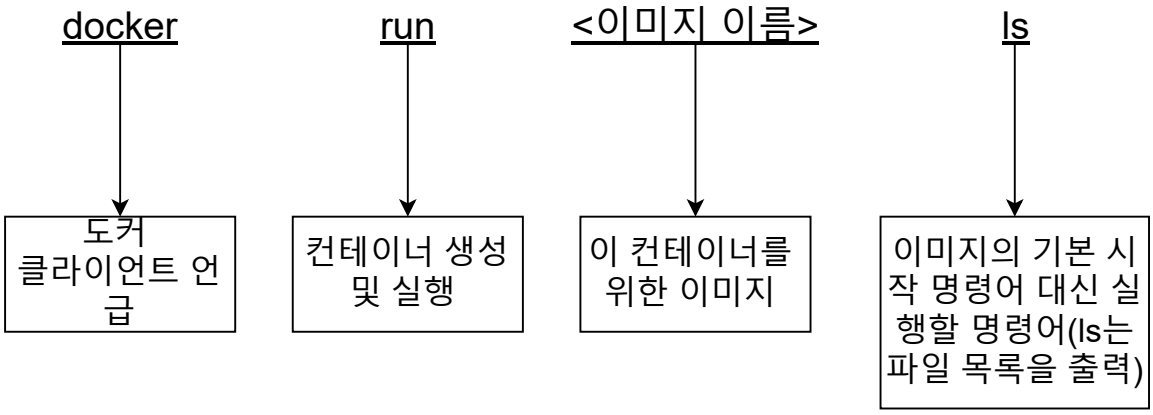
# 기본적인 도커 클라이언트 명령어 알아보기

# 도커 이미지 복습




***ex) docker run hello-world***

## 작동 순서 복습

이미지의 기본 시작 명령어
출력)

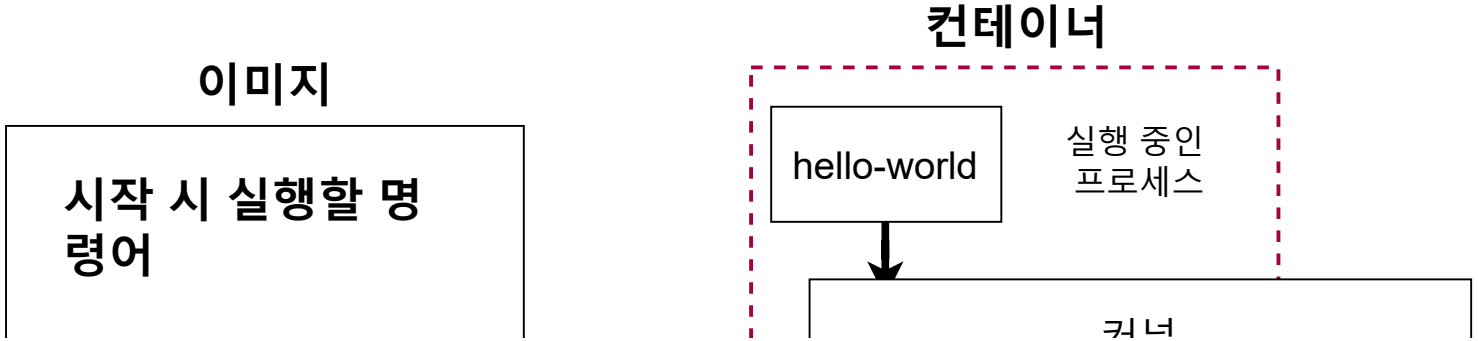
1. 도커 클라이언트에 명령어 입력후 도커 서버로 보냄

2. 도커 서버에서 컨테이너를 위한 이미지가 이미 캐쉬가 되어 있는지 확인

3. 없으면 도커 허브에서 다운 받아옴 있다면 그 이미 가지고 있는 이미지로 컨테이너 생성

## 이미지로 컨테이너 생성하는 순서 복습 (아래 도표 참조)

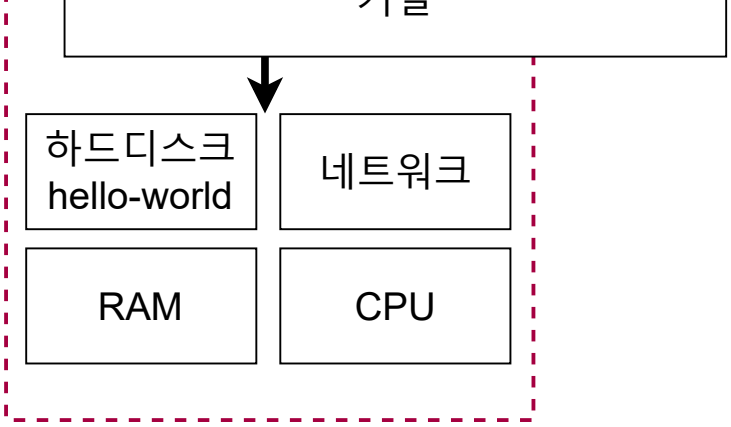
1. 먼저 파일 스냅숏 돼있는 것을 컨테이너의 하드 디스크 부분에 올린다.



