7. 여러 컨테이너의 운용 관리

- Docker에서 웹 어플리케이션을 환경 제품으로 운용할 때는 어플리케이션 서버, 로그 서버, 프록시 서버 등 여러 개의 컨테이너를 연계하여 작동

7.1 여러 컨테이너 관리의 개요

---

1. 웹 3계층 시스템 아키텍처

- 인프라 아키텍처
  - 여러 개의 서버에 기능과 역할을 분할 구성
- 어플리케이션 개발 기술 및 플랫폼 기술 기술에 정통한 IT 아키텍트를 중심으로 인프라 처리 방식을 결정
- 웹 3계층 아키텍처는 웹 어플리케이션의 대표적인 인프라 아키텍처 중 하나

프론트 서버

- 클라이언트에게 HTTP 요청/응답을 하는 서버 기능을 가짐
- 웹 프로트 서버 또는 웹 서버라 불림
- Nginx, IIS(Internet Information Server) 등 사용
- 부하가 많이 생길 수 있으며, 로드밸런스 기능 필요

어플리케이션 서버

- 서버의 업무를 처리를 실행
- 결제 처리, 수주 처리 등

데이터베이스(DB) 서버

- 영구 데이터를 관리하기 위한 서버
- RDBMS(Relational Database Management System)인 MySQL, PostgreSQL, Oracle Database 등 사용
- NoSQL은 병렬분산처리 및 대량의 데이터 축적에 이점(사용자의 액세스 처리가 많은 온라인 시스템에 많이 사용)
  - 주요 방식 : KVS(Key-Value 스토어), 도큐먼트 데이터베이스 등
  - Redis, MongoDB 등이 존재
- 높은 가용성이 요구 -> 클러스터링 및 다중화 기술 사용
- 장애 대비 -> 데이터 백업, 원격지 보관 등 사용
- 많은 부하, 병목현상 -> 운용 상황에 따른 퍼포먼스 튜닝 필요

클라우드 시스템에서는 오토스케일 기능 사용 권장

---

2. 영구 데이터 관리

- 영구 데이터는 적절히 관리할 필요함

데이터 백업 및 복원

- 적절한 보안 대책이나 운영 규칙에 따라 데이터 백업
- 클라우드 서비스를 이용한 해외 저가의 데이터센터에 백업 가능

로그 수집

- 여러 개의 서버로 된 분산 환경에서는 로그 수집 전용 서버를 권장

하나의 컨테이너에는 하나의 프로세스를 구성하는 것이 대원칙!

- 컨테이너에 저장 : 어플리케인션 실행 모듈, 각종 라이브러리의 모듈, 미들웨어 설정 파일 등
- 별도의 인프라 아키텍처에 저장 : 시스템 가동 후 생성되는 영구 데이터

---

3. Docker Compose

- 여러 컨테이너를 모아서 일괄적으로 관리할 수 있는 툴
- 'docker-compose.yml'라는 파일에 컨테이너의 구성 정보를 정의
  - 웹 어플리케이션의 의존관계(DB, 큐, 캐시, 어플리케이션 등)를 모아서 설정
  - 여러 개의 컨테이너를 모아서 시작/정지 가능

7.2 웹 어플리케이션을 로컬에서 움직여 보자

- 전에 받은 샘플 파일 사용

```
git clone https://github.com/asashiho/dockertext2
cd dockertext2/chap07/
```

---

1. Compose 구성파일의 작성

Ex) Dockerfile

```
# Base Image
FROM python:3.6

# Maintainer
LABEL maintainer "Shiho ASA"

# Upgrade pip
RUN pip install --upgrade pip

# Install Path
ENV APP_PATH /opt/imageview

# Install Python modules needed by the Python app
COPY requirements.txt $APP_PATH/
RUN pip install --no-cache-dir -r $APP_PATH/requirements.txt

# Copy files required for the app to run
COPY app.py $APP_PATH/
COPY templates/ $APP_PATH/templates/
COPY static/ $APP_PATH/static/

# Port number the container should expose
EXPOSE 80
```

```
# Run the application
CMD ["python", "/opt/imageview/app.py"]
```

