Introduction to Docker

Learning Outcomes

On successful completion of this module, students should be able to:

- **Use Docker, (bio)conda, and git to create reproducible analysis environments and generate reproducible results**
- **Use the Linux command line environment including access/use of a High-Performance Compute (HPC) cluster**
- Write Rmarkdown documents to generate reproducible research reports
- Analyse gene expression microarrays in order to identify differentially expressed genes, enriched GO terms, pathways, and gene sets
- Develop simple Shiny applications

Computer Science 101

What do you already know?

Q1. What is a computer?

Q2. What are the key pieces of hardware?

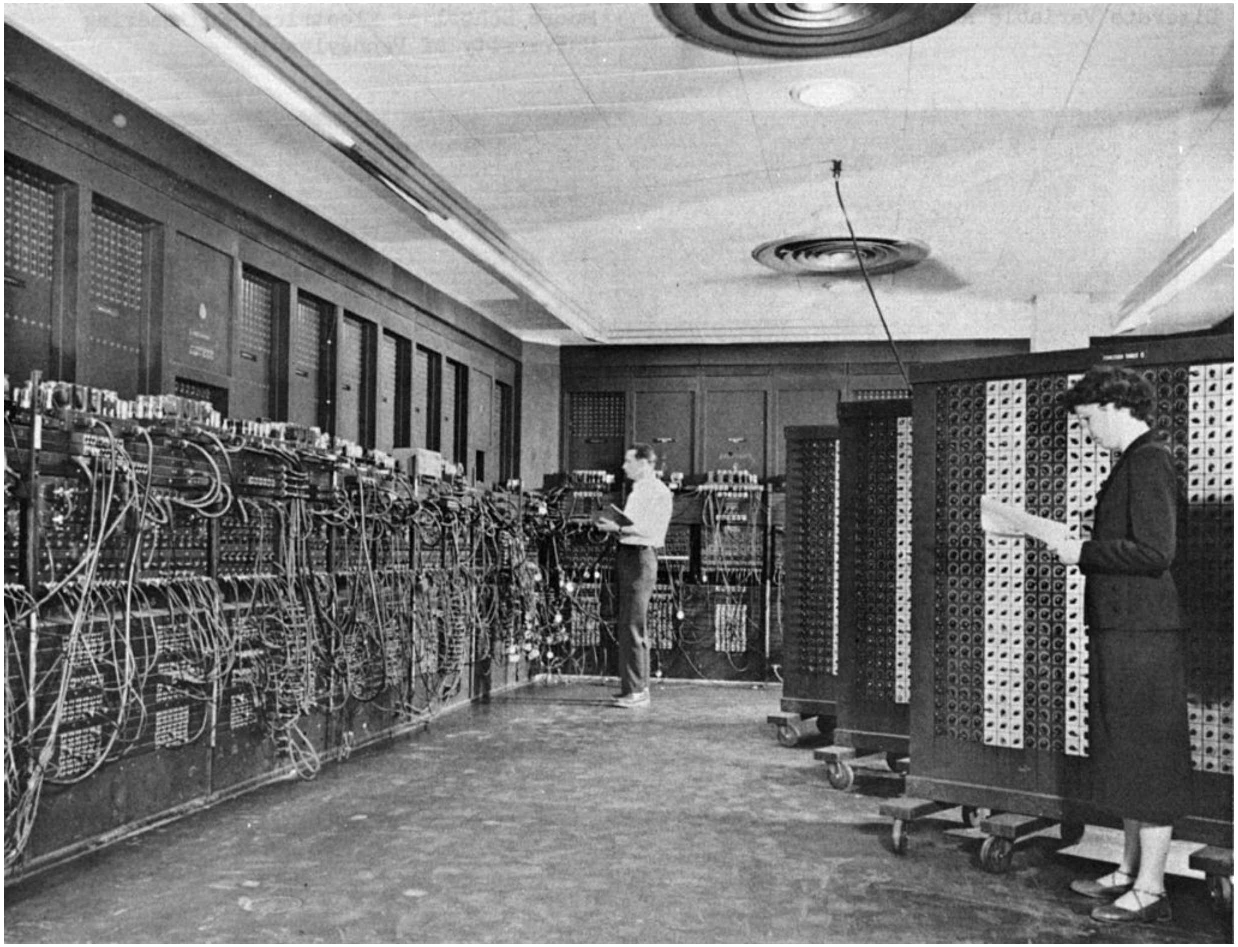
Q3. What is a bit? A byte?

Q4. What is a Virtual Machine?

Q5. What is HTTP/HTTPS?

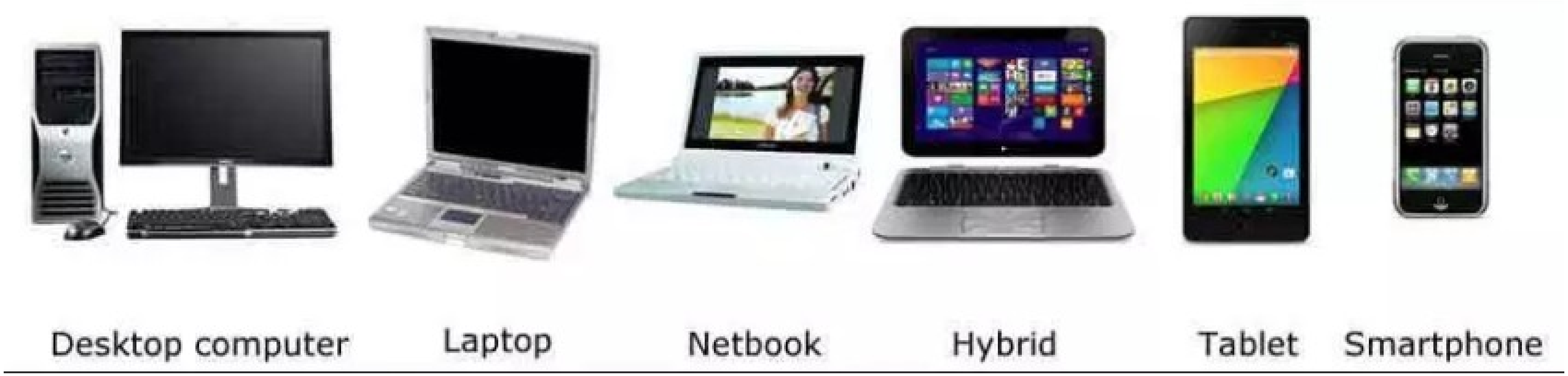
What is a computer?

A general-purpose machine that takes in information (data) by a process called input, stores and processes it, and then generates some kind of output (result)



What is a computer?

