Docker networking, Swarm a Kubernetes

Obsahom tohto dokumentu je ukážka práce so základnými konceptmi práce s počítačovou sieťou v rámci docker prostredia. Ďalšou časťou dokumentu je ukážka optimalizačných a orchestračných nástrojov pre správu docker kontajnerov a to Swarm a Kubernetes.

Docker Networking

Docker predstavuje jeden z virtualizačných prístupov umožňujúcich jednoduché vytváranie a nasadzovanie aplikácií. Je to jednoduchá forma virtualizácie, kde nad kernelom Host zariadenia sa spúšťa izolované prostredie (kontajner). Pri vytváraní sietí robí docker engine automatickú úpravu iptables na host zariadení.

Docker poskytuje 6 hlavných network typov:

- **Bridge**: Default type. Predstavuje izolované sieťové prostredie umožňujúce komunikáciu medzi kontajnermi. Viacero kontajnerov v tej istej Bridge sieti dokáže vzájomne komunikovať. Vzájomná komunikácia medzi kontajnermi v rôznych bridge sieťach možná nie je. Poskytuje automatické DNS rozoznávanie pre používateľom špecifikované siete. Kontajner nepriradený do žiadnej siete je automaticky v default bridge sieti. Pre sprístupnenie služby kontajnera z bridge siete je potrebné daný port vypublikovať.
- **Host**: Sieť spadajúca do štandardnej host siete = bez izolácie. Dostupné len na OS Linux. Docker kontajner spadá pod IP adresu host zariadenia.
- **Overlay**: Spája viaceré Docker deamons – umožňuje Swarm službu = distribuovaná komunikácia medzi docker prostrediami bežiacimi na rôznych serveroch.
- **IPVlan**: Poskytuje plnú kontrolu nad IPv4 adresáciou. Docker kontajner sa javí ako nové zariadenie v tej istej sieti ako host. Môže byť tiež pripojený ku konkrétnym VLAN v prípade že host zariadenie podporuje trunk a tagovanie.
- **MACVlan**: Podobné ako IPVlan, avšak navyše umožňuje priradiť kontajneru vlastnú MAC adresu = javí sa ako zariadenie priamo pripojené do siete.
- **None**: Kompletná izolácia od Host a ďalších kontajnerov.

V rámci tohto cvičenia budeme pracovať hlavne s Bridge a Host typom sietí:

1. Nainštalujte docker engine do Vášho zariadenia:

```
sudo apt install docker.io
```

2. Zobrazte štandardne existujúce docker siete:

```
sudo docker network ls
```

```
┌──(kali㉿kali)-[~]
└─$ sudo docker network ls
[sudo] password for kali:
NETWORK ID     NAME      DRIVER    SCOPE
888ea32984e6   bridge    bridge    local
d883bac92501   host      host      local
227f79464827   none      null      local
```

3. Vytvorte novú sieť v docker prostredí a nazvite ju apsDockerNetwork:

```
sudo docker network create --driver=bridge --
subnet=192.168.10.0/24 --ip-range=192.168.10.0/28 --
gateway=192.168.10.254 apsDockerNetwork
```

4. Overte úspešnosť vytvorenia docker siete a zobrazte detailné informácie o nej:

```
sudo docker network ls
```

```
┌──(kali㉿kali)-[~]
└─$ sudo docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
093322cfe3d2   apsDockerNetwork    bridge    local
888ea32984e6   bridge              bridge    local
d883bac92501   host                host      local
227f79464827   none                null      local
```

```
sudo docker network inspect apsDockerNetwork
```

```
┌──(kali㉿kali)-[~]
└─$ sudo docker network inspect apsDockerNetwork
[
    {
        "Name": "apsDockerNetwork",
        "Id": "093322cfe3d2c0239cbbbcc436c3bf8e67225e6a88257b4039e76de4dab52f39",
        "Created": "2023-10-10T15:06:23.294990973-04:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.10.0/24",
                    "IPRange": "192.168.10.0/25",
                    "Gateway": "192.168.10.254"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

5. Spusťte dvojicu kontajnerov. Prvý nech je kontajner Apline (čistý linux) a druhý Nginx (spustená web služba na porte 80).

```
sudo docker run -dit --name alpineContainer --network
apsDockerNetwork alpine

sudo docker run -dit --name nginxContainer --network
apsDockerNetwork nginx
```

6. Overte, že došlo k úspešnému spusteniu kontajnerov:

```
sudo docker container ls
```

```
┌──(kali㉿kali)-[~]
└─$ sudo docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
1c8da88368c1   nginx     "/docker-entrypoint.…"   9 seconds ago    Up 7 seconds    80/tcp    nginxContainer
717a5c1fbdde   alpine    "/bin/sh"                26 seconds ago   Up 23 seconds             alpineContainer
```

7. Skontrolujte konfiguráciu docker kontajnerov v kontexte sieťových nastavení:

```
sudo docker network inspect apsDockerNetwork
```

```
"Containers": {
    "1c8da88368c1df4106eea1d4fd1ec3ccef39a140b1a6deaa8087dbd4115fb164": {
        "Name": "nginxContainer",
        "EndpointID": "6b273bcd64cb48a0c98c96c883fe5fe281ae1f95ea25aafafd841be53d995bc0",
        "MacAddress": "02:42:c0:a8:0a:02",
        "IPv4Address": "192.168.10.2/24"
        "IPv6Address": ""
    },
    "717a5c1fbddead9a8d6122785bf2e5c1ed4e2eed2d1e4a702162e2a13710e438": {
        "Name": "alpineContainer",
        "EndpointID": "252efeab8641538023333f3cbe548092d557c4ffcad79ae3621f504ca4923c16",
        "MacAddress": "02:42:c0:a8:0a:01",
        "IPv4Address": "192.168.10.1/24",
        "IPv6Address": ""
    }
```

8. Skontrolujte sieťové nastavenie na lokálnom zariadení:

```
Ifconfig
```

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.10.12  netmask 255.255.255.0  broadcast 10.10.10.255
```

9. Otestujte dostupnosť webovej služby z lokálneho Host zariadenia (služba nebude dostupná):

```
curl 192.168.10.2:80
```

