VIRTUALIZATION & DOCKER

191224 / nenw*
@ SPARCS Wheel Seminar

Table-of-Contents

Objective

완전히 이해하는 정도

> **적당히 개념만 잡고 필요 시 구글링이 가능한 정도**

들어만 본 정도

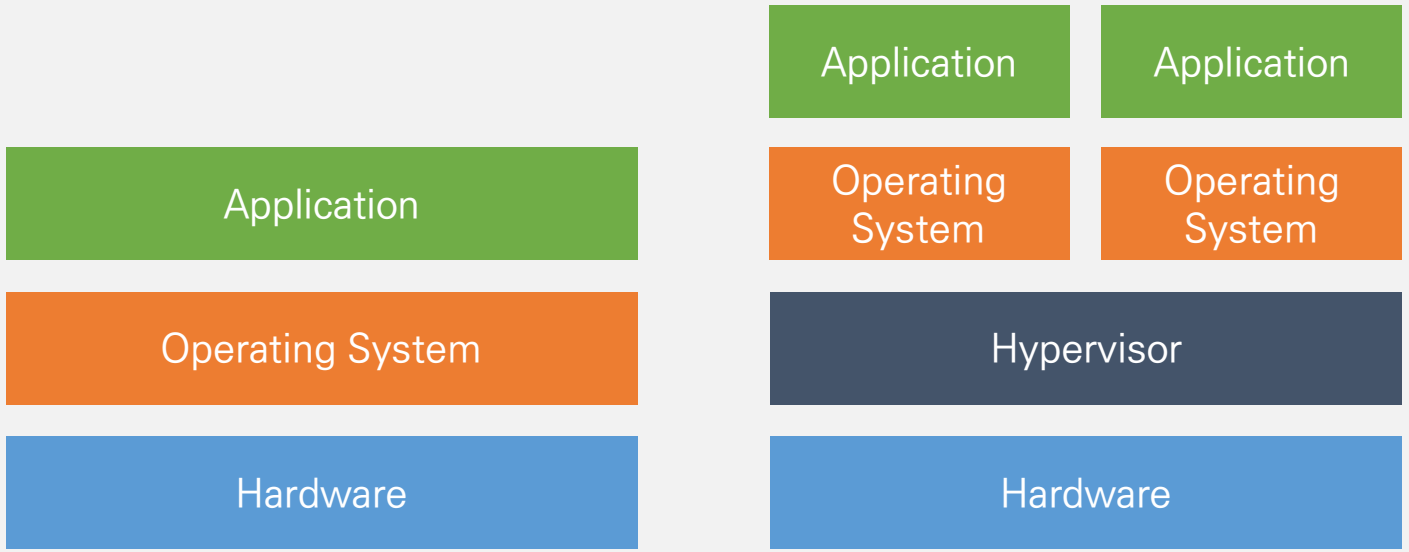
01

VIRTUALIZATION

가상화

» 리소스를 추상화 (큰 의미)
  - 컴퓨팅, 네트워킹, 스토리지, …
  - 물리적인 자원들을 숨기고 논리적인 자원을 만들어내는 것

» 하나의 컴퓨터로 여러 개의 컴퓨터를 사용 (매우 좁은 의미)



w/o Virtualization

w/ Virtualization

가상화를 사용하는 이유

» 자원의 효율적 사용

» 상황에 유연하게 대처할 수 있음

» 캡슐화 / 격리, 획일화된 환경

전가상화

» 완전히 가상화, 언제나 개입

» OS는 하이퍼바이저를 모름

» 모든 명령을 중재하기 때문에 비교적 느림
  - Dom0

반가상화

» OS를 적당히 수정해서 하이퍼바이저에게 직접 요청 (Hyper Call)

» 게스트 OS가 오픈 소스가 아니라면...

가상화 솔루션

- Xen (garam에서 사용중임)
- ESXi
- KVM
- 등등….

