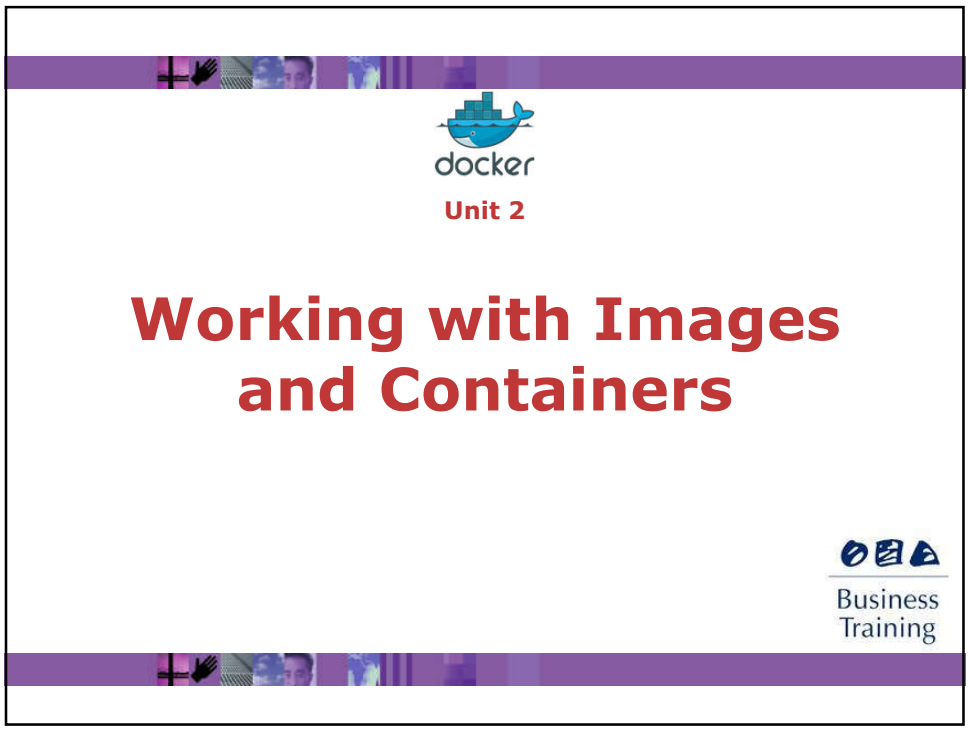


1

Outline

- Structure of a Docker image
- Images and repositories
- Managing images
- Launching containers with Docker Run
- Interacting with containers

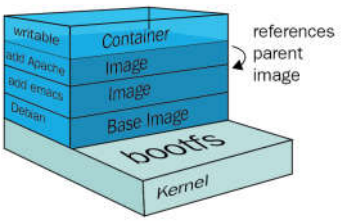
2

Structure of a Docker image

3

What is an image?

- **An image consists of a number of layers that are combined into a single virtual filesystem accessible for Docker applications. This is achieved by using a special technique which combines multiple layers into a single view.**
  - Image = files + metadata
  - The metadata can indicate a number of things, e.g.:
    - the author of the image
    - the command to execute in the container when starting it
    - environment variables to be set
    - etc.
- **Images are made of layers, conceptually stacked on top of each other.**
- **Each layer can add, change, and remove files and/or metadata.**
- **Images can share layers to optimize disk usage, transfer times, and memory use.**



4

Working with images and containers

Docker storage drivers

- **A Docker storage driver is the main component to enable and manage container images.**
  - Two main technologies are used for that:
    - Copy-on-write and,
    - Stackable image layers.

Storage driver category	Storage drivers
Union filesystems	AUFS, Overlay, Overlay2
Snapshotting filesystems	Btrfs, ZFS
Copy-on-write block devices	Devicemapper

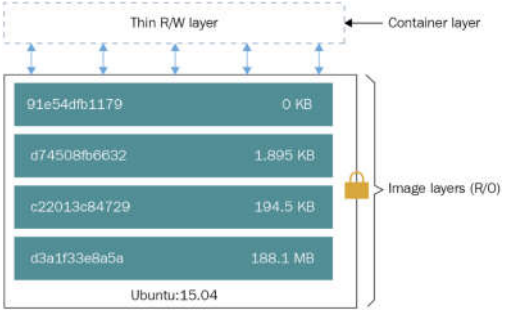
M.Romdhani, 2020 5

5

Working with images and containers

Container image layers

- **As previously mentioned, a Docker image contains a number of layers that are combined into a single filesystem using a storage driver.**
- **The layers (also called intermediate images) are generated when commands are executed during the Docker image build process.**
  - Usually, Docker images are created using a Dockerfile, the syntax of which will be described later. Each layer represents an instruction in the image's Dockerfile.
- **Each layer, except thevery last one, is read-only.**



M.Romdhani, 2020 6

6

Example for a Java webapp

- **Layers**
  - Ubuntu base layer
  - Packages and configuration files added by our local IT
  - JRE
  - Tomcat
  - Our application's dependencies
  - Our application code and assets
  - Our application configuration

