

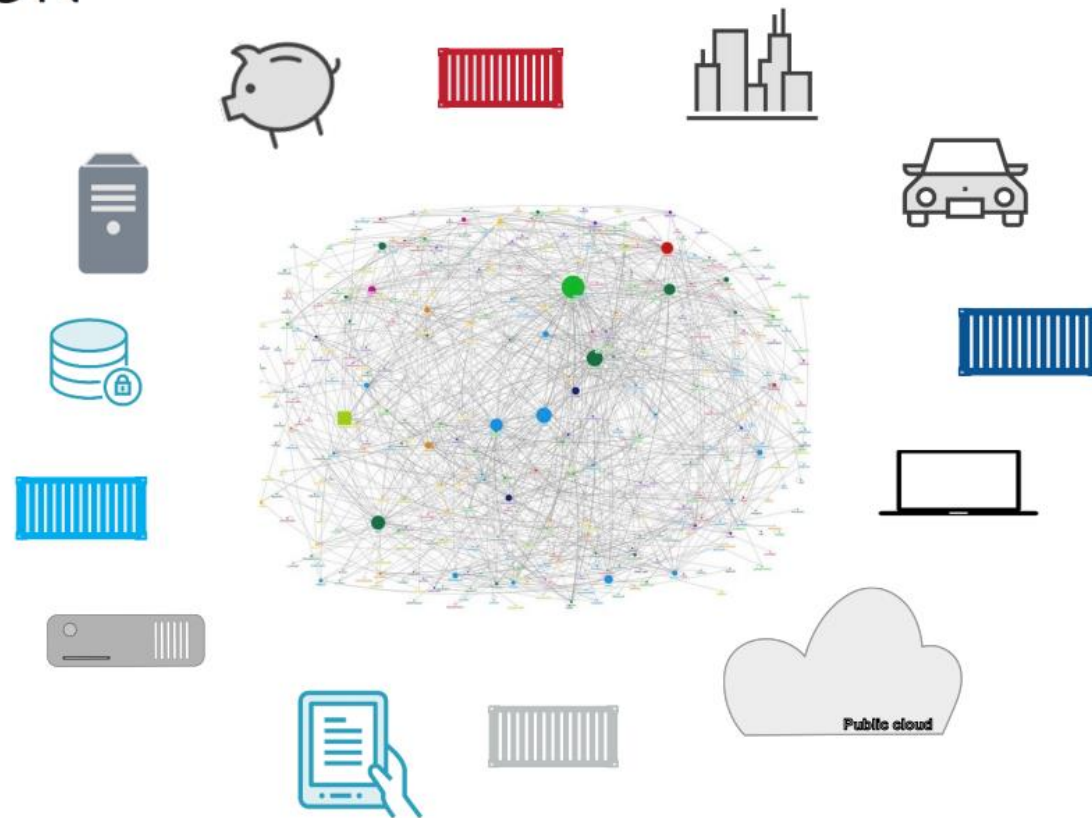
NETWORKING

COURSE INTRODUCTION

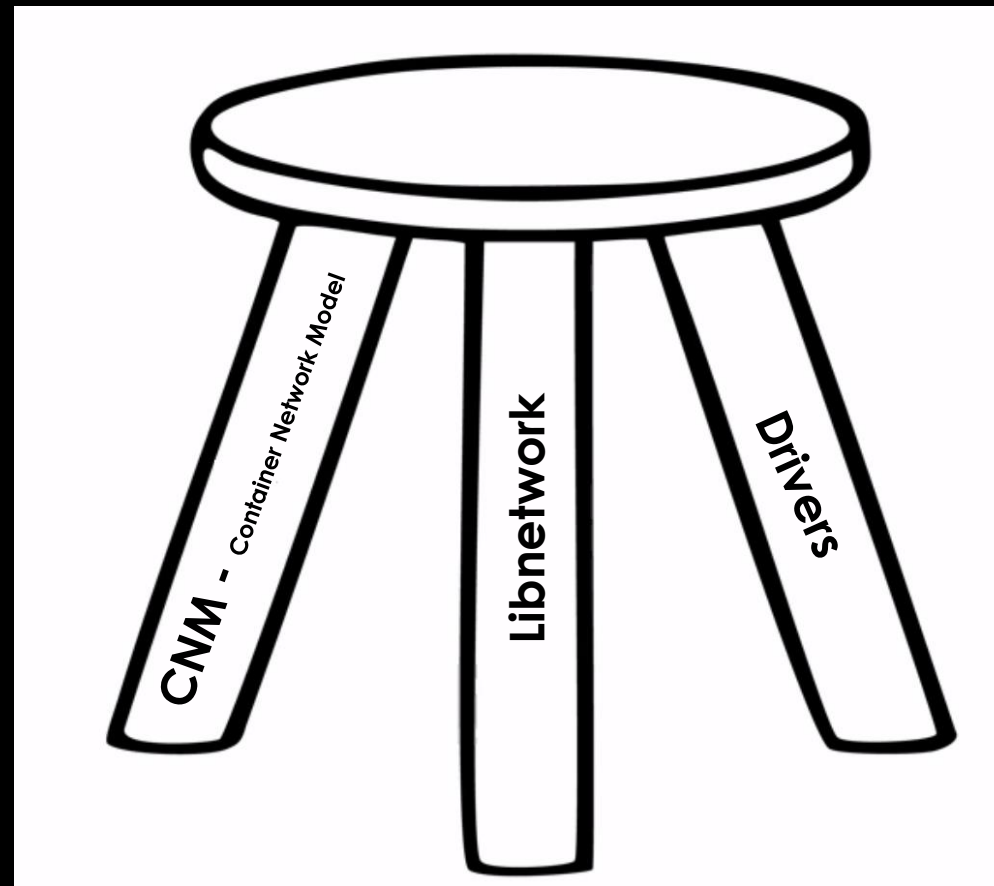
Networks are complex

Networks are huge

Networks are central



THE THREE PILLARS OF DOCKER NETWORKING



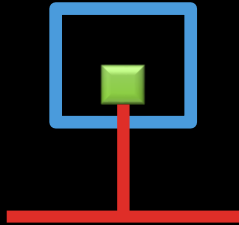
NETWORK

Master plan / Design

CNM

DNA:

- Sandbox
- Endpoint
- Network



Control plane

Libnetwork

Central place for
all docker networking
logic ,API ,UX etc...
Real-world implantation of CNM

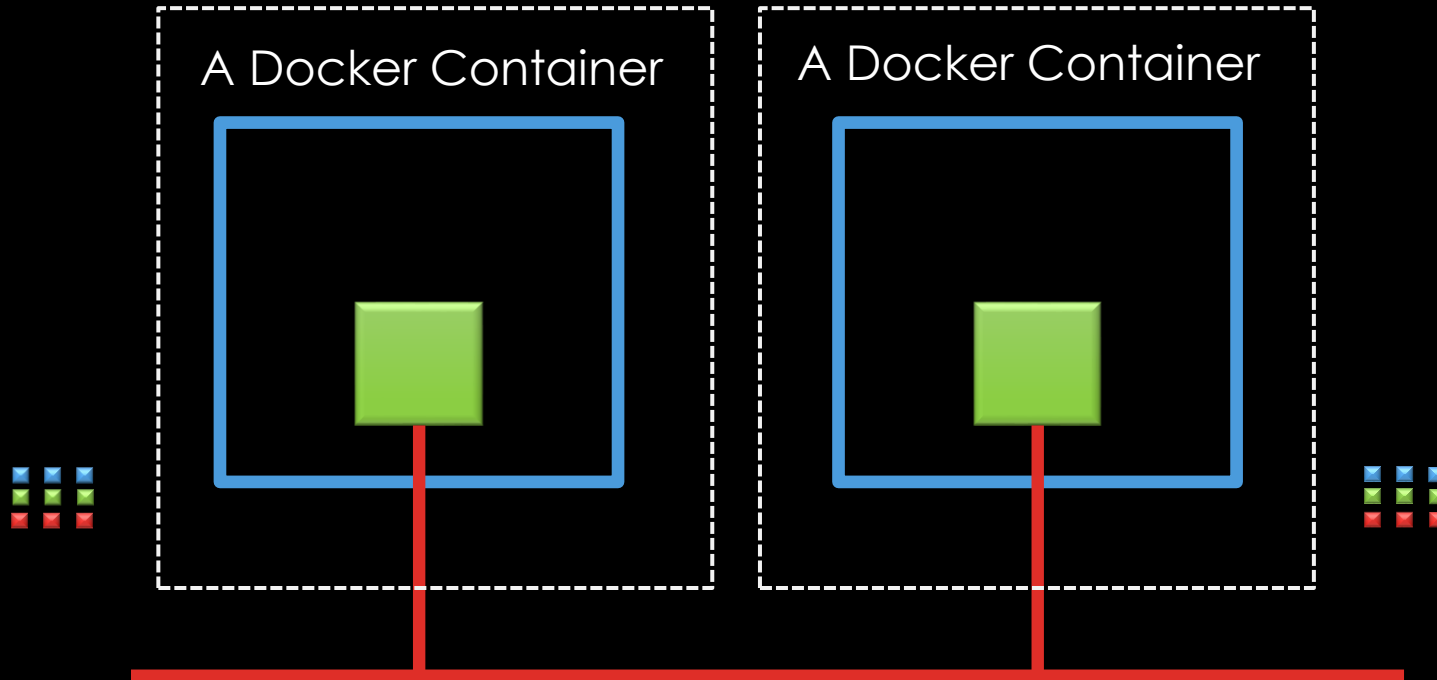
Data plane

Drivers

Network-specific detail

- Overlay
- MACVLAN
- IPVLAN
- Bridge

NETWORK - CNM



Sandbox Network

Isolated area of OS
Containers full network stack

Endpoint

Network interface like eth0

Network

Connected Endpoint

NETWORK – BASIC CLI

```
C:\Users\Administrator>docker network
```

```
Usage:  docker network COMMAND
```

```
Manage networks
```

```
Commands:
```

