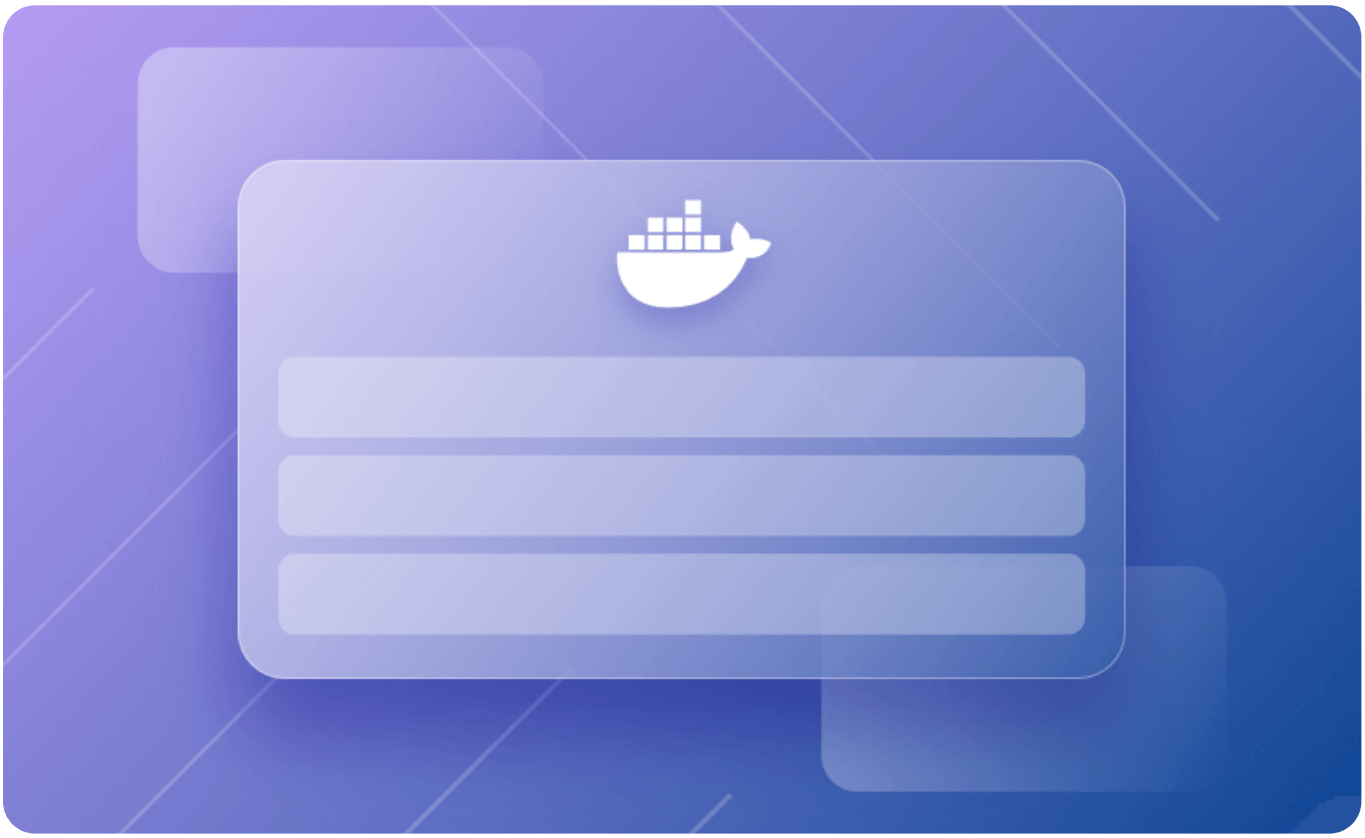
DOCKER

Docker Cheat Sheet – 36 Docker CLI Commands

James Walker

23 Jun 2023 · 15 min read



Need a quick reference for commonly used `docker` CLI commands? You're in the right place.

The `docker` CLI is the command-line tool used to interact with a Docker installation. You need it to start containers, build images, interact with the Docker resources on your machine, and manage configuration settings.

In this article, we'll provide a brief description for the majority of the main `docker` commands. You can use this list to learn more about Docker's functionality, or simply find out how to use a particular feature.

Docker CLI Cheat Sheet

Here's a quick table of contents:

If you need more help installing and getting started with Docker, check out our Docker tutorial for beginners.

Let's dive into the cheat sheet!

General Commands

First up, here are some basics to get you started:

`docker version` – Displays detailed information about your Docker CLI and daemon versions.

`docker system info` – Lists data about your Docker environment, including active plugins and the number of containers and images on your system.

`docker help` – View the help index, a reference of all the supported commands.

`docker <command> --help` – View the help information about a particular command, including detailed information on the supported option flags.

Build Images

These commands relate to building new images from Dockerfiles:

`docker build .` – Build the Dockerfile in your working directory into a new image.

`docker build -t example-image:latest .` – Build the Dockerfile in your working directory and tag the resulting image as `example-image:latest`.

`docker build -f docker/app-dockerfile` – Build the Dockerfile at the `docker/app-dockerfile` path.

