

Docker Commands Cheat Sheet

Registry & Repository

Repository refers to the hosted collection of the images creating the file system for containers. Registry refers to the host containing the repositories and providing an HTTP API that helps manage the repositories.

Docker has its central registry with thousands of repositories. But before you use the images from this registry, make sure you verify them to avoid security issues.

- **docker login:** To log in to a registry.
- **docker logout:** To log out from a registry.
- **docker search:** It will search the registry for the image.
- **docker pull:** It will pull an image from the registry to the local machine.
- **docker push:** It will push an image to the registry from the local machine.

Images

You can refer to images as the templates for the Docker containers. You can run the following commands to work with the images:

- **docker images:** It will display all images.
- **docker import:** You can create an image from a tarball.
- **docker build:** You can create an image from Dockerfile.
- **docker commit:** You can create an image from a container and temporarily pause it if it is running.
- **docker rmi:** It will remove an image.
- **docker load:** It will load an image from a tar archive as STDIN, including images and tags.
- **docker:** It will save an image to a tar archive stream to STDOUT with all parent layers, tags, & versions.
- **docker history:** To display the history of an image
- **docker tag:** It will tag the image to a name.
- **docker load < my_image.tar.gz:** It will load an image from the mentioned file along with its history.

- **docker save my_image:my_tag | gzip > my_image.tar.gz:** It will save an existing file.
- **cat my_container.tar.gz | docker import - my_image:my_tag:** It will import the container as an image from the mentioned file without its history; thus, the file size is small.
- **docker export my_container | gzip > my_container.tar.gz:** It will export the container.
- **docker push repo[:tag]:** It will push an image or repo from the registry.
- **docker pull repo[:tag]:** It will pull an image or repo from the registry.
- **docker search text:** It will allow you to search for an image in the official registry.

Container

Containers are the isolated Docker process that contains the code to be executed.

- **docker create:** It will create a container without starting it.
- **docker rename:** To rename the container.
- **docker run:** It will create and start the container in one task.
- **docker rm:** It will delete a container.
- **docker update:** It will update a container's resource limits.

Usually, a container will start and stop immediately if you run it without any option.

- **docker run -td container_id:** It will keep the container running, -t will allocate a pseudo-TTY session, and -d will detach the container automatically.
- **docker run --rm:** It will remove the container once it stops.
- **docker run -v \$HOSTDIR:\$DOCKERDIR:** It will map a directory on the host to a docker container.
- **docker rm -v:** It will remove the volumes associated with the container.
- **docker run --log-driver=syslog:** Docker 1.10 comes with the logging driver for each container, and it will run docker with a custom log driver.
- **docker start:** It will start a container, so it is running.
- **docker stop:** It will stop a running container.
- **docker restart:** It stops and starts a container.
- **docker pause:** It will pause a running container, "freezing" it in place.

- **docker unpause:** It will unpause a running container.
- **docker wait:** It will block until the running container stops.
- **docker kill:** It sends a SIGKILL to a running container.
- **docker attach:** It will connect to a running container.
- **docker run -it -c 512 agileek/cpuset-test:** It lets you limit the CPU, either using a percentage of all CPUs or by using specific cores. 1024 means 100% of the CPU, so if you want the container to take 50% of all CPU cores, you should specify 512.
- **docker run -it --cpuset-cpus=0,4,6 agileek/cpuset-test:** CPU cores using cpuset-cpus.
- **docker run -it -m 300M ubuntu:14.04 /bin/bash:** Setting memory constraints on Docker.
- **docker run --rm -it --cap-add SYS_ADMIN --device /dev/fuse sshfs:** Setting Linux capabilities using cap-add. It lets you mount a FUSE-based filesystem, and you need to combine both --cap-add and --device
- **docker run -it --device=/dev/ttyUSB0 debian bash:** Providing access to single device.
- **docker run -it --privileged -v /dev/bus/usb:/dev/bus/usb debian bash:** Providing access to all devices.
- **docker ps:** It will display the running containers.
- **docker logs:** Provide the logs from the container. (You can use a custom log driver, but logs are only available for json-file and journald in 1.10).
- **docker inspect:** It checks all the information on a container (including IP address).
- **docker events:** It will get the events from the container.
- **docker port:** It will display the public-facing port of the container.
- **docker top:** It will display the running processes in the container.
- **docker stats:** It will display the containers' resource usage statistics.
- **docker diff:** It will display the changed files in the container's FS.
- **docker ps -a:** It will display the running and stopped containers.
- **docker stats --all:** It will display a list of all containers, default shows just running.
- **docker cp:** It will copy the files or folders between a container and the local filesystem.

