**2장 도커의 기본적인 사용법**

***2-1* 컨테이너로 Hello World 실행하기**

**컨테이너 실행 명령어**

**docker container run diamol/ch02-hello-diamol**

**컨테이너 실행 과정**

1. 이미지를 이용해 컨테이너를 실행하려면 먼저 이미지(컨테이너의 패키지)가 있어야 함
2. 명령을 처음 실행했던 시점에는 이미지를 갖고 있지 않았음 (unable to find image locally라는 메시지가 출력)
3. 도커에서 이미지를 내려받음(pull)
4. 내려받은 이미지를 사용해 컨테이너를 실행

이미지에는 애플리케이션을 실행하는 데 필요한 모든 내용과 함께 애플리케이션을 실행하는 방법이 정의되어 있음

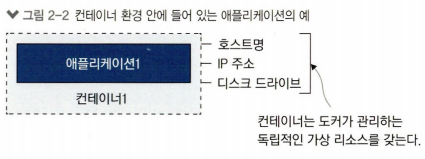
**도커를 사용하는 워크플로: 빌드->공유->실행**

**빌드**: 애플리케이션을 컨테이너에서 실행할 수 있도록 패키징

**공유**: 다른 사람이 패키지를 사용할수 있도록 공유

**실행**: 패키지를 내려받은 사람이 컨테이너를 통해 애플리케이션을 실행

***2-2* 컨테이너란 무엇인가?**

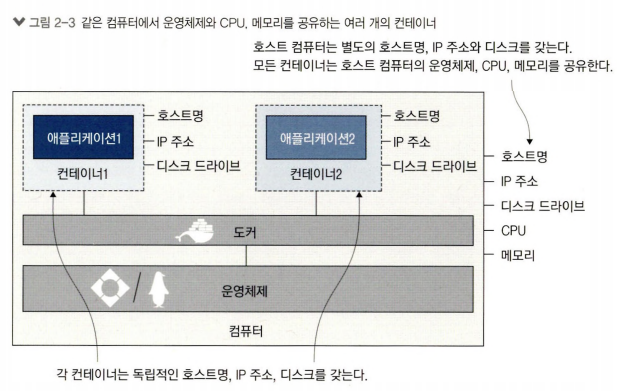


호스트명. IP 주소. 파일 시스템까지 이들은 모두 도커가 만들어낸 가상 리소스

상자 안에서는 상자 밖의 환경을 볼 수 없다.

이 상자는 어떤 컴퓨터상에서 동작하는 것이고 이 컴퓨터는 상자를 여러 개 실행할 수 있다

서로 독립적인 환경을 갖지만 상자가 실행되는 컴퓨터의 CPU와 메모리, 운영체제를 공유



이러한 구도가 중요한 이유는 **격리(isolation)**와 **밀집(density)**이라는 얼핏 보면 모순돼 보이는 조건을 동시에 만족할 수 있기 때문

**밀집**: 컴퓨터에 CPU와 메모리가 허용하는 한 되도록 많은 수의 애플리케이션을 실행하는 것을 의미

-> 그러나 서로 다른 여러 애플리케이션을 동시에 실행하는 데는 제약이 따름

ex) 다른 런타임 버전, 호환되지 않는 버전의 라이브러리, 리소스 부족 등

가상머신을 사용 시

: 가상머신은 컨테이너와 달리 호스트 컴퓨터의 운영체제를 공유하지 않고 별도의 운영체제를 필요

**문제점** : 각각의 운영체제는 애플리케이션이 사용해야 할 CPU와 메모리 자원을 상당량 차지, 운영체제의 라이선스 비용과 운영체제 업데이트 설치 부담

컨테이너 사용 시

: 각각의 컨테이너는 호스트 컴퓨터의 운영체제를 공유하므로 필요한 리소스가 크게 경감, 그만큼 실행도 빠르고 가상 머신에 비해 더 많은 수의 애플리케이션을 실행 가능