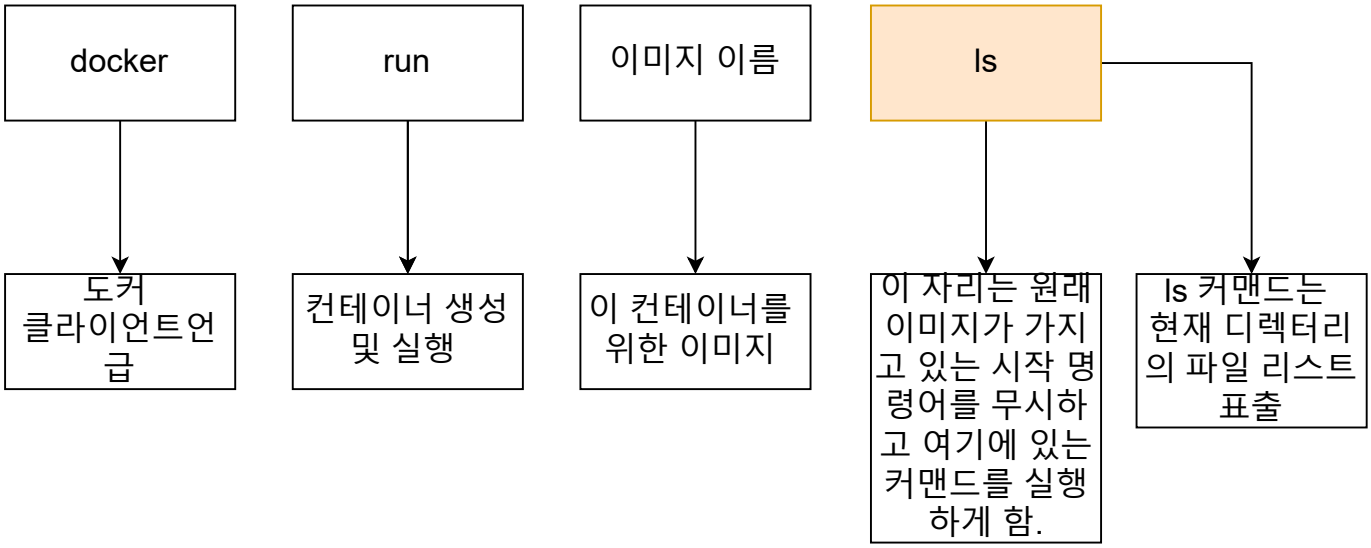
대신 실행할 명령어(ls는 파일 목록을

**파일 스냅숏**

hello-world



# 이미지 내부 파일 시스템 구조 보기




## *ex) docker run alpine ls*

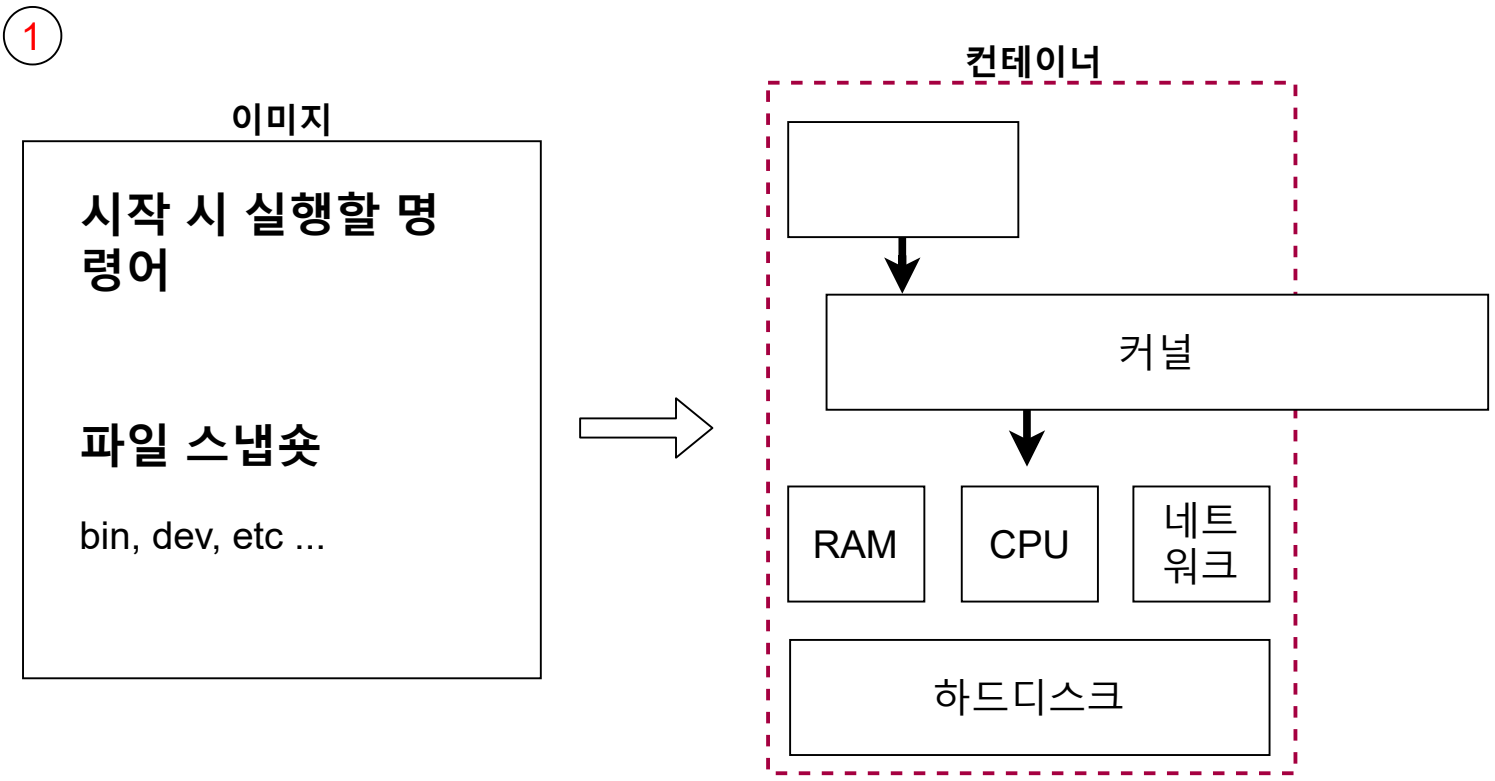
```
jaewon@Jaewonui-MacBookPro ~ % docker run alpine ls
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
df20fa9351a1: Pull complete
Digest: sha256:185518070891758909c9f839cf4ca393ee977ac378609f700f60a771a2dfe321
Status: Downloaded newer image for alpine:latest
bin
dev
etc
home
lib
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

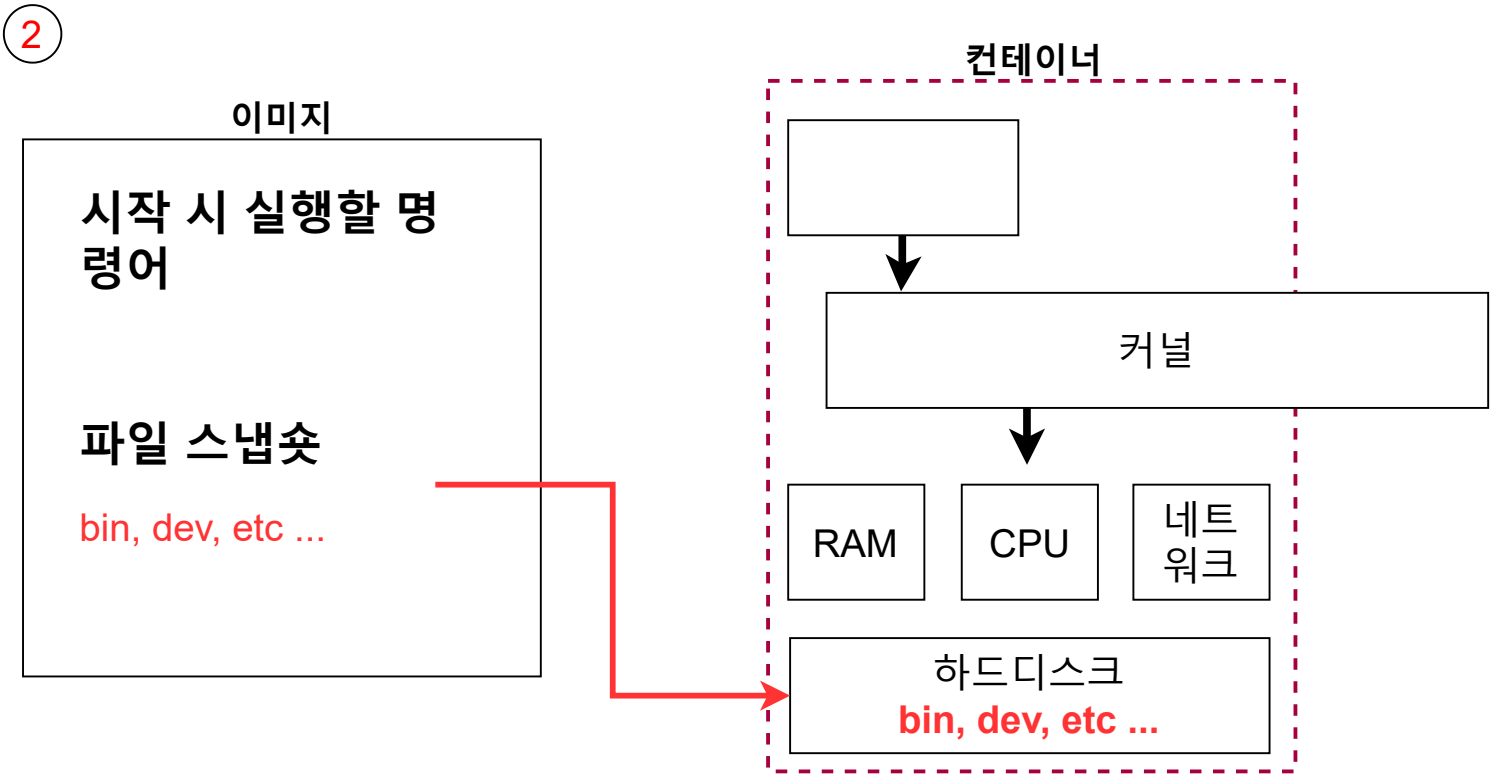
## 설명 (아래 도표 참조)

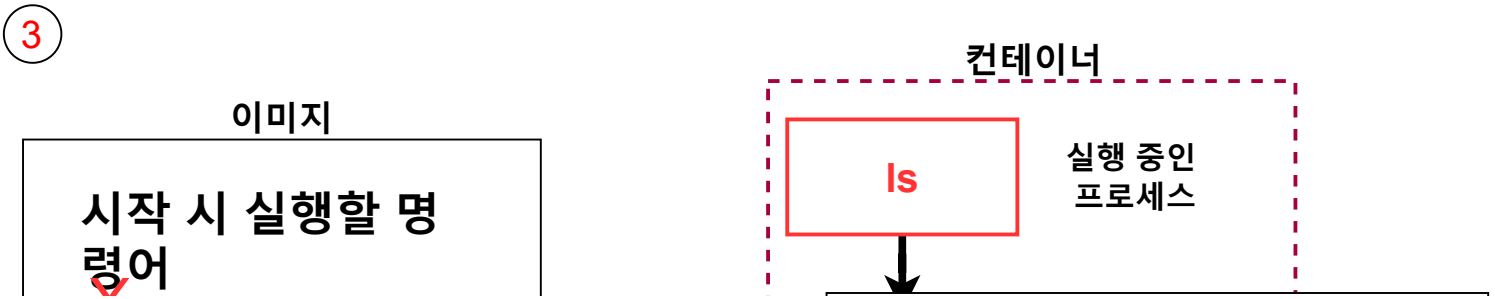
1. Alpine 이미지를 이용해서 컨테이너를 생성.

2. 생성할 때 Alpine 이미지 안에 들어있던 파일 스냅샷들 (bin, dev, etc 등등..)이
   컨테이너 안에 있는 하드 디스크로 다운로드됨

3. 이미지 이름 뒤에 다른 명령어를 더 붙여서 원래 이미지 안에 들어있는 기본 커맨드는 무시가 되고 ls 명령어가 실행됨


1
이미지
시작 시 실행할 명령어
파일 스냅숏
bin, dev, etc ...
컨테이너
커널
RAM
CPU
네트워크
하드디스크

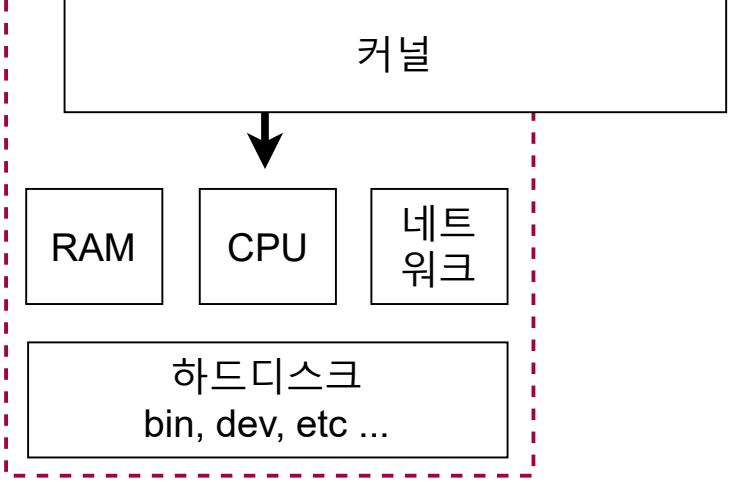

2
이미지
시작 시 실행할 명령어
파일 스냅숏
bin, dev, etc ...
컨테이너
커널
RAM
CPU
네트워크
하드디스크
bin, dev, etc ...


3
이미지
시작 시 실행할 명령어
컨테이너
ls
실행 중인 프로세스

X

**파일 스냅숏**

bin, dev, etc ...




## 하지만 어떻게 Alpine 이미지를 이용해 ls 명령어를 실행 가능?

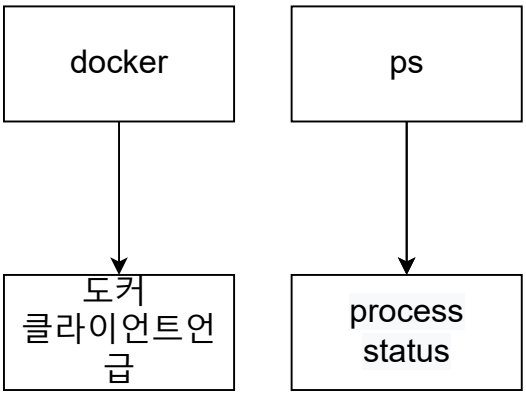
## hello-world 이미지로는 ls 명령어 사용 불가능

1. docker run hello-world ls 하면  아래와 같은 오류가 뜸.
   설명을 보면 excutable file not found. (실행할 수 있는 파일을 못 찾음)

