Docker 실습

FastAPI + 리버스 프록시 (Nginx/Caddy)

Dockerfile부터 Docker Compose까지
실전 컨테이너 관리 완전 정복

김경훈 kyunghoon@core.today

목차

실습 개요

이 실습에서는 **Dockerfile로 시작하여 불편함을 경험**한 후, **Docker Compose의 필요성을 자연스럽게 체감**하는 방식으로 학습합니다.

> 💡 **학습 철학**
>
> "불편함을 경험해야 해결책의 가치를 알 수 있다"

예상 소요 시간: 60-90분

학습 목표

- Dockerfile로 컨테이너를 직접 빌드하고 실행하기
- Docker 네트워크를 통한 컨테이너 간 통신 이해
- Nginx와 Caddy 리버스 프록시 설정 및 비교
- **수동 관리의 불편함과 복잡성 체감**
- Locust로 성능 테스트하며 관리의 어려움 경험
- **Docker Compose로 전환하여 효율성 체감**
- 멀티 컨테이너 애플리케이션 관리 자동화

사전 준비사항

✅ 필수 소프트웨어

- Docker Desktop 설치 (Windows/Mac/Linux)
- Python 3.7+ 설치 (성능 테스트용)

✅ 사전 지식

- PowerShell 또는 터미널 기본 사용법
- 기본적인 HTTP 및 REST API 개념

프로젝트 구조

실습에 사용할 파일 구조입니다:

```
lgcns/
├── app/
│   ├── Dockerfile
│   ├── main.py
│   └── requirements.txt
├── nginx/
│   └── nginx.conf
├── caddy/
│   └── Caddyfile
├── performance/
│   ├── locustfile.py
│   └── requirements.txt
├── docker-compose.nginx.yml
└── docker-compose.caddy.yml
```

리버스 프록시

> **클라이언트 요청을 받아 백엔드 서버로 전달하는 중간 서버**

주요 장점

- 로드 밸런싱으로 트래픽 분산
- SSL/TLS 종료 처리로 백엔드 부담 감소
- 정적 파일 캐싱으로 성능 향상
- 보안 강화 (백엔드 서버 숨김)
- 압축 및 최적화

Docker 네트워크

> **컨테이너 간 통신을 위한 가상 네트워크**

핵심 개념

같은 네트워크 내 컨테이너는 **이름으로 서로 통신**할 수 있습니다.

예시: nginx 컨테이너에서 fastapi 컨테이너로 접근

```
http://fastapi:8000/
```

IP 주소 대신 **컨테이너 이름**을 사용하여 간편하고 안정적으로 통신합니다.

Dockerfile이란?

> Docker 이미지를 자동으로 빌드하기 위한 텍스트 파일

주요 특징

- 텍스트 파일로 버전 관리가 용이
- 재현 가능한 이미지 빌드
- 레이어 캐싱으로 빠른 빌드
- 명령어 기반의 선언적 정의

Dockerfile 주요 명령어

기본 명령어

- **FROM**: 베이스 이미지 지정
- **RUN**: 명령어 실행 (패키지 설치 등)
- **COPY/ADD**: 파일 복사
- **WORKDIR**: 작업 디렉토리 설정
- **ENV**: 환경 변수 설정
- **EXPOSE**: 포트 노출
- **CMD/ENTRYPOINT**: 컨테이너 실행 명령어

확장자 없이 Dockerfile 이란 이름으로 아래 텍스

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0"]
```

Dockerfile 베스트 프랙티스

- 최소한의 레이어 사용 (RUN 명령어 결합)
- 자주 변경되지 않는 레이어를 앞에 배치
- .dockerignore 파일로 불필요한 파일 제외
- 멀티 스테이지 빌드로 이미지 크기 최소화
- 명확한 버전 태그 사용 (latest 지양)
- 보안을 위해 non-root 사용자로 실행

Docker 이미지 레이어

> 각 Dockerfile 명령어는 읽기 전용 레이어를 생성

레이어 캐싱의 장점

- 변경되지 않은 레이어는 재사용
- 빌드 시간 대폭 단축
- 네트워크 전송량 감소
- 디스크 공간 효율적 사용
- 동일한 베이스 이미지 공유

Docker Compose란?

> 여러 컨테이너를 정의하고 실행하기 위한 도구

왜 필요한가?

- 여러 컨테이너를 한 번에 관리
- YAML 파일로 인프라를 코드로 정의
- 개발 환경과 프로덕션 환경을 동일하게 유지
- 네트워크와 볼륨을 자동으로 생성
- 서비스 간 의존성 관리
- 팀원들과 환경을 쉽게 공유

Docker Compose 파일 구조