10. Pripojte sa na kontajner alpineContanier a z neho otestujte dostupnosť webovej služby:

```
sudo docker exec -it alpineContainer sh
apk add curl

ifconfig
ping nginxContainer
ping 192.168.10.2
curl 192.168.10.2:80
```

```
PING nginxContainer (192.168.10.2): 56 data bytes
64 bytes from 192.168.10.2: seq=0 ttl=64 time=1.607 ms
64 bytes from 192.168.10.2: seq=1 ttl=64 time=0.198 ms
^C
── nginxContainer ping statistics ──
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.198/0.902/1.607 ms
/ #
/ # curl 192.168.10.2:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

11. Vytvorte ďalší alpine kontajner, no tentokrát bez špecifikácie bridge siete:

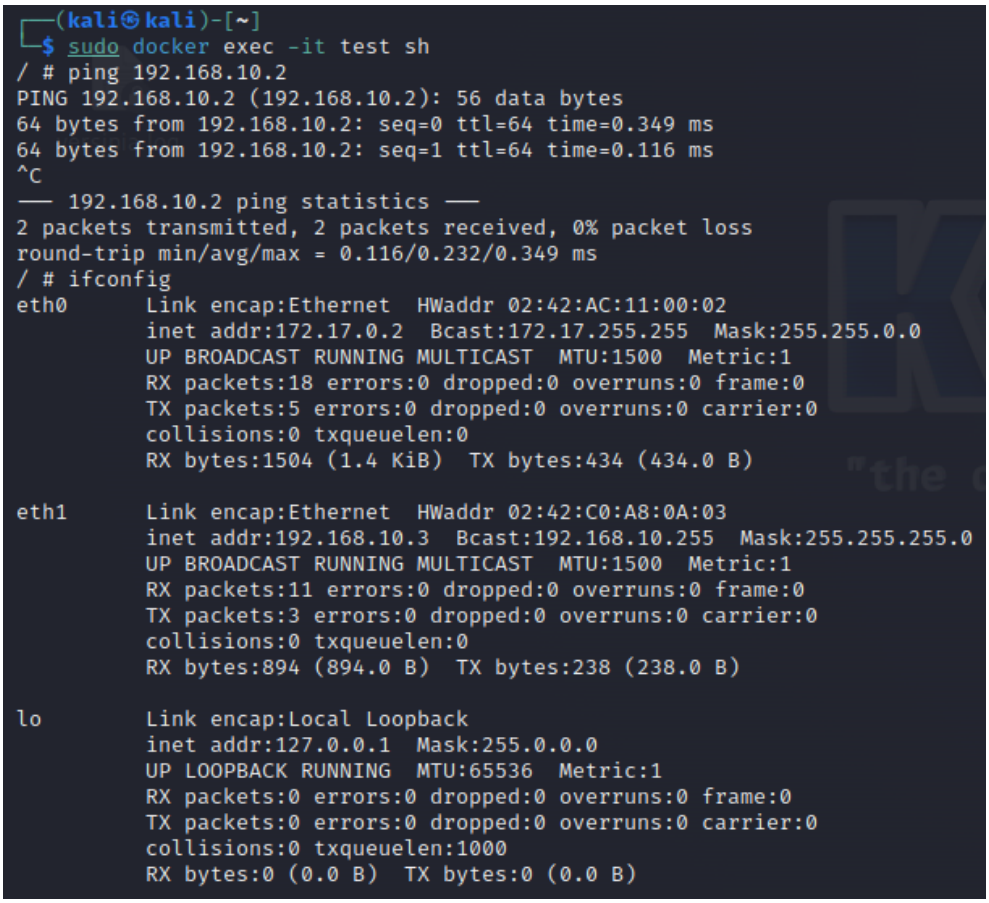
```
sudo docker run -dit --name test alpine
docker exec -it test sh
ifconfig
```

Pozn.: Komunikácia medzi kontajnermi nebude možná vzhľadom k tomu, že sa nachádzajú v rôznych sieťach, čo je možné overiť príkazom ifconfig.

```
┌──(kali㉿kali)-[~]
└─$ sudo docker exec -it test sh
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1252 (1.2 KiB)  TX bytes:0 (0.0 B)
```

12. Pripojte kontajner test do siete apsDockerNetwork

```
sudo docker network connect apsDockerNetwork test
ping 192.168.10.2
ifconfig
```



13. Odpojte pripojené rozhranie kontajnera Test:

```
sudo docker network disconnect apsDockerNetwork test
```

14. Skúste z Host zariadenia pripojenie na web službu nginx kontajnera pod adresou localhost:80 (komunikácia by ísť nemala – dôvod je ten, že služba nie je vypublikovaná mimo docker siete). Pozor: ak použijete konkrétnu adresu nginx kontajnera, tak web bude dostupný (dôvod je ten, že Host má jeden zo svojich rozhraní pripojený do docker siete).

15. Zastavte nginx docker kontajner a spusťte ho tak, aby bol súčasťou Host siete = automaticky vypublikoval svoj port:

```
sudo docker container stop nginxContainer
sudo docker container rm nginxContainer
sudo docker run --rm -d --network host --name nginxContainer
nginx
curl localhost:80
```

Pozn.: Využitím prepínača –publish out_port:in_port je možné vypublikovať službu aj z kontajnera bežiaceho v bridge sieti. Nasledujúci príkaz ukazuje možné nastavenie a vypublikovanie služby bežiacej na porte 80 (v kontajneri) na port 8080 (dostupné z host zariadenia).

```
sudo docker run -dit --name nginxContainer --network
apsDockerNetwork --publish 8080:80 nginx
```

16. Zastavte vytvorené kontajnery a zmažte vytvorenú bridge sieť:

```
sudo docker container stop test
sudo docker container stop alpineContainer
sudo docker container stop nginxContainer

sudo docker container rm test
sudo docker container rm alpineContainer
sudo docker container rm nginxContainer

sudo docker network rm apsDockerNetwork
```

17. Nainštalujte docker-compose a vytvorte vyššie vytvorené sieťové prostredie využitím docker-compose.yml súboru:

```
sudo apt install docker-compose

nano docker-compose.yml


version: '3.9'

networks:
  apsDockerNetwork:
    driver: bridge

services:
  alpineContainer:
    image: alpine
    container_name: alpine_container
    networks:
      - apsDockerNetwork

  nginxContainer:
    image: nginx
    container_name: nginx_container
    networks:
      - apsDockerNetwork

sudo docker-compose up -d
sudo docker-compose down
```

Docker Swarm

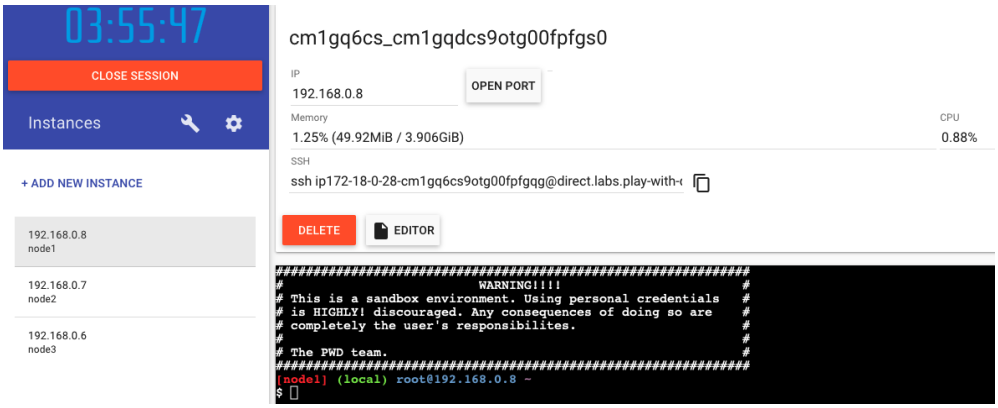
Docker Swarm je nástroj pre zabezpečenie orchestrácie kontajnerov, pričom jeho úlohou je spravovať skupinu kontajnerov a riadiť ich z jedného miesta tak, aby nasadzované aplikácie mali vysokú dostupnosť, spoľahlivosť a bolo ich možné jednoducho škálovať.

Základné pojmy v kontexte docker swarm technológie sú:

- **Node (Uzol)**: fyzické alebo VM zariadenie, na ktorom je prevádzkovaný nástroj swarm. Množinu uzlov je možné nazvať klaster.
- **Swarm Manager**: uzol, ktorý riadi vytváraný klaster.
- **Worker Node**: uzol, ktorý prevádzkuje kontajner a poskytuje výpočtové zdroje.
- **Service**: skupina kontajnerov, ktoré sú spustené na rôznych uzloch v rámci klastra. Môže to byť aplikácia alebo časť aplikácie.
- **Overlay Network**: sieť umožňujúca kontajnerom komunikovať medzi rôznymi uzlami v klastri.
- **Rolling Updates**: postupná aktualizácia služieb, umožňujúca postupný prechod na novú verziu na základe princípu blue-green, čo umožňuje minimálny výpadok aplikácii.

Swarm kontroluje počet inštancii kontajnerov v rámci klastra (vzhľadom k počtu požadovaných replík). Ak niektorý z kontajnerov bude manuálne zmazaný, alebo dôjde v ňom k chybe, tak Swarm automaticky vytvorí nový. V prípade manuálneho vymazania celej služby dôjde k vymazaniu všetkých prislúchajúcich kontajnerov na každom uzle.

Pre demonštráciu práce s docker swarm je vhodné mať viacero inštancii VM, kde bude vytváraná služba nasadzovaná. Pre jednoduchosť je možné použiť webovú aplikáciu dostupnú na: https://labs.play-with-docker.com. Jediným obmedzením je, že vytvárané prostredie bude aktívne len 4 hod, to ale pre naše účely stačí. Po prihlásení vytvorte 3 inštancie VM:



V našom prípade prvý uzol (node1) bude Master a zvyšné uzly (node2 a node3) budú worker zariadenia. Cieľom bude nasadiť webový server nginx, pričom pre zabezpečenie škálovateľnosti a lepšej dostupnosti je požiadavka, aby server bežal minimálne v 2 replikách.

Najskôr spustime master uzol:

