Docker Swarm Classic

<< Back to Docker main

Warning
Minden amit itt leírok már elavult. A swarm 1.0, vagy swarm classic helyét már átvette a swarm mode. Viszont fontos tudni, hogy mi tartozik a swarm classic-ba és a swarm mode-ba, mivel ezek folyton kevered? fogalmak a fórumokon, fontos tisztán látni.

Contents

Bevezet?

https://docs.docker.com/swarm/reference/manage/

- A Swarm classic nem része az alap Docker programnak, ez egy független kiegészítése, amit külön kell telepíteni, ugyan úgy mint pl a docker-machine-t.
- A Swarm classic szabványos docker konténereket hoz létre a master és anode VM-eken. Külön konténert kap a swarm manager és külön konténert kap a swarm agent. Tehát a hagyományos Docker eszközkészletet használtja fel a swarm classic. Ezen konténerek segítségével telepíti aztán a swarm classic a megfelel? service-eket (konténereket) a node VM-eken. Mindenre a swarm nev? image-et fogjuk használni. Attól függ hogy mit fog csinálni a swarm konténer, amit az image-ból létrehozunk, hogy milyen CMI paramétereket kapott. (master, node vagy list)
- A swarm kezelésére nincsenek dedikált docker és compose parancsok. Be kell 'jelentkezni' swarm módban a manager VM-re, majd onnantól kezdve minden kiadott docker és compose parancs az egész cluster-re vonatkozik, nem a manager VM-en futó docker démonra. (Swarm mode-ban ez nem így van, ott saját parancs készlete van már a swarm-nak)
- A scheduling stratégiákat a swarm cluster létrehozásakor meg kell adni, a service-ek telepítése közben erre már nincs lehet?ség. Tehát a docker-compose.yml fájl deploy szekcióját nem veszi figyelembe a swarm classic, az kizárólag a swarm mode-nak szól.
- A swarm classic-kban nincs beépített load balancer szolgáltatás. Magunknak kell létrehozni. A load-balancer a végpontok listáját a discovery service szolgáltatótól kell hogy lekérdezze (pl. consul)

Kétféleképpen hozhatunk létre swarm classic cluster-t:

- Saját kez?leg hozzuk létre a docker-t futtató virtuális vagy fizikai gépeket. Ezeken manuálisan, docker parancsokkal hozzuk létre a master és node swarm konténereket.
- Használhatjuk a 'docker-machin' parancsot a '--swarm' kapcsolóval, hogy a megfelel? swarm cluster-hez is hozzáadja a létrehozott VM-et. Ez a funkció VM driver-t?l függetlenül m?ködik, ez a funkció be van építve a 'docker-machine' programba, azonban kizárólag swarm classic cluster-t tud kezelni. (Bizonyos driver-ek, talán az Amazon támogatják, hogy a létrehozott VM swarm mode cluster tagja legyen)

Bármelyik megoldást is választjuk a swarm cluster létrehozására, szükség van egy discovery szolgáltatóra, ahova a node-ok regisztrálják magukat. A discovery szolgáltatás egy KV (Key-Value) pár regiszter, ahova minden node beírja saját magát. A master a discovery szolgáltatót folyamatosan poll-ozza, hogy hozzájusson az aktuális member listához.

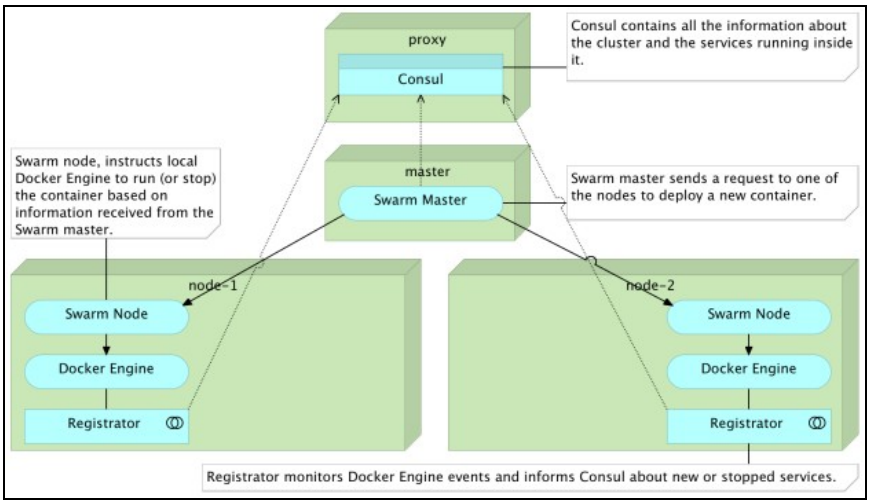
Note
Ez a swarm mode-ban (swarm 2.0) már nincs így. Discovery service nélkül is képes a master megtalálni a node-okat.

Összesen 6 féle discovery API-t támogat a swarm classic. Kett?re kérnék itt ki részletesebben, amik gyakran keverednek swarm mode leírásokkal.
Forrás1: https://docs.docker.com/swarm/discovery/#docker-hub-as-a-hosted-discovery-service
Forrás2: https://docs.docker.com/swarm/reference/manage/

- **token**: A DockerHub által nyújtott, ingyenes discovery szolgáltatás. Produkciós környezetbe nagyon nem ajánlott.

