Docker 치트시트

1. Docker 기본 개념

- Docker: 컨테이너 기반의 가상화 플랫폼
- 이미지 (Image): 컨테이너를 만들기 위한 읽기 전용 템플릿
- 컨테이너 (Container): 이미지에서 생성된 실행 가능한 인스턴스
- Dockerfile: 이미지를 빌드하기 위한 스크립트 파일
- 레지스트리 (Registry): 이미지를 저장하고 공유하는 저장소 (Docker Hub 등)
- 볼륨 (Volume): 데이터를 영구적으로 저장하기 위한 메커니즘
- 네트워크 (Network): 컨테이너 간 통신을 위한 네트워크 설정

2. 컨테이너 생명주기 관리

컨테이너 실행

- `docker run [OPTIONS] IMAGE [COMMAND] [ARG...]`: 새 컨테이너를 생성하고 실행
  - `-d, --detach`: 백그라운드에서 실행
  - `-p, --publish <host_port>:<container_port>`: 포트 포워딩
  - `-P, --publish-all`: Dockerfile에서 EXPOSE된 모든 포트를 호스트의 랜덤 포트에 연결
  - `-v, --volume <host_path>:<container_path>[:ro]`: 볼륨 마운트 (`:ro`는 읽기 전용)
  - `--mount`: 더 명시적인 마운트 옵션
  - `--name <name>`: 컨테이너 이름 지정
  - `-e, --env <key>=<value>`: 환경 변수 설정
  - `--env-file <file>`: 환경 변수 파일 사용
  - `--rm`: 컨테이너 종료 시 자동으로 삭제
  - `-it, --interactive --tty`: 상호작용 가능한 TTY 할당 (셸 접속 시 필수)
  - `--network <network>`: 컨테이너를 특정 네트워크에 연결
  - `--restart <policy>`: 재시작 정책 (`no`, `on-failure`, `unless-stopped`, `always`)
  - `--memory <limit>`: 메모리 제한 (예: 512m, 1g)
  - `--cpus <number>`: CPU 제한 (예: 0.5, 2)
  - `--user <uid>:<gid>`: 컨테이너 실행 사용자 지정
  - `--workdir <path>`: 작업 디렉토리 설정
  - `--hostname <name>`: 컨테이너 호스트명 설정

컨테이너 조회 및 관리

- `docker ps`: 실행 중인 컨테이너 목록
  - `-a, --all`: 중지된 컨테이너 포함 모든 컨테이너
  - `-q, --quiet`: 컨테이너 ID만 표시
  - `-s, --size`: 컨테이너 크기 정보 포함
  - `--filter`: 필터 조건 적용 (예: `--filter status=running`)
  - `--format`: 출력 형식 지정
- `docker start <container>`: 중지된 컨테이너 시작
- `docker stop <container>`: 컨테이너 중지 (SIGTERM 신호)
- `docker kill <container>`: 컨테이너 강제 종료 (SIGKILL 신호)
- `docker restart <container>`: 컨테이너 재시작
- `docker pause <container>`: 컨테이너 일시 정지
- `docker unpause <container>`: 컨테이너 일시 정지 해제
- `docker wait <container>`: 컨테이너 종료까지 대기
- `docker attach <container>`: 실행 중인 컨테이너에 연결

컨테이너 삭제

- `docker rm <container>`: 컨테이너 삭제
  - `-f, --force`: 실행 중인 컨테이너 강제 삭제
  - `-v, --volumes`: 연결된 볼륨도 함께 삭제
- `docker rm $(docker ps -aq)`: 모든 컨테이너 삭제
- `docker stop $(docker ps -aq)`: 모든 컨테이너 중지
- `docker container prune`: 중지된 모든 컨테이너 삭제