```
version: '3.8'
services:
fastapi:
build: ./app
networks:
- app-network
nginx:
image: nginx:alpine
ports:
- "80:80"
volumes:
- ./nginx/nginx.conf:/etc/nginx/conf.d/default.conf
depends_on:
- fastapi
networks:
- app-network
networks:
app-network:
driver: bridge
```

- **services**: 실행할 컨테이너 정의
- **networks**: 네트워크 정의
- **volumes**: 볼륨 정의

Docker Compose 주요 명령어

- **up -d**: 서비스 시작 (백그라운드)
- **down**: 서비스 중지 및 삭제
- **ps**: 실행 중인 서비스 확인
- **logs [-f]**: 로그 확인 (실시간)
- **restart**: 서비스 재시작
- **stop/start**: 서비스 중지/시작
- **build**: 이미지 다시 빌드
- **exec**: 실행 중인 컨테이너에 명령 실행

Dockerfile vs Docker Compose

Dockerfile	Docker Compose
목적: 이미지 빌드	**목적:** 다중 컨테이너 관리
범위: 단일 이미지	**범위:** 여러 서비스
파일 형식: Dockerfile	**파일 형식:** YAML
사용 시점: 빌드 시	**사용 시점:** 실행 시

💡 **Compose가 Dockerfile을 사용**

Docker Compose의 장점

- **단순성**: 복잡한 명령어를 YAML로 간단히 정의
- **재현성**: 동일한 환경을 어디서든 재현
- **격리성**: 프로젝트별 독립된 환경
- **협업**: 팀원들과 환경을 쉽게 공유
- **개발 효율**: 로컬 개발 환경 구축 자동화
- **CI/CD**: 자동화 파이프라인에 통합 용이

언제 무엇을 사용할까?

Dockerfile만 사용

- 단일 컨테이너 애플리케이션
- 이미지를 빌드해서 레지스트리에 푸시
- CI/CD 파이프라인에서 이미지 빌드

Docker Compose 사용 (권장)

- 여러 컨테이너가 필요한 애플리케이션
- 로컬 개발 환경
- 테스트 환경
- 소규모 프로덕션 환경
- 마이크로서비스 아키텍처

Part 1

Dockerfile로 수동 관리하기

"직접 해보면서 불편함을 체감합니다"

실습 1 - Dockerfile로 Nginx 구성 (수동)

실습 목표

Dockerfile만 사용하여 FastAPI와 Nginx를 수동으로 구성합니다.

모든 명령어를 직접 입력하며 관리의 복잡성을 경험합니다.

💡 **주목:** 이 과정에서 느끼는 불편함이 이번 실습의 핵심 학습 포인트입니다!

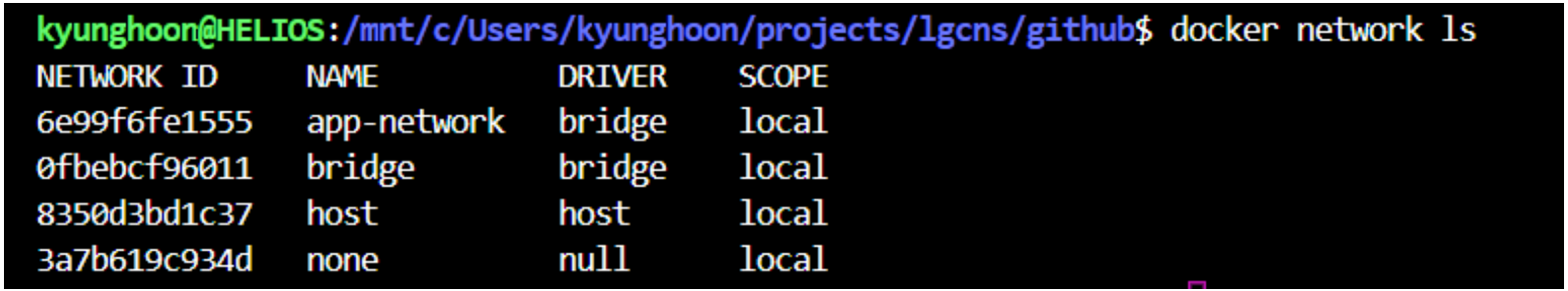
실습 1 - Step 1

Docker 네트워크 생성

명령어

