Docker 기본명령2

Docker Image 관련 명령들

$ docker image build [OPTIONS] PATH | URL | -

Dockerfile 및 "context" 에서 이미지 빌드합니다.
context는 지정된 PATH 또는 URL에 있는 파일 세트입니다.
빌드 프로세스는 컨텍스트의 모든 파일을 참조할 수 있습니다. 예를 들어 빌드는 COPY 명령을 사용하여 컨텍스트에서 파일을 참조할 수 있습니다.

URL 파라미터는 세 가지 종류의 리소스(Git repositories, pre-packaged tarball contexts, 일반 텍스트 파일)를 참조할 수 있다.

기본적으로 docker build 명령은 빌드 컨텍스트의 루트에서 Dockerfile을 찾습니다.
-f, --file, 옵션을 사용하면 대신 사용할 대체 파일의 경로를 지정할 수 있습니다. -f 옵션으로 지정한 경로는 빌드 컨텍스트 내의 파일이어야 합니다. 만약 상대 경로를 지정하면 빌드 컨텍스트의 루트를 기준으로 해석됩니다.
업로드된 컨텍스트에 파일 또는 디렉터리가 없으면 docker build는 해당 파일 또는 디렉터리가 없음 오류를 반환합니다.
컨텍스트가 없거나 호스트 시스템의 다른 곳에 있는 파일을 지정하는 경우 이런 일이 발생할 수 있습니다.
컨텍스트는 보안상의 이유로 현재 디렉터리(및 해당 하위 디렉터리)로 제한되며 원격(remote) Docker 호스트에서 반복 가능한 빌드를 보장합니다.
이는 ADD ../file이 작동하지 않는 이유이기도 합니다.

```
$ docker build https://github.com/docker/rootfs.git#container:docker
$ docker build http://server/context.tar.gz
$ docker build - < Dockerfile
$ docker build .
# -f, --file 옵션으로 빌드할 파일을 지정
$ docker build -f ctx/Dockerfile http://server/ctx.tar.gz
# 아래의 명령은 현재 디렉터리를 빌드 컨텍스트로 사용하고 STDIN에서 Dockerfile을 읽습니다.
$ curl example.com/remote/Dockerfile | docker build -f - .
# 아래의 명령은 Dockerfile을 찾는 대신 debug 파일의 내용을 사용하고, /home/me/myapp을 빌드 컨텍스트의 루트로 사용합니다.
$ docker build -f /home/me/myapp/dockerfiles/debug /home/me/myapp
$ docker build -t vieux/apache:2.0 .          # -t, --tag 옵션으로 결과 이미지에 태그 지정
$ docker build -t mybuildimage --target build-env .    # 특정 스테이지(stage)만 빌드
```

$ docker image history [OPTIONS] IMAGE

이미지 히스토리 표시

$ docker image import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]

파일 시스템 이미지를 생성하기 위해 tarball에서 콘텐츠를 가져옵니다.

```
$ docker import https://example.com/exampleimage.tgz
$ docker import /path/to/exampleimage.tgz
$ cat exampleimage.tgz | docker import - exampleimagelocal:new
```

$ docker image inspect [OPTIONS] IMAGE [IMAGE...]

하나 이상의 이미지에 대한 자세한 정보 표시

$ docker image load [OPTIONS]

tar 아카이브 또는 STDIN에서 이미지 로드

```
$ docker load < [path/to/image_file.tar]
$ docker load --input [path/to/image_file.tar]
$ docker load --quiet --input [path/to/image_file.tar]
```

$ docker image ls [OPTIONS] [REPOSITORY[:TAG]]

이미지 리스트 확인

```
$ docker images
$ docker images --all
# 이미지의 long ID 확인
$ docker images --no-trunc
# 다이제스트 확인
$ docker images --digests
$ docker images --filter dangling=true
$ docker images "[*name*]"
$ docker images --format 'table {{.Repository}}₩t{{.Tag}}₩t{{.ID}}₩t{{.CreatedAt}}₩t{{.Size}}'
```

