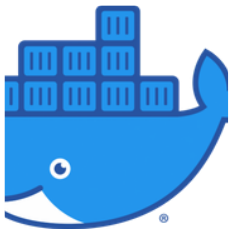
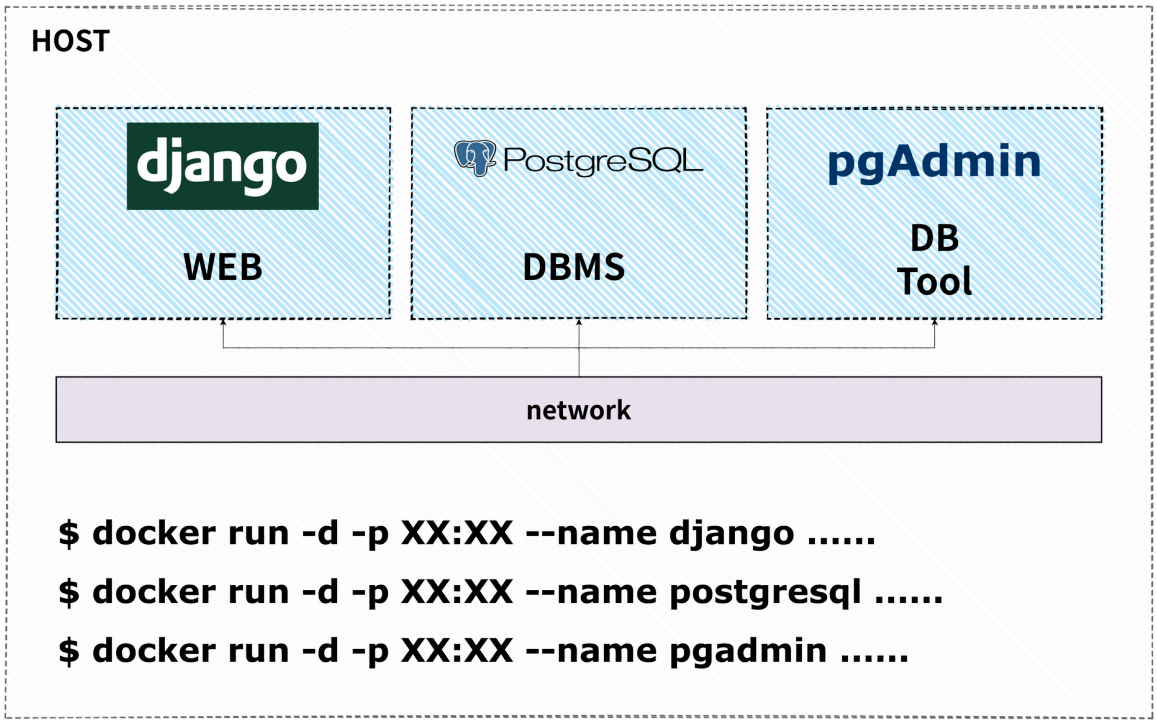


Ch 7. Docker Compose

7. Docker Compose

> 출처
> - Docker Compose : https://docs.docker.com/compose/

"Docker Compose"라는 새로운 용어가 등장 했습니다. Compose는 *구성하다* 라는 뜻을 가진 영어 단어인데, 갑자기 도커와 함께 어울리게 된 이유는 무엇일까요? 이 수수께끼를 풀기 위해서 간단한 그림을 보여 드리겠습니다.



Python 기반의 웹 프레임워크 **django**, 관계형 DBMS **Postgresql**, Postgresql 모니터링 **pgAdmin** 이 세 가지를 모두 한꺼번에 컨테이너로 구성하려고 합니다. 각각을 컨테이너로 구동해야 되니까... 이미지를 받아오고 `docker run` 을 실행하고...다시 또 실행하고...포트는 어떻게 연결하지? 세팅하려는 모든 컨테이너를 각각 구성하려니 명령어를 어떻게 실행해야 할지, 필요한 계정은 어떻게 설정해야 할지 하나도 감이 잡히지 않습니다.