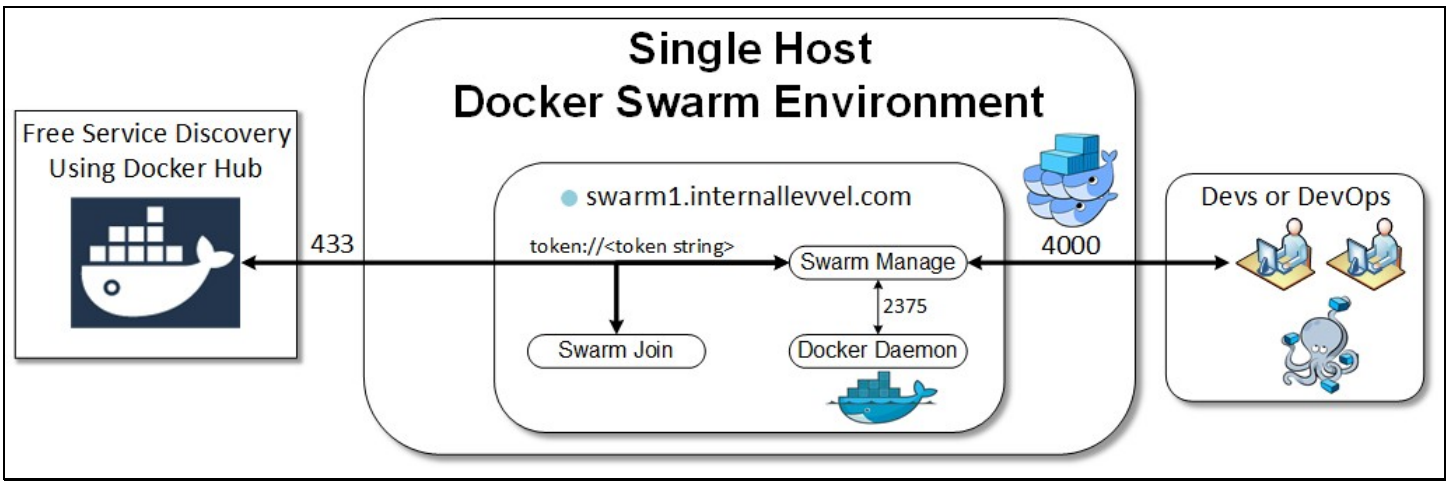
- **consul**: A Consul egy ingyenes, lokálisan futó discovery szolgáltatás, ezt írja magáról: "Consul is a distributed service mesh to connect, secure, and configure services across any runtime platform and public or private cloud Service registry, integrated health checks, and DNS and HTTP interfaces enable any service to discover and be discovered by other services"
- **etcd**: Ez is egy KV store. Úgy tudom hogy ez lett az új Swarm mode beépített service discovery megoldása, de a Kubernetes is ezt használja.
- fájl alapú: Egy fájlba kézzel beírjuk, hogy hol vannak a kliensek. Nyilván ez is csak játékra való, mint a DockerHub alapú megoldás

...



Swarm cluster DockerHub discovery service használatával

Ahogy már írtam fentebb, a swarm master-nek mindenképpen szüksége van egy discovery szolgáltatóra, hogy meg tudja találni a node-okat. A DocerkHub egy ingyenes, bárki számára elérhet? discovery szolgáltatást nyújt. Ez azonban csak tesztelési és fejlesztési célokra használható, produkciós környezetben nem. Cserébe nagyon egyszer? a használata, mert nem kell hozzá extra szoftver komponenseket telepíteni, össz-vissz annyira van szükségünk, hogy a swarm-ot futtató VM-eknek legyen publikus internet elérése, hogy elérjék a DockerHub-ot.



Swarm inicializál

Els?ként regisztrálni kell az új swarm cluster-ünket a Docker Hub-on, ami a discovery szolgáltatás szerepét tölti most be. A regisztráció végén a Docker Hub vissza fogja adni az új cluster azonosítóját, amivel a node-ok regisztrálni tudják magukat, és amivel a master bekérdezhet a discovery a Docker Hub-ba, hogy lekérdezze a cluster node-ok listáját. A regisztrációt a '**swarm'** nev? image futtatásával tehetjük meg, aminek az argumentuma a '**create'**

A swarm konténert **--rm** kapcsolóval futtatjuk, vagyis el is távolítjuk, amint lefutott.

```
# docker run --rm swarm create

Token based discovery is now deprecated and might be removed in the future.
It will be replaced by a default discovery backed by Docker Swarm Mode.
Other mechanisms such as consul and etcd will continue to work as expected.
7b1602e9fc114f1f47ad7ad4df41521c
```

A swarm konténer kiírja a konzolra az új cluster azonosítóját, majd kilép. Ahogy láthatjuk.

Note
Láthatjuk, hogy a docker figyelmeztet minket, hogy a token alapú swarm classic használata már elavult.

A regisztrációt a swarm konténer itt végzi el: https://registry-1.docker.io/v2/. Ha nem tudja elérni ezt a publikus host-ot, akkor a konténer hibát fog dobni.

Swarm ready VM-ek legyártása

A **docker-machine** parancs létre tudja úgy hozni a VM-eket, hogy azok már elve csatlakoznak egy docker swarm classic cluster-hez vagy master-ként vagy worker-ként. Sajnos a **docker-machine** a **swarm mode**-ot már nem támogatja semmilyen formában. Azonban bizonyos VM szolgáltatók driver-e lehet?vé teszi, hogy az új VM-em swarm mode tagjaként jöjjön eleve létre (azt hiszem a DigitalOchean ilyen).