```
docker swarm init --advertise-addr 192.168.0.8
```

```
[node1] (local) root@192.168.0.8 ~
$ docker swarm init --advertise-addr 192.168.0.8
Swarm initialized: current node (kk9d7n3pa739s0czvt663aefl) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-1z5b1kz4gvgiz52russqsuewbg5dpur1duakfa0zprdx78seif-4x3q0nojydrtznqyivzyj0nuw 192.168.0.8:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Aktuálne je v klastri jeden jediný node a to master node, čo je možné zistiť príkazom:

```
docker node ls
```

```
[node1] (local) root@192.168.0.8 ~
$ docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
kk9d7n3pa739s0czvt663aefl *   node1      Ready     Active         Leader           24.0.7
```

Teraz je možné na worker uzloch vykonať príkaz na ich registráciu do klastra:

```
docker swarm join --token SWMTKN-1-
1z5b1kz4gvgiz52russqsuewbg5dpur1duakfa0zprdx78seif-
4x3q0nojydrtznqyivzyj0nuw 192.168.0.8:2377
```

```
[node2] (local) root@192.168.0.7 ~
$ docker swarm join --token SWMTKN-1-1z5b1kz4gvgiz52russqsuewbg5dpur1duakfa0zprdx78seif-4x3q0nojydrtznqyivzyj0nuw 192.168.0.8:2377
This node joined a swarm as a worker.
```

```
[node3] (local) root@192.168.0.6 ~
$ docker swarm join --token SWMTKN-1-1z5b1kz4gvgiz52russqsuewbg5dpur1duakfa0zprdx78seif-4x3q0nojydrtznqyivzyj0nuw 192.168.0.8:2377
This node joined a swarm as a worker.
```

Úspešnosť registrácie worker uzlov je možné overiť na Master uzle príkazom **docker node ls**:

```
[node1] (local) root@192.168.0.8 ~
$ docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
kk9d7n3pa739s0czvt663aefl *   node1      Ready     Active         Leader           24.0.7
fg37goklkhijlc7xyxdeh41zu     node2      Ready     Active                          24.0.7
2wspars6zj8v852scrz7xmgxo     node3      Ready     Active                          24.0.7
```

Ako je možné vidieť, tak manažér aktuálne spravuje 2 uzly, na ktoré môže podľa potreby nasadzovať kontajnery. Teraz vytvorme jednu inštanciu nginx servera využitím definície služby:

```
docker service create –name swarmnginx nginx
```

Overenie vytvorenia služby je možné príkazom

```
docker service ls
```

```
[node1] (local) root@192.168.0.8 ~
$ docker service ls
ID             NAME          MODE         REPLICAS   IMAGE          PORTS
sghlv3y6kr6a   swarmnginx    replicated   1/1        nginx:latest
```

Detailný výpis charakterizujúci nastavenie služby je možné zobraziť príkazom **docker service inspect swarmnginx**. Tento výpis je avšak ťažko čitateľný. Vhodnejší výpis na overenie počtu aktuálne pracujúcich replík a ich umiestnenie je možné zobraziť príkazom:

```
docker service ps swarmnginx
```

```
[node1] (local) root@192.168.0.8 ~
$ docker service ps swarmnginx
ID             NAME            IMAGE          NODE      DESIRED STATE   CURRENT STATE            ERROR     PORTS
mzdjlornrg55   swarmnginx.1    nginx:latest   node1     Running         Running 4 minutes ago
```

Ako je možné vidieť, tak inštancia nginx kontajnera je prevádzkovaná na Master uzle (node1). O tomto sa je tiež možné presvedčiť zadaním príkazu na zobrazenie aktuálne pracujúcich kontajnerov príkazom **docker container ls**.

```
[node1] (local) root@192.168.0.8 ~
$ docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
cc840e32b13d   nginx:latest   "/docker-entrypoint.…"   5 minutes ago   Up 5 minutes   80/tcp    swarmnginx.1.mzdjlornrg55nvxuz9o0dhlwy
```

Vhodnejšie je ale prevádzkovať viacero inštancii (replík), čo je možné zabezpečiť aktualizáciou danej služby. Pre lepšiu viditeľnosť zabezpečme, aby boli vytvorené 4 repliky a pozorujme ich rozmiestnenie na aktuálne pracujúce uzly:

```
docker service update swarmnginx –replicas 4
```

```
[node1] (local) root@192.168.0.8 ~
$ docker service ps swarmnginx
ID             NAME             IMAGE          NODE      DESIRED STATE   CURRENT STATE            ERROR     PORTS
mzdjlornrg55   swarmnginx.1     nginx:latest   node1     Running         Running 12 minutes ago
4uypbcvsdy0q   swarmnginx.2     nginx:latest   node3     Running         Running 44 seconds ago
qmk5qfkeid3k   swarmnginx.3     nginx:latest   node2     Running         Running 44 seconds ago
myr7wk3kp2r9   swarmnginx.4     nginx:latest   node3     Running         Running 44 seconds ago
```

```
[node1] (local) root@192.168.0.8 ~
$ docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS     NAMES
cc840e32b13d   nginx:latest   "/docker-entrypoint.…"   12 minutes ago   Up 12 minutes   80/tcp    swarmnginx.1.mzdjlornrg55nvxuz9o0dhlwy
```

```
[node2] (local) root@192.168.0.7 ~
$ docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED              STATUS              PORTS     NAMES
c60e5563af5e   nginx:latest   "/docker-entrypoint.…"   About a minute ago   Up About a minute   80/tcp    swarmnginx.3.qmk5qfkeid3kzh4tlyj695wls
```

```
[node3] (local) root@192.168.0.6 ~
$ docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED              STATUS              PORTS     NAMES
f640c8dba483   nginx:latest   "/docker-entrypoint.…"   About a minute ago   Up About a minute   80/tcp    swarmnginx.4.myr7wk3kp2r9ay81d6t3do2k3
7cf97f23ff59   nginx:latest   "/docker-entrypoint.…"   About a minute ago   Up About a minute   80/tcp    swarmnginx.2.4uypbcvsdy0q8z0md3izvofyh
```

Ako je možné vidieť, tak na uzle 3 pracuje dvojica kontajnerov. Teraz skúsme vymazať jeden z existujúcich kontajnerov a overiť, že swarm automaticky zareaguje a vytvorí nový kontajner.

```
[node3] (local) root@192.168.0.6 ~
$ docker container stop f640c8dba483
f640c8dba483
```

```
[node1] (local) root@192.168.0.8 ~
$ docker service ps swarmnginx
ID             NAME                 IMAGE          NODE      DESIRED STATE   CURRENT STATE
mzdjlornrg55   swarmnginx.1         nginx:latest   node1     Running         Running 20 minutes ago
4uypbcvsdy0q   swarmnginx.2         nginx:latest   node3     Running         Running 9 minutes ago
qmk5qfkeid3k   swarmnginx.3         nginx:latest   node2     Running         Running 9 minutes ago
x1zjkb4z45cx   swarmnginx.4         nginx:latest   node1     Running         Running 4 seconds ago
myr7wk3kp2r9    \_ swarmnginx.4     nginx:latest   node3     Shutdown        Complete 9 seconds ago
```

