Docker

Docker

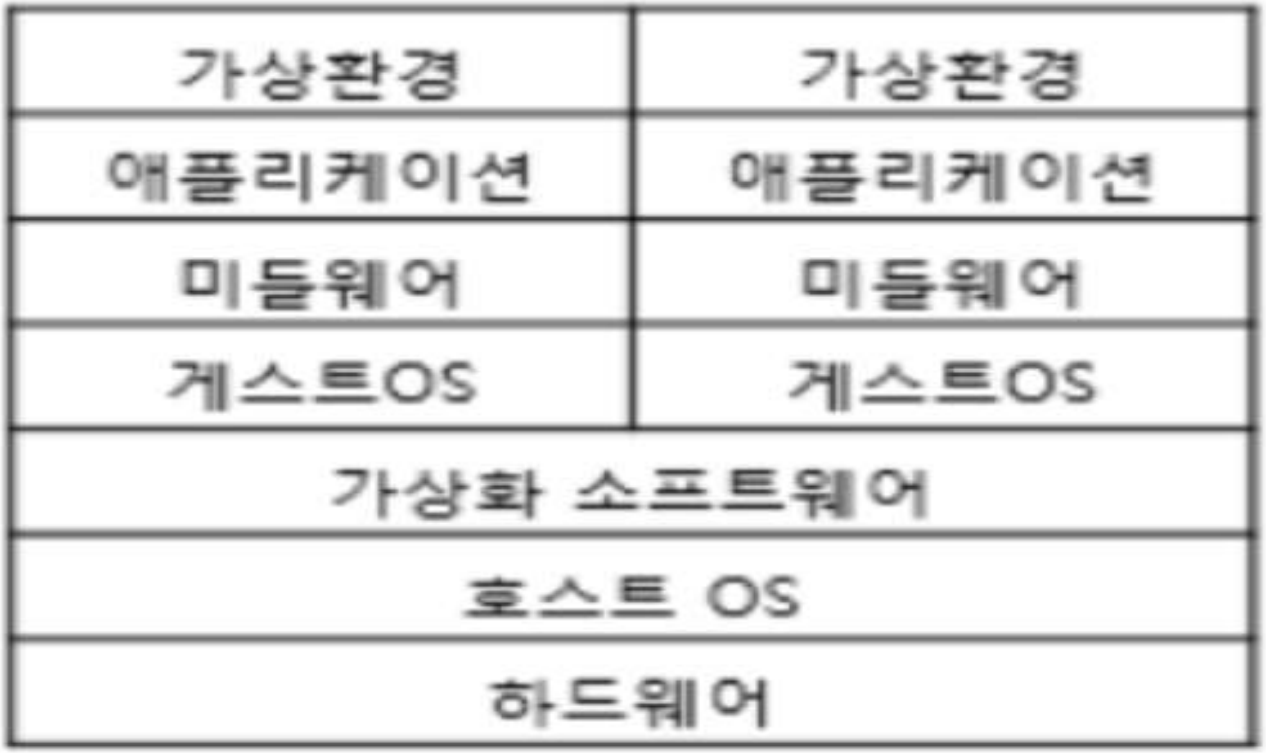
- Container 와 가상화
  - ✓ 가상화
    - 가상화는 서버, 스토리지, 네트워크 및 기타 물리적 시스템에 대한 가상 표현을 생성하는 데 사용할 수 있는(추상화) 기술
    - 가상화 소프트웨어는 물리적 하드웨어 기능을 모방하여 하나의 물리적인 컴퓨터에서 여러 가상 시스템을 동시에 실행
    - 기업은 가상화를 사용해 하드웨어 리소스를 효율적으로 사용하여 투자 대비 이익을 더 많이 얻을 수 있고 클라우드 컴퓨팅 서비스를 지원하여 조직의 인프라를 더욱 효율적으로 관리할 수 있음

Docker

- Container 와 가상화
  - ✓ 가상화 이점
    - 효율적인 리소스 사용
    - 자동화된 IT 관리
    - 신속한 재해 복구
  - ✓ 가상화 서비스
    - 서버 가상화
    - 스토리지 가상화
    - 네트워크 가상화
      - SDN(Software-Defined Networking)
        - 물리적 환경의 데이터 라우팅에서 라우팅 관리를 인수하여 트래픽 라우팅을 제어
        - 애플리케이션 트래픽보다 영상 통화 트래픽을 우선적으로 처리하도록 시스템을 프로그래밍해서 모든 온라인 회의에서 일관된 통화 품질을 보장하도록 할 수 있음
      - 네트워크 기능 가상화
    - 데이터 가상화
    - 애플리케이션 가상화
    - 데스크톱 가상화

Docker

- ❖ Container 와 가상화
  - ✓ 가상화 방식
    - ● 호스트 운영체제 가상화



Docker

- ❖ Container 와 가상화
  - ✓ 가상화 방식
    - ● 호스트 운영체제 가상화
      - ◆ 물리적 하드웨어 위에 설치된 호스트 운영체제 위에 가상화 소프트웨어 와 가상 머신에 설치된 운영체제(게스트 운영체제)를 움직이는 방식
      - ◆ 가상의 하드웨어를 Emulating 하기 때문에 호스트 운영체제에 크게 제약 사항이 없음
      - ◆ 운영체제 위에 운영체제가 얹히는 방식이기 때문에 오버헤드가 클 수 있음
      - ◆ 호스트 운영체제 가상화 소프트웨어
        - VMware Workstation
        - Oracle의 VirtualBox
        - Microsoft의 Virtual PC

Docker

- ❖ Container 와 가상화
  - ✓ 가상화 방식
    - ● 하이퍼바이저 가상화
      - ◆ 하이퍼바이저 가상화는 호스트 운영체제 없이 하드웨어에 하이퍼바이저를 설치하여 사용하는 가상화 방식
      - ◆ 하이퍼바이저는 한 컴퓨터에서 여러 가상 머신을 관리하는 소프트웨어 구성 요소로 각 가상 머신이 할당된 리소스를 얻고 다른 가상 머신의 작동을 방해하지 않도록 함
      - ◆ 하이퍼바이저 가상화는 호스트 운영체제 와 별도로 개별 시스템처럼 행동하기 때문에 처리 오버헤드가 존재하지 않음
      - ◆ 별도의 호스트 운영체제가 없기 때문에 오버헤드가 적고 하드웨어를 직접 제어하기 때문에 효율적으로 리소스를 사용할 수 있음
      - ◆ 자체적으로 머신에 대한 관리 기능이 없기 때문에 관리를 위한 컴퓨터나 콘솔(CLI)이 필요함
      - ◆ 서버 가상화 기술에서는 주류 방식으로 사용
      - ◆ 하드웨어와 운영체제 사이에서 물리적 시스템의 하드웨어에 직접 설치되는 경우를 베어메탈 하이퍼바이저라고 함

Docker

- ❖ Container 와 가상화
  - ✓ 가상화 방식
    - ● 하이퍼바이저 가상화
      - ◆ 구현 방식
        - ▪ 전가상화(Full-Virtualization)
          - 하드웨어를 완전히 가상화 하는 방식으로 Hardware Virtual Machine 이라고도 함
          - 게스트 운영체제(가상화 된 운영체제)가 하드웨어에 접근하면서 제어를 요구하는데 이 때 CPU는 가상화가 지원 가능한지 아닌지를 확인하는데 지원을 하지 않으면 게스트 운영체제는 운영할 수 없으며 지원 가능하면 하드웨어 제어를 요구하고 이후 하드웨어를 제어할 수 있게 됨
          - 하이퍼바이저를 구동하면 DOM0 이라는 관리용 가상 머신이 구동되는데 모든 가상 머신의 하드웨어 접근이 이 DOM0로 이루어 짐
          - 하드웨어를 완전히 가상화하기 때문에 게스트 운영체제의 별다른 수정이 필요 없음
          - 하이퍼바이저가 모든 명령을 중재하기 때문에 성능이 비교적 느림(하이퍼바이저는 중재이외의 다른 문제들도 처리해야함)
          - VMWare ESX Server, MS의 Hyper V 가 대표적인 전가상화 소프트웨어

Docker

- ❖ Container 와 가상화
  - ✓ 가상화 방식
    - ● 하이퍼바이저 가상화
      - ◆ 구현 방식
        - ▪ 전가상화(Full-Virtualization)

Docker

- ❖ Container 와 가상화
  - ✓ 가상화 방식
    - ● 하이퍼바이저 가상화
      - ◆ 구현 방식
        - ▪ 반가상화(Para-Virtualization)
          - 전가상화의 가장 큰 단점인 성능 저하 문제를 해결하기 위해 하이퍼콜(Hyper Call)이라는 인터페이스를 통해 하이퍼바이저에게 직접 요청을 보냄
          - 반가상화는 게스트 운영체제들에게 자원을 어떻게 분배할지 같은 관리적 문제만 다루는 방식으로 이렇게 하면 기존에 하이퍼바이저가 하던 번역 일은 게스트 운영체제 자체에서 수행하고 하드웨어로 전송하는데 게스트 운영체제의 커널을 수정하여 이러한 역할이 가능하도록 만드는 것이 반가상화와 전가상화의 차이
          - 하이퍼바이저에게 Hyper Call 요청을 할 수 있도록 각 운영체제의 커널을 수정해야 하므로 오픈 소스 운영체제(Linux)가 아니면 반가상화를 이용하기가 쉽지 않았는데 윈도우에서 Xen에서 제공하는 툴로 가능해 짐
          - 모든 명령을 D0M0를 통해 하이퍼바이저에게 요청하는 전가상화에 비해 성능이 빠름
          - Xen, KVM이 대표적인 반가상화 소프트웨어

Docker

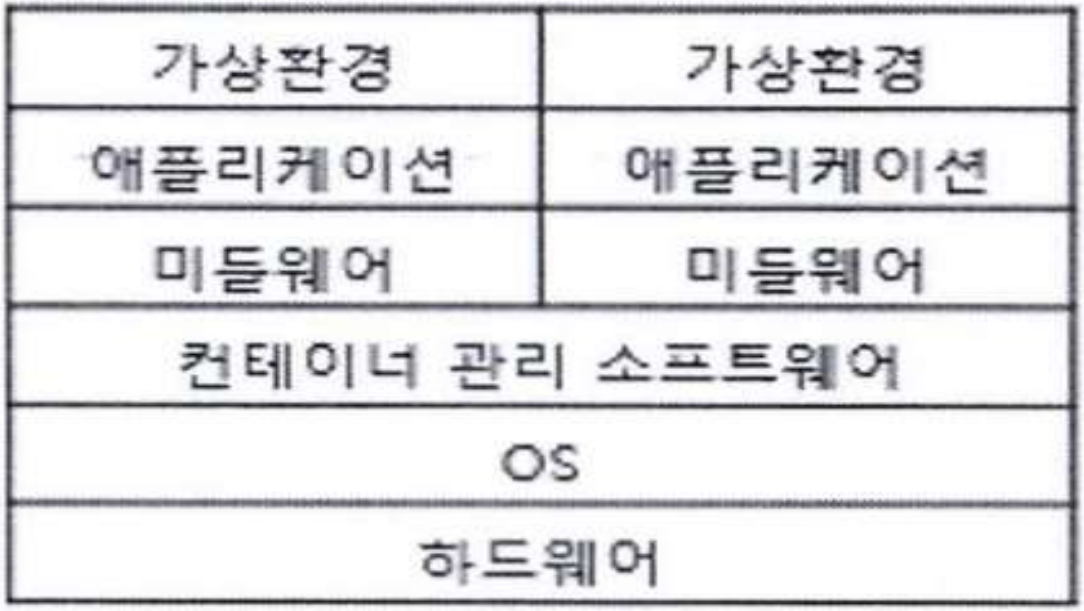
- ❖ Container 와 가상화
  - ✓ 가상화 방식
    - ● 하이퍼바이저 가상화
      - ◆ 구현 방식
        - ▪ 반가상화(Para-Virtualization)