Ex) Compose 정의 파일 확인

```
root@Ubuntu [19시 36분 01초] [~/dockertext2/chap07] [master *]
-> # cat docker-compose.yml
version: '3.3'
services:
  # WebServer config
  webserver:
    build: .
    ports:
     - "80:80"
    depends_on:
     - redis

  # Redis config
  redis:
    image: redis:4.0
```

- 'webserver'와 'redis'라는 이름의 서비스 2개를 정의
- webserver
    1. Dockerfile에 정의한 구성에 따라 이미지 build
    2. 외부:내부 port 80번으로 공개하여 컨테이너 시작
    3. 'redis' 서비스에 의존(depends_on)
- redis
    1. Docker Hub의 공식 이미지인 redis는 버전 4.0을 베이스 이미지로 사용하여 컨테이너 시작

---

2. 여러 Docker 컨테이너 시작

- Docker Compose : 이미지 다운로드나 빌드를 하나의 명령으로 모두 실행
- 'docker-compos' 설치
    - `apt  install docker-compos`

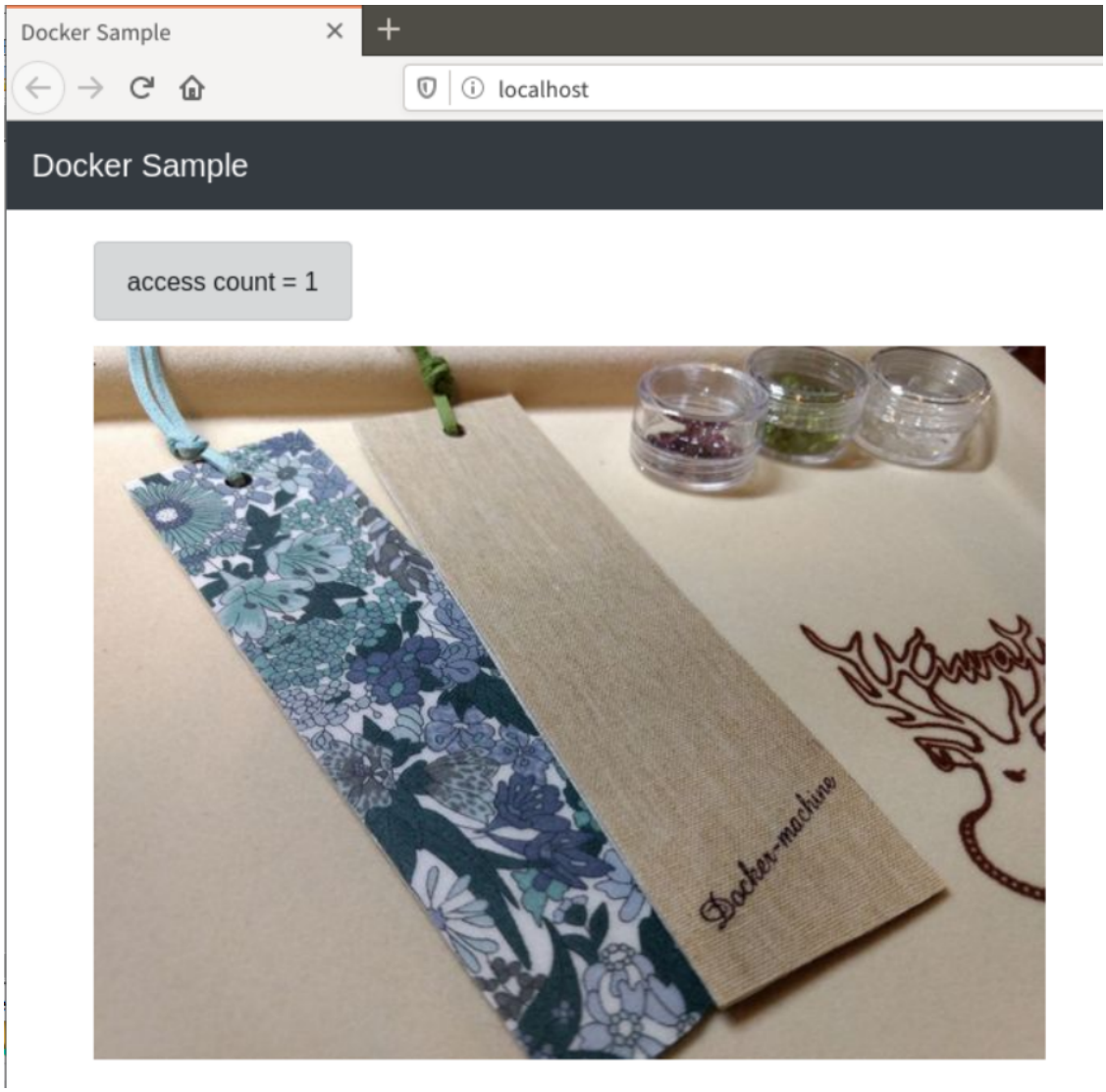
Ex) 샘플 어플리케이션의 컨테이너 시작

```
root@Ubuntu [21시 02분 50초] [~/dockertext2/chap07] [master *]
-> # docker-compose up
Creating network "chap07_default" with the default driver
Pulling redis (redis:4.0)...
~생략~
Building webserver
Step 1/11 : FROM python:3.6
3.6: Pulling from library/python
~생략~
Successfully built 9b7cc1d07312
Successfully tagged chap07_webserver:latest
WARNING: Image for service webserver was built because it did not already exist.
To rebuild this image you must use `docker-compose build` or `docker-compose up
--build`.
Creating chap07_redis_1 ...
```

```
Creating chap07_redis_1 ... done
Creating chap07_webserver_1 ...
Creating chap07_webserver_1 ... done
Attaching to chap07_redis_1, chap07_webserver_1
redis_1      | 1:C 08 Mar 12:13:13.815 # oO0OoO0OoO0Oo Redis is starting
oO0OoO0OoO0Oo
redis_1      | 1:C 08 Mar 12:13:13.818 # Redis version=4.0.14, bits=64,
commit=00000000, modified=0, pid=1, just started
redis_1      | 1:C 08 Mar 12:13:13.818 # Warning: no config file specified,
using the default config. In order to specify a config file use redis-server
/path/to/redis.conf
redis_1      | 1:M 08 Mar 12:13:13.820 * Running mode=standalone, port=6379.
redis_1      | 1:M 08 Mar 12:13:13.820 # WARNING: The TCP backlog setting of 511
cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower
value of 128.
redis_1      | 1:M 08 Mar 12:13:13.820 # Server initialized
redis_1      | 1:M 08 Mar 12:13:13.820 # WARNING overcommit_memory is set to 0!
Background save may fail under low memory condition. To fix this issue add
'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the
command 'sysctl vm.overcommit_memory=1' for this to take effect.
redis_1      | 1:M 08 Mar 12:13:13.820 # WARNING you have Transparent Huge Pages
(THP) support enabled in your kernel. This will create latency and memory usage
issues with Redis. To fix this issue run the command 'echo never >
/sys/kernel/mm/transparent_hugepage/enabled' 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_1      | 1:M 08 Mar 12:13:13.820 * Ready to accept connections
~생략~
webserver_1  |  * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
webserver_1  |  * Restarting with stat
webserver_1  |  * Debugger is active!
webserver_1  |  * Debugger PIN: 209-120-066
```

