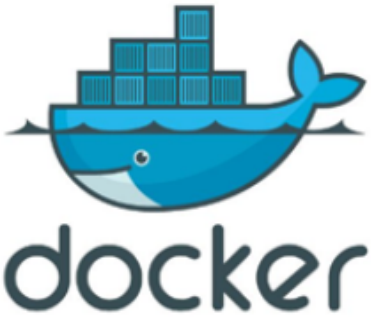
도커와 쿠버네티스

도커(Docker)와 쿠버네티스(kubernetes)

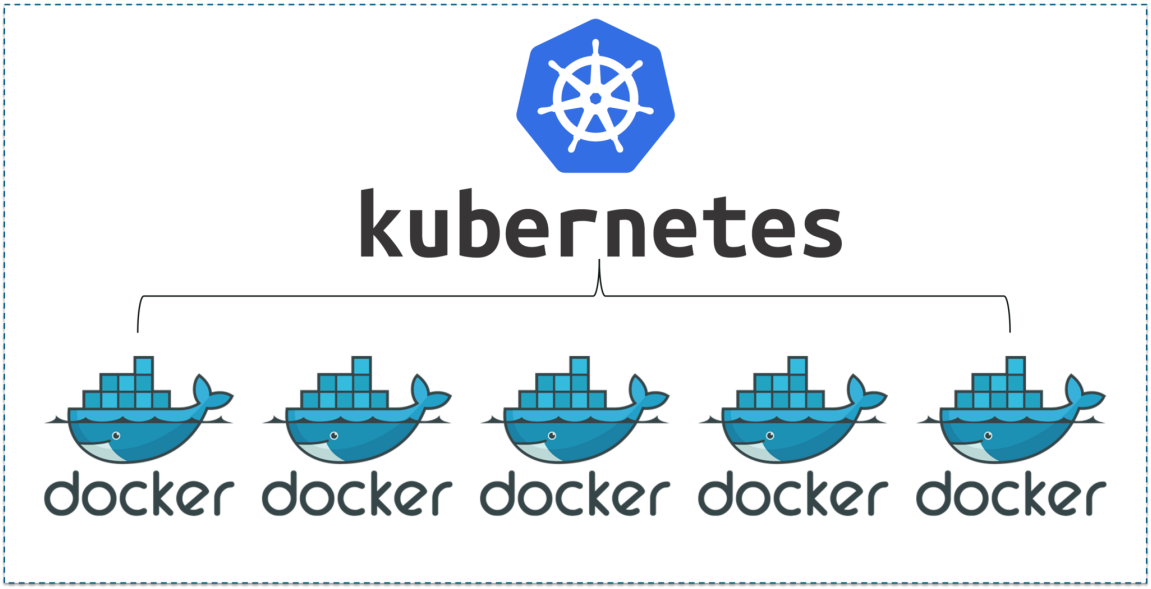


도커란 “여러 컨테이너를 관리/예약하는 플랫폼”

= 기술적인 개념이자 도구

쿠버네티스란 “여러 컨테이너를 관리 예약하는 도구”

= 이러한 도커를 관리하는 툴



컨테이너와 이미지

이미지(image)

: 필요한 프로그램과 라이브러리, 소스를 설치한 뒤 만든 하나의 파일

ex. 윈도우를 사용할 때 필요한 것이 윈도우 “이미지” (=윈도우 설치파일)

