Témalabor beszámoló

Távközlési és Médiainformatikai Tanszék

Készítette:	**Madács Botond Pál**
Neptun-kód:	**I9VGKZ**
Ágazat:	**Felhő alapú hálózatok**
E-mail cím:	**madacsbotondpal@gmail.com**
Konzulens(ek):	**Dr. Maliosz Markosz**
	Dr. Simon Csaba
E-mail címe(ik):	**maliosz@tmit.bme.hu**
	simon@tmit.bme.hu

Téma címe: Felhő alapú hálózatok

Feladat

Virtualizációs tecnhológiák megismerése, tanulmányozása és gyakorlatban történő kipróbálása..Egy Nginx alapú webszolgáltatás létrehozása, Docker konténer technológia segítségével, majd a Kubernetes konténer menedzsment rendszerbe telepítése és végül OpenStackbe felett kialakítani az infrastruktúrát.

2019/2020. 1. félév

Virualizációs technológiák áttekintése

Operációs rendszer szintű virtualizáció

Más néven ez a konténer alapú virtualizáció. A konténeres virtualizáció új lehetőségeket nyitott a virtuális gépek szervezésében. Ebben a konténerek szervezése szolgáltatás alapon történik, minden szolgáltatásnak konténert, saját környezetet definiálunk, amelyek között a célszerűségnek megfelelően linkeket képzünk. Például létrehozhatunk egy webservert, ami egy konténeren fut és hozzá egy adatbázis-konténert. A konténerek többszörözése szükség esetén könnyen megtehető, így egy adott szolgáltatást pillanatok alatt több példányban futtathatunk. Pár operációs rendszer szintű virtualizációs megoldás, BSD-ken: Jail, Linux alatt: OpenVZ, Windowson: Parallels Virtuozzo. Előnye, hogy nagyon kis költségű, konténerenként nincs szükség további vendég kernelre. Erőforrás gazdálkodási szempontból is problémamentes. Viszont biztonsági szempontból kevésbé jó az izoláció. A hypervisoros megoldással szemben nincs virtuális hardver, egy kernelen fut és több userspace példánnyal, jobb erőforrás kihasználás, közel natív teljesítménnyel.

A konténerek támogatása nagyrészt az OpenVZ és Linux-Vserver projektnek köszönhetően került be a Linux kernelbe. A folyamatok szeparálása 6 nagy területre bontható szét, **mnt: a** fájlrendszer különböző “nézeteinek” biztosítása, **pid:** folyamatok azonosítása, eltérő környezetben előfordulhat ugyanaz a PID (program azonosító szám), mint a gazda gépen, **net:** hálózati kapcsolat szeparálása saját IP cím, forgalomirányítási (routing) táblázat és virtuális hálózati eszközök használatával, **ipc:** folyamatok közti kommunikáció, **uts:** hosztnevek szeparációja, **user:** a felhasználó és csoport azonosítók szeparálása. Tehát a Linux megoldása szerint a virtuális gépek semmi másból nem állnak, mint korlátozott, gazda operációs rendszerből indított folyamatokból. Emiatt van lehetőségünk akár egyetlen programot is indítani.

Docker

Docker egy Open Source projekt, első verziója 2013. márciusában jelent meg. Közvetlen elődjének a 2008-ban megjelent LXC (Linux containers) tekinthető. Az LXC egy parancshalmaz, amely konténer alapú, de teljes virtuális gépeket kezel. Ezzel szemben a Docker erőfeszítéseket tett a konténerformátumok egységesítéséért és az egyetlen program kedvéért létező konténerek működéséért. Ez a megközelítés lehetővé teszi a különböző szerepű virtuális gépek igény szerinti elindítását vagy leállítását, ezért könnyebb alkalmazkodni a változó terhelésű rendszerek hullámzó kihasználtságához. Nagyobb forgalom esetén újabb virtuális gépeket indíthatunk, amikor pedig csökken a forgalom, akkor leállíthatjuk ezeket. Emiatt egy-egy szerver jobb teljesítménnyel működhet, kevesebb fizikai eszközre van szükség, jobb helykihasználtságot tesz lehetővé, kevesebb eszköz miatt kevesebb karbantartásra van szükség, ami csökkenti az esetleges hibaforrások számát is.