PCs



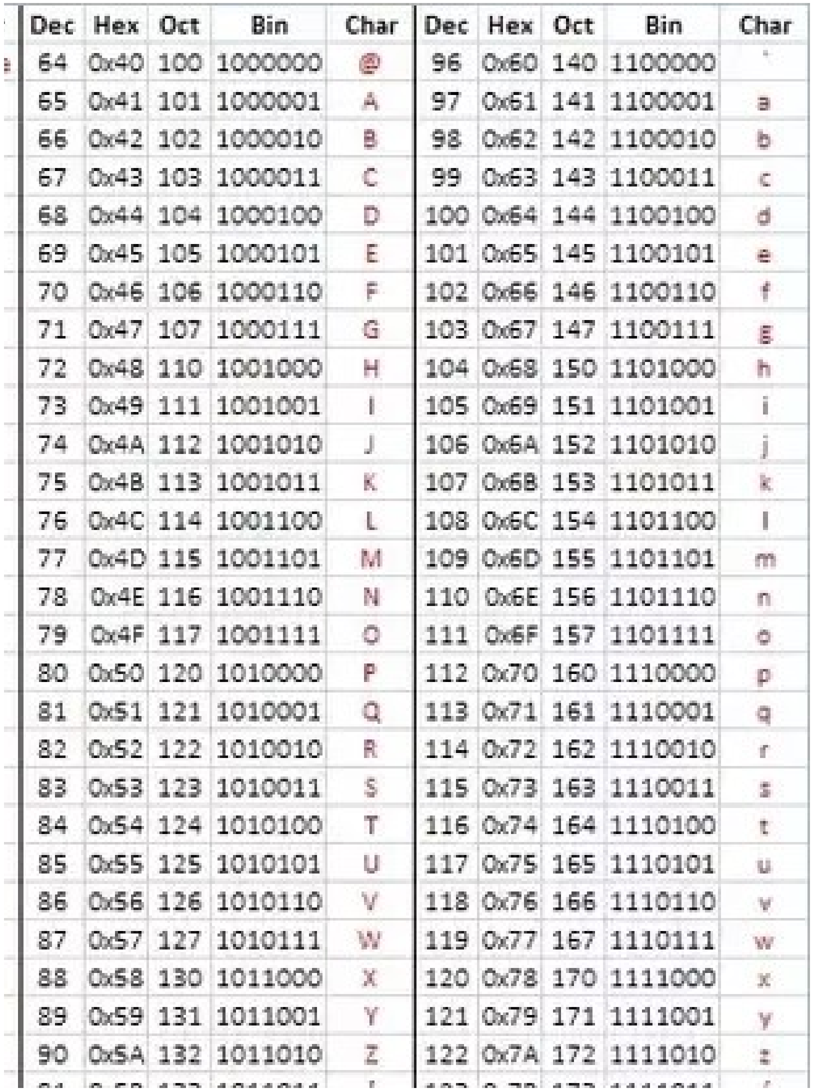
Server

Supercomputer / HPC cluster

How is information encoded?

- Bit (b)
  - Binary digit (0/1), basic unit of information
  - Physically - punched hole / magnetic polarity / stored electrical charge
  - 4 bits (nibble) = 1 hexadecimal digit (0-f)
    – often used for memory addresses

- Byte (B) = 8 bits / 2 hexadecimals
  - One ASCII character (First Unicode Byte)
  - 1,000 bytes = 1 kilobyte (KB)
  - 1,000 KB = 1 megabyte (MB)
  - 1,000 MB = 1 gigabyte (GB)
  - 1,000 GB = 1 Terabyte (TB)
  - 1,000 TB = 1 Petabyte (PB)

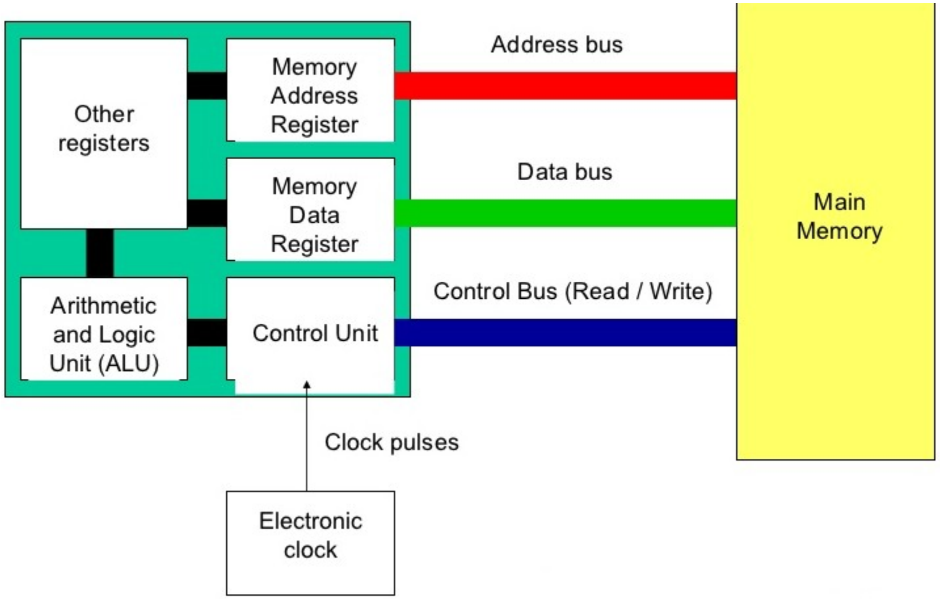


Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
64	0x40	100	1000000	@	96	0x60	140	1100000	`
65	0x41	101	1000001	A	97	0x61	141	1100001	a
66	0x42	102	1000010	B	98	0x62	142	1100010	b
67	0x43	103	1000011	C	99	0x63	143	1100011	c
68	0x44	104	1000100	D	100	0x64	144	1100100	d
69	0x45	105	1000101	E	101	0x65	145	1100101	e
70	0x46	106	1000110	F	102	0x66	146	1100110	f
71	0x47	107	1000111	G	103	0x67	147	1100111	g
72	0x48	110	1001000	H	104	0x68	150	1101000	h
73	0x49	111	1001001	I	105	0x69	151	1101001	i
74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
80	0x50	120	1010000	P	112	0x70	160	1110000	p
81	0x51	121	1010001	Q	113	0x71	161	1110001	q
82	0x52	122	1010010	R	114	0x72	162	1110010	r
83	0x53	123	1010011	S	115	0x73	163	1110011	s
84	0x54	124	1010100	T	116	0x74	164	1110100	t
85	0x55	125	1010101	U	117	0x75	165	1110101	u
86	0x56	126	1010110	V	118	0x76	166	1110110	v
87	0x57	127	1010111	W	119	0x77	167	1110111	w
88	0x58	130	1011000	X	120	0x78	170	1111000	x
89	0x59	131	1011001	Y	121	0x79	171	1111001	y
90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z

What are the key hardware / software components?

Hardware (Physical devices of system)

- Input (Keyboard, mouse, etc.)
- Output (Monitor, printer, etc.)
- Processor (CPU, GHz)
- Memory (RAM, volatile)
- Storage (HDD, SSD)



Software (Programs that run (on) computers)

- BIOS - Basic Input Output System (non-volatile flash memory, POST, load OS to RAM)
- OS - Operating system (kernel, device drivers, user interface)
- Application software (Web browser, spreadsheets, etc.)

What is Unix/Linux

Unix (originally developed at Bell labs in 1970s) is a family of operating systems with some powerful features:

Stable / Secure - Generally less prone to crashes / hacks

Efficient multitasking - Designed for a multiuser environment

Minimalist, modular code ("Do one thing and do it well") written mostly in C – portable

