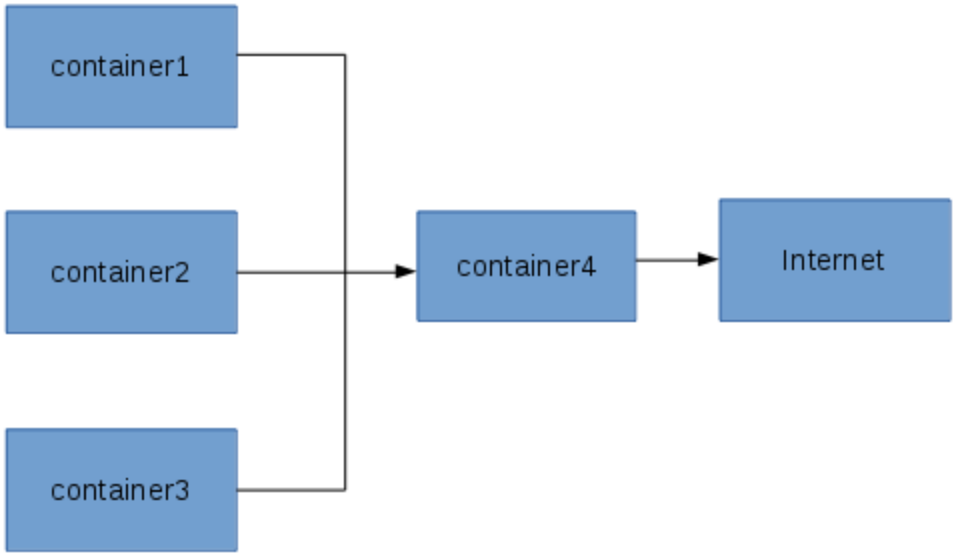
Restrict Internet Access - Docker Container

Asked 8 years, 5 months ago Modified 1 year, 11 months ago Viewed 81k times

92

I have a situation to restrict internet access of the container in load balancer network. for example in that below picture



Only **container4** connects to the Internet; other **three** only communicate through **container4** with the outside world. For example if **container1** needs smtp support, it will forward smtp request to **container4** to get access.

No container other than **container4** should be allowed to access the Internet directly! This should be enforced on Docker level.

I believe it will be configurable on **docker network creation**, can any one explain how to achieve this?

docker docker-network

Share Improve this question Follow

edited Feb 19, 2017 at 16:59
knaperek
2,253 ● 26 ● 42

asked Oct 7, 2016 at 9:20
Bilal Usean
2,474 ● 3 ● 25 ● 51

5 Answers

Sorted by: Highest score (default)

As found here, I got this to work with docker-compose. Save as `docker-compose.yml`:

113

```
version: '3'

services:
  outgoing-wont-work:
    image: alpine
    networks:
      - no-internet
    command: ping -c 3 google.com # will crash

  internal-will-work:
    image: alpine
    networks:
      - no-internet
    command: ping -c 3 internal-and-external

  internal-and-external:
    image: alpine
    networks:
      - no-internet
      - internet
    command: ping -c 3 google.com

networks:
  no-internet:
    driver: bridge
    internal: true
  internet:
    driver: bridge
```

Then run `docker-compose up -d`, `docker-compose ps` will show something like this after a few seconds:

```
              Name                            Command               State    Ports
-------------------------------------------------------------------------------------
dco_inet_internal-and-external_1   ping -c 3 google.com             Exit 0
dco_inet_internal-will-work_1      ping -c 3 internal-and-ext ...   Exit 0
dco_inet_outgoing-wont-work_1      ping -c 3 google.com             Exit 1
```

Share Improve this answer Follow

edited Mar 19, 2019 at 16:16

answered Aug 22, 2018 at 9:51

exic

2,698 ● 2 ● 26 ● 30

3 I wish this worked for me. We're running Traefik as a reverse proxy in front of the application containers and Squid as a proxy for outgoing HTTP(S) requests *from* the application container to the internet. I tried your solution but the app container still has access to the internet, I think through the network it shares with Traefik. – Tobias Sep 2, 2021 at 16:52

49

Network creation for access internet

> docker network create --subnet=172.19.0.0/16 internet

Network creation for block internet access

> docker network create --internal --subnet 10.1.1.0/24 no-internet

If you want to connect docker container into internet

```
docker network connect internet container-name
```

If you want to block internet access

```
docker network connect no-internet container-name
```

Note

in internal network we can't expose ports to connect outside world, please refer this question for more details

Share Improve this answer Follow

edited May 23, 2017 at 11:46
Community Bot
1 1

answered Feb 20, 2017 at 4:33

Bilal Usean
2,474 3 25 51

14 Do you know how to configure this with Docker Compose? – knaperek Feb 24, 2017 at 13:01

I don't know @JozefKnaperek – Bilal Usean Feb 24, 2017 at 13:05

The containers on the `no-internet` network will still be able to make DNS queries; read this – Bharat Khatri Jan 27, 2018 at 15:49

1 @BharatKhatri Yeah, but not through the internet. – Spooky Mar 25, 2018 at 9:28

2 @JozefKnaperek see my answer for a solution with docker-compose. – exic Aug 22, 2018 at 9:51

10

As stated in Bilal's answer, the internal network is a good solution if you do not need to expose the ports.