A virtualizáció során technikai szempontból figyelembe kell venni a használt architektúrát, például az x86-os architektúra nem volt könnyen virtualizálható, mert a 250 utasításából 17 megsérti a klasszikus feltételeket. A Docker alapvetően két fő részre bontható, a Docker Engine a hoszt gépen, a szerveren fut, feladata a konténerek kezelése, futtatása, illetve a menedzselése. A

Docker Hub pedig egy felhő alapú szolgáltatás, ami a képfájlok publikálását teszi lehetővé, GitHub szerű funkcionális palettával rendelkezik. Ez lehet akár a hoszt gépen is vagy egy távolról elérhető gépen is.

Működése és tesztelése nagyon gyors, másodpercek alatt fordul és utána azonnal fut. Ha a lokális buildje sikeres, akkor fel lehet tölteni a Docker Hubba vagy más privát registry-be. Hiba esetén rollback alkalmazható, azaz vissza lehet térni egy korábbi verzióra.

Docker konténereket kézzel is létre lehet hozni, amiket utána be lehet állítani manuálisan, de ezzel az az egyik nagy baj, hogy időigényes és nehézkes, hiba esetén sok időbe telhet megtalálni a hiba okát és utána javítani azt. Sok-sok konténer esetén ez nem alternatíva. Másik megoldás a konténerek létrehozására az úgynevezett Dockerfile elkészítése. Ennek a segítségével szöveges formában megadható, hogy milyen parancsokat hajtson végre a Docker.

Kubernetes

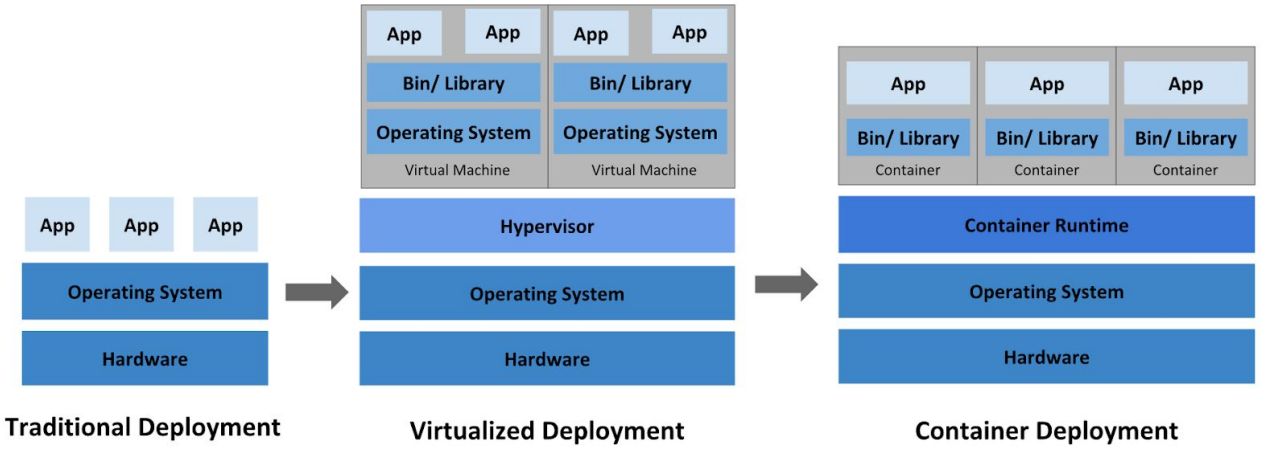
A Kubernetes (K8s) egy hordozható, open-source platform a konténerizált terhelés és szolgáltatások menedzselésére, ami megkönnyíti a dekleratív konfigurációt és automatizációt is. Nagy és gyorsan növő ökoszisztémával rendelkezik. A Kubernetes név egy görög szóból jön, aminek a jelentése hajóskapitány. A Google 2014-ben tette nyílt forráskódúvá a projektet.