A **docker-machine ls** parancs-nak van egy SWARM oszlopa. De ez is kizárólag a swarm classic-ra vonatkozik. Ha a VM tagja egy swarm mode cluster-nek, akkor sajnos a SWARM oszlopban ez nem fog megjelenni:

```
# docker-machine ls
NAME   ACTIVE   DRIVER   STATE     URL   SWARM   DOCKER    ERRORS
mg5    -        kvm      Stopped                 Unknown
```

A swarm classic ready gépeket a **--swarm** kapcsolóval kell létrehozni. A docker-machine-nek nagyon sok további paramétere van, ami azt szabályozza, hogy az új VM hogyan és milyen szerepben csatlakozzon a swarm classic cluster-hez.
Els?ként **--swarm-master** kapcsolóval adjunk hozzá egy master node-ot a cluster-hez. A discover szolgáltatást a **--swarm-discovery** kapcsolóval kell megadni. Mivel a **DocekrHub**-os token-t használtuk, itt is ezt a protokollt kell megadni.

```
docker-machine create -d kvm --kvm-network "docker-network" --kvm-disk-size "5000" \
--swarm --swarm-master --swarm-discovery="token://7b1602e9fc114f1f47ad7ad4df41521c" \
swarm-master
...
Configuring swarm...
Checking connection to Docker...
```

Tip
Természetesen a **swarm classic konténereket** manuálisan is létre lehet hozni, nem kell hozzá a **docker-machine --swarm** parancs. Ez csak egy segítség. Ilyenkor a docker-machine ugyan azt csinálja, amit mi is csinálni fogunk a Cluster létrehozás manuálisan fejezetben.

Ha a **--swarm-master** kapcsolót elhagyjuk, akkor normál worker-t adunk hozzá:

```
docker-machine create -d kvm --kvm-network "docker-network" --kvm-disk-size "5000" \
--swarm --swarm-discovery="token://7b1602e9fc114f1f47ad7ad4df41521c" node1
...
Configuring swarm...
Checking connection to Docker...
```

```
docker-machine create -d kvm --kvm-network "docker-network" --kvm-disk-size "5000" \
--swarm --swarm-discovery="token://7b1602e9fc114f1f47ad7ad4df41521c" node2
...
Configuring swarm...
Checking connection to Docker...
```

VM-ek ellen?rzése

```
# docker-machine ls
NAME           ACTIVE      DRIVER   STATE     URL                         SWARM                   DOCKER
node1          -           kvm      Running   tcp://192.168.42.236:2376   swarm-master            v18.05.0-ce
node2          -           kvm      Running   tcp://192.168.42.152:2376   swarm-master            v18.05.0-ce
swarm-master   * (swarm)   kvm      Running   tcp://192.168.42.54:2376    swarm-master (master)   v18.05.0-ce
```

Note
A **docker-machine ls** parancsában a SWARM oszlop kizárólag a **--swarm** kapcsolóval létrehozott swarm classic cluster-re vonatkozik. A Swarm mode-al létrehozott cluster esetében ez az oszlop nem lesz kitöltve.

Láthatjuk a host gép route táblájában, hogy két KVM-re vonatkozó route bejegyzés van, ezek a **virbr0** és a **virbrDocker**.

- A **virbr0** a **KVM** docker-machine driver által létrehozott hálózat, amivel a VM-ek egymás között tudnak csak kommunikálni. Ez a **default** nev? KVM hálózatnak felel meg a 'virsh net-list' listába. Ezt a hálózatot automatikusan hozza létre a driver, nincs rá hatásunk.
- A **virbrDocker** pedig az a hálózat amit mi definiáltunk korábban **docker-network** néven. Ez a hálózat kilát a publikus internetre. Ezt adtuk meg a VM-ek létrehozásakor a **docker-machine**-nek.

```
# route
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
...
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
192.168.123.0   0.0.0.0         255.255.255.0   U     0      0        0 virbrDocker
```

Nézzük meg, hogy ezek melyik hálózatok a KVM szerint

```
# virsh net-list
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              active     yes           yes
 docker-network       active     yes           yes

# virsh net-info default
Name:           default
Bridge:         virbr0
```

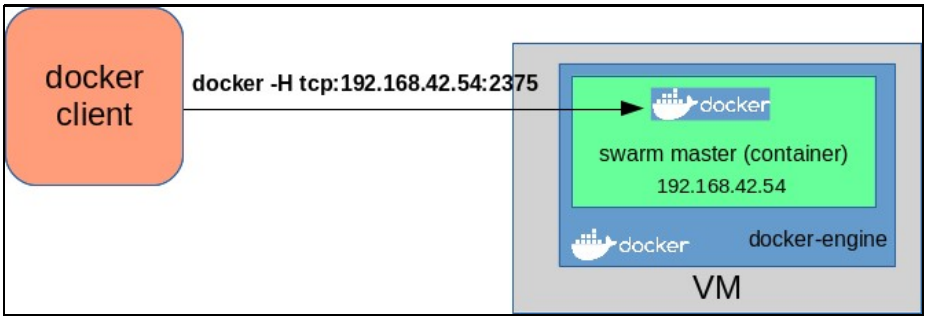
```
# virsh net-info docker-network
Name:           docker-network
Bridge:         virbrDocker
```

Swarm cluster ellen?rzése

A swarm classic cluster tagjait ki tudjuk listázni a **swarm image** segítségével, ha megkérjük, hogy kérje le a discovery agent-t?l (ami most a **DockerHub**) a node-ok listáját. Ezt bárhol lefuttathatjuk, nem csak a VM-eken. A swarm konténer le fogja kérdezni a Docker Hub-tol a node-ok listáját. Azért van három node, mert a master VM-en is fut egy swarm agent.

```
# docker run --rm swarm list token://7b1602e9fc114f1f47ad7ad4df41521c
192.168.42.152:2376
192.168.42.236:2376
192.168.42.54:2376
```

Ahhoz hogy utasításokat tudjunk kiadni a swarm cluster-en, a docker kliensünkkel rá kell csatlakozni a swarm manager konténerben futó docker démonra. Tehát arra a docker-démonra kell csatlakozni ami a konténert natívan futtatja a VM-en, hanem arra swarm docker démonra a swarm konténer belsejében:



Ezt kétféle képen tehetjük meg.