Docker

- ❖ Container 와 가상화
  - ✓ 가상화 방식
    - Container 가상화
      - ◆ 호스트 운영체제 위에 Container 관리 소프트웨어를 설치하여 논리적으로 Container를 나누어 사용
      - ◆ Container는 애플리케이션 동작을 위한 라이브러리 와 애플리케이션 등으로 구성해서 각각 개별 서버처럼 사용
      - ◆ 장점: Container 가상화는 오버헤드가 적어 가볍고 빠른 장점이 있음
      - ◆ 단점: 다양한 운영체제를 사용할 수 없고 보안적으로 완전히 격리되지 않음
      - ◆ 컨테이너 가상화 소프트웨어
        - OpenVZ
        - LXC
        - Linux VServer
        - Docker
        - Oracle Solaris Zones

Docker

- ❖ Container 와 가상화
  - ✓ 가상화 방식
    - Container 가상화

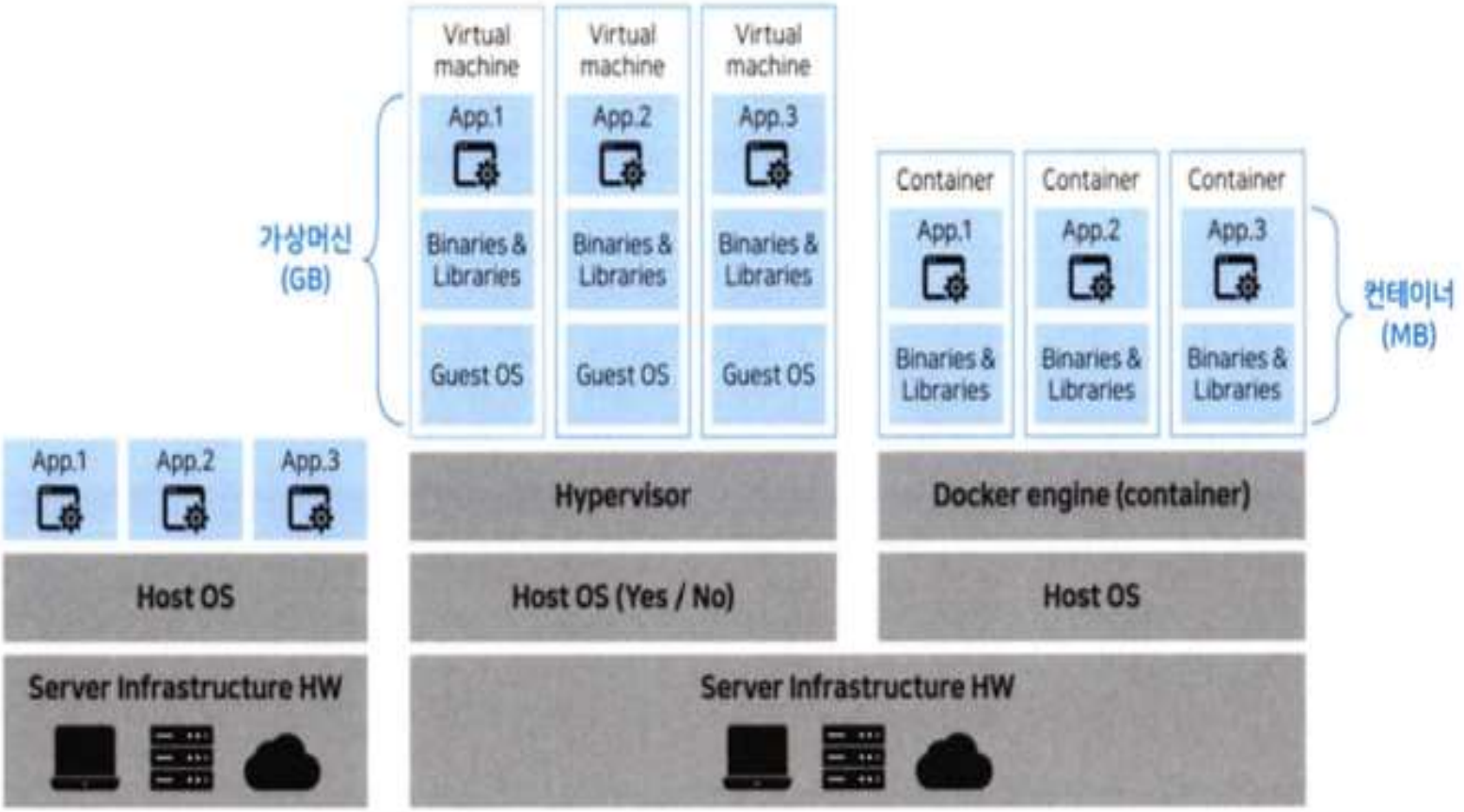


Docker

- ❖ Container 와 가상화
  - ✓ Container
    - 각 애플리케이션에는 운영체제가 아닌 의존성 요소만 포함됨
    - 애플리케이션 인터페이스는 호스트 운영체제와 직접 연결되며 게스트 운영체제 같은 추가 레이어가 없기 때문에 성능은 향상되고 리소스도 낭비되지 않고 Image 파일 크기도 작음
    - Container의 경우 호스트 운영체제의 프로세스 수준에서 격리하는데 Container들이 의존성 요소를 공유하지는 않음
    - Docker 엔진에서는 Container용 Linux 네임스페이스와 컨트롤 그룹을 생성해 이를 처리하는데 Docker의 보안이 Linux 커널 프로세스 격리를 기반으로 하는 이유이기도 하며 이런 솔루션은 충분히 검증됐지만 가상 머신이 제공하는 전체 운영체제 기반 격리보다는 덜 안전하다고 여겨지기도 함
    - 장점
      - ◆ 하이퍼바이저 와 게스트 운영체제가 없기 때문에 가벼움(수십 메가바이트)
      - ◆ 경량이기 때문에 만들어진 Image 복제, 이관, 배포가 쉬움
      - ◆ 게스트 운영체제를 부팅하지 않기 때문에 애플리케이션 시작 시간이 빠름
      - ◆ 가상 머신보다 경량이므로 더 많은 애플리케이션을 실행할 수 있음

