Docker Containers and images

START HERE

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

`docker compose up -d`
Start a whole stack

`docker ps`
List running containers

`docker system prune`
Clean up unused data

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

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

`docker stop` *`<container>`*
Stop it

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

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

RUN (cont.)

`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.

CONTAINERS

`docker ps`
List running containers

CONTAINERS (cont.)

`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 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".

INSIDE A CONTAINER (cont.)

`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.

`docker images`
List local images

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

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

IMAGES (cont.)

`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 prune`
Remove unused volumes

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

VOLUMES AND NETWORKS (cont.)

`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

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

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

`docker compose build`
Build service images

CLEANUP AND INFO

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