1. A docker kliensünknek a **-H tcp:ip:port** paraméterrel megadjuk a swarm konténer címét és portját
2. Használjuk a **docker-machine env** parancs **--swarm** kapcsolóját, ami ezt elvégzi helyettünk. Már korábban is láthattuk, hogy a **docker-machine evn <gépnév>** paranccsal transzparens módon tudtunk csatlakozni a távoli démonra a lokális klienssel. Ekkor minden kiadott docker parancs a távoli démonon futott le. Ha a **--swarm** kapcsolót is használjuk, és a távoli docker démon futtat swarm manager konténert, akkor közvetlen a swarm konténer belsejébe futó docker démon-ra fog csatlakozni szintén transzparens módon a lokális docker kliens.

Ha a **--swarm** kapcsolót is használjuk, akkor innent?l kezdve minden kiadott docker parancs már a swarm classic cluster-en fog lefutni:

```
# eval "$(docker-machine env --swarm swarm-master)"
```

Innent?l kezdve minden kiadott docker parancs az egész cluster-re fog vonatkozni.

A **docker info** láthatjuk, hogy 4 konténert mutat, holott a master-t futtató VM-en még semmilyen konténert nem telepítettünk. Ezek mind swarm konténerek, három worker és egy master.

```
# docker info
Containers: 4
 Running: 4
 Paused: 0
 Stopped: 1
Images: 4
Server Version: swarm/1.2.9
Role: primary
Strategy: spread
Filters: health, port, containerslots, dependency, affinity, constraint, whitelist
Nodes: 3
 node1: 192.168.42.236:2376
  ? ID: 7AKJ:NJMX:6OW3:OWNJ:Q2AV:SXPH:SM4S:GY3B:TZMQ:QJIV:4UEB:6O65|192.168.42.236:2376
  ? Status: Healthy
  ? Containers: 1 (1 Running, 0 Paused, 0 Stopped)
  ? Reserved CPUs: 0 / 1
  ? Reserved Memory: 0 B / 1.021 GiB
  ? Labels: kernelversion=4.9.93-boot2docker, operatingsystem=Boot2Docker ..
  ? UpdatedAt: 2018-07-08T08:15:56Z
  ? ServerVersion: 18.05.0-ce
 node2: 192.168.42.152:2376
  ? ID: 2K5X:CTUY:VOBF:AF5K:GBWU:3ICY:YJWJ:ALAY:6MDP:HAP4:JYHB:ZGRJ|192.168.42.152:2376
  ? Status: Healthy
  ? Containers: 1 (1 Running, 0 Paused, 0 Stopped)
  ? Reserved CPUs: 0 / 1
  ? Reserved Memory: 0 B / 1.021 GiB
  ? Labels: kernelversion=4.9.93-boot2docker, operatingsystem=Boot2Docker ..
  ? UpdatedAt: 2018-07-08T08:15:56Z
  ? ServerVersion: 18.05.0-ce
 swarm-master: 192.168.42.54:2376
  ? ID: LHJT:XZ2H:YAEH:COC7:DQXM:PRCV:O4EA:3BEE:OZXC:R5GV:2DIC:GW75|192.168.42.54:2376
  ? Status: Healthy
  ? Containers: 2 (2 Running, 0 Paused, 0 Stopped)
  ? Reserved CPUs: 0 / 1
  ? Reserved Memory: 0 B / 1.021 GiB
  ? Labels: kernelversion=4.9.93-boot2docker, operatingsystem=Boot2Dock..
  ? UpdatedAt: 2018-07-08T08:15:35Z
  ? ServerVersion: 18.05.0-ce
```

A **ps** parancs is az egész cluster-re vonatkozik:

```
# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  PORTS                                    NAMES
8d8167f3b9bc   swarm:latest   "/swarm join --adv..."   2375/tcp                                 node2/swarm-agent
4d5b7b11a8dd   swarm:latest   "/swarm join --adv..."   2375/tcp                                 node1/swarm-agent
ea5bf0ae71e3   swarm:latest   "/swarm join --adv..."   2375/tcp                                 swarm-master/swarm-agent
7b135d1f0e64   swarm:latest   "/swarm manage --t..."   2375/tcp, 192.168.42.54:3376->3376/tcp   swarm-master/swarm-agent-master
```

Látható, hogy egy-egy swarm agent konténer fut a node nev? gépeken és egy agent és a master fut a swarm-master nev? gépen. Az agent konténerek számára a 2375 (alapértelmezett) port van kinyitva, itt beszélgetnek a master-rel. A master meg a 3376-os porton érhet? el.

Nézzünk megy egy agent konténert:

```
# docker inspect 8d8167f3b9bc
    ...
            "IP": "192.168.42.152",
            "Addr": "192.168.42.152:2376",
            "Name": "node2",
        },
        "Name": "/swarm-agent",
        "Config": {

            "ExposedPorts": {
                "2375/tcp": {}
            },
            "Env": [
                "SWARM_HOST=:2375"
            ],
            "Cmd": [
                "join",
                "--advertise",
                "192.168.42.152:2376",
                "token://7b1602e9fc114f1f47ad7ad4df41521c"
            ],
            "Volumes": {
                "/.swarm": {}
```

Láthatjuk hogy az agent-nek a 2375-ös portja van nyitva, a swarm volume van felcsatolva, és hogy a konténer az alábbi argumentumokat kapta meg a docker run parancsban.

join --advertise "192.168.42.152:2376" token://7b1602e9fc114f1f47ad7ad4df41521c

Swarm cluster 'Consul' használatával

A consule egyike azon 5 API-nak, amit használhatunk mint discovery szolgáltatás a swarm cluster-ünkben. A consule honlap ezt írja magukról:
Consul is a distributed service mesh to connect, secure, and configure services across any runtime platform and public or private cloud Service registry, integrated health checks, and DNS and HTTP interfaces enable any service to discover and be discovered by other services
https://www.consul.io