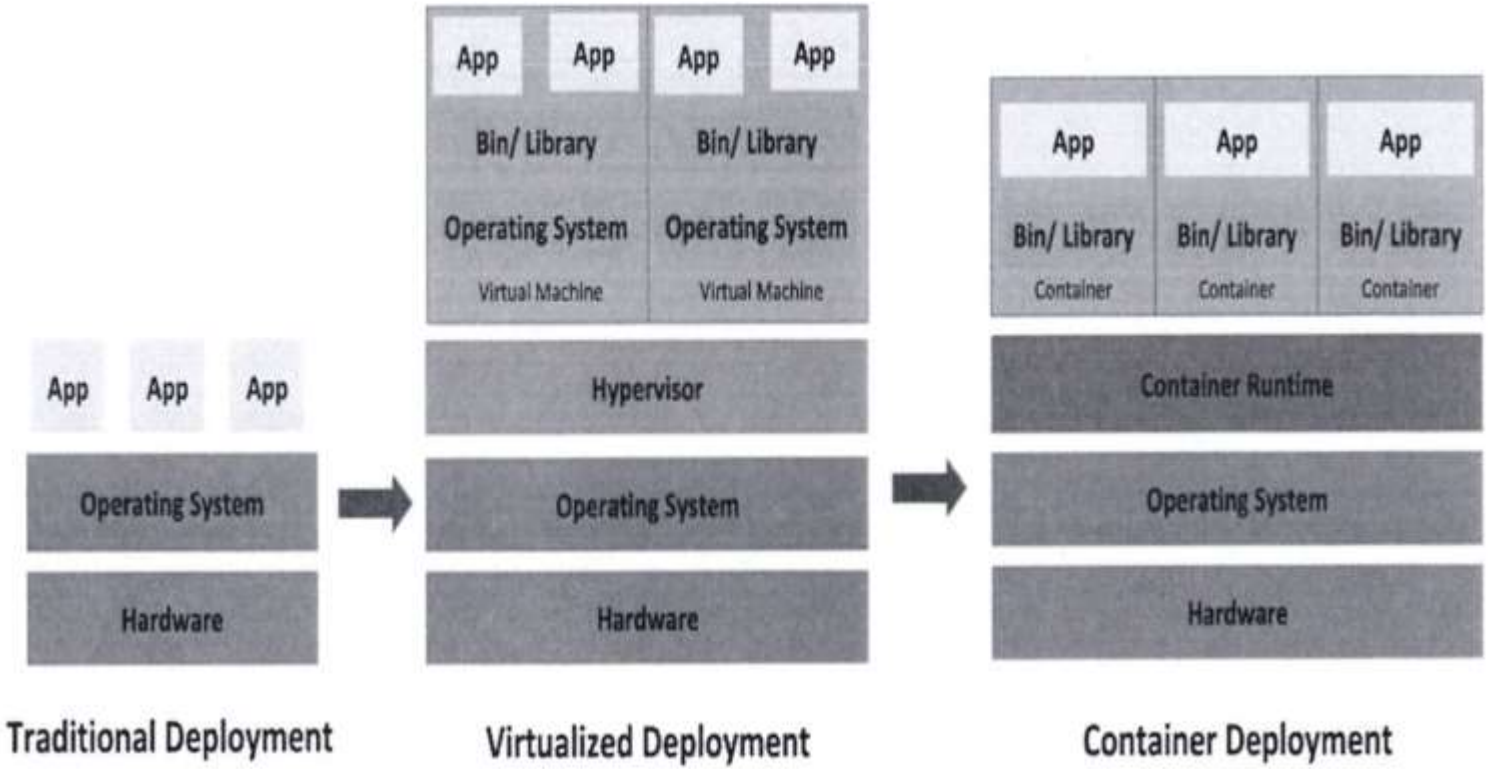
Docker

- Container 와 가상화
  - ✓ 로컬 서버 vs 가상 머신 vs Container



Docker

- Container 와 가상화
  - ✓ 애플리케이션 배포 방식의 변화

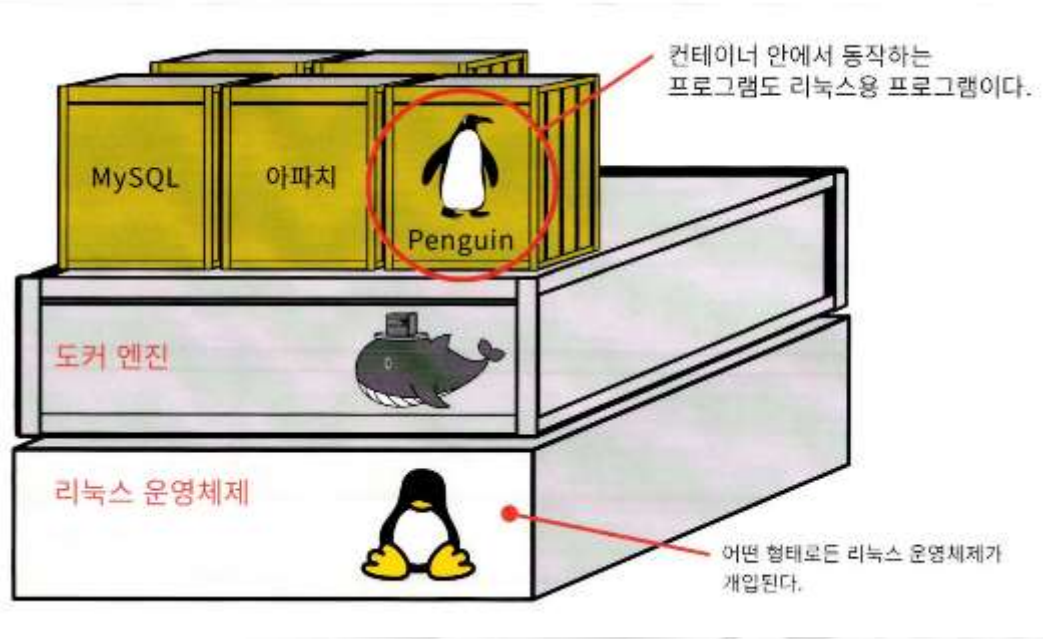


Docker

- Docker
  - ✓ Docker는 Container형 가상화 기술을 구현하기 위한 상주 애플리케이션과 이 애플리케이션을 조작하기 위한 명령행 도구로 구성되는 애플리케이션
  - ✓ 같이 사용되는 프로그램 과 데이터를 격리시키는 기능을 제공하는 애플리케이션 – Container 가상화
  - ✓ Container 와 Docker 엔진
    - 개인용 컴퓨터 또는 서버 상의 환경을 조립형 창고 같은 작은 방으로 분할해서 독립된 영역에 데이터나 프로그램을 두는 것으로 이 독립된 영역을 Container 라고 하고 이 Container를 다루는 기능을 제공하는 소프트웨어 중 하나가 Docker
    - Docker를 사용하려면 Docker 소프트웨어의 본체인 Docker 엔진을 설치해야 하고 Docker 엔진을 사용해 Container를 생성하고 구동시킬 수 있음
  - ✓ 특징
    - 마이크로 서비스 전환은 물론 DevOps, Testing 등 다양한 분야에서 활용
    - Linux Container 구현체의 사실상(de-facto) 표준

Docker

- Docker
  - ✓ Docker는 Linux를 사용
    - Linux 운영체제가 필요
    - 윈도우나 Mac 운영체제에서도 Docker를 구동할 수는 있지만 이 경우 내부적으로 Linux가 사용되며 Container에서 동작시킬 프로그램도 Linux 용 프로그램
    - Docker가 Linux 운영체제에서 사용하는 것을 전제로 만들어졌기 때문이며 윈도우나 Mac 운영체제에서 Docker를 사용하는 경우에도 내부적으로는 Linux 운영체제를 사용



Docker

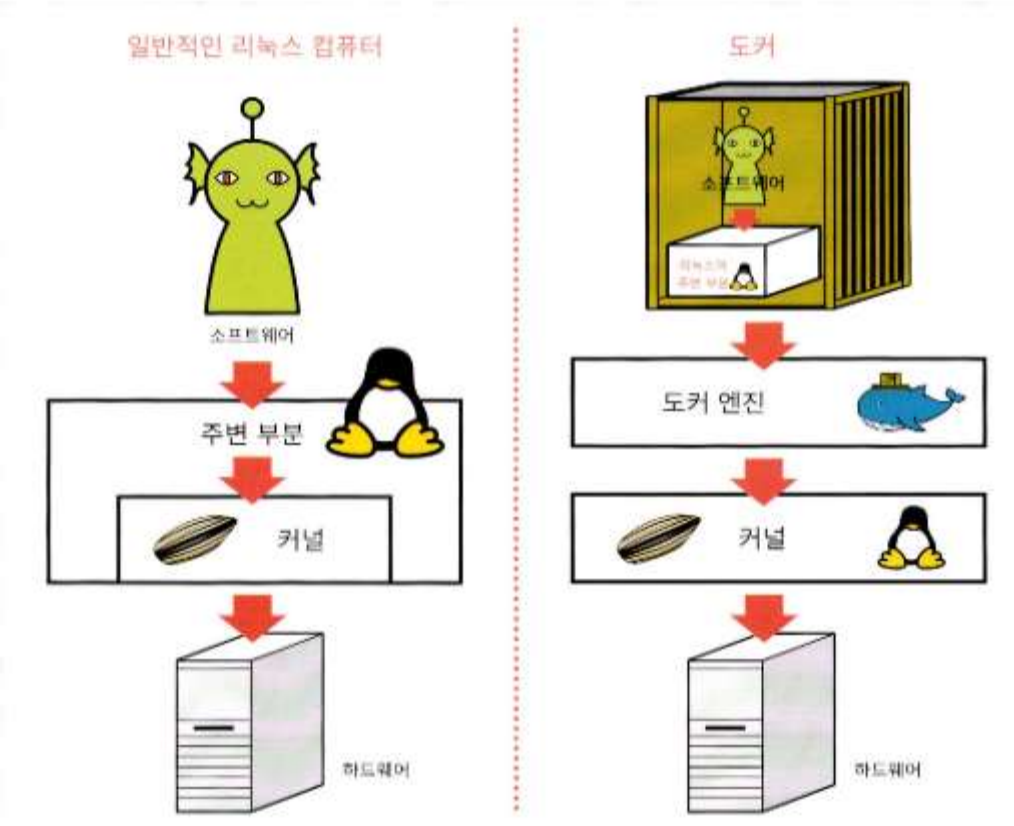
- Docker
  - ✓ LXC(Linux Container)
    - OS 수준의 가상화 구현: 단일 Linux 커널을 사용하는 제어 호스트에서 여러 개의 격리된 Linux 시스템(Container) 실행
    - Linux 커널에서 제공하는 주요 기능
      - chroot(change root): 특정 디렉토리를 최상위 디렉토리인 root로 인식하게끔 설정하는 Linux 명령
        - 유저 격리
        - 파일 격리
        - 네트워크 격리
      - cgroups: 자원(CPU, 메모리, 블록I/O, 네트워크 등)을 제한하고 격리시키는 Linux 커널 기능
      - 네임스페이스(Namespace)
        - 프로세스를 독립시켜주는 가상화 기술(Linux 커널 리소스의 분리)
        - 운영 환경에 대한 애플리케이션 보기의 완전한 분리를 허용
        - 프로세스 트리, 네트워킹, 사용자 ID 및 마운트 된 파일 시스템 포함

Docker

- Docker
  - ✓ 데이터나 프로그램을 독립된 환경에 격리하는 이유
    - 대부분의 프로그램은 프로그램 단독으로 동작하는 것이 아니라 어떤 실행 환경이나 라이브러리 또는 다른 프로그램을 이용해 동작
    - PHP로 작성된 프로그램을 실행하려면 PHP 실행 환경이 필요하고 Python으로 작성된 프로그램은 외부 라이브러리를 사용하는 경우가 많고 다른 소프트웨어 역시 단일 프로그램이 아니라 여러 개의 프로그램으로 구성된 경우가 많은데 워드프레스는 MySQL 데이터베이스를 따로 갖추지 않으면 사용할 수 없고 다른 프로그램과 특정한 디렉토리를 공유하거나 같은 경로에 설정 정보를 저장하는 경우도 있어서 프로그램 하나를 업데이트하면 다른 프로그램에도 영향을 미치고 실행 환경이나 라이브러리, 디렉토리 나 설정 파일에서도 이런 현상이 발생할 수 있음
    - Docker Container를 사용해 프로그램을 격리하면 여러 프로그램이 한 서버에서 실행되면서 발생하는 문제를 대부분 해결할 수 있음