$ docker image prune [OPTIONS]

사용하지 않은 이미지 제거

```
# 사용하지 않는 로컬 Docker 이미지 삭제
$ docker image prune
# 사용하지 않는 모든 이미지를 제거
$ docker image prune -a 또는 $ docker image prune --all
# 2017-01-04T00:00:00 이전에 생성된 이미지 삭제
$ docker image prune -a --force --filter "until=2017-01-04T00:00:00"
# 생성된 지 10일이 지난 이미지 삭제
$ docker image prune -a --force --filter "until=240h"
# label을 이용한 삭제
$ docker image prune --filter="label=maintainer=john"
```

$ docker image pull [OPTIONS] NAME[:TAG|@DIGEST]

레지스트리에서 이미지 다운로드
대부분의 이미지는 Docker Hub 레지스트리에서 기본 이미지 위에 생성됩니다.

```
# 특정 이미지 또는 이미지 세트 다운로드 (docker pull 은 docker image pull의 shorthand이다.) tag를 지정하지 않으면 Docker 엔진은 ':latest' 태그를 기본
$ docker image pull debian
# 태그를 지정
$ docker pull ubuntu:22.04
# 어떤 경우에는 이미지를 최신 버전으로 업데이트하지 않고 고정된 버전의 이미지를 사용하는 것을 선호합니다. Docker는 이미지의 다이제스트(digest)로 이
# 다이제스트(digest)로 이미지를 가져올 때 가져올 이미지 버전을 정확하게 지정합니다. 이렇게 하면 이미지를 해당 버전에 "고정"할 수 있으며 사용 중인 이
# 다이제스트는 불변 식별자(immutable identifier) 입니다.
$ docker pull ubuntu@sha256:26c68657ccce2cb0a31b330cb0be2b5e108d467f641c62e13ab40cbec258c68d
```

$ docker image tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]

SOURCE_IMAGE를 참조하는 TARGET_IMAGE 태그를 생성합니다.
전체 이미지 이름의 형식과 구성요소는 다음과 같습니다.

[HOST[:PORT_NUMBER]/]PATH

- HOST: 선택적 레지스트리 호스트 이름(hostname)은 이미지가 있는 위치를 지정합니다.
  호스트 이름은 표준 DNS 규칙을 준수해야 하지만 밑줄(underscore)을 포함하지 않을 수 있습니다.
  호스트 이름이 지정되지 않은 경우 명령은 기본적으로 registry-1.docker.io에 있는 Docker의 공개 레지스트리를 사용합니다.
  docker.io는 Docker의 공개 레지스트리에 대한 표준 참조입니다.
- PORT_NUMBER: 호스트 이름이 있는 경우 선택적으로 :8080 형식의 레지스트리 포트 번호가 뒤에 올 수 있습니다.
- PATH: 경로는 슬래시(/)로 구분된 구성 요소로 구성됩니다. 각 구성 요소에는 소문자, 숫자, 구분자가 포함될 수 있습니다.
  구분자는 마침표(period), 하나 또는 두 개의 밑줄(underscore), 하나 이상의 하이픈(hyphen)으로 정의됩니다.
  구성 요소는 구분자로 시작하거나 끝날 수 없습니다.

  ✅ Docker 공개 레지스트리의 경우 경로 형식은 다음과 같습니다.
  [NAMESPACE/]REPOSITORY: 첫 번째 선택적 구성 요소는 일반적으로 사용자 또는 조직의 네임스페이스입니다.
  두 번째 필수 구성 요소는 저장소 이름입니다. 네임스페이스를 지정하지 않으면 Docker는 library를 기본 네임스페이스로 사용합니다.