02

DOCKER

OS-Level Virtualization

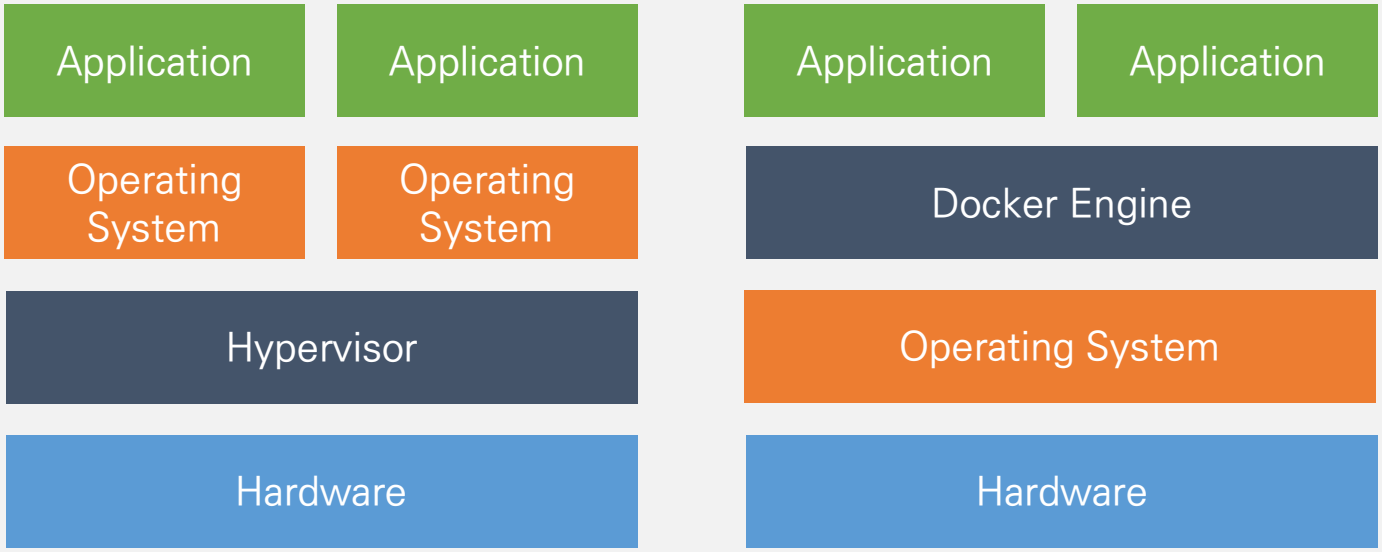
» OS까지는 같이 쓰고 그 위에서 가상화를 하자!

» Linux: cgroups, namespace
- cgroups: CPU, 메모리, 디스크 I/O 등 격리
- namespace: PID, hostname, 네트워크, IPC, 파일시스템 등 격리

Docker

» OS-Level Virtualization 플랫폼

» “컨테이너”
- 직육면체의 규격화된 사이즈
- 화물이 들어간다


Application
Application
Operating System
Operating System
Hypervisor
Hardware
Application
Application
Docker Engine
Operating System
Hardware

장점

» 가볍다!
- 퍼포먼스가 좋음
- 전체 OS를 깔 필요가 없어 비효율을 줄일 수 있음

» DockerHub에서 이미지를 가져올 수 있음

» 그 외에도 수많은 장점들이…
- 빌드용으로 도커를 쓰면 디펜던시 관리로 인한 탈모를 예방할 수 있다

Image / Container

» Image
- 시스템의 스냅샷
- 실행 파일, 라이브러리, 기타 파일 등을 하나로 묶은 것
- 컨테이너 실행에 필요한 것들

» Container
- 이미지를 실행시킨 상태

Exercise

» Docker 설치
  - https://docs.docker.com/install/

» `$ docker search ubuntu`
  - 이미지를 검색

» `$ docker pull ubuntu:bionic`
  - Bionic(18.04) 버전의 ‘ubuntu’ 이미지를 다운로드

Exercise

» `$ docker run -i -t -d --name hello ubuntu:bionic /bin/bash`
- hello라는 이름으로 ubuntu 컨테이너를 만들고 /bin/bash 실행
- i: interactive, t: tty, d: daemon

» `$ docker ps -a`
- 모든 컨테이너 목록 출력

» `$ docker attach hello`
- hello에 stdio 연결 (ctrl + p 〉 ctrl + q 로 detach)

Exercise

» `$ docker exec -i -t hello /bin/bash`
  - hello에서 /bin/bash를 실행하고 stdio 연결

Image 생성 - Dockerfile

» Dockerfile: 이미지를 생성하기 위한 파일
- FROM (이미지명) : 특정 이미지로부터 시작
- COPY (소스) (타겟) : 폴더/파일을 소스에서 파일로 복사
- RUN (명령어) : 명령어를 실행
- WORKDIR (폴더) : 특정 폴더로 이동
- ENV (키)=(값) : 환경변수를 설정
- ENTRYPOINT [“echo”, “hello”] : 컨테이너 구동 시 echo hello 실행

» `$ docker build --tag 이름:버전 .`
- 이름:버전으로 도커 이미지 생성

Image 생성 – commit

» `$ docker commit (컨테이너) (이미지)`
  - 변경 사항을 이미지로 저장

DockerHub

» DockerHub : 도커 이미지들이 올라오는 곳
  - https://hub.docker.com

» 매우 많은 이미지들이 미리 올라와있다!