→ connect	Connect a container to a network
→ create	Create a network
→ disconnect	Disconnect a container from a network
→ inspect	Display detailed information on one or more networks
→ ls	List networks
→ prune	Remove all unused networks
→ rm	Remove one or more networks

```
Run 'docker network COMMAND --help' for more information on a command.
```

NETWORKING: SINGLE – HOST CLI

- Create bridge

```
docker network create -d bridge --subnet 10.0.0.1/24 ps-bridge
```

- More information about bridge

```
docker inspect ps-bridge
```

CONNECT BETWEEN CONTAINER

- Create Container name C1 with ps-bridge

```
docker run -dt --name c1 --network ps-bridge alpine sleep 1d
```

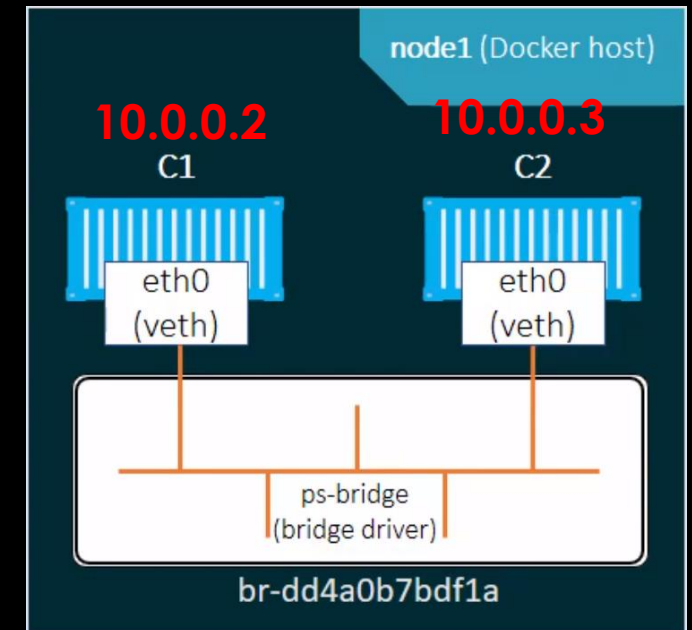
- Create Container name C2 with ps-bridge

```
docker run -dt --name c2 --network ps-bridge alpine sleep 1d
```

inspect network

```
docker inspect ps-bridge
```

```
"ConfigOnly": false,
"Containers": {
  "5c9baab6fbc83aa47578c86a1cf987d6f9ecb18e0cf5a1": {
    "Name": "c2",
    "EndpointID": "cb693e055e3f41b22b77df8adcfe",
    "MacAddress": "02:42:0a:00:00:03",
    "IPv4Address": "10.0.0.3/24",
    "IPv6Address": ""
  },
  "846fe69e1bfd71065250c6df44322fec9f246fb1139c9a": {
    "Name": "c1",
    "EndpointID": "40e0041b6d8913fe1fb67a6e48a",
    "MacAddress": "02:42:0a:00:00:02",
    "IPv4Address": "10.0.0.2/24",
    "IPv6Address": ""
  }
}
```



CONNECT BETWEEN CONTAINER

- Check container C1 and C2 with ps-bridge

```
docker exec -it c1 sh
```

- Check ip

```
/# ip a
```

```
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN
    link/ipip 0.0.0.0 brd 0.0.0.0
3: ip6tnl0@NONE: <NOARP> mtu 1452 qdisc noop state DOWN
    link/tunnel6 00:00:00:00:00:00:00:00:00:00:00:00
    link/ether 02:42:0a:00:00:02 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.2/24 brd 10.0.0.255 scope global eth0
        valid_lft forever preferred_lft forever
```

- Check connecting to C2

```
/# ping 10.0.0.3
```

By Name

```
/# ping C2
```


CONNECT BETWEEN CONTAINER MORE THEN ONE NETWORK

Create networks bridge1 and bridge2:

```
docker network create -d bridge --subnet 11.1.0.1/24 bridge1
```

```
docker network create -d bridge --subnet 11.2.0.1/24 bridge2
```

- Create Container name C1 with bridge1

```
docker run --rm -dt --name C1 --network bridge1 alpine sleep 1d
```

- Create Container name C2 with bridge1

```
docker run --rm -dt --name C2 --network bridge1 alpine sleep 1d
```

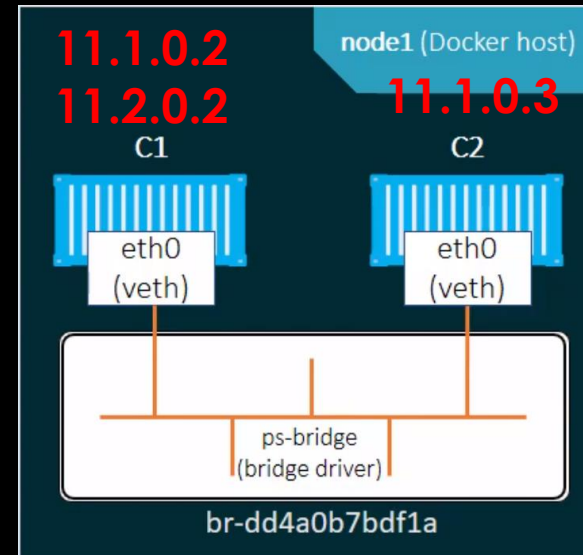
- Connect bridge2 to Container C1

```
docker network connect bridge2 C1
```

- inspect network

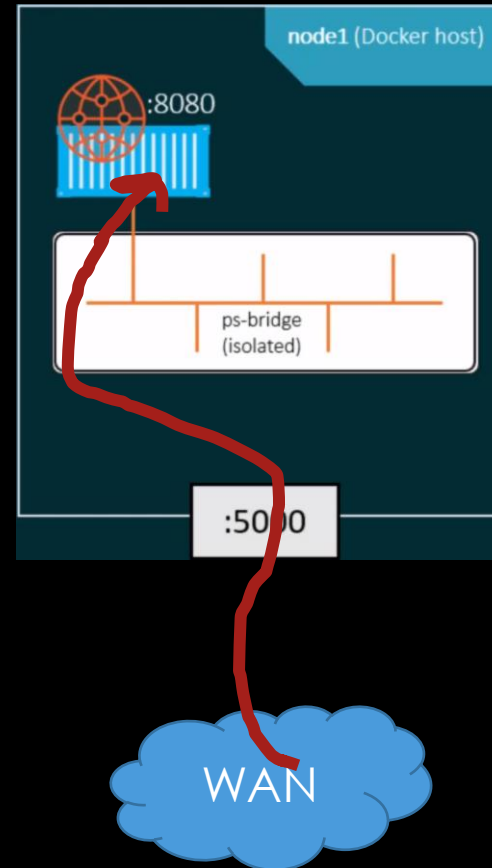
```
docker inspect bridge2
```

```
"networks": {  
  "bridge1": {  
    "IPAMConfig": null,  
    "Links": null,  
    "Aliases": [ "8ee675d81482"  
    ],  
    "NetworkID": "858f726849449",  
    "EndpointID": "74f8486567e7",  
    "Gateway": "11.1.0.1",  
    "IPAddress": "11.1.0.2",  
    "IPPrefixLen": 24,  
    "IPv6Gateway": "",  
    "GlobalIPv6Address": "",  
    "GlobalIPv6PrefixLen": 0,  
    "MacAddress": "02:42:0b:01:",  
    "DriverOpts": null  
  },  
  "bridge2": {  
    "IPAMConfig": {},  
    "Links": null,  
    "Aliases": [ "8ee675d81482"  
    ],  
    "NetworkID": "704025b5c18f4",  
    "EndpointID": "53ea642c8938",  
    "Gateway": "11.2.0.1",  
    "IPAddress": "11.2.0.2",  
    "IPPrefixLen": 24,  
    "IPv6Gateway": "",  
    "GlobalIPv6Address": "",  
    "GlobalIPv6PrefixLen": 0,  
    "MacAddress": "02:42:0b:01:",  
    "DriverOpts": null  
  }  
}
```



