



---

# **Subject: Containerization using Dockers**

**Module Number: 2**

**Module Name: Docker Images and Repositories**

# Docker Images and Repositories

## Syllabus:

- Docker Hub.
- Docker Image Layers.
- Listing Docker Images.
- Pulling Images.
- Searching Images.
- Building Docker Images – Using Commands, Using Docker File.
- Working with Docker file.
- pushing image to the Docker Hub.
- Deleting an Image.
- Running Docker Registry.



## Aim

**The aim of this module is to introduce Docker Image and Repository Implementation and Maintenance.**



# Docker Images and Repositories

## Objectives:

- Explain the concepts, functions and components of docker containers.
- Understand the use and benefits of containerization.
- Articulate Docker compose to create multi-container applications.
- Design the best practices of Dockerfiles and Image building.
- To Understand Run Docker Commands on the command line.
- Create and run Docker containers.

## Course Outcomes:

- Develop Containerized applications and implement continuous integration using Docker, explaining different types of cloud deployment and service models.
- Create own images and build the repository.
- Utilize Docker Orchestration and Service discovery features.
- Apply development tools, frameworks, platforms, libraries and packages to test hardware and software systems.
- Evaluate the fundamentals of solution architecture to provision cloud infrastructure.

## Understanding dockers and containers

- Docker is a tool designed to make it easier to deploy and run applications using containers.
- Container is a standardized unit of software that allows developer to package an application with all its code, components and dependencies and ship it out as one package.
- And docker will take care of shipping the container to all possible platforms in a standard way, so that the application runs quickly and reliably in any computing environment.
- Thus, developers can just focus on coding and packaging the software with its dependencies without worrying about deployment.

# Docker Images and Repositories

- Thus, Docker container can be used to resolve issues which occurred in previous example.
- In this case, the developer will create a tomcat docker image ( A Docker Image is nothing but a blueprint to deploy multiple containers of the same configurations ) using a base image like Ubuntu, which is already existing in Docker Hub (Docker Hub has some base docker images available for free) .
- Now this image can be used by the developer, the tester and the system admin to deploy the tomcat environment. We will discuss about docker images in detail in upcoming slides.



# Docker Images and Repositories

## Docker Hub

- A Docker registry is a repository for Docker images.
- The registry can be either a user's local (private) repository or a public repository like Docker Hub, AWS and Google's own registries.
- Docker Hub is a service provided by Docker and is the world's largest cloud-based repository of container images.
- It helps in finding ,creating, testing, storing and sharing container images with a team.
- It can store 1 or more versions of a specific Docker image.



