

General Commands

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

docker container COMMAND

ls [OPTIONS]	list running containers
ls --all -a	list all containers
ls --size -s	list running containers sizes
start stop pause unpause restart rm CONTAINER	start, stop, pause, unpause, restart or remove container
logs --tail -n 10 CONTAINER	show the last 10 lines of logs
logs [OPTIONS] CONTAINER	fetch the logs of a container
logs --timestamps -t CONTAINER	show logs including timestamps
run [OPTIONS] IMAGE [COMMAND] [ARG...]	create a new container and run a command into it
run --name CONTAINER nginx:1.22.0	create CONTAINER using nginx image tagged 1.22.0
run -p 8080:80 IMAGE	maps the host port 8080 to the created container port 80
run --detach -d IMAGE	run container in background and print container ID
run --rm -it IMAGE CMD	create a container, run a cmd on it interactively, then delete the container
run --net NETWORK IMAGE	--net connects a container to NETWORK
run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True -v mysql-db:/var/lib/mysql mysql	create a named volume mysql-db pointing to the container directory /var/lib/mysql
run -d --name nginx -p 80:80 -v \$(pwd):/usr/share/nginx/html nginx	create a bind mount between the host current directory and /usr/share/nginx/html
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	create a named volume between the host directory \$HOME/docker/volumes/postgres and the container directory /var/lib/postgresql/data
run --name postgres-db -e POSTGRES_PASSWORD=password --v \$HOME/docker/volumes/postgres:/var/lib/postgresql/data -p 2000:5432 -d postgres	create a named volume between the host directory \$HOME/docker/volumes/postgres and the container directory /var/lib/postgresql/data
run -d --name postgres-db -e POSTGRES_PASSWORD=password --mount type=bind,source="\$pwd",target=/var/lib/postgresql/data -p 2000:5432 -d postgres	create a bind mount between the host current directory and the /var/lib/postgresql/ directory in the container.
run --health-cmd="curl -f localhost:9200/_cluster/health false" --health-interval=5s --health-retries=3 --health-timeout=2s --health-start-period=15s IMAGE	health check
	more docker container run options here
top CONTAINER [ps OPTIONS]	display the running processes of a container
	ps OPTIONS here



docker container COMMAND (cont)

<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

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

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

<code>ls</code>	list volumes
-----------------	--------------



By Boulard
cheatography.com/boulard/

Published 22nd September, 2022.
Last updated 27th September, 2022.
Page 2 of 7.

Sponsored by **CrosswordCheats.com**
Learn to solve cryptic crosswords!
<http://crosswordcheats.com>

docker volume (cont)

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

docker images	List the most recently created images
docker images -f 'dangling=true' -q	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 repo:tag away from the image ID, leaving it as <none>:<none> 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.



docker images (cont)

`docker rmi $(docker images -f "dangling=true" -q)` 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 `repo:tag` away from the image ID, leaving it as `<none>:<none>` 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-compose.yml` 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.yml -f docker-compose.test.yml up` Build test container(s) for services using `docker-compose.yml` as base config overridden by `docker-compose-test.yml`

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

`up -d|--detach` Create container(s) for service(s) in the background.

`up --build`

`down [OPTIONS]` Stops and removes containers, networks, volumes, and images created by up.

`down rmi`

`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

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`



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

Published 22nd September, 2022.
Last updated 27th September, 2022.
Page 4 of 7.

