

Your First Docker Containers

A step-by-step tutorial and lab

This lab goes with the lecture *Docker Images & Containers Commands*. It assumes you have never used Docker before. By the end you should be able to explain what Docker is, say how an **image** differs from a **container**, and use the everyday commands for both.

Part 0 Getting ready

1. Start **Docker Desktop** and wait until it reports that the engine is running.
2. Open a terminal. On Windows use **PowerShell** or **Windows Terminal**. On macOS or Linux use **Terminal**.
3. Type commands by hand. Text pasted from slides or a word processor often has its dashes replaced by a long dash that Docker does not understand.

Typing options

Options start with one dash and a letter (`-d`) or two dashes and a word (`--name`). There is never a space after the dashes: write `--version`, not `-- version`.

Part 1 What Docker is

1.1 Concept

A program needs more than its own code. It needs the right language runtime, the right libraries and the right settings. If any of these differ on another computer, the program may fail there. This is the “but it works on my machine” problem.

Docker solves it by packing an application *together with everything it needs* into one package. The package runs the same way on your laptop, on a classmate’s laptop and on a server in the cloud.

Image and container

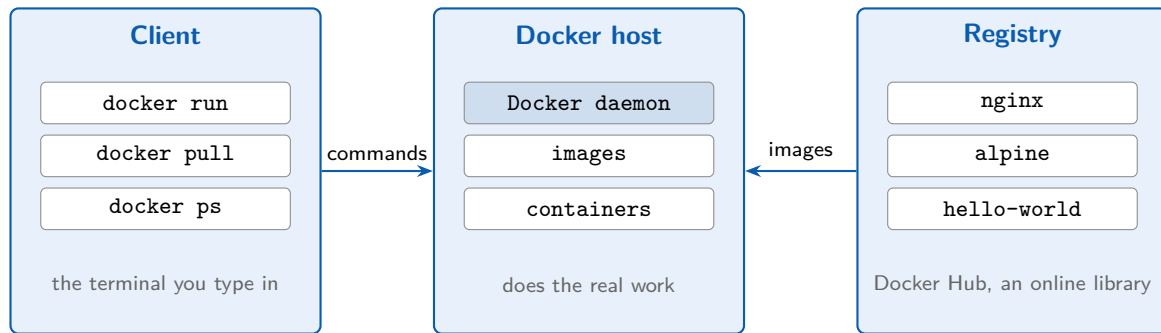
An **image** is a read-only package: the application’s code, a runtime, system tools and libraries. It is stored on disk and does nothing by itself.

A **container** is a running copy made from an image. It can be started, stopped and deleted. One image can produce any number of containers.

Comparison	Image	Container
Cooking	a recipe	a dish cooked from the recipe
Programming	a class	an object created from the class

If you add salt to one dish, the recipe does not change, and neither do the other dishes cooked from it. Containers and images behave the same way.

Docker itself has three parts:



- The **client** is the `docker` command you type. It only passes your request on.
- The **daemon** (also called the server or engine) does the work: it downloads images, and creates, starts and stops containers. Docker Desktop keeps the daemon running on your computer.
- A **registry** is an online store of images. The default one is **Docker Hub**. When you ask for an image that is not on your computer, the daemon downloads it from there.

1.2 Commands

Show the version of the client.

```
docker --version
```

Example output

```
Docker version 27.3.1, build ce12230
```

Show the client and the server. The output has a `Client` section and a `Server` section. These are the first two boxes of the diagram.

```
docker version
```

Run a container from an image. `hello-world` is a tiny test image provided by Docker.

```
docker container run hello-world
```

Example output

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

The four numbered steps in that output are the architecture diagram in words: client, daemon,

registry, image, container. The first lines show the image being downloaded (“pulled”) because it was not yet on your computer.

If Docker is not running

`Cannot connect to the Docker daemon` or `error during connect` means the server side is not running. Start Docker Desktop, wait a minute and try again.