EXPOSE PORT

(host) (container)
-p 5000:8080



תרגיל 4

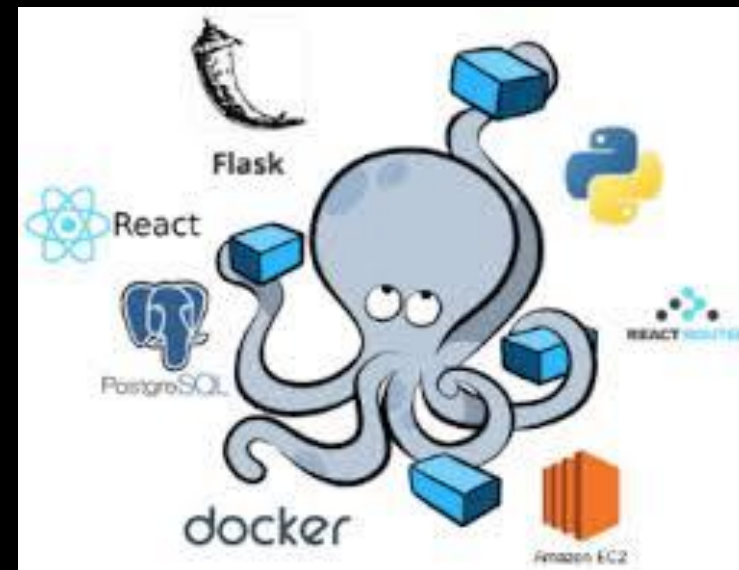
1. ייצר Bridge בשם first-Bridge למרחב כתובות 10.0.0.1/24
2. תבדוק באמצעות פקודה inspect את המידע על ה Bridge
3. הרם container שירץ ברקע של alpine בשם C100 שמחובר לרשת first-Bridge
4. הרם container שירץ ברקע של alpine בשם C200 שמחובר לרשת first-Bridge
5. תבדוק שוב באמצעות פקודה inspect את המידע על ה Bridge
 1. בדוק ש C100 ו C200 משויכים אליו, תרשום את הכתובות IP שקיבלו בצד.
6. כנס באמצעות bash ל C100
7. בדוק איזה כתובת קיבלת
8. בצע Ping ל C200 באמצעות כתובת IP
9. בצע Ping ל C200 באמצעות שם

תרגיל 5

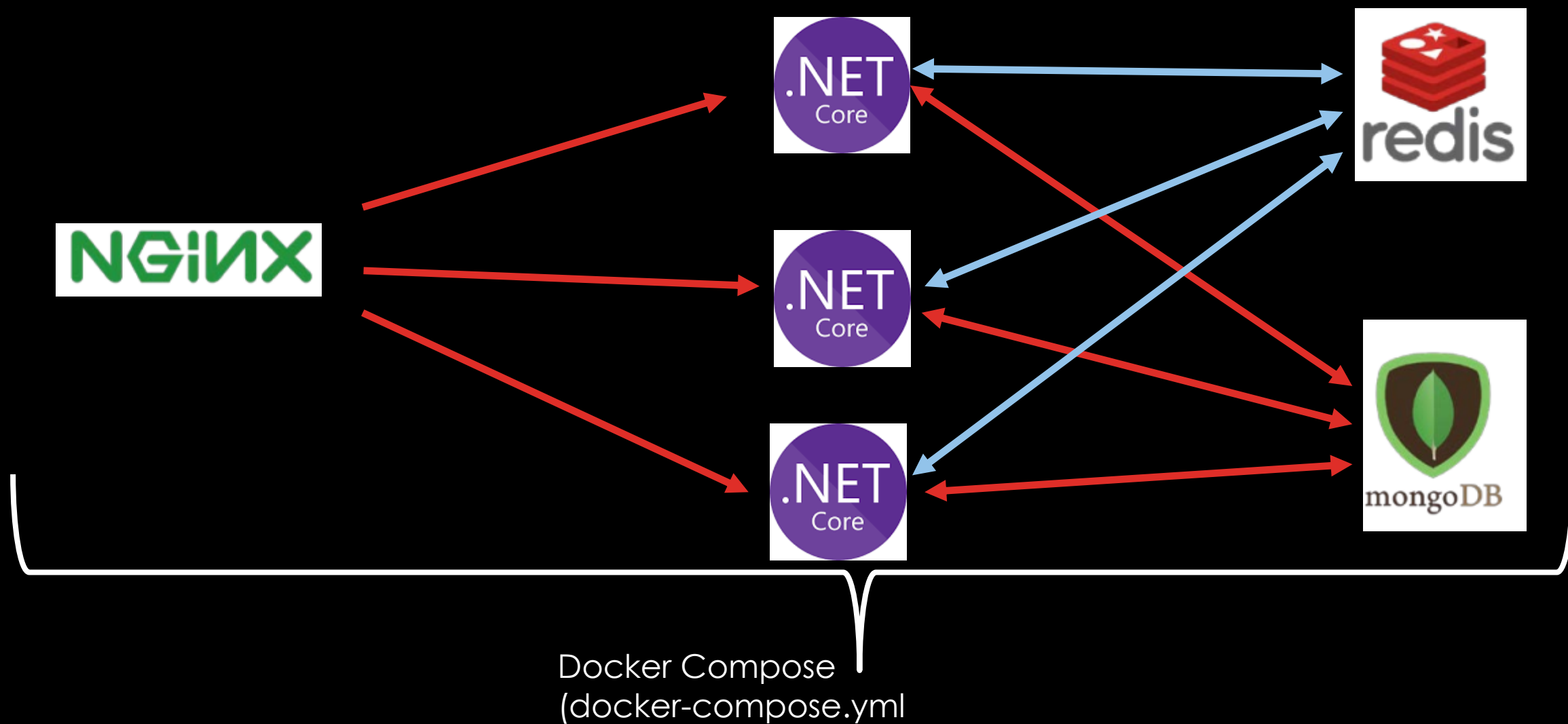
<https://github.com/shaloml/docker-network-sample/blob/master/README.md>

DOCKER COMPOSE

- Start, stop , rebuild of our services
- View status of our running services
- Stream the log output of running services
- Run a one-off command on a service



