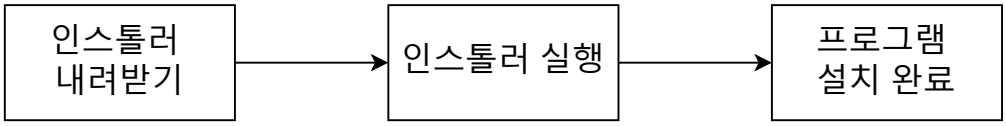
도커를 쓰는 이유

도커를 쓰는 이유는 무엇인가요?

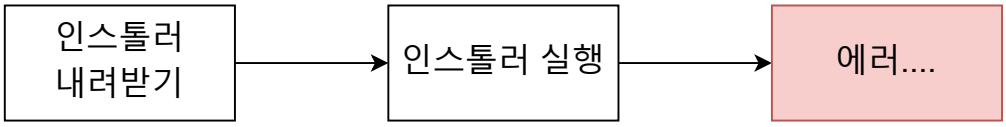
우선 결론부터 얘기하자면 어떠한 프로그램을 다운로드하는 과정을 굉장히 간단하게 만들기 위해서입니다.

먼저 예를 통해 이해해 보겠습니다.

도커 없이 프로그램받을 때 원래 프로그램을 다운로드하고 실행하는 순서



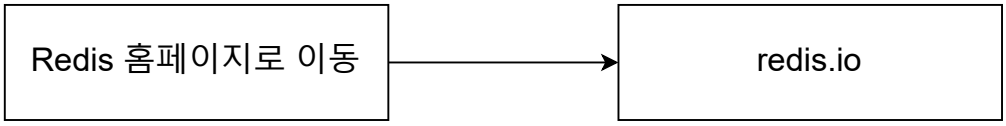
하지만....



갖고 있는 서버, 패키지 버전, 운영체제 등등에 따라
프로그램을 설치하는 과정 중에 많은 에러들이 발생하게 됩니다..
그것만이 아니라 설치 과정이 다소 복잡합니다...

프로그램 다운 받는 것을 도커 없이 원래 방식으로 다운 받을때와 도커를 이용해서 받을 때의 차이점을 한번 봐보겠습니다.

원래 Redis 받는 과정 !



Installation

Download, extract and compile Redis with:

```
$ wget http://download.redis.io/releases/redis-6.0.4.tar.gz
$ tar xzf redis-6.0.4.tar.gz
```