Exercise 1: first contact

1. Run `docker --version` and note which version you have.
2. Run `docker version`. Find the `Client` section and the `Server` section in the output. Do not go on until both are there.
3. Run `docker container run hello-world`.
4. In the output, find the lines that show the image being downloaded, then read the four numbered steps and match each one to a box in the diagram.
5. Run the same command a second time. Compare the two outputs: the download lines are gone, because the image is now stored on your computer and Docker reuses it.

Part 2 Running containers

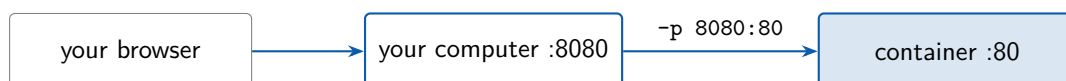
2.1 Concept

`docker container run` does two things: it *creates* a new container from an image and *starts* it. If the image is not on your computer, Docker downloads it first. Every `run` makes a brand-new container, even when the image is the same.

In this part the image is **nginx**, a widely used web server. You do not install nginx. The image already contains it.

Three ideas matter when you run a container:

- **Foreground or background.** By default a container runs attached to your terminal, which stays busy until the container stops. A *detached* container runs in the background and gives the terminal back.
- **Ports.** A container has its own private network. A server inside it cannot be reached from your computer until you *publish* a port.
- **Names.** Every container has a name. If you do not choose one, Docker invents a random one such as `quirky_tesla`.



With `-p 8080:80`, a request to port 8080 on your computer is passed to port 80 inside the container. The order is always **host port first, container port second**.

2.2 Commands

The general form. Square brackets mean “optional”. Only the image is required.

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

Option	Meaning
<code>-d, --detach</code>	Run in the background and print the container ID
<code>-p, --publish</code>	Publish a container port on a port of your computer
<code>--name</code>	Give the container a name
<code>-i, -t</code>	Interactive, with a terminal (Part 5)
<code>--rm</code>	Remove the container automatically when it stops (Part 5)
<code>-e, --env</code>	Set an environment variable (Part 5)
<code>-v, --network, --restart</code>	Volumes, networks and restart policy (later lectures)

Run in the foreground. The terminal stays attached and shows the server's log. The "Welcome to nginx!" page is at <http://localhost:8080>. Press Ctrl + C to stop the container.

```
docker container run -p 8080:80 nginx
```

Run in the background. Docker prints the new container's ID and returns the prompt.

```
docker container run -p 8080:80 -d nginx
```

Example output

```
7c9e4b1f0a6d2e83b5c1f4a97d0e6b2c8a1f3d5e7b9c0a2d4f6e8b1c3a5d7f9e
```

Run in the background with a name.

```
docker container run -p 8080:80 -d --name myweb nginx
```

See which containers are running. Part 3 covers this command fully.

```
docker container ls
```

Example output

CONTAINER ID	IMAGE	COMMAND	STATUS	PORTS	NAMES
7c9e4b1f0a6d	nginx	"/docker-..."	Up 1 minute	0.0.0.0:8080->80/tcp	myweb

Options go before the image name

Anything written *after* the image name is handed to the program inside the container, not to Docker. Write `docker container run -d nginx`, not `docker container run nginx -d`.

Two errors you will meet

`container name "/myweb" is already in use`: names must be unique. Choose another name or remove the old container.

`port is already allocated`: two containers cannot share one host port. Choose another host port, for example `-p 8081:80`.

A note on the lecture slides

The slides use `-p 80:80`. That works too, but port 80 on your own computer is sometimes taken by another program, so this lab uses host ports from 8080 upwards.

