7주차

- 먼저 터미널에 밑에 코드를 입력하여 노드 이미지 실행함.

  `$ docker run node`
- node : 도커 허브의 공식 이미지

  off 방법 : `$ docker ps`
- 인터렉티브 모드 시작

  `$ docker run -it node`

[실행 화면]

```
Welcome to Node.js v14.9.0.
Type ".help" for more information.
>
```

- 여기 노드에서 몇몇 기본 명령을 실행할 수 있음.

[종료 방법]

Control+C 를 2번 눌러 종료하면 됨.

[확인 방법]

```
$ docker ps 또는 $ docker ps -a

$ docker exec
```

- 애플리케이션이 실행 되고 있는 와중에 컨테이너 내에 추가적인 명령어를 실행 할 수 있음

[주의점]

- ! exec 실행 시 주의 할 점 : 실행 중인 컨테이너의 이름이 필요하다는 것임.
- `$ docker exec vigorous_dewdney npm init` : 노드 프로젝트를 생성하는 `npm init`
  몇 가지 질문은 하지만 생성이 안 되고 금방 끝나는 모습을 볼 수가 있음.
- "exec 사용할 시 인터렉티브 모드에서 시작해야 함."

[사용 방법]

- `$ docker exec -it vigorous_dewdney npm init`

  → 이렇게 하면 질문이 나옴.

- `package name` : { 사용할 이름 입력 후 엔터 } 이러고 나면 설치가 됨.
- 알 수 있는 점
  - 시스템에 node 또는 npm을 설치하지 않고, 내부 컨테이너에서 실행이 되는 것을 알 수가 있다.
  - **제한점이 있음.**
    그 컨테이너에 프로젝트가 있다면 여기에서도 바로 프로제그를 생성할 수 있지만 그 프로젝트엔 접근할 수 없기 때문에

[컨테이너 중지 방법]

```
$ docker ps
$ docker stop vigorous_dewdney
```

유틸리티 컨테이너를 만들려면 자체 이미지가 필요함.

준비물 : dockerfile 생성

코드

```
FROM node:14-alpine
WORKDIR /app
CMD npm init : npm init 에만 실행할 수 있도록 해주는 명령어.
```

→ ?? CMD 가 없다면 사용자에게 모든 권한을 부여하고, 사용자가 이 이미지에 대한 모든 명령을 실행할 수 있도록 해줌

[빌드]

```
$ docker build -t node-util .
```

→ 노드 유틸리티임을 의미하는 node-util 태그를 지정하고 실행

[실행 결과]

```
Sending build context to Docker daemon 732.2kB
Step 1/2 : FROM node:14-alpine
14-alpine: Pulling from library/node
cbdbe7a5bc2a: Downloading 1.493MB/2.813MB
```

→ 이런 식으로 노드 알파인 이미지를 다운로드 하는 것을 볼 수 있음

```
$ docker run node-util npm init : docker run 으로 node-util 이미지에 넣어보는 방법
$ docker run -it node-util npm init : 디폴트로 이 명령은 컨테이너 내부의 app 폴더에
```

- 그리고 난 후 이것들을 가지고 로컬 폴더에 미러링을 하려고 함.
- 그렇게 하고 나면 뭐가 달라질까?
  - 컨테이너에서 생성한 것들을 호스트 머신에서도 사용 가능
- dockerfile 만든 파일을 우클릭 후 경로 복사
  - `$ docker run -it -v {경로 붙여넣기}:/app node-util npm init`
  - 이때 경로 붙여넣기 후 **"꼭 파일 이름은 제거"**
- 다 하고 나면 package.json 이 생성된 것을 볼 수 있음.
- 따라서 호스트 머신에서 node와 npm을 완전히 언인스톨할 수 있게 됨을 볼 수 있음.
- 그래서 호스트 머신에서 유틸리티 기능+npm init 도움으로 프로젝트 생성 가능함.

[장점]

- 컨테이너에서 실수로 몬든 것을 삭제하는 명령을 실행하여 바운드 마운트로 인해 호스트 머신에서도 콘텐츠를 삭제하는 경우를 방지 가능함.

※ `ENTRYPOINT ["executable"]` : CMD 명령과 매우 유사하지만 한 가지 차이점이 있음.

- CMD ["executable"] 가 구문에 있다면 docker run의 이미지 이름 뒤에 그 명령으로 덮어쓰여짐.
- ENTRYPOINT ["executable"] : docker run에서 이미지 이름 뒤에 입력하는 모든 것들이 ENTRYPOINT ["executable"] 뒤에 추가됨.

[dockerfile 코드]

```
FROM node:14-alpine
WORKDIR /app
ENTRYPOINT [ "npm" ]
```

$ docker run -it -v {dockerfile 경로 복사 후 파일 이름은 삭제}:/app mynpm init
** 이때 npm을 안 붙이는 이유는 이미 코드 위에 npm을 넣어놨기에 init만 입력하면 됨

npm install 의 경우

$ docker run -it -v {dockerfile 경로 복사 후 파일 이름은 삭제}:/app mynpm install
package.json 에서 찾은 모든 종속성이 설치가 됨.

→ !! 다 설치되고 난 후 $ docker ps 하여 확인하면 알 수 있는 것. 항상 컨테이너가 종료된다는 점

[docker 프로젝트를 할 때 많이 볼 수 있는 유용한 것]

- dockerfile /app 쪽에 설치 되어있는 package.json 에 수동적으로 종속성을 추가로 할 수 잇음.

<package.json 파일 내부 안>

```
{
   "author":"",
   "license": "ISC",
   "dependencies" {이 줄을 추가하면 됨. 이름과 버전을 알 경우}
}
```

docker 프로젝트를 할 때 많이 보이는 패턴

- `$ docker run -it -v {dockerfile 경로 복사 후 파일 이름은 삭제}:/app mynpm install express --save` : 명시적 종속성 설치 가능함.
- 다 하고 난 후 옆을 보면 node_modules 파일들이 생성 되어있는 것을 확인 할 수 있음.
- 바인딩 마트로 인해 이 로컬 폴더로 미러링 되는 거.

- 역자주 : 컨테이너의 app 폴더에 인스톨 된 것들이 바인딩 마운트로 된 폴더에 미러링 됨.

[단점]

- 터미널의 명령 프롬프트에서 위 코드처럼 꽤 긴 명령을 실행해야 함.
- 해결 방법 : 도커 컴포즈
    - (참고): 여기에서 도커 컴포즈를 사용하면 아주 유용함.