Docker 기본명령1 (docker run)

Docker는 격리된 컨테이너에서 프로세스를 실행합니다. 컨테이너는 호스트에서 실행되는 프로세스입니다.
사용자가 `docker run`을 실행하면 실행되는 컨테이너 프로세스는 자체 파일 시스템, 자체 네트워킹 및 호스트와 별도로 격리된 자체 프로세스 트리를 갖는다는 점에서 격리됩니다.

$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]

Detached vs foreground

Docker 컨테이너를 실행할 때 컨테이너를 백그라운드(background)에서 "detached" 모드로 실행할지 기본 포그라운드(foreground) 모드로 실행할지 먼저 결정해야 합니다.
Detached 모드에서 컨테이너를 시작하려면 -d=true 또는 -d 옵션만 사용합니다.
--rm 옵션도 지정하지 않는 한, 설계상 분리 모드에서 시작된 컨테이너는 컨테이너를 실행하는 데 사용된 루트 프로세스가 종료될 때 종료됩니다.

분리된 컨테이너에 다음과 같이 `service x start` 명령을 전달하지 마세요.

$ docker run -d -p 80:80 my_image service nginx start $ docker run -d --name topdemo ubuntu:22.04 /usr/bin/top -b

컨테이너 내부에서 nginx 서비스를 시작하는 데 성공했습니다. 그러나 루트 프로세스(service nginx start)가 반환되고 분리된 컨테이너가 설계대로 중지된다는 점에서 분리된 컨테이너 패러다임에 실패합니다.
이로 인해 nginx 서비스가 시작되지만 사용할 수 없습니다. 대신 nginx 웹 서버와 같은 프로세스를 시작하려면 다음을 수행하세요.

$ docker run -d -p 80:80 my_image nginx -g 'daemon off;'

분리된 컨테이너에 다시 연결하려면 `docker attach` 명령을 사용합니다.

포그라운드(foreground) 모드(-d 옵션이 없는 경우 기본값)에서 docker run은 컨테이너에서 프로세스를 시작하고 콘솔을 프로세스의 표준 입력, 표준 출력, 표준 에러에 연결할 수 있습니다.
심지어 TTY(대부분의 명령줄 실행 파일이 기대하는 방식)인 것처럼 행세하여 시그널(signal)을 전달할 수도 있습니다.

옵션	설명
-a=[]	Attach to `STDIN`, `STDOUT` and/or `STDERR` -a를 지정하지 않으면 Docker는 stdout, stderr 둘 다 연결합니다. 다음과 같이 연결할 세 가지 표준 스트림(STDIN, STDOUT, STDERR)을 지정할 수 있습니다. $ docker run -a stdin -a stdout -i -t ubuntu /bin/bash
-t	pseudo-tty 할당. << 과거 히스토리 ... >> -t 옵션은 Unix/Linux가 터미널 엑세스를 처리하는 방법을 지정합니다. 과거에는 hardline 연결이었지만 나중에는 모뎀(modem) 기반 연결이었습니다. 이것들은 물리적인 장치 드라이버가 있습니다. 일반화된 네트워크가 사용되기 시작하자 pseudo-terminal 드라이버가 개발되었습니다. 프로그램에 직접 쓸 필요 없이 사용할 수 있는 터미널 기능이 무엇인지 이해하는 것(stty, curses에 대한 man page) 사이에 괴리가 생기기 때문입니다. Shell과 같은 대화형 프로세스의 경우 컨테이너 프로세스에 tty를 할당하려면 `-i -t`를 함께 사용해야 합니다. `-i -t`는 나중에 예제에서 볼 수 있듯이 `-it`로 자주 사용합니다. 다음과 같이 클라이언트가 파이프에서 표준 입력을 수신할 때 -t를 지정하는 것은 금지된다. $ echo test | docker run -i busybox cat
--sig-proxy=true	수신된 모든 signal을 프로세스로 프록시(proxy)합니다. (TTY 모드가 아닌 경우에만 해당)
-i	연결되지 않은 경우(not attached)에도 표준입력(STDIN)을 열어 둔다.

★ NOTE ★
컨테이너 안에서 PID 1 로 실행되는 프로세스는 Linux에서 특별하게 다뤄진다.
기본적으로 모든 시그널을 무시한다.
따라서 해당 프로세스는 종료하도록 코딩되지 않는 한 SIGINT 또는 SIGTERM에서 종료되지 않는다.

Container identification

1. 사용자는 세 가지 방법으로 컨테이너를 식별할 수 있다.

