Docker Containers and images

START HERE

`docker run -d -p <host>:<ctr> <image>`
Start a container

`docker stop <container>`
Stop it

`docker ps`
List running containers

`docker build -t <name> .`
Build an image

`docker logs -f <container>`
Follow its output

`docker compose up -d`
Start a whole stack

`docker exec -it <container> sh`
Open a shell inside

`docker system prune`
Clean up unused data

Dockerfile recipe → `docker build` → **Image** template → `docker run` → **Container** running instance → `docker push` → **Registry** Docker Hub

CONCEPTS

An image is a read-only template; a container is a running instance of it. Removing a container does not remove its image.

Data written inside a container is lost when it is removed. Keep data in a volume or bind mount.

Containers on the same user-defined network reach each other by name.

RUN

`docker run <image>`
Create and start a container

`docker run -d <image>`
Run in the background

`docker run -it <image> sh`
Interactive shell

`docker run --rm <image>`
Delete the container on exit
Also removes its anonymous volumes.

`docker run --name <name> <image>`
Give the container a name

`docker run -p <host>:<ctr> <image>`
Publish a port
Format is [HOST_IP:]HOST_PORT:CONTAINER_PORT[/PROTO]; bind 127.0.0.1:80:8080 to keep it local.

`docker run -v <name>:<path> <image>`
Mount a named volume

`docker run -v <dir>:<path> <image>`
Bind-mount a host folder

`docker run -e KEY=VALUE <image>`
Set an environment variable
Passing only KEY copies it from your shell.

`docker run --env-file <file> <image>`
Load variables from a file

`docker run --network <net> <image>`
Join a network

`docker run --restart unless-stopped <image>`
Restart after crashes and reboots
Policies: no, on-failure[:max], always, unless-stopped.

`docker run -w <dir> <image>`
Set the working directory

CONTAINERS

`docker ps`
List running containers

`docker ps -a`
List all, including stopped

`docker stop <container>`
Stop gracefully
Sends SIGTERM, then SIGKILL after 10 s on Linux.

`docker start <container>`
Start a stopped container

`docker restart <container>`
Stop and start again

`docker rm <container>`
Remove a stopped container

`docker rm -f <container>`
Force-remove a running one

`docker ps -q`
Print only container IDs

`docker ps --filter status=exited`
Filter the list

`docker ps --format json`
Format output as JSON

`docker stats`
Live CPU and memory usage

INSIDE A CONTAINER

`docker exec -it <container> sh`
Open a shell in a running container

`docker exec <container> <cmd>`
Run one command
Wrap chains in a shell: sh -c "a && b".

`docker logs <container>`
Show output

`docker logs -f <container>`
Follow output live

`docker logs --tail <n> <container>`
Only the last lines

`docker logs --since <duration> <container>`
Only recent output, e.g. 10m

`docker cp <container>:<path> <dest>`
Copy out of a container

`docker cp <src> <container>:<path>`
Copy into a container

`docker inspect <object>`
Low-level details as JSON

IMAGES

`docker pull <image>`
Download an image
No tag means :latest.

IMAGES (cont.)

`docker images`
List local images

`docker build -t <name> .`
Build from a Dockerfile here

`docker build -f <file> -t <name> .`
Use another Dockerfile

`docker build --build-arg KEY=VALUE .`
Pass a build variable

`docker build --no-cache .`
Ignore the build cache

`docker tag <image> <repo>:<tag>`
Add a name to an image

`docker push <repo>:<tag>`
Upload to a registry
Private registries need the host in the name: host:5000/repo:tag.

`docker login`
Log in to a registry

`docker rmi <image>`
Remove an image

VOLUMES AND NETWORKS

`docker volume create <name>`
Create a named volume

`docker volume ls`
List volumes

`docker volume rm <name>`
Remove a volume

`docker volume inspect <name>`
Show volume details

`docker volume prune`
Remove unused volumes

`docker network create <name>`
Create a bridge network
Bridge is the default driver.

`docker network ls`
List networks

`docker network connect <net> <container>`
Attach a running container

COMPOSE

`docker compose up -d`
Create and start the stack

`docker compose down`
Stop and remove containers and networks

`docker compose ps`
List the stack's containers

`docker compose logs -f <service>`
Follow service logs

COMPOSE (cont.)

`docker compose exec <service> sh`
Shell in a service

`docker compose up --build`
Rebuild images, then start

`docker compose build`
Build service images

`docker compose pull`
Pull service images

`docker compose restart <service>`
Restart a service

`docker compose stop`
Stop without removing

`docker compose run --rm <service> <cmd>`
One-off command in a service

`docker compose down -v`
Also remove volumes
Deletes named volumes declared in the file: data is lost.

`docker compose config`
Show the resolved file

CLEANUP AND INFO

`docker system df`
Show disk usage

`docker system prune`
Remove stopped containers and dangling data
Volumes are kept unless you add --volumes.

`docker system prune -a`
Also remove unused images

`docker version`
Client and server versions

RECIPES

Web app with a database

```
docker network create app
docker run -d --name db --
network app \
  -v pgdata:/var/lib/
postgresql/data \
  -e POSTGRES_PASSWORD=<pw>
postgres
docker run -d --network app -p
8080:80 <image>
```

Throwaway shell with your code

```
docker run --rm -it -v "$PWD":/
work -w /work \
  <image> sh
```

Rebuild and follow a service

```
docker compose up -d --build
<service>
docker compose logs -f
<service>
```

Reset everything for a project

```
docker compose down -v --
remove-orphans
docker system prune -f
```