1장

Docker 이해 및 설치

전체 내용

Docker란?

Docker 설치하기

Docker 도움말 활용하기

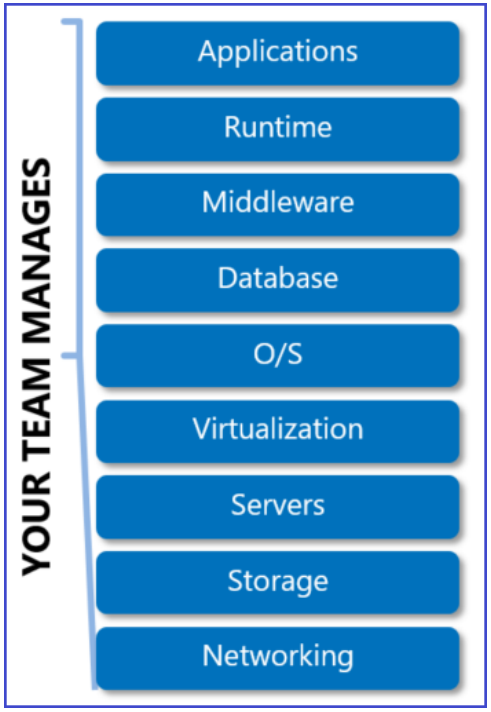
쉬어 가는 코너

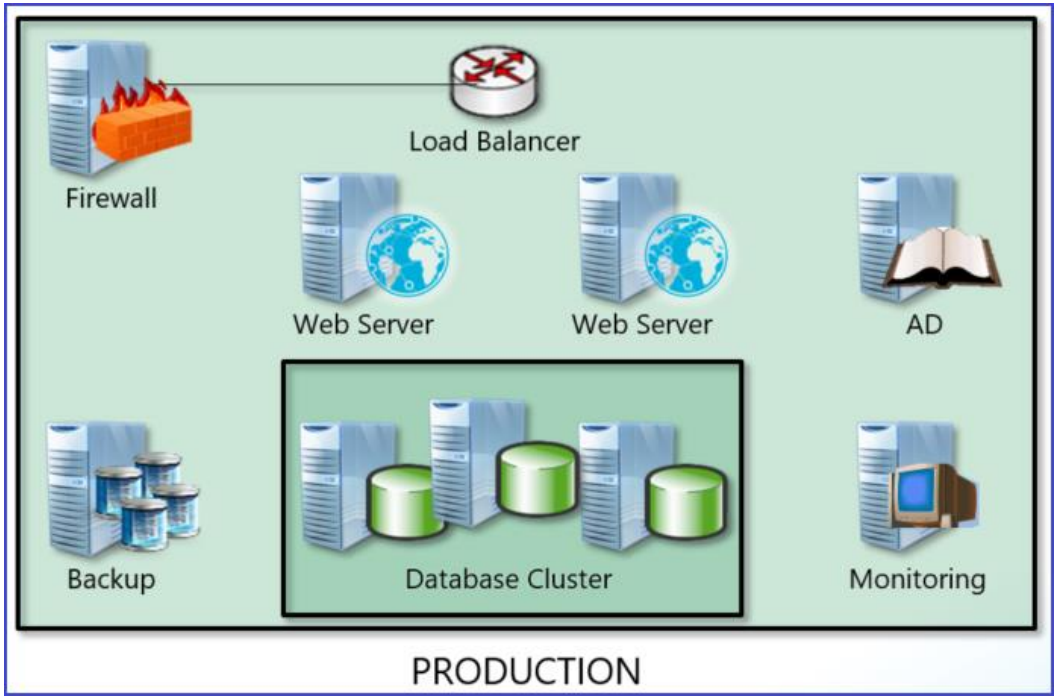
1-Docker란?

- Application으로 서비스하는 것이 우리의 목표
- Container란?
- Docker란?
- Docker의 필요성

1-Docker란?

- Application으로 서비스하는 것이 우리의 목표
  - 회사 업무를 지원하기 위해서 이메일, 포탈, 협업, 제조 시스템, 고객관리시스템을 준비하기 위해 Database server, Web server가 필요하다
  - 이를 위해서 Networking, Storage, Server H/W, Server S/W(OS), Virtualization 등등의 Infrastructure가 추가적으로 필요하다



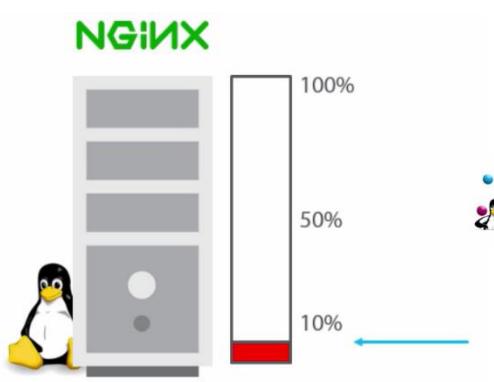


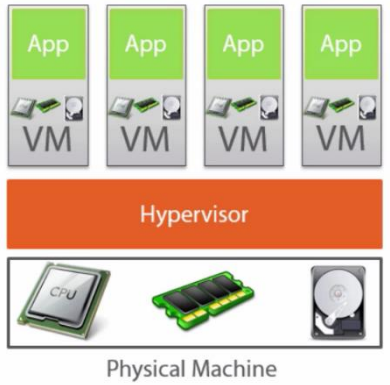
1-Docker란?

- Application으로 서비스하는 것이 우리의 목표
  - 이렇게 구성하기 위해서는 다음과 같은 문제점들이 있다
    - 서버들을 확장하기 위해 전용 H/W 및 S/W **필요**
    - 추가적으로 여러 가지 환경(Prod, Staging, Test, Dev) **필요**
    - 서버 이용률이 매우 낮음(10~20%)
      - 한 서버에 하나의 서비스만 운영하는 것이 일반적이다
    - 서버 확장을 위해서는 모든 것이 준비되어 있어야 하며, 또한 서비스의 일시 중지가 요구되므로 유연하지 못함

1-Docker란?

