Docker Compose

Docker Compose란 무엇인가 ?

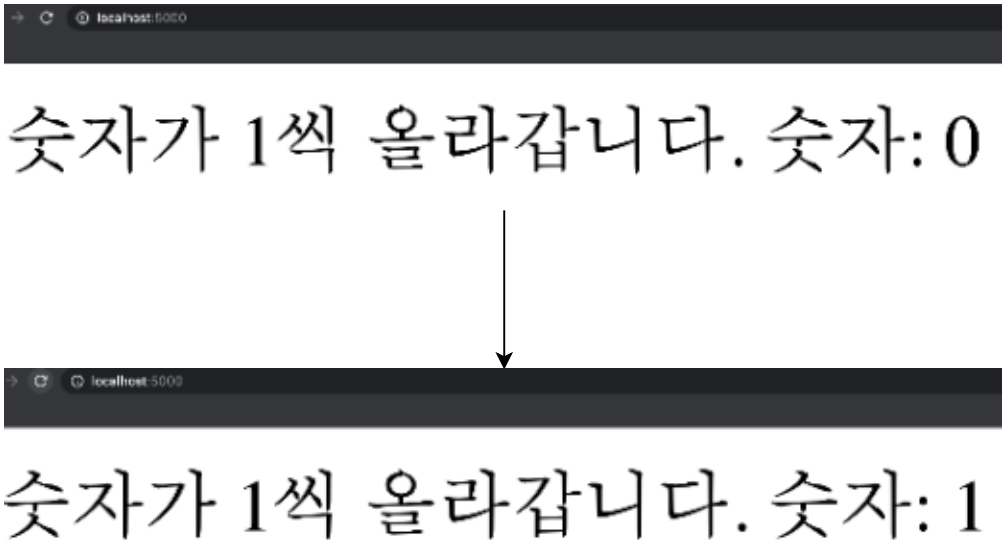
이제부터 docker compose란 것을 배우겠습니다.
우선 이것이 무엇인지 알아보겠습니다.

docker compose는 다중 컨테이너 도커 애플리케이션을 정의하고 실행하기 위한 도구입니다

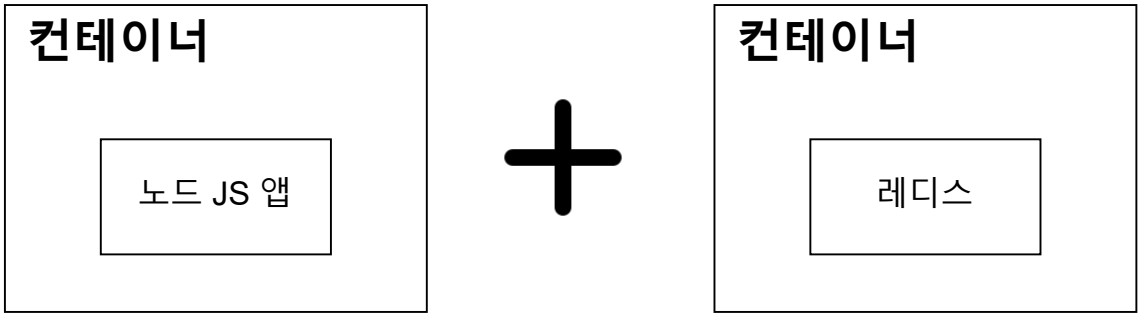
이렇게만 정의만 봐서는 무엇인지 잘 감이 안 올 것입니다.
그래서 이제부터는 이 Compose를 이용해서
새로운 애플리케이션을 만들면서 Docker Compose에 대해 더욱 배워보는 시간을 가져보겠습니다.

Docker Compose를 이용해서 만들 어플리케이션 소개

페이지를 리프레쉬했을 때 숫자 0부터 1씩 계속 올라가는
간단한 앱을 만들어보면서 Docker Compose를 배워보겠습니다.