그래서 위와 같은 멀티 컨테이너를 구동하기 위한 구원투수가 나타났으니 그 이름이 바로 **Docker Compose** 입니다. Docker Compose 는 도커 컴포넌트 중의 하나로서, 여러 개의 컨테이너를 정의하고 실행하는 역할을 합니다. 기존에 학습했던 이미지 빌드용 파일인 `Dockerfile` 과 더불어 **`docker-compose.yml`** 이라는 새로운 설정 파일이 등장하는데 사용법이 간결하고 직관적이기 때문에 크게 부담갖지 않으셔도 됩니다. 오히려 이렇게 쉽게 컨테이너를 제어할 수 있다는 것에 감탄하게 됩니다.

7.1 Docker Compose 설치 및 개요

7.1.1 Docker Compose 설치

Docker Compose 를 사용하기 위해 먼저 설치를 진행하겠습니다. 설치 과정은 매우 간단합니다.

> 💡 Ubuntu 가상머신이 아닌 Docker Desktop을 설치한 경우 이미 Docker Compose 가 내장되어 있어 별도로 설치하실 필요 없습니다.

1) Docker Compose 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
```

2) 실행 권한 부여

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

3) 버전 확인

```
docker-compose --version
```

```
ggingmin@ubuntu-server:~$ docker-compose --version
docker-compose version 1.29.2, build 5beceа4c
```

위와 같이 버전이 출력되면 정상적으로 설치된 것입니다. 이제 즐겁게 사용하는 일만 남았네요😁

7.1.2 docker-compose.yml

