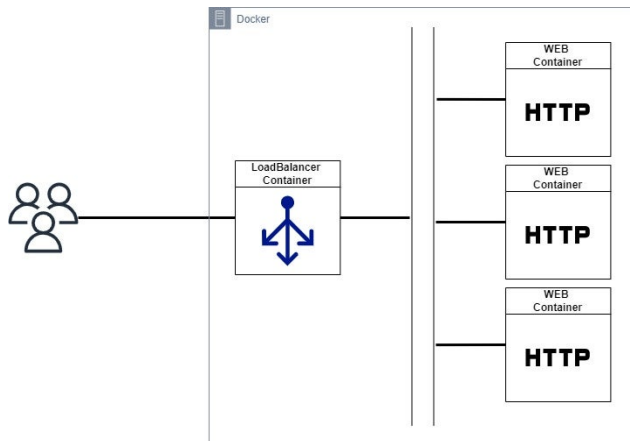


Docker 3tier

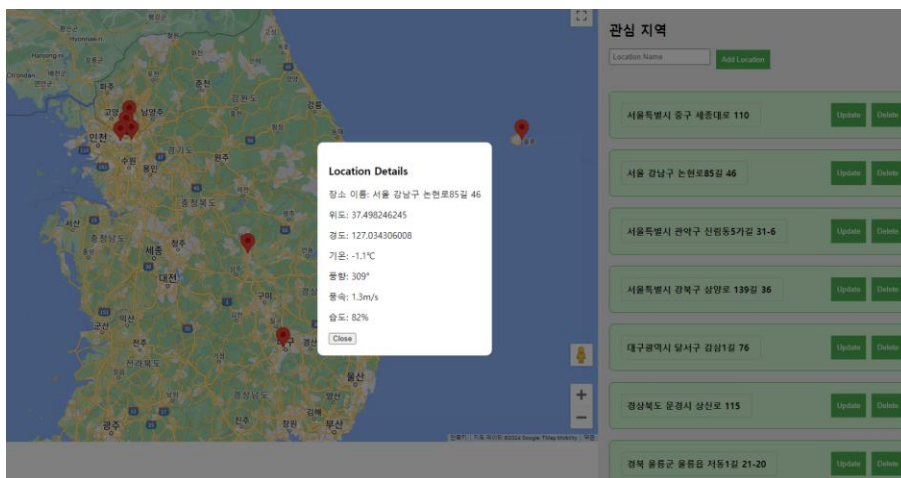
실습 조건



-> Web, was, db 를 컨테이너로 구성해서 3tier 를 구축

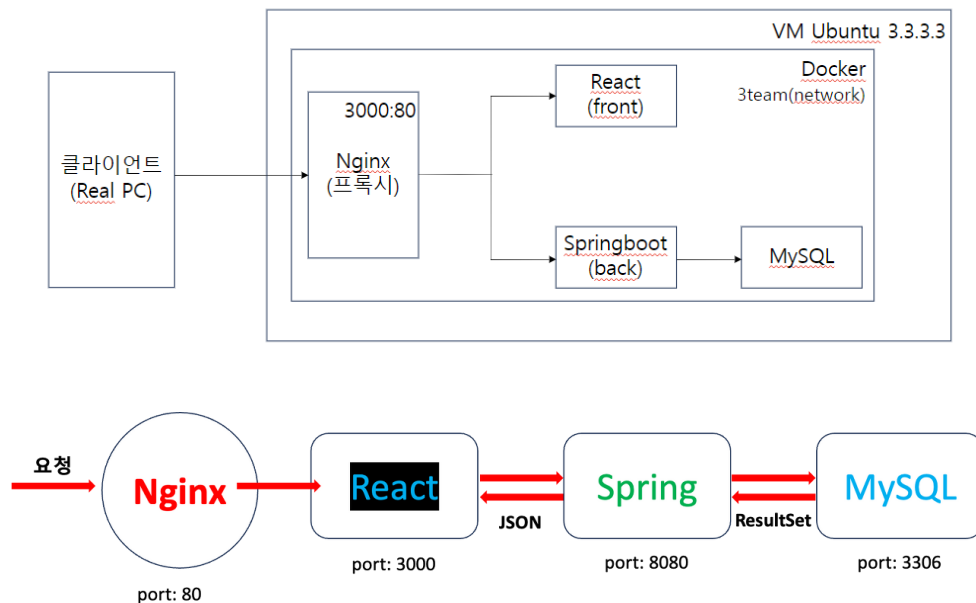
실습 목표

Api 발표 때 만들었던 장소별 좌표 api 서비스를 컨테이너 환경에서 구축



다음과 같은 구조로 컨테이너를 구축해봄

Nginx + React + springboot + DB



-> Nginx 가 프록시 서버로 기능

클라이언트가 react 애플리케이션에 접속, 요청. React 는 인터페이스 랜더링

클라이언트가 정적 파일에 접근하면, nginx 는 해당 파일을 찾아서 클라이언트에게 전달

백엔드 API 에 요청을 보내면, nginx 를 거쳐 백엔드 서버에 전달. 이때 프록시에 설정된 경로를 따라 이동

백엔드 서버가 요청을 처리하고 응답을 생성

응답이 nginx 를 통해 클라이언트로 전달

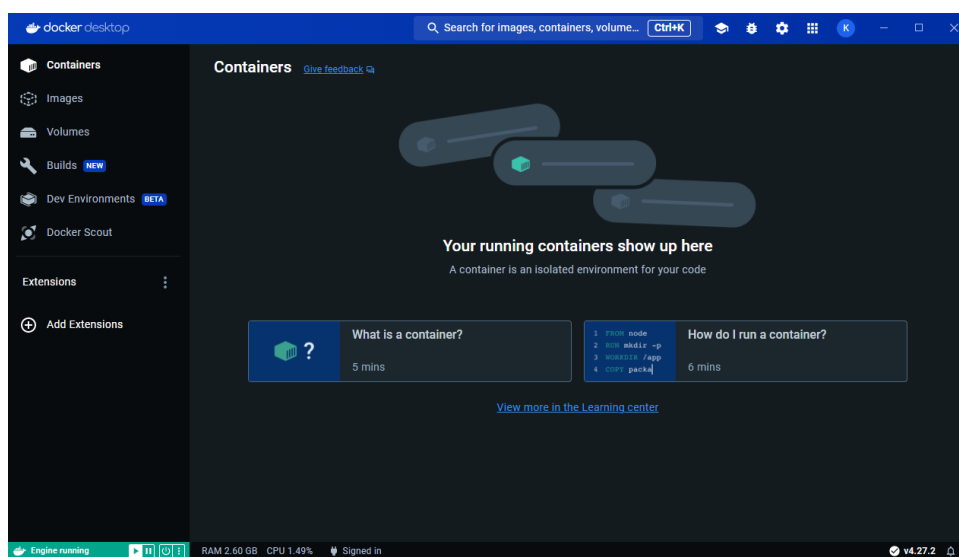
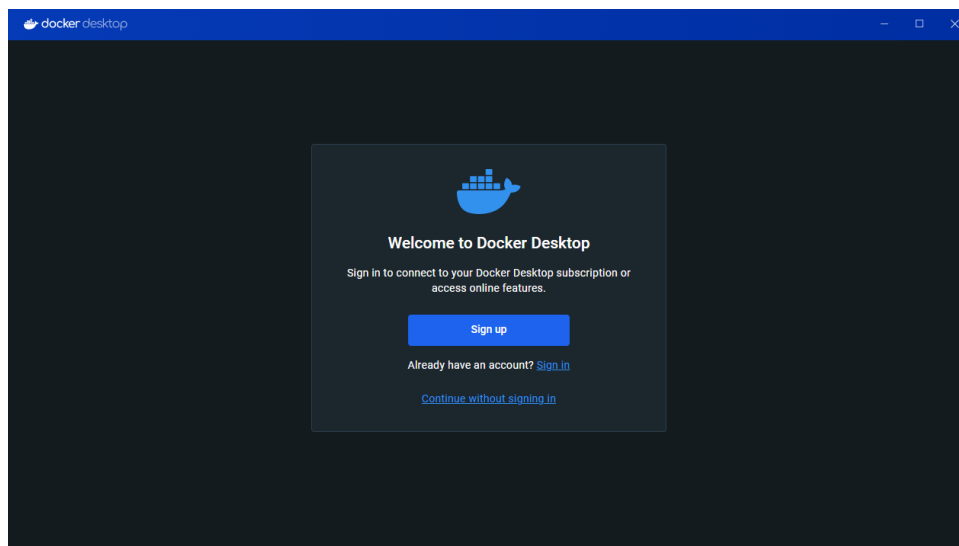
윈도우 Docker

실습 전, 윈도우 환경에서도 docker 를 해볼 수 있지 않을까 해서 docker 를 구축해봄

물론, 어차피 react, springboot 코드를 build 하려면 여기서 해야됨...

