

2026/04/08~15

S-개발자 4기

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

내용 공유 >>>

<http://bit.ly/4cbwMse>

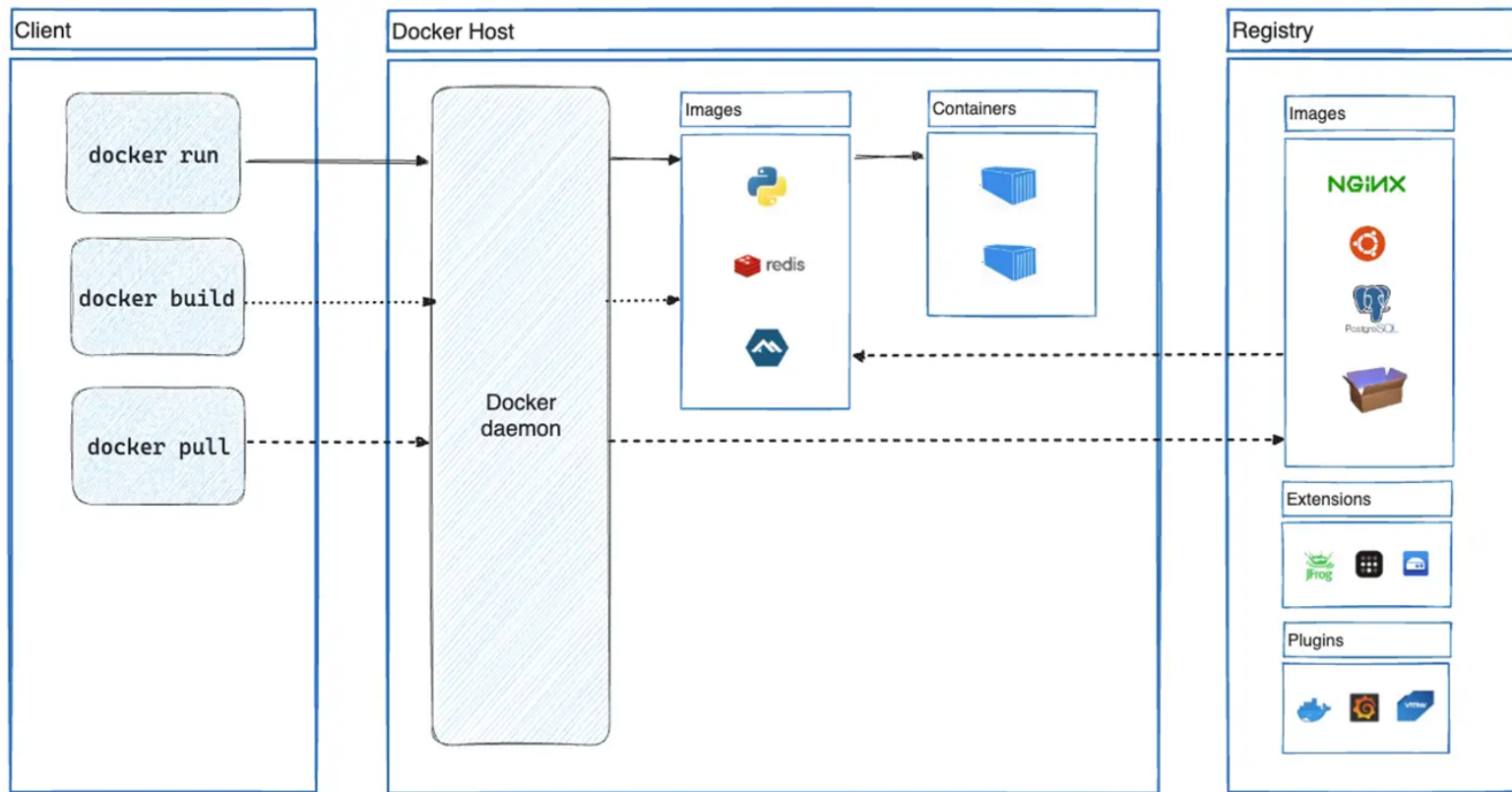
붉은색만 숫자입니다. 대소문자를 구분해서 입력해 주세요.
수업 종료 후 링크가 삭제될 수 있으니, 필요한 내용은 미리 백업해 두세요.

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 저장**하므로 컨테이너에서 무엇을 하든지 원래 이미지는 영향을 받지 않음
- 생성된 컨테이너는 각기 **독립된 파일 시스템**을 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 정지 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