Ako je možné vidieť, manuálne zastavenie kontajnera na uzle 3 spôsobilo automatické spustenie kontajnera na uzle 1. Aby bolo možné pristupovať k prevádzkovanej službe aj z vonkajšieho sveta, tak je potrebné isté požadované porty vypublikovať, čo je možné vykonať príznakom publish. Najskôr ale zmažme aktuálne existujúcu službu a nasaďme ju znova špecifikovaním všetkých potrebných parametrov:

```
docker service rm swarmnginx
docker service create --name swarmnginx \
                      --replicas 3 \
                      --publish published=8080,target=80 \
                      nginx
```

V tejto chvíli je možné pristupovať k webovej službe na akomkoľvek uzle na porte 8080. Nasledujúci obrázok znázorňuje aktuálny stav vytvoreného prostredia a úspešnosť dosiahnuteľnosti cieľovej služby.

```
[node1] (local) root@192.168.0.8 ~
$ docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
244a64fa0e17   nginx:latest   "/docker-entrypoint.…"   3 minutes ago   Up 3 minutes   80/tcp    swarmnginx.3.lsr4m3wf80xwkevasbq81ywpv
[node1] (local) root@192.168.0.8 ~
$ docker service ls
ID             NAME         MODE         REPLICAS   IMAGE          PORTS
98ps9ue9h7kj   swarmnginx   replicated   3/3        nginx:latest   *:8080->80/tcp
[node1] (local) root@192.168.0.8 ~
$ docker service ps swarmnginx
ID             NAME           IMAGE          NODE      DESIRED STATE   CURRENT STATE           ERROR     PORTS
cbz73pxglh3k   swarmnginx.1   nginx:latest   node2     Running         Running 3 minutes ago
i3il8zzns3i1   swarmnginx.2   nginx:latest   node3     Running         Running 3 minutes ago
lsr4m3wf80xw   swarmnginx.3   nginx:latest   node1     Running         Running 3 minutes ago
[node1] (local) root@192.168.0.8 ~
$ curl 192.168.0.7:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

Štandardne platí, že všetky služby v rámci Swarm prostredia sú dostupné vďaka tomu, že všetky uzly sú súčasťou docker ingress overlay siete, ktorá má v sebe zabudovaný automatizovaný load balancing a umožňuje prepojenie všetkých kontajnerov v rámci klastra. Požiadavka na akýkoľvek uzol v sieti, je vďaka LB doručená na niektorý z dostupných kontajnerov (aj požiadavka prijatá na kontajner, na ktorom žiaden kontajner nie je):

```
[node1] (local) root@192.168.0.8 ~
$ docker service update swarmnginx --replicas 2
swarmnginx
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
[node1] (local) root@192.168.0.8 ~
$ docker service ps swarmnginx
ID             NAME           IMAGE          NODE      DESIRED STATE   CURRENT STATE            ERROR     PORTS
cbz73pxglh3k   swarmnginx.1   nginx:latest   node2     Running         Running 23 minutes ago
i3il8zzns3i1   swarmnginx.2   nginx:latest   node3     Running         Running 23 minutes ago
[node1] (local) root@192.168.0.8 ~
$ curl 192.168.0.8:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

Ako je možné vidieť na vyššie uvedenom obrázku, tak došlo k zníženiu počtu replík na 2, výsledkom čoho bolo, že na node1 aktuálne nie je nasadený žiaden kontajner. Napriek tomu, v prípade, že bol poslaný request na dané zariadenie, tak došlo k úspešnému pripojeniu sa k webovej službe.

Pre nasadzovanie služieb definovaných v docker compose súbore v kontexte swarm je možné použiť koncept **stack**, ktorý umožňuje špecifikovať spustenie celých aplikácii.

```
version: '3.8'

services:
  swarmnginx:
      image: nginx
      ports:
        - "8080:80"
      deploy:
        replicas: 3
```

```
[node1] (local) root@192.168.0.8 ~
$ cat docker-compose.yml
version: '3.8'

services:
  swarmnginx:
    image: nginx
    ports:
      - "8080:80"
    deploy:
      replicas: 3

[node1] (local) root@192.168.0.8 ~
$ docker stack deploy -c docker-compose.yml swarmstack
Updating service swarmstack_swarmnginx (id: oq8olqd0w6f3sl8wmbjekkm4u)
[node1] (local) root@192.168.0.8 ~
$ docker service ls
ID             NAME                     MODE         REPLICAS   IMAGE          PORTS
oq8olqd0w6f3   swarmstack_swarmnginx    replicated   3/3        nginx:latest   *:8080->80/tcp
[node1] (local) root@192.168.0.8 ~
$ docker service ps swarmstack_swarmnginx
ID             NAME                       IMAGE          NODE      DESIRED STATE   CURRENT STATE
2482dyhpikm7   swarmstack_swarmnginx.1    nginx:latest   node1     Running         Running about a minute ago
2hyjpnkvlw6y   swarmstack_swarmnginx.2    nginx:latest   node2     Running         Running about a minute ago
jk683nye408e   swarmstack_swarmnginx.3    nginx:latest   node3     Running         Running about a minute ago
```

Po úprave docker-compose súboru a opätovného nasadenia stacku dôjde k automatickému znovunasadeniu služieb, pričom staré budú vypnuté a nové spustené:

```
[node1] (local) root@192.168.0.8 ~
$ cat docker-compose.yml
version: '3.8'

services:
  swarmnginx:
    image: nginx
    ports:
      - "8081:80"
    deploy:
      replicas: 4

[node1] (local) root@192.168.0.8 ~
$ docker stack deploy -c docker-compose.yml swarmstack
Updating service swarmstack_swarmnginx (id: oq8olqd0w6f3sl8wmbjekkm4u)
[node1] (local) root@192.168.0.8 ~
$ docker service ps swarmstack_swarmnginx
ID             NAME                       IMAGE          NODE      DESIRED STATE   CURRENT STATE
1ikm3e3bjf2m   swarmstack_swarmnginx.1    nginx:latest   node1     Running         Running 4 seconds ago
2482dyhpikm7    \_ swarmstack_swarmnginx.1   nginx:latest   node1     Shutdown        Shutdown 5 seconds ago
e3m5tx1mlmoc   swarmstack_swarmnginx.2    nginx:latest   node2     Running         Starting less than a second ago
2hyjpnkvlw6y    \_ swarmstack_swarmnginx.2   nginx:latest   node2     Shutdown        Shutdown less than a second ago
jk683nye408e   swarmstack_swarmnginx.3    nginx:latest   node3     Running         Running 4 minutes ago
i88ze6vn3eku   swarmstack_swarmnginx.4    nginx:latest   node1     Running         Running 8 seconds ago
[node1] (local) root@192.168.0.8 ~
$ docker service ls
ID             NAME                     MODE         REPLICAS   IMAGE          PORTS
oq8olqd0w6f3   swarmstack_swarmnginx    replicated   4/4        nginx:latest   *:8081->80/tcp
[node1] (local) root@192.168.0.8 ~
$ curl 192.168.0.7:8081
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

Komunikácia medzi kontajnermi je štandardne posielaná bez zabezpečenia (len riadiaca komunikácia medzi swarm uzlami je šifrovaná). Swarm umožňuje zabezpečiť komunikáciu prebiehajúcu medzi uzlami zabezpečením šifrovania pri prenose dát Overlay sieťou, čo je možné nastaviť príznakom **–opt encrypted**:

```
docker network create --driver overlay --ingress   --
subnet=10.11.0.0/16 --gateway=10.11.0.2 --opt encrypted  secure-
ingress
```

Pred tým, ako bude vytvorená nová ingress sieť, tak je potrebné tú starú zmazať. Zmazanie je ale možné až po tom, ako žiadna služba nevyužíva danú sieť. Po nasadení nového stacku sú automaticky všetky uzly priradené do tejto novej siete, ktorá je zároveň šifrovaná, čo je možné overiť nasledujúcimi príkazmy:

```
[node1] (local) root@192.168.0.8 ~
$ docker network create --driver overlay --ingress   --subnet=10.11.0.0/16 --gateway=10.11.0.2 --opt encrypted  secure-ingress
ubhxvu9th263w5y13wvk3sfsa
[node1] (local) root@192.168.0.8 ~
$ docker stack deploy -c docker-compose.yml swarmstack
Creating service swarmstack_swarmnginx
[node1] (local) root@192.168.0.8 ~
$ docker network inspect secure-ingress
[
    {
        "Name": "secure-ingress",
        "Id": "ubhxvu9th263w5y13wvk3sfsa",
        "Created": "2023-12-20T18:36:35.065643691Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.11.0.0/16",
                    "Gateway": "10.11.0.2"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": true,
```

```
    "Containers": {
        "bcf5a61b7f122c7369fedc871ea886651835b531690ad64b96cd0761a092dac2": {
            "Name": "swarmstack_swarmnginx.3.1e3mobzdxfn674gds29u2gfze",
            "EndpointID": "b30b70d5ff6dc27410c835053bb6b353d1a30915c92e672927fef62f08b076ed",
            "MacAddress": "02:42:0a:0b:00:06",
            "IPv4Address": "10.11.0.6/16",
            "IPv6Address": ""
        },
        "secure-ingress-sbox": {
            "Name": "secure-ingress-endpoint",
            "EndpointID": "5d9b5960c21af89accf9022bf4e14b606520469799fe1646d3c8e23dd1e51095",
            "MacAddress": "02:42:0a:0b:00:08",
            "IPv4Address": "10.11.0.8/16",
            "IPv6Address": ""
        }
    },
    "Options": {
        "com.docker.network.driver.overlay.vxlanid_list": "4101",
        "encrypted": ""
    },
    "Labels": {},
    "Peers": [
        {
            "Name": "f4d6f962de79",
            "IP": "192.168.0.8"
        },
        {
            "Name": "a17fd1c3573d",
            "IP": "192.168.0.6"
        },
```

Ako je možné vidieť, tak vytvorený kontajner sa nachádza vo vytvorenej šifrovanej sieti.

Kubernetes

Kubernetes (k8s) je orchestračný nástroj pre správu (automatizované nasadzovanie a škálovanie) komplexných kontajnerovo založených aplikácii a mikroslužieb. Základná terminológia v kontexte kubernetes technológie je nasledovna:

- **Pod**: skupina pozostávajúca z jedného alebo viacerých, logicky súvisiacich, kontajnerov zdieľajúcich zdroje na rovnakom uzle.
- **Node**: fyzický server alebo VM v klastri slúžiaci na spúšťanie a prevádzkovanie kontajnerov.
- **Klaster**: skupina uzlov.
- **Master**: uzol, ktorý riadi klaster.
- **Deployment**: objekt, ktorý definuje stav aplikácie a riadi spôsob jej nasadzovania.
- **Service**: predstavuje prístup k skupine zariadení.
- **ReplicaSet**: definuje počet kópií pod-ov, ktoré majú byť spustené a udržiava ich počet konzistentný.
- **Ingress**: umožňuje prístup k službám v klastri z vonkajšieho sveta.

Pre jednoduchosť bude práca s Kubernetes demonštrovaná využitím **Minikube** technológie, ktorá predstavuje prácu s klastrami na jednom zariadení. V prípade práce na viacerých zariadeniach je potrebné použiť technológiu **Kubeadm** a zabezpečiť networking využitím jedného z plugin-ov (napr. Calico). Inštalácia a spustenie minikube je možná využitím nasledujúcich príkazov:

```
curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
sudo usermod -aG docker $USER && newgrp docker
minikube start –driver=docker
minikube dashboard
```

Minikube dashboard je používateľské rozhranie, vďaka ktorému je možné interagovať s minikube prostredím a spravovať nasadzované aplikácie. Základné zobranie informácii o klastri je možné zobraziť príkazom:

```
kubectl cluster-info
```

```
┌──(kali㉿kali)-[~]
└─$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Vytvorenie jednoduchého uzla pre nginx kontajner je možné využitím príkazov:

```
kubectl run kubnginx –image nginx
kubectl get pods
```

```
┌──(kali㉿kali)-[~]
└─$ kubectl run kubnginx --image nginx
pod/kubnginx created

┌──(kali㉿kali)-[~]
└─$ kubectl get pods
NAME       READY   STATUS              RESTARTS   AGE
kubnginx   0/1     ContainerCreating   0          7s
```

Bližšie informácie (IP adresa, kontajner ID, …) o vytvorenom Pod-e je možné zobraziť tiež príkazom:

```
kubectl describe pod kubnginx
```

```
┌──(kali㉿kali)-[~]
└─$ kubectl get pods
NAME       READY   STATUS    RESTARTS   AGE
kubnginx   1/1     Running   0          2m57s

┌──(kali㉿kali)-[~]
└─$ kubectl describe pod kubnginx
Name:             kubnginx
Namespace:        default
Priority:         0
Node:             minikube/192.168.49.2
Start Time:       Thu, 21 Dec 2023 01:43:04 -0500
Labels:           run=kubnginx
Annotations:      <none>
Status:           Running
IP:               10.244.0.5
IPs:
  IP:  10.244.0.5
Containers:
  kubnginx:
    Container ID:   docker://2587abd29347d2fc1f909df2127c2634388420a52f859f5c7eb447cf088ce4a0
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:bd30b8d47b230de52431cc71c5cce149b8d5d4c87c204902acf2504435d4b4c9
```

Ďalším užitočným príkazom je príkaz:

```
kubectl get pods -o wide
```

```
┌──(kali㉿kali)-[~]
└─$ kubectl get pods -o wide
NAME       READY   STATUS    RESTARTS   AGE   IP           NODE
kubnginx   1/1     Running   0          38m   10.244.0.5   minikube
```

K vytvorenému kontajneru pracujúcemu v rámci Pod-u sa je možné dostať referenciou na názov Pod-u, pričom štandarde sa používa otvorenie jeho terminálu alebo zobrazenie logových správ:

```
kubectl exec -ti kubnginx -- bash
kubectl logs kubnginx
```

