DOCKER DEVOPS ENGINEER CHEAT SHEET

Legend for explanations

For every command you will see:

Command pattern	: The general syntax.
Word-by-word	: Explanation of each word/flag in that command.
Example	: A realistic DevOps-style example.
Why it’s used	: When/why a DevOps engineer runs it in real life.

1. Check Docker installation and version

Command pattern:
```
  docker --version
```

Word-by-word:

docker	-> The Docker command-line client.
--version	-> Prints the Docker client version information.

Example:
```
  docker --version
```

Why it’s used:
  - To verify that Docker is installed and accessible in PATH.
  - To check which version is running before debugging, upgrading, or matching versions across environments.

2. Get general Docker system info

Command pattern:
```
  docker info
```

Word-by-word:

docker	-> Docker CLI.
info	-> Shows detailed information about the Docker daemon and system (number of containers, images, storage driver, etc.).

Example:
```
  docker info
```

Why it’s used:
  - To quickly see how many containers/images are present.
  - To check storage drivers, cgroup drivers, root directory, etc.
  - Often used when debugging Docker host issues.

3. List local images

```
Command pattern:
  docker images

Word-by-word:
  docker   -> Docker CLI.
  images   -> Lists all images stored locally on this Docker host.

Example:
  docker images

Why it’s used:
  - To see which images are already downloaded/built.
  - To check image sizes and tags before cleaning up or pushing.

--------------------------------------------------------------------------------
4. Search images on Docker Hub
--------------------------------------------------------------------------------

Command pattern:
  docker search IMAGE_NAME

Word-by-word:
  docker       -> Docker CLI.
  search       -> Searches image names on Docker Hub (or configured registry).
  IMAGE_NAME   -> Name or keyword of the image to search for.

Example:
  docker search nginx

Why it’s used:
  - To discover official or popular community images for a technology.
  - To see basic info like stars and descriptions before pulling.

--------------------------------------------------------------------------------
5. Pull an image from registry
--------------------------------------------------------------------------------

Command pattern:
  docker pull IMAGE[:TAG]

Word-by-word:
  docker       -> Docker CLI.
  pull         -> Download an image from a registry.
  IMAGE        -> The image name (e.g., nginx, httpd, ubuntu).
  :TAG         -> Optional tag (version), defaults to "latest" if omitted.

Example:
  docker pull nginx:1.25

Why it’s used:
  - To download a specific image version before running containers.
  - Ensures all environments are using the same image tag.

--------------------------------------------------------------------------------
6. Remove a local image
```

Command pattern:

```
docker rmi IMAGE
```

Word-by-word:

docker	-> Docker CLI.
rmi	-> Remove image (delete it from local cache).
IMAGE	-> Image name or ID.

Example:

```
docker rmi nginx:1.25
```

Why it’s used:
- To clean up unused images and free disk space.
- To remove wrong/broken images before pulling the correct one.

7. Run a container (interactive + detached + named)

Command pattern:

```
docker run -itd --name CONTAINER_NAME IMAGE
```

Word-by-word:

docker	-> Docker CLI.
run	-> Create and start a new container from an image.
-i	-> Keep STDIN open (interactive).
-t	-> Allocate a pseudo-TTY (terminal).
-d	-> Run container in detached (background) mode.
--name	-> Assign a custom name to the container.
CONTAINER_NAME	-> The custom container name you choose.
IMAGE	-> The image to run (e.g., httpd, nginx, ubuntu).

Example:

```
docker run -itd --name ct1 httpd
```

Why it’s used:
- Quickly start a background web server with a readable name.
- -itd is common when you want a container that you can exec into later,
  but that runs in the background as a service.

8. Run a container with port mapping

Command pattern:

```
docker run -d --name CONTAINER_NAME -p HOST_PORT:CONTAINER_PORT IMAGE
```

Word-by-word:

docker	-> Docker CLI.
run	-> Create and start a container.
-d	-> Detached mode (run in background).
--name	-> Set container name.
CONTAINER_NAME	-> Custom container name.

```
  -p                    -> Publish/forward container port to host.
  HOST_PORT:CONTAINER_PORT
                        -> HOST_PORT is port on host machine,
                           CONTAINER_PORT is port inside the container.
  IMAGE                 -> Image name.

Example:
  docker run -d --name web1 -p 8080:80 nginx

Why it’s used:
  - To expose services running inside the container to the outside world.
  - Standard DevOps pattern for running web apps, APIs, dashboards, etc.

-------------------------------------------------------------------------------
9. Run a container with a bind mount (host directory into container)
-------------------------------------------------------------------------------

Command pattern:
  docker run -d --name CONTAINER_NAME -v HOST_DIR:CONTAINER_DIR IMAGE

Word-by-word:
  docker                -> Docker CLI.
  run                   -> Create and start container.
  -d                    -> Detached mode.
  --name                -> Container name.
  CONTAINER_NAME        -> Your chosen name.
  -v                    -> Volume/bind mount flag.
  HOST_DIR:CONTAINER_DIR
                        -> HOST_DIR is a path on host,
                           CONTAINER_DIR is path inside container.
  IMAGE                 -> Image to run.

Example:
  docker run -d --name web2 -p 8081:80 -v /opt/html:/usr/share/nginx/html nginx

Why it’s used:
  - To persist data or serve local application code into containers.
  - Common for development setups or when external config is stored on host.

-------------------------------------------------------------------------------
10. List running containers
-------------------------------------------------------------------------------

Command pattern:
  docker ps

Word-by-word:
  docker   -> Docker CLI.
  ps       -> "Process status" style listing of containers (running only).

Example:
  docker ps

Why it’s used:
  - To see currently running containers, names, IDs, ports, and uptime.
```

  - First step when debugging "is my container running?" questions.

11. List ALL containers (running + stopped)

Command pattern:

```
  docker ps -a
```

Word-by-word:

docker	-> Docker CLI.
ps	-> List containers.
-a	-> Show all containers (including exited/stopped).

Example:

```
  docker ps -a
```

Why it’s used:
  - To see history of containers, including crashed or stopped ones.
  - Helpful when cleaning up or investigating failures.

12. Stop a running container

Command pattern:

```
  docker stop CONTAINER
```

Word-by-word:

docker	-> Docker CLI.
stop	-> Gracefully stop a running container (SIGTERM, then SIGKILL).
CONTAINER	-> Container name or ID.

Example:

```
  docker stop web1
```

Why it’s used:
  - To gracefully stop a service/container before redeploy, upgrade or debug.

13. Start a stopped container

Command pattern:

```
  docker start CONTAINER
```

Word-by-word:

docker	-> Docker CLI.
start	-> Start an existing, stopped container.
CONTAINER	-> Container name or ID.

Example:

```
  docker start web1
```

Why it’s used:

```
  - To bring back a previously created container without creating a new one.

--------------------------------------------------------------------------------
14. Restart a container
--------------------------------------------------------------------------------

Command pattern:
  docker restart CONTAINER

Word-by-word:
  docker      -> Docker CLI.
  restart     -> Stop and then start the container.
  CONTAINER   -> Container name or ID.

Example:
  docker restart web1

Why it’s used:
  - Quick way to apply environment changes or recover from transient issues.

--------------------------------------------------------------------------------
15. Remove a container
--------------------------------------------------------------------------------

Command pattern:
  docker rm CONTAINER

Word-by-word:
  docker      -> Docker CLI.
  rm          -> Remove (delete) a container.
  CONTAINER   -> Container name or ID.

Example:
  docker rm web1

Why it’s used:
  - To clean up stopped containers and free names/resources.
  - Typical step after testing or replacing deployments.

--------------------------------------------------------------------------------
16. Force remove a running container
--------------------------------------------------------------------------------

Command pattern:
  docker rm -f CONTAINER

Word-by-word:
  docker      -> Docker CLI.
  rm          -> Remove container.
  -f          -> Force removal (sends SIGKILL if needed).
  CONTAINER   -> Container name or ID.

Example:
  docker rm -f web1
```

Why it’s used:
  - When a container does not stop gracefully or is stuck/hung.
  - Useful in emergency cleanup scripts, but use with care.

----

17. View logs of a container

Command pattern:
```
  docker logs CONTAINER
  docker logs -f CONTAINER
```

Word-by-word:

docker	-> Docker CLI.
logs	-> Show logs (STDOUT/STDERR) of a container.
-f	-> "Follow" the logs in realtime (like tail -f).
CONTAINER	-> Container name or ID.

Example:
```
  docker logs -f web1
```

Why it’s used:
  - To debug application behavior inside containers.
  - -f is heavily used for live debugging and monitoring during incidents.

----

18. Execute a command inside a running container

Command pattern:
```
  docker exec -it CONTAINER COMMAND [ARGS...]
```

Word-by-word:

docker	-> Docker CLI.
exec	-> Run a command in an already running container.
-i	-> Keep STDIN open.
-t	-> Allocate a pseudo-TTY.
CONTAINER	-> Container name or ID.
COMMAND	-> The command to run (e.g., bash, sh).
[ARGS...]	-> Optional extra arguments for that command.

