그림과 실습으로 배우는 도커 & 쿠버네티스

Chapter 1. 도커란 무엇인가?

01. 도커란 무엇인가?

> 데이터 또는 프로그램을 격리시키는 기능을 제공하는 소프트웨어

> 컨테이너는 데이터나 프로그램을 두는 것, 이러한 컨테이너를 다루는 기능을 제공하는 소프트웨어가 도커

> 컨테이너는 도커 엔진이 있어야 만들 수 있음, 또한 도커 엔진 이외에도 이미지가 필요함

> 도커는 리눅스 컴퓨터에서 사용

> 데이터나 프로그램을 독립된 환경에 격리해야 하는 이유는 프로그램 간 공유에 있음
> 예를들어 이스템 A, B 가 어떠한 같은 특정 프로그램과 연동되는 상황에서 각각 연동되는 버전이 다를 경우를 생각하면 됨

02. 서버와 도커

> 서버(Server)란 어떤 서비스(Service)를 제공(Serve)하는 것을 가리키는 말

> 기능적 의미의 서버는 '무슨무슨 기능을 제공한다' 는 의미
> ex) 웹 서버 = 웹 기능을 제공, 메일 서버 = 메일 기능을 제공

> 물리적 의미의 서버는 '물리적 컴퓨터로서의 서버' 를 의미
> 개인용 컴퓨터는 개인이 사용하지만 서버는 여러 사람이 원격으로 접근

> 해 사용

> 서버의 기능은 소프트웨어가 제공하는 것으로 '여러 가지 소프트웨어를 한 컴퓨터에 설치할 수 있다'
> ex) 웹 서버용 소프트웨어를 설치 → 웹 서버가 된다, 메일 서버용 소프트웨어를 설치 → 메일 서버가 된다

> 서버의 운영체제로는 주로 리눅스가 사용됨

> 일반적으로 한 대의 서버 컴퓨터에는 웹 서버를 한 벌(아파치 한 벌)밖에 실행하지 못하지만, 도커의 컨테이너 기술을 활용하면 여러 개의 웹 서버를 올릴 수 있음
> 즉, 도커를 이용하면 물리적 환경의 차이, 서버 구성의 차이를 무시할 수 있으므로 운영 서버와 개발 서버의 환경 차이로 인한 문제를 원천적으로 방지할 수 있음

Etc. 도커와 가상화 기술의 차이

> 가상화 기술은 가상의 물리 서버를 만드는 것과 같음, 여기서 '가상' 이라는 말은 물리적인 대상을 소프트웨어로 대체했다는 의미
> 즉, 메인보드와 CPU, 메모리 등의 물리적인 부품을 소프트웨어로 구현하는 것
> 실질적으로 물리 서버와 동등한 것이므로 당연히 운영체제도 아무 것이나 설치할 수 있고, 그 위에서 어떤 소프트웨어를 구동해도 무방

> 이와 달리 도커는 컨테이너에서 리눅스가 동작하는 것처럼 보이지만 실제 리눅스가 동작하는 것은 아님
> 운영체제의 기능 중 일부를 호스트 역할을 하는 물리 서버에 맡겨 부담을 덜어 둔 형태
> 즉, 컨테이너는 운영체제의 일부 기능을 호스트 컴퓨터에 의존하기 때문에 물리 서버에도 리눅스 기능이 필요하며, 컨테이너의 내용도 리눅스 운영체제가 될 수 밖에 없음

Etc. 도커와 AWS EC2의 차이

> AWS EC2 에도 도커의 컨테이너와 비슷한 '인스턴스' 개념이 존재
> EC2 역시 가상화 기술
> 즉, 각각의 인스턴스가 완전히 독립된 컴퓨터처럼 동작
> 다만, 인스턴스는 컨테이너와 마찬가지로 AMI 라는 이미지로부터 생성하므로 인스턴스를 배포하는 방법은 도커와 비슷

Etc. 도커와 호스팅 서비스

> AWS EC2 가 이에 해당, 이들 서비스를 사용하면 별도로 가상 서버를 만들지 않아도 컨테이너 이미지를 그대로 실행할 수 있음

Chapter 2. 도커의 동작 원리

01. 도커의 동작 원리

> 운영체제 위에 도커 엔진이 동작하고 그 위에서 컨테이너가 동작

> 모든 컨테이너에는 '리눅스 운영체제 비슷한 무언가'가 들어 있음

> 운영체제는 소프트웨어나 프로그램의 명령을 하드웨어에 전달하는 역할
> 운영체제는 '커널' 이라는 부분과 '그 외의 주변 부분'으로 구성됨
> 주변 부분이 프로그램의 연락 내용을 커널에 전달하고 커널이 하드웨어를 다룸

