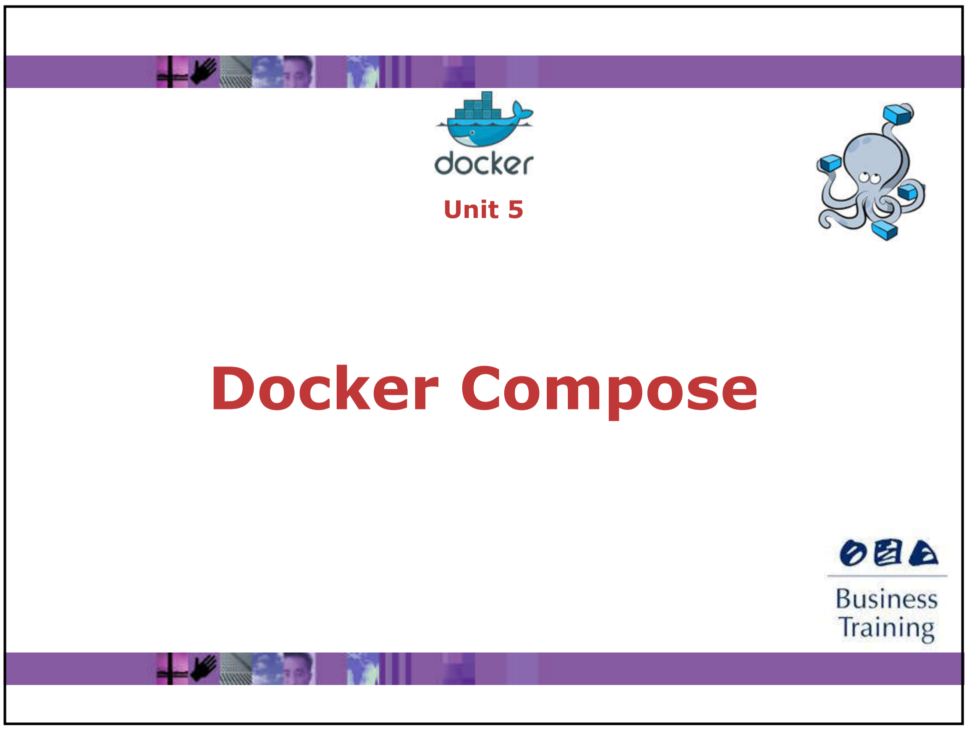


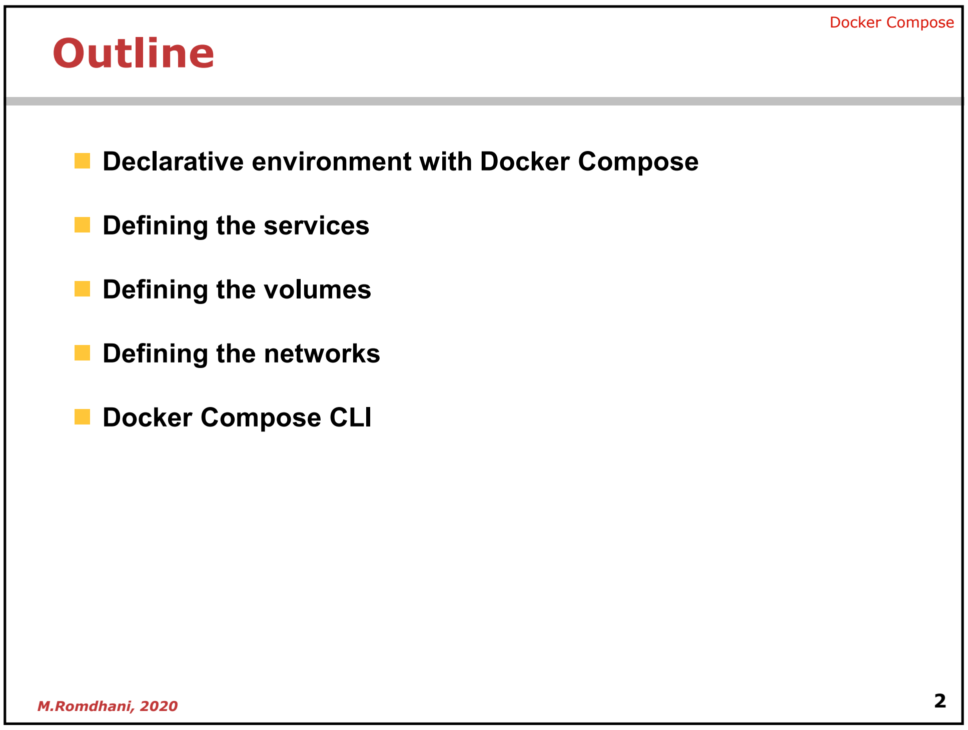
docker

Unit 5

Docker Compose

Business Training

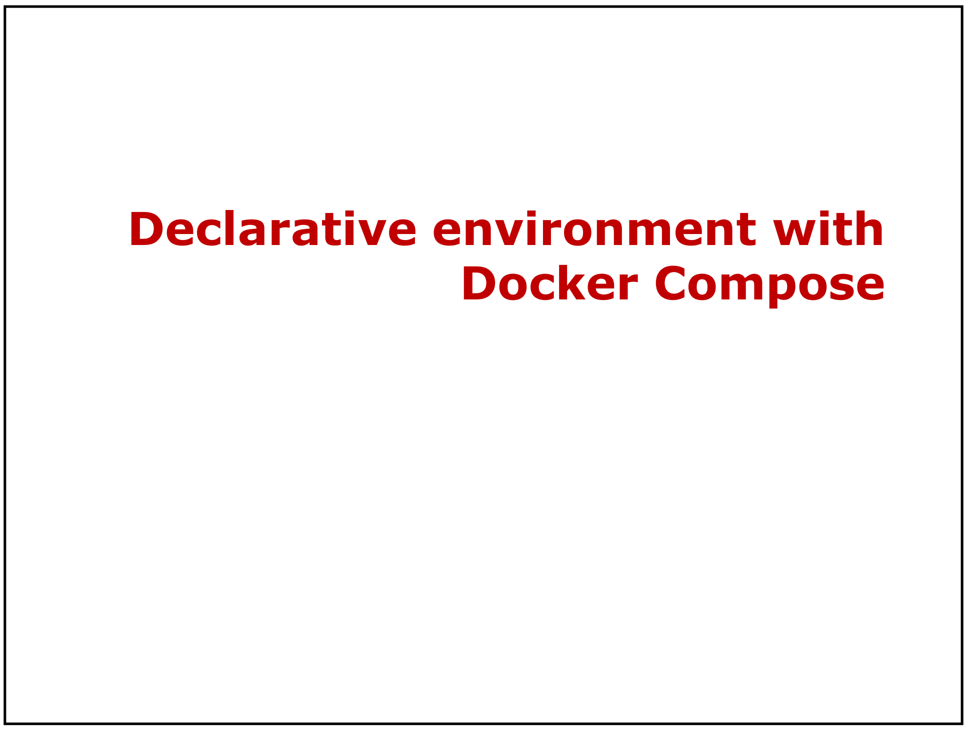
1



Outline

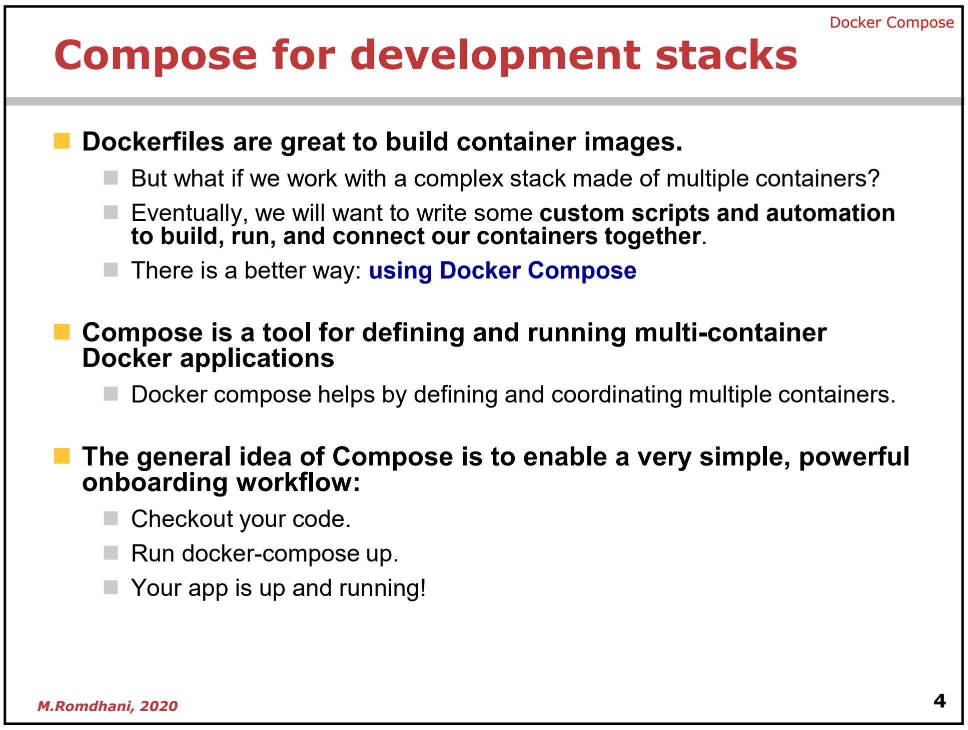
- Declarative environment with Docker Compose
- Defining the services
- Defining the volumes
- Defining the networks
- Docker Compose CLI

2



Declarative environment with Docker Compose

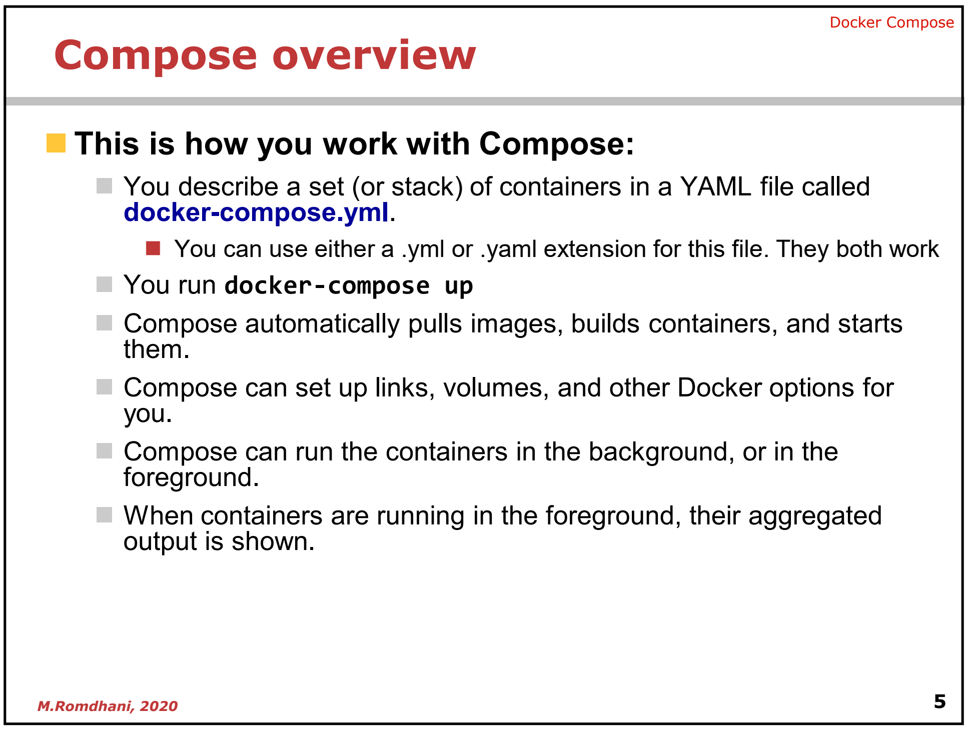
3



Compose for development stacks

- **Dockerfiles are great to build container images.**
  - But what if we work with a complex stack made of multiple containers?
  - Eventually, we will want to write some **custom scripts and automation to build, run, and connect our containers together**.
  - There is a better way: **using Docker Compose**
- **Compose is a tool for defining and running multi-container Docker applications**
