[과제 2] 해설 및 참고자료

💡 과제 2에 대한 해설을 부록의 형태로 준비하였습니다. `docker-compose.yml` 의 각 옵션에 대한 정확한 사용법과 의미를 짚고 넘어가보도록 하겠습니다.

> 출처
> - FastAPI : https://fastapi.tiangolo.com/

```
version: "3.9" services: db: image: postgres:13 ports: - "5432:5432" restart:
always volumes: - pgdata:/var/lib/postgresql/data environment: -
POSTGRES_USER=admin - POSTGRES_PASSWORD=postgres - POSTGRES_DB=postgres
pgadmin: image: dpage/pgadmin4 ports: - "5050:80" restart: always volumes: -
pgadmindata:/var/lib/pgadmin environment: PGADMIN_DEFAULT_EMAIL:
admin@example.com PGADMIN_DEFAULT_PASSWORD: admin logging: driver: none web:
build: . command: bash -c "alembic upgrade head && uvicorn main:app --host
0.0.0.0 --port 8000 --reload" restart: always volumes: - .:/code ports: -
"8000:8000" depends_on: - db volumes: pgdata: pgadmindata:
```

1. db

1.1 컨테이너와 호스트는 각각 5432 포트로 통신

```
ports: - "5432:5432"
```

도커 컴포즈를 통해 포트포워딩을 하기 위해서는 `ports` 옵션을 적용해야합니다. `[호스트 포트]:[컨테이너 포트]` 로 작성합니다. 이렇게 포트포워딩을 하는 것은 단일 컨테이너를 작동시키는 `docker run -p [호스트 포트]:[컨테이너 포트]...` 의 명령어와 같은 역할을 수행하며 실행 명령어가 아닌 설정파일에 각 컨테이너별로 포트를 세팅하여 한꺼번에 실행할 수 있다는 이점이 있습니다.

1.2 컨테이너가 종료되면 항상 재시작

```
restart: always
```

컨테이너가 종료되는 이유는 다양합니다. 웹 서비스를 개발할 때 컨테이너가 종료된다면 아주 높은 확률로 웹서버가 오류에 의해서 종료 되었을 경우임을 추측 해볼 수 있습니다.

각 서비스는 특정 포트를 통해 서로와 통신하고자 합니다. 이렇게 특정 포트에서 다른 서비스와 통신하려고 하는 상태를 listening 이라고 하죠. 만약 listening 대상의 서비스가 완벽히 준비가 되어 있지 않아 연결이 불가능하다면 다시 웹서버를 재시작해서 상대 서비스가 준비가 완료되면 통신을 시작할 수 있도록 설정해주어야 합니다.

1.3 환경변수 3가지 설정

```
environment: - POSTGRES_USER=admin - POSTGRES_PASSWORD=postgres -
POSTGRES_DB=postgres
```

환경변수는 각 컨테이너 환경에서 사용하고자 하는 고정된 값을 할당하기 위해 사용합니다. 일반적으로 docker hub에 공개된 공식 이미지의 경우 다음과 같이 어떤 환경변수의 설정이 필수이고 선택인지를 명시하고 있습니다.

ex 1) PostgreSQL

Environment Variables

The PostgreSQL image uses several environment variables which are easy to miss. The only variable required is `POSTGRES_PASSWORD`, the rest are optional.

Warning: the Docker specific variables will only have an effect if you start the container with a data directory that is empty; any pre-existing database will be left untouched on container startup.

`POSTGRES_PASSWORD`

This environment variable is required for you to use the PostgreSQL image. It must not be empty or undefined. This environment variable sets the superuser password for PostgreSQL. The default superuser is defined by the `POSTGRES_USER` environment variable.

Note 1: The PostgreSQL image sets up `trust` authentication locally so you may notice a password is not required when connecting from `localhost` (inside the same container). However, a password will be required if connecting from a different host/container.

Note 2: This variable defines the superuser password in the PostgreSQL instance, as set by the `initdb` script during initial container startup. It has no effect on the `PGPASSWORD` environment variable that may be used by the `psql` client at runtime, as described at https://www.postgresql.org/docs/current/libpq-envars.html. `PGPASSWORD`, if used, will be specified as a separate environment variable.

`POSTGRES_USER`

This optional environment variable is used in conjunction with `POSTGRES_PASSWORD` to set a user and its password. This variable will create the specified user with superuser power and a database with the same name. If it is not specified, then the default user of `postgres` will be used.

Be aware that if this parameter is specified, PostgreSQL will still show `The files belonging to this database system will be owned by user "postgres"` during initialization. This refers to the Linux system user (from `/etc/passwd` in the image) that the `postgres` daemon runs as, and as such is unrelated to the `POSTGRES_USER` option. See the section titled "Arbitrary `--user` Notes" for more details.

`POSTGRES_DB`

This optional environment variable can be used to define a different name for the default database that is created when the image is first started. If it is not specified, then the value of `POSTGRES_USER` will be used.

ex 2) MYSQL

Environment Variables

When you start the `mysql` image, you can adjust the configuration of the MySQL instance by passing one or more environment variables on the `docker run` command line. Do note that none of the variables below will have any effect if you start the container with a data directory that already contains a database: any pre-existing database will always be left untouched on container startup.

See also https://dev.mysql.com/doc/refman/5.7/en/environment-variables.html for documentation of environment variables which MySQL itself respects (especially variables like `MYSQL_HOST`, which is known to cause issues when used with this image).