Identifier type	Example value
UUID long identifier	"f78375b1c487e03c9438c729345e54db9d20cfa2ac1fc3494b6eb60872e74778"
UUID short identifier	"f78375b1c487"
Name	"evil_ptolemy"

UUID 식별자는 Docker 데몬에서 나온다. --name 옵션으로 컨테이너 이름을 명시하지 않으면 Docker 데몬이 임의의 문자열 이름을 생성한다.
이름을 지정하면 컨테이너의 의미를 추가하는 편리한 방법이 될 수 있다. 이름을 지정하면 Docker 네트워크 내에서 컨테이너를 참조할 때 해당 이름을 사용할 수 있다.

이는 백그라운드(background), 포그라운드(foreground) 컨테이너 모두 해당한다.

기본 브릿지 네트워크의 컨테이너들은 이름으로 통신하려면 연결되어야 한다.

마지막으로 자동화를 돕기 위해 --cidfile 옵션으로 사용자가 선택한 파일에 컨테이너 ID를 쓰도록 할 수 있다. 이는 일부 프로그램이 프로세스 ID를 파일에 기록하는 방법과 유사하다.

2. Image[:tag]
컨테이너를 식별하는 수단은 아니지만 image[:tag]를 추가하여 컨테이너를 실행할 이미지의 버전을 지정할 수 있습니다.
$ docker run ubuntu:22.04

3. Image[@digest]
v2 이상의 이미지 형식을 사용하는 이미지들은 '다이제스트(digest)'라고 하는 컨텐츠 주소 지정 식별자가 있습니다. 이미지를 생성하는 데 사용되는 입력이 변경되지 않는 한 다이제스트 값은 예측 가능하고 참조 가능합니다.

Network settings

옵션	설명
--dns=[]	컨테이너에 대한 사용자정의 DNS서버 설정
--network="bridge" (기본값)	컨테이너를 네트워크에 연결 'bridge' : 기본 Docker 브릿지에 네트워크 스택 생성 'none' : 네트워킹 하지 않음 'container:<name|id>' : 다른 컨테이너의 네트워크 스택 재사용 'host': Docker 호스트 네트워크 스택 사용
--network-alias=[]	컨테이너에 대한 네트워크 범위 별칭 추가
--add-host=""	/etc/hosts(호스트:IP)에 한 줄 추가
--mac-address=""	컨테이너의 이더넷 디바이스의 MAC주소를 설정
--ip=""	컨테이너의 이더넷 디바이스의 IPv4주소를 설정
--ip6=""	컨테이너의 이더넷 디바이스의 IPv6주소를 설정
--link-local-ip=[]	하나 이상의 컨테이너의 이더넷 디바이스의 link local IPv4/IPv6 주소를 설정

기본적으로 모든 컨테이너에는 네트워킹이 활성화되어 있으며 나가는(outgoing) 커넥션을 만들 수 있다.
사용자는 `docker run --network none`을 사용하여 들어오고(incoming), 나가는(outgoing) 네트워킹을 완전히 비활성화할 수 있다.
이와 같은 경우 파일을 통해서 I/O를 수행하거나 STDIN, STDOUT만 수행한다.

포트 게시(Publishing ports) 및 다른 컨테이너에 대한 연결(linking)은 기본값(bridge)에서만 작동한다.
linking은 레거시 기능이다. 항상 linking 보다 Docker 네트워크 드라이버를 사용하는 것을 선호해야 한다.

컨테이너는 기본적으로 호스트와 동일한 DNS 서버를 사용하지만 --dns를 사용하여 이를 재정의할 수 있다.

기본적으로 MAC 주소는 컨테이너에 할당된 IP 주소를 사용하여 생성된다.
--mac-address 옵션으로 컨테이너의 MAC 주소(format:12:34:56:78:9a:bc)를 명시적으로 설정할 수 있다.
Docker는 수동으로 지정한 MAC 주소가 고유한(unique) 값인지 체크하지 않는다.