이 애플리케이션의 전체적인 구조



어플리케이션 소스 작성하기

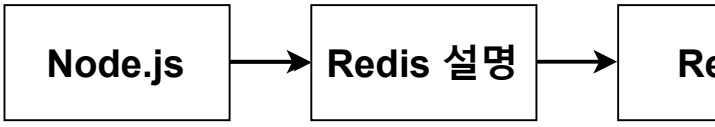
먼저 애플리케이션을 위한 소스코드를 작성해보겠습니다.

우선 새로운 프로젝트를 만들 새로운 폴더를 하나 더 만듭니다.

package.json → NPM INIT **server.js**

기본적인 노드 부분 완성하기

레디스 부분 완성하기



그전에 간단하게 Redis 설명 !

레디스란 ?

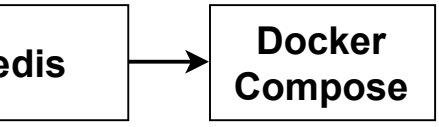
Redis(REmote Dictionary Server)는 메모리 기반의 키-값 구조
데이터 관리 시스템이며, 모든 데이터를 메모리에 저장하고

레디스를 쓰는 이유?

메모리에 저장을 하기 때문에 Mysql 같은 데이터베이스에 데이터를 저장하는 것과
데이터를 불러올 때 훨씬 빠르게 처리할 수가 있으며,
비록 메모리에 저장하지만 영속적으로도 보관이 가능하다.
그래서 서버를 재부팅해도 데이터를 유지할 수 있는 장점이 있다.

Node.js 환경에서 Redis 사용 방법

- 먼저 redis-server를 작동시켜주셔야 합니다.
- 그리고 redis 모듈을 다운받습니다.
- 레디스 모듈을 받은 후 레디스 클라이언트를 생성하기 위해서 Redis에서 제공하는 createClient() 함수를 이용해서 redis.createClient로


edis
Docker
Compose

Redis에서 제공하는 createClient() 함수를 이용해서 redis.createClient로 레디스 클라이언트를 생성해준다

- 하지만 여기서 redis server가 작동하는 곳과 Node.js 앱이 작동하는 곳이 다른 곳이라면 host 인자와 port 인자를 명시해주어야 한다.

  예를 들자면)

```
const client = redis.createClient({
host: "https://redis-server.com",
port: 6379
});
```

만약 Redis 서버가 작동하는 곳이 redis-server.com이라면 Host 옵션을 위에 처럼 주면 된다. 그리고 레디스의 기본 포트는 6379번 입니다.

도커 환경에서 레디스 클라이언트 생성 시 주의사항

보통 도커를 사용하지 않는 환경에서는 Redis 서버가 작동되고 있는 곳의 host 옵션을
URL로 위에 처럼 주면 되지만, 도커 Compose를 사용할 때는 host 옵션을 docker-compose.yml 파일에 명시한 컨테이너 이름으로 주면 된다.

```
const client = redis.createClient({
host: "redis-server",
port: 6379
});
```