1. Docker Hub로부터 redis:4.0의 이미지가 먼저 다운됨
2. webserver에서 사용할 이미지가 Dockerfile을 바탕으로 Biuld됨
3. 'redis' -> 'webserver'순으로 서비스 컨테이너 동작

- 컨테이너 동작 확인



Ex) 컨테이너 확인

```
chap07 (master*) # docker-compose ps                              ~/dockertext2/chap07
       Name                     Command               State          Ports
-------------------------------------------------------------------------------------
chap07_redis_1       docker-entrypoint.sh redis ...   Up      6379/tcp
chap07_webserver_1   python /opt/imageview/app.py     Up      0.0.0.0:80->80/tcp
```

- 'docker-compose.yml' 또는 'docker-compose.yaml'이 존재하는 디렉터리에서 명령 실행

---

3. 여러 Docker 컨테이너 정지

- 'docker-compose.yml' 또는 'docker-compose.yaml'이 존재하는 디렉터리에서 명령 실행

Ex) 컨테이너 정지

```
[21:45:52] root:chap07 git:(master*) # docker-compose stop
Stopping chap07_webserver_1 ... done
Stopping chap07_redis_1     ... done
```

Ex) 리소스 삭제

```
[21:50:51] root:chap07 git:(master*) # docker-compose down
Removing chap07_webserver_1 ... done
Removing chap07_redis_1     ... done
Removing network chap07_default
```

7.3 Docker Compose를 사용한 여러 컨테이너의 구성 관리

---

1. docker-compose.yml의 개요.