이미지 이름 뒤의 선택적으로 붙는 TAG는 사람이 읽을 수 있는 매니페스트 식별자입니다. (이미지의 특정 버전 또는 변종)
TAG가 지정되지 않으면 명령은 기본적으로 latest 를 사용합니다.

```
# ID가 "0e5574283393"인 로컬 이미지에 "version1.0" 태그를 사용하여 "fedora/httpd"로 태그를 지정
$ docker tag 0e5574283393 fedora/httpd:version1.0
# 이름이 "httpd"인 로컬 이미지에 "version1.0" 태그를 사용하여 "fedora/httpd"로 태그를 지정
$ docker tag httpd fedora/httpd:version1.0
# 공개 Docker 레지스트리가 아닌 개인 레지스트리에 이미지를 푸시하려면 레지스트리 호스트 이름과 포트(필요한 경우)를 포함해야 합니다.
$ docker tag 0e5574283393 myregistryhost:5000/fedora/httpd:version1.0
```

$ docker image rm [OPTIONS] IMAGE [IMAGE...]

host에서 하나 이상의 이미지 제거
이미지에 태그(tag)가 여러 개 있는 경우 태그와 함께 이미지를 지정하면 태그만 제거됩니다.
만약 태그가 이미지에 대한 유일한 태그인 경우 이미지와 태그가 모두 제거됩니다.
-f 옵션을 사용하지 않으면 실행 중인 컨테이너의 이미지를 제거할 수 없습니다.
short ID / long ID 또는 이미지 tag 또는 이미지 다이제스트(digest)로 이미지를 제거할 수 있습니다.
이미지에 참조하는 태그가 하나 이상 있으면 이미지를 제거하기 전에 해당 태그를 모두 제거해야 합니다.
태그에 의해 이미지가 제거되면 다이제스트 참조가 자동으로 제거됩니다.
-f 옵션으로 short ID 또는 long ID로 지정하면 해당 ID와 일치하는 모든 이미지의 태그를 해제하고 이미지도 제거합니다.

```
$ docker rmi fd484f19954f
$ docker rmi test1:latest
$ docker rmi -f fd484f19954f
$ docker rmi localhost:5000/test/busybox@sha256:cbbf2f9a99b47fc460d422812b6a5adff7dfee951d8fa2e4a98caa0382cfbdbf
```

$ docker image save [OPTIONS] IMAGE [IMAGE...]

하나 이상의 이미지를 tar 아카이브에 저장 (기본적으로 STDOUT으로 스트리밍)
표준 출력 스트림에 대한 tar 저장소를 생성합니다.
제공된 각각의 인수에 대해 모든 상위 레이어(parent layers), 모든 태그 + 버전, 지정된 repo:tag를 포함합니다.

```
$ docker save busybox > busybox.tar
# 이미지의 모든 태그를 저장
$ docker save --output busybox.tar busybox
$ docker save -o fedora-all.tar fedora

$ docker save -o fedora-latest.tar fedora:latest
$ docker save myimage:latest | gzip > myimage_latest.tar.gz
# 저장할 이미지의 특정 태그를 선별적으로 선택
$ docker save -o ubuntu.tar ubuntu:lucid ubuntu:saucy
```

$ docker image push [OPTIONS] NAME[:TAG]

레지스트리(registry)에 이미지 업로드
푸시하는 도중에 CTRL + c 로 인터럽트 걸면 중단됩니다.

```
# 태그 지정 후 push
# 이 예제는 레지스트리는 registry-host라는 호스트에 있고 포트 5000에서 수신 대기중
$ docker image tag rhel-httpd:latest registry-host:5000/myadmin/rhel-httpd:latest
$ docker image push registry-host:5000/myadmin/rhel-httpd:latest

# 이미지에 태그를 여러개 생성 후 모두 Docker Hub에 푸시
# -a (or --all-tags) 옵션으로 로컬 이미지의 모든 태그를 푸시
$ docker image tag myimage registry-host:5000/myname/myimage:latest
$ docker image tag myimage registry-host:5000/myname/myimage:v1.0.1
$ docker image tag myimage registry-host:5000/myname/myimage:v1.0
$ docker image tag myimage registry-host:5000/myname/myimage:v1
$ docker image push --all-tags registry-host:5000/myname/myimage
```

Docker Container 관련 명령들 (docker run 제외)