```
docker network create app-network
docker network ls
```

💡 **배운 점:** 컨테이너끼리 통신하려면 네트워크를 먼저 만들어야 합니다.



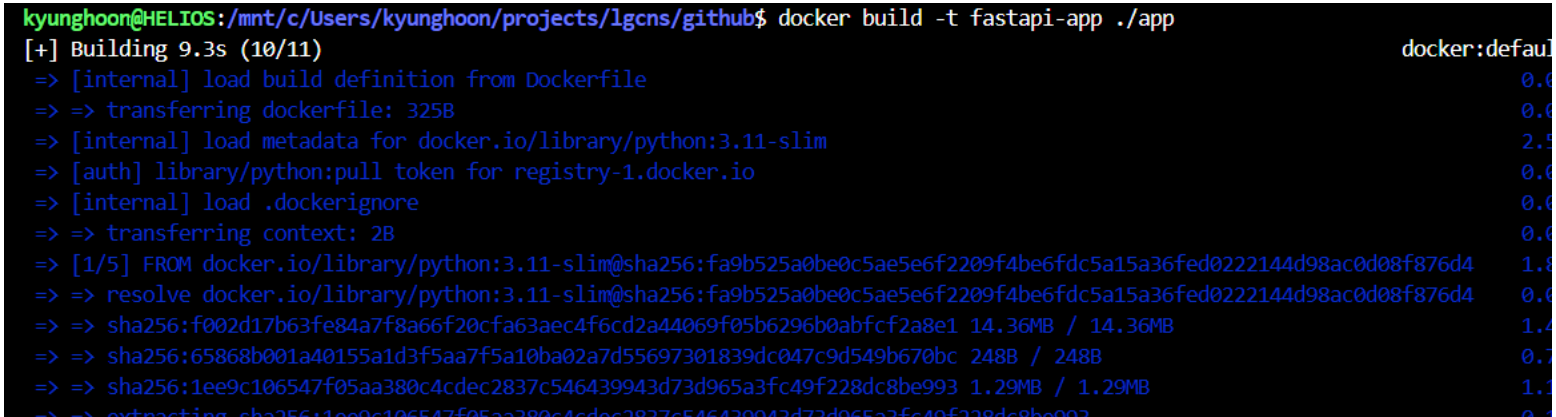
실습 1 - Step 2

FastAPI 이미지 빌드

명령어

```
docker build -t fastapi-app ./app
docker images | grep fastapi-app
```

💡 **배운 점:** 이미지를 빌드하는 데 시간이 걸립니다. 코드를 수정할 때마다 다시 빌드해야 합니다.



실습 1 - Step 3

FastAPI 컨테이너 실행

명령어

```
docker run -d \
--name fastapi-app \
--network app-network \
--network-alias fastapi \
fastapi-app
```

⚠ **주의:** --network-alias fastapi를 잊으면 Nginx가 연결할 수 없습니다!

실습 1 - Step 4

Nginx 컨테이너 실행

명령어 (Powershell)

```
docker run -d \
--name nginx-proxy \
--network app-network \
-p 80:80 \
-v "${PWD}\nginx\nginx.conf:/etc/nginx/conf.d/default.conf:ro" \
nginx:alpine
```

💡 **Windows 경로 주의:** PowerShell에서는 ${PWD} 사용

실습 1 - Step 5

테스트

명령어