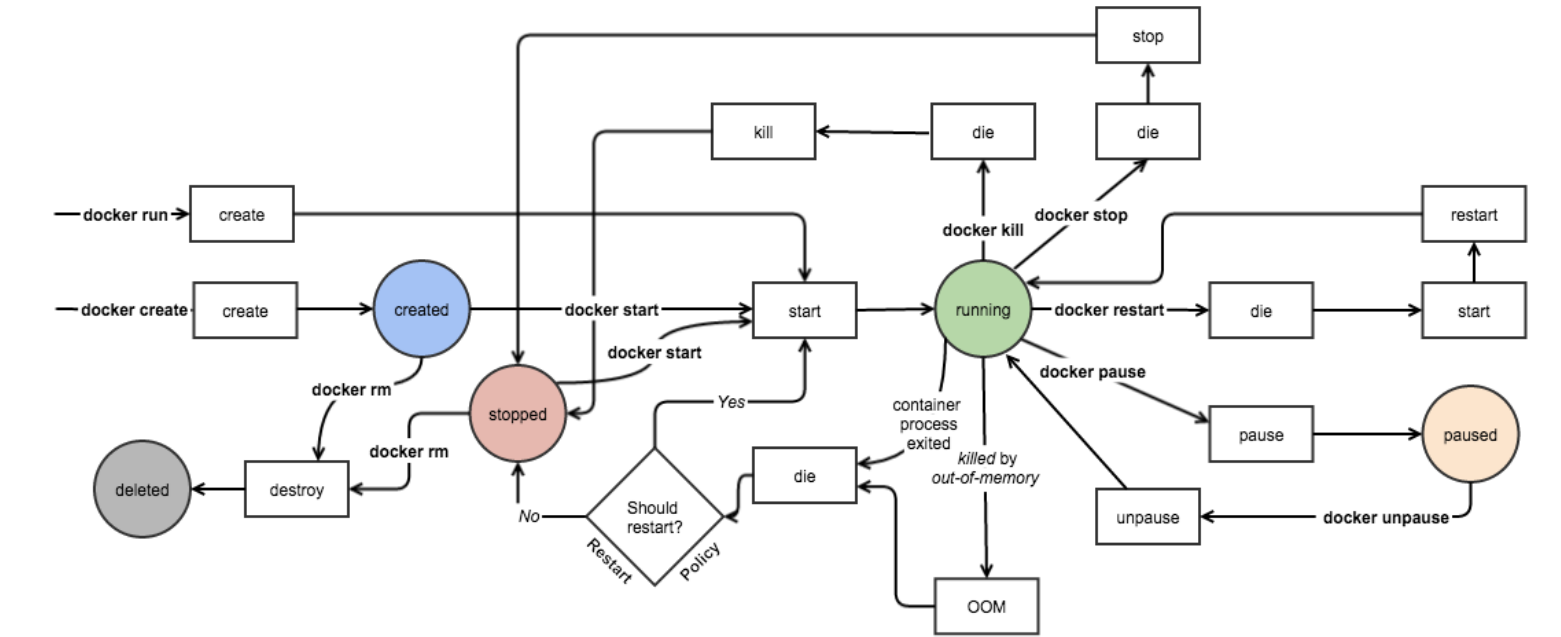
기타 명령

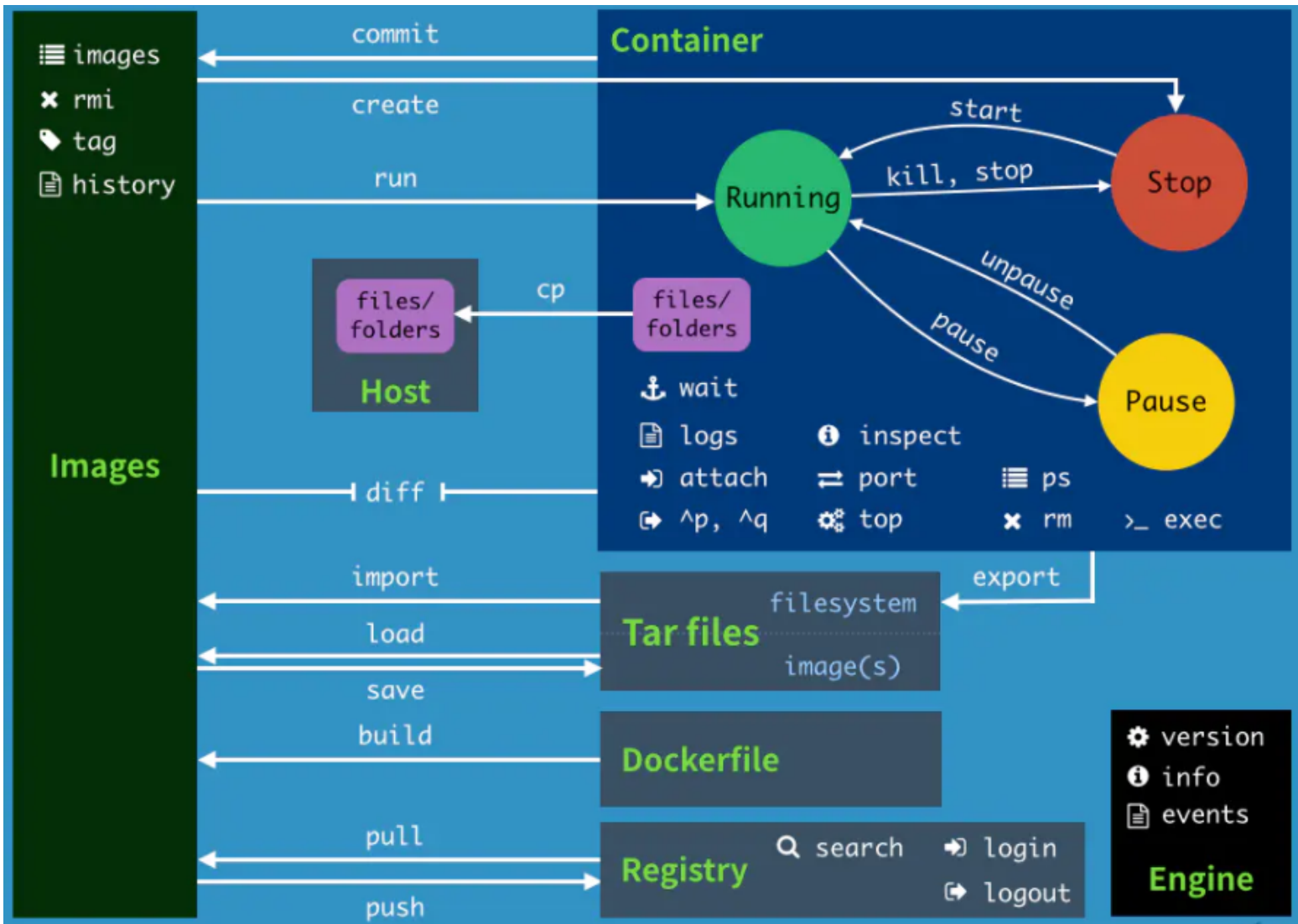
명령	의미
docker builder * docker buildx *	빌드 관련 관리 명령
docker checkpoint *	(실험 중인 기능) 체크 포인트 관리 명령
docker config *	(Swarm 관련 명령) 도커 설정을 관리하는 명령
docker events	도커 서버에서 발생한 이벤트 정보 보기
docker image *	이미지 관리 명령
docker info	시스템 전반의 정보 보기
docker inspect	도커 객체(이미지, 컨테이너, 네트워크 등)에 대한 자세한 정보 보기

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, Dockerfile은 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	커맨드를 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 셸을 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 셸 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 импорт */
```

```
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 디렉토리를 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:

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

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

| IMAGE                | ID           | i Info →   | U In Use     |
|----------------------|--------------|------------|--------------|
|                      |              | 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:a3e1481c7a703c53cef384197aef31925a7d2f796bd4cd9bd0e2799b2e9c82f size: 856

### 도커 허브에 이미지 등록을 확인

New Insights on the state of AI agents from 800+ builders and leaders. Download your copy →

hub

ExploreMy Hub

Search Docker Hub

CtrlK

M

myanjini  
Docker Personal

Repositories

Hardened Images

Collaborations

Settings

Default privacy

Notifications

Billing

Usage

Pulls

Storage

Upgrade plan

Repositories

All repositories within the myanjini namespace.

Search by repository name

All content

Create a repository

| Name                     | Last Pushed ↑ | Contains | Visibility | Scout    |
|--------------------------|---------------|----------|------------|----------|
| myanjini/echo            | 1 minute ago  | IMAGE    | Public     | Inactive |
| myanjini/contents-cloner | 3 days ago    | IMAGE    | Public     | Inactive |
| myanjini/mynginx         | 4 days ago    | IMAGE    | Public     | Inactive |
| myanjini/catalog         | 4 months ago  | IMAGE    | Public     | Inactive |
| myanjini/hello-app       | 5 months ago  | IMAGE    | Public     | Inactive |
| myanjini/buildah-example | 5 months ago  | IMAGE    | Public     | Inactive |
| myanjini/jib-example     | 5 months ago  | IMAGE    | Public     | Inactive |

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 ID를 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
dir 명령어 입력 후 Ctrl+C 를 3번 입력
got 3 SIGTERM/SIGINTs, forcefully exiting
```

← 컨테이너 내부의 출력

← 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

셸을 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
```

```
exit
```

← 셸이 제공되는 것을 확인

→ exit 를 입력하면 해당 셸을 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
```

```
c:\docker\go> docker container ls -a
```

← detach 모드로 전환 → 해당 컨테이너는 실행 상태를 유지

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					
feb0f553b9e	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 셸을 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 셸 (Dockerfile의 CMD 명령)
root 17 0 0 05:38 pts/1 00:00:00 /bin/bash ⇐ exec 명령으로 실행한 셸
root 25 17 0 05:38 pts/1 00:00:00 ps -ef
root@8879d52cf047:/# exit ⇐ PID 17 의 셸을 종료
exit

모든 컨테이너를 삭제 => 윈도우의 경우, powershell에서

```
PS C:\Users\myanj> docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
8879d52cf047	ubuntu	"/bin/bash"	About an hour ago	Up About an hour
stupefied_faraday				
7eae0d282ed8	ubuntu	"/bin/bash"	About an hour ago	Exited (0) About an hour ago
objective_nobel				
febf0f553b9e	ubuntu	"/bin/bash"	About an hour ago	Exited (0) About an hour ago
vibrant_montalcini				
ebd61ea7bfb1	myanjini/echo	"go run /echo/main.go"	About an hour ago	Up About an hour
0.0.0.0:9999->8080/tcp, [::]:9999->8080/tcp		happy_tharp		
297e3b2a6ec1	myanjini/echo	"go run /echo/main.go"	About an hour ago	Up About an hour
0.0.0.0:8888->8080/tcp, [::]:8888->8080/tcp		upbeat_kowalevski		
378901e21c4e	myanjini/echo	"go run /echo/main.go"	About an hour ago	Up About an hour
0.0.0.0:7777->8080/tcp, [::]:7777->8080/tcp		quizzical_napier		
69712c8d3bd1	myanjini/echo:latest	"go run /echo/main.go"	3 hours ago	Up 3 hours
0.0.0.0:8282->8080/tcp, [::]:8282->8080/tcp		myecho		

```
PS C:\Users\myanj> docker container rm -f $(docker container ls -a -q)
```

~~~~~

! 모든 상태의 컨테이너 ID를 조회

+++ 강제로 삭제

```
PS C:\Users\myanj> docker container ls -a
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|-------|---------|---------|--------|-------|-------|
|--------------|-------|---------|---------|--------|-------|-------|

Dockerfile 작성 시 유의 사항 1 - 사용자 입력이 발생하지 않도록 하고, 포그라운드 프로세스로 실행되도록 해야 함

우분투 컨테이너에 아파치 웹 서버를 설치하고 hello.html 파일을 /var/www/html 디렉터리에 추가

```
작업 디렉터리를 생성하고 이동
c:\docker\go> mkdir ../webserver && cd ../webserver

c:\docker\webserver>
```

```
우분투 컨테이너를 실행하고 아파치 웹 서버를 설치

c:\docker\webserver> docker container run -it --name myweb --rm -p 8888:80 ubuntu
root@fef4707e5a7f:/#
root@fef4707e5a7f:/# apt-get update
root@fef4707e5a7f:/# apt-get install apache2
:
0 upgraded, 43 newly installed, 0 to remove and 2 not upgraded.
Need to get 26.3 MB of archives.
After this operation, 109 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
:
root@fef4707e5a7f:/# apachectl start
```

AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message

```
root@fef4707e5a7f:/# echo "ServerName localhost" >> /etc/apache2/apache2.conf
root@fef4707e5a7f:/# apachectl restart
```

```
root@fef4707e5a7f:/# ls /var/www/html
index.html
```

```
root@fef4707e5a7f:/# vi /var/www/html/hello.html
bash: vi: command not found
```

```
root@fef4707e5a7f:/# Ctrl + p + q
c:\docker\webserver>
```

c:\docker\webserver\hello.html 파일을 생성

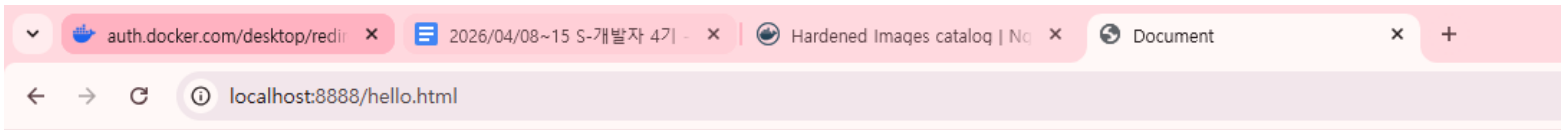
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello Docker !!!!</h1>
</body>
</html>
```

호스트의 hello.html 파일을 컨테이너 내부로 복사

```
c:\docker\webserver> docker container ls
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS
NAMES
fef4707e5a7f   ubuntu    "/bin/bash"             9 minutes ago Up 9 minutes   0.0.0.0:8888->80/tcp, [::]:8888->80/tcp
myweb
```

```
c:\docker\webserver> docker container cp .\hello.html myweb:/var/www/html/
Successfully copied 2.05kB to myweb:/var/www/html/
```

브라우저를 이용해서 컨테이너 동작을 확인  
http://localhost:8888/hello.html



Hello Docker !!!!

동일한 기능을 제공하는 **Dockerfile**을 작성하고 컨테이너 이미지를 생성

c:\docker\webserver\Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install apache2
RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf
ADD hello.html /var/www/html/
CMD apachectl start
```

c:\docker\webserver> docker image build -t mywebserver:v1 .

```
[+] Building 14.3s (7/9)          docker:desktop-linux
=> [internal] load build definition from Dockerfile          0.1s
=> => transferring dockerfile: 226B                          0.0s
=> [internal] load metadata for docker.io/library/ubuntu:l  0.1s
=> [internal] load .dockerignore                          0.1s
=> => transferring context: 2B                                0.0s
=> [1/5] FROM docker.io/library/ubuntu:latest@sha256:84e77 0.1s
=> => resolve docker.io/library/ubuntu:latest@sha256:84e77 0.1s
=> [internal] load build context                          0.1s
=> => transferring context: 280B                             0.1s
=> [2/5] RUN apt-get update                               11.8s
=> ERROR [3/5] RUN apt-get install apache2                2.0s
-----
> [3/5] RUN apt-get install apache2:
0.382 Reading package lists...
1.326 Building dependency tree...
1.539 Reading state information...
1.833 The following additional packages will be installed:
1.833 adduser apache2-bin apache2-data apache2-utils ca-certificates krb5-locales
1.833 libapr1t64 libaprutil1-dbd-sqlite3 libaprutil1-ldap libaprutil1t64
1.834 libbrotli1 libcurl4t64 libexpat1 libgdbm-compat4t64 libgdbm6t64
1.834 libgssapi-krb5-2 libicu74 libjansson4 libk5crypto3 libkeyutils1 libkrb5-3
1.834 libkrb5support0 libldap-common libldap2 liblua5.4-0 libnghttp2-14
1.834 libperl5.38t64 libpsl5t64 librtmp1 libsasl2-2 libsasl2-modules
1.835 libsasl2-modules-db libsasl2-modules-gssapi-mit libsasl2-modules-gssapi-heimdal
1.835 libsasl2-modules-ldap libsasl2-modules-otp libsasl2-modules-sql perl-doc
1.835 libterm-readline-gnu-perl libterm-readline-perl-perl make libtap-harness-archive-perl
1.918 The following NEW packages will be installed:
1.918 adduser apache2 apache2-bin apache2-data apache2-utils ca-certificates
1.918 krb5-locales libapr1t64 libaprutil1-dbd-sqlite3 libaprutil1-ldap
1.918 libaprutil1t64 libbrotli1 libcurl4t64 libexpat1 libgdbm-compat4t64
1.918 libgdbm6t64 libgssapi-krb5-2 libicu74 libjansson4 libk5crypto3 libkeyutils1
1.918 libkrb5-3 libkrb5support0 libldap-common libldap2 liblua5.4-0 libnghttp2-14
1.919 libperl5.38t64 libpsl5t64 librtmp1 libsasl2-2 libsasl2-modules
1.919 libsasl2-modules-db libsasl2-modules-gssapi-mit libsasl2-modules-gssapi-heimdal
1.919 libsasl2-modules-ldap libsasl2-modules-otp libsasl2-modules-sql perl-doc
1.920 libterm-readline-gnu-perl libterm-readline-perl-perl make libtap-harness-archive-perl
```

1.937 0 upgraded, 43 newly installed, 0 to remove and 2 not upgraded.  
1.937 Need to get 26.3 MB of archives.  
1.937 After this operation, 109 MB of additional disk space will be used.  
1.937 Do you want to continue? [Y/n] Abort.

-----  
  
1 warning found (use docker --debug to expand):  
- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 6)  
Dockerfile:3  
-----  
1 | FROM ubuntu  
2 | RUN apt-get update  
3 | >>> RUN apt-get install apache2  
4 | RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf  
5 | ADD hello.html /var/www/html/  
-----  
ERROR: failed to build: failed to solve: process "/bin/sh -c apt-get install apache2" did not complete successfully:  
exit code: 1

### Dockerfile 수정

```
FROM ubuntu
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update
RUN apt-get install apache2 -y
RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf
ADD hello.html /var/www/html/
CMD apachectl start
```

ENV DEBIAN\_FRONTEND=noninteractive  
[https://www.cyberciti.biz/faq/explain-debian\\_frontend-apt-get-variable-for-ubuntu-debian/](https://www.cyberciti.biz/faq/explain-debian_frontend-apt-get-variable-for-ubuntu-debian/)

### 이미지 빌드

c:\docker\webserver> docker image build -t mywebserver:v1 .

c:\docker\webserver> docker image ls

| IMAGE                | ID           | DISK USAGE | i Info →<br>CONTENT SIZE | U<br>In Use |
|----------------------|--------------|------------|--------------------------|-------------|
| example/echo:latest  | 708142e76f91 | 1.23GB     | 304MB                    |             |
| myanjini/echo:latest | 708142e76f91 | 1.23GB     | 304MB                    |             |
| mywebserver:v1       | 11c24e95b1ad | 373MB      | 107MB                    |             |
| ubuntu:latest        | 84e77dee7d1b | 119MB      | 31.7MB                   | U           |

### 컨테이너 실행

c:\docker\webserver> docker container run -p 9999:80 mywebserver:v1

c:\docker\webserver>                    ⇐ 제어권이 반환 → 컨테이너가 종료

```
c:\docker\webserver> docker container ls -a
```

| CONTAINER ID                            | IMAGE          | COMMAND                  | CREATED        | STATUS     | PORTS          |
|-----------------------------------------|----------------|--------------------------|----------------|------------|----------------|
| NAMES                                   |                |                          |                |            |                |
| f29bc681782f                            | mywebserver:v1 | "/bin/sh -c 'apachec..." | 52 seconds ago | Exited (0) | 51 seconds ago |
| nifty_lewin                             |                |                          |                |            |                |
| fef4707e5a7f                            | ubuntu         | "/bin/bash"              | 25 minutes ago | Up         | 25 minutes     |
| 0.0.0.0:8888->80/tcp, [::]:8888->80/tcp | myweb          |                          |                |            |                |

## Dockerfile 수정

```
FROM ubuntu
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update
RUN apt-get install apache2 -y
RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf
ADD hello.html /var/www/html/
CMD apachectl -D FOREGROUND
```