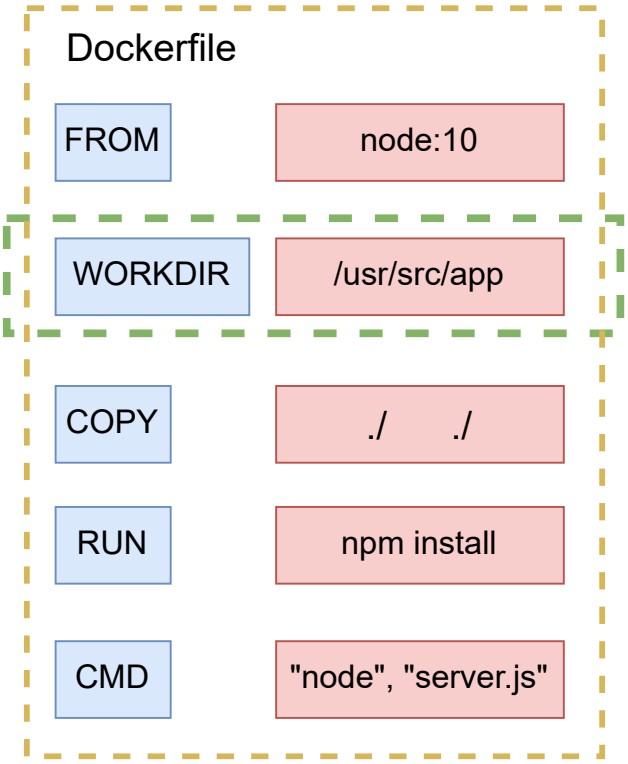
실제로 노드 앱에 레디스로 간단한 기능 구현하기

페이지를 리프레시할 때 숫자 0부터 1씩 계속 올라가는 기능

```
client.set("number", 0);

// App
const app = express();
app.get('/', (req, res) => {

  client.get("number", (err, number) => {
    res.send("Number is going up " + number);
    client.set("number", parseInt(number) + 1);
  });

});
```

Dockerfile 작성하기

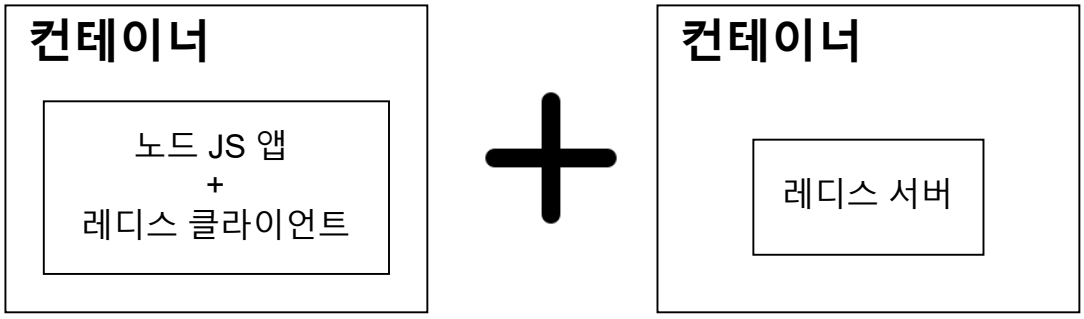
Nodejs를 위한 이미지를 만들기 위해서
Dockerfile을 작성하는 것이기에
저번에 노드 js 앱을 위한 Dockerfile과
똑같이 만들어 주면 됩니다.




```
FROM node:10

WORKDIR /usr/src/app

COPY ./ ./

RUN npm install

CMD [ "node", "server.js" ]
```

Docker Containers간 통신할 때 나타나는 에러

이제 어플리케이션 소스와 도커 파일까지 작성했으니
실제로 어플을 실행해 보겠습니다.
우선 어플이 어떤 식으로 실행이 되는지 살펴보겠습니다.

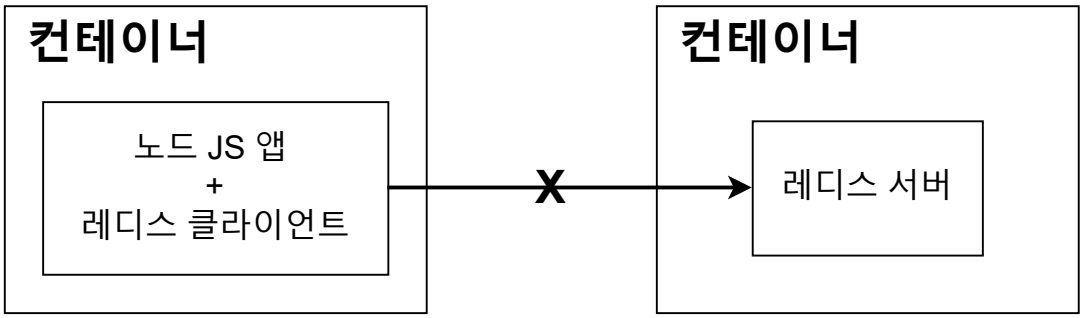


이제는 컨테이너를 하나하나 실행해 보겠습니다.
먼저 레디스 클라이언트가 작동하려면
레디스 서버가 켜져있어야 하기 때문에
먼저 레디스 서버를 위한 컨테이너를 실행하고
노드 js를 위한 컨테이너를 실행하겠습니다.