Sponsored by **CrosswordCheats.com**
Learn to solve cryptic crosswords!
<http://crosswordcheats.com>

docker service (Swarm) (cont)

<code>create --name mydb --replicas 2 --env MYVAR=foo --env MYVAR2=bar redis:3.0.6</code>	Create a mydb service specifying 2 env variables
<code>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</code>	
<code>create --name frontend --network <net> -p 8081:80 nginx</code>	Creates a network attached to <net>. exposes port 80 reachable through the host port 8081
<code>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</code>	Create a postgres "db" service with a named volume, attached to the backend network
<code>create --name SERVICE --network NET IMAGE</code>	Creates a SERVICE and attach it to the existing NET network. The swarm extends NET to each node running the service.
<code>create --name pgserv -e POSTGRES_HOST_AUTH_METHOD=trust --health-cmd="pg_isready -U postgres exit 1" --health-start-period 120s postgres:latest</code>	Creates a pgserv service with health checks executed every 30sec (default) but the failure counts begins after 120s
<code>create --name SERVICE -p HOST_PORT:CONTAINER_PORT --replicas 5 --detach=false REGISTRY_HOST:REGISTRY_HOST/IMAGE</code>	Create a service from a custom registry
<code>ls</code>	List services running in the swarm
<code>ps SERVICE</code>	List the tasks of one or more services
<code>inspect SERVICE</code>	Display detailed information on one or more services
<code>logs SERVICE</code>	Batch-retrieves logs present at the time of execution
<code>rm SERVICE</code>	Removes the specified services from the swarm.
<code>update [OPTIONS] SERVICE</code>	Update a service
<code>update --replicas 10 --reserve-cpu 4 --reserve-memory 16G SERVICE</code>	Change cpu and ram and max tasks.
<code>update --mount-add type=volume,source=other-volume,target=/somewhere-else SERVICE</code>	Add a named volume <code>other-volume</code> pointing to <code>/somewhere-else</code> . note the syntax <code>--option-add & *-option-rm</code>
<code>update --mount-rm PATH myservice</code>	Remove the PATH volume. a path always begins with a /
<code>update --secret-add source=ssh-2,target=ssh-2 --secret-rm ssh-1 myservice</code>	Add/remove secret. note the syntax <code>--option-add & --option-rm</code>
<code>update --rollback SERVICE</code>	Rollback a service to its previous state
<code>update --image IMAGE SERVICE</code>	Change the image of a service
<code>update --publish-rm HOST_PORT --publish-add HOST_PORT:CONTAINER_PORT> SERVICE</code>	Replace a port in containers of a service



docker service (Swarm) (cont)

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 -compose.yml</code> base config and the <code>docker -compose.prod.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}}"	Output tasks with the Name, Image and State
STACK	
rm [OPTIONS] STACK [STACK...]	Removes one or more <stack>
services [OPTIONS] STACK	Lists the services that are running as part of the specified stack



docker stack (swarm) (cont)

```
services --filter name=web --filter name=db myapp
```

List both the web and db services

Secrets (swarm)

```
create [OPTIONS] SECRET [file|-]
```

```
create psq_user psq_user.txt
```

Create the psq_user secret containing the value in psq_user.txt

```
echo SECRET_PASS | docker secret create psq_pass -
```

Create the secret psq_pass 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

```
docker container run -d -p 5000:5000 --restart=always -v $(pwd)/registry-data:/var/lib/regi
stry --name registry registry:2
```

start the registry container. A production-ready registry must be protected by TLS and should ideally use an access-control mechanism.

```
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=/ce
rts/domain.crt -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key registry
```

start a secured registry using host `./registry-data` folder and `./certs/` for the signed cert and the private key

```
docker service create --name <registry> -p 5000:5000 registry
```

creates a registry in a swarm

```
docker login <REGISTRY_HOST>:<REGISTRY_PORT>
```

1/2 Login to a private registry

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

2/2 Create a service from an image on a private registry

```
docker login <REGISTRY_HOST>:<REGISTRY_PORT>
```

1/2 Login to a private registry

```
docker tag <IMAGE_ID> <REGISTRY_HOST>:<REGISTRY_PORT>/<APPNAME>:<AP
PVERSION>
```

2/2 Tag an image to a private registry

```
docker push <REGISTRY_HOST>:<REGISTRY_PORT>/<IMAGE>:<TAG>
```

3/3 Push an image to a private registry