`MYSQL_ROOT_PASSWORD`

This variable is mandatory and specifies the password that will be set for the MySQL `root` superuser account. In the above example, it was set to `my-secret-pw`.

`MYSQL_DATABASE`

This variable is optional and allows you to specify the name of a database to be created on image startup. If a user/password was supplied (see below) then that user will be granted superuser access (corresponding to `GRANT ALL`) to this database.

`MYSQL_USER`, `MYSQL_PASSWORD`

These variables are optional, used in conjunction to create a new user and to set that user's password. This user will be granted superuser permissions (see above) for the database specified by the `MYSQL_DATABASE` variable. Both variables are required for a user to be created.

Do note that there is no need to use this mechanism to create the root superuser, that user gets created by default with the password specified by the `MYSQL_ROOT_PASSWORD` variable.

ex 3) NGINX

Using environment variables in nginx configuration (new in 1.19)

Out-of-the-box, nginx doesn't support environment variables inside most configuration blocks. But this image has a function, which will extract environment variables before nginx starts.

Here is an example using docker-compose.yml:

```
web:
  image: nginx
  volumes:
   - ./templates:/etc/nginx/templates
  ports:
   - "8080:80"
  environment:
   - NGINX_HOST=foobar.com
   - NGINX_PORT=80
```

By default, this function reads template files in `/etc/nginx/templates/*.template` and outputs the result of executing `envsubst` to `/etc/nginx/conf.d`.

So if you place `templates/default.conf.template` file, which contains variable references like this:

```
listen       ${NGINX_PORT};
```

outputs to `/etc/nginx/conf.d/default.conf` like this:

```
listen       80;
```

This behavior can be changed via the following environment variables:

- `NGINX_ENVSUBST_TEMPLATE_DIR`
  - A directory which contains template files (default: `/etc/nginx/templates`)
  - When this directory doesn't exist, this function will do nothing about template processing.
- `NGINX_ENVSUBST_TEMPLATE_SUFFIX`
  - A suffix of template files (default: `.template`)
  - This function only processes the files whose name ends with this suffix.

1.4 `/var/lib/postgresql/data` 를 `pgdata` 볼륨으로 할당

```
volumes: - pgdata:/var/lib/postgresql/data
```

볼륨을 할당하는 이유는 영속성 유지, 즉 서비스에서 생성된 데이터를 유실되지 않도록 보관하기 위함입니다. postgres 이미지로 컨테이너를 실행하면 `/var/lib/postgresql/data` 에 생성된 데이터가 저장되며, 호스트에 `pgdata` 라는 이름의 볼륨으로 경로를 할당해줍니다.

2. `pgadmin`

pgadmin은 PostgreSQL을 이용하기 위한 DB 툴입니다. MySQL Workbench, Oracle SQL Developer 등과 같은 역할을 하며, 본 과제에서는 컨테이너에서 이를 실행했습니다.

2.1 컨테이너가 종료되면 항상 재시작

```
restart: always
```

2.2 환경변수 2가지 설정

```
environment: PGADMIN_DEFAULT_EMAIL: admin@example.com
PGADMIN_DEFAULT_PASSWORD: admin
```

2.3 `/var/lib/pgadmin` 를 `pgadmindata` 볼륨으로 할당

```
volumes: - pgadmindata:/var/lib/pgadmin
```

pgadmin 내부에서 생성된 데이터를 `pgadmindata` 라는 이름의 볼륨으로 할당하였습니다.

3. `web`

3.1 템플릿으로 제공된 Dockerfile 을 빌드한 이미지 사용

```
build: .
```

```
build: context: . dockerfile: Dockerfile
```

레지스트리에서 이미지를 받아오지 않고 소스코드를 빌드하여 이미지로 사용하는 경우에는 `build` 를 사용합니다. 만약 `docker-compose.yml` 과 같은 경로에 `Dockerfile` 이 있다면 위와 같이 작성할 수 있으며, 다른 경로에 있다면 아래와 같이 `context` 옵션에 `docker-compose.yml` 을 기준으로하는 상대경로를 작성해주어야 합니다. `dockerfile` 옵션은 파일의 이름이 `Dockerfile` 이 아닌 경우에 파일명을 명시하면 해당 파일을 빌드하여 사용할 수 있도록 합니다.

3.2 컨테이너와 호스트는 각각 8000 포트로 통신

```
ports: - "8000:8000"
```

3.3 postgreSQL(db) 서비스가 시작된 이후에 시작

```
depends_on: - db
```

`depends_on` 은 컨테이너간 의존성을 설정하는 옵션입니다. 의존성에 대한 설정은 두 가지 케이스에 대해 영향을 줍니다.

첫 번째는 컨테이너의 실행/중지 순서입니다. 과제를 예시로 들자면 `web` 은 `db` 가 시작한 이후에 실행/중지 됩니다. 중요한 것은 컨테이너에 시작하라는 사인을 순서에 맞게 전달할 뿐 서비스가 접속 가능한 상태인지는 확인하지 않습니다.
두 번째는 의존성 컨테이너 동반 실행입니다. 만약 `docker-compose up web` 으로 `web` 이라는 서비스를 실행하는 경우 의존성으로 설정된 `db` 서비스까지 함께 실행됩니다.

3.4 컨테이너가 종료되면 항상 재시작

```
restart: always
```