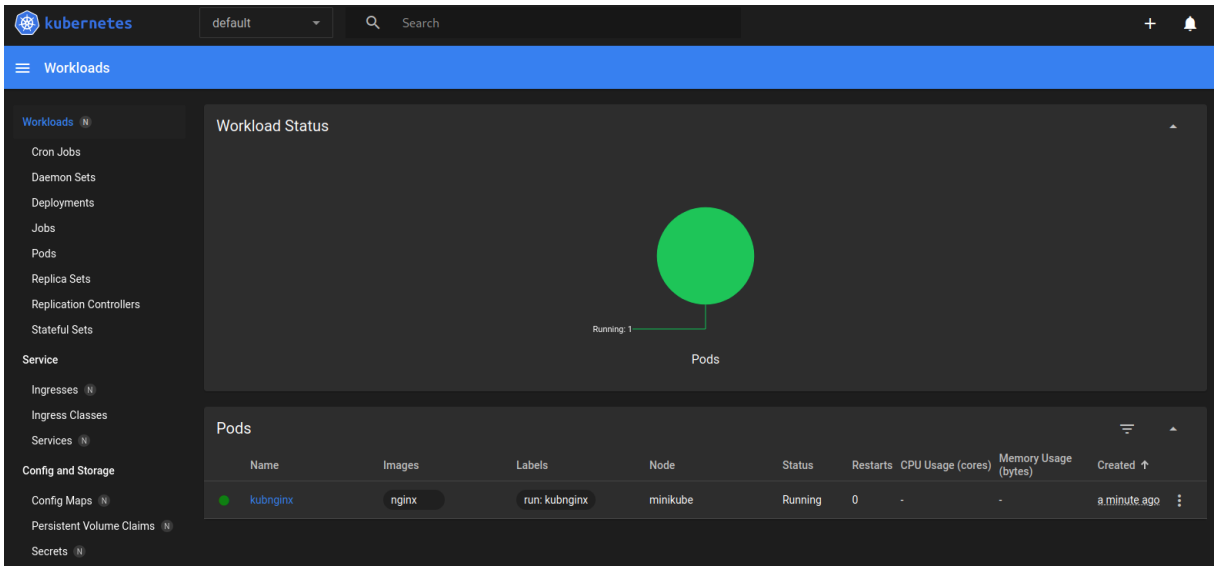
```
┌──(kali㉿kali)-[~]
└─$ kubectl exec -ti kubnginx -- bash
root@kubnginx:/#
root@kubnginx:/# curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@kubnginx:/# exit
exit

┌──(kali㉿kali)-[~]
└─$ kubectl logs kubnginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/12/21 06:43:40 [notice] 1#1: using the "epoll" event method
2023/12/21 06:43:40 [notice] 1#1: nginx/1.25.3
```

Detailný výpis o vytvorených zdrojoch v rámci kubernetes prostredia je možné tiež zobraziť využitím otvoreného ovládacieho panelu.



Zdroje v rámci kubernetes je možné definovať využitím YAML súborov, ktoré obsahujú 4 hlavné časti:

- **apiVersion**: definuje kubernetes API (v1, apps/v1beta1, extensions/v1beta1)
- **kind**: definuje typ vytváraného objektu (pod, replicaSet, deployment, service)
- **metadata**: charakteristika objektu (label, name) definované v podobe objektu
- **spec**: špecifikuje nasadzovaný kontajner (image, …)

Nasadenie nginx kontajnera môže byť vykonané využitím konfiguračného súboru nasledovne (častejšie využívaný prístup, pričom názov môže byť ľubovoľný - napr. pod-def.yml):

```
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx
  labels:
    app-name: my-nginx-app
spec:
  containers:
    - name: kubnginx
      image: nginx
      ports:
        - containerPort: 80
```

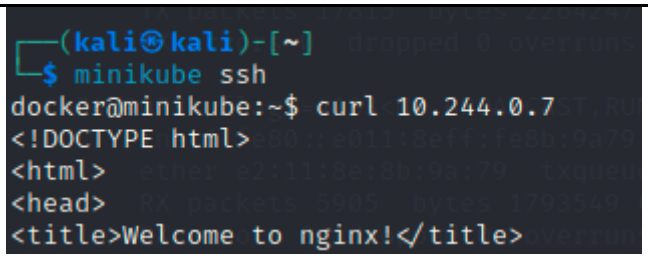
Dôležitým atribútom je **labels**, ktorým sa je možné neskôr na daný pod odkazovať. Nasadenie vytvoreného Pod objektu je možné využitím príkazu:

```
kubectl create -f pod-def.yml
kubectl get pods
```

```
┌──(kali㉿kali)-[~]
└─$ kubectl get pods -o wide
NAME       READY   STATUS    RESTARTS   AGE   IP           NODE
my-nginx   1/1     Running   0          27m   10.244.0.7   minikube
```

Overiť funkčnosť webovej služby je možné priamo z minikube kontajnera, na ktorý sa je ale potrebné pripojiť protokolom SSH:

```
minikube ssh
curl 10.244.0.7
```

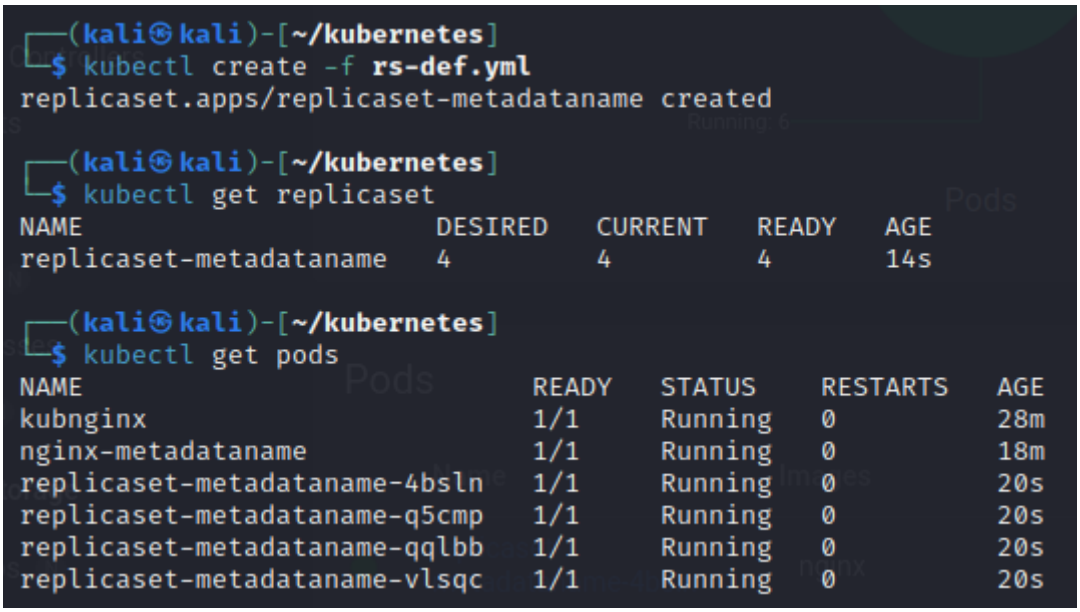


Štandardne je potrebné, aby daný Pod fungoval z pohľadu redundancie vo viacerých replikách, čo je možné dosiahnuť využitím ReplicaSet objektu. ReplicaSet kontroluje stav bežiacich Pod-ov a v prípade výpadku niektorého z podov automaticky nasadí nový tak, aby bol ich počet konzistentný:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: replicaset-metadataname
  labels:
      app: my-nginx-website
      tier: fe
spec:
  replicas: 4
  template:
      metadata:
         name: myapp-pod
         labels:
            app: myapp
      spec:
         containers:
            - name: nginx
               image: nginx
  selector:
      matchLabels:
         app: myapp
```

Selektor pri ReplicaSet musí byť použitý, lebo ReplicaSet riadi nie len aktuálne vytvárané Pod-y ale aj všetky pred tým vytvorené (súčasťou počtu replík bude aj predtým vytvorený Pod).