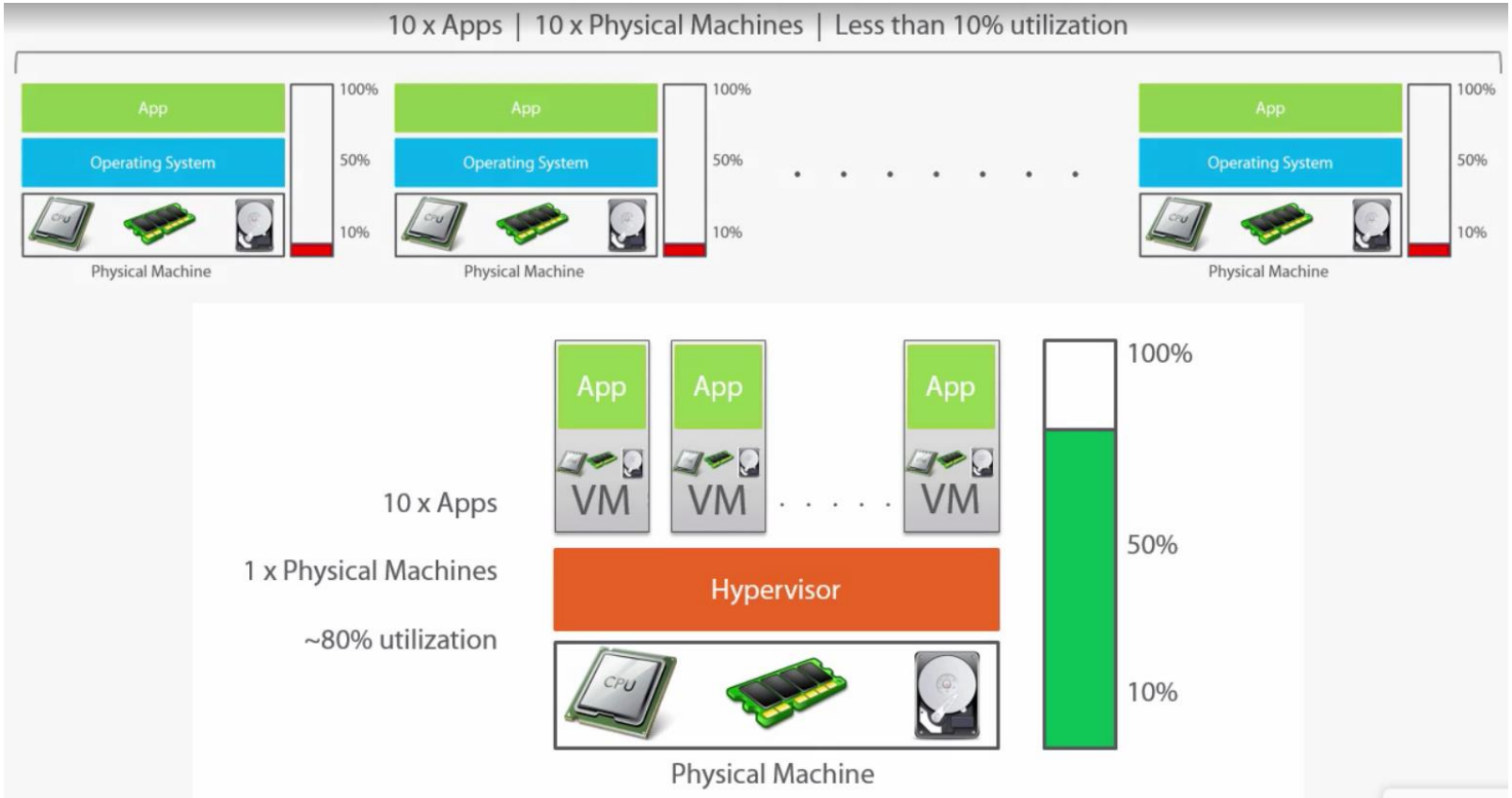
- Application으로 서비스하는 것이 우리의 목표
  - 이것을 해결한 것이 바로 **하나의 물리적인 Server에 여러 개의 Virtual Machine을 운영하는 것**이다
  - 다양한 종류의 Hypervisor 기술을 사용하여 다양한 OS에 다양한 Application(=Service)를 제공하여 **자원 활용률을 증가시킨다**





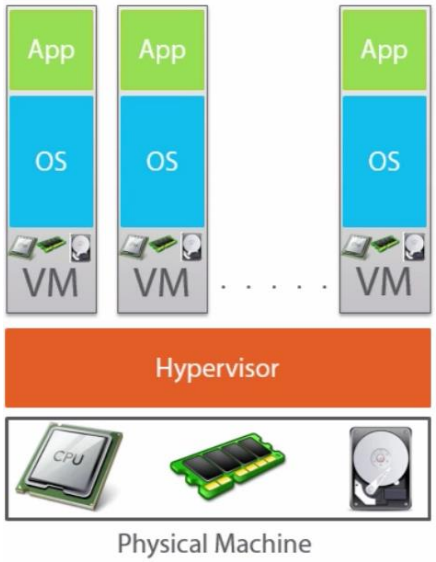
1-Docker란?

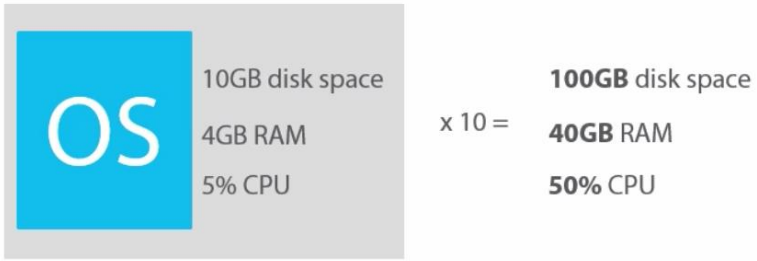
- Application으로 서비스하는 것이 우리의 목표
  - 10% 정도의 자원 활용률을 높이기 위해 Virtualization 기술을 사용하면 효율적으로 자원을 이용하여 서비스할 수 있다



1-Docker란?