> 도커에서는 컨테이너가 완전히 분리돼 있으므로 밑바탕이 되는 리눅스 운영체제의 주변 부분이 컨테이너 속 프로그램의 명령을 전달 받을 수 없음, 따라서 컨테이너 속에 운영체제의 주변 부분이 들어 있어 프로그램의 명령을 전달받고 이를 밑바탕이 되는 커널에 전달하는 구조로 되어 있음

> 주변 부분만 컨테이너에 넣고 커널은 밑바탕에 있는 것을 빌려 쓰는 형태 덕분에 도커의 가장 큰 특징인 '가벼움'을 얻을 수 있음

> 도커는 밑바탕에서 리눅스 운영체제가 동작하는 것을 전제로 하는 구조로 되어 있기 때문에 리눅스 운영체제에서만 동작할 수 있음
> 즉, 어떤 형태로든 리눅스 운영체제를 갖춰야 함

02. 도커 허브와 이미지, 그리고 컨테이너

> 이미지는 컨테이너를 만드는 데 사용
> 컨테이너로도 이미지를 만들 수 있음, 이는 컨테이너로부터 이미지를 만들 수 없었다면 여러 개의 컨테이너를 일일이 수정할 수밖에 없을 것인데, 개조된 컨테이너로부터 이미지를 만들고 나면 새로 만든 이미지를 사용해 개조된 컨테이너를 여러 개 만들 수 있음
> 예를 들어 소프트웨어나 시스템을 넣은 새로운 이미지를 만들면 다수의 서버를 준비하는 작업이 매우 간단해 짐

> 동일한 컨테이너를 여러 개 만들지 않더라도 이러한 특성을 이용해 다른 물리 서버에 설치된 도커 엔진으로 컨테이너를 이동시킬 수 있음
> 컨테이너는 도커 엔진만 설치되어 있으면 구동이 가능하므로 다른 서버나 컴퓨터에 도커 엔진을 설치하고 새로운 도커 엔진에 이미지를 이용해 똑같은 컨테이너를 생성하면 됨
> 사실 컨테이너 자체가 이동하는 것은 아니지만 이미지를 통해 컨테이너가 이동한 것과 같은 효과를 얻을 수 있음

> 도커 허브는 공개된 컨테이너 이미지가 모여있는 곳
> ex) 구글 플레이 스토어

03. 도커 컨테이너의 생애주기와 데이터 저장

> 컨테이너는 '쓰고 버리는' 일회용품 같은 것

> 컨테이너의 생애주기는 컨테이너를 '만들고', '실행하고', '폐기한' 다음, 다시 컨테이너를 '만드는' 일련의 과정
> 컨테이너의 생애주기 = 컨테이너를 만들고 → 실행하고 → 종료하고 → 폐기하는 과정

> 컨테이너를 폐기했다면 컨테이너에 들어있던 데이터는 어떻게 될까?
> 도커사 설치된 물리적 서버(호스트)의 디스크를 마운트하여 이 디스크에 데이터를 저장

04. 도커의 장점과 단점

> 도커의 장점
> 1) 독립된 환경
> 2) 이미지를 만들 수 있다
> 3) 컨테이너에 '커널을 포함시킬 필요가 없다'
> 4) 한 대의 물리 서버에 여러 대의 서버를 띄울 수 있다
> 5) 서버 관리가 용이하다
> 6) 서버 고수가 아니어도 다루기 쉽다 ⇒ 터미널에 명령을 직접 입력해야 한다는 것 외에는 장애물이 없음, 서버에 대해 잘 모르는 초보자라도 명령어만 익히면 컨테이너를 사용할 수 있음

> 도커의 단점
> 1) 리눅스 운영체제를 사용하는 기술이므로 리눅스용 소프트웨어밖에 지원하지 않는다
> 2) 물리 서버 한 대에 여러 대의 서버를 띄우는 형태이므로 호스트 서버에 문제가 생기면 모든 컨테이너에 영향이 미친다
> 3) 컨테이너 여러 개 사용하는 형태를 가정하므로 컨테이너 하나를 장기간에 걸쳐 사용할 때는 그리 큰 장점을 느끼기 어렵다 ⇒ 애초에 도커를 사용하려면 반드시 도커 엔진을 구동해야 하는데, 컨테이너를 하나밖에 사용하지 않는다면 도커 엔진이 단순한 오버헤드에 지나지 않기 때문

> 도커의 용도
> 1) 팀원 모두에게 동일한 개발환경 제공하기(=동일한 환경을 여러 개 만들기)
> 2) 새로운 버전의 테스트(=격리된 환경을 이용)
> 3) 동일한 서버가 여러 대 필요한 경우(=컨테이너 밖과 독립된 성질을 이용)