Docker 1,2

1.Docker

Docker는 2013년 3월 Docker, Inc(구 dotCloud)에서 출시한 오픈 소스 컨테이너 프로젝트입니다.
2014년 현재 나온 지 2년도 채 되지 않았지만 전 세계적으로 큰 인기를 끌고 있습니다.

2010년을 넘어서면서 서버 시장은 급속히 클라우드 환경으로 옮겨갔습니다. 이렇다 보니 물리 서버를 구입하기 보다는 요금만 내고 가상 서버를 빌려 쓰게 되었습니다. 특히 물리적인 서버를 구축하려면 서버 구입과 설치에 상당한 시간이 걸립니다. 하지만 클라우드 환경에서는 1대가 되었든 1,000대가 되었든 클릭 몇 번 만으로 가상 서버를 만들어낼 수 있게 되었습니다.

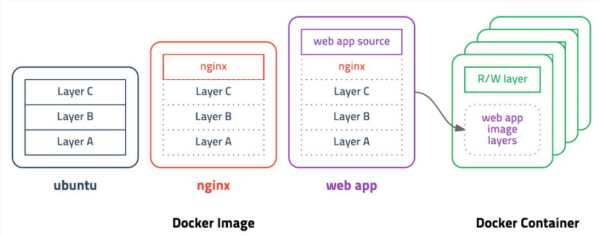
생성된 가상 서버에 각종 소프트웨어를 설치하고 설정을 해야 하는데, 서버가 한 두 대라면 쉽게 설정을 할 수 있지만 서버 개수가 많아지면 사람이 하기가 어려워집니다. 따라서 클라우드 환경에서 설치와 배포가 큰 어려움으로 다가왔습니다.

이런 상황에서 Immutable Infrastructure라는 패러다임이 나왔습니다. Immutable Infrastructure는 호스트 OS와 서비스 운영 환경을 분리하고, 한 번 설정한 운영 환경은 변경하지 않는다(Immutable)는 개념입니다. 즉 서비스 운영 환경을 이미지로 생성한 뒤 서버에 배포하여 실행합니다. 이때 서비스가 업데이트되면 운영 환경 자체를 변경하지 않고, 이미지를 새로 생성하여 배포하는 것입니다. 클라우드 플랫폼에서 서버를 쓰고 버리는 것처럼, Immutable Infrastructure도 서비스 운영 환경 이미지를 한 번 쓰고 버립니다.

1.Docker

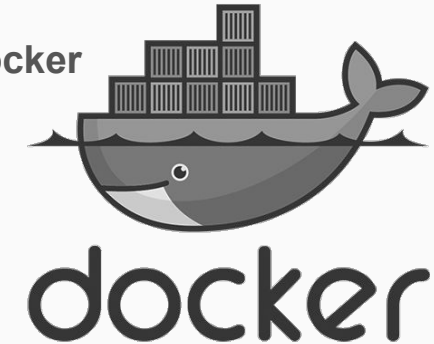
Immutable infrasturucture에는 여러 가지 장점이 있습니다.

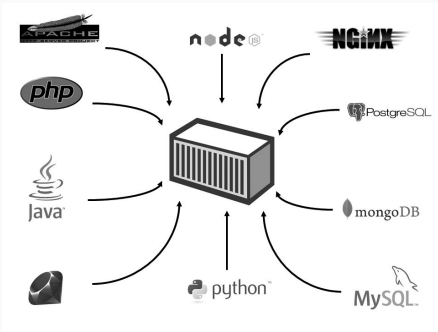
- 편리한 관리 : 운영 환경을 이미지로 생성했기 때문에 이미지 자체만 관리하면 됩니다.
  - Docker에서 Image란?
    - 컨테이너를 실행할 수 있는 실행파일, 설정 값 들을 가지고 있는 것



- 확장 : 이미지 하나로 서버를 계속 생성할 수 있습니다.
  따라서 클라우드 플랫폼의 자동 확장(Auto Scaling)
  기능과 연동하면 손쉽게 서비스를 확장할 수 있습니다.
- 테스트 : 이미지를 실행하기만 하면 서비스 운영 환경과
  동일한 환경이 구성되기 때문에 테스트가 쉽습니다.
- 가볍다 : 운영체제와 서비스 운영 환경을 분리하여 가볍고 어디서든 실행 가능한 환경을 제공합니다.