- Application으로 서비스하는 것이 우리의 목표
  - 하지만 VM이 많아진다는 것은 **관리해야 할 Server가 많아지는 것과 동일하다**
  - VM으로 서비스를 하면
    - 물리적인 서버(Host Machine)에 비하여 **성능이 떨어진다**
    - Host Machine의 고장으로 인한 피해를 줄이기 위해 **Clustering 기술이 필요**하며, 당연히 **SAN Storage가 필요하게 되어 비용이 증대된다**
    - 이러한 VM을 효과적으로 관리하기 위해서는 **추가적인 소프트웨어 구매가 필수적이어서** 계속해서 관리 비용이 많이 드는 결과를 초래한다

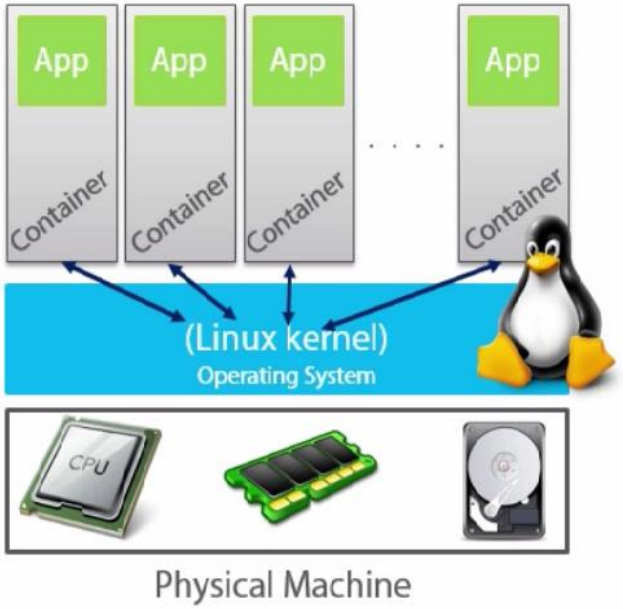




Numbers and values shown here are for illustration purposes only. Real world values will differ.

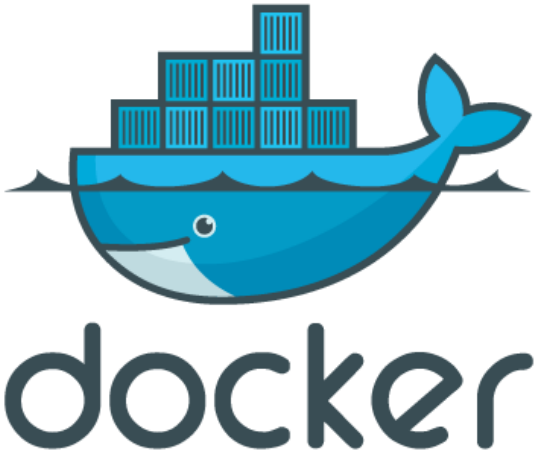
1-Docker란?

- Application으로 서비스하는 것이 우리의 목표
  - 초기 구입 및 관리 비용을 줄이면서 효과적으로 Application을 서비스하기 위해서 VM보다 가벼운 Container 기술을 이용하는 것이 Docker다
  - Linux Kernel을 공유하는 Container 기술을 사용하면 VM에 비하여 CPU, RAM, Disk를 덜 사용하게 된다
    - Web Service를 서비스할 때 VM에서 운영하는 것과 Container에서 제공할 때 CPU 사용율과 Memory 사용량을 확인해 보면 알 수 있다



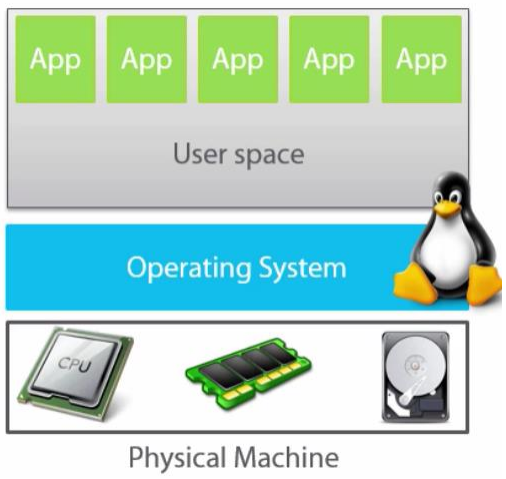
1-Docker란?

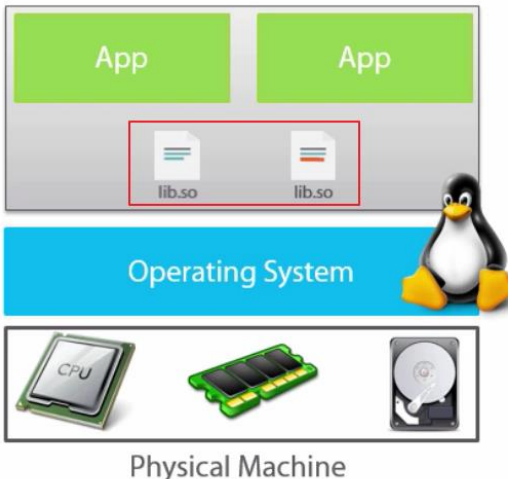
- Container란?
  - 배(ship)에 물건을 선적하여 운행할 때, 폭풍우를 만나면 물건끼리 부딪혀서 파손되기도 한다
  - 개인의 물건들을 안전하게 배송하려면 각 물건을 Container에 넣어서 Container를 Dock에 선적하여 운행하면 된다
  - Container는 서로의 영역을 침범하지 않는 독립적인 공간을 제공하는 것이다
  - Linux도 이러한 Container 기술을 사용하여 Application간의 충돌을 방지하고, 각 Container들은 공통의 자원인 CPU, RAM, Disk, Network를 사용하여 쉽고 빠르고 자원 활용률을 증대하여 서비스를 한다

