

General Commands

<code>docker version</code>	
<code>docker version --format '{{.Server.Version}}'</code>	Get the server version
<code>docker version --format '{{json .}}'</code>	Dump raw JSON data
<code>docker info -D</code>	All docker commands to output debug info

docker container COMMAND

<code>ls [OPTIONS]</code>	list running containers
<code>ls --all -a</code>	list all containers
<code>ls --size -s</code>	list running containers sizes
<code>ls -q</code>	display container IDs of running container (q stands for quiet)
<code>ls --format '{{printf "%-40s" .Names}} {{.Ports}}'</code>	display name and ports of running containers
<code>start stop pause unpause restart rm CONTAINER</code>	start, stop, pause, unpause, restart or remove container
<code>logs --tail -n 10 CONTAINER</code>	show the last 10 lines of logs
<code>logs [OPTIONS] CONTAINER</code>	fetch the logs of a container
<code>logs --timestamps -t CONTAINER</code>	show logs including timestamps
<code>run [OPTIONS] IMAGE [COMMAND] [ARG...]</code>	create a new container and run a command into it
<code>run --name CONTAINER nginx:1.22.0</code>	create CONTAINER using nginx image tagged 1.22.0
<code>run -p 8080:80 IMAGE</code>	maps the host port 8080 to the created container port 80
<code>run --detach -d IMAGE</code>	run container in background and print container ID
<code>run --rm -it IMAGE CMD</code>	create a container, run a cmd on it interactively, then delete the container
<code>run --net NETWORK IMAGE</code>	--net connects a container to NETWORK
<code>run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True -v mysql--db:/var/lib/mysql mysql</code>	create a named volume mysql-db pointing to the container directory /var/lib/mysql
<code>run -d --name nginx -p 80:80 -v \$(pwd):/usr/share/nginx/html nginx</code>	create a bind mount between the host current directory and /usr/share/nginx/html
<code>run --name postgres-db -e POSTGRES_PASSWORD=password --mount type=volume,source=\$HOME/docker/volumes/postgres,target=/var/lib/postgresql/data -p 2000:5432 -d postgres</code>	create a named volume between the host directory \$HOME/docker/volumes/postgres and the container directory /var/lib/postgresql/data
<code>run --name postgres-db -e POSTGRES_PASSWORD=password --v \$HOME/docker/volumes/postgres:/var/lib/postgresql/data -p 2000:5432 -d postgres</code>	create a named volume between the host directory \$HOME/docker/volumes/postgres and the container directory /var/lib/postgresql/data
<code>run -d --name postgres-db -e POSTGRES_PASSWORD=password --mount type=bind,source="\$pwd",target=/var/lib/postgresql/data -p 2000:5432 -d postgres</code>	create a bind mount between the host current directory and the /var/lib/postgresql/ directory in the container.



docker container COMMAND (cont)

<code>run --health-cmd="curl -f localhost:9200/_cluster/health false" --health-interval=5s --health-retries=3 --health-timeout=2s --health-start-period=15s IMAGE</code>	health check
<code>ps -a --filter volume=VOLUME</code>	display all containers using VOLUME
<code>ps --no-trunc</code>	prevent truncating output
	more docker container run options here
<code>top CONTAINER [ps OPTIONS]</code>	display the running processes of a container
	ps OPTIONS here
<code>rm --force -f CONTAINER</code>	force the removal of a running container (uses SIGKILL)
<code>stats [OPTIONS] CONTAINER</code>	display a live stream of running container(s) resource usage statistics
<code>stats --all -a CONTAINER</code>	display a live stream of ALL running container(s) resource usage statistics
<code>inspect --size -s --pretty CONTAINER</code>	display detailed information on one or more containers with size
<code>update -c 4 -m 8G CONTAINER</code>	update cpu and ram of a container. see https://docs.docker.com/engine/reference/commandline/container_update/#options
<code>commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]</code>	Create a new image from a container's changes excepts on volumes.
<code>commit --change "ENV DEBUG=true" CONTAINER [REPOSITORY[:TAG]]</code>	Apply Dockerfile instruction to the created image
<code>exec -it CONTAINER sh -c "test -d /some/dir && echo 'It Exists'"</code>	test if a folder exists in a container
<code>docker container inspect CONTAINER --format '{{json .NetworkSettings}}' jq</code>	pretty print container's network info