$ docker container attach [OPTIONS] CONTAINER

실행 중인 컨테이너에 로컬(local) 표준 입력(input), 표준 출력(output), 표준 에러(error) 스트림을 연결합니다.
컨테이너의 ID 또는 이름을 사용하여 표준 입력, 표준 출력, 표준 에러 세 가지의 조합을 연결
진행 중인 출력을 보거나 대화식으로 제어할 수 있습니다.
attach 명령은 ENTENPOINT/CMD 프로세스의 출력을 표시합니다.
이는 실제로 프로세스가 그 당시 터미널과 상호 작용하지 않을 수 있는데도 attach 명령이 정지된 것처럼 나타날 수 있습니다.

Docker 호스트의 다른 세션에서 동일한 컨테이너 프로세스에 여러번 attach 할 수 있습니다.
컨테이너를 중지(stop)하려면 `CTRL + c`를 사용하세요. 그러면 SIGKILL 을 컨테이너로 보냅니다. 만약 --sig-proxy가 true(기본값)이면 CTRL + c는 SIGINT를 컨테이너로 보냅니다.
컨테이너를 실행(run)할 때 `-i`, `-t`로 실행된 경우 `CTRL + p, CTRL + q` 를 순서대로 입력하면 컨테이너를 빠져나와도 실행 상태를 유지할 수 있습니다.

```
$ docker run -d --name topdemo ubuntu:22.04 /usr/bin/top -b
$ docker attach topdemo

$ docker run -dit --name topdemo2 ubuntu:22.04 /usr/bin/top -b
$ docker attach topdemo2
```

$ docker container commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

컨테이너의 변경사항에서 새 이미지 만들기
이를 통해 대화형 shell을 실행하여 컨테이너를 디버깅하거나 작업 데이터 세트를 다른 서버로 내보낼 수 있습니다.
commit 작업에는 컨테이너 내부에 마운트된 볼륨에 포함된 데이터가 포함되지 않습니다.

기본적으로 commit되는 컨테이너와 해당 프로세스는 이미지가 commit되는 동안 일지 중지(pause)됩니다. 이렇게 하면 commit을 생성하는 과정에서 데이터 손상이 발생할 가능성이 줄어듭니다. 만약 이 동작을 원하지 않으면 --pause 옵션을 false로 설정하십시오.

--change 옵션은 생성된 이미지에 Dockerfile 명령(instructions)을 적용합니다. 지원되는 Dockerfile 명령:
CMD|ENTRYPOINT|ENV|EXPOSE|LABEL|ONBUILD|USER|VOLUME|WORKDIR

```
# 컨테이너 ID를 지정
$ docker commit c3f279d17e0a  svendowideit/testimage:version3
$ docker commit --change "ENV DEBUG=true" c3f279d17e0a  svendowideit/testimage:version3
$ docker commit --change='CMD ["apachectl", "-DFOREGROUND"]' -c "EXPOSE 80" c3f279d17e0a  svendowideit/testimage:version4
```

$ docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-
$ docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH

컨테이너와 로컬 파일 시스템 간에 파일/폴더 복사
SRC_PATH 또는 DEST_PATH에 `-`가 지정된 경우 STDIN 또는 STDOUT에서 tar 아카이브를 스트리밍할 수도 있습니다.

`-`을 SRC_PATH로 사용하면 STDIN의 내용을 tar 아카이브로 스트리밍합니다.
`-`을 DEST_PATH로 사용하면 리소스 내용이 tar 아카이브로 STDOUT으로 스트리밍됩니다.
CONTAINER는 실행 중이거나 중지된 컨테이너일 수 있습니다.

docker cp 명령은 컨테이너 경로가 컨테이너의 /(root) 디렉터리에서 상대적이라고 가정합니다. 이는 경로 앞에 슬래시(/) 제공이 선택 사항임을 의미합니다.
즉, container_name:/tmp/foo/myfile.txt와 container_name:tmp/foo/myfile.txt 는 동일합니다.