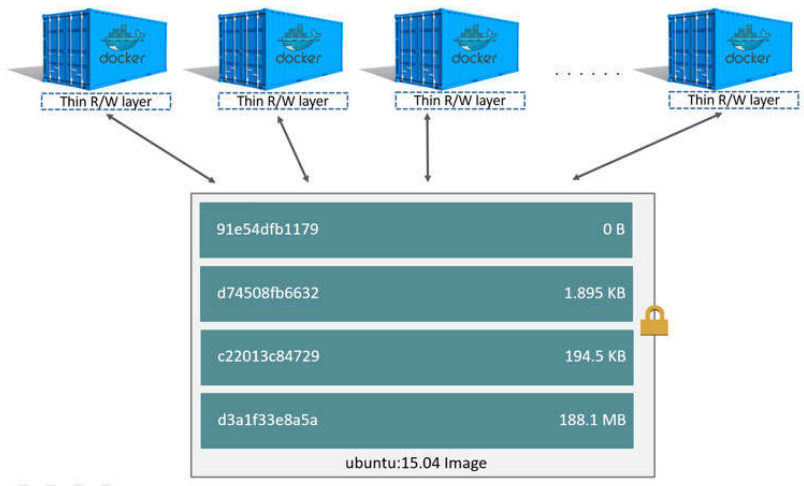
7

Differences between containers and images

- **Docker Image**
  - An **immutable template** for containers
  - Can be pulled and pushed towards a registry
  - Image names have the form **[registry/][user/]name[:tag]**
  - The default for the tag is latest
- **Docker Container**
  - A running instance instance of an image
  - Can be started, stopped, restarted, …
  - Maintains changes within the filesystems
  - New image can be created from current container state

8

Multiple containers sharing the same image



9

Images namespaces

- There are three namespaces:
  1. Official images
     - e.g. `ubuntu, busybox ...`
  2. User (and organizations) images
     - e.g. `mromdhani/myapp`
  3. Self-hosted images
     - e.g. `registry.example.com:5000/my-private/image`

10

Images and repositories

11

Docker registries

- **A registry is a storage and content delivery system, holding named Docker images, available in different tagged versions.**
- **Docker supports several types of docker registry:**
  - Public registry
  - Private registry
- **Docker Hub is the public registry managed by the Docker project.**
  - It hosts a large set of container images, including those provided by major open source projects, such as MySQL, Nginx, Apache, and so on, as well as customized container images developed by the community.

12

search, pull & push images

- Searching in the registry:
  - `$ docker search <term>`
- Download or update an image from the registry:
  - `$ docker pull <image>`
- Upload an image to the registry:
  - `$ docker push <image>`

13

Managing images

14

Working with images and containers

Listing local images

- To list current images use `docker image ls` (or `docker images`)

```
$ docker image ls    # Long form
$ docker images ls   # short form, legacy style
```

- To filter images by criteria use the --filter (or -f) option

```
$ docker images --filter "dangling=true "
```

  - Dangling images are untagged images that have no relationship to any tagged images

- To customize the listing format use –format option

```
docker images --format "{{.Repository}}: {{.Tag}}: {{.Size}}
```


15

Working with images and containers

Inspecting images

- Inspect the nginx image using the following command

```
$ docker inspect nginx
```

  - The output can be overwhelming.

- Filter the tags and hash of the nginx image using --format (or -f) option.

```
$ docker  inspect -f "{{.RepoTags}} {{.Config.Image}} " nginx
```

- Filter the tags and hash of the nginx image using the jq tool

```
$  inspect nginx | jq ".[]| (.RepoTags, .Config.Image)"
```


16

Removing Docker images

- To remove images use `docker image rm` (or `docker rmi`)

```
$ docker image rm    # Long form
$ docker rmi   # short form, legacy style
```

- Remove Dangling images, using `docker rmi` or `docker prune`

```
$ docker rmi -f $(docker images -f "dangling=true" -q)
$ docker image prune
```


17

Save and loading Docker images

- **You can use save/load process to move an image from one host to the other.**
  - This is similar to pushing and pulling from a repository but without the overhead of uploading and downloading the image.
- **Saving an image as a tar archive**

```
$ docker save my-alpine-with-curl:latest -o alpine-saved.tar
```

- **Loading an image from tar archive**

```
$ docker load -i  alpine-saved.tar
```


18

Working with images and containers

Viewing the history of an image

- To view the history of an image use the **docker history** command.
- The history command lists all the layers composing an image.
- For each layer, it shows its creation time, size, and creation command.
- When an image was built with a Dockerfile, each layer corresponds to a line of the Dockerfile.
- Example

```
$ docker history my-alpine-with-curl
IMAGE          CREATED        CREATED BY                                      SIZE
4a6f1e6bea54   34 hours ago   /bin/sh                                         2.93MB
<missing>      2 weeks ago    /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>      2 weeks ago    /bin/sh -c #(nop) ADD file:0c455f22e3…          5.6MB
```


19

Working with images and containers

Image and tags

- Images can have tags.
- Tags define image versions or variants.
- `docker pull ubuntu` will refer to ubuntu:latest.
- The :latest tag is generally updated often.