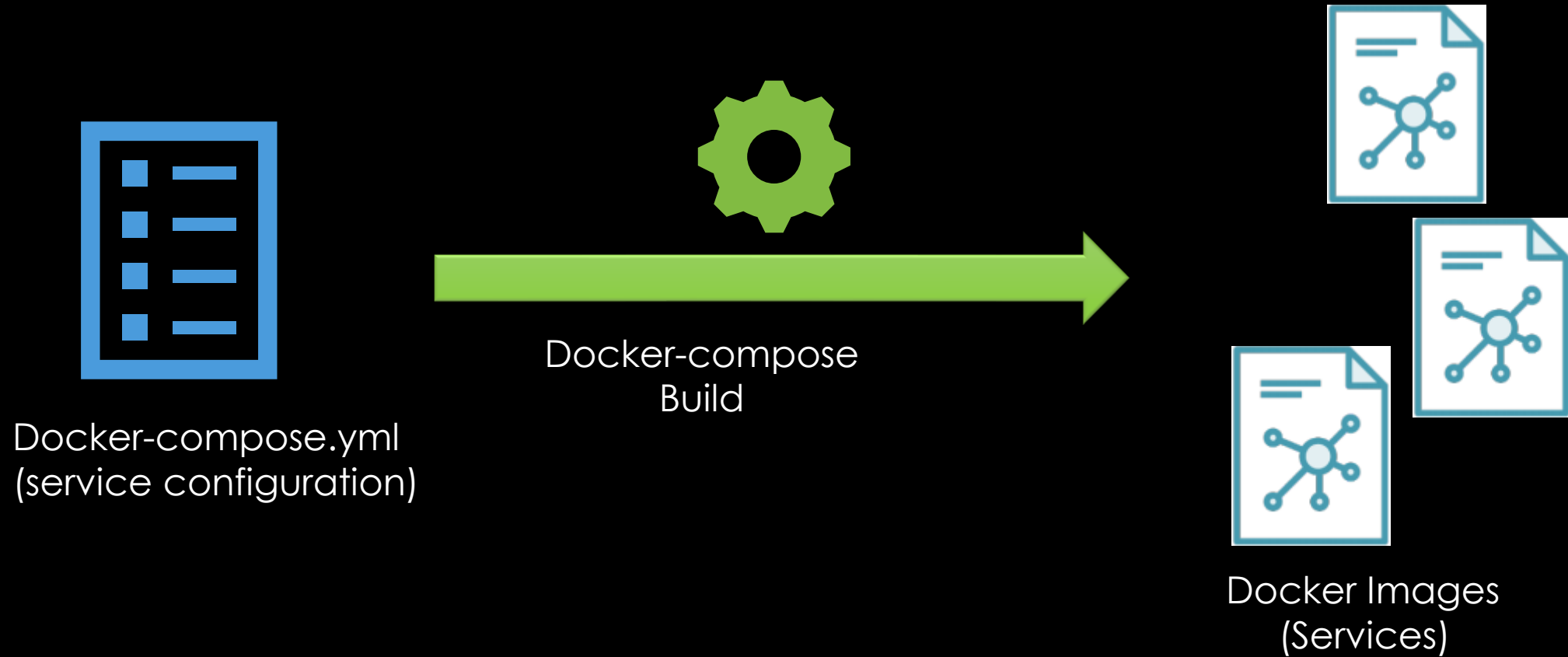
THE NEED FOR DOCKER COMPOSE



DOCKER COMPOSE WORKFLOW



THE DOCKER-COMPOSE.YML FILE



THE DOCKER-COMPOSE.YML FILE

```
version: '2'
```

```
services:
```



mongoDB

```
docker-compose.yml
```

KEY SERVICE CONFIGURATION OPTIONS

build

environment

image

networks

ports

volumes

DOCKER-COMPOSE.YML EXAMPLE

```
Version:'3.4'
services:
  node:
    build:
    context: .
    dockerfile: node.dockerfile
networks:
  -nodeapp-network

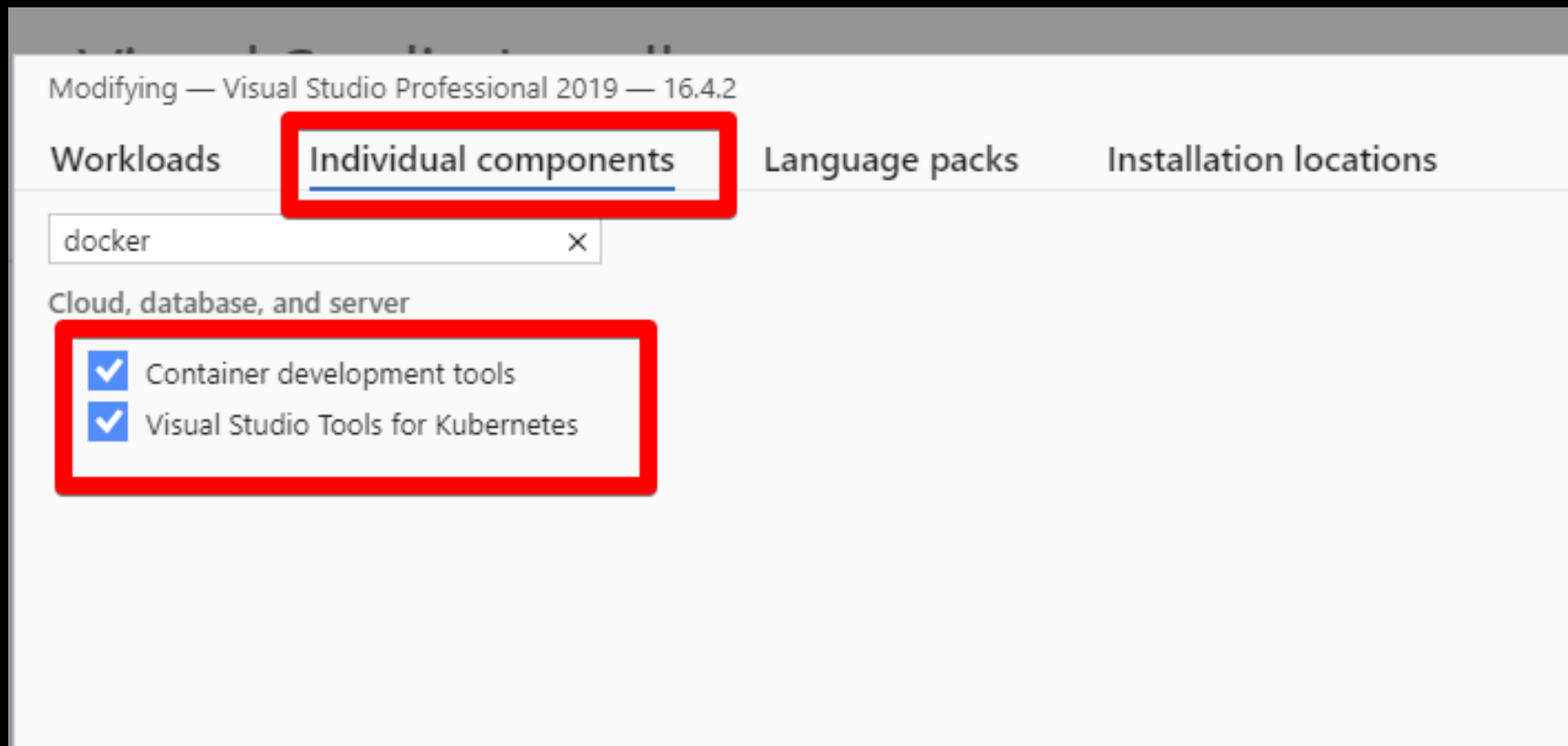
mongodb:
  image: mongo
  networks:
    - nodeapp-network
networks:
  nodeapp-network
  driver: bridge
```

```
version: '3' # Docker-compose yml version for docker compose builder
services: # Here we define our architecture services / roles
  nginx: # or web_role , api_role , db_mysql , db_redis ....
    build: # Dockerfile to build or use image:
      #to pull from DockerHub (public/private) or git
      context: . # folder where the dockerfile is located
      dockerfile: #[dockerfile-name]
    ports: # Ports we wish to expose source:target
      - 8080:8080
    volumes: # Which Bind mounts or Volumes to mount source:target
      - type: bind # Shortsyntax -> ./source:/code
        source: /source
        target: /code
      -type: volume # Shortsyntax ->logicalVolume01:/var/log
        source: logicalVoloume01
        target: /var/log
    network: # Define netwroks per service / role
      - db_layer
      - app_layer
  services: #.....
    image: redis
networks: # custome configuration per network defined
  db_layer:
  app_layer:

volumes:
  logicalVoloume01:
```