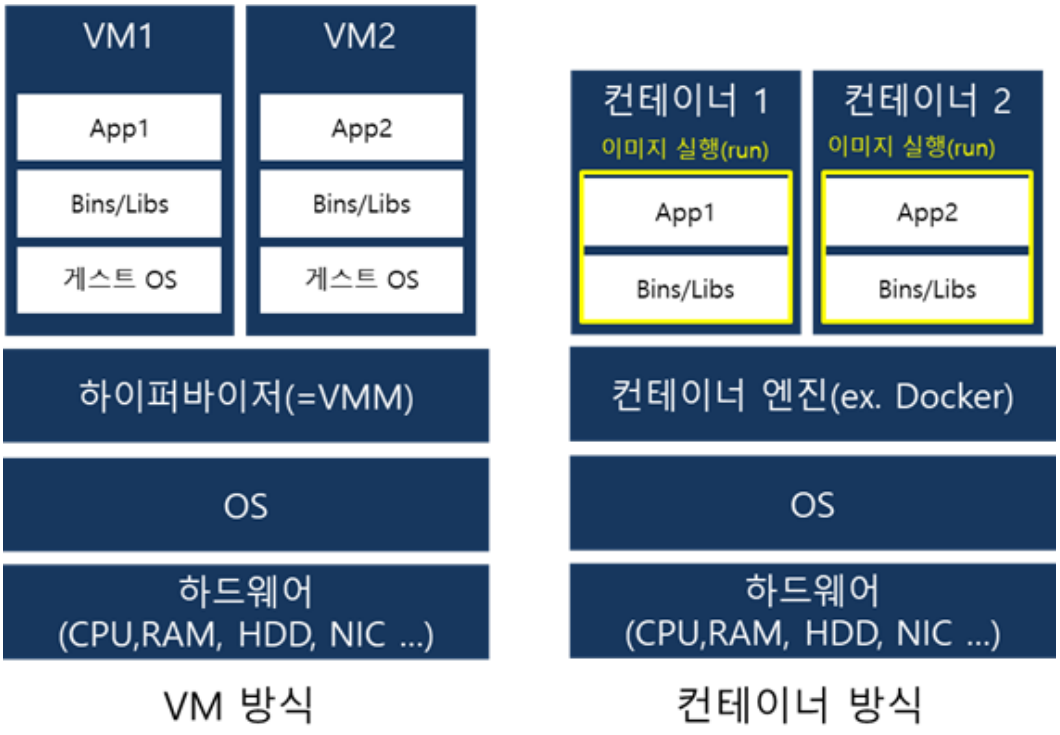
컨테이너(container)

: 애플리케이션을 구성하는 다양한 프로그램과 실행환경을 박스로 묶어 추상화한 개념

: 도커는애플리케이션의 실행에 필요한 라이브러리와 바이너리, 기타 구성 파일을 ‘이미지’ 단위로 빌드하여 패키지로 배포

: 이미지를 격리하여 독립된 공간에서 실행한 가상환경

= 이미지를 받고 설치한 것이 “컨테이너”



: 특징

1) 실행에 필요한 모든 환경이 준비되어 있으므로 어떤 환경에서도 애플리케이션을 오류 없이

동작시킬 수 있다

2) 하이퍼바이저와 게스트 OS를 필요로 하지 않아 가볍다 (민첩성)

→ 일반적으로 컨테이너에는 OS가 포함 X, 크기가 가볍고 때문에 컨테이너 복제와 배포가 용이함

⇒ VM에 비해 애플리케이션을 실행, 배포하는 과정이 가볍기 때문에 하나의 물리 서버에서

더 많은 애플리케이션을 구동시킬 수 있다.

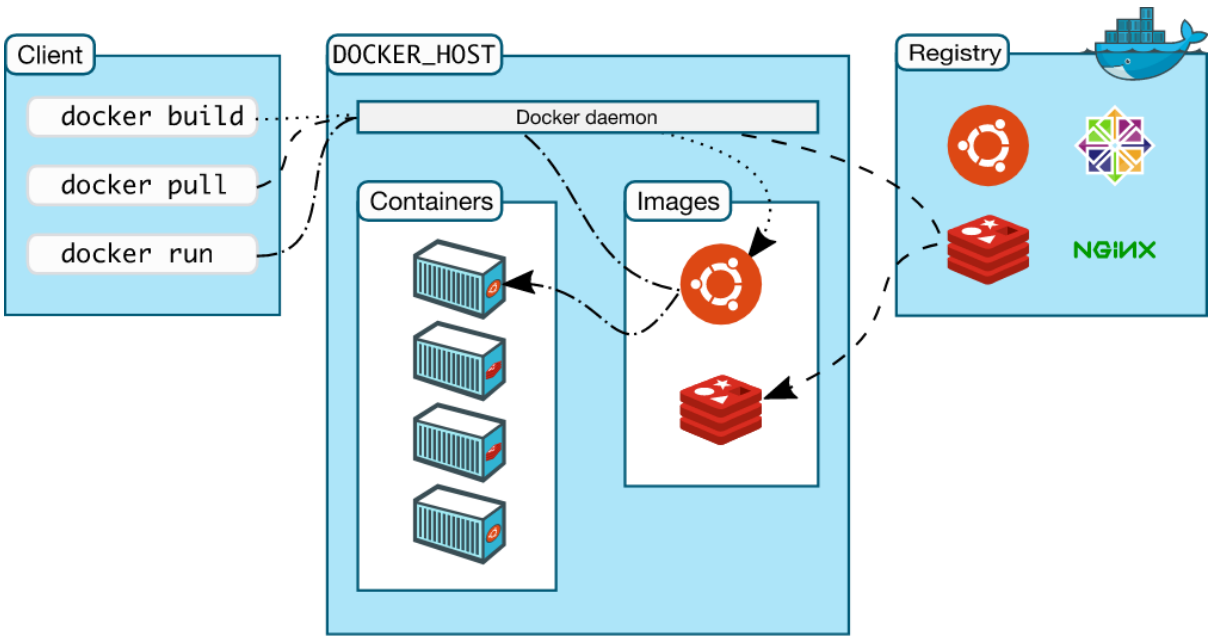
도커란?

