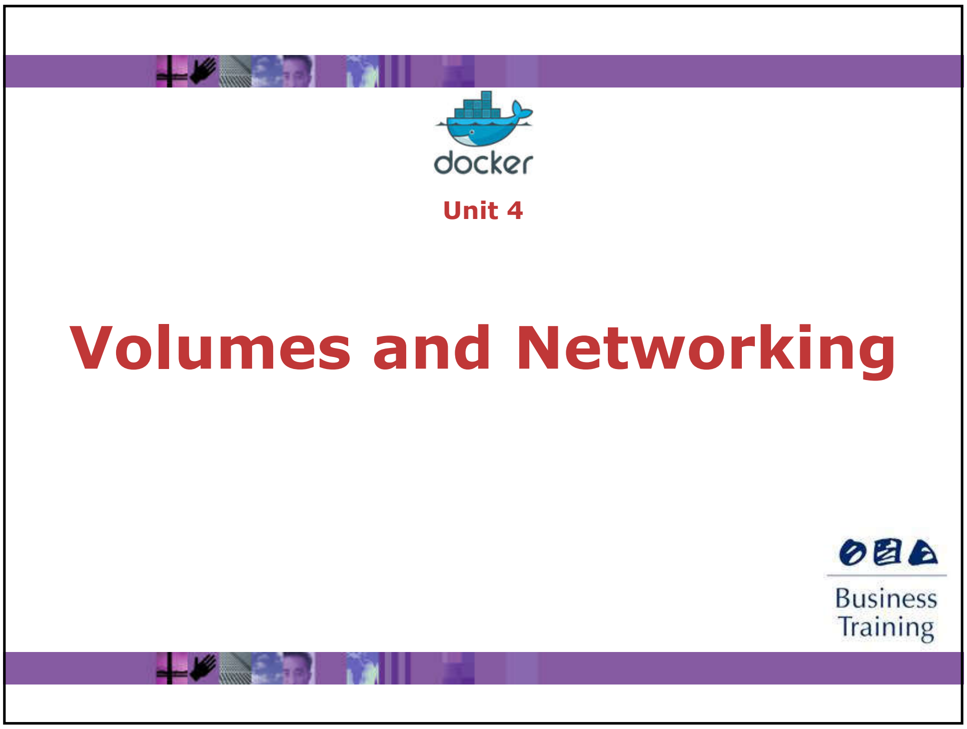


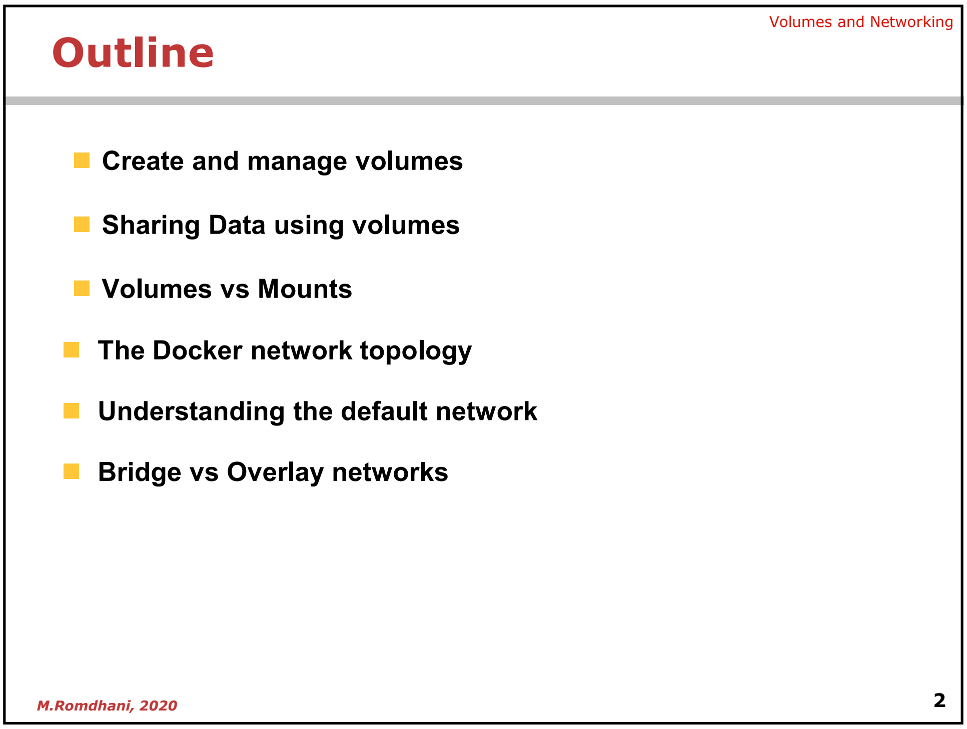
docker

Unit 4

Volumes and Networking

Business Training

1



Volumes and Networking

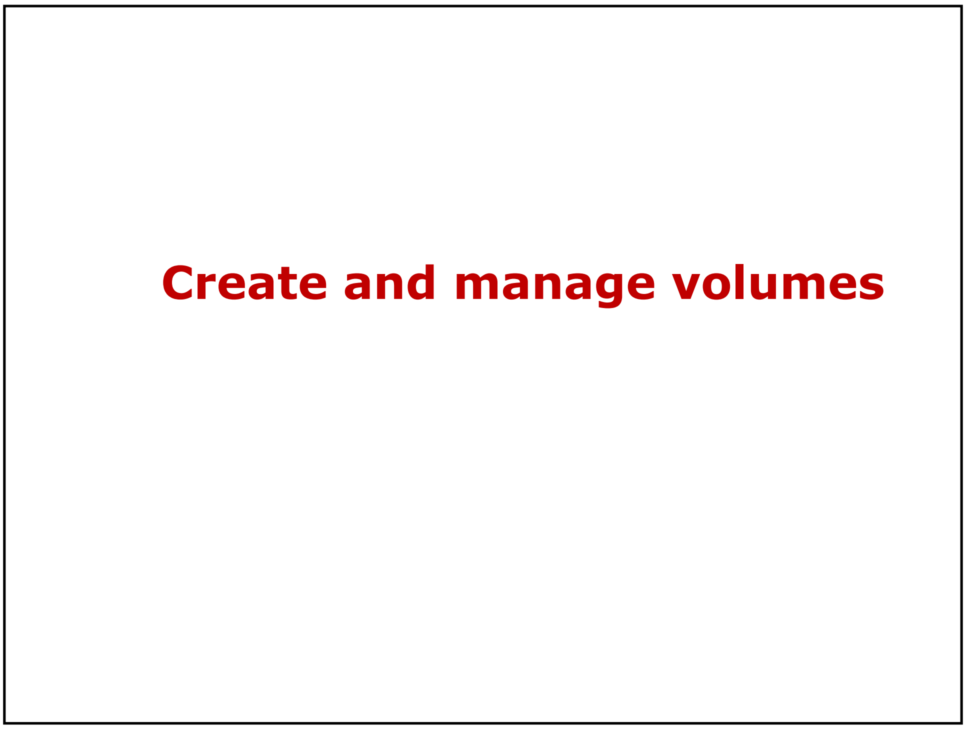
Outline

- Create and manage volumes
- Sharing Data using volumes
- Volumes vs Mounts
- The Docker network topology
- Understanding the default network
- Bridge vs Overlay networks

M.Romdhani, 2020

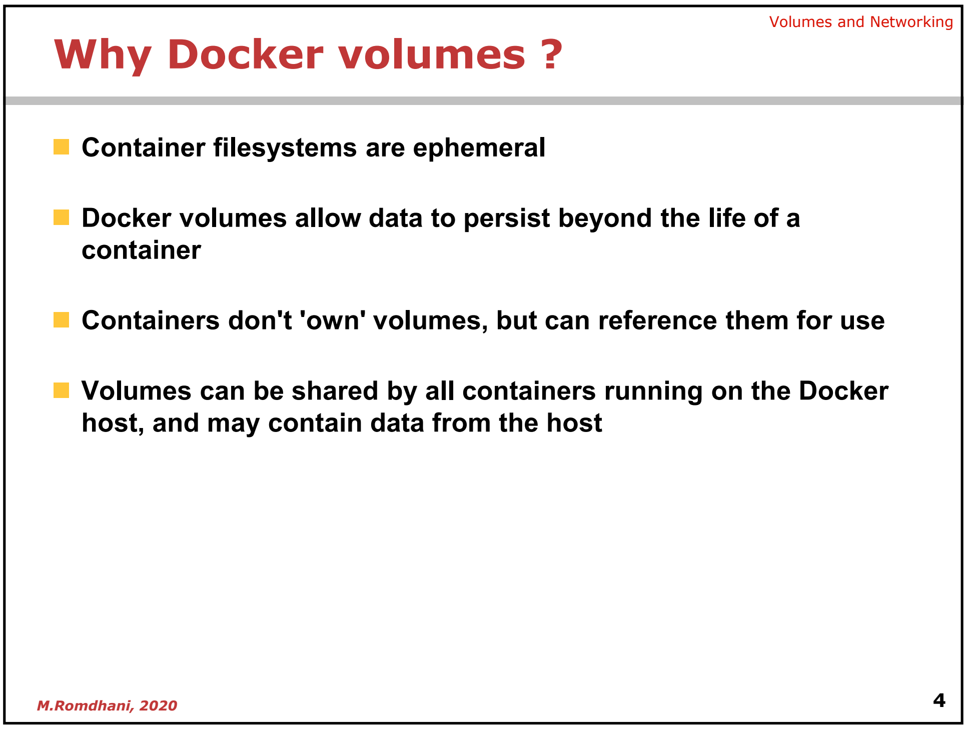
2

2



Create and manage volumes

3



Volumes and Networking

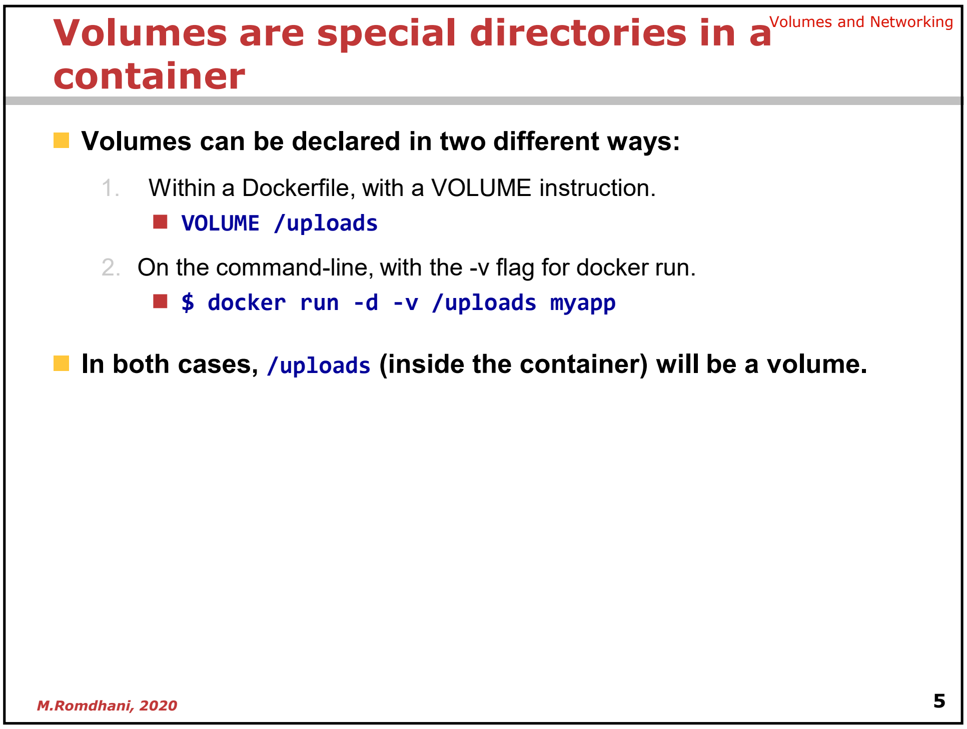
Why Docker volumes ?

- Container filesystems are ephemeral
- Docker volumes allow data to persist beyond the life of a container
- Containers don't 'own' volumes, but can reference them for use
- Volumes can be shared by all containers running on the Docker host, and may contain data from the host

M.Romdhani, 2020

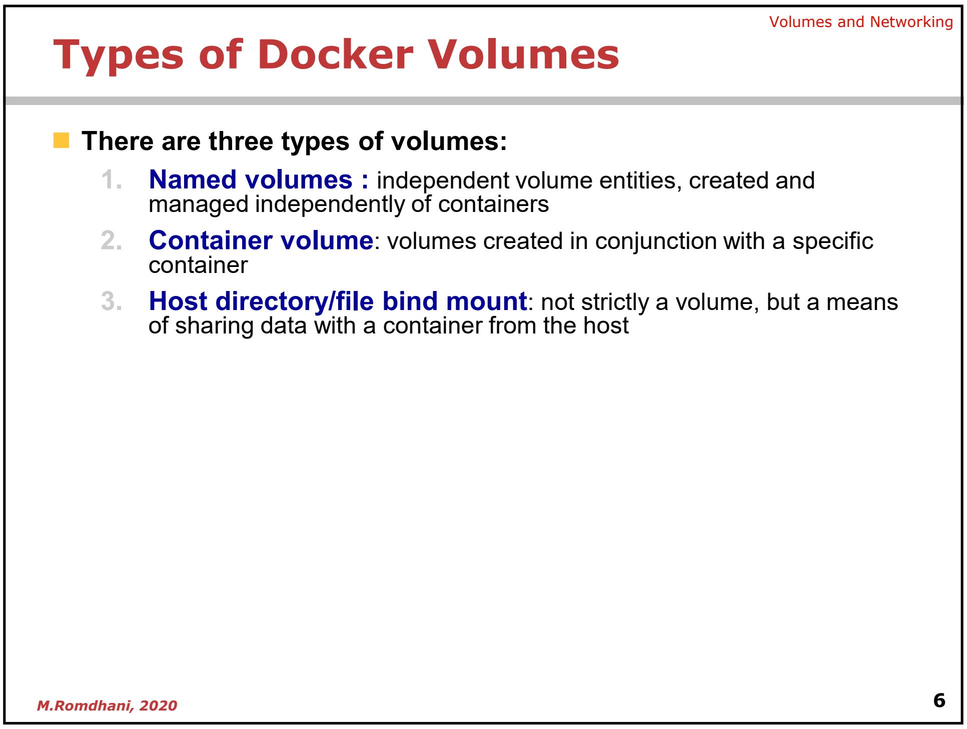
4

4



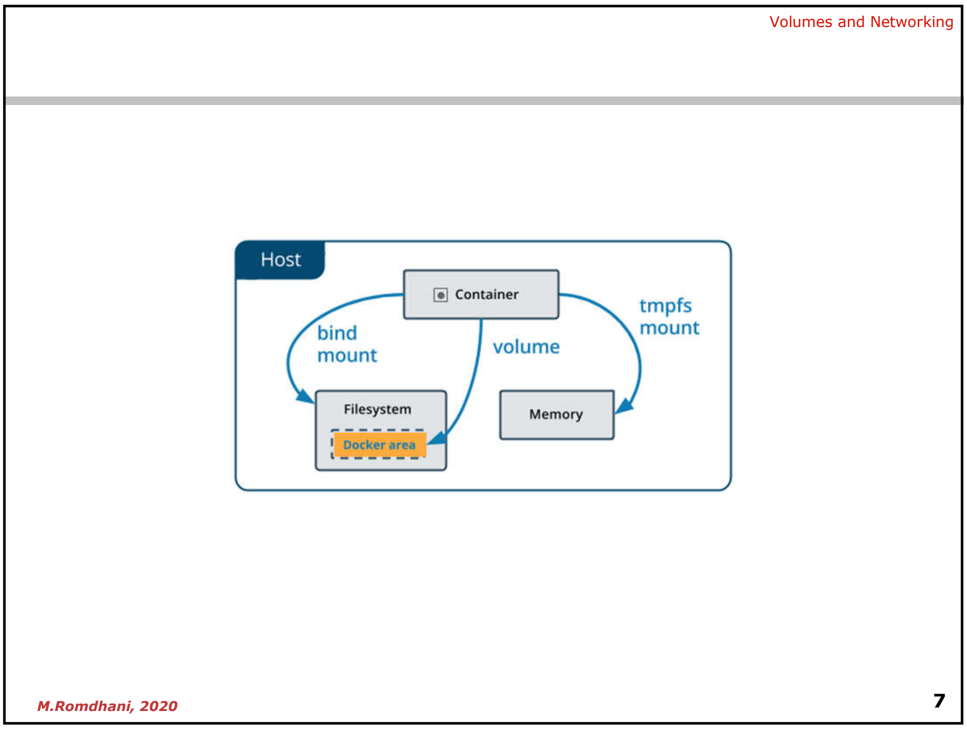
Volumes are special directories in a container