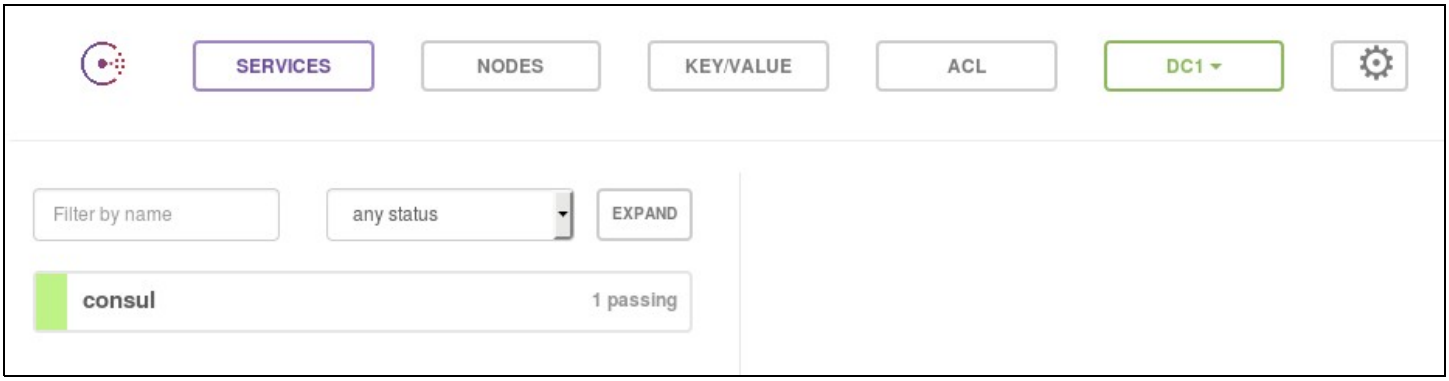
Tip
A Docker Hub-os discover szolgáltatással ellentétben ez már használható produkciós környezetben is.

Consule telepítése

A conluse szintén telepíthet? docker konténerként, majd ide fogják magukat regisztrálni a node-ok, és innen fogja lekérdezni a node listát a master. A conlue -nak nem fogunk most külön VM-et létrehozni, a host docker démonján fog futni:

```
# docker run -d -p 8400:8400 -p 8500:8500 -p 8600:53/udp -h consul progrium/consul -server -bootstrap -ui-dir /ui
```

Az utolsó paraméter (/ui) hatására lesz web-es admin konzolunk is. Ha elindult a konténer, akkor a webes konzolt itt érhetjük el: http://localhost:8500/ui/



Hozzuk újra létre a master-t futtat VM-et kapásból swarm classic módban. (**--swarm** és **--swarm-master**). A **--swarm-discovery** értékének most a **consul** címét kell megadni. Innen fogja a master lekérdezni a node-k listáját és a telepített service-k listáját.

```
docker-machine create -d kvm --kvm-network "docker-network" --kvm-disk-size "5000" \
--swarm --swarm-master \
--swarm-discovery="consul://192.168.0.105:8500" \
consule-master
```

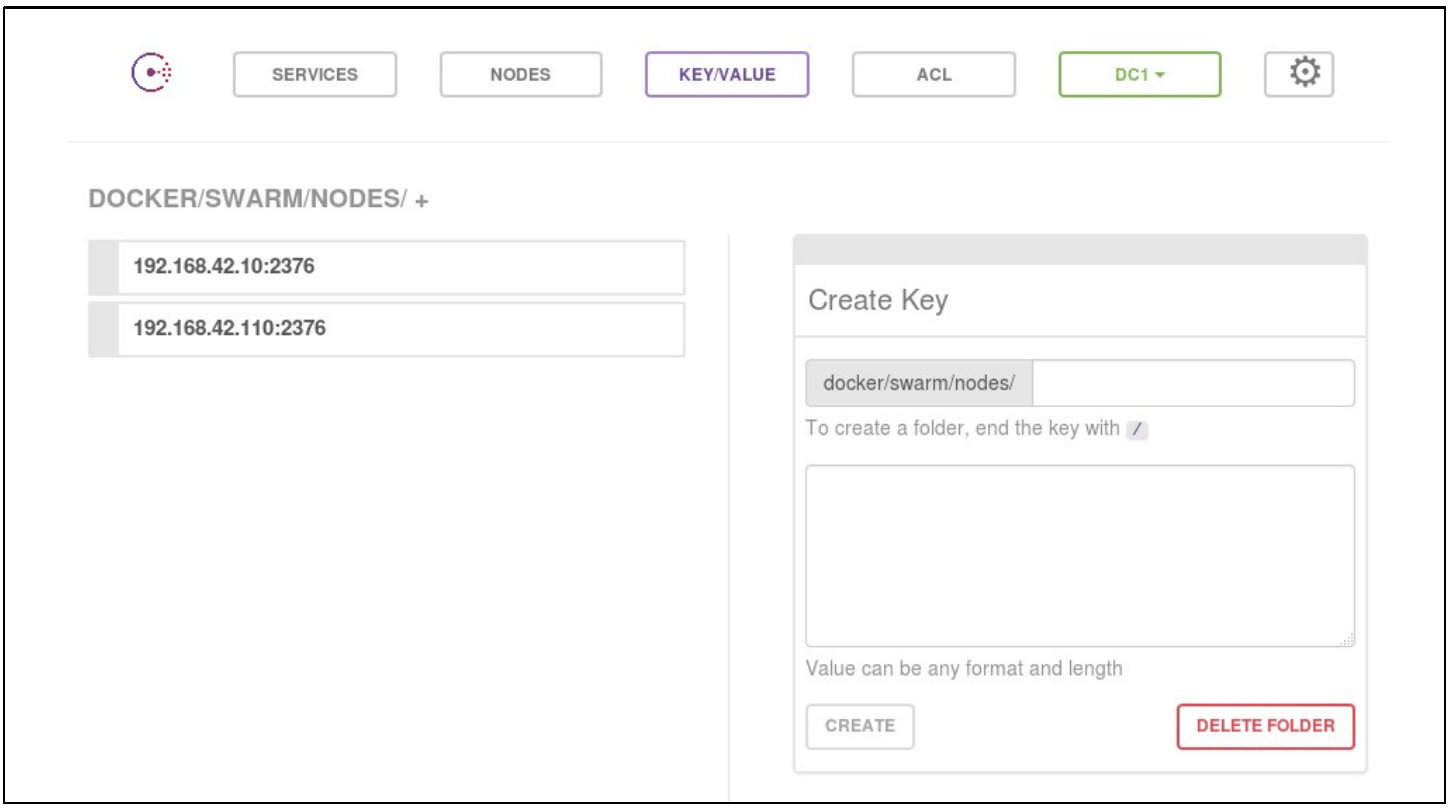
Hozzunk létre node-okat is a **--swarm-master** kapcsoló elhagyásával. A discovery szolgáltatás címe változatlan.

```
docker-machine create -d kvm --kvm-network "docker-network" --kvm-disk-size "5000" \
--swarm \
--swarm-discovery="consul://192.168.0.105:8500" \
consule-node1
```