: 리눅스 컨테이너를 만들고, 사용할 수 있는 **컨테이너화 기술**

: 애플리케이션에 국한되지 않고 의존성 및 파일 시스템까지 패키징하여 빌드, 배포, 실행을 단순화

→ 컨테이너를 실행하기 위한 모든 정보를 이미지화 하여 컨테이너의 이동성과 유연성을 높였다는 점에서 기존의 컨테이너들과 차이를 가진다

: 도커의 구조&작동원리



Docker daemon - 클라이언트의 명령을 REST API로 받아

컨테이너, 이미지, 네트워크 그리고 볼륨을 관리

*볼륨- 컨테이너에서 생성된 데이터들을 의미

*REST API

-어떤 자원에 대해 CRUD(Create, Read, Update, Delete)연산을 수행하기 위해 UR(Resource)로 요청을 보내는 것

Docker client - dockerd에 명령을 전달하기 위한 수단으로 docker 명령어를 사용하면 Docker API가 REST API형식으로 dockerd 소켓에 전달

Registry - 컨테이너의 이미지가 보관되는 곳 (=/= 리포지터리)

* 레지스트리를 리포지터리를 여러개 가지는 보관 서비스

*리포지터리는 하나의 이미지에 대해 태그를 사용하여 다양한 출시 버전을 함께 보관하는 곳

: 도커의 한계점

서비스가 커지면 커질수록 관리해야 하는 컨테이너의 양이 급격히 증가

→ 각각의 서버마다 도커를 하나씩 설치하기 때문에 도커의 내용을 확인하기 위해서 서버마다 들어가서 도커를 확인해야 함(번거로움)

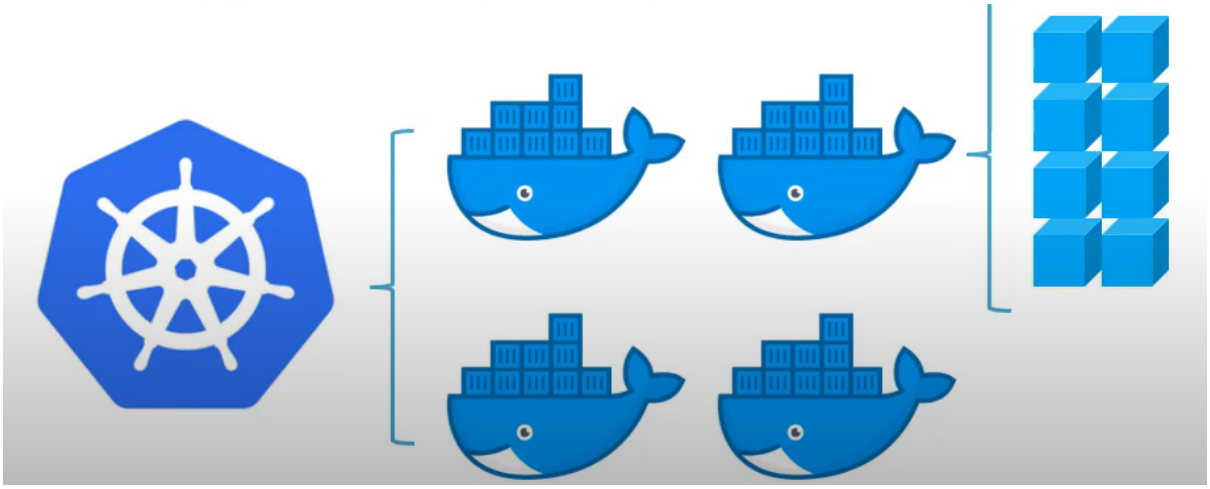
→ 관리자가 도커를 사용하여 관리를 한다고 하더라도 쉽지 않다

⇒ 이러한 문제를 해결하기 위해 오케스트레이션 툴들의 등장

*오케스트레이션(orchestration)

: 마치 오케스트라를 지휘하는 지휘자처럼 여러 컨테이너 애플리케이션을 여러 대의 호스트에 배치하고, 관리하는 작업을 자동화 하는 방법을 제공하여 적절한 배치와 구성을 관리해주는 것

쿠버네티스란?