Immutable Infrastructure를 구현한 프로젝트인 Docker





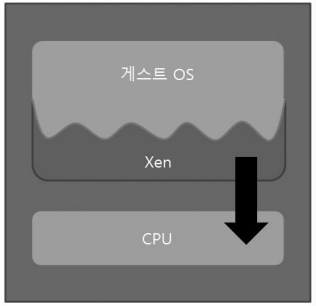
1-1.가상 머신과 Docker - 가상머신

가상 머신

가상 머신은 편하긴 하지만 성능이 좋지 못한 것이 단점이었습니다. 지금까지 CPU에 가상화를 위한 기능들이 많이 추가되었지만 아직도 가상 머신은 리얼 머신에 비해 속도가 느립니다.

전가상화(Full Virtualization)의 느린 속도를 개선하기 위해 반가상화(Paravirtualization) 방식이 개발되었고, 현재 널리 쓰이고 있습니다.

CPU의 가상화 지원을 이용한 전가상화



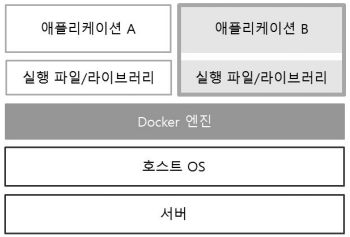
Xen 반가상화

하지만 가상 머신 자체는 완전한 컴퓨터라서 항상 게스트OS를 설치해야 합니다. 그래서 이미지 안에 OS가 포함되기 때문에 이미지 용량이 커집니다.

1-1.가상 머신과 Docker - Docker

Docker

Docker는 반가상화보다 좀더 경량화된 방식입니다. 그림과 같이 게스트 OS를 설치하지 않습니다. Docker 이미지에 서버 운영을 위한 프로그램과 라이브러리만 격리해서 설치할 수 있고, OS 자원(시스템 콜)은 호스트와 공유합니다. 이렇게 되면서 이미지 용량이 크게 줄어들었습니다.

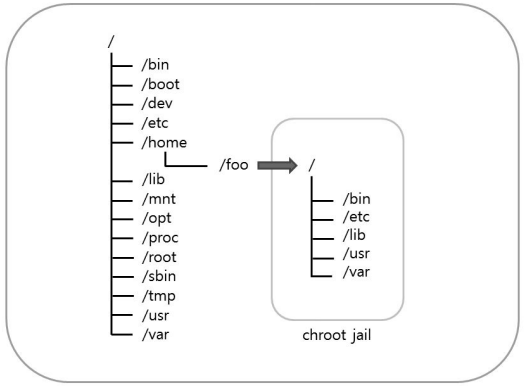


Docker는 가상 머신과는 달리 이미지 생성과 배포에 특화된 기능을 제공합니다. Git에서 소스를 관리하는 것처럼 이미지 버전 관리 기능을 제공합니다. 또한, 중앙 관리를 위해 저장소에 이미지를 올리고, 받을 수 있습니다(Push/Pull). 그리고 GitHub처럼 Docker 이미지를 공유할 수 있는 Docker Hub도 제공합니다(GitHub처럼 유료 개인 저장소도 제공합니다).

다양한 API를 제공하기 때문에 원하는 만큼 자동화를 할 수 있어 개발과 서버 운영에 매우 유용합니다.

1-1.가상 머신과 Docker - 리눅스 컨테이너

오래 전부터 리눅스/유닉스 환경은 chroot라는 명령을 제공했습니다. chroot는 파일시스템에서 루트 디렉터리(/)를 변경하는 명령입니다. chroot로 특정 디렉터리를 루트 디렉터리로 설정하면 chroot jail(감옥)이라는 환경이 생성되는데, chroot jail안에서는 바깥의 파일과 디렉터리에 접근할 수 없습니다. 이처럼 chroot는 디렉터리 경로를 격리하기 때문에 서버 정보 유출과 피해를 최소화 하는데 주로 사용되었습니다.