A node-ok listázását bármelyik gépen futtathatjuk, ami eléri a host-ot. Ez a VM-en futó docker-engine-ben fog létrehozni egy új swarm konténert a lekérdezés idejére. Paraméterként megkapja a consul címét, ahonnan elkéri a node-ok listáját. Látható, hogy mind két node-ot megtalálta. Egy van a master VM-en, amit automatikusan hozott létre a **docker-machine** parancs, és egy van a node számára létrehozott VM-en.

```
# docker run --rm swarm list consul://192.168.0.105:8500
time="2018-07-08T21:55:56Z" level=info msg="Initializing discovery without TLS"
192.168.42.10:2376
192.168.42.110:2376
```

A Consul webes konzolján is megnézhetjük a node-ok listáját a /nodes alatt: http://localhost:8500/ui/#/dc1/kv/docker/swarm/nodes/



Most a lokális docker kliensünkkel csatlakozzunk rá a swarm master-re ahogy azt már fentebb tettük:

```
# eval "$(docker-machine env --swarm consule-master)"
```

És nézzük meg hány konténer jött létre összesen a cluster-ben:

```
# docker info
Containers: 3
 Running: 3
 Paused: 0
 Stopped: 0
Images: 2
Server Version: swarm/1.2.9
Role: primary
Strategy: spread
Filters: health, port, containerslots, dependency, affinity, constraint, whitelist
Nodes: 2
 consule-master: 192.168.42.10:2376
  ? ID: KD2G:QYRW:Z63A:O2GV:4CU4:2GWK:IR5D:NUWX:EAIH:H4ON:ODCH:6QUB|192.168.42.10:2376
  ? Status: Healthy
  ? Containers: 2 (2 Running, 0 Paused, 0 Stopped)
  ? Reserved CPUs: 0 / 1
  ? Reserved Memory: 0 B / 1.021 GiB
  ? Labels: kernelversion=4.9.93-boot2docker,
  ? UpdatedAt: 2018-07-08T22:08:15Z
  ? ServerVersion: 18.05.0-ce
 consule-node1: 192.168.42.110:2376
  ? ID: V2ZU:666C:DI7V:4RC7:KWGI:Y4LW:AYL5:IXQO:2QAE:4HOP:EHHC:2D7Y|192.168.42.110:2376
  ? Status: Healthy
  ? Containers: 1 (1 Running, 0 Paused, 0 Stopped)
  ? Reserved CPUs: 0 / 1
  ? Reserved Memory: 0 B / 1.021 GiB
  ? Labels: kernelversion=4.9.93-boot2docker
```

Cluster létrehozása manuálisan

A **swarm classic cluster**-t nem csak a **docker-machine** paranccsal lehet létrehozni, bár kétségtelen hogy úgy a legegyszer?bb. Létrehozhatjuk swarm cluster nélkül is a VM-eket, majd oda SSH-val belépve, manuálisan is felépíthetjük a cluster-t (vagyis elindíthatjuk a swarm konténereket). Nézzünk erre egy token-es (DockerHub) példát.

Cluster regisztrálása

Most is DockerHub-ot fogunk használni discovery szolgáltatóként, az egyszer?ség végett. Ugyan úgy regisztráljuk a cluster-t a rögtön megsz?n? swarm konténerrel ahogy azt már tettük.

```
# docker run --rm swarm create

Token based discovery is now deprecated and might be removed in the future.
It will be replaced by a default discovery backed by Docker Swarm Mode.
Other mechanisms such as consul and etcd will continue to work as expected.
4d245124e7feba5224a117bdd83c9acc
```

https://docs.docker.com/swarm/reference/manage/
http://harrylee.me/2016/08/23/Docker-Docker-Swarm-with-Docker-Machine-Quick-Setup-Guide/

Master létrehozása

https://www.w3cschool.cn/doc_docker_1_10/docker_1_10-swarm-install-manual-index.html

Hozzuk létre a master VM-jét swarm nélkül. A docker-machine-al készítünk egy docker ready gépet.

```
# docker-machine create -d kvm --kvm-network "docker-network" manager
Running pre-create checks...
...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env manager
```

