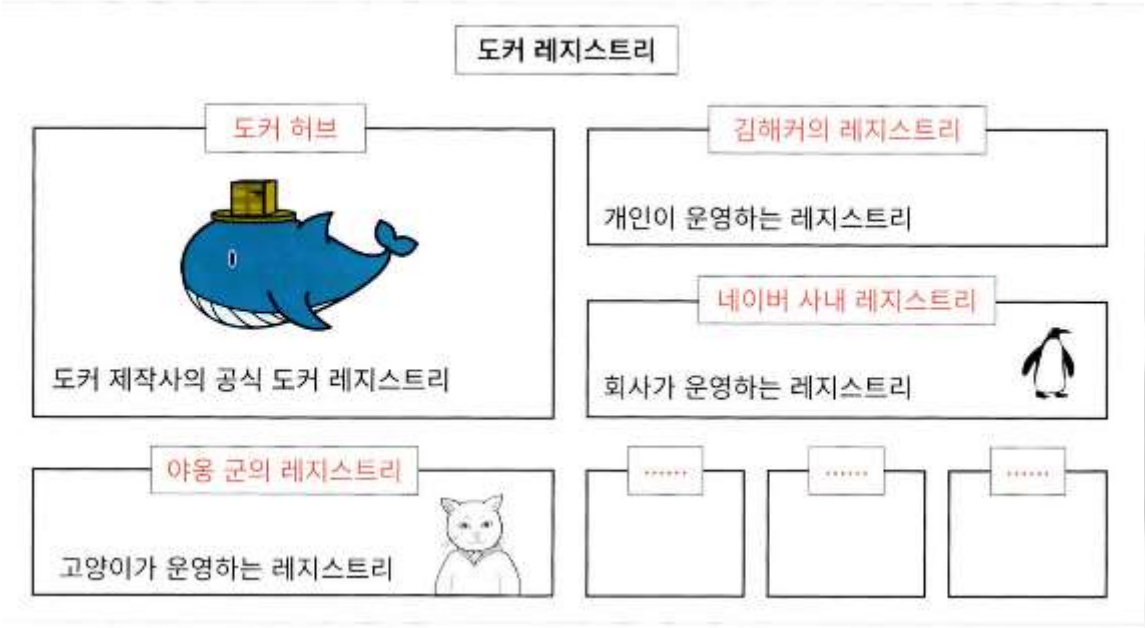
Command

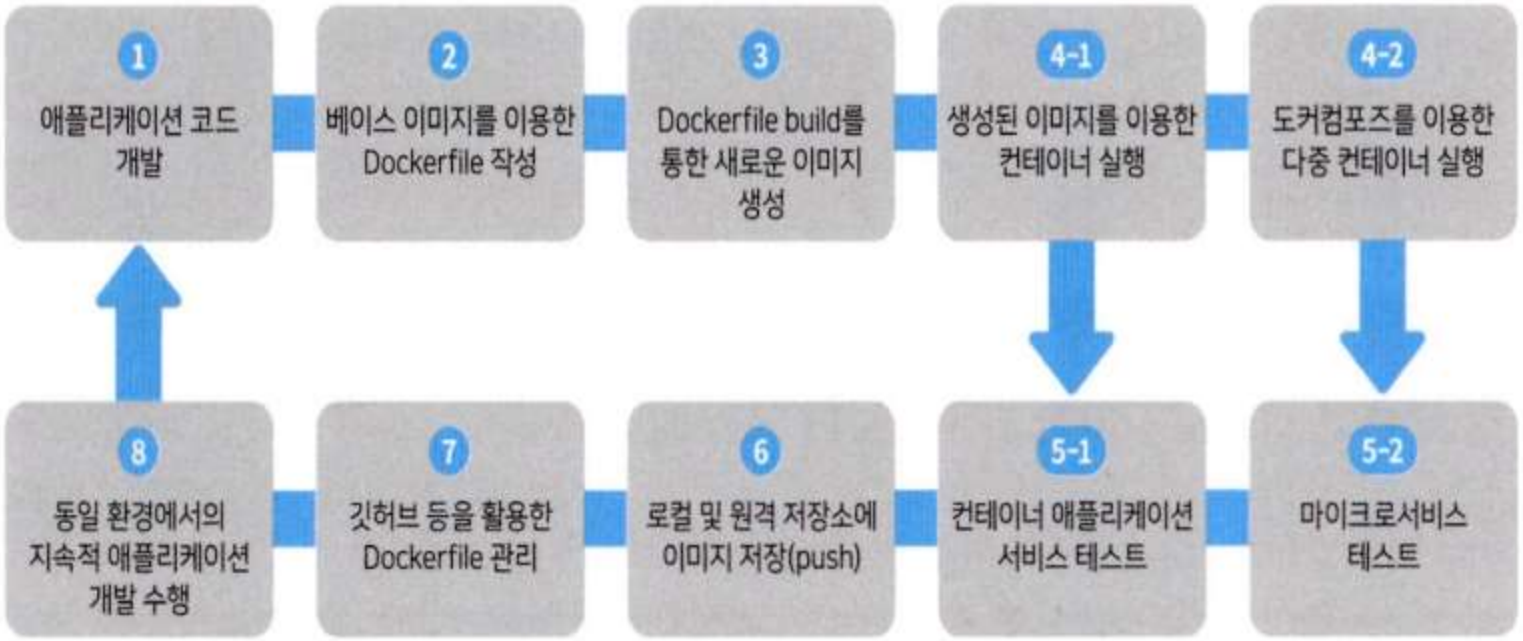
Docker Registry

- Docker Registry
  - ✓ Image를 배포하는 장소가 Docker Registry
  - ✓ Docker Hub는 Docker 제작사에서 운영하는 공식 Docker Registry로 Apache나 MySQL, Ubuntu의 공식 Image 모두 Docker Hub에 참여해 Docker Hub에서 Image를 배포하는데 run 커맨드를 사용할 때 내려받는 Image를 제공하는 사이트
  - ✓ Private Registry를 직접 생성할 수 있으며 CSP의 Registry 이용 가능
  - ✓ Repository는 Registry를 구성하는 단위



Docker Container Service

❖ Docker Container Application Service Development Flow



Docker Container Service

- Docker Container Application Service Development Flow
  - ✓ Life Cycle
    - ① 애플리케이션 코드 개발: 특정 서비스 구동을 위한 애플리케이션 코드 및 웹 화면 구성 등을 위한 코드를 개발
    - ② Base Image를 이용한 Dockerfile 작성: 개발에 필요한 인프라 구성 요소를 Dockerfile에 작성하는데 Docker Hub를 통해 base image를 다운로드하고 다양한 구동 명령어 (FROM, RUN, CMD. ENDPOINT, ENV, ADD 등) 와 작성한 애플리케이션 코드, 라이브러리, 여러 도구를 Dockerfile에 포함시키는 과정
    - ③ Dockerfile build를 통한 새로운 Image 생성: docker build 명령을 통해 작성한 Dockerfile을 실행하는 단계로 단계 별로 실행되는 로그를 화면에서 확인하며 오류 발생 내용도 확인
    - ⑤ 생성된 Image를 이용한 컨테이너 실행: docker images 명령을 통해 생성된 Image를 확인하고 Image를 통한 컨테이너를 구동(docker run)
    - ④ Docker Compose를 이용한 다중 컨테이너 실행: Docker 실행 옵션을 작성한 docker-compose.yml을 통해 다중 컨테이너 간 실행 순서, 네트워크, 의존성 등을 통합 관리할 수 있고 마이크로서비스 개발에 활용하는데 하나의 docker-compose.yml 파일이 아닌 여러 개의 yml 파일로 구성된 서비스 개발도 가능
    - ⑤ 컨테이너 애플리케이션 서비스 테스트: nginx를 이용한 웹 애플리케이션 컨테이너 서비스였다면 연결하는 IP 와 포트 번호를 이용하여 웹 브라우저를 이용한 페이지 연결을 확인
    - ⑤ 마이크로서비스(MSA) 테스트

Docker Container Service

- Docker Container Application Service Development Flow
  - ✓ Life Cycle
    - ⑥ 로컬 및 원격 저장소에 Image 저장: 로컬 및 원격에 있는 Image 저장소에 생성한 Image를 저장하여 다른 팀 간의 공유 및 지속적인 Image 관리를 수행
    - ⑦ Git Hub 등을 활용한 Dockerfile 관리: Dockerfile 코드를 Git Hub 사이트에 저장 및 관리할 수 있고 Docker Hub 사이트와 연동하게 되면 자동화된 빌드 기능을 이용한 Image 생성도 가능
    - ⑧ 동일 환경에서의 지속적 애플리케이션 개발 수행: 이전 과정을 통해 업무용 애플리케이션 Image를 지속적으로 개발, 운영 및 관리

Docker Container Service

- ❖ IaC – 코드형 인프라
  - ✓ Docker Container를 이용한 Application 개발 과정에서 눈 여겨 볼 것은 컨테이너 동작에 필요한 모든 내용을 사전에 코드로 작성하여 Ansible, Chef, Puppet, Vagrant 와 같은 Infra Provisioning 도구로 자동화하게 되면 필요할 때마다 애플리케이션 및 서버 환경을 적은 비용으로 빠르게 개발, 배포, 확장할 수 있는데 이러한 개념이 IaC(Infrastructure as Code)
  - ✓ 개발자는 애플리케이션 개발, 테스트, 배포 시마다 모든 인프라 구성 요소를 하나 하나 수동적으로 체크하거나 맞출 필요가 없고 변경 불가능한 인프라 환경에서 언제든 동일한 상태에서의 개발이 가능해지며 버전 업이나 패치 등의 작업이 필요하면 기존 Image를 변경하지 않고 해당 작업을 수행한 새로운 Image를 생성하여 신규 인프라 서버로 사용 가능

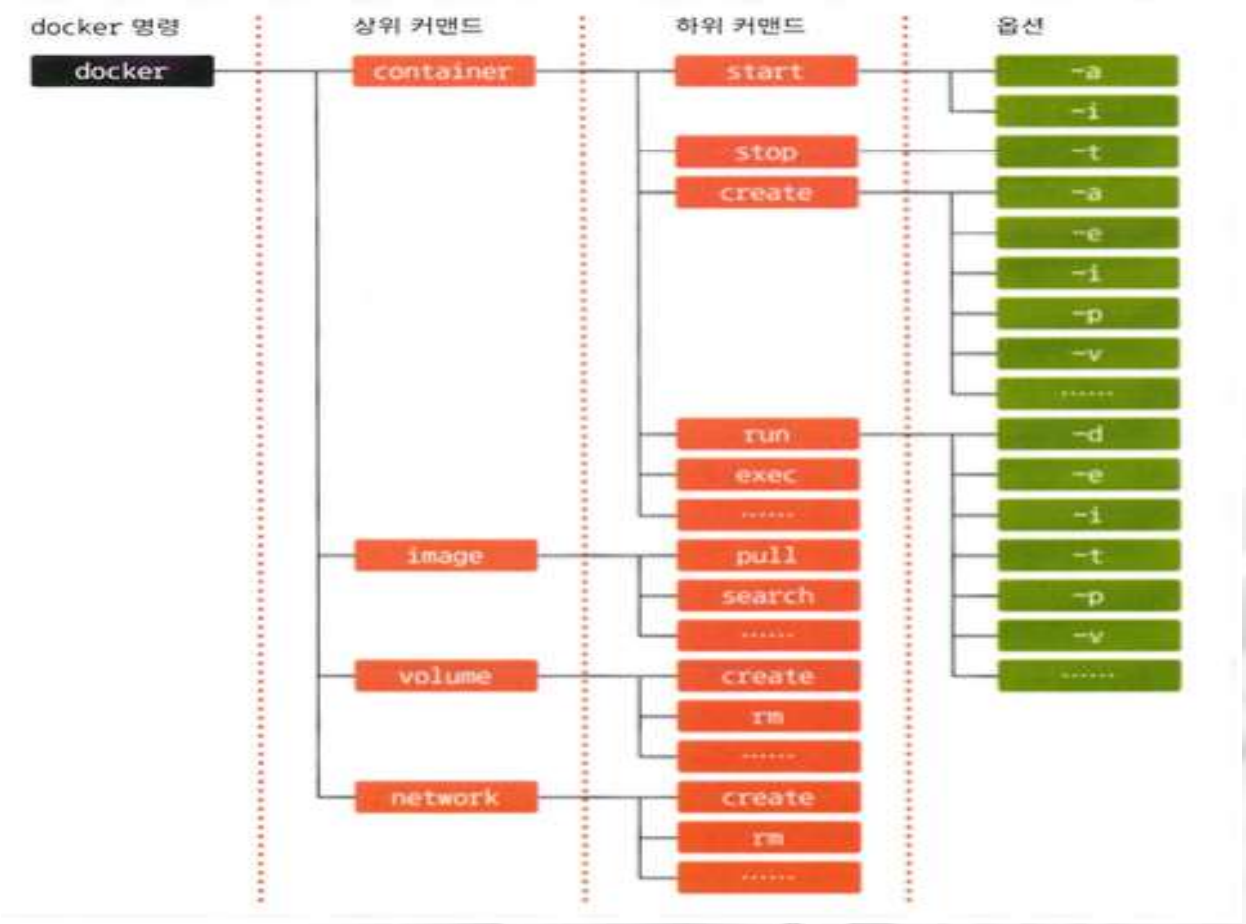
Docker Command

- Docker 명령어 사용
  - ✓ Docker의 작업 명령어는 docker로 시작
  - ✓ 기본 형식

    docker 명령어 옵션 대상 인자
    - penguin Image를 다운로드: docker image pull penguin
    - penguin Image를 실행: docker container run penguin
    - penguin 컨테이너 시작: docker container start penguin
    - penguin 컨테이너를 백그라운드에서 시작: docker container run -d penguin –mode=1
  - ✓ 커맨드는 상위 커맨드 와 하위 커맨드로 나누어지는 경우도 있는데 상위 커맨드를 생략해도 되는 경우가 있음

Docker Command

- Docker 명령어 사용
  - ✓ Docker 명령



Docker Command

- Docker 명령어 종류
  - ✓ Image 조작을 위한 하위 커맨드

하위 커맨드	내용	생략 가능 여부	주요 옵션
pull	도커 허브 등의 리포지토리에서 이미지를 내려받음	O	거의 사용하지 않음
rm	도커 이미지를 삭제	*2	거의 사용하지 않음
ls	내려 받은 이미지의 목록을 출력	X	거의 사용하지 않음
build	도커 이미지를 생성	O	-t

Docker Command

- Docker 명령어 종류
  - ✓ 컨테이너 조작을 위한 하위 커맨드

하위 커맨드	내용	생략 가능 여부	주요 옵션
start	컨테이너를 실행	O	-i
stop	컨테이너를 정지	O	거의 사용하지 않음
create	도커 이미지로부터 컨테이너를 생성	O	--name -e -p -v
run	도커 이미지를 내려받고 컨테이너를 생성해 실행함(다운로드는 필요한 경우에만). docker image pull, docker container create, docker container start라는 세 개의 명령을 하나로 합친 것과 같다.	O	--name -e -p -v -d -i -t
rm	정지 상태의 컨테이너를 삭제	O	-f -v
exec	실행 중인 컨테이너 속에서 프로그램을 실행	O	-i -t
ls	컨테이너 목록을 출력	*1	-a
cp	도커 컨테이너와 도커 호스트 간에 파일을 복사	O	거의 사용하지 않음
commit	도커 컨테이너를 이미지로 변환	O	거의 사용하지 않음

Docker Command

- Docker 명령어 종류
  - ✓ 볼륨 조작을 위한 하위 커맨드

하위 커맨드	내용	생략 가능 여부	주요 옵션
create	볼륨을 생성	X	--name
inspect	볼륨의 상세 정보를 출력	X	거의 사용하지 않음
ls	볼륨의 목록을 출력	X	-a
prune	현재 마운트되지 않은 볼륨을 모두 삭제	X	거의 사용하지 않음
rm	지정한 볼륨을 삭제	X	거의 사용하지 않음

Docker Command

- Docker 명령어 종류
  - ✓ 네트워크 조작을 위한 하위 커맨드

하위 커맨드	내용	생략 가능 여부	주요 옵션
connect	컨테이너를 도커 네트워크에 연결	X	거의 사용하지 않음
disconnect	컨테이너의 도커 네트워크 연결을 해제	X	거의 사용하지 않음
create	도커 네트워크를 생성	X	거의 사용하지 않음
inspect	도커 네트워크의 상세 정보를 출력	X	거의 사용하지 않음
ls	도커 네트워크의 목록을 출력	X	거의 사용하지 않음
prune	현재 컨테이너가 접속하지 않은 네트워크를 모두 삭제	X	거의 사용하지 않음
rm	지정한 네트워크를 삭제	X	거의 사용하지 않음

Docker Command

- Docker 명령어 종류
  - ✓ 그 이외 커맨드

상위 커맨드	내용
checkpoint	현재 상태를 일시적으로 저장한 후, 나중에 해당 시점의 상태로 되돌릴 수 있다. 현재 실험적 기능이다.
node	도커 스웜의 노드를 관리하는 기능
plugin	플러그인을 관리하는 기능
secret	도커 스웜의 비밀값 정보를 관리하는 기능
service	도커 스웜의 서비스를 관리하는 기능
stack	도커 스웜 또는 쿠버네티스에서 여러 개의 서비스를 합쳐 구성한 스택을 관리하는 기능
swarm	도커 스웜을 관리하는 기능
system	도커 엔진의 정보를 확인하는 기능