```
curl http://localhost/
curl http://localhost/health
curl http://localhost/api/items/1?q=test
```

> ✅ **예상 응답:**
> {"message":"Hello from FastAPI behind Nginx!"}

실습 1 - Step 6

디버깅 & 정리

문제 발생 시

```
docker logs nginx-proxy
docker logs fastapi-app
docker network inspect app-network
docker restart nginx-proxy
```

정리하기

```
docker stop fastapi-app nginx-proxy
docker rm fastapi-app nginx-proxy
docker network rm app-network
```

실습 2 - Dockerfile로 Caddy 구성 (수동)

실습 목표

이번에는 Caddy로 **똑같은 작업을 반복**합니다.

반복 작업의 번거로움을 체감합니다

실습 2 회고

더 불편한 점:

똑같은 작업을 반복해야 한다

명령어를 하나라도 빼먹으면 안 된다

포트나 설정을 바꾸면 명령어를 다시 입력해야 한다

실수하기 쉽다

실습 2 - Step 1

환경 정리 & 네트워크 재생성

이전 환경 정리

```
docker stop fastapi-app nginx-proxy 2>/dev/null
docker rm fastapi-app nginx-proxy 2>/dev/null
docker network rm app-network 2>/dev/null
```

네트워크 재생성

```
docker network create app-network
```

> 😓 불편: 네트워크를 또 만들어야 합니다...

실습 2 - Step 2

FastAPI & Caddy 실행

FastAPI 재실행

```
docker run -d \
--name fastapi-app \
--network app-network \
--network-alias fastapi \
fastapi-app
```

Caddy 실행 (포트 8080)

```
docker run -d \
--name caddy-proxy \
--network app-network \
-p 8080:8080 \
-p 8443:443 \
-v "${PWD}\caddy\Caddyfile:/etc/caddy/Caddyfile:ro" \
caddy:alpine
```

> 😓 불편: 똑같은 명령어를 또 입력해야 합니다.

실습 2 - Step 3

테스트 & 정리

테스트 (포트 8080)

```
curl http://localhost:8080/
curl http://localhost:8080/health
curl http://localhost:8080/api/items/1?q=test
```

정리

```
docker stop fastapi-app caddy-proxy
docker rm fastapi-app caddy-proxy
docker network rm app-network
```

실습 3 - 두 프록시 동시 실행

실습 목표

Nginx와 Caddy를 **동시에 실행**하여 비교합니다.

관리 복잡도가 기하급수적으로 증가합니다!

실습 3 회고

훨씬 더 불편한 점:

컨테이너가 3개로 늘어나니 관리가 복잡하다

하나라도 실행 순서를 틀리면 오류가 난다

전체를 재시작하려면 명령어를 여러 번 입력해야 한다

로그를 확인하려면 각각 따로 봐야 한다

이거... 더 좋은 방법이 없을까? 🤔

실습 3 - 전체 실행

두 프록시 동시 실행

1. 네트워크 & FastAPI

```
docker network create app-network
docker build -t fastapi-app ./app
docker run -d \
--name fastapi-app \
--network app-network \
--network-alias fastapi \
fastapi-app
```

2. Nginx & Caddy

```
docker run -d \
--name nginx-proxy \
--network app-network \
-p 80:80 \
-v "${PWD}\nginx\nginx.conf:/etc/nginx/conf.d/default.conf:ro" \
nginx:alpine
docker run -d \
--name caddy-proxy \
--network app-network \
-p 8080:8080 \
-p 8443:443 \
-v "${PWD}\caddy\Caddyfile:/etc/caddy/Caddyfile:ro" \
caddy:alpine
```

😱 매우 불편함: 5개 이상의 명령어를 입력해야 합니다!

실습 3 - 테스트 & 비교

응답 헤더 비교

Nginx 테스트

