2026/04/08~15

S-개발자 4기

박창렴 / myanjini@gmail.com / 010-2982-7033

내용 공유 >>>

http://bit.ly/4cbwMse

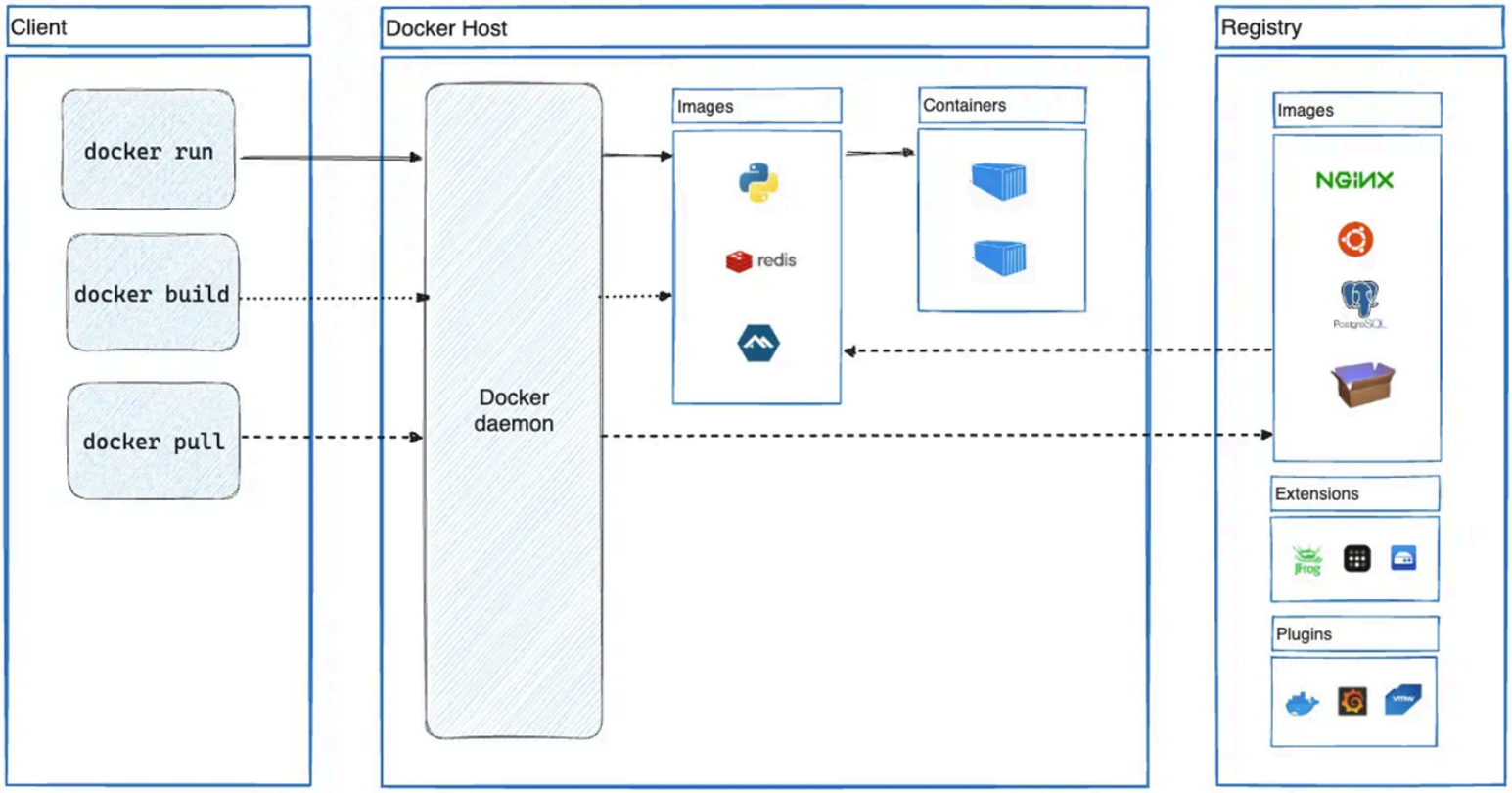
붉은색만 숫자입니다. 대소문자를 구분해서 입력해 주세요.
수업 종료 후 링크가 삭제될 수 있으니, 필요한 내용은 미리 백업해 두세요.

참고

https://pyrasis.com/jHLsAlwaysUpToDateDocker

도커 구성 요소

https://docs.docker.com/get-started/overview/#docker-architecture



도커 객체(Docker objects)

이미지

- 컨테이너를 생성할 때 필요한 요소 (가상 머신을 생성할 때 사용하는 iso 파일과 비슷한 개념)
- 여러 개의 계층으로 된 바이너리 파일로 존재
- 컨테이너를 생성하고 실행할 때 읽기 전용으로 사용됨
- 저장소(registry)/리포지토리(repository):태그(tag)

컨테이너

- 도커 이미지로 생성한 해당 이미지의 목적에 맞는 파일이 들어 있는 파일 시스템과 격리된 시스템 자원 및 네트워크를 사용할 수 있는 독립된 공간
- 컨테이너는 이미지를 읽기 전용으로 사용하되 이미지에서 변경된 사항만 컨테이너 계층에 저장하므로 컨테이너에서 무엇을 하든지 원래 이미지는 영향을 받지 않음
- 생성된 컨테이너는 각기 독립된 파일 시스템을 제공받으며 호스트와 분리돼 있으므로 특정 컨테이너에서 어떤 애플리케이션을 설치하거나 삭제해도 다른 컨테이너와 호스트는 변화가 없음

도커 CLI 커맨드(Docker CLI Command)

- https://docs.docker.com/engine/reference/commandline/docker/
- 도커를 조작하기 위한 인터페이스
- 서브 커맨드 ⇒ 구체적인 작업을 지정
- 사용법 ⇒ docker COMMAND --help