네트워크	설명
none	컨테이너에 네트워킹이 없음
bridge(default)	veth 인터페이스를 통해 컨테이너를 브릿지에 연결 네트워크를 bridge로 설정하면 컨테이너는 Docker의 기본 네트워킹 설정을 사용한다. 일반적으로 docker0 라는 이름으로 host에 브릿지가 설정되고 컨테이너에 대한 한 쌍의 veth 인터페이스가 생성된다. veth 쌍의 한쪽은 브릿지에 연결된 호스트에 유지되고 다른 쪽은 컨테이너의 네임스페이스 내부에 배치된다. 브릿지 네트워크에 있는 컨테이너들에게 IP 주소가 할당되고 트래픽은 이 브릿지를 통해 컨테이너로 라우팅된다. 컨테이너들은 기본적으로 그들의 IP를 통해 커뮤니케이션을 할 수 있다. 컨테이너 이름으로 통신하려면 연결되어 있어야 한다.
host	컨테이너 내부의 호스트 네트워크 스택을 사용 네트워크를 host로 설정하면 컨테이너는 호스트의 네트워크 스택을 공유하고 호스트의 모든 인터페이스를 컨테이너에서 사용할 수 있다. 컨테이너의 호스트 이름은 호스트 시스템의 호스트 이름과 일치한다. --mac-address 옵션은 host 네트워크 모드에서는 유효하지 않다. host 네트워크 모드에서도 컨테이너는 기본적으로 자신만의 UTS 네임스페이스를 갖는다. 따라서 --hostname, --domainname은 host 네트워크 모드에서 허용되며 오직 컨테이너 내부의 호스트 이름과 도메인 이름만 변경한다. --hostname과 유사하게 --add-host, --dns, --dns-search, --dns-option 옵션을 host 네트워크 모드에서 사용할 수 있다. 이러한 옵션들은 컨테이너 내부의 /etc/hosts 또는 /etc/resolv.conf를 업데이트한다. 호스트 머신의 /etc/hosts, /etc/resolv.conf는 변경되지 않는다. 기본(default) 브릿지 모드와 비교하여 host 모드는 호스트 머신의 네이티브 네트워킹 스택을 사용하기 때문에 훨씬 더 나은 네트워킹 성능을 제공하지만 bridge는 Docker 데몬을 통해 한 단계의 가상화를 거쳐야 합니다. 운영 로드 밸런서 또는 고성능 웹 서버와 같이 네트워킹 성능이 중요한 경우 이 모드에서 컨테이너를 실행하는 것이 좋습니다.

container:
<name|id>
	이름이나 ID를 통해 지정된 다른 컨테이너의 네트워크 스택을 사용
	네트워크가 container로 설정되면 컨테이너는 '다른 컨테이너'의 네트워크 스택을 공유합니다.
	위에서 말한 '다른 컨테이너'의 이름은 --network container:<name|id> 형식으로 제공되어야 합니다.
	
	--add-host, --hostname, --dns, --dns-search, --dns-option, --mac-address 는 container 네트워크 모드에서 유효하지 않으며
	--publish, --publish-all, --expose 또한 container 네트워크 모드에서 유효하지 않습니다.
	
	$ docker run -d --name redis example/redis --bind 127.0.0.1
	$ # use the redis container's network stack to access localhost
	$ docker run --rm -it --network container:redis example/redis-cli -h 127.0.0.1

NETWORK	컨테이너를 사용자 생성 네트워크에 연결(docker network create 명령 사용)
	Docker 네트워크 드라이버 또는 외부 네트워크 드라이버 플러그인을 사용하여 네트워크를 생성할 수 있습니다.
	여러 컨테이너를 동일한 네트워크에 연결할 수 있습니다.
	사용자 정의 네트워크에 연결되면 컨테이너는 다른 컨테이너의 IP 주소나 이름만 사용하여 쉽게 통신할 수 있습니다.
	다중 호스트 연결을 지원하는 overlay 네트워크 또는 사용자 정의 플러그인의 경우 동일한 다중 호스트 네트워크에 연결되어 있지만 서로 다른 엔진에서 시작된 컨테이너도 이러한 방식으로 통신할 수 있습니다.
	
	$ docker network create -d bridge my-net
	$ docker run --network=my-net -itd --name=container3 busybox

/etc/hosts 파일 관리하기	컨테이너에는 /etc/hosts에 컨테이너 자체의 hostname 뿐만 아니라 localhost 및 기타 몇 가지 일반적인 사항을 정의하는 행(line)이 있습니다. --add-host 플래그로 /etc/hosts에 라인을 추가할 수 있습니다. 컨테이너가 기본 브리지 네트워크에 연결되어 있고 다른 컨테이너와 연결되어 있는 경우 컨테이너의 /etc/hosts 파일은 연결된 컨테이너의 이름으로 업데이트됩니다. Docker는 컨테이너의 /etc/hosts 파일을 실시간으로 업데이트할 수 있기 때문에 컨테이너 내부의 프로세스가 비어있거나 불완전한 /etc/hosts 파일을 읽는 상황이 발생할 수 있습니다. 대부분의 경우 다시 읽기를 시도하면 문제가 해결됩니다.

