Docker 치트시트

Docker 핵심 개념

- Docker: 컨테이너 기반의 애플리케이션 가상화 플랫폼입니다.
- 이미지 (Image): 컨테이너 실행에 필요한 파일과 설정을 담은 읽기 전용 템플릿입니다.
- 컨테이너 (Container): 이미지의 실행 가능한 인스턴스입니다. 독립된 환경에서 애플리케이션이 구동됩니다.
- Dockerfile: 이미지를 자동으로 빌드하기 위한 일련의 명령어가 담긴 스크립트 파일입니다.
- 레지스트리 (Registry): 이미지를 저장하고 배포하는 저장소입니다. (예: Docker Hub)
- 볼륨 (Volume): 컨테이너가 종료되어도 데이터를 영구적으로 보존하기 위한 저장 메커니즘입니다.
- 네트워크 (Network): 컨테이너끼리 혹은 외부와 통신하기 위한 가상 네트워크 설정입니다.

컨테이너 생명주기 관리

컨테이너 실행

- `docker run [옵션] IMAGE [명령어] [인자...]`: 새로운 컨테이너를 생성하고 실행합니다.
  - `-d, --detach`: 컨테이너를 백그라운드에서 실행합니다.
  - `-p, --publish <호스트포트>:<컨테이너포트>`: 호스트와 컨테이너의 포트를 연결(포트 포워딩)합니다.
  - `-P, --publish-all`: Dockerfile에 설정된 모든 포트를 호스트의 무작위 포트에 연결합니다.
  - `-v, --volume <호스트경로>:<컨테이너경로>[:ro]`: 볼륨을 마운트합니다. (`:ro` 추가 시 읽기 전용)
  - `--mount`: 더 상세하고 명시적인 마운트 옵션을 설정합니다.
  - `--name <이름>`: 컨테이너에 고유한 이름을 부여합니다.
  - `-e, --env <키>=<값>`: 환경 변수를 설정합니다.
  - `--env-file <파일>`: 환경 변수 설정 파일을 사용합니다.
  - `--rm`: 컨테이너가 종료될 때 자동으로 컨테이너를 삭제합니다.
  - `-it`: 터미널 입력을 주고받을 수 있도록 TTY를 할당합니다. (셸 접속 시 필수)
  - `--network <네트워크>`: 컨테이너를 특정 네트워크에 연결합니다.
  - `--restart <정책>`: 재시작 정책을 설정합니다. (`no`, `on-failure`, `unless-stopped`, `always`)
  - `--memory <제한>`: 사용할 메모리 용량을 제한합니다. (예: 512m, 1g)
  - `--cpus <개수>`: 사용할 CPU 코어 수를 제한합니다. (예: 0.5, 2)
  - `--user <uid>:<gid>`: 컨테이너를 실행할 사용자 계정을 지정합니다.
  - `--workdir <경로>`: 컨테이너 내부의 작업 디렉토리를 설정합니다.
  - `--hostname <이름>`: 컨테이너의 호스트 이름을 설정합니다.

컨테이너 조회 및 상태 관리

- `docker ps`: 현재 실행 중인 컨테이너 목록을 확인합니다.
  - `-a, --all`: 실행 여부와 관계없이 모든 컨테이너를 표시합니다.
  - `-q, --quiet`: 컨테이너 ID만 간략히 출력합니다.
  - `-s, --size`: 컨테이너가 차지하는 디스크 용량 정보를 포함합니다.
  - `--filter`: 특정 조건(예: `status=running`)으로 목록을 필터링합니다.
  - `--format`: 출력 형식을 사용자 정의합니다.
- `docker start <ID/이름>`: 중지된 컨테이너를 다시 시작합니다.
- `docker stop <ID/이름>`: 실행 중인 컨테이너에 종료 신호(SIGTERM)를 보내 중지시킵니다.
- `docker kill <ID/이름>`: 컨테이너를 즉시 강제 종료(SIGKILL)합니다.
- `docker restart <ID/이름>`: 컨테이너를 재시작합니다.
- `docker pause <ID/이름>`: 실행 중인 컨테이너의 모든 프로세스를 일시 정지합니다.
- `docker unpause <ID/이름>`: 일시 정지된 컨테이너를 다시 가동합니다.
- `docker wait <ID/이름>`: 컨테이너가 종료될 때까지 대기하며 종료 코드를 출력합니다.
- `docker attach <ID/이름>`: 실행 중인 컨테이너의 표준 입력/출력에 연결합니다.

컨테이너 삭제

- `docker rm <ID/이름>`: 중지된 컨테이너를 삭제합니다.
  - `-f, --force`: 실행 중인 컨테이너를 강제로 삭제합니다.
  - `-v, --volumes`: 컨테이너에 연결된 익명 볼륨도 함께 제거합니다.
- `docker rm $(docker ps -aq)`: 모든 컨테이너를 일괄 삭제합니다.
- `docker stop $(docker ps -aq)`: 모든 컨테이너를 일괄 중지합니다.
- `docker container prune`: 실행 중이지 않은 모든 컨테이너를 한꺼번에 삭제합니다.