컨테이너 정보 및 모니터링

- `docker logs <container>`: 컨테이너 로그 확인
  - `-f, --follow`: 실시간 로그 스트리밍
  - `--tail <N>`: 마지막 N줄만 표시
  - `--since <timestamp>`: 특정 시간 이후 로그
  - `--until <timestamp>`: 특정 시간까지 로그
  - `-t, --timestamps`: 타임스탬프 포함
- `docker exec -it <container> <command>`: 실행 중인 컨테이너에서 명령 실행
  - `-d, --detach`: 백그라운드에서 실행
  - `-e, --env`: 환경 변수 설정
  - `-u, --user`: 실행 사용자 지정
  - `-w, --workdir`: 작업 디렉토리 지정
- `docker inspect <container|image>`: 상세 정보 JSON 형식으로 표시
- `docker top <container>`: 컨테이너 내 실행 중인 프로세스
- `docker stats`: 실시간 리소스 사용량 (CPU, 메모리, 네트워크, I/O)
  - `--no-stream`: 한 번만 출력
  - `--format`: 출력 형식 지정
- `docker port <container>`: 포트 매핑 정보
- `docker diff <container>`: 컨테이너의 파일 시스템 변경사항
- `docker cp <container>:<src_path> <dest_path>`: 컨테이너와 호스트 간 파일 복사

3. 이미지 관리

기본 이미지 명령어

- `docker images`: 로컬에 저장된 이미지 목록
  - `-a, --all`: 중간 이미지 포함 모든 이미지
  - `-q, --quiet`: 이미지 ID만 표시
  - `--filter`: 필터 조건 적용
  - `--format`: 출력 형식 지정
- `docker build -t <name>:<tag> <path>`: Dockerfile로 이미지 빌드
  - `--no-cache`: 캐시 사용 안 함
  - `--build-arg <key>=<value>`: 빌드 시 변수 전달
  - `-f, --file <path/to/Dockerfile>`: Dockerfile 경로 지정
  - `--target <stage>`: 멀티스테이지 빌드에서 특정 스테이지까지만 빌드
  - `--progress`: 빌드 진행 상황 출력 형식
- `docker pull <name>[:<tag>]`: 레지스트리에서 이미지 가져오기
- `docker push <name>[:<tag>]`: 레지스트리에 이미지 푸시
- `docker rmi <image>`: 로컬 이미지 삭제
  - `-f, --force`: 강제 삭제
- `docker rmi $(docker images -q -f "dangling=true")`: 목적 없는 이미지 삭제
- `docker image prune`: 사용되지 않는 이미지 삭제
  - `-a, --all`: 사용되지 않는 모든 이미지 삭제

이미지 정보 및 관리

- `docker tag <source_image> <target_image>`: 이미지에 태그 지정
- `docker history <image>`: 이미지 레이어 히스토리 확인
- `docker save -o <output.tar> <image>`: 이미지 tar로 저장
- `docker load -i <input.tar>`: 이미지 tar에서 로드
- `docker export -o <output.tar> <container>`: 컨테이너를 tar로 내보내기
- `docker import <tarball>`: tar에서 이미지 가져오기

4. 네트워크 및 볼륨 관리

네트워크

- `docker network ls`: Docker 네트워크 목록
- `docker network create --driver bridge <name>`: 브리지 네트워크 생성
  - `--subnet`: 서브넷 지정
  - `--gateway`: 게이트웨이 지정
- `docker network rm <name>`: 네트워크 삭제
- `docker network inspect <name>`: 네트워크 정보
- `docker network connect <network> <container>`: 네트워크에 컨테이너 연결
- `docker network disconnect <network> <container>`: 네트워크에서 컨테이너 분리
- `docker network prune`: 사용되지 않는 네트워크 삭제

볼륨

- `docker volume ls`: Docker 볼륨 목록
- `docker volume create <name>`: 볼륨 생성
  - `--driver`: 볼륨 드라이버 지정
  - `--label`: 볼륨에 라벨 추가
- `docker volume rm <name>`: 볼륨 삭제
- `docker volume inspect <name>`: 볼륨 정보
- `docker volume prune`: 사용되지 않는 볼륨 삭제

5. Docker Compose (다중 컨테이너 애플리케이션)

기본 명령어

- `docker-compose up`: `docker-compose.yml` 파일의 서비스 생성 및 시작
  - `-d`: 백그라운드에서 실행
  - `--build`: 시작 전 빌드