이미지 관리 명령

```
C:\Users\myanj> docker image --help
```

명령	의미
docker image build docker build	Dockerfile로 이미지 빌드
docker image history docker history	이미지 생성 기록 보기
docker iamge import docker import	(docker container export 명령으로 가져온) tar 아카이브로 이미지 만들기
docker image inspect	이미지 상세 정보 보기
docker image load docker load	(docker image save로 출력한) tar 아카이브로 이미지 로드
docker image ls docker images	이미지 목록 보기
docker image prune	불필요한 이미지(태그가 붙어 있지 않아 컨테이너로 사용되지 않는 이미지) 삭제
docker image pull docker pull	레지스트리로부터 이미지 가져오기
docker image push docker push	레지스트리로 이미지 전송
docker image rm docker rmi	이미지 삭제
docker image save docker save	(docker image load 명령으로 읽을 수 있는) 이미지 tar 아카이브로 출력
docker image tag docker tag	기존의 이미지에 태그를 붙이기

컨테이너 관리 명령

```
C:\Users\myanj> docker container --help
```

명령	의미
docker container attach docker attach	실행 중인 컨테이너에 표준 입출력 연결
docker container commit docker commit	컨테이너 내부에서 변경된 파일을 베이스로 이미지 만들기
docker container cp docker cp	파일이나 폴더를 컨테이너와 호스트 환경 간에 복사
docker container create docker create	새 컨테이너 만들기
docker container diff docker diff	컨테이너 내부에서 변경된 파일 검사

docker container exec docker exec	실행 중인 컨테이너 내부에서 커맨드 실행
docker container export docker export	컨테이너 파일 시스템을 tar 아카이브 형식으로 꺼내기
docker container inspect	컨테이너의 자세한 정보 보기
docker container kill docker kill	실행 중인 컨테이너(도커가 만든 PID 1 프로세스)에 신호 보내기
docker container logs docker logs	컨테이너 로그 취득
docker container ls docker ps	컨테이너 목록 보기
docker container pause docker pause	컨테이너로 실행 중인 모든 프로세스 일시 중지
docker container port docker port	컨테이너 포트 매핑 보기
docker container prune	정지 상태의 컨테이너 모두 삭제
docker container rename docker rename	컨테이너 이름 변경
docker container restart docker restart	컨테이너 다시 시작
docker container rm docker rm	컨테이너 삭제
docker container run docker run	컨테이너 이미지를 이용해서 새로운 컨테이너는 생성하고 실행
docker container start docker start	정지 상태의 컨테이너를 시작
docker container stats docker stats	컨테이너의 자원 이용 상태 보기
docker container stop docker stop	실행 상태의 컨테이너를 중지
docker container top docker top	컨테이너 내부에서 실행 중인 프로세스 보기
docker container unpause docker unpause	(docker pause 명령으로) 일시 정시 상태의 프로세스 재개
docker container update docker update	컨테이너 설정 업데이트
docker container wait docker wait	컨테이너의 종료를 기다리고 나서 종료 코드 보기

볼륨 관리 명령

```
C:\Users\myanj> docker volume --help
```

볼륨은 컨테이너의 라이프 사이클과 별개의 독립적인 영역
컨테이너를 삭제하면 컨테이너에서 변경된 파일은 제거되지만, 볼륨은 명시적으로 삭제하지 않는 한 내용이 유지됨
볼륨은 여러 컨테이너에 걸쳐 공유되며 호스트 환경의 디렉터리를 공유할 수 있음

명령	의미
docker volume create	볼륨 만들기
docker volume inspect	볼륨에 대한 자세한 정보 보기
docker volume ls	볼륨 목록 보기
docker volume prune	불필요한 볼륨 삭제
docker volume rm	볼륨 삭제

네트워크 관리 명령

```
C:\Users\myanj> docker network --help
```

도커는 컨테이너 마다 호스트 환경과 독립적인 네트워크와 네트워크 주소를 할당할 수 있으며, 동일한 포트를 수신하는 여러 개의 컨테이너를 기동할 수 있음
도커는 내부에 DNS 서버를 가지고 있으며, 컨테이너 이름(서비스 이름)을 사용하여 다른 컨테이너와 통신할 수 있음

명령	의미
docker network connect	컨테이너를 네트워크에 연결
docker network create	네트워크 만들기
docker network disconnect	컨테이너를 네트워크와 분리
docker network inspect	네트워크에 대한 자세한 정보 보기
docker network ls	네트워크 목록 보기
docker network prune	불필요한 네트워크 삭제
docker network rm	네트워크 삭제