  - Docker compose helps by defining and coordinating multiple containers.
- **The general idea of Compose is to enable a very simple, powerful onboarding workflow:**
  - Checkout your code.
  - Run docker-compose up.
  - Your app is up and running!

4



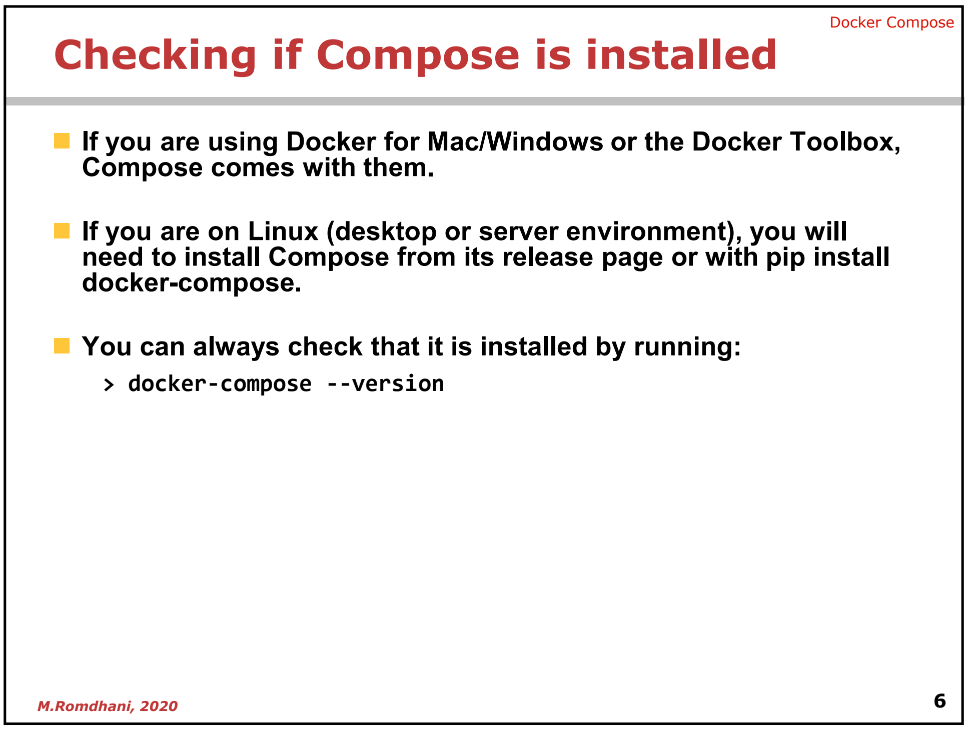
Docker Compose

Compose overview

- **This is how you work with Compose:**
  - You describe a set (or stack) of containers in a YAML file called **docker-compose.yml**.
    - You can use either a .yml or .yaml extension for this file. They both work
  - You run `docker-compose up`
  - Compose automatically pulls images, builds containers, and starts them.
  - Compose can set up links, volumes, and other Docker options for you.
  - Compose can run the containers in the background, or in the foreground.
  - When containers are running in the foreground, their aggregated output is shown.