Compose 정의 파일

- 시스템 안에서 가동하는 여러 서버들의 구성을 모아서 정의
- YAML 형식으로 기술
- YAML
  - 구조화된 데이터를 표현하기 위한 데이터 포맷
  - 들여쓰기로 데이터 계층 구조를 나타냄
  - '탭'이 아니라 '스페이스'로 들여쓰기 사용
  - 데이터 맨 앞에 '-'를 붙이면 배열을 나타냄
- 확장자 : ".yml", ".yaml"
- 컨테이너의 서비스(services:), 네트워크(networks:), 볼륨(volumes:)을 정의
- 버전에 따라애 따라 기술 가능 항목이 다름(버전 지정이 없으면, 1.0 버전으로 작동)
- 버전 관계

Compose 정의 파일의 버전	Docker Engine의 버전
3.7	18.06.0+
3.6	18.02.0+
3.5	17.12.0+
3.4	17.09.0+
3.3	17.06.0+
3.2	17.04.0+
3.1	1.13.1+
3.0	1.13.0+
2.4	17.12.0+
2.3	17.06.0+
2.2	1.13.0+
2.1	1.12.0+
2.0	1.10.0+
1.0	1.9.1.+

- 간단하게 Version 1, 2, 3으로만 나누어 사용 가능

Ex) compose 정의 예

```
# 버전을 지정
version: '3'

# 서비스 정의
services:
  webserver:
    image: ubuntu
    ports:
     - "80:80"
    networks:
     - webnet
    depends_on:
     - redis

  redis:
    image: redis:4.0
    networks:
     - webnet

# 네트워크 정의
networks:
  webnet:

# 데이터 볼륨 정의
volumes:
```

```
  data-volume:
```

2.1 이미지 지정

- image: 이미지의 이름 또는 이미지 ID 중 하나를 지정
- 로컬 환경 -> Docker Hub 순으로 베이스 이미지 자동 사용
- 태그 지정이 없으렴 최신 버전(latest) 사용(제품 환경에서는 버전을 지정 권장)

Ex) 이미지 태그 지정

```
services:
  webserver:
    image: asashiho/dockersample:1.0
```

2.2 이미지 빌드

- build : Dockerfile의 파일 경로 지정
- Dockerfile에 따라 자동으로 빌드
- 일반적으로 'Dockerfile'과 'docker-compose.yml'은 같은 디렉터리에 존재

Ex) build 지정

```
services:
  webserver:
    build: .    # 피리어드로 커런트 디렉터리를 나타냄
```

Ex) Dockerfile 작성

```
FROM ubuntu
```

- test용 Dockerfile

Ex) 컨테이너 생성

```
root@Ubuntu ~/test % docker-compose up --build
Creating network "test_default" with the default driver
Building webserver
Step 1/1 : FROM ubuntu
 ---> 72300a873c2c

Successfully built 72300a873c2c
Successfully tagged test_webserver:latest
Creating test_webserver_1 ...
Creating test_webserver_1 ... done
Attaching to test_webserver_1
test_webserver_1 exited with code 0
```

- 실행 할 때는 '3.x'버전으로 설정 - Ex) '3.3'
- 자동으로 build되어 컨테이너 생성
- `--build` : 이미지를 새로 build함

Ex) Dockerfile과 컨텍스트 지정

```
services:
  webserver:
    build:
      context: /data
      dockerfile: Dockerfile-alternate
```

- '/date'에 저장되어 있는 'Dockerfile-alternate'라는 이름의 'Dockfile'을 build
- 절대 경로 및 상대 경로 지정 가능
- context : 디렉터리 경로 및 Git repository의 URL 지정

