나는 도커 장인이야! 3주차 2부

문성훈 발표 / 2024. 11. 10 (일)

Docker & Kubernates 실전 가이드 : 55~66장

Volumes VS Bind mounts

- Volumes와 bind mounts의 주된 차이점은 무엇일까?

Volumes VS Bind mounts

- 주된 차이점 : ‘도커(컨테이너) 외부에서 데이터를 직접 수정할 수 있냐의 여부’

Anonymous Volumes VS Named Volumes

- Volume에는 **익명볼륨**과 **명명된 볼륨**(혹은 이름있는 볼륨)으로 나뉜다.
- 그렇다면 이 둘의 차이는 또 무엇일까?

Anonymous Volumes VS Named Volumes

- Anonymus Volumes
  - 여러 컨테이너에서 볼륨 같이 쓰기 불가능
- Named Volumes
  - 여러 컨테이너에서 볼륨 같이 쓰기 가능

Anonymous Volumes VS Named Volumes

타입/유형	익명 볼륨	이름있는 볼륨
생성 목적	특정 컨테이너에 대해 사용할 때	특정 컨테이너에 종속되지 않음 -> 여러 컨테이너에서 같이 사용할 수 있음.
지속성	컨테이너 종료/재시작 시 유지 (하지만 --rm 사용 시엔 자동 삭제)	컨테이너 종료/재시작 시 유지 (Docker CLI로 직접 삭제 해야함)
공유 가능 여부	다른 컨테이너와 공유 불가	다른 컨테이너와 공유 가능
재사용 가능 여부	익명이므로 재사용 불가 (같은 이미지에서도 사용 불가)	동일한 컨테이너에서 재사용 가능 (재시작 포함)
사용 예시	docker run -v /app/data ...	docker run -v data:/app/data ... -> 볼륨이름:컨테이너_내부_위치

Anonymous Volumes VS Named Volumes

추가 설명 : Anonymous Volumes

- Dockerfile에서 정의 가능

```
VOLUME [ "/app/feedback" ] <- 외부로 드러낼 컨테이너 안에 내부 위치
```

- 익명 볼륨은 주기적으로 관리해주는게 좋음.
  - —rm 으로 실행안하면, 익명 볼륨이 계속 쌓이기 때문.
    - 뭐 물론,, Named Volume도 쌓이겠지만, 걔는 컨테이너 간에 공유해서 쓰기때문에 부담이 덜하니깐!!
  - (비하인드) 4달전에 지인이 서버빌려줘서 도커 열심히 썼는데… 한날 내 서버인줄 알고 자연스레 prune했다가.. 친구한테 혼났다는…

Anonymous Volumes VS Named Volumes

추가 설명 : Named Volumes

- 어떻게든 컨테이너가 제거되어도, 볼륨은 남아있음.
  - (익명볼륨의 경우엔, rm명령어로 실행시, 자동제거가 되었지만… Named는 삭제되지 않음.)
  - 그래서 컨테이너 상태(종료, 삭제 등)와 상관없이 데이터 유지를 위해 사용
- (익명볼륨과 달리) 명명된 볼륨은 docker CLI에서 정의 함
  - 즉, Dockerfile에서 명명된 볼륨을 VOLUME [“/app/data”] 이런식으로 정의하지 못함.
  - docker run .... -v "{지정할_볼륨이름}:{컨테이너 내부 경로}” 꼭 콜론앞에 경로가 아닌 이름
- 그럼에도 호스트 파일 시스템에서 수정은 불가능함
  - 왜냐면 로컬(호스트) 머신내 어디에 저장될지 모르니깐.