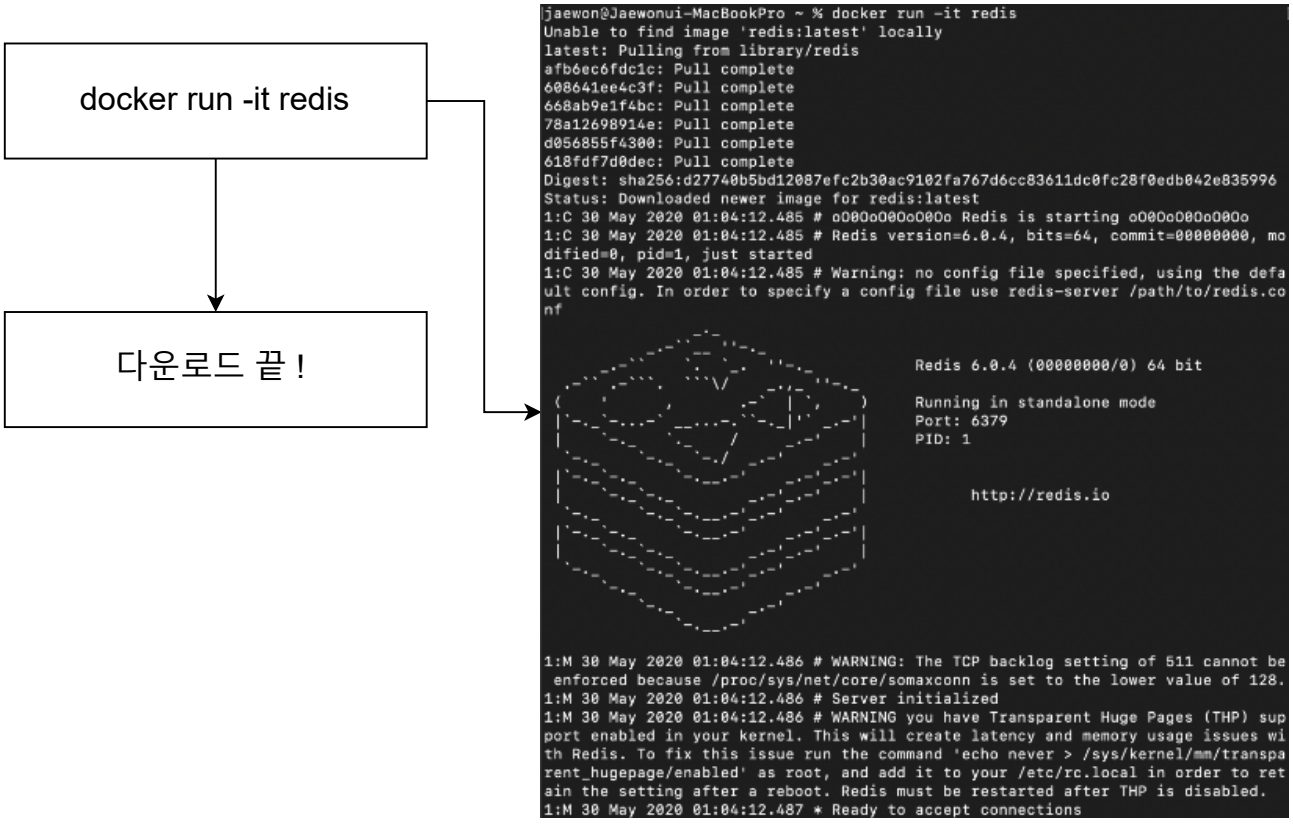
```
$ cd redis-6.0.4
$ make
```

```
jaewon@Jaewonui-MacBookPro ~ % wget http://download.redis.io/releases/redis-6.0.
4.tar.gz
zsh: command not found: wget
jaewon@Jaewonui-MacBookPro ~ %
```

에러 발생... wget 이 없기에...

그래서 wget을 받은 후 다시 Redis 받아야 함...
이런 식으로 어떠한 특정 프로그램을 받을 때
거기에 맞는 부수적인 것들도 계속 받으면서
설치하는 과정이 복잡해지고 에러도 많이 생김...

도커로 Redis 받는 과정



위에서 보는 것과 같이 도커를 이용하여 프로그램을 설치하면 예상치 못한 에러도 덜 발생하며, 설치하는 과정도 훨씬 간단해진 것을 볼 수 있습니다.
이러한 이유로 도커를 사용하고 있습니다.

도커란 무엇인가?

도커는 무엇인가요 ?

컨테이너를 사용하여 응용프로그램을 더 쉽게 만들고 배포하고 실행할 수 있도록 설계된 도구 이며 컨테이너 기반의 오픈소스 가상화 플랫폼이며 생태계입니다.

그러면 컨테이너란 무엇인가요 ?

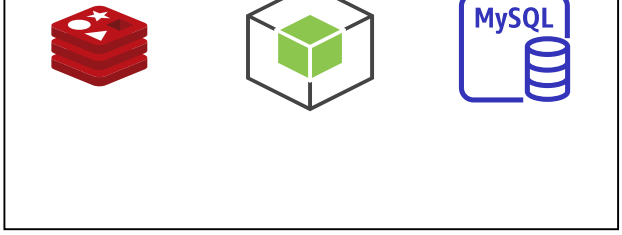
일반적인 컨테이너의 개념

컨테이너

이런 식으로 컨테이너에
물건을 넣고
다양한 운송 수단으로
쉽게 옮길 수 있습니다.

서버에서의 컨테이너의 개념

컨테이너



이런 식으로 컨테이너 안에 다양한 프로그램, 실행환경을
컨테이너로 추상화하고 동일한 인터페이스를 제공하여
프로그램의 배포 및 관리를 단순하게 해 줍니다.
일반 컨테이너의 개념에서 물건을 손쉽게 운송해주는 것처럼
프로그램을 손쉽게 이동 배포 관리를 할 수 있게 해 줍니다.
AWS, Azure, Google cloud 등
어디에서든 실행 가능하게 해 줍니다.

