunde API. Voor onze testen hebben we succesvol gebruik gemaakt van de gratis beschikbare CloudFlare nameservers. Een lijst van ondersteunde DNS-providers en daarbij horende configuratie is beschikbaar op de git pagina van dit project.

2.12 Onze toekomstplannen met Kubernetes

Gezien onze reeds opgebouwde kennis van het Kubernetes ecosysteem, en de stevig groeiende populariteit ervan, gaan we Kubernetes zeker blijven gebruiken voor het vervolg van het project. Het staat nog niet vast hoe we dit gaan aanpakken, al is het wel essentieel om virtualisatiemogelijkheden te demonstreren. Dit kan bijvoorbeeld door het opzetten van OpenStack en Kubernetes naast elkaar, waarbij diensten zoals vb. OpenStack *Keystone* (authenticatie) en OpenStack *Cinder* (block storage) gedeeld worden. Het is ook mogelijk om Kubernetes volledig gevirtualiseerd uit te rollen binnen OpenStack.

We merken ook groeiende interesse vanaf de open source community en ontwikkelaars zoals RedHat in projecten zoals *KubeVirt*, *Virtlet* en *KataContainers*. Deze projecten hebben als doel een vorm van virtualisatie mogelijk te maken vanaf Kubernetes.

Tot nu toe hebben we enkel gebruik gemaakt van IPv4-gebaseerde netwerkcommunicatie. De komende maanden wordt dit verder uitbreidt naar dual-stack IPv4 + IPv6 netwerkcommunicatie om in de eerste plaats communicatie met onze IPv6-only IoT opstellingen, zijnde voornamelijk IEEE 802.15.4 gebaseerde draadloze sensoren actuatorknopen, mogelijk te maken. Hiervoor zijn al voorbereidende stappen genomen zoals het migreren naar een dual-stack compatibele CNI: Calico.

3 Applicaties uitgerold op de Kubernetes cloud

3.1 Opzetten eigen Docker registry

Kubernetes werkt aan de hand van containerisatie. Alvorens containers kunnen gestart worden moeten de bijhorende container images eerst beschikbaar zijn op de nodes waarop ze uitgerold worden. In het geval van Docker kunnen deze lokaal gebouwd worden vanaf de *DockerFile* broncode of kunnen deze afgehaald worden vanaf een *registry*. Dit kan gebeuren vanaf de Docker's publieke registry, genaamd Docker Hub. Om verschillende redenen kan het interessant zijn de images zelf te hosten in een private registry. Dit elimineert de noodzaak om (incrementele, nog niet gecachte) container images af te halen van een externe server op Internet.

Het opzetten van een eigen registry binnen de Kubernetes cluster zelf is praktisch en komt delay en bandbreedte ten goede wat zorgt voor het sneller uitrollen van deployments. Kubernetes verzorgt ook de hoge beschikbaarheid door de registry pod(s) te orchestreren op een nieuwe node indien de oude node faalt en kan tevens schaling verzorgen.

Het uitrollen van een private Docker registry hebben we gerealiseerd aan de hand van een deployment die de door Docker beschikbaar gestelde registry image (`https://hub.docker.com/_/registry`) uitrolt. Een LoadBalancer service voorziet connectiviteit aan de hand van een IP adres binnen ons departement's Intranet. Dit gaat vergezeld met een subdomein DNS entry om eenvoudiger te werken.