```
jaewon@Jaewonui-MacBookPro ~ % docker run hello-world ls
docker: Error response from daemon: OCI runtime create failed: container_linux.go:349: startin
g container process caused "exec: \"ls\": executable file not found in $PATH": unknown.
```

# 컨테이너들 나열하기

## 현재 실행중인 컨테이너 나열




## 실습

1. 2개의 Terminal을 작동시킵니다.

2. 첫 번째 Terminal에서 container 하나를 실행
   (하지만 이때 컨테이너를 바로 켰다가 바로 끄면
   3번을 할 때 이미 프로세스가 꺼져있기 때문에 리스트에서 볼 수 없다).

3. 그리고 두 번째 Terminal에서  docker ps 로 확인.

4. 그러면 꺼져있는 container도 확인하고 싶다면?

```
jaewon@Jaewonui-MacBookPro ~ % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
459ab08c1f57   alpine    "ping google.com"        18 seconds ago   Up 17 seconds             sad_mccarthy
```

**이미지 설명**

1. CONTAINER ID : 컨테이너의 고유한 아이디 해쉬값.

실제로는 더욱 길지만 일부분만 표출.

2. IMAGE : 컨테이너 생성 시 사용한 도커 이미지.

3. COMMAND : 컨테이너 시작 시 실행될 명령어.

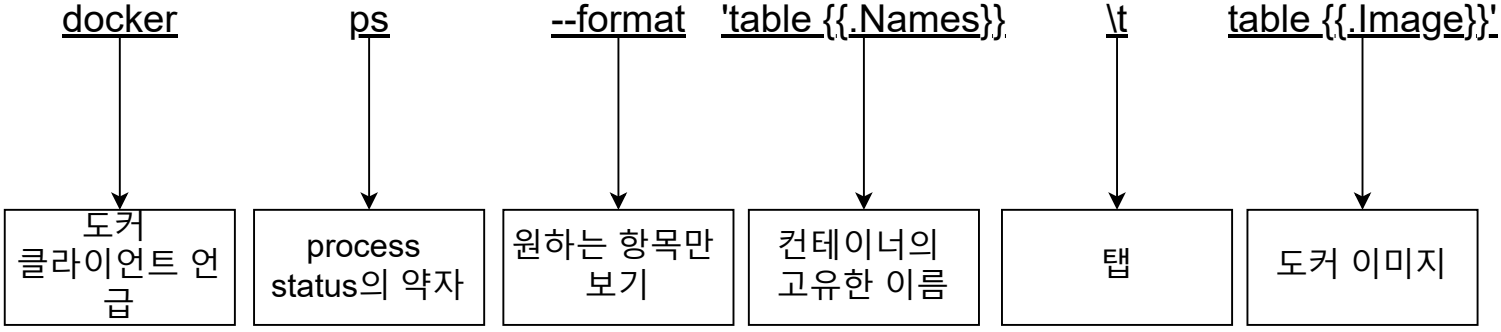
대부분 이미지에 내장되어 있으므로 별도 설정이 필요 X.

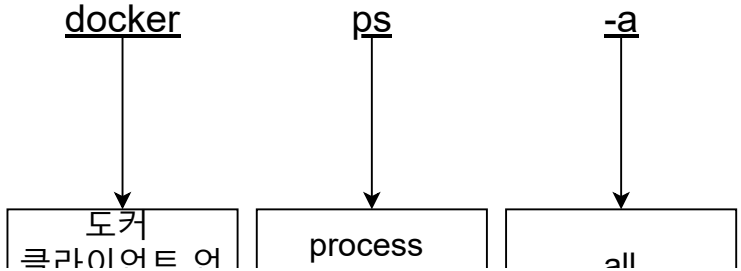
4. CREATED : 컨테이너가 생성된 시간.

5. STATUS : 컨테이너의 상태입니다.

실행 중은 Up, 종료는 Exited, 일시정지 Pause.

6. PORTS : 컨테이너가 개방한 포트와 호스트에 연결한 포트.


docker
ps
--format
'table {{.Names}}
\t