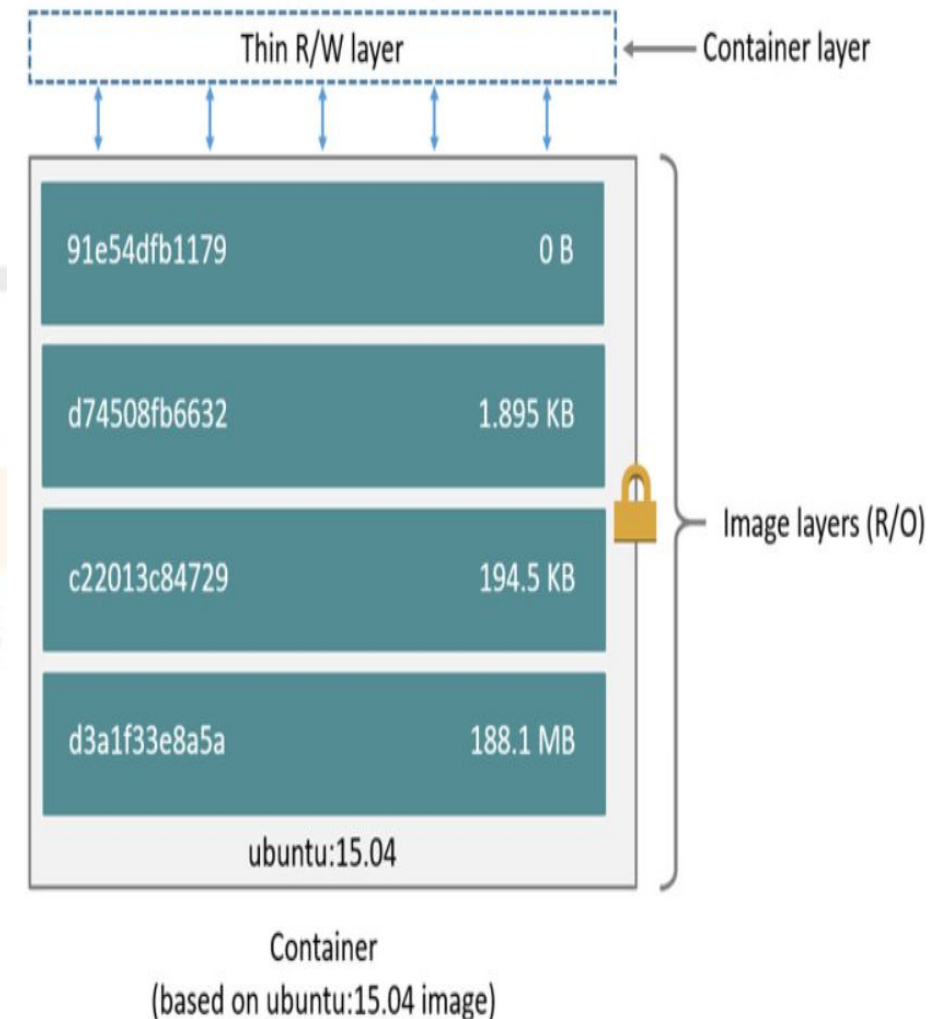
## Docker image

- Docker images are the building blocks of a Docker container.
- A Docker container image is a read only template which is lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.
- Docker container images become containers at runtime when they run on Docker Engine or else we can say that a docker container is a running instance of a docker Image.

# Docker Images and Repositories

## Format of Docker image

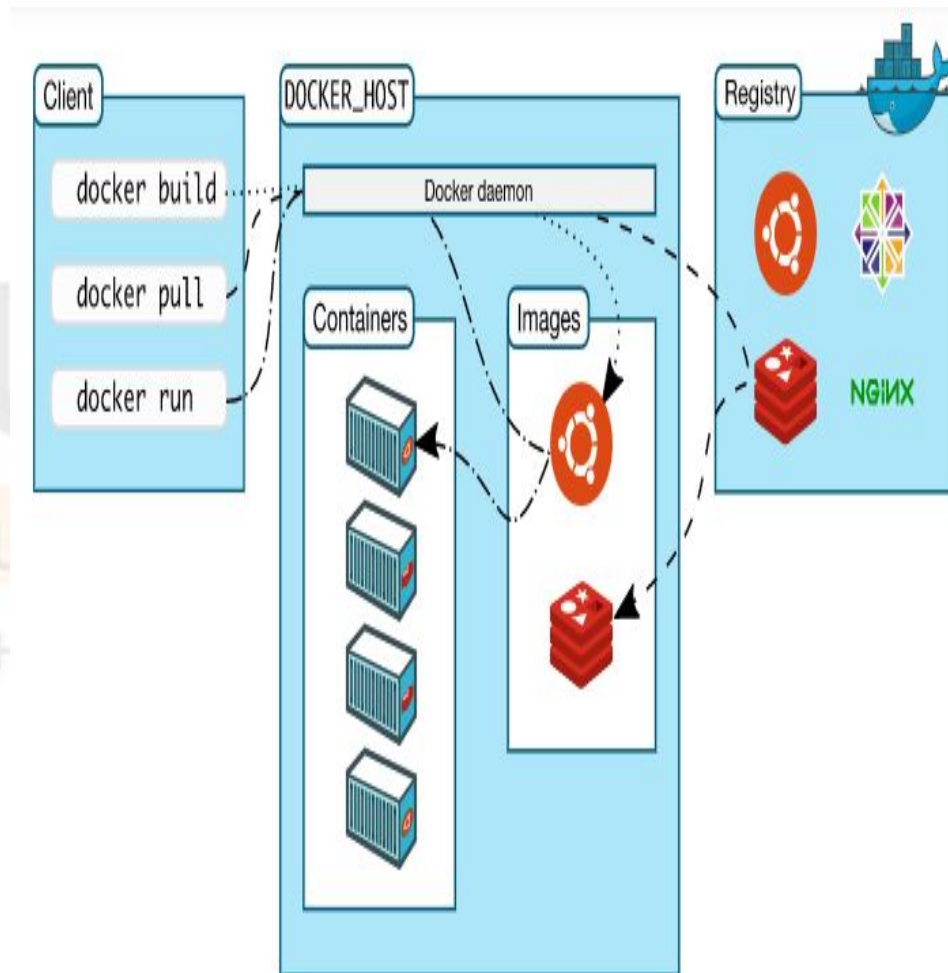
- It is a file comprised of multiple layers that is used to execute code in a docker container.
- A storage driver handles the details about the way these layers interact with each other. Different storage drivers are available, which have advantages and disadvantages in different situations.
- The major difference between a container and an image is the top writable layer. All writes to the container that add new or modify existing data are stored in this writable layer. When the container is deleted, the writable layer is also deleted. The underlying image remains unchanged.