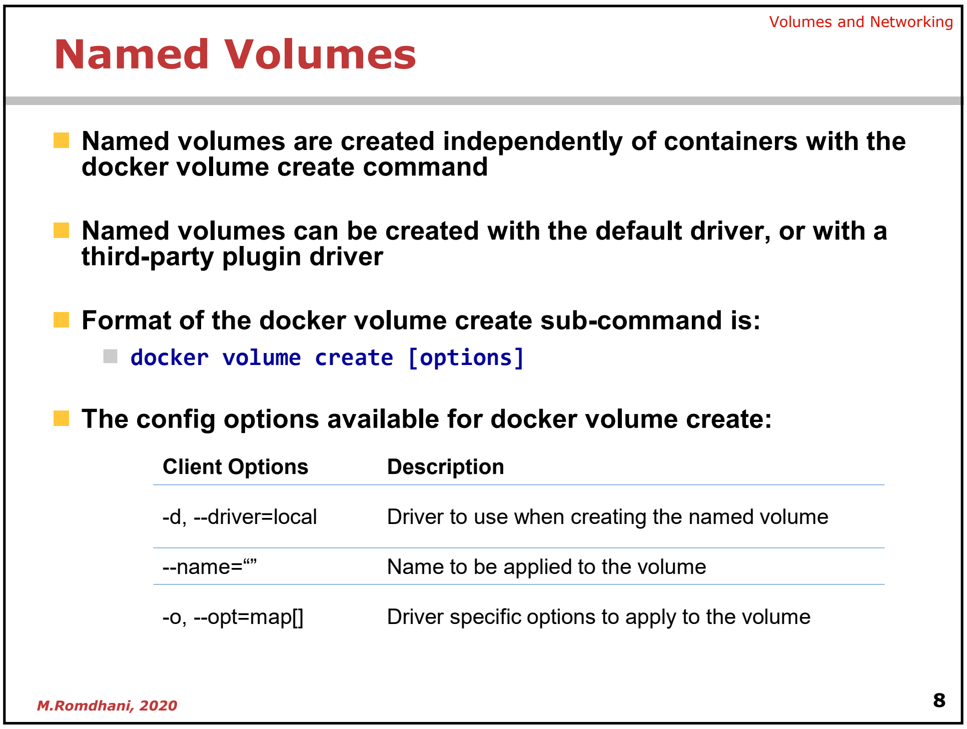
- **Volumes can be declared in two different ways:**
  1. Within a Dockerfile, with a VOLUME instruction.
     - `VOLUME /uploads`
  2. On the command-line, with the -v flag for docker run.
     - `$ docker run -d -v /uploads myapp`
- **In both cases, `/uploads` (inside the container) will be a volume.**



Types of Docker Volumes

- **There are three types of volumes:**
  1. **Named volumes :** independent volume entities, created and managed independently of containers
  2. **Container volume**: volumes created in conjunction with a specific container
  3. **Host directory/file bind mount**: not strictly a volume, but a means of sharing data with a container from the host

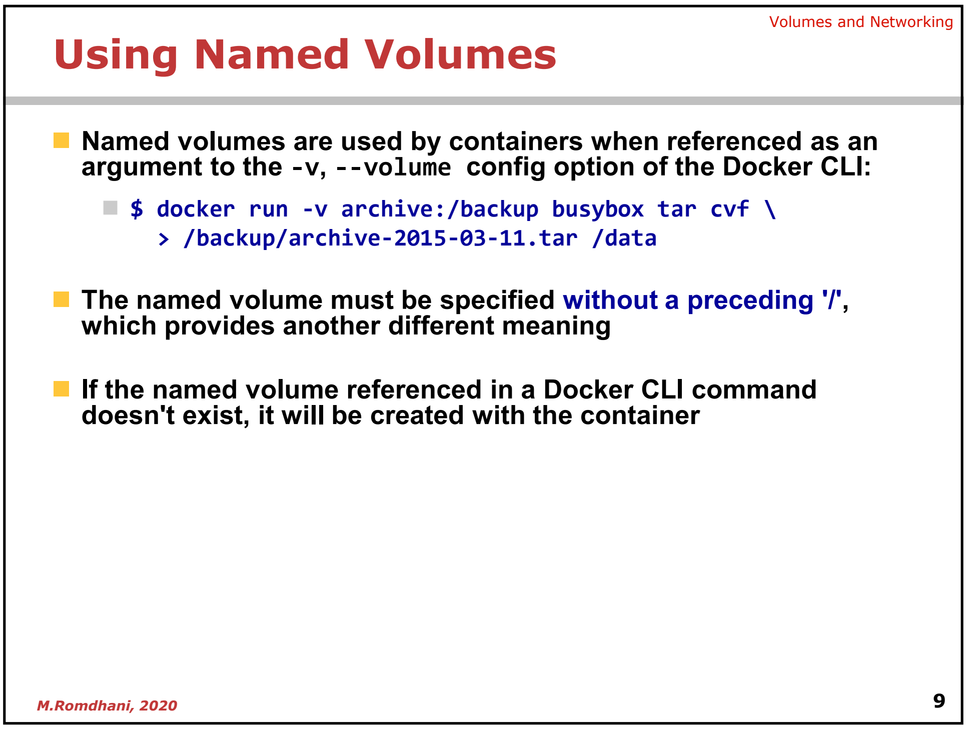




Named Volumes

- Named volumes are created independently of containers with the docker volume create command
- Named volumes can be created with the default driver, or with a third-party plugin driver
- Format of the docker volume create sub-command is:
  - `docker volume create [options]`
- The config options available for docker volume create:

Client Options	Description
-d, --driver=local	Driver to use when creating the named volume
--name=“”	Name to be applied to the volume
-o, --opt=map[]	Driver specific options to apply to the volume