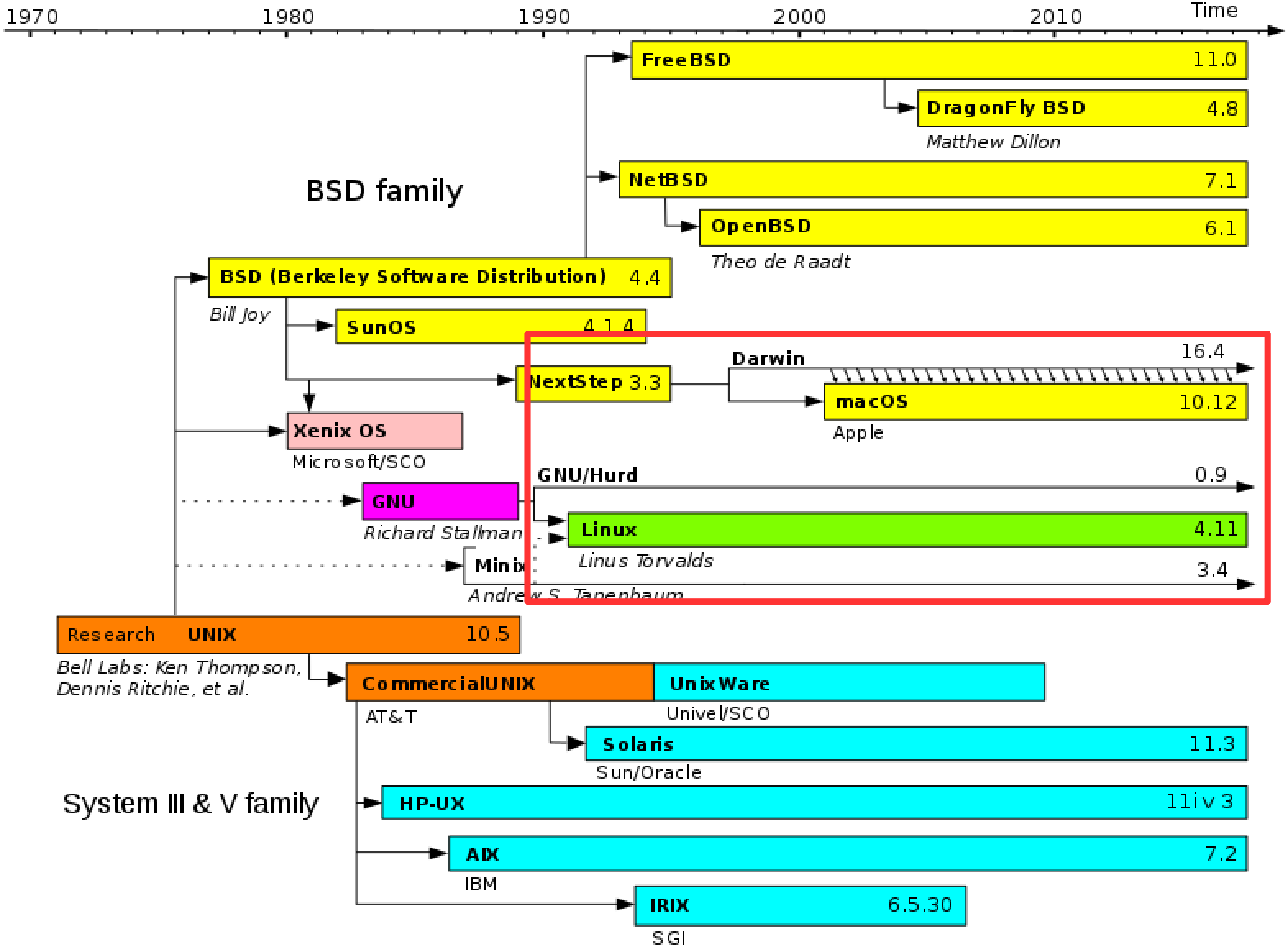
Unix shell – command line interpreter/interface (CLI), user enters text in a window to execute commands

Unified **File System – "**Everything is a file" (documents/directories/devices/processes)

Linux is a family of **free/open-source OS**es (Linux distributions) built on the **Linux kernel**. Modern variants typically use the **X11 Windows System** plus a **desktop environment** to provide a **GUI.**

Most compute clusters (supercomputers) however run Unix/Linux – we usually need to use these types of systems to handle large-scale genomics analyses.

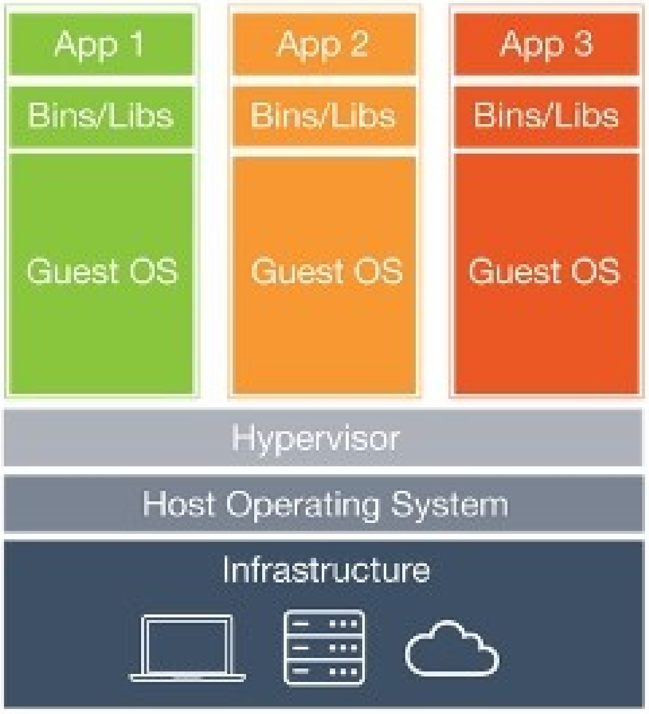
What is Unix/Linux



What is a Virtual Machine (VM)?

Run one operating system emulated within another operating system

- Oracle VMWare / VirtualBox / Microsoft Hyper-V



Virtual Machines

Some network fundamentals

Protocol: Standard that defines communication procedures/formats between two or more devices over a network

Port: Logical communication endpoint, numbered 0 to 65535 (16-bit), 0 to 1023 are the ‘well-known ports’ or system ports

Examples:

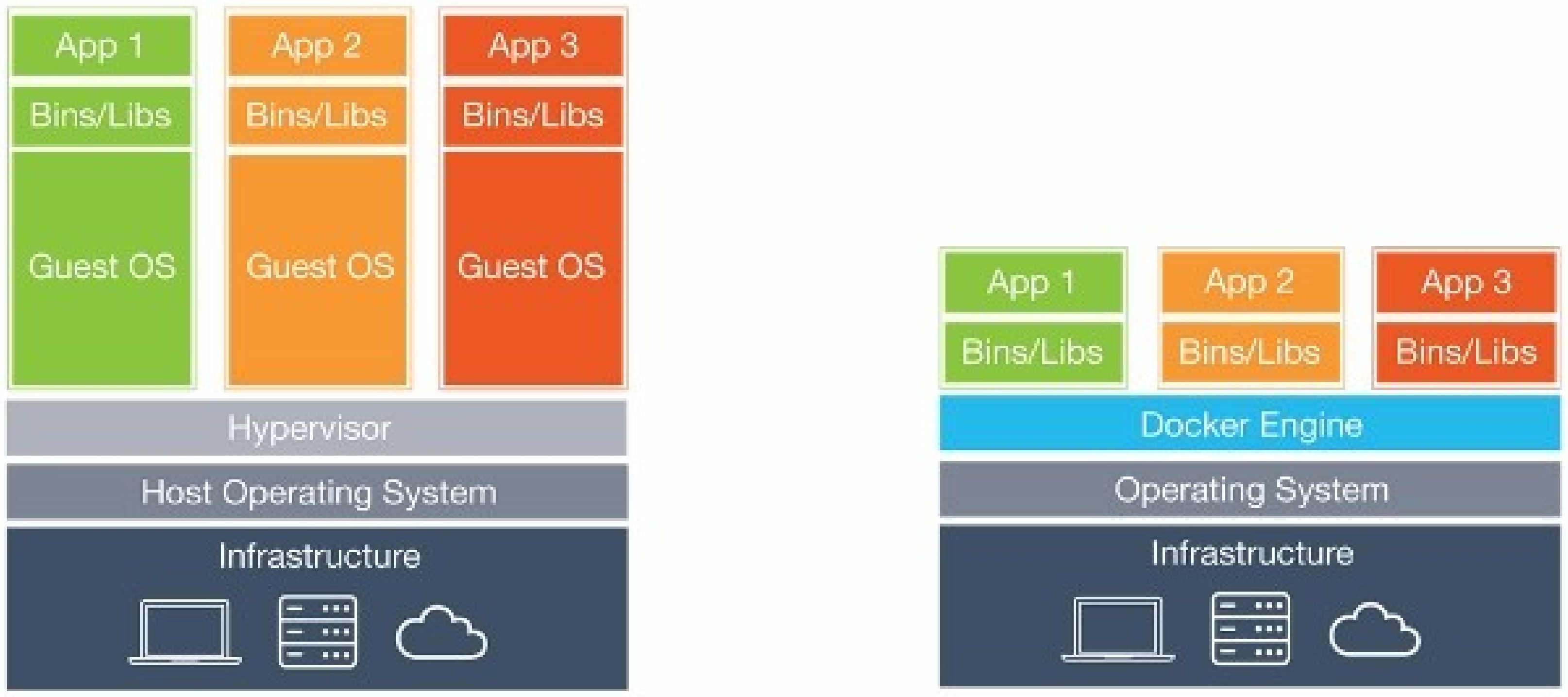
- HTTP(S) – (Secure) Hypertext Transfer Protocol, client/server–based protocol used to transfer web pages (HTML) across a network (Ports 80 and 443)
- FTP – File Transfer Protocol, unencrypted file transfer (Ports 20 and 21)
- SSH – Secure Shell, secure (encrypted) connection between devices for remote commands / file transfer (Port 22)