Lépjünk be SSH-val az új VM-re.

```
# docker-machine ssh manager
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
              \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 18.05.0-ce, build HEAD : b5d6989 - Thu May 10 16:35:28 UTC 2018
Docker version 18.05.0-ce, build f150324
docker@manager:~$
```

Mikor a manager-t létrehozzuk, akkor ugyan azt a **swarm** image-t futtatjuk, amivel regisztráltuk a cluster-t a Docker Hub-on, csak most nem a create hanem a manager paraméterrel, innen fogja tudni a swarm, hogy manager konténert kell létrehoznia.
A **docker-machine --swarm** üzemmódja pont ezt csinálta meg, azzal a különbséggel, hogy a master VM-en egy node-ot is mindig létrehoz a master mellé.

```
docker@manager:~$ docker run -d -p 4000:4000 swarm manage -H :4000 --advertise 192.168.42.83:4000 token://4d245124e7feba5224a117bdd83c9acc
Unable to find image 'swarm:latest' locally
latest: Pulling from library/swarm
d85c18077b82: Pull complete
1e6bb16f8cb1: Pull complete
85bac13497d7: Pull complete
Digest: sha256:406022f04a3d0c5ce4dbdb60422f24052c20ab7e6d41ebe5723aa649c3833975
Status: Downloaded newer image for swarm:latest
19e2e236a8bc1aab1d7aab5bad17fb66af7184380457f7ee4de3a4e046c599ae
```

Node-ok létrehozása

Hozzunk létre egy swarm nélküli VM-et a node-nak is:

```
# docker-machine create -d kvm --kvm-network "docker-network" node1
...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env node1
```

Lépjünk be SSH-val az új VM-re:

```
# docker-machine ip node1
192.168.42.164



# docker-machine ssh node1
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
```

```
                \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 18.05.0-ce, build HEAD : b5d6989 - Thu May 10 16:35:28 UTC 2018
Docker version 18.05.0-ce, build f150324
```

Itt is ugyan úgy a swarm image-ból fogunk csinálni egy konténert, de a manage paraméter helyett a **join** paramétert adjuk mega a konténernek, innen fogja tudni, hogy node-ot kell létrehozni.

```
docker@node1:~$ docker run -d \
>   --restart=always swarm join \
>   --addr=192.168.42.164:2376 \
>   token://4d245124e7feba5224a117bdd83c9acc
Unable to find image 'swarm:latest' locally
latest: Pulling from library/swarm
d85c18077b82: Pull complete
1e6bb16f8cb1: Pull complete
85bac13497d7: Pull complete
Digest: sha256:406022f04a3d0c5ce4dbdb60422f24052c20ab7e6d41ebe5723aa649c3833975
Status: Downloaded newer image for swarm:latest
34e10a5463faa9b42556914822e5a20281cebdbcd7b0539a90cc6fea0f876e2c
```

Ha ugyan ezen a VM-en nyomunk egy **docker ps**-t, akkor láthatjuk, hogy létrejött az egy szem swarm konténer, ami **node** üzemmódban fut.

```
docker@node1:~$ docker ps
CONTAINER ID  IMAGE  COMMAND                CREATED          STATUS         PORTS     NAMES
34e10a5463fa  swarm  "/swarm join --addr=?" 30 seconds ago   Up 29 seconds  2375/tcp  kind_goldberg
```

Mi jött létre

Egy rögtön megsz?n? swarm konténert futtassunk a list paraméterrel, hogy lekérje a DockerHub-tól a node-ok listáját.

```
# docker run --rm swarm list token://4d245124e7feba5224a117bdd83c9acc
192.168.42.231:2376
192.168.42.164:2376
```

Ha a master VM-en az ott futó docker klienssel rácsatlakozunk a swarm master konténerre, akkor listázni tudjuk a swarm cluster részleteit.

```
$ docker -H :4000 info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: swarm/1.2.9
Role: primary
Strategy: spread
Filters: health, port, containerslots, dependency, affinity, constraint, whitelist
Nodes: 2
  (unknown): 192.168.42.231:2376
   ? ID:
   ? Status: Pending
   ? Containers: 0
   ? Reserved CPUs: 0 / 0
   ? Reserved Memory: 0 B / 0 B
   ? Labels:
   ? UpdatedAt: 2018-07-06T23:38:41Z
   ? ServerVersion:
  (unknown): 192.168.42.164:2376
   ? ID:
   ? Status: Pending
   ? Containers: 0
   ? Reserved CPUs: 0 / 0
   ? Reserved Memory: 0 B / 0 B
   ? Labels:
   ? UpdatedAt: 2018-07-06T23:38:41Z
   ? ServerVersion:
```

Scheduling

Note
docker classic-ban nem tudjuk még a compose YAML fájlban megadni a kiosztási stratégiát (deploy szekció)!. A példányszámot a compose scale parancsával kell megadni, a kiosztási stratégiát pedig a swarm master létrehozásakor kell definiálni. Ez az egész cluster-re vonatkozni fog, és nem lehet megváltoztatni.

Szolgáltatás konténer száma

Írjunk egy nagyon egyszer? service-t. A compose fájlnak adjuk a flock.yml nevet az alábbi tartalommal

```
bird:
  image: dockerinaction/ch12_painted
  command: bird
  restart: always
```

Tehát egy darab **bird** nev? service-t definiálunk, úgy hogy mindig indítsa újra a docker, ha leáll.