컨테이너 정보 및 모니터링

- `docker logs <ID/이름>`: 컨테이너의 로그 기록을 확인합니다.
  - `-f, --follow`: 로그 출력을 실시간으로 스트리밍합니다.
  - `--tail <N>`: 마지막 N줄의 로그만 보여줍니다.
  - `--since <시점>`: 특정 시간 이후의 로그를 보여줍니다.
  - `-t, --timestamps`: 로그에 타임스탬프를 함께 표시합니다.
- `docker exec -it <ID/이름> <명령어>`: 실행 중인 컨테이너 내부에서 새로운 명령을 수행합니다.
  - `-d, --detach`: 명령을 백그라운드에서 실행합니다.
  - `-u, --user`: 명령을 실행할 사용자 계정을 지정합니다.
- `docker inspect <ID/이름>`: 컨테이너나 이미지의 상세 정보를 JSON 형식으로 조회합니다.
- `docker top <ID/이름>`: 컨테이너 내부에서 실행 중인 프로세스 목록을 확인합니다.
- `docker stats`: 컨테이너별 리소스 사용량(CPU, 메모리, 네트워크 등)을 실시간으로 확인합니다.
- `docker port <ID/이름>`: 컨테이너의 포트 매핑 정보를 확인합니다.
- `docker diff <ID/이름>`: 이미지와 비교하여 컨테이너 파일 시스템의 변경 사항을 확인합니다.
- `docker cp <ID/이름>:<경로> <호스트경로>`: 컨테이너와 호스트 간에 파일을 복사합니다.

이미지 관리

기본 이미지 명령어

- `docker images`: 로컬에 저장된 이미지 목록을 확인합니다.
  - `-a, --all`: 중간 레이어 이미지를 포함한 모든 이미지를 표시합니다.
- `docker build -t <이름>:<태그> <경로>`: Dockerfile이 있는 경로를 기준으로 이미지를 빌드합니다.
  - `--no-cache`: 빌드 시 캐시를 사용하지 않고 새로 만듭니다.
  - `--build-arg <키>=<값>`: 빌드 시점에 사용할 변수를 전달합니다.
  - `-f, --file <경로>`: 사용할 Dockerfile의 경로를 직접 지정합니다.
- `docker pull <이름>[:<태그>]`: 레지스트리에서 이미지를 다운로드합니다.
- `docker push <이름>[:<태그>]`: 이미지를 레지스트리에 업로드합니다.
- `docker rmi <이미지ID>`: 로컬 이미지를 삭제합니다.
  - `-f, --force`: 이미지를 사용하는 컨테이너가 있더라도 강제로 삭제합니다.
- `docker image prune`: 이름 없는(dangling) 이미지를 모두 삭제하여 정리합니다.

이미지 정보 및 고급 관리

- `docker tag <원본> <대상>`: 기존 이미지에 새로운 태그를 부여합니다.
- `docker history <이미지>`: 이미지를 구성하는 각 레이어의 생성 이력을 확인합니다.
- `docker save -o <파일명.tar> <이미지>`: 이미지를 tar 파일로 저장합니다.
- `docker load -i <파일명.tar>`: tar 파일로부터 이미지를 불러옵니다.
- `docker export -o <파일명.tar> <컨테이너>`: 컨테이너의 파일 시스템을 tar 파일로 내보냅니다.
- `docker import <파일명.tar>`: tar 파일로부터 이미지를 생성합니다.

네트워크 및 볼륨 관리

가상 네트워크 관리

- `docker network ls`: Docker 네트워크 목록을 확인합니다.
- `docker network create --driver bridge <이름>`: 새로운 가상 네트워크를 생성합니다.
- `docker network rm <이름>`: 네트워크를 삭제합니다.
- `docker network inspect <이름>`: 네트워크의 상세 설정 정보를 확인합니다.
- `docker network connect <네트워크> <컨테이너>`: 컨테이너를 특정 네트워크에 연결합니다.
- `docker network disconnect <네트워크> <컨테이너>`: 컨테이너의 네트워크 연결을 해제합니다.

- `docker network prune`: 사용 중이지 않은 모든 네트워크를 삭제합니다.

볼륨 관리
- `docker volume ls`: Docker 볼륨 목록을 확인합니다.
- `docker volume create <이름>`: 새로운 데이터 볼륨을 생성합니다.
- `docker volume rm <이름>`: 볼륨을 삭제합니다.
- `docker volume inspect <이름>`: 볼륨의 상세 정보를 확인합니다.
- `docker volume prune`: 컨테이너에서 사용하지 않는 모든 볼륨을 삭제합니다.