Introduction to Docker

Basic concepts and terminology

What is Docker?

- Program to create, manage and run containers (lightweight, standalone, executable packages that include all code, system tools, and libraries to necessary run an application)



Virtual Machines

Containers

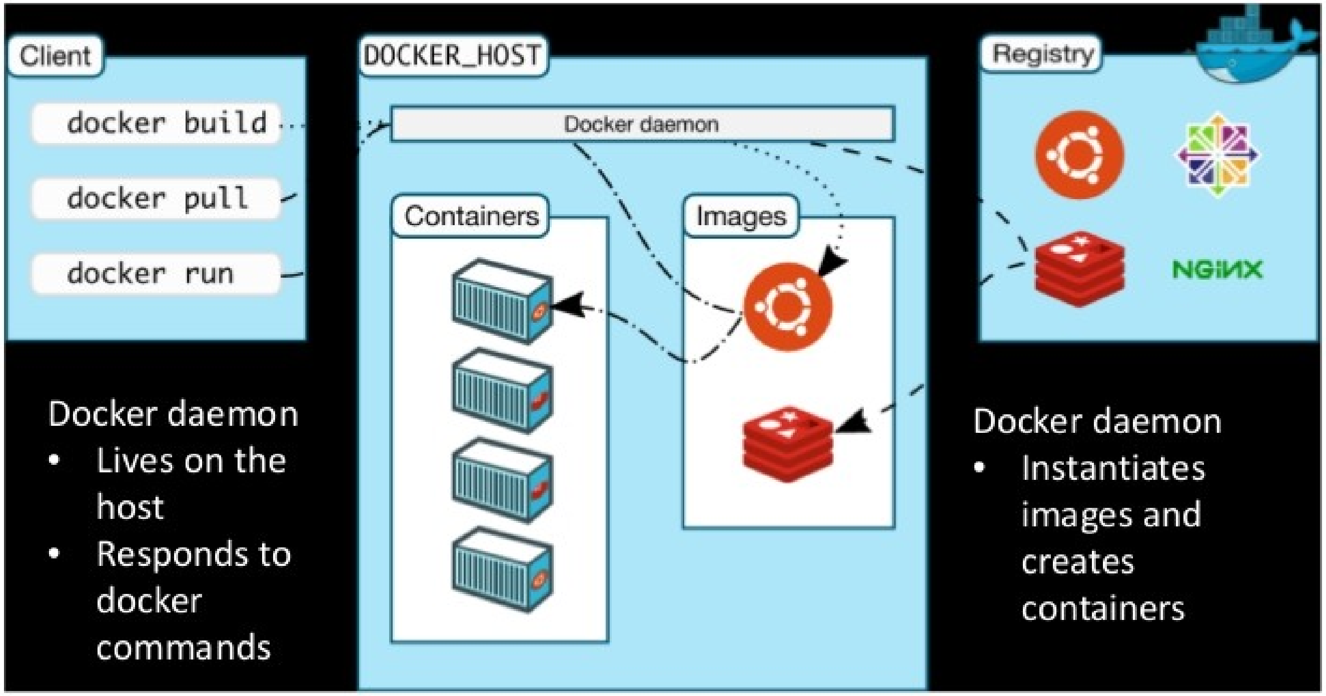
Basic concepts and terminology

Docker Image: read-only template with instructions for creating a container

Docker Container: a runnable instance of an image

Docker Registry: ‘App-store’ for Docker images. Docker is configured to use **Docker Hub** by default.

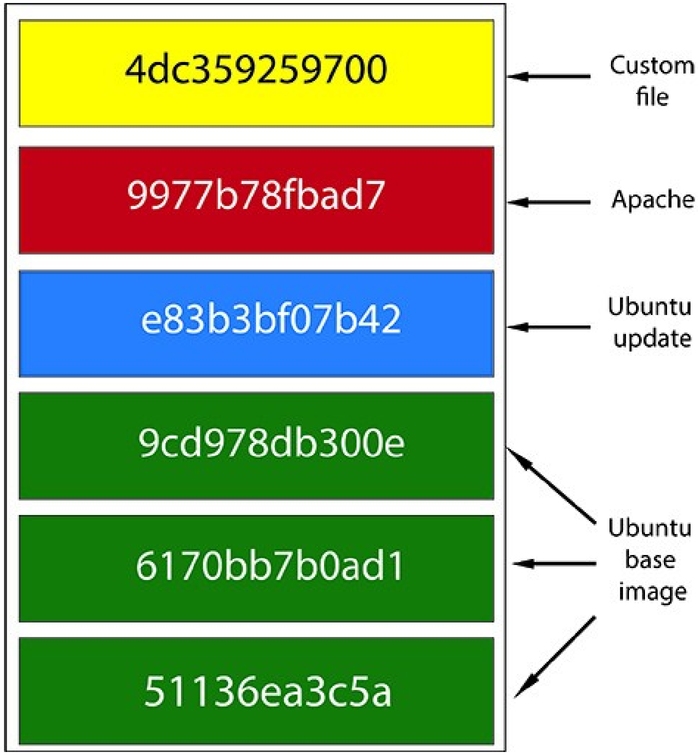
Dockerfile: set of instructions to build an image