Docker Command

❖ Docker 명령어 종류

✓ 단독으로 사용되는 커맨드

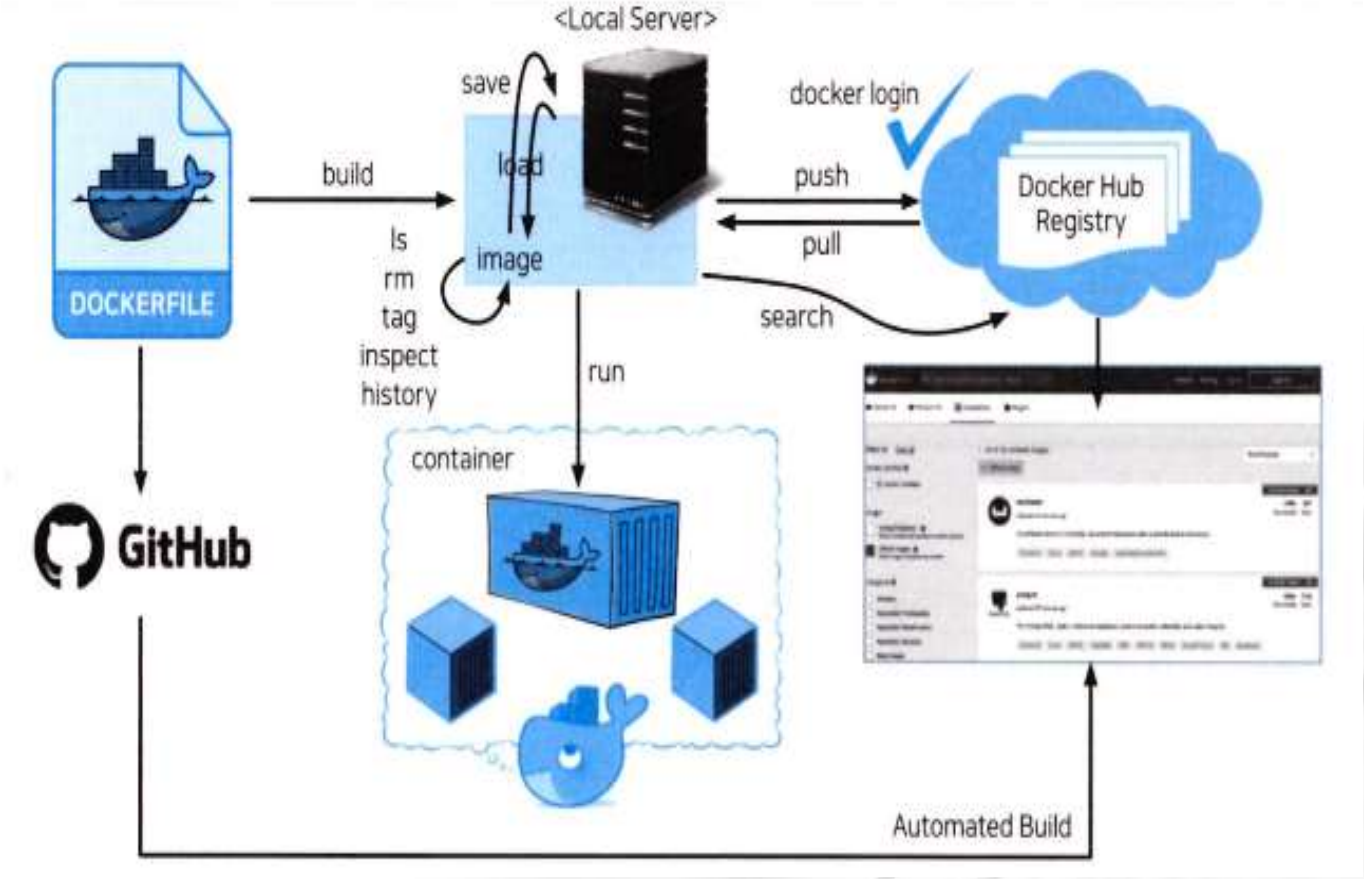
단독 커맨드	내용	주요 옵션
login	도커 레지스트리에 로그인	-u -p
logout	도커 레지스트리에 로그아웃	거의 사용하지 않음
search	도커 레지스트리를 검색	거의 사용하지 않음
version	도커 엔진 및 명령행 도구의 버전을 출력	거의 사용하지 않음

Docker Image Command

- Docker Image
  - ✓ Docker 이미지는 컨테이너 형태로 소프트웨어를 배포하기 위해 필요한 모든 요소를 실행할 수 있는 포맷으로 컴파일 및 빌드한 패키지
  - ✓ Docker 이미지는 독립적이기 때문에 의존성을 고려할 필요가 없으며 경량화된 패키지이므로 비교적 작은 용량으로도 제 역할을 수행할 수 있음
  - ✓ Docker 이미지는 특정 시점의 Docker 컨테이너 상태를 담은 스냅숏이라고 할 수 있음
  - ✓ Docker 이미지를 통해 동일한 환경을 가진 여러 개의 컨테이너를 손쉽게 생성할 수 있음
  - ✓ Docker 이미지는 여러 개의 레이어로 구성되어 있고 Docker Hub와 같은 중앙 저장소에 저장되어 관리

Docker Image Command

- Docker Image
  - ✓ docker image 관련 명령



Docker Image Command

- docker search - image 검색
  - ✓ Docker Hub에는 모든 Image의 기반이 되는 운영체제(CentOS, Ubuntu 등) Repository, Language Runtime 이나 Middleware 등 이 관리되는 Repository가 있기 때문에 Docker Image를 직접 만드는 대신 제공되는 Image를 사용할 수 있음
  - ✓ docker search 명령을 사용하면 Docker Hub에 등록된 Repository를 검색할 수 있음
  - ✓ docker search [options] 검색_키워드
  - ✓ mysql image를 5개로 제한해서 검색

    docker search --limit 5 mysql
    - mysql과 관련된 Repository 목록을 확인 할 수 있는데 검색 결과 첫 번째에 나오는 mysql Repository는 Repository 이름에 네임스페이스가 생략돼 있는데 이 Repository가 mysql 공식 Repository
    - 공식 Repository의 네임스페이스는 일률적으로 library 이므로 이 Repository의 이름도 library/mysql이 되는데 공식 Repository의 네임스페이스는 생략 가능

Docker Image Command

- 이미지 다운로드: docker [image] pull [OPTIONS] name[:TAG | @IMAGE_DIGEST]
  - ✓ jenkins image 다운로드

    docker image pull jenkins/jenkins:lts
  - ✓ debian image 다운로드

    docker pull debian

```
Using default tag: latest
latest: Pulling from library/debian
1e4aec178e08: Pull complete
Digest:
    sha256:43ef0c6c3585d5b406caa7a0f232ff5a19c1402aeb415f68bcd1cf9d10180a
    f8
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
```

Docker Image Command

- 이미지 다운로드: docker [image] pull [OPTIONS] name[:TAG | @IMAGE_DIGEST]
  - ✓ docker image pull 결과 확인
    - Image 이름 뒤에 :태그를 포함하지 않으면 자동으로 최신 버전(latest)으로 지정되므로 기본 태그 값이 latest라고 출력되는데 특정 버전을 지정하게 되면 latest 대신 지정한 버전 명이 포함됨
    - library는 Image를 저장하고 있는 네임스페이스로 Docker 이미지명의 기본 형식은 <네임스페이스>/<이미지명>:<태그>이고 별도로 특정 Registry를 지정하지 않으면 자동으로 Docker Hub의 라이브러리가 네임스페이스로 지정
    - Image의 분산 해시(distribution hash) 값을 표시하는데 다운로드한 Image는 여러 계층(layer)으로 만들어지는데 그 중 Image의 핵심 정보를 바이너리 형태의 정보로 제공
    - Digest는 Docker Hub에서 관리하는 Image의 고유 식별 값을 의미하고 이 값을 포함한 조회는 docker images --digests 옵션을 사용
    - status는 다운로드 한 Image 정보가 로컬에 저장되었음을 나타내는 상태 표시
    - docker.io는 Docker Hub 의 Image 저장소 주소

Docker Image Command

- 이미지 다운로드: docker [image] pull [OPTIONS] name[:TAG | @IMAGE_DIGEST]
  - ✓ docker image pull 옵션
    - -a, --all-tags: 저장소에 태그로 지정된 여러 Image를 모두 다운로드
    - --disable-content-trust: Image 검증 작업을 건너뛰는 것으로 기본값은 true
    - --platform: 플랫폼을 지정하는 것으로 --platform=linux 형태로 설정
    - -q, --quiet: Image 다운로드 과정에서 나타나는 상세 출력 숨김

Docker Image Command

- 이미지 다운로드: docker [image] pull [OPTIONS] name[:TAG | @IMAGE_DIGEST]
  - ✓ 이미지 다운로드
    - 버전 정보 설정

      docker pull debian:latest
    - 저장소 설정

      docker pull docker.io/library/debian:latest

Docker Image Command

- ❖ 이미지 다운로드: docker [image] pull [OPTIONS] name[:TAG | @IMAGE_DIGEST]
  - ✓ 이미지 다운로드
    - ● 외부 Registry 활용: 구글에서 제공하는 샘플 Image

      docker pull gcr.io/google-samples/hello-app:1.0

      1.0: Pulling from google-samples/hello-app
      fbad7aa519f7: Pull complete
      fda4ba87f6fb: Pull complete
      74c357f2d0d6: Pull complete
      Digest:
      sha256:845f77fab71033404f4cfceaa1ddb27b70c3551ceb22a5e7f4498cdda6c9daea
      Status: Downloaded newer image for gcr.io/google-samples/hello-app:1.0
      gcr.io/google-samples/hello-app:1.0

Docker Image Command

- Image 정보 확인: docker images - docker image ls
  - ✓ 관련 정보
    - REPOSITORY: Image 이름
    - TAG: 버전 정보로 Image를 내려받을 때 따로 지정하지 않으면 latest(최신 버전}를 내려받는데 특정 버전을 설치하고자 하는 경우는 Image_이름:버전_넘버 형태로 설정
    - IMAGE ID: Image 식별자로 본래는 64글자이지만 앞에서부터 12글자만 출력
    - CREATED: Image 생성 후 경과된 시간
    - SIZE: Image의 전체 용량

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
debian	latest	54e726b437fb	7 days ago	124MB
mysql	latest	57da161f45ac	8 days ago	517MB
jenkins/jenkins	lts	f16216f97fcb	8 days ago	467MB
mariadb	latest	5986006dc2c5	9 days ago	400MB
mongo	latest	a440572ac3c1	13 days ago	639MB
busybox	latest	66ba00ad3de8	6 weeks ago	4.87MB
jaspeen/oracle-xe-11g	latest	52fbd1fe2d7a	7 years ago	792MB

Docker Image Command

- docker image 세부 정보 확인: docker image inspect
  - ✓ docker image inspect [OPTIONS] Image이름
    - --format 이나 -f 옵션으로 JSON 형식의 정보 중 원하는 형식의 정보만 출력 가능(go 템플릿으로 설정)
    - 세부 정보
      - image ID: "Id"
      - 생성일: "Created"
      - Docker 버전: "DockerVersion"
      - CPU 아키텍처: "Architecture"
      - 이미지 다이제스트 정보: "RootFS"
      - 이미지 레이어 저장 정보: "GraphDriver"

Docker Image Command

- ❖ docker image 세부 정보 확인: docker image inspect
  - ✓ 실습
    - 이미지 조회: docker search httpd
    - 이미지 다운로드: docker image pull httpd:latest
    - 이미지 상세 조회: docker image inspect httpd
    - 이미지 생성 날짜: docker image inspect --format="{{ .Created}}" httpd
    - 이미지와 태그 확인: docker image inspect --format="{{.RepoTags}}" httpd
    - 운영체제 조회: docker image inspect --format="{{ .Os}}" httpd

Docker Image Command

- docker image history
  - ✓ 이미지 구성을 위해 사용된 레이블 정보와 각 레이어의 수행 명령, 크기 등 을 조회
  - ✓ 형식

    docker image history Image이름
  - ✓ docker image history httpd

```
IMAGE          CREATED        CREATED BY                                      SIZE
COMMENT
a3e79aafef7f   2 months ago   CMD ["httpd-foreground"]                          0B
buildkit.dockerfile.v0
<missing>      2 months ago   EXPOSE map[80/tcp:{}]                            0B
buildkit.dockerfile.v0
<missing>      2 months ago   COPY httpd-foreground /usr/local/bin/ # buil...   138B
buildkit.dockerfile.v0
<missing>      2 months ago   STOPSIGNAL SIGWINCH                              0B
buildkit.dockerfile.v0
<missing>      2 months ago   RUN /bin/sh -c set -eux;    savedAptMark="$(a...
69.2MB    buildkit.dockerfile.v0
<missing>      2 months ago   ENV HTTPD_PATCHES=                               0B
buildkit.dockerfile.v0
...
```

Docker Image Command

- docker image history
  - ✓ docker image history httpd
    - 출력 결과 중 CREATED BY 열을 보면 특정 이미지를 구성하기 위해 사용된 명령과 환경 설정 정보 등을 볼 수 있는데 정보 중 용량을 가지고 있는 라인이 레이어
    - CMD, EXPOSE, ENV, WORKDIR 등의 명령을 통해 베이스 이미지에 필요한 설정 정보를 결합하여 새로운 이미지를 생성

Docker Image Command

- ❖ docker image history
  - ✓ 데비안 리눅스와 달리 Apache 웹 서버 이미지인 httpd는 다운로드한 레이어 수가 더 많음
    - 데비안

      Using default tag: latest

      latest: Pulling from library/debian

      6d11c181ebb3: Pull complete

      Digest: sha256:27586f4609433f2f49a9157405b473c62c3cb28a581c413393975b4e8496d0ab

      Status: Downloaded newer image for debian:latest

      docker.io/library/debian:latest

Docker Image Command

- ❖ docker image history
  - ✓ 데비안 리눅스와 달리 Apache 웹 서버 이미지인 httpd는 다운로드한 레이어 수가 더 많음
    - httpd

      Using default tag: latest

      latest: Pulling from library/httpd

      14c9d9d19932: Pull complete

      f5db40045454: Pull complete

      4f4fb700ef54: Pull complete

      ac0ad684e55d: Pull complete

      b59792d2b7f1: Pull complete

      0ffcdbb5bd41: Pull complete

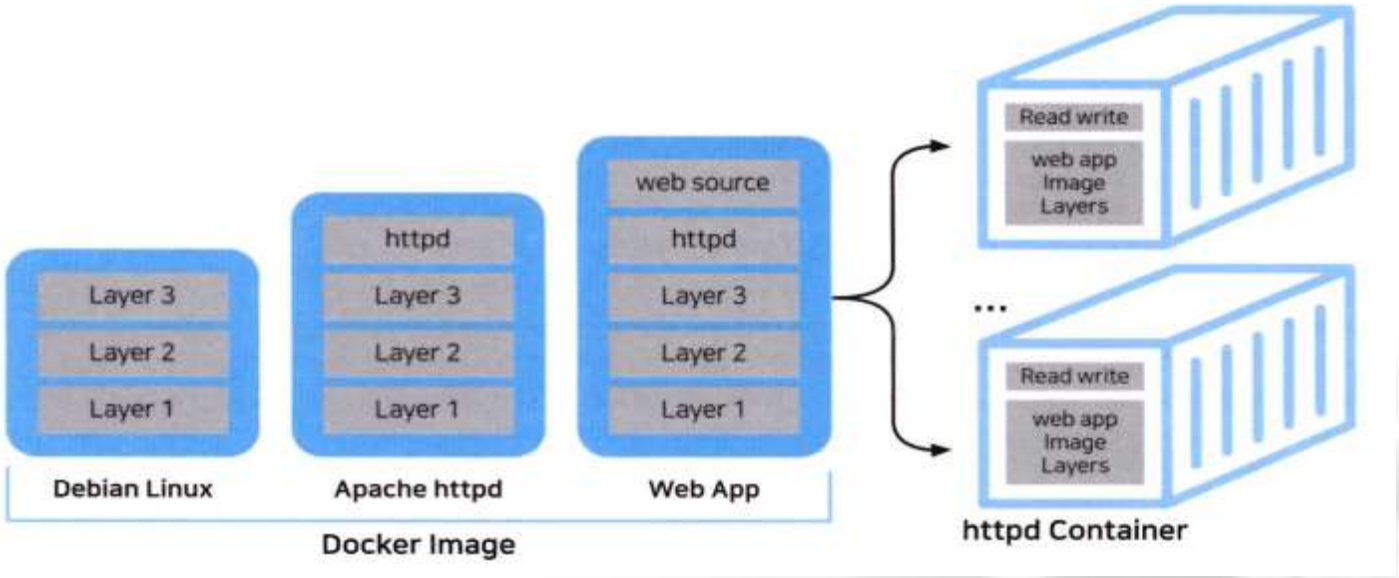
      Digest: sha256:7204bce27072f97f244337ebe93c1dfc93d358d103beefc4107ee359d74d9148