Example:
```
  docker exec -it web1 /bin/bash
```

Why it’s used:
  - To "ssh into" a container for debugging (without actual SSH).
  - To run inspection commands like ls, ps, curl inside container context.

----

19. Attach to a running container

Command pattern:
```
  docker attach CONTAINER
```

Word-by-word:

docker	-> Docker CLI.
attach	-> Attach your terminal to a running container’s main process.
CONTAINER	-> Container name or ID.

Example:
```
  docker attach web1
```

Why it’s used:
- To see live output or interact with the main process (if it is interactive).
- Less commonly used than exec for debugging, but useful in some scenarios.

20. Copy files between host and container

Command pattern:
```
  docker cp SRC_PATH CONTAINER:DEST_PATH
  docker cp CONTAINER:SRC_PATH DEST_PATH
```

Word-by-word:

docker	-> Docker CLI.
cp	-> Copy files/directories.
SRC_PATH	-> File or directory path on host or container.
CONTAINER:DEST_PATH	-> Destination path inside container.
CONTAINER:SRC_PATH	-> Source inside container.
DEST_PATH	-> Destination on host.

Example:
```
  docker cp web1:/var/log/nginx/access.log ./access.log
```

Why it’s used:
- To extract logs, config files, or artifacts from containers.
- To copy new config or scripts into running containers (for debugging).

21. Inspect container details (low-level JSON)

Command pattern:
```
  docker inspect CONTAINER_OR_IMAGE
```

Word-by-word:

docker	-> Docker CLI.
inspect	-> Show detailed JSON metadata.
CONTAINER_OR_IMAGE	-> Name or ID of container or image.

Example:
```
  docker inspect web1
```

Why it’s used:
- To see container IP, mounts, env variables, entrypoint, etc.
- Often used in scripts and debugging network/storage issues.

22. Show processes running inside a container

Command pattern:
```
  docker top CONTAINER
```

Word-by-word:
```
  docker      -> Docker CLI.
  top         -> Show running processes (like Linux top/ps).
  CONTAINER   -> Container name or ID.
```

Example:
```
  docker top web1
```

Why it’s used:
- To see what processes are running inside the container.
- Quick health check during incidents.

----
23. Show resource usage for containers

Command pattern:
```
  docker stats
  docker stats CONTAINER
```

Word-by-word:
```
  docker      -> Docker CLI.
  stats       -> Live stream of CPU, memory, network, I/O usage.
  CONTAINER   -> Optional specific container name/ID.
```

Example:
```
  docker stats web1
```

Why it’s used:
- To monitor performance and resource usage in real-time.
- Helpful to identify memory leaks or high CPU containers.

----
24. Build an image from a Dockerfile

Command pattern:
```
  docker build -t IMAGE_NAME[:TAG] PATH
```

Word-by-word:
```
  docker               -> Docker CLI.
  build                -> Build an image from a Dockerfile context.
  -t                   -> Tag for the resulting image.
  IMAGE_NAME[:TAG]     -> Name (and optional tag) of the image to create.
  PATH                 -> Build context path (directory with Dockerfile).
```

Example:
```
  docker build -t myapp:1.0 .
```

Why it’s used:
- To create custom application images from source code and Dockerfile.
- Standard DevOps workflow for CI/CD pipelines.

25. Tag an image

Command pattern:

```
  docker tag SOURCE_IMAGE TARGET_IMAGE
```

Word-by-word:

docker	-> Docker CLI.
tag	-> Assign a new tag (name) to an existing image.
SOURCE_IMAGE	-> Existing image name or ID.
TARGET_IMAGE	-> New repository:tag name (often with registry prefix).

Example:

```
  docker tag myapp:1.0 myregistry.example.com/team/myapp:1.0
```

Why it’s used:
- To prepare images with proper registry/repository names before pushing.
- To create versioned tags for releases.

26. Push an image to a registry

Command pattern:

```
  docker push IMAGE
```

Word-by-word:

docker	-> Docker CLI.
push	-> Upload an image to a registry.
IMAGE	-> Image name (must include registry/repo if not Docker Hub).

Example:

```
  docker push myregistry.example.com/team/myapp:1.0
```

Why it’s used:
- To publish images so that other environments (staging, prod, k8s nodes) can pull and run them.