```
curl http://localhost/
curl -I http://localhost/
```

Caddy 테스트

```
curl http://localhost:8080/
curl -I http://localhost:8080/
```

정리

```
docker stop fastapi-app nginx-proxy caddy-proxy
docker rm fastapi-app nginx-proxy caddy-proxy
docker network rm app-network
```

실습 4 - Locust로 성능 테스트

실습 목표

성능 테스트를 하면서 수동 관리의 불편함이 **극대화**됩니다.

혼돈의 극치!

환경을 바꿀 때마다 6개 이상의 명령어...

실습 4 - Locust 설치

성능 테스트 준비

Locust 설치

```
cd performance
pip install -r requirements.txt
locust --version
```

💡 **Locust:** Python 기반 성능 테스트 도구

웹 UI로 실시간 모니터링 가능

실습 4 - Locust 설치

Locust 성능 테스트 준비

Locust 설치

```
cd performance
pip install -r requirements.txt
locust --version
```

Nginx 환경 준비

```
cd ..
docker network create app-network
docker build -t fastapi-app ./app
docker run -d --name fastapi-app --network app-network \
--network-alias fastapi fastapi-app
docker run -d --name nginx-proxy --network app-network \
-p 80:80 -v "${PWD}\nginx\nginx.conf:/etc/nginx/conf.d/default.conf:ro" \
nginx:alpine
```

> 😰 복잡함: 명령어를 4개나 입력해야 합니다.

Nginx 성능 테스트

Locust 실행

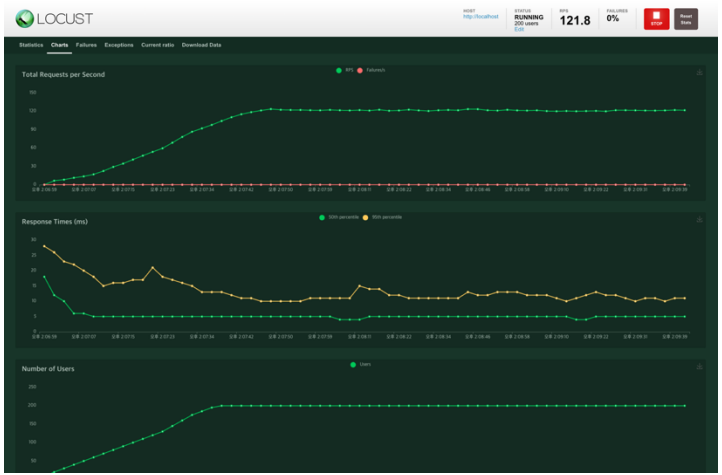
```
cd performance
locust -f locustfile.py --host=http://localhost
```

웹 UI 접속

```
http://localhost:8089
테스트 설정:
- Number of users: 50
- Spawn rate: 10
- 5분 정도 실행
```

결과 저장

```
Stop 버튼 클릭
Download Data 탭에서 statistics.csv 다운로드
```



실습 4 - 환경 전환

Caddy로 환경 전환

Nginx 환경 정리

```
# Locust 종료 (Ctrl+C)
cd ..
docker stop fastapi-app nginx-proxy
docker rm fastapi-app nginx-proxy
docker network rm app-network
```

Caddy 환경 구성

```
docker network create app-network
docker run -d --name fastapi-app --network app-network \
--network-alias fastapi fastapi-app
docker run -d --name caddy-proxy --network app-network \
-p 8080:8080 -p 8443:443 \
-v "${PWD}\caddy\Caddyfile:/etc/caddy/Caddyfile:ro" \
caddy:alpine
```

😱 매우 불편함: 환경을 바꾸려면 6개 이상의 명령어를 다시 입력!

실습 4 - Caddy 테스트

Caddy 성능 테스트 & 정리

Caddy 테스트

```
cd performance
locust -f locustfile.py --host=http://localhost:8080
# 동일한 테스트 수행 (50 users, 5분)
```

헤드리스 자동화 (선택)

```
locust -f locustfile.py --host=http://localhost:8080 \
--users 100 --spawn-rate 10 --run-time 60s \
--headless --html report-caddy.html
```

최종 정리