Docker Desktop 으로 데스크탑 환경에서 docker 를 사용할 수 있게 함

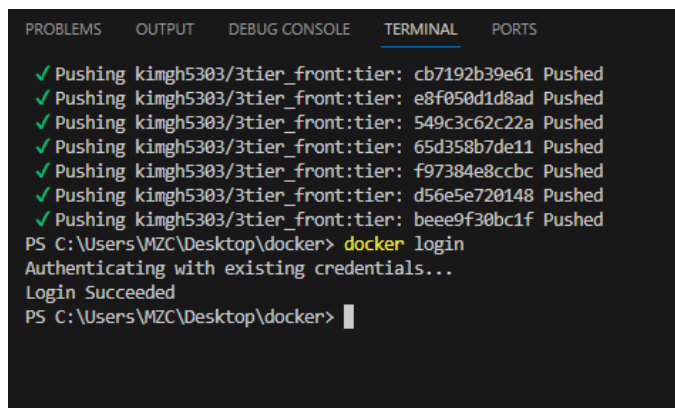
Docker Desktop 설치



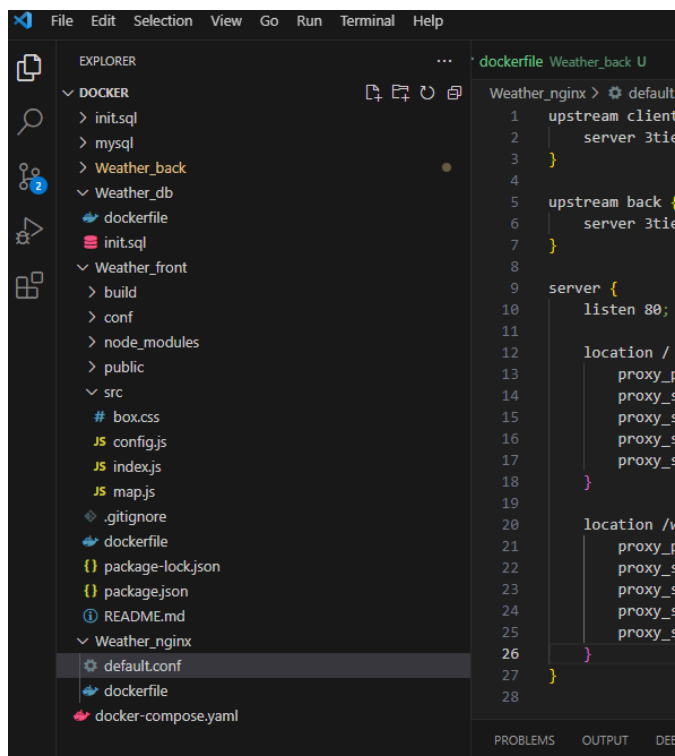
보다 편리하게 docker 를 사용하기 위해 IDE 를 사용하면 됨

Vscode 터미널 환경에서 사용 가능

Vscode terminal docker



```
✓ Pushing kimgh5303/3tier_front:tier: cb7192b39e61 Pushed
✓ Pushing kimgh5303/3tier_front:tier: e8f050d1d8ad Pushed
✓ Pushing kimgh5303/3tier_front:tier: 549c3c62c22a Pushed
✓ Pushing kimgh5303/3tier_front:tier: 65d358b7de11 Pushed
✓ Pushing kimgh5303/3tier_front:tier: f97384e8ccbc Pushed
✓ Pushing kimgh5303/3tier_front:tier: d56e5e720148 Pushed
✓ Pushing kimgh5303/3tier_front:tier: beee9f30bc1f Pushed
PS C:\Users\MZC\Desktop\docker> docker login
Authenticating with existing credentials...
Login Succeeded
PS C:\Users\MZC\Desktop\docker> 
```



The screenshot shows the VS Code interface with the Explorer view on the left and the Editor view on the right. The Explorer view shows a project structure with folders like DOCKER, Weather_back, Weather_db, Weather_front, and Weather_nginx. The Editor view shows the content of the Dockerfile, which defines a multi-stage build for a Docker image.

```
Weather_nginx > default
1 upstream client
2 | server 3tie
3 }
4
5 upstream back
6 | server 3tie
7 }
8
9 server {
10     listen 80;
11
12     location /
13         proxy_pass http://127.0.0.1:8080;
14     proxy_set_header Host $host;
15     proxy_set_header X-Real-IP $remote_addr;
16     proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
17     proxy_set_header X-Forwarded-Proto $scheme;
18 }
19
20 location /w
21     proxy_pass http://127.0.0.1:8080;
22     proxy_set_header Host $host;
23     proxy_set_header X-Real-IP $remote_addr;
24     proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
25     proxy_set_header X-Forwarded-Proto $scheme;
26 }
27
28 }
```

컨테이너 파일을 만들기 전 코드를 수정해야 함

기존에는 인프라를 생각하지 않았기 때문에 코드에 ip, port 를 작성했지만

컨테이너는 환경에 구애받지 않도록 하기 위해 환경변수로 설정해야함

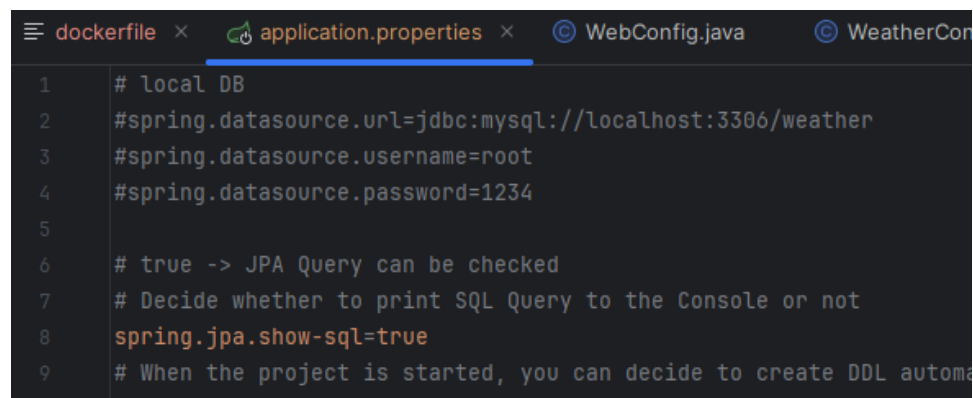
React 코드 수정

다음과 같이 get, post, put, delete 전부 ip 부분을 없애줌

```
const closeModal = () => {  
  setSelectedLocation(null);  
};  
  
const fetchLocations = async () => {  
  try {  
    // const response = await axios.get(`${serverIP}/weathers/locations`);  
    const response = await axios.get('/weathers/locations');  
    setLocations(response.data.data || []);  
  } catch (error) {  
    console.error('Error fetching locations:', error);  
  }  
};
```

Springboot 코드 수정

마찬가지로 application.properties 에서 db 연결 주소를 전부 주석처리

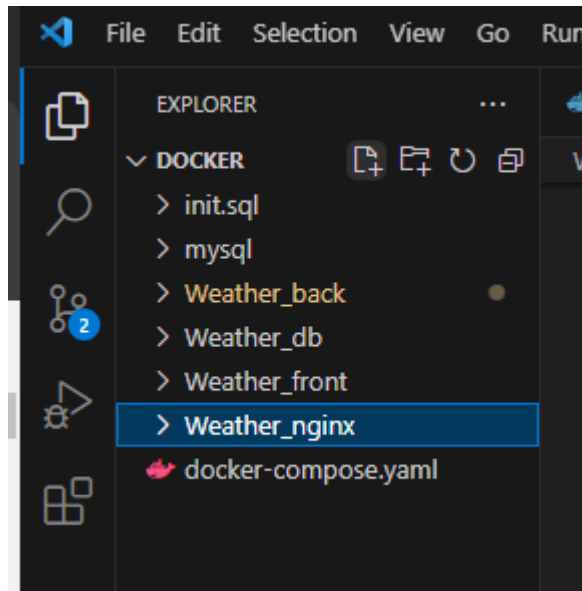


```
# local DB  
#spring.datasource.url=jdbc:mysql://localhost:3306/weather  
#spring.datasource.username=root  
#spring.datasource.password=1234  
  
# true -> JPA Query can be checked  
# Decide whether to print SQL Query to the Console or not  
spring.jpa.show-sql=true  
# When the project is started, you can decide to create DDL automa
```