      Status: Downloaded newer image for httpd:latest

      docker.io/library/httpd:latest

Docker Image Command

- Docker 유니언 파일 시스템



Docker Image Command

- Docker 유니언 파일 시스템
  - ✓ httpd
    - Docker 이미지 구조의 기본 운영체제 레이어들을 쌓음
    - 운영체제 베이스 이미지 위에 Apache 웹 서버를 설치한 레이어를 올림
    - Apache 웹 서비스에 필요한 리소스 정보 및 환경 정보가 포함된 레이어를 올림
    - 이미지는 불변의 읽기 전용 레이어들의 집합 구조인 유니언 파일 시스템
    - Docker 이미지를 실행하면 여러 개의 컨테이너를 구동할 수 있는데 각각의 컨테이너에서 발생한 모든 변경 정보를 저장하기 위해 읽고-쓰기 레이어를 두고 저장
  - ✓ 그림을 보면 이미지를 실행하여 여러 개의 컨테이너를 구동한 것을 볼 수 있는데 처음 내려받은 이미지는 수백 메가 용량을 가지고 있지만 컨테이너를 구동할 때마다 이미지를 내려받지 않고 로컬에 저장된 이미지를 계속 사용하며 이미지 레이어의 상단에 있는 웹 애플리케이션 소스 레이어의 환경 설정 및 리소스 설정이 변경되어 이미지로 변경되더라도 기존 레이어를 제외한 변경된 웹 소스 레이어만 내려받아 사용하기 때문에 효율적

Docker Image Command

- ❖ 읽고 쓰기 레이어
  - ✓ 호스트 운영체제에서 확인
    - ● Docker 저장소에 사용되는 스토리지 드라이버 조회
      - ◆ $ docker info | grep Storage

        WARNING: bridge-nf-call-iptables is disabled

        WARNING: bridge-nf-call-ip6tables is disabled

        Storage Driver: overlay2
      - ◆ overlay2
        - ▪ overlay2는 Linux 배포판에 대해서 선호되는 스토리지 드라이버
        - ▪ Container의 Storage를 제한하기 위해서는 Docker Engine 자체의 기능을 이용하지 않고 Storage Driver의 도움이 필요

Docker Image Command

- ❖ 읽고 쓰기 레이어
  - ✓ 호스트 운영체제에서 확인
    - ● Docker의 모든 데이터 및 로그 정보 저장소 확인
      - ◆ sudo su -
      - ◆ cd /var/lib/docker
      - ◆ ls

        buildkit containers engine-id image network overlay2 plugins runtimes swarm tmp volumes

Docker Image Command

- ❖ 읽고 쓰기 레이어
  - ✓ 호스트 운영체제에서 확인
    - ● 이미지 레이어 데이터 확인
      - ◆ cd image/overlay2/layerdb/sha256/
      - ◆ ls

02f8c31257289b5c1901a3e78de1f0fdad6efa10522e46960f18b13407feba4b
054df1200f3e1d70e2ebeafa03c965d0dd34e7dab90b4ea24963352e5590b573
18e07544fb833059a49110efedd57b923421de8d5b7a24e72313710976fe873a
19fe9b91aa30a4b437a57ce4a7a52b4cd98ee6f283e0db3b0b204b7f2fd10779
20312b5745843a1d409495e32b3cb4aaf48fc5f02eba6ae34ec2dcee82a5886a
5467c0f933c7741ed2d345c50462b3699253bccd19b9bb0ba10514a1efabed72
881dbaddd6cfed0aff1e4efa183e338ce6b5f16b6ffb39373731bbf77b5939f3
8badcfdd9076e3c27484535c1e425870eac2895f15d23039e0e6b2ca6f9caf29
9450f98d66df133758fb2e1a5ef26a11c0f4acc5d5ce69b3db771d6040e90463
fc96334e6952158c406789d23f6644be6f80e8277c17a2851d6e998bf7b8d545

Docker Image Command

- ❖ 읽고 쓰기 레이어
  - ✓ 호스트 운영체제에서 확인
    - ● httpd 의 이미지 레이어
      - ◆ $ docker image inspect --format="{{ .RootFS.Layers }}" httpd

        [sha256:054df1200f3e1d70e2ebeafa03c965d0dd34e7dab90b4ea24963352e5590b573
        sha256:d39a942a42952672fee8dd744376e2395e0caf193463f52100d292468967f59c
        sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
        sha256:2aeff8d3e2159b6ee484c889f2edae8f689027d2a80406c8fb964d79fc548726
        sha256:2db08903db836722fc2ed533faa2f46166de12af6fbe01dd64fcbfe8d95e3cd1
        sha256:cdac3ce322a3d473a7138d89959e45fa9fb2a29ead16d3e08afd6919c438cca0]

Docker Image Command

- 읽고 쓰기 레이어
  - ✓ 호스트 운영체제에서 확인
    - 일치하는 디렉터리의 diff로 이동하고 cached-id 파일 정보를 통해 실제 데이터가 저장된 디렉터리의 다이제스트값을 확인
      - sudo su –
      - cd /var/lib/docker/image/overlay2/layerdb/sha256/054df1200f3e1d70e2ebeafa03c965d0dd34e7dab90b4ea24963352e5590b573
      - ls

      cache-id diff size tar-split.json.gz
      - # cat cache-id

      2896ea143bfeb8c7c471274838a851b6ee45c3969088838bb89ab07a9e6683d0

Docker Image Command

- ❖ 읽고 쓰기 레이어
  - ✓ 호스트 운영체제에서 확인
    - ● 이미지 실행을 통해 만들어지는 컨테이너의 최상위 경로/영역
      - ◆ cd /var/lib/docker/overlay2/
      - ◆ ls

2896ea143bfeb8c7c471274838a851b6ee45c3969088838bb89ab07a9e6683d0
297d8f9b9537e6d33a2d36089b8371ae35facd5de62cb3accf2780b1e301f8e4
574a4320eef5ecf40fc756e87d55d5c869c454dd09fcce6cf4c94dd8f4adc0c4
70e05f1d6e95e51eb9e8f1c8540e046ccfc1515c30422ea6414581dcd3847e97
7ebcac8da76f7d6deb5d62296d63b28e893cc40a789b75040974c84a46247235
7fdc65bfc566179744e83eb3a364e6066736a96d7398bb774b7a58617d668851
afb6b0e347cfc4c41584a0f594a74f1fe44542262341c99f810ed17553449068
b78ca71401f74d83acbb030395445d190a2a5766a401337c69145b04600fa9c6
cd11dbcedac8b3edd6cf65ad7e25fdbcaf72d9317ed0ea67b057ea5611700e67
fe7e250d219910e2af9e93112d6d2a52ffe315589af769724f15f0e2b0ed727e

Docker Image Command

- ❖ 읽고 쓰기 레이어
  - ✓ 호스트 운영체제에서 확인
    - ● 이미지 실행을 통해 만들어지는 컨테이너의 최상위 경로/영역
      - ◆ cd 2896ea143bfeb8c7c471274838a851b6ee45c3969088838bb89ab07a9e6683d0
      - ◆ ls

        committed diff link
      - ◆ cd diff
      - ◆ ls

        bin boot dev etc home lib media mnt opt proc root run sbin srv sys tmp usr var

Docker Image Command

- ❖ 읽고 쓰기 레이어
  - ✓ 호스트 운영체제에서 확인
    - ● 다른 터미널을 이용해서 실행
      - ◆ docker run -it -p 80:80 --name=webserver1 httpd /bin/bash
      - ◆ ls

        bin dev home media opt root sbin sys usr

        boot etc lib mnt proc run srv tmp var
      - ◆ touch Hello-docker

        Hello-docker bin dev home media opt root sbin sys usr

        boot etc lib mnt proc run srv tmp var

Docker Image Command

- ❖ 읽고 쓰기 레이어
  - ✓ 호스트 운영체제에서 확인
    - ● 이전 터미널에서 수행
      - ◆ # find / -name 'Hello-docker'

        /var/lib/docker/overlay2/d5c83793c9c1931702036b9dc388228ad0fe6e3ce274957a8827086858bb292a/merged/Hello-docker

        /var/lib/docker/overlay2/d5c83793c9c1931702036b9dc388228ad0fe6e3ce274957a8827086858bb292a/diff/Hello-docker

Docker Image Command

- ❖ 업로드
  - ✓ Docker Hub 로그인 관련 명령
    - docker login
    - docker logout
  - ✓ Image 이름 과 태그
    - docker의 태그는 Registry에 업로드를 상정한 Image 이름
    - 태그는 Registry의 주소와 버전 표기를 추가해 정식 명칭으로 생성

      adamsoft.com(Registry 주소)/adam(레포지토리 이름):13(버전)
    - 태그 이름 부여

      docker tag 원본_Image_이름[:버전] 참조_Image_이름[:버전]
    - docker Hub에 업로드

      docker push 이름

Docker Image Command

- ❖ 업로드
  - ✓ Image 이름 과 태그
    - docker images - docker image ls

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
jenkins/jenkins	lts	641faa6c9469	2 weeks ago	498MB
debian	latest	f1dc429dc895	2 weeks ago	139MB
httpd	latest	251be5552bf4	8 weeks ago	177MB
gcr.io/google-samples/hello-app	1.0	dd1b12fcb609	7 months ago	27.2MB

Docker Image Command

- ❖ 업로드
  - ✓ Image 이름 과 태그
    - docker image tag 251be5552bf4 myhttpd:1.0
    - docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
jenkins/jenkins	lts	641faa6c9469	2 weeks ago	498MB
debian	latest	f1dc429dc895	2 weeks ago	139MB
httpd	latest	251be5552bf4	8 weeks ago	177MB
myhttpd	1.0	251be5552bf4	8 weeks ago	177MB
gcr.io/google-samples/hello-app	1.0	dd1b12fcb609	7 months ago	27.2MB

Docker Image Command

- 업로드
  - ✓ Image 이름 과 태그
    - docker image tag httpd:latest myhttpd:2.0
    - docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
jenkins/jenkins	lts	641faa6c9469	2 weeks ago	498MB
debian	latest	f1dc429dc895	2 weeks ago	139MB
httpd	latest	251be5552bf4	8 weeks ago	177MB
myhttpd	1.0	251be5552bf4	8 weeks ago	177MB
myhttpd	2.0	251be5552bf4	8 weeks ago	177MB
gcr.io/google-samples/hello-app	1.0	dd1b12fcb609	7 months ago	27.2MB

Docker Image Command

- ❖ 업로드
  - ✓ Image 이름 과 태그
    - 업로드 하고자 하는 경우 태그 설정: docker image tag httpd:latest ggangpae1/myhttpd:3.0
    - docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
jenkins/jenkins	lts	641faa6c9469	2 weeks ago	498MB
debian	latest	f1dc429dc895	2 weeks ago	139MB
ggangpae1/myhttpd	3.0	251be5552bf4	8 weeks ago	177MB
httpd	latest	251be5552bf4	8 weeks ago	177MB
myhttpd	1.0	251be5552bf4	8 weeks ago	177MB
myhttpd	2.0	251be5552bf4	8 weeks ago	177MB
gcr.io/google-samples/hello-app	1.0	dd1b12fcb609	7 months ago	27.2MB

Docker Image Command

- ❖ 업로드
  - ✓ Image 이름 과 태그
    - ● 업로드: docker push ggangpae1/myhttpd:3.0

      The push refers to repository [docker.io/ggangpae1/myhttpd]

      347fbe85df8e: Mounted from library/httpd

      67dd04ce35fa: Mounted from library/httpd

      b1c326bcb393: Mounted from library/httpd

      5f70bf18a086: Mounted from library/httpd

      6f334184ae05: Mounted from library/httpd

      2bd1a2222589: Mounted from library/httpd

      3.0: digest: sha256:63b28cbc60e4d8dfe3900b56aeadc21456187acbf56d1b0697dda3238224eca3 size: 1572

Docker Image Command

- ❖ 업로드
  - ✓ Image 이름 과 태그
    - ● 다운로드: docker pull ggangpae1/myhttpd:3.0

      3.0: Pulling from ggangpae1/myhttpd

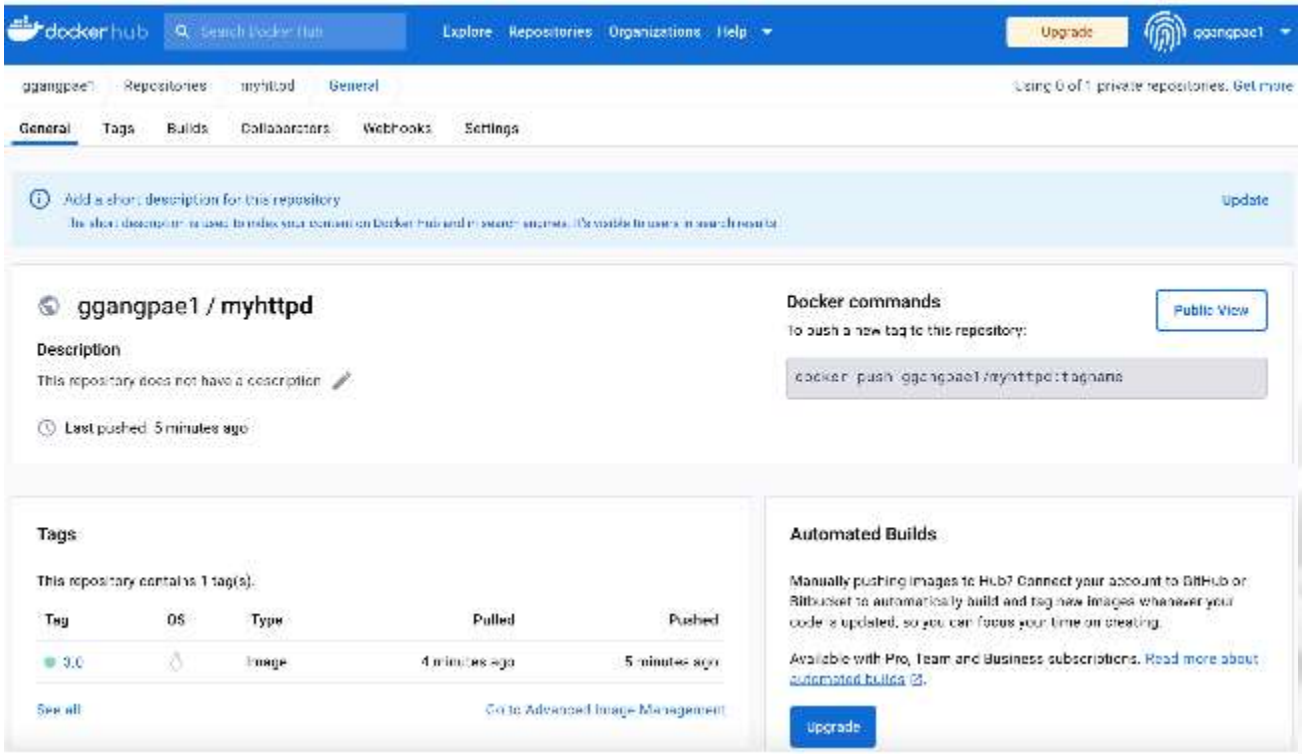
      Digest:
      sha256:d866e5c91f31fc6a122aaf37149cc67ba2ca0de68ae73ab206747a190937967e

      Status: Image is up to date for ggangpae1/myhttpd:3.0

      docker.io/ggangpae1/myhttpd:3.0

Docker Image Command

- ❖ 업로드
  - ✓ Docker Hub에서 확인
    - https://hub.docker.com



Docker Image Command

- Docker Image를 파일로 관리
  - ✓ 원본 Image의 레이어 구조까지 복제를 수행해서 tar(Tape Archiver) 확장자로 Image를 저장
  - ✓ 명령어
    - 저장: docker image save [옵션] Image_이름 > 파일_경로
    - 로드: docker image load < 파일_경로

Docker Image Command

- Docker Image를 파일로 관리
  - ✓ Image 저장 및 로드
    - Image 다운로드: docker pull eclipse/mysql
    - Image 확인: docker images
    - Image 저장: docker image save eclipse/mysql > test-mysql57.tar
    - 파일 확인: ls 또는 dir
    - 원본 Image 삭제: docker image rm eclipse/mysql
    - Image 확인: docker images
    - Image 가져오기: docker image load < test-mysql57.tar
    - Image 확인: docker images
  - ✓ Image의 용량을 줄이고자 하는 경우는 tar 다음에 .gz를 추가
  - ✓ 모든 Image 저장: docker image save -o 파일명 $(docker image ls -q)

Docker Image

- Docker Image 삭제
  - ✓ 이미지 삭제: docker image rm(docker rmi) [옵션] {Image 이름[:태그] | Image ID}
    - 이미지 다운로드: docker pull ubuntu:14.04
    - 이미지 확인: docker images
    - 이미지 삭제: docker image rm ubuntu:14.04
    - 이미지 확인: docker images