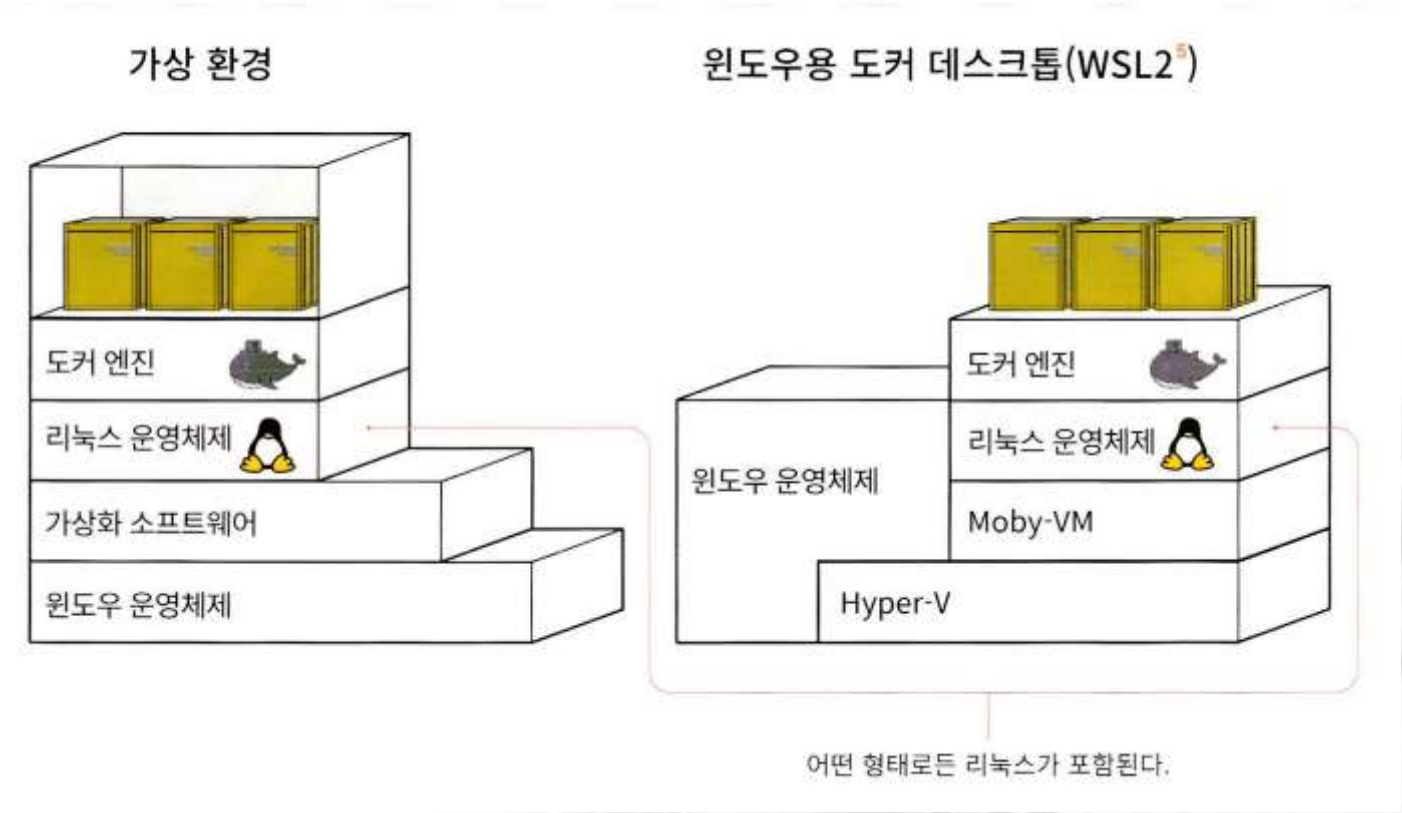
Docker

- Docker
  - ✓ 동작 원리
    - Container 안에는 운영체제 와 유사한 것이 들어있음



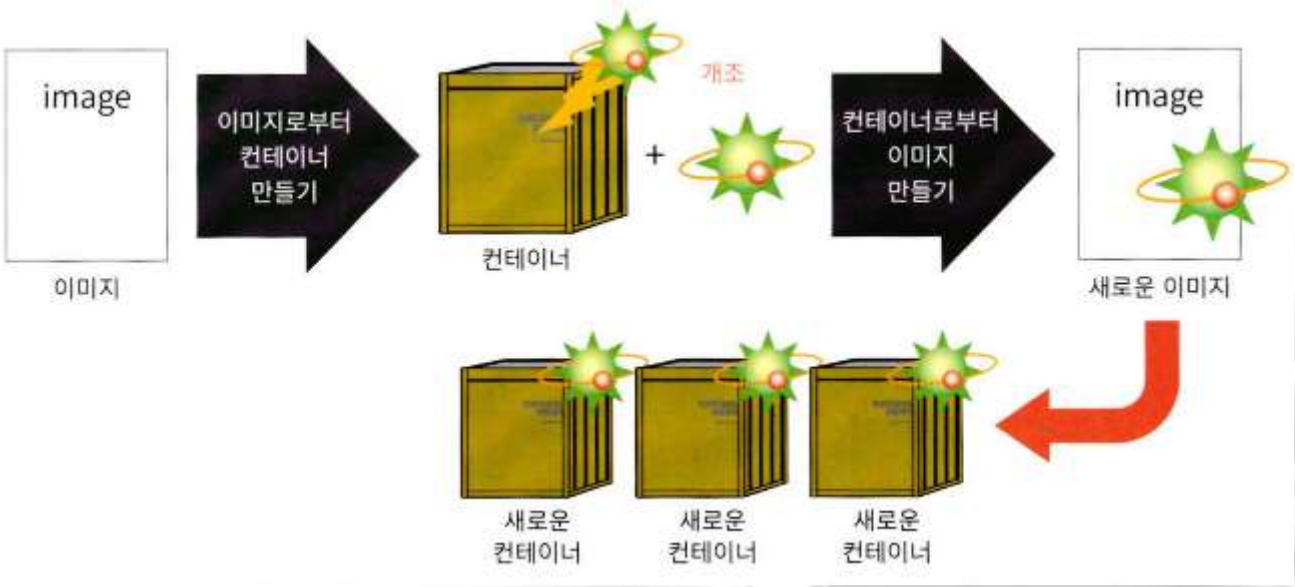
Docker

- Docker
  - ✓ 동작 원리
    - 윈도우 와 Mac 운영체제에서 Docker 구동: Linux 운영체제를 끌어들여서 사용



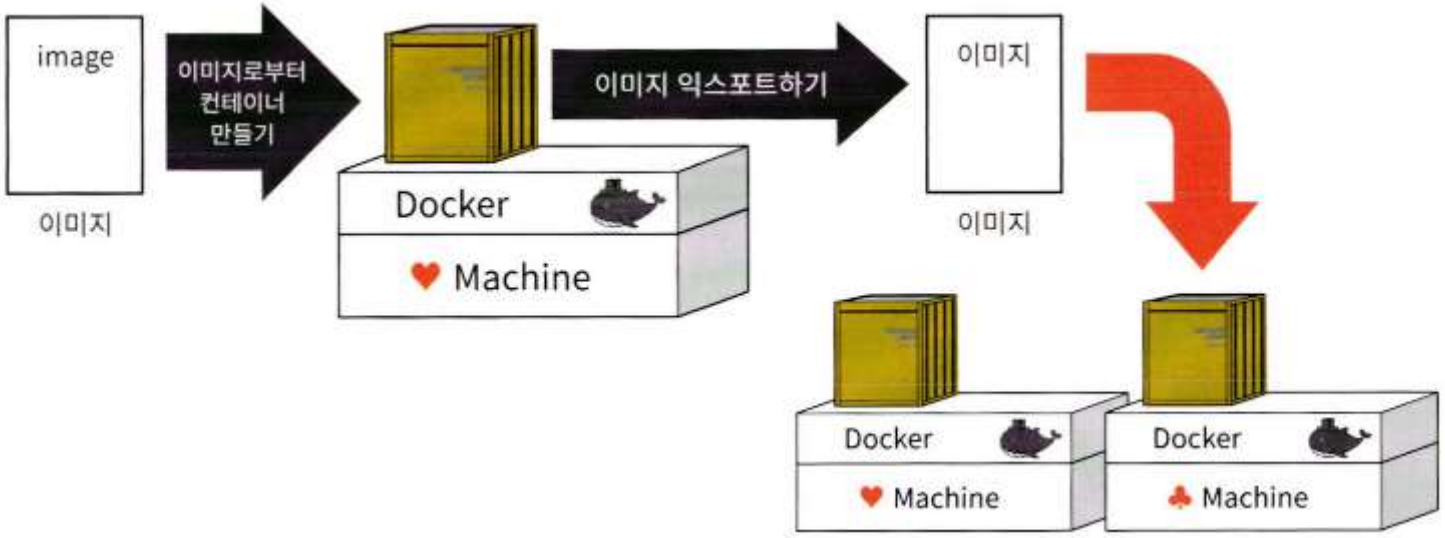
Docker

- Docker
  - ✓ Image 와 Container
    - Image는 Container를 만들어내는 설계도(클래스) 역할을 수행하는 것으로 하나만 있으면 동일한 Container(인스턴스)를 여러 개 생성할 수 있음
    - Container로도 Image를 생성할 수 있음 – 기존 Image를 수정해서 새로 만드는 것이 가능



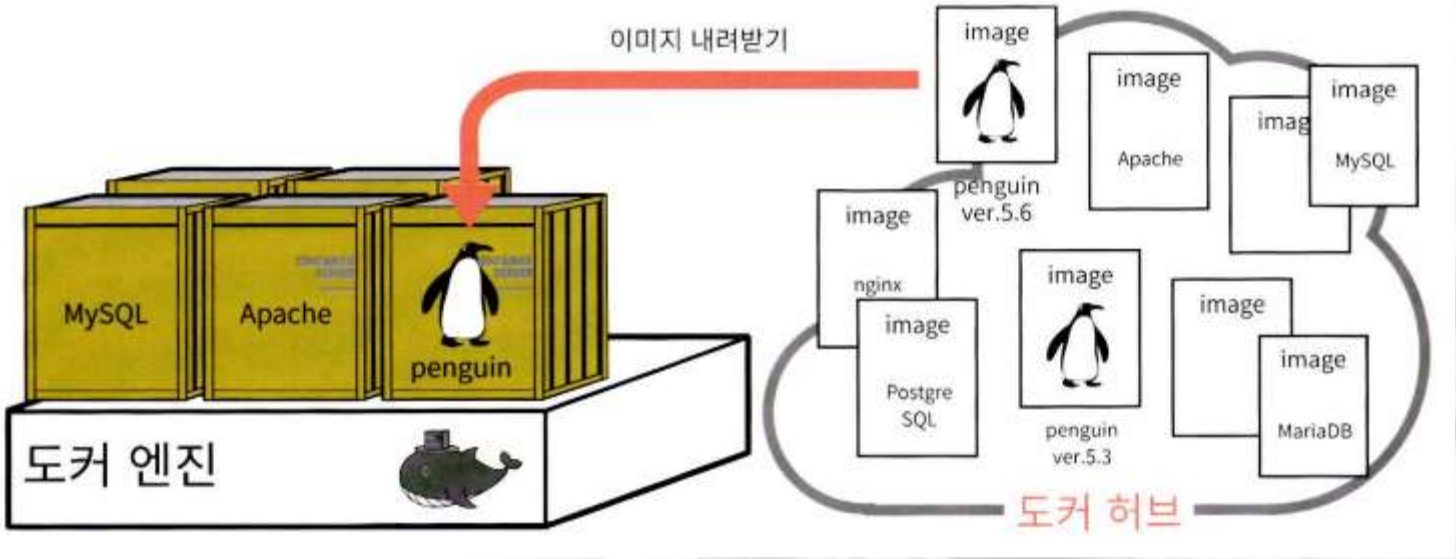
Docker

- Docker
  - ✓ Docker 엔진 간에 이동이 가능
    - 동일한 Container를 여러 개 만들지 않더라도 다른 물리 서버에 설치된 Docker 엔진으로 Container를 이동시킬 수 있음
    - Container는 Docker 엔진만 설치되어 있으면 구동이 가능하므로 다른 서버나 컴퓨터에 Docker 엔진을 설치하고 새로운 Docker 엔진에 Image를 이용해 동일한 Container를 생성하면 되는데 Container 자체가 이동하는 것은 아니지만 Image를 통해 Container가 이동한 것과 같은 효과를 얻을 수 있음