  - `--force-recreate`: 강제 재생성
  - `--scale <service>=<num>`: 서비스 스케일링
- `docker-compose down`: 컨테이너, 네트워크, 볼륨 중지 및 삭제
  - `-v, --volumes`: 볼륨도 삭제
  - `--rmi all`: 사용 이미지 삭제
  - `--remove-orphans`: 고아 컨테이너 삭제
- `docker-compose ps`: 서비스 컨테이너 상태
- `docker-compose logs -f <service_name>`: 실시간 로그
- `docker-compose build <service_name>`: 서비스 이미지 빌드
- `docker-compose exec <service_name> <command>`: 서비스 컨테이너에서 명령 실행
- `docker-compose run --rm <service_name> <command>`: 일회성 명령 실행
- `docker-compose config`: 구문 확인 및 검증
- `docker-compose scale <service>=<num>`: 특정 서비스의 컨테이너 수 조절

고급 명령어

- `docker-compose start <service>`: 중지된 서비스 시작
- `docker-compose stop <service>`: 서비스 중지
- `docker-compose restart <service>`: 서비스 재시작
- `docker-compose pause <service>`: 서비스 일시 정지
- `docker-compose unpause <service>`: 서비스 일시 정지 해제
- `docker-compose pull`: 서비스 이미지 업데이트
- `docker-compose top`: 서비스의 실행 중인 프로세스

6. 시스템 정리 및 관리

정리 명령어

- `docker system prune`: 사용 안 하는 리소스 삭제
  - `-a, --all`: 중지된 컨테이너와 안 쓰는 이미지 삭제
  - `--volumes`: 볼륨 포함 삭제
- `docker system df`: 디스크 사용량
- `docker builder prune`: 사용 안 하는 빌더 캐시 삭제
- `docker container prune`: 중지된 컨테이너 삭제
- `docker image prune`: 사용 안 하는 이미지 삭제
- `docker network prune`: 사용 안 하는 네트워크 삭제
- `docker volume prune`: 사용 안 하는 볼륨 삭제

시스템 정보

- `docker version`: Docker 버전 정보
- `docker info`: Docker 시스템 정보
- `docker system events`: 실시간 Docker 이벤트

7. Dockerfile 주요 명령어

- `FROM <image>`: 베이스 이미지 지정
- `RUN <command>`: 이미지 빌드 시 명령 실행
- `CMD ["command", "arg1", "arg2"]`: 컨테이너 시작 시 기본 명령
- `ENTRYPOINT ["executable", "param1", "param2"]`: 컨테이너 진입점
- `COPY <src> <dest>`: 파일/디렉토리 복사
- `ADD <src> <dest>`: 파일 추가 (URL, tar 자동 압축 해제 지원)
- `WORKDIR <path>`: 작업 디렉토리 설정
- `ENV <key>=<value>`: 환경 변수 설정
- `EXPOSE <port>`: 포트 노출 선언
- `VOLUME ["<path>"]`: 볼륨 마운트 포인트 생성
- `USER <user>`: 실행 사용자 설정
- `ARG <name>[=<default>]`: 빌드 시점 변수
- `LABEL <key>=<value>`: 메타데이터 추가
- `HEALTHCHECK`: 컨테이너 상태 확인 명령

8. 보안 실천 (Security Practices)

- 최신 버전 사용: `docker pull`로 최신 이미지 받기
- 권한 최소화: 필요 최소 권한으로 실행 (가능하면 root 권한 사용 안 함)
- 서드파티 이미지 주의: 신뢰할 수 있는 소스 사용
- 네트워크 제한: 꼭 필요한 네트워크만 연결
- 롤링 업데이트 사용: 무중단 업데이트로 서비스 지속성 유지
- 비밀 정보 관리: Docker Secrets나 환경 변수로 안전하게 관리
- 이미지 스캔: 취약점 검사 도구 사용
- 리소스 제한: CPU, 메모리 제한으로 시스템 보호