Docker Image

- Docker Image 삭제
  - ✓ 이미지 삭제 시 latest 버전을 제외한 나머지는 반드시 태그명을 명시
  - ✓ 관련된 이미지를 모두 삭제할 때는 -f 옵션 추가
  - ✓ 셸 스크립트 변수 활용
    - 이미지 전체 삭제 - docker rmi $ (docker -images -q)
    - 특정 이미지 이름이 포함된 것만 삭제 - docker rmi $(docker images | grep debian)
    - 반대로 특정 이미지 이름이 포함된 것만 제외하고 모두 삭제 - docker rmi $(docker images | grep -v centos)
  - ✓ 사용되지 않는 image 파기
    - 사용 중이 아닌 모든 이미지 제거: docker image prune -a
    - 사용 중이 아닌 48시간 이전(--filter)의 모든 이미지(-a)를 별도 확인 없이(-f) 제거: docker image prune -a -f --filter "until=48h"

Docker Image Command

- docker image history
  - ✓ 기존 컨테이너 와 이미지 삭제
    - 모든 Docker 컨테이너 삭제

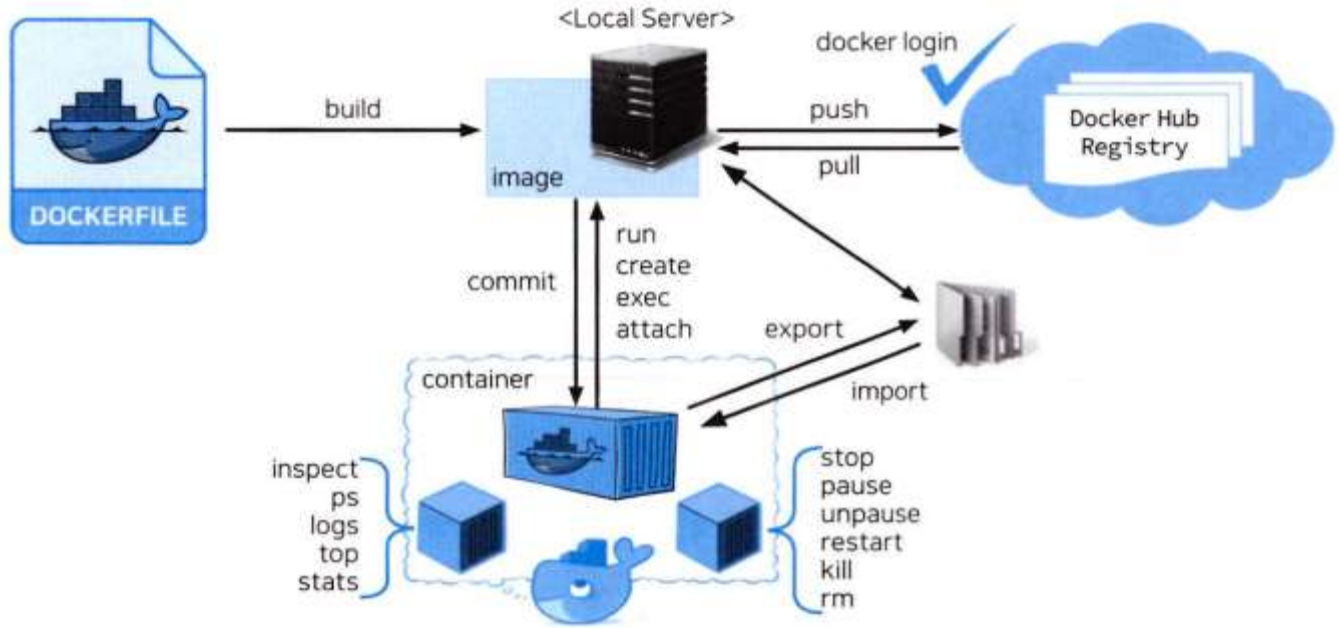
      docker stop $(docker ps -a -q)

      docker rm $(docker ps -a -q)
    - 모든 Docker 이미지 삭제

      docker rmi $(docker images -q)

Docker Container Command

- Container로 애플리케이션 실행
  - ✓ 이미지는 읽기 전용의 불변 값으로 만들어지는데 이러한 이미지를 바탕으로 Docker 엔진은 컨테이너를 생성할 수 있는데 이때 이미지와 함께 읽고 쓰기가 가능한 레이어를 추가해서 만들어지는 것이 컨테이너
  - ✓ 이미지와 마찬가지로 컨테이너 명령도 dockerd daemon 이 제공하는 docker CLI API를 통해 제공
  - ✓ Docker 이미지는 컨테이너 동작과 관련된 콘텐츠를 제공하고 이를 바탕으로 컨테이너의 동작이 이루어지므로 컨테이너 명령 대부분이 서비스 실행 및 운영과 관련되어 있음



Docker Container Command

- Container 는 Process
  - ✓ Docker 컨테이너는 Docker 이미지를 기반으로 만들어지는 snapshot
  - ✓ snapshot은 읽기 전용의 Docker 이미지 레이어를(불변의 유니언 파일 시스템 - UFS) 복제한 것이고 그 위에 읽고 쓰기가 가능한 컨테이너 레이어를 결합하면 컨테이너
  - ✓ 컴퓨터 애플리케이션의 동작은 프로세스를 통해 이루어지는데 컨테이너는 격리된 공간에서 프로세스가 동작하는 기술
  - ✓ 컨테이너는 바로 프로세스 격리 기술(namespaces, cgroups, chroot)의 표준으로 정의된 OCI(Open Container Initiative)로 컨테이너 포맷과 런타임에 대한 개방형 업계 표준을 만들기 위한 목적으로 리눅스 파운데이션(Linux Foundation)의 지원을 받아 구성된 프로젝트
  - ✓ 명령어 docker run~을 사용하면 컨테이너가 동작하게 되고 가상의 격리 환경에 독립된 프로세스가 동작하는데 서버 호스트 운영체제가 독립적으로 동작하는 것과 유사
  - ✓ 리눅스 호스트 운영체제를 부팅하면 PID 1번은 init(systemd) 프로세스가 동작하며 이 프로세스는 나머지 모든 시스템 프로세스의 부모 프로세스가 됨

Docker Container Command

- ❖ Container로 애플리케이션 실행
  - ✓ Image 다운로드 와 Container 생성 및 실행 과정
    - gihyodocker/echo:latest image 다운로드

      docker image pull gihyodocker/echo:latest
    - 다운로드 받은 image 실행 – 컨테이너 실행

      docker container run -t -p 9000:8080 gihyodocker/echo:latest
    - 다른 터미널을 실행시켜 확인 – 브라우저에서 실행 가능

      curl http://localhost:9000/

Docker Container Command

- Container로 애플리케이션 실행
  - ✓ 컨테이너 실행 과정
    - 컨테이너의 실행을 위해 docker run 명령을 사용하면 해당 Docker 이미지 복사본 스냅숏 레이어 위에 읽고 쓰기가 가능한 컨테이너 레이어를 추가한 뒤 docker start 명령으로 컨테이너를 시작
    - 실행된 컨테이너를 조회하는 방법은 docker ps 명령을 사용하는 것으로 리눅스 명령어인 ps(Process Status)는 리눅스 호스트에서 실행 중인 프로세스를 조회하는 방법을 제공하는데 이처럼 컨테이너 또한 프로세스라는 의미에서 docker ps 명령어를 사용

Docker Container Command

- Container로 애플리케이션 실행
  - ✓ 컨테이너 수동 제어
    - 컨테이너 생성: docker create -it --name container-test1 ubuntu:14.04
    - 실행 중인 컨테이너 확인: docker ps
    - 모든 컨테이너 컨테이너 확인: docker ps -a
    - 컨테이너 실행: docker start container-test1
    - 실행 중인 컨테이너 확인: docker ps
    - 컨테이너 접속: docker attach container-test1
    - root@01a74288d04c:/# exit
    - 컨테이너 삭제: docker rm container-test1
  - ✓ 동일 작업 수행
    - 컨테이너 생성 및 실행: docker run -it --name container-test1 ubuntu:14.04 bash
    - root@01a74288d04c:/# exit
    - 컨테이너 삭제: docker rm container-test1

Docker Container Command

- ❖ Container로 애플리케이션 실행
  - ✓ Docker Container의 생명 주기
    - ● 실행 상태
      - ◆ docker container run 명령의 인자로 지정된 Docker image를 기반으로 Container가 생성되면 이 image를 생성했던 Dockerfile에 포함된 CMD 및 ENTRYPOINT 인스트럭션에 정의된 애플리케이션이 실행되며 이 애플리케이션이 실행 중인 상태가 Container의 실행 중 상태가 됨
      - ◆ HTTP 요청을 받는 서버 애플리케이션이면 오류로 인해 종료되지 않는 한 실행 중 상태가 지속되므로 실행 기간이 길며 이에 비해 명령이 바로 실행되고 끝나는 명령행 도구 등의 Container는 실행 중 상태가 길게 유지되지 않음
      - ◆ 실행이 끝나면 정지 상태가 됨
    - ● 정지 상태
      - ◆ 실행 중 상태에 있는 Container를 사용자가 명시적으로 정지하거나 Container에서 실행된 애플리케이션이 정상/오류 여부를 막론하고 종료된 경우에는 Container가 자동으로 정지 상태가 됨
      - ◆ Container를 정지시키면 가상 환경으로서는 더 이상 동작하지 않지만 디스크에 Container가 종료되던 시점의 상태가 저장돼 남기 때문에 정지시킨 Container를 다시 실행할 수 있음

Docker Container Command

- ❖ Container로 애플리케이션 실행
  - ✓ Docker Container의 생명 주기
    - ● 파기 상태
      - ◆ 정지 상태의 Container는 명시적으로 파기하지 않는 이상 디스크에 그대로 남아 있음
      - ◆ Container를 자주 생성하고 정지해야 하는 상황에서는 디스크를 차지하는 용량이 점점 늘어나므로 불필요한 Container를 삭제하는 것이 바람직
      - ◆ 같은 image로 새로운 Container를 생성했다고 해도 각 Container가 실행된 시각 등이 서로 다르고 애플리케이션의 처리 결과도 이에 따라 달라질 수 있기 때문에 완전히 같은 Container를 새로 생성할 수는 없음

Docker Container Command

- Container로 애플리케이션 실행
  - ✓ docker run
    - docker image pull, docker container create, docker container start + command 의 기능을 하나로 합친 것과 같아서 Image를 내려받은 상태가 아니라면 먼저 Image를 내려받음
    - 기본 형식: docker run [옵션] Image (인자)
    - 사용할 수 있는 인자는 Image의 종류에 따라 달라지는데 MySQL Image처럼 로그인에 사용할 아이디 나 패스워드, 인코딩이나 정렬 순서를 인자로 받는 경우가 있지만 아무 인자도 지정하지 않는 경우도 많음

Docker Container Command

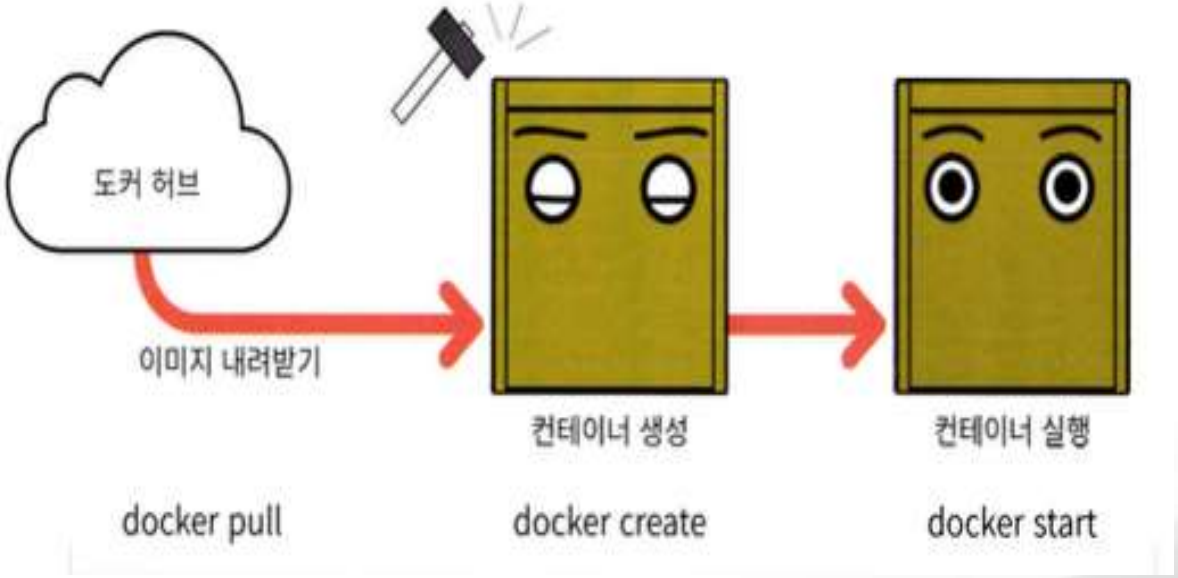
- ❖ Container로 애플리케이션 실행
  - ✓ docker run
    - ● 옵션
      - ◆ --name 컨테이너이름
        - ▪ 컨테이너 이름을 지정하는 옵션으로 미지정시 자동으로 부여
      - ◆ –p [Host Port]:[Container Port]
        - ▪ 외부 포트 와 컨테이너 내부 포트를 port forwarding
      - ◆ -P, --publish-all=[true | false]
        - ▪ 컨테이너 내부의 노출된 포트를 호스트 임의의 포트에 게시
      - ◆ –v, --volume=호스트경로:컨테이너경로
        - ▪ 볼륨 마운트
      - ◆ --net
        - ▪ 네트워크 연결
      - ◆ -i, --interface
        - ▪ 대화식 모드 연결
      - ◆ -t
        - ◆ 단말 디바이스 할당

Docker Container Command

- ❖ Container로 애플리케이션 실행
  - ✓ docker run
    - ● 옵션
      - ◆ -d, --detach=true
        - ▪ 백그라운드에서 컨테이너 실행 후 컨테이너 ID 등록
        - ▪ 데몬 형태로 동작하는 소프트웨어의 컨테이너는 -d. -i, -t 옵션을 사용하는 경우가 많으며 이들 옵션은 -dit와 같이 합쳐서 사용하면 편리
      - ◆ -e, --env
        - ▪ 환경 변수 지정
      - ◆ --rm
        - ▪ 컨테이너 종료 시 자동으로 컨테이너 파기
      - ◆ --restart
        - ▪ 컨테이너 종료 시 적용할 재시작 정책 지정(no | on-failure | on-failure:횟수 | always)
      - ◆ -h 컨테이너의호스트이름
        - ▪ 컨테이너의 호스트 이름을 지정하는 옵션으로 미지정시 컨테이너 ID가 호스트 이름으로 설정
      - ◆ --link=[container:container_id]
        - ▪ 동일 호스트의 다른 컨테이너 와 연결 설정으로 IP가 아닌 컨테이너 이름을 이용해 통신

Docker Container Command

- Container로 애플리케이션 실행
  - ✓ Image 다운로드 와 Container 생성 및 실행 과정



Docker Container Command

- ❖ Container로 애플리케이션 실행
  - ✓ 한번만 실행되는 컨테이너 와 데몬 형태로 동작하는 컨테이너
    - -d는 컨테이너를 백그라운드로 실행하는 옵션이고 –I 와 -t는 컨테이너 내부에 터미널로 접속하기 위해 사용하는 옵션
    - -d를 붙이지 않고 컨테이너를 실행하면 실행된 컨테이너가 프로그램의 실행을 마칠 때까지 터미널의 제어를 차지하므로 그 다음 명령을 입력할 수 없는 상태가 되며 -i, -t 옵션을 붙이지 않으면 컨테이너 안의 파일 시스템에 접근할 수 없음
    - 한 번만 실행되는 컨테이너는 실행하자마자 종료되므로 컨테이너가 터미널의 제어를 차지하더라도 일시적인 것이므로 문제가 되지 않음
    - 데몬처럼 계속적으로 실행되는 프로그램은 저절로 종료되지 않으므로 한번 터미널의 제어를 넘기면 이를 되찾아 오기가 번거로움
    - 컨테이너 속 파일 시스템을 다루려면 -i 또는 -t 옵션을 사용해야 하지만 실행하자마자 곧장 종료되는 컨테이너라면 컨테이너 속 파일 시스템에 손을 댈 일도 없으므로 불필요한 옵션

Docker Container Command

- ❖ Container로 애플리케이션 실행
  - ✓ Image 다운로드 와 Container 생성 및 실행 과정
    - ● 실행 중인 Container 확인

      docker ps

Docker Container Command

- ❖ Container로 애플리케이션 실행
  - ✓ Image 다운로드 와 Container 생성 및 실행 과정
    - docker ps 옵션
      - ◆ 컨테이너의 목록을 출력하는 명령으로 –a 옵션을 사용하면 정지 상태의 컨테이너를 포함해서 목록을 출력
      - ◆ 항목
        - CONTAINER ID: 컨테이너 식별자로 무작위 문자열이 할당되는데 64글자이지만 앞에서부터 12글자만 출력
        - IMAGE: 컨테이너를 만들 때 사용한 Image의 이름
        - COMMAND: 컨테이너 실행 시에 실행하도록 설정된 프로그램의 이름
        - CREATED: 컨테이너 생성 후 경과된 시간
        - STATUS: 컨테이너의 현재 상태로 실행 중이라면 Up이 그리고 종료된 상태라면 Exited
        - PORTS: 컨테이너에 할당된포트 번호로 호스트 포트 번호 -> 컨테이너 포트 번호 형식으로 출력되는데 포트 번호가 동일할 경우 -> 뒷부분은 출력되지 않음
        - NAMES: 컨테이너 이름