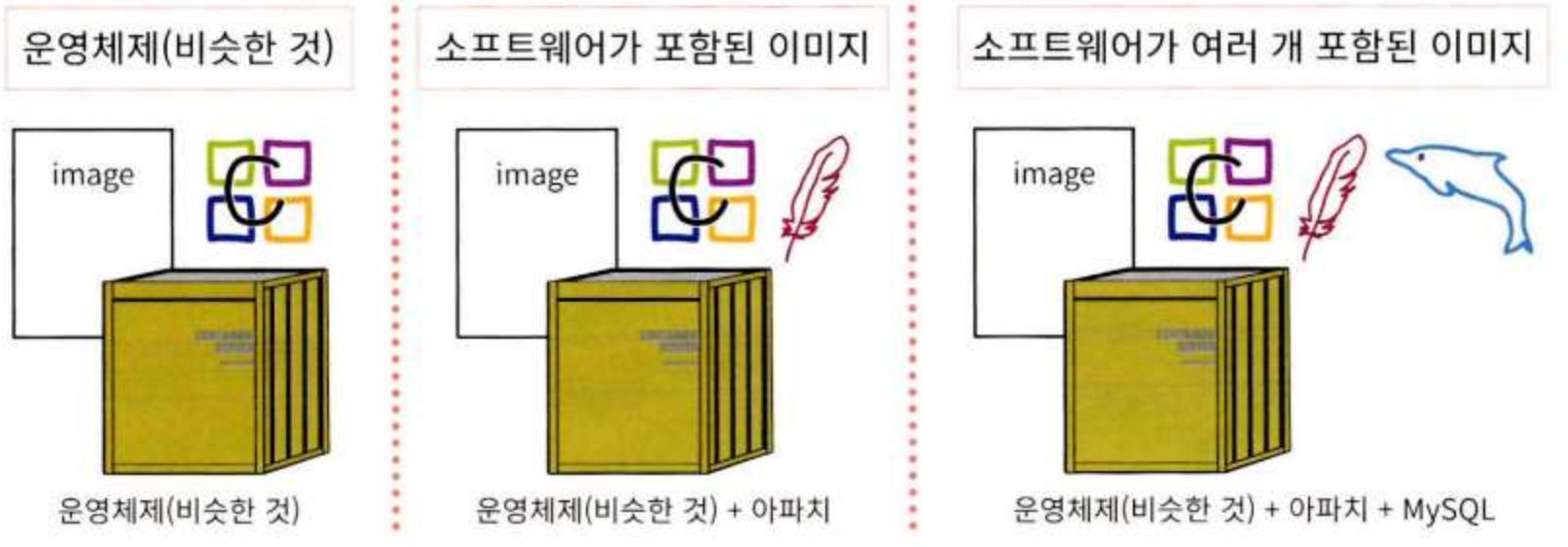
Docker

- Docker
  - ✓ Docker Hub 와 Docker Image
    - Image는 주로 Docker Hub에서 구하게 되는데 Docker Hub는 공식적으로 운영되는 Docker Registry(Docker Image를 배포하는 서비스)로 공개된 Container Image가 모여 있는 곳
    - https://hub.docker.com/



Docker

- Docker
  - ✓ Docker Hub 와 Docker Image
    - Docker Image의 종류



Docker

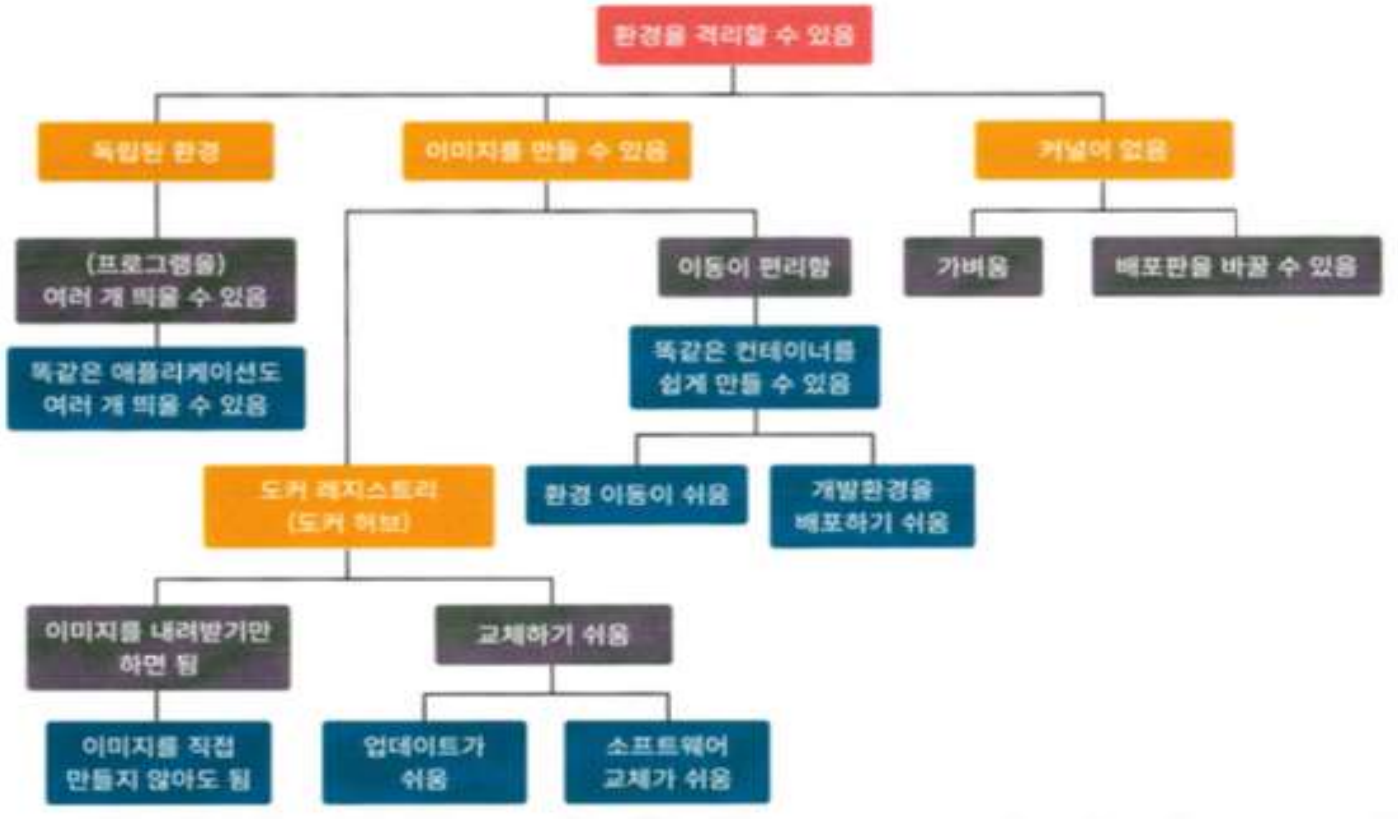
- Docker
  - ✓ 다양한 형태로 조합이 가능한 Container
    - Docker를 사용할 때의 원칙 중 하나로 한 Container에 한 프로그램 이라는 것이 있는데 하나의 프로그램만 담긴 Container를 사용한다는 의미로 보안 및 유지 관리 측면에서 유리하기 때문에 많이 쓰이는 정책
    - 워드프레스를 사용하려면 아파치(웹 서버 소프트웨어), MySQL 같은 DBMS, 워드프레스로 세 가지 소프트웨어가 필요한데 Docker를 사용해 워드프레스를 구축하는 방법은 이들을 각각 별도의 Container로 구성할 수도 있고 한 Container에 모두 집어넣는 방법도 있음

Docker

- Docker
  - ✓ Docker Container는 쓰고 버리는 일회용품
  - ✓ 데이터 저장
    - Container를 폐기하면 데이터도 소멸됨
    - Docker가 설치된 물리적 서버의 디스크를 마운트해서 데이터를 저장하는 것이 가능

Docker

- Docker
  - ✓ 특성



Docker

- Docker
  - ✓ 장점
    - 애플리케이션 배포에 특화돼 있기 때문에 애플리케이션 개발 및 운영을 Container 중심으로 할 수 있음
    - 가상화 소프트웨어보다 더 가볍게 동작
    - Docker는 이식성이 뛰어나서 로컬 머신의 Docker 환경에서 실행하던 Container를 다른 컴퓨터에 있는 Docker 환경에 배포하거나 반대로 다른 서버의 Docker 환경에서 동작하던 Container를 로컬로 가져올 수 있는데 개발 환경 과 운영 환경을 거의 동등하게 재현할 수 있음
    - 물리적 환경의 차이나 서버 구성의 차이를 무시할 수 있음
    - Container 간의 연동이나 클라우드 플랫폼 지원 등 여러 면에서 장점이 있음

Docker

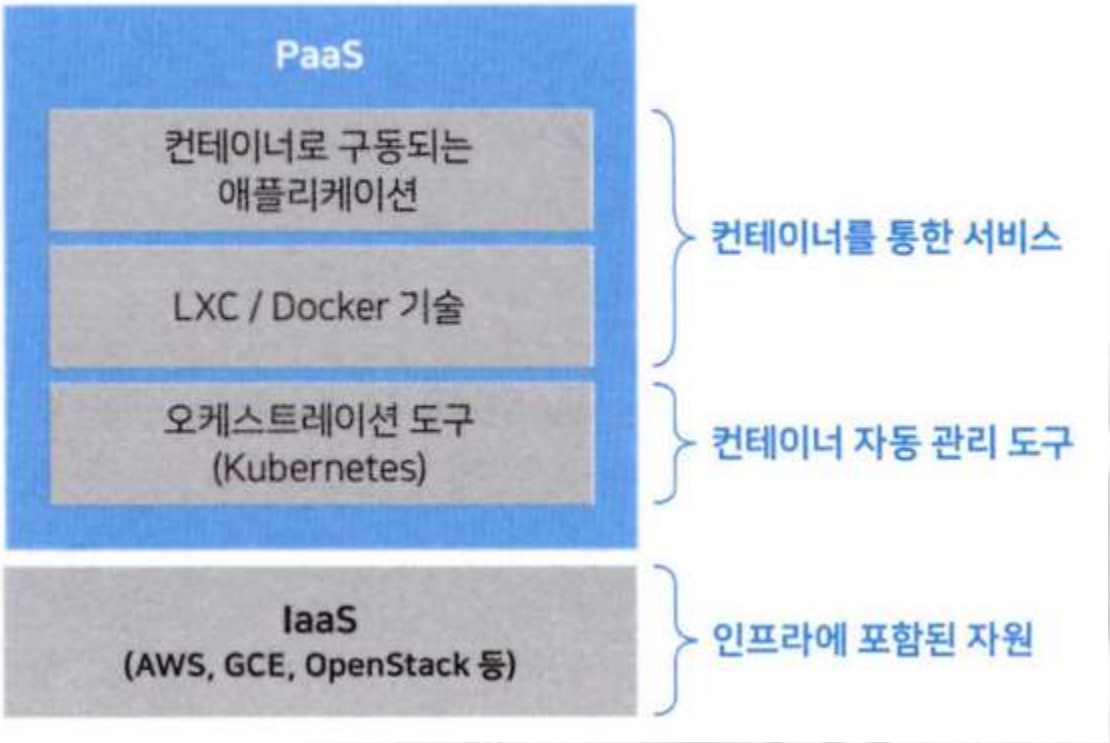
- Docker
  - ✓ 추천하지 않는 경우
    - Docker Container의 내부는 Linux 계열 운영체제와 같은 구성을 취하는 것이 많은데 Container는 운영체제의 동작을 완전히 재현하지는 못하기 때문에 좀 더 엄밀한 Linux 계열 운영체제의 동작이 요구되는 가상 환경을 구축해야 한다면 기존 방법대로 VMWare나 VirtualBox 같은 가상화 소프트웨어를 사용하는 것이 나음
    - FreeBSD 같은 비 Linux 환경이 필요한 경우에도 Docker가 적합하지 않음