Majd indítsuk el 10 példányban a bird nev? szolgáltatásunkat.

```
# docker-compose -f flock.yml scale bird=10
```

Most nézzük meg mi jött létre:

```
# docker ps
CONTAINER ID        IMAGE         COMMAND             NAMES
5606975758f4        painted   "/magic.sh bird"   node2/bird_bird_4
816db74ca4d5        painted   "/magic.sh bird"   swarm-master/bird_bird_7
9bec876d658c        painted   "/magic.sh bird"   swarm-master/bird_bird_2
d3481b35d14d        painted   "/magic.sh bird"   node1/bird_bird_9
1e5994ed2820        painted   "/magic.sh bird"   node2/bird_bird_5
44729be69074        painted   "/magic.sh bird"   node1/bird_bird_6
f2101718f2ab        painted   "/magic.sh bird"   swarm-master/bird_bird_8
9c42193745e2        painted   "/magic.sh bird"   node2/bird_bird_1
6c1e97342499        painted   "/magic.sh bird"   node1/bird_bird_10
aa97325e8cb6        painted   "/magic.sh bird"   node1/bird_bird_3
```

Láthatjuk, hogy 10 konténer jött létre, 3-3 a node2-ön és a swarm-master-en, és 4 a node1-en. A konténer neve mindig a node nevével kezd?dik, majd jön az image név majd a sorszáma a replikának. Ahogy láthatjuk, a swarm egyenl?en akarta elosztani a node-ok között a konténereket. Ez a "spread" elosztási stratégia, ez az alapértelmezett. Mivel a swarm cluster létrehozásakor nem adtunk meg stratégiát, ezért a "spread" scheduling stratégiát használja a cluster.

Kiosztási stratégiák

A kiosztási stratégiát a master létrehozásakor kell megadni a **--swarm-strategy** kapcsolóval. Három stratégia közül választhatunk:

spread

Egyenl?en akarja elosztani, mindig a leg kevéssé leterhelt node-ra. Ha két node azonosan van leterhelve, akkor arra rakja, amin kevesebb konténer fut. Akkor hatásos, ha a konténereken limitálva van az er?forrás foglalás, és ha definiált er?forrás limitek nem térnek el nagyban egymástól.

Ez az alapértelmezett, ha nem adunk meg semmit, akkor mindig spread lesz.

```
# docker-machine create -d kvm --kvm-network "docker-network" --kvm-disk-size "5000" \
--swarm --swarm-master \
--swarm-discovery="token://7b1602e9fc114f1f47ad7ad4df41521c" \
--swarm-strategy=spread
swarm-master



# docker info
....
Strategy: spread
```

BinPack

Mindig egy node-ot megpróbál maximálisan kihasználni, miel?tt egy új node-ra tenni az új konténert. Csak akkor tud jól m?ködni, vagy bárhogy m?ködni, ha minden konténert er?forrás megkötésekkel hozunk létre. Akkor hasznos, ha nagyon nagy a különböz? er?forrást igényl? konténerek variációja, de ugyanakkor meg van adva az er?forrást megkötés minden konténerhez. Ezzel ellentétben a Spread akkor volt jó, ha hasonló volt az er?forrás igénye az összes konténernek.

Random

Filterek

A filterek hasonlón m?ködnek a swarm cluster-ben mint az itables filterek. Mikor egy újonnan létrehozandó konténernek keresi a swarm a helyét, hogy melyik node-ra rakja, akkor els?ként a szóba jöhet? node-okat végig futtatja a filter láncon. Egy node akkor marad bent a lehetséges node-ok listájában, ha a láncon sehol nem akadt fent.

A filter paramétereket a compose vagy a run -ba kell **-e <filter név>==<filter érték>** formában. pl:
docker run -d -e affinity:image==nginx nginx

Öt gyári filter lánc van:
health, port, containerslots, dependency, affinity, constraint, whitelist

affinity

Egy megkötés már a node-ra telepített konténerekre. Azt mondja meg, hogy csak olyan node-ko jöhetnek szóba, ahol már van ilyen konténer, amit megadtunk az effinity paramétereként.

Azon node-ok, ahova az nginx már fel van rakva:

```
# docker run -d -e affinity:image==nginx -p 80:80 nginx
```

Azon node-ok ahova az nginx még nincs felrakva:

```
# docker run -d -e affinity:image!=nginx -p 8080:8080 haproxy
```

health

constraint

Megkötés bármilyen VM tulajdonságra. Vagy a beépített tulajdonság változókat használjuk, vagy sajátokat adhatunk meg a docker-machine create --engine- label használatával, amire kés?bb hivatkozhatunk.

Van 5 beépített tulajdonság:

- node ?The name or ID of the node in the cluster
- storagedriver ?The name of the storage driver used by the node
- executiondriver ?The name of the execution driver used by the node
- kernelversion ?The version of the Linux kernel powering the node
- operatingsystem ?The operating system name on the node

Ezeket a docker info parancs minden node-ra megmutatja:

```
# docker info
node1: 192.168.42.236:2376
  ? ID: 2K5X:CTUY:VOBF:AF5K:GBWU:3ICY:YJWJ:ALAY:6MDP:HAP4:JYHB:ZGRJ|192.168.42.152:2376
  ? Labels: kernelversion=4.9.93-boot2docker, operatingsystem=Boot2Docker 18.05.0-ce
```

A custom címkéket a **--engine-label** kapcsolóval kell megadni a VM létrehozásakor:

```
# docker-machine create -d kvm --kvm-network "docker-network" \
--swarm --swarm-discovery token://7b1602e9fc114f1f47ad7ad4df41521c \
--engine-label size=small \
little-machine
```

Majd így hivatkozhatunk rá a **run** parancsban:

```
# docker run -d -e constraint:size==small postgres
```

Ez azt jelenti, hogy csak olyan node-ok jöhetnek szóba az új konténer futtatására, amik **size=small** címkével lettek létrehozva.

port

dependency