Docker daemon
- Lives on the host
- Responds to docker commands

Docker daemon
- Instantiates images and creates containers

Basic concepts and terminology

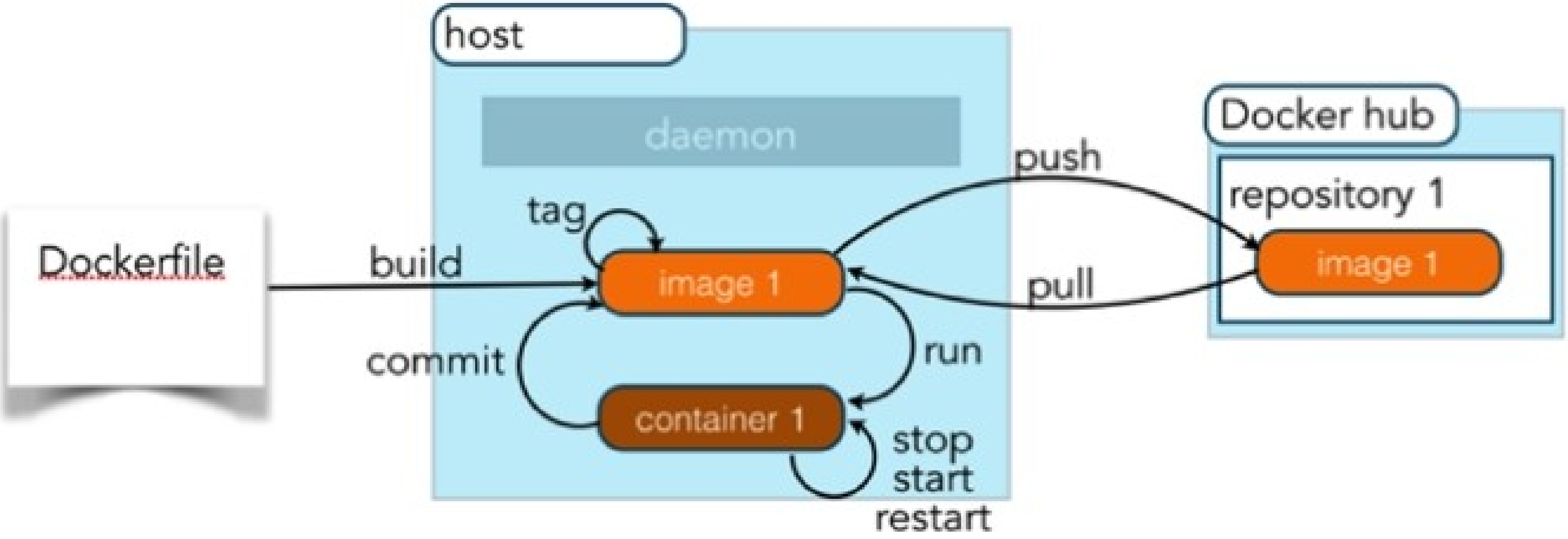


Each (RUN) instruction in a **Dockerfile** creates a **layer** in the image.

After modification, **only layers which have changed are rebuilt**.

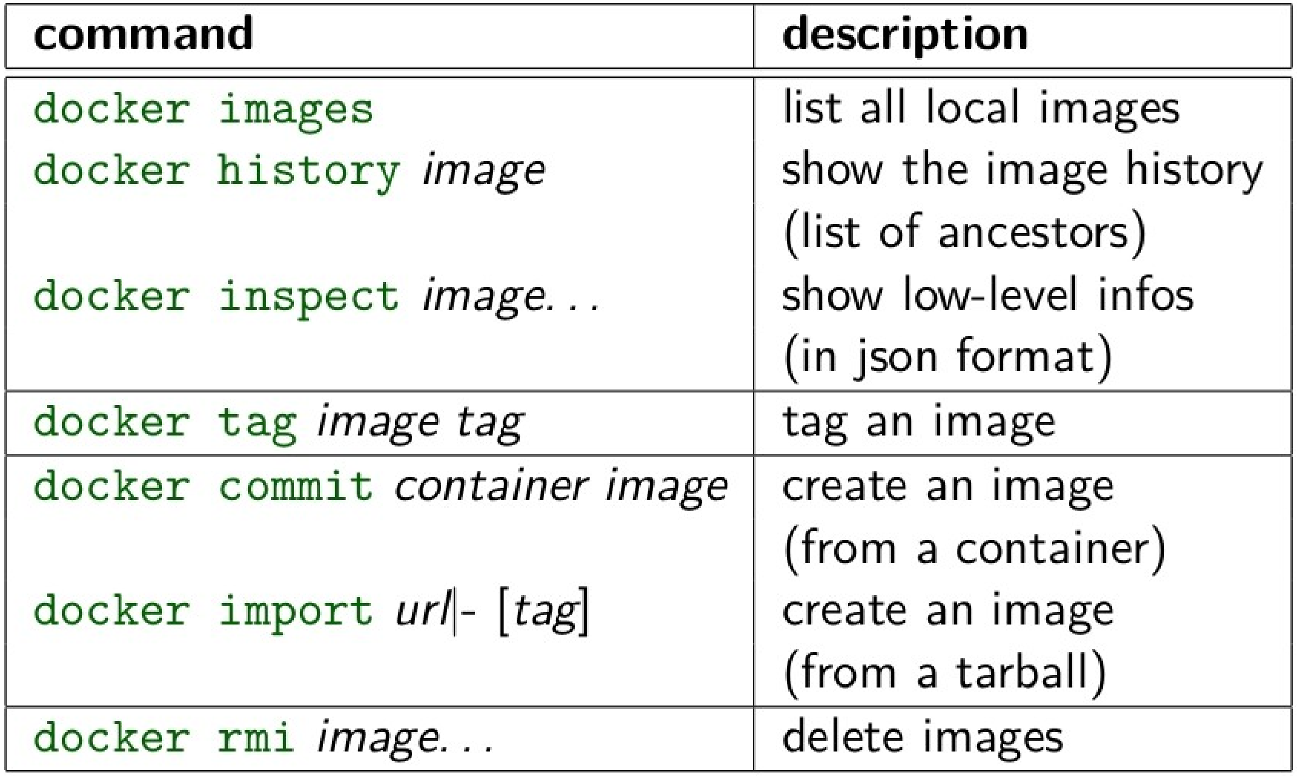
The **final layer is a read-write filesystem** to allow the container to create / modify its local files.