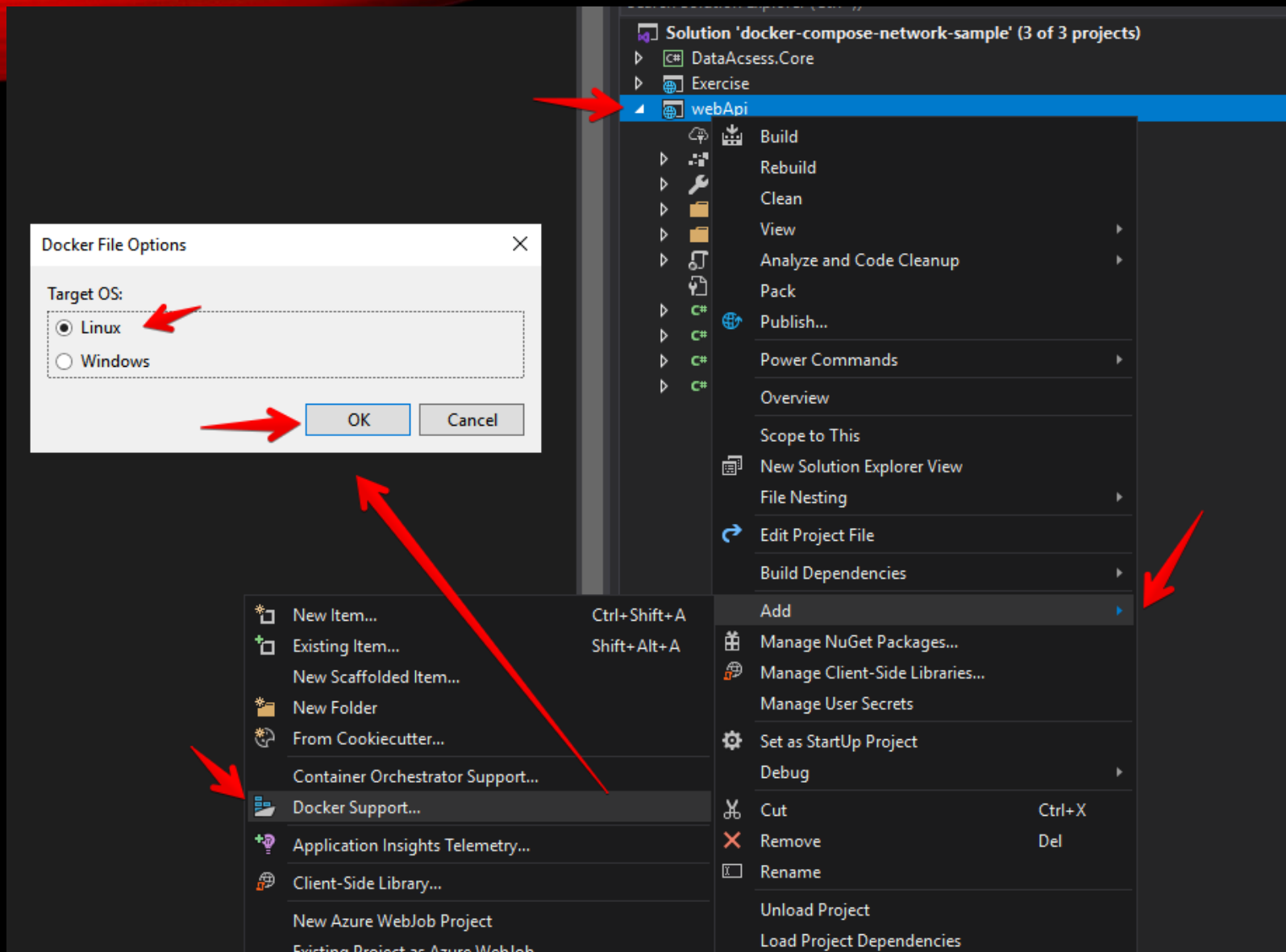
DOCKER TOOL - VISUAL STUDIO

Open Microsoft visual studio installer



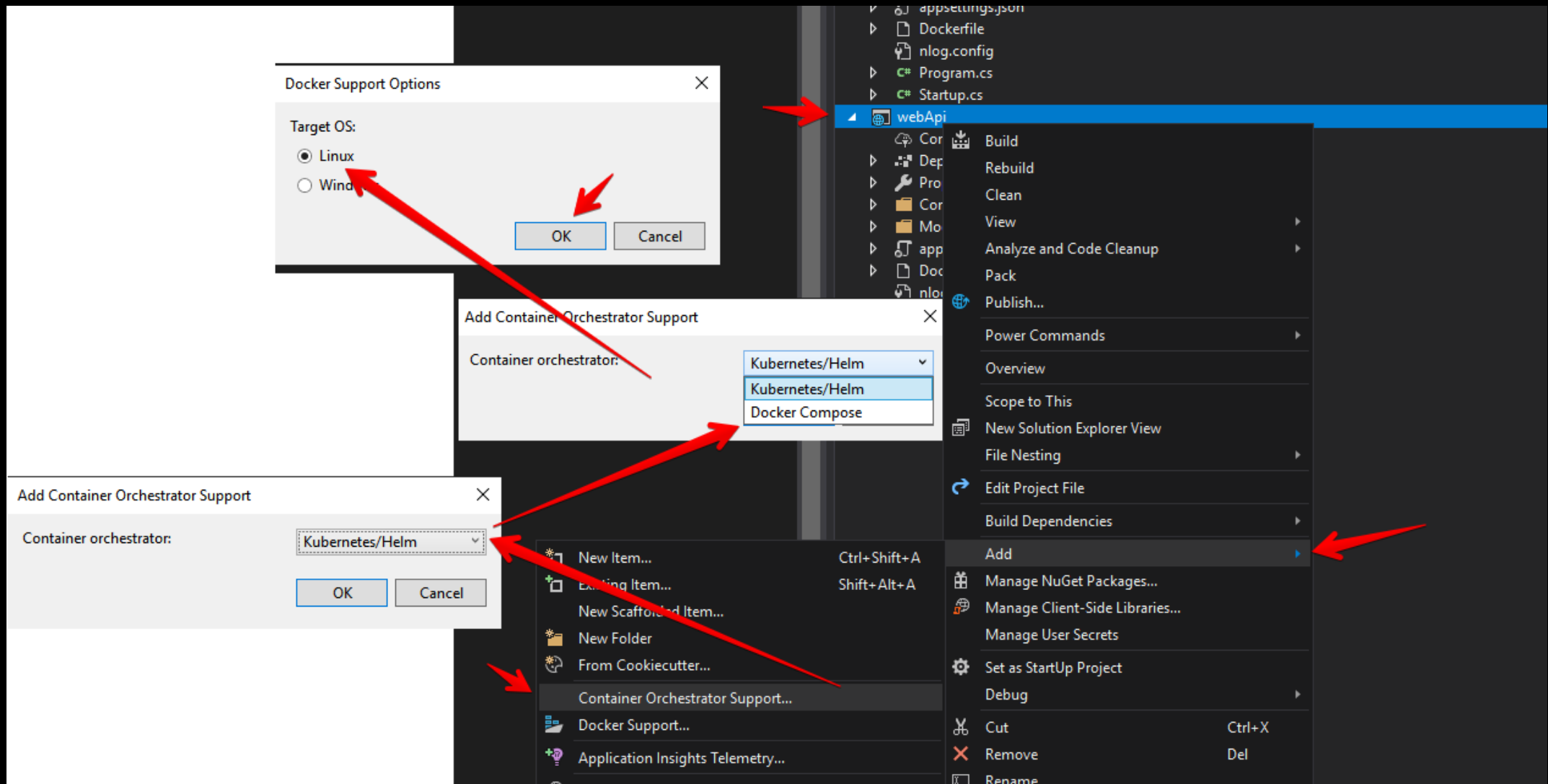
DOCKER ON VISUAL STUDIO

Add Dockerfile



DOCKER ON VISUAL STUDIO

Add docker-compose



DOCKER ON VISUAL STUDIO

management

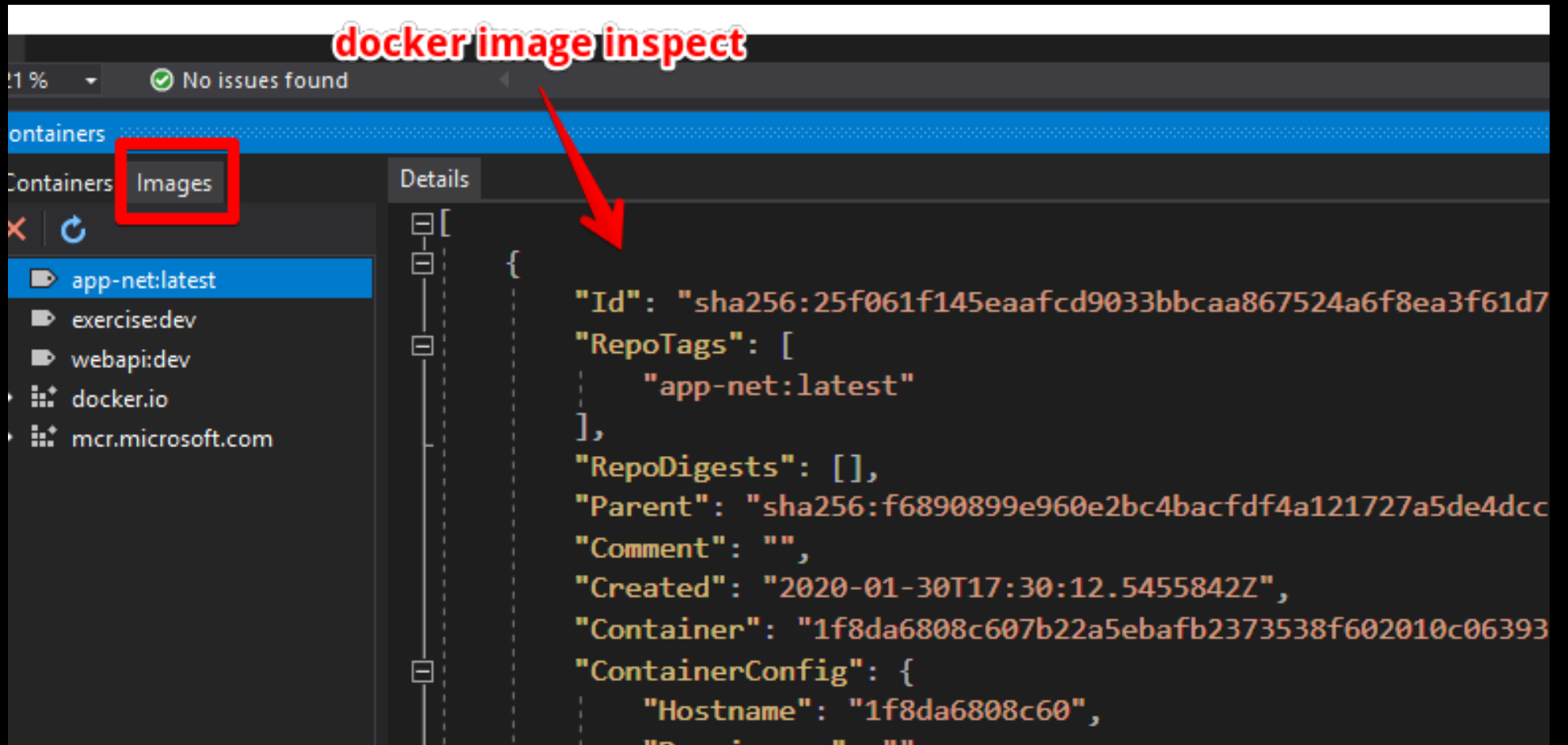
The screenshot shows the Visual Studio Docker extension interface. On the left, the 'Containers' pane lists 'Exercise' and 'webApi' under 'Solution Containers'. A red arrow labeled 'containers' points to this pane. Above the 'Containers' pane, a red arrow labeled 'exec' points to the 'Containers' tab. In the main area, the 'Environment' tab is selected and highlighted with a red box. A red arrow labeled 'information' points to this tab. The 'Environment' tab displays a table of environment variables.