## 이미지 빌드

```
c:\docker\webserver> docker image build -t mywebserver:v2 .
```

```
[+] Building 0.6s (10/10) FINISHED          docker:desktop-linux
=> [internal] load build definition from Dockerfile      0.1s
=> => transferring dockerfile: 275B                     0.0s
=> [internal] load metadata for docker.io/library/ubuntu:l  0.1s
=> [internal] load .dockerignore                       0.0s
=> => transferring context: 2B                          0.0s
=> [1/5] FROM docker.io/library/ubuntu:latest@sha256:84e77 0.0s
=> => resolve docker.io/library/ubuntu:latest@sha256:84e77 0.0s
=> [internal] load build context                       0.0s
=> => transferring context: 32B                          0.0s
=> CACHED [2/5] RUN apt-get update                     0.0s
=> CACHED [3/5] RUN apt-get install apache2 -y         0.0s
=> CACHED [4/5] RUN echo "ServerName localhost" >> /etc/ 0.0s
=> CACHED [5/5] ADD hello.html /var/www/html/          0.0s
=> exporting to image                                  0.2s
=> => exporting layers                                    0.0s
=> => exporting manifest sha256:ea31b05a8cd5035b0fdf8d5cdd 0.0s
=> => exporting config sha256:b59803b00b86e381da578339753f 0.0s
=> => exporting attestation manifest sha256:62fef05f26db27 0.0s
=> => exporting manifest list sha256:5079c4e02e3c784e727c4 0.0s
=> => naming to docker.io/library/mywebserver:v2         0.0s
=> => unpacking to docker.io/library/mywebserver:v2      0.0s
```

1 warning found (use docker --debug to expand):

- JSONArgsRecommended: JSON arguments recommended for CMD to prevent unintended behavior related to OS signals (line 7)

```
c:\docker\webserver> docker image ls
```

| IMAGE                | ID           | DISK USAGE | i Info → U In Use  |
|----------------------|--------------|------------|--------------------|
|                      |              |            | CONTENT SIZE EXTRA |
| example/echo:latest  | 708142e76f91 | 1.23GB     | 304MB              |
| myanjini/echo:latest | 708142e76f91 | 1.23GB     | 304MB              |
| mywebserver:v1       | 11c24e95b1ad | 373MB      | 107MB U            |
| mywebserver:v2       | 5079c4e02e3c | 373MB      | 107MB              |
| ubuntu:latest        | 84e77dee7d1b | 119MB      | 31.7MB U           |

## 컨테이너 실행

```
c:\docker\webserver> docker container run -p 7777:80 mywebserver:v2
```

⇐ attach 모드로 실행했기 때문에 컨테이너 내부의 출력이 보여지고 있음

## detach 모드로 전환해서 컨테이너의 동작을 확인

```
c:\docker\webserver> docker container run -p 7777:80 mywebserver:v2
```

Ctrl + c 를 세 번 입력

got 3 SIGTERM/SIGINTs, forcefully exiting

```
c:\docker\webserver> curl http://localhost:7777/hello.html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello Docker !!!!</h1>
</body>
</html>
```

## Dockerfile 작성 시 유의 사항 2 - 도커 이미지 크기가 커지지 않도록 유의

- 이미지 크기를 줄이기 위해서는 애플리케이션 실행에 꼭 필요한 아티팩트(부산물)와 런타임만 포함  
⇒ PUSH, PULL할 때 불필요한 시간과 대역폭 사용을 방지  
⇒ 잠재적 보안 취약점을 가지고 있는 도구가 포함되는 것을 방지 (예: JDK에 포함된 보안약점)
- 이미지 빌드 및 배포 시간 단축을 위해서는 이미지 크기를 최대한 작게 유지하는 것이 중요
- Dockerfile의 모든 명령어는 빌드 과정에서 레이어를 생성하고, 생성된 레이어는 컨테이너에서도 유지
- 실제 컨테이너에서 사용하지 못 하는 파일(또는 디렉터리)이 이미지 레이어로 존재하면 의미 없는 저장 공간만 차지
- 방법1. Dockerfile을 작성할 때 &&를 이용해서 여러 RUN 명령어를 하나로 묶어서 실행하도록 작성
- 방법2. 다단계 도커 빌드
  - Docker 17.05에는 다단계 빌드를 만들 수 있는 새로운 기능이 추가

- 빌드 프로세스 중에 여러 중간 도커 이미지를 만들고 한 단계에서 다른 단계로 필수 아티팩트만 선택적으로 복사
- 다단계 빌드가 제공되기 전에는 최적화된 이미지 생성을 위해 **빌더 패턴**을 사용

## #1 레이어 최소화 필요성 실습

Dockerfile의 각 라인은 레이어 형태로 이미지에 저장되므로, 지시문을 결합해서 최적화하는 것이 필요

```
c:\docker\webserver> mkdir ../layer && cd ../layer
```

```
c:\docker\layer> code .
```

명령어마다 **RUN** 지시문을 사용하는 경우

C:\docker\layer\Dockerfile

```
FROM ubuntu
RUN mkdir /test
RUN fallocate -l 100m /test/dummy
RUN rm /test/dummy
```

```
c:\docker\layer> docker image build -t falloc_100m .
```

```
c:\docker\layer> docker image ls
```

| IMAGE                     | ID                  | DISK USAGE   | i Info →<br>CONTENT SIZE | U In Use<br>EXTRA |
|---------------------------|---------------------|--------------|--------------------------|-------------------|
| example/echo:latest       | 708142e76f91        | 1.23GB       | 304MB                    |                   |
| <b>falloc_100m:latest</b> | <b>7474b3c86e57</b> | <b>222MB</b> | <b>29.8MB</b>            |                   |
| myanjini/echo:latest      | 708142e76f91        | 1.23GB       | 304MB                    |                   |
| mywebserver:v1            | 11c24e95b1ad        | 373MB        | 107MB                    | U                 |
| mywebserver:v2            | 5079c4e02e3c        | 373MB        | 107MB                    | U                 |
| <b>ubuntu:latest</b>      | <b>84e77dee7d1b</b> | <b>119MB</b> | <b>31.7MB</b>            | <b>U</b>          |

하나의 **RUN** 지시문에 여러 개의 명령어를 결합해서 사용

```
FROM ubuntu
RUN mkdir /test && \
  fallocate -l 100m /test/dummy && \
  rm /test/dummy
```

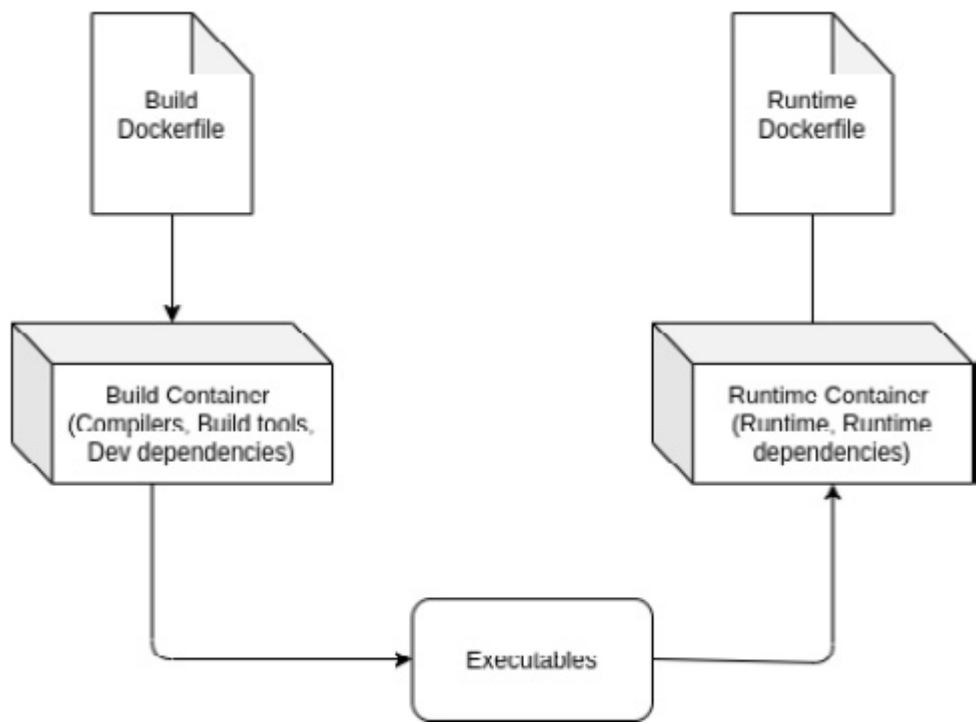
```
c:\docker\layer> docker image build -t falloc_0m .
```

```
c:\docker\layer> docker image ls
```

| IMAGE                   | ID                  | DISK USAGE   | i Info →<br>CONTENT SIZE | U In Use<br>EXTRA |
|-------------------------|---------------------|--------------|--------------------------|-------------------|
| example/echo:latest     | 708142e76f91        | 1.23GB       | 304MB                    |                   |
| <b>falloc_0m:latest</b> | <b>39997e24070c</b> | <b>117MB</b> | <b>29.7MB</b>            |                   |
| falloc_100m:latest      | 7474b3c86e57        | 222MB        | 29.8MB                   |                   |
| myanjini/echo:latest    | 708142e76f91        | 1.23GB       | 304MB                    |                   |
| mywebserver:v1          | 11c24e95b1ad        | 373MB        | 107MB                    | U                 |
| mywebserver:v2          | 5079c4e02e3c        | 373MB        | 107MB                    | U                 |
| <b>ubuntu:latest</b>    | <b>84e77dee7d1b</b> | <b>119MB</b> | <b>31.7MB</b>            | <b>U</b>          |