20

Image and tags

- **Don't specify tags:**
  - When doing rapid testing and prototyping.
  - When experimenting.
  - When you want the latest version.
- **Do specify tags:**
  - When going to production.
  - To ensure that the same version will be used everywhere.
  - To ensure repeatability later.

21

Launching containers with Docker Run

22

Start a new container


```
$ docker [container] run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

- **Most Useful run options:**
  - **--name** Give the container a symbolic name
  - **-v, --volume=[]** Bind mount a volume
  - **-p, --publish=[]** Publish a container's port(s) to the host
  - **-e, --env=[]** Set environment variables
  - **--link=[]** Add link to another container
  - **--restart="no"** Restart policy (no, on-failure[:max-retry], always)
  - **--rm=false** Automatically remove the container when it exits
  - **-d, --detach=false** Run container in background and print container ID
  - **-i, --interactive=false** Keep STDIN open even if not attached
  - **-t, --tty=false** Allocate a pseudo-TTY

23

See your contrainers

- **List containers**
  - The running containers:

```
$ docker ps  # or docker container ls
```

- **All containers:**

```
$ docker ps –a # or docker container ls -a
```

- **Show Metadata of a container**

```
$ docker inspect <container>
```

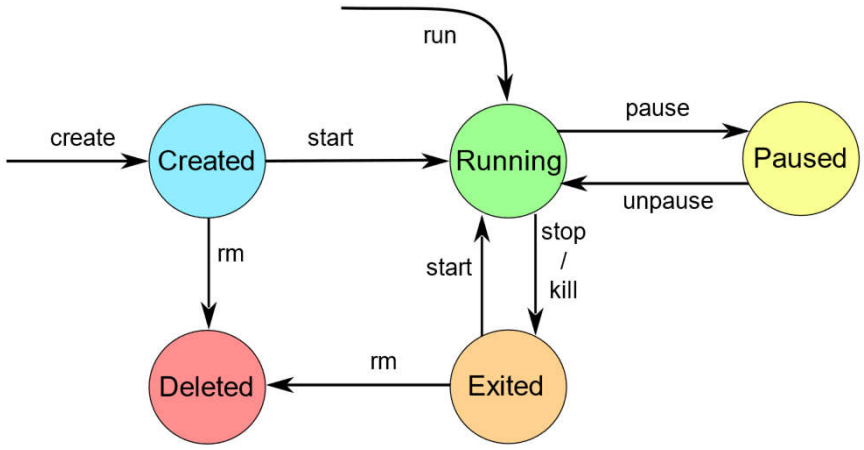
- **Output a special field of the metadata:**

```
$ docker inspect --format='{{.Image}}' <container>
```


24

Docker containers lifecycle

Working with images and containers



M.Romdhani, 2020

25

25

Container Lifecycle commands

Working with images and containers

- **Stop running containers:**
  ```
  $ docker stop <container..>
  ```
- **Start stopped containers:**
  ```
  $ docker start <container..>
  ```
- **Pause and Resume containers:**
  ```
  $ docker pause <container..>
  $ docker unpause <container..>
  ```
- **Remove containers:**
  ```
  $ docker rm <container..>
  ```
- **Kill running containers:**
  ```
  $ docker kill <container..>
  ```
- **Clean Up Trick**
  ```
  $ docker rm $(docker ps -a -q -f status=exited)
  ```

M.Romdhani, 2020

26

26

Interacting with containers

27

Attaching to a background container

- **To enter the shell of that running container we can use the docker attach command:**
  - Start a container in detached mode

```
$ docker run -it -d --name my-ubuntu2 ubuntu
```

  - Attach to the terminal of the container

```
$ docker attach my-ubuntu2
```

- **To detach from a container issue the keys `^P^Q (i.e CTRL-P CTRL-Q)`**
  - Be carful: Issuing `^C` will terminate the container !

28

Working with images and containers

Executing commands on containers

- **The docker exec command runs a new command in a running container.**
  - The command started using docker exec only runs while the container's primary process (PID 1) is running, and it is not restarted if the container is restarted.
  - The command will run in the default directory of the container.
- **Examples**

```
$ docker exec -it my-ubuntu2 /bin/bash
$ docker exec -it -e VAR=1 my-ubuntu2 bash
$ docker exec -it w /root  my-ubuntu2 pwd
```


29

Working with images and containers

View the logs of a container

- **Use docker logs to view the container output.**

```
$ docker logs 0a8
Fri Feb 20 00:39:52 UTC 2015
Fri Feb 20 00:39:53 UTC 2015
...
```

  - **We specified a prefix of the full container ID. You can, of course, specify the full ID.**
  - **The logs command will output the entire logs of the container. (Sometimes, that will be too much. Let's see how to address that.)**
- **View only the tail of the logs**
  - To avoid being spammed with many pages of output, we can use the **--tail** option:

```
$ docker logs --tail 10 0a8  # displays the last 10 lines
```


30

Follow the logs in real time

- Just like with the standard UNIX command tail -f, we can follow the logs of our container:

```
$ docker logs --tail 1 --follow 0a8
```

  - This will display the last line in the log file.
  - Then, it will continue to display the logs in real time.
  - Use ^C to exit.