Docker에 대한 약간의 설명

Docker Architecture

Docker는 클라이언트-서버 아키텍처를 사용한다.
Docker 클라이언트는 Docker 데몬(Daemon)과 통신
Docker 데몬 은 Docker 컨테이너를 빌드, 실행 및 배포하는 작업을 많이 수행
Docker 클라이언트와 데몬은 같은 시스템에서 실행될 수 있습니다. 또는 Docker 클라이언트를 원격(remote) Docker 데몬에 연결할 수 있습니다.
Docker 클라이언트와 데몬은 UNIX 소켓 또는 네트워크 인터페이스를 통하여 REAT API를 사용하여 통신합니다.
또다른 Docker 클라이언트는 Docker Compose이며, Docker Compose는 우리가 컨테이너 세트로 구성된 애플리케이션으로 작업할 수 있게 해준다.

Docker Daemon

Docker 데몬(dockerd)은 Docker API요청을 수신하고 이미지, 컨테이너, 네트워크 및 볼륨과 같은 Docker objects을 관리합니다.
데몬은 Docker 서비스를 관리하기 위해 다른 데몬과 통신할 수도 있습니다.

Docker Client

Docker 클라이언트 (docker)는 많은 Docker 사용자가 Docker와 상호 작용하는 기본 방법입니다.
사용자가 docker run과 같은 명령을 사용하면 클라이언트는 이 명령들을 Docker 데몬에게 보냅니다.
Docker 클라이언트는 둘 이상의 데몬과 통신 할 수 있습니다.

Docker Registries

Docker 레지스트리는 Docker 이미지를 저장합니다.
Docker Hub는 누구나 사용할 수 있는 공개 레지스트리이며, Docker는 기본적으로 Docker Hub에서 이미지를 찾습니다.
자신만의 개인 레지스트리를 실행할 수도 있습니다.

docker pull 또는 docker run 명령을 사용하면 Docker는 구성된 레지스트리에서 필요한 이미지를 가져옵니다.
docker push 명령을 사용하면 Docker는 이미지를 구성된 레지스트리에 푸시합니다.

Docker objects

Docker를 사용할 때 이미지(image), 컨테이너(container), 네트워크(network), 볼륨(volume), 플러그인(plugin) 및 기타 object를 생성하고 사용하게 됩니다.

1) 이미지(Images)

이미지는 Docker 컨테이너를 생성하기 위한 지침(instruction)이 포함된 읽기 전용(read-only) 템플릿입니다.
가끔 이미지는 몇몇 추가적인 사용자 정의가 포함된 다른 이미지를 기반으로 합니다.
예를 들어 ubuntu 이미지를 기반으로 하는 이미지를 빌드할 수 있지만 Apache 웹 서버와 애플리케이션은 물론 애플리케이션을 실행하는 데 필요한 구성 상세 정보도 설치합니다.

자신만의 이미지를 만들 수도 있고, 다른 사람이 만들고 레지스트리에 게시한 이미지만 사용할 수도 있습니다.
자신만의 이미지를 빌드하려면 이미지를 생성하고 실행하는 데 필요한 단계를 정의하기 위한 간단한 구문을 사용하여 Dockerfile을 생성합니다.
Dockerfile에 있는 각각의 인스트럭션(Instruction)은 이미지 안에서 레이어(layer)를 생성합니다.
Dockerfile을 변경하고 이미지를 다시 빌드하면 변경된 레이어만 다시 빌드됩니다. 이는 다른 가상화 기술과 비교할 대 이미지를 매우 가볍고 작고 빠르게 만드는 이유 중 하나입니다.

2) 컨테이너(Containers)

컨테이너는 실행 가능한 이미지 인스턴스입니다.
Docker API 또는 CLI를 사용하여 컨테이너를 생성, 시작, 중지, 이동 또는 삭제할 수 있습니다.
컨테이너를 하나 이상의 네트워크에 연결하거나, 컨테이너에 스토리지를 연결하거나, 현재 상태를 기반으로 새 이미지를 생성할 수도 있습니다.

기본적으로 컨테이너는 다른 컨테이너 및 해당 호스트 시스템과 비교적 잘 격리되어 있습니다.
컨테이너의 네트워크, 스토리지 또는 기타 하위 시스템이 다른 컨테이너 또는 호스트 시스템으로 부터 얼마나 격리되는지 제어할 수 있습니다.

컨테이너는 해당 이미지와 컨테이너를 생성하거나 시작할 때 제공하는 구성 옵션으로 정의됩니다.
컨테이너가 제거되면 영구 저장소에 저장되지 않은 상태에 대한 모든 변경 사항이 사라집니다.

다음 명령은 ubuntu 컨테이너를 실행하고, 로컬 명령줄 세션에 대화형으로 연결하고, /bin/bash 를 실행합니다.

```
$ docker run -i -t ubuntu /bin/bash
```

위 명령을 실행하면 다음과 같은 일들이 발생합니다.(기본 레지스트리 구성을 사용한다고 가정)

1. 로컬에 ubuntu 이미지가 없으면 `docker pull ubuntu`를 수동으로 실행한 것처럼 Docker는 구성된 레지스트리에서 해당 이미지를 가져옵니다.
2. Docker는 마치 `docker container create` 명령을 수동으로 실행한 것처럼 새 컨테이너를 생성합니다.
3. Docker는 컨테이너에 읽기-쓰기 파일 시스템을 최종 레이어로 할당합니다. 이를 통해 실행 중인 컨테이너는 컨테이너 내부 로컬 파일 시스템에서 파일과 디렉토리를 생성하거나 수정할 수 있습니다.
4. 네트워킹 옵션을 지정하지 않았으므로 Docker는 컨테이너를 default network에 연결하기 위한 네트워크 인터페이스를 생성합니다.
   여기에는 컨테이너에 IP주소를 할당하는 작업이 포함됩니다. 기본적으로 컨테이너는 호스트 머신의 네트워크 연결을 사용하여 외부 네트워크에 연결할 수 있습니다.

5. Docker는 컨테이너를 시작하고 `/bin/bash`를 실행합니다.
   컨테이너는 대화형으로 실행되고 터미널에 연결되므로(-i 및 -t 플래그로 인해)
   Docker가 출력을 터미널에 기록하는 동안 키보드를 사용하여 입력을 할 수 있습니다.
6. `/bin/bash` 명령을 종료하기 위해 `exit`를 실행하면 컨테이너가 중지되지만 제거되지는 않습니다.
   다시 시작하거나 제거할 수 있습니다.

Docker는 각 컨테이너에 고유 ID를 할당합니다.
full container ID는 64자리 16진수 문자열입니다.
그러나 대부분의 경우 컨테이너 ID의 짧은 버전이면 충분합니다.
short container ID는 full container ID의 처음 12자리를 나타냅니다.

예시) full container ID : acdea168264a08f9aaca0dfc82ff3551418dfd22d02b713142a6843caa2f61bf
short container ID : acdea168264a