27. Login / Logout to a registry

Command patterns:

```
  docker login [REGISTRY]
  docker logout [REGISTRY]
```

Word-by-word:

docker	-> Docker CLI.
login	-> Authenticate to a registry with username/password or token.
logout	-> Clear local auth credentials for registry.

```
  [REGISTRY]  -> Optional registry hostname (default is Docker Hub).

Example:
  docker login myregistry.example.com
  docker logout myregistry.example.com

Why it’s used:
  - Required before pushing/pulling private images.
  - DevOps uses this in automation scripts and CI/CD.

--------------------------------------------------------------------------------
28. List networks
--------------------------------------------------------------------------------

Command pattern:
  docker network ls

Word-by-word:
  docker        -> Docker CLI.
  network       -> Manage container networks.
  ls            -> List objects (here: networks).

Example:
  docker network ls

Why it’s used:
  - To see bridge, host, overlay, and custom networks.
  - Useful when debugging connectivity between containers.

--------------------------------------------------------------------------------
29. Create a user-defined bridge network
--------------------------------------------------------------------------------

Command pattern:
  docker network create NETWORK_NAME

Word-by-word:
  docker        -> Docker CLI.
  network       -> Network management command group.
  create        -> Create a new network.
  NETWORK_NAME  -> Name of the new Docker network.

Example:
  docker network create app-net

Why it’s used:
  - To isolate applications on custom networks and control DNS/aliases.
  - Important for multi-container app topologies.

--------------------------------------------------------------------------------
30. Connect / disconnect containers to network
--------------------------------------------------------------------------------

Command patterns:
  docker network connect NETWORK_NAME CONTAINER
docker network disconnect NETWORK_NAME CONTAINER
```

Word-by-word:

docker	-> Docker CLI.
network	-> Network management group.
connect	-> Attach container to network.
disconnect	-> Detach container from network.
NETWORK_NAME	-> Target network.
CONTAINER	-> Container name or ID.

Example:

```
docker network connect app-net web1
```

Why it’s used:
  - To dynamically connect services to different networks.
  - Helpful when restructuring topology without recreating containers.

----
31. List volumes

Command pattern:

```
docker volume ls
```

Word-by-word:

docker	-> Docker CLI.
volume	-> Volume management group.
ls	-> List volumes.

Example:

```
docker volume ls
```

Why it’s used:
  - To see named volumes used for persistent data.
  - Useful in backup and cleanup operations.

----
32. Create a named volume

Command pattern:

```
docker volume create VOLUME_NAME
```

Word-by-word:

docker	-> Docker CLI.
volume	-> Volume management group.
create	-> Create a new named volume.
VOLUME_NAME	-> Name of the volume.

Example:

```
docker volume create db-data
```

Why it’s used:
  - To persist databases or stateful data beyond container lifecycle.

33. Run container with a named volume

Command pattern:

```
docker run -d --name CONTAINER_NAME -v VOLUME_NAME:CONTAINER_DIR IMAGE
```

Word-by-word:

docker	-> Docker CLI.
run	-> Start a new container.
-d	-> Detached mode.
--name	-> Container name.
CONTAINER_NAME	-> Name of container.
-v	-> Volume mount flag.
VOLUME_NAME:CONTAINER_DIR	-> Named volume to container path mapping.
IMAGE	-> Image name.

Example:

```
docker run -d --name db1 -v db-data:/var/lib/mysql mysql:8
```

Why it’s used:
- To keep database data across container restarts/redeployments.

34. System-wide Docker disk usage

Command pattern:

```
docker system df
```

Word-by-word:

docker	-> Docker CLI.
system	-> Manage Docker as a whole (system-level).
df	-> Disk usage summary ("disk free" style).

Example:

```
docker system df
```

Why it’s used:
- To see how much space images, containers, volumes, and build cache use.

35. Prune unused data (careful!)

Command patterns:

```
docker system prune
docker system prune -a
```

Word-by-word:

docker	-> Docker CLI.
system	-> System management group.
prune	-> Remove unused data (containers, networks, build cache).
-a	-> Also remove unused images (not just dangling ones).

```
Example:
  docker system prune -a

Why it’s used:
  - To clean up disk space on Docker hosts.
  - Common DevOps maintenance task; dangerous in production if misused.

--------------------------------------------------------------------------------
36. Remove all stopped containers
--------------------------------------------------------------------------------

Command pattern:
  docker container prune

Word-by-word:
  docker       -> Docker CLI.
  container    -> Container management group.
  prune        -> Remove all stopped containers after confirmation.

Example:
  docker container prune

Why it’s used:
  - To clean up old/stopped containers regularly.

--------------------------------------------------------------------------------
37. Remove unused images
--------------------------------------------------------------------------------

Command pattern:
  docker image prune
  docker image prune -a

Word-by-word:
  docker   -> Docker CLI.
  image    -> Image management group.
  prune    -> Remove unused images.
  -a       -> Remove all unused images, not just dangling.

Example:
  docker image prune -a

Why it’s used:
  - To recover disk space taken by unused images (build leftovers, old tags).

--------------------------------------------------------------------------------
38. Docker Compose: up services
--------------------------------------------------------------------------------

Command pattern (Compose v2):
  docker compose up
  docker compose up -d

Word-by-word:
  docker           -> Docker CLI.
  compose          -> Docker Compose v2 subcommand (multi-container apps).
  up               -> Create and start services defined in compose file.
  -d               -> Run in detached mode (background).

Example:
  docker compose up -d

Why it’s used:
  - To start full application stacks (db + api + frontend + cache) with one
    command based on docker-compose.yml.

--------------------------------------------------------------------------------
39. Docker Compose: stop and remove services
--------------------------------------------------------------------------------

Command pattern:
  docker compose down

Word-by-word:
  docker           -> Docker CLI.
  compose          -> Compose command group.
  down             -> Stop and remove containers, networks, etc created by up.

Example:
  docker compose down

Why it’s used:
  - To tear down a complete stack cleanly between tests or deployments.

--------------------------------------------------------------------------------
40. Docker Compose: view service logs
--------------------------------------------------------------------------------

Command pattern:
  docker compose logs
  docker compose logs -f SERVICE

Word-by-word:
  docker         -> Docker CLI.
  compose        -> Compose command group.
  logs           -> Show logs from all or specific services.
  -f             -> Follow logs in real time.
  SERVICE        -> Service name from compose file (e.g., web, db).

Example:
  docker compose logs -f web

Why it’s used:
  - To debug multi-service applications with a single command.
  - Essential during development and production incident triage.

--------------------------------------------------------------------------------
41. Docker Compose: list running services
--------------------------------------------------------------------------------
```

```
Command pattern:
  docker compose ps

Word-by-word:
  docker   -> Docker CLI.
  compose  -> Compose group.
  ps       -> Show containers/services managed by Compose stack.

Example:
  docker compose ps

Why it’s used:
  - To quickly see which services are up, their ports, and status.

--------------------------------------------------------------------------------
42. Show image history (layers)
--------------------------------------------------------------------------------

Command pattern:
  docker history IMAGE

Word-by-word:
  docker   -> Docker CLI.
  history  -> Show history of image layers.
  IMAGE    -> Image name or ID.

Example:
  docker history myapp:1.0

Why it’s used:
  - To debug image size, build steps, and layering.
  - Helpful for optimizing Dockerfiles.

--------------------------------------------------------------------------------
43. Save and load images (tar files)
--------------------------------------------------------------------------------

Command patterns:
  docker save -o FILE.tar IMAGE
  docker load -i FILE.tar

Word-by-word:
  docker        -> Docker CLI.
  save          -> Export image as tar archive.
  -o FILE.tar   -> Output tar file path.
  IMAGE         -> Image name/ID to export.
  load          -> Import image from tar archive.
  -i FILE.tar   -> Input tar file.

Example:
  docker save -o myapp.tar myapp:1.0
  docker load -i myapp.tar

Why it’s used:
  - To move images between air-gapped environments or without registry.
```

44. Export and import containers (filesystem)

Command patterns:

```
  docker export CONTAINER > FILE.tar
  docker import FILE.tar IMAGE_NAME
```

Word-by-word:

docker	-> Docker CLI.
export	-> Export container filesystem to tar (no history).
CONTAINER	-> Container name or ID.
> FILE.tar	-> Shell redirection to file.
import	-> Create image from tarball.
IMAGE_NAME	-> Name for new image.

Example:

```
  docker export web1 > web1_fs.tar
  docker import web1_fs.tar web1-image:fs
```

Why it’s used:
- To create images from running containers or share entire filesystem state.

Quick mental model for "docker run"

Think of:

```
  docker run [OPTIONS] IMAGE [COMMAND] [ARGS...]
```

- docker	: The CLI tool.
- run	: "Create + start container from image".
- [OPTIONS]	: Flags for interactive mode, ports, volumes, env, etc.
- IMAGE	: Template filesystem + default command.
- [COMMAND]	: Optional override of default command.
- [ARGS...]	: Arguments for that command.

Common patterns you will use daily as a DevOps engineer:

```
  docker run -it --rm IMAGE bash
  docker run -d --name NAME -p HOST:CONTAINER IMAGE
  docker run -d --name NAME -v HOST:CONTAINER IMAGE
  docker exec -it NAME bash
  docker logs -f NAME
  docker compose up -d
  docker compose logs -f SERVICE
```

Use this sheet as a quick map from "command words" to their meaning and day to day DevOps use cases.