이제 모든 준비는 끝났고 docker build 를 위한 환경을 만들어줌

Docker 라는 폴더를 만들고 그 안에 React, springboot 프로젝트, db, nginx

디렉토리를 만들어줌

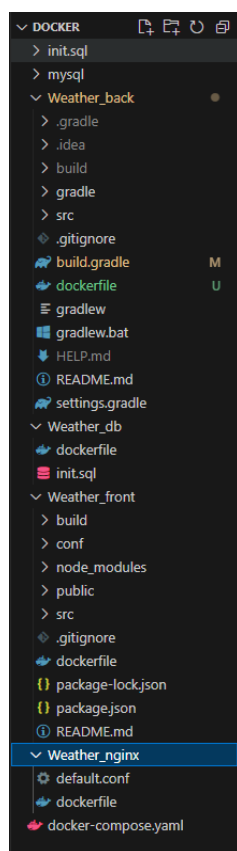


동시에 Docker compose 를 이용하기 위해 yaml 파일을 만들어줌

Docker compose 란?

-> 여러 개의 컨테이너를 정의하고 실행하기 위한 도구

각 디렉토리 안에 dockerfile 을 생성해줌



이후 각각의 dockerfile 을 작성해줌

React dockerfile

```
Weather_front > dockerfile
1  # 기반 이미지로 Node.js 13.12.0 Alpine 이미지 사용
2  FROM node:13.12.0-alpine
3
4  # 작업 디렉터리를 /react-to-do/frontend로 설정
5  WORKDIR /react-to-do/frontend
6
7  # 환경 변수 PATH에 /app/node_modules/.bin을 추가
8  ENV PATH /app/node_modules/.bin:$PATH
9
10 # package.json과 package-lock.json을 복사하여 이미지의 /react-to-do/frontend 디렉터리에 붙여넣음
11 COPY package.json package-lock.json ./
12
13 # npm install 명령어를 실행하여 package.json에 명시된 의존성 패키지 설치
14 RUN npm install
15
16 # 현재 디렉터리의 모든 파일을 이미지의 /react-to-do/frontend 디렉터리에 복사
17 COPY . ./
18
19 # 컨테이너가 시작될 때 npm start 명령어를 실행하여 React 앱을 시작
20 CMD ["npm", "start"]
```

Nginx dockerfile

```
Weather_nginx > dockerfile
1  # 기반 이미지로 nginx 공식 이미지를 사용
2  FROM nginx
3
4  # 로컬의 default.conf 파일을 nginx의 설정 디렉터리로 복사
5  COPY ./default.conf /etc/nginx/conf.d/default.conf
6
7  # 컨테이너가 시작될 때 nginx를 백그라운드에서 실행하지 않고, 포그라운드에서 실행하도록 설정
8  ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Springboot dockerfile

```
Weather_back > dockerfile
1  # FROM <이미지명>:<태그>
2  FROM openjdk:17
3  # Docker 기본 이미지, openjdk:[프로젝트의 jdk버전]
4  ARG JAR_FILE=/build/libs/*.jar
5  # ARG
6  # 변수선언. JAR_FILE을 *.jar파일로 지정
7  COPY ${JAR_FILE} app.jar
8  # COPY
9  # JAR_FILE을 컨테이너의 app.jar로 복사
10 ENTRYPOINT ["java", "-jar", "/app.jar"]
11 # ENTRYPOINT
12 # 컨테이너 시작 시 스크립트 실행
```


DB dockerfile

```
Weather_db > dockerfile
1 FROM mysql:8.0.22
2
3 ENV MYSQL_DATABASE=weather \
4     MYSQL_ROOT_HOST=%' \
5     MYSQL_ROOT_PASSWORD=1234 \
6     TZ='Asia/Seoul'
7
8 CMD ["mysqld", "--character-set-server=utf8mb4", "--collation-server=utf8mb4_unicode_ci"]
```

DB init.sql 작성

```
Weather_db > init.sql
1 CREATE DATABASE IF NOT EXISTS weather;
2
3 USE weather;
4
5 CREATE TABLE IF NOT EXISTS location (
6     loc_id VARCHAR(100) PRIMARY KEY,
7     loc_name VARCHAR(100),
8     loc_latitude DOUBLE,
9     loc_longitude DOUBLE
10 );
```

이제 file 설정은 끝났고 yaml 파일과 nginx 의 conf 파일을 작성해줌

Yaml 을 루트 디렉토리에 작성

윈도우 환경에서는 networks 옵션이 필요 없으므로 주석 처리

Docker-compose.yaml 작성

```
docker-compose.yaml
1  version: "3.2"
2  services:
3    nginx:
4      restart: always
5      container_name: 3tier/nginx
6      build:
7        context: ../Weather/nginx
8        dockerfile: dockerfile
9      image: kimgh5303/3tier/nginx:tier
10     ports:
11       - "3000:80"
12     # networks:
13     #   - 3team
14
15     front:
16       container_name: 3tier/front
17       build:
18         context: ../Weather/front
19         dockerfile: dockerfile
20       image: kimgh5303/3tier/front:tier
21       volumes:
22         - ../Weather/front:/app
23         - /app/node_modules
24
25     back:
26       container_name: 3tier/back
27       restart: on-failure
28       build:
29         context: ../Weather/back
30         dockerfile: dockerfile
31       image: kimgh5303/3tier/back:tier
32       ports:
33         - "8080:8080"
34       # networks:
35       #   - 3team
36
37     environment:
38       SPRING_DATASOURCE_URL: jdbc:mysql://3tier/db:3306/weather?useSSL=false&allowPublicKeyRetrieval=true
39       SPRING_DATASOURCE_USERNAME: "root"
40       SPRING_DATASOURCE_PASSWORD: "1234"
41     depends_on:
42       - db
```

```
43     db:
44       container_name: 3tier/db
45       build:
46         context: ../Weather/db
47         dockerfile: dockerfile
48       image: kimgh5303/3tier/db:tier
49       restart: unless-stopped
50       ports:
51         - "3306:3306"
52       # networks:
53       #   - 3team
54       volumes:
55         - ../mysql/conf.d:/etc/mysql/conf.d
56         - ../init.sql:/docker-entrypoint-initdb.d/init.sql
```

프록시 설정을 위해 nginx 설정 변경

Default.conf 파일 작성

