

1. Containers

A lightweight virtual OS that run processes in full isolation.

1.1 Lifecycle

- `docker create` creates a container but does not start it.
- `docker rename` allows the container to be renamed.
- `docker run` creates and starts a container in one operation.
- `docker rm` deletes a container.
- `docker update` updates a container's resource limits.
- `docker run --rm` : remove the container after it stops.
- `docker run -v $HOSTDIR:$DOCKERDIR` : map the directory (\$HOSTDIR) on the host to a `docker container ($DOCKERDIR)`.
- `docker rm -v` : remove the volumes associated with the container.
- `docker run --log-driver=syslog` : run docker with a custom log driver.

1.2 Starting and Stopping

- `docker start` starts a container so it is running.
- `docker stop` stops a running container.
- `docker restart` stops and starts a container.
- `docker pause` pauses a running container, "freezing" it in place.
- `docker unpause` will unpause a running container.
- `docker wait` blocks until running container stops.
- `docker kill` sends a SIGKILL to a running container.
- `docker attach` will connect to a running container.

1.3 CPU Constraints

CPU can be limited either using a percentage over all CPUs, or by using specific cores.

- `-c` or `cpu-shares`: 1024 means 100% of the CPU, so if we want the container to take 50% of all CPU cores, we should specify 512 for instance, `docker run -ti --c 512 ...cpuset-cpus`
- `o`: use only some CPU cores, for instance, `docker run -ti --cpuset-cpus=0,4,6 ...`

1.4 Memory Constraints

Memory can be limited using `-m` flag, for instance, `docker run -it -m 300M ubuntu:14.04 /bin/bash`

1.5 Capabilities

`cap-add` and `cap-drop`: Add or drop linux capabilities.

- Mount a FUSE based filesystem:
 - `docker run --rm -it --cap-add SYS_ADMIN --device /dev/fuse sshfs`
- Give access to a single device:
 - `docker run -it --device=/dev/ttyUSB0 debian bash`

- Give access to all devices:
 - `docker run -it --privileged -v /dev/bus/usb:/dev/bus/usb debian bash`

1.6 Info

- `docker ps` shows running containers.
- `docker logs` gets logs from container. (You can use a custom log driver, but logs is only available for `json-file` and `journald` in 1.10).
- `docker inspect` looks at all the info on a container (including IP address).
- `docker events` gets events from container.
- `docker port` shows public facing port of container.
- `docker top` shows running processes in container.
- `docker stats` shows containers' resource usage statistics.
- `docker diff` shows changed files in the container's FS.
- `docker ps -a` shows running and stopped containers

1.7 Import / Export

- `docker cp` copies files or folders between a container and the local filesystem.
- `docker export` turns container filesystem into tarball archive stream to STDOUT.

1.8 Executing Commands

`docker exec` to execute a command in container.

2. Images

A template or blueprint for docker containers.

2.1 Lifecycle

- `docker images` shows all images.
- `docker import` creates an image from a tarball.
- `docker build` creates image from Dockerfile.
- `docker commit` creates image from a container, pausing it temporarily if it is running.
- `docker rmi` removes an image.
- `docker load` loads an image from a tar archive as STDIN, including images and tags (as of 0.7).
- `docker save` saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions (as of 0.7).

2.2. Info

- `docker history` shows history of image.
- `docker tag` tags an image to a name (local or registry).

2.3. Cleaning up

- `docker rmi` remove specific images.
- `docker-gc` a tool to clean up images that are no longer used by any containers in a safe manner.

2.4. Load/Save image

- `docker load < my_image.tar.gz` load an image from file
- `docker save my_image:my_tag | gzip > my_image.tar.gz` save an existing image

2.5. Import/Export container

- `cat my_container.tar.gz | docker import - my_image:my_tag` import a container as an image from file
- `docker export my_container | gzip > my_container.tar.gz` export an existing container

3. Networks

A small def goes here

3.1. Lifecycle

- `docker network create`
- `docker network rm`

3.2. Info

- `docker network ls`
- `docker network inspect`

3.3. Connection

- `docker network connect`
- `docker network disconnect`

4. Registry & Repository

A repository is a hosted collection of tagged images that together create the file system for a container.

A registry is a host -- a server that stores repositories and provides an HTTP API for managing the uploading and downloading of repositories.

Docker.com hosts its own index to a central registry which contains a large number of repositories.

- `docker login` to login to a registry.
- `docker logout` to logout from a registry.
- `docker search` searches registry for image.
- `docker pull` pulls an image from registry to local machine.
- `docker push` pushes an image to the registry from local machine.

5. Volumes

Docker volumes are free-floating filesystems. They don't have to be connected to a particular container. You should use volumes mounted from data-only containers for portability.

5.1. Lifecycle

- `docker volume create`
- `docker volume rm`

5.2. Info

- `docker volume ls`
- `docker volume inspect`

6. Exposing ports

- `docker run -p 127.0.0.1:$HOSTPORT:$CONTAINER-PORT --name CONTAINER -t docker_image` mapping the container port to the host port using `-p`
- `EXPOSE <CONTAINERPORT>` expose port `CONTAINERPORT` at runtime (see `dockerfile`)
- `docker port CONTAINER $CONTAINERPORT` check the mapped port

7. Tips

7.1. Get IP address

```
> docker inspect some_docker_id | grep IPAddress | cut -d '"' -f 4
or install jq:
> docker inspect some_docker_id | jq -r '.[0].NetworkSettings.IPAddress'
or using a go template:
> docker inspect -f '{{.NetworkSettings.IPAddress}}' <container_name>
```

7.2. Get port mapping

```
docker inspect -f '{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} -> {{{index $conf 0}.HostPort}} {{end}}' <containername>
```

7.3. Find containers by regular expression

```
for i in $(docker ps -a | grep "REGEXP_PATTERN" | cut -f1 -d" "); do echo $i; done
```

7.4. Get Environment Settings

```
docker run --rm ubuntu env
```

7.5. Kill running containers

```
docker kill $(docker ps -q)
```

7.6. Delete old containers

```
docker ps -a | grep 'weeks ago' | awk '{print $1}' | xargs docker rm
```

7.7. Delete stopped containers

```
docker rm -v $(docker ps -a -q -f status=exited)
```

7.8. Delete dangling images

```
docker rmi $(docker images -q -f dangling=true)
```

7.9. Delete all images

```
docker rmi $(docker images -q)
```

7.10. Delete dangling volumes

```
docker volume rm $(docker volume ls -q -f dangling=true)
```