The Official YAML Web Site

YAML 1.2 --- YAML : YAML Ain't Markup Language™ What It Is : YAML is a human friendly data serialization standard for all programming languages.

https://yaml.org/

`*.yml` 혹은 `*.yaml` 이라는 파일 형식을 본 적이 있으신가요? 이는 `YAML Ain't Markup Language` 라는 문구의 약자를 따서 만들어졌다고 합니다. 원래는 `Yet Another Markup Language` 이라는 이름이었으나 이후 의미가 변경되었습니다. `JSON` 이나 `XML` 같이 시스템 간 데이터 교환을 위해 만들어 졌으며, `key-value` 구조를 기본으로 합니다. 이외 문법적 중요사항은 다음과 같습니다.

- **대소문자를 구분합니다.**
- **구조를 구분할 때 들여쓰기로 탭 대신 스페이스를 사용합니다.**
- **값으로는 문자열(string), 숫자(number), 불린(boolean) 을 모두 취할 수 있습니다.**
- **`:` 바로 뒤는 한 칸을 떼고 작성합니다.**
- **값을 나열하기 위해서는 `-` 를 입력한 후 한 칸을 떼고 사용합니다.**
- **주석 표기는 `#` 를 사용합니다.**

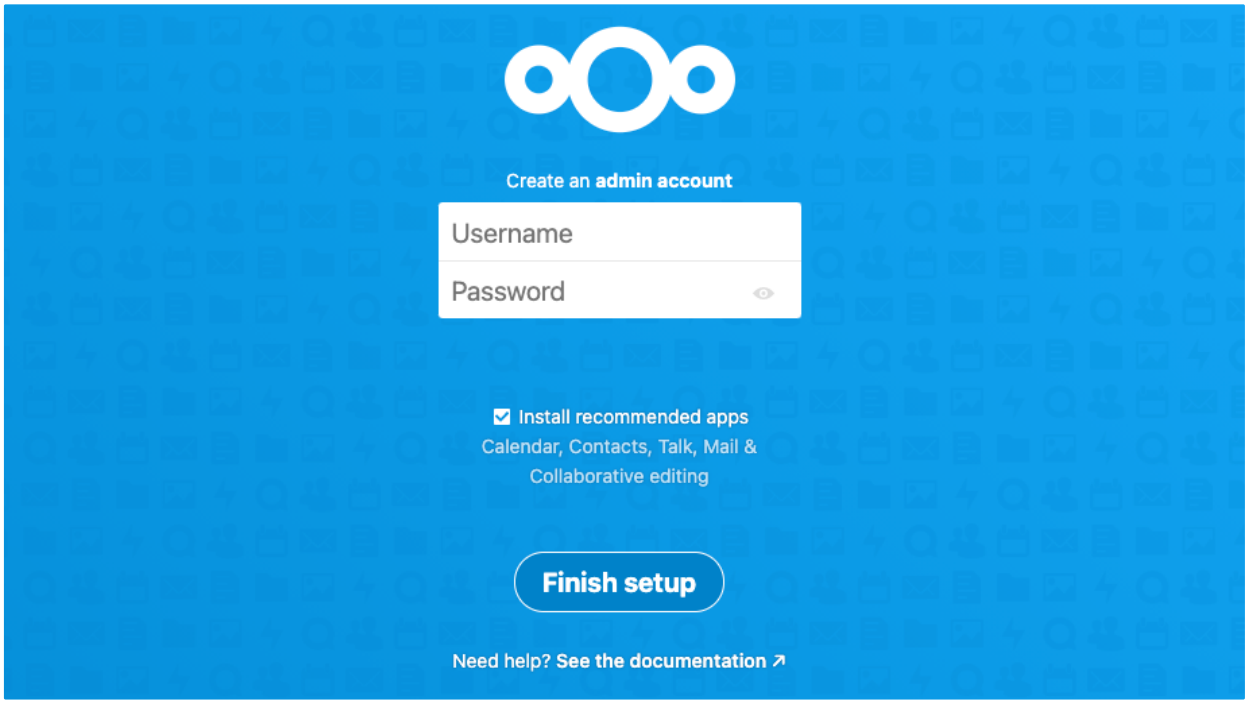
Docker Compose 를 이용하기 위한 설정파일인 `docker-compose.yml` 의 구조는 다음과 같습니다.

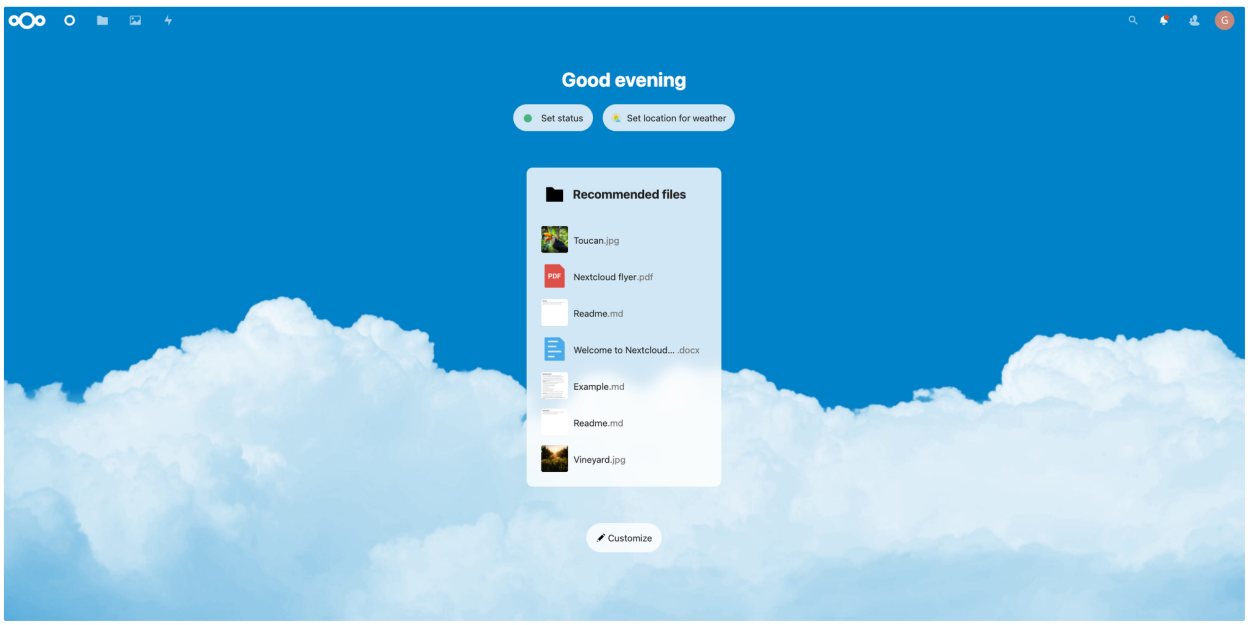
설치형 클라우드인 Nextcloud 와 관계형 DB postgresql 컨테이너를 함께 구성할 수 있도록 내용을 준비하였습니다.