## #2 빌더 패턴

- 최적의 도커 이미지 생성을 위해 사용하는 방법
- 두 개의 도커 이미지를 사용
  - 각각의 도커 이미지는 필수 아티팩트(부산물)만 포함
  - 첫번째 도커 이미지 ⇒ Builder ⇒ 소스 코드를 실행 파일로 만들기 위한 빌드 환경 ⇒ 빌드에 필요한 컴파일러, 빌드 도구, 개발 종속성을 포함
  - 두번째 도커 이미지 ⇒ Runtime ⇒ 첫번째 도커 컨테이너가 생성한 실행 파일을 실행하기 위한 런타임 환경 ⇒ 실행 파일, 종속성 및 런타임 도구만 포함
  - 첫번째 도커 컨테이너가 생성한 실행 파일을 두번째 도커 컨테이너로 전달하는 스크립트가 필요



## #3 다단계 도커 빌드

- Docker 17.05에 새로 추가된 기능
- 하나의 Dockerfile에 여러 개의 FROM 문을 사용해 빌드 단계를 정의하고, --from 플래그를 사용해 각 단계에서 생성된 아티팩트를 참조할 수 있도록 하는 구문
- 각 단계는 0 부터 순서대로 부여되는 번호 또는 AS 절을 사용해 부여한 별칭을 이용해서 참조
- 도커 호스트에 복사하지 않고 도커 이미지간에 실행 파일을 복사할 수 있으므로 실행 파일을 도커 이미지에 복사하기 위한 쉘 스크립트를 필요로 하지 않음

```
c:\docker\layer> mkdir ../multi-stage && cd ../multi-stage
```

```
c:\docker\multi-stage> code .
```

```
C:\docker\multi-stage-build\helloworld.go
```

```
package main
import "fmt"
func main() {
    fmt.Println("Hello World")
}
```



c:\docker\multi-stage\Dockerfile

```
FROM      golang
WORKDIR   /myapp
COPY      helloworld.go .
RUN       go env -w GO111MODULE=auto
RUN       go build -o helloworld.exe .

FROM      alpine
WORKDIR   /myapp
COPY      --from=0 /myapp/helloworld.exe .
ENTRYPOINT ["./helloworld.exe"]
```

c:\docker\multi-stage> docker image build -t multi-stage:v1 .

c:\docker\multi-stage> docker image ls

|                      |              |            | i Info →     | U In Use |
|----------------------|--------------|------------|--------------|----------|
| IMAGE                | ID           | DISK USAGE | CONTENT SIZE | EXTRA    |
| example/echo:latest  | 708142e76f91 | 1.23GB     | 304MB        |          |
| falloc_0m:latest     | 39997e24070c | 117MB      | 29.7MB       |          |
| falloc_100m:latest   | 7474b3c86e57 | 222MB      | 29.8MB       |          |
| multi-stage:v1       | dec9e1c06511 | 16.8MB     | 5.3MB        |          |
| myanjini/echo:latest | 708142e76f91 | 1.23GB     | 304MB        |          |
| mywebserver:v1       | 11c24e95b1ad | 373MB      | 107MB        | U        |
| mywebserver:v2       | 5079c4e02e3c | 373MB      | 107MB        | U        |
| ubuntu:latest        | 84e77dee7d1b | 119MB      | 31.7MB       | U        |

Dockerfile을 수정 ⇒ 별명을 부여하는 방식으로

```
FROM      golang      AS Builder
WORKDIR   /myapp
COPY      helloworld.go .
RUN       go env -w GO111MODULE=auto
RUN       go build -o helloworld.exe .

FROM      alpine      AS Runtime
WORKDIR   /myapp
COPY      --from=Builder /myapp/helloworld.exe .
ENTRYPOINT ["./helloworld.exe"]
```

# 프로그램 설치

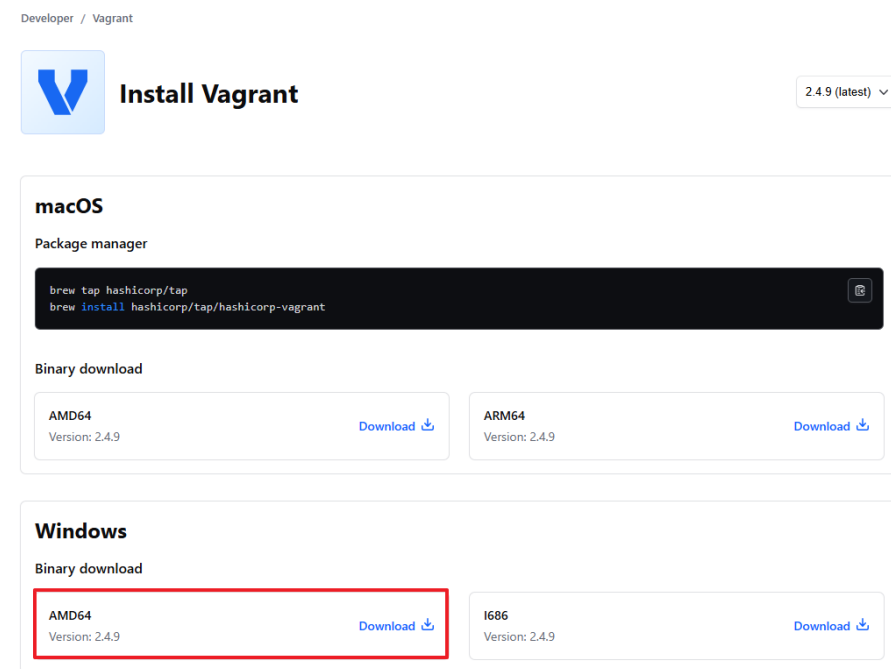
## VirtualBox

<https://www.virtualbox.org/wiki/Downloads>



## vagrant (리부팅 필요)

<https://developer.hashicorp.com/vagrant/install>



## 쿠버네티스(kubernetes)

<https://kubernetes.io/docs/home/>

Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation ([CNCF](https://www.cncf.io/)).