table {{.Image}}'
도커 클라이언트 언급
process status의 약자
원하는 항목만 보기
컨테이너의 고유한 이름
탭
도커 이미지


docker
ps
-a
도커 클라이언트 언
process
all

특별한 설정을 하지 않은 경우 출력되지 않습니다.

뒤에 가서 더 자세히 설명합니다.

7. NAMES : 컨테이너 고유한 이름.

컨테이너 생성 시 --name 옵션으로 이름을 설정하지 않으면

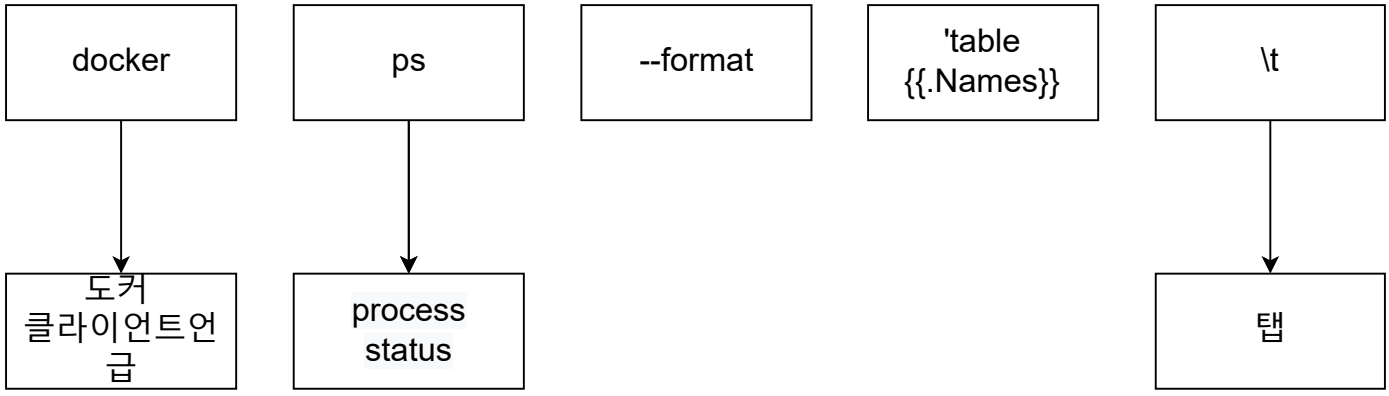
도커 엔진이 임의로 형용사와 명사를 조합해 설정.

id와 마찬가지로 중복이 안되고 docker rename명령어로

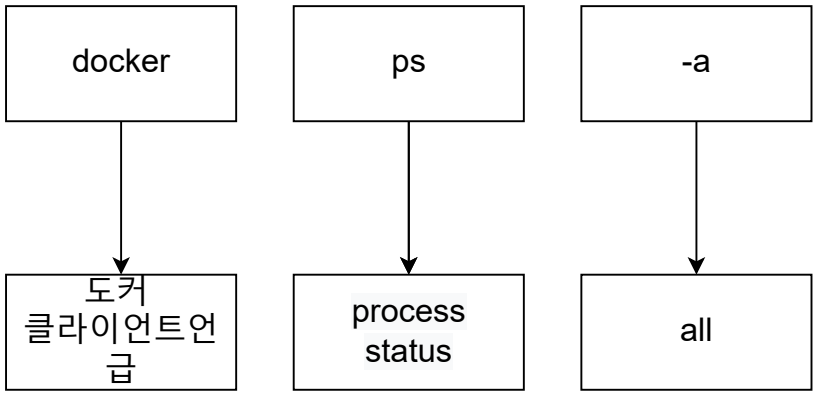
이름을 변경할 수 있습니다.

docker rename original-name changed-name

## 원하는 항목만 보기




## 모든 컨테이너 나열

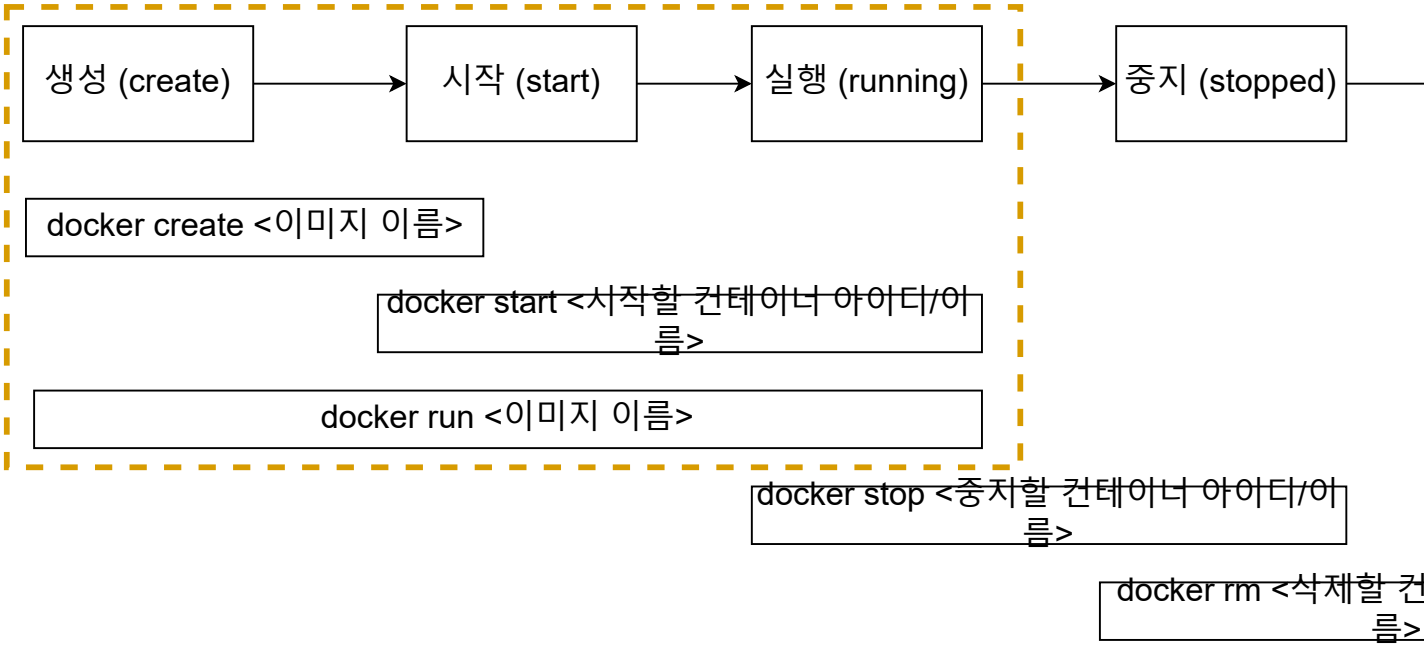


| 클라이언트 단 급 | status의 약자 | all |
| --- | --- | --- |

table {{.Image}}'

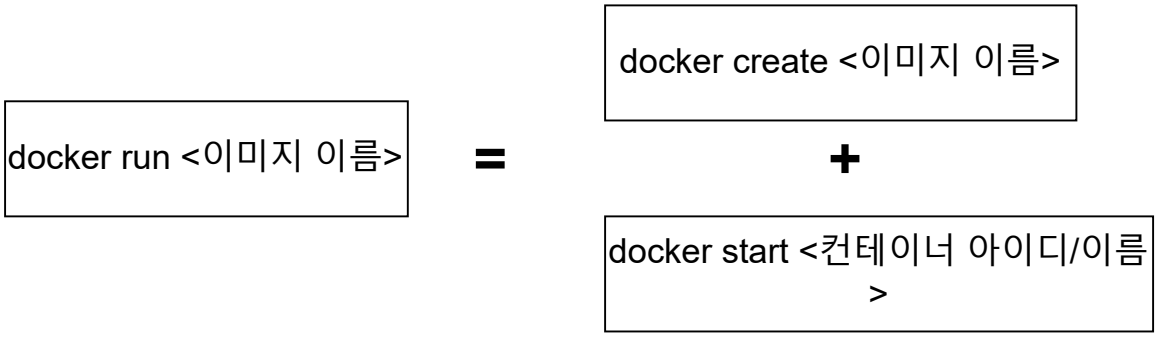
# 도커 컨테이너의 생명주기

## 생명주기

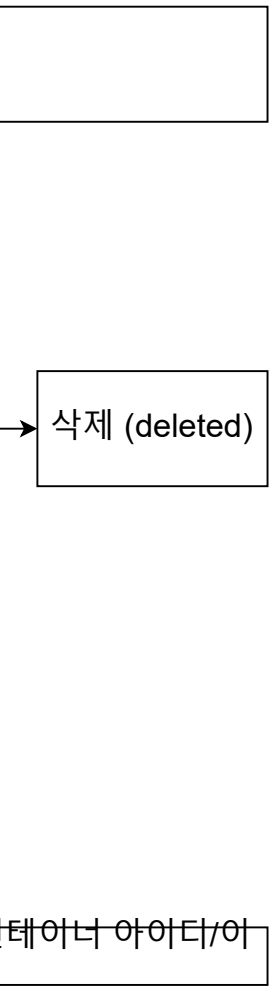


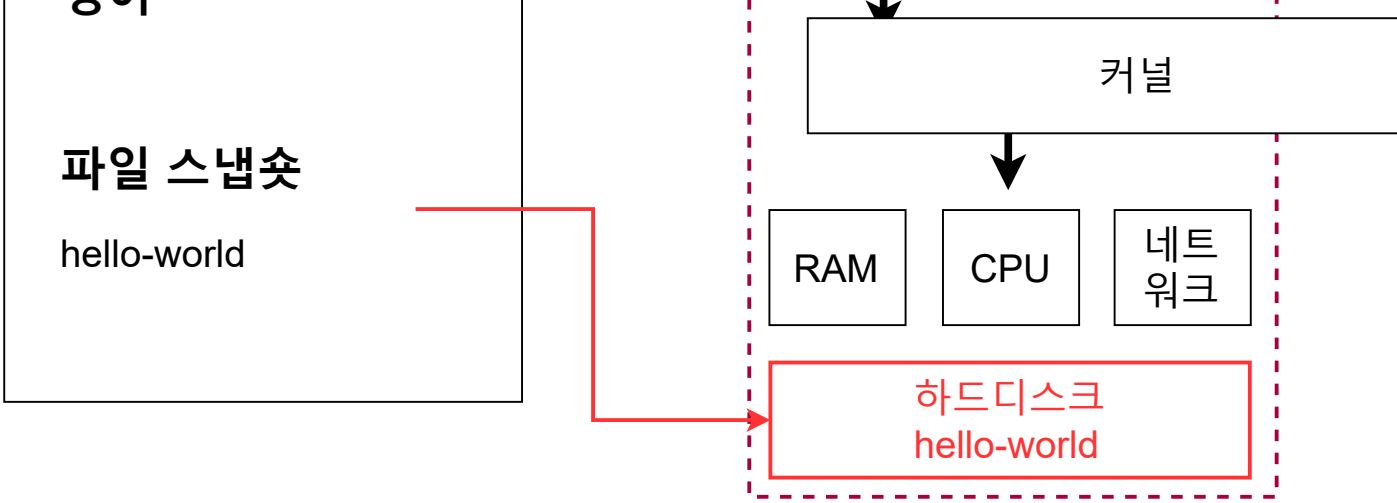

## 하이라이트 된 부분에 대한 자세한 설명

***지금까지 docker run <이미지 이름>으로 컨테이너 생성 실행했는데 이걸 docker create과 docker run으로  쪼개서 봐보겠습니다.***

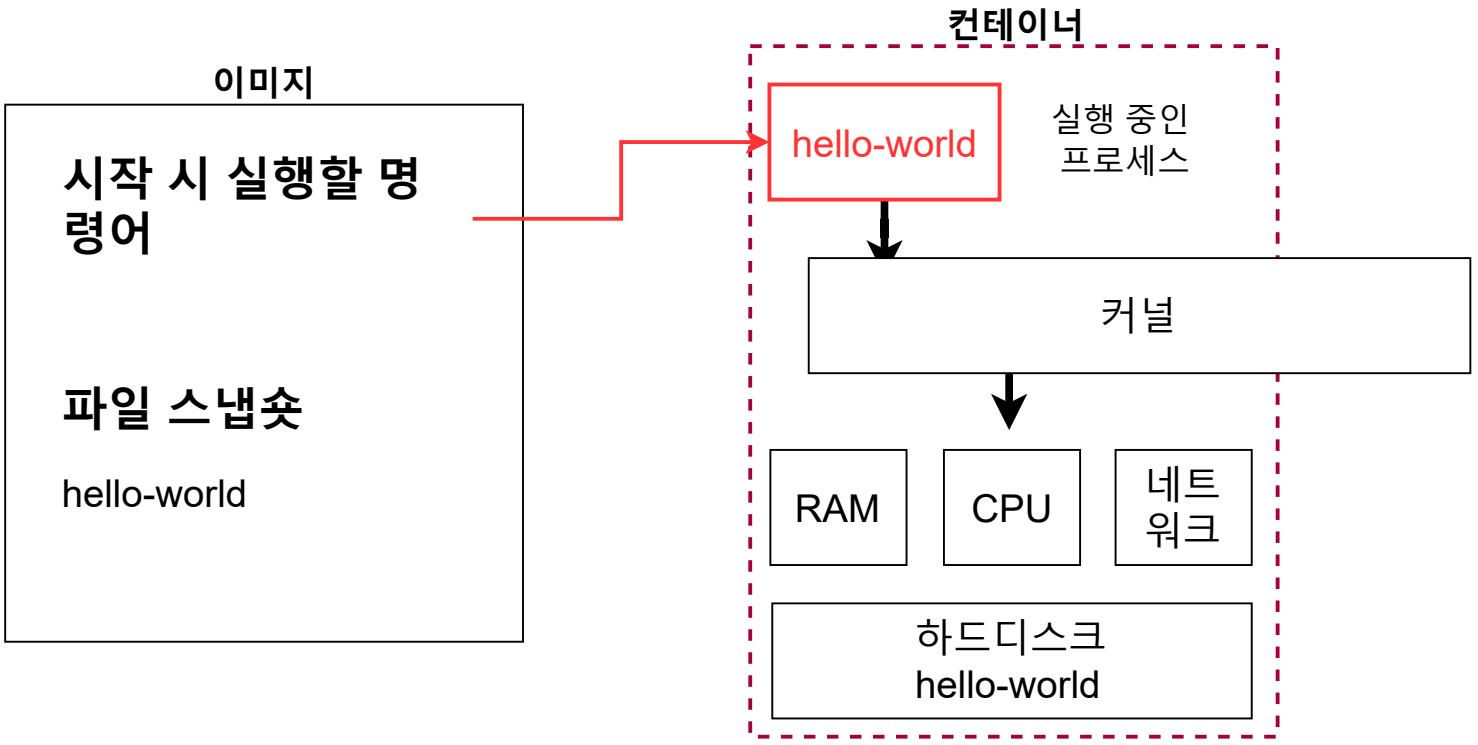


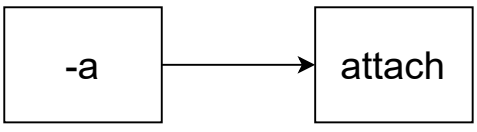

***docker create <이미지 이름>***

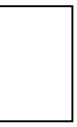

삭제 (deleted)
테이너 아이디/아




***docker start <컨테이너 아이디/이름>***



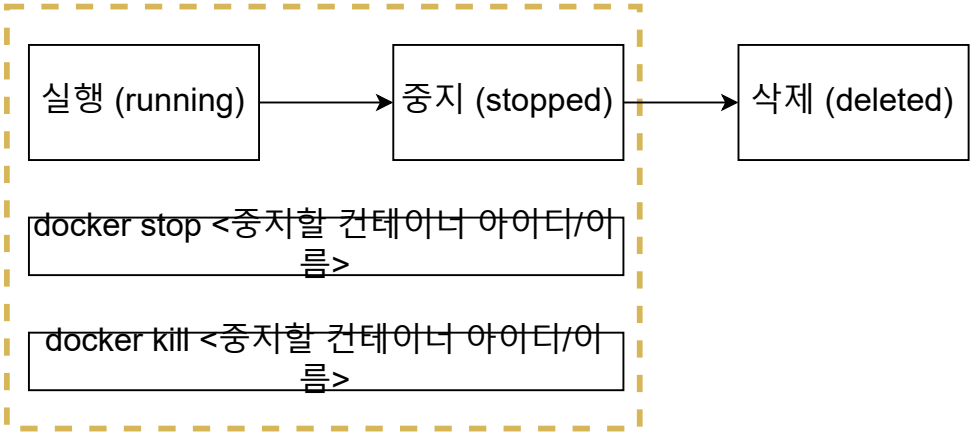