Name	Value
ASPNETCORE_ENVIRONMENT	Development
ASPNETCORE_URLS	http://+:80
DOTNET_RUNNING_IN_CONTAINER	true
DOTNET_USE_POLLING_FILE_WATCHER	1
NUGET_FALLBACK_PACKAGES	/root/.nuget/fallbackpackages
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

At the bottom of the interface, the 'Containers' tab is active, showing 'Locals' and 'Watch 1'.

DOCKER ON VISUAL STUDIO

management

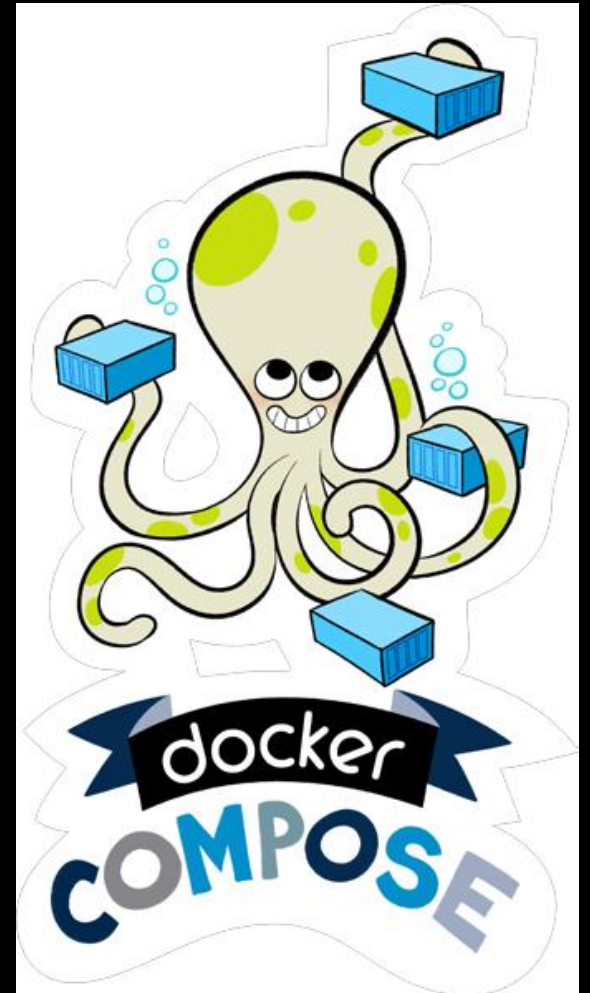


DEMO



KEY DOCKER COMPOSE COMMANDS

- `docker-compose build`
- `docker-compose up`
- `docker-compose down`
- `docker-compose logs`
- `docker-compose ps`
- `docker-compose stop`
- `docker-compose start`
- `docker-compose rm`



BUILDING SERVICES

`docker-compose build`



Build or rebuild services
defined in
`docker-compose.yml`

BUILDING SPECIFIC SERVICES

```
docker-compose build mongo
```



Only build/rebuild
mongo service

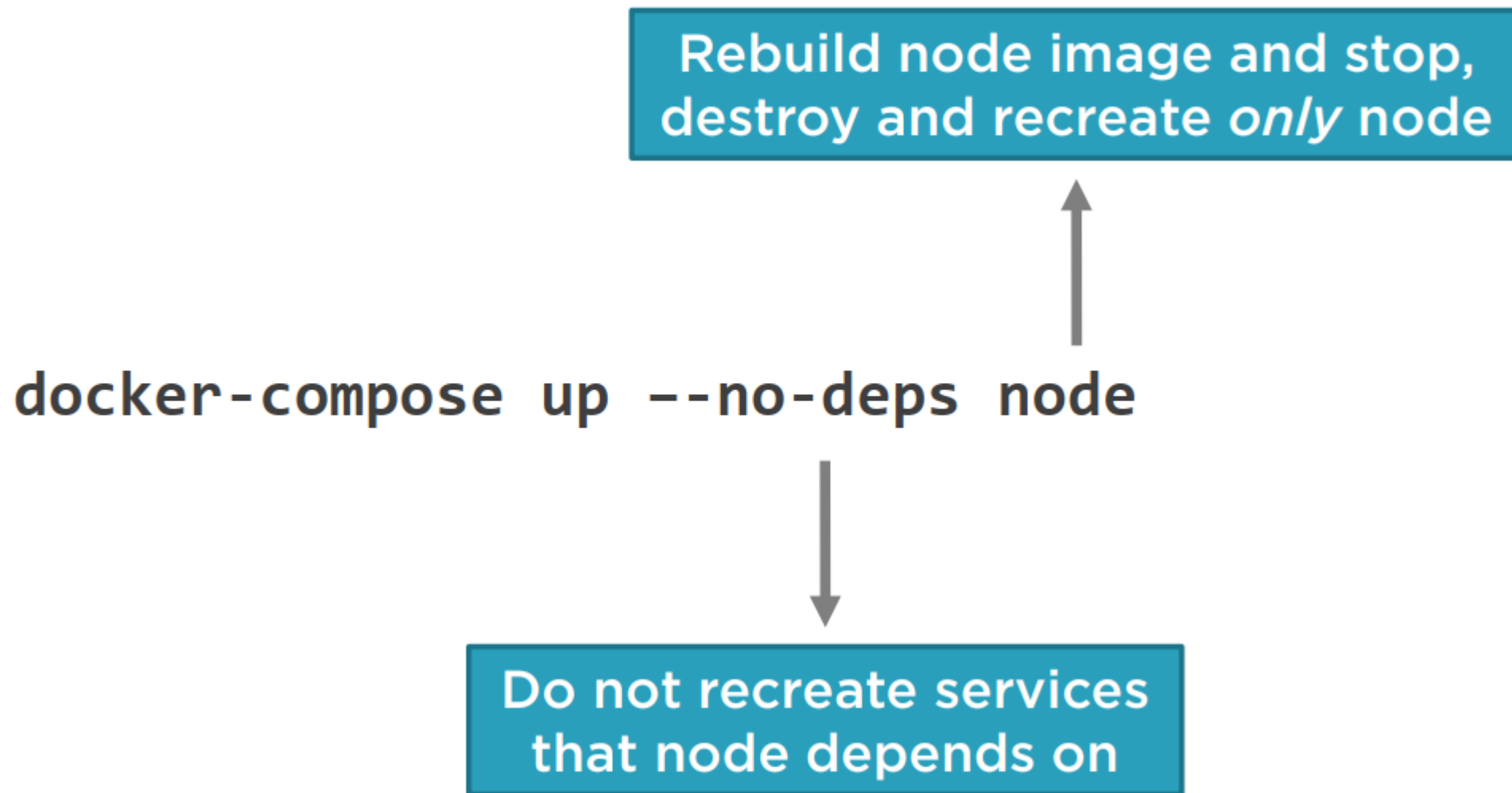
CREATING AND STARTING CONTAINERS

`docker-compose up`



Create and start the
containers

CREATING AND STARTING CONTAINERS



STOP AND REMOVE CONTAINERS

`docker-compose down`



Take all of the containers
down (stop and remove)