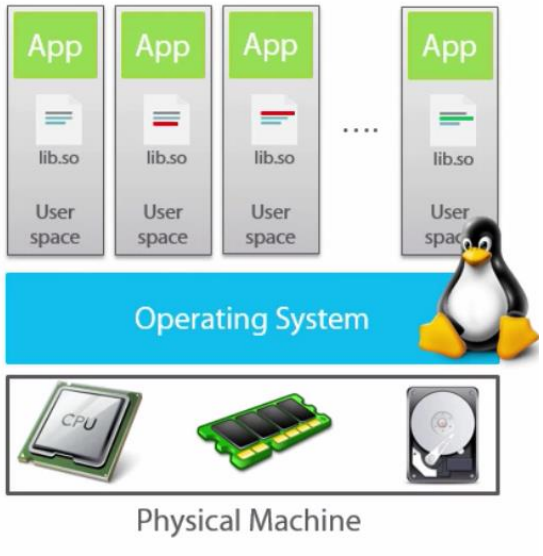


1-Docker란?

- Container란?
  - 한 운영 체제의 하나의 User space 상황에서 .dll 파일을 사용하는 여러 개의 Application이 실행되는 경우가 있다
  - 이 때 각각 Application의 이름은 동일하지만 Version이 다른 경우라면 난감하게 된다
  - 이러한 상황을 해결한 것이 바로 Application마다 서로 방해가 되지 않도록 하는 독립적인 실행 환경을 제공하면 된다
  - 즉, 각각의 Application마다 각각의 고유한 User space를 제공하여 각각에 필요한 Version의 .dll 파일을 사용하도록 하면 된다.
  - 이것이 바로 Container 기술이다

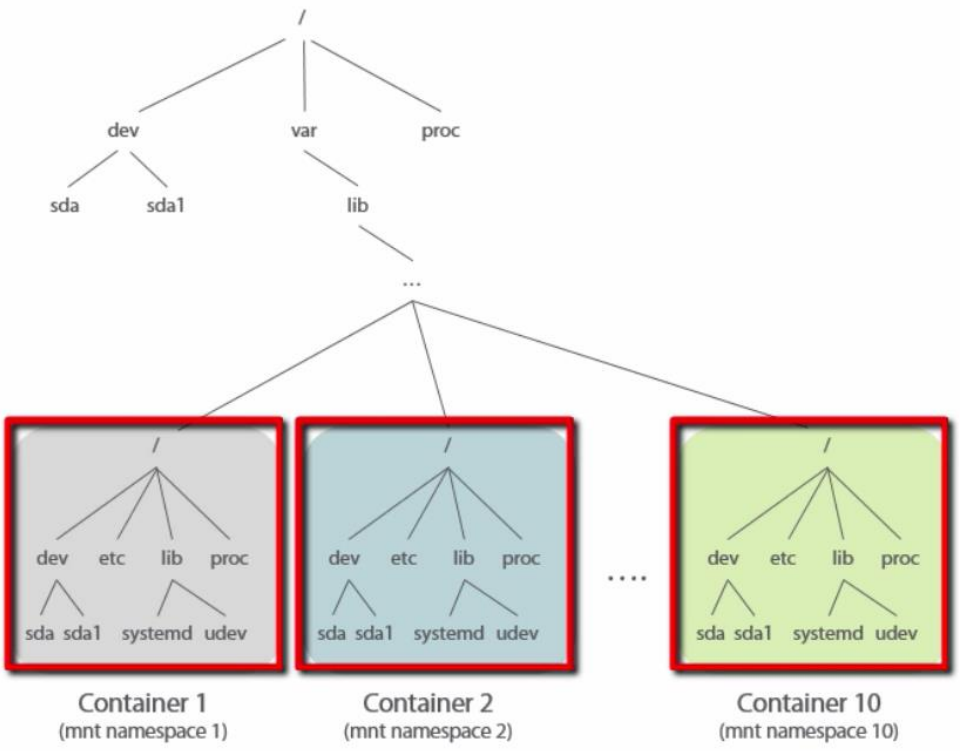


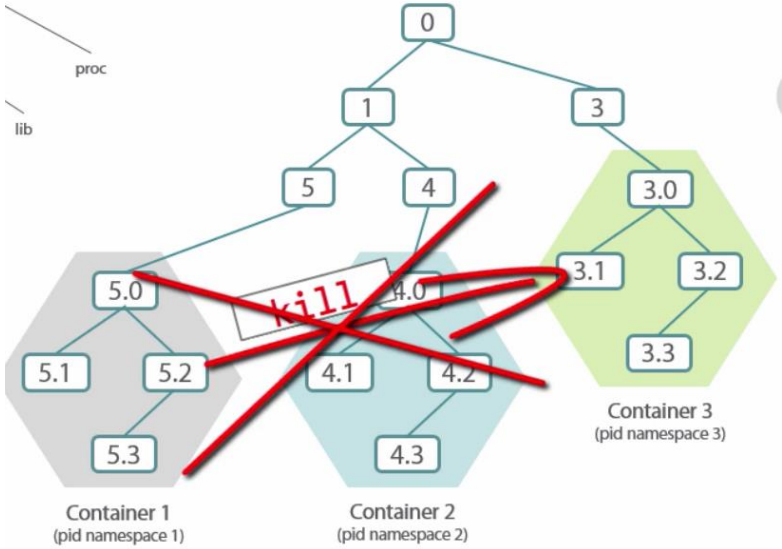




1-Docker란?