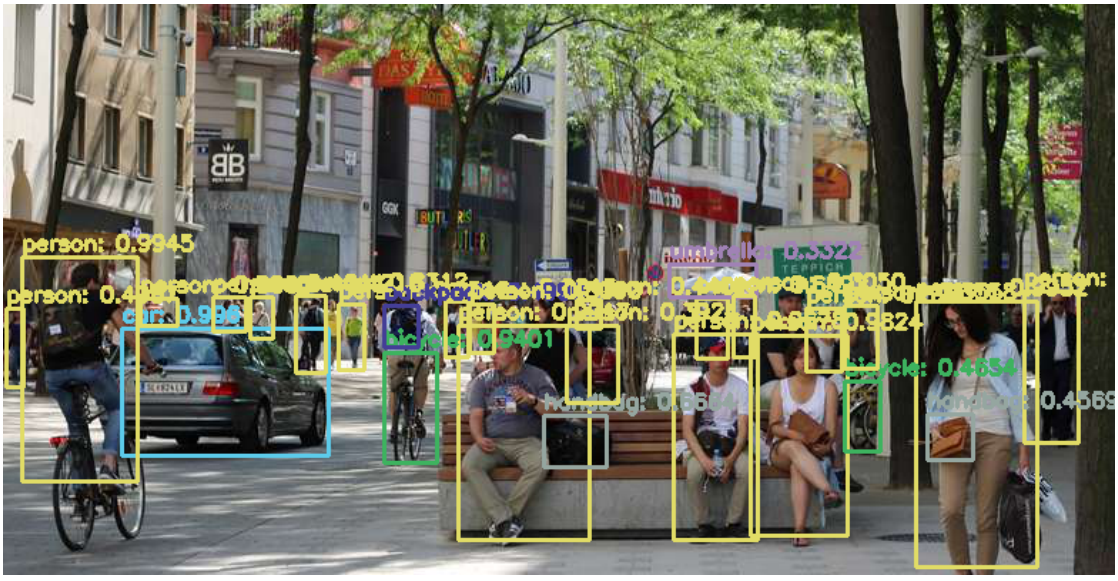
3.2 Object detectie toegepast op afbeeldingen

Voor een studentenproject is in functie van een KMO een object detectie systeem opgezet binnen Kubernetes. Er wordt gebruik gemaakt van het *You Only Look Once* (YOLO) real-time object detectie algoritme dat werkt via een neuraal netwerk. Men kan werken via gewone CPU of kan de rekenkracht verschuiven naar CUDA-geaccelereerde grafische kaarten van Nvidia.

Er is een Docker container gemaakt waarbij de objectdetectie kan getraind worden. Dit is ook geïntegreerd binnen onze Kubernetes cluster. We beschikken slechts over één workstation met voldoende krachtige Nvidia grafische kaart (RTX 2070) waardoor we het simultaan trainen op meerdere machines niet hebben getest. Wegens het ontbreken van een voldoende groot aantal aan geannoteerde afbeeldingen voor het trainen van de detectie gewichten hebben we in de plaats daarvan enige tijd getraind met de Common Objects in Context (COCO) dataset (`http://cocodataset.org`).

Er is een object detectie omgeving opgezet aan de hand van openCV in combinatie met een webapplicatie waar afbeeldingen naartoe gestuurd worden. Een afbeelding zoals Figuur 5(a) wordt geanalyseerd aan de hand van de op voorhand getrainde gewichten van de COCO dataset. De gedetecteerde objecten worden omkaderd en voorzien van objecttype en probabiliteit waarna deze afbeelding wordt teruggestuurd via de webapplicatie. Het resultaat is beschikbaar in Figuur 5(b).

(a) Input afbeelding



(b) Output afbeelding: gedetecteerde objecten omkaderd en annotatie van type en probabiliteit.

Figuur 5: Beeldherkenning toegepast met de COCO dataset

Om schaalbaarheid aan te tonen is gebruik gemaakt van zuivere CPU rekenkracht: onze Kubernetes cluster beschikt momenteel over een honderdtal threads en slechts één GPU. Binnen Kubernetes is er een horizontal pod autoscaler uitgerold die extra pods creëert in het geval van langdurige hoge belasting. Deze worden bij het normaliseren van de belasting na verloop van tijd teruggeschaald tot de standaard ingestelde hoeveelheid.

Deze applicatie is vanwege COVID-19 niet volledig afgewerkt zoals initieel voorzien aangezien het grotendeels is gerealiseerd door een uitwisselingsstudent die vroeger dan verwacht is teruggekeerd naar zijn thuisland.

3.3 ETROpy online programmeeromgeving

De ETROpy online programeeromgeving is een samentrekking van ons ETRO (electronics and Informatics) departement en py van python. Het is ontwikkeld als webapplicatie waarop studenten programmeeroefeningen kunnen maken aan de hand van een web-gebaseerde code editor die vervolgens automatisch gevalideerd worden. ETROpy is primair gericht op onze 1e bachelor studenten die python krijgen, al maakt de generieke opzet het mogelijk om voor quasi elke programmeertaal te valideren. Zo zijn momenteel Python 2, Python 3, Java, C, C++ en C# beschikbaar.