Ahhoz, hogy megértsük, hogy miért hasznos a Kubernetes, vissza kell menni egy kicsit az időben. Kezdetekben a vállalatok minden alkalmazást fizikai szervereken futtattak, ez több okból is probléma volt. Nem volt minden egyes alkalmazásnak megszabva, hogy mennyi erőforrást használhat, ezért például megtörténhetett az is, hogy egy alkalmazás az összes erőforrást lefoglalta és elvette más alkalmazásoktól, amik így nem tudtak normálisan működni. Erre megoldás lehet, hogy egy-egy alkalmazás különböző fizikai szerveren futott, de ez nem túl hatékony megoldás és nagyon költséges lehetett.

Erre megoldást jelentett a virtualizáció, ennek a segítségével több virtuális gépet (Virtual Machine, VM) lehetett futtatni egy fizikai szerveren. A virtualizáció segítségével a futó alkalmazások egymástól izolálva futottak, ez jobb védelmet is jelentett, mert alkalmazások között az információ nem áramolhat át szabadon. A virtualizáció jobb kihasználtságot és skálázhatóságot tett lehetővé, mert könnyedén lehetett újabb példányokat elindítani vagy leállítani, amivel jobb processzorkihasználtság érhető el. Minden VM saját operációs rendszerrel rendelkezik, ami virtuális hardverre épül.

A következő lépcsőfoka a fejlődésnek az OS alapú virtualizáció (konténer alapú virtualizáció). A konténeres megoldás hasonlít a VM-re, de lazítottak az izoláción és közös operációs rendszert használnak az alkalmazások. Emiatt a konténerek sokkal „könnyebbek”. Hasonlóan a VM-hez a konténereknek meg van saját filerendszerük, processzoruk és memóriájuk. A népszerűségének és gyors elterjedésének okai: Agilis alkalmazás készítés és telepítés, folyamatos fejlesztés, integráció, könnyű megfigyelhetőség, nem csak OS szintű információkról és metrikákról, környezetfüggetlen fejlesztés, applikáció-centrikus menedzsment, erőforrás izoláció, erőforrás kihasználás.

Tehát a konténerek jó megoldást jelentenek, hogy alkalmazásokat futtassunk, de ezekre a konténerekre figyelni is kell, hogy megbizonyosodjunk afelől, hogy minden helyesen működik és nincs leállás. Például mi történik, akkor, ha egy konténer működése leáll és egy másik konténert kellene helyette indítani? Erre jött válaszként a Kubernetes, ami egy frameworkot kínál, hogy elosztott rendszereket rugalmasan futtathassunk. A K8s gondoskodik a skálázódásról, kiesés esetén indít újabb konténereket. A K8s-el lehetőségünk van a load balancera, tárhely orkesztrációra, automatizált rolloutsra és rollbackre, önjavításra hiba esetén.



1. ábra Deployment változásai

Feladat kidolgozása

1. Docker telepítése

A Dockert egy Ubuntu 18.04 LTS operációs rendszert futtató virtuális gépre telepítettem, ami VirtualBoxon ment. Docker telepítésének lépései a hivatalos dokumentációban megtalálható, ebből elkészített scriptemmel gyorsan és egyszerűen telepíthetővé tettem az újabb gépekhez.