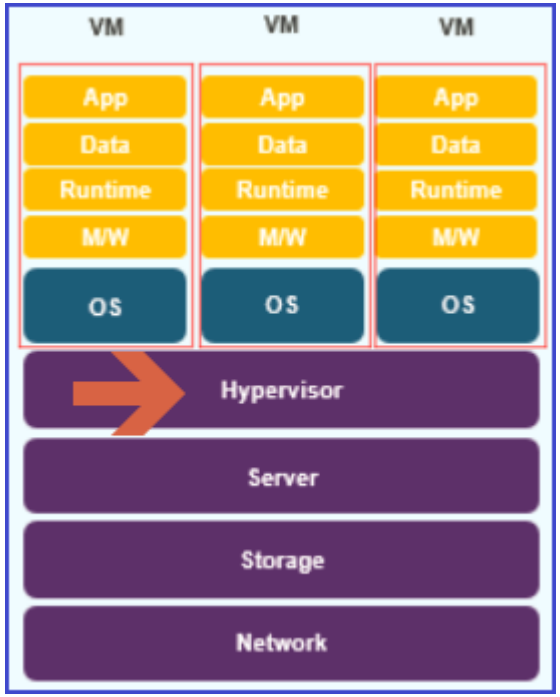
- Container란?
  - 각 Container들은 OS의 Kernel을 공유한다
  - 각 User Space(=Container) 단위로 root filesystem을 갖도록 하는 것이 container 기술이다
  - 각 Container 단위로 systemd를 갖도록 하면 process간에 kill하지 못한다

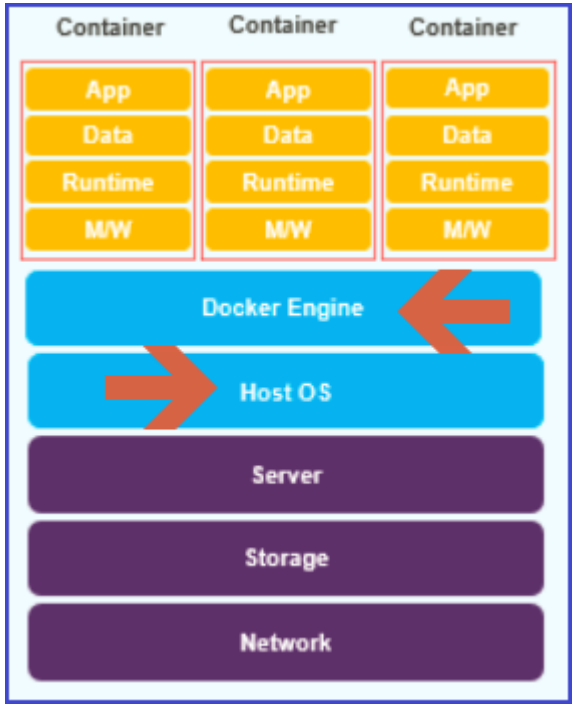




1-Docker란?

- VM과 Container 비교
  - VM
    - Hypervisor로 하드웨어를 가상화하고, Hypervisor 위에서 Guest OS가 설치된 VM들을 구동한다
  - Container
    - OS 레벨에서 CPU, RAM, Disk, Network 등의 자원을 격리하여 Container에 할당하기 때문에 Guest OS(VM)가 따로 필요 없다





1-Docker란?

- VM과 Container 비교
  - 효율성
    - 기업환경에서는 안정적인 운영을 위해, 1개의 Virtual Machine에 1개의 서비스를 구동하는 것을 권장한다. 이의 경우 VM의 모든 자원을 사용하는 것이 아니기 때문에 성능적 오버헤드가 발생한다
    - 반면 Container의 경우, OS 커널을 공유하기 때문에 자원을 필요한 만큼 효율적으로 사용할 수 있다
  - 신속성
    - 사용자의 서비스 요청량이 증가함에 따라, 기업에서는 VM이나 Container를 추가적으로 배포한다
    - VM의 크기는 최소 몇 GB이지만, Container인 경우 Guest OS가 없어 MB단위의 크기를 가진다
    - 결과적으로 VM은 배포하는데 수 분에서 수십 분의 시간이 소요되지만, Container는 배포에 소요되는 시간이 수 초에 불과하다

1-Docker란?

- VM과 Container 비교
  - 라이센스 비용 절감
    - 가상화 서버의 경우 VM의 개수만큼 Guest OS의 라이센스 비용이 발생한다
    - 반면에 Container인 경우 Host OS 1대의 라이센스 비용만 발생한다. 만약 서버의 수가 많아진다면 비용의 차이도 기하급수적으로 증가할 것이다
  - 안정성
    - VM의 경우 정확히 할당된 자원 내에서 VM이 운영되기 때문에, Container에 비해 안정적으로 운영할 수 있다
    - 반면에 Container들은 OS 커널을 공유하기 때문에, 하나의 Container가 무리하게 자원을 사용하게 될 수 있다
    - 이를 해결하기 위해서 실행할 Container에 CPU, Memory와 같은 자원 할당량을 사전에 지정시킬 수 있다
    - 또는 Orchestration을 하는 Docker Swarm, Kubernetes으로 해결할 수 있다