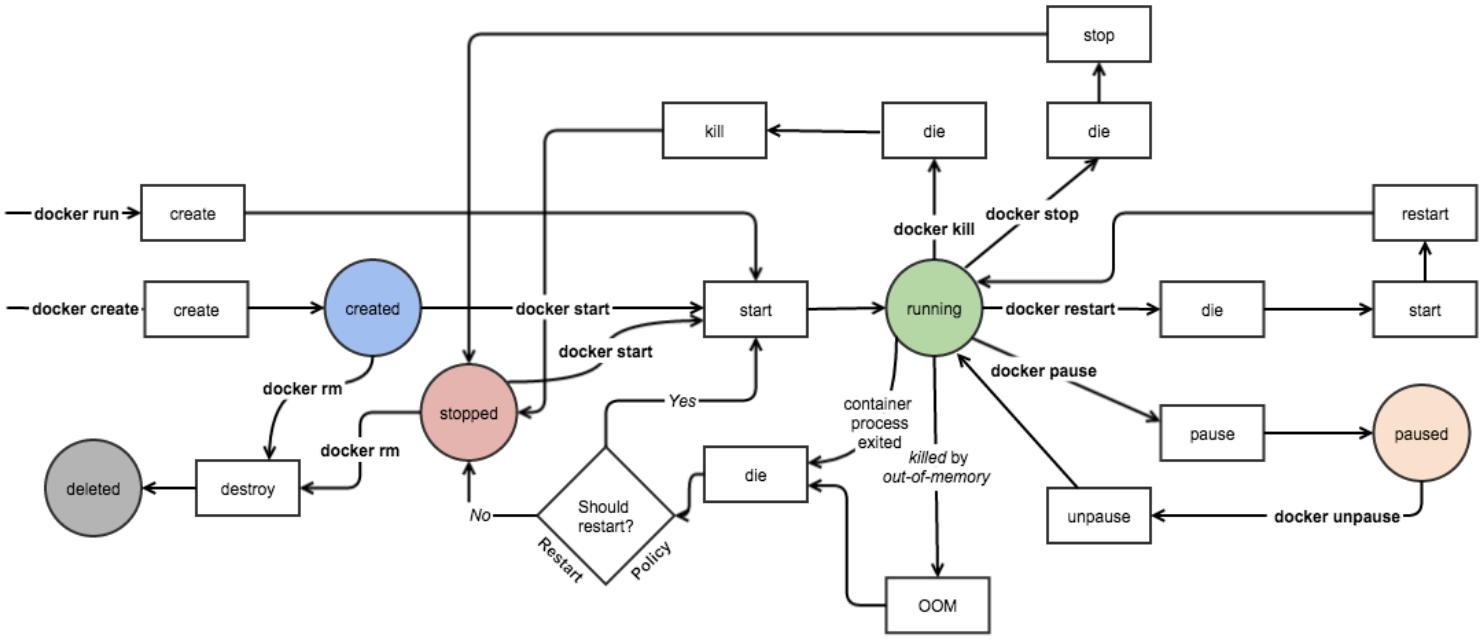
기타 명령

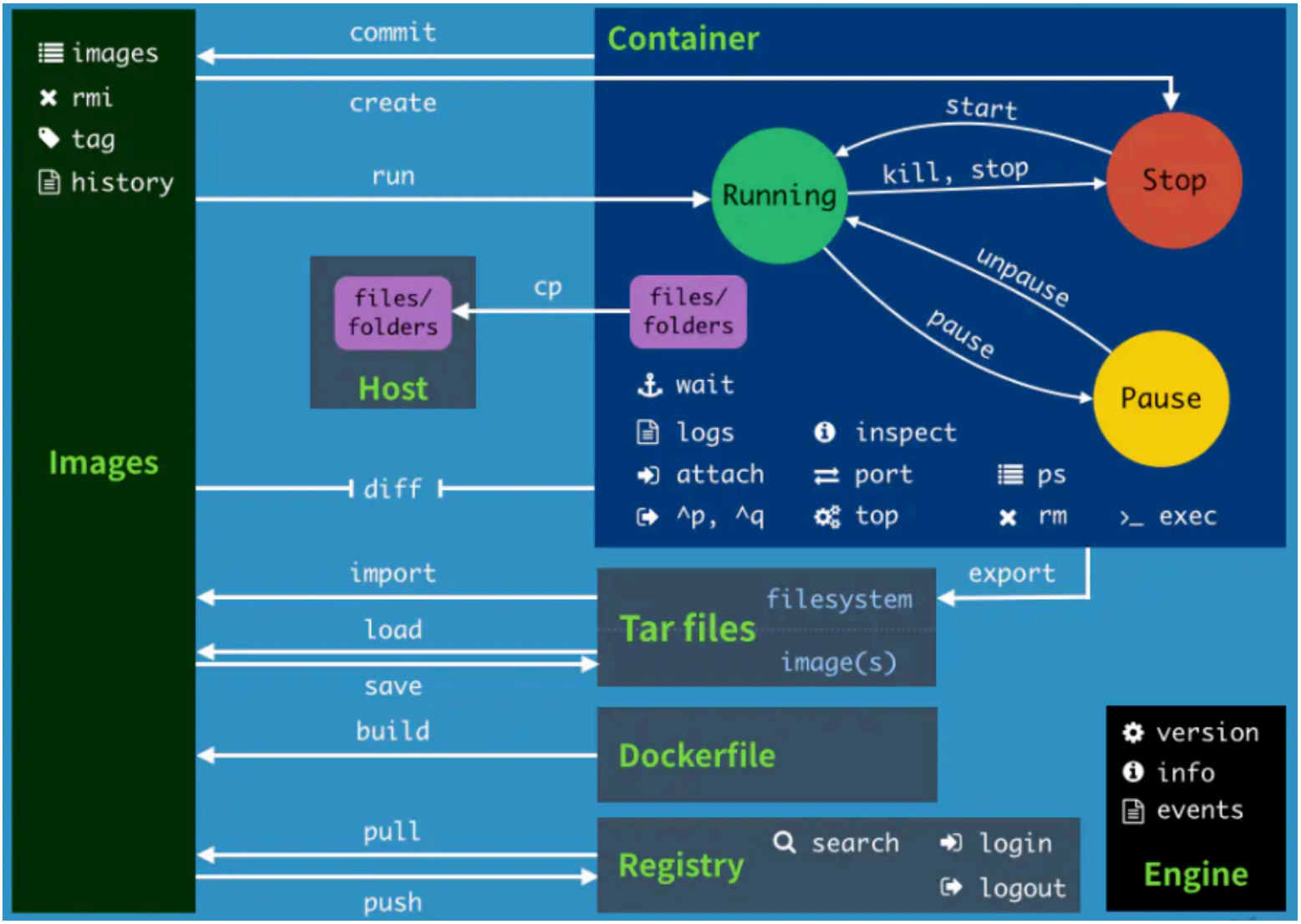
명령	의미
docker builder * docker buildx *	빌드 관련 관리 명령
docker checkpoint *	(실험 중인 기능) 체크 포인트 관리 명령
docker config *	(Swarm 관련 명령) 도커 설정을 관리하는 명령
docker events	도커 서버에서 발생한 이벤트 정보 보기
docker image *	이미지 관리 명령
docker info	시스템 전반의 정보 보기
docker inspect	도커 객체(이미지, 컨테이너, 네트워크 등)에 대한 자세한 정보 보기

docker login	도커 레지스트리에 로그인
docker logout	도커 레지스트리에 로그아웃
docker manifest *	(실험 중인 기능) 매니페스트 관리 명령
docker network *	네트워크 관리 명령
docker plugin *	플러그인 관리 명령
docker search	도커 허브에서 이미지 검색
docker system *	도커 관리 명령
docker trust	이미지 서명 관리 명령
docker version	도커 버전 정보 보기
docker volume *	볼륨 관리 명령