Docker

- Docker
  - ✓ 특징
    - Container 가상화는 프로세스 가상화로 Container 엔진인 Docker 와 오케스트레이션 도구인 Kubernetes는 호스트 운영체제의 커널을 공유하고 그 위에 실행 파일 및 라이브러리, 기타 구성 파일 등을 Image로 빌드하여 패키지로 배포하는 방식
    - Container는 최신 기술이 아니며 오랜 시간 동안의 변화를 통해 Linux Container(Linux Container, LXC) 기술로 완벽해졌고 LXC 기술을 차용한 Docker를 통해 Container의 생성과 배포를 위한 완벽한 가상화 솔루션, Container 표준화로 자리 잡음
    - Container는 코드와 모든 종속성을 패키지화하는 표준 소프트웨어 단위로 애플리케이션이 한 컴퓨팅 환경에서 다른 컴퓨팅 환경으로 빠르고 안정적으로 실행되도록 하는데 Docker Container Image는 애플리케이션을 실행하는 데 필요한 모든 것(코드, 런타임, 라이브러리 등)을 포함하는 경량의 독립형 실행 가능 소프트웨어 패키지라고 정의할 수 있음
    - Docker Container Image는 Docker Hub로부터 내려받거나 Dockerfile을 통해 생성하여 Docker 엔진을 이용해 실행하면 Container 서비스가 됨
    - Docker의 주요 도구
      - LXC를 이용한 Container 구동하는 containerd : Linux 및 윈도우용 데몬으로 Image 전송 및 스토리지에서 Container 실행 및 감독, 네트워크 연결까지 호스트 시스템 전체 Container의 라이프 사이클을 관리
      - 통합 Buildkit: Docker 파일의 설정 정보를 이용하여 Docker Image를 빌드하는 오픈 소스 도구이며 빠르고 정확하게 여러 가지 아키텍처 향상 기능을 제공
      - Docker CLI: Docker 명령을 수행하는 기본적인 방법은 CLI로 제공

Docker

- Docker
  - ✓ 주 용도는 PaaS 서비스를 가능하게 하는 소프트웨어 개발 환경을 제공하는 것인데 Container 서비스에 대한 자동화된 관리, 트래픽 라우팅, 로드 밸런싱 등을 쉽게 하려면 오케스트레이션 기능이 추가로 요구됨

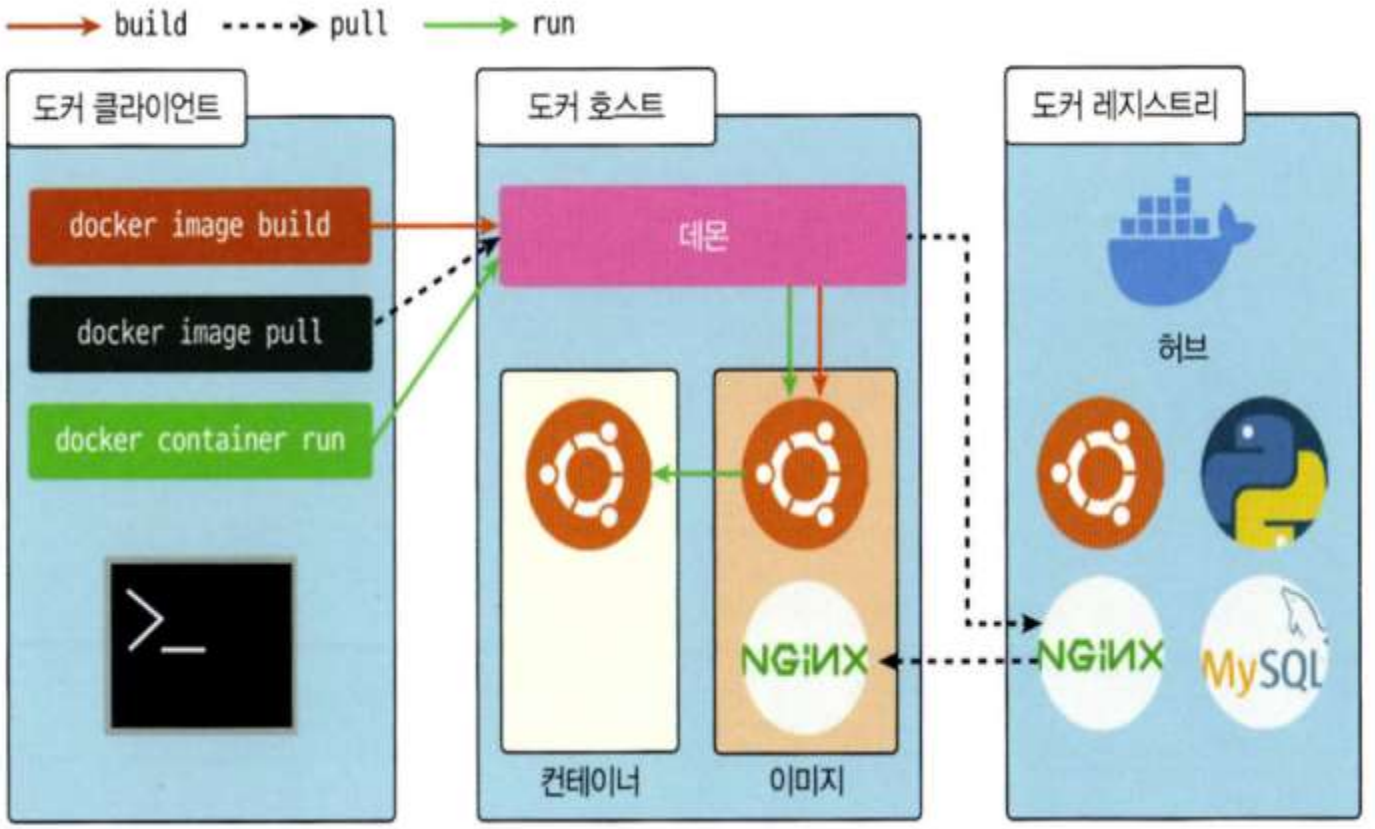


Docker

- Docker
  - ✓ Docker의 주요 구성 요소
    - Docker Client: 도커에 명령을 내릴 수 있는 CLI(Command Line Interface) 도구를 의미하는데 도커 클라이언트를 이용해 컨테이너, 이미지, 볼륨 등을 관리할 수 있음
    - Docker Host: 도커를 설치한 서버 혹은 가상 머신
    - Docker Engine: Docker를 이용한 애플리케이션 실행 환경 제공을 위한 핵심 요소
    - Docker-Compose: 의존성 있는 독립된 Container에 대한 구성 정보를 yaml 코드로 작성하여 일원화된 애플리케이션 관리를 가능하게 하는 도구
    - Docker Registry: Docker Container Image를 push/pull 할 수 있는 레지스트리 구축에 사용하는 도구
    - Docker Swarm: 여러 Docker 호스트를 클러스터로 구축하여 관리할 수 있는 Docker 오케스트레이션 도구
    - Docker Hub: 전 세계 Docker 사용자들과 함께 Docker Container Image를 공유하는 클라우드 서비스

Docker

- Docker
  - ✓ Docker의 주요 구성 요소



Docker

- Docker
  - ✓ Docker Engine
    - Docker는 2013년 3월에 닷클라우드의 엔지니어였던 솔로몬 하익스가 오픈 소스로 공개 발표
    - Docker는 기존의 Linux Container(LXC) 기술을 이용하여 애플리케이션을 Container로 사용할 수 있게 Go 언어로 만들었고 출시 이후 꾸준한 기술 개발을 통해 Container 가상화를 이용한 차세대 클라우드 인프라 솔루션의 표준이 됨
    - 초기 Docker는 Linux Container 기술인 LXC를 기반으로 하는 Container였지만 이후 0.9.0 버전부터는 libcontainer OCI를 이용하였고 1.11.0 이후 버전부터는 runC OCI를 이용하며 현재 버전에 포함된 runC 라이브러리는 운영체제에서 독립적으로 사용되는 일종의 드라이버로 이를 통해 호스트 운영체제 의존성이 제거되면서 Linux 플랫폼에 의존적인 LXC를 대체하게 됐고 전체적인 구조는 dockerd, containerd 데몬과 runC 라이브러리를 이용해 Container들을 관리
    - Docker Container 기술은 가상화된 공간을 만들기 위해 Linux 커널에서 Container를 제어하는 chroot, cgroup, namespace API를 런타임으로 사용함으로써 프로세스 단위의 격리 환경을 만들 수 있으며 Container는 호스트 운영체제의 커널을 공유하여 사용하고 애플리케이션 Container 자체에는 필요한 실행 파일과 라이브러리만 존재하기 때문에 만들어진 Container Image의 용량이 가상 머신의 수 기가 바이트에 비해 수 메가바이트로 작고 배포 시간도 빠름

Docker

- Docker
  - ✓ 엔진 설치
    - Docker는 운영체제가 64bit 인 경우만 동작
    - Docker Hub 사이트에 접속해서 회원 가입
      https://hub.docker.com/signup?next=%2Feditions%2Fcommunity%2Fdocker-ce-desktop-mac%3Fref%3Dlogin

Docker

- Docker
  - ✓ 엔진 설치
    - Windows: https://hub.docker.com/editions/community/docker-ce-desktop-windows
      - 윈도우에서는 Linux 커널을 다운로드 받아야 함 – https://learn.micr운영체제oft.com/en-us/windows/wsl/install
        - wsl --install
        - wsl --set-default-version 2

    - Mac: https://hub.docker.com/editions/community/docker-ce-desktop-mac

Docker

- Docker
  - ✓ 엔진 설치
    - Linux(Ubuntu)에서 설치
      - 기존 버전 제거: sudo apt remove $(dpkg --get-selections docker.io docker-compose docker-compose-v2 docker-doc podman-docker containerd runc | cut -f1)
      - Docker에서 제공하는 공식 GPG(GNU Privacy Guard) key 추가
        - sudo apt update
        - 보안 웹 통신(인증서 관리)과 데이터 다운로드를 위해 필수적인 두 가지 도구를 설치: sudo apt install ca-certificates curl
        - GPG 키(소프트웨어 저장소의 신뢰성을 확인하는 디지털 열쇠)를 안전하게 보관할 전용 디렉토리 생성: sudo install -m 0755 -d /etc/apt/keyrings
        - Docker 공식 서버에서 우리는 진짜 Docker가 맞습니다 라고 증명하는 디지털 서명(GPG 키)을 다운로드하여 내 컴퓨터에 저장하는 명령: sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
        - Docker 보안 열쇠(GPG 키)를 시스템의 모든 사용자가 읽을 수 있도록 권한을 조정하는 설정: sudo chmod a+r /etc/apt/keyrings/docker.asc

Docker

- Docker
  - ✓ 엔진 설치
    - Linux(Ubuntu)에서 설치
      - ◆ 데비안 계열의 Docker repository 추가

        sudo tee /etc/apt/sources.list.d/docker.sources <<EOF

        Types: deb

        URIs: https://download.docker.com/linux/ubuntu

        Suites: $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}")