```
kubectl create -f replicaset-def.yml
kubectl get replicaset
kubectl get pods
```



V prípade, že je potrebné zmeniť obsah vytvorenej repliky, tak je na to možné použiť príkaz replace. Na vymazanie repliky je zas možné použiť príkaz delete:

```
kubectl replace -f rs-def.yml
kubectl delete replicaset replicaset-metadataname
```

Ďalším veľmi užitočným objektom je **Deployment**, ktorý obsahuje všetko vyššie spomenuté Deployment = [ReplicaSet[Pod]]. Umožňuje spravovať nasadzovanie verzii aplikácii použitím postupných aktualizácii. Umožňuje spravovať verzie aplikácii a ich rollback. Konfiguračný súbor vyzerá nasledovne:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
      app: mywebsite
      tier: frontend
spec:
  replicas: 4
  template:
      metadata:
      name: myapp-pod
      labels:
        app: myapp
      spec:
        containers:
        - name: nginx
          image: nginx
  selector:
      matchLabels:
        app: myapp
```

Spustiť vytvorený deployment je možné využitím nasledujúcich príkazov:

```
kubectl create -f dep-def.yml
kubectl get deployments
kubectl get replicaset
kubectl get pods
```

```
┌──(kali㉿kali)-[~/kubernetes]
└─$ kubectl get deployments
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
frontend   3/4     4            3           12s

┌──(kali㉿kali)-[~/kubernetes]
└─$ kubectl get replicaset
NAME                  DESIRED   CURRENT   READY   AGE
frontend-6f5f4b44ff   4         4         4       25s

┌──(kali㉿kali)-[~/kubernetes]
└─$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
frontend-6f5f4b44ff-4rhsp   1/1     Running   0          36s
frontend-6f5f4b44ff-9×2nl   1/1     Running   0          36s
frontend-6f5f4b44ff-v7xf6   1/1     Running   0          36s
frontend-6f5f4b44ff-wn5sp   1/1     Running   0          36s
kubnginx                    1/1     Running   0          42m
nginx-metadataname          1/1     Running   0          32m
```

Kubernetes umožňuje rollout, čo je proces postupného nasadenie novej verzie aplikácie. Každý rollout (aktualizácia) vytvorí nové revízne číslo, vďaka čomu sa je neskôr možné vrátiť k pôvodnej verzii aplikácie. Štandardne sa používa Rolling Update technika, ktorá pri nasadzovaní novej aplikácie postupne, jeden Pod za druhým, najskôr vypne starú verziu a vytvorí Pod s novou. Výsledkom je, že aplikácia je stále funkčná, akurát niektoré Pod-y majú ešte starú verziu a niektoré už aktualizovanú.

Pre aktualizáciu obsahu, aplikovanie zmien a spätný návrat na predchádzajúcu verziu je možné použiť nasledujúce sériu príkazov:

```
┌──(kali㉿kali)-[~/kubernetes]
└─$ kubectl apply -f dep-def.yml
deployment.apps/frontend configured

┌──(kali㉿kali)-[~/kubernetes]
└─$ kubectl rollout status deployment/frontend
Waiting for deployment "frontend" rollout to finish: 3 of 4 updated replicas are available...
deployment "frontend" successfully rolled out

┌──(kali㉿kali)-[~/kubernetes]
└─$ kubectl rollout history deployment/frontend
deployment.apps/frontend
REVISION  CHANGE-CAUSE
2         <none>
3         <none>


┌──(kali㉿kali)-[~/kubernetes]
└─$ kubectl rollout undo deployment/frontend
deployment.apps/frontend rolled back

┌──(kali㉿kali)-[~/kubernetes]
└─$ kubectl rollout status deployment/frontend
Waiting for deployment "frontend" rollout to finish: 3 out of 4 new replicas have been updated...
Waiting for deployment "frontend" rollout to finish: 3 out of 4 new replicas have been updated...
Waiting for deployment "frontend" rollout to finish: 3 out of 4 new replicas have been updated...
Waiting for deployment "frontend" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "frontend" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "frontend" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "frontend" rollout to finish: 3 of 4 updated replicas are available...
deployment "frontend" successfully rolled out
```

```
kubectl apply -f dep-def.yml
kubectl rollout status deployment/frontend
kubectl rollout history deployment/frontend
kubectl rollout undo deployment/myapp-deployment
```

Štandardne každý Pod má priradenú IP adresu z rozsahu 10.244.0.0/16. Pre úspešnú komunikáciu z inými Pod-mi na iných Node-s mimo lokálnej siete je potrebné použiť plugin zabezpečujúci vytvorenie komunikačnej siete (napr. calico).

Pre sprístupnenie konkrétnych služieb, bežiacich vo vnútri Pod-u, aj z vonkajšieho sveta, je možné použiť objekt Service. Komunikácia je štandardne povolená len medzi Pod-mi navzájom a medzi Pod-mi a lokálnym zariadením, kde sú nasadené. Pre zabezpečenie komunikácie z vonka sveta je potrebné premostiť port dostupný z externých zariadení s portom vo vnútri Pod-u.

Vytvorme nasledujúci Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx
  labels:
      app: my-nginx-app
spec:
  containers:
      - name: kubnginx
      image: nginx
      ports:
      - containerPort: 80
```

Definovanie služby je možné využitím nasledujúceho konfiguračného súboru:

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service

spec:
  type: NodePort
  ports:
  - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
      app: my-nginx-app
```

Pozn.: Ak existuje viacero Pod-ov s tým istým selektorom, tak služba je namapovaná na každý jeden Pod a používa náhodný algoritmus na rozkladanie záťaže.

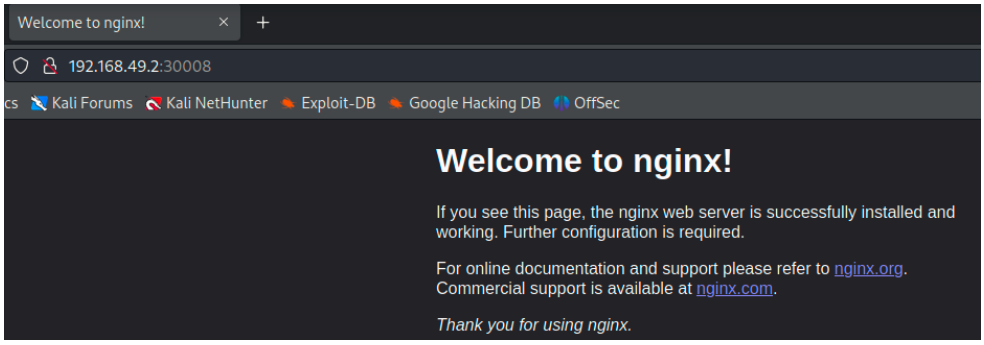
Dostupnosť služby je možné zobraziť nasledujúcim príkazom:

```
minikube service myapp-service --url
```

```
┌──(kali㉿kali)-[~]
└─$ kubectl apply -f pod-def.yml
pod/my-nginx created

┌──(kali㉿kali)-[~]
└─$ kubectl apply -f s.yml
service/myapp-service created

┌──(kali㉿kali)-[~]
└─$ minikube service myapp-service --url
http://192.168.49.2:30008
```