```
version: "3.9" services: db: image: postgres:alpine environment: -
POSTGRES_PASSWORD=nextcloud - POSTGRES_DB=nextcloud - POSTGRES_USER=nextcloud
restart: always volumes: - db_data:/var/lib/postgresql/data nc: image:
nextcloud:apache environment: - POSTGRES_HOST=db - POSTGRES_PASSWORD=nextcloud
- POSTGRES_DB=nextcloud - POSTGRES_USER=nextcloud ports: - "80:80" restart:
always volumes: - nc_data:/var/www/html depends_on: - db volumes: nc_data:
db_data:
```

정상적으로 컨테이너가 구동된 이후 브라우저를 통해 접속하면 다음과 같이 관리자 계정을 생성하는 창이 나옵니다. 원하시는 값을 입력해주시고 하단의 `Finish setup` 버튼을 누르시면 됩니다.

계정생성이 완료되고 필요한 모듈의 설치가 완료되면 대시보드가 뜹니다. 이제 이 컨테이너를 여러분들만의 구글 드라이브, 네이버 클라우드 처럼 사용하실 수 있습니다.





7.2 Docker Compose 명령어

Docker Compose 에서 사용되는 명령은 기존에 사용하던 Docker 명령어와 비슷하지만 기능적으로 다른 것들이 있어 혼동하지 않도록 주의해야 합니다.

명령어는 기본적으로 `docker-compose.yml` 파일이 위치한 곳의 경로에서 실행합니다.

1) `up` - 멀티 컨테이너 생성 및 실행

컨테이너의 이름은 별도로 설정하지 않으면 `[docker-compose.yml 파일이 위치한 디렉토리명]_[서비스명]_[번호]` 의 형태로 정의됩니다.

▼ 옵션

up

Aa 명령어	이름	기능
	`-d, --detach`	컨테이너를 백그라운드에서 실행합니다.
	`--build`	컨테이너를 생성하기 전에 이미지를 빌드합니다.
	`--no-build`	실행 대상 이미지가 존재하지 않더라도 빌드하지 않습니다.
	`--abort-on-container-exit`	여러 컨테이너들 중 하나라도 종료되면 모두 종료됩니다. 주의❗ `--detach` 와 함께 사용할 수 없습니다.

개수 5

```
sudo docker-compose up
```

```
ggingmin@ubuntu_server:~/nextcloud$ sudo docker-compose up
Creating network "nextcloud_default" with the default driver
Creating volume "nextcloud_nc_data" with default driver
Creating volume "nextcloud_db_data" with default driver
Pulling nc (nextcloud:apache)...
apache: Pulling from library/nextcloud
99046ad9247f: Pull complete
3875fa64ab1e: Pull complete
e9329a8f553a: Pull complete
9bb327f9b0a4: Pull complete
051b56f0e6a3: Pull complete
da02d3111b48: Pull complete
98ca514d99e4: Pull complete
a4ff74d025cd: Pull complete
d094ddd610df: Pull complete
8adafae068ec: Pull complete
70fd73853886: Pull complete
23a25cfceae7: Pull complete
df2efaa2c941: Pull complete
a3057a184756: Pull complete
3d71491f2ea2: Pull complete
6164fe9a1ff1: Pull complete
5025772d3c9a: Pull complete
Digest: sha256:99d94124b2024c9f7f38dc12144a92bc0d68d110bcfd374169ebb7e8df0adf8e
Status: Downloaded newer image for nextcloud:apache
Pulling db (postgres:alpine)...
alpine: Pulling from library/postgres
a0d0a0d46f8b: Already exists
5034a66b99e6: Pull complete
82e9eb77798b: Pull complete
314b9347faf5: Pull complete
2625be9fae82: Pull complete
5ec8358e2a99: Pull complete
2e9ccfc29d86: Pull complete
2a4d94e5dde0: Pull complete
Digest: sha256:a70babcd0e8f86272c35d6efcf8070c597c1f31b3d19727eece213a09929dd55
Status: Downloaded newer image for postgres:alpine
Creating nextcloud_db_1 ... done
Creating nextcloud_nc_1 ... done
Attaching to nextcloud_db_1, nextcloud_nc_1
```

2) `ps` - 컨테이너 조회

`docker ps` 혹은 `docker container ls` 를 사용해도 실행 중인 컨테이너를 조회할 수 있습니다. 다만, `docker-compose ps` 를 통해 조회했을 때의 출력 양식에서 약간 차이가 납니다.

▼ 옵션

ps

Aa 명령어	이름	기능
	`-q, --quiet`	컨테이너 ID만 출력합니다.
	`--services`	정의된 서비스 명을 출력합니다.
	`-a, --all`	종료된 컨테이너를 포함하여 모든 컨테이너를 출력합니다.

개수 3