M.Romdhani, 2020 5

5



Docker Compose

Checking if Compose is installed

- **If you are using Docker for Mac/Windows or the Docker Toolbox, Compose comes with them.**
- **If you are on Linux (desktop or server environment), you will need to install Compose from its release page or with pip install docker-compose.**
- **You can always check that it is installed by running:**

```
> docker-compose --version
```

M.Romdhani, 2020 6

6

Launching Our First Stack with Compose

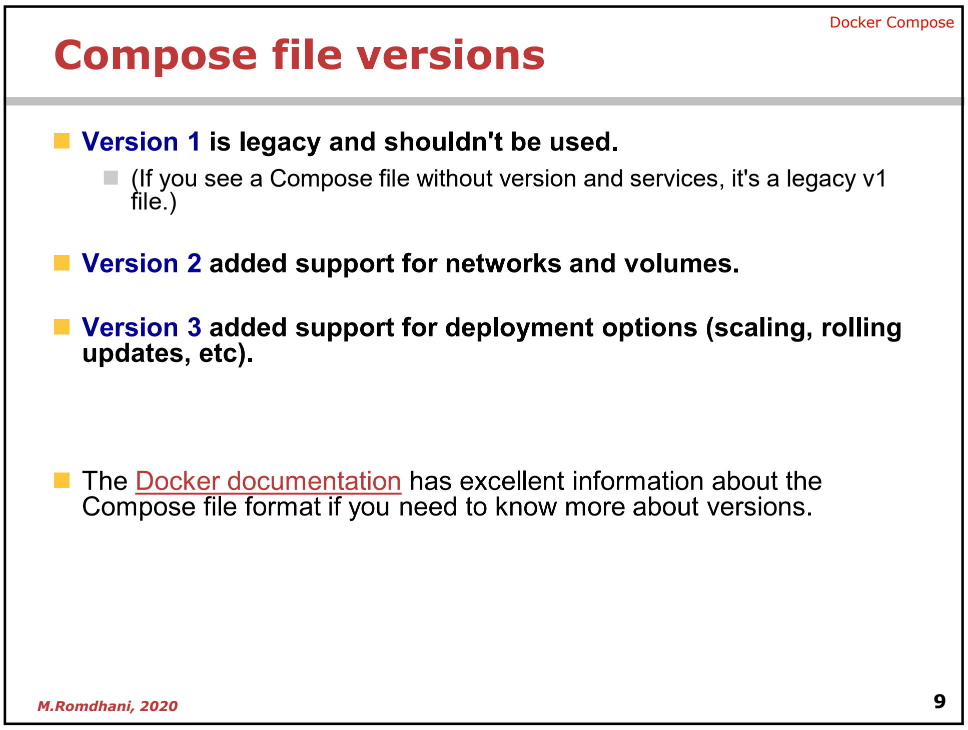
- This is an example of Docker Compose file

- To start the app:
  - `docker-compose up`

```
version: '3'
services:
  db:
     image: mysql
     container_name: mysql_db
     restart: always
     environment:
        - MYSQL_ROOT_PASSWORD="secret"
  web:
    image: apache
    build: ./webapp
    depends_on:
       - db
    container_name: apache_web
    restart: always
    ports:
      - "8080:80"
```

Compose file structure

- **version is mandatory.**
  - We should use "2" or later; version 1 is deprecated.
- **services is mandatory.**
  - A service is one or more replicas of the same image running as containers.
- **networks is optional and indicates to which networks containers should be connected.**
  - By default, containers will be connected on a private, per-compose-file network.
- **volumes is optional and can define volumes to be used and/or shared by the containers.**

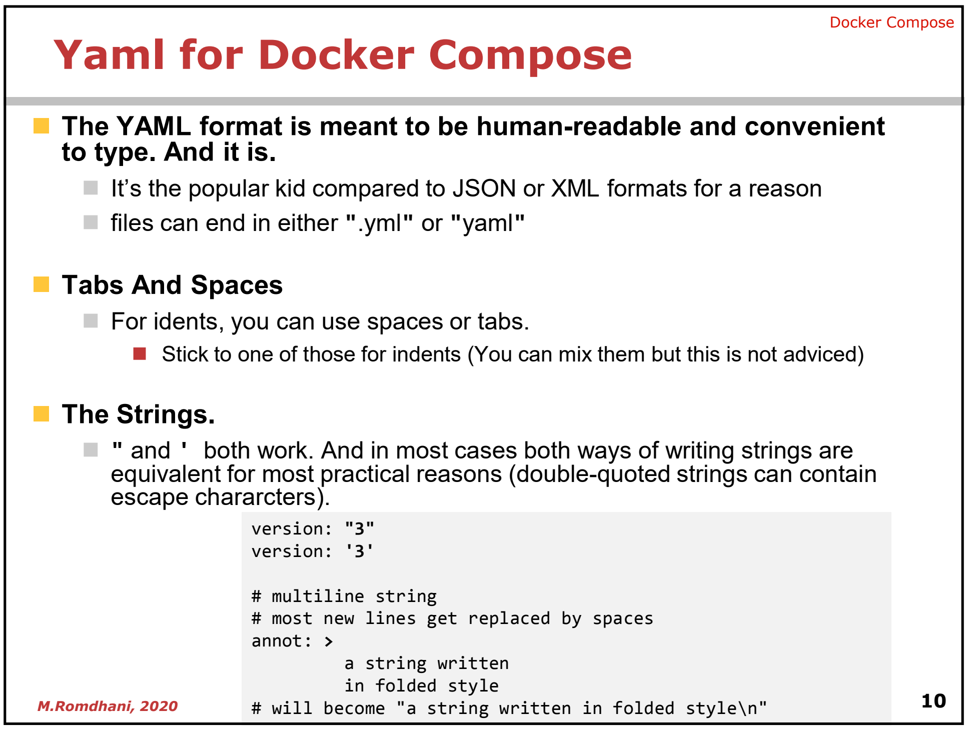


Compose file versions

- **Version 1 is legacy and shouldn't be used.**
  - (If you see a Compose file without version and services, it's a legacy v1 file.)
- **Version 2 added support for networks and volumes.**
- **Version 3 added support for deployment options (scaling, rolling updates, etc).**
- The Docker documentation has excellent information about the Compose file format if you need to know more about versions.

M.Romdhani, 2020 9

9



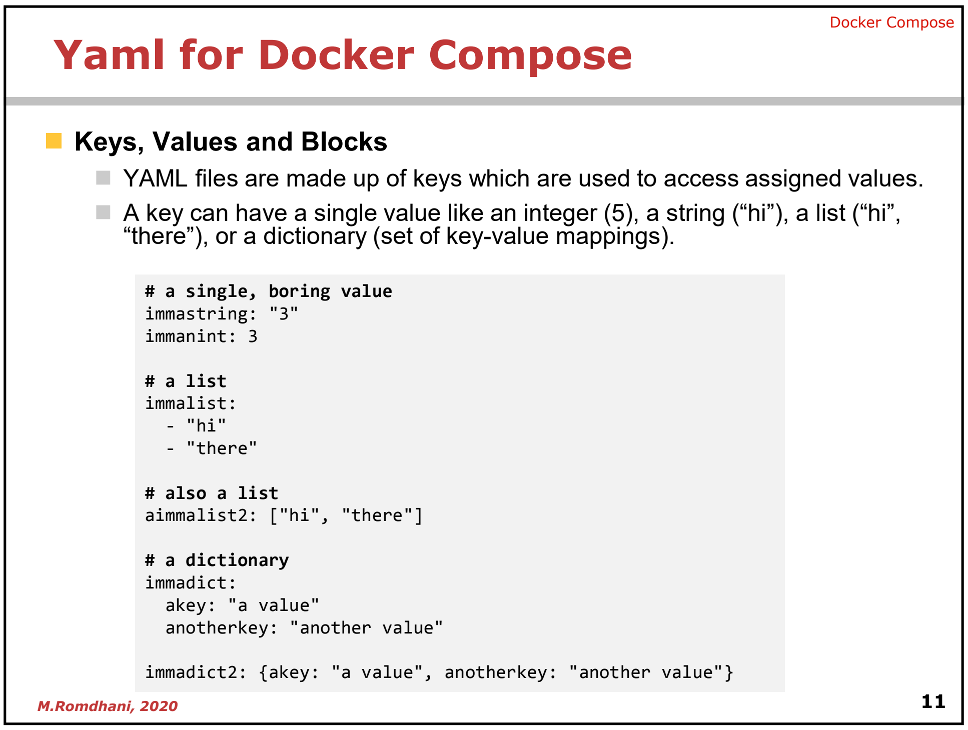
Yaml for Docker Compose

- **The YAML format is meant to be human-readable and convenient to type. And it is.**
  - It’s the popular kid compared to JSON or XML formats for a reason
  - files can end in either ".yml" or "yaml"
- **Tabs And Spaces**
  - For idents, you can use spaces or tabs.
    - Stick to one of those for indents (You can mix them but this is not adviced)
- **The Strings.**
  - " and ' both work. And in most cases both ways of writing strings are equivalent for most practical reasons (double-quoted strings can contain escape chararcters).

```
version: "3"
version: '3'

# multiline string
# most new lines get replaced by spaces
annot: >
         a string written
         in folded style
# will become "a string written in folded style\n"
```