Bind mounts

- 로컬 호스트내 특정 위치를 컨테이너로 마운트 함

Named Volume (CLI)

```
docker run .... -v "{지정할_볼륨이름}:{컨테이너 내부 경로}"
// 콜론앞에 경로가 아닌게 붙으면, 네임드 볼륨이 됨
```

Bind Mounts (CLI)

```
docker run .... -v "{로컬경로}:{컨테이너 내부 경로}"
// 콜론앞에 경로가 아닌게 붙으면, 네임드 볼륨이 됨
```

Bind mounts

타입/유형	익명 볼륨	이름있는 볼륨	Bind mounts
생성 목적	특정 컨테이너에 대해 사용할 때	특정 컨테이너에 종속되지 않음 -> 여러 컨테이너에서 같이 사용할 수 있음.	호스트 파일 시스템의 위치 즉, 어느 컨테이너에도 종속되지 않음
지속성	컨테이너 종료/재시작 시 유지 (하지만 --rm 사용 시엔 자동 삭제)	컨테이너 종료/재시작 시 유지 (Docker CLI로 직접 삭제 해야함)	- 컨테이너 종료/재시작 시 유지 – 호스트 파일 시스템에서 삭제 가능
공유 가능 여부	다른 컨테이너와 공유 불가	다른 컨테이너와 공유 가능	다른 컨테이너와 공유 가능
재사용 가능 여부	익명이므로 재사용 불가 (같은 이미지에서도 사용 불가)	동일한 컨테이너에서 재사용 가능 (재시작 포함)	동일한 컨테이너에서 재사용 가능 (재시작 포함)
사용 예시	docker run -v /app/data ...	docker run -v data:/app/data ... -> 볼륨이름**:**컨테이너_내부_위치	docker run -v /path/to/code:/app/code ... ->이름있는 볼륨에서 {볼륨이름}이 아닌 {경로}로 작성

.dockerignore

- 이미지를 빌드할 때 “COPY”에서 제외 할 디렉토리/파일 명시
  - ‘/node_modules’를 지정하면, 이미지 빌드 시 node_modules 디렉토리 제외
    -> **이미지 크기 감소 효과**
    -> 빌드 시간 단축
  - ‘node_modules’외에도 .git 같은거도 뺌.
  - 그 외에도, 민감 정보가 담긴 파일들 제외 가능

ARGument VS **ENV**ironment

- arg는 하나의 Dockerfile에서 여러개의 다양한 이미지를 만들 때 사용
  - Dockerfile에서 빌드 시점에만 사용 가능한 변수

- env는 하나의 이미지에서 여러개의 다양한 컨테이너를 만들 때 사용
  - 컨테이너 실행 시점에도 사용할 수 있는 변수

ARGument VS **ENV**ironment

타입/유형	ARG	ENV
설명	**- Dockerfile 내부에서만 사용 가능(빌드시점)** - CMD 또는 애플리케이션 코드에서는 접근 불가	**- Dockerfile 및 애플리케이션 코드에서 사용 가능**
설정 방법	이미지 빌드 시 설정 (`docker build`의 `--build-arg` 옵션 사용)	컨테이너 종료/재시작 시 유지 (Docker CLI로 직접 삭제 해야함)
사용 예시	Dockerfile를 수정하지 않고, 여러개의 값을 가진 다양한 이미지를 만들고 싶을때	하나의 이미지로 유연한 변수를 가지는 컨테이너로 실행하고 싶을때
주의 사항	이것도 도커 레이어를 추가함. 그래서 캐싱전략을 위해 순서를 잘 조작해야함.	

ARGument VS **ENV**ironment

ENV

ENV는 도커파일, 애플리케이션에서 사용 가능

```
ENV {환경변수_이름} {기본_값}
```

ENV 사용 예시 (Dockerfile)

```
ENV PORT 80
EXPOSE $PORT // 도커파일에서 환경변수를 사용할때 `$이름` 사용
```

활용 예시 (Dockerfile - **도커 파일 내에서도 활용 가능**

ARGument VS ENVironment

ENV

ENV는 도커파일, 애플리케이션에서 사용 가능

```
docker run .... --env PORT=8080

또는

docker run .... -e PORT=8080

또는

docker run .... --env-file ./.env
// .env 파일에 PORT=8080
```

DockerCLI에서 환경변수(ENV) 주입 방법