Volumes and Networking

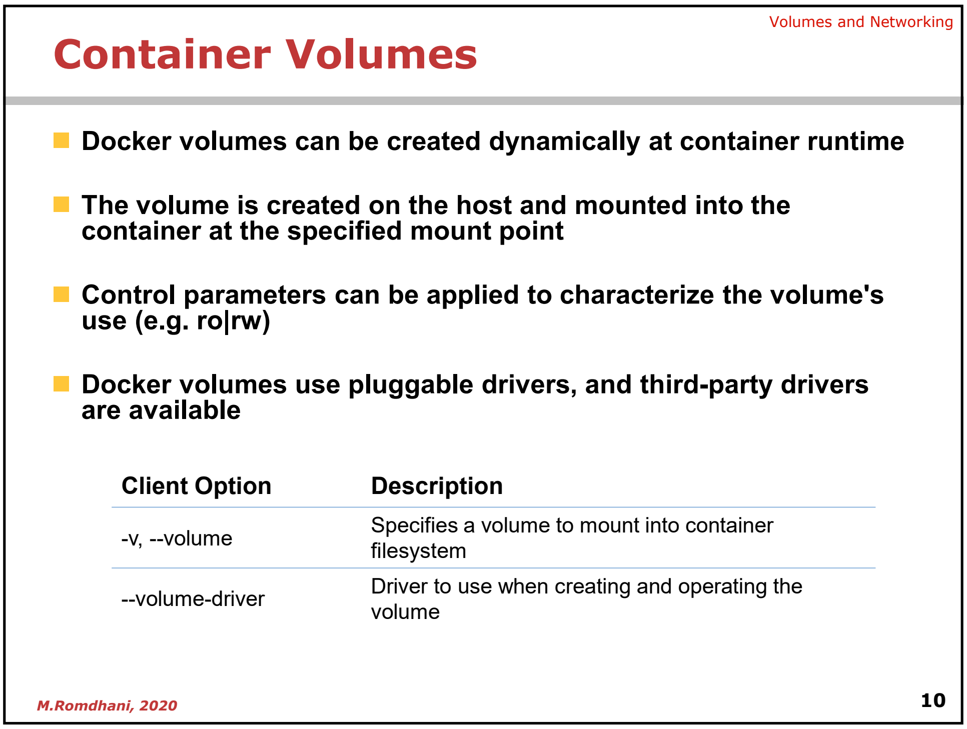
Using Named Volumes

- Named volumes are used by containers when referenced as an argument to the `-v`, `--volume` config option of the Docker CLI:

  -
    ```
    $ docker run -v archive:/backup busybox tar cvf \
      > /backup/archive-2015-03-11.tar /data
    ```
- The named volume must be specified without a preceding '/', which provides another different meaning
- If the named volume referenced in a Docker CLI command doesn't exist, it will be created with the container

M.Romdhani, 2020 9

9



Volumes and Networking

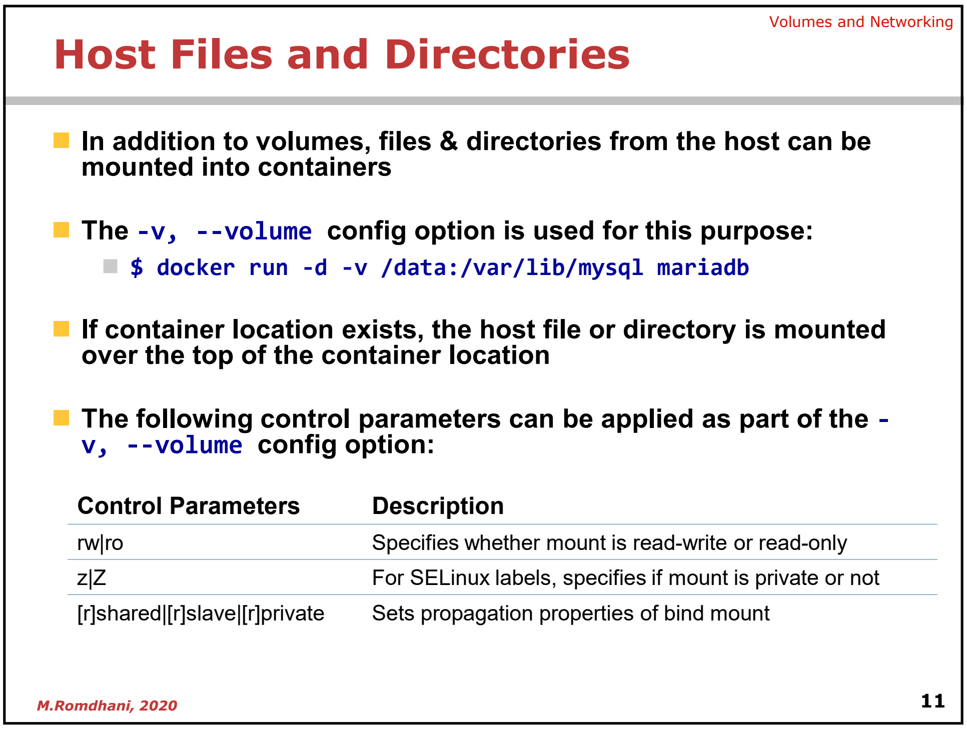
Container Volumes

- Docker volumes can be created dynamically at container runtime
- The volume is created on the host and mounted into the container at the specified mount point
- Control parameters can be applied to characterize the volume's use (e.g. ro|rw)
- Docker volumes use pluggable drivers, and third-party drivers are available

Client Option	Description
-v, --volume	Specifies a volume to mount into container filesystem
--volume-driver	Driver to use when creating and operating the volume

M.Romdhani, 2020 10

10



Volumes and Networking

Host Files and Directories

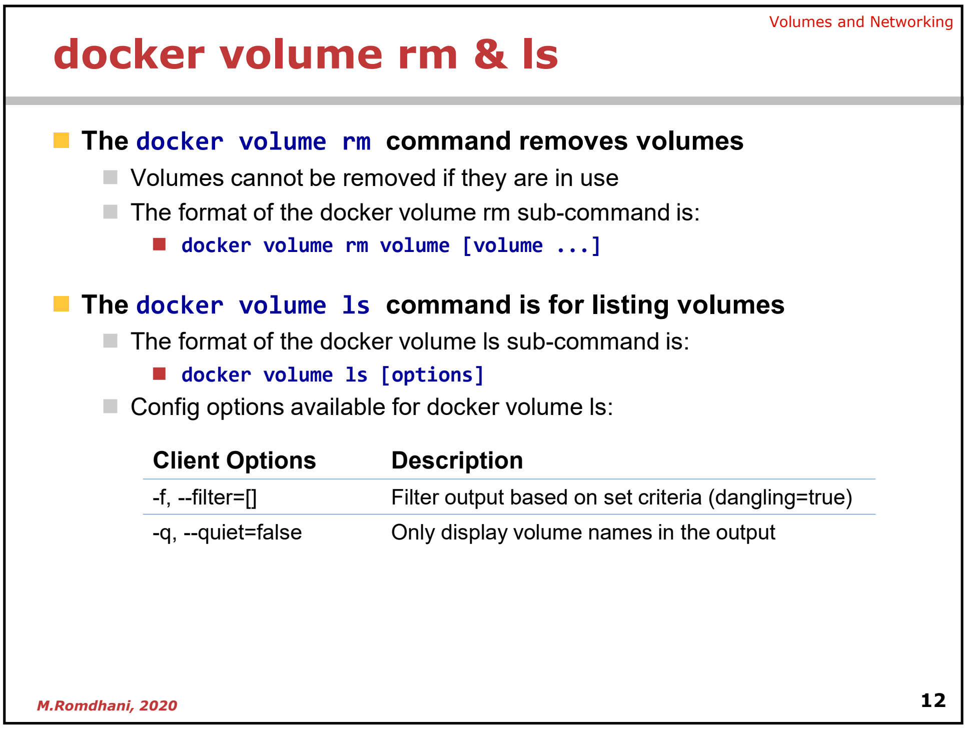
- In addition to volumes, files & directories from the host can be mounted into containers
- The `-v, --volume` config option is used for this purpose:
  - `$ docker run -d -v /data:/var/lib/mysql mariadb`
- If container location exists, the host file or directory is mounted over the top of the container location
- The following control parameters can be applied as part of the `-v, --volume` config option:

Control Parameters	Description
rw|ro	Specifies whether mount is read-write or read-only
z|Z	For SELinux labels, specifies if mount is private or not
[r]shared|[r]slave|[r]private	Sets propagation properties of bind mount

M.Romdhani, 2020

11

11



Volumes and Networking

docker volume rm & ls

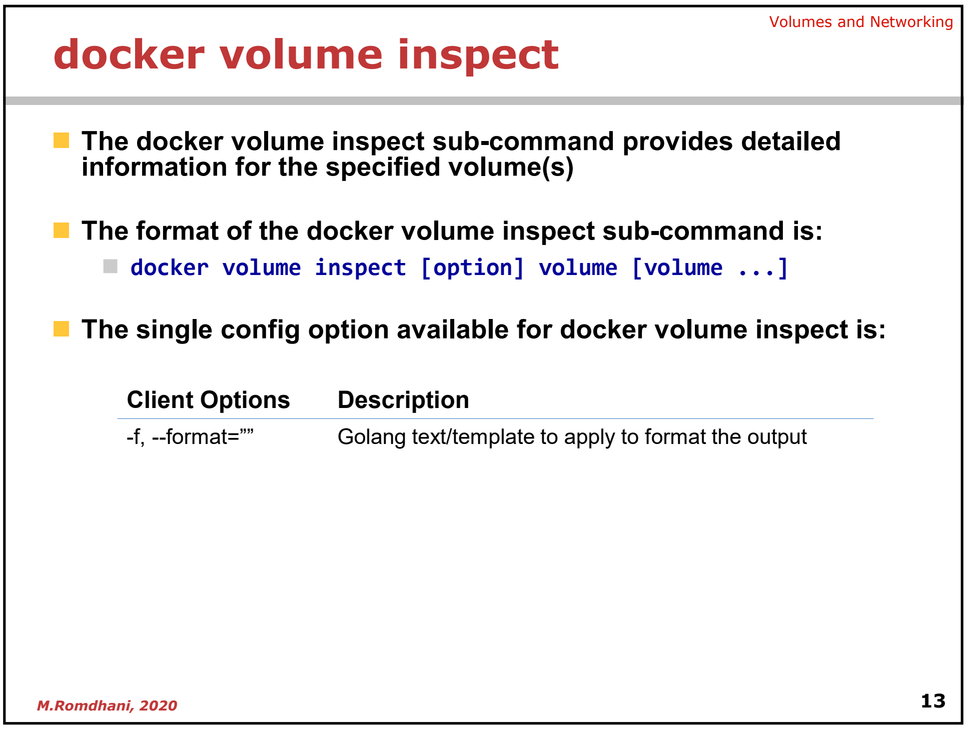
- The `docker volume rm` command removes volumes
  - Volumes cannot be removed if they are in use
  - The format of the docker volume rm sub-command is:
    - `docker volume rm volume [volume ...]`
- The `docker volume ls` command is for listing volumes
  - The format of the docker volume ls sub-command is:
    - `docker volume ls [options]`
  - Config options available for docker volume ls:

Client Options	Description
-f, --filter=[]	Filter output based on set criteria (dangling=true)
-q, --quiet=false	Only display volume names in the output

M.Romdhani, 2020

12

12



Volumes and Networking

docker volume inspect

- The docker volume inspect sub-command provides detailed information for the specified volume(s)
- The format of the docker volume inspect sub-command is:
  - `docker volume inspect [option] volume [volume ...]`
- The single config option available for docker volume inspect is:

Client Options	Description
-f, --format=""	Golang text/template to apply to format the output

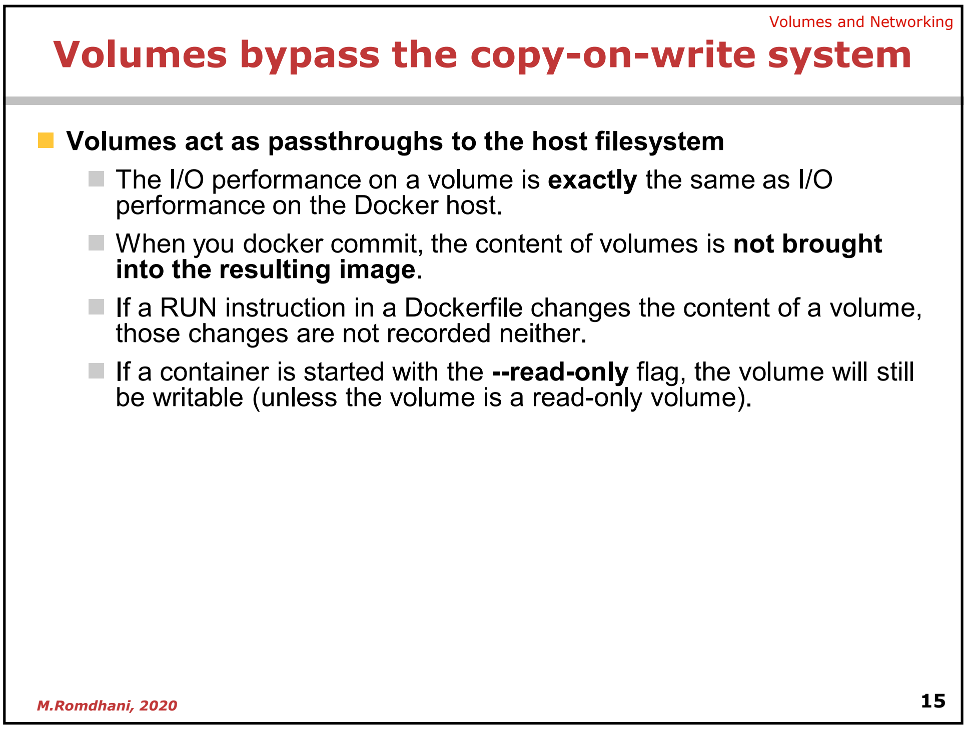
M.Romdhani, 2020

13

13

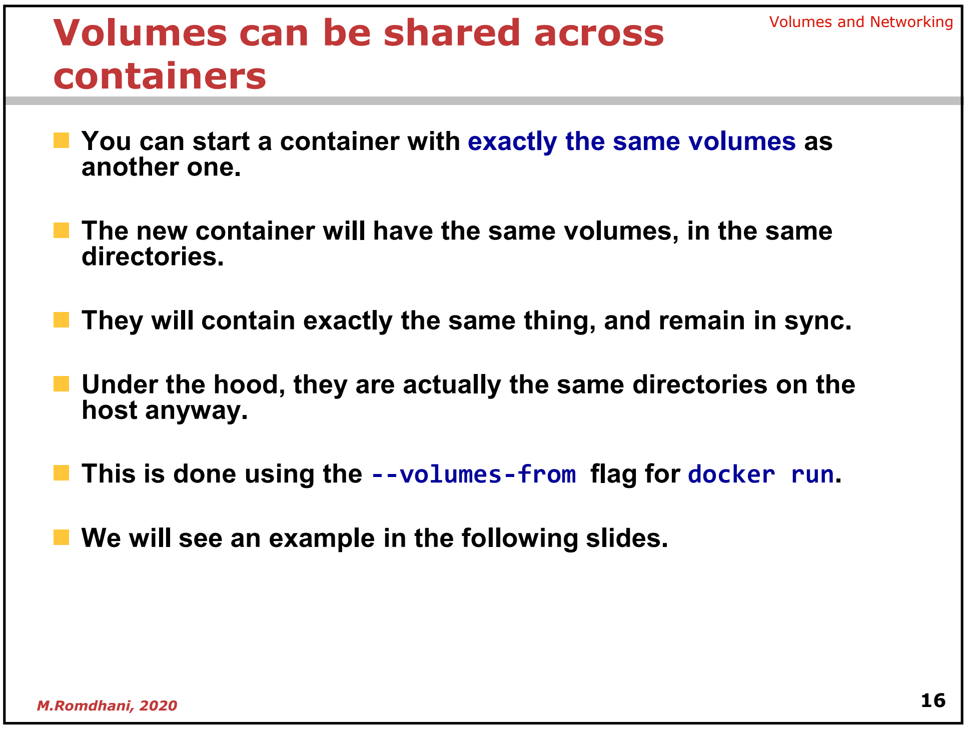
Share volumes across containers

14



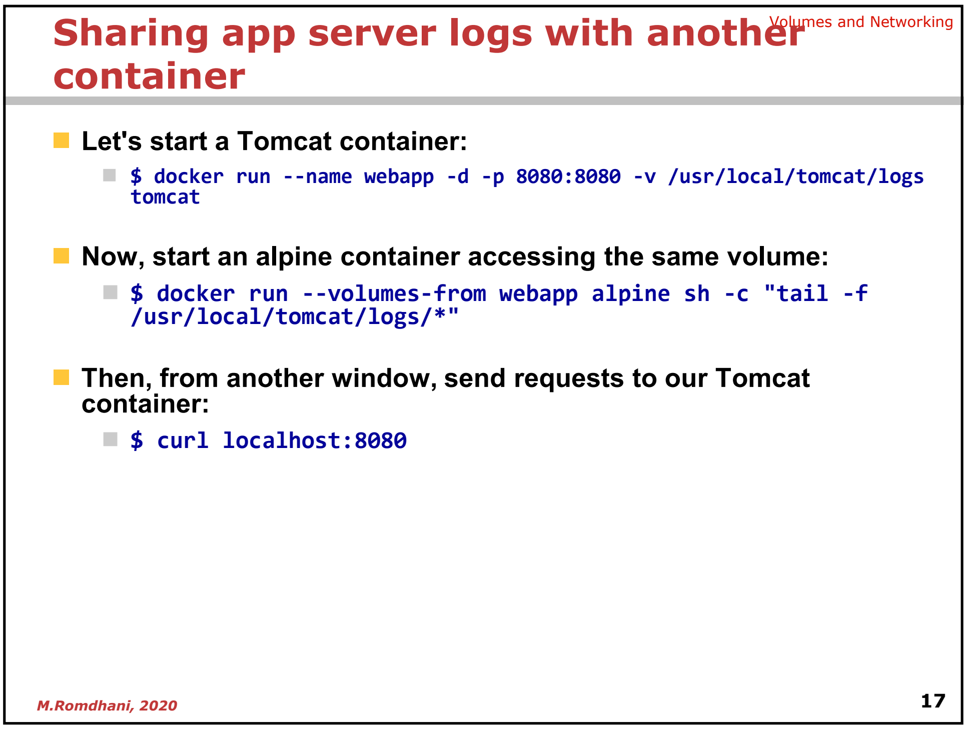
Volumes bypass the copy-on-write system

- **Volumes act as passthroughs to the host filesystem**
  - The I/O performance on a volume is **exactly** the same as I/O performance on the Docker host.
  - When you docker commit, the content of volumes is **not brought into the resulting image**.
  - If a RUN instruction in a Dockerfile changes the content of a volume, those changes are not recorded neither.
  - If a container is started with the **--read-only** flag, the volume will still be writable (unless the volume is a read-only volume).



Volumes can be shared across containers

- **You can start a container with exactly the same volumes as another one.**
- **The new container will have the same volumes, in the same directories.**
- **They will contain exactly the same thing, and remain in sync.**
- **Under the hood, they are actually the same directories on the host anyway.**
- **This is done using the `--volumes-from` flag for `docker run`.**
- **We will see an example in the following slides.**



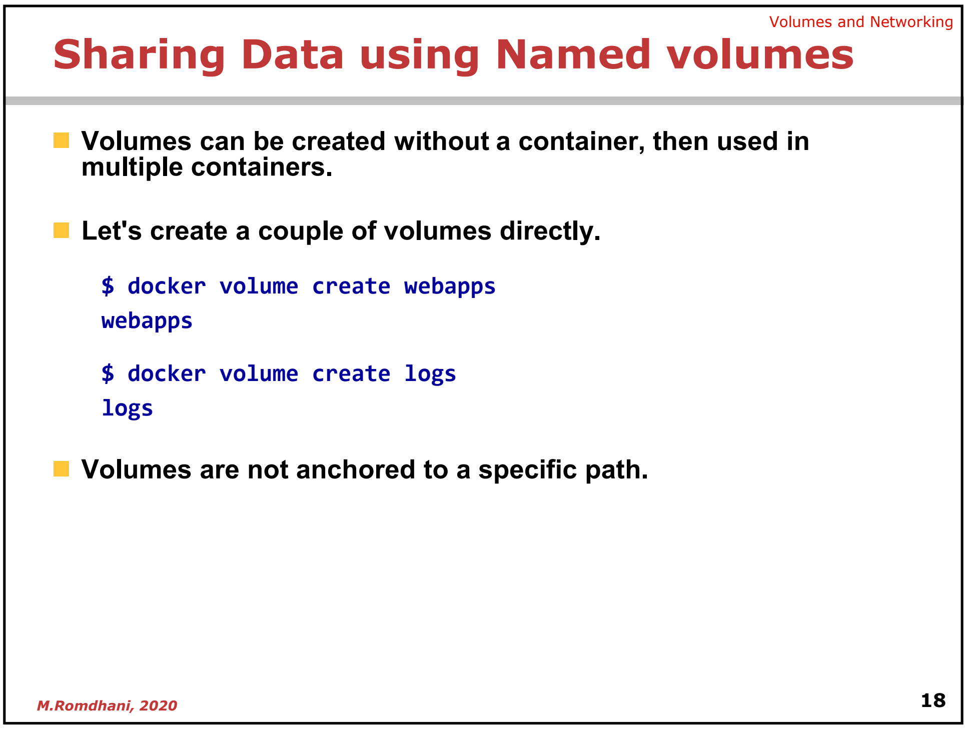
Sharing app server logs with another container

- Let's start a Tomcat container:

  -
    ```
    $ docker run --name webapp -d -p 8080:8080 -v /usr/local/tomcat/logs
    tomcat
    ```
- Now, start an alpine container accessing the same volume:

  -
    ```
    $ docker run --volumes-from webapp alpine sh -c "tail -f
    /usr/local/tomcat/logs/*"
    ```
- Then, from another window, send requests to our Tomcat container:

  -
    ```
    $ curl localhost:8080
    ```

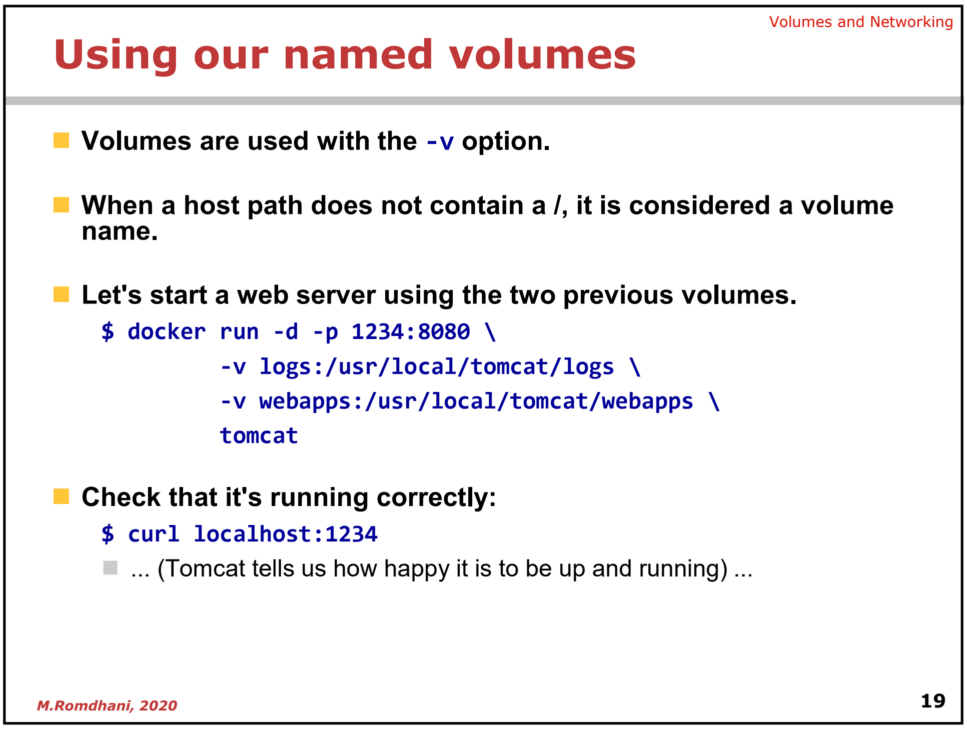


Sharing Data using Named volumes

- Volumes can be created without a container, then used in multiple containers.
- Let's create a couple of volumes directly.

```
$ docker volume create webapps
webapps

$ docker volume create logs
logs
```

- Volumes are not anchored to a specific path.



Volumes and Networking

Using our named volumes

- Volumes are used with the `-v` option.
- When a host path does not contain a /, it is considered a volume name.
- Let's start a web server using the two previous volumes.

```
$ docker run -d -p 1234:8080 \
         -v logs:/usr/local/tomcat/logs \
         -v webapps:/usr/local/tomcat/webapps \
         tomcat
```

- Check that it's running correctly:

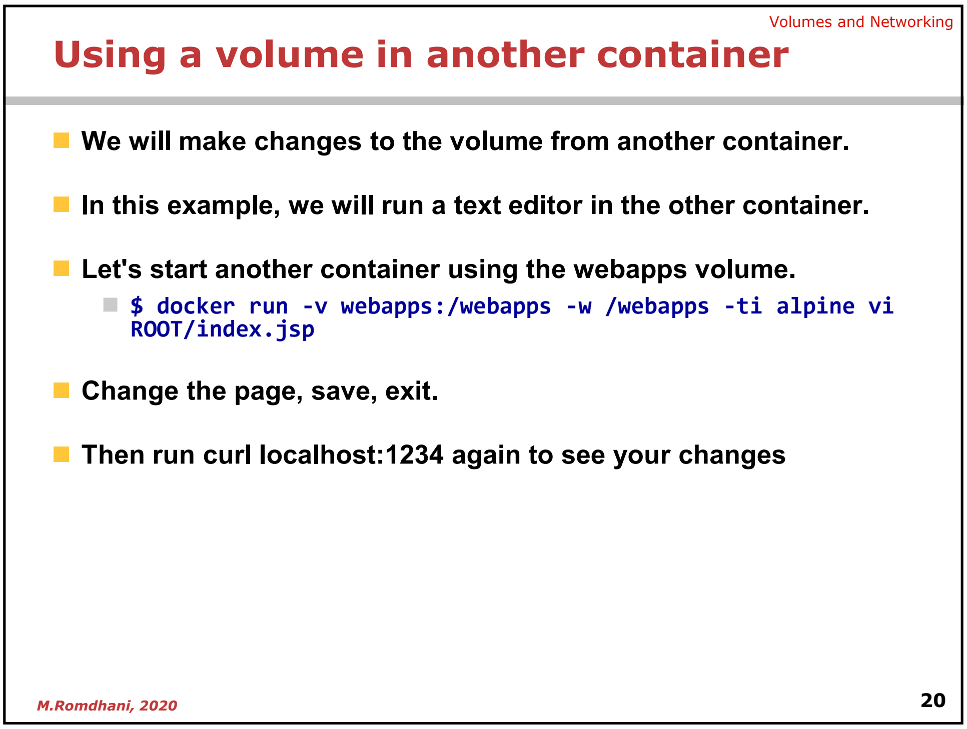
```
$ curl localhost:1234
```

  - ... (Tomcat tells us how happy it is to be up and running) ...

M.Romdhani, 2020

19

19



Volumes and Networking

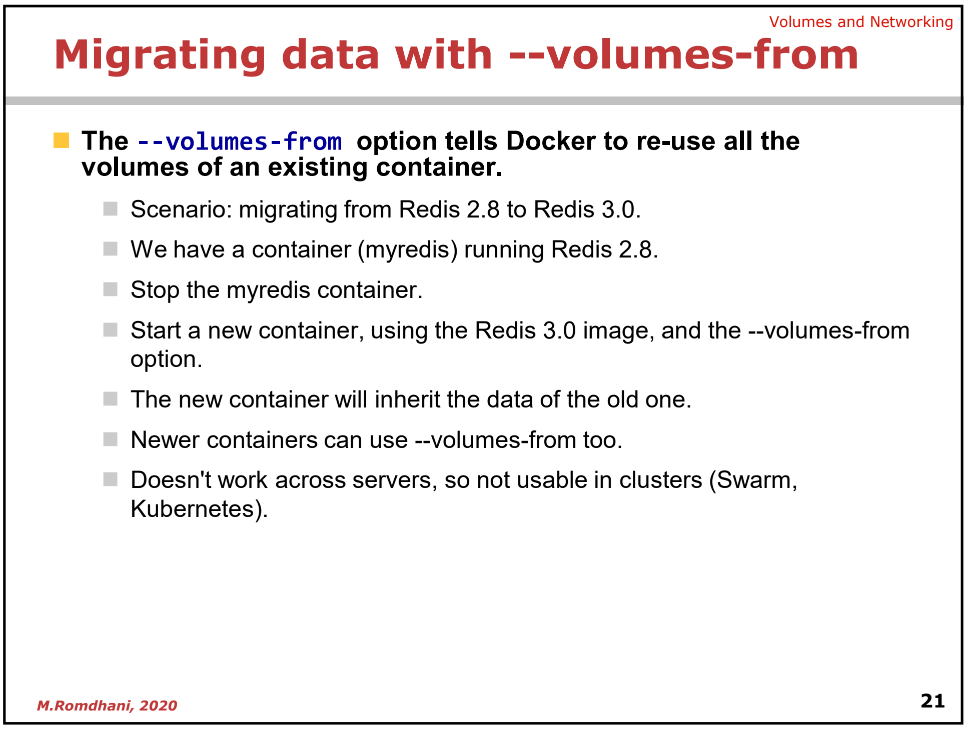
Using a volume in another container

- We will make changes to the volume from another container.
- In this example, we will run a text editor in the other container.
- Let's start another container using the webapps volume.
  - `$ docker run -v webapps:/webapps -w /webapps -ti alpine vi ROOT/index.jsp`
- Change the page, save, exit.
- Then run curl localhost:1234 again to see your changes

M.Romdhani, 2020

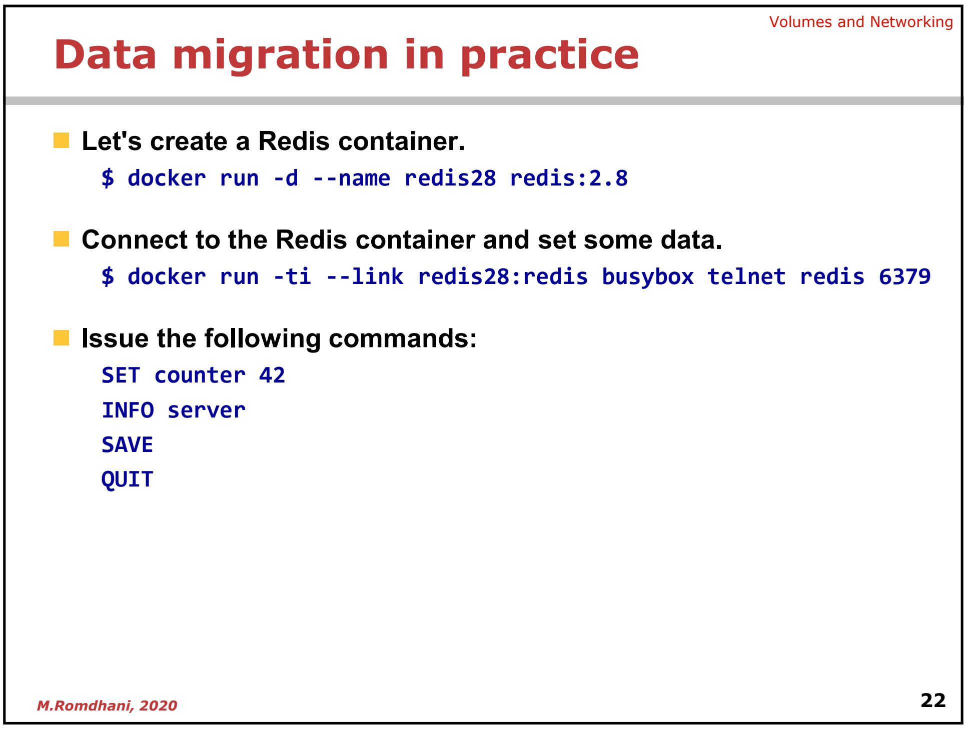
20

20



Migrating data with --volumes-from

- **The `--volumes-from` option tells Docker to re-use all the volumes of an existing container.**
  - Scenario: migrating from Redis 2.8 to Redis 3.0.
  - We have a container (myredis) running Redis 2.8.
  - Stop the myredis container.
  - Start a new container, using the Redis 3.0 image, and the --volumes-from option.
  - The new container will inherit the data of the old one.
  - Newer containers can use --volumes-from too.
  - Doesn't work across servers, so not usable in clusters (Swarm, Kubernetes).



Data migration in practice

- **Let's create a Redis container.**

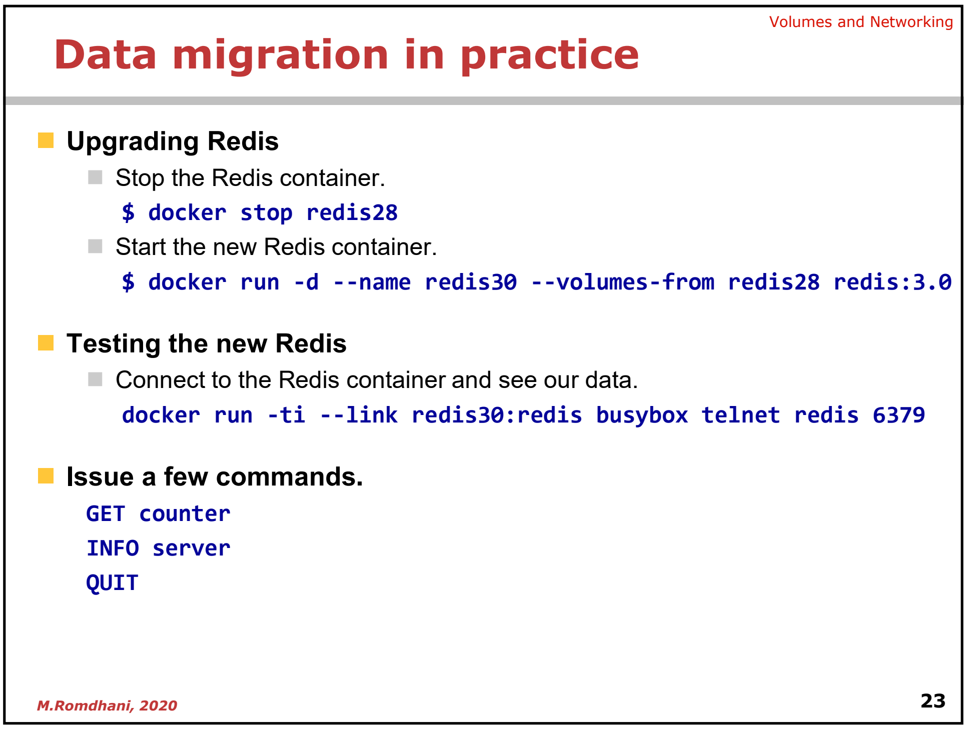
```
$ docker run -d --name redis28 redis:2.8
```

- **Connect to the Redis container and set some data.**

```
$ docker run -ti --link redis28:redis busybox telnet redis 6379
```

- **Issue the following commands:**

```
SET counter 42
INFO server
SAVE
QUIT
```



Data migration in practice

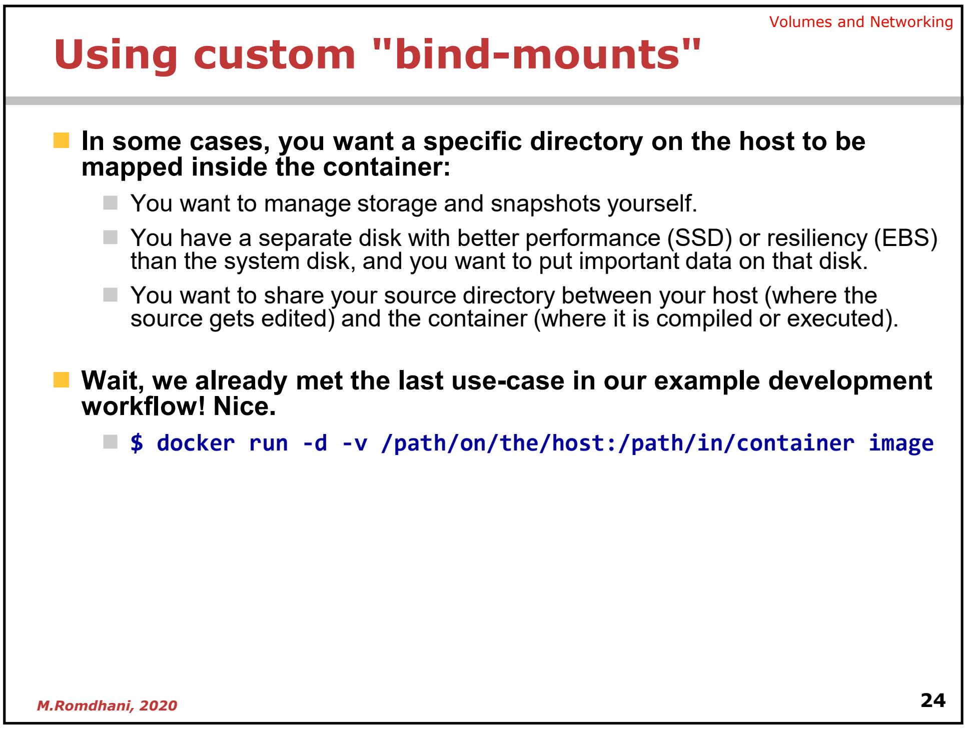
- **Upgrading Redis**
  - Stop the Redis container.