```
Weather_nginx > default.conf
1  upstream client {
2      server 3tier_front:3000;
3  }
4
5  upstream back {
6      server 3tier_back:8080;
7  }
8
9  server {
10     listen 80;
11
12     location / {
13         proxy_pass http://client; # 컨테이너 이름으로 프록시 설정
14         proxy_set_header Host $host;
15         proxy_set_header X-Real-IP $remote_addr;
16         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
17         proxy_set_header X-Forwarded-Proto $scheme;
18     }
19
20     location /weathers {
21         proxy_pass http://back; # 컨테이너 이름으로 프록시 설정
22         proxy_set_header Host $host;
23         proxy_set_header X-Real-IP $remote_addr;
24         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
25         proxy_set_header X-Forwarded-Proto $scheme;
26     }
27 }
28 |
```

구성은 끝났고, 이제 docker 를 Build 와 동시에 컨테이너 생성후 실행해 보겠음

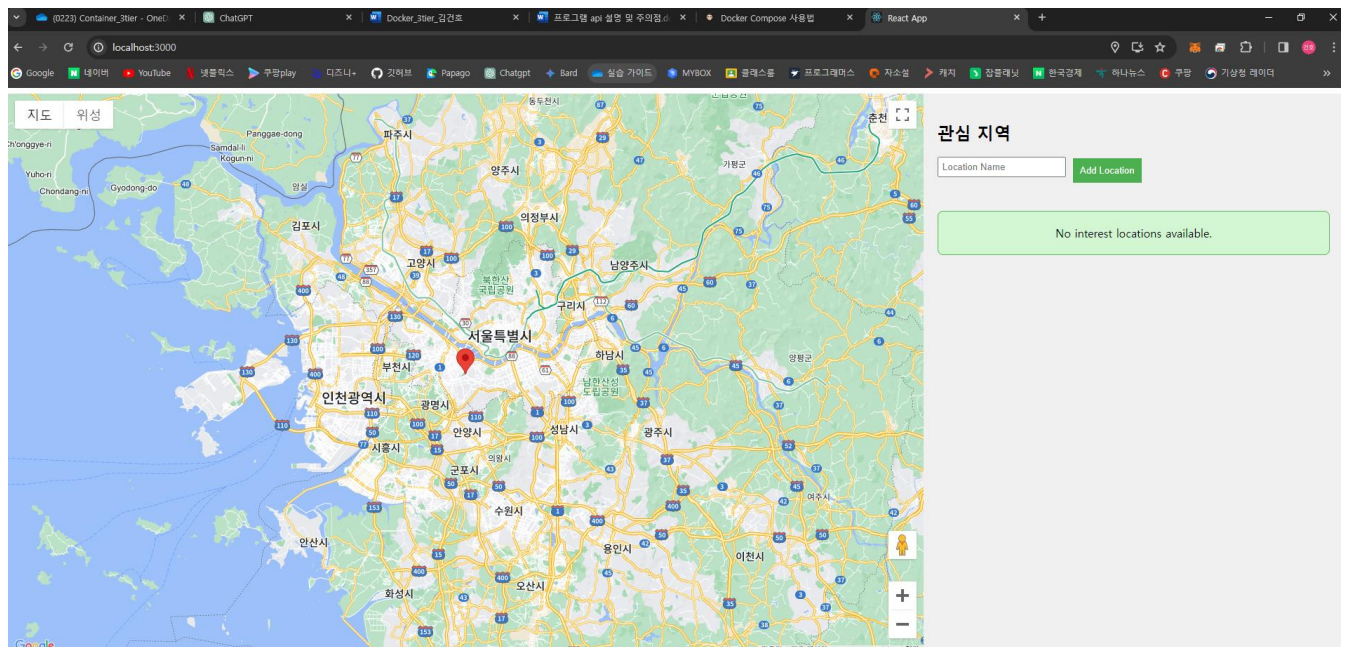
```
PS C:\Users\WZC\Desktop\docker> docker-compose up -d --build
2024/02/26 05:16:52 http2: server: error reading preface from client ../pipe/docker_engine: file has already been closed
[+] Building 0.0s (0/0) docker:default
2024/02/26 05:16:52 http2: server: error reading preface from client ../pipe/docker_engine: file has already been closed
[+] Building 1.7s (7/9)
[+] Building 2.1s (19/24)
[+] Building 9.1s (31/32)
-> [front 1/5] FROM docker.io/library/node:13.12.0-alpine@sha256:cc85e728fab3827ada20a181ba280cae1f8b625f256e2c86b9094d9bfe834766
-> [front internal] load build context
-> => transferring context: 3.47MB
-> CACHED [nginx 2/2] COPY ./default.conf /etc/nginx/conf.d/default.conf
-> [nginx] exporting to image
-> => exporting layers
-> => writing image sha256:4031e3c6ac792df646c1a5ba116ff5ab0f3aa32082f7ebb6107ea0437343e6b4
-> => naming to docker.io/kinggh5303/3tier_nginx:tier
-> [back internal] load build definition from dockerfile
-> => transferring dockerfile: 423B
-> [back internal] load metadata for docker.io/library/openjdk:17
-> [back auth] library/openjdk:pull token for registry-1.docker.io
-> [back internal] load .dockerignore
-> => transferring context: 2B