docker image COMMAND

<code>ls</code>	List images. same as <code>docker images</code>
<code>inspect -f --format='{{.Config.Cmd}}' IMAGE</code>	Check available command (ex: sh or bash) available on the image
<code>pull nginx</code>	Pull the "latest" nginx image from dockerhub (default repo)
<code>pull nginx:1.11.9</code>	Pull image nginx 1.11.9 from dockerhub
<code>history [OPTIONS] IMAGE</code>	Show the history of the IMAGE (layers)
<code>tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]</code>	Create a tag
<code>push [OPTIONS] NAME[:TAG]</code>	Push the USER/IMAGE:TAG image to DockerHub (default registry)
<code>build [OPTIONS] PATH URL -</code>	Build an image from a Dockerfile. see options here



docker image COMMAND (cont)

build --tag -t IMAGE .	Build an image searching for a dockerfile in the host current directory and tag it <code>my_cus t_image</code>
build --target builder -t IMAGE:latest .	Build from Dockerfile but stops at build stage "builder" in a multi-stage build dockerfile.

docker volume

A **volume gives** full control of the storage** from the container. A new directory is created within Docker's storage directory on the host machine, and Docker manages that directory's content

Bind mounts gives full control of the storage from the host and containers. It's not a secure option. **A file or directory on the host machine is mounted into a container.** The file or directory is referenced by its full or relative path on the host machine

more info on mount types here

ls	list volumes
inspect VOLUME	display detailed information on a volume
create --name VOLUME	create a volume
rm -f VOLUME	Remove a volume. -f to force
prune -f	Remove all unused local volumes. -f to force
docker container run --rm --volumes-from CONTAINER -v \$(pwd):/backup ubuntu tar cvf /backup/backup.tar /NAMED_VOL	backup data from a container

docker network

ls	
create <network>	
inspect -v <network> --pretty	
connect <network> <container>	connect a running container to a network
rm <network>	remove a network
prune	remove all unused networks

a **bridge network** is an isolated network on a single engine install (=single host)

create -d|--driver bridge NET

an **overlay network** is an isolated network on a **swarm** (=across host)

create -d|--driver overlay NET create an overlay network used to enable communication between containers more info here

All swarm service management traffic is encrypted by default, using the AES algorithm in GCM mode. Manager nodes in the swarm rotate the key used to encrypt gossip data every 12 hours.

create --opt encrypted NET encrypt application data. enables IPSEC encryption at the level of the vxlan.

create --driver overlay --attachable NET create an overlay network which can be used by swarm services or **standalone containers**

create =d|--driver bridge NET containers across the host



docker images

<code>docker images</code>	List the most recently created images
<code>docker images -f 'dangling=true' -q</code>	display untagged images that are the leaves of the images tree (not intermediary layers). These images occur when a new build of an image takes the <code>repo:tag</code> away from the image ID, leaving it as <code><none>:<none></code> or untagged. A warning will be issued if trying to remove an image when a container is presently using it. By having this flag it allows for batch cleanup.
<code>docker rmi \$(docker images -f "dangling=true" -q)</code>	Clean up untagged images that are the leaves of the images tree (not intermediary layers). These images occur when a new build of an image takes the <code>repo:tag</code> away from the image ID, leaving it as <code><none>:<none></code> or untagged. A warning will be issued if trying to remove an image when a container is presently using it.

docker compose

Compose can only create services locally

A **Service** is a set of replicated containers

`docker -co mpo se .yaml` is the default file used but we can use the `-f` option to use another filename

`up [OPTIONS] [SERVICE...]` Builds, (re)creates, and then starts a set of defined services.