Exercise 2: run web servers

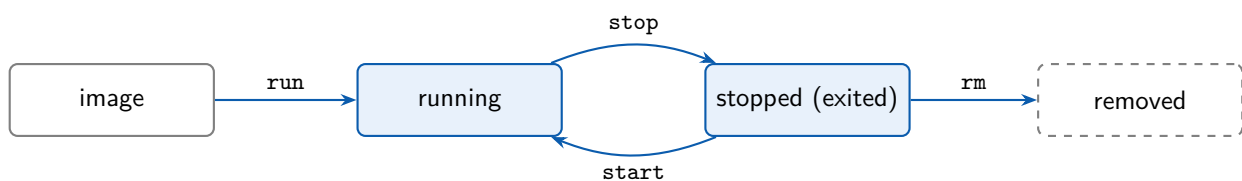
1. Run an nginx container in the *foreground*, publishing container port 80 on host port **8081**.
2. Open **`http://localhost:8081`** in a browser. Refresh the page a few times and watch a new line appear in the terminal for each refresh.
3. Press Ctrl + C to stop the container. Refresh the browser and confirm that the page no longer loads.
4. Run nginx again on host port 8081, this time *detached* and with the name `site1`. Confirm that the page loads and that your terminal is free.
5. Run a second detached nginx container named `site2` on host port **8082**. Open **`http://localhost:8082`**.
6. Run `docker container ls`. Confirm that `site1` and `site2` are both listed, each with its own port in the **PORTS** column.
7. Try to start one more container that is also named `site2`, on host port 8083. Read the error message Docker gives you.

You now have two web servers running from one image. Leave them running for Exercise 3.

Part 3 Managing containers

3.1 Concept

A container moves through a small number of states, and each command below moves it from one state to another.



- A **stopped** container still exists. It keeps its name, its settings and its files, and can be started again. A container also stops by itself when its main program ends, which is what happened to `hello-world`.
- A **removed** container is gone for good.
- Removing a container never removes the image it came from.

Docker has two styles for most commands. Both work, and you will see both online.

Longer style	Shorter style
<code>docker container run</code>	<code>docker run</code>
<code>docker container ls</code>	<code>docker ps</code>
<code>docker container ls -a</code>	<code>docker ps -a</code>
<code>docker container stop</code>	<code>docker stop</code>
<code>docker container rm</code>	<code>docker rm</code>

3.2 Commands

List running containers.

```
docker container ls
```

List all containers, including stopped ones. In the **STATUS** column, **Up** means running and **Exited** means stopped.

```
docker container ls -a
```

Example output

CONTAINER ID	IMAGE	COMMAND	STATUS	NAMES
7c9e4b1f0a6d	nginx	"/docker-..."	Up 5 minutes	myweb
3f2a9c1d7b44	hello-world	"/hello"	Exited (0) 20 minutes ago	brave_curie

Stop a running container.

```
docker container stop myweb
```

Start a stopped container again. It keeps the port and name it was created with.

```
docker container start myweb
```

Remove a stopped container. You can use its name, or the first few characters of its ID.

```
docker container rm myweb
```

Docker refuses to remove a container that is still running:

Example output

```
Error response from daemon: cannot remove container "/myweb": container is running:
stop the container before removing or force remove
```

Stop all containers, then remove all containers. `docker container ls -a -q` prints only the container IDs, and `$( )` passes that list to the outer command.

```
docker container stop $(docker container ls -a -q)
docker container rm $(docker container ls -a -q)
```

In the shorter style these are `docker stop $(docker ps -a -q)` and `docker rm $(docker ps -a -q)`.

Watch out

`$()` works in PowerShell and in Terminal on macOS and Linux, but not in the old Windows Command Prompt (`cmd.exe`). These two commands affect *every* container on the machine. That is fine on your own laptop and dangerous on a shared server.

Exercise 3: manage your containers

This exercise continues from Exercise 2, with `site1` and `site2` running.