로컬 머신 경로는 절대 경로, 상대 경로 둘 다 가능합니다.
이 명령은 로컬 시스템의 상대 경로를 'docker cp'가 실행되는 현재 작업 디렉터리를 기준으로 해석합니다.

```
# 로컬 파일을 컨테이너에 복사
$ docker cp ./some_file CONTAINER:/work
# 컨테이너에서 로컬 시스템으로 파일 복사
$ docker cp CONTAINER:/var/logs/ /tmp/app_logs
# 컨테이너에서 stdout으로 파일을 복사
$ docker cp CONTAINER:/var/logs/app.log - | tar x -O | grep "ERROR"
```

$ docker container create [OPTIONS] IMAGE [COMMAND] [ARG...]

지정된 이미지에서 새 컨테이너를 생성
docker create 명령은 docker run 명령과 대부분의 옵션을 공유합니다.

```
$ docker container create -i -t --name mycontainer alpine
```

$ docker container diff CONTAINER

컨테이너의 파일 시스템의 파일 또는 디렉터리의 변경 사항을 검사합니다.
컨테이너가 생성된 이후 컨테이너의 파일 시스템에서 변경된 파일과 디렉터리를 표시
세 가지 유형의 변경 사항이 추적됩니다.

A	파일 또는 디렉터리가 추가됨(added)
D	파일 또는 디렉터리가 삭제됨(deleted)
C	파일 또는 디렉터리가 수정됨(changed)

```
$ docker diff [full container ID|short container ID|container name]
```

$ docker container exec [OPTIONS] CONTAINER COMMAND [ARG...]

실행 중인(running) 컨테이너에서 명령 실행합니다.
주어진 COMMAND는 컨테이너의 기본 디렉터리에서 실행됩니다. 만약 컨테이너의 기초가 되는 이미지의 Dockerfile에 WORKDIR 명령에 지정된 사용자 디렉터리가 있는 경우 이 디렉터리가 대신 사용됩니다.
주어진 COMMAND는 실행 파일이어야 합니다. 연결되거나(chained) 인용된(quoted) 명령은 작동하지 않습니다. 예를 들어,
`$ docker exec -it my_container sh -c "echo a && echo b"` 는 작동하지만
`$ docker exec -it my_container "echo a && echo b"`는 작동하지 않습니다.

```
# 실행 중인 컨테이너에서 대화형 shell 세션 시작
$ docker exec --interactive --tty [container_name] [/bin/bash]
# 실행 중인 컨테이너의 백그라운드에서 명령 실행
$ docker exec --detach [container_name] [command]
# 특정 명령을 실행할 작업 디렉터리를 선택
$ docker exec --interactive -tty --workdir [path/to/directory] [container_name] [command]
# 컨테이너의 백그라운드에서 명령을 실행하되 stdin을 열어둔 상태를 유지
$ docker exec --interactive --detach [container_name] [command]
# 실행 중인 Bash 세션에서 환경 변수 설정(--env 여러개 지정 가능)
$ docker exec --interactive --tty --env [variable_name]=[value] [container_name] [/bin/bash]
# 특정 사용자로 명령 실행
$ docker exec --user [user] [container_name] [command]
```

$ docker container export [OPTIONS] CONTAINER

컨테이너의 파일 시스템을 tar 아카이브로 내보낸다.

```
# 아래 두 명령의 결과는 갔음
$ docker export red_panda > latest.tar
$ docker export --output="latest.tar" red_panda
```

$ docker container inspect [OPTIONS] CONTAINER [CONTAINER...]

하나 이상의 컨테이너에 대한 자세한 정보 표시

$ docker container kill [OPTIONS] CONTAINER [CONTAINER...]

하나 이상의 실행 중인 컨테이너 종료
컨테이너 내부의 메인 프로세스에 SIGKILL 시그널(기본값) 또는 --signal 옵션으로 지정된 시그널이 전송됩니다.
ID 또는 이름으로 컨테이너를 지정할 수 있습니다.