Docker Container Command

- Container로 애플리케이션 실행
  - ✓ Image 다운로드 와 Container 생성 및 실행 과정
    - 실행 중인 Container 중지
      - 특정 Container 중지: docker stop ContainerID 또는 Name
      - 모든 Container 중지: docker stop $(docker ps -a -q)
    - Container 삭제
      - 특정 Container 삭제: docker rm ContainerID 또는 Name
      - 모든 Container 삭제: docker rm $(docker ps -a -q)

Docker Container Command

- Container로 애플리케이션 실행
  - ✓ Apache 컨테이너를 이용한 실습 – Image 이름은 httpd 컨테이너 이름은 apa000ex1

Docker Container Command

- Container로 애플리케이션 실행
  - ✓ Apache 컨테이너를 이용한 실습
    - 컨테이너 생성 및 실행: docker run --name apa000ex1 -d httpd
    - ps 커맨드로 컨테이너 실행 상태를 확인: docker ps
    - stop' 커맨드로 컨테이너 정지: docker stop apa000ex1
    - ps 커맨드로 컨테이너 실행 상태를 확인: docker ps
    - rm 커맨드로 apa000ex1 컨테이너를 삭제: docker rm apa000ex1
    - ps 커맨드와 인자로 컨테이너가 삭제되었는지 확인: docker ps -a

Docker Container Command

- Container로 애플리케이션 실행
  - ✓ 웹 서비스 실행을 위한 nginx 컨테이너 실행
    - 이미지 다운로드: docker pull nginx:1.18
    - 이미지 확인: docker images

      nginx 1.18 c2c45d506085 22 months ago 133MB
    - 컨테이너 실행: docker run --name webserver1 -d nginx:1.18
    - 컨테이너 확인: docker ps

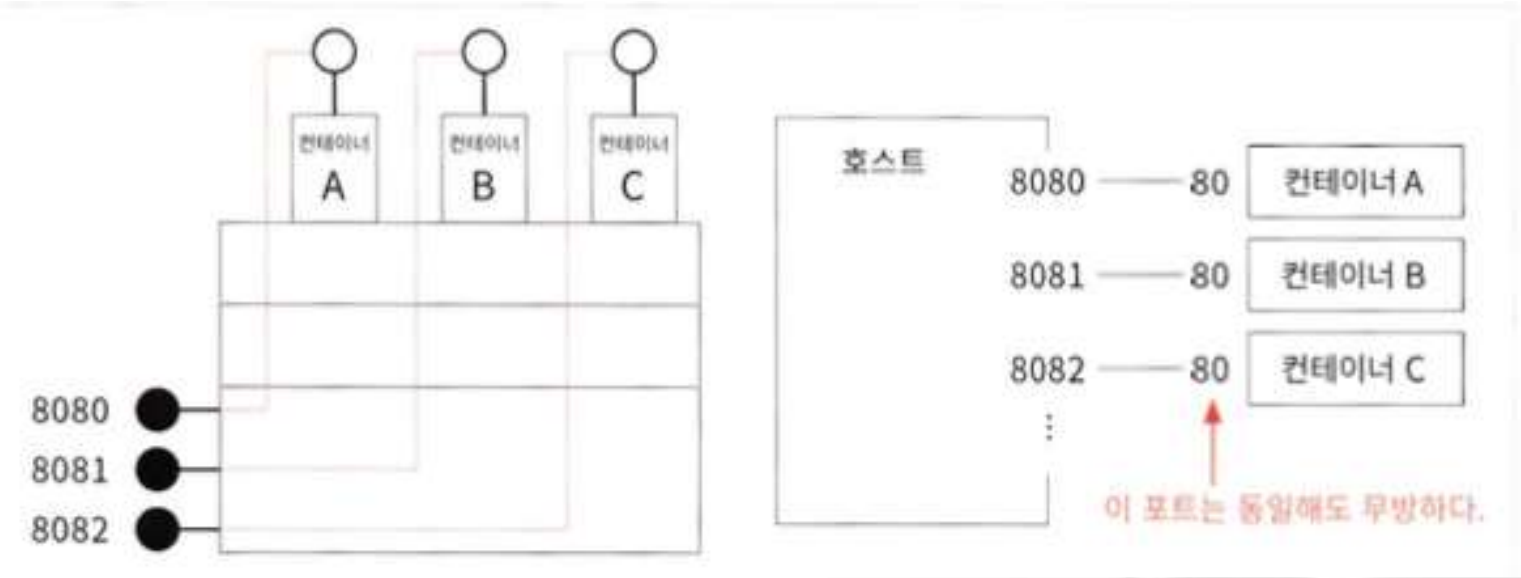
      6f8dba236131 nginx:1.18 "/docker-entrypoint...." 47 seconds ago Up 47 seconds 0.0.0.0:8001->80/tcp webserver1

Docker Container Command

- 생성한 컨테이너에 접근
  - ✓ 외부에서 컨테이너 내부에 개방된 포트에 접근하려면 외부와 접속하기 위한 설정이 필요한데 이를 위해 포트를 설정
  - ✓ Apache는 서버에서 정해둔 포트(80번 포트) 에서 웹 사이트에 대한 접근을 기다리다가 사용자가 이 포트를 통해 접근해 오면 요청에 따라 웹 사이트의 페이지를 제공하지만 컨테이너 속에서 실행 중 인 Apache는 외부와 직접 연결되지 않았기 때문에 외부에서 접근할 수 없음
  - ✓ port forwarding
    - 호스트 머신의 포트를 Container 포트와 연결해 Container 밖에서 온 통신을 Container 포트로 전달
    - 이 기능 덕분에 Container 포트를 Container 외부에서도 이용할 수 있음
    - docker container run 명령에서 -p 옵션을 붙이면 port forwarding을 지정할 수 있는데 -p 옵션값은 호스트_포트_번호:Container_포트_번호 형식으로 기술하면 됨

Docker Container Command

❖ 생성한 컨테이너에 접근



Docker Container Command

- 생성한 컨테이너에 접근
  - ✓ Apache 컨테이너를 이용한 실습 – Image 이름은 httpd 컨테이너 이름은 apa000ex2으로 하고 포트 설정은 8080:80
    - 컨테이너 생성 및 실행: docker run --name apa000ex2 -d -p 8080:80 httpd
    - ps 커맨드로 컨테이너 실행 상태를 확인: docker ps
    - 웹 브라우저를 통해서 Apache에 접근할 수 있는지 확인: http://localhost:8080
    - stop' 커맨드로 컨테이너 정지: docker stop apa000ex2
    - rm 커맨드로 apa000ex2 컨테이너를 삭제: docker rm apa000ex2
    - ps 커맨드와 인자로 컨테이너가 삭제되었는지 확인: docker ps -a

Docker Container Command

- 다양한 이미지
  - ✓ 리눅스 운영체제가 담긴 이미지

이미지 이름	컨테이너의 내용	컨테이너 실행에 주로 사용되는 옵션 및 인자
ubuntu	우분투	-d 없이 -it 옵션만 사용. 인자로는 /bin/bash 등 셸 명령어를 지정한다.
centos	CentOS	-d 없이 -it 옵션만 사용. 인자로는 /bin/bash 등 셸 명령어를 지정한다.
debian	데비안	-d 없이 -it 옵션만 사용. 인자로는 /bin/bash 등 셸 명령어를 지정한다.
fedora	페도라	-d 없이 -it 옵션만 사용. 인자로는 /bin/bash 등 셸 명령어를 지정한다.
busybox	BizyBox	-d 없이 -it 옵션만 사용. 인자로는 /bin/bash 등 셸 명령어를 지정한다.
alpine	알파인 리눅스	-d 없이 -it 옵션만 사용. 인자로는 /bin/bash 등 셸 명령어를 지정한다.

Docker Container Command

- 다양한 이미지
  - ✓ 웹 서버 및 데이터베이스 서버용 이미지

이미지 이름	컨테이너의 내용	컨테이너 실행에 주로 사용되는 옵션 및 인자
httpd	Apache	-d로 백그라운드로 실행. -p로 포트 번호 지정
nginx	Nginx	-d로 백그라운드로 실행. -p로 포트 번호 지정
mysql	MySQL	-d를 사용. 실행 시 -e MYSQL_ROOT_PASSWORD와 같이 루트 패스워드를 지정
postgres	PostgreSQL	-d를 사용. 실행 시 -e POSTGRES_ROOT_PASSWORD와 같이 루트 패스워드를 지정
mariadb	MariaDB	-d를 사용. 실행 시 -e MYSQL_ROOT_PASSWORD와 같이 루트 패스워드를 지정

Docker Container Command

- 다양한 이미지
  - ✓ 프로그래밍 언어의 런타임 이미지

이미지 이름	컨테이너의 내용	컨테이너 실행에 주로 사용되는 옵션 및 인자
openjdk	자바 런타임	-d를 사용하지 않고, 인자로 java 명령 등을 지정해 도구 형태로 사용한다.
python	파이썬 런타임	-d를 사용하지 않고, 인자로 python 명령 등을 지정해 도구 형태로 사용한다.
php	PHP 런타임	웹 서버가 포함된 것과 실행 명령만 포함된 것으로 나뉘어 제공된다.
ruby	루비 런타임	웹 서버가 포함된 것과 실행 명령만 포함된 것으로 나뉘어 제공된다.
perl	펄 런타임	-d를 사용하지 않고, 인자로 perl 명령 등을 지정해 도구 형태로 사용한다.
gcc	C/C++ 컴파일러	-d를 사용하지 않고, 인자로 gcc 명령 등을 지정해 도구 형태로 사용한다.
node	Node.js	-d를 사용하지 않고, 인자로 app 명령 등을 지정해 도구 형태로 사용한다.
registry	도커 레지스트리	-d 옵션을 사용해 백그라운드로 실행하며, -p 옵션으로 포트 번호를 지정한다.
wordpress	WordPress	-d 옵션을 사용해 백그라운드로 실행하며, -p 옵션으로 포트 번호를 지정한다. MySQL 또는 MariaDB가 필요하다. 접속에 필요한 패스워드는 -e 옵션으로 지정한다.
nextcloud	NextCloud	-d 옵션을 사용해 백그라운드로 실행한다. -p 옵션으로 포트 번호를 지정한다.
redmine	Redmine	-d 옵션을 사용해 백그라운드로 실행하며, -p 옵션으로 포트 번호를 지정한다. PostgreSQL 또는 MySQL이 필요하다.

Docker Container Command

- 다양한 이미지
  - ✓ 웹 서비스 실행을 위한 nginx 컨테이너 실행
    - 이미지 다운로드: docker pull nginx
    - 컨테이너 실행: docker run --name webserver1 -d -p 8001:80 nginx
    - 컨테이너 확인: docker ps
    - 컨테이너의 리소스 사용량 실시간 확인: docker stats webserver1
    - 컨테이너의 실행 중인 프로세스 확인: docker top webserver1
    - 컨테이너의 접근 로그를 Docker 명령으로 확인: docker logs -f webserver1
    - 컨테이너 중지: docker stop webserver1
    - 컨테이너 실행: docker start webserver1

Docker Container Command

- 관리 명령어
  - ✓ 실행 중인 Container를 제외한 모든 Container 파기
    - docker container prune
  - ✓ 사용하지 않는 Docker image 및 Container, 볼륨, 네트워크 등 모든 Docker 리소스를 일괄적으로 삭제
    - docker system prune
  - ✓ 사용 현황 확인
    - docker container stats

Docker Container Command

- Ubuntu 설치
  - ✓ Ubuntu image 검색: docker search --limit 25 ubuntu

```
(base) adam@Adams-MacBook-Pro docker % docker search --limit 25 ubuntu
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys…	12861	[OK]	
dorowu/ubuntu-desktop-lxde-vnc	Docker image to provide HTML5 VNC interface …	571		[OK]
websphere-liberty	WebSphere Liberty multi-architecture images …	280	[OK]	
rastasheep/ubuntu-sshd	Dockerized SSH service, built on top of offi…	255		[OK]
consol/ubuntu-xfce-vnc	Ubuntu container with "headless" VNC session…	241		[OK]
ubuntu-upstart	DEPRECATED, as is Upstart (find other proces…	113	[OK]	
ansible/ubuntu14.04-ansible	Ubuntu 14.04 LTS with ansible	98		[OK]
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5	ubuntu-16-nginx-php-phpmyadmin-mysql-5	50		[OK]
open-liberty	Open Liberty multi-architecture images based…	48	[OK]	
ubuntu-debootstrap	DEPRECATED; use "ubuntu" instead	44	[OK]	
i386/ubuntu	Ubuntu is a Debian-based Linux operating sys…	25		
nuagebec/ubuntu	Simple always updated Ubuntu docker images w…	24		[OK]
1and1internet/ubuntu-16-apache-php-5.6	ubuntu-16-apache-php-5.6	14		[OK]
1and1internet/ubuntu-16-apache-php-7.0	ubuntu-16-apache-php-7.0	13		[OK]
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mariadb-10	ubuntu-16-nginx-php-phpmyadmin-mariadb-10	11		[OK]
1and1internet/ubuntu-16-nginx-php-5.6-wordpress-4	ubuntu-16-nginx-php-5.6-wordpress-4	9		[OK]
1and1internet/ubuntu-16-nginx-php-5.6	ubuntu-16-nginx-php-5.6	8		[OK]
1and1internet/ubuntu-16-apache-php-7.1	ubuntu-16-apache-php-7.1	7		[OK]
1and1internet/ubuntu-16-nginx-php-7.1	ubuntu-16-nginx-php-7.1	5		[OK]
1and1internet/ubuntu-16-nginx-php-7.0	ubuntu-16-nginx-php-7.0	4		[OK]
1and1internet/ubuntu-16-sshd	ubuntu-16-sshd	1		[OK]
smartentry/ubuntu	ubuntu with smartentry	1		[OK]
1and1internet/ubuntu-16-php-7.1	ubuntu-16-php-7.1	1		[OK]
1and1internet/ubuntu-16-rspec	ubuntu-16-rspec	0		[OK]
ossobv/ubuntu	Custom ubuntu image from scratch (based on o…	0		

Docker Container Command

- Ubuntu 설치
  - ✓ Ubuntu image 다운로드: docker pull ubuntu

```
(base) adam@Adams-MacBook-Pro docker % docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
35807b77a593: Pull complete
Digest: sha256:9d6a8699fb5c9c39cf08a0871bd6219f0400981c570894cd8cbea30d3424a31f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Docker Container Command

- Ubuntu 설치
  - ✓ Ubuntu 컨테이너 생성: docker create -it --name ubuntu_server ubuntu

```
(base) adam@Adams-MacBook-Pro docker % docker create -it --name ubuntu_server ubuntu
09634fa86dfb2c9af132e900269d6fc97b5a0276b48e5caff6d563612c5cb9e4
(base) adam@Adams-MacBook-Pro docker % docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS    PORTS     NAMES
09634fa86dfb   ubuntu    "bash"    23 seconds ago   Created             ubuntu_server
```

Docker Container Command

- Ubuntu 설치
  - ✓ Ubuntu 컨테이너 실행: docker start ubuntu_server

```
(base) adam@Adams-MacBook-Pro docker % docker start ubuntu_server
ubuntu_server
```

Docker Container Command

- Ubuntu 설치
  - ✓ Ubuntu 컨테이너 접속: docker attach ubuntu_server

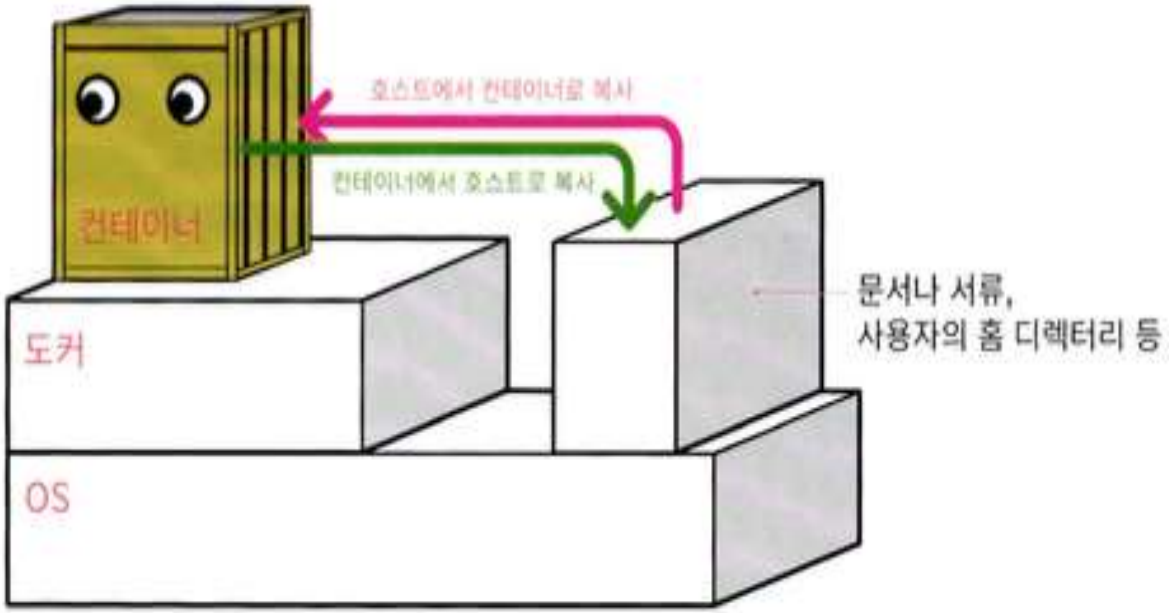
```
(base) adam@Adams-MacBook-Pro docker % docker attach ubuntu_server
root@09634fa86dfb:/#
```