docker build -t 도커 아이디/앱 이름.
docker run 이미지 이름

```
Error: Redis connection to redis-server:6379 failed -
getaddrinfo ENOTFOUND redis-server redis-server:6379
    at GetAddrInfoReqWrap.onlookup [as oncomplete] (dns.js:56:26)
Emitted 'error' event at:
    at RedisClient.on_error (/usr/src/app/node_modules/redis/index.js:406:14)
    at Socket.<anonymous> (/usr/src/app/node_modules/redis/index.js:279:14)
    at Socket.emit (events.js:198:13)
    at emitErrorNT (internal/streams/destroy.js:91:8)
    at emitErrorAndCloseNT (internal/streams/destroy.js:59:3)
    at process._tickCallback (internal/process/next_tick.js:63:19)
```

왜 이런 에러가 일어날까요?



서로 다른 컨테이너에 있는데 이렇게 컨테이너 사이에는 아무런 설정 없이는 접근을 할 수 없기에
노드 JS 앱에서 레디스 서버에 접근을 할 수 없습니다.

그러면 어떻게 컨테이너 사이에 통신을 할 수 있게 해 줄까요?

멀티 컨테이너 상황에서 쉽게 네트워크를 연결시켜주기 위해서 Docker Compose를 이용하면 됩니다.

도커 Compose 파일 작성하기

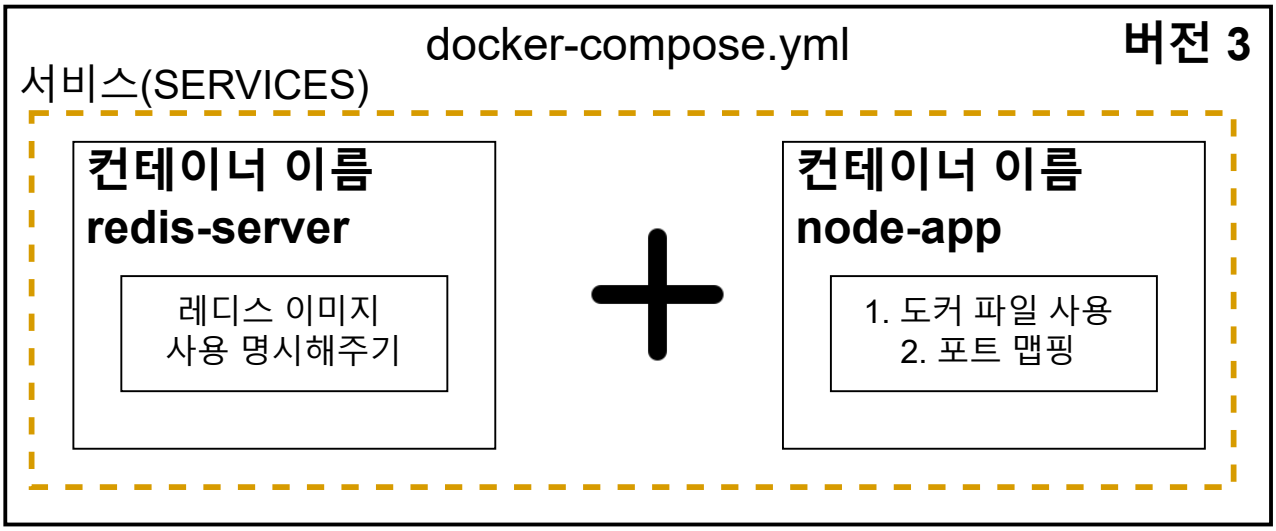
도커 컴포즈가 컨테이너 사이에 네트워크를 연결시켜 준다는 걸 알았습니다.
그러면 본격적으로 도커 컴포즈 파일을 작성해겠습니다.