--signal 플래그는 컨테이너로 전송되는 시스템 콜 시그널을 지정합니다. 이 시그널은 SIGINT와 같은 SIG<NAME> 형식의 시그널 이름 또는 커널의 syscall 테이블 위치에 일치하는 부호 없는 숫자(예: 2)일 수 있습니다.

기본값(SIGKILL)은 컨테이너를 종료하지만 --signal을 통해 설정된 시그널은 컨테이너의 메인 프로세스에 따라 종료가 아닐 수 있습니다.
shell 형태의 ENTRYPOINT 및 CMD는 /bin/sh -c의 자식 프로세스로 실행되며, 이는 시그널은 전송하지 않습니다.
이는 실행 파일이 컨테이너의 PID 1이 아니며 UNIX 시그널은 수신하지 않음을 의미합니다.

```
$ docker kill my_container
$ docker kill --signal=SIGHUP my_container
# 시그널을 이름이나 번호로 지정 가능, 시그널의 'SIG' 접두사는 선택 사항이므로 다음 예제 3개는 동일합니다.
$ docker kill --signal=SIGHUP my_container
$ docker kill --signal=HUP my_container
$ docker kill --signal=1 my_container
```

$ docker container logs [OPTIONS] CONTAINER

컨테이너 로그 가져오기
--follow 옵션은 컨테이너의 STDOUT 및 STDERR에서 새 출력을 계속 스트리밍합니다.

--details 옵션은 컨테이너를 생성할 때 --log-opt에 제공되는 환경 변수 및 레이블과 같은 추가 속성을 보여줍니다.

```
$ docker logs [container_name]
$ docker logs -f [container_name]
$ docker logs [container_name] --tail [number of lines]
$ docker logs -t [container_name]
$ docker logs [container_name] --until [time]
```

$ docker container ls [OPTIONS]
$ docker ps [OPTIONS]

컨테이너 목록 확인
✅ 옵션이 매우 많으므로 문서 참고

```
$ docker ps --no-trunc
$ docker ps --size
$ docker ps -a
$ docker ps -a --format 'table {{.ID}}₩t{{.Image}}₩t{{.Command}}₩t{{.CreatedAt}}₩t{{.Status}}'
$ docker ps --filter "label=color"
$ docker ps --filter "name=nostalgic"
$ docker ps -a --filter 'exited=0'
$ docker ps -a --filter 'exited=137'
$ docker ps --filter status=running
$ docker ps --filter status=paused
$ docker ps --filter status=exited -q
$ docker ps --filter ancestor=ubuntu:22.04
$ docker ps --filter ancestor=d0e008c6cf02
$ docker ps -f before=9c3527ed70ce
$ docker ps -f since=6e63f6ff38b0
$ docker ps --filter volume=remote-volume --format "table {{.ID}}₩t{{.Mounts}}"
$ docker ps --filter network=net1
$ docker ps --filter network=8c0b4110ae930dbe26b258de9bc34a03f98056ed6f27f991d32919bfe401d7c5
$ docker ps --filter publish=80
$ docker ps --filter expose=8000-8080/tcp
```

$ docker container pause CONTAINER [CONTAINER...]

하나 이상의 컨테이너 내의 모든 프로세스를 일시 중지합니다.
전통적으로 프로세스를 일시중단할 때 SIGSTOP 시그널이 사용되며 이는 일시중단 중인 프로세스에서 관찰할 수 있습니다.

```
$ docker pause my_container
```

$ docker container port CONTAINER [PRIVATE_PORT[/PROTO]]

컨테이너에 매핑된 포트를 표시

```
$ docker port
$ docker port conatiner_name 8100/tcp
$ docker port conatiner_name 8100/udp
$ docker port conatiner_name 8100
```

$ docker container prune [OPTIONS]

중지(stop) 상태의 컨테이너를 모두 제거

```
# 5분 이상 전에 생성된 stop 상태의 컨테이너를 제거
$ docker container prune --force --filter "until=5m"
```

```
# 지정된 시각 이전에 생성된 stop 상태의 컨테이너 제거
$ docker container prune --force --filter "until=2017-01-04T13:10:00"
```

$ docker container rename CONTAINER NEW_NAME

컨테이너 이름 변경

```
$ docker rename my_container my_new_container
```

$ docker container restart [OPTIONS] CONTAINER [CONTAINER...]