If you do need to expose the ports, the below solution using iptables does the job for my requirements:

```
docker network create --subnet 172.19.0.0/16 no-internet
sudo iptables --insert DOCKER-USER -s 172.19.0.0/16 -j REJECT --reject-with icmp-port-
unreachable
sudo iptables --insert DOCKER-USER -s 172.19.0.0/16 -m state --state
RELATED,ESTABLISHED -j RETURN
```

Then add

```
--network no-internet
```

when you run your docker container. For instance:

```
$ docker run -it --network no-internet ubuntu:focal /bin/bash
root@9f2181f79985:/# apt update
Err:1 http://archive.ubuntu.com/ubuntu focal InRelease
  Temporary failure resolving 'archive.ubuntu.com'
```

Share Improve this answer Follow

answered Oct 21, 2020 at 13:33

beledouxdenis
221 ● 3 ● 5

Do you know how to do this on windows host machine? – dkregen Dec 11, 2021 at 4:03

1 I can confirm this works, so thank you! 2 questions: 1) Why does the REJECT come before the RELATED,ESTABLISHED line? Doesn't this order mean that REJECT takes precedence and renders latter rule useless? 2) Why icmp-port-unreachable for the reject reason? – John Lee Mar 3, 2022 at 15:43

4

Another option, if you need to expose ports on a container without internet access, but want to let it talk to other containers would be to provide a bogus DNS configuration. This isn't a perfect solution though, since it doesn't prevent direct IP access to the outside world.

docker-compose.yaml

```
version: '3'

services:
  service1:
    image: alpine
    command: sh -c 'ping service2 -c 1; ping google.com -c 1'
    dns: 0.0.0.0
  service2:
    image: alpine
    command: sh -c 'ping service1 -c 1; ping google.com -c 1'
    dns: 0.0.0.0
```

```
isolated> docker-compose up
Recreating isolated_service1_1 ... done
Recreating isolated_service2_1 ... done
Attaching to isolated_service2_1, isolated_service1_1
service1_1  | PING service2 (172.18.0.2) 56(84) bytes of data.
service1_1  | 64 bytes from isolated_service2_1.isolated_default (172.18.0.2):
icmp_seq=1 ttl=64 time=0.038 ms
service1_1  |
service1_1  | --- service2 ping statistics ---
service1_1  | 1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
service1_1  | rtt min/avg/max/mdev = 0.038/0.038/0.038/0.000 ms
service2_1  | PING service1 (172.18.0.3) 56(84) bytes of data.
service2_1  | 64 bytes from isolated_service1_1.isolated_default (172.18.0.3):
icmp_seq=1 ttl=64 time=0.093 ms
service2_1  |
service2_1  | --- service1 ping statistics ---
service2_1  | 1 packets transmitted, 1 received, 0% packet loss, time 0ms
service2_1  | rtt min/avg/max/mdev = 0.093/0.093/0.093/0.000 ms
service1_1  | ping: google.com: Temporary failure in name resolution
service2_1  | ping: google.com: Temporary failure in name resolution
isolated_service1_1 exited with code 2
isolated_service2_1 exited with code 2
```

Share Improve this answer Follow

answered Apr 16, 2020 at 5:34

Keegan
12.2k ● 1 ● 30 ● 38

It isn't perfect but yes it might mitigate most of internet access attempts from softwares. – secavfr May 8, 2020 at 13:43

5 I would like to disagree it dose not restrict anything and therefor so far from perfect... – Lenard Jul 18, 2020 at 22:01

This works well for requests that require DNS resolution and is a good workaround for cases where the network based restriction isn't feasible. Our setup involves a Traefik instance in front of the application container and in that case, the other solutions don't seem to work. – Tobias Sep 2, 2021 at 16:50

3

For blocking outgoing (internet) while exposing ports to the internal LAN network

`--internal` or `internal:true` does not allow exposing ports to the internal network.

1. backup your current rules

```
sudo iptables-save > iptables.backup
```

2. (optional) flush your existing rules under `DOCKER-USER`

```
sudo iptables -F DOCKER-USER
```

3. add these rules in this order (assuming the docker IP range is `10.0.1.0/24`)

```
sudo iptables -A DOCKER-USER -s 10.0.1.0/24 -d 192.168.0.0/16 -j RETURN
sudo iptables -A DOCKER-USER -s 10.0.1.0/24 -d 172.16.0.0/12 -j RETURN
sudo iptables -A DOCKER-USER -s 10.0.1.0/24 -d 10.0.0.0/8 -j RETURN
sudo iptables -A DOCKER-USER -s 10.0.1.0/24 -j REJECT --reject-with icmp-port-
unreachable
sudo iptables -A DOCKER-USER -j RETURN
```

4. `sudo iptables --list --line-numbers` should show:

```
Chain DOCKER-USER (1 references)
num  target     prot opt source               destination
1    RETURN     all  --  10.0.1.0/24          192.168.0.0/16
2    RETURN     all  --  10.0.1.0/24          172.16.0.0/12
3    RETURN     all  --  10.0.1.0/24          10.0.0.0/8
4    REJECT     all  --  10.0.1.0/24          anywhere             reject-with icmp-
port-unreachable
5    RETURN     all  --  anywhere             anywhere
```

`docker-compose.yml`:

```
networks:
  no_internet:
    driver: bridge
    ipam:
      config:
        - subnet: 10.0.1.0/24

services:
  some-service:
    image: some-image
    networks:
      - no_internet
    ports:
      - '123:123'
```

Share Improve this answer Follow

edited Mar 24, 2023 at 15:40

answered Mar 24, 2023 at 15:34

mastrchee

31 ● 2

gotcha: iptables does not survive reboots – fiat Oct 17, 2023 at 22:23