1. List the running containers, then list all containers. In the second list, find the stopped `hello-world` containers from Exercise 1 and the `nginx` container you stopped with Ctrl + C.
2. Stop `site2`. Confirm that `http://localhost:8082` no longer loads and that the list of all containers shows `site2` as `Exited`.
3. Start `site2` again and confirm that the page is back.
4. Try to remove `site1` while it is running, and read the error. Then stop it and remove it.
5. Try to run a new container named `site3` on host port 8082, which `site2` is already using. Read the error. List all containers: `site3` was created but could not start. Remove `site3`.
6. Stop all containers with one command, then remove all containers with one command.
7. List all containers and confirm that the list is empty.

Part 4 Looking inside a container

4.1 Concept

A detached container is invisible. Docker gives you three ways to see what it is doing:

- its **logs**: everything the program inside has printed;
- its **processes**: what is running inside it;
- a **shell** inside it, so you can look at its files and run commands there.

A container holds one application and what that application needs. There is no desktop and there are no background services, which is why containers are described as *lightweight*.

Images never change; containers are disposable

Each container has its own private file system, copied from the image. Changes you make inside a container stay in that container. The image is never modified, so every new container starts from the same clean state, and when a container is removed its changes go with it. This is a strength: you always know exactly what you are starting from. Next lecture covers *volumes*, which keep data that must survive.

4.2 Commands

Show a container's logs. For `nginx` there is one line for each page request.

```
docker container logs myweb
```

Example output

```
172.17.0.1 - - [08/Oct/2026:10:15:02 +0000] "GET / HTTP/1.1" 200 615 "-" "Mozilla/5.0 ..."
```

Follow the logs live. New lines appear as they are written. Ctrl + C ends the `logs` command only. The container keeps running.

```
docker container logs -f myweb
```

Show the processes running in a container.

```
docker container top myweb
```

Example output

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	2101	2080	0	10:14	?	00:00:00	nginx: master process nginx -g daemon off;
101	2145	2101	0	10:14	?	00:00:00	nginx: worker process

Open a shell inside a running container. `exec` runs an extra command in a container that is already running. `-it` makes it interactive, and `bash` is the shell to start.

```
docker container exec -it myweb bash
```

Example output

```
root@7c9e4b1f0a6d:/#
```

The prompt changes because you are now inside the container, which is a small Linux system even when your laptop runs Windows or macOS. Some things to try there:

```
ls /
cat /etc/os-release
ls /usr/share/nginx/html
echo "<h1>Hello from my container</h1>" > /usr/share/nginx/html/index.html
exit
```

`/usr/share/nginx/html/index.html` is the page nginx serves, so the `echo` line replaces the welcome page. `exit` leaves the shell. The container keeps running, because nginx, its main program, has not stopped.

Exercise 4: inspect and change a container

1. Run a detached nginx container named `site1` on host port 8081. Open `http://localhost:8081` and refresh the page three times.
2. Show the logs of `site1` and find the lines for your three requests. Each contains `GET / HTTP/1.1`.
3. Follow the logs live, refresh the page and watch a line arrive. Press Ctrl + C, then list the running containers to confirm that `site1` is still up.
4. Show the processes running in `site1`. Notice that all of them belong to nginx.
5. Open a shell inside `site1`. List the folder `/usr/share/nginx/html`, then replace `index.html` with a heading that contains your own name. Leave the shell.
6. Refresh the browser and confirm that your page is shown.

7. Stop and remove `site1`. Then run a new container with the same name, port and image.
8. Refresh the browser (Ctrl + F5 forces a full reload). The original “Welcome to nginx!” page is back. Your edit lived only in the container you removed, and the image was never changed.
9. Stop and remove `site1`.

Part 5 Interactive containers

5.1 Concept

A container does not have to be a background server. It can be a throw-away Linux machine that you work in directly, and then discard.

The `[COMMAND]` part of `run` chooses the program to start inside the container in place of the image’s default. A container lives exactly as long as that main program: when the program ends, the container stops.

Each container is **isolated**. It has its own file system, its own process list, its own network and its own hostname. It cannot see the programs running on your laptop, and whatever goes wrong inside it stays inside it.