Docker Container Command

- Ubuntu 설치
  - ✓ Ubuntu 명령 수행: cat etc/issue
  - ✓ Ubuntu 명령 수행: apt-get update(apt-get update --allow-unauthenticated)
  - ✓ Ubuntu 명령 수행: apt-get upgrade
  - ✓ Ubuntu 명령 수행: apt-get install vim

컨테이너 와 호스트 간의 공유

❖ 컨테이너 와 호스트 간에 파일 복사



컨테이너 와 호스트 간의 공유

- 컨테이너 와 호스트 간에 파일 복사
  - ✓ 명령어: docker cp source destination
    - docker cp 호스트경로 컨테이너이름:컨테이너경로
    - docker cp 컨테이너이름:컨테이너경로 호스트경로

컨테이너 와 호스트 간의 공유

- 컨테이너 와 호스트 간에 파일 복사
  - ✓ Apache 컨테이너 와 의 파일 복사

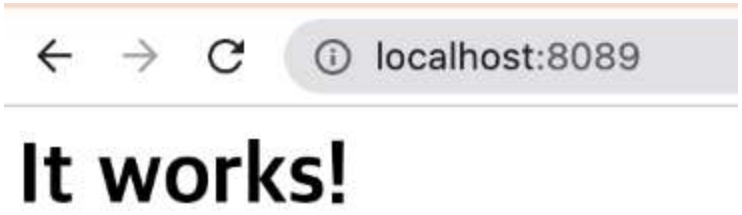
컨테이너이름: apa000ex19

Image 이름: httpd

포트 설정: 8089 -> 80

컨테이너 와 호스트 간의 공유

- 컨테이너 와 호스트 간에 파일 복사
  - ✓ Apache 컨테이너 와 의 파일 복사
    - 컨테이너 생성: docker run --name apa000ex19 -d -p 8089:80 httpd
    - 브라우저에서 localhost:8089 를 입력하고 확인



컨테이너 와 호스트 간의 공유

- 컨테이너 와 호스트 간에 파일 복사
  - ✓ Apache 컨테이너 와 의 파일 복사
    - 컨테이너에서 호스트로 파일을 복사: docker cp apa000ex19:/usr/local/apache2/htdocs/index.html index.html

컨테이너 와 호스트 간의 공유

- 컨테이너 와 호스트 간에 파일 복사
  - ✓ Apache 컨테이너 와 의 파일 복사
    - 현재 디렉토리에 index.html 파일을 생성하고 작성

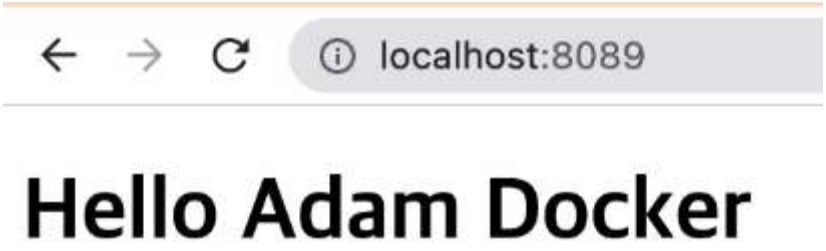
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Hello Adam Docker</h1>
</body>
</html>
```

컨테이너 와 호스트 간의 공유

- 컨테이너 와 호스트 간에 파일 복사
  - ✓ Apache 컨테이너 와 의 파일 복사
    - 현재 디렉토리에 index.html 파일을 생성하고 작성: docker cp index.html apa000ex19:/usr/local/apache2/htdocs/index.html

컨테이너 와 호스트 간의 공유

- ❖ 컨테이너 와 호스트 간에 파일 복사
  - ✓ Apache 컨테이너 와 의 파일 복사
    - ● 브라우저에서 localhost:8089 를 입력하고 확인



컨테이너 와 호스트 간의 공유

- Container로 애플리케이션 실행
  - ✓ 웹 서비스 실행을 위한 nginx 컨테이너 실행
    - 컨테이너 실행: docker run --name webserver1 -d -p 8001:80 nginx
    - 컨테이너의 리소스 사용량 실시간 확인: docker stats webserver1
    - 컨테이너의 실행 중인 프로세스 표시: docker top webserver1
    - 접근 로그 확인: docker logs -f webserver1
    - 컨테이너 중지: docker stop webserver1
    - 컨테이너 시작: docker start webserver1
    - 메인 화면 수정: index.html 파일을 생성하고 작성

      <h1> Hello Adam Docker </h1>
    - index.html 파일을 컨테이너에 복사: docker cp index.html webserver1:/usr/share/nginx/html/index.html

컨테이너 와 호스트 간의 공유

- ❖ Container로 애플리케이션 실행
  - ✓ 웹 서비스 실행을 위한 nginx 컨테이너 실행
    - ● 브라우저에서 확인

컨테이너 와 호스트 간의 공유

- Container로 애플리케이션 실행
  - ✓ 웹 서비스 실행을 위한 nginx 컨테이너 실행
    - 프로세스 일시 중지: docker pause webserver1
    - 컨테이너 확인: docker ps
    - 프로세스 일시 중단 해제: docker unpause webserver1
    - 컨테이너 확인: docker ps
    - 컨테이너 재시작: docker restart webserver1
    - 컨테이너 확인: docker ps
    - docker stop은 해당 컨테이너에서 실행되는 모든 프로세스를 KILL 하는 반면 docker pause는 컨테이너에서 실행되는 프로세스를 잠깐 중단(SIGSTOP) 시킴

컨테이너 와 호스트 간의 공유

- Container로 애플리케이션 실행
  - ✓ 파이썬 실행 환경

```
from random import shuffle
from time import sleep
gamenum = input('로또 게임 회수를 입력하세요: ')
for i in range(int(gamenum)):
    balls = [x+1 for x in range(45)]
    ret = []
    for j in range(6):
        shuffle(balls)          # balls를 무작위로 섞고,
        number = balls.pop()    # balls의 제일 마지막 숫자를 추출하고, 제거
        ret.append(number)      # 추출된 숫자를 ret에 추가
    ret.sort()
    print('로또번호[%d]: ' %(i+1), end='')
    print(ret)
    sleep(1)
```

```
로또 게임 회수를 입력하세요: 2
로또번호[1]: [5, 14, 26, 32, 44, 45]
로또번호[2]: [15, 21, 25, 26, 28, 35]
```

컨테이너 와 호스트 간의 공유

- Container로 애플리케이션 실행
  - ✓ 파이썬 실행 환경
    - 현재 디렉토리에 lotto.py 파일을 만들고 작성

```
from random import shuffle
from time import sleep
gamenum = input('로또 게임 회수를 입력하세요: ')
for i in range(int(gamenum)):
    balls = [x+1 for x in range(45)]
    ret = []
    for j in range(6):
        shuffle(balls)                                  # balls를 무작위로 섞고,
        number = balls.pop()                # balls의 제일 마지막 숫자를 추출하고, 제
      거
        ret.append(number)                  # 추출된 숫자를 ret에 추가
    ret.sort()
    print('로또번호[%d]: ' %(i+1), end='')
    print(ret)
    sleep(1)
```

컨테이너 와 호스트 간의 공유

- Container로 애플리케이션 실행
  - ✓ 파이썬 실행 환경
    - 파이선 컨테이너 실행: docker run -it -d --name=python_test -p 8900:8900 python
    - 파일 복사: docker cp lotto.py python_test:/
    - 컨테이너에 명령 수행: docker exec -it python_test bash
    - 파이썬 확인: python
    - 파이썬 종료: exit()
    - 파이썬 관련 도구 확인: pip list
    - 파이썬 모듈 확인: python -c 'help("modules")'
    - 외부로 나가기: exit
    - 외부에서 파이썬 컨테이너 코드 실행: docker exec -it python_test python /lotto.py

컨테이너 와 호스트 간의 공유

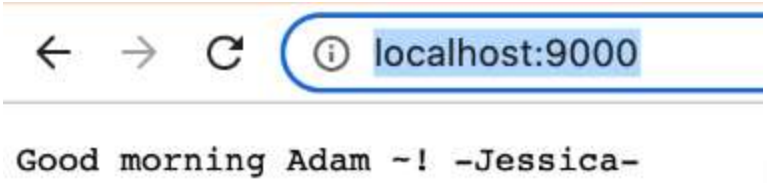
- Container로 애플리케이션 실행
  - ✓ node 실행 환경
    - 노드 실행 파일을 현재 디렉토리에 만들고 작성 – index.js

```
let http = require('http');
let content = function(req, resp){
 resp.end("Good morning Adam ~! -Jessica-" +"₩n");
 resp.writeHead(200);
}

let  web = http.createServer(content);
web.listen(8000);
```

컨테이너 와 호스트 간의 공유

- Container로 애플리케이션 실행
  - ✓ node 실행 환경
    - node 이미지 다운로드: docker pull node
    - node 컨테이너 실행: docker run -dit -p 9000:8000 --name=nodejs node
    - 컨테이너로 파일 복사: docker cp index.js nodejs:/index.js
    - 외부에서 실행: docker exec -it nodejs node /index.js
    - 브라우저에서 확인



컨테이너 와 호스트 간의 공유

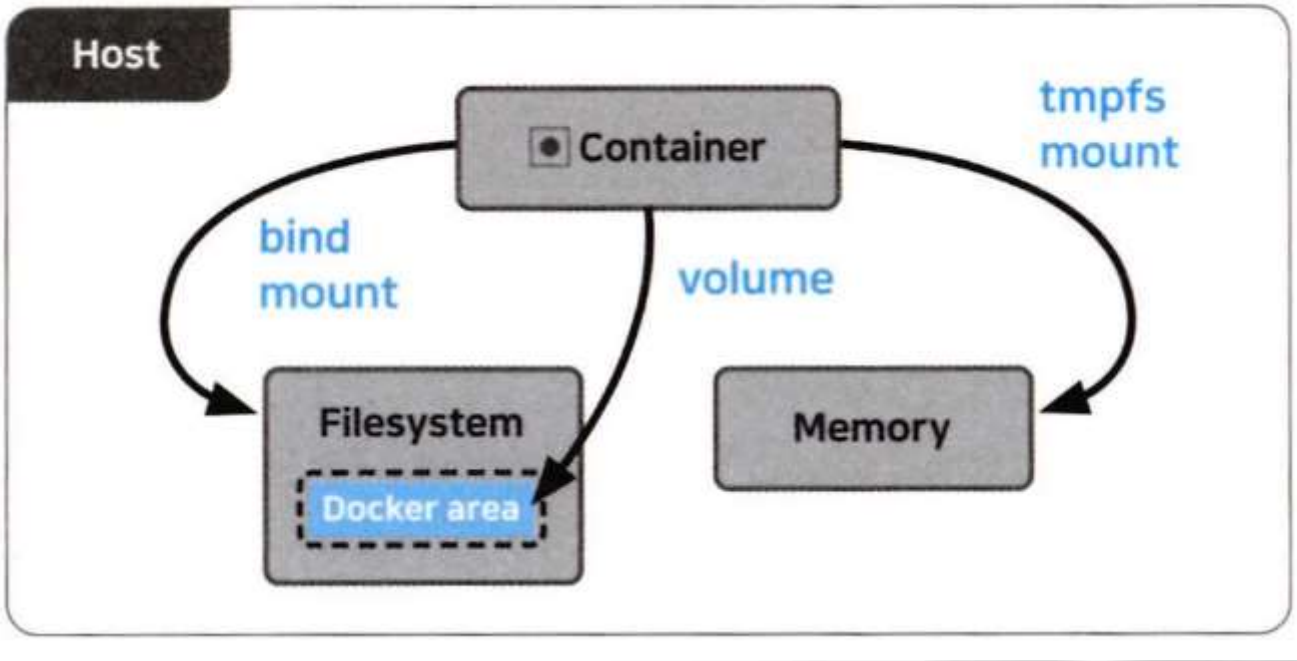
- ❖ Container를 이미지로 생성
  - ✓ docker commit
    - 컨테이너를 이미지로 생성: docker commit -a 'adam' nodejs webserver:1.0
    - 이미지 확인: docker images
    - Docker 로그인: docker login
    - 태그 생성: docker tag webserver:1.0 ggangpae1/webserver:1.0
    - Docker Hub에 이미지 업로드: docker push ggangpae1/webserver:1.0
    - 업로드 한 이미지 실행: docker run -dit --name webserver2 -p 8009:8000 ggangpae1/webserver:1.0
    - 명령 수행: docker exec webserver2 node /index.js

Docker 볼륨 활용

- 볼륨 마운트
  - ✓ 볼륨: 스토리지의 한 영역을 분할한 것
  - ✓ 마운트: 연결
  - ✓ Docker는 유니언 파일 시스템을 사용하는데 하나의 이미지로부터 여러 컨테이너를 만들 수 있는 방법을 제공하고 이미지에 변경된 내용을 저장할 수 있도록 해주는데 데이터베이스, 웹 프로그램 등 업무에서 사용하는 애플리케이션에서 발생하는 데이터에 접근하고 이것을 공유하기 위해서 Docker 볼륨 기능을 사용할 수 있으며 제공하는 서비스의 데이터와 로직은 반드시 분리되어야 하기 때문에 애플리케이션에서 발생하는 여러 가지 상황이 데이터에 영향을 주지 않고 언제든 다른 컨테이너로 이전할 수 있다면 운영자는 데이터를 안전하게 관리하고 운영하는 것이 가능
  - ✓ Docker 볼륨은 컨테이너에서 생성, 재사용할 수 있고 호스트 운영체제에서 직접 접근이 가능
  - ✓ 보존되어야 하는 데이터를 유지(데이터 영속성과 지속성)하기 위한 메커니즘을 제공
  - ✓ 일반적으로 컨테이너 내부의 데이터는 컨테이너의 라이프사이클과 연관되어 컨테이너 종료 시 삭제되는데 이를 영속적으로 유지하기 위한 방법으로 Docker 볼륨을 사용하면 컨테이너가 삭제되어도 볼륨은 독립적으로 운영되기 때문에 함께 삭제되지 않는 특징이 있음

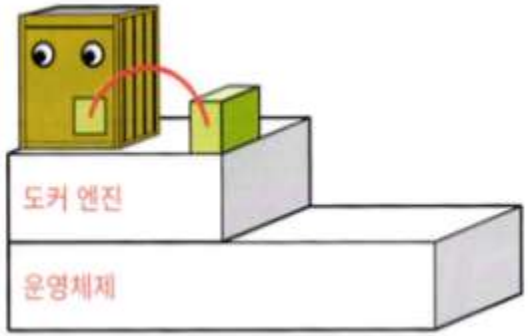
Docker 볼륨 활용

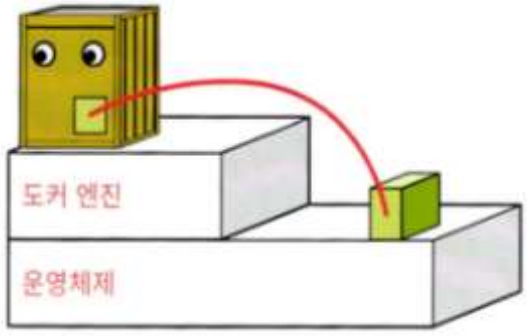
❖ 볼륨 마운트



Docker 볼륨 활용

- ❖ 볼륨 마운트
  - ✓ 마운트 방식
    - 볼륨 마운트: Docker 엔진이 관리하는 영역 내에 만들어진 볼륨을 컨테이너에 디스크 형태로 마운트 하는 방식
    - 바인드 마운트: Docker가 설치된 컴퓨터의 디렉토리 즉 Docker 엔진에서 관리하지 않는 영역의 기존 디렉토리를 컨테이너에 마운트하는 방식으로 디렉토리가 아닌 파일 단위로도 마운트가 가능





Docker 볼륨 활용

- 볼륨 마운트
  - ✓ 스토리지 마운트 방식

항목	볼륨 마운트	바인드 마운트
스토리지 영역	볼륨	디렉터리 또는 파일
물리적 위치	도커 엔진의 관리 영역	어디든지 가능
마운트 절차	볼륨을 생성한 후 마운트	기존 파일 또는 폴더를 마운트
내용 편집	도커 컨테이너를 통해서	일반적인 파일과 같이
백업	절차가 복잡함	일반적인 파일과 같이

Docker 볼륨 활용

- ❖ 볼륨 마운트
  - ✓ 볼륨 마운트 영역 생성 및 삭제
    - docker volume create 이름
    - docker volume rm 이름
  - ✓ 마운트 절차
    - 스토리지를 마운트하려면 먼저 마운트 될 스토리지를 생성해야 함
    - 볼륨 마운트의 경우는 마운트와 동시에 볼륨(스토리지 영역)을 만들 수도 있지만 이 방법은 권장하지 않음
  - ✓ 볼륨마운트 명령
    - docker run (생략) -v 볼륨이름:컨테이너_마운트_경로 (생략)

Docker 볼륨 활용

- ❖ 볼륨 마운트
  - ✓ 볼륨 마운트 실습
    - 볼륨 생성: docker volume create my-appvol-1
    - 볼륨 확인: docker volume ls