-> [back internal] load build context
```

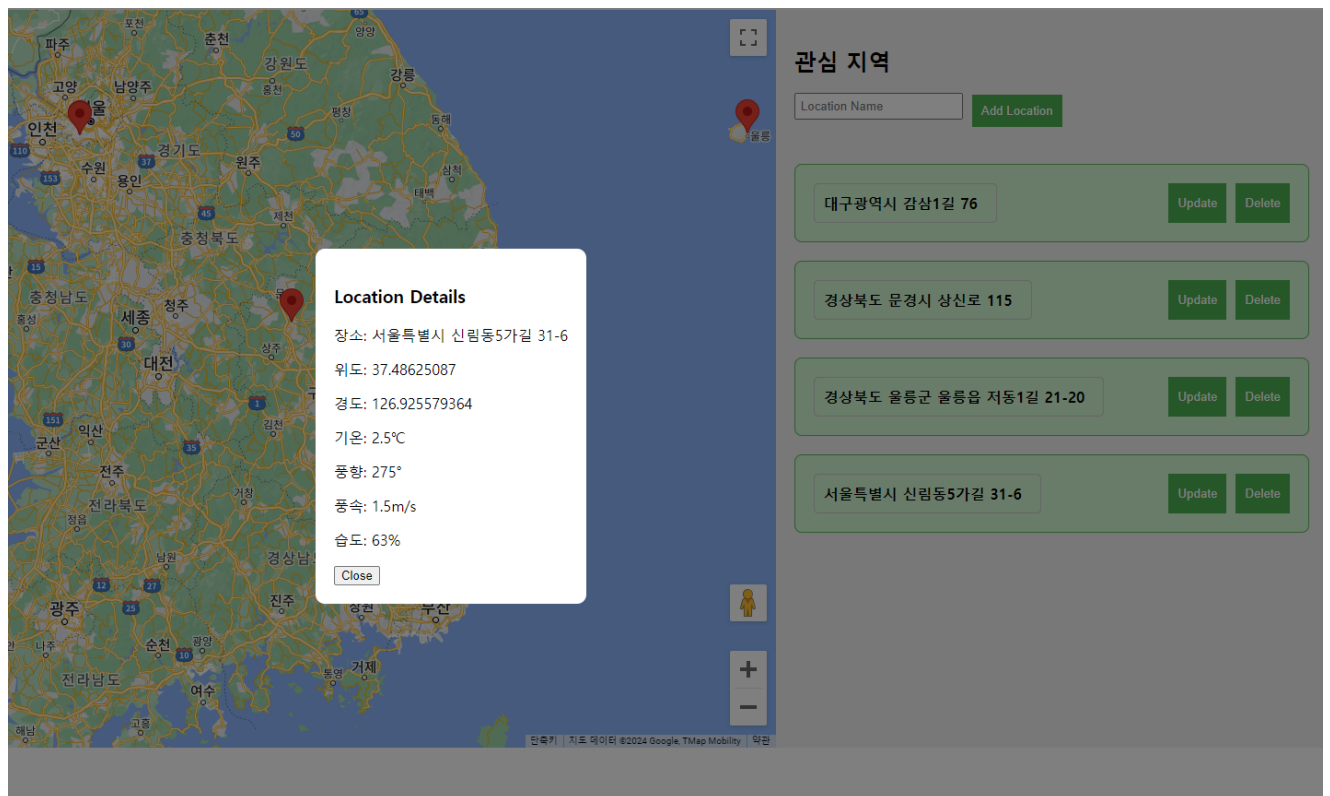
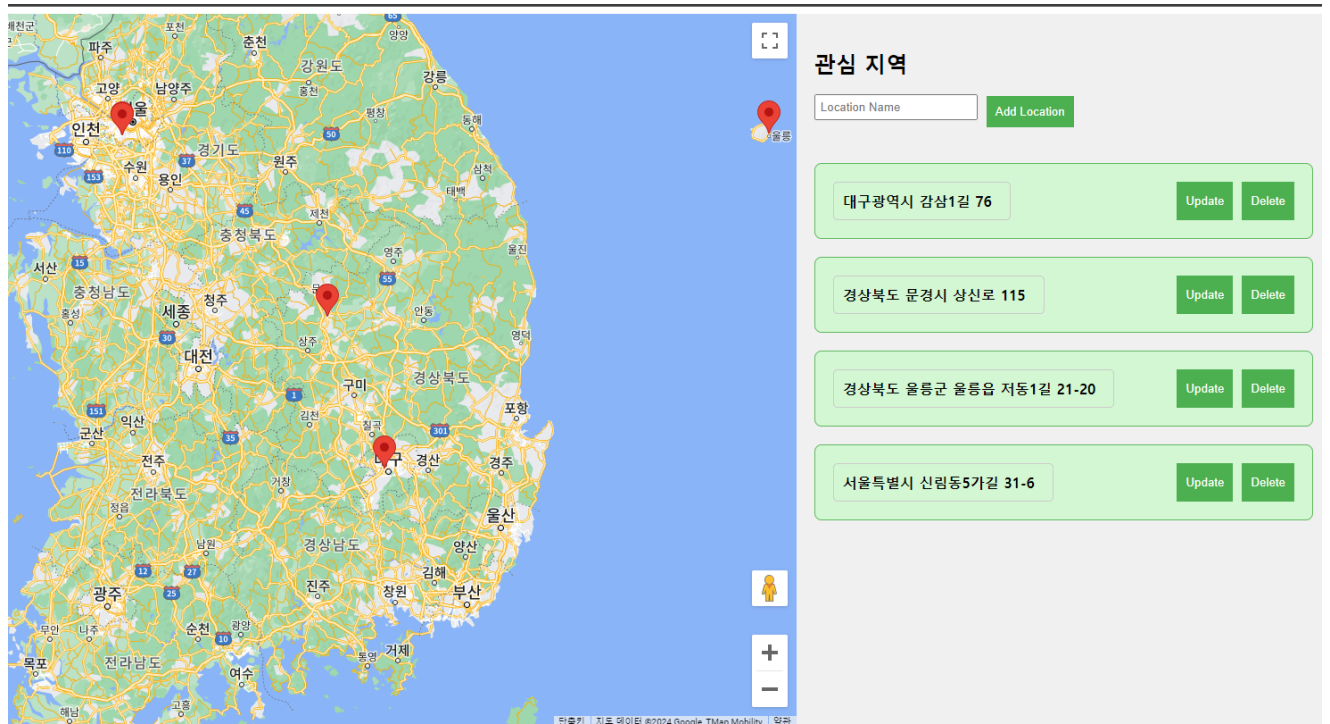
```
=> => exporting layers
=> => writing image sha256:c366a373d4bde7d9738982361bdfbdaa133b72bf3499126d506cccbdd8a8dfc5
=> => naming to docker.io/kingh5303/3tier_front:tier
[+] Running 4/4
✔ Container 3tier_db Started
✔ Container 3tier_nginx Started
✔ Container 3tier_front Started
✔ Container 3tier_back Started
PS C:\Users\VMC\Desktop\docker> docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
ed4bdf8158e9   kingh5303/3tier_back:tier           "java -jar /app.jar"    43 seconds ago Up 28 seconds 0.0.0.0:8080->8080/tcp            3tier_back
1f6343973a57   kingh5303/3tier_nginx:tier          "nginx -g 'daemon of..." 43 seconds ago Up 41 seconds 0.0.0.0:3000->80/tcp              3tier_nginx
1327810ae6fe   kingh5303/3tier_db:tier             "docker-entrypoint.s..." 43 seconds ago Up 20 seconds 0.0.0.0:3306->3306/tcp, 33060/tcp 3tier_db
d3a6aba5279e   kingh5303/3tier_front:tier         "docker-entrypoint.s..." 43 seconds ago Up 42 seconds                               3tier_front
PS C:\Users\VMC\Desktop\docker> docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
kingh5303/3tier_front   tier       c366a373d4bd   50 seconds ago  956MB
kingh5303/3tier_nginx   tier       4031e3c6ac79   2 hours ago    187MB
kingh5303/3tier_back    tier       8dd076cb2b9c   29 hours ago   524MB
kingh5303/3tier_db      tier       682427cf779    3 years ago    545MB
PS C:\Users\VMC\Desktop\docker>
```

-> 잘 만들어진것을 확인

확인을 위해 접속해보겠습니다

Localhost:3000 으로 접속

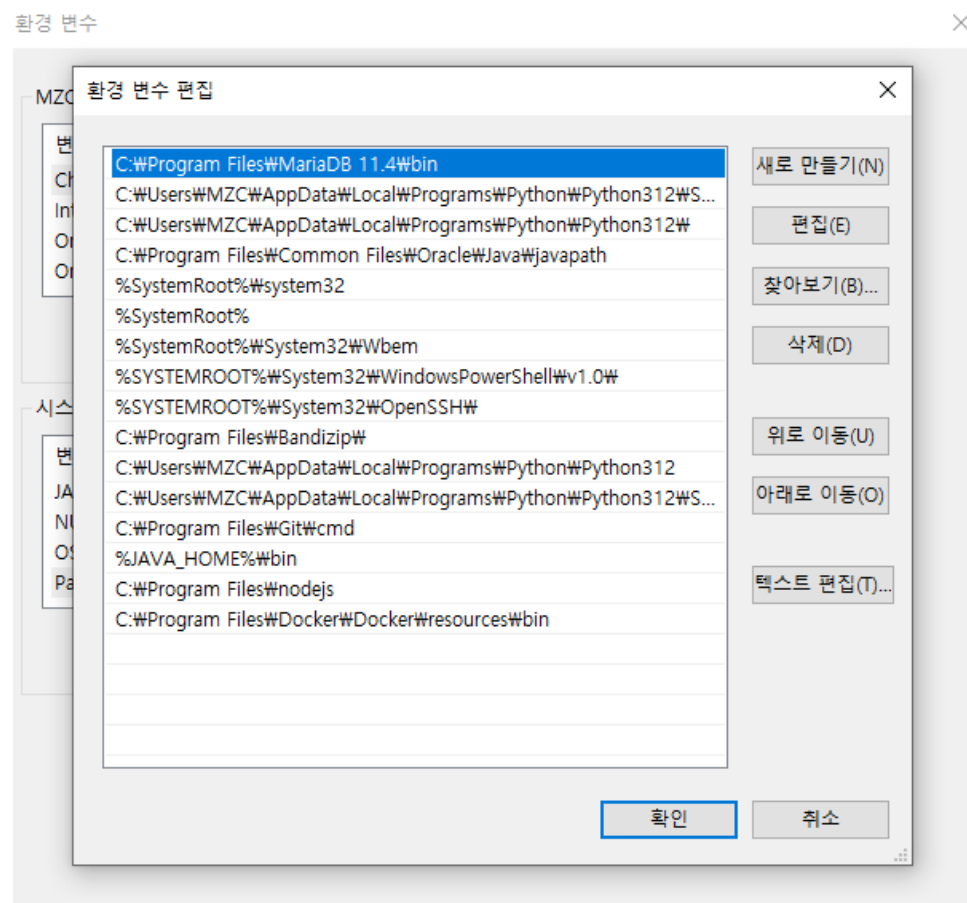
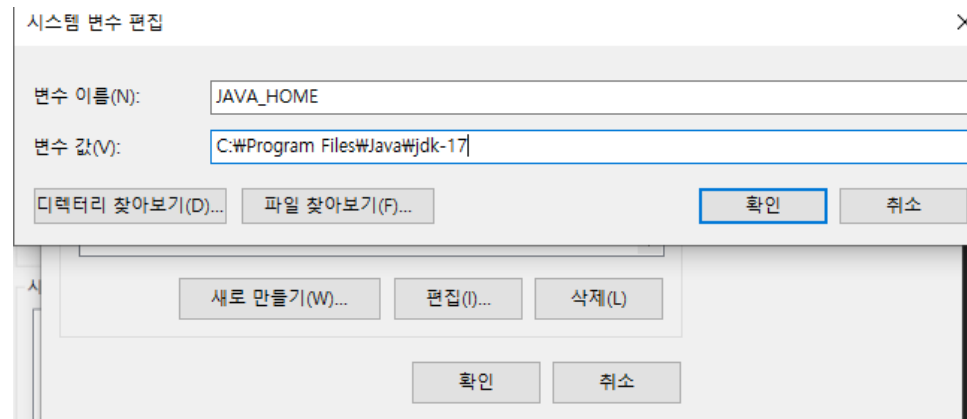




-> 잘 구동되는것을 확인!

TS) 코드에는 아무 문제 없는데 혹시 구동 오류가 나면 거의 대부분 환경변수 오류일 가능성이 높음

다음과 같이 nodejs, mariadb(mysql), java 환경 변수를 설정해주면 됨

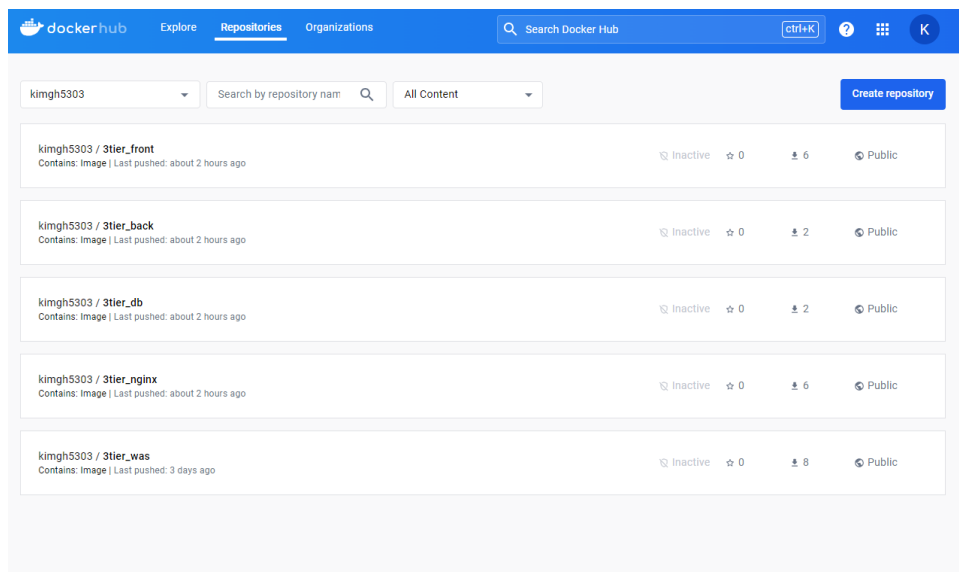


VM Ubuntu Docker

본격적으로 가상머신에서의 컨테이너 실습을 해보겠습니다

윈도우에서 Docker-compose push 를 하면

조금 전에 윈도우에서 만들었던 이미지 파일들이 docker hub 에 올라갔을것임



이제 이 이미지 파일들을 가상 머신에 pull 해올 것임

일단 3team 라는 network 를 하나 만들어줌

Docker network create 3team