Exercise

```
»   $ cat Dockerfile
FROM node:current-alpine

WORKDIR /usr/src/app
COPY package*.json ./

RUN npm install
COPY . .

EXPOSE 3008
ENTRYPOINT ["npm", "start"]

»   $ docker build --tag slink-app:1.0 .
```

Union Mounting

» Union Mounting
- Readonly 파일시스템들의 Layer를 쌓는 느낌
- 그러나 하나의 파일시스템처럼 보이게 됨
- ex) Alpine Linux + nodejs + …

Volume, Bind

» 컨테이너가 삭제될 때 기본적으로 데이터도 삭제됨
- 따라서 특정 디렉토리를 호스트에 저장하는 옵션을 제공함 (-v)

» Volume
- 도커가 관리하는 볼륨이 생성돼 데이터가 저장됨
- `/var/lib/docker/volume/` 에 존재
- `$ docker volume`

» Bind
- 그 외 호스트의 어딘가를 컨테이너에 마운트함

Networking

» bridge
  - 격리된 네트워크, docker0 브릿지 (기본)
  - expose : 포트를 노출 (-p 8080:80 : 8080포트를 80포트로 노출)

» host (--net=host)
  - 호스트와 같은 네트워크 사용

» container (--net=container:(hash))
  - 다른 컨테이너와 같은 네트워크 사용

» none
  - 격리만 된 상태

Reboot Policy

» 자동으로 컨테이너를 실행할 수 있음 (--restart)

» no
- 자동 재시작을 사용하지 않음

» on-failure
- 오류로 죽으면 재시작

» always
- 그냥 무조건 재시작 (직접 죽이면 도커 재시작시 같이 재시작)

» unless-stopped
- always랑 같지만 정지되면 도커 재시작해도 같이 재시작 X

03

DOCKER COMPOSE

Background

» 도커 아규먼트가 주렁주렁 달림

» 컨테이너 간의 의존성이 생김

» 설정파일을 하나 만들어서 그걸로 관리하자!

How To

» yaml 파일을 하나 만들어서 설정을 저장
  - docker-compose.yml

» `$ docker-compose up`
  - `$ docker-compose up -d` : 데몬으로 실행

» `$ docker-compose down`

```yaml
version: '3'
services:
        pydio:
                image: pydio/cells
                restart: always
                environment:
                        CELLS_BIND: 0.0.0.0:8080
                        CELLS_EXTERNAL: cloud.nenw.dev
                        CELLS_NO_SSL: 0
                ports:
                        - 3003:8080
                volumes:
                        - pydio:/root/.config/pydio/cells
                        - static:/root/.config/pydio/cells/static/pydio
                        - data:/root/.config/pydio/cells/data
                depends_on:
                        - db
                networks:
                        pydio:
                                aliases:
                                        - pydio

        db:
                image: mariadb
                restart: always
                environment:
                        MYSQL_ROOT_PASSWORD: [redacted]
                        MYSQL_DATABASE: pydio
                        MYSQL_USER: pydio
                        MYSQL_PASSWORD: [redacted]
                command: [mysqld, --character-set-server=utf8mb4, --collation-server=utf8mb4_unicode_ci]
                expose:
                        - "3306"
                volumes:
                        - db:/var/lib/mysql
                networks:
                        pydio:
                                aliases:
                                        - db

volumes:
        pydio:
        data:
        static:
        db:

networks:
        pydio:
```

Exercise

```
»    $ cat docker-compose.yml
version: '3'
   services:
       slink:
           image: slink-app:1.0
           restart: always
           environment:
               PORT: 8080
           volumes:
               - ./links.json:/usr/src/app/links.json
           ports:
               - 3009:8080

»    $ docker-compose up
```

그 외

» ipam 으로 IP 주소 할당을 관리할 수 있음

» Docker Swarm으로 분산 서버를 관리할 수 있음

» 도커 스웜 대신에 Kubernetes를 이용할 수도 있음

Q & A

THANKS FOR LISTENING