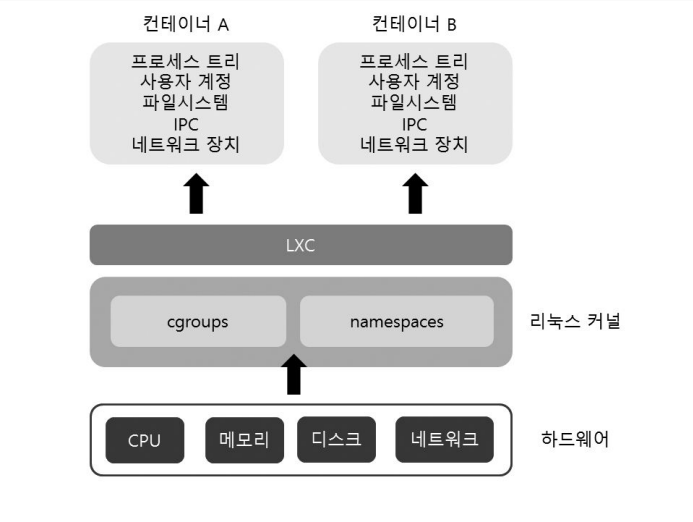
chroot는 chroot jail에 들어갈 실행 파일과 공유 라이브러리를 직접 준비해야 하고 설정 방법이 복잡합니다. 또한, 완벽한 가상 환경이 아니기 때문에 각종 제약이 많습니다. 이후 리눅스는 LXC(LinuX Container)라는 시스템 레벨 가상화를 제공했습니다.

1-1.가상 머신과 Docker - 리눅스 컨테이너

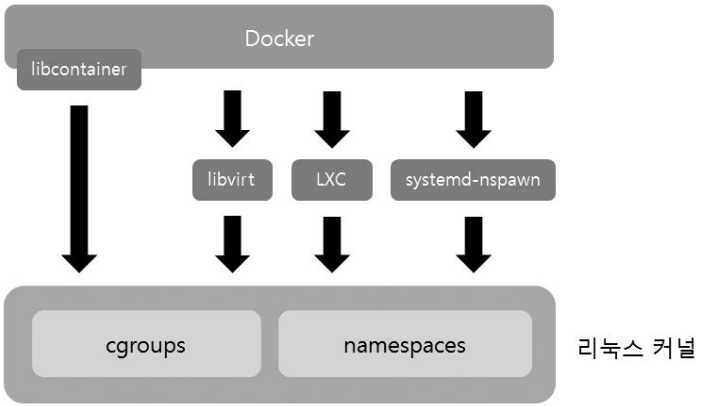
LXC는 컴퓨터를 통째로 가상화하여 OS를 실행하는 것이 아닌 리눅스 커널 레벨에서 제공하는 일종의 격리(Isolate)된 가상 공간입니다. 이 가상 공간에는 OS가 설치되지 않기 때문에 가상 머신이라 하지 않고, 컨테이너라 부릅니다.

리눅스 커널의 Control Groups(cgroups)는 CPU, 메모리, 디스크, 네트워크 자원을 할당하여 완전한 형태의 가상 공간을 제공합니다. 또한, 프로세스 트리, 사용자 계정, 파일시스템, IPC 등을 격리시켜 호스트와 별개의 공간을 만듭니다. 이것을 Namespace isolation(namespaces)이라고 합니다.

LXC는 리눅스 커널의 cgroups와 namespaces 기능을 활용하여 가상 공간을 제공합니다.