도커 컨테이너 라이프사이클 / Docker Commands from Managing Container Lifecycle

http://docker-saigon.github.io/post/Docker-Internals/





Dockerfile

https://docs.docker.com/engine/reference/builder/

도커 이미지를 빌드하는데 필요한 명령어를 순서대로 기술한 텍스트 파일

Dockerfile 지시어

FROM은 베이스 이미지를 지정하는 지시어로, Dockerile은 FROM 지시어부터 시작해야 하며, 이 이미지를 베이스로해 이어지는 다음 단계의 지시어의 실행 결과(변경 내용)를 축적하며 이미지를 생성

지시어(Instruction)	의미
FROM	베이스 이미지 지정
RUN	베이스 이미지에 새로운 레이어를 추가해 커맨드를 실행하고, 결과를 빌드 이미지에 반영
CMD	컨테이너를 시작할 때 실행할 커맨드를 설정
LABEL	이미지에 레이블을 설정
EXPOSE	컨테이너에서 공개하는 포트 번호 설정

ENV	환경 변수 설정
ADD	이미지에 파일 복사 (압축 파일의 경우 압축을 해제한 후 복사)
COPY	이미지에 파일 복사
ENTRYPOINT	컨테이너를 시작할 때 실행할 커맨드 설정
VOLUME	볼륨이 마운트될 위치를 설정
USER	커맨드를 실행할 사용자 ID 설정
WORKDIR	컨맨드를 실행할 작업 디렉터리를 설정
ARG	빌드 시에만 사용되는 변수 설정
ONBUILD	이 이미지를 베이스 이미지로 사용하여 이미지를 빌드할 때 실행할 커맨드 설정
STOPSIGNAL	컨테이너를 중지시킬 때의 시그널 번호를 설정
HEALTHCHECK	헬스체크를 위한 커맨드 설정
SHELL	기본으로 사용할 쉘을 지정

docker image build = docker build = docker buildx build

~~~~~~~~~~~~~~~~~~~~~~
BuildKit을 이용해 빌드를 수행
Docker 이미지 빌드 프로세스의 성능과 기능을 개선한 빌드 시스템
도커 18.09 이후에 도입
디폴트로 비활성화되어 있으며, 환경 변수를 통해 활성화 가능
향상된 캐싱, 병렬 처리, 시크릿 전달, 멀티 플랫폼 빌드 등 향상된 기능을 제공

- Dockerfile 및 컨텍스트(context)에서 Docker 이미지를 빌드
  ~~~~~~~~~~   ~~~~~~~~~~~~~~~~~~
  ¦   ¦
  ¦   +-- 지정된 경로(PATH) 또는 URL에 있는 파일 세트
  ¦   build 프로세스는 컨텍스트에 있는 모든 파일을 참조할 수 있음
  ¦
  +-- 컨테이너에 설치해야 하는 패키지(FROM), 추가해야 하는 소스코드(COPY, ADD),
      실행해야 하는 명령어와 쉘 스크립트(RUN) 등을 기록한 파일

Go 기반 컨테이어 애플리케이션 개발

#1 작업 디렉터리 생성

```
c:\> mkdir docker\go
c:\> cd docker\go
c:\docker\go> code .                ⇐ 현재 디렉터리를 프로젝트 루트로 설정한 상태로 VSCode를 실행
```

#2 소스 코드 작성

```
c:\docker\go\main.go
```

```
/* 8080 포트로 HTTP 요청을 대기하다가, /로 요청이 들어오면 Hello Docker!!!를 응답 */


/* 프로그램의 진입점인 main 패키지를 지정 */
package main


/* 표준 입출력과 문자열 형식을 처리하는 Go 패키지,
   로그를 출력하기 위한 패키지,
   HTTP 서버와 클라이언트 관련 기능을 제공하는 패키지를 임포트 */
import (
        "fmt"
        "log"
        "net/http"
)

func main() {
        http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
                log.Println("received request")
                fmt.Fprintf(w, "Hello Docker!!")
        })

        log.Println("start server")
        server := &http.Server{Addr: ":8080"}
        if err := server.ListenAndServe(); err != nil {
                log.Println(err)
        }
}
```

#3 도커 이미지 생성

Dockerfile 작성

c:\docker\Dockerfile

```
FROM golang:1.23.1                      # 베이스 이미지를 지정 (저장소 이름이 없으므로 도커 허브 공식 이미지)
                                        # https://docs.docker.com/engine/reference/builder/#from
                                        # https://hub.docker.com/_/golang

RUN mkdir /echo                         # 컨테이너 내부에 /echo 디렉터리를 생성
                                        # https://docs.docker.com/engine/reference/builder/#run

COPY main.go /echo                      # 호스트의 main.go 파일을 컨테이너의 /echo/main.go로 복사
                                        # https://docs.docker.com/engine/reference/builder/#copy

CMD ["go", "run", "/echo/main.go"]      # 컨테이너 실행 시 go run /echo/main.go 명령어를 실행
                                        # https://docs.docker.com/engine/reference/builder/#cmd
```