# docker stop vs docker kill

***도커의 생명주기 중에서 중지 부분***

***docker stop과 docker kill로 중지할 수 있습니다.***




## Stop과 Kill 은 어떤 차이가 있을까?

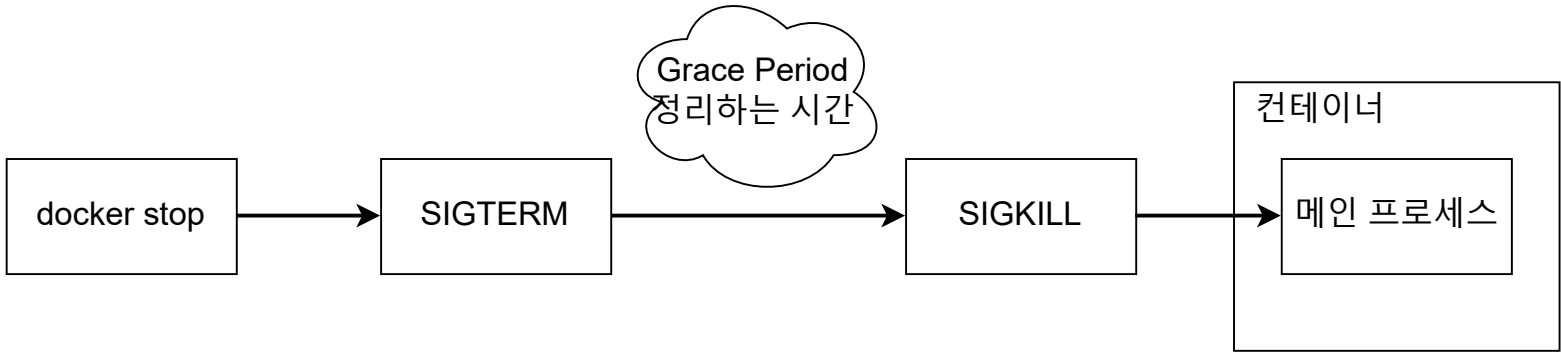
공통점은 둘 다 실행중인 컨테이너를 중지시킵니다.

하지만

1. Stop은 Gracefully 하게 중지를 시킵니다.
   자비롭게 그동안 하던 작업들을 ( 메시지를 보내고 있었다면 보내고 있던 메시지 ) 완료하고 컨테이너를 중지시킨다.

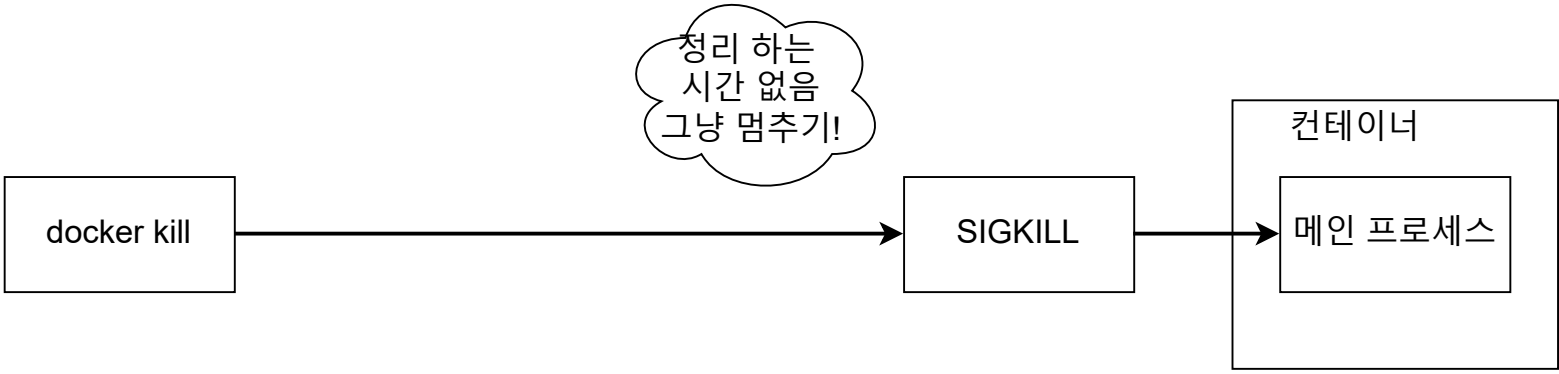
2. Kill 같은 경우는 Stop과 달리 어떠한 것도 기다리지 않고 바로 컨테이너를 중지시킨다.

***docker stop <컨테이너 아이디/이름>***




***docker kill <컨테이너 아이디/이름>***

***docker kill <컨테이너 아이디/이름>***



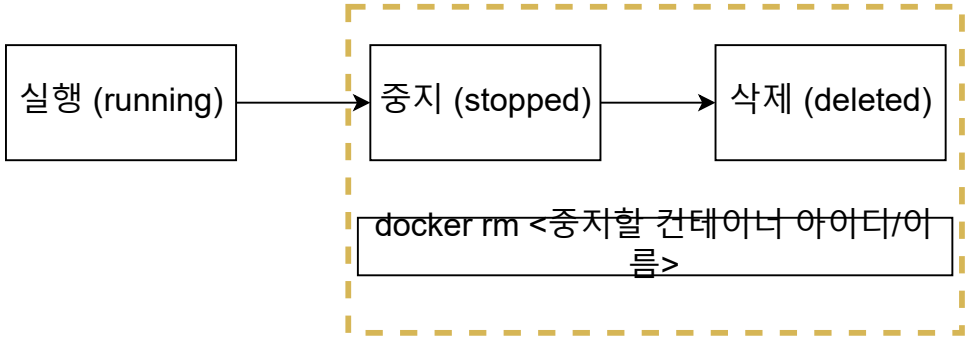

## 실제로 stop과 kill 체험해보기

1. docker run alpine ping google.com

2. 이걸 stop으로 한번 멈춰보고  kill로 한번 멈춰보자

# 도커 컨테이너 삭제하기

## *도커의 생명주기중에서 삭제 부분*




### 중지된 컨테이너를 삭제하고 싶다면?

***docker rm <아이디/ 이름>***

-실행 중인 컨테이너는 먼저 중지한 후에 삭제 가능.

### 모든 컨테이너를 삭제하고 싶다면?

***docker rm `docker ps -a -q`***

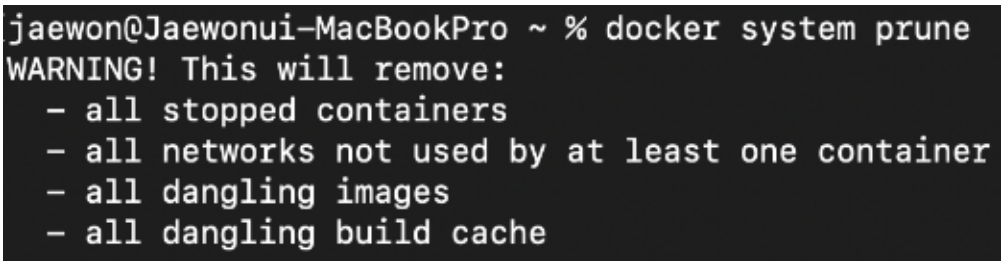
### 이미지를 삭제하고 싶다면 ?

***docker rmi <이미지 id>***

### 한 번에 사용하지 않는 컨테이너, 이미지, 네트워크 모두 삭제하고 싶다면?

***docker system prune***

- 도커를 쓰지 않을 때 모두 정리하고 싶을 때 사용해주면 좋음


jaewon@Jaewonui-MacBookPro ~ % docker system prune
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- all dangling build cache