```
sudo docker-compose ps
```

```
ggingmin@ubuntu_server:~/nextcloud$ sudo docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATUS          PORTS                                   NAMES
690920bb2936   postgres:alpine    "docker-entrypoint.s…"   14 minutes ago   Up 11 seconds   5432/tcp                                nextcloud_db_1
8c6607c91407   nextcloud:apache   "/entrypoint.sh apac…"   14 minutes ago   Up 9 seconds    0.0.0.0:80->80/tcp, :::80->80/tcp   nextcloud_nc_1
ggingmin@ubuntu_server:~/nextcloud$ sudo docker-compose ps
     Name                   Command               State               Ports
-------------------------------------------------------------------------------------
nextcloud_db_1   docker-entrypoint.sh postgres    Up      5432/tcp
nextcloud_nc_1   /entrypoint.sh apache2-for ...   Up      0.0.0.0:80->80/tcp,:::80->80/tcp
```

3) `run` - 컨테이너 내부에서 명령 실행

명령어를 사용할 때 주의할 점은 `run` 명령어 실행의 인수를 컨테이너명이 아닌 `docker-compose.yml` 에 정의된 서비스명으로 입력해야 하는 것입니다.

```
sudo docker-compose run [서비스명] [실행 대상 명령] sudo docker-compose run db bash
```

```
ggingmin@ubuntu_server:~/nextcloud$ sudo docker-compose run db bash
Creating nextcloud_db_run ... done
bash-5.1#
```

4) `start` - 생성되어 있는 컨테이너 실행

```
sudo docker-compose start
```

```
ggingmin@ubuntu_server:~/nextcloud$ sudo docker-compose start
Starting nc ... done
Starting db ... done
```

5) `stop` - 생성되어 있는 컨테이너 종료

```
sudo docker-compose stop
```

```
ggingmin@ubuntu_server:~/nextcloud$ sudo docker-compose stop
Stopping nextcloud_db_1 ... done
Stopping nextcloud_nc_1 ... done
```

6) `down` - 컨테이너 종료 및 삭제

```
sudo docker-compose down
```

```
ggingmin@ubuntu_server:~/nextcloud$ sudo docker-compose down
Stopping nextcloud_db_1 ... done
Stopping nextcloud_nc_1 ... done
Removing nextcloud_db_run_6073673595df ... done
Removing nextcloud_db_1                ... done
Removing nextcloud_nc_1                ... done
Removing network nextcloud_default
```

7.3 docker-compose.yml 작성

1) `version`

```
version: "3.9"
```

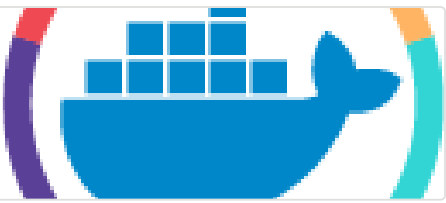
Docker Compose 파일은 최상단에 버전을 정의하도록 되어있습니다. 각 버전별로 명령어 혹은 표기법이 다르기 때문에 정상적으로 작동하지 않는 경우 버전을 체크해보아야 합니다.

더불어 설치된 도커 엔진 버전과의 호환성도 반드시 확인하여야 합니다.

Compose file version 3 reference

These topics describe version 3 of the Compose file format. This is the newest version. There are several versions of the Compose file format -

https://docs.docker.com/compose/compose-file/compose-file-v3/



2) `services`