5.2 Commands

Start a container and work inside it. `alpine` is a very small Linux image, only a few megabytes. `-i` keeps your keyboard connected and `-t` gives you a proper terminal. They are nearly always written together as `-it`. `sh` is the shell to run.

```
docker container run -it --name sandbox alpine sh
```

Inside, `ps` lists only the shell and `ps` itself, which shows the isolation:

```
hostname
ps
exit
```

Here `exit` stops the container, because `sh` was its main program. Compare this with Part 4, where leaving an `exec` shell left the container running.

Remove the container automatically when it stops. With `--rm` nothing is left behind, which is ideal for quick experiments.

```
docker container run -it --rm alpine sh
```

Run a single command and finish. The container starts, runs `cat`, prints the result and stops.

```
docker container run --rm alpine cat /etc/os-release
```

Set an environment variable in the container. Many images are configured this way. `env` prints all variables, so `COURSE=cloud` appears in its output.

```
docker container run --rm -e COURSE=cloud alpine env
```

Exercise 5: a disposable Linux machine

1. Run an interactive `alpine` container named `box` with the shell `sh`.
2. Inside it, run `hostname`, `ps` and `cat /etc/os-release`. Notice that only two processes exist and that the system reports itself as Alpine Linux.
3. Break the container on purpose: run `rm /bin/ls`, then run `ls`. The command is gone. Leave the container with `exit`.
4. List all containers and confirm that `box` is shown as `Exited`.
5. Run a second interactive `alpine` container, this time with `--rm` and without a name. Run `ls /` inside it. It works: the damage stayed in `box`, and the image is intact. Exit.
6. List all containers again. `box` is still there, but the second container left no trace. Remove `box`.
7. Run one more `alpine` container with `--rm` that sets the variable `STUDENT` to your name and runs `env`. Find `STUDENT` in the output.

Part 6 Images

6.1 Concept

Every container you have run came from an image stored on your computer.

- An image is identified by a **name** and a **tag**, written `name:tag`. The tag is a version label. If you leave it out, Docker uses `latest`, so `nginx` means `nginx:latest`.
- One name can have many tags. `nginx:alpine` is nginx built on the small Alpine Linux, and is much smaller than `nginx:latest`.
- Images are downloaded (**pulled**) from a registry. `run` pulls automatically when an image is missing.
- Images take up disk space until you remove them. Docker will not remove an image while any container, running or stopped, still depends on it.

A note on the lecture slides

Some slides write image commands with the plural, as in `docker images ls`. The management command is the singular `image`: `docker image ls`, `docker image rm` and `docker image prune`. The plural exists only as the short command `docker images` on its own, which lists images.

6.2 Commands

List the images on your computer. The short form is `docker images`.

```
docker image ls
```

Example output

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	fd484f19954f	2 weeks ago	192MB
alpine	latest	91ef0af61f39	5 weeks ago	7.8MB
hello-world	latest	d2c94e258dcb	17 months ago	13.3kB

Download an image without running it.

```
docker image pull nginx:alpine
```

Remove an image. The short form is `docker rmi`. You can give the name or the first few characters of the image ID.

```
docker image rm nginx:alpine
docker rmi fd484f
```

If a container still uses the image, Docker refuses:

Example output

```
Error response from daemon: conflict: unable to remove repository reference "nginx"
(must force) - container 7c9e4b1f0a6d is using its referenced image fd484f19954f
```

The clean fix is to remove that container first.