    ```
    $ docker stop redis28
    ```
  - Start the new Redis container.

    ```
    $ docker run -d --name redis30 --volumes-from redis28 redis:3.0
    ```
- **Testing the new Redis**
  - Connect to the Redis container and see our data.

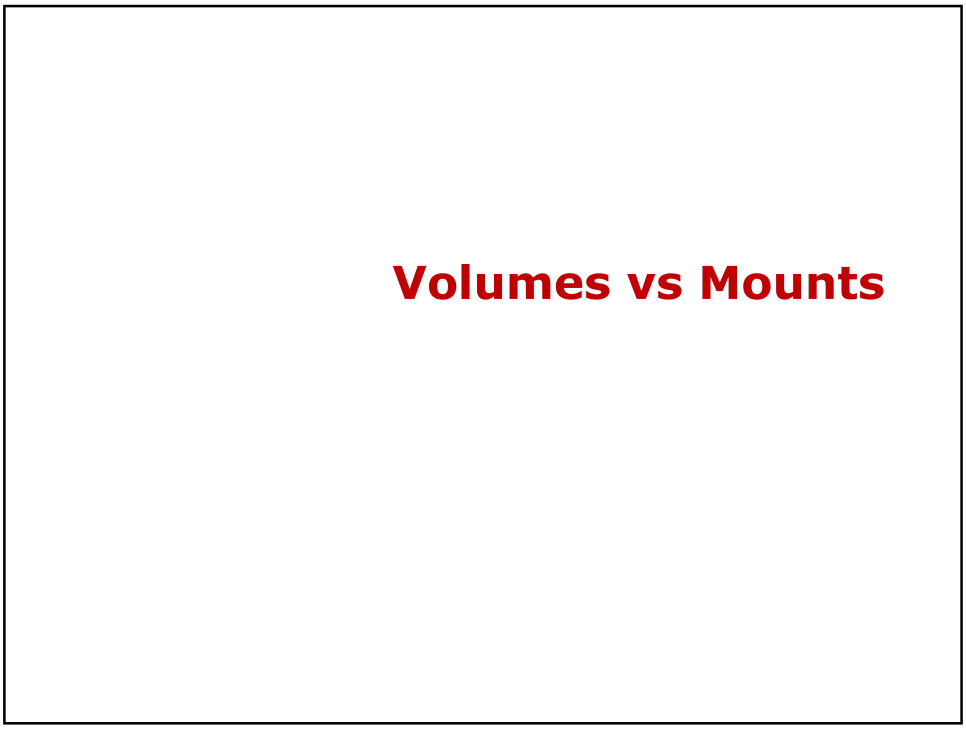
    ```
    docker run -ti --link redis30:redis busybox telnet redis 6379
    ```
- **Issue a few commands.**

  ```
  GET counter
  INFO server
  QUIT
  ```



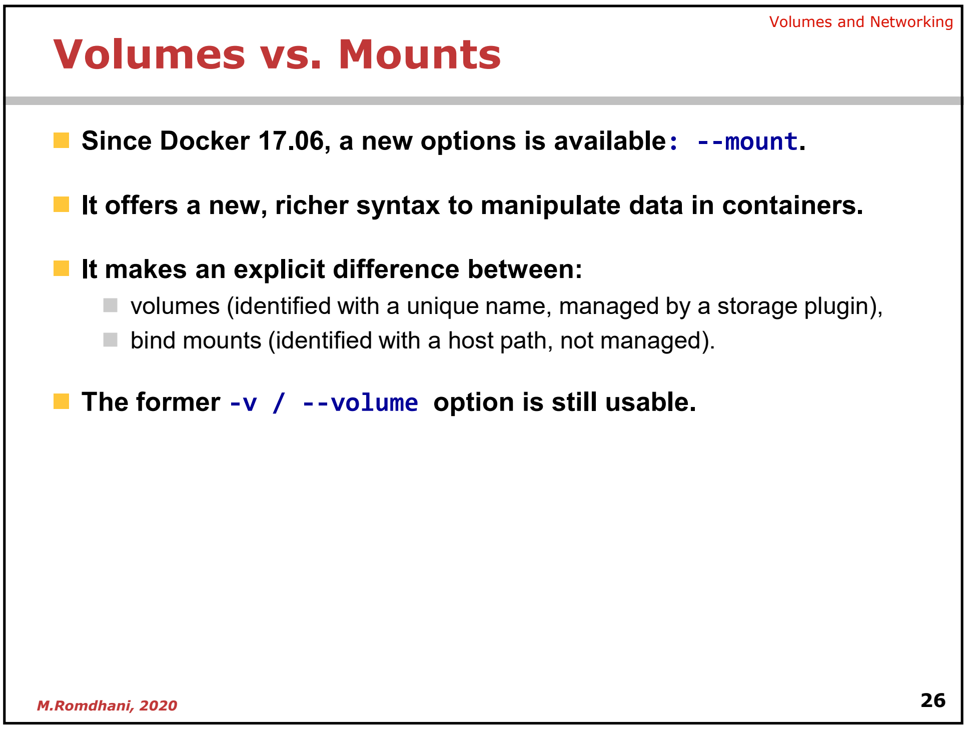
Using custom "bind-mounts"

- **In some cases, you want a specific directory on the host to be mapped inside the container:**
  - You want to manage storage and snapshots yourself.
  - You have a separate disk with better performance (SSD) or resiliency (EBS) than the system disk, and you want to put important data on that disk.
  - You want to share your source directory between your host (where the source gets edited) and the container (where it is compiled or executed).
- **Wait, we already met the last use-case in our example development workflow! Nice.**
  - `$ docker run -d -v /path/on/the/host:/path/in/container image`



Volumes vs Mounts

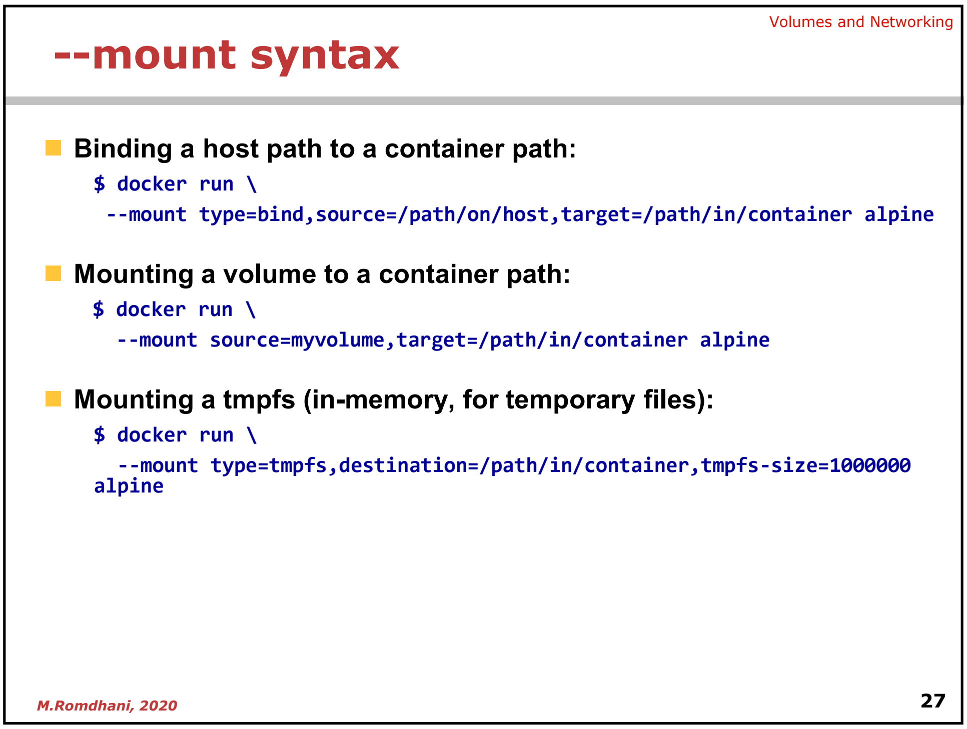
25



Volumes vs. Mounts

- **Since Docker 17.06, a new options is available: `--mount`.**
- **It offers a new, richer syntax to manipulate data in containers.**
- **It makes an explicit difference between:**
  - volumes (identified with a unique name, managed by a storage plugin),
  - bind mounts (identified with a host path, not managed).
- **The former `-v / --volume` option is still usable.**

26



--mount syntax

- Binding a host path to a container path:

```
$ docker run \
 --mount type=bind,source=/path/on/host,target=/path/in/container alpine
```

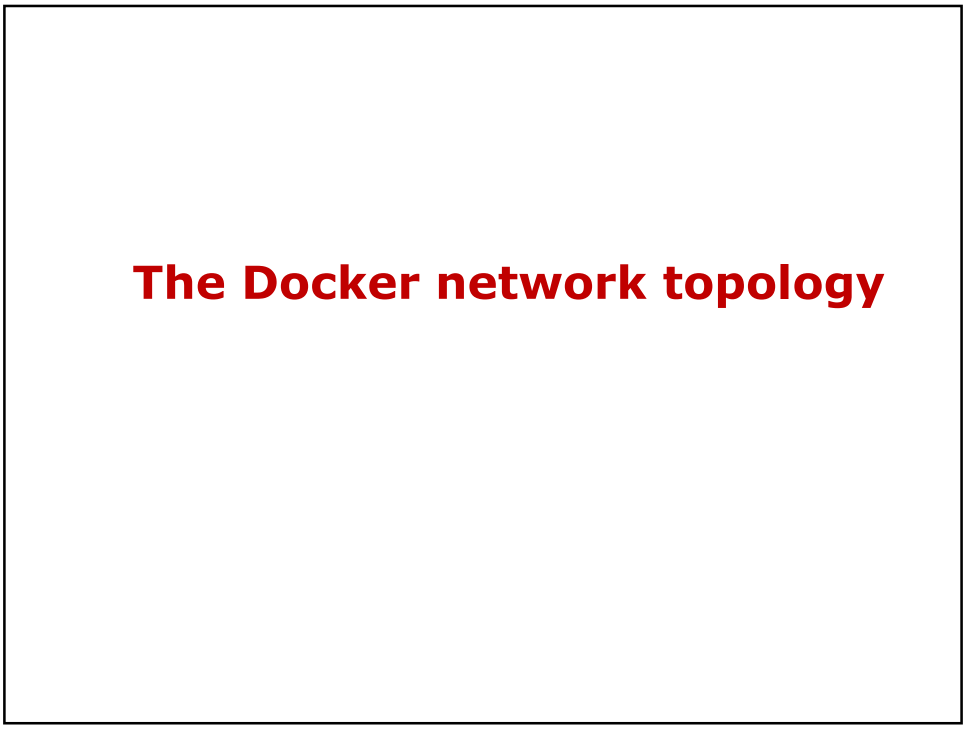
- Mounting a volume to a container path:

```
$ docker run \
  --mount source=myvolume,target=/path/in/container alpine
```

- Mounting a tmpfs (in-memory, for temporary files):

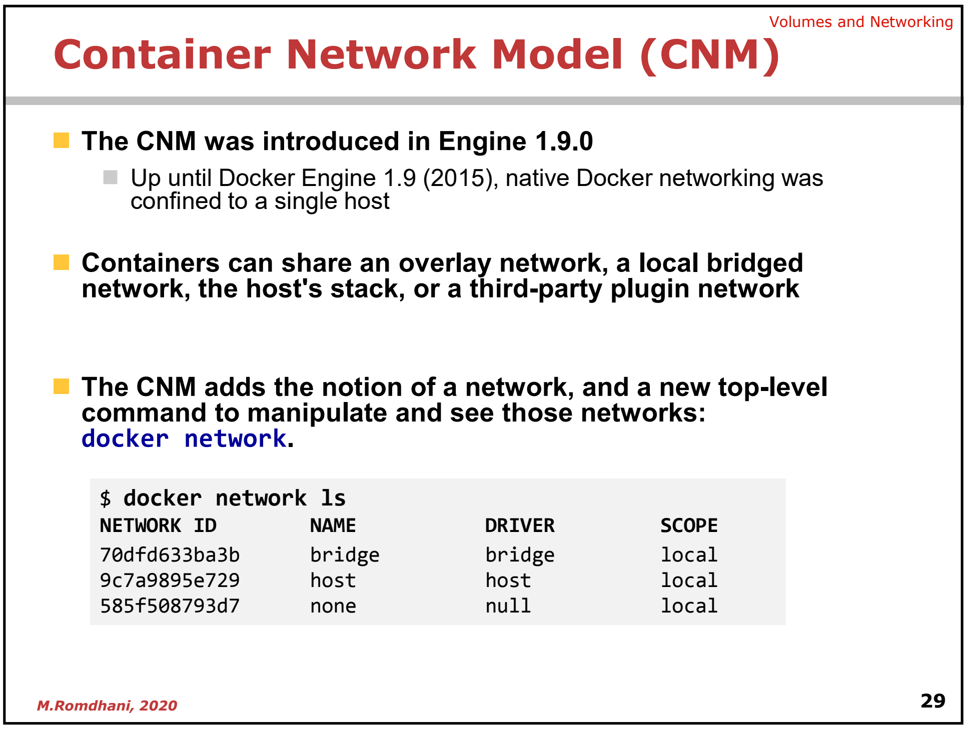
```
$ docker run \
  --mount type=tmpfs,destination=/path/in/container,tmpfs-size=1000000
alpine
```