# 실행 중인 컨테이너에 명령어 전달

## 이미 실행 중인 컨테이너에 명령어를 전달하고 싶다면 ?

## *docker exec <컨테이너 아이디>*

1. 먼저 터미널 2개를 실행합니다.

2. 첫 번째 터미널에서 컨테이너 하나를 실행합니다.
   (docker run alpine ping localhost)

3. 두 번째 터미널에서 컨테이너가 잘 작동하고 있는지 확인하고 다른 명령어를 전달합니다.

*똑같은 결과를 내주는 것!* *docker run* *<이미지 이름>*

## docker run vs docker exec

1. docker run 은 새로 컨테이너를 만들어서 실행

2. docker exec 은 이미 실행 중인 컨테이너에 명령어를 전달

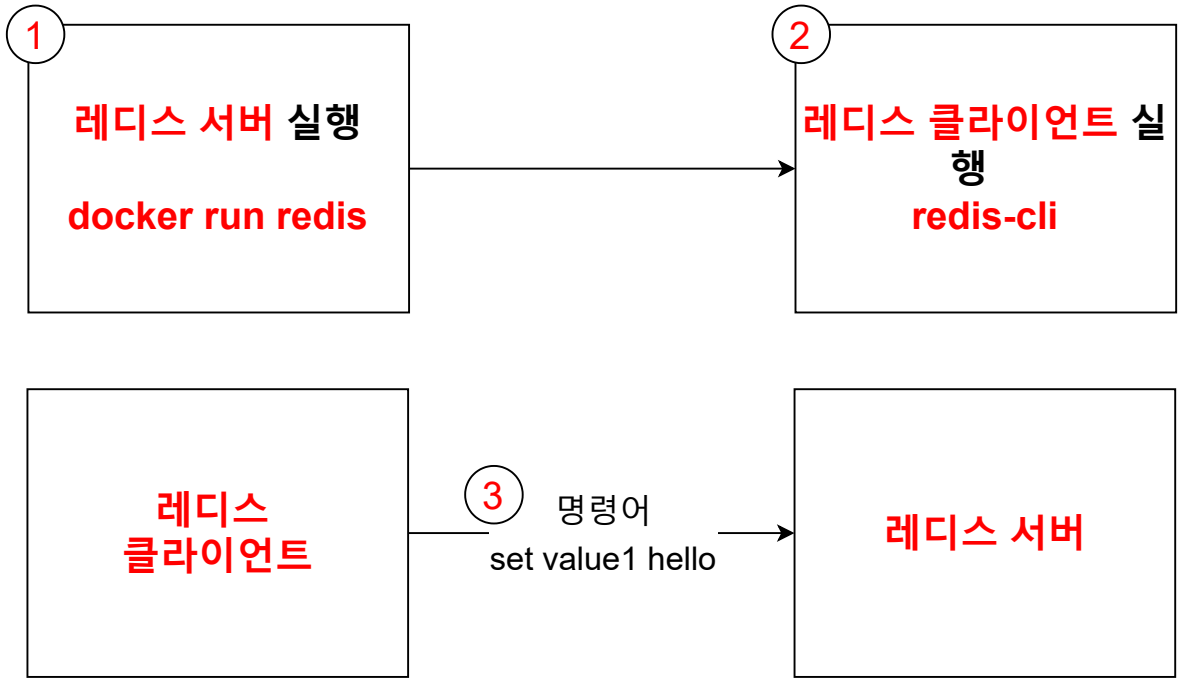
# 레디스를 이용한 컨테이너 이해

**레디스를 도커 환경에서 실행을 해서 컨테이너를 더욱**

**이해해 보는 시간을 갖겠습니다.**

**그러기 위해서 레디스를 실행을 해야겠는데요.**

**그전에 레디스를 어떻게 이용하며 작동하는지 알아보겠습니다.**




**위에 도표와 같이 먼저 레디스 서버를 실행한 후, 레디스 클라이언트를 통해서 서버에 명령어를 전달해 줘야 합니다.**

1. 먼저 첫 번째 터미널을 실행 후, 레디스 서버를 작동시키자

docker run redis

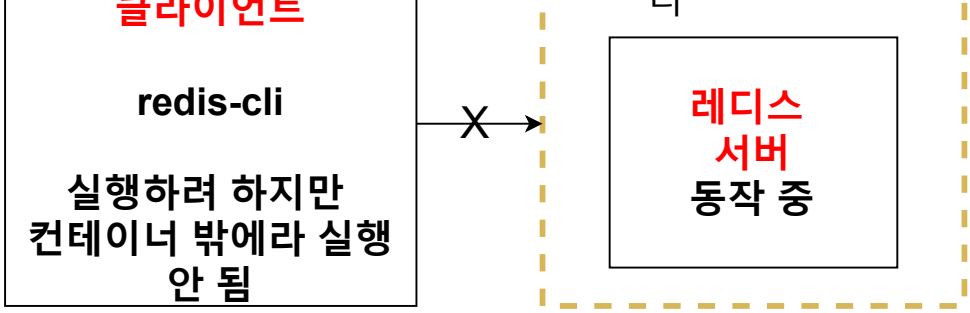
2. 그 후 레디스 클라이언트를 켜야 하는데 첫 번째 터미널에서는 아무것도 할 수 없다.

그러니 두 번째 터미널을 켜서 레디스 클라이언트를 작동시킨다.

redis-cli

3. 하지만 에러가 났다..... 무엇이 잘못된 것일까요???

## 현재 레디스 클라이언트와 서버




**레디스 클라이언트가 레디스 서버가 있는 컨테이너 밖에서**
**실행을 하려 하니 레디스 서버에 접근을 할 수가 없기**

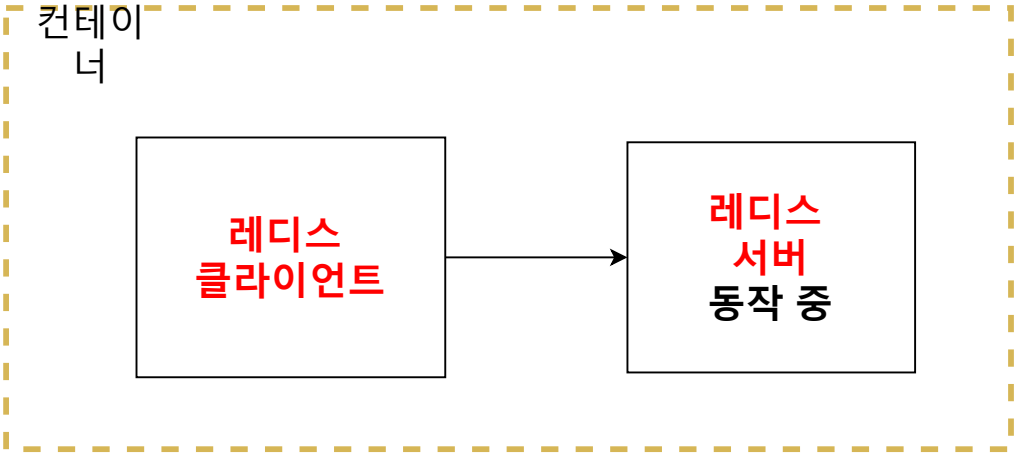
**그러면 어떻게 해야 할까요 ...?**

*답은 레디스 클라이언트도 컨테이너 안에서 실행을 시켜야 합니다.*

1. 먼저 이전과 똑같이 첫 번째 터미널을 킨 후, 레디스 서버를 작동시키자