```
root@docker:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
57ddad9d7af0        3team              bridge              local
48a27696a51d        bridge             bridge              local
77572c5604bc        host               host                local
7035b762f286        none               null                local
fdabd13b0bd6        root_default       bridge              local
root@docker:~#
```

루트 디렉토리에 마찬가지로 docker-composet.yml 파일을 작성

이때 아까 주석 처리했던 networks 옵션을 활성화해주고 여기서는 build 를 할 필요가 없기 때문에 build 옵션을 지워줌

```
GNU nano 6.2 dock
version: "3.2"
services:
  nginx:
    restart: always
    container_name: 3tier_nginx
    image: kimgh5303/3tier_nginx:tier
    ports:
      - "3000:80"
    networks:
      - 3team
  front:
    restart: always
    container_name: 3tier_front
    image: kimgh5303/3tier_front:tier
    networks:
      - 3team
  back:
    container_name: 3tier_back
    restart: on-failure # 실패 시 재시작 정책 설정
    image: kimgh5303/3tier_back:tier
    ports:
      - "8080:8080"
    networks:
      - 3team
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://3tier_db:3306/weather?useSSL=false&allowPublicKeyRetrieval=true
      SPRING_DATASOURCE_USERNAME: "root"
      SPRING_DATASOURCE_PASSWORD: "1234"
    depends_on:
      - db
  db:
    container_name: 3tier_db
    image: kimgh5303/3tier_db:tier
    restart: unless-stopped
    ports:
      - "3306:3306"
    networks:
      - 3team
    volumes:
      - ./mysql/conf.d:/etc/mysql/conf.d
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
networks:
  3team:
    external: true
```

여기까지 완료했으면 docker-compose 를 설치해줌

GitHub 릴리스 페이지에서 바이너리 파일을 다운로드

```
Sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-  
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

실행 권한 부여: 다운로드한 파일에 실행 권한을 부여

```
sudo chmod +x /usr/local/bin/docker-compose
```

/usr/local/bin 에 대한 실행 경로를 설정하기 위해 심볼릭 링크를 생성

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

설치했으면 잘 됐는지 확인

Docker-compose --version

```
root@docker:~# docker-compose --version
docker-compose version 1.29.2, build 5becea4c
root@docker:~#
```

이제 모든 구성은 끝났고 실행해 봄

마찬가지로 docker compose 한 번에 실행

이번에는 이미지가 없으니 pull 해 올 것임

아까 yaml 파일에서 줬던 image 옵션이 pull 해 올 것임

```
root@docker:~# nano docker-compose.yml
root@docker:~# docker-compose up -d
Pulling nginx (kimgh5303/3tier/nginx:tier)...
tier: Pulling from kimgh5303/3tier/nginx
e1caac4eb9d2: Pull complete
88f6f236f401: Pull complete
c3ea3344e711: Pull complete
cc1bb4345a3a: Pull complete
da8fa4352481: Pull complete
c7f80e9cda2: Pull complete
19a8e9824cb6: Pull complete
b753e8d60101: Pull complete
Digest: sha256:d0e62e60b67bdd5d5e4477aa41f485026d65f77dd764edd5ad317506e94f4897
Status: Downloaded newer image for kimgh5303/3tier/nginx:tier
Pulling front (kimgh5303/3tier/front:tier)...
tier: Pulling from kimgh5303/3tier/front
aad63a933944: Pull complete
a00bd932208e: Pull complete
c57f2c59b937: Pull complete
f3446470f297: Pull complete
b3a330e6cc86: Pull complete
f81a129c42bd: Pull complete
e16c4e787049: Pull complete
58ebe78a7c96: Pull complete
Digest: sha256:64a0478acb88148775bf4a60112e6035e7f8f95132fdf1f4df235aca1c2b21b4
Status: Downloaded newer image for kimgh5303/3tier/front:tier
Pulling db (kimgh5303/3tier/db:tier)...
tier: Pulling from kimgh5303/3tier/db
a076a020afff: Pull complete
f6c208f3f901: Pull complete
88a9455a0165: Pull complete
406c9b8427c6: Pull complete
7c88599c0b25: Pull complete
25b5c6debda5: Pull complete
43a5816f1617: Pull complete
69dd1fbf9190: Pull complete
5346a60dcee8: Pull complete
ef28da371fc9: Pull complete
fd04d93b852: Pull complete
050c49742ea2: Pull complete
Digest: sha256:c8b8cfaf090279b0277fdd9afefd94e925629395855fb92c0c4b3b2d02d4e0b4
Status: Downloaded newer image for kimgh5303/3tier/db:tier
Pulling back (kimgh5303/3tier/back:tier)...
tier: Pulling from kimgh5303/3tier/back
38a980f2cc8a: Pull complete
de849f1cfbe6: Pull complete
a7203ca35e75: Pull complete
8abfa2bffc31: Pull complete
Digest: sha256:f06942fda234071a50f41cf8ff1feb695dc1146a4f378245f999f2f8c0bbf539
Status: Downloaded newer image for kimgh5303/3tier/back:tier
Creating 3tier/nginx ... done
Creating 3tier/front ... done
Creating 3tier/db ... done
Creating 3tier/back ... done
root@docker:~#
```

잘 받아와졌으면 확인해보겠습니다

이미지 파일 확인

```
Every 1.0s: docker images
```

docker: Sun				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
kimgh5303/3tier_front	tier	be4767ea003a	3 hours ago	956MB
kimgh5303/3tier_nginx	tier	4031e3c6ac79	3 hours ago	187MB
kimgh5303/3tier_back	tier	8dd076cb2b9c	29 hours ago	524MB
kimgh5303/3tier_db	tier	682427cfd779	3 years ago	545MB