27



The Docker network topology

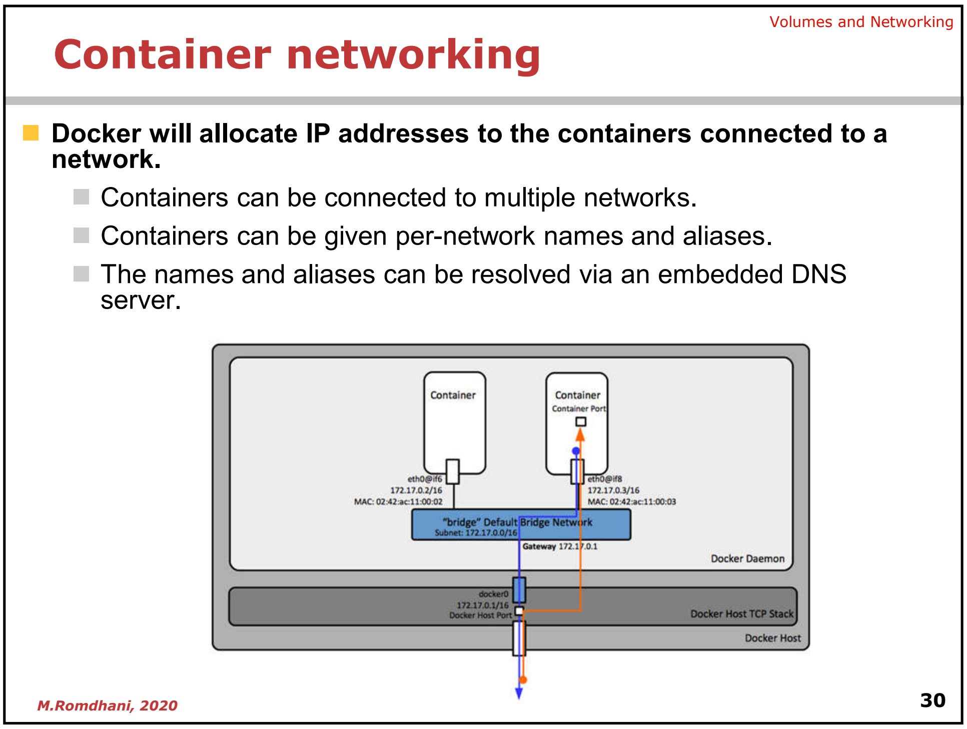
28



Container Network Model (CNM)

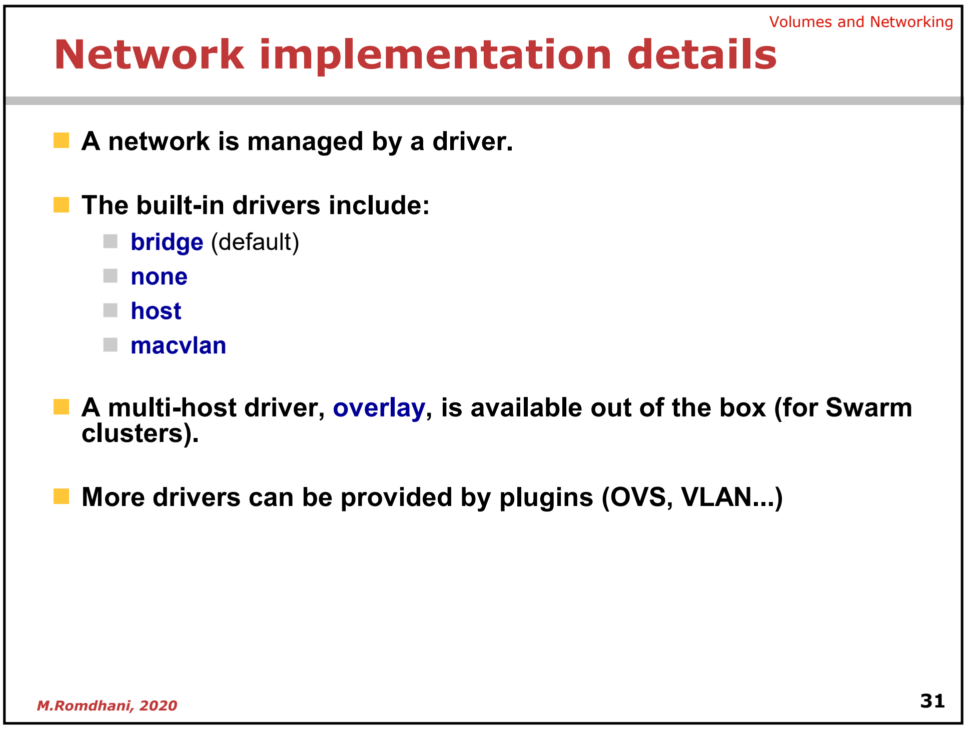
- **The CNM was introduced in Engine 1.9.0**
  - Up until Docker Engine 1.9 (2015), native Docker networking was confined to a single host
- **Containers can share an overlay network, a local bridged network, the host's stack, or a third-party plugin network**
- **The CNM adds the notion of a network, and a new top-level command to manipulate and see those networks: `docker network`.**

```
$ docker network ls
NETWORK ID        NAME          DRIVER        SCOPE
70dfd633ba3b      bridge        bridge        local
9c7a9895e729      host          host          local
585f508793d7      none          null          local
```



Container networking

- **Docker will allocate IP addresses to the containers connected to a network.**
  - Containers can be connected to multiple networks.
  - Containers can be given per-network names and aliases.
  - The names and aliases can be resolved via an embedded DNS server.

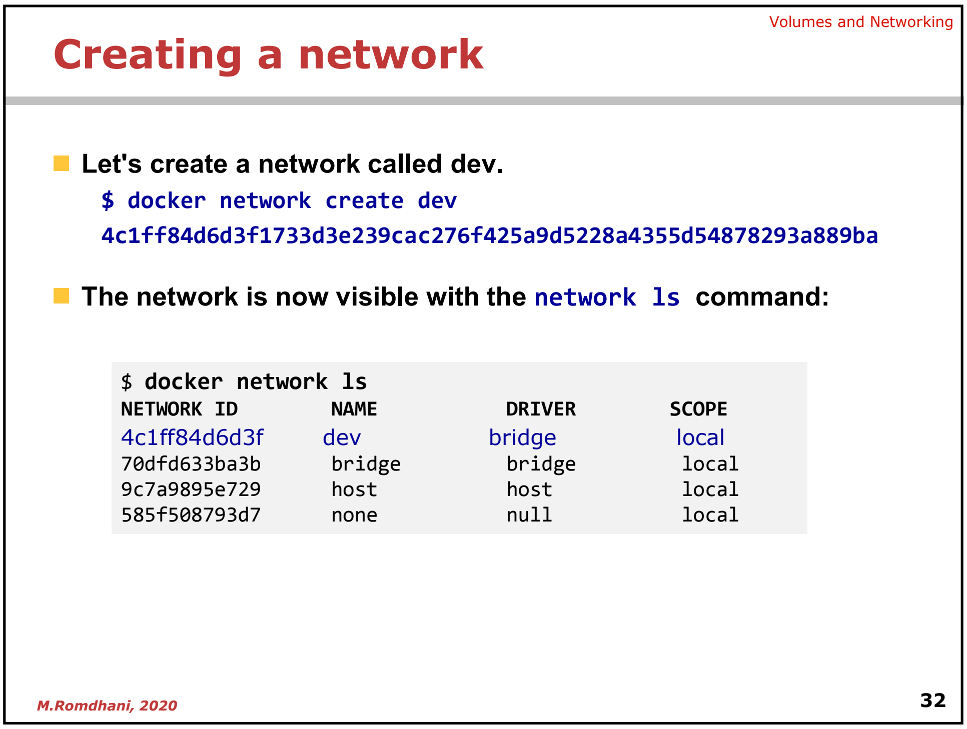


Volumes and Networking

Network implementation details

- A network is managed by a driver.
- The built-in drivers include:
  - bridge (default)
  - none
  - host
  - macvlan
- A multi-host driver, overlay, is available out of the box (for Swarm clusters).
- More drivers can be provided by plugins (OVS, VLAN...)

M.Romdhani, 2020 31



Volumes and Networking

Creating a network

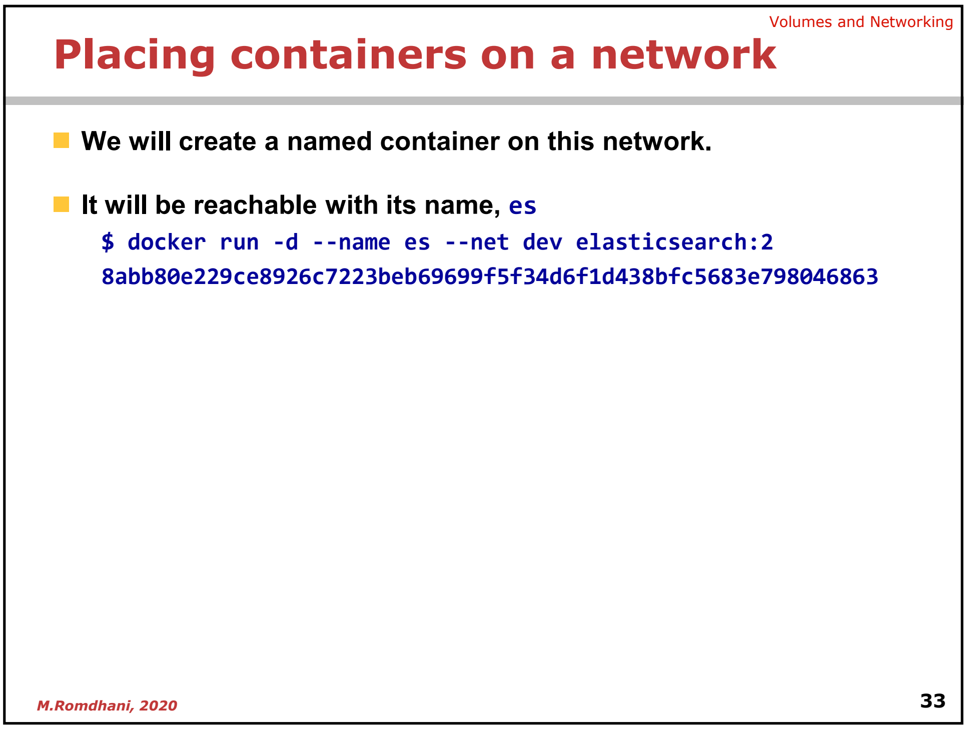
- Let's create a network called dev.

```
$ docker network create dev
4c1ff84d6d3f1733d3e239cac276f425a9d5228a4355d54878293a889ba
```

- The network is now visible with the `network ls` command:

```
$ docker network ls
NETWORK ID          NAME           DRIVER        SCOPE
4c1ff84d6d3f       dev            bridge        local
70dfd633ba3b        bridge         bridge         local
9c7a9895e729        host           host           local
585f508793d7        none           null           local
```

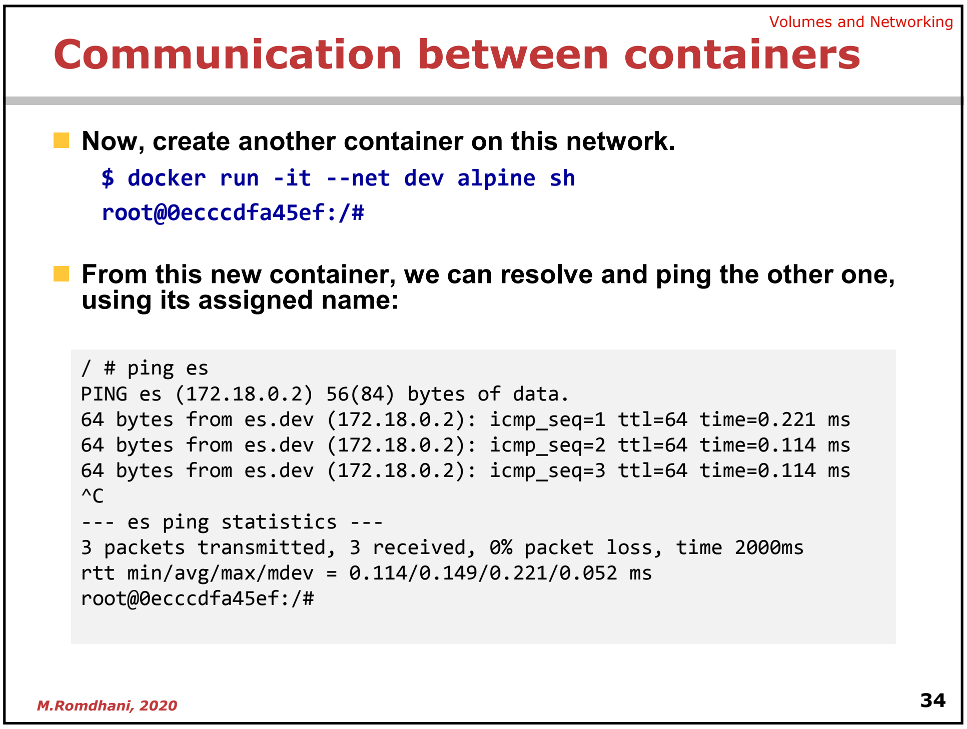
M.Romdhani, 2020 32



Placing containers on a network

- We will create a named container on this network.
- It will be reachable with its name, es

```
$ docker run -d --name es --net dev elasticsearch:2
8abb80e229ce8926c7223beb69699f5f34d6f1d438bfc5683e798046863
```



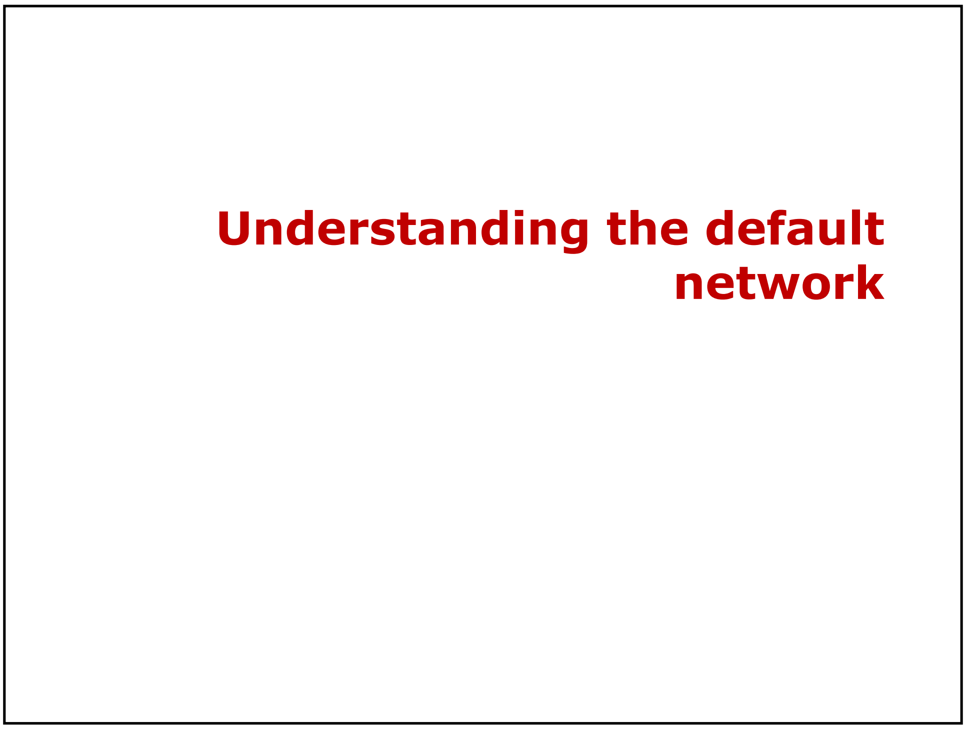
Communication between containers

- Now, create another container on this network.

```
$ docker run -it --net dev alpine sh
root@0ecccdfa45ef:/#
```