# Docker Images and Repositories

## Docker Architecture

- The Docker client talks to the Docker daemon, which builds, pulls or runs Docker containers using Docker images.
- The Docker client and daemon can run on the same system, or we can connect a Docker client to a remote Docker daemon.
- The daemon(server) receives commands from the docker client through CLI or REST API's
- Docker client and daemon can be present on the same host machine or different hosts.



## Benefit of using Docker image layers

### Image Version control, component reuse and rollback :

- Each Docker image file is made up of a series of layers that are combined into a single image.
- A layer is created when the image changes. Every time a user specifies a command, such as run or copy, a new layer gets created.
- Docker reuses these layers to build new containers, which accelerates the building process.

## Docker image commands

Command	Description
<code>docker image build</code>	Build an image from a Dockerfile
<code>docker image history</code>	Show the history of an image
<code>docker image import</code>	Import the contents from a tarball to create a filesystem image
<code>docker image inspect</code>	Display detailed information on one or more images
<code>docker image load</code>	Load an image from a tar archive or STDIN
<code>docker image ls</code>	List images
<code>docker image prune</code>	Remove unused images
<code>docker image pull</code>	Pull an image or a repository from a registry
<code>docker image push</code>	Push an image or a repository to a registry
<code>docker image rm</code>	Remove one or more images
<code>docker image save</code>	Save one or more images to a tar archive (streamed to STDOUT by default)
<code>docker image tag</code>	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE



# Docker Images and Repositories

## Listing Docker images

- Docker images have intermediate layers that increase reusability, decrease disk usage, and speed up docker build by allowing each step to be cached. These intermediate layers are not shown by default.
- The ps command provides several columns of information:
- Container ID – a unique alphanumeric number for each container
- Image – The base operating system image the container is based on, Command – The command that launched the container.
- Created – How long ago the container was created
- Status – Uptime or downtime
- Ports – Specifies any ports forwarded to the container for networking, Name – A memorable name assigned by the Docker software

```
dejan@dejan-VirtualBox:~$ sudo docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
ad33d63a2f71	hello-world	"/hello"	3 minutes ago
Exited (0) 3 minutes ago		wizardly_hertz	
c5e95317294d	hello-world	"/hello"	4 minutes ago
Exited (0) 4 minutes ago		cranky_shannon	

```
dejan@dejan-VirtualBox:~$
```

## Listing Docker images

```
$ docker image ls [OPTIONS] [REPOSITORY[:TAG]]
```

### Options

Name, shorthand	Default	Description
<code>--all</code> , <code>-a</code>		Show all images (default hides intermediate images)
<code>--digests</code>		Show digests
<code>--filter</code> , <code>-f</code>		Filter output based on conditions provided
<code>--format</code>		Pretty-print images using a Go template
<code>--no-trunc</code>		Don't truncate output

## Listing Docker images

- Example and sample output : List the most recently created images.

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<none>	<none>	77af4d6b9913	19 hours ago	1.089 GB
committ	latest	b6fa739cedf5	19 hours ago	1.089 GB
<none>	<none>	78a85c484f71	19 hours ago	1.089 GB
docker	latest	30557a29d5ab	20 hours ago	1.089 GB
<none>	<none>	5ed6274db6ce	24 hours ago	1.089 GB
postgres	9	746b819f315e	4 days ago	213.4 MB
postgres	9.3	746b819f315e	4 days ago	213.4 MB
postgres	9.3.5	746b819f315e	4 days ago	213.4 MB
postgres	latest	746b819f315e	4 days ago	213.4 MB



## Listing Docker images

- Example and sample output : to list all images in the “java” repository, run the following command.
- The [REPOSITORY[:TAG]] value must be an “exact match”.