```
docker run redis
```

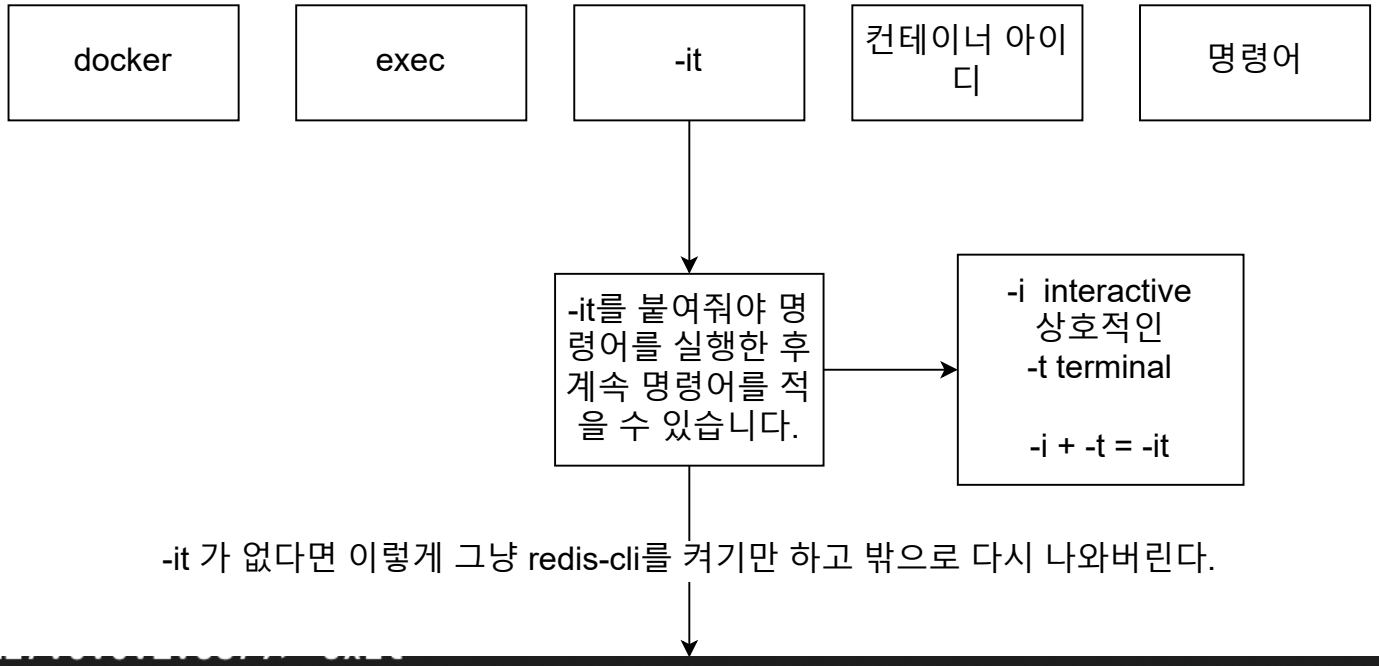
2. 이제는 이전 시간에 배운 exec을 써먹을 차례입니다.
   이미 실행 중인 컨테이너에 명령어를 전달할 때 exec을 쓴다고 배웠습니다.
   그러니 redis 서버가 실행 중인 컨테이너에 exec을 이용하여 redis 클라이언트도




첫번째 터미널