도커 이미지 빌드 및 확인

```
c:\docker\go> docker image build --help

Usage:  docker buildx build [OPTIONS] PATH | URL | -

Start a build

Aliases:
```

```
  docker buildx build, docker buildx b

Options:
      --add-host strings              Add a custom host-to-IP mapping (format: "host:ip")
      --allow strings                 Allow extra privileged entitlement (e.g., "network.host", "security.insecure")
      --attest stringArray            Attestation parameters (format: "type=sbom,generator=image")
      --build-arg stringArray         Set build-time variables
      --build-context stringArray     Additional build contexts (e.g., name=path)
      --builder string                Override the configured builder instance (default "default")
      --cache-from stringArray        External cache sources (e.g., "user/app:cache", "type=local,src=path/to/dir")
      --cache-to stringArray          Cache export destinations (e.g., "user/app:cache",
                                      "type=local,dest=path/to/dir")
      --cgroup-parent string          Optional parent cgroup for the container
  -f, --file string                   Name of the Dockerfile (default: "PATH/Dockerfile")
      --iidfile string                Write the image ID to the file
      --label stringArray             Set metadata for an image
      --load                          Shorthand for "--output=type=docker"
      --metadata-file string          Write build result metadata to the file
      --network string                Set the networking mode for the "RUN" instructions during build
                                      (default "default")
      --no-cache                      Do not use cache when building the image
      --no-cache-filter stringArray   Do not cache specified stages
  -o, --output stringArray            Output destination (format: "type=local,dest=path")
      --platform stringArray          Set target platform for build
      --progress string               Set type of progress output ("auto", "plain", "tty"). Use plain
                                      to show container output (default "auto")
      --provenance string             Shorthand for "--attest=type=provenance"
      --pull                          Always attempt to pull all referenced images
      --push                          Shorthand for "--output=type=registry"
  -q, --quiet                         Suppress the build output and print image ID on success
      --sbom string                   Shorthand for "--attest=type=sbom"
      --secret stringArray            Secret to expose to the build (format: "id=mysecret[,src=/local/secret]")
      --shm-size bytes                Size of "/dev/shm"
      --ssh stringArray               SSH agent socket or keys to expose to the build (format:
                                      "default|<id>[=<socket>|<key>[,<key>]]")
  -t, --tag stringArray               Name and optionally a tag (format: "name:tag")
      --target string                 Set the target build stage to build
      --ulimit ulimit                 Ulimit options (default [])

c:\docker\go> docker buildx build -t example/echo:latest .
[+] Building 152.5s (9/9) FINISHED      docker:desktop-linux
 => [internal] load build definition from Dockerfile    0.1s
 => => transferring dockerfile: 183B                    0.0s
 => [internal] load metadata for docker.io/library/gol  6.9s
 => [auth] library/golang:pull token for registry-1.do  0.0s
 => [internal] load .dockerignore                       0.0s
 => => transferring context: 2B                         0.0s
 => [1/3] FROM docker.io/library/golang:1.23.1@sha25  143.6s
 => => resolve docker.io/library/golang:1.23.1@sha256:  0.1s
 => => sha256:32d3574b34bd65a6cf89a80e5bd9 126B / 126B  1.0s
 => => sha256:4f4fb700ef54461cfa02571ae0db9a 32B / 32B  0.8s
 => => sha256:e7bff916ab0c126c9d94 74.00MB / 74.00MB  128.7s
 => => sha256:1eb015951d08f558e980 92.26MB / 92.26MB  138.7s
 => => sha256:a173f2aee8e962ea19db 64.39MB / 64.39MB  122.9s
```

```
 => => sha256:a47cff7f31e941e78bf63 24.05MB / 24.05MB  11.8s
 => => sha256:cdd62bf39133c498a16f 49.56MB / 49.56MB  107.1s
 => => extracting sha256:cdd62bf39133c498a16f7a7b1b655  1.5s
 => => extracting sha256:a47cff7f31e941e78bf63ca19f081  0.5s
 => => extracting sha256:a173f2aee8e962ea19db1e418ae84  1.7s
 => => extracting sha256:1eb015951d08f558e9805d427f6d3  1.6s
 => => extracting sha256:e7bff916ab0c126c9d943f0c481a9  2.8s
 => => extracting sha256:32d3574b34bd65a6cf89a80e5bd93  0.0s
 => => extracting sha256:4f4fb700ef54461cfa02571ae0db9  0.0s
 => [internal] load build context                      0.1s
 => => transferring context: 790B                      0.0s
 => [2/3] RUN mkdir /echo                              1.0s
 => [3/3] COPY main.go /echo                           0.1s
 => exporting to image                                 0.5s
 => => exporting layers                                0.2s
 => => exporting manifest sha256:2bae16f1b27bc069ecd48  0.0s
 => => exporting config sha256:98719096f93a77b70c2b3af  0.0s
 => => exporting attestation manifest sha256:0025a4d0c  0.0s
 => => exporting manifest list sha256:e50db4d09d8cd292  0.0s
 => => naming to docker.io/example/echo:latest         0.0s
 => => unpacking to docker.io/example/echo:latest      0.1s
```

이미지 생성 확인

```
c:\docker\go> docker image ls
                                      i Info →   U  In Use
IMAGE                ID             DISK USAGE   CONTENT SIZE
example/echo:latest
                     a3e1481c7a70       1.23GB          304MB
```

도커 허브에 이미지 등록

```
c:\docker\go> docker image push example/echo:latest
The push refers to repository [docker.io/example/echo]
e7bff916ab0c: Waiting
2ae31906c633: Waiting
1eb015951d08: Waiting
32d3574b34bd: Waiting
cdd62bf39133: Waiting
a47cff7f31e9: Waiting
4f4fb700ef54: Waiting
3e0c1b642ded: Waiting
a173f2aee8e9: Waiting
e4e9b89b7879: Waiting
push access denied, repository does not exist or may require authorization: server message: insufficient_scope:
authorization failed
```