```
services: db: ... nc: ...
```

서비스는 Compose 에서 실행할 컨테이너라고 생각하시면 이해가 빠릅니다. 각 서비스 별로 그에 맞는 환경의 컨테이너를 구성하기 위해 내부에 다양한 옵션을 추가합니다. 혼동하지 말아야 하는 것은 서비스명과 컨테이너명은 다른 개념이라는 것입니다.

만약 원하는 컨테이너명을 설정하고 싶다면 각 서비스 하위에 `container_name` 키와 설정하려는 값을 추가하면 됩니다.

3) `image`

```
services: db: image: postgres:alpine ... nc: image: nextcloud:apache ...
```

이미지는 말 그대로 컨테이너로 실행할 대상 이미지를 설정하는 것입니다. `Dockerfile` 에서 `FROM` 을 통해 베이스 이미지를 설정했던 것 기억 나시나요? 그 과정이 `image` 라는 키의 값을 작성하는 것으로 바뀌었다고 보시면 됩니다.

4) `build`

```
services: db: ... nc: build: context: . dockerfile: Dockerfile ...
```

이후 이어지는 Spring Boot + MariaDB 실습에서 다룰 예정이지만 간단히 기능을 살펴보겠습니다.

이미 로컬에 이미지가 있거나 Docker Hub에 이미지가 있다면 이미지명과 태그만으로 쉽게 내려받아 컨테이너를 구성할 수 있습니다. 하지만 일반적으로 작성한 `Dockerfile` 에서 빌드된 이미지를 기반으로 컨테이너를 실행하죠.

Compose 에서는 실행할 이미지명 대신에 빌드할 `Dockerfile` 의 정보를 입력하여 이를 빌드하고 바로 이미지로 사용할 수 있습니다. `dockerfile` 옵션을 사용하면 파일의 이름이 `Dockerfile` 이 아닌 것도 빌드 대상으로 지정할 수 있으며 경로 역시 동일 경로가 아닌 다른 경로를 지정할 수 있습니다.

5) `command`

```
services: db: ... nc: build: context: . dockerfile: Dockerfile command: java -
jar app.jar
```

생성된 컨테이너에 어떤 명령을 내릴지 세팅합니다. 보통 컴파일러나 특정 언어로 작성된 어플리케이션을 명령어로 실행해야 하는 경우에 사용됩니다.

6) `ports`

```
services: db: ... nc: ports: "80:80" ...
```

포트포워딩을 설정하는 항목으로 `docker run -p 80:80` 와 동일한 기능을 한다고 보시면 됩니다. 다만, yaml 파일에서는 `XX:YY` 의 형식이 시간값으로 해석될 수 있기 때문에 안전하게 따옴표 처리를 하시는 것을 권장합니다.

7) `depends_on`

```
services: db: ... nc: depends_on: - db ...
```

특정 서비스가 먼저 시작되면 이어서 시작할 수 있도록 설정하는 명령어 입니다. 위 예제 소스를 보면 `nc` 라는 서비스에 `db` 가 `depends_on` 으로 걸려있는데, 이는 `db` 서비스가 시작되면 `nc` 서비스가 시작되도록 순서를 정하는 것입니다. 다만, `db` 가 완전히 초기화 되어 리스닝 상태까지 도달 했는지는 확인하지 않습니다. 단순히 시작이 되었느냐, 아니냐 만을 가지고 서비스를 시작합니다.

8) `environment`

```
services: db: ... nc: environment: - POSTGRES_HOST=db -
POSTGRES_PASSWORD=nextcloud - POSTGRES_DB=nextcloud - POSTGRES_USER=nextcloud
...
```

환경변수를 설정하는 항목이며, DB 계정 및 초기 DB 세팅 등에 주로 사용됩니다. 이외 필요에 따라 각 컨테이너별 환경변수를 할당할 수 있습니다. `docker run -e` 와 유사한 기능이라고 볼 수 있습니다.

9) `volumes`