A Docker telepítésének a lépései a következőek:

I. Ha van már feltelepítve előző verzió akkor azt letöröljük

II. Az apt-get update segítségével a csomagok helyi indexének/adatbázisának frissítése

III. Az apt-transport-https, ca-certificates, curl, software-properties-common csomagok telepítése

IV. Docker GPG kulcsának telepítése és a kulcs verifikálása

V. Docker repository hozzáadása az apt forrásokhoz

VI. A docker-ce, docker-ce-cli és containerd.io csomagok telepítése

VII. Egy konténer letöltése és futtatása, ha sikerült, akkor a docker telepítése megtörtént, ha nem sikerül futtatni, akkor valami probléma van

A Docker telepítés után mindig a sudo paranccsal használtam, nem root felhasználóként belépve. Ez elkerülhető lehet úgy, hogy minden Dockert használó felhasználót hozzáadjuk a Docker csoporthoz. Ez veszélyes lehet, mert ezután lehetősége van a felhasználónak, hogy konténereket indítson vagy menedzseljen, amivel root jogokat szerezhet a hoszt gépen.

2. Docker konténer/dockerfile létrehozása

A webszerver elkészítéséhez az nginx webszerver alkalmazást használtam alapnak. Ahhoz, hogy Docker konténert létre tudjunk hozni, kell egy Dockerfile-t készíteni, amiben leírjuk azokat a lépéseket, amik a konténer létrehozásához szükségesek. A Dockerfile első lépése, hogy letöltsük az nginx-et, majd létrehozunk egy mappát, amiben a weblapunk adatait tároljuk. A következő lépésben felcsatolunk egy volume-ot, hogy a konténer leállítása után is megmaradjon minden adatunk. Ezután kitöröljük a weblap default.confját és átmásoljuk a sajátunkat. Ezután rekurzívan átmásoljuk a weblap tartalmát a hosztról a konténerbe, majd jogokat adunk erre. Végső lépésként megnyitjuk a 80-as portot a HTTP, 443-as portot pedig a HTTPS forgalom számára. A docker build segítségével ezután elkészítettem az image-et, amit feltöltöttem a Docker hubra. (Docker hubon madacsbotondpal/nginx-re rákeresve megtalálható.)

3. Kubernetes telepítése

Szükség volt második virtuális gépre, ehhez az előzőekben használt virtuális gépet szerettem volna klónozni. Ezzel több problémám is volt, a MAC címeket manuálisan kellett cserélni, mert a VirtualBox azokat is lemásolta. Ezután abból volt problémája a Kubernetesnek, hogy megegyeztek a gépnevek és nem tudott rácsatlakozni a masterre a slave vagy nem találta meg a master clustert a slave. Ezen problémák elkerülése miatt telepítettem inkább egy friss ubuntu-t, amin az előzőleg megírt docker scriptet lefuttattam. A K8s telepítéséhez is egy scriptet használtam. Ebben a scriptben az alábbi lépések voltak: a swap kikapcsolása, a csomagok helyi indexének frissítése, K8s GPG kulcsának telepítése és ellenőrzése, a csomagok helyi indexének ismételt frissítése, majd ezután három csomag telepítése, név szerint: kubelet, kubeadm, kubectl.

Miután mindkét gépen telepítettem a K8s-t, el kellett indítani egy master node-ot. A klaszter hálózathoz az úgynevezett Weave net alapú virtuális hálózatot használtam. A masternek szánt node-on elindítottam egy scriptet, aminél megadtam a master IP címét. Ez a script lefuttatja a kubeadm initet, létrehozza a .kube mappát és bemásol egy config fájlt. Miután a script lefutott kiír egy parancsot, amivel a slave-ekkel tudunk kapcsolódni majd a masterre. (Ezt ajánlatos elmenteni, mert ezzel tudunk csak rácsatlakozni a masterre.)

```
kubeadm join 10.0.2.15:6443 --token  --discovery-token-ca-cert-hash
```

Miután lefuttattuk a slave-en vagy slave-eken a parancsot megnézhetjük, hogy milyen állapotban vannak a node-jaink. Láthatjuk, hogy futnak-e, mi a nevük és, hogy mikor indultak el.

```
osboxes@master:~/Docker/Kubernetes$ sudo kubectl get nodes
NAME      STATUS   ROLES    AGE    VERSION
master    Ready    master   7d2h   v1.16.3
osboxes   Ready    <none>   7d     v1.16.3
```