```
cd ..
docker stop fastapi-app caddy-proxy
docker rm fastapi-app caddy-proxy
docker network rm app-network
```

Part 1 회고

지금까지 우리는...

✅ 실습 1-3: 기본 구성의 복잡함 체감

✅ 실습 4: 성능 테스트로 불편함 극대화

"이건 너무 불편하다!"

Part 2

Docker Compose로 전환하기

"이제 편리함을 경험할 시간입니다"

실습 5 - Docker Compose 도입

실습 목표

Docker Compose를 사용하여 **같은 작업을 단 1-2개 명령어로** 수행합니다.

혁명적 편리함!

Part 1의 불편함이 얼마나 해소되는지 직접 체감합니다

실습 5 - Docker Compose 실행

단 1개 명령어로!

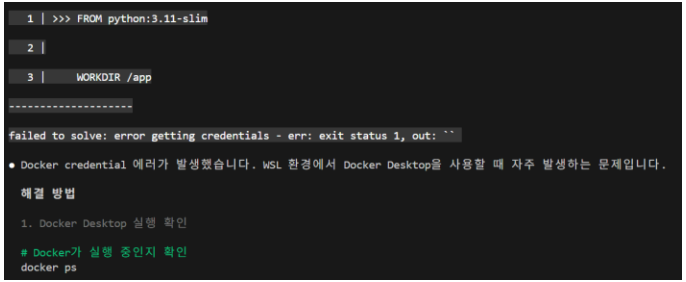
이전에는 5-6개 명령어가 필요했던 작업이...

```
docker-compose -f docker-compose.nginx.yml up -d
```

🎉 끝! 이게 전부입니다!

에러가 났다면 GPT와 함께..!

소프트웨어 업그레이드와 함께 늘 새로운 에러가 날 수 있습니다.



실습 5 - Docker Compose 정리

정리도 단 1개 명령어!

이전에는 3-4개 명령어가 필요했던 작업이...

```
docker-compose -f docker-compose.nginx.yml down
```

네트워크까지 자동으로 정리!

Part 2 회고

편리한 점:

명령어가 1-2개로 줄었다!

네트워크를 신경 쓸 필요가 없다!

실행 순서가 자동으로 관리된다!

설정 변경이 쉽다 (YAML만 수정)!

정리도 한 번에!

 **실무에서 쓸 만하다!**

Part 1 vs Part 2 비교

작업	Dockerfile	Docker Compose	개선도
실행	5-6개 명령어	1개 명령어	**6배 감소** 🚀
환경 전환	10개 이상	2개 명령어	**5배 감소** 🚀
정리	3-4개 명령어	1개 명령어	**4배 감소** 🚀

유용한 명령어

Docker Compose

```
docker-compose -f [file] up -d    # 실행
docker-compose -f [file] down     # 중지 및 삭제
docker-compose -f [file] logs -f  # 로그 확인
docker-compose -f [file] ps       # 상태 확인
```

일반 Docker

```
docker ps                         # 실행 중인 컨테이너
docker logs -f [container]        # 로그 실시간 확인
docker exec -it [container] sh    # 내부 접속
```

트러블슈팅

❌ 컨테이너 이름 충돌

증상: The container name is already in use

✅ **해결:** docker rm -f [container-name]

❌ 포트 이미 사용 중

증상: port is already allocated

✅ **해결:** docker stop [container-name]

실습 체크리스트

Part 1: Dockerfile (수동 관리)

실습 1: Nginx 구성 (수동)

실습 2: Caddy 구성 (수동)

실습 3: 두 프록시 동시 실행

실습 4: 성능 테스트

Part 2: Docker Compose (자동화)

실습 5: Docker Compose 도입

실습 6: 성능 테스트 (자동화)

실습 완료를 축하합니다! 

- ✅ Dockerfile의 한계를 이해했습니다
- ✅ Docker Compose의 가치를 체감했습니다
- ✅ 리버스 프록시 설정을 할 수 있습니다
- ✅ 성능 테스트를 수행할 수 있습니다
- ✅ 실무에서 사용할 준비가 되었습니다!

"불편함을 경험했기에, 이제 더 나은 도구를 선택할 수 있습니다."

Locust로 nginx 성능 테스트 하신 스크린샷을 올려주세요

시간이 되시는 분들은 caddy도 성능 테스트하여 비교해 보세요