```
DRIVER    VOLUME NAME
local
    2b75f18ceba7323c65989cb609336b0c923fa91c40f32b82bd52d6019294d9ac
local
    8cde13fb6a7991d359ccd9f7cf38b1db4eb3edac9254d2520901c73e59e2b95a
local
    15e6df0a206260866d8167ac09adad1c1a68ba6395bad7d7a62181bff2e48bec
local
    39e92b43ce88d6efce89952ea1eef3396b846997478daea1f33d4b5ef436cba5
local     c25c99b101529ed35222dcd0cdef35b9db139f8f8733dd3cdf282fcf2a712376
local     my-appvol-1
```

Docker 볼륨 활용

- ❖ 볼륨 마운트
  - ✓ 볼륨 마운트 실습
    - 볼륨 검사: docker volume inspect my-appvol-1

```
[
    {
        "CreatedAt": "2023-02-20T07:06:59Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/my-appvol-1/_data",
        "Name": "my-appvol-1",
        "Options": {},
        "Scope": "local"
    }
]
```

Docker 볼륨 활용

- ❖ 볼륨 마운트
  - ✓ 볼륨 마운트 생성
    - 생성한 볼륨을 이용해서 ubuntu:20.04 이미지 실행(기본 볼륨에 연결: -v 옵션 이용)
      docker run -d --name vol-test1 -v my-appvol-1:/var/log ubuntu:20.04
    - 생성한 볼륨을 이용해서 ubuntu:20.04 이미지 실행(볼륨을 생성하면서 연결)

      docker run -d --name vol-test2 -v my-appvol-2:/var/log ubuntu:20.04

Docker 볼륨 활용

- ❖ 볼륨 마운트
  - ✓ 볼륨 마운트 생성
    - ● 볼륨 확인: docker volume ls