4. Kubernetes pod, deployment és loadbalancer készítése

A 2. feladatrészben elkészült docker image-ből készíthetünk egy K8s podot. Ehhez egy egyszerű yaml fájlt kell írni. Ez a yaml fájl tartalmazza a pod nevét, az indítani kívánt docker image nevét, a konténer nevét és azt is, hogy milyen portokat szeretnénk kinyitni. Miután elkészült a yaml fájlunk a *kubectl apply –f <nev.yaml>* parancsot lefuttatva elindíthatjuk azt, ha sikeres volt az indítás, akkor a *kubectl get pods* vagy a *kubectl get all* paranccsal megnézhetünk. Ha ott megjelenik, akkor utolsó lépésként a *curl <master_IP>:<port>* paranccsal megnézhetjük, hogy mit tartalmaz a weblapunk.

Ha sikeres volt minden, akkor következő lépésként el kell készítenünk egy deploymentet, ami az előző pod.yaml-hoz hasonló. A legnagyobb különbség, hogy itt megadhatjuk a replikák számát, ami megadja, hogy hány pod induljon el.

Azért, mert több podot hoztunk létre szükségünk volt valamire, ami elosztja a terhelést a podok között és jobb kihasználtságot eredményez. Erre a legjobb megoldás egy loadbalancer létrehozása, amit a service Kubernetes objektum tud biztosítani. A loadbalancert is egy yaml fájlban kell specifikálni. Ebben meg kell adnunk externalIP-nek a master node-unk IP címét, és azt, hogy melyik portot vagy portokat szeretnénk megnyitni, amin keresztül elérhetővé válik a szolgáltatásunk.

Fontos, hogy a podok közt perzisztencia legyen, és a tartalmuk megegyezzen, és a podok leállítása után is megmaradjanak az ezekből származó logok. A perzisztencia megoldására több lehetőségünk is van. Az én megoldásom az NFS (Network File System) alkalmazása volt. A master node-ra fel kellett telepíteni az nfs-kernel-servert. Miután ez megtörtént létre kellett hoznunk egy mappát, amit szeretnénk majd megosztani. Ha kész van a megosztott mappa akkor megadhatjuk, hogy mely IP címek érhetik el. Ha a beállítás megtörtént, akkor utolsó lépésben újraindítjuk az nfs-kernel-servert. Ha ezzel kész vagyunk, akkor az összes slave-re, amin szeretnénk elérni a weblap tartalmát, feltelepítjük az nfs-commont. Miután mindennel elkészültünk a megosztott mappát felcsatoljuk az összes podhoz és azok elérik az azonos tartalmat.

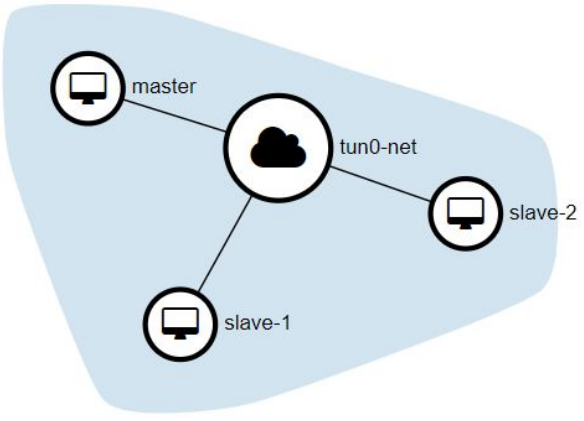
```
osboxes@master:~/Docker/Kubernetes$ sudo kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginxserver-57dcb9bdf4-4ndzm    1/1     Running   0          33s
pod/nginxserver-57dcb9bdf4-6kszs    1/1     Running   0          33s
pod/nginxserver-57dcb9bdf4-h2k7q    1/1     Running   0          33s
pod/nginxserver-57dcb9bdf4-mkhhw    1/1     Running   0          33s
pod/nginxserver-57dcb9bdf4-q98pk    1/1     Running   0          33s

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
 AGE
service/kubernetes   ClusterIP      10.96.0.1       <none>        443/TCP
 2m4s
service/web-service  LoadBalancer   10.108.102.87   10.0.2.15     80:31007/TCP
 30s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginxserver    5/5     5            5           33s

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/nginxserver-57dcb9bdf4    5         5         5       33s
```