Docker Compose (다중 컨테이너 관리)

기본 명령어
- `docker-compose up`: `docker-compose.yml` 설정을 바탕으로 서비스를 생성하고 시작합니다.
  - `-d`: 백그라운드 모드로 서비스를 구동합니다.
  - `--build`: 시작 전 이미지를 새로 빌드합니다.
- `docker-compose down`: 컨테이너, 네트워크, 볼륨 등을 정지시키고 삭제합니다.
- `docker-compose ps`: 현재 서비스의 컨테이너 상태를 확인합니다.
- `docker-compose logs -f <서비스명>`: 특정 서비스의 로그를 실시간으로 확인합니다.
- `docker-compose build <서비스명>`: 특정 서비스의 이미지만 빌드합니다.
- `docker-compose exec <서비스명> <명령어>`: 실행 중인 서비스 컨테이너 내에서 명령을 수행합니다.
- `docker-compose config`: 설정 파일의 문법 오류를 검증하고 내용을 확인합니다.

고급 제어
- `docker-compose start/stop/restart`: 서비스를 시작, 정지 또는 재시작합니다.
- `docker-compose pull`: 서비스에 정의된 이미지들을 최신 상태로 내려받습니다.
- `docker-compose top`: 각 서비스 내부에서 실행 중인 프로세스들을 확인합니다.

시스템 최적화 및 관리
- `docker system prune`: 중지된 컨테이너, 사용되지 않는 네트워크 및 이미지 등을 한꺼번에 정리합니다.
  - `-a, --all`: 사용되지 않는 이미지까지 포함하여 더 강력하게 정리합니다.
  - `--volumes`: 볼륨 데이터까지 모두 삭제합니다.
- `docker system df`: Docker가 사용 중인 전체 디스크 용량 현황을 확인합니다.
- `docker version`: 설치된 Docker의 버전을 확인합니다.
- `docker info`: 시스템 전체의 상세 상태 및 설정 정보를 확인합니다.

Dockerfile 주요 명령어 설명
- `FROM <이미지>`: 베이스가 될 이미지를 지정합니다. 모든 빌드의 시작점입니다.
- `RUN <명령어>`: 이미지 빌드 과정에서 실행할 명령어를 기술합니다. (패키지 설치 등)
- `CMD ["명령어", "인자"]`: 컨테이너가 시작될 때 기본으로 실행할 명령을 지정합니다.
- `ENTRYPOINT ["명령어"]`: 컨테이너 실행 시 고정적으로 수행될 명령을 지정합니다.
- `COPY <원본> <대상>`: 호스트의 파일이나 디렉토리를 이미지 내부로 복사합니다.
- `ADD <원본> <대상>`: COPY 기능에 더해 URL 다운로드 및 tar 압축 해제 기능이 포함됩니다.
- `WORKDIR <경로>`: 이후 명령어가 실행될 작업 디렉토리를 설정합니다.
- `ENV <키>=<값>`: 이미지 내부에서 사용할 환경 변수를 정의합니다.
- `EXPOSE <포트>`: 컨테이너가 대기할 포트를 명시합니다. (실제 포트 개방은 run 시 수행)
- `VOLUME ["경로"]`: 호스트와 공유할 마운트 포인트를 생성합니다.
- `USER <사용자>`: 이후 명령을 수행할 계정을 지정합니다.
- `ARG <이름>`: 빌드 시점에만 유효한 변수를 정의합니다.

보안 및 운영 모범 사례
- 이미지 보안: `docker pull` 시 공식 이미지를 우선 사용하고 항상 최신 버전을 유지합니다.
- 권한 최소화: 가능한 root 대신 일반 사용자 권한으로 컨테이너를 실행합니다.
- 리소스 관리: CPU와 메모리 제한을 설정하여 특정 컨테이너가 시스템 전체를 점유하지 않도록 합니다.
- 데이터 보존: 중요한 데이터는 반드시 볼륨이나 바인드 마운트를 사용해 컨테이너 외부에 저장합니다.
- 이미지 스캔: 취약점 검사 도구를 사용해 이미지의 보안 결함을 사전에 파악합니다.

유용한 활용 예제
```
# 단순 웹 서버(Nginx) 구동 예시
docker run -d -p 80:80 --name my-web
nginx

# MySQL 데이터베이스 구동 (데이터 보존을 위한
볼륨 사용)
docker run -d -p 3306:3306 -v
mysql_data:/var/lib/mysql \
  -e MYSQL_ROOT_PASSWORD=mypassword --
name my-db mysql:8.0

# 로컬 디렉토리를 연결하여 개발 환경 구성하기
docker run -it -v $(pwd):/app --
workdir /app ubuntu:22.04 /bin/bash

# 실시간 리소스 모니터링 (가독성 높은 포맷)
docker stats --format "table
{{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.NetIO}}"
```