1-1.가상 머신과 Docker - 리눅스 컨테이너



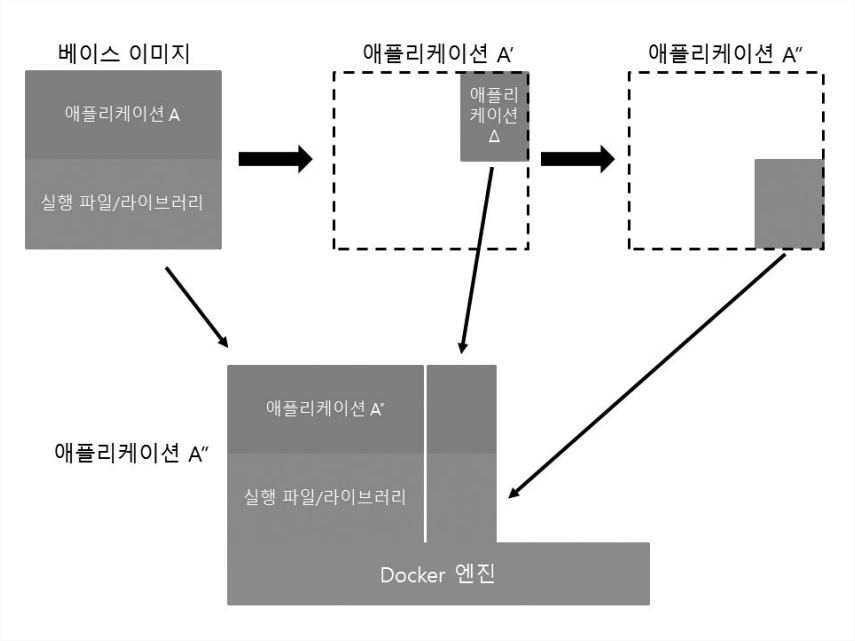
LXC는 격리된 공간만 제공할 뿐 개발 및 서버 운영에 필요한 부가 기능이 부족했습니다. Docker는 리눅스 커널의 cgroups와 namespaces를 기반으로 하여 이미지, 컨테이너 생성 및 관리 기능과 다양한 부가 기능을 제공합니다.

Docker가 처음 개발될 당시에는 LXC를 기반으로 구현을 하였지만 버전 0.9부터는 LXC를 대신하는 libcontainer를 개발하여 사용하고 있습니다. 내부적으로는 실행 드라이버(exec driver)라고 하는데 libcontainer는 native, LXC는 lxc로 표시됩니다. 실행 옵션에 따라 libcontainer를 사용하지 않고 LXC를 사용할 수도 있습니다.

1-2. Docker 이미지와 컨테이너

Docker는 이미지와 컨테이너라는 개념이 있습니다.
먼저 베이스 이미지가 있는데, 리눅스 배포판의
유저랜드만 설치된 파일을 뜻합니다.

유저랜드
OS는 메모리 사용을 기준으로 커널 공간과 유저 공간으로 나눌 수 있습니다. 여기서 유저 공간에서 실행되는 실행 파일과 라이브러리를 유저랜드(userland)라고 합니다. 리눅스는 커널만으로 부팅할 수 없으므로 부팅에 필요한 최소 실행 파일과 라이브러리 조합을 뜻하기도 합니다. 보통 리눅스 배포판에서 유저랜드는 부팅에 필요한 실행 파일과 라이브러리 그리고 고유의 패키징 시스템을 포함합니다.



매번 베이스 이미지에 필요한 프로그램과 라이브러리, 소스를 설치하면 용량이 큰 이미지가 중복되어 생성될 것이라고 생각하기 쉽습니다. Docker 이미지는 베이스 이미지에서 바뀐 부분(Δ)만 이미지로 생성하고, 실행할 때는 베이스 이미지와 바뀐 부분을 합쳐서 실행합니다.