De door studenten opgestuurde oplossingen worden gevalideerd aan de hand van verschillende test-cases en worden vergeleken met de op voorhand vastgelegde uitkomst. Studenten kunnen eender welke code doorsturen, dus ook potentieel gevaarlijke code. Om problemen hieromtrent te vermijden en de overhead minimaal te houden, is gekozen voor de geïsoleerde omgeving die containers bieden. Dit biedt tevens de transparantie om verschillende programmeertalen te integreren en vermijdt het quasi onmogelijke hanteren van programmeertaal-specifieke trucs om onveilige code uit te schakelen.

In oudere versies van het project wordt hiervoor zuiver gebruik gemaakt van Docker, die rechtstreeks wordt aangestuurd vanaf de machine waarop de ETROpy webapplicatie staat. Dit werkt goed maar is beperkt tot de resources beschikbaar op de gegeven machine waardoor het geen schaalbare oplossing is.

In de huidige versie is het rechtstreeks aanspreken van Docker vanaf de in python geschreven webapplicatie vervangen door interactie met de Kubernetes cluster. Hiervoor is gebruik gemaakt van Kubernetes' officiële python client (`https://github.com/kubernetes-client/python`).

Binnen Kubernetes is de *Namespace* "etropy" aangemaakt. Er is een *ServiceAccount*, *Role* en *RoleBinding* aangemaakt waarbij de nodige rechten zijn toegekend om pods te beheren binnen de "etropy" namespace. Het token dat bewaard is als *Secret* van de *ServiceAccount* kan geëxtraheerd worden om toegang te geven tot de Kubernetes cluster. Dit biedt de mogelijkheid om pods te beheren binnen de "etropy" namespace vanaf de ETROpy webapplicatie. Binnen deze namespace is de *NetworkPolicy* ingesteld om *ingress* en *egress* verkeer te weigeren wat het verder isoleren van de ETROpy validatieomgeving garandeert. Volgende stappen worden doorlopen bij het doorsturen van een oplossing ter validatie:

- Aanmaken pod met container image van de gewenste programmeertaal omgeving (vb. python, gcc, java, mono) en wachten tot deze beschikbaar is.
- Overbrengen broncode oefening naar de pod. Dit kan oftewel via een gedeelde PersistentVolume, of via het doorsturen van een file write of append via *stdin* in een interactieve connectie.
- Compileren van de broncode indien nodig. Eventuele *stderr* compilatiefouten, responscodes en timeouts worden afgevangen waardoor het valideren wordt geannuleerd.

- Valideren van de test-cases: ø
  - De test input wordt doorgestuurd.
  - Het testprogramma wordt uitgevoerd op deze test input.
  - Eventuele *stderr*, responscodes en timeouts worden afgevangen. Indien er geen fouten zijn wordt de stdout test output vergeleken met het gekende resultaat.
  - Stoppen bij een fout resultaat of doorgaan met de volgende test-case.
- Verwijderen van de pod en resultaten verwerken.

De ETROpy webapplicatie zelf kan ook gemigreerd worden naar de Kubernetes cluster in plaats van rechtstreeks op een machine te draaien. Hiervoor moet een eerst container image gemaakt worden die alle benodigde bibliotheken bevat (e.g. python, django, python's kubernetes client, etc.) en de ETROpy webapplicatie zelf. De configuratie specifieke parameters zoal mailinstellingen, Kubernetes API en token, debug opties, etc. worden omgezet naar omgevingsvariabelen zodat deze eenvoudig door te geven zijn aan de container.

Een Kubernetes *Deployment* verzorgt het uitrollen van de ETROpy webapplicatie met de gepaste omgevingsvariabelen. Wanneer men werkt binnen dezelfde "etropy" namespace kan men het benodigde *ServiceAccount Token* rechtstreeks inladen; dit token hoeft dus niet manueel overgenomen te worden als omgevingsvariabele. Een *PersistentVolumeClaim* van de *StorageClass Ceph RBD* is voorzien om de opgaven, test-cases, en oplossingen van de studenten te bewaren alsook de webapplicaties' database. Een Kubernetes Service voorziet toegang tot de relevante pod(s). Er is gekozen voor een *ClusterIP* type Service in combinatie met een nginx gebaseerde publiek toegangelijke *Ingress* op `https://etropy.etrovub.be`. Deze Ingress staat ingesteld om aan automatisch TLS-certificaat beheer te doen aan de hand van Kubernetes' cert-manager uitbreiding.