이미지 태그를 변경

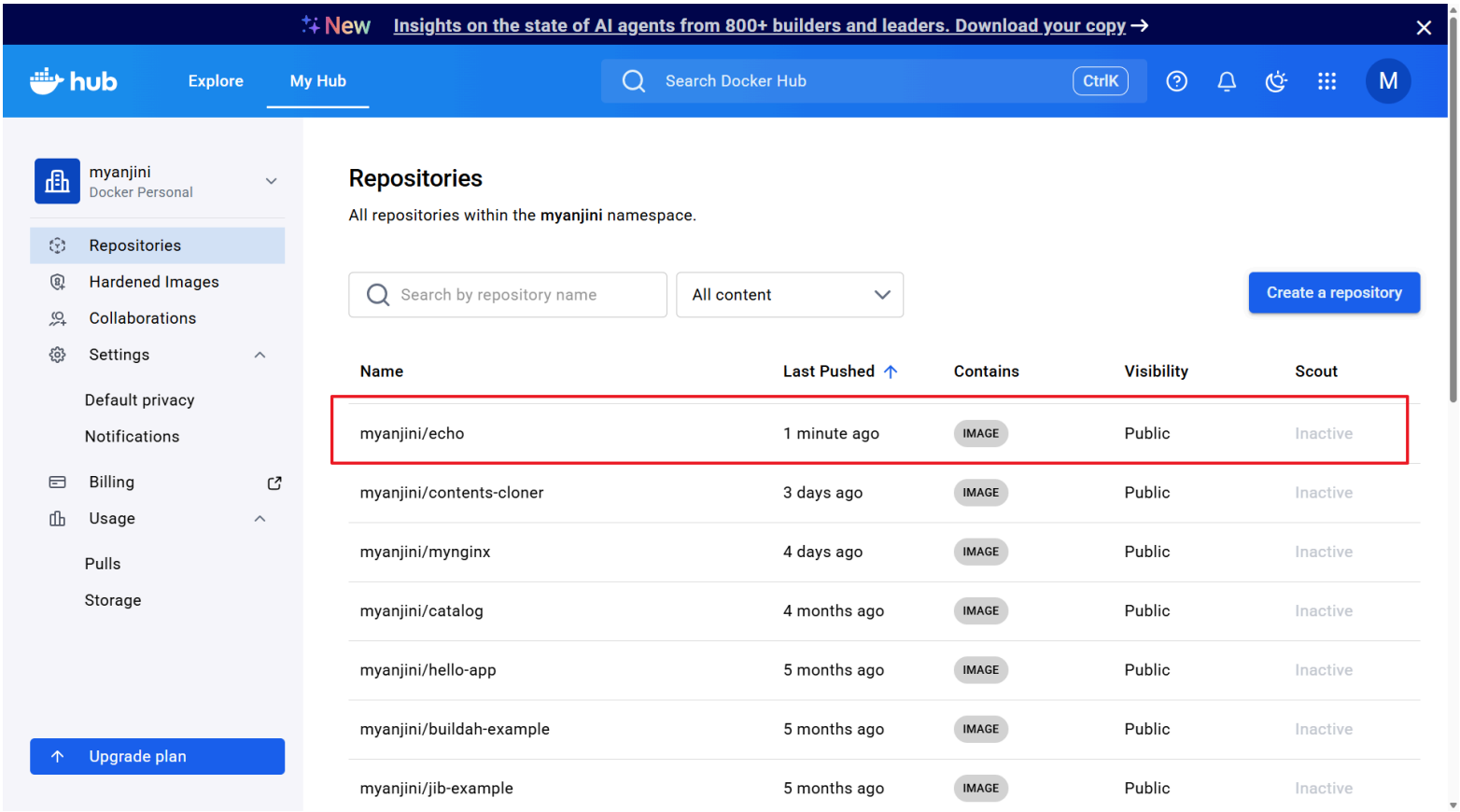
```
c:\docker\go> docker image tag example/echo:latest myanjini/echo:latest
                                                 ~~~~~~~~
                                                 본인의 도커 허브 username 을 사용
```

```
c:\docker\go> docker image ls
                                    i Info →   U  In Use
IMAGE                ID             DISK USAGE   CONTENT SIZE
example/echo:latest
                     a3e1481c7a70       1.23GB          304MB
myanjini/echo:latest
                     a3e1481c7a70       1.23GB          304MB
```

이미지 등록

```
c:\docker\go> docker image push myanjini/echo:latest
The push refers to repository [docker.io/myanjini/echo]
a173f2aee8e9: Layer already exists
4f4fb700ef54: Layer already exists
a47cff7f31e9: Layer already exists
e7bff916ab0c: Layer already exists
32d3574b34bd: Layer already exists
1eb015951d08: Layer already exists
3e0c1b642ded: Pushed
e4e9b89b7879: Pushed
2ae31906c633: Pushed
cdd62bf39133: Layer already exists
latest: digest: sha256:a3e1481c7a703c53c6ef384197aef31925a7d2f796bd4cd9bd0e2799b2e9c82f size: 856
```

도커 허브에 이미지 등록을 확인



도커 허브에 등록된 이미지를 이용해서 컨테이너를 실행

```
c:\docker\go> docker container run -d -p 8282:8080 --name myecho --rm myanjini/echo:latest
                                  ~~ ~~~~~~~~~~~~ ~~~~~~~~~~~~~~ ~~~~ ~~~~~~~~~~~~~~~~~~~~
                                  ¦  ¦            ¦              ¦    컨테이너 실행에 사용할 이미지
                                  ¦  ¦            ¦              +-- 컨테이너가 종료되면 자동으로 삭제
                                  ¦  ¦            +-- 컨테이너 이름 ⇒ 지정하지 않으면 랜덤하게 생성
                                  ¦  +--  포트 포워딩 ⇒ 호스트의 8282 포트를 컨테이너 내부에 8080 포트로 연결
                                  +-- detach 모드로 실행
69712c8d3bd155c27e5da65dd888244a2ebd502adfd8fdc486be9a1d0d52c1ed        ⇐ 컨테이너 ID

c:\docker\go> docker container ls
CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS          PORTS
NAMES
69712c8d3bd1   myanjini/echo:latest   "go run /echo/main.go"   13 seconds ago   Up 13 seconds
0.0.0.0:8282->8080/tcp, [::]:8282->8080/tcp   myecho
```