Ex) build 인수 지정

```
services:
  webserver:
    build:
      args:
        projectno: 1
        user: asa
```

- 'projectno=1', 'user=asa'라는 값을 build할 때의 변수로 전달
- bool 연산자(true/false/yes/no)의 경우 ""(따옴표) 사용
- 변수값은 Docker Compose를 실행하는 머신 위에서만 유효

2.3 컨테이너 안에서 작동하는 명령 지정

- command : 컨테이너에서 작동하는 명령
- 베이스 이미지에 지정된 명령이 있으면 덮어씀

Ex) 컨테이너 안에서 작동하는 명령 지정

```
command: /bin/bash
```

Ex) entrypoint 지정

```
entrypoint:
    - php
    - -d
    - memory_limit=-1
```

- entrypoint도 덮어쓸 수 있음

2.4 컨테이너 간 연결

- links : 다른 컨테이너에 대한 링크
- 버전 3부터는 links 항목을 사용하지 않더라도 한 네트워크 안에 있는 서비스끼리는 서로 통신이 가능
- 컨테이너명과 별도로 alias명를 붙이는게 가능(서비스명:alias명)

Ex) links 지정

```
links:
    - logserver
    - logserver:log01
```

2.5 컨테이너 간 통신

- ports : 컨테이너가 공개하는 포트 지정
- '호스트머신의 포트 번호:컨테이너의 포트 번호'
- '컨테이너의 포트 번호'만 사용
    - 호스트 머신의 포트 번호 : 랜덤값
- YAML은 xx:yy 형식을 시간으로 해석
    - 포트 설정시 ""(따옴표)로 묶기

Ex) 공개 포트 지정

```
ports
  - "3000"
  - "8000:8000"
  - "49100:22"
  - "127.0.0.1:8001:8001"
```

Ex) 컨테이너 내부에만 공개하는 포트 지정

```
expose:
  - "3000"
  - "8000"
```

- pexose: 링크 기능을 사용하여 연결하는 컨테이너에게만 포트를 공개 할때 사용(외부 노출 X)
- 로그 서버와 같이 호스트 머신에 직접 액세스하지 않고, 웹 어플리케이션 서버 기능을 갖고 있는 컨테이너를 경유해서만 액세스하고 싶은 경우 등에 사용
- 설정한 포트로는 호스트에서는 접근할 수 없음

2.6 서비스 의존관계 정의

- depends_on : 여러 서비스의 의존관계를 정의

Ex) 의존관계 지정

```
services:
  webserver:
    build: .
    deponds_on:
      - db
      - redis
  redis:
    image: redis
  db:
    image: postgres
```

- db -> redis -> webserver 순으로 시작
- **시작 순서만 제어할 뿐, 어플리케이션이 이용 가능 할 때까지 기다리는 제어 없음**
    - 즉, 의존관계에 있는 'db' 서비스의 준비가 끝날 때까지 기다리지 않음
    - 어플리케이션 측에서 이에 대한 대책이 필요

2.7 컨테이너 환경변수 지정

- environment : 컨테이너 안의 환경변수 지정

- YAML 배열 형식 또는 해시 형식 중 하나로 변수 지정

Ex) 환경변수 지정

```
# 배열 형식으로 지정
environment:
  - HOGE=fuga
  - FOO

# 해시 형식으로 지정
environmet:
  HOGE: fuga
  FOO:
```

Ex) envfile

```
HOGE=fuga
FOO=bar
```

- 설정할 환경변수가 많으면, 다른 파일에 환경변수를 정의하고, 그 파일을 읽어 들일 수 있음
- 일반적으로 '환경변수 파일'(envfile)과 'docker-compose.yml'은 같은 디렉터리에 존재

Ex) 환경변수 파일 읽어 들이기

```
env_file: envfile
```

- env_file : 환경변수 파일을 읽어 들임

Ex) 여러 개의 환경변수 파일 읽어 들이기

```
env_file:
  - ./envfile
  - ./app/envfile2
  - /tmp/envfile3
```

어플리케이션 안에서 사용하는 API 키와 같은 비밀정보의 관리는 컨테이너 오케스트레이션 툴의 기능을 사용하는 것을 권장(Kubernetes 등)