- **docker export:** It will turn the container filesystem into a tarball archive stream to STDOUT.
- **docker exec:** To execute a command in a container.
- **docker exec -it foo /bin/bash:** To enter a running container, attach a new shell process to a running container called foo.
- **docker commit container image:** It will commit a new docker image.

Dockerfile Cheat Sheet

It is a config file that will set up a Docker container whenever you run a docker build on it. To create docker files, you can use any of the following text editors and their syntax highlighting modules.

- Sublime Text 2
- Atom
- Vim
- Emacs
- TextMate
- VS Code

The following are some instructions that you can use while working with Dockerfile:

- **FROM:** It will set the Base Image for subsequent instructions.
- **MAINTAINER** (deprecated - use LABEL instead): It will set the Author field of the generated images.
- **RUN:** It will execute any commands in a new layer on top of the current image and then commit the results.
- **CMD:** It will offer the defaults for an executing container.
- **EXPOSE:** It will tell the Docker that the container listens on the specified network ports at runtime.
- **ADD:** It will copy the new files, directories, or remote files to the container.
- **COPY:** It will copy the new files or directories to a container. It copies as root regardless of the USER/WORKDIR settings by default. Use `--chown=<user>:<group>` to provide the ownership to another user/group.
- **ENTRYPOINT:** It will configure a container that will run as an executable.

- **VOLUME:** It will create a mount point for externally mounted volumes or other containers.
- **USER:** It will set the user name for the following RUN / CMD / ENTRYPOINT commands.
- **WORKDIR:** It will set the working directory.
- **ARG:** It lets you define a build-time variable.
- **ONBUILD:** It will add a trigger instruction when the image is used as the base for another build.

Networks

Docker has a featured network, allowing the containers to connect. You can create three network interfaces with Docker, namely bridge, host, and none.

By default, the new container is launched into the bridge network. To establish communication among several containers, you need a new network for launching containers in it. It lets the containers communicate while being isolated from other containers not connected to the network.

- **docker network create NAME:** It will create a new network of bridge type by default.
- **docker network rm NAME:** It will remove one or more networks specified by name and make sure that no containers are connected to the deleted network.
- **docker network ls:** It will list all the networks.
- **docker network inspect NAME:** It will show the detailed information on one or more networks.
- **docker network connect NETWORK CONTAINER:** It will connect a container to a network
- **docker network disconnect NETWORK CONTAINER:** It will disconnect a container from a network.

Volumes

Docker has volumes that are free-floating filesystems. So there is no need to be connected to a particular container. You can use volumes mounted from data-only containers for portability. As per Docker 1.9.0, it comes with the named volumes that replace data-only containers.

- **docker volume create:** to create volumes.
- **docker volume rm:** To remove volumes.

- **docker volume ls:** To list the volumes.
- **docker volume inspect:** To inspect the volumes.

Orchestrate

Orchestration manages the container's life cycle, especially in dynamic environments. You can use it for controlling and automating several tasks for containers.

Among a long list of Docker orchestration tools, the most commonly used orchestration tools are Docker Swarm, Kubernetes, and Mesos. In this Docker cheat sheet, we are using Docker Swarm commands.

- **Docker swarm init --advertise-addr 10.1.0.2:** Initialize the swarm mode and listen to a specific interface.
- **Docker swarm join --token<manager-token> 10.1.0.2:2377:** It will join an existing swarm as a manager node.
- **Docker swarm join --token<worker-token> 10.1.0.2:2377:** It will join a swarm as a worker node.
- **Docker node ls:** It will list all nodes in the swarm.
- **Docker service create --replicas 3 -p 80:80 name -webngix:** It will create a service from an image on the existing port and deploy three instances.
- **Docker service ls:** It will list services running in a swarm.
- **Docker service scale web=5:** It will scale the service.
- **docker service ps web:** It will list the tasks of service.

