신뢰성 확보

Health Check

- 개요
  - ✓ 도커는 컨테이너를 시작할 때마다 애플리케이션의 기본적인 상태를 확인
  - ✓ 컨테이너를 실행하면 내부에서 애플리케이션 실행 파일이나 자바 혹은 닷넷 런타임 또는 셸 스크립트 같은 특정한 프로세스가 실행되는데 도커가 확인하는 것은 이 프로세스의 실행 상태인데 만약 이 프로세스가 종료됐다면 컨테이너도 종료 상태가 되는데 이것만으로도 환경과 상관없이 기본적인 헬스 체크는 가능
  - ✓ 해당 프로세스가 비정상 종료됐거나 컨테이너가 종료됐다면 개발자도 애플리케이션의 상태가 비정상임을 알 수 있음
  - ✓ 클러스터 환경에서는 플랫폼이 종료된 컨테이너를 재시작하거나 새 컨테이너로 교체하는 작업을 대신 수행
  - ✓ 웹 애플리케이션을 실행 중인 컨테이너에서 처리 용량을 뛰어넘는 수의 요청이 들어오는 순간 웹 애플리케이션은 503 Service Unavailable 오류를 발생하지만 컨테이너의 프로세스는 여전히 정상적으로 실행 중 인 상태
  - ✓ 애플리케이션이 동작을 멈췄어도 도커는 컨테이너를 정상이라 판단

Health Check

❖ 헬스 체크

✓ 헬스 체크를 위한 파이썬 애플리케이션 작성: app.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, Docker!"

# 헬스체크용 엔드포인트
@app.route('/health')
def health_check():
    return "OK", 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

Health Check

❖ 헬스 체크

✓ 이미지 빌드를 위한 Dockerfile 작성

```
# 1. 베이스 이미지 설정
FROM python:3.9-slim

# 2. 필요한 패키지 설치 (curl은 헬스체크 명령에 자주 사용됩니다)
RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/*

# 3. 작업 디렉토리 설정 및 파일 복사
WORKDIR /app
COPY app.py .
RUN pip install flask

# 4. 헬스체크 설정 ★ 핵심 부분
# --interval: 체크 간격 (기본 30s)
# --timeout: 이 시간 안에 응답 없으면 실패 (기본 30s)
# --start-period: 컨테이너 시작 후 헬스체크를 유예할 시간
# --retries: 연속 실패 시 'unhealthy'로 간주할 횟수 (기본 3)
HEALTHCHECK --interval=10s --timeout=3s --start-period=5s --retries=3 ₩
  CMD curl -f http://localhost:8080/health || exit 1

# 5. 앱 실행
EXPOSE 8080
CMD ["python", "app.py"]
```

Health Check

- 헬스 체크
  - ✓ 이미지 빌드: docker build -t my-health-app .
  - ✓ 컨테이너 실행: docker run -d --name health-test my-health-app
  - ✓ 상태 확인: docker ps

```
CONTAINER ID   IMAGE           COMMAND           CREATED         STATUS
PORTS      NAMES
59cd0ffefdbc   my-health-app   "python app.py"   2 seconds ago   Up 2 seconds (health:
starting)   8080/tcp   health-test
```

```
CONTAINER ID   IMAGE           COMMAND           CREATED         STATUS
PORTS      NAMES
59cd0ffefdbc   my-health-app   "python app.py"   32 seconds ago   Up 32 seconds
(healthy)   8080/tcp   health-test
```

  - ✓ docker ps를 입력했을 때 STATUS 항목

starting: 컨테이너가 막 실행되어 헬스체크를 기다리는 중

healthy: curl 명령이 성공하여 정상 작동 중

unhealthy: 설정된 횟수만큼 헬스체크가 실패함

Health Check

- 헬스 체크
  - ✓ 헬스 체크가 중요한 이유
    - 자동 복구: Docker Swarm이나 Kubernetes 같은 오케스트레이션 도구는 unhealthy 상태의 컨테이너를 자동으로 재시작하거나 교체
    - 무중단 배포: 로드밸런서(Nginx 등)와 연동할 때 새 컨테이너가 healthy가 될 때까지 트래픽을 보내지 않도록 제어할 수 있음

Health Check

- Docker Compose 헬스 체크
  - ✓ docker-compose.yml 파일에서 서비스 하위에 healthcheck 항목을 추가하면 Dockerfile에 이미 헬스 체크가 있더라도 Compose 파일에 작성한 내용이 우선순위를 가짐
  - ✓ 기본 형식

```
services:
  web:
    image: my-health-app
    ports:
      - "8080:8080"
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/health"]
      interval: 10s
      timeout: 5s
      retries: 3
      start_period: 5s
```

Health Check

- Docker Compose 헬스 체크
  - ✓ DB 준비 후 웹 서버 실행

```
version: '3.8'

services:
  db:
    image: postgres:15
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydb
    healthcheck:
      # pg_isready는 postgres에서 제공하는 헬스체크 전용 도구입니다.
      test: ["CMD-SHELL", "pg_isready -U user -d mydb"]
      interval: 5s
      timeout: 5s
      retries: 5
    networks:
      - my-network
```

Health Check

- Docker Compose 헬스 체크
  - ✓ DB 준비 후 웹 서버 실행

```
web:
  build: .
  ports:
    - "8080:8080"
  environment:
    DB_HOST: db
  depends_on:
    db:
      # 핵심: db 서비스가 단순히 '실행'되는 게 아니라 'healthy' 상태여야 함
      condition: service_healthy
  networks:
    - my-network

networks:
  my-network:
```

Health Check

- Docker Compose 헬스 체크
  - ✓ DB 준비 후 웹 서버 실행
    - test: 실행할 명령어
    - interval: 헬스 체크를 수행하는 주기
    - timeout: 이 시간 안에 응답이 없으면 실패로 간주
    - retries: 연속으로 몇 번 실패해야 unhealthy로 바꿀지 결정
    - start_period: 앱 초기화 시간을 벌어주는 것으로 이 시간 동안의 실패는 횟수에 포함하지 않음
    - 기존의 depends_on은 컨테이너가 시작만 되면 바로 다음 컨테이너를 실행하지만 condition: service_healthy를 사용하면, DB가 내부 설정을 마치고 실제 쿼리를 받을 준비가 끝날 때까지 웹 서버 실행을 우직하게 기다려주는데 덕분에 "DB 연결 실패"로 웹 서버가 동작하지 않는 현상을 방지할 수 있음