컨테이너 동작을 확인 **(**명령 프롬프트에서**)**

```
c:\docker\go> curl -X GET http://localhost:8282
Hello Docker ^..^!!
```

-i -t -d 옵션

detach 모드로 실행 ⇒ 컨테이너 **ID**를 출력하고**,** 호스트로 제어권을 반환

```
c:\docker\go> docker container run -d -p 7777:8080 myanjini/echo
378901e21c4efd974fb654839f1d89df8739ffed7c03732bc2acd49451395f46
```

실행 상태의 컨테이너 목록을 조회

```
c:\docker\go> docker container ls
CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS          PORTS
NAMES
378901e21c4e   myanjini/echo          "go run /echo/main.go"   11 seconds ago   Up 10 seconds
0.0.0.0:7777->8080/tcp, [::]:7777->8080/tcp   quizzical_napier
69712c8d3bd1   myanjini/echo:latest   "go run /echo/main.go"   2 hours ago      Up 2 hours
0.0.0.0:8282->8080/tcp, [::]:8282->8080/tcp   myecho
```

attach 모드로 실행 ⇒ 제어권을 컨테이너 내부로 전달 → 호스트의 입력이 컨테이너 종료 전까지 실행되지 않음

```
c:\docker\go> docker container run -p 8888:8080 myanjini/echo
2026/04/08 05:23:54 start server                ⇐ 컨테이너 내부의 출력
dir 명령어 입력 후 Ctrl+C 를 3번 입력
got 3 SIGTERM/SIGINTs, forcefully exiting       ⇐ detach 모드로 전환
```

```
c:\docker\go>dir                                    ⇐ 큐에 쌓여있던 명령어가 실행
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: 9243-B031

 c:\docker\go 디렉터리

2026-04-08  오전 11:44    <DIR>          .
2026-04-08  오전 11:38    <DIR>          ..
2026-04-08  오전 11:50               144 Dockerfile
2026-04-08  오후 12:11               759 main.go
               2개 파일                 903 바이트
               2개 디렉터리  95,584,976,896 바이트 남음

c:\docker\go> docker container ls
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS
NAMES
297e3b2a6ec1   myanjini/echo          "go run /echo/main.go"   2 minutes ago   Up 2 minutes
0.0.0.0:8888->8080/tcp, [::]:8888->8080/tcp   upbeat_kowalevski        ⇐ 실행 상태를 유지하고 있는 것을 확인
378901e21c4e   myanjini/echo          "go run /echo/main.go"   3 minutes ago   Up 3 minutes
0.0.0.0:7777->8080/tcp, [::]:7777->8080/tcp   quizzical_napier
69712c8d3bd1   myanjini/echo:latest   "go run /echo/main.go"   2 hours ago     Up 2 hours
0.0.0.0:8282->8080/tcp, [::]:8282->8080/tcp   myecho
```

컨테이너의 동작을 확인

```
c:\docker\go> curl http://localhost:8888
Hello Docker ^..^!!
```

-it 옵션과 함께 컨테이너를 실행 ⇒ 호스트의 입력을 컨테이너로 전달

```
c:\docker\go> docker container run -it -p 9999:8080 myanjini/echo
2026/04/08 05:27:25 start server
Ctrl + p + q                ⇐ detach string을 입력 → detach 모드로 전환 ← --detach-keys 옵션으로 변경 가능
```

쉘을 실행(=입출력을 필요)하는 컨테이너를 실행

```
c:\docker\go> docker container run ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
689b91d88a0f: Pull complete
b22f29e93d86: Download complete
Digest: sha256:84e77dee7d1bc93fb029a45e3c6cb9d8aa4831ccfcc7103d36e876938d28895b
Status: Downloaded newer image for ubuntu:latest
```

실행 상태의 컨테이너를 조회 ⇒ 우분투 컨테이너가 출력되지 않음

```
c:\docker\go> docker container ls
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS
NAMES
```

```
ebd61ea7bfb1   myanjini/echo          "go run /echo/main.go"   4 minutes ago   Up 4 minutes
0.0.0.0:9999->8080/tcp, [::]:9999->8080/tcp   happy_tharp
297e3b2a6ec1   myanjini/echo          "go run /echo/main.go"   7 minutes ago   Up 7 minutes
0.0.0.0:8888->8080/tcp, [::]:8888->8080/tcp   upbeat_kowalevski
378901e21c4e   myanjini/echo          "go run /echo/main.go"   8 minutes ago   Up 8 minutes
0.0.0.0:7777->8080/tcp, [::]:7777->8080/tcp   quizzical_napier
69712c8d3bd1   myanjini/echo:latest   "go run /echo/main.go"   2 hours ago     Up 2 hours
0.0.0.0:8282->8080/tcp, [::]:8282->8080/tcp   myecho
```

모든 상태의 컨테이너를 조회

```
c:\docker\go> docker container ls  -a
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS                      PORTS
NAMES
febf0f553b9e   ubuntu                 "/bin/bash"              29 seconds ago   Exited (0) 28 seconds ago
vibrant_montalcini                 ⇐ 컨테이너가 실행과 동시에 종료된 것을 확인할 수 있음 ⇐ stdio를 필요로 함
ebd61ea7bfb1   myanjini/echo          "go run /echo/main.go"   4 minutes ago    Up 4 minutes
0.0.0.0:9999->8080/tcp, [::]:9999->8080/tcp   happy_tharp
297e3b2a6ec1   myanjini/echo          "go run /echo/main.go"   8 minutes ago    Up 8 minutes
0.0.0.0:8888->8080/tcp, [::]:8888->8080/tcp   upbeat_kowalevski
378901e21c4e   myanjini/echo          "go run /echo/main.go"   8 minutes ago    Up 8 minutes
0.0.0.0:7777->8080/tcp, [::]:7777->8080/tcp   quizzical_napier
69712c8d3bd1   myanjini/echo:latest   "go run /echo/main.go"   2 hours ago      Up 2 hours
0.0.0.0:8282->8080/tcp, [::]:8282->8080/tcp   myecho
```