`docker compose up` builds the image from the dockerfile in the build section of the compose file only if not found in cache

`-f docker-compose.yaml -f docker-compose.test.yaml up` Build test container(s) for services using `docker -co mpo se .yaml` as base config overridden by `docker -co mpo se .test.yaml`

`-f docker-compose.yml -f docker-compose.prod.yml config > output.yml up`

`up --build` rebuild the images even if found in cache

`up -d|--detach`

`up -p PROJECT_NAME` set a project name that will be the containers names prefix instead of the parent directory of the `compose file`

`down [OPTIONS]`

`down --rmi local|all` removes images of the service as well as containers. Remove images used by services. "local" remove only images that don't have a custom tag



docker compose (cont)

start stop pause unpause restart kill [SERVICE...]	Start, stop, pause, unpause, restart, kill q service(s) and its container(s)
logs [OPTIONS] [SERVICE...]	Displays log output from containers of a service
ps [OPTIONS] [SERVICE...]	List containers of a service
top [SERVICE...]	Display the running processes of a service
build [OPTIONS] [SERVICE...]	Build or rebuild images but don't start the container
-f FILE build [OPTIONS] [SERVICE...]	Build or rebuild images specifying a Dockerfile
VAR=VALUE docker-compose build	pass variables at build time to compose
VAR=VALUE docker-compose up	pass variables at run time to compose
build --build-arg GIT_PERSONAL_ACCESS_TOKEN="{TOKEN}"	Pass variables into Dockerfile through Docker Compose during build

docker service (Swarm)

A service is a set of replicated containers

A single **service** can have **multiple tasks** and each one will launch a **container**

```
create --name mydb --replicas 3 redis:7-bullseye
```

```
create --name mydb --replicas 2 --env MYVAR=foo --env MYVAR2=bar redis:3.0.6
```

Create a mydb service specifying 2 env variables

```
create --name redis --secret source=ssh-key,target=ssh --secret source=app-key,-target=app,uid=1000,gid=1001,mode=0400 redis:3.0.6
```

```
create --name frontend --network <net> -p 8081:80 nginx
```

Creates a network attached to <net>. exposes port 80 reachable through the host port 8081

```
create --name db --network backend --mount type=volume,source=db-data,target=/var/lib/postgresql/data -e POSTGRES_HOST_AUTH_METHOD=trust --replicas 1 postgres:9.4
```

Create a postgres "db" service with a named volume, attached to the backend network

```
create --name SERVICE --network NET IMAGE
```

Creates a SERVICE and attach it to the existing NET network. The swarm extends NET to each node running the service.

```
create --name pgserv -e POSTGRES_HOST_AUTH_METHOD=trust --health-cmd="pg_isready -U postgres || exit 1" --health-start-period 120s postgres:latest
```

Creates a pgserv service with health checks executed every 30sec (default) but the failure counts begins after 120s

```
create --name SERVICE -p HOST_PORT:CONTAINER_PORT --replicas 5 --detach=false REGISTRY_HOST:REGISTRY_HOST/IMAGE
```

Create a service from a custom registry

```
ls
```

List services running in the swarm

```
ps SERVICE
```

List the tasks of one or more services



docker service (Swarm) (cont)

inspect SERVICE	Display detailed information on one or more services
logs SERVICE	Batch-retrieves logs present at the time of execution
rm SERVICE	Removes the specified services from the swarm.
update [OPTIONS] SERVICE	Update a service
update --replicas 10 --reserve-cpu 4 --reserve-memory 16G SERVICE	Change cpu and ram and max tasks.
update --mount-add type=volume,source=other-volume,target=/somewhere-else SERVICE	Add a named volume <code>other-volume</code> pointing to <code>/somewhere-else</code> . note the syntax <code>--option-add</code> & <code>*--option-rm</code>
update --mount-rm PATH myservice	Remove the PATH volume. a path always begins with a /
update --secret-add source=ssh-2,target=ssh-2 --secret-rm ssh-1 myservice	Add/remove secret. note the syntax <code>--option-add</code> & <code>--option-rm</code>
update --rollback SERVICE	Rollback a service to its previous state
update --image IMAGE SERVICE	Change the image of a service
update --publish-rm HOST_PORT --publish-add HOST_PORT:CONTAINER_PORT> SERVICE	Replace a port in containers of a service
scale SERVICE=REPLICAS	Scale one or multiple replicated services. a replica is a task
update --force SERVICE	Force update of a service to rebalance the load across the swarm.