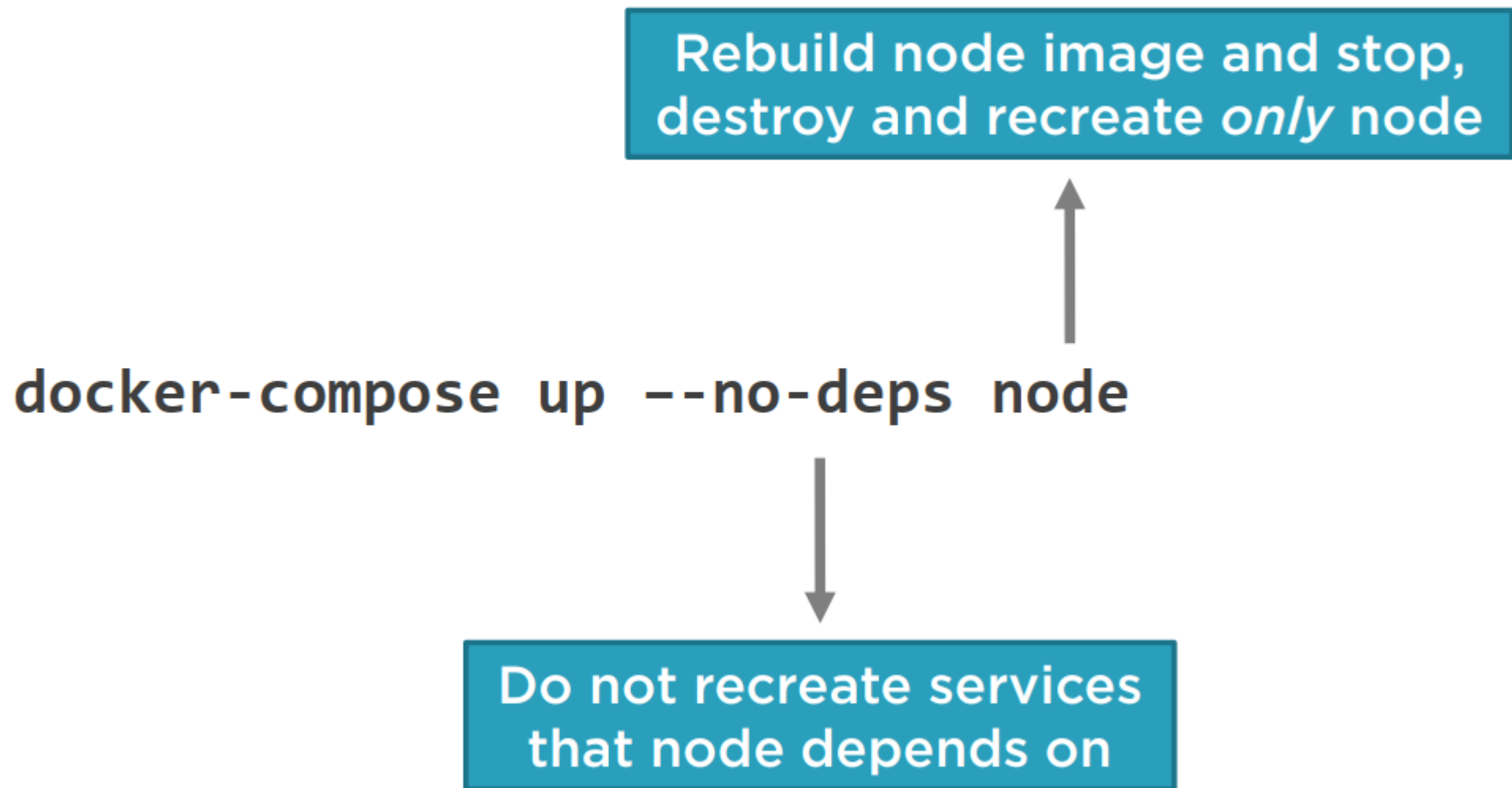
STOP AND REMOVE CONTAINERS, IMAGES, VOLUMES

```
docker-compose down --rmi all --volumes
```

