Docker And Docker Compose

- How to use certificate in Docker Compose file
- QNAP Network / Docker Network
- Docker Compose With Frontend and Backend Network

How to use certificate in Docker Compose file

Once you have obtained your Cloudflare Zero Trust certificate and private key, and assuming that you have a Docker Compose file for your application, you can use them in your Compose file by adding a volume to mount the certificate and private key files directly to the container or service that needs them.

Here is an example of how you can add a volume to mount the certificate and private key files for a container running an Nginx service:

```
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./certs:/etc/nginx/certs:ro
    restart: always

volumes:
  certs:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /path/to/certs
```

In the above example, we have added a new volume for `certs` and mounted it to the `/etc/nginx/certs` directory in the Nginx container. We also specified that the directory is read-only (`:ro`).

Make sure to replace `/path/to/certs` with the actual path to the directory where you have saved your Cloudflare Zero Trust certificate and private key.

You can then reference the certificate and private key files in your `nginx.conf` file, like so:

```
server {
  listen 443 ssl;
  ssl_certificate /etc/nginx/certs/example.com.crt;
  ssl_certificate_key /etc/nginx/certs/example.com.key;
  # rest of your Nginx configuration
}
```

This assumes that you have named your certificate and private key files as `example.com.crt` and `example.com.key`, respectively.

With this setup, your Nginx service will be using your Cloudflare Zero Trust certificate for SSL/TLS connections.

QNAP Network / Docker Network

Preberi na naslovu

Docker Network command

```
docker network create --help

Usage:  docker network create [OPTIONS] NETWORK

Create a network

Options:
      --attachable           Enable manual container attachment
      --aux-address map      Auxiliary IPv4 or IPv6 addresses used by Network driver (default
map[])
      --config-from string   The network from which to copy the configuration
      --config-only          Create a configuration only network
  -d, --driver string        Driver to manage the Network (default "bridge")
      --gateway strings      IPv4 or IPv6 Gateway for the master subnet
      --ingress              Create swarm routing-mesh network
      --internal             Restrict external access to the network
      --ip-range strings     Allocate container ip from a sub-range
      --ipam-driver string   IP Address Management Driver (default "default")
      --ipam-opt map         Set IPAM driver specific options (default map[])
      --ipv6                 Enable IPv6 networking
      --label list           Set metadata on a network
  -o, --opt map              Set driver specific options (default map[])
      --scope string         Control the networks scope
      --subnet strings       Subnet in CIDR format that represents a network segment
```

Docker Command

Without UI, you can also use Docker command to create a network that belongs to `Qnet driver` and run a container with `--net` argument.

DHCP mode

Create a new network named `qnet-dhcp-eth0`

```
$ docker network create \
  -d qnet \
  --opt=iface=eth0 \
  --ipam-driver=qnet \
  --ipam-opt=iface=eth0 \
  qnet-dhcp-eth0

# you get id
9e3b67877569d6da0f5587c736c4981b3206c7f31bd22e7acdfe1a347e41122c
```

Run container with Qnet DHCP mode

Run a container and connect it to `qnet-dhcp-eth0` network.

```
$ docker run --rm -it --net=qnet-dhcp-eth0 alpine ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 02:42:B8:3D:2B:07
          inet addr:192.168.80.118  Bcast:0.0.0.0  Mask:255.255.254.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1 errors:0 dropped:1 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:60 (60.0 B)  TX bytes:0 (0.0 B)
```

Static mode

Create a new network named qnet-static-eth0.

```
docker network create
  -d qnet \
  --opt=iface=eth0 \
  --ipam-driver=qnet \
  --ipam-opt=iface=eth0 \
  --subnet=192.168.80.0/23 \
  --gateway=192.168.80.254 \
  qnet-static-eth0

# you get id
85fbe06a66d82ba8109d304e1b891598d7c21e9f6c9a99a34f586250c7d8b92d
```

Run container with Qnet Static mode

Run a container and connect it to `qnet-static-eth0 network`.

```
$ docker run --rm -it --net=qnet-static-eth0 --ip=192.168.80.119 alpine ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 02:42:55:B1:84:92
          inet addr:192.168.80.119  Bcast:0.0.0.0  Mask:255.255.254.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:9 errors:0 dropped:4 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:940 (940.0 B)  TX bytes:0 (0.0 B)
```

Docker Compose

There is another way to create a container with Qnet driver.

Write **docker-compose.yml**

Specify `qnet` driver for this network.

docker-compose.yml

```
version: '2'

services:
```

```
  qnet_dhcp:
    image: alpine
    command: ifconfig eth0
    networks:
      - qnet-dhcp

  qnet_static:
    image: alpine
    command: ifconfig eth0
    networks:
      qnet-static:
        ipv4_address: 192.168.80.119

networks:
  qnet-dhcp:
    driver: qnet
    driver_opts:
      iface: "eth0"
    ipam:
      driver: qnet
      options:
        iface: "eth0"

  qnet-static:
    driver: qnet
    driver_opts:
      iface: "eth0"
    ipam:
      driver: qnet
      options:
        iface: "eth0"
      config:
        - subnet: 192.168.80.0/23
          gateway: 192.168.80.254
```

Docker Compose With Frontend and Backend Network

Problem :

Application is conected to frontend and access database on backend. Database is not accessible to outside world. Appliaction should have to have static ip.

Solution:

Define two separate networks in your Docker Compose file, one for the frontend and one for the backend. You can also assign static IP addresses to your containers using the ipv4_address attribute.

Example:

Here is an example docker-compose.yml file that demonstrates this setup:

```
version: '3'

networks:
  frontend:
    ipam:
      driver: default
      config:
        - subnet: 172.16.1.0/24
  backend:
    ipam:
      driver: default
      config:
        - subnet: 172.16.2.0/24
```

```
services:
  web:
    image: my-app
    networks:
      frontend:
        ipv4_address: 172.16.1.10
      backend:
        ipv4_address: 172.16.2.10
    environment:
      DATABASE_URL: postgres://dbuser:dbpass@database/dbname
    depends_on:
      - database

  database:
    image: postgres
    networks:
      backend:
        ipv4_address: 172.16.2.20
    environment:
      POSTGRES_USER: dbuser
      POSTGRES_PASSWORD: dbpass
      POSTGRES_DB: dbname
```

Explanation:

In the above example, we define two networks, frontend and backend, each with its own subnet. We also define two services, web and database.

The web service is connected to both networks, frontend and backend, with static IP addresses of 172.16.1.10 and 172.16.2.10, respectively. It also has an environment variable DATABASE_URL that references the hostname of the database service.

The database service is connected only to the backend network, with a static IP address of 172.16.2.20.

With this setup, the web service can access the database service using the DATABASE_URL environment variable, and the database service is only accessible within the backend network and is not exposed to the public internet.