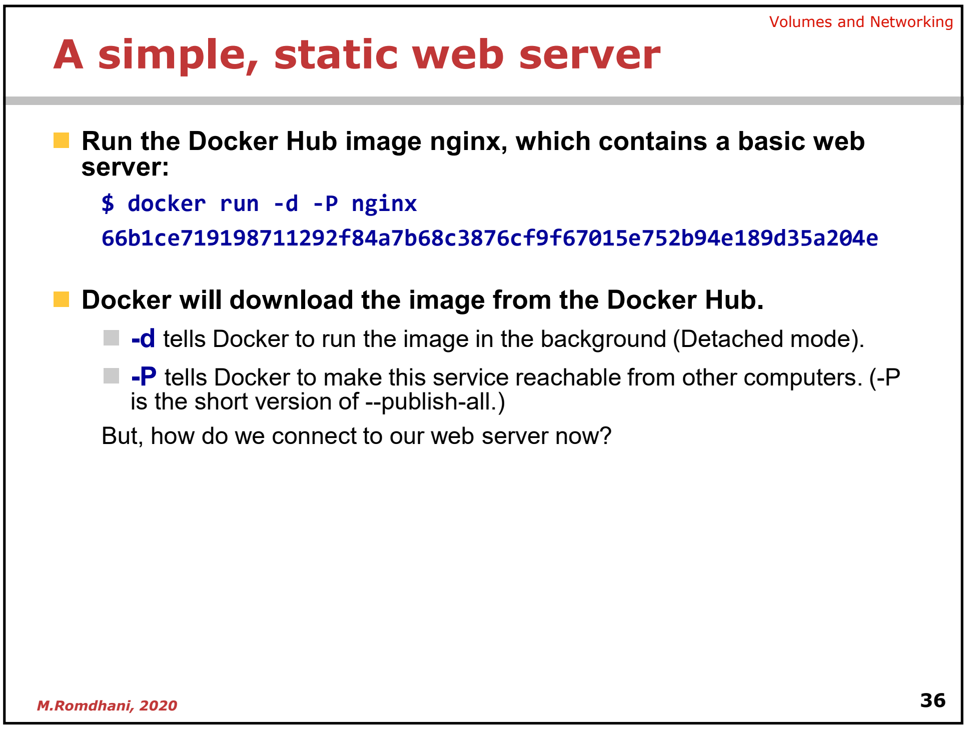
- From this new container, we can resolve and ping the other one, using its assigned name:

```
/ # ping es
PING es (172.18.0.2) 56(84) bytes of data.
64 bytes from es.dev (172.18.0.2): icmp_seq=1 ttl=64 time=0.221 ms
64 bytes from es.dev (172.18.0.2): icmp_seq=2 ttl=64 time=0.114 ms
64 bytes from es.dev (172.18.0.2): icmp_seq=3 ttl=64 time=0.114 ms
^C
--- es ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.114/0.149/0.221/0.052 ms
root@0ecccdfa45ef:/#
```



Understanding the default network

35



Volumes and Networking

A simple, static web server

- **Run the Docker Hub image nginx, which contains a basic web server:**

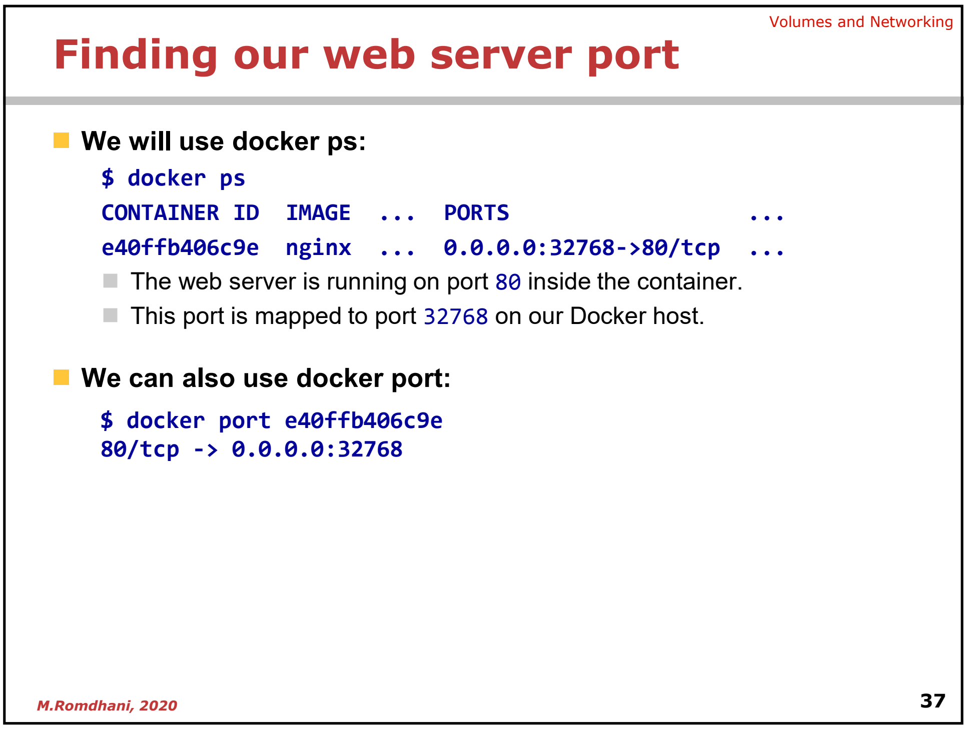
```
$ docker run -d -P nginx
66b1ce719198711292f84a7b68c3876cf9f67015e752b94e189d35a204e
```

- **Docker will download the image from the Docker Hub.**
  - **-d** tells Docker to run the image in the background (Detached mode).
  - **-P** tells Docker to make this service reachable from other computers. (-P is the short version of --publish-all.)

  But, how do we connect to our web server now?

M.Romdhani, 2020 36

36



Finding our web server port

- **We will use docker ps:**

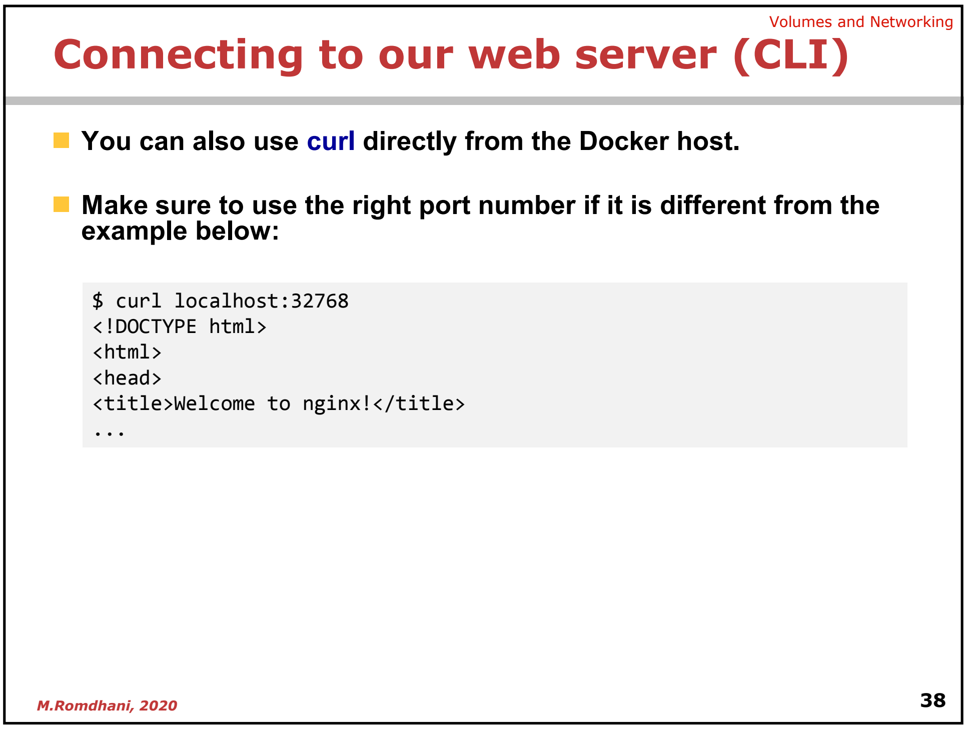
```
$ docker ps
CONTAINER ID  IMAGE  ...  PORTS                   ...
e40ffb406c9e  nginx  ...  0.0.0.0:32768->80/tcp  ...
```

  - The web server is running on port `80` inside the container.
  - This port is mapped to port `32768` on our Docker host.

- **We can also use docker port:**

```
$ docker port e40ffb406c9e
80/tcp -> 0.0.0.0:32768
```

37



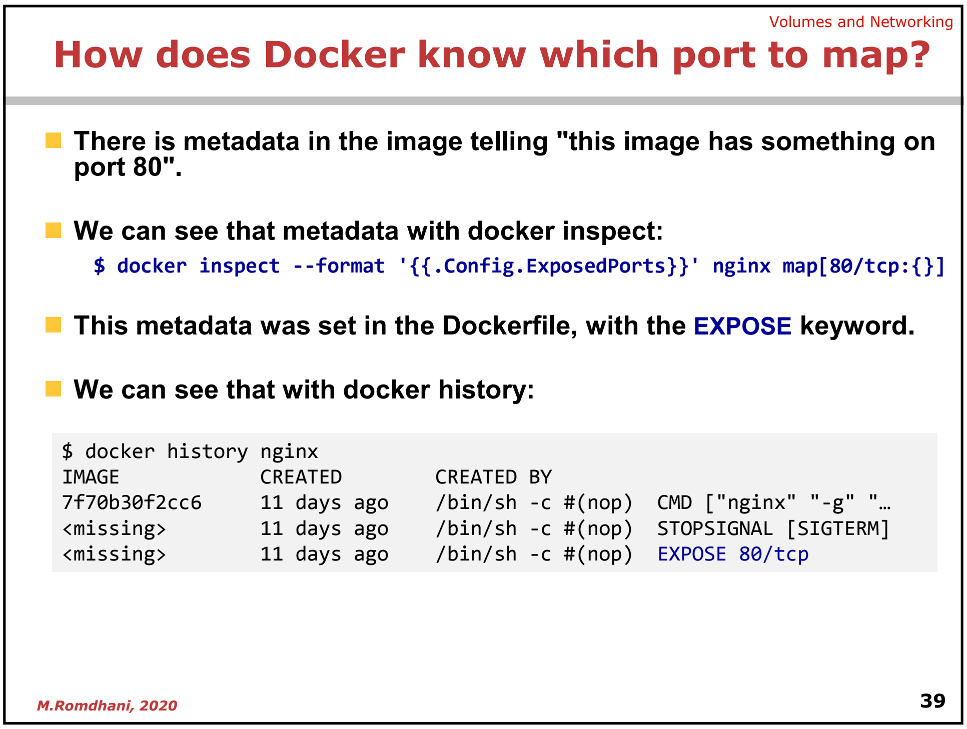
Connecting to our web server (CLI)

- **You can also use curl directly from the Docker host.**

- **Make sure to use the right port number if it is different from the example below:**

```
$ curl localhost:32768
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

38



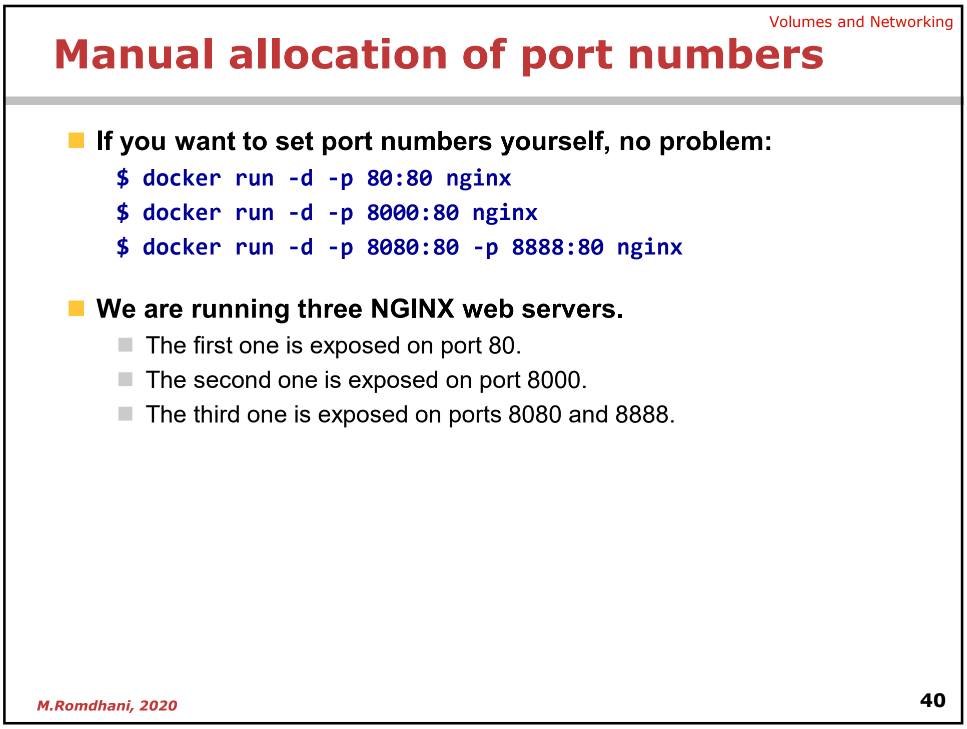
How does Docker know which port to map?

- There is metadata in the image telling "this image has something on port 80".
- We can see that metadata with docker inspect:

```
$ docker inspect --format '{{.Config.ExposedPorts}}' nginx map[80/tcp:{}]
```

- This metadata was set in the Dockerfile, with the EXPOSE keyword.
- We can see that with docker history:

```
$ docker history nginx
IMAGE            CREATED        CREATED BY
7f70b30f2cc6     11 days ago    /bin/sh -c #(nop)  CMD ["nginx" "-g" "…
<missing>        11 days ago    /bin/sh -c #(nop)  STOPSIGNAL [SIGTERM]
<missing>        11 days ago    /bin/sh -c #(nop)  EXPOSE 80/tcp
```

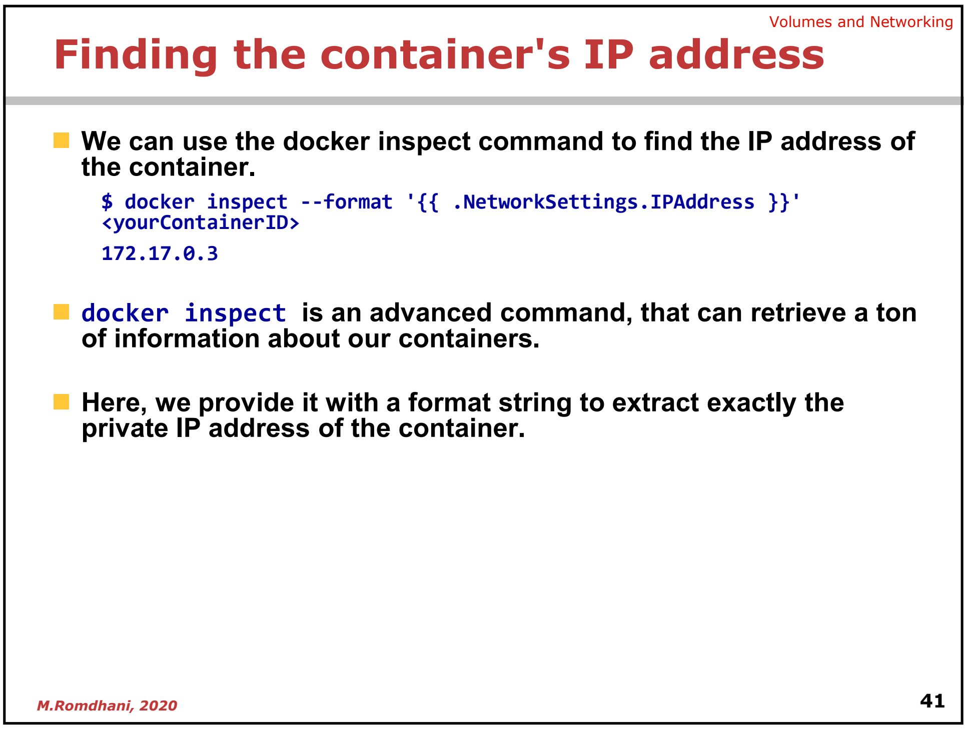


Manual allocation of port numbers

- If you want to set port numbers yourself, no problem:

```
$ docker run -d -p 80:80 nginx
$ docker run -d -p 8000:80 nginx
$ docker run -d -p 8080:80 -p 8888:80 nginx
```

- We are running three NGINX web servers.
  - The first one is exposed on port 80.
  - The second one is exposed on port 8000.
  - The third one is exposed on ports 8080 and 8888.

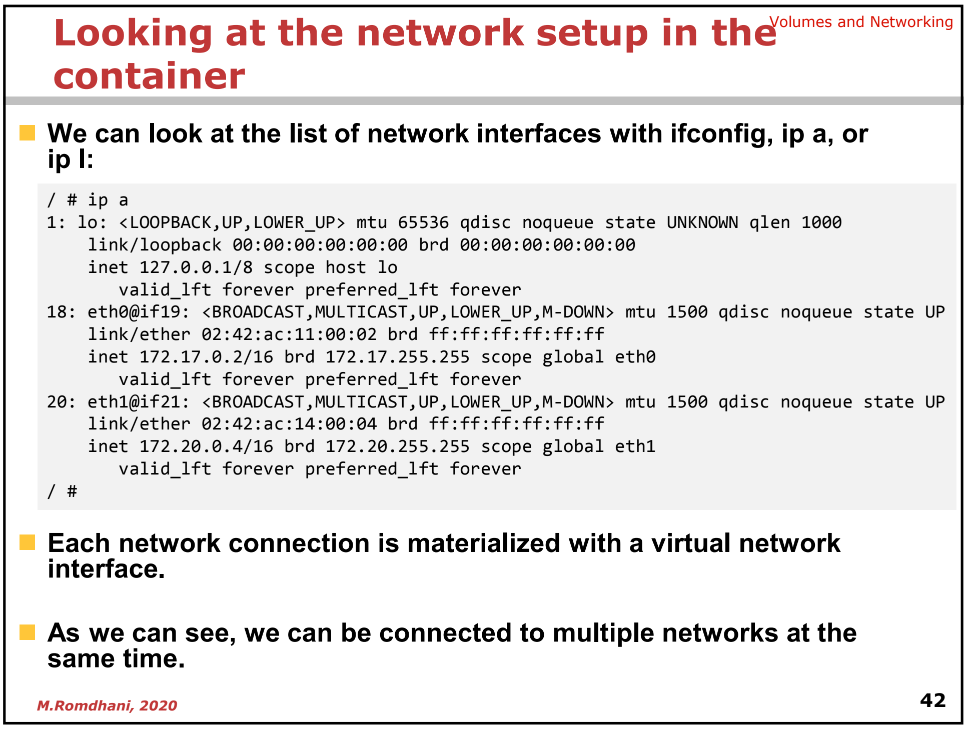


Finding the container's IP address

- We can use the docker inspect command to find the IP address of the container.

```
$ docker inspect --format '{{ .NetworkSettings.IPAddress }}'
<yourContainerID>
172.17.0.3
```

- `docker inspect` is an advanced command, that can retrieve a ton of information about our containers.
- Here, we provide it with a format string to extract exactly the private IP address of the container.



Looking at the network setup in the container

- We can look at the list of network interfaces with ifconfig, ip a, or ip l:

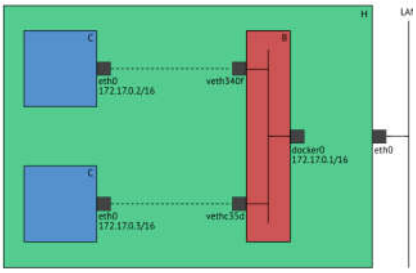
```
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
18: eth0@if19: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
20: eth1@if21: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:14:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.20.0.4/16 brd 172.20.255.255 scope global eth1
       valid_lft forever preferred_lft forever
/ #
```

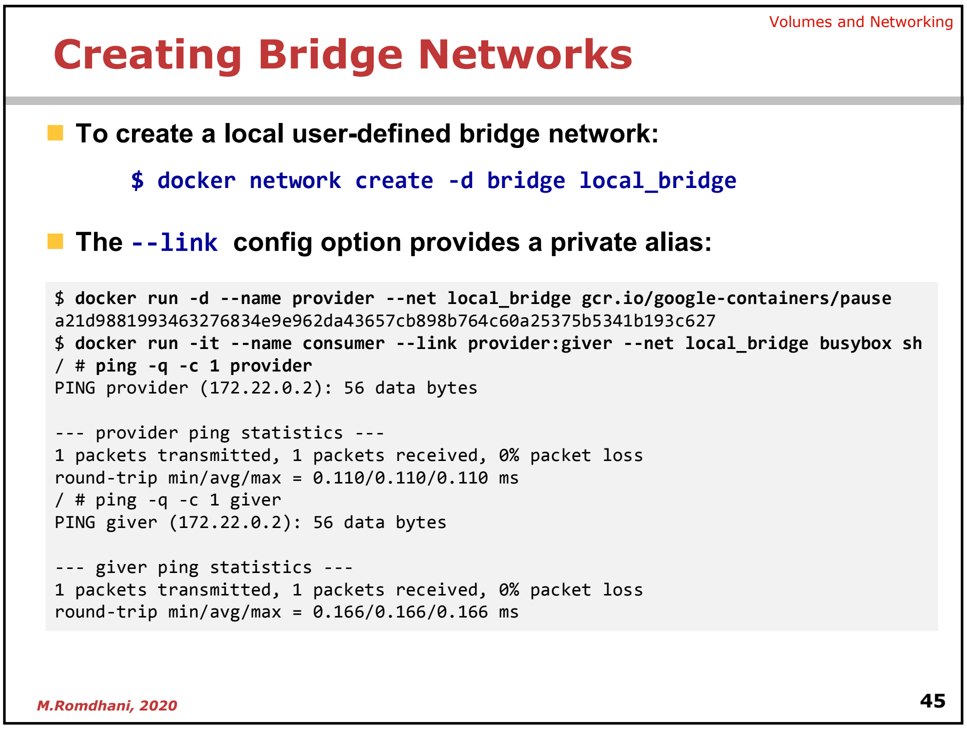
- Each network connection is materialized with a virtual network interface.
- As we can see, we can be connected to multiple networks at the same time.

Bridge vs Overlay networks

Bridge Networking

- **At start-up, the Docker daemon creates three default networks, called 'bridge', 'none', 'host', and 'Container'**
  - Bridge
    - By default, any container invoked will be connected to this bridge, with its network stack contained within its own NET namespace, but with its primary network interface connected to the bridge
  - None
    - With --net=none, the container is created with just a loopback interface to allow container-specific services to communicate
  - Host
    - The container shares the same NET namespace and network stack (interfaces, ports etc.) as the host
  - Container
    - The container shares the network stack of another specified container





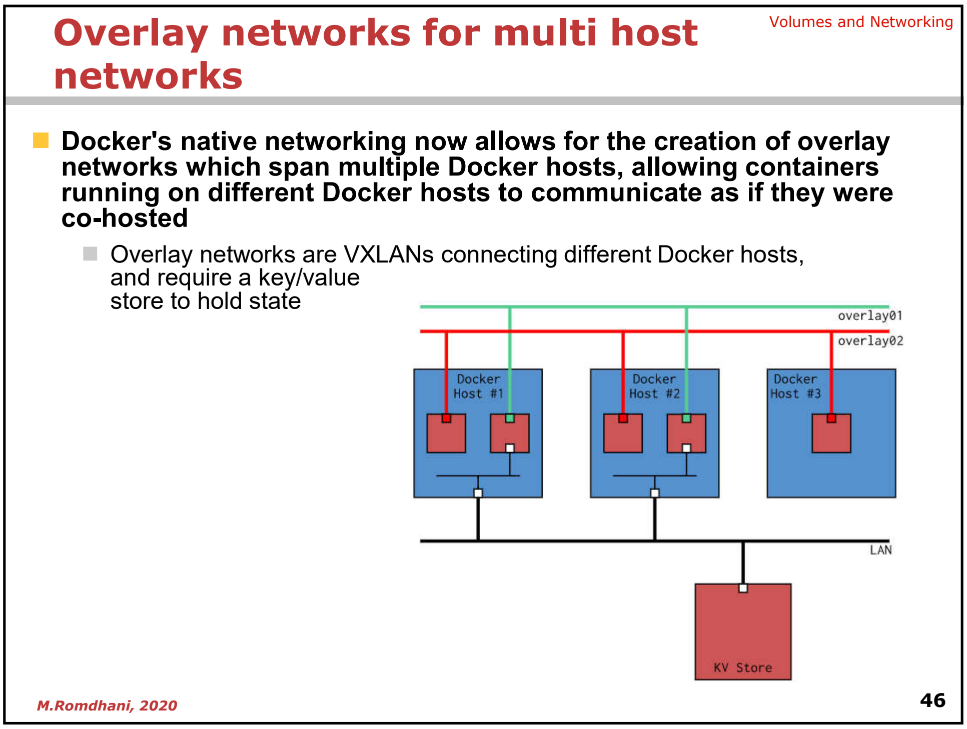
Creating Bridge Networks

- **To create a local user-defined bridge network:**

```
$ docker network create -d bridge local_bridge
```

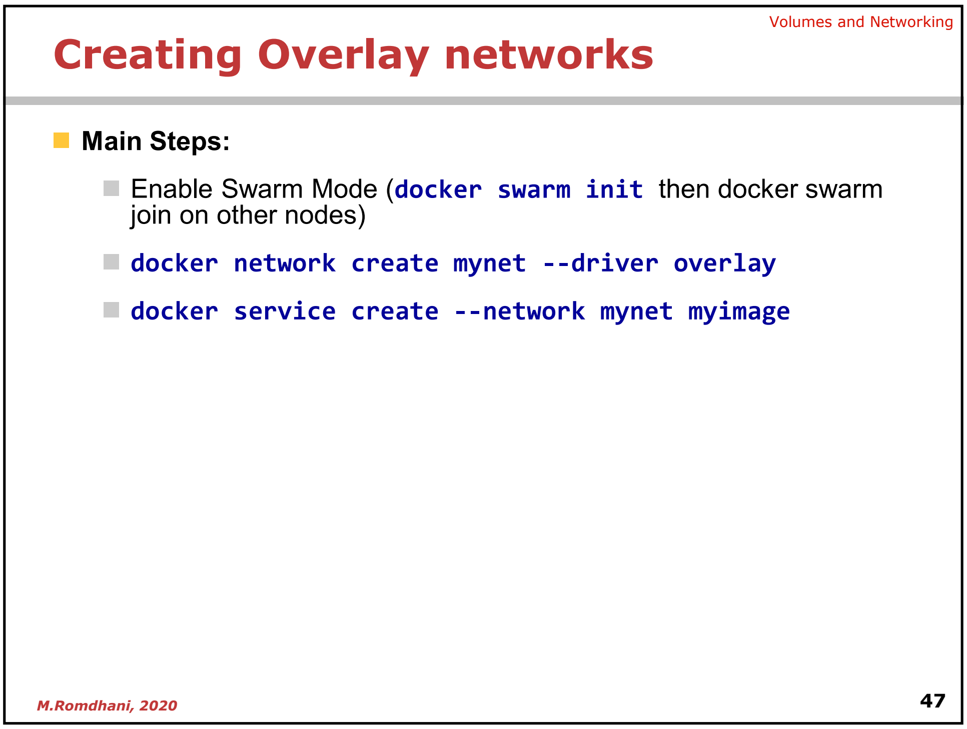
- **The `--link` config option provides a private alias:**

```
$ docker run -d --name provider --net local_bridge gcr.io/google-containers/pause
a21d9881993463276834e9e962da43657cb898b764c60a25375b5341b193c627
$ docker run -it --name consumer --link provider:giver --net local_bridge busybox sh
/ # ping -q -c 1 provider
PING provider (172.22.0.2): 56 data bytes

--- provider ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.110/0.110/0.110 ms
/ # ping -q -c 1 giver
PING giver (172.22.0.2): 56 data bytes

--- giver ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.166/0.166/0.166 ms
```



Overlay networks for multi host networks

- **Docker's native networking now allows for the creation of overlay networks which span multiple Docker hosts, allowing containers running on different Docker hosts to communicate as if they were co-hosted**
  - Overlay networks are VXLANs connecting different Docker hosts, and require a key/value store to hold state



Creating Overlay networks

- **Main Steps:**
  - Enable Swarm Mode (`docker swarm init` then docker swarm join on other nodes)
  - `docker network create mynet --driver overlay`
  - `docker service create --network mynet myimage`