        Components: stable

        Signed-By: /etc/apt/keyrings/docker.asc

        EOF
      - ◆ 패키지 업데이트

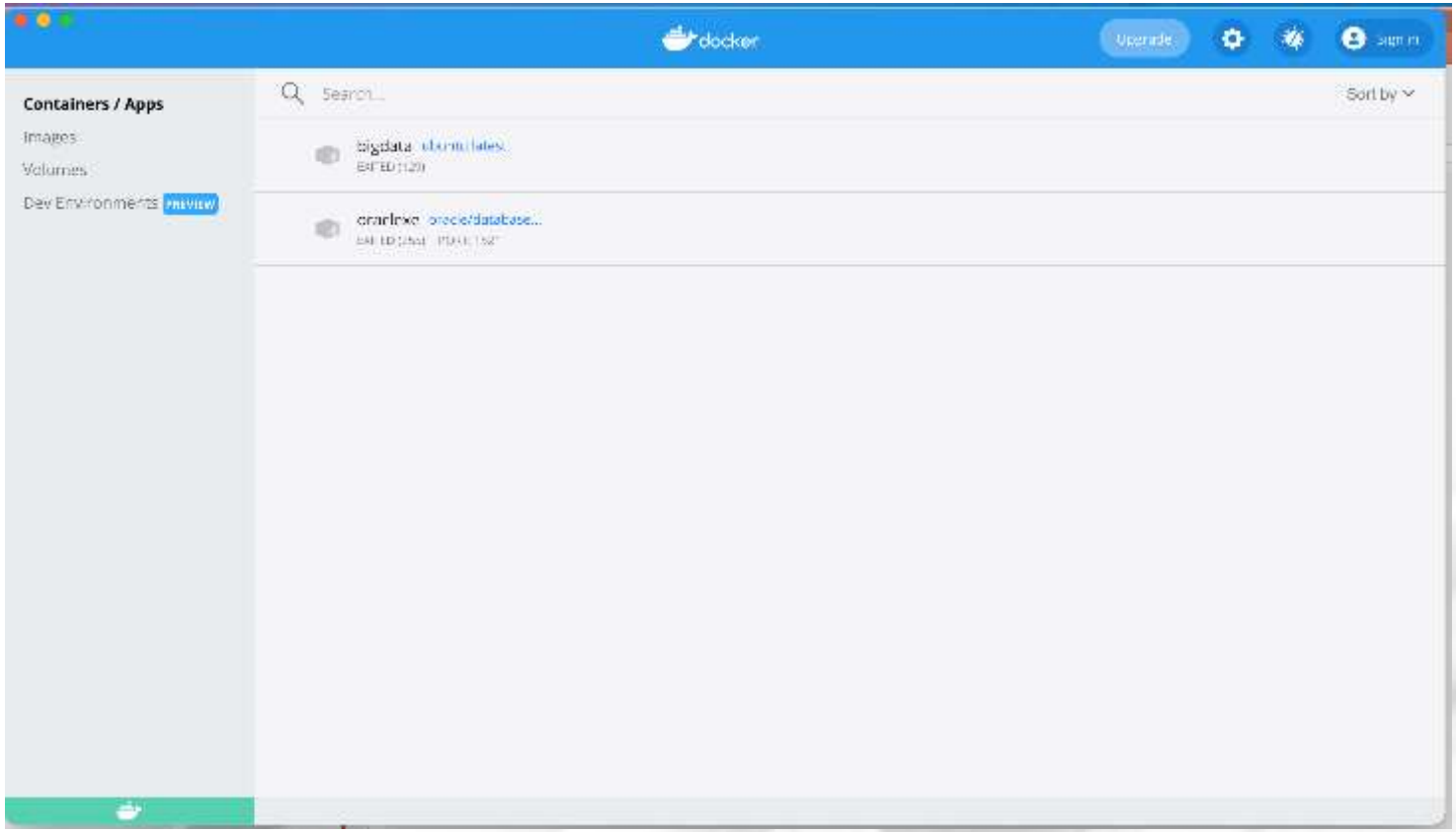
        sudo apt update

Docker

- ❖ Docker
  - ✓ 엔진 설치
    - ● Linux(Ubuntu)에서 설치
      - ◆ Docker CE 설치: sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
      - ◆ 설치 확인: sudo docker version
      - ◆ 서비스를 시작할 때 수행되도록 설정: sudo systemctl enable docker
      - ◆ 서비스 시작: sudo service docker start
      - ◆ 데몬 확인: sudo systemctl status docker
      - ◆ Docker 그룹에 현재 사용자 등록
        - ▪ sudo usermod -aG docker $(whoami)
        - ▪ sudo chmod 666 /var/run/docker.sock
        - ▪ sudo service docker restart

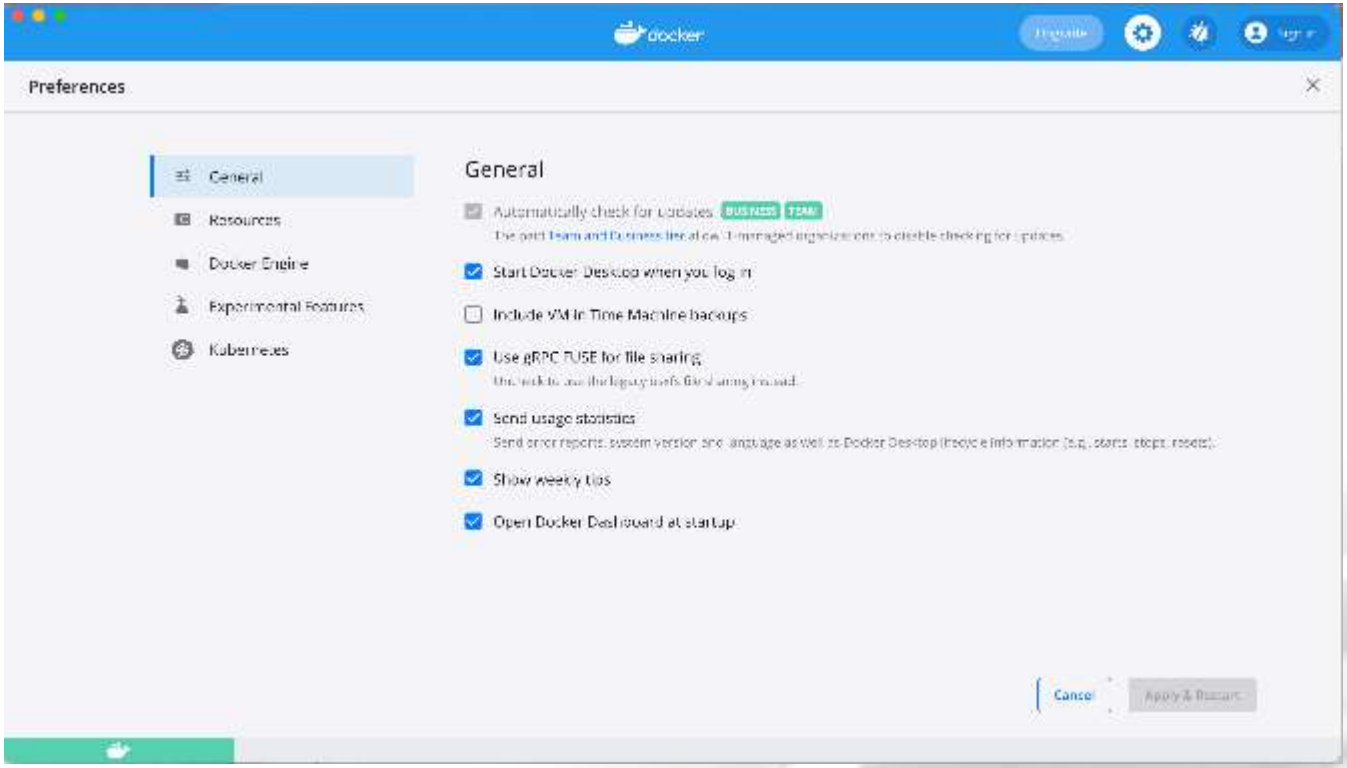
Docker

- Docker 기본 설정
  - ✓ Docker Dashboard – 설치된 image 와 Container 확인 가능



Docker

- Docker 기본 설정
  - ✓ Docker Preference – 자동 실행 및 자동 업데이트 및 각종 환경 설정 가능



Docker

- Docker 기본 설정
  - ✓ Docker Preference
    - General: 자동 로그인 및 자동 업데이트 설정
    - Resources: CPU, RAM, Disk Size 설정
    - Docker Engine: JSON 포맷으로 Docker를 설정할 수 있는데 윈도우용/Mac 운영체제용 Docker의 설정 화면에 나오지 않는 사항을 변경하려면 이곳에서 JSON 문자열을 수정
    - Experimental Features: 현재 실험하고 있는 기능
    - Kubernetes: 쿠버네티스 사용 설정

Docker

- Docker 기본 사용
  - ✓ Image 다운로드: docker pull hello-world
  - ✓ Image 확인: docker images

```
adam@adam:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
hello-world   latest    feb5d9fea6a5   19 months ago    13.3kB
```

Docker

- Docker 기본 사용
  - ✓ Image를 Container로 생성: docker run hello-world
  - ✓ Container 확인: docker ps -a

```
adam@adam:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

adam@adam:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
e490de125bfb   hello-world   "/hello"   9 seconds ago    Exited (0) 8 seconds ago              elated_khayyam
```

Docker

- Docker 정보 확인
  - ✓ Docker 버전 확인 – docker version

```
Client:
 Cloud integration: v1.0.35-desktop+001
 Version:           24.0.5
 API version:       1.43
 Go version:        go1.20.6
 Git commit:        ced0996
 Built:             Fri Jul 21 20:32:30 2023
 운영체제/Arch:           darwin/arm64
 Context:           desktop-linux
```

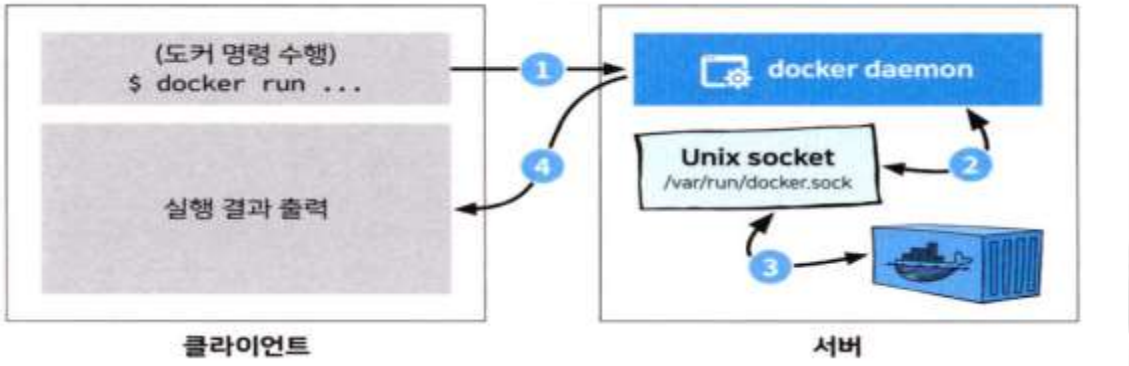
Docker

- Docker 정보 확인
  - ✓ Docker 버전 확인 – docker version

```
Server: Docker Desktop 4.22.0 (117440)
 Engine:
  Version:          24.0.5
  API version:      1.43 (minimum version 1.12)
  Go version:       go1.20.6
  Git commit:       a61e2b4
  Built:            Fri Jul 21 20:35:38 2023
  운영체제/Arch:          linux/arm64
  Experimental:     false
 containerd:
  Version:          1.6.21
  GitCommit:        3dce8eb055cbb6872793272b4f20ed16117344f8
 runc:
  Version:          1.1.7
  GitCommit:        v1.1.7-0-g860f061
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Docker

- ❖ Docker 정보 확인
  - ✓ Docker 버전 확인 – docker version
    - 설치된 Docker 엔진은 클라이언트와 서버로 구분
    - 클라이언트는 Docker 명령을 받고 결과를 출력하는 역할을 수행하고 서버는 Docker 엔진 즉 Docker Daemon을 이용해 Container 시작, 운영, 정지 등을 담당
    - Docker 버전 정보에 나타나는 클라이언트와 서버의 상호 실행 원리



- ◆ 클라이언트는 Docker 명령을 수행하는 명령줄을 제공
- ◆ 수행된 Docker 명령은 서버의 Docker 데몬으로 전달
- ◆ Docker 데몬은 docker-socket이 보유한 Docker API를 이용해 Container 생성
- ◆ 수행된 Container에 포함된 서비스 결과를 클라이언트에 전달

Docker

- Docker 정보 확인
  - ✓ 시스템에 설치된 Docker 구성 정보 - docker info
    - 커널 정보, 현재 Container 수 및 Image 수를 출력
    - 사용 중인 스토리지 드라이버에 따른 풀 이름
    - 데이터 파일, 메타 데이터 파일, 사용된 데이터 공간, 총 데이터 공간, 사용된 메타 데이터 공간, 총 메타 데이터 공간 정보

Docker

- Docker 정보 확인
  - ✓ 시스템에 설치된 Docker 구성 정보 - docker info