`docker build --build-arg foo=bar .` – Build an image and set the `foo` build argument to the value bar.

`docker build --pull .` – Instructs Docker to pull updated versions of the images referenced in FROM instructions in your Dockerfile, before building your new image.

`docker build --quiet .` – Build an image without emitting any output during the build. The image ID will still be emitted to the terminal when the build completes.

Run Containers

After building an image, use these commands to run containers:

`docker run example-image:latest` – Run a new container using the `example-image:latest` image. The output from the container's foreground process will be shown in your terminal.

`>docker run example-image:latest demo-command` – Supplying an argument after the image name sets the command to run inside the container; it will be appended to the image's entrypoint. (It's possible to override the entrypoint with the `docker run` command's `--entrypoint` flag.)

`docker run --rm example-image:latest` – The `--rm` flag instructs Docker to automatically remove the container when it exits instead of allowing it to remain as a stopped container.

`docker run -d example-image:latest` – Detaches your terminal from the running container, leaving the container in the background.

`docker run -it example-image:latest` – Attaches your terminal's input stream and a TTY to the container. Use this command to run interactive commands inside the container.

`docker run --name my-container example-image:latest` – Names the new container `my-container`.

`docker run --hostname my-container example-image:latest` – Set the container's hostname to a specific value (it defaults to the container's name).

`docker run --env foo=bar example-image:latest` – Set the value of the `foo` environment variable inside the container to `bar`.

`docker run --env-file config.env example-image:latest` – Populate environment variables inside the container from the file `config.env`. The file should contain key-value pairs in the format `foo=bar`.

`docker run -p 8080:80 example-image:latest` – Bind port 8080 on your Docker host to port 80 inside the container. It allows you to visit `localhost:8080` to access the network service listening on port 80 inside the container.

`docker run -v /host-directory:/container-directory example-image:latest` – Bind mount `/host-directory` on your host to `/container-directory` inside the container. The directory's contents will be visible on both sides of the mount.

`docker run -v data:/data example-image:latest` – Mount the named Docker volume called `data` to `/data` inside the container.

`docker run --network my-network example-image:latest` – Connect the new container to the Docker network called `my-network`.

`docker run --restart unless-stopped example-image:latest` – Set the container to start automatically when the Docker daemon starts, unless the container has been manually stopped. Other restart policies are also supported.

`docker run --privileged example-image:latest` – Run the container with privileged access to the host system. This should usually be disabled to maintain security.

Manage Containers

After you've started some containers, you can use the following set of commands to manage them:

`docker ps` – List all the containers currently running on your host. (Learn more: How to use `docker ps` command)

`docker ps -a` – List *every* container on your host, including stopped ones.

`docker attach <container>` – Attach your terminal to the foreground process of the container with the ID or name `<container>`.

`docker commit <container> new-image:latest` – Save the current state of `<container>` to a new image called `new-image:latest`.

`docker inspect <container>` – Obtain all the information Docker holds about a container, in JSON format.

`docker kill <container>` – Send a SIGKILL signal to the foreground process running in a container, to force it to stop.

`docker rename <container> my-container` – Rename a specified container to `my-container`.

`docker pause <container>` and `docker unpause <container>` – Pause and unpause the processes running within a specific container.

`docker stop <container>` – Stop a running container.

`docker start <container>` – Start a previously stopped container.

`docker rm <container>` – Delete a container by its ID or name. Use the `-f` (force) flag to delete a container that's currently running.

Read more: How to Stop and Remove Docker Containers.

Copy to and From Containers

The `docker cp` command facilitates bi-directional copying between containers and your host machine:

`docker cp example.txt my-container:/data` – Copy example.txt from your host to `/data` inside the `my-container` container.

`docker cp my-container:/data/example.txt /demo/example.txt` – Copy `/data/example.txt` out of the `my-container` container, to `/demo/example.txt` on your host.

If you need to move files or folders between two containers, you should copy from the first container to your host, then onwards into the second container.

Execute Commands in Containers

The `docker exec` command allows you to run a new process inside a currently running container:

`docker exec my-container demo-command` – Run `demo-command` inside `my-container`; the process' output will be shown in your terminal

`docker exec -it my-container demo-command` – Run a command interactively by attaching your terminal's input stream and a pseudo-TTY.

Access Container Logs

`docker logs <container>` – This command streams the existing log output from a container into your terminal window, then exits.

`docker logs <container> --follow` – This variation emits all existing logs, then continues to stream new logs into your terminal as they're stored.

`docker logs <container> -n 10` – Get the last 10 logs from a container.

Logs are collated from the standard output and error streams emitted by the container's foreground process.

View Container Resource Utilization

`docker stats <container>` – Stream a container's resource utilization information into your terminal. The output includes CPU, memory, and I/O usage, as well as the number of processes running within the container.

Manage Images

The following commands interact with images stored on your Docker host:

`docker images` – List all stored images.

`docker rmi <image>` – Delete an image by its ID or tag. Deletion of images which have multiple tags must be forced using the -f flag.