M.Romdhani, 2020 10

10



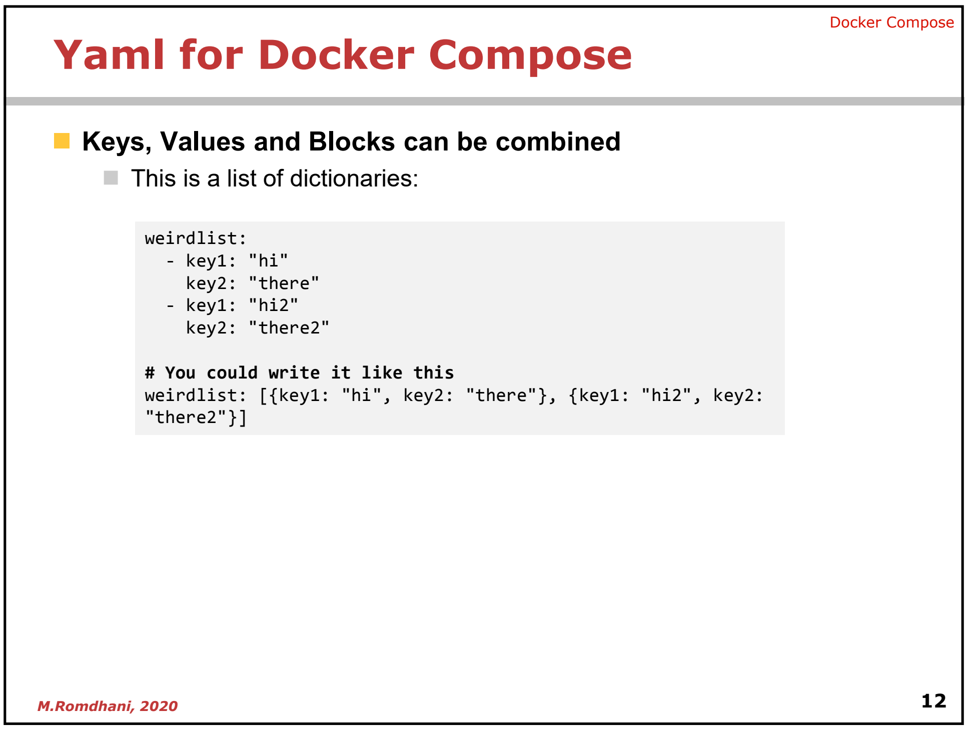
Yaml for Docker Compose

- **Keys, Values and Blocks**
  - YAML files are made up of keys which are used to access assigned values.
  - A key can have a single value like an integer (5), a string (“hi”), a list (“hi”, “there”), or a dictionary (set of key-value mappings).

```
# a single, boring value
immastring: "3"
immanint: 3

# a list
immalist:
  - "hi"
  - "there"

# also a list
aimmalist2: ["hi", "there"]

# a dictionary
immadict:
  akey: "a value"
  anotherkey: "another value"

immadict2: {akey: "a value", anotherkey: "another value"}
```

M.Romdhani, 2020 11

11



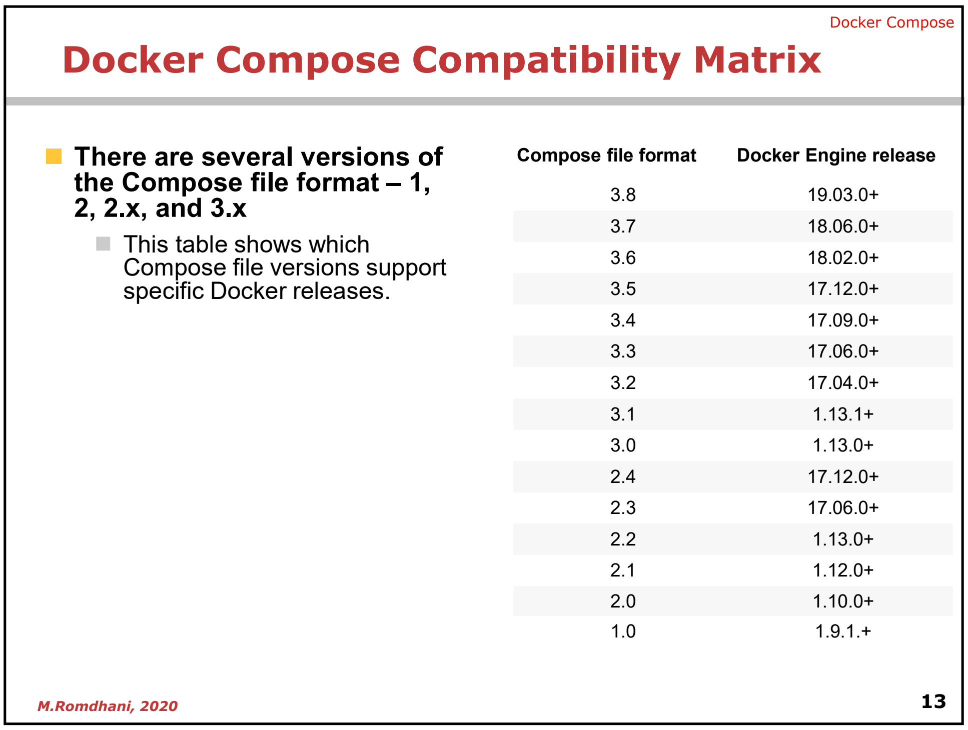
Yaml for Docker Compose

- **Keys, Values and Blocks can be combined**
  - This is a list of dictionaries:

```
weirdlist:
  - key1: "hi"
    key2: "there"
  - key1: "hi2"
    key2: "there2"

# You could write it like this
weirdlist: [{key1: "hi", key2: "there"}, {key1: "hi2", key2:
"there2"}]
```

M.Romdhani, 2020 12

12

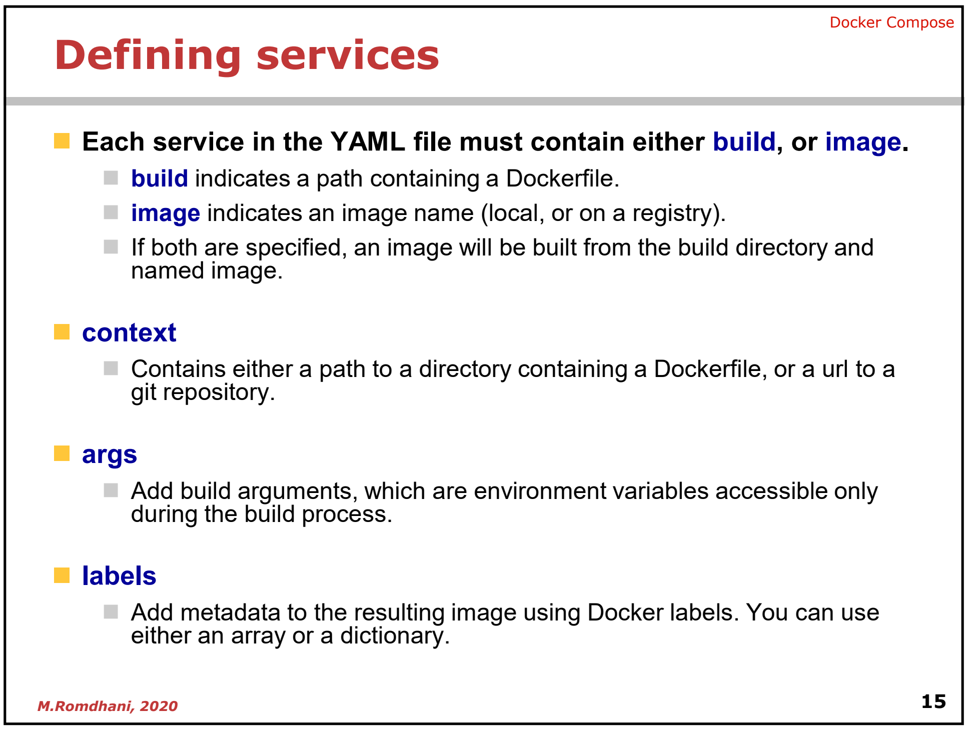


Docker Compose Compatibility Matrix

- **There are several versions of the Compose file format – 1, 2, 2.x, and 3.x**
  - This table shows which Compose file versions support specific Docker releases.