외부와 독립된 환경을 제공하므로 밀집과 격리가 동시에 달성

***2-3* 컨테이너를 원격 컴퓨터처럼 사용하기**

**컨테이너를 실행하고 실행된 컨테이너에 터미널을 통해 접근하는 명령어**

docker container run --interactive --tty diamol/base

--interactive: 컨테이너에 접속한 상태

--tty: 터미널 세션을 통해 컨테이너를 조작하겠다는 의미

-> 컨테이너 내부에서 해당 운영체제의 명령행 인터페이스를 사용하듯 명령을 실행 가능

**현재 실행 중인 모든 컨테이너에 대한 정보를 확인하는 명령어**

docker container Is

**대상 컨테이너에서 실행 중인 프로세스의 목록을 보여주는 명령어**

〉 docker container top f1

f1은 container Id의 일부

-> 컨테이너에서 실행 중인 프로세스가 두 개 이상이라면. 모든 프로세스에 대한 정보가 출력

**대상 컨테이너에서 수집된 모든 로그를 출력하는 명령어**

> docker container logs fl

**대상 컨테이너의 상세한 정보를 보여주는 명령어**

> docker container inspect fl

-> 이 정보는 자동 처리에 유리한 JSON 포맷으로 이루어짐

***2-4* 컨테이너를 사용해 웹 사이트 호스팅 하기**

**상태와 상관없이 모든 컨테이너의 목록을 확인하는 명령어**

〉docker container Is --all

* 컨테이너 내부의 애플리케이션이 실행 중이어야 컨테이너의 상태도 실행 중
* 컨테이너가 종료되어도 컨테이너는 사라지지 않음 -> 나중에 컨테이너를 다시 실행하거나, 로그를 확인하거나, 컨테이너의 파일 시스템에 새로운 파일을 복사하거나 외부로 복사해 올 수 있음

**컨테이너에서 웹 사이트를 호스팅하는 명령어**

docker container run —detach --publish 8088：80 diamol/ch02-hello-diamol-web

-> 이 명령을 실행하면 컨테이너 ID만 출력되며 이 컨테이너는 종료되지 않고백그라운드에서 계속 동작함

--detach： 컨테이너를 백그라운드에서 실행하며 컨테이너 ID를 출력

--publish： 컨테이너의 포트를 호스트 컴퓨터에 공개

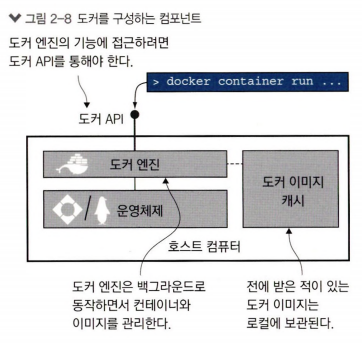
**실행 중인 컨테이너의 상태를 확인할 수 있는 명령어**

〉 docker container stats e53

e53은 container Id의 일부

-> 이 명령어를 실행하면 실시간으로 컨테이너의 CPU, 메모리, 네트워크, 디스크 사용량을 볼 수 있음

***2-5* 도커가 컨테이너를 실행하는 원리**



**도커 엔진**

* 도커의 관리 기능을 맡는 컴포넌트
* 새로운 이미지가 필요하면 이미지를 내려받으며, 기존 이미지가 있다면 전에 내려받은 이미지를 사용(로컬 이미지 캐시를 담당)
* 도커 리소스를 만드는 일도 담당
* 항시 동작하는 백그라운드 프로세스
* **도커 API**를 통해 맡은 기능을 수행
  + **도커 API**: 표준 HTTP 기반 REST API
  + **도커 명령행 인터페이스(도커 CLI)**: 도커 API의 클라이언트. 도커 명령을 사용할 때 실제로 도커 API를 호출하는 역할

원격 컴퓨터에서 실행 중인 도커를 조작할 수 있도록 CLI가 요청을 전달하는 곳을 변경할 수 있음 -> 빌드 환경, 운영 환경, 테스트 환경 등 서로 다른 환경에서 동작하는 컨테이너를 관리하려면 이런 방법을 사용해야 함

**containerd**

* 도커엔진이 컨테이너를 실제로 관리하기 위해 사용하는 컴포넌트
* 호스트 운영체제가 제공하는 기능을 통해 컨네이터, 즉 가상 환경을 만듦
* CNCF에서 관리하는 오픈 소스 프로젝트
* 컨테이너는 개방형 컨테이너 이니셔티브(Open Container Initiative, OCI)라는 이름으로 공개된 개방형 표준