Als de webapplicatie wordt uitgerold binnen de "etropy" namespace waarin de studentencode reeds wordt geëvalueerd, dan is het van belang om ook een extra *NetworkPolicy* uitzondering toe te voegen om *Ingress* en *Egress* verkeer toe te laten naar de benodigde pods: de webapplicatie en eventuele tijdelijke cert-manager ACME-client pods. Dit is nodig omdat de reeds standaard voorziene NetworkPolicy alle netwerkverkeer blokkeert.

3.4 Cryptografische demoapplicatie en Gitlab

Voor lopend onderzoek naar nieuwe cryptografische technieken binnen onze groep wordt er momenteel een demoapplicatie ontwikkeld. Deze bestaat uit meerdere componenten en applicaties. De broncode van deze componenten wordt gehost op een interne Gitlab server en wordt door de Gitlab CI automatisch gebouwd, getest en uiteindelijk ook uitgerold op onze Kubernetes cluster. Het automatisch bouwen en testen gebeurt ook op Kubernetes: de zogenaamde *Gitlab runner* schaalt automatisch over de infrastructuur. Het automatisch uitrollen stelt in staat om kort op de

bal te spelen tussen de verschillende programmeurs: een aanpassing in een “core” library heeft onmiddellijk effect in een testomgeving.

Eén van de componenten bestaat uit een webservice die een aantal cryptografische validaties moet uitvoeren en de resultaten opslaat in een PostgreSQL database. De database staat opgeslagen op een *Ceph RBD* block opslag volume. Om de schaalbaarheid van de demoapplicatie en de onderliggende technieken te meten, zal ook hier de autoscaling van Kubernetes worden toegepast. Hierbij kijken we ook naar schaalbaarheid met een heterogene verzameling van CPU en GPU: de cryptografische workload kan vermoedelijk sterk versneld worden op grafische processors, waarvoor via een ander project geld werd vrijgemaakt. We zullen de komende maanden de servers uitbreiden met drie mid-range GPUs, in eerste plaats om de performance op GPU te kunnen meten, maar ook om te bekijken hoe een heterogene omgeving, zijnde een mix van AMD processoren, AMD GPUs en een Nvidia GPU (RTX 2070 in een workstation) optimaal kan benut worden voor dit soort toepassing.

4 Sidetrack: Network Function Virtualization

Network Function Virtualization (NFV) is een concept waarbij men network services die traditioneel geïmplementeerd zijn op dedicated hardware tracht te vervangen door gevirtualiseerde versies. NFV wordt vaak vermeld in de context van telcos, en kan in dat opzicht worden beschouwd als de toepassing van cloud concepten en technologieën op de telco infrastructuur.

De voordelen van NFV zijn gelijkaardig aan deze van cloud platformen: schaalbaarheid, flexibiliteit, snelle uitrol, etc. Door het gebruik van virtualisatie op COTS servers kan men de hardware efficiënter benutten. Het virtualiseren van network services is bovendien bevorderlijk voor innovatie, wat in de telco wereld traditioneel geleidelijker gebeurt.

NFV speelt ook een rol in 5G. Naast de nieuwe specificaties voor de radio interface, die voor een grotere bandbreedte en lagere latency moeten zorgen, wordt ook de architectuur van het core netwerk in 5G vernieuwd en zal deze op een cloud native manier worden geïmplementeerd. NFV moet toelaten om de 5G core infrastructuur gemakkelijk te schalen, en dient ook als een platform voor het uitrollen van de vele nieuwe diensten die 5G mogelijk zou kunnen maken.