컨테이너 확인

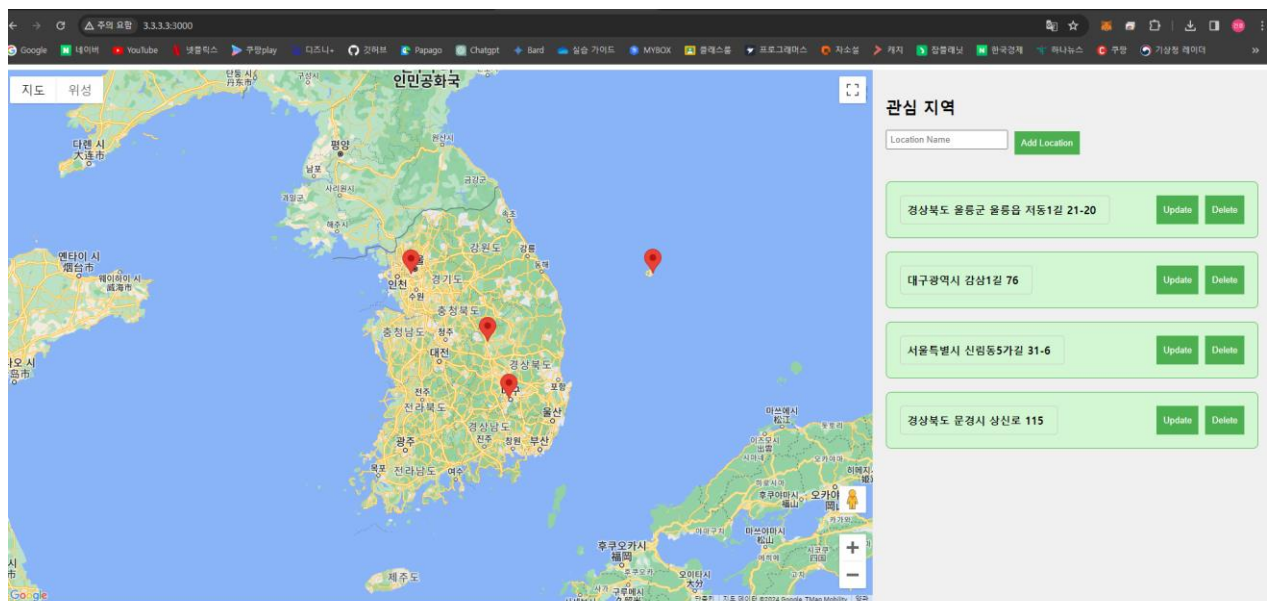
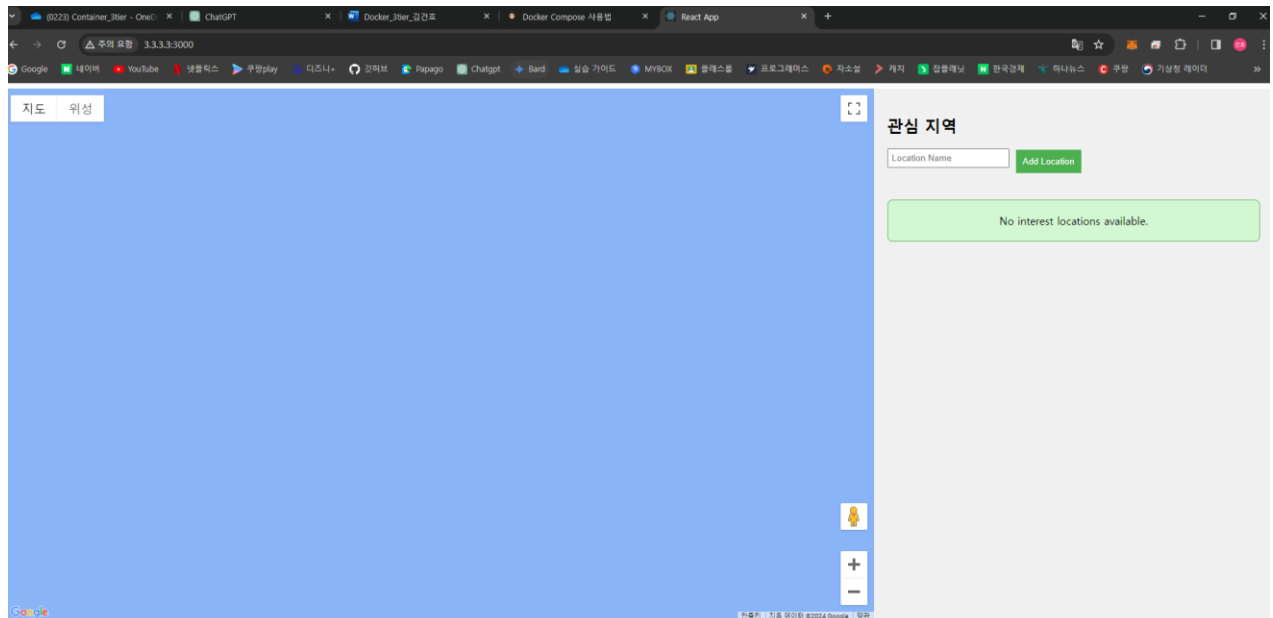
```
Every 1.0s: docker ps -a
```

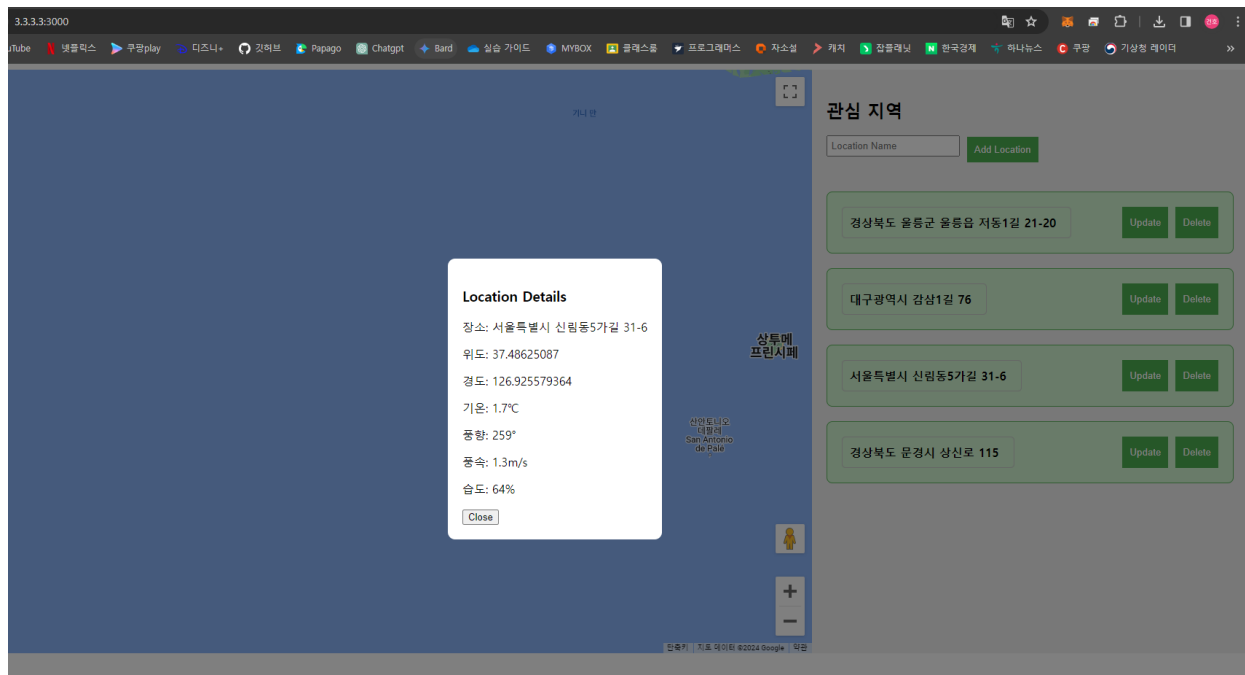
docker: Sun Feb 25 20:39:27 2024

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
738f833f98ae	kimgh5303/3tier_back:tier	"java -jar /app.jar"	2 hours ago	Up 2 hours
s 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp				3tier_back
8aad6a2f9209	kimgh5303/3tier_front:tier	"docker-entrypoint.s..."	2 hours ago	Up 2 hours
s				3tier_front
8fe6272fd730	kimgh5303/3tier_nginx:tier	"nginx -g 'daemon of..."	2 hours ago	Up 2 hours
s 0.0.0.0:3000->80/tcp, :::3000->80/tcp				3tier_nginx
6f0fb093e64a	kimgh5303/3tier_db:tier	"docker-entrypoint.s..."	2 hours ago	Up 2 hours
s 0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp				3tier_db

최종적으로 브라우저에서 확인해보겠습니다

3.3.3.3:3000 으로 접속





성공적으로 된 것을 확인!