9. 유용한 예제

```
# 웹 서버 실행
docker run -d -p 80:80 --name webserver
nginx

# 데이터베이스 실행 (볼륨 사용)
docker run -d -p 3306:3306 -v
mysql_data:/var/lib/mysql \
  -e MYSQL_ROOT_PASSWORD=password --name
mysql mysql:8.0

# 개발 환경 설정
docker run -it -v $(pwd):/workspace \
  --name dev-env ubuntu:20.04 bash

# 로그 모니터링
docker logs -f --tail 100 container_name

# 컨테이너 리소스 사용량 확인
docker stats --format "table
{{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}"
```

10. Docker 보안 및 모범 사례

보안 강화

```
# 비루트 사용자로 컨테이너 실행
docker run --user 1000:1000 nginx

# 읽기 전용 파일시스템
docker run --read-only nginx

# 보안 옵션 설정
docker run --security-opt no-new-
privileges nginx

# 네트워크 격리
docker run --network none nginx

# 리소스 제한
docker run --memory=512m --cpus=1.0
nginx

# 환경 변수 보안
docker run -e MYSQL_ROOT_PASSWORD_FILE=/
run/secrets/mysql_root_password \
  --secret mysql_root_password nginx
```

이미지 보안 스캔

```
# Docker Scout로 취약점 스캔
docker scout quickview nginx:latest
docker scout cves nginx:latest

# Trivy로 보안 스캔
trivy image nginx:latest

# 이미지 서명 검증
docker trust inspect nginx:latest
```

멀티스테이지 빌드 최적화

```
# Build stage
FROM node:18-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

# Production stage
FROM node:18-alpine AS production
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nextjs -u 1001
WORKDIR /app
COPY --from=builder --
chown=nextjs:nodejs /app/node_modules ./
node_modules
COPY --chown=nextjs:nodejs . .
USER nextjs
EXPOSE 3000
CMD ["npm", "start"]
```

11. Docker Compose 고급 기능

환경별 설정

```
# docker-compose.yml
version: '3.8'
services:
  web:
    image: nginx:${NGINX_VERSION:-
latest}
    environment:
      - NODE_ENV=${NODE_ENV:-
development}
    env_file:
      - .env.local
    configs:
      - source: nginx_config
        target: /etc/nginx/nginx.conf

configs:
  nginx_config:
    file: ./nginx.conf

# .env.production
NGINX_VERSION=1.21
NODE_ENV=production
```

서비스 의존성 관리

```yaml
version: '3.8'
services:
  db:
    image: postgres:13
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U
postgres"]
      interval: 30s
      timeout: 10s
      retries: 3

  web:
    build: .
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
```

볼륨과 네트워크 고급 설정

```yaml
version: '3.8'
services:
  app:
    image: nginx
    volumes:
      - type: bind
        source: ./config
        target: /etc/nginx
        read_only: true
      - type: volume
        source: app_data
        target: /var/www
    networks:
      - frontend
      - backend

volumes:
  app_data:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /path/to/host/data

networks:
  frontend:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/16
  backend:
    driver: bridge
    internal: true
```

12. 프로덕션 배포 전략

Blue-Green 배포

```bash
# Blue 환경 (현재 운영)
docker-compose -f docker-
compose.blue.yml up -d

# Green 환경 (새 버전) 배포
docker-compose -f docker-
compose.green.yml up -d

# 트래픽 전환 (로드밸런서 설정 변경)
# Blue 환경 종료
docker-compose -f docker-
compose.blue.yml down
```

롤링 업데이트

```bash
# 서비스 업데이트
docker service update --image nginx:1.21
web

# 배치 크기 설정
docker service update --update-
parallelism 2 web

# 업데이트 실패 시 롤백
docker service rollback web
```

헬스체크 및 모니터링

```yaml
version: '3.8'
services:
  web:
    image: nginx
    healthcheck:
      test: ["CMD", "curl", "-f",
"http://localhost"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
    deploy:
      replicas: 3
      update_config:
        parallelism: 1
        delay: 10s
        failure_action: rollback
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
```

13. Docker Swarm 클러스터 관리

클러스터 초기화

```bash
# 매니저 노드 초기화
docker swarm init --advertise-addr
192.168.1.100

# 워커 노드 조인
docker swarm join --token SWMTKN-1-xxx
192.168.1.100:2377

# 클러스터 정보 확인
docker node ls
docker info
```