```
$ docker images java
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
java	8	308e519aac60	6 days ago	824.5 MB
java	7	493d82594c15	3 months ago	656.3 MB
java	latest	2711b1d6f3aa	5 months ago	603.9 MB

## Listing Docker images

- If both REPOSITORY and TAG are provided, only images matching that repository and tag are listed.
- Example: To find all local images in the “java” repository with tag “8” you can use following command:

```
$ docker images java:8
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
java	8	308e519aac60	6 days ago	824.5 MB

## Pulling Docker images

- Used to pull an image or repository from a registry.
- Docker Hub contains many pre-built images that you can pull and try without needing to define and configure your own.
- To download a particular image, or set of images (i.e., a repository), use docker pull.

# Docker Images and Repositories

## Pulling Docker images

```
$ docker image pull [OPTIONS] NAME[:TAG|@DIGEST]
```

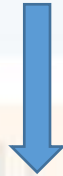
### Options

Name, shorthand	Default	Description
<code>--all-tags</code> , <code>-a</code>		Download all tagged images in the repository
<code>--disable-content-trust</code>	true	Skip image verification
<code>--platform</code>		<div>API 1.32+</div> Set platform if server is multi-platform capable
<code>--quiet</code> , <code>-q</code>		Suppress verbose output

## Searching images

- Search the Docker Hub for images.

```
$ docker search [OPTIONS] TERM
```



Name, shorthand	Default	Description
<code>--filter</code> , <code>-f</code>		Filter output based on conditions provided
<code>--format</code>		Pretty-print search using a Go template
<code>--limit</code>	25	Max number of search results
<code>--no-trunc</code>		Don't truncate output

# Docker Images and Repositories

## Searching images by name

- This example displays images with a name containing 'busybox'

```
$ docker search busybox
```

```
root@ubuntu:~#  
root@ubuntu:~# docker search wordpress
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
wordpress	The WordPress rich content management syst...	1751	[OK]	
bitnami/wordpress	Bitnami Docker Image for WordPress	40		[OK]
appcontainers/wordpress	Centos/Debian Based Customizable Wordpress...	34		[OK]
centurylink/wordpress	Wordpress image with MySQL removed.	13		[OK]
trafex/wordpress	Lightweight WordPress container with Nginx...	6		[OK]
scjalliance/wordpress	WordPress with GD and FreeType	3		[OK]
deardoooley/wordpress	Production-ready minimal Wordpress images.	2		[OK]
dsteinkopf/wordpress	wordpress clone plus some php extensions	1		[OK]
owncloud/wordpress	Simple WordPress image for our websites	1		[OK]
bakudankun/wordpress-ja	WordPress Japanese Edition forked from off...	1		[OK]
miqueladell/wordpress	Just a tailor made Wordpress.	0		[OK]