```
DRIVER    VOLUME NAME
local     my-appvol-1
local     my-appvol-2
```

Docker 볼륨 활용

- ❖ 볼륨 마운트
  - ✓ 볼륨 마운트 생성
    - 볼륨 마운트 확인: docker inspect --format="{{.Mountpoint}}" my-appvol-1

      /var/lib/docker/volumes/my-appvol-1/_data
    - 볼륨 제거(연결된 이미지가 존재해서 에러 발생): docker volume rm my-appvol-1

      Error response from daemon: remove my-appvol-1: volume is in use - [359dacf0b52800913b634dba985816c1f9e94558aa0d9bb46baeeff9f25dbe87]
    - 컨테이너 중지: docker stop vol-test1 vol-test2
    - 컨테이너 삭제: docker rm vol-test1 vol-test2

      vol-test1

      vol-test2
    - 볼륨 삭제: docker volume rm my-appvol-1 my-appvol-2

      my-appvol-1

      my-appvol-2

Docker 볼륨 활용

- ❖ 볼륨 마운트
  - ✓ 볼륨 마운트 생성
    - 볼륨 확인: docker volume ls

      DRIVER VOLUME NAME

      (base) adam@choongangui-MacBookPro ~ %

Docker 볼륨 활용

- 바인드 마운트
  - ✓ 특징
    - Docker 볼륨 기법에 비해 사용이 제한적
    - 호스트 파일 시스템 절대경로: 컨테이너 내부 경로를 직접 마운트하여 사용
    - 사용자가 파일 또는 디렉토리를 생성하면 해당 호스트 파일 시스템의 소유자 권한으로 연결이 되고 존재하지 않는 경우 자동 생성
    - 자동 생성된 디렉토리는 루트 사용자 소유
    - 컨테이너 실행 시 지정하여 사용하고 컨테이너 제거 시 바인드 마운트는 해제되지만 호스트 디렉토리는 유지
  - ✓ 바인드 마운트 명령
    - docker run (생략) -v 스토리지_실제_경로:컨테이너_마운트_경로 (생략)

Docker 볼륨 활용

❖ 바인드 마운트

✓ 바인드 마운트 실습

- 생성할 컨테이너 정보
  - 컨테이너 이름: apa00ex20
  - Image 이름: httpd
  - 포트 번호: 8090
- 옵션
  - 컨테이너 마운트 경로: /usr/local/apache2/htdocs
  - 실제 디렉토리: apa_folder
  - 마운트 원본 경로: /Users/adam/Documents/apa_folder

Docker 볼륨 활용

- ❖ 바인드 마운트
  - ✓ 바인드 마운트 실습
    - 디스크에 apa_folder라는 이름으로 마운트 원본이 될 디렉토리 생성
    - docker run --name apa000ex20 -d -p 8090:80 -v /Users/adam/Documents/apa_folder:/usr/local/apache2/htdocs httpd
    - 웹 브라우저에서 확인: http://localhost:8090
      - ◆ Index of / 가 출력됨
    - apa_folder 디렉토리에 index.html 파일을 배치 한 후 웹 브라우저에서 확인
    - 컨테이너 중지: docker stop apa000ex20
    - 컨테이너 삭제: docker rm apa000ex20

Docker 볼륨 활용

- ❖ 바인드 마운트
  - ✓ 바인드 마운트 실습
    - 디렉토리를 생성하고 마운트: docker run -dit --name bind-test1 --mount type=bind,source=/Users/adam/Documents/target1,target=/var/log centos:8
    - 디렉토리를 생성하지 않고 마운트: docker run -dit --name bind-test2 -v /Users/adam/Documents/target2:/var/log centos:8
    - 디렉토리를 생성하지 않고 디렉토리에 읽고 쓰기 바인드 마운트: docker run -dit --name bind-test3 -v /Users/adam/Documents/target_ro:/app1:ro -v /Users/adam/Documents/target_wo:/app2:rw centos:8

Docker 볼륨 활용

- tmpfs mount
  - ✓ 개요
    - 일시적으로 마운트하는 것으로 호스트 메모리에만 지속되는 데이터를 활용하고자 하는 경우 사용
    - 컨테이너 실행 시 지정하여 사용하고 컨테이너 제거 시 자동 해제
    - 마운트: docker run -dit --name tempfs-test --tmpfs /var/www/html httpd:2

Docker 볼륨 활용

- MySQL 영속성 유지
  - ✓ 볼륨 생성: docker volume create mysql-data-vol
  - ✓ 볼륨 확인: docker volume ls
  - ✓ 볼륨과 연결해서 MySQL 5.7 실행: docker run -it --name=mysql-vtest -e MYSQL_ROOT_PASSWORD=wnddkd -e MYSQL_DATABASE=dockertest -v mysql-data-vol:/var/lib/mysql -d mysql
  - ✓ Mac에서 실행이 안되면 옵션 추가: --platform linux/amd64
  - ✓ MySQL 컨테이너 접속: docker exec -it mysql-vtest bash
  - ✓ 루트 접속: mysql -uroot -pwnddkd dockertest
  - ✓ 데이터베이스 작업: create table mytable(c1 int, c2 char(100));
  - ✓ 데이터베이스 작업: insert into mytable values(1, 'adam');
  - ✓ 데이터베이스 작업: exit
  - ✓ 데이터 확인: bash-4.2# ls /var/lib/mysql/dockertest/

    db.opt mytable.frm mytable.ibd
  - ✓ 콘솔 종료: exit

Docker 볼륨 활용

- MySQL 영속성 유지
  - ✓ 컨테이너 삭제 후 이전 과 동일한 방법으로 컨테이너를 생성하고 테이블이 계속 유지되는지 확인

Docker 볼륨 활용

- 웹 서비스의 로그 정보 보호 및 분석을 위한 바인드 마운트 설정
  - ✓ 로그 정보를 저장할 nginx-log 디렉토리 생성
  - ✓ 웹 서버를 디렉토리와 바인드 마운트해서 실행: docker run -d -p 8011:80 -v /Users/adam/Documents/nginx-log:/var/log/nginx nginx:1.19
  - ✓ 브라우저에서 접속: http://localhost:8011

Docker 볼륨 활용

- 컨테이너 간 데이터 공유를 위한 데이터 컨테이너 생성
  - ✓ 데이터 공유를 위한 컨테이너 생성: docker create -v /data-volume --name=datavol ubuntu:20.04
  - ✓ 공유 연결할 컨테이너 생성: docker run -it --volumes-from datavol ubuntu:20.04
  - ✓ 컨테이너에 텍스트 파일 생성: echo 'testing data container' > /data-volume/test-volume.txt
  - ✓ 컨테이너에 텍스트 파일 확인: cat /data-volume/test-volume.txt

    testing data container
  - ✓ 공유 연결할 컨테이너 생성: docker run -it --volumes-from datavol ubuntu:20.04
  - ✓ 컨테이너에 텍스트 파일 생성: echo 'testing data container2' > /data-volume/test-volume2.txt
  - ✓ 컨테이너에 텍스트 파일 확인: cat /data-volume/test-volume2.txt

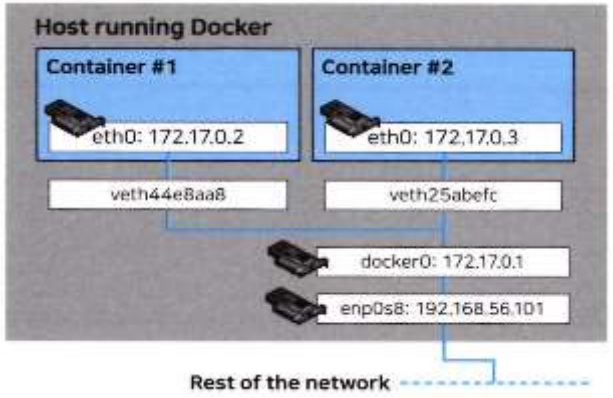
    testing data container2

Docker 볼륨 활용

- 파일 시스템 마운트의 한계
  - ✓ 컨테이너의 마운트 대상 디렉터리가 이미 존재하고 이미지 레이어에 이 디렉터리의 파일이 포함되어 있는 경우 이미 존재하는 대상 디렉터리에 마운트하면 마운트의 원본 디렉터리가 기존 디렉터리를 완전히 대체해서 이미지에 포함돼 있던 원래 파일은 사용할 수 없음
  - ✓ 호스트 컴퓨터의 파일 하나를 컨테이너에 이미 존재하는 디렉토리로 마운트 하는 경우 디렉토리의 파일이 합쳐져 이미지에서 온 파일과 호스트에서 마운트 된 파일이 모두 나타나게 되는데 호스트 컴퓨터가 윈도우인 경우는 이 기능을 제공하지 않음

여러 개의 컨테이너 연동

- Docker Network
  - ✓ 개요
    - Docker 컨테이너 및 서비스는 Docker 네트워크를 통해 격리된 컨테이너 간의 네트워크 연결 뿐 만 아니라 Docker 외의 다른 애플리케이션 워크로드 와도 연결이 가능
    - Docker 네트워크의 하위 시스템 연결을 위해 Docker 네트워크 드라이버를 사용하여 상호 간 통신이 가능
    - Docker 레퍼런스(https://docs.docker.com/network/)의 Docker 네트워크 정의: Docker 설치 시 기본적으로 제공되는 docker0는 소프트웨어적으로 구현된 가상 이더넷 브리지(virtual ethernet bridge) 네트워크이고 이것을 통해 격리된 컨테이너들의 상호 간 통신을 제공
    - 별도의 브리지 네트워크를 생성하여 연결 값으로 설정하지 않는 한 실행되는 모든 컨테이너는 docker0 브리지에 연결되어 172.17.0.0/16의 CIDR(Classless Inter-Domain Routing 은 클래스 없는 도메인 간 라우팅 기법) 범위로 IP 주소가 할당되는데 /16은 최대 65,536개의 IP 주소 범위를 소유



여러 개의 컨테이너 연동

- ❖ Docker Network
  - ✓ 개요
    - ● 네트워크 인터페이스
      - ◆ enp0s8: Ubuntu 리눅스의 네트워크 카드로 오라클 VirtualBox에서 제공하는 IP 주소를 할당받아 사용 중(centos에서는 ens33 등의 이름을 사용)
      - ◆ docker0: Docker 설치 시 기본적으로 제공되는 브리지 네트워크로 172.17.0.1 주소를 갖고 docker0 브리지는 소프트웨어적인 스위치 방식으로 동작하며 일반적인 스위치 방식과는 다르게 DHCP로 연결된 컨테이너에 사전에 정의된 IP 풀을 할당
      - ◆ vethxxxxxxx: OSI 7 계층 서비스 모델의 2계층 서비스로 컨테이너 내부에 제공되는 네트워크 인터페이스 eth0와 한 쌍으로 제공되어 docker0 와 가상의 터널링 네트워크를 제공
      - ◆ eth0: Docker 컨테이너에 생성되는 기본 네트워크 인터페이스명으로 docker0를 게이트웨이로 사용하는데순차적으로 IP 주소를 할당받거나 사용자가 동일 대역의 IP 주소를 지정할 수 있음

여러 개의 컨테이너 연동

- Docker Network
  - ✓ 기본 브리지 네트워크 모드
    - Docker 네트워크 드라이버 방식 조회: docker network ls

      NETWORK ID NAME DRIVER SCOPE

      206c22e2ffb3 bridge bridge local

      b8fd2f9cbc57 host host local

      fe4dc31ab13d none null local
    - 전체 네트워크 조회: docker network inspect bridge
    - ubuntu 이미지 실행: docker run -dit --name container1 ubuntu:20.04
    - ubuntu 이미지 실행: docker run -dit --name container2 ubuntu:20.04

여러 개의 컨테이너 연동

- Docker Network
  - ✓ 기본 브리지 네트워크 모드
    - 네트워크 확인: docker inspect container1 | grep IPAddress

      "SecondaryIPAddresses": null,

      "IPAddress": "172.17.0.3",

      "IPAddress": "172.17.0.3",
    - 네트워크 확인: docker inspect container2 | grep IPAddress

      "SecondaryIPAddresses": null,

      "IPAddress": "172.17.0.4",

      "IPAddress": "172.17.0.4",

여러 개의 컨테이너 연동

- Docker Network
  - ✓ 웹 서비스를 제공하는 nginx 컨테이너가 브리지 모드의 네트워크 와 연결되는 과정
    - nginx 컨테이너 포트 80을 호스트 포트 8080에 연결하여 실행: docker run -d --name=nginx-net -p 8080:80 nginx:1.19
    - 브라우저에서 확인: localhost:8080

여러 개의 컨테이너 연동

- Docker Network
  - ✓ 웹 서비스를 제공하는 nginx 컨테이너가 브리지 모드의 네트워크 와 연결되는 과정
    - 접근 과정
      - 윈도우 웹 브라우저에 http://호스트IP:연결된포트 입력을 통해 접속
      - 호스트 운영체제에 8080 포트가 열려 있음을 확인하고 해당 포트가 연결된 컨테이너를 찾음
      - 해당 컨테이너가 연결된 브리지 네트워크의 프라이빗 IP(172.17.0.2)와 포트 번호로 사용자가 입력 한 외부 IP와 포트 번호가 변환되는데 이때 사용되는 서비스가 네트워크 주소 포트 변환이고 NAT에서는 발신자의 사설망 to 외부망 IP를 변환해 주는 역할만 수행했다면 여기에 포트까지 바꿔서 보내는 역할을 수행

여러 개의 컨테이너 연동

- Docker Network
  - ✓ 네트워크 관련 커맨드

커맨드	내용	생략 가능 여부	주요 옵션
connect	네트워크에 컨테이너를 새로이 접속	X	거의 사용하지 않음
disconnect	네트워크에서 컨테이너의 접속을 끊음	X	거의 사용하지 않음
create	네트워크를 생성	X	거의 사용하지 않음
inspect	네트워크의 상세 정보를 확인	X	거의 사용하지 않음
ls	네트워크의 목록을 확인	X	거의 사용하지 않음
prune	현재 아무 컨테이너도 접속하지 않은 네트워크를 모두 삭제	X	거의 사용하지 않음
rm	지정한 네트워크를 삭제	X	거의 사용하지 않음

여러 개의 컨테이너 연동

- MySQL 컨테이너 실행 시에 필요한 옵션 과 인자
  - ✓ 컨테이너 생성 및 실행: docker run --name 컨테이너이름 –dit --net=네트워크이름 -e MYSQL_ROOT_PASSWORD=root비밀번호 –e MYSQL_DATABASE=데이터베이스_이름 –e MYSQL_USER=MYSQL_사용자이름 –e MYSQL_PASSWORD=MySQL_패스워드 -p 소호스트 포드번호:컨테이너포트번호 mysqlImage이름 --charaterset-set-server=문자_인코딩 --collation-server=정렬순서
  - ✓ 사용된 옵션
    - --net은 컨테이너를 연결할 Docker 네트워크 나머지는 모두 -e 옵션으로 환경변수를 설정하기 위해 사용
    - 환경 변수란 운영체제에서 다양한 설정 값을 저장하는 장소를 가리키는 것으로 컨테이너의 설정 값을 환경 변수를 통해 전달하는 경우가 많은데 어떤 환경 변수를 사용할지는 컨테이너의 종류에 따라 달라짐
    - 패스워드는 루트 패스워드와 일반 사용자 패스워드 두 가지를 설정
    - 루트는 모든 권한을 가진 사용자로 이 권한이 없으면 할 수 없는 작업이 있지만 매번 루트 사용자로 접속할 경우 보안 측면에서 문제가 생기기 때문에 제한된 권한을 가진 일반 사용자로 전환하는 것이 일반적

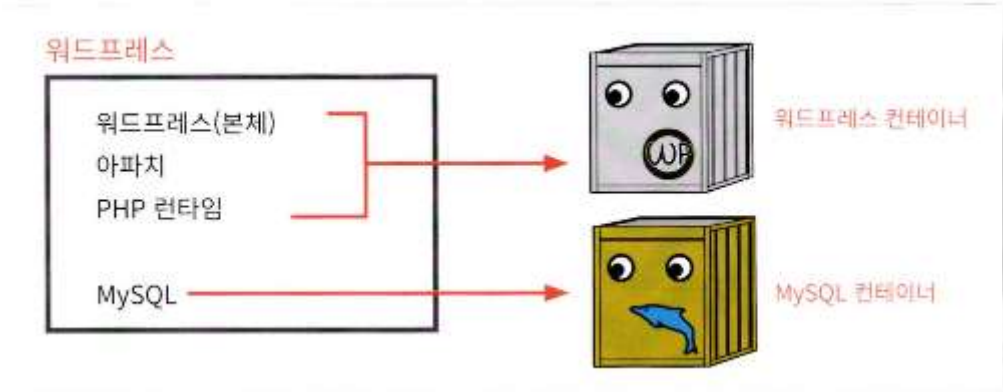
여러 개의 컨테이너 연동

- MySQL 컨테이너 실행 시에 필요한 옵션 과 인자
  - ✓ 사용된 옵션
    - 인자는 한국어를 사용하기 위한 것이 두 가지 와 인증 방식을 변경하는 것이 한 가지 사용되었는데 이들 값은 Docker에서 사용되는 옵션이 아니라 MySQL 컨테이너에서만 사용되는 값

항목	인자	값	의미
문자 인코딩	--character-set-server=	utf8mb4	문자 인코딩으로 UTF8을 사용
정렬 순서	--collation-server=	utf8mb4_unicode_ci	정렬 순서로 UTF8을 따름

여러 개의 컨테이너 연동

- ❖ 워드프레스 구축
  - ✓ 워드프레스
    - 웹 사이트를 만들기 위한 소프트웨어로 서버에 설치해 사용
    - 워드프레스는 워드프레스 프로그램 외에도 Apache나 데이터베이스, PHP 런타임 등을 필요로 함
    - 워드프레스는 데이터베이스로 MySQL 및 MariaDB를 지원
    - 컨테이너는 워드프레스 공식 Image를 사용
    - 이 Image는 워드프레스 프로그램 본체와 Apache, PHP 런타임을 함께 포함하고 있어서 매우 편리한데 이 컨테이너 와 MySQL 컨테이너가 있으면 워드프레스를 사용할 수 있음
    - 데이터베이스 서버를 꼭 컨테이너가 아니더라도 Docker 외부에 두는 방식도 가능



여러 개의 컨테이너 연동

- ❖ 워드프레스 구축
  - ✓ Docker 네트워크 생성/삭제
    - 워드프레스는 워드프레스 컨테이너와 MySQL 컨테이너로 구성
    - 워드프레스는 블로그 생성 도구와 같은 것으로 웹 사이트 작성자가 작성한 내용을 데이터베이스에 저장하고 웹 사이트 열람자의 요청에 따라 웹 페이지를 보여주는데 프로그램이 MySQL에 저장된 데이터를 읽고 쓸 수 있어야 하기 때문에 두 컨테이너가 연결돼 있어야 함
    - 컨테이너를 두 개 만들기만 해서는 두 컨테이너가 연결되지 않으므로 가상 네트워크를 만들고 이 네트워크에 두 개의 컨테이너를 소속시켜 두 컨테이너를 연결
    - 가상 네트워크를 만드는 커맨드가 docker network create로 커맨드 뒤로 네트워크 이름을 기재하면 됨
    - container 또는 image 상위 커맨드와 마찬가지로 네트워크를 삭제할 때는 docker network rm, 네트워크 목록을 출력할 때는 docker network ls 커맨드를 사용

여러 개의 컨테이너 연동

- 워드프레스 구축
  - ✓ 워드프레스 컨테이너 실행 시 필요한 옵션 과 인자
    - docker run --name 컨테이너_이름 -dit –-net=네트워크_이름 -p 포트_설정 -e W0RDPRESS_DB_HOST=데이터베이스_컨테이너_이름 -e WORDPRESS_DB_NAME=데이터베이스_이름 -e WORDPRESS_DB_USER=데이터베이스_사용자_이름 -e WORDPRESS_DB_PASSWORD=데이터베이스_패스워드 wordpress
    - 사용된 옵션
      - -e 옵션은 MySQL 접속과 관련된 옵션

여러 개의 컨테이너 연동

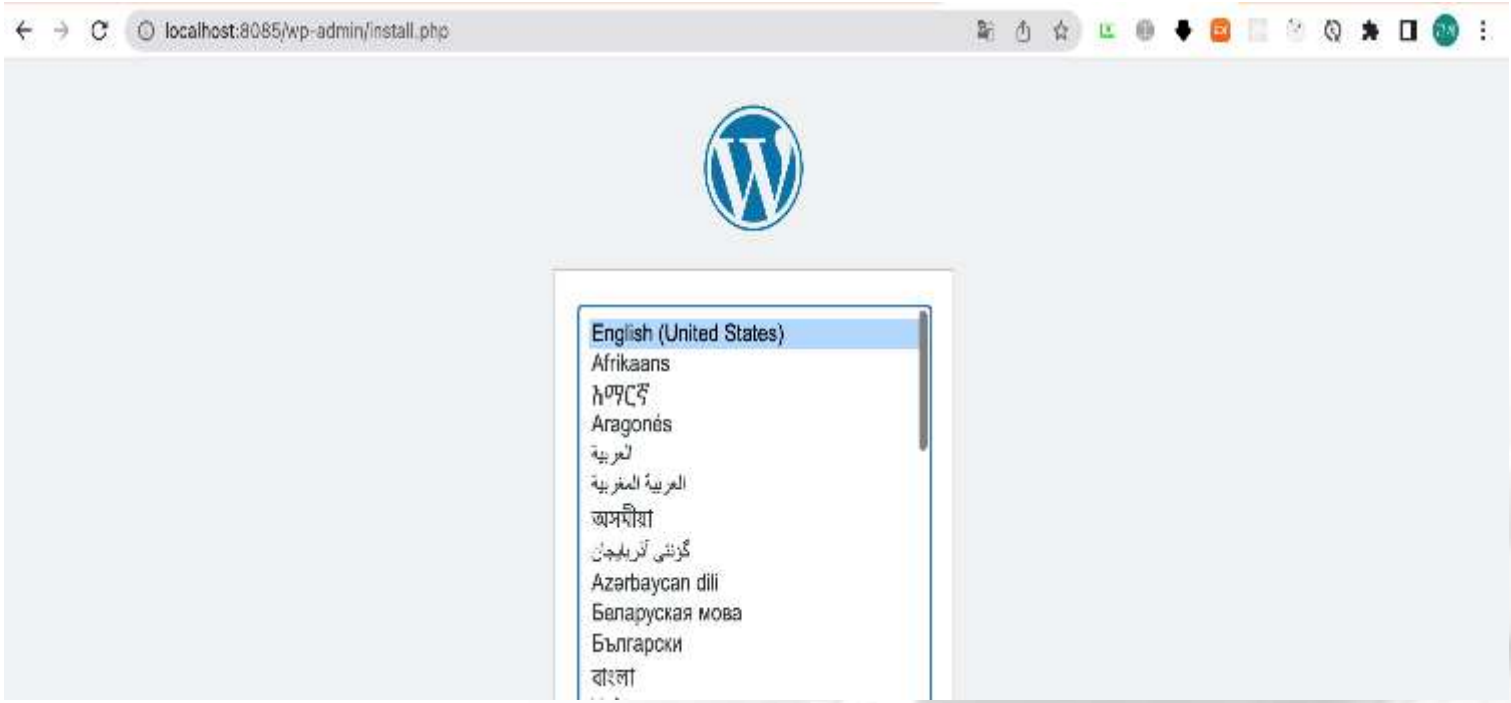
- 워드프레스 및 MySQL 컨테이너 생성 과 연동
  - ✓ 실습

여러 개의 컨테이너 연동

- ❖ 워드프레스 및 MySQL 컨테이너 생성 과 연동
  - ✓ 실습
    - network create 커맨드로 네트워크 생성: docker network create wordpress000net1
    - MySQL 컨테이너를 생성 및 실행: docker run --name mysql000ex11 -dit --net=wordpress000net1 -e MYSQL_ROOT_PASSWORD=myrootpass -e MYSQL_DATABASE=wordpress000db -e MYSQL_USER=wordpress000kun -e MYSQL_PASSWORD=wkunpass mysql --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci
    - WordPress 컨테이너를 생성 및 실행: docker run --name wordpress000ex12 -dit --net=wordpress000net1 -p 8085:80 -e WORDPRESS_DB_HOST=mysql000ex11 -e WORDPRESS_DB_NAME=wordpress000db -e WORDPRESS_DB_USER=wordpress000kun -e WORDPRESS_DB_PASSWORD=wkunpass wordpress
    - 컨테이너가 실행 중인지 확인: docker ps

여러 개의 컨테이너 연동

- ❖ 워드프레스 및 MySQL 컨테이너 생성 과 연동
  - ✓ 실습
    - ● 웹 브라우저에서 확인 – http://localhost:8085



여러 개의 컨테이너 연동

- ❖ 워드프레스 및 MySQL 컨테이너 생성 과 연동
  - ✓ 실습
    - ● 뒷정리
      - ◆ docker stop wordpress000ex12
      - ◆ docker stop mysql000ex11
      - ◆ docker rm wordpress000ex12
      - ◆ docker rm mysql000ex11
      - ◆ docker network rm wordpress000net1

여러 개의 컨테이너 연동

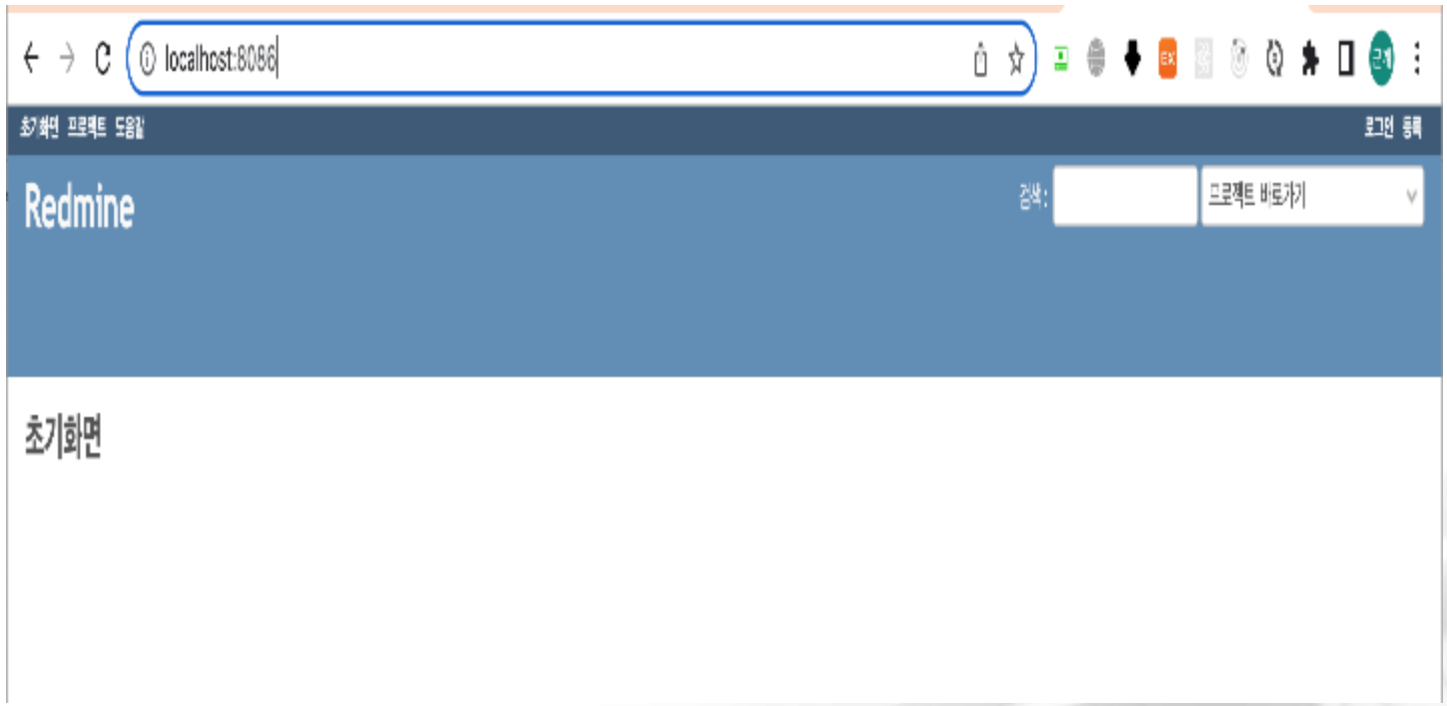
- Redmine과 MySQL 컨테이너 연결
  - ✓ Redmine
    - Redmine은 오픈 소스 프로그램으로 웹 기반의 프로젝트 관리와 버그 추적 기능을 제공하는 도구
    - 화면 기반의 프로젝트 관리에 도움이 되도록 달력과 간트 차트를 제공하고 일정 관리 기능을 제공

여러 개의 컨테이너 연동

- Redmine과 MySQL 컨테이너 연결
  - ✓ 네트워크 생성: docker network create redmine000net2
  - ✓ MySQL 컨테이너 생성 및 실행: docker run --name mysql000ex13 -dit --net=redmine000net2 -e MYSQL_ROOT_PASSWORD=myrootpass -e MYSQL_DATABASE=redmine000db -e MYSQL_USER=redmine000kun -e MYSQL_PASSWORD=rkunpass mysql --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci --default-authentication-plugin=mysql_native_password
  - ✓ Redmine 컨테이너 생성 및 실행: docker run -dit --name redmine000ex14 --network redmine000net2 -p 8086:3000 -e REDMINE_DB_MYSQL=mysql000ex13 -e REDMINE_DB_DATABASE=redmine000db -e REDMINE_DB_USERNAME=redmine000kun -e REDMINE_DB_PASSWORD=rkunpass redmine

여러 개의 컨테이너 연동

- Redmine과 MySQL 컨테이너 연결
  - ✓ 확인



여러 개의 컨테이너 연동

- Wordpress 와 MariaDB 컨테이너 실행하기
  - ✓ 네트워크 생성: docker network create wordpress000net4
  - ✓ MariaDB 컨테이너 생성 및 실행: docker run --name mariadb000ex17 -dit --net=wordpress000net4 -e MYSQL_ROOT_PASSWORD=mariarootpass -e MYSQL_DATABASE=wordpress000db -e MYSQL_USER=wordpress000kun -e MYSQL_PASSWORD=wkunpass mariadb --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci --default-authentication-plugin=mysql_native_password
  - ✓ WordPress 컨테이너 생성 및 실행: docker run --name wordpress000ex18 -dit --net=wordpress000net4 -p 8088:80 -e WORDPRESS_DB_HOST=mariadb000ex17 -e WORDPRESS_DB_NAME=wordpress000db -e WORDPRESS_DB_USER=wordpress000kun -e WORDPRESS_DB_PASSWORD=wkunpass wordpress

여러 개의 컨테이너 연동

- Maria DB
  - ✓ Daemon이 아닌 형태로 컨테이너 생성: docker run --name mariadb -d -p 외부에서접속할 포트번호:MariaDB포트번호 -e MYSQL_ROOT_PASSWORD=비밀번호 Image이름
  - ✓ 외부 접속 허용
    - MariaDB가 설치된 Docker 컨테이너로 접속

      $ docker exec -it [mariadb컨테이너이름] bash
    - vim 설치되어 있지 않으면 설치

      # apt update

      # apt upgrade

      # apt install vim
    - MariaDB 외부 접근 허용을 위한 설정 파일을 수정

      # vim /etc/mysql/mariadb.conf.d/50-server.cnf

      bind-address 의 값을 0.0.0.0 으로 수정