먼저는 docker compose 파일을 생성해볼게요.
docker-compose.yml

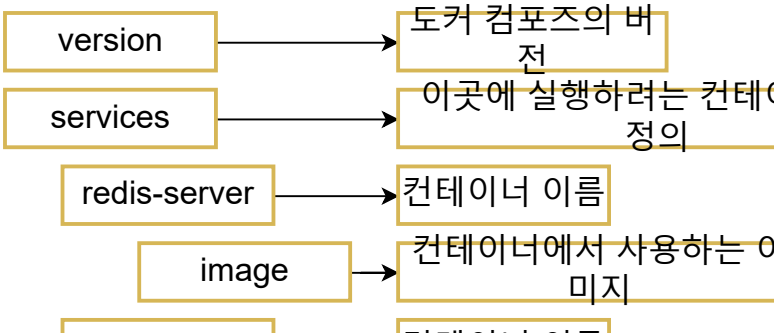
Docker Compose 파일은 확장자가 yaml이나 yml인데 이것은 무엇인가요?

YAML ain't markup language의 약자이며,
일반적으로 구성 파일 및 데이터가 저장되거나 전송되는 응용 프로그램에서 사용되고
원래는 XML이나 json 포맷으로 많이 쓰였지만,
좀 더 사람이 읽기 쉬운 포맷으로 나타난 게 yaml입니다.

이 어플을 위한 docker-compose파일 구조




```yaml
version: "3"
services:
  redis-server:
    image: "redis"
  node-app:
```



도커 컴포즈가 컨테이너 사이에 네트워크를 연결 시켜 준다는 걸 알았습니다.
그러면 본격적으로 도커 컴포즈 파일을 작성해겠습니다.

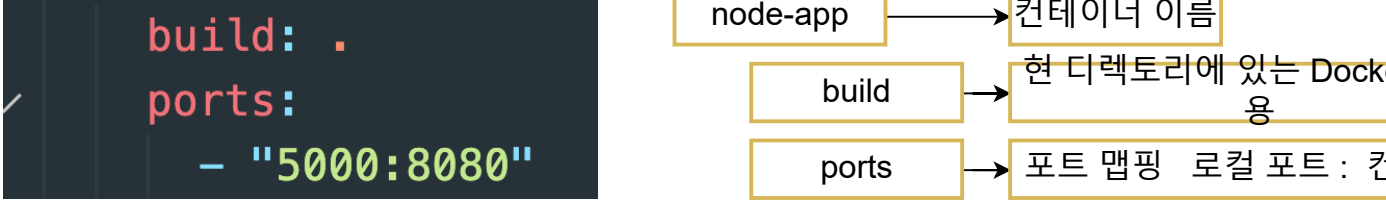
도커 클라이언트에 넣었던 명령어들

docker build -t johnahn/node-app .

docker run -p 5000:8000 johnahn/node-app

이너들을

아



이렇게 도커 컴포즈를 다 작성한 후에
도커 컴포즈를 이용해서 앱을 실행해보겠습니다.

docker-compose	up

erfile 사

컨테이너 포트

이제는 실제로 docker-compose를 이용하여 앱을 실행해보자.