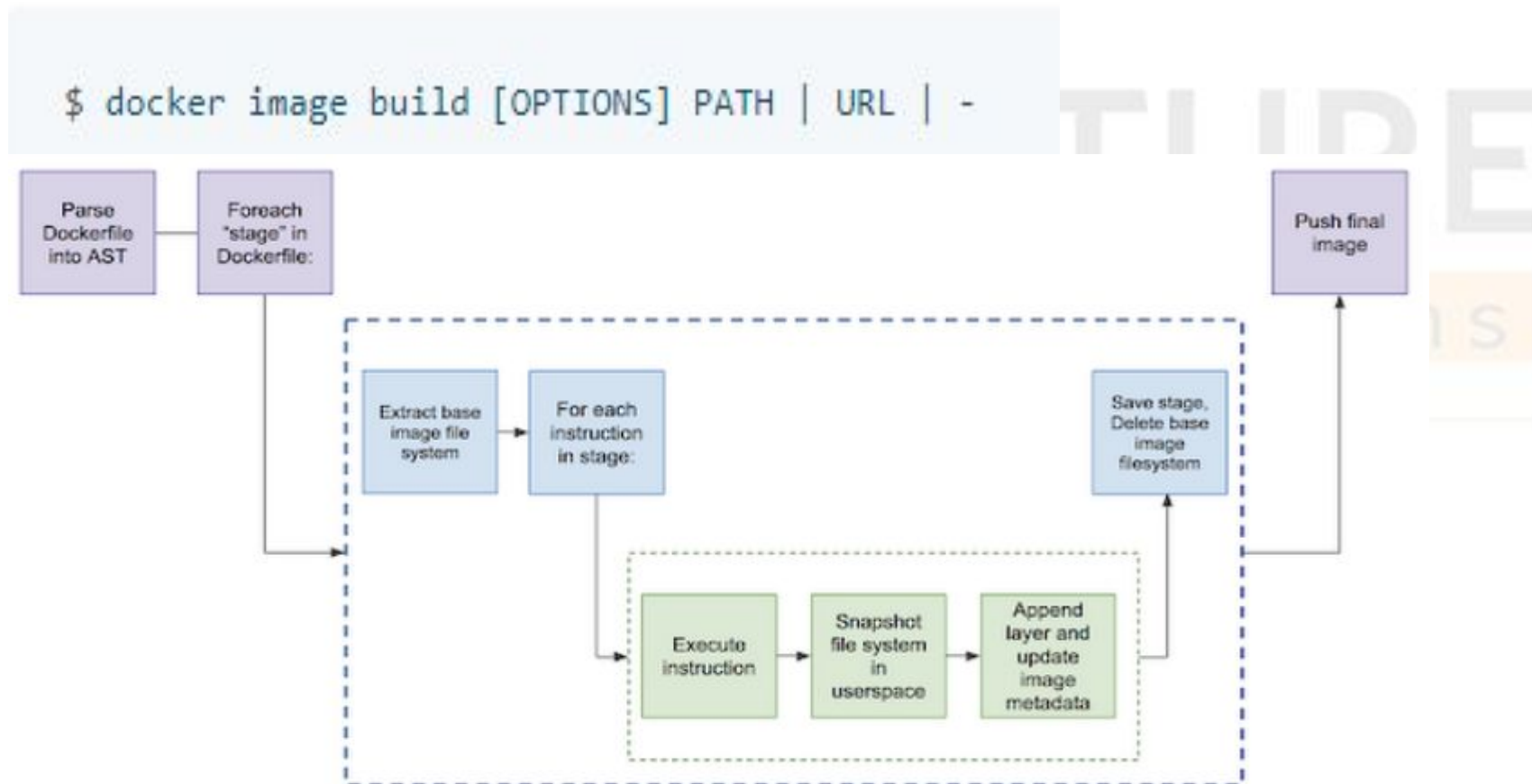
## Searching images by name

- Sample output:

NAME	DESCRIPTION	STARS	OFFICIAL	AUTO
busybox	Busybox base image.	316	[OK]	
progrium/busybox		50		[OK]
radial/busyboxplus	Full-chain, Internet enabled, busybox made...	8		[OK]
odise/busybox-python		2		[OK]
azukiapp/busybox	This image is meant to be used as the base...	2		[OK]
ofayau/busybox-jvm	Prepare busybox to install a 32 bits JVM.	1		[OK]
shingonoide/archlinux-busybox	Arch Linux, a lightweight and flexible Lin...	1		[OK]
odise/busybox-curl		1		[OK]
ofayau/busybox-libc32	Busybox with 32 bits (and 64 bits) libs	1		[OK]
peelsky/zulu-openjdk-busybox		1		[OK]
skomma/busybox-data	Docker image suitable for data volume cont...	1		[OK]
elektritter/busybox-teamspeak	Lightweight teamspeak3 container based on...	1		[OK]
socketplane/busybox		1		[OK]
oveits/docker-nginx-busybox	This is a tiny NginX docker image based on...	0		[OK]
ggtools/busybox-ubuntu	Busybox ubuntu version with extra goodies	0		[OK]
nikfoundas/busybox-confd	Minimal busybox based distribution of confd	0		[OK]
openshift/busybox-http-app		0		[OK]
jlllopis/busybox		0		[OK]
swyckoff/busybox		0		[OK]
powellquiring/busybox		0		[OK]
williamyeh/busybox-sh	Docker image for BusyBox's sh	0		[OK]
simplexsys/busybox-cli-powered	Docker busybox images, with a few often us...	0		[OK]
fhisamoto/busybox-java	Busybox java	0		[OK]
scottabernethy/busybox		0		[OK]
marcloup/busybox-solr		0		[OK]

## Building Docker images

- Build an image from a Dockerfile.