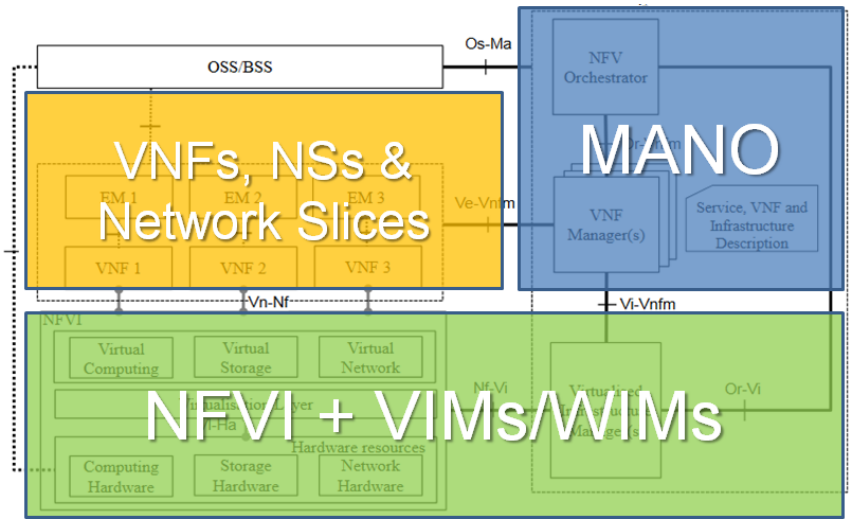
Enkele voorbeelden van network services die met behulp van NFV gevirtualiseerd kunnen worden:

- traditionele netwerk services: routers, firewalls, load balancers, content delivery networks
- 5G packet core en nieuwe services zoals edge computing, AR/VR, etc.
- deployen van orchestration engines zoals kubernetes

4.1 Architectuur

Figuur 6 geeft de algemene architectuur weer die gebruikt wordt in NFV. Deze kan opgedeeld worden in 3 componenten. De verantwoordelijkheid van de verschillende componenten en de interfaces tussen de componenten zijn beschreven in specificaties uitgebracht door ETSI.

- De NFV infrastructure (NFVI) biedt de eigenlijke compute, storage, en network middelen die nodig zijn voor het aanmaken van VMs en containers die deel uitmaken van de VNFs. Virtual Infrastructure Managers (VIMs) zijn verantwoordelijk voor het aanmaken van VMs en containers op de NFVI en dienen als een plugin tussen NFVI en MANO. De WAN Infrastructure Manager (WIM) kan gebruikt worden voor het opzetten van connectiviteit tussen datacenters of zelfs vershillende VIMs, door bijvoorbeeld gebruik te maken van MPLS.
- Virtual Network Functions (VNFs) zijn de bouwstenen van NFV. VNFs bundelen de VM of container images die bepaalde functionaliteit bieden, samen



Figuur 6: Algemene architectuur binnen Network Function Virtualization

met metadata over de vereiste resources zoals geheugen, rekenkracht, en connectiviteit voor deze VMs of containers. Bovendien kunnen ook operaties, in de vorm van scripts, voor configuratie en monitoring worden gedefiniëerd. Network Services (NS) zijn samenstellingen van één of meerdere VNFs, en laten zo toe om meer complexe functionaliteit op te bouwen. Network slices maken deel uit van de 5G netwerkarchitectuur, en laten toe om de fysische netwerk infrastructuur op te delen in geïsoleerde partities met verschillende quality-of-service vereisten. NFV ondersteunt het aanmaken van dergelijke network slices.

- De Management and Orchestration (MANO) component is verantwoordelijk voor het beheer van de life-cycle, de configuratie en de monitoring van de VNFs, NSs, en network slices.

NFV kan geïntegreerd worden in reeds bestaande OSS/BSS systemen via de northbound interface.

4.2 Open Source MANO (OSM)

Er bestaan verschillende platformen die NFV implementeren. De 2 grote spelers zijn Open Networking Automation Platform (ONAP) een Linux Foundation project, en ETSI's Open Source MANO (OSM). Cloudify, Rift zijn 2 alternatieven die verder werken op ONAP en OSM en zijn ook commerciële producten die support aanbieden. We kozen voor OSM als onderwerp voor een verdere analyse aangezien het de referentie implementatie is van de ETSI NVF specificaties.

De installatie van OSM gebeurt met behulp van een installatie script dat de nodige dependencies installeert, en een Docker swarm of Kubernetes cluster aanmaakt waarop de OSM services uitgerold worden. De optie voor het deployen op een reeds bestaande cluster wordt niet beschreven in de documentatie.