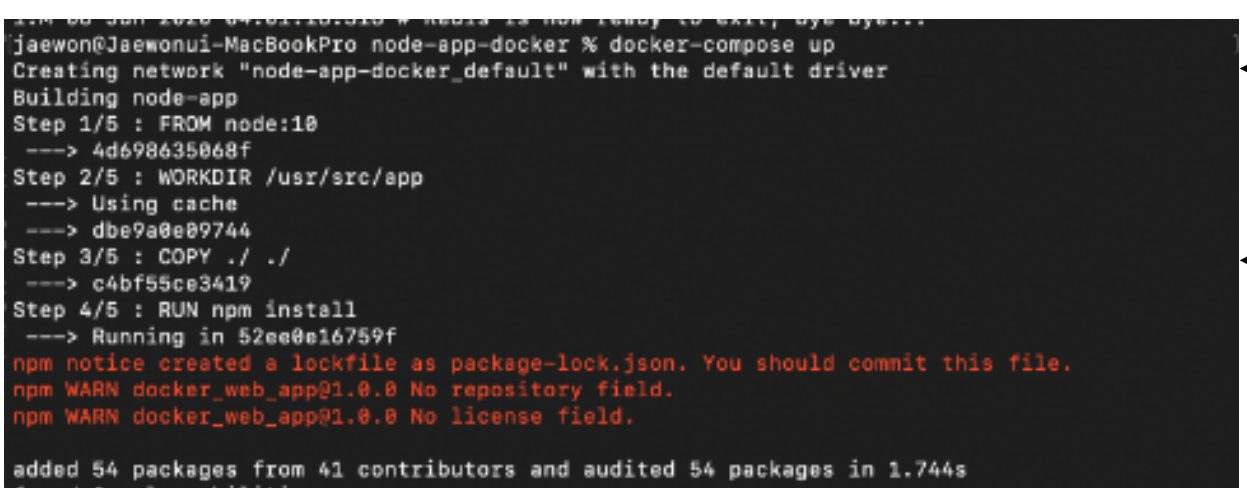
실행할때는 docker-compose up 명령어를 사용하면 된다.

Docker Compose를 이용하여 앱을 실행 할때는
지금까지 앱을 실행할때 처럼 복잡하지 않고
이미 모든 정보들이 compose 파일 안에 들어있기에
그냥 docker compose up 해주면 된다.

하지만 여기서 몇가지 기억하면 좋은 점이 있다.

docker-compose up 과 비슷한 명령어 몇개가 있는데 상황에 따라 쓰면 좋다

docker-compose build	이미지를 빌드하기만 하며, 컨테이너를
docker-compose up	이미지가 존재하지 않을 경우에만 빌드 시작한다.
docker-compose up --build	필요치 않을때도 강제로 이미지를 빌드 시작한다.
docker-compose up --no-build	이미지 빌드 없이, 컨테이너를 시작한다 실패)