## Docker file

- Docker can build images automatically by reading the instructions from a Dockerfile.
- A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using `docker build` users can create an automated build that executes several command-line instructions in succession.
- The `docker build` command builds an image from a Dockerfile and a context.
- The build's context is the set of files at a specified location `PATH` or `URL`. The `PATH` is a directory on your local filesystem. The `URL` is a Git repository location.
- The build is run by the Docker daemon, not by the CLI. The first thing a build process does is send the entire context (recursively) to the daemon.

```
$ docker build .
```

```
Sending build context to Docker daemon 6.51 MB
```

```
...
```

## Docker file

- A Docker image consists of read-only layers each of which represents a Dockerfile instruction. Consider following docker file.

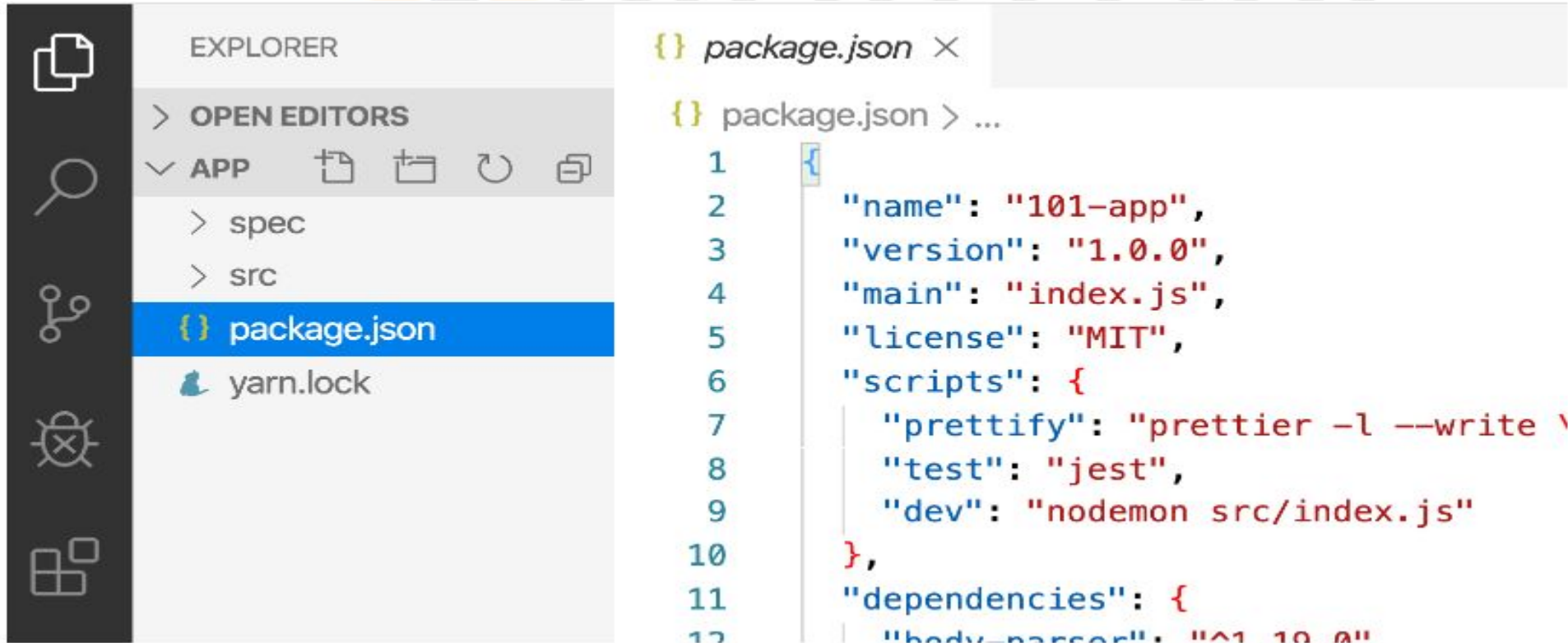
```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Each instruction creates one layer:

- `FROM` creates a layer from the `ubuntu:18.04` Docker image.
- `COPY` adds files from your Docker client's current directory.
- `RUN` builds your application with `make`.
- `CMD` specifies what command to run within the container.

## Docker file

- When you run an image and generate a container, you add a new *writable layer* (the “container layer”) on top of the underlying layers.
- All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this writable container layer.



```
{} package.json ×
{} package.json > ...
1  {
2    "name": "101-app",
3    "version": "1.0.0",
4    "main": "index.js",
5    "license": "MIT",
6    "scripts": {
7      "prettify": "prettier -l --write \
8      "test": "jest",
9      "dev": "nodemon src/index.js"
10   },
11   "dependencies": {
12     "body-parser": "1.18.0"
```

## Pushing Docker images

- Push an image or a repository to a registry.
- Concurrent uploads: By default the Docker daemon will push five layers of an image at a time.