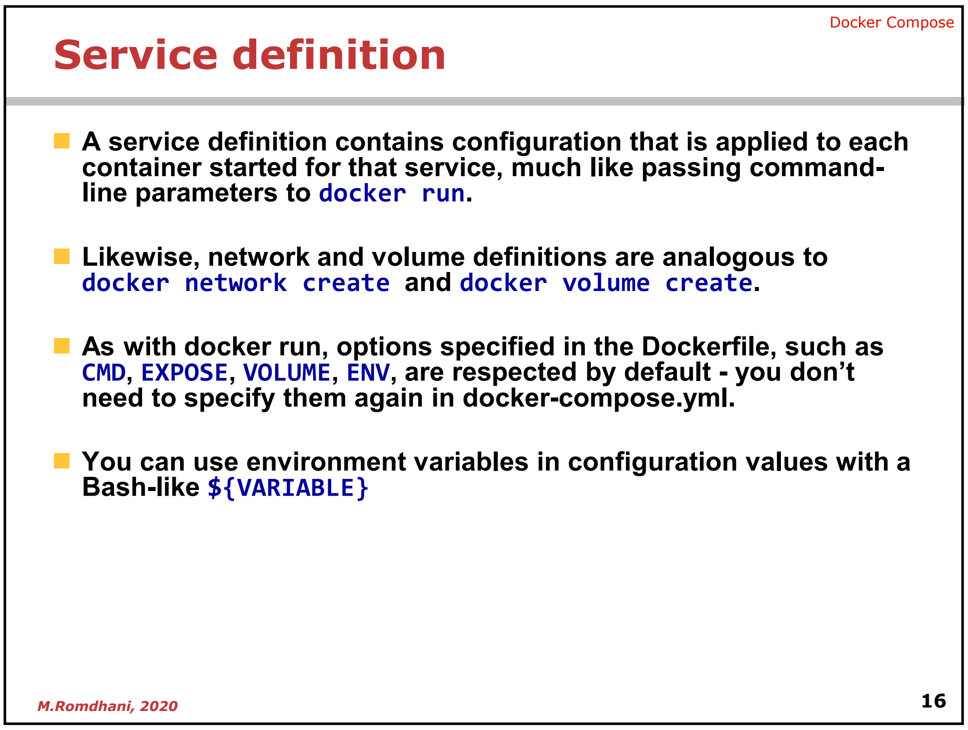
Compose file format	Docker Engine release
3.8	19.03.0+
3.7	18.06.0+
3.6	18.02.0+
3.5	17.12.0+
3.4	17.09.0+
3.3	17.06.0+
3.2	17.04.0+
3.1	1.13.1+
3.0	1.13.0+
2.4	17.12.0+
2.3	17.06.0+
2.2	1.13.0+
2.1	1.12.0+
2.0	1.10.0+
1.0	1.9.1.+

Defining the services



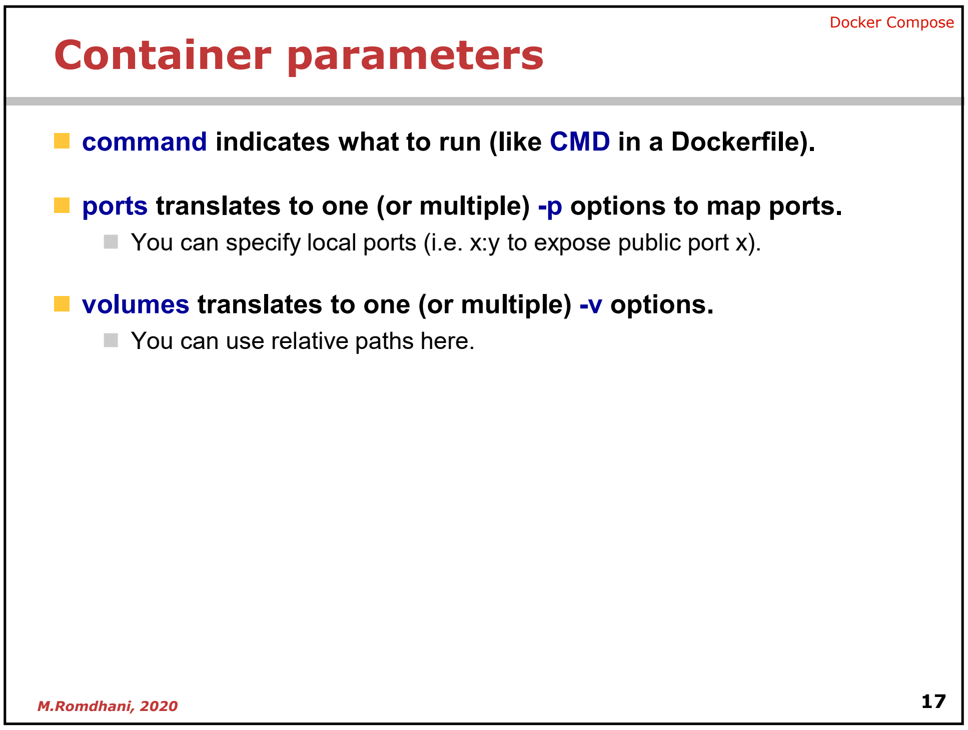
Defining services

- **Each service in the YAML file must contain either build, or image.**
  - build indicates a path containing a Dockerfile.
  - image indicates an image name (local, or on a registry).
  - If both are specified, an image will be built from the build directory and named image.
- **context**
  - Contains either a path to a directory containing a Dockerfile, or a url to a git repository.
- **args**
  - Add build arguments, which are environment variables accessible only during the build process.
- **labels**
  - Add metadata to the resulting image using Docker labels. You can use either an array or a dictionary.



Service definition

- **A service definition contains configuration that is applied to each container started for that service, much like passing command-line parameters to `docker run`.**
- **Likewise, network and volume definitions are analogous to `docker network create` and `docker volume create`.**
- **As with docker run, options specified in the Dockerfile, such as `CMD`, `EXPOSE`, `VOLUME`, `ENV`, are respected by default - you don't need to specify them again in docker-compose.yml.**
- **You can use environment variables in configuration values with a Bash-like `${VARIABLE}`**



Docker Compose

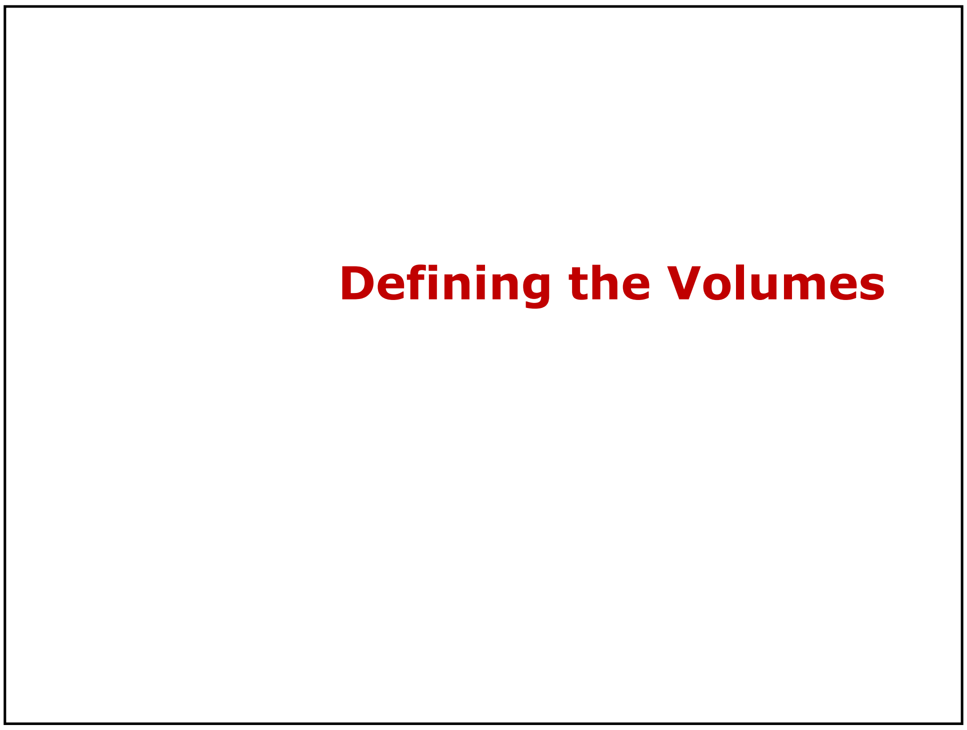
Container parameters

- **command indicates what to run (like CMD in a Dockerfile).**
- **ports translates to one (or multiple) -p options to map ports.**
  - You can specify local ports (i.e. x:y to expose public port x).
- **volumes translates to one (or multiple) -v options.**
  - You can use relative paths here.