2.8 컨테이너 정보 설정

- container_name : Docker Compose로 생성되는 컨테이너에 이름을 붙임
- Docker 컨테이너명은 고유해야하므로 커스텀명을 지정하면 여러 컨테이너로 스케일랗 수 없어짐

Ex) 컨테이너명 지정

```
container_name web-container
```

Ex) 컨테이너 라벨 설정

```
# 배열 형식으로 지정
labels:
  - "com.example.description=Accounting webapp"
  - "com.example.department=Finance"

# 해시 형식으로 지정
labels:
  com.example.description: "Accounting webapp"
  com.example.department: "Finance"
```

- labals : 컨테이너에 라벨을 붙임
- 설정한 라벨 확인은 `docker-compose config` 명령 사용

2.9 컨테이너 데이터 관리

- volumes : 컨테이너에 볼륨 마운트
- 일반적으로는 현재 디렉터리를 불륨으로 마운트

Ex) 볼륨 지정

```
volumes:
  - /var/lib/mysql
  - cache/:/tmp/cache
```

- '호스트의 디렉터리 경로:컨테이너의 디렉터리 경로' 가능

Ex) 읽기전용 볼륨 지정

```
volumes:
  - ~/configs:/etc/configs/:ro
```

- ro : 읽지 전용으로 마운트
- 설정 파일이 저장된 볼륨 등 쓰기 금지를 원할 때 사용

Ex) 볼륨 마운트 지정

```
volumes_from:
    - log
```

volumes_from : 다른 서비스나 컨테이너의 전체 볼륨을 마운트

- log라는 이름의 컨테이너로 마운트

7.4 Docker Compose를 사용한 여러 컨테이너의 운용

---

1. Docker Compose의 버전 확인

Ex) Docker Compose의 버전 확인

```
root@Ubuntu ~/test % docker-compose --version
docker-compose version 1.17.1, build unknown
```

---

2. Docker Compose의 기본 명령

- `docker-compose` 명령 사용을 위해서는 'docker-compose.yml' 또는 'docker-compose.yaml' 파일이 필요(같은 디렉터리내에 존재)
    - '-f' 옵션을 사용해 정의 파일명 지정 가능
- Docker Compose의 주요 서브 명령

서브 명령	설명
up	컨테이너 생성/시작
ps	컨테이너 목록 표시
logs	컨테이너 로그 출력
run	컨테이너 실행
start	컨테이너 시작
stop	컨테이너 정지
restart	컨테이너 재시작
pause	컨테이너 일시정지
unpause	컨테이너 재가동
port	공개 포트 번호 표시
config	구성확인
kill	실행중인 컨테이너 강제 중지
rm	컨테이너 삭제
down	리소스 삭제

Docker Compose 명령

https://docs.docker.com/compose/reference/

Ex) docker-compose.yml을 바탕으로 컨테이너 생성/시작

```
docker-compose -f ./sample/docker-compose.yml up
```

Ex) 특정 컨테이너 조작

```
docker-compose stop webserver
```

3.1 여러 컨테이너 생성

`docker-compose up [옵션] [서비스명 ...]`

* 주요 옵션

-d : 서비스 실행 후 콘솔로 빠져나옴(백그라운드 실행)

--no-deps : 링크 서비스를 시작하지 않음

--force-recreate: 컨테이너를 지우고 새로 만듬

--build : 서비스 시작 전 이미지를 새로 만듬(Dockerfile 필요)
--no-build : 이미지를 새로 만들지 않음
-t, --timeout : 컨테이너의 타임아웃을 초로 지정(기본값:10초)
--scale SERVICE=서비스 수 : 서비스 수를 지정

Ex) docker-compose.yml

```
version: '3.3'

services:
  server_a:
    image: nginx

  server_b:
    image: redis
```