하나 이상의 컨테이너를 다시 시작

```
$ docker restart my_container
```

$ docker container rm [OPTIONS] CONTAINER [CONTAINER...]

하나 이상의 컨테이너를 제거

```
# 컨테이너 제거
$ docker rm [container1 container2 ...]
$ docker rm --force [container1 container2 ...]
# 컨테이너 및 해당 볼륨 제거
$ docker rm --volumes [container]
# /redis 링크에서 참조되는 컨테이너 제거
$ docker rm /redis
# 이렇게 하면 기본 브릿지 네트워크에서 /webapp과 /redis 컨테이너 사이의 링크가 제거되어 두 컨테이너 간의 모든 네트워크 통신이 제거됩니다.
$ docker rm --link /webapp/redis
```

$ docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]

🔴 이전 파일에 정리된 내용 참고 🔴

$ docker container start [OPTIONS] CONTAINER [CONTAINER...]

하나 이상의 stop 상태의 컨테이너를 시작

```
$ docker start my_container
```

$ docker container stats [OPTIONS] [CONTAINER...]

컨테이너 리소스 사용 통계의 실시간 스트림 표시

```
$ docker stats --all
# 디폴트 format On Linux :
# "table {{.ID}}₩t{{.Name}}₩t{{.CPUPerc}}₩t{{.MemUsage}}₩t{{.MemPerc}}₩t{{.NetIO}}₩t{{.BlockIO}}₩t{{.PIDs}}"
# 디폴트 format On Windows :
# "table {{.ID}}₩t{{.Name}}₩t{{.CPUPerc}}₩t{{.MemUsage}}₩t{{.NetIO}}₩t{{.BlockIO}}"
$ docker stats --format "table {{.Container}}₩t{{.CPUPerc}}₩t{{.MemUsage}}"
$ docker stats --all --format "table {{.Container}}₩t{{.CPUPerc}}₩t{{.MemUsage}}" fervent_panini 5acfcb1b4fd1 humble_visvesvaraya big_heisenberg
```

$ docker container stop [OPTIONS] CONTAINER [CONTAINER...]

하나 이상의 실행 중인(running) 컨테이너 중지
컨테이너 내부의 메인 프로세스는 SIGTERM을 수신하고 유예 기간 후에 SIGKILL을 수신합니다.
첫 번째 시그널은 컨테이너 Dockerfile의 STOPSIGNAL 명령 또는 docker run 의 --stop-signal 옵션으로 변경할 수 있습니다.

```
$ docker stop my_container
```

$ docker container top CONTAINER [ps OPTIONS]

컨테이너의 실행 중인 프로세스 표시

$ docker container unpause CONTAINER [CONTAINER...]

하나 이상의 컨테이너 내의 모든 프로세스 일시 중지를 해제

```
$ docker unpause my_container
```

$ docker container update [OPTIONS] CONTAINER [CONTAINER...]

하나 이상의 컨테이너 구성 업데이트
Block IO, CPU 점유율, 메모리, 스왑(Swap), 리스타트 정책(Restart policy) 등등 구성을 변경한다.

```
$ docker update --cpu-shares 512 abebf7571666
$ docker update --cpu-shares 512 -m 300M abebf7571666 hopeful_morse
$ docker update --restart=on-failure:3 abebf7571666 hopeful_morse
```

$ docker container wait CONTAINER [CONTAINER...]

하나 이상의 컨테이너가 중지(stop)될 때까지 차단(block)한 다음 종료(exit) 코드를 표시
컨테이너가 중지될 때까지 차단한다. 다른 터미널에서 해당 컨테이너를 stop 하면 0이 리턴된다.

```
# 이렇게 wait을 걸어두고 해당 컨테이너가 stop 하면 0이 출력된다.
$ docker wait my_container
```