Interaction with container

You can use the following commands to interact with the container.

- **Docker exe -ti container_name command.sh:** It will run a command in the container.
- **Docker logs -ft container name:** It will follow the container log.
- **Docker commit -m "commit message" -a "author" container_name username/image_name: tag:** It will save the running container as an image.

Build

You can use the following commands to build the images from a Docker file.

- **Docker build -t myapp :1.0-** will build an image from the Docker file and tag it.

- **Docker images**- it will list all the images that are locally stored
- **Docker rmi alpine: 3.4** will delete an image from the Docker Store.

Cleanup

To optimize the usage of the resources, you need to clean up the resources frequently to maintain the performance. You can run the following commands to clean up resources.

- **Docker image prune**: It will clean an unused/dangling image
- **Docker image prune -a**: It will remove an image not used in a container.
- **Docker system prune**: It will prune the entire system.
- **Docker swarm leave**: It will leave a swarm.
- **docker stack rm stack_name**: It will remove a swarm.
- **Docker kills \$ (docker ps -q)**: It will v.
- **docker rm \$(docker ps -a -q)**: It will delete all stopped containers
- **docker rmi \$(docker images -q)**: It will delete all images.

Services

Let's now take a sneak peek at the commands used to view the running services, run the services, view all service logs, and scale the services.

- **Docker service ls**: It will list all services running in a swarm.
- **Docker stack services stack_name**: It will display all running services.
- **Docker service logs stack_name service_names**: It will display all service logs.
- **Docker service scale stack_name_service_name= replicas**: It will scale a service across qualified nodes.

Docker-compose Cheat Sheet

Compose is a tool that helps you to define and run multi-container Docker applications. With Compose, you get to work with a YAML file to configure your application services. With the help of the following commands, you can simply create and start all the services from your configuration.

- **docker-compose start**: It will start the container.
- **docker-compose stop**: It will stop the container.

- **docker-compose pause:** It will pause the container.
- **docker-compose unpause:** It will unpause the container.
- **docker-compose ps:** It will list all the containers.
- **docker-compose up:** It aggregates the output of each container (essentially running `docker-compose logs --follow`).
- **Docker-compose down:** It stops containers and removes containers, networks, volumes, and images created by up.
- **Docker-compose -f <docker-compose-file> up:** It will start up your application
- **docker-compose stop-run docker-compose** in detached mode using `-d` flag, and then you can stop it whenever needed.

Basic example

```
# docker-compose.yml

version: '2'

services:

  web:

    build: .

    # build from Dockerfile

    context: ./Path

    dockerfile: Dockerfile

    ports:

      - "5000:5000"

    volumes:

      - ./code

  redis:

    image: redis
```

Reference

- Building

web:

build using the Dockerfile

build: .

build using the custom Dockerfile

build:

context: ./dir

dockerfile: Dockerfile.dev

build from image

image: ubuntu

image: ubuntu:14.04

image: tutum/influxdb

image: example-registry:4000/postgresql

image: a4bc65fd

- Ports

ports:

- "3000"

- "8000:80" # guest:host

expose ports to linked services (not to host)

expose: ["3000"]

- Commands

commands to execute

command: bundle exec thin -p 3000

command: [bundle, exec, thin, -p, 3000]

overriding the entrypoint

entrypoint: /app/start.sh

entrypoint: [php, -d, vendor/bin/phpunit]

- Environment variables

environment variables

environment:

RACK_ENV: development

environment:

- RACK_ENV=development

environment vars from the specified file

env_file: .env

env_file: [.env, .development.env]

- Dependencies

makes the `db` service available as the hostname `database`

(implies depends_on)

links:

- db:database

- redis

make sure `db` is alive before starting

depends_on:

- db

- Other options

make this service extend another

extends:

file: common.yml # optional

service: web app

volumes:

- /var/lib/mysql

- ./_data:/var/lib/mysql