컨테이너 기반의 애플리케이션을 개발하고 배포할 수 있도록 설계된 오픈 소스 플랫폼  
컨테이너 오케스트레이션 도구의 사실상 표준(de facto standard)

~~~~~

- | - Orchestration
- | - 복잡한 단계를 관리하고 요소들의 유기적인 관계를 미리 정의해 손쉽게 사용하도록 서비스를 제공
- | - 다수의 컨테이너를 유기적으로 연결, 실행, 종료할 뿐만 아니라 상태를 추적하고 보존하는 등
- | 컨테이너를 안정적으로 사용할 수 있게 만들어주는 것
- +--- 하나 이상의 목적을 위해 독립적으로 작동하는 프로세스

컨테이너 인프라 환경

- 하나의 리눅스 운영체제 커널에서 여러 개의 컨테이너가 격리된 상태로 실행되는 인프라 환경
- 가상화 상태에서 동작하는 프로세스 보다 가볍고 빠르게 동작
- 컨테이너, 컨테이너 관리, 개발 환경 구성 및 배포 자동화, 모니터링으로 구성
 - 컨테이너 도구 : 도커(docker), 컨테이너디(containerd), 크라이오(CRI-O), 파드맨(podman)
 - 컨테이너 관리 도구 : 쿠버네티스(kubernetes), 도커 스웜(docker swarm), 메소스(mesos), 노마드(nomad)
 - 지속적 통합과 배포를 위한 도구 : 젠킨스(Jenkins), 뱀부(Bamboo), 깃허브 액션(Github Action), 팀시티(Teamcity)
 - 모니터링 데이터 수집 도구 : 프로메테우스, 데이터독(DataDog), 인플럭스DB(InfluxDB), 뉴렐릭(New Relic)
 - 데이터를 시각화하는 도구 : 그라파나, 키바나(Kibana), 크로노그래프(Chronograf)

쿠버네티스 컴포넌트

<https://kubernetes.io/docs/concepts/overview/components/>

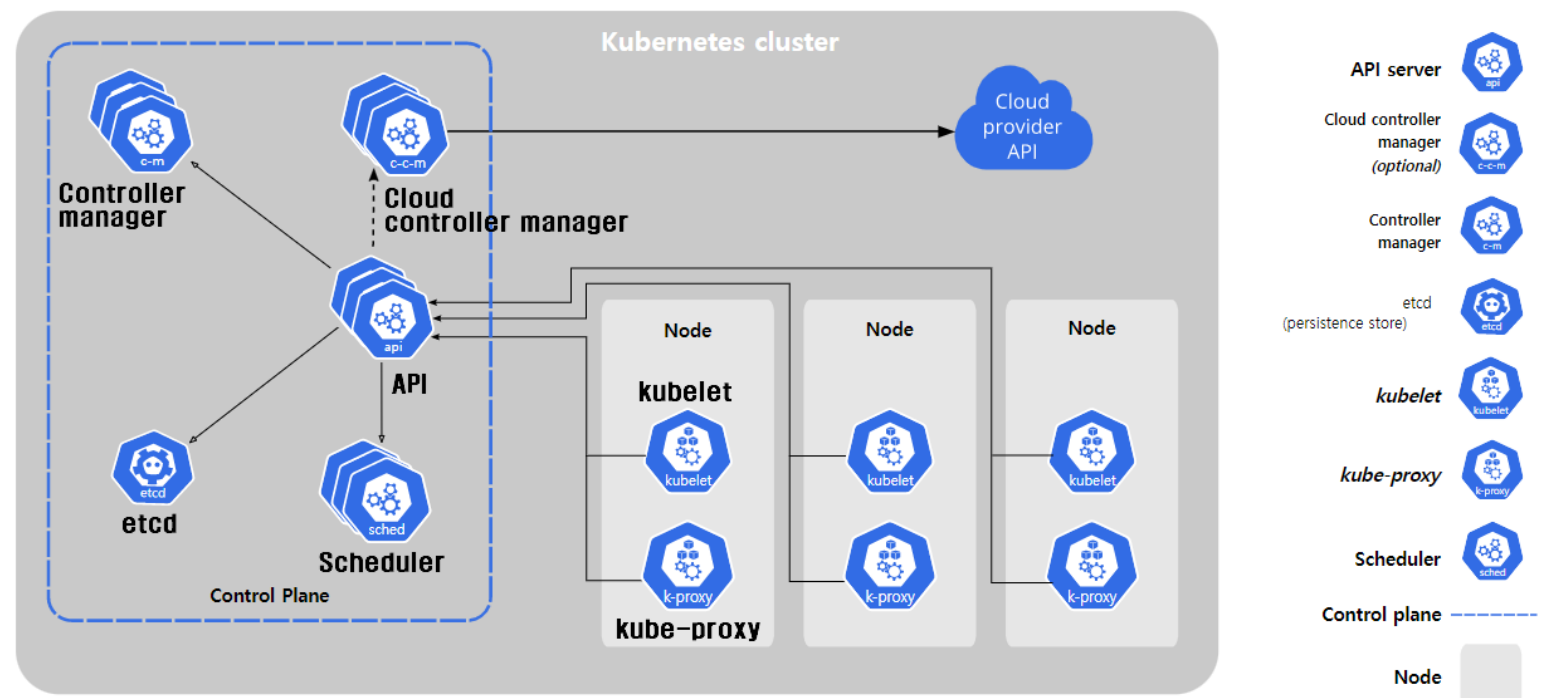
<https://kubernetes.io/ko/docs/concepts/overview/components/>

master node - 전체 쿠버네티스 시스템을 제어하고 관리하는 쿠버네티스 **컨트롤 플레인을 실행하는 노드**

worker node - 실제 배포되는 컨테이너 애플리케이션을 실행하는 노드 ⇒ **컴퓨팅 머신**

쿠버네티스 클러스터는 컨테이너화된 애플리케이션을 실행하는 노드라고 하는 워커 머신의 집합으로, 모든 클러스터는 최소한 개의 워커 노드를 가진다.

워커 노드는 애플리케이션 워크로드의 구성요소인 파드를 호스트한다. 컨트롤 플레인은 워커 노드와 클러스터 내 파드를 관리한다. 프로덕션 환경에서는 일반적으로 컨트롤 플레인이 여러 컴퓨터에 걸쳐 실행되고, 클러스터는 일반적으로 여러 노드를 실행하므로 내결함성과 고가용성이 제공된다.



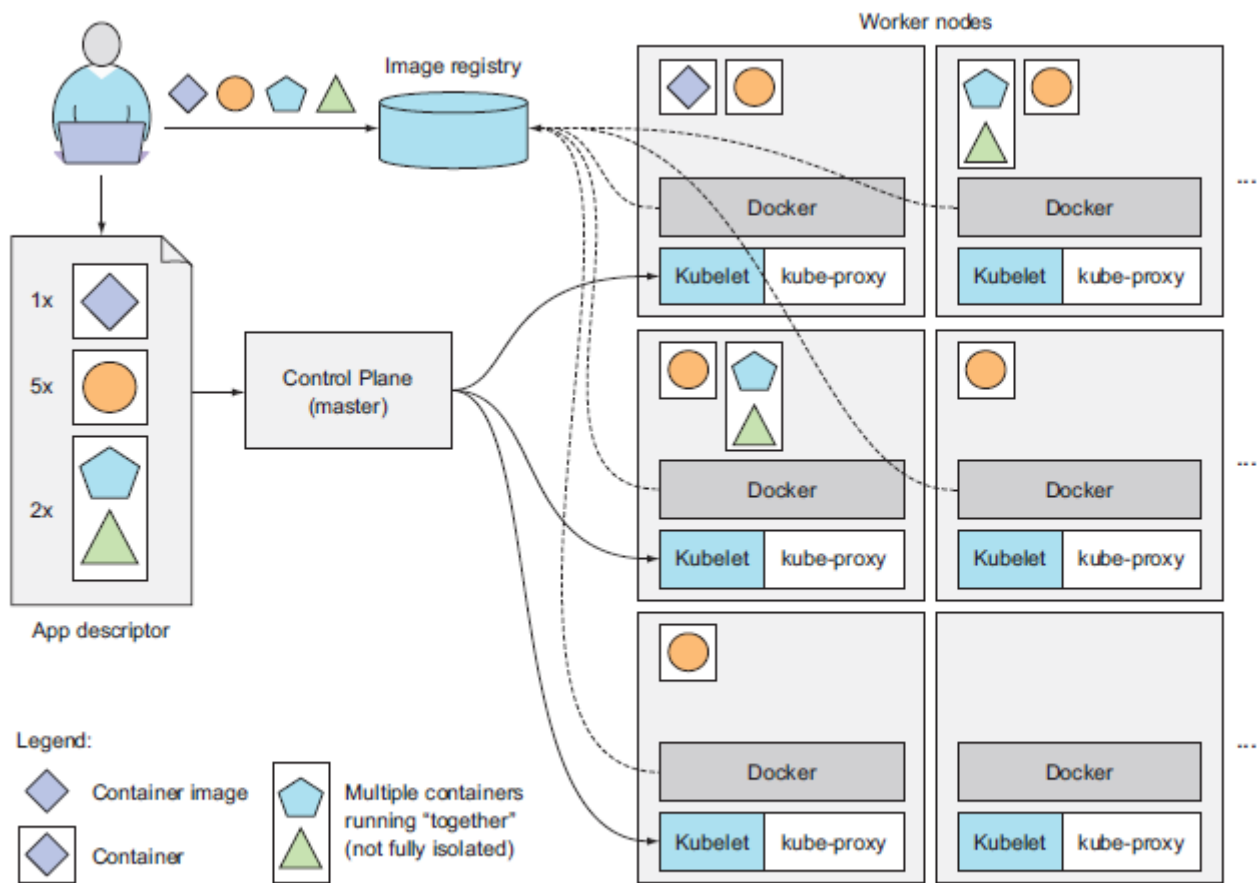
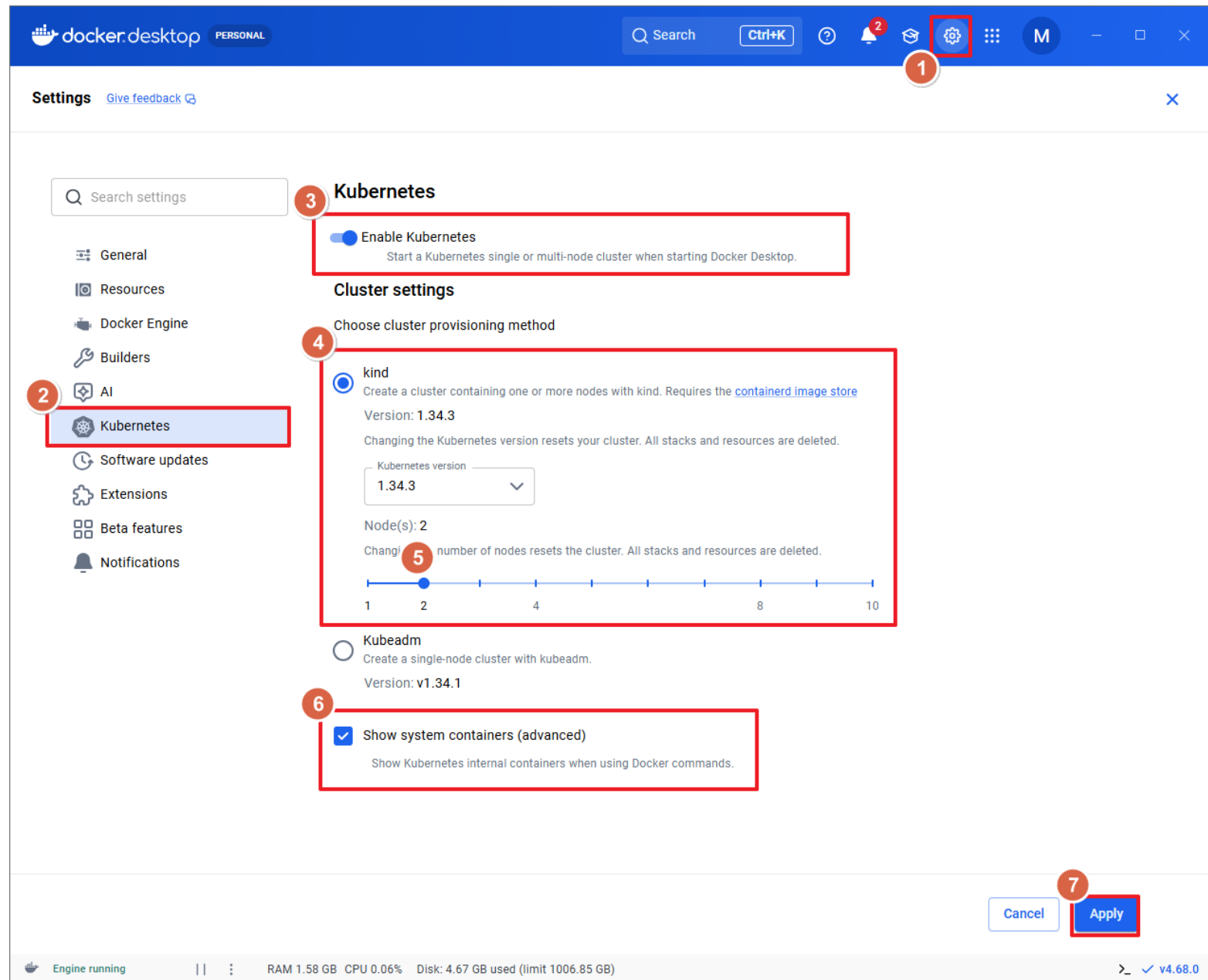


Figure 1.10 A basic overview of the Kubernetes architecture and an application running on top of it

Docker Desktop에서 kubernetes 클러스터 생성



실행 확인