docker node (swarm)

ls	
ps NODE	
promote demote NODE [NODE...]	promote or demote a node to manager or worker
rm -f NODE [NODE...]	remove a node from a swarm. -f to force
inspect --pretty self NODE [NODE...]	Display detailed and pretty-printed info on one or more nodes
inspect --format '{{ .Status.Addr }}' self NODE [NODE...]	get the node IP address
ls -f "role=manager" -f node.label=region=region-a	list manager nodes having a region label set to region-a
update [OPTIONS] NODE	update a node
update --label-add LABEL_KEY=LABEL_VALUE NODE [LABEL_KEY=LABEL_VALUE NODE...]	update <node> adding a label key/value



docker stack (swarm)

stacks are compose for production swarms and accepts compose files but can't build images

deploy --compose-file -c docker-compose.yml --compose-file -c docker-compose.prod.yml STACK	Deploy or update the prod stack using the <code>docker -co mpo se.yml</code> base config and the <code>docker -co mpo se.p ro d.yml</code> for specific prod conf
cat <docker-compose.yml> docker stack deploy --compose-file - <stack>	Create or update a stack using the std input (- opt)
deploy -c <docker-compose-1.yml> -c <docker-compose-2.yml><stack>	Deploy a stack using multiple compose files. It must be exec on a Manager node
ls [OPTIONS]	Lists the stacks.
ls --format "table {{.Name}}: {{.Services}}"	Output stacks with the Name and Services
ps [OPTIONS] STACK	
ps -f "name=redis.1" -f "name=redis.7" STACK	List the tasks that are part of the STACK named <code>redis.1</code> and <code>redis.7</code>
ps -f "node=NODE_01" -f "node=NODE_02" STACK	List the tasks from <code>NODE_01</code> and <code>NODE_02</code>
ps -f "desired-state=running" -f "desired-state=ready"	List the tasks which desired-state is running or ready
ps --format "table {{.Name}}: {{.Image}}, {{.CurrentState}}" STACK	Output tasks with the Name, Image and State
rm [OPTIONS] STACK [STACK...]	Removes one or more <stack>
services [OPTIONS] STACK	Lists the services that are running as part of the specified stack
services --filter name=web --filter name=db myapp	List both the web and db services

Secrets (swarm)

create [OPTIONS] SECRET [file -]	
create psql_user psql_user.txt	Create the <code>psql_user</code> secret containing the value in <code>psql_user.txt</code>
echo SECRET_PASS docker secret create psql_pass -	Create the secret <code>psql_pass</code> from stdin
ls [OPTIONS]	List secrets
ls --filter label=project	List all secrets with a "project" label
inspect SECRET	List info about SECRET
docker service create --name SERVICE --secret SECRET_USER --secret SECRET_PASS -e POSTGRES_PASSWORD_FILE=/run/secrets/SECRET_PASS -e POSTGRES_USER_FILE=/run/secrets/SECRET_USER postgres	Use secrets inside the containers of a service

A secret can be a username, a password, a key or whatever shouldn't be seen from the outside.



Docker Registry

<code>docker container run -d -p 5000:5000 --restart=always -v \$(pwd)/registry-data:/var/lib/registry --name registry registry:2</code>	start the registry container. A production-ready registry must be protected by TLS and should ideally use an access-control mechanism.
<code>docker run -d -p 5000:5000 --name <registry> --restart unless-stopped -v \$(pwd)/registry-data:/var/lib/registry -v \$(pwd)/certs:/certs -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key registry</code>	start a secured registry using host <code>./registry-data</code> data folder and <code>./certs/</code> for the signed cert and the private key
<code>docker service create --name <registry> -p 5000:5000 registry</code>	creates a registry in a swarm
<code>docker login <REGISTRY_HOST>:<REGISTRY_PORT></code>	1/2 Login to a private registry
<code>docker service create --name SERVICE -p HOST_PORT:CONTAINER_PORT --replicas 5 --detach=false REGISTRY_HOST:REGISTRY_HOST/IMAGE</code>	2/2 Create a service from an image on a private registry
<code>docker login <REGISTRY_HOST>:<REGISTRY_PORT></code>	1/2 Login to a private registry
<code>docker tag <IMAGE_ID> <REGISTRY_HOST>:<REGISTRY_PORT>/<APPNAME>:<APPVERSION></code>	2/2 Tag an image to a private registry
<code>docker push <REGISTRY_HOST>:<REGISTRY_PORT>/<IMAGE>:<TAG></code>	3/3 Push an image to a private registry



By **Boulard**
cheatography.com/boulard/

Published 22nd September, 2022.
 Last updated 29th September, 2022.
 Page 8 of 8.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>