서비스 배포

```bash
# 서비스 생성
docker service create --name web --
replicas 3 nginx:latest

# 서비스 스케일링
docker service scale web=5

# 서비스 업데이트
docker service update --image nginx:1.21
web

# 서비스 제거
docker service rm web
```

스택 배포

```yaml
# docker-stack.yml
version: '3.8'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    deploy:
      replicas: 3
      placement:
        constraints:
          - node.role == worker
      resources:
        limits:
          cpus: '0.5'
          memory: 512M
        reservations:
          cpus: '0.25'
          memory: 256M
```

```bash
# 스택 배포
docker stack deploy -c docker-stack.yml
myapp
```

14. 컨테이너 오케스트레이션 고급 기능

시크릿 관리

```bash
# 시크릿 생성
echo "mysecretpassword" | docker secret
create db_password -

# 서비스에 시크릿 연결
docker service create --name db \
  --secret db_password \
  postgres:13

# 시크릿 목록 확인
docker secret ls
```

컨피그 관리

```bash
# 컨피그 생성
docker config create nginx_config
nginx.conf

# 서비스에 컨피그 연결
docker service create --name web \
  --config source=nginx_config,target=/
etc/nginx/nginx.conf \
  nginx:latest
```

네트워크 오버레이

```bash
# 오버레이 네트워크 생성
docker network create --driver overlay
--attachable mynetwork

# 서비스에 네트워크 연결
docker service create --name web \
```

```
  --network mynetwork \
  nginx:latest
```

15. 성능 최적화 및 모니터링

이미지 최적화

```
# Alpine Linux 사용
FROM node:18-alpine

# 멀티스테이지 빌드로 크기 최소화
FROM node:18-alpine AS deps
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production && npm
cache clean --force

FROM node:18-alpine AS runner
WORKDIR /app
COPY --from=deps /app/node_modules ./
node_modules
COPY . .
RUN addgroup -g 1001 -S nodejs && \
    adduser -S nextjs -u 1001 && \
    chown -R nextjs:nodejs /app
USER nextjs
EXPOSE 3000
CMD ["npm", "start"]
```

리소스 모니터링

```
# 컨테이너 리소스 사용량 실시간 모니터링
docker stats --format "table
{{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.NetIO}}\t{{.BlockIO}}"

# 특정 컨테이너의 상세 정보
docker inspect container_name | jq '.
[0].State'

# 로그 모니터링
docker logs -f --tail 100 container_name
```

성능 튜닝

```
# 컨테이너 메모리 제한
docker run --memory=1g --memory-swap=2g
nginx

# CPU 제한
docker run --cpus=2.0 nginx

# I/O 제한
docker run --device-read-bps /dev/
sda:1mb --device-write-bps /dev/sda:1mb
nginx

# 네트워크 대역폭 제한
docker run --network-alias web nginx
```

16. 트러블슈팅 및 디버깅

일반적인 문제 해결

```
# 컨테이너가 시작되지 않는 경우
docker logs container_name
docker inspect container_name

# 디스크 공간 부족
docker system df
docker system prune -a

# 네트워크 연결 문제
docker network ls
docker network inspect network_name

# 볼륨 문제
docker volume ls
docker volume inspect volume_name
```

디버깅 도구

```
# 컨테이너 내부 접속
docker exec -it container_name /bin/bash

# 컨테이너 프로세스 확인
docker top container_name

# 컨테이너 파일시스템 변경사항
docker diff container_name

# 컨테이너 이벤트 모니터링
docker events --filter
container=container_name
```

로그 관리

```
# 로그 드라이버 설정
docker run --log-driver=json-file --log-
opt max-size=10m --log-opt max-file=3
nginx

# 중앙화된 로깅 (ELK 스택)
docker run --log-driver=fluentd --log-
opt fluentd-address=localhost:24224
nginx

# 로그 로테이션
docker run --log-opt max-size=10m --log-
opt max-file=3 nginx
```