Ex) 여러 컨테이너의 일괄 생성 및 시작

```
oot@Ubuntu ~/test % docker-compose up
Pulling server_b (redis:latest)...
latest: Pulling from library/redis
68ced04f60ab: Already exists
7ecc253967df: Already exists
765957bf98d4: Already exists
52f16772e1ca: Pull complete
2e43ba99c3f3: Pull complete
d95576c71392: Pull complete
Digest: sha256:6b9920bdc913ebeeed5cd5327decabe9fa829de425b52b3f28a7215ee7c7c457
Status: Downloaded newer image for redis:latest
Creating test_server_a_1 ...
Creating test_server_b_1 ...
Creating test_server_a_1
Creating test_server_b_1 ... done
Attaching to test_server_a_1, test_server_b_1
```

- [Ctrl + C]로 컨테이너 정지 가능

Ex) 여러 컨테이너를 백그라운드 시작

```
root@Ubuntu ~/test % docker-compose up -d
Starting test_server_a_1 ...
Starting test_server_a_1
Starting test_server_b_1 ...
Starting test_server_b_1 ... done
```

Ex) Docker 이미지 빌드

```
docker-compose up --build
```

- 컨테이너 시작 시 Dockerfile을 가지고 build

Ex) 컨테이너 갯수 지정
`docker-compose up --scale [서비스명=수]`

```
root@Ubuntu ~/test % docker-compose up --scale server_a=10 --scale server_b=20
Starting test_server_a_1 ...
~생략~
Creating test_server_a_7
Starting test_server_b_1 ... done
~생략~
Creating test_server_b_20 ... done
~생략~
```

- server_a는 10개, server_b는 20개로 컨테이너 시작
  - 삭제 : "docker container rm `docker container ls -a -f name=server -q`"

3.2 여러 컨테이너 확인

`docker-compose ps [옵션] [서비스명]`
`docker-compose logs [옵션] [서비스명]`

Ex) 여러 컨테이너의 상태 확인

```
root@Ubuntu ~/test % docker-compose ps
     Name                    Command               State    Ports
-------------------------------------------------------------------
test_server_a_1   nginx -g daemon off;             Up      80/tcp
test_server_b_1   docker-entrypoint.sh redis ...   Up      6379/tcp
```

Ex) 컨테이너 ID 확인

```
root@Ubuntu ~/test % docker-compose ps -q
c8e605bc26b105efbb5f02bab021e784161b705b44e756b842590287ffa7d02d
a442e341cad9327caf9365c0e7fcd56b99278193f9c4f392357a80cddb59a0ca
```

Ex) Docker 명령을 사용한 컨테이너 확인

```
root@Ubuntu ~/test % docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS               NAMES
a442e341cad9        redis               "docker-entrypoint.s…"   About a minute
ago   Up About a minute   6379/tcp            test_server_b_1
c8e605bc26b1        nginx               "nginx -g 'daemon of…"   About a minute
ago   Up About a minute   80/tcp              test_server_a_1
```

Ex) Docker 명령을 사용한 로그 확인

```
root@Ubuntu ~/test % docker-compose logs
Attaching to test_server_b_1, test_server_a_1
server_b_1  | 1:C 09 Mar 2020 07:19:04.584 # oOOoooOOooOOoo Redis is starting
oOOoooOOooOOoo
server_b_1  | 1:C 09 Mar 2020 07:19:04.584 # Redis version=5.0.7, bits=64,
commit=00000000, modified=0, pid=1, just started
~생략~
server_b_1  | 1:M 09 Mar 2020 07:19:14.126 * DB loaded from disk: 0.000 seconds
server_b_1  | 1:M 09 Mar 2020 07:19:14.126 * Ready to accept connections
```

3.3 컨테이너에서 명령 실행

`docker-compose run 서비스명 [명령] [인수]`

Ex) 컨테이너에서 명령 실행

```
root@Ubuntu ~/test % docker-compose run server_a /bin/bash
root@a99ab456b20f:/#
```

- `docker-compose up`을 통해 시작한 'server_a'라는 컨테이너에서 '/bin/bash' 실행

3.4 여러 컨테이너 시작/정지/재시작

`docker-compose start [옵션] [서비스명]`
`docker-compose stop [옵션] [서비스명]`
`docker-compose restart [옵션] [서비스명]`

Ex) 컨테이너 일괄 시작/정지/재시작

```
root@Ubuntu ~/test % docker-compose start
Starting server_a ... done
Starting server_b ... done
root@Ubuntu ~/test % docker-compose stop
Stopping test_server_b_1 ... done
Stopping test_server_a_1 ... done
root@Ubuntu ~/test % docker-compose restart
Restarting test_server_b_1 ... done
Restarting test_server_a_1 ... done
```