Het installatie script laat ook toe om optioneel vim-emu te installeren, een multi-PoP emulatie platform dat gebruikt kan worden voor het lokaal ontwikkelen en prototyping van network services. Er zijn een aantal beperkingen aan dit emulatie platform. Zo biedt het enkel ondersteuning voor container gebaseerde VNFs en kunnen day-1 en day-2 operaties voor configuratie en management niet toegepast worden.

Na de installatie moet een VIM geconfigureerd worden, waarop de network services kunnen geïnstantieerd worden. OSM biedt hiervoor ondersteuning voor OpenStack, VMware vCloud Director, Amazon Web Services, Microsoft Azure, Eclipse fog05, en vim-emu. Tot op heden hebben we enkel de interactie met vim-emu getest, maar we plannen ook de integratie met OpenStack verder te onderzoeken.

4.3 Kritiek

De eerste white paper uitgebracht door ETSI waarin de NFV visie werd beschreven dateert reeds van 2012. Hoewel werd aangegeven dat NFV een revolutie teweeg zou brengen in de telco wereld, is daar tot nog toe niet veel van waargenomen. Er zijn hiervoor een aantal redenen.

De stabiliteit en functionaliteit van de platformen voldoet nog niet aan de vereisten van productie omgevingen. Voor OSM specifiek ontbreken bijvoorbeeld nog de ondersteunging voor het automatisch plaatsen van VNFs in een geoptimaliseerde manier, en integratie met container orchestratie platformen, zoals kubernetes. Aan veel van deze features wordt echter wel actief gewerkt. Bovendien worden de specificaties uitgebracht door ETSI nog steeds verder ontwikkeld.

Verder wordt er ook geopperd dat virtualizatie op basis van COTS hardware niet de nodige quality of service kan voorzien die vereist is voor het functioneren van bepaalde network services. Enhanced Platform Awareness kan hierop een antwoord bieden, door gebruik te maken van technologieën zoals cpu pinning, NUMA topology awareness, huge tables, PCIe passthrough, SR-IOV, etc. die latency en throughput kunnen bevorderen.

4.4 Planning

Momenteel bekijken we de integratie van OSM met de laatste OpenStack release, Ussuri, die net is uitgebracht. Het gebruik van OpenStack, in plaats van het vim-emu platform, zal ook toelaten om meer complexe network services te testen. Daarnaast zullen we ook het gebruik van charms voor day-1 en day-2 operaties verder bestuderen.

5 Besluit

Op het moment van testen blijkt de tot dan toe recentste OpenStack versie *train* onvoldoende compatibel met de nieuwste generatie besturingssystemen. Daarom is er gekozen te wachten op de OpenStack Ussuri versie, welke sinds halfweg mei 2020 beschikbaar is en compatibel is met onder meer CentOS/RHEL 8 en Ubuntu 20.04 LTS. In afwachting is er een Kubernetes cluster uitgerold die container orchestratie voorziet, in ons geval voor Docker-gebaseerde containers. Na een primaire evaluatie van Kubernetes gaan wij ervan uit dat dit voor een groot deel van de KMO doeleinden volstaat waardoor het uitrollen van een volwaardige virtualisatie oplossing zoals OpenStack optioneel is. OpenStack of vergelijkbare oplossingen blijven wel essentieel wanneer volwaardige virtualisatie nodig is, al zijn er momenteel ook grootschalige projecten zoals KubeVirt met als doel virtualisatie vanaf Kubernetes mogelijk te maken.

Dit verslag voorziet de belangrijkste concepten om aan de slag te gaan met Kubernetes. Een beknopte installatiegids is voorzien en de door ons uitgerolde plugins zijn beschreven. De door ons tot nu toe uitgerolde applicaties binnen onze Kubernetes cluster zijn ook beschreven.

De komende periode gaan we ons toespitsen op meerdere aspecten. Kubernetes communicatie aan de hand van IPv6 zal geëvalueerd worden. We zullen evalueren in welke mate het mogelijk is om aan virtualisatie te doen aan de hand van projecten zoals KubeVirt en gaan dit vergelijken met een volwaardige OpenStack oplossing. Aangezien de noodzakelijke cloud infrastructuur in eigen beheer nu voorhanden is gaan we ook de eerste stappen zetten richting uitbreiding naar de edge en interactie met publieke cloud.