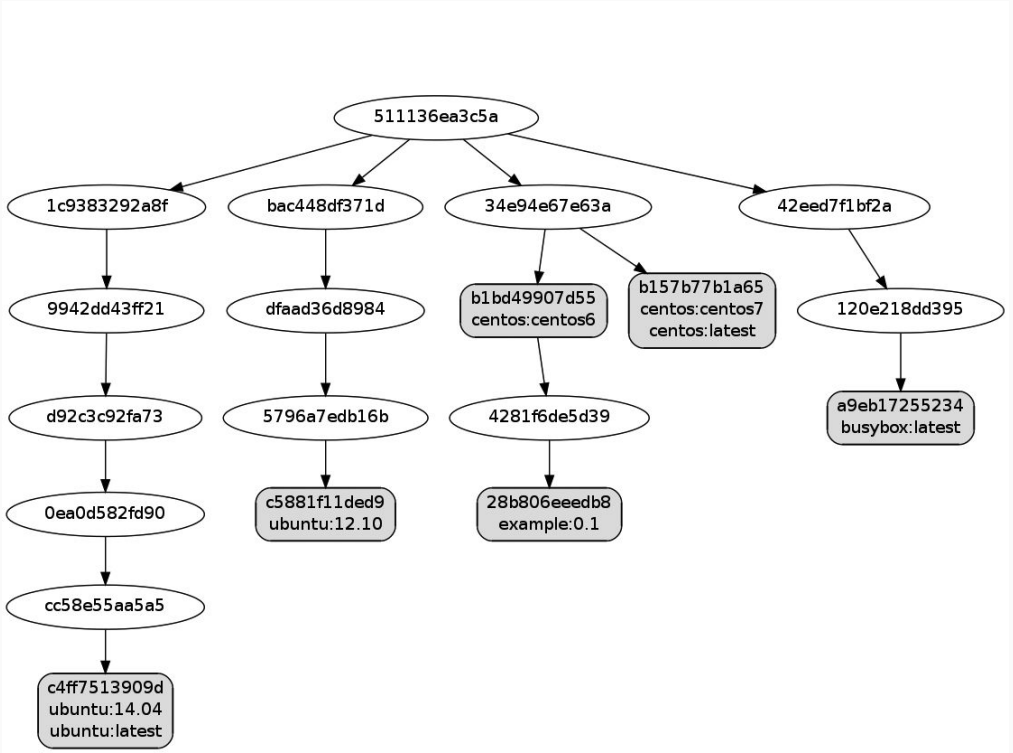
1-2. Docker 이미지와 컨테이너

Docker 이미지는 16진수로 된 ID로 구분하고,
각각의 이미지는 독립적입니다.
centos:centos7 이미지는 511136ea3c5a,
34e94e67e63a, b517b77b1a65가 조합된 것입니다.

centos:centos6 이미지에 서비스 운영에 필요한
프로그램을 설치한 뒤 Docker 이미지를 생성하면
example:0.1과 같은 형태가 됩니다.

즉 Docker는 이미지를 통째로 생성하지 않고, 바뀐 부분만
생성한 뒤 부모 이미지를 계속 참조하는 방식으로
동작합니다. Docker에서는 이를 레이어라고 합니다.

Docker 이미지는 파일이기 때문에 저장소에 올린 뒤 다른
곳에서 받을 수 있습니다. 그리고 저장소에 올릴 때는 자식
이미지와 부모 이미지를 함께 올립니다. 받을 때도
마찬가지로 부모 이미지를 함께 받습니다. 이후에는 내용이
바뀐 이미지만 주고받습니다.



1-2. Docker 이미지와 컨테이너

Union mount, Union File System
생성된 Docker 이미지는 읽기 전용 상태입니다. 여기서 내용이 바뀌면 이미지를 수정하지 않고, 쓰기 이미지를 생성한 뒤 내용을 기록합니다. 이러한 방식을 Union mount라고 하며 Union mount를 지원하는 파일시스템을 Union File System이라 합니다.

Docker 컨테이너는 이미지를 실행한 상태입니다. 이미지로 여러 개의 컨테이너를 만들 수 있습니다. 운영체제로 보면 이미지는 실행 파일이고 컨테이너는 프로세스입니다. 이미 실행된 컨테이너에서 변경된 부분을 이미지로 생성할 수도 있습니다.

Docker는 특정 실행 파일 또는 스크립트를 위한 실행 환경이라 보면 됩니다. 리눅스/유닉스 계열은 파일 실행에 필요한 모든 구성요소가 잘게 쪼개어져 있습니다. 시스템 구조가 단순해지고 명확해지는 장점이 있지만 의존성 관계를 해결하기가 어려워지는 단점이 있습니다. 그래서 리눅스 배포판 별로 미리 컴파일된 패키지(rpm, deb 등)라는 시스템이 나왔습니다. 하지만 서버를 실행할 때마다 일일이 소스를 컴파일하거나 패키지를 설치하고, 설정하려면 상당히 귀찮습니다.

서버가 한 두 대라면 큰 어려움이 없겠지만 클라우드 환경에서는 서버를 몇 십, 몇 백개를 생성해야 합니다. 서버 구성을 미리 해놓은 Docker 이미지를 사용하면 실행할 서버가 몇 개가 되든 손쉽게 해결할 수 있습니다.

2.Docker 설치하기-Windows

Windows에서는 Docker Toolbox(예전 버젼은 Boot2Docker)를 이용하여 Docker를 사용할 수 있습니다. 다음 URL에서 docker-install.exe를 받습니다.

https://www.docker.com/products/docker-toolbox