```
jaewon@Jaewonui-MacBookPro ~ % docker run redis
1:C 02 Jun 2020 10:55:51.648 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 02 Jun 2020 10:55:51.648 # Redis version=6.0.4, bits=64, commit=00000000, mo
dified=0, pid=1, just started
1:C 02 Jun 2020 10:55:51.648 # Warning: no config file specified, using the defa
ult config. In order to specify a config file use redis-server /path/to/redis.co
nf
1:M 02 Jun 2020 10:55:51.649 * Running mode=standalone, port=6379.
1:M 02 Jun 2020 10:55:51.649 # WARNING: The TCP backlog setting of 511 cannot be
 enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
1:M 02 Jun 2020 10:55:51.649 # Server initialized
1:M 02 Jun 2020 10:55:51.649 # WARNING you have Transparent Huge Pages (THP) sup
```

```
port enabled in your kernel. This will create latency and memory usage issues wi
th Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transpa
rent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to ret
ain the setting after a reboot. Redis must be restarted after THP is disabled.
1:M 02 Jun 2020 10:55:51.650 * Ready to accept connections
```




-it 가 없다면 이렇게 그냥 redis-cli를 켜기만 하고 밖으로 다시 나와버린다.

```
jaewon@Jaewonui-MacBookPro ~ % docker exec 2f3118e8ba88 redis-cli
jaewon@Jaewonui-MacBookPro ~ %
```

두번째 터미널

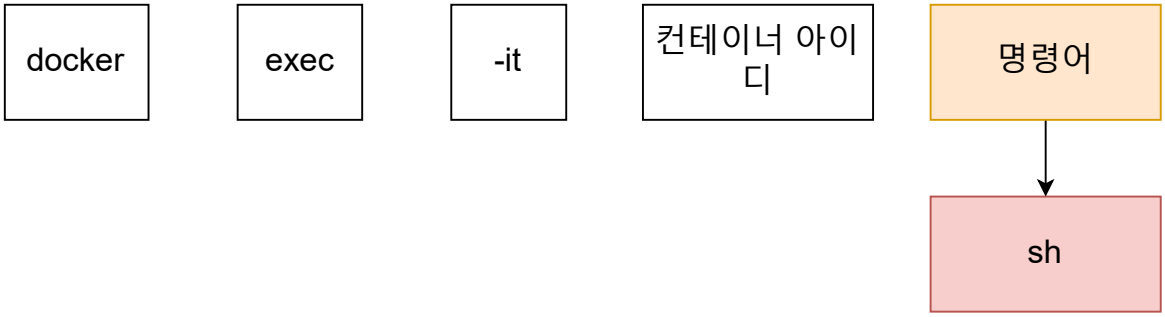
```
jaewon@Jaewonui-MacBookPro ~ % docker ps
CONTAINER ID        IMAGE               COMMAND
2f3118e8ba88        redis               "docker-entrypoint.s…"
b998e78244fc        mongo               "docker-entrypoint.s…"
jaewon@Jaewonui-MacBookPro ~ % docker exec -it 2f3118e8ba88 red
127.0.0.1:6379> set value1 hello
OK
127.0.0.1:6379> get value1
"hello"
```

# 실행 중인 컨테이너에서 터미널 생활 즐기기

## 지금까지 실행 중인 컨테이너에 명령어를 전달할 때에는


**이런 식으로 명령어 하나 입력해야 할 때마다**
**이 모든 것을 계속 입력해줬어야 하는 데 요.**
**이러한 문제점을 해결해주기 위해 컨테이너 안에**
**쉘이나 터미널 환경으로 접속을 해줄 수가 있습니다.**
**그 방법은 마지막 명령어를 sh로 주시면 됩니다.**

## 실행 중인 컨테이너에 쉘 환경으로 접속하기




## 컨테이너에 쉘 환경으로 접근해 보기

1. 먼저 첫 번째 터미널을 실행한 후, alpine 이미지를 이용해서 컨테이너를 실행합니다.

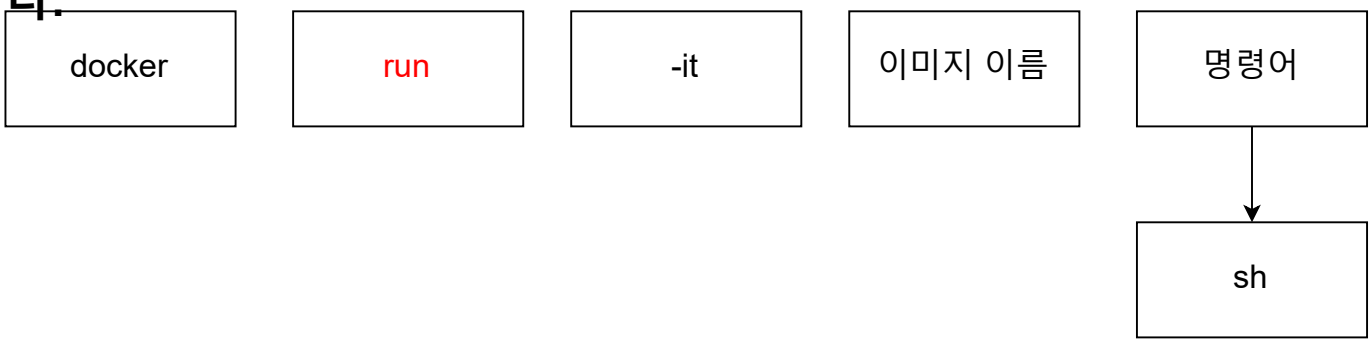
```
docker run alpine ping localhost
```

2. 그 후 exec을 이용하고 마지막 명령어 부분에 sh를 입력후 컨테이너 안에서 터미널 환경을 구축

```
docker exec -it  컨테이너 아이디 sh
```

3. 그 안에서 여러 가지 터미널에서 원래 할 수 있는 작동들을 해봅니다.
   ex)
   - ls  -컨테이너 디렉토리에 있는 내용(디렉토리, 파일) 확인
   - touch new-file - 파일 생성
   - export hello=hi  echo $hello  -변수 생성 출력

**exec 대신에 run을 하는 것도 가능합니다.**




**이 터미널 환경에서 나오려면 Control + D**