`docker tag <image> example-image:latest` – Add a new tag (`example-image:latest`) to an existing image (`<image>`).

Pull and Push Images

`docker push example.com/user/image:latest` – Push an image from your Docker host to a remote registry. The image is identified by its tag, which must reference the registry you're pushing to.

`docker pull example.com/user/image:latest` – Manually pull an image from a remote registry to make it available on your host.

When the image's tag omits a registry URL, the Docker Hub registry will be used as the default.

Manage Networks

These commands administer the Docker networks on your host:

`docker create network my-network` – Create a new network called `my-network`; it will default to using the `bridge` driver.

`docker create network my-network -d host` – Use the `-d` flag to select an alternative driver, such as `host`.

`docker network connect <network> <container>` – Connect a container to an existing network.

`docker network disconnect <network> <container>` – Remove a container from a network it's currently connected to.

`docker network ls` – List all the Docker networks available on your host, including built-in networks such as `bridge` and `host`.

`docker network rm <network>` – Delete a network by its ID or name. This is only possible when there are no containers currently connected to the network.

Manage Volumes

The following commands relate to the management of storage volumes:

`docker volume create my-volume` – Create a new named volume called `my-volume`.

`docker volume ls` – List the volumes present on your host.

`docker volume rm` – Delete a volume, which will destroy the data within it. The volume must not be used by any container.

Use Configuration Contexts

Configuration contexts allow you to connect to multiple Docker daemon instances from a single installation of the Docker CLI.

`docker context create my-context --host=tcp://host:2376,ca=~/ca-file,cert=~/cert-file,key=~/key-file` – Create a new context called `my-context` to connect to a specified Docker host.

`docker context update <context>` – Modify the configuration of a named context; the command accepts the same arguments as `docker context create`.

`docker context ls` – List the contexts available in your Docker config file.

`docker context use <context>` – Switch to a named context. Subsequent `docker` commands will be executed against the Docker host configured in the newly selected context.

`docker context rm <context>` – Delete a context by its name.

Create SBOMs

Docker now has integrated SBOM generation capabilities. SBOMs are indexes of the packages included in your container images.

`docker sbom example-image:latest` – Produce an SBOM for the image tagged `example-image:latest`. The SBOM will be shown in your terminal.

`docker sbom example-image:latest --output sbom.txt` – Produce an SBOM and save it to `sbom.txt`.

`docker sbom example-image:latest --format spdx-json` – Produce an SBOM in a standard machine-parseable format, such as SPDX (`spdx-json`), CycloneDX (`cyclonedx-json`), or Syft JSON (`syft-json`).

Scan for Vulnerabilities

Docker also has a built-in image vulnerability scanner that's powered by Snyk:

`docker scan example-image:latest` – Scan for vulnerabilities in the image tagged `example-image:latest`. The results will be shown in your terminal.

`docker scan example-image:latest --file Dockerfile` – The `--file` argument supplies the path to the Dockerfile that was used to build the image. When the Dockerfile is available, more detailed vulnerability information is produced.

`docker scan example-image:latest --severity high` – Only report vulnerabilities that are `high` severity or higher. The `--severity` flag also supports `low` and `medium` values.

Docker Hub Account

These commands interact with your Docker Hub account:

`docker login` – Login to your account. You'll be prompted to supply credentials interactively. You must login before you can push images. Logging in also helps you avoid hitting public pull rate limits.

`docker logout` – Logs you out of your account.

`docker search nginx` – Searches Docker Hub for images matching the supplied search term (`nginx` , in this example).

Clean Up Unused Resources

It's normal for a regularly used Docker installation to accumulate a large number of resources, many of which become redundant as you create replacements. These commands will clean up your environment:

`docker system prune` – Removes unused data, including dangling image layers (images with no tags).

`docker system prune -a` – Extends the prune process by deleting all unused images, instead of only dangling ones.

`docker system prune --volumes` – Includes volume data in the prune process. This will delete any volumes that aren't used by a container.

`docker image prune` – Removes dangling images, without affecting any other types of data.

`docker image prune -a` – Removes all unused images.

`docker network prune` – Removes unused networks.

`docker volume prune` – Removes unused volumes.

`docker system df` – Reports your Docker installation's total disk usage.

The `prune` commands will prompt you to confirm your intentions before any resources are deleted. You can disable the prompt by setting the `-f` (force) flag.

Key Points

The commands shown in this article are a quick reference for the most commonly used Docker CLI capabilities. If you need more information on any of the commands included in this article, you can check out the official documentation on the Docker website.

Looking for more Docker guides and tutorials? Check out our other articles on the Spacelift blog.

We encourage you also to explore how integrating with third party tools is easily done at Spacelift. You have the possibility of installing and configuring them before and after runner phases or you can simply bring your own Docker image and leverage them directly.

Written by

James Walker

James Walker is the founder of Heron Web, a UK-based software development studio providing bespoke solutions for SMEs. He has experience managing complete end-to-end web development workflows with DevOps, CI/CD, Docker, and Kubernetes. James is also a technical writer and has written extensively about the software development lifecycle, current industry trends, and DevOps concepts and technologies.

jhwalker.net