Restart policies (--restart)

Docker 실행할 때 --restart 옵션으로 컨테이너 종료(exit)할 때 재시작하거나 또는 재시작하지 않아야 한는 리스타트 규정을 명시할 수 있습니다.
컨테이너에서 리스타트 정책이 활성화되면 $ docker ps 에서 Up 또는 Restarting으로 표시됩니다.
$ docker event 를 사용하여 리스타트 정책이 적용되는지 확인하는 것도 유용할 수 있습니다.

Policy	Result
no (기본값)	컨테이너가 종료될 때 자동으로 재시작(restart) 하지 않습니다.
on-failure[:max-retries]	컨테이너가 0이(non-zero) 아닌 종료 상태로 종료되는 경우에만 재시작합니다. 선택적으로 Docker 데몬이 시도하는 재시작 재시도(retries) 횟수를 제한합니다. # 최대 리스타트 시도 횟수 10으로 Redis 컨테이너가 실행됩니다. $ docker run --restart=on-failure:10 redis
always	종료 상태에 관계없이 항상 컨테이너를 재시작합니다. always를 지정하면 Docker 데몬이 컨테이너를 무한히 재시작하려고 시도합니다. 컨테이너의 현재 상태와 관계없이 컨테이너는 데몬 시작 시 항상 시작됩니다. # 컨테이너가 종료(exit)되면 Docker가 컨테이너를 항상 다시 시작하도록 리스타트 정책으로 Redis 컨테이너가 실행됩니다. $ docker run --restart=always redis
unless-stopped	Docker 데몬이 중지(stop)되기 전에 컨테이너가 중지된 상태인 경우를 제외하고 데몬 시작 시점을 포함하여 종료(exit) 상태에 관계없이 항상 컨테이너를 재시작합니다.

Exit Status

docker run의 종료 코드(exit code)는 컨테이너가 실행되지 못한 이유 또는 종료된 이유에 대한 정보를 제공합니다.
docker run이 0이 아닌(non-zero) 코드로 종료되면 종료 코드는 chroot 표준을 따릅니다.

Clean up (--rm)

기본적으로 컨테이너의 파일 시스템은 컨테이너가 종료된 후에도 유지됩니다. 즉, 기본적으로 모든 데이터가 유지됩니다.
그러나 단기 포그라운드(foreground) 프로세스를 실행하는 경우 이러한 컨테이너 파일 시스템이 실제로 살일 수 있습니다.
대신 Docker가 컨테이너를 자동으로 정리하고 컨테이너가 종료(exit)될 때 파일 시스템을 제거하도록 하려면 --rm 플래그를 추가할 수 있습니다.

--rm : 컨테이너가 종료되면 자동으로 컨테이너 제거

--rm 플래그를 설정하면 Docker는 컨테이너가 제거될 때 컨테이너와 연결된 익명 볼륨도 제거합니다. 이는 docker rm -v my-container를 실행하는 것과 유사합니다.
이름 없이 지정된 볼륨만 제거됩니다. 예를 들어, 다음과 같이 실행하면
$ docker run --rm -v /foo -v awesome:/bar busybox top
/foo의 볼륨은 제거되지만 /bar의 볼륨은 제거되지 않습니다. --volumes-from을 통해 상속된 볼륨은 동일한 로직으로 제거됩니다.
원래 볼륨이 이름으로 지정된 경우에는 제거되지 않습니다.

Overriding Dockerfile image defaults

도커 파일에서 이미지를 빌드하거나 커밋할 때 이미지가 컨테이너로 시작될 때 적용되는 여러 기본 매개변수를 설정할 수 있습니다.

Dockerfile 명령 중 4개(FROM, MAINTAINER, RUN, ADD)는 런타임 시 재정의될 수 없습니다. 다른 모든 것에는 docker run에 해당 재정의가 있습니다.

- CMD (default command or options)
  위에 있는 docker run 명령의 커맨드라인을 다시 한 번 보면
  `$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]`
  IMAGE를 만든 사람이 이미 Dockerfile CMD 명령을 사용하여 기본 COMMAND를 제공했을 수 있으므로 이 명령은 선택 사항입니다.
  운영자(이미지에서 컨테이너를 실행하는 사람)로서 새로운 COMMAND 를 지정하기만 하면 Dockerfile에 있는 CMD 명령을 재정의할 수 있습니다.

이미지가 ENTRYPOINT도 지정하면 CMD 또는 COMMAND가 ENTRYPOINT에 인수로 추가(append)됩니다.