```
C:\Users\myanj> kubectl version
```

```
Client Version: v1.34.1
```

```
Kustomize Version: v5.7.1
```

```
Server Version: v1.34.3
```

```
C:\Users\myanj> kubectl version --output=json
```

```
{
  "clientVersion": {
    "major": "1",
    "minor": "34",
    "gitVersion": "v1.34.1",
    "gitCommit": "93248f9ae092f571eb870b7664c534bfc7d00f03",
    "gitTreeState": "clean",
    "buildDate": "2025-09-09T19:44:50Z",
    "goVersion": "go1.24.6",
```

```
    "compiler": "gc",
    "platform": "windows/amd64"
  },
  "kustomizeVersion": "v5.7.1",
  "serverVersion": {
    "major": "1",
    "minor": "34",
    "emulationMajor": "1",
    "emulationMinor": "34",
    "minCompatibilityMajor": "1",
    "minCompatibilityMinor": "33",
    "gitVersion": "v1.34.3",
    "gitCommit": "df11db1c0f08fab3c0baee1e5ce6efbf816af7f1",
    "gitTreeState": "clean",
    "buildDate": "2025-12-09T14:59:13Z",
    "goVersion": "go1.24.11",
    "compiler": "gc",
    "platform": "linux/amd64"
  }
}
```

노드 정보를 조회

C:\Users\myanj> kubectl get nodes

NAME	STATUS	ROLES	AGE	VERSION
desktop-control-plane	Ready	control-plane	4m52s	v1.34.3
desktop-worker	Ready	<none>	4m38s	v1.34.3

노드 갯수를 변경한 후 다시 노드 정보를 조회

C:\Users\myanj> kubectl get nodes

NAME	STATUS	ROLES	AGE	VERSION
desktop-control-plane	Ready	control-plane	45s	v1.34.3
desktop-worker	Ready	<none>	31s	v1.34.3
desktop-worker2	Ready	<none>	32s	v1.34.3

C:\Users\myanj> docker container ls

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c4ecad73396b	docker/desktop-cloud-provider-kind:v0.5.0	"/bin/cloud-provider..."	13 minutes ago	Up 13 minutes	2375-2376/tcp	kind-cloud-provider
cf4d6336bd37	docker/desktop-containerd-registry-mirror:v0.0.3	"/bin/registry"	13 minutes ago	Up 13 minutes (healthy)		kind-registry-mirror
a91436db2fb6	kindest/node:v1.34.3	"/usr/local/bin/entr..."	14 minutes ago	Up 14 minutes		desktop-worker2
74806e04192b	kindest/node:v1.34.3	"/usr/local/bin/entr..."	14 minutes ago	Up 14 minutes	127.0.0.1:50069->6443/tcp	desktop-control-plane
35f76180ff82	kindest/node:v1.34.3	"/usr/local/bin/entr..."	14 minutes ago	Up 14 minutes		desktop-worker

애플리케이션 배포 및 확인

디플로이먼트 생성

C:\Users\myanj> kubectl create deployment mynginx --image=nginx
deployment.apps/mynginx created

```
C:\Users\myanj> kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/mynginx-645865c456-jrkn8	0/1	ImagePullBackOff	0	9m19s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	38m

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/mynginx	0/1	1	0	9m19s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/mynginx-645865c456	1	1	0	9m19s

```
C:\Users\myanj> kubectl delete deployment mynginx
deployment.apps "mynginx" deleted from default namespace
```

```
C:\Users\myanj> kubectl get all
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	44m

```
C:\Users\myanj> docker image pull nginx
```

```
C:\Users\myanj> kubectl create deployment mynginx --image=nginx
deployment.apps/mynginx created
```

```
C:\Users\myanj> kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/mynginx-645865c456-6bcnv	1/1	Running	0	16s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	49s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/mynginx	1/1	1	1	16s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/mynginx-645865c456	1	1	1	16s

```
C:\Users\myanj> kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
mynginx-645865c456-6bcnv	1/1	Running	0	2m20s

```
C:\Users\myanj> kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE
mynginx-645865c456-6bcnv	1/1	Running	0	2m29s	10.244.2.2	desktop-worker	<none>
<none>							

NodePort 타입의 서비스를 생성

```
kubectl expose (-f FILENAME | TYPE NAME) [--port=port] [--protocol=TCP|UDP|SCTP] [--target-port=number-or-name]
[--name=name] [--external-ip=external-ip-of-service] [--type=type] [options]
```

⇒ Kubernetes 리소스(예: Pod, Deployment 등)에 대해 Service 객체를 생성해서 네트워크 접근이 가능하도록 설정

```
C:\Users\myanj> kubectl expose deployment mynginx --type=NodePort --port=80 --target-port=80 --name=myweb
service/myweb exposed
```

```
C:\Users\myanj> kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/mynginx-645865c456-6bcnv	1/1	Running	0	12m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	12m
service/myweb	NodePort	10.96.28.193	<none>	80:30720/TCP	23s