```
$ docker push [OPTIONS] NAME[:TAG]
```



Name, shorthand	Default	Description
<code>--all-tags</code> , <code>-a</code>		Push all tagged images in the repository
<code>--disable-content-trust</code>	true	Skip image signing
<code>--quiet</code> , <code>-q</code>		Suppress verbose output

## Pushing Docker images

- Example: Push a new image to a registry.
- First save the new image by finding the container ID (using `docker container ls`) and then committing it to a new image name. Note that only a-z0-9-\_. are allowed when naming images.

```
$ docker container commit c16378f943fe rhel-httpd:latest
```

## Pushing Docker images

- Now, push the image to the registry using the image ID.
- In this example, the registry is on host named registry-host and listening on port 5000. To do this, tag the image with the host name or IP address, and the port of the registry:

```
$ docker image tag rhel-httpd:latest registry-host:5000/myadmin/rhel-httpd:latest  
  
$ docker image push registry-host:5000/myadmin/rhel-httpd:latest
```

Check that this worked by running:

```
$ docker image ls
```

You should see both `rhel-httpd` and `registry-host:5000/myadmin/rhel-httpd` listed.



## Deleting Docker image

- **docker image rm:** Remove one or more images.

```
$ docker image rm [OPTIONS] IMAGE [IMAGE...]
```



Name, shorthand	Default	Description
<code>--force</code> , <code>-f</code>		Force removal of the image
<code>--no-prune</code>		Do not delete untagged parents

## Deleting Docker image

- **docker rm:** Remove one or more containers.

```
$ docker rm [OPTIONS] CONTAINER [CONTAINER...]
```



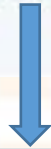
Name, shorthand	Default	Description
<code>--force</code> , <code>-f</code>		Force the removal of a running container (uses SIGKILL)
<code>--link</code> , <code>-l</code>		Remove the specified link
<code>--volumes</code> , <code>-v</code>		Remove anonymous volumes associated with the container



## Deleting Docker image

- **docker rmi:** Remove one or more images.

```
$ docker rmi [OPTIONS] IMAGE [IMAGE...]
```



Name, shorthand	Default	Description
<code>--force</code> , <code>-f</code>		Force removal of the image
<code>--no-prune</code>		Do not delete untagged parents

## Deleting Docker image

### **docker rmi:**

- Removes (and un-tags) one or more images from the host node.
- If an image has multiple tags, using this command with the tag as a parameter only removes the tag.
- If the tag is the only one for the image, both the image and the tag are removed.
- If an image has one or more tags referencing it, you must remove all of them before the image is removed.
- Refer to example and sample output in following slide.

## Deleting Docker image

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
test1	latest	fd484f19954f	23 seconds ago	7 B (virtual 4.96
test	latest	fd484f19954f	23 seconds ago	7 B (virtual 4.96
test2	latest	fd484f19954f	23 seconds ago	7 B (virtual 4.96

```
$ docker rmi fd484f19954f
```

Error: Conflict, cannot delete image fd484f19954f because it is tagged in multiple repositories, use -f  
2013/12/11 05:47:16 Error: failed to remove one or more images

```
$ docker rmi test1:latest
```

Untagged: test1:latest

```
$ docker rmi test2:latest
```

Untagged: test2:latest

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
test	latest	fd484f19954f	23 seconds ago	7 B (virtual 4.96