: 위에서 언급한 문제를 해결하기 위해 나타남

: 다수의 컨테이너를 자동으로 운영하기 위한 오케스트레이션 도구

→ 쿠버네티스가 각각의 도커를 한꺼번에 관리가 가능

: 특징

1) 자동화된 복구(self healing)

: 특정 컨테이너가 죽었다면 그 컨테이너를 복제 생성하여 서비스를 유지

2) 로드밸런싱(Load Balancing)

: 사용자의 접속량이 준비된 리소스가 감당할 수 있는 선을 넘을 경우

자동으로 새로운 컨테이너를 생성

: 반대로 니즈가 줄어든다면 컨테이너의 숫자를, 지정해둔 최소 숫자로 자동으로 조절

3) 무중단(Falult Tolerance - FT) 서비스

: 점진적인 업데이트를 제공하기 때문에 서비스를 중단하지 않고도 애플리케이션을 업데이트 할 수 있다

4) 호환성(Vendor Lock In 해결)

*Vendor Lock In

사용자가 클라우드의 환경을 이전하고 싶을 때, 서로 다른 업체의 클라우드 제품 간에 호환문제가 발생하여 이전하기 어려운 상황을 말함

: 도커 컨테이너를 기반으로 하는 오픈 소스로, 사용자들이 특정 업체에 종속되지 않고 클라우드 환경들을 이전할 수 있음

*이슈

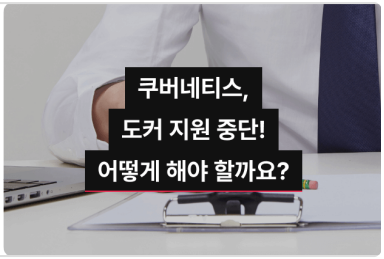
2020.12.8 기준 쿠버네티스 v1.20부터 도커 지원을 중지한다고 발표

도커가 CRI를 사용하는 런타임으로 더 이상 사용되지 않는 것일 뿐, 도커를 더이상 개발 도구로 쓸 수 없는 것은 아님

= 쿠버네티스 런타임에서 도커를 제거하더라도 도커에서 만든 컨테이너 이미지를 등록하고

실행하는 것은 가능.

→ 개발은 여전히 도커로 진행하되, 이를 실행하는 컨테이너 런타임만 바꾸면 상관 X



Q. 쿠버네티스가 도커 지원을 중단했다면 쿠버네티스가 더이상 안 쓰이게 되는 것인지?

A. 결론만 말씀드리자면 아닙니다..!

쿠버네티스에서 도커를 컨테이너 런타임으로 많이 사용하긴 했지만

도커만 지원하는 것은 아니기 때문에

1) 도커심의 대체품인 크리도커드(cri-dockerd)를 사용하거나

2)다른 컨테이너 런타임(ex. cri-o, containerd)를 사용하면 됩니다.

쿠버네티스가 도커 지원을 중단한 이유는 **호환성 문제!**

기존에 쿠버네티스는 컨테이너런타임과 통신할 때 CRI라는 표준 인터페이스를 API로 사용했는데, 도커는 이를 지원하지 않았다. 때문에 도커와 쿠버네티스 API는 '도커심(Dockershim)을 사용하여 이 둘을 연결하여 사용할 수 있었다. 그러나 도커심은 배포 속도를 느리게 하고 유지관리자에게 큰 부담을 주는 문제가 있었기 때문에 쿠버네티스 v1.20 이후부터는 도커심을 지원하지 않기로 해서 도커 지원이 중단된 것입니다.

*cri-o - 훨씬 가볍게 컨테이너 실행이 가능하지만 컨테이너 생성 및 이미지 빌드 기능을 제공하지 않아, kaniko, buildash, Podman, Skpeo와 같이 이미지를 빌드할 수 있는 툴을 함께 써야 함

*containerd - 도커에서 분리된 런타임으로 도커심과 같은 중단매체 필요 X