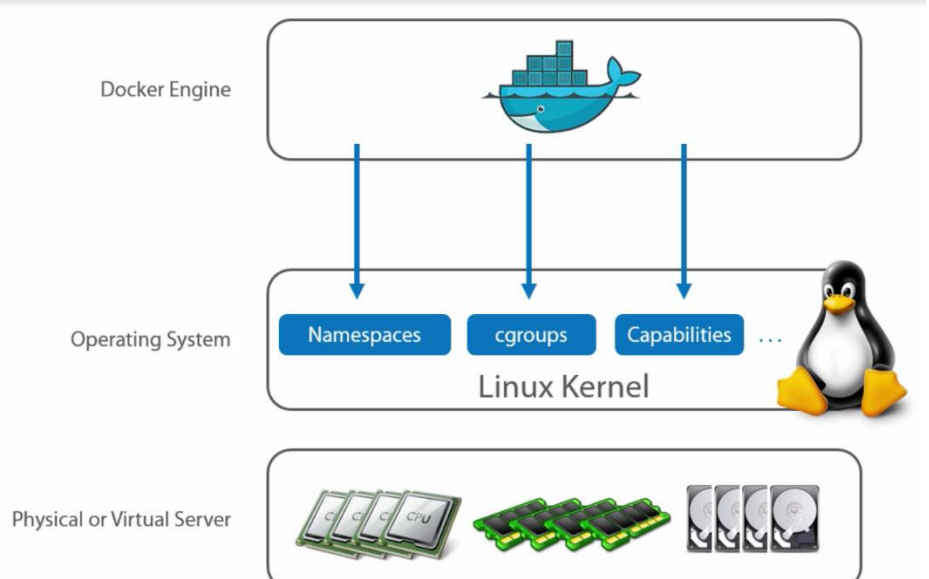
VM과 Docker 비교하기

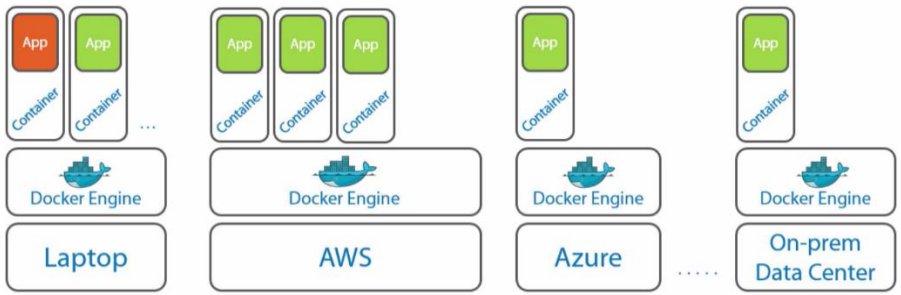
- VM vs. Docker

조건	VM	Docker
메모리 필요량	많은 메모리 양 필요	적은 메모리 양 요구
부팅 시간	오랜 부팅 시간	부팅 시간이 엄청 빠름
성능	많은 VM 실행으로 불안정한 성능	성능 좋음
확장 여부	Scale out 어려움	Scale out하기 쉬움
효율성	효율성이 낮음	효율성이 높음
이동성	다른 Platform으로 이동하면 호환성 문제 발생	다른 Platform으로 이동 가능
공간 할당	데이터 볼륨의 공유가 어려움	데이터 볼륨이 공유되고, 다른 컨테이너도 같이 사용할 수 있다

1-Docker란?

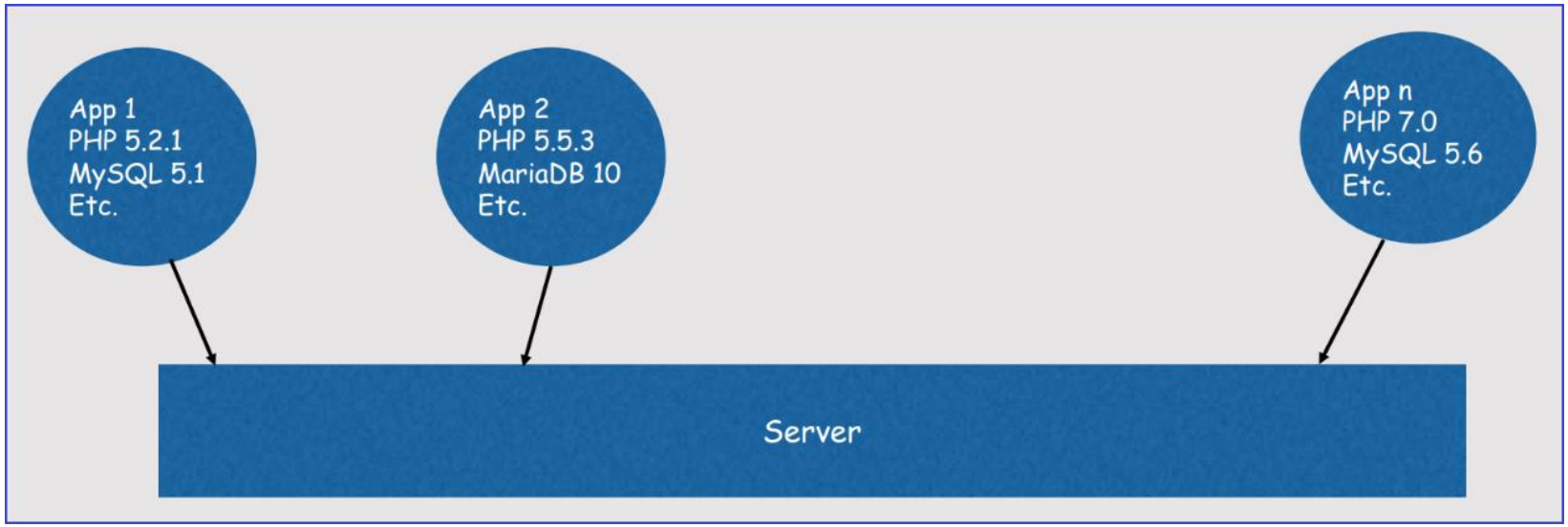
- Docker란?
  - 2013년도에 Docker Inc.에서 출시하였고, Go Language로 작성되었다
  - Docker는 Linux Container에 여러 기능을 추가하여 Application을 Container 형식으로 쉽게 사용할 수 있는 Open Source Project다
  - Docker는 VM에 비해 **성능 손실이 거의 없고**, 상대적으로 **가볍고**, 어디서든 실행할 수 있고**(이식성)**, 이미지를 **중앙에 저장하여** 편리하게 사용할 수 있다
  - Public Cloud에서 운영하는 수 많은 VM에 Application을 설치하고 구성하는 문제를 해결한 것이 Docker 기술이다





1-Docker란?