시작하지는 않는다

조하며, 컨테이너를

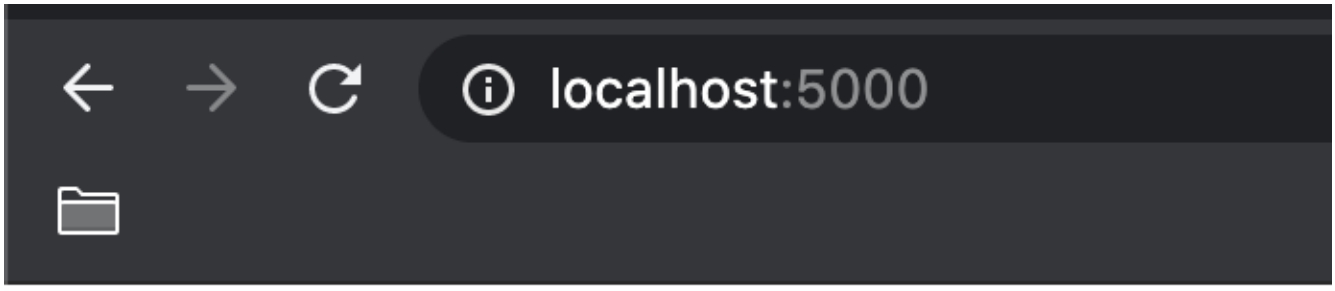
조하며, 컨테이너를

. (이미지가 없을시

ode-app 컨테이너와
dis-server 컨테이너 통
을 위한 네트워크 생성

ode-app의 dockerfile
읽으며 이미지 생성중

```
found 0 vulnerabilities

Removing intermediate container 52ee0e16759f
 ---> 3b38a4974674
Step 5/5 : CMD [ "node", "server.js" ]
 ---> Running in b950175b31cb
Removing intermediate container b950175b31cb
 ---> af340761b8e3
Successfully built af340761b8e3
Successfully tagged node-app-docker_node-app:latest
WARNING: Image for service node-app was built because it did not already exist. To rebuild t
his image you must use `docker-compose build` or `docker-compose up --build`.
Creating node-app-docker_node-app_1     ... done
Creating node-app-docker_redis-server_1 ... done
Attaching to node-app-docker_redis-server_1, node-app-docker_node-app_1
redis-server_1  | 1:C 06 Jun 2020 04:02:46.662 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0
Oo
redis-server_1  | 1:C 06 Jun 2020 04:02:46.662 # Redis version=6.0.4, bits=64, commit=000000
00, modified=0, pid=1, just started
redis-server_1  | 1:C 06 Jun 2020 04:02:46.662 # Warning: no config file specified, using th
e default config. In order to specify a config file use redis-server /path/to/redis.conf
redis-server_1  | 1:M 06 Jun 2020 04:02:46.664 * Running mode=standalone, port=6379.
redis-server_1  | 1:M 06 Jun 2020 04:02:46.665 # WARNING: The TCP backlog setting of 511 can
not be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
redis-server_1  | 1:M 06 Jun 2020 04:02:46.665 # Server initialized
redis-server_1  | 1:M 06 Jun 2020 04:02:46.665 # WARNING you have Transparent Huge Pages (TH
P) support enabled in your kernel. This will create latency and memory usage issues with Red
is. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enab
led' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot
. Redis must be restarted after THP is disabled.
redis-server_1  | 1:M 06 Jun 2020 04:02:46.665 * Ready to accept connections
node-app_1      | Running on port 8080
```



Number is going up 0

이제는 실제로 docker-compose를 이용하여 앱을 실행해보자.

포트 맵핑이 5000:8080으로 되어 있으니 localhost:5000을 통해서 가면 된다.

node-app 컨테이너와
edis-server 컨테이너
생성

edis-server에서 오는
로그들

node-app에서 오는
로그

Docker compose stop & down

도커 컴포즈를 통해 작동시킨 컨테이너들을
한꺼번에 중단 시키려면
docker compose down으로 할 수 있습니다.

실제로 한번 해보

docker compose up --build

docker-compose up vs docker compose up --build

docker-compose up 이미지가 없을 때 이미지를 빌드하고

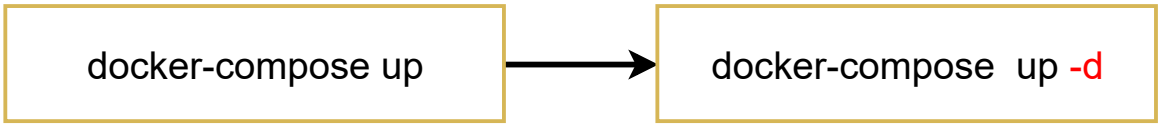
docker-compose up --build 이미지가 있든 없든 이미지를 빌드하

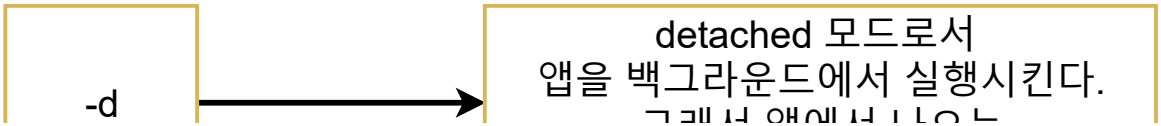
다른 터미널은 켜

docker compose down

굳이 다른 터미널 켜지 않고 하나의 터미널로 해결하고

docker compose를 컨테이너를 실행할 때





컨테이

고 컨테이너

그래서 앱에서 나오는
output을 표출하지 않는다.

그래서 이렇게 앱을 실행한다면 하나의 터미널에서 앱을 작동시키고 중단시킬 수 있다.