Ex) 특정 컨테이너 재시작

```
root@Ubuntu ~/test % docker-compose restart server_a
Restarting test_server_a_1 ... done
```

3.5 여러 컨테이너 일시정지/재개

`docker-compose pause [서비스명]`
`docker-compose unpause [서비스명]`

Ex) 컨테이너 일시정지/재개

```
root@Ubuntu ~/test % docker-compose pause
Pausing test_server_a_1 ... done
Pausing test_server_b_1 ... done
root@Ubuntu ~/test % docker-compose unpause
Unpausing test_server_b_1 ... done
Unpausing test_server_a_1 ... done
```

3.6 서비스 구성 확인

`docker-compose port [옵션] <서비스명> <프라이빗_포트_번호>`

- 서비스의 공개용 포트 확인
  - 주요 옵션
    --protocol=proto : 프로토콜, tcp 또는 udp
    --index=index : 컨테이너의 인덱스 수

`docker-compose pause [서비스명]`

Ex) docker-compose.yml

```
root@Ubuntu ~/test % cat docker-compose.yml
version: '3.3'

services:
  webserver:
    image: nginx
    ports:
    - 80:80/tcp

  server_b:
    image: redis
```

- 'docker-compose.yml'을 수정후, `docker-compose up -d` 실행

Ex) 공개포트 확인

```
root@Ubuntu ~/test % docker-compose port webserver 80
0.0.0.0:80
```

Ex) 구성 확인

```
root@Ubuntu ~/test % docker-compose config
services:
  server_b:
    image: redis
  webserver:
    image: nginx
    ports:
    - protocol: tcp
      published: 80
      target: 80
version: '3.3'
```

- protocol: 포트 프로토콜(tcp)
- published: 호스트OS에서 공개할 포트(8080)
- target: 컨테이너 내부 포트(80)

3.7 여러 컨테이너 강제 정지/삭제

`docker-compose kill [옵션] [서비스명]`

- 컨테이너 강제 정지, 시그널 송신
- `kill -l`로 시그널 종류 확인 가능

`docker-compose rm [옵션] [서비스명]`

Ex) 컨테이너에 시그널 송신

```
root@Ubuntu ~/test % docker-compose kill -s SIGINT
Killing test_webserver_1 ... done
Killing test_server_b_1  ... done
root@Ubuntu ~/test % docker-compose ps
      Name                     Command               State    Ports
-------------------------------------------------------------------
test_server_b_1    docker-entrypoint.sh redis ...   Exit 0
test_webserver_1   nginx -g daemon off;            Exit 0
```

- 옵션을 지정하지 않으면 'SIGKILL'이 송신

Ex) 여러 컨테이너 일괄 삭제

```
root@Ubuntu ~/test % docker-compose rm
Going to remove test_webserver_1, test_server_b_1
Are you sure? [yN] y
Removing test_webserver_1 ... done
Removing test_server_b_1  ... done
```

- '-f' : 확인 메시지 없이 강제 삭제

3.8 여러 리소스 일괄 삭제

`docker-compose down [옵션]`

- 실행중인 컨테이너 정지 후, Docker 이미지, 네트워크, 데이터 볼륨을 일괄 삭제
  - 주요 옵션
    --rmi all : 모든 이미지를 삭제
    --rmi local : 커스텀 태그가 없는 이미지만 삭제
    -v, --volumes : Compose 정의 파일의 데이터 볼륨을 삭제

Ex) 여러 이미지 삭제

```
root@Ubuntu ~/test % docker-compose down --rmi all
Stopping test_server_b_1  ... done
Stopping test_webserver_1 ... done
Removing test_server_b_1  ... done
Removing test_webserver_1 ... done
Removing network test_default
Removing image nginx
Removing image redis
```

- 실행중인 컨테이너 정지 후, 모든 Docker 이미지 삭제(Docker Compose에 정의된 이미지)
- `--rmi all` 옵션이 없으면, Docker 이미지는 삭제되지 않고 존재