Core commands and options



Core commands and options

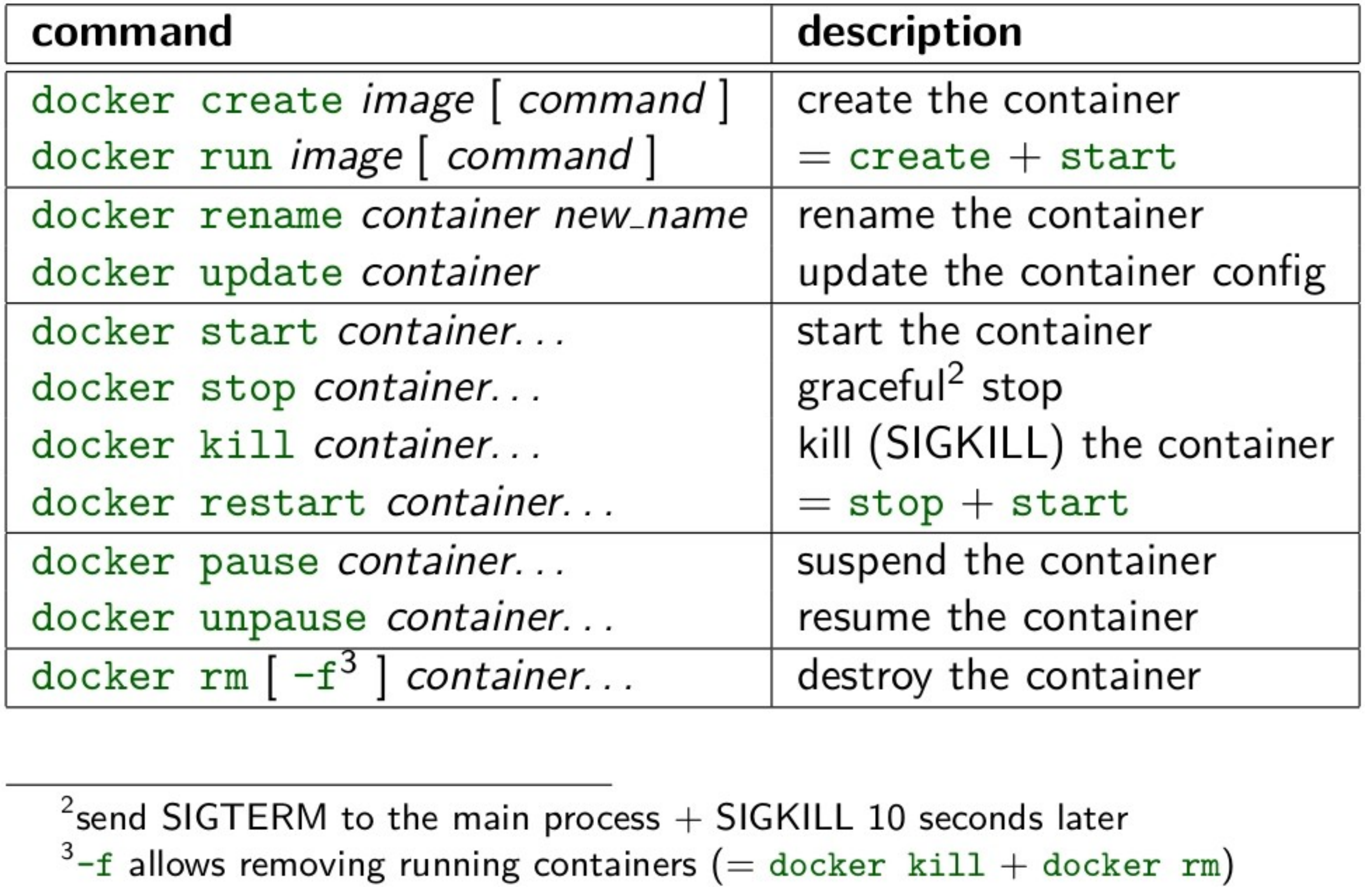
Docker image commands



command	description
`docker images`	list all local images
`docker history` *image*	show the image history (list of ancestors)
`docker inspect` *image. . .*	show low-level infos (in json format)
`docker tag` *image tag*	tag an image
`docker commit` *container image*	create an image (from a container)
`docker import` *url*|- [*tag*]	create an image (from a tarball)
`docker rmi` *image. . .*	delete images

Core commands and options

Docker container commands:



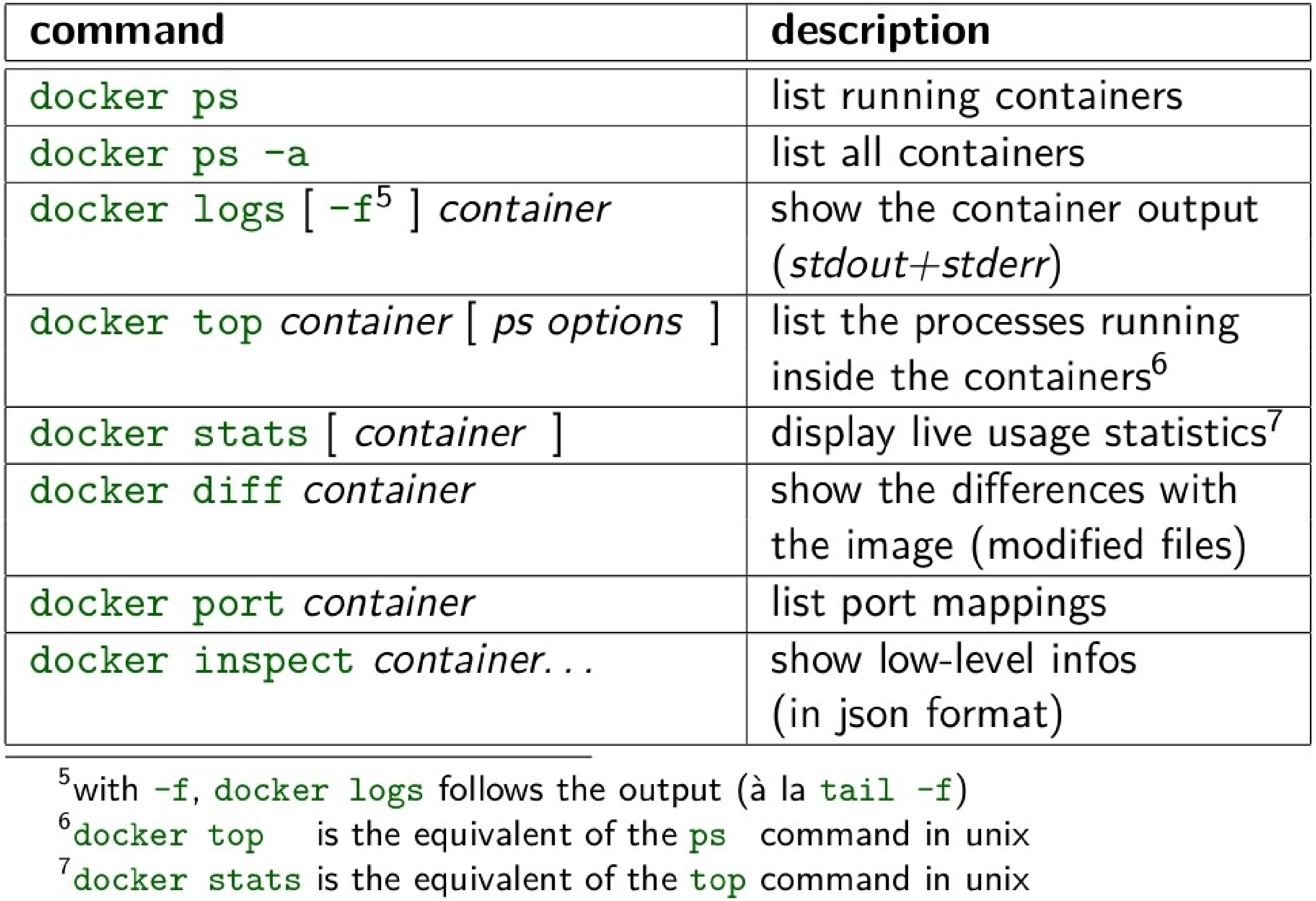
command	description
`docker create` *image* [*command*]	create the container
`docker run` *image* [*command*]	= `create` + `start`
`docker rename` *container new_name*	rename the container
`docker update` *container*	update the container config
`docker start` *container...*	start the container
`docker stop` *container...*	graceful[2] stop
`docker kill` *container...*	kill (SIGKILL) the container
`docker restart` *container...*	= `stop` + `start`
`docker pause` *container...*	suspend the container
`docker unpause` *container...*	resume the container
`docker rm` [`-f`[3]] *container...*	destroy the container

[2] send SIGTERM to the main process + SIGKILL 10 seconds later

[3] `-f` allows removing running containers (= `docker kill` + `docker rm`)

Core commands and options

Docker container commands:



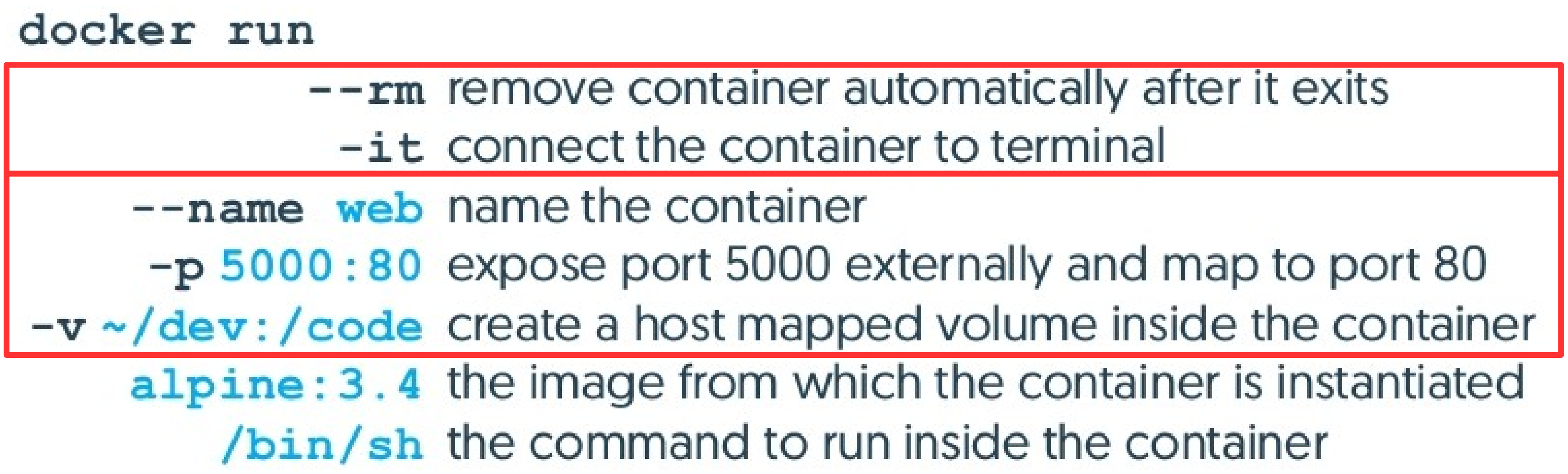
command	**description**
`docker ps`	list running containers
`docker ps -a`	list all containers
`docker logs` [`-f`[5]] *container*	show the container output (*stdout*+*stderr*)
`docker top` *container* [*ps options*]	list the processes running inside the containers[6]
`docker stats` [*container*]	display live usage statistics[7]
`docker diff` *container*	show the differences with the image (modified files)
`docker port` *container*	list port mappings
`docker inspect` *container*. . .	show low-level infos (in json format)

[5] with `-f`, `docker logs` follows the output (à la `tail -f`)

[6] `docker top` is the equivalent of the `ps` command in unix

[7] `docker stats` is the equivalent of the `top` command in unix

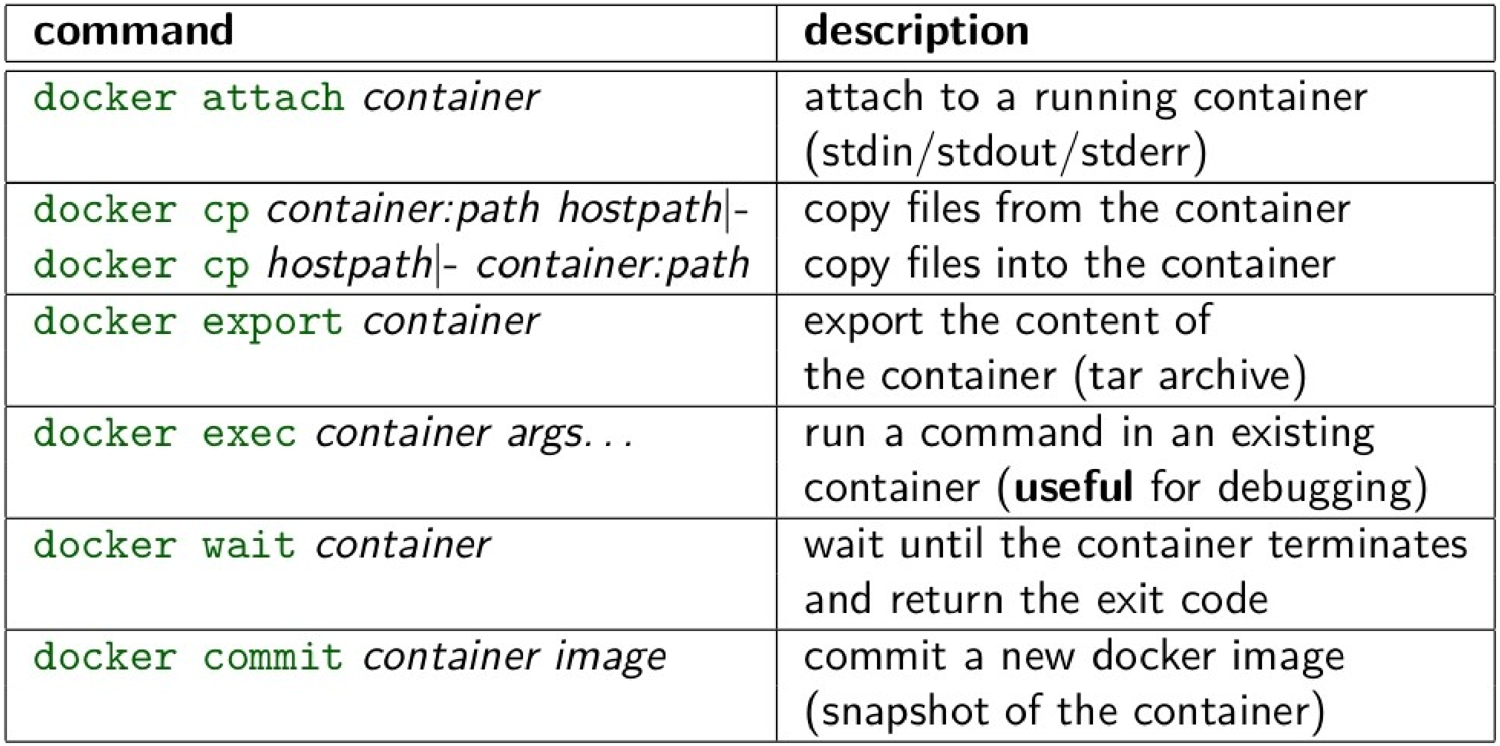
Core commands and options



```
docker run
               --rm  remove container automatically after it exits
                -it  connect the container to terminal
         --name web  name the container
          -p 5000:80  expose port 5000 externally and map to port 80
-v ~/dev:/code  create a host mapped volume inside the container
        alpine:3.4  the image from which the container is instantiated
            /bin/sh  the command to run inside the container
```

Core commands and options

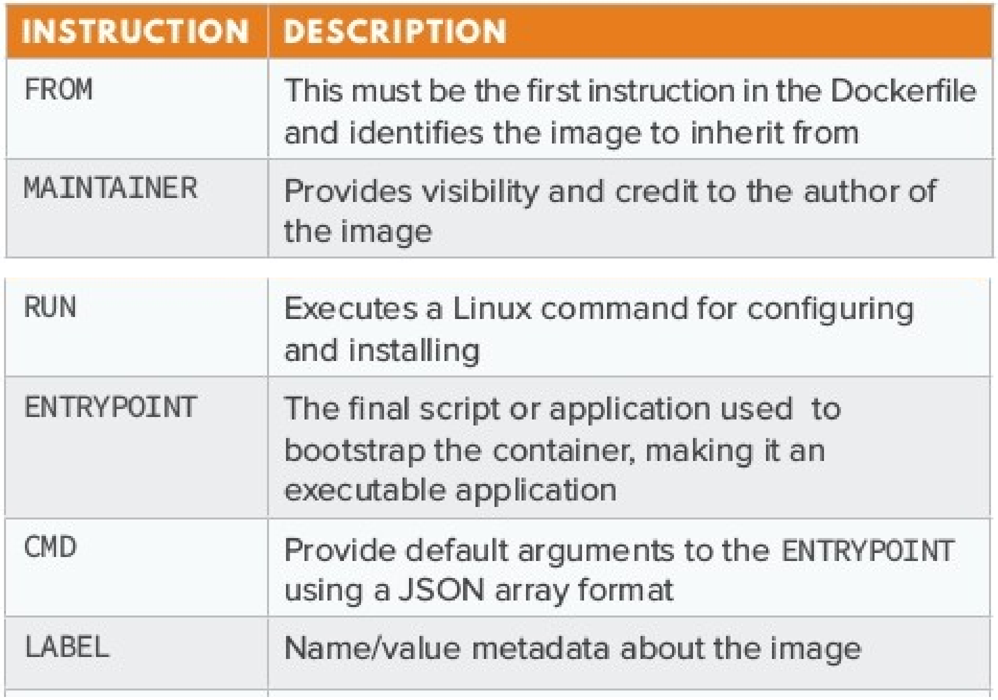
Docker container commands:



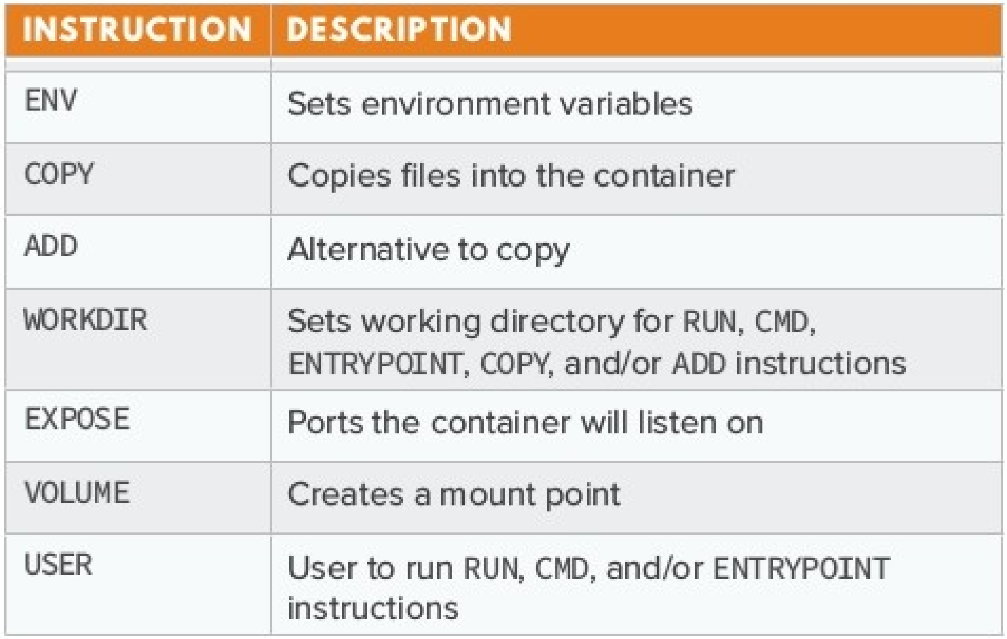
command	**description**
docker attach *container*	attach to a running container (stdin/stdout/stderr)
docker cp *container:path hostpath*|- docker cp *hostpath*|- *container:path*	copy files from the container copy files into the container
docker export *container*	export the content of the container (tar archive)
docker exec *container args...*	run a command in an existing container (**useful** for debugging)
docker wait *container*	wait until the container terminates and return the exit code
docker commit *container image*	commit a new docker image (snapshot of the container)

Dockerfiles

Rather than building an image manually layer by layer, create a script.



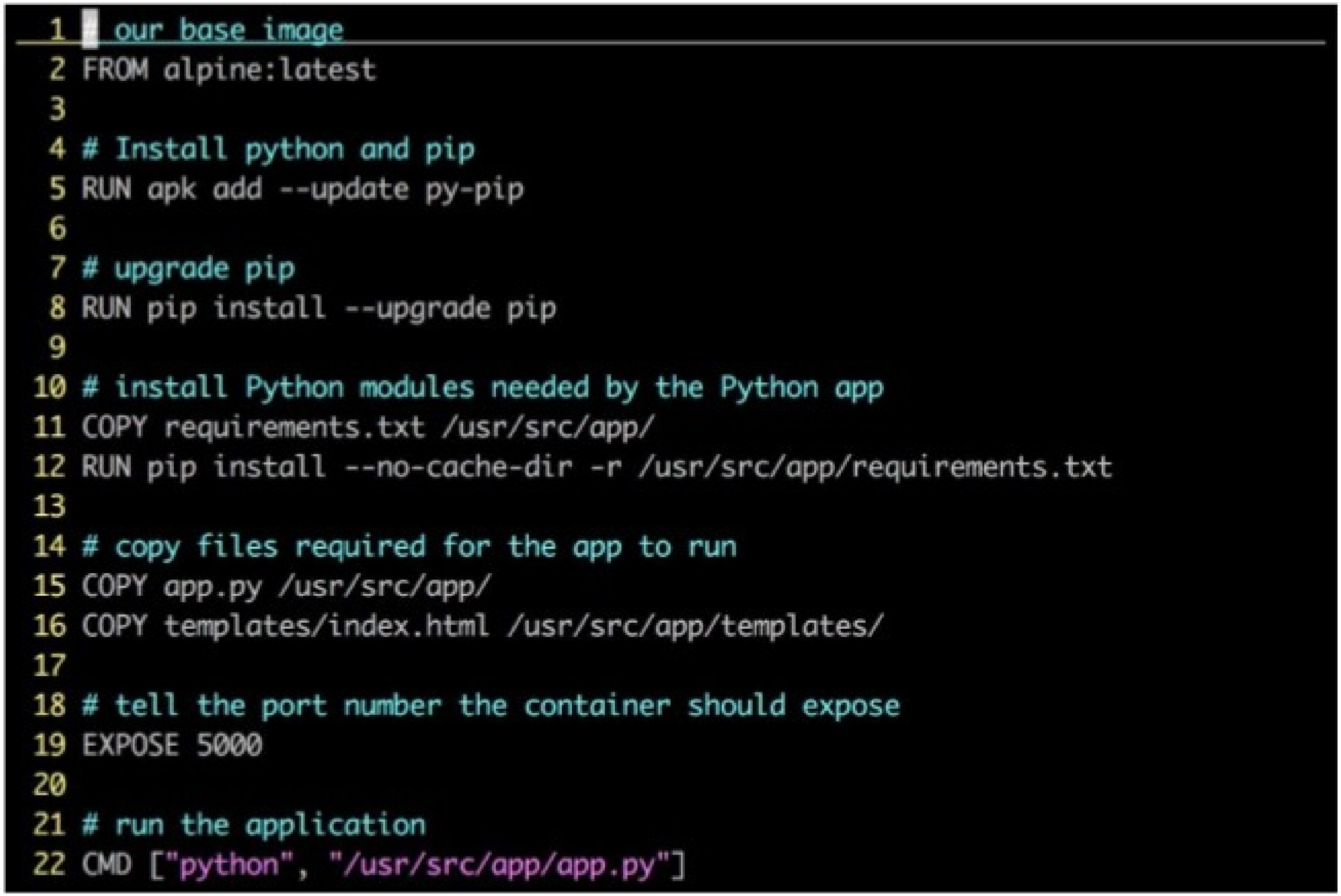
INSTRUCTION	DESCRIPTION
FROM	This must be the first instruction in the Dockerfile and identifies the image to inherit from
MAINTAINER	Provides visibility and credit to the author of the image
RUN	Executes a Linux command for configuring and installing
ENTRYPOINT	The final script or application used to bootstrap the container, making it an executable application
CMD	Provide default arguments to the ENTRYPOINT using a JSON array format
LABEL	Name/value metadata about the image



INSTRUCTION	DESCRIPTION
ENV	Sets environment variables
COPY	Copies files into the container
ADD	Alternative to copy
WORKDIR	Sets working directory for RUN, CMD, ENTRYPOINT, COPY, and/or ADD instructions
EXPOSE	Ports the container will listen on
VOLUME	Creates a mount point
USER	User to run RUN, CMD, and/or ENTRYPOINT instructions

Dockerfiles

Rather than building an image manually layer by layer, create a script.



```
# our base image
FROM alpine:latest

# Install python and pip
RUN apk add --update py-pip

# upgrade pip
RUN pip install --upgrade pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```