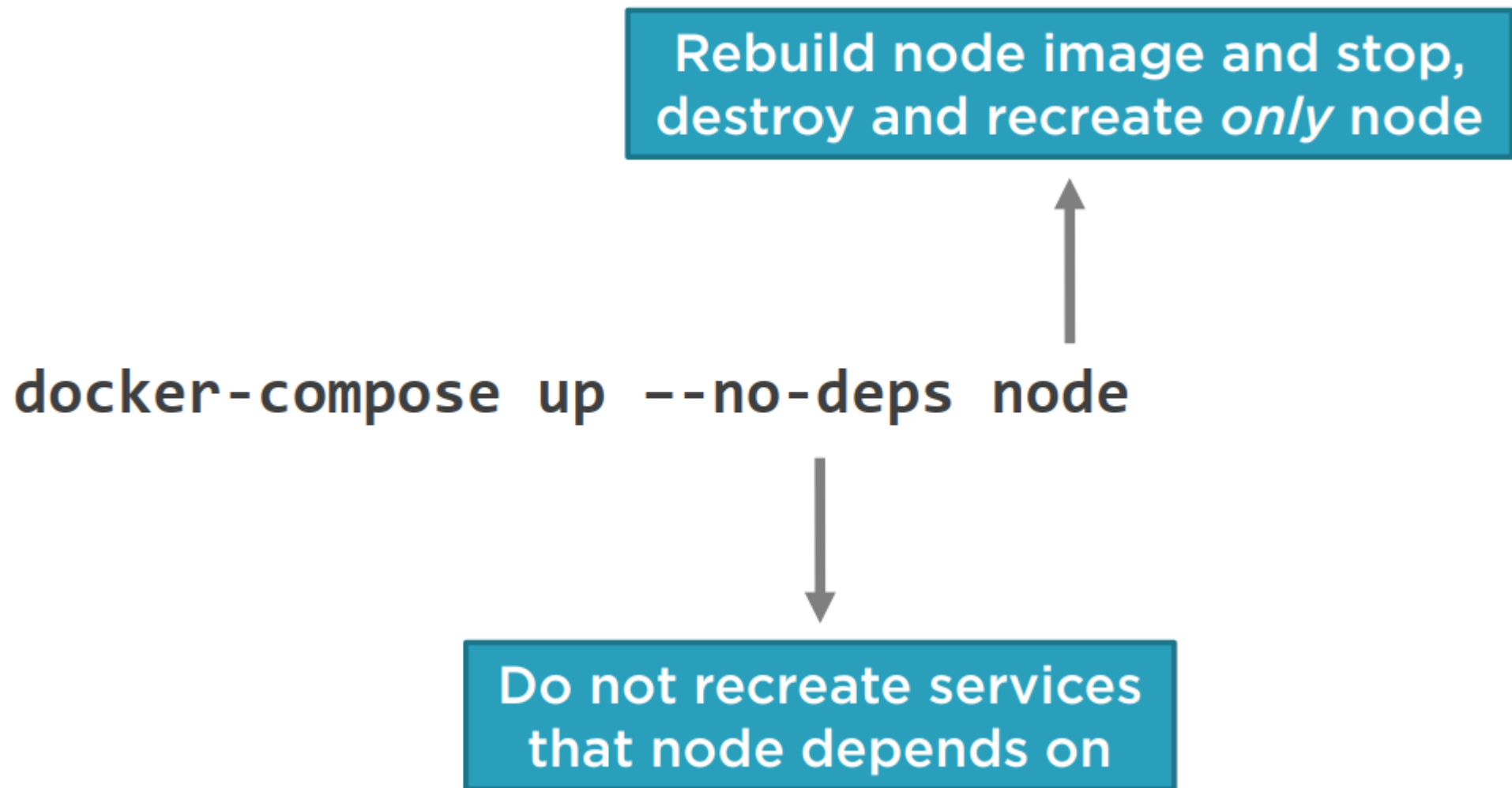
Remove all
volumes

Remove all
images

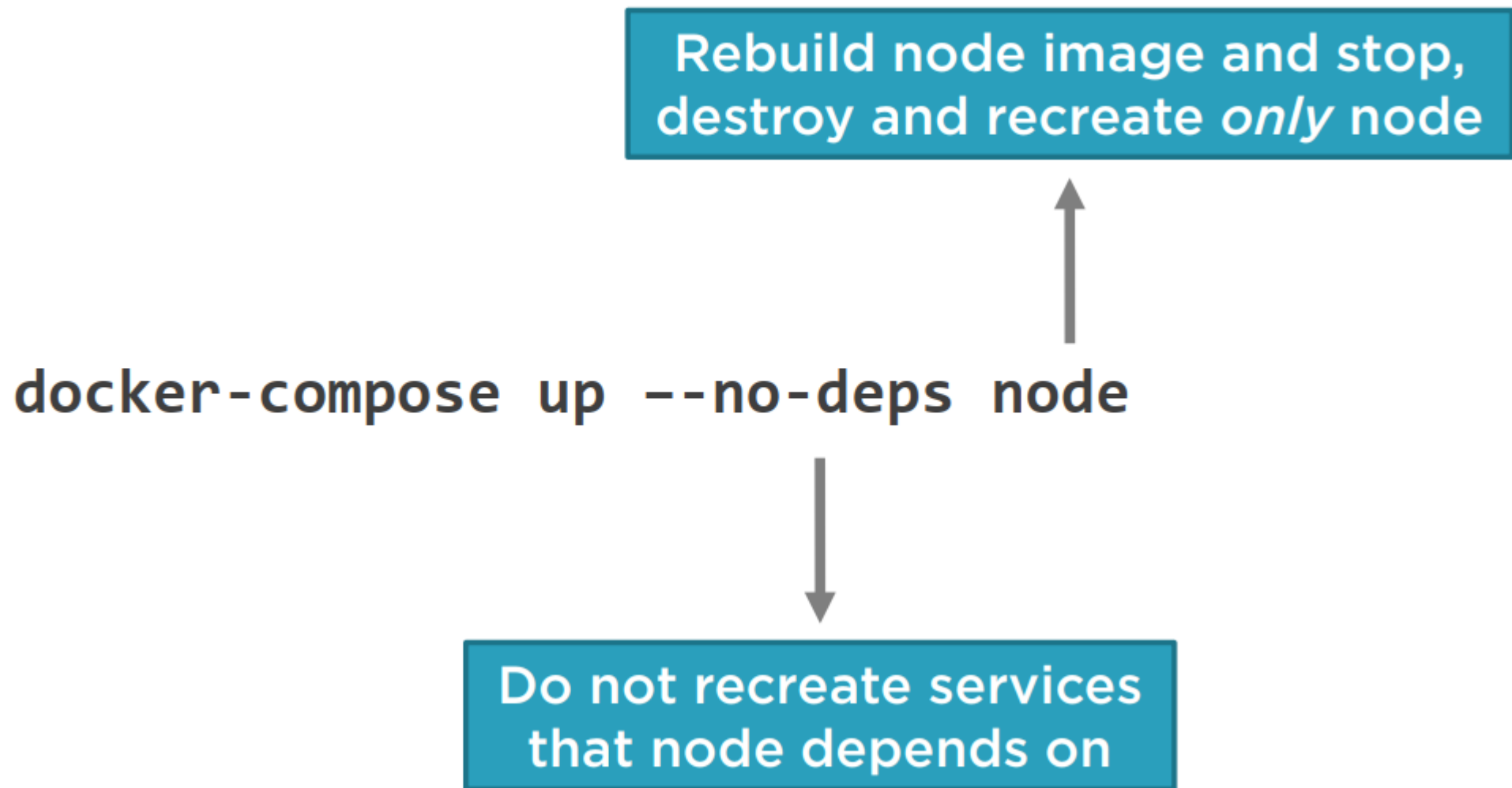
CREATING AND STARTING CONTAINERS



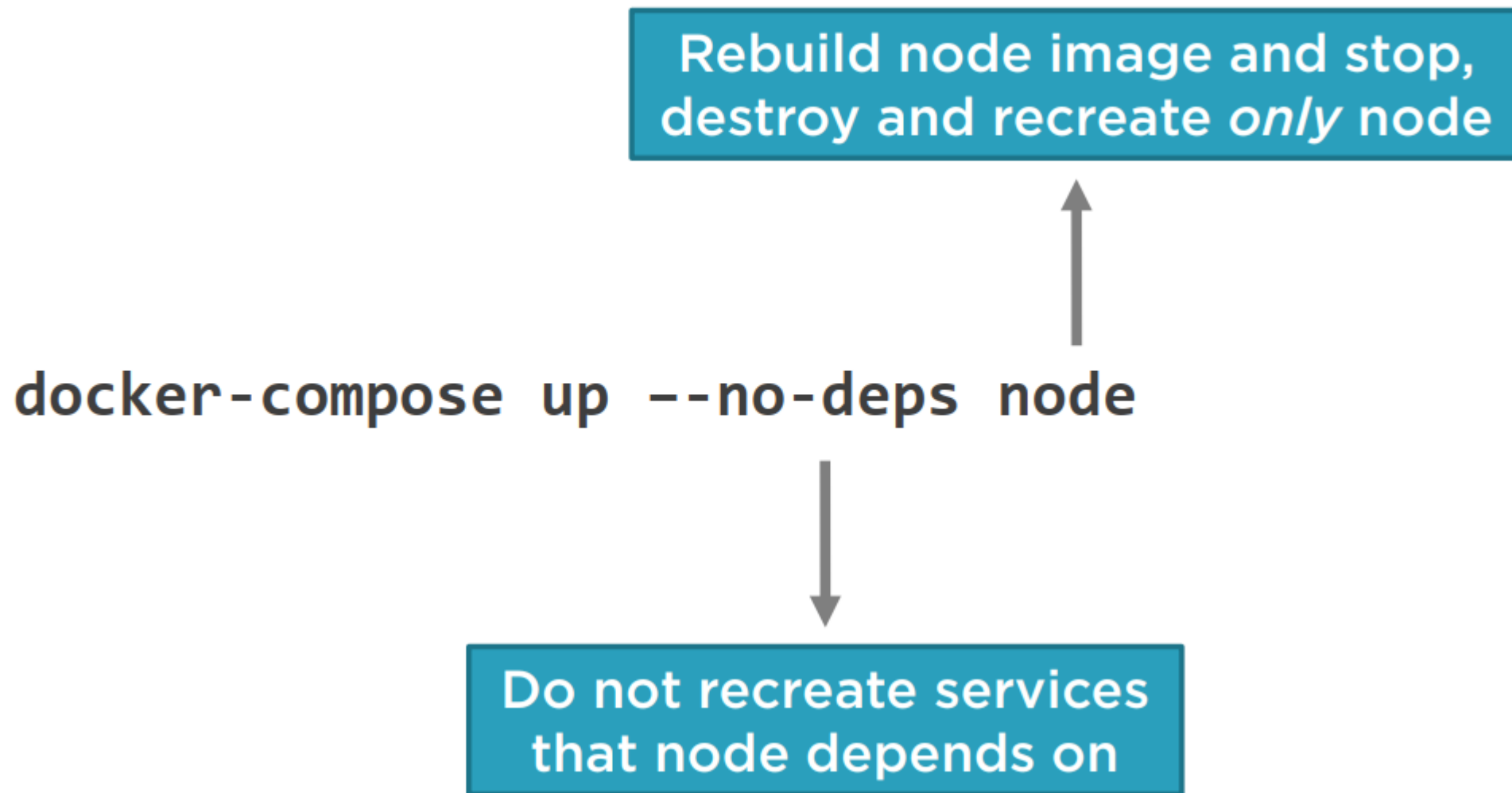
CREATING AND STARTING CONTAINERS



CREATING AND STARTING CONTAINERS



CREATING AND STARTING CONTAINERS



DOCKER-COMPOSE - TIPS

- How to keep your images small
- Store data on volumes
- **use bind mounts is during development**, when you may want to mount your source directory or a binary you just built into your container. For production, use a volume instead, mounting it into the same location as you mounted a bind mount during development.
-

DOCKER-COMPOSE - TIPS

- **Use name of container**

use the name of the service as your dns and you will be able to talk to other containers without the need to create a network. That's a better option to keep your docker-compose files as simple as possible.

- **Use a .env file to replace variables in your docker-compose.yml file**

- **Assign an ip to your container**

```
version: '3.7'
services:
  redis:
    image: redis:4.0.9-alpine
    init: true
    container_name: redis
    networks:
      dockernet:
        ipv4_address: 172.20.0.2
networks:
  dockernet:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/16
```

- **Use the build flag to rebuild your containers**

Sometimes you want to force a rebuild of your containers with docker-compose, do it like this:

```
docker-compose up -d --build
```

```
# Ignore the cache:
docker-compose build --no-cache
```

DOCKER-COMPOSE - TIPS

- **Use name of container**

use the name of the service as your dns and you will be able to talk to other containers without the need to create a network. That's a better option to keep your docker-compose files as simple as possible.

- **Use a .env file to replace variables in your docker-compose.yml file**

- **Assign an ip to your container**

```
version: '3.7'
services:
  redis:
    image: redis:4.0.9-alpine
    init: true
    container_name: redis
    networks:
      dockernet:
        ipv4_address: 172.20.0.2
networks:
  dockernet:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/16
```

- **Use the build flag to rebuild your containers**

Sometimes you want to force a rebuild of your containers with docker-compose, do it like this:

```
docker-compose up -d --build
```

```
# Ignore the cache:
docker-compose build --no-cache
```

תרגיל DOCKER-COMPOSE

תרגיל 5

Thanks