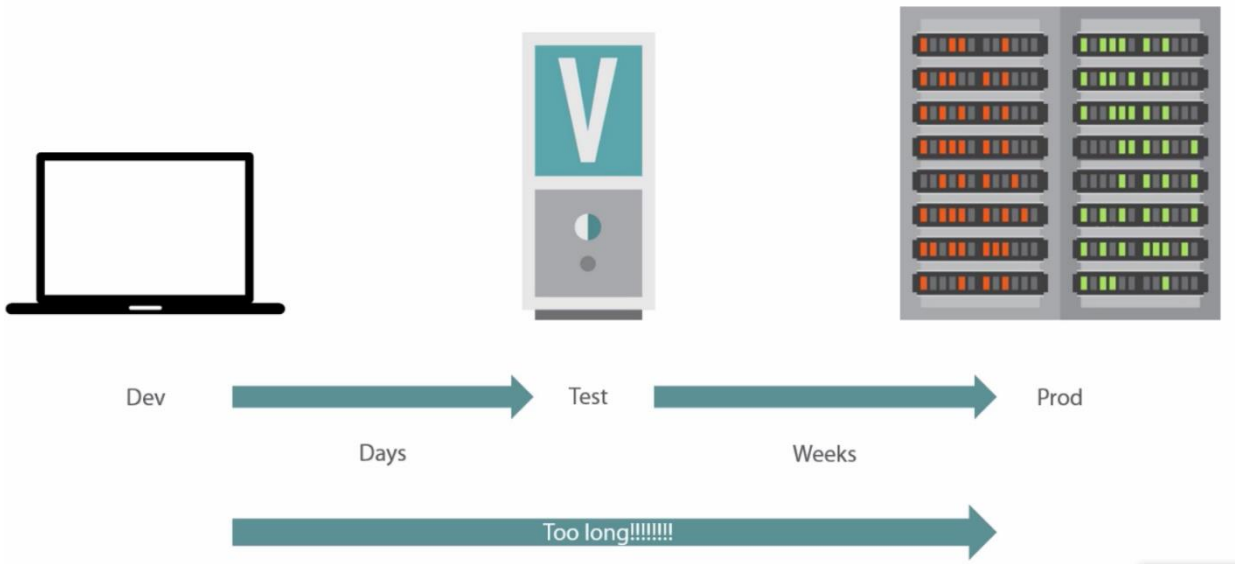
- 개발 부분에서 Docker의 필요성
  - 하나의 서버에 여러 개의 Project 운영하기
    - PHP Web Application을 개발을 하여 운영을 할 때, **한 서버에 여러 개의 Web site를 운영할 수 없다**. 그 이유는 각 Project마다 다른 버전의 PHP를 사용하고, Database 버전도 다른 것을 사용하기 때문이다



    - 이것을 해결하기 위해 Host 컴퓨터에 여러 개의 VM을 설치하여 운영할 수도 있지만, VM 운영을 무겁고, 많은 자원을 사용하므로 성능 부분에 문제가 있다

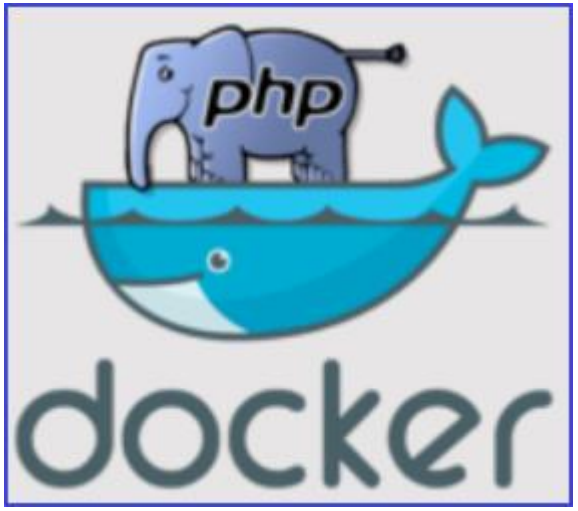
1-Docker란?

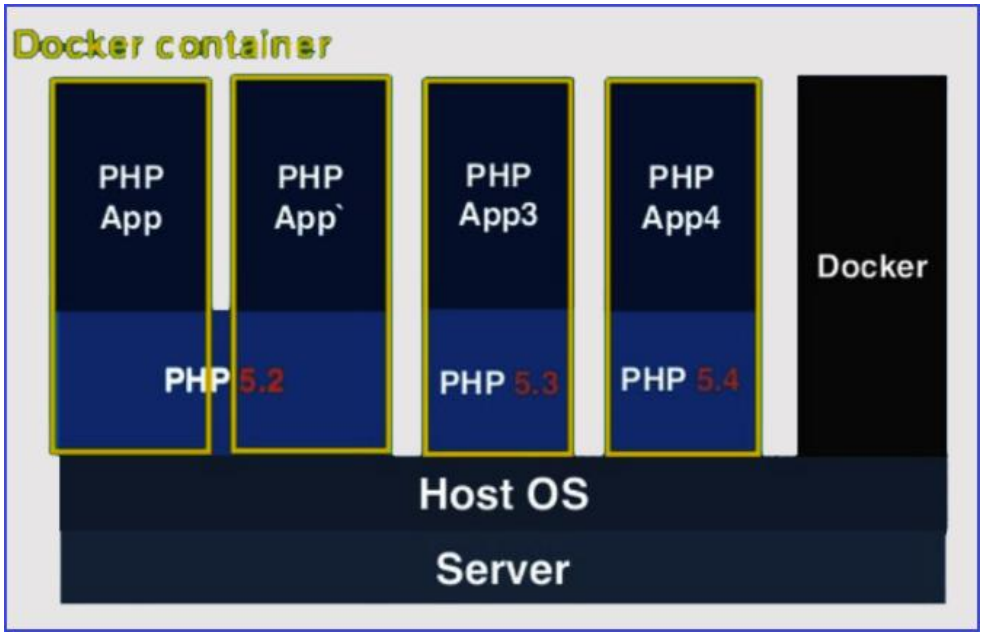
- 개발 부분에서 Docker의 필요성
  - **“개발, 테스트, 서비스 환경을 하나로 통일”**하여 효율적으로 관리할 필요가 있다
    - 개발 환경(개발자 노트북)에서 OS, Middleware를 설치하고 개발한 프로그램을 개발자가 설치 및 운영을 한 후 다음 담당자에게 넘긴다
    - 테스트 환경(회사 서버 컴퓨터)에서도 동일하게 작업을 하여 테스트를 한다
    - 실제 운영 서버에 개발한 프로그램을 설치하여 운영할 때 정상적으로 안되는 경우가 종종 있다



1-Docker란?