```
$ docker rmi test:latest
```

Untagged: test:latest  
Deleted: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8

## Summary

- Docker is a tool designed to make it easier to deploy and run applications using containers, ensuring that the application works seamlessly in any environment.
- Container is a standardized unit of software that allows developer to package an application with all its code, components and dependencies and ship it out as one package.
- Docker container images become containers at runtime when they run on Docker Engine.
- Docker Hub is a repository for docker images.

## Summary

- Docker is a tool designed to make it easier to deploy and run applications using containers, ensuring that the application works seamlessly in any environment.
- Container is a standardized unit of software that allows developer to package an application with all its code, components and dependencies and ship it out as one package.
- Docker container images become containers at runtime when they run on Docker Engine.
- Docker Hub is a repository for docker images.

## Self Assessment Questions

1. \_\_\_\_\_ is a running instance of a docker Image.
  - a. Docker file
  - b. Docker container
  - c. Docker hub
  - d. None

**Answer: b**

## Self Assessment Questions

2. \_\_\_\_\_ is a repository of docker images.
- a. Docker swarm
  - b. Docker engine
  - c. Docker hub
  - d. None

**Answer: c**

## Self Assessment Questions

3. A layer is created when the image changes. Each Docker image file is made up of a series of layers . These intermediate layers help to:
- a. increase reusability
  - b. decrease disk usage
  - c. speed up docker build
  - d. All of the above

**Answer: d**



## Self Assessment Questions

4. Which of the following list commands would be suitable to get the desired output as shown below?

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
java	8	308e519aac60	6 days ago	824.5 MB

- a. \$ docker list java: 8
- b. \$ docker images list java: 8
- c. \$ docker list java
- d. \$ docker images list java: 8

**Answer: b**

## Self Assessment Questions

5. Which of the followings commands is used to remove unused images?
- a. docker image rm
  - b. docker image ls
  - c. docker image prune
  - d. docker image push

**Answer: c**

## Self Assessment Questions

6. The docker build command builds an image from a Dockerfile and a *context*. The build's context is the set of files at a specified location PATH or URL. The PATH is \_\_\_\_\_.

- a. a directory on your local filesystem
- b. a Git repository location
- c. both
- d. none

**Answer: a**

## Self Assessment Questions

7. State true or false: The command to push an image to the registry:

```
$ docker push [OPTIONS] NAME[:TAG]
```

- a. True
- b. False

**Answer: a**

## Self Assessment Questions

8. If an image has multiple tags, using docker rmi command :

- a. Removes only the tag
- b. Removes the image
- c. Removes both the image and tag
- d. none

**Answer: a**

## Self Assessment Questions

9. If the tag is only one for the image, using docker rmi command :
- a. Removes only the tag
  - b. Removes the image
  - c. Removes both the image and tag
  - d. none

**Answer: c**

## Self Assessment Questions

10. By default the Docker daemon will push \_\_\_\_\_ layers of an image at a time.

- a. four
- b. five
- c. three
- d. As many as required

**Answer: b**



## Assignment

1. Register yourself with Docker Hub. Create a public/ private repository. Try pulling, pushing docker images using command line in docker desktop and run those containers. You can get help from git hub for docker images.
2. Perform the orientation and setup experiment using <https://docs.docker.com/get-started/>.
3. Experiment with a sample application, build app's container image and start app container using [https://docs.docker.com/get-started/02\\_our\\_app/](https://docs.docker.com/get-started/02_our_app/).
4. Create a Simple Compose file to run Django Application

## Video Links

Topics	Video Link	Notes
Get started with docker-orientation and setup	<a href="https://www.youtube.com/watch?v=3c-iBn73dDE&amp;feature=emb_rel_pause">https://www.youtube.com/watch?v=3c-iBn73dDE&amp;feature=emb_rel_pause</a>	Introduction to Docker and Installation and Deployment
Docker images	<a href="https://youtu.be/QBOcKdh-fwQ">https://youtu.be/QBOcKdh-fwQ</a>	Explained Docker Images
How to push and pull docker image from docker hub	<a href="https://youtu.be/EIHY_CY5J0k">https://youtu.be/EIHY_CY5J0k</a>	Push and Pull Concepts with respect to Docker file and Hub
Dockerfile, create and build image from dockerfile	<a href="https://youtu.be/LQjaJINkQXY">https://youtu.be/LQjaJINkQXY</a>	How to create Compose file and Run the images

## E-Book Links and References

Topics	Video Link
Introduction to docker image	<a href="https://www.docker.com/why-docker">https://www.docker.com/why-docker</a> <a href="https://docker-curriculum.com/#hello-world">https://docker-curriculum.com/#hello-world</a> <a href="https://docs.docker.com/get-started/overview/">https://docs.docker.com/get-started/overview/</a>
Docker installation on windows	<a href="https://ropenscilabs.github.io/r-docker-tutorial/04-Docker-hub.html">https://ropenscilabs.github.io/r-docker-tutorial/04-Docker-hub.html</a> <a href="https://docs.docker.com/engine/reference/commandline/">https://docs.docker.com/engine/reference/commandline/</a>
Docker Repositories	<a href="https://docs.docker.com/develop/develop-images/dockerfile_best-practices/">https://docs.docker.com/develop/develop-images/dockerfile_best-practices/</a>