M.Romdhani, 2020

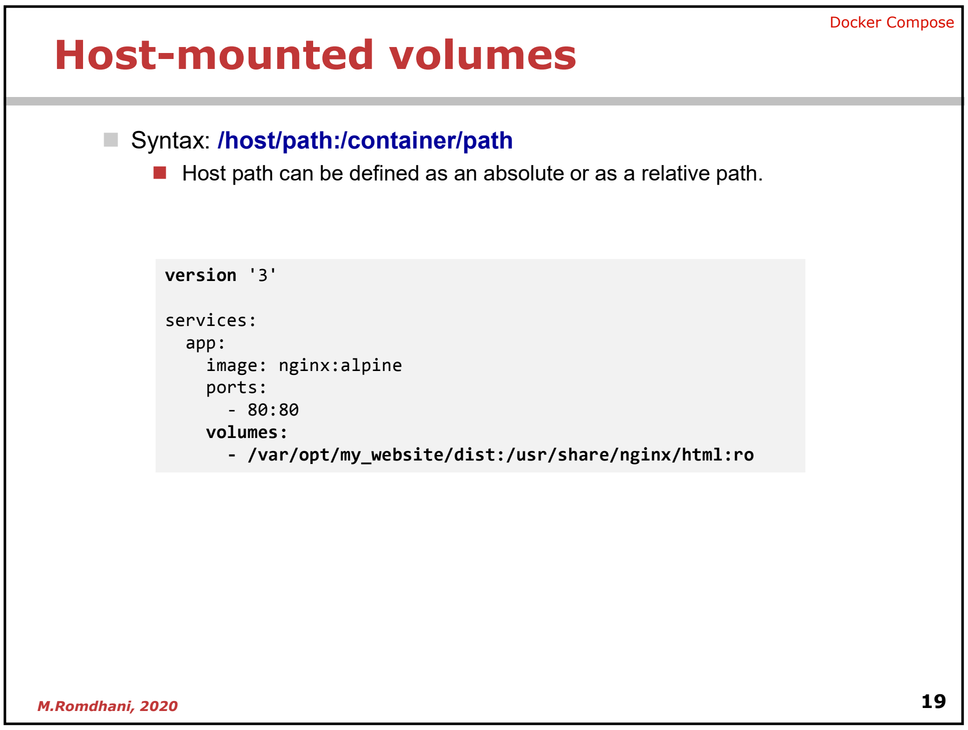
17

17



Defining the Volumes

18



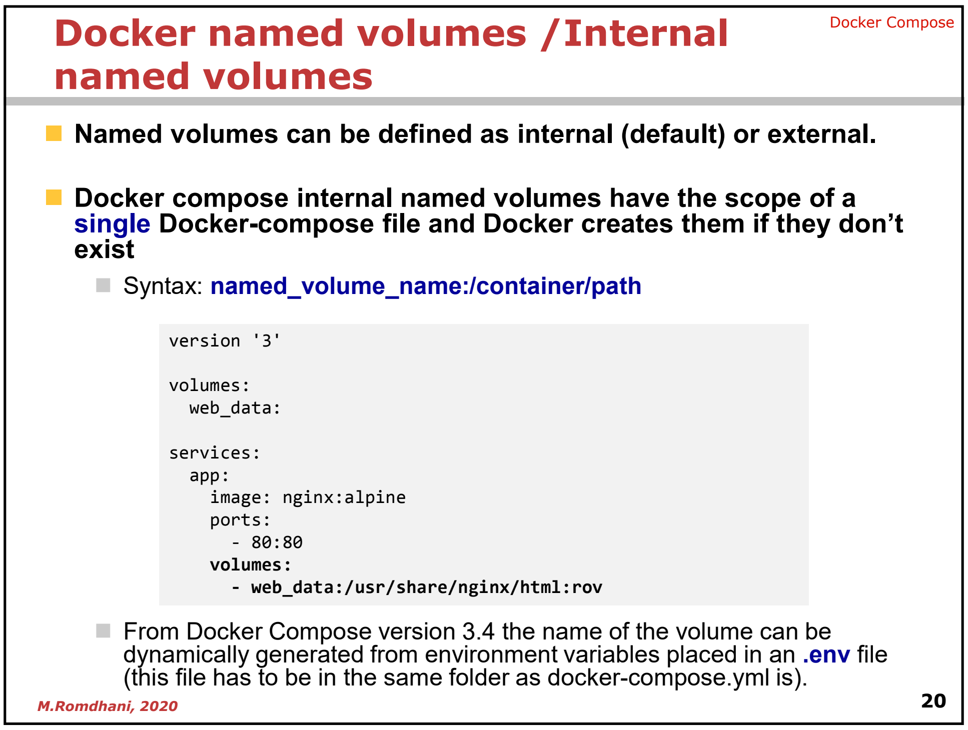
Host-mounted volumes

- Syntax: **/host/path:/container/path**
  - Host path can be defined as an absolute or as a relative path.

```
version '3'

services:
  app:
    image: nginx:alpine
    ports:
      - 80:80
    volumes:
      - /var/opt/my_website/dist:/usr/share/nginx/html:ro
```

M.Romdhani, 2020 19

19



Docker named volumes /Internal named volumes

- **Named volumes can be defined as internal (default) or external.**
- **Docker compose internal named volumes have the scope of a single Docker-compose file and Docker creates them if they don't exist**
  - Syntax: **named_volume_name:/container/path**

```
version '3'

volumes:
  web_data:

services:
  app:
    image: nginx:alpine
    ports:
      - 80:80
    volumes:
      - web_data:/usr/share/nginx/html:rov
```

  - From Docker Compose version 3.4 the name of the volume can be dynamically generated from environment variables placed in an **.env** file (this file has to be in the same folder as docker-compose.yml is).

M.Romdhani, 2020 20

20

Docker named volumes / External named volumes

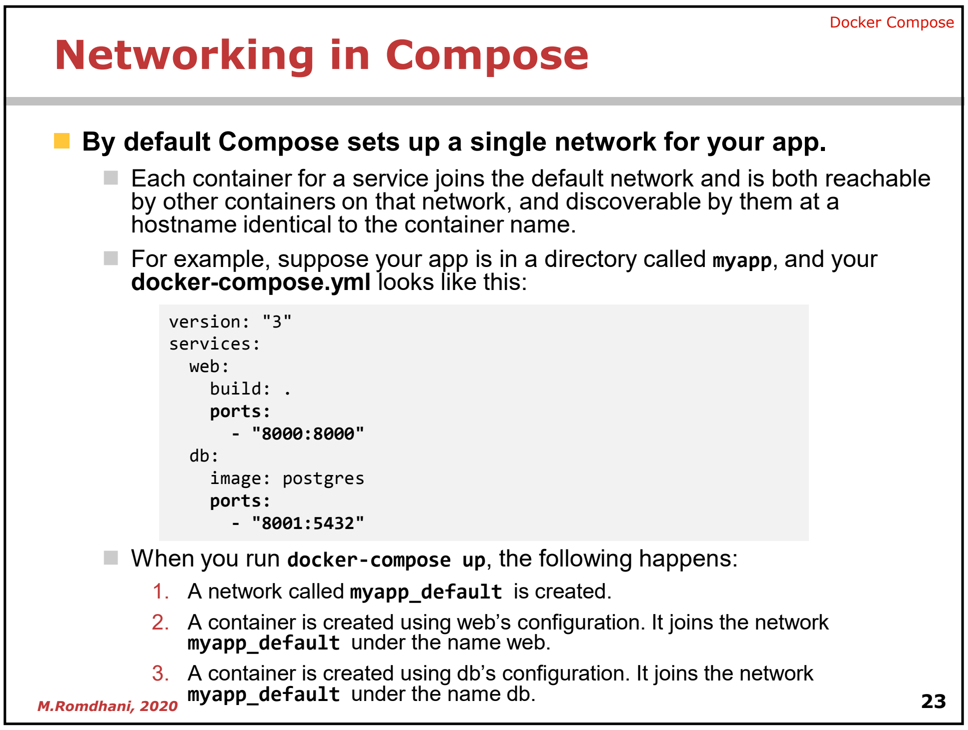
- **Docker compose external named volumes can be used across the Docker installation and they need to be created by the user (otherwise fails) using the docker volume create command.**
  - The same syntax in yaml as the internal volumes
- **Example:**
  - Defines web_data volume:

```
docker volume create --driver local \
    --opt type=none \
    --opt device=/var/opt/my_website/dist \
    --opt o=bind web_data
```