~~~~~  
targetPort:nodePort  
노드의 nodePort로 접근하면 선택된 노드의 80 포트로 전달

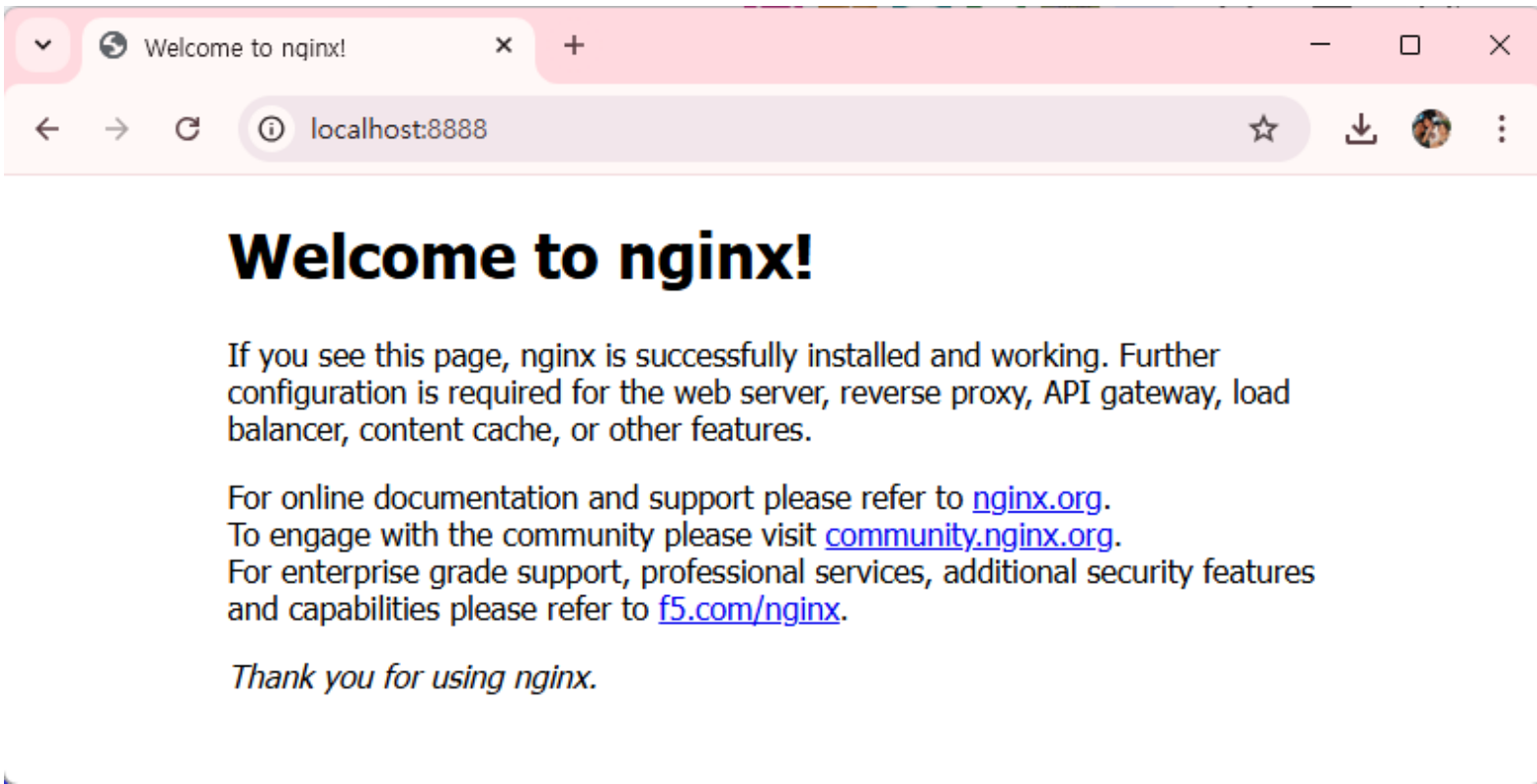
| NAME                    | READY | UP-TO-DATE | AVAILABLE | AGE |
|-------------------------|-------|------------|-----------|-----|
| deployment.apps/mynginx | 1/1   | 1          | 1         | 12m |

| NAME                               | DESIRED | CURRENT | READY | AGE |
|------------------------------------|---------|---------|-------|-----|
| replicaset.apps/mynginx-645865c456 | 1       | 1       | 1     | 12m |

포트 포워딩

```
kubectl port-forward TYPE/NAME [options] [LOCAL_PORT:]REMOTE_PORT [...[LOCAL_PORT_N:]REMOTE_PORT_N]
```

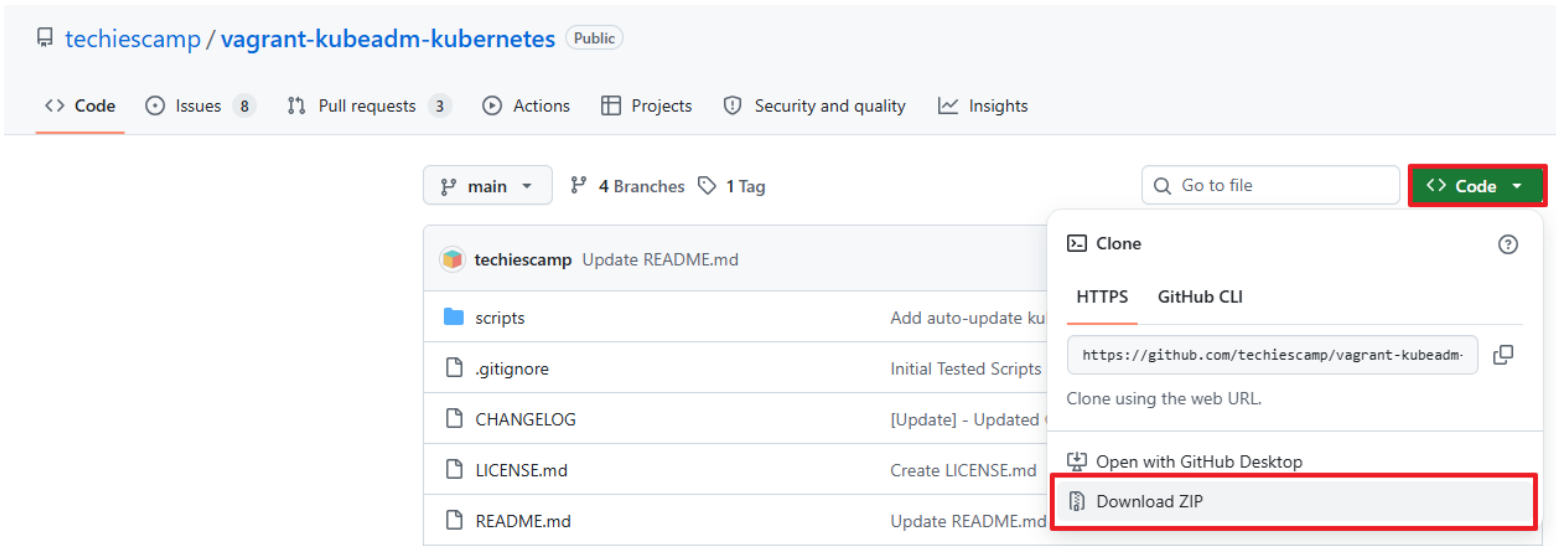
```
C:\Users\myanj> kubectl port-forward service/myweb 8888:80
Forwarding from 127.0.0.1:8888 -> 80
Forwarding from [::1]:8888 -> 80
```



## Automated Kubernetes Cluster Setup on Vagrant

<https://devopscube.com/kubernetes-cluster-vagrant/>

<https://github.com/techiescamp/vagrant-kubeadm-kubernetes>



내려받은 zip 파일을 C:\ 아래에 압축해제

```
C:\Users\myanj> cd c:\vagrant-kubeadm-kubernetes-main
```

```
c:\vagrant-kubeadm-kubernetes-main> code .
```

```
c:\vagrant-kubeadm-kubernetes-main> vagrant up
```

```
c:\vagrant-kubeadm-kubernetes-main> vagrant status
```

Current machine states:

|              |                      |
|--------------|----------------------|
| controlplane | running (virtualbox) |
| node01       | running (virtualbox) |
| node02       | running (virtualbox) |

This environment represents multiple VMs. The VMs are all listed above with their current state. For more information about a specific VM, run `vagrant status NAME`.