```
(base) adam@choongangui-MacBookPro ~ % docker info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc., v0.10.0)
  compose: Docker Compose (Docker Inc., v2.15.1)
  dev: Docker Dev Environments (Docker Inc., v0.0.5)
  extension: Manages Docker extensions (Docker Inc., v0.2.17)
  sbom: View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc., 0.6.0)
  scan: Docker Scan (Docker Inc., v0.23.0)

Server:
 Containers: 4
  Running: 2
  Paused: 0
  Stopped: 2
 Images: 4
 Server Version: 20.10.22
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 2
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: inactive
 Runtimes: io.containerd.runc.v2 io.containerd.runtime.v1.linux runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 9ba4b250366a5ddde94bb7c9d1def331423aa323
 runc version: v1.1.4-0-g5fd4c4d
 init version: de40ad0
 Security Options:
  seccomp
   Profile: default
  cgroupns
 Kernel Version: 5.15.49-linuxkit
 Operating System: Docker Desktop
 OSType: linux
 Architecture: x86_64
 CPUs: 4
 Total Memory: 7.675GiB
```

Docker

- Docker 정보 확인
  - ✓ 디스크 사용량 확인 - docker system df

```
(base) adam@choongangui-MacBookPro ~ % docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          4         4         2.272GB   77.81MB (3%)
Containers      4         0         1.32GB    1.32GB (100%)
Local Volumes   3         3         339.5MB   0B (0%)
Build Cache     0         0         0B        0B
```

Docker

- Docker 정보 확인
  - ✓ 디스크 사용량 확인 - docker system df -v

```
(base) adam@choongangui-MacBookPro ~ % docker system df -v
Images space usage:

REPOSITORY            TAG       IMAGE ID       CREATED       SIZE      SHARED SIZE   UNIQUE SIZE   CONTAINERS
mysql                 latest    57da161f45ac   7 days ago    517.5MB   0B            517.5MB       1
mariadb               latest    5986086dc2c5   9 days ago    400.5MB   77.81MB       322.7MB       1
mongo                 latest    a440572ac3c1   13 days ago   639.3MB   77.81MB       561.5MB       1
jaspeen/oracle-xe-11g latest    52fad1fe2d7a   7 years ago   792.1MB   0B            792.1MB       1

Containers space usage:

CONTAINER ID   IMAGE                 COMMAND                  LOCAL VOLUMES   SIZE      CREATED       STATUS                          NAMES
42d4898ddd62   mariadb               "docker-entrypoint.s…"   1               0B        5 hours ago   Exited (255) 19 minutes ago     mariadb
4e39ce0113fe   mongo                 "docker-entrypoint.s…"   1               0B        6 hours ago   Exited (255) 19 minutes ago     mongodb-container
64fc98f62975   jaspeen/oracle-xe-11g "/entrypoint.sh "        0               1.32GB    6 hours ago   Exited (137) 2 hours ago        oracle11g
04eae573fcc2   mysql                 "docker-entrypoint.s…"   1               0B        8 hours ago   Exited (0) 2 hours ago          mysql

Local Volumes space usage:

VOLUME NAME                                                        LINKS     SIZE
2b75f18ceba7323c65989cb609336b0c923fa91c40f32b82bd52d6019294d9ac   1         197.1MB
39e92b43ce88d6efce89952ea1eef3396b846997478daaa1f33d4b5ef436cba5   1         0B
8cde13fb6a7991d359ccd9f7cf38b1db4eb3edac9254d2528981c73e59e2b95a   1         142.4MB

Build cache usage: 0B

CACHE ID   CACHE TYPE   SIZE      CREATED   LAST USED   USAGE     SHARED
```

Docker

- Docker 정보 확인
  - ✓ Docker 관련 이벤트 정보를 표시하는 명령 - docker system events
    - 롤링 로그이며 한 번에 최대 1,000 개의 이벤트를 보유
    - 실습
      - 터미널에서 docker system events 수행
      - 다른 터미널에서 docker run -itd -p 80:80 --name=webapp nginx 명령 수행

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
e886f0f47ef5: Pull complete
9a9138853e32: Pull complete
598a42ec6587: Pull complete
82e490cc2043: Pull complete
948128637a91: Pull complete
e4cad15ac3f6: Pull complete
096332b242c2: Pull complete
Digest:
sha256:32da30332506740a2f7c34d5dc70467b7f14ec67d912703568daff790
ab3f755
Status: Downloaded newer image for nginx:latest
dcb12d9827a9fdf2cca3bd716a454eb0c4858c72cc15f84d1067d9e1dfb79b0
b
```

Docker

- ❖ Docker 정보 확인
  - ✓ Docker 관련 이벤트 정보를 표시하는 명령 - docker system events
    - ● 실습
      - ◆ Docker 명령이 실행되면 이전 터미널에 이벤트 로그 기록됨

```
(base) adam@ADAMui-MacBookPro ~ % docker system events
2023-10-02T12:00:27.050910750+09:00 image pull nginx:latest
(maintainer=NGINX Docker Maintainers <docker-maint@nginx.com>,
name=nginx)
2023-10-02T12:00:27.170213334+09:00 container create
dcb12d9827a9fdf2cca3bd716a454eb0c4858c72cc15f84d1067d9e1dfb79b0
b (image=nginx, maintainer=NGINX Docker Maintainers <docker-
maint@nginx.com>, name=webapp)
2023-10-02T12:00:27.198475500+09:00 network connect
4d022fe950e08f3c460f6e9b2833454119593efcbf3dcbf5c674351673608335
(container=dcb12d9827a9fdf2cca3bd716a454eb0c4858c72cc15f84d1067d9
e1dfb79b0b, name=bridge, type=bridge)
2023-10-02T12:00:27.306766376+09:00 container start
dcb12d9827a9fdf2cca3bd716a454eb0c4858c72cc15f84d1067d9e1dfb79b0
b (image=nginx, maintainer=NGINX Docker Maintainers <docker-
maint@nginx.com>, name=webapp)
```

Docker

- ❖ Docker 정보 확인
  - ✓ Docker 관련 이벤트 정보를 표시하는 명령 - docker system events
    - ● 실습
      - ◆ 웹 애플리케이션 중지: docker stop webapp
      - ◆ 이전 터미널에 로그 기록됨

2023-10-02T12:03:59.128819918+09:00 container die dcb12d9827a9fdf2cca3bd716a454eb0c4858c72cc15f84d1067d9e1dfb79b0b (execDuration=211, exitCode=0, image=nginx, maintainer=NGINX Docker Maintainers <docker-maint@nginx.com>, name=webapp)

Docker

- Docker 정보 확인
  - ✓ Docker 관련 이벤트 정보를 표시하는 명령 - docker system events
    - filter 옵션
      - docker system events --filter 'type=image'
      - docker system events --filter 'event=stop'
      - docker system events --filter 'container=webapp'
      - docker system events --filter 'container=webapp' --filter 'event=stop'
    - 지난 24시간 동안의 로그를 출력.
      - docker system events --since 24h
    - JSON 형식으로 로그 출력.
      - docker system events --format '{{json .}}'

Docker

- 컨테이너를 실행하자 마자 터미널을 이용해서 접속
  - ✓ -it 옵션

    docker container run -it ubuntu
  - ✓ 쉘 명령 수행(hostname, date)

    / # hostname

    29b4f4344443

    / # date

    Thu Oct 10 04:34:28 UTC 2024

Docker

- 컨테이너에 터미널을 이용해서 접속
  - ✓ 현재 실행 중 인 컨테이너 확인 – 새로운 터미널을 열어서 수행

```
adam@itstudy:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
PORTS                                  NAMES
e3869d0ed28c   ubuntu    "/bin/bash"              2 minutes ago   Up 2 minutes
elastic_davinci
46aebd1eb28f   nginx     "/docker-entrypoint...."   4 minutes ago   Up 3 minutes
0.0.0.0:80->80/tcp, [::]:80->80/tcp   webapp

adam@itstudy:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
PORTS                                  NAMES
e3869d0ed28c   ubuntu    "/bin/bash"              2 minutes ago   Up 2 minutes
elastic_davinci
46aebd1eb28f   nginx     "/docker-entrypoint...."   4 minutes ago   Up 3 minutes
0.0.0.0:80->80/tcp, [::]:80->80/tcp   webapp
```

Docker

- 컨테이너에 터미널을 이용해서 접속
  - ✓ 컨테이너에서 실행 중인 프로세스 확인: docker container top 컨테이너ID

```
adam@itstudy:~$ docker container top 29b4f4344443
UID                 PID                 PPID                C                   STIME               TTY
TIME                CMD
root                18286               18265               0                   04:33               pts/0
00:00:00            /bin/sh
```

Docker

- 컨테이너에 터미널을 이용해서 접속
  - ✓ 컨테이너의 상세 정보 확인: docker container inspect 컨테이너ID

    adam@itstudy:~$ docker container inspect 29b4f4344443

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	18286	18265	0	04:33	pts/0	00:00:00	/bin/sh

Docker

- 컨테이너에 터미널을 이용해서 접속
  - ✓ 컨테이너의 로그 정보 확인 확인: docker container logs 컨테이너ID

    adam@itstudy:~$ docker container logs 29b4f4344443

    / # 호스트name

    29b4f4344443

    / # date

    Thu Oct 10 04:34:28 UTC 2024

Docker

- 컨테이너에 터미널을 이용해서 접속
  - ✓ 도커 초기화
    - 모든 컨테이너 종료 및 삭제: docker container rm -f $(docker container ls -aq)
    - 도커 이미지 삭제: docker image rm $(docker image ls -aq)