  - docker-compose.yml

```
version '3'
volumes:
  web_data:
    external: true
services:
  app:
    image: nginx:alpine
    ports:
      - 80:80
    volumes:
      - web_data:/usr/share/nginx/html:ro
```

Defining the networks



Networking in Compose

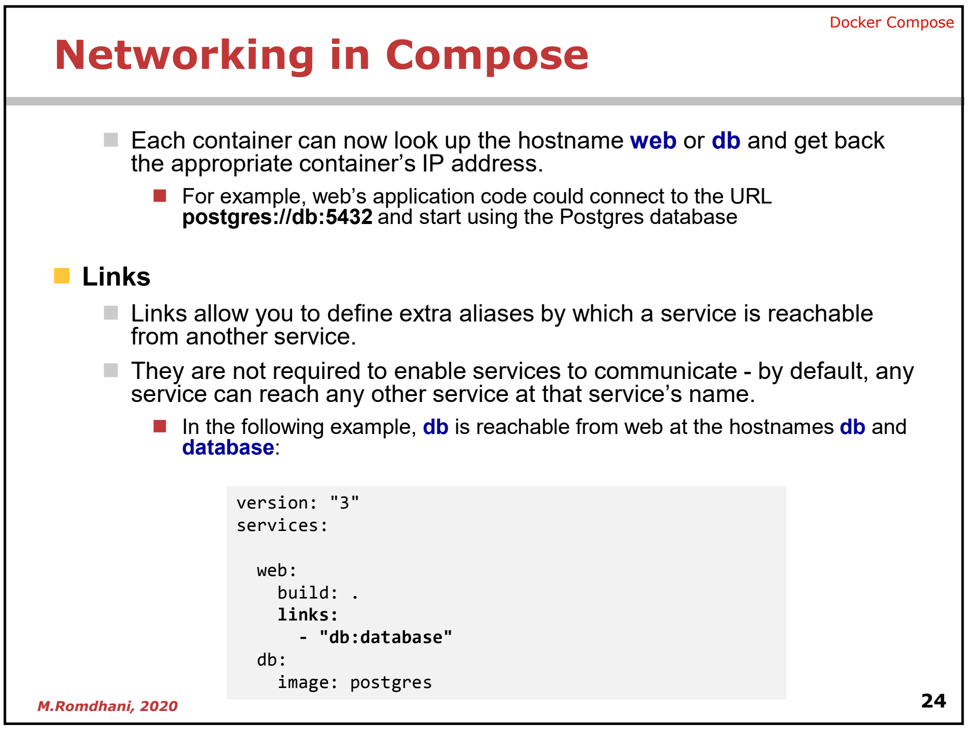
- **By default Compose sets up a single network for your app.**
  - Each container for a service joins the default network and is both reachable by other containers on that network, and discoverable by them at a hostname identical to the container name.
  - For example, suppose your app is in a directory called `myapp`, and your **docker-compose.yml** looks like this:

```
version: "3"
services:
  web:
    build: .
    ports:
      - "8000:8000"
  db:
    image: postgres
    ports:
      - "8001:5432"
```

  - When you run `docker-compose up`, the following happens:
    1. A network called `myapp_default` is created.
    2. A container is created using web's configuration. It joins the network `myapp_default` under the name web.
    3. A container is created using db's configuration. It joins the network `myapp_default` under the name db.

M.Romdhani, 2020 **23**

23



Networking in Compose

  - Each container can now look up the hostname **web** or **db** and get back the appropriate container's IP address.
    - For example, web's application code could connect to the URL **postgres://db:5432** and start using the Postgres database
- **Links**
  - Links allow you to define extra aliases by which a service is reachable from another service.
  - They are not required to enable services to communicate - by default, any service can reach any other service at that service's name.
    - In the following example, **db** is reachable from web at the hostnames **db** and **database**:

```
version: "3"
services:

  web:
    build: .
    links:
      - "db:database"
  db:
    image: postgres
```

M.Romdhani, 2020 **24**

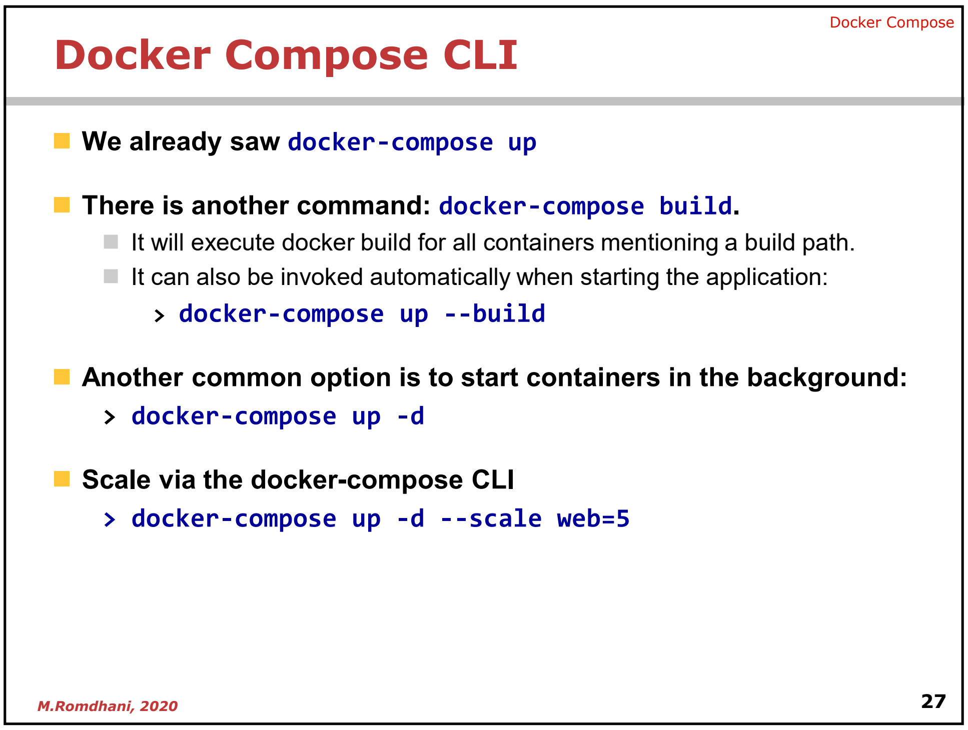
24

Specify custom networks

- **Instead of just using the default app network, you can specify your own networks with the top-level networks key.**
  - This lets you create more complex topologies and specify custom network **drivers** and **options**.
  - Each service can specify what networks to connect to with the service-level networks key, which is a list of names referencing entries under the top-level networks key.

```
version: "3"
services:
  proxy:
    build: ./proxy
    networks:
      - frontend
  app:
    build: ./app
    networks:
      - frontend
      - backend
  db:
    image: postgres
    networks:
      - backend
networks:
  frontend:
    # Use a custom driver
    driver: custom-driver-1
  backend:
    # Use a custom driver which takes special options
    driver: custom-driver-2
    driver_opts:
      foo: "1"
      bar: "2"
```

Docker Compose CLI



Docker Compose CLI

- **We already saw `docker-compose up`**
- **There is another command: `docker-compose build`.**
  - It will execute docker build for all containers mentioning a build path.
  - It can also be invoked automatically when starting the application:

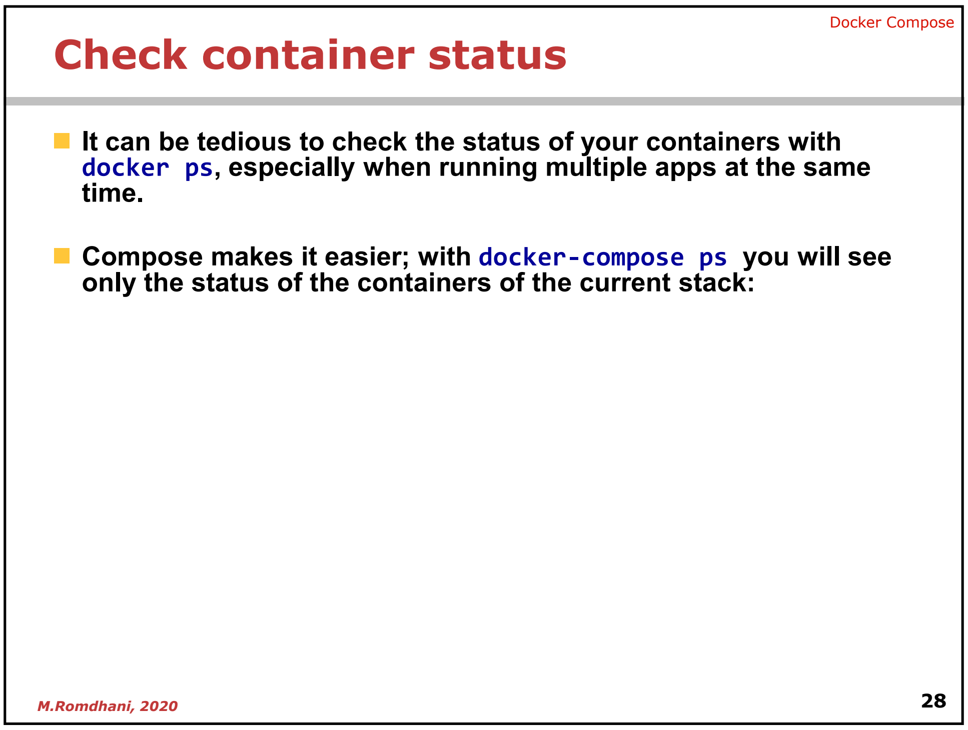
```
> docker-compose up --build
```

- **Another common option is to start containers in the background:**

```
> docker-compose up -d
```

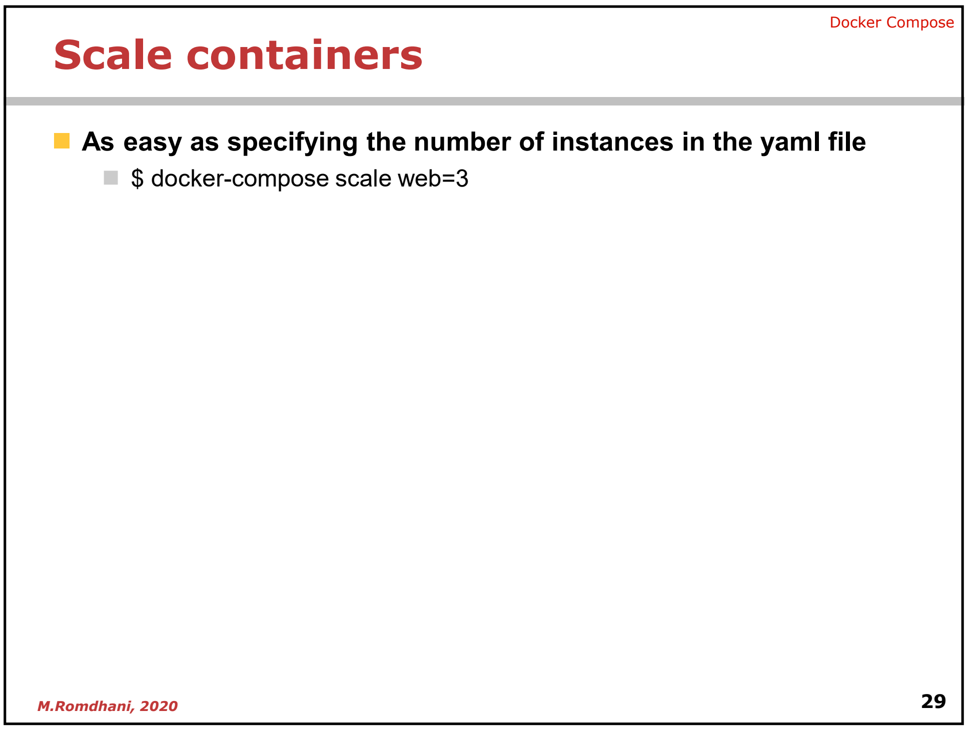
- **Scale via the docker-compose CLI**

```
> docker-compose up -d --scale web=5
```



Check container status

- **It can be tedious to check the status of your containers with `docker ps`, especially when running multiple apps at the same time.**
- **Compose makes it easier; with `docker-compose ps` you will see only the status of the containers of the current stack:**



Docker Compose

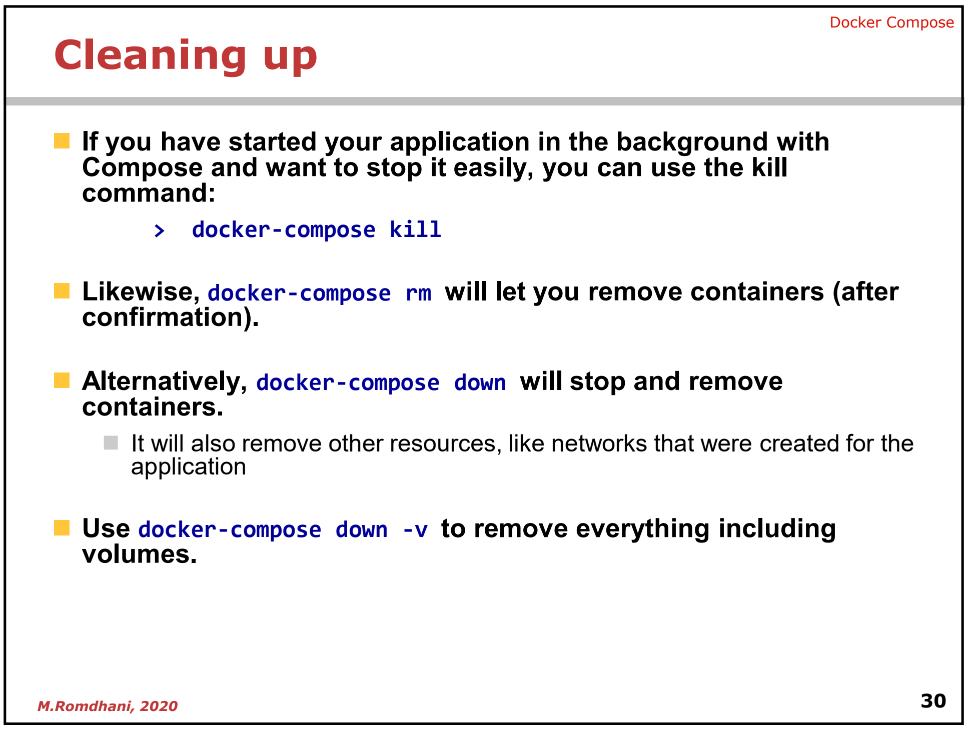
Scale containers

- **As easy as specifying the number of instances in the yaml file**
  - $ docker-compose scale web=3

M.Romdhani, 2020

29

29



Docker Compose

Cleaning up

- **If you have started your application in the background with Compose and want to stop it easily, you can use the kill command:**

```
> docker-compose kill
```

- **Likewise, `docker-compose rm` will let you remove containers (after confirmation).**
- **Alternatively, `docker-compose down` will stop and remove containers.**
  - It will also remove other resources, like networks that were created for the application
- **Use `docker-compose down -v` to remove everything including volumes.**

M.Romdhani, 2020

30

30