Poslednou úlohou je zamedziť prístupu medzi dvojicou podov. Pre tento účel je možné využiť objekt NetworkPolicy. Avšak, aby bolo možné aplikovať politiky, tak je potrebné pracovať s ovládačom sieťovej karty, ktorý aplikovanie politík umožňuje. Príkladom je Cilium, ktorý je možné inštalovať nasledovne:

```
curl -LO https://github.com/cilium/cilium-
cli/releases/download/v0.4/cilium-linux-amd64.tar.gz
tar xzvf cilium-linux-amd64.tar.gz
sudo mv cilium /usr/local/bin
cilium install
cilium status
```

```
┌──(kali㉿kali)-[~]
└─$ cilium status
    /¯¯\
 /¯¯\__/¯¯\    Cilium:         OK
 \__/¯¯\__/    Operator:       OK
 /¯¯\__/¯¯\    Hubble:         1 warnings
 \__/¯¯\__/    ClusterMesh:    1 warnings
    \__/

Deployment        cilium-operator          Desired: 1, Ready: 1/1, Available: 1/1
DaemonSet         cilium                   Desired: 1, Ready: 1/1, Available: 1/1
Containers:       cilium                   Running: 1
                  cilium-operator          Running: 1
Image versions    cilium                   quay.io/cilium/cilium:v1.9.4: 1
                  cilium-operator          quay.io/cilium/operator-generic:v1.9.4: 1
Warnings:         hubble-relay             hubble-relay            hubble relay is not deployed
                  clustermesh-apiserver    clustermesh-apiserver   clustermesh is not deployed
```

Od tejto chvíle je možné aplikovať na sériu Pod-ov pravidlá komunikácie. Štandardne dokáže komunikovať každý Pod s každým. V prípade, že aplikujeme na daný Pod politiku, bez povolenia

komunikácie, tak je všetka komunikácia defaultne zakázaná, čo je možné nastaviť nasledujúcim výpisom:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: network-policy

spec:
  podSelector:
      matchLabels:
        app: my-nginx-app
  policyTypes:
      - Ingress
      - Egress
```

Ak by bolo v PolicyTypes definované napr. len Ingress, tak Egress komunikácia by bola naďalej bez obmedzenia. Povolenie kompletnej komunikácie so zariadením my-nginx zo zariadenia my-nginx-app v oboch smeroch je možné definovať nasledovne:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: network-policy

spec:
  podSelector:
      matchLabels:
        app: my-nginx
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: my-nginx-app
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: my-nginx-app
```

Na obmedzenie komunikácie s databázou je možné použiť nasledovnú politiku:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-policy
spec:
  podSelector:
      matchLabels:
        name: my-db-mysql
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: be
      ports:
      - protocol: TCP
          port: 3306
  policyTypes:
  - Ingress
```

Príklad povolenia všetkej vstupujúcej komunikácie v clustri je nasledovný:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress
spec:
  podSelector: {}
  ingress:
  - {}
  policyTypes:
  - Ingress
```

Komplexný príklad na záver

Majme informačný systém pozostávajúci s troch prvkov a to backend, frontend a databázu. Označme ich ako my-server, my-web a my-db.

1. Nasaďme vyššie uvedené uvedené kontajnery a otestujme funkčnosť komunikácie medzi nimi navzájom (politiky je možné nasadiť štandardným príkazom apply, pričom viacero politík môže byť definovaných v rámci toho istého yaml súboru - môžu byť ale rozdelené aj do viacerých súborov):

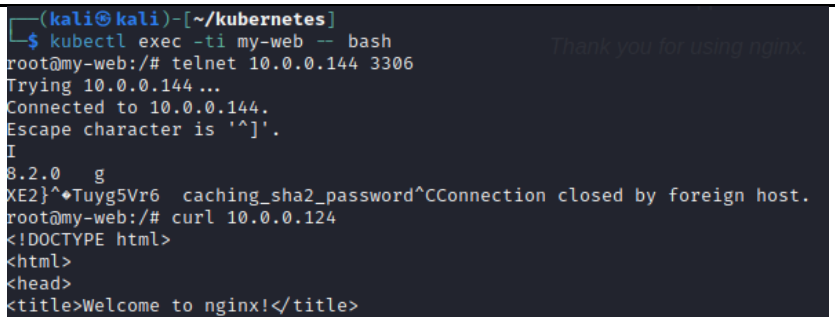
   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: my-web
     labels:
        role: fe
   spec:
     containers:
     - name: nginx
   ```

```
    image: nginx
     ports:
     - name: web
       containerPort: 80
       protocol: TCP
---
apiVersion: v1
kind: Pod
metadata:
  name: my-server
  labels:
      role: be
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - name: http
       containerPort: 80
       protocol: TCP
---
apiVersion: v1
kind: Pod
metadata:
  name: my-db
  labels:
      name: mysql
spec:
  containers:
  - name: mysql
    image: mysql:latest
    env:
    - name: "MYSQL_USER"
      value: "mysql"
    - name: "MYSQL_PASSWORD"
      value: "mysql"
    - name: "MYSQL_DATABASE"
      value: "test"
    - name: "MYSQL_ROOT_PASSWORD"
      value: "rootpass"
    ports:
    - name: http
      containerPort: 3306
       protocol: TCP
```

```
┌──(kali㉿kali)-[~/kubernetes]
└─$ kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE     IP           NODE
my-db       1/1     Running   0          4m13s   10.0.0.144   minikube
my-server   1/1     Running   0          4m13s   10.0.0.124   minikube
my-web      1/1     Running   0          4m13s   10.0.0.119   minikube
```

Pre test pripojenia sa na databázu je možné do kontajnera nainštalovať protokol telnet a následne sa pripojiť k databáze.

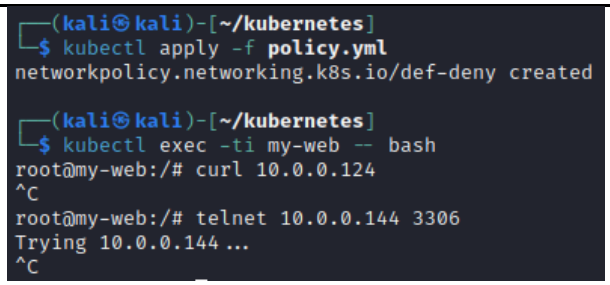
```
kubectl exec -ti my-web -- bash
apt update && apt install telnet -y
telnet 10.0.0.144 3306
curl 10.0.0.124
```



Ako je možné vidieť, tak všetka komunikácia je povolená. Rovnaký výstup je možné získať aj z Pod-u my-server.

2. Nasaďte politiku, ktorá zakáže všetku komunikáciu v smere inside v klastri:

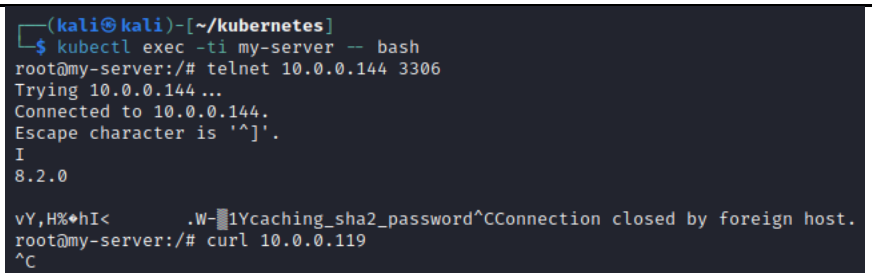
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: def-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```



3. Nastavte politiku tak, aby sa na databázu dalo pripojiť len z backend kontajnera:

```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: permit-db
spec:
  podSelector:
     matchLabels:
       name: mysql
  policyTypes:
  - Ingress
  ingress:
```

```
  - from:
      - podSelector:
          matchLabels:
            role: be
      ports:
      - protocol: TCP
      port: 3306
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: permit-be
spec:
  podSelector:
      matchLabels:
        role: be
  policyTypes:
  - Ingress
  ingress:
  - {}
```



4. Doplňte bezpečnostnú politiku tak, aby sa na web dalo pripojiť odkiaľkoľvek:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: permit-fe
spec:
  podSelector:
      matchLabels:
        role: fe
  policyTypes:
  - Ingress
  ingress:
  - {}
```

Overenie sieťových politík je možné využitím nasledujúcich príkazov:

```
kubectl get networkpolicies
kubectl describe networkpolicies permit-db
```