가상 환경이라 **현재 위치를 못 잡는다...**

분명 외부 api 와의 통신은 된다면 이것도 돼야 할텐데 안된다...

정확한 이유는 잘 모르겠다.....

Dockerhub :

kimgh5303/3tier_nginx
 kimgh5303/3tier_front
 kimgh5303/3tier_back
 kimgh5303/3tier_db

Git 링크 : https://github.com/kimgh5303/Weather_API

발표 자료 > API_3 팀 > 개발 > 프로그램 api 설명 및 주의점 .docx 참고

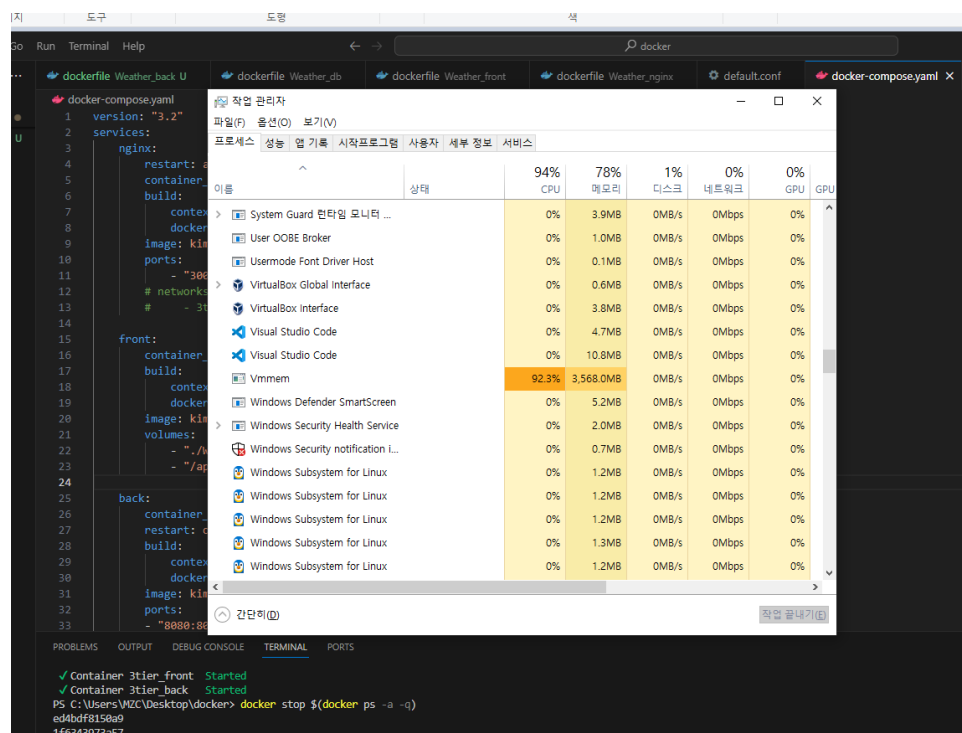
추가 TS) VMMEM 자원 이슈

<https://meaownworld.tistory.com/entry/docker%EB%A1%9C-%EC%9D%B8%ED%95%9C-%EB%A9%94%EB%AA%A8%EB%A6%AC-%EB%B0%B0%ED%84%B0%EB%A6%AC-%EC%86%8C%EB%AA%A8-%EC%95%84%EB%81%BC%EA%B8%B0-feat-vmem-%ED%94%84%EB%A1%9C%EC%84%B8%EC%8A%A4>

다음과 같이 cpu 가동률이 비정상적으로 높게 치솟고 메모리를 많이 차지하는 프로세스를 볼 수 있음

바로 vmmem 이란 것인데 이것은 윈도우에서 가상머신 프로세스 및 메모리 관리를 담당함

Docker desktop 을 실행하면서 vmmem 이 동작된 것임



이를 알아보기

-> 리눅스에서 파일 액세스할 때, 리눅스 OS 는 그 정보를 캐시로 사용하기 위해 메모리에 보존하고 메모리가 부족해 더 이상 보존할 수 없을 때까지 반복됨

그러니까 vmmem 이 실행시킨 WSL 이 메모리가 꽉 차면 메모리를 추가 할당하면서 vmmem 윈도우의 메모리 점유율이 점점 높아가는 것임...

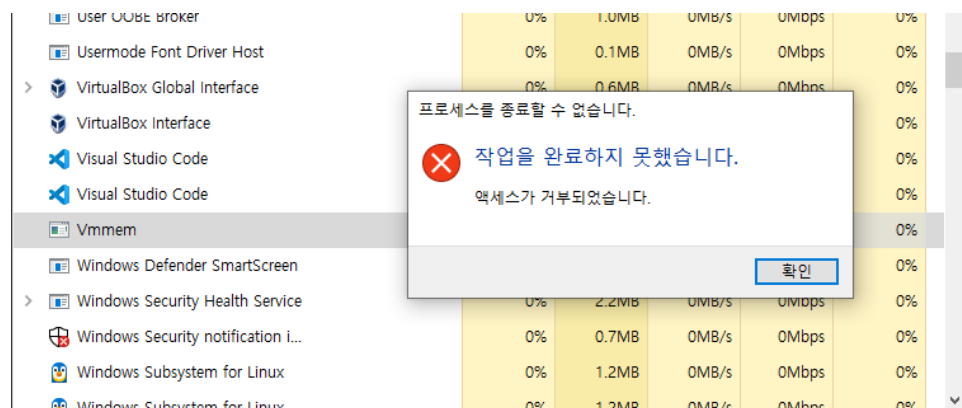
※ wsl 이란? 윈도우에서 리눅스 환경을 실행할 수 있는 윈도우 기능

이를 위한 방법은 여러가지 있었지만 다 자원을 제한하거나 부팅을 해야됨

하지만 노트북은 절전 모드를 주로 사용하기 때문에 이렇게 할 수가 없음

제일 간단한 방법은 docker desktop 을 안쓸때 꺼버리는 방법임

그래서 vmmem 을 끄려고 했지만



이렇게 종료가 되지 않음...

Wsl 이 켜져 있고 이는 커널이기 때문에 사용자가 함부로 끌 수가 없음

(커널을 사용자가 함부로 건드릴 수 없는 이유는 Docker 실습의 커널 참고)

결국 CMD 를 관리자 권한으로 실행시켜 꺼야함

Cmd 에 관리자 권한으로 들어가서

```
C:\> 관리자: 명령 프롬프트
Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>wsl --shutdown

C:\Windows\system32>
C:\Windows\system32>
```

이렇게 wsl --shutdown 을 입력해주면

Usermode Font Driver Host	0%	0.1MB	0MB/s	0Mbps	0%
> VirtualBox Global Interface	0%	0.5MB	0MB/s	0Mbps	0%
VirtualBox Interface	0%	3.9MB	0MB/s	0Mbps	0%
Visual Studio Code	0%	1.3MB	0MB/s	0Mbps	0%
Visual Studio Code	0%	1.5MB	0MB/s	0Mbps	0%
Windows Defender SmartScreen	0.1%	5.3MB	0MB/s	0Mbps	0%
> Windows Security Health Service	0%	2.0MB	0MB/s	0Mbps	0%
Windows Security notification i...	0%	0.7MB	0MB/s	0Mbps	0%
> Windows Subsystem for Linux ...	0%	2.8MB	0MB/s	0Mbps	0%
Windows Wireless LAN 802.11 ...	0%	0.5MB	0MB/s	0Mbps	0%
> Windows 기본 잠금 화면	0%	5.4MB	0MB/s	0Mbps	0%

Wsl 커널이 종료되고 cpu, 메모리가 낭비되는 것을 막을 수 있음

wsl 을 사용하는 사람들은 꽤 오래전부터 인식하던 이슈라고 한다

MS 사에서도 이를 인식하고는 있다는데 아직까지 수정이 없는것으로 보인다...

빨리 수정해주기를....

윈도우를 서버로 잘 쓰지 않고 리눅스를 자주 사용하기 때문에 알아두면 좋을 것 같다