Force the removal. `-f` removes the image even though a stopped container still refers to it. Use it as a last resort.

```
docker image rm -f nginx
docker rmi -f fd484f
```

Remove unused images in bulk. With `-a`, every image not used by any container is removed. Without `-a`, only “dangling” images (leftovers with no name) are removed. Docker asks you to confirm with `y`.

```
docker image prune -a
```

Exercise 6: work with images

1. List your images. You should have `nginx`, `alpine` and `hello-world`.
2. Pull the image `nginx:alpine`. List the images again and compare the sizes of `nginx:latest` and `nginx:alpine`.
3. Run a detached container named `site1` on host port 8081 from `nginx:alpine`. Confirm that `http://localhost:8081` shows the same welcome page.
4. Try to remove the image `nginx:alpine` and read the error.
5. Stop and remove `site1`, then remove `nginx:alpine` again. This time it works.
6. Remove the `hello-world` image using `docker rmi` and the first characters of its image ID.
7. Remove the `alpine` image by name.
8. List the images and confirm that only `nginx:latest` is left. Keep it for the final exercise.

Part 7 Final exercise

This exercise uses commands from every part. No commands are given. Look back through the handout, or at the summary on the next page, whenever you need to.

Exercise 7: two copies of one website

Your team wants two copies of the same website running at once: one for testing a change and one left untouched for comparison.

1. Start two detached nginx containers named `web1` and `web2`. `web1` must answer on **`http://localhost:8081`** and `web2` on **`http://localhost:8082`**.
2. Show both containers running with a single command.
3. Open a shell inside `web1` and replace its home page with a message of your own.
4. Open both addresses in the browser. `web1` shows your message and `web2` still shows the welcome page: two containers from one image, each with its own files.
5. Show the logs of `web2` only.
6. Show the processes running in `web1`.
7. Stop `web2`, confirm that its page no longer loads, and start it again.
8. Stop all containers and remove all containers.
9. Remove all unused images.
10. Confirm that your machine is clean: the list of all containers and the list of images should both show only a header line.

Command summary

Words in capitals are placeholders that you replace with a real name.

Command	What it does
<code>docker --version</code>	Show the Docker version
<code>docker version</code>	Show client and server details
<code>docker container run IMAGE</code>	Create and start a new container from an image
<code>docker container ls</code>	List running containers (short: <code>docker ps</code>)
<code>docker container ls -a</code>	List all containers, including stopped ones
<code>docker container stop NAME</code>	Stop a running container
<code>docker container start NAME</code>	Start a stopped container again
<code>docker container rm NAME</code>	Remove a stopped container
<code>docker container logs NAME</code>	Show what the container has printed
<code>docker container top NAME</code>	Show the processes inside the container
<code>docker container exec -it NAME bash</code>	Open a shell inside a running container
<code>docker image ls</code>	List images (short: <code>docker images</code>)
<code>docker image pull IMAGE</code>	Download an image from the registry
<code>docker image rm IMAGE</code>	Remove an image (short: <code>docker rmi</code>)
<code>docker image prune -a</code>	Remove all images not used by any container

Options for `docker container run`

Option	Meaning
<code>-d</code>	Detach: run in the background and print the container ID
<code>-p HOST:CONTAINER</code>	Publish a container port on a port of your computer
<code>--name NAME</code>	Choose the container's name
<code>-i</code> and <code>-t</code>	Interactive, with a terminal. Normally written together as <code>-it</code>
<code>--rm</code>	Remove the container automatically when it stops
<code>-e KEY=VALUE</code>	Set an environment variable inside the container

When something goes wrong

Message contains	What to do
<code>Cannot connect to the Docker daemon</code>	Docker Desktop is not running. Start it and wait.
<code>port is already allocated</code>	Another container or program uses that host port. Choose another.
<code>container name ... is already in use</code>	A container with that name exists, perhaps stopped. Remove it or pick a new name.
<code>container is running: stop the container</code>	Stop the container before removing it.
<code>unable to remove repository reference</code>	A container still uses the image. Remove that container first.
<code>unknown flag</code> or <code>unknown shorthand flag</code>	Check the dashes: no space after them, and typed by hand.
<code>No such container</code>	Check the spelling with <code>docker container ls -a</code> .
<code>requires at least 1 argument</code>	The <code>\$()</code> list was empty: there was nothing to stop or remove.

Reference

Docker documentation: <https://docs.docker.com>