- 개발 부분에서 Docker를 사용하여 문제점 해결하기
  - docker는 각각 필요한 software에 호환되는 환경을 구축할 수 있다
  - docker는 각 Project별로 disk 공간을 다르게 사용할 수 있다
  - 필요한 software도 쉽게 구현할 수 있다
  - 필요한 software를 쉽게 배포 및 이식할 수 있다
    - 회사 서버에서 제작한 Docker 이미지를 AWS, Azure 등에서도 동일하게 서비스할 수 있다 (Docker 이식성)





1-Docker란?

- 운영 부분에서 Docker의 필요성
  - 부하가 많이 걸리는 대규모 Web Service를 제공할 때 하나의 물리적 서버로서는 한계가 있다
  - 이를 해결하기 위해 Cloud Infra를 사용할 수도 있다
  - 하지만 간단하게 docker swarm(clustering 기술)을 사용하여 동적으로 대처를 할 수 있다

2-Docker 설치하기

Docker 설치하기

Docker 정보 확인하기

Docker 관리자 추가하기

2-Docker 설치하기

- Docker 설치하기
  - CentOS 7에 Docker 설치하기
    - **curl** -sSL http://**get.docker.com | sh**
  - Docker 서비스의 자동 시작 구성 여부 확인하기
    - systemctl **list-unit-files** | grep **docker**
  - 시스템이 시작할 때 자동으로 Docker 서비스 시작하기
    - systemctl **enable** docker
  - Docker 서비스 시작하기
    - systemctl **start** docker
  - Docker 서비스 실행 여부 확인하기
    - systemctl **status** docker

systemctl enable docker --now

2-Docker 설치하기

- Docker 정보 확인하기
  - 실행된 Docker 서버 및 클라이언트 버전 확인하기
    - **docker version**
  - Docker 세부 정보 확인하기
    - **docker info**

```
[root@centos1 ~]# docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 17.06.0-ce
```

    - 다운로드 받은 docker image도 없고, 실행된 docker container도 없다
    - Docker의 **Root Directory 내용 확인**
      - ls -l **/var/lib/docker**

```
Total Memory: 1.797GiB
Name: centos1
ID: A2IL:AYXO:IDI2:K3Z2:UTB6:FHV4
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
```

```
[root@centos1 docker]# ls -l /var/lib/docker
total 0
drwx------. 2 root root  6 Aug 14 21:30 containers
drwx------. 4 root root 40 Aug 14 21:30 devicemapper
drwx------. 3 root root 25 Aug 14 21:30 image
drwxr-x---. 3 root root 18 Aug 14 21:30 network
```

3-Docker 도움말 활용하기

- docker help 사용하기
- 하위 메뉴가 “있는” 명령어 도움말 사용하기
- 하위 메뉴가 “없는” 명령어 도움말 사용하기

3-Docker 도움말 활용하기

- docker help 사용하기
  - docker 명령어를 모두 외울 수가 없으므로 도움말을 잘 활용해야 한다
  - **docker help** (=docker --help)
  - Management command는 하위 메뉴가 있다

```
Management Commands:
  builder     Manage builds
  config      Manage Docker configs
  container   Manage containers
  engine      Manage the docker engine
  image       Manage images
  network     Manage networks
```

  - command는 하위 메뉴가 없다

```
Commands:
  attach      Attach local standard input, output,
  build       Build an image from a Dockerfile
  commit      Create a new image from a container'
  cp          Copy files/folders between a contain
  create      Create a new container
  diff        Inspect changes to files or director
  events      Get real time events from the server
  exec        Run a command in a running container
```

3-Docker 도움말 활용하기

- 하위 메뉴가 **있는** 명령어 도움말 사용하기
  - **docker image** --help
    - docker image **ls**
    - docker image **ls --help**
    - docker image **ls --no-trunc**
    - **docker image rm** --help
    - docker image rm **alpine**
  - **docker container** --help
    - docker container **ls**
    - docker container **ls -a**
    - **docker container rm** mycon (= **docker rm** mycon)
      - ## alias가 있어서 동일한 명령어가 있을 수 있다
  - **docker system** --help
    - docker system **df**
    - docker system **info** (=docker info)
    - docker system **prune**
    - docker system prune **-a**

3-Docker 도움말 활용하기

- 하위 메뉴가 **없는** 명령어 도움말 사용하기
  - **docker run** --help
    - docker run **alpine**
    - docker run **-it --name myalpine** alpine **/bin/sh**
    - docker run **-d -p 80:80** nginx
  - **docker stats** --help
    - **docker stats -a**
      - **docker run -d -p 80:80 nginx**
      - **docker run -it alpine /bin/sh**
        - **ping www.google.com**
  - **docker rename** --help
    - docker rename ae2c89eecb95 **mycon**
      - docker ps
  - **docker exec** --help
    - docker exec -it **mycon** /bin/sh

4-쉬어 가는 코너

docker create, docker run, docker exec의 차이점

4-쉬어 가는 코너

- docker create, docker run, docker exec의 차이점
  - **docker create**
    - container를 생성만 하고 **start는 하지 않는다**
  - **docker run**
    - container를 생성한 후 **start까지 하는 것이다**
  - **docker exec**
    - **이미 실행 중인 container에** 명령어를 실행하는 것이다

  - 각 명령어의 도움말을 참고한다
    - docker create **--help**
    - docker run **--help**
    - docker exec **--help**

4-쉬어 가는 코너

- docker create, docker run, docker exec의 차이점
  - 예제
    - **docker create** --name busybox-con1 -it **busybox** /bin/sh
      - docker ps (##container가 없음)
      - docker ps -a (##container가 중지됨)
      - **docker start** busybox-con1
      - docker ps (##container가 실행되고 있음)
    - **docker run** --name busybox-con2 -it **busybox** /bin/sh
      - container가 실행되어 /bin/sh에 접속함
      - ping www.google.com -c 3
      - exit
      - docker ps (##busybox-con1만 실행중임)
    - **docker exec** busybox-con1 **mkdir /yslee**
    - **docker exec -it** busybox-con1 **/bin/sh**