```
services: db: volumes: - db_data:/var/lib/postgresql/data ... nc: volumes: -
nc_data:/var/www/html ... volumes: nc_data: db_data: ...
```

볼륨을 세팅하는 항목이며, `docker run -v` 와 같은 역할을 합니다. 컨테이너가 삭제되어도 데이터 유실이 되지 않도록 호스트의 일부 영역을 할당하게 되며, `[볼륨명]:[할당할 호스트 경로]` 를 작성하면 됩니다. 더불어 `services` 와 같은 위계에 `volumes` 를 작성하고 그 하위에 서비스에 설정한 볼륨명을 작성해줍니다.

10) `restart`

```
services: db: ... nc: ... restart: always ...
```

서비스 재시작을 설정할 수 있습니다. 기본값은 재시작을 하지 않는 것이지만 웹서비스의 경우 재시작을 `always` 로 설정하는 경우가 많습니다. 이는 `db` 가 완전히 리스닝이 가능한 상태가 되기 전에 웹 서비스에서 Connection을 생성하려는 경우 에러가 발생해 서비스가 비정상 종료가 되기 때문입니다. 재시작을 하다가 `db` 가 Connection 생성이 가능한 상태가 되면 웹 서비스가 정상적으로 실행됩니다.

11) `expose`

```
services: db: ... expose: - 5432 ... nc: ...
```

`Dockerfile` 에도 `EXPOSE` 라는 명령어가 있었죠. 실제로 포트포워딩을 수행하지 않고 어떤 포트를 개방해야 하는지 명시하는 역할을 했었습니다. `docker-compose.yml` 에서는 조금 다른 역할을 하기 때문에 주의해야 합니다.

`expose` 에서 지정된 포트를 통해 통신을 가능케 하는 것은 맞습니다. 하지만, 호스트 OS 에서의 접근은 불가능하고 연결된 타 서비스(컨테이너)와의 통신만 가능합니다. Compose 에서 연결된 서비스라는 것은 `docker-compose.yml` 에 작성된 `services` 하위의 항목을 가리킵니다. 이들은 동일한 네트워크 대역에 위치하므로 기본적으로 통신이 가능한 상태입니다.

7.4 Spring Boot + MariaDB

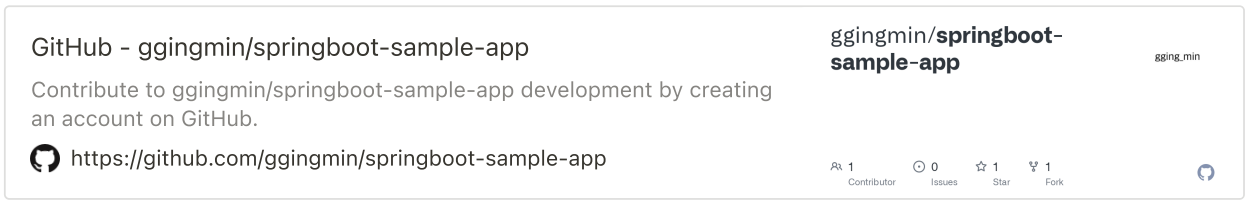
Docker Compose 로 좀 더 실용적인 프로젝트를 하나 준비했습니다. 바로 실무에서 많이 사용되는 Spring Boot 와 MariaDB로 API를 구성한 것인데요, 사용된 언어와 프레임워크 버전 등은 다음과 같습니다.

> 💡 **Spring Boot + MariaDB**
>
> Java : OpenJDK 8
> Spring Boot : 2.5.4
> MariaDB : 10.6.4

1) 예제소스 받기

템플릿 소스 코드는 다음의 명령을 통해 내려받으실 수 있습니다.

```
git clone https://github.com/ggingmin/springboot-sample-app.git
```



2) 환경변수

실습에 필요한 환경변수는 다음과 같습니다. 각 서비스에 맞게 `docker-compose.yml` 내에 작성해주어야 정상적으로 컨테이너가 작동합니다.

```
services: mariadb: ... environment: - MARIADB_ROOT_PASSWORD=root -
MARIADB_DATABASE=testdb ... sampleapp: ... environment: -
MARIADB_ADDRESS=mariadb - MARIADB_USERNAME=root - MARIADB_PASSWORD=root ...
```