-it 옵션과 함께 실행

```
c:\docker\go> docker container run -it ubuntu
root@7eae0d282ed8:/# exit                ⇐ 쉘이 제공되는 것을 확인
exit                                        → exit 를 입력하면 해당 쉘을 종료하므로 컨테이너가 함께 종료

c:\docker\go> docker container ls -a
CONTAINER ID   IMAGE                  COMMAND                  CREATED              STATUS
PORTS                                      NAMES
7eae0d282ed8   ubuntu                 "/bin/bash"              About a minute ago   Exited (0) 14 seconds ago
objective_nobel                        ⇐ /bin/bash 를 exit (종료) 했기 때문에 컨테이너가 종료됨
febf0f553b9e   ubuntu                 "/bin/bash"              2 minutes ago        Exited (0) 2 minutes ago
vibrant_montalcini
ebd61ea7bfb1   myanjini/echo          "go run /echo/main.go"   7 minutes ago        Up 7 minutes
0.0.0.0:9999->8080/tcp, [::]:9999->8080/tcp   happy_tharp
297e3b2a6ec1   myanjini/echo          "go run /echo/main.go"   10 minutes ago       Up 10 minutes
0.0.0.0:8888->8080/tcp, [::]:8888->8080/tcp   upbeat_kowalevski
378901e21c4e   myanjini/echo          "go run /echo/main.go"   11 minutes ago       Up 11 minutes
0.0.0.0:7777->8080/tcp, [::]:7777->8080/tcp   quizzical_napier
69712c8d3bd1   myanjini/echo:latest   "go run /echo/main.go"   2 hours ago          Up 2 hours
0.0.0.0:8282->8080/tcp, [::]:8282->8080/tcp   myecho

c:\docker\go> docker container run -it ubuntu
root@8879d52cf047:/# Ctrl + p + q          ⇐ detach 모드로 전환 → 해당 컨테이너는 실행 상태를 유지
c:\docker\go> docker container ls -a
```

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS                      PORTS
NAMES
8879d52cf047   ubuntu                 "/bin/bash"              25 seconds ago   Up 25 seconds
stupefied_faraday                                  ⇐ 실행 상태를 유지
7eae0d282ed8   ubuntu                 "/bin/bash"              2 minutes ago    Exited (0) 49 seconds ago
objective_nobel
febf0f553b9e   ubuntu                 "/bin/bash"              3 minutes ago    Exited (0) 3 minutes ago
vibrant_montalcini
ebd61ea7bfb1   myanjini/echo          "go run /echo/main.go"   7 minutes ago    Up 7 minutes
0.0.0.0:9999->8080/tcp, [::]:9999->8080/tcp   happy_tharp
297e3b2a6ec1   myanjini/echo          "go run /echo/main.go"   11 minutes ago   Up 11 minutes
0.0.0.0:8888->8080/tcp, [::]:8888->8080/tcp   upbeat_kowalevski
378901e21c4e   myanjini/echo          "go run /echo/main.go"   11 minutes ago   Up 11 minutes
0.0.0.0:7777->8080/tcp, [::]:7777->8080/tcp   quizzical_napier
69712c8d3bd1   myanjini/echo:latest   "go run /echo/main.go"   2 hours ago      Up 2 hours
0.0.0.0:8282->8080/tcp, [::]:8282->8080/tcp   myecho
```

실행 상태의 컨테이너로 접속 방법

```
c:\docker\go> docker container attach 88              ⇐ 컨테이너 ID가 88로 시작하는 컨테이너로 attach
root@8879d52cf047:/# read escape sequence             ⇐ Ctrl + p + q 입력해서 detach 모드로 전환

c:\docker\go> docker container exec -it 88 /bin/bash
root@8879d52cf047:/# exit                         ⇐ exec 명령어로 실행한 쉘을 종료 → 컨테이너는 실행 상태를 유지
exit

c:\docker\go> docker container ls
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS
NAMES
8879d52cf047   ubuntu                 "/bin/bash"              3 minutes ago    Up 3 minutes
stupefied_faraday                                  ⇐ 실행 상태를 유지하고 있음
ebd61ea7bfb1   myanjini/echo          "go run /echo/main.go"   11 minutes ago   Up 11 minutes
0.0.0.0:9999->8080/tcp, [::]:9999->8080/tcp   happy_tharp
297e3b2a6ec1   myanjini/echo          "go run /echo/main.go"   14 minutes ago   Up 14 minutes
0.0.0.0:8888->8080/tcp, [::]:8888->8080/tcp   upbeat_kowalevski
378901e21c4e   myanjini/echo          "go run /echo/main.go"   15 minutes ago   Up 15 minutes
0.0.0.0:7777->8080/tcp, [::]:7777->8080/tcp   quizzical_napier
69712c8d3bd1   myanjini/echo:latest   "go run /echo/main.go"   2 hours ago      Up 2 hours
0.0.0.0:8282->8080/tcp, [::]:8282->8080/tcp   myecho

c:\docker\go> docker container exec -it 88 /bin/bash
root@8879d52cf047:/# ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 05:34 pts/0    00:00:00 /bin/bash ⇐ 컨테이너 실행 시 실행한 쉘 (Dockerfile의 CMD 명령)
root          17     0  0 05:38 pts/1    00:00:00 /bin/bash ⇐ exec 명령으로 실행한 쉘
root          25    17  0 05:38 pts/1    00:00:00 ps -ef
root@8879d52cf047:/# exit                             ⇐ PID 17 의 쉘을 종료
exit
```