5. OpenStack deployment

Utolsó lépésként az eddig elkészített rendszert feltelepítjük egy OpenStack által futtatott virtuális gépekből álló klaszterbe. Elsőnek fel kellett tölteni egy Ubuntu image-t, amire a 16.04 LTS-t használtam. Ebből az image-ből létrehoztam 3 virtuális gépet, az egyik lesz a master, a többi pedig a slave. Az OpenStackben megadható, hogy mennyi erőforrást használhasson, melyik helyi hálózatot használják és, hogy milyen SSH kulccsal lehessen felcsatlakozni rá.

Ezek után a master node kapott egy floating IP címet, hogy elérhessem a saját gépemről is, ne csak belső hálózatról. Valami oknál fogva nem sikerült az első SSH kulccsal csatlakozni a slave-ekre, ezért a masteren létrehozott SSH kulccsal csatlakoztam a másik két gépre.

Miután az összes gépet el tudtam érni letöltöttem githubról az eddig használt scriptjeimet. Mindegyik gépen lefuttattam a docker telepítését, majd a K8s telepítését. A masteren ezután elindítottam az API szervert és rácsatlakoztam a slave-ekkel is. Miután mindez megtörtént a *kubectl apply –f <vmi.yaml>* paranccsal lefuttattam a dep.yaml-t és a loadbalancer.yaml-t, amivel az egész szolgáltatás elindult.



Összefoglalás

A félév során (szerintem) hátrányból indultam, mert kevés Linux ismerettel csöppentem bele egy olyan környezetbe, ahol többnyire minden Linuxon fut. Úgy gondolom, hogy sokat tanultam és az ebből fakadó hátrányomat sikerült mérsékelnem. Ehhez hozzátartozik még, hogy új dolgokat tanultam, köztük a Dockerről és Kubernetesről is rengeteg dolgot tudtam meg. A félév végére feladatnak kaptuk, hogy készítsünk egy webszolgáltatást. Ehhez arra volt szükségem, hogy végiggondoljam azt, hogy mit tanultam a félév során és apró lépésekben építsem meg a szolgáltatást. Az elkészített szolgáltatás képes lenne élesben is kiszolgálni egy weblapot. (Persze ehhez néhány dolgot változtatni kellene.) A githubra feltöltött scriptek és a Docker Hubra feltöltött image segítségével bárhol, bármikor, gyorsan felépíthető a szolgáltatásom. Külön köszönet témalaboros hallgatótársamnak Kovács Leventének, hogy türelmes volt és válaszolt minden kérdésemre.

Irodalomjegyzék

Saját scriptjeim, dockerfile

https://github.com/madacsbotondpal/nginxtemalab

https://hub.docker.com/repository/docker/madacsbotondpal/nginx

Felhasznált források

Kubernetes telepítése és master futtatása: https://mediacaching.blog.hu/2018/09/12/kubernetes_hpa_beallitasa?token=8e4b8499ac4a8aa57a59c14b943ba8dd&fbclid=IwAR1iq-5uN10v7zFyzqKNiQ8wuPpeaYhwXwfF-aJzdHfh3NZLy8b4f_93198

Kubernetesről információ: https://kubernetes.io/docs/

Docker telepítés, információk: https://docs.docker.com/install/