예시1) $ sudo docker run -it example1 "Hello\tWorld\tof Docker Run"

- ENTRYPOINT (default command to execute at runtime)
  이미지의 ENTRYPOINT는 컨테이너가 시작될 때 실행할 것을 지정한다는 점에서 COMMAND와 유사하지만 (의도적으로) 재정의하기가 더 어렵습니다.
  ENTRYPOINT는 컨테이너에 기본 특성이나 동작을 제공하므로 ENTRYPOINT를 설정할 때 컨테이너를 마치 바이너리인 것처럼 실행하고 기본 옵션을 사용하여 완료할 수 있으며 COMMAND를 통해 더 많은 옵션을 전달할 수 있습니다.
  그러나 때로는 운영자가 컨테이너 내에서 다른 것을 실행하려고 할 수도 있으므로 문자열을 사용하여 새로운 ENTRYPOINT를 지정함으로써 실행시 기본 ENTRYPOINT를 재정의할 수 있습니다.

  $ docker run -it --entrypoint /bin/bash example/redis
  $ docker run -it --entrypoint /bin/bash example/redis -c ls -l
  $ docker run -it --entrypoint /usr/bin/redis-cli example/redis --help
  $ docker run -it --entrypoint="" mysql bash

  --entrypoint를 전달하면 이미지에 설정된 모든 기본 명령(즉, 이미지를 빌드하는 데 사용된 Dockerfile의 모든 CMD 명령)이 지워집니다.

- EXPOSE (incoming ports)
  --expose=[] : 컨테이너 내부의 포트 또는 포트 범위를 노출합니다. 이것들은 Dockerfile의 'EXPOSE' 명령에 의해 노출된 것 외에 추가됩니다.
  -P(대문자) : 노출된 모든 포트를 호스트 인터페이스에 공개합니다.
  -p=[] : 컨테이너의 포트 또는 포트 범위를 호스트에 공개.
  format: ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort | containerPort hostPort와 containerPort는 모두 포트 범위로 지정할 수 있습니다.
  hostPort, containerPort 두 가지 모두에 대해 범위를 지정할 때 범위의 containerPort 수는 범위의 hostPort 수와 일치해야 합니다.
  예시1) -p 1234-1236:1234-1236/tcp

  hostPort에만 범위를 지정할 때 containerPort는 범위가 아니어야 합니다. 이 경우 containerPort는 지정된 hostPort 범위 내의 어딘가에 공개됩니다.
  예시2) -p 1234-1236:1234/tcp

  실제 매핑을 보려면 `docker port`를 사용하세요
  --link="" : 다른 컨테이너에 link 추가 (<name or id>:alias or <name or id>)

  EXPOSE 명령을 제외하고 이미지 개발자는 네트워킹을 많이 제어할 수 없습니다.
  EXPOSE 명령은 서비스를 제공하는 초기 수신 포트를 정의합니다. 이러한 포트는 컨테이너 내부의 프로세스에 사용할 수 있습니다.
  운영자는 --expose 옵션을 사용하여 노출된 포트에 추가할 수 있습니다.

  컨테이너의 내부 포트를 노출하기 위해 운영자는 -P 또는 -p 플래그를 사용하여 컨테이너를 시작할 수 있습니다.
  노출된 포트는 호스트(host)에서 엑세스할 수 있으며 해당 포트는 호스트에 연결할 수 있는 모든 클라이언트에서 사용할 수 있습니다.

  -P(대문자) 옵션은 모든 포트를 호스트 인터페이스에 게시합니다. Docker는 노출된 각 포트를 호스트의 임의 포트에 바인딩합니다.
  포트 범위는 `/proc/sys/net/ipv4/ip_local_port_range`에 정의된 임시 포트 범위 내에 있습니다.
  단일 포트 또는 포트 범위를 명시적으로 매핑하려면 -p(소문자) 플래그를 사용합니다.

  컨테이너 내부(서비스가 수신 대기하는 곳)의 포트 번호는 컨테이너 외부(클라이언트가 연결되는 곳)에 노출된 포트 번호와 일치할 필요가 없습니다.
  예를 들어 컨테이너 내부에서 HTTP 서비스는 포트 80에서 수신 대기합니다. 따라서 이미지 개발자는 Dockerfile에 EXPOSE 80을 지정합니다.
  런타임 시 포트 80은 호스트의 42800에 바인딩될 수 있습니다.
  호스트 포트와 노출된 포트 간의 매핑을 찾으려면 `docker port`를 사용하세요.

  운영자가 기본 브리지 네트워크에서 새 클라이언트 컨테이너를 시작할 때 만약 --link를 사용하는 경우 클라이언트 컨테이너는 개인 네트워킹 인터페이스를 통해 노출된 포트에 액세스할 수 있습니다.
  사용자 정의 네트워크에서 컨테이너를 시작할 때 --link를 사용하면 연결되는 컨테이너에 대해 명명된 별칭이 제공됩니다.