도커 이미지와 도커 컨테이너 정의

도커 이미지와 컨테이너 정의 간단히 살펴보기

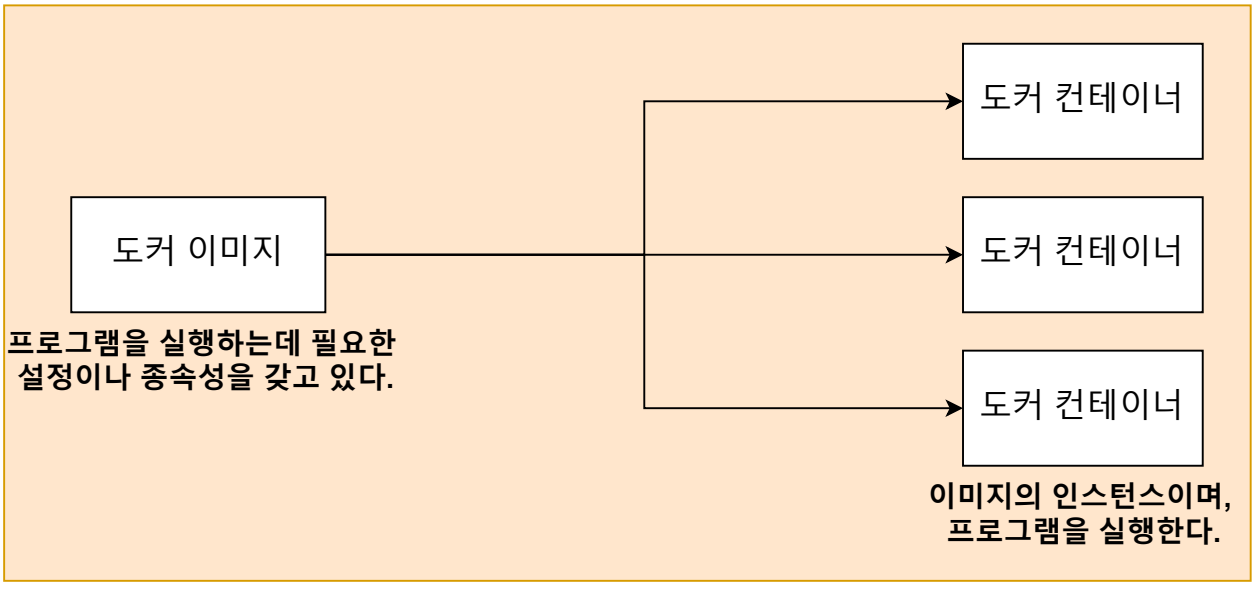
컨테이너는 코드와 모든 종속성을 패키지화하여 응용 프로그램이 한 컴퓨팅 환경에서 다른 컴퓨팅 환경으로 빠르고 안정적으로 실행되도록 하는 소프트웨어의 표준 단위다.

↓

현재까지 여러 가지 방향으로 컨테이너를 정의할 때 간단하고 편리하게 프로그램을 실행시켜주는 것으로 정의를 내리고 있습니다.

컨테이너 이미지는 코드, 런타임, 시스템 도구, 시스템 라이브러리 및 설정과 같은 응용 프로그램을 실행하는 데 **필요한 모든 것을 포함**하는 가볍고 독립적이며 실행 가능한 **소프트웨어 패키지** 입니다.

또한 **컨테이너 이미지는** 런타임에 컨테이너가 되고 도커 컨테이너의 경우 도커 엔진에서 실행될 때 이미지가 컨테이너가 된다.
리눅스와 윈도우 기반 애플리케이션 모두에서 사용할 수 있는 컨테이너화 된 소프트웨어는 인프라에 관계없이 항상 동일하게 실행됩니다.
컨테이너는 소프트웨어를 환경으로부터 격리시키고 개발과 스테이징의 차이에도 불구하고 균일하게 작동하도록 보장한다.



여기서는 간단하게
도커 이미지는 프로그램을 실행하는데 필요한 설정이나 종속성을 갖고 있으며
도커 이미지를 이용해서 컨테이너를 생성하며
도커 컨테이너를 이용하여 프로그램을 실행한다.
이렇게 핵심만 기억해주시면 됩니다 ^^

앞으로 계속 컨테이너와 이미지에 대해서 설명하기 때문에
여기서는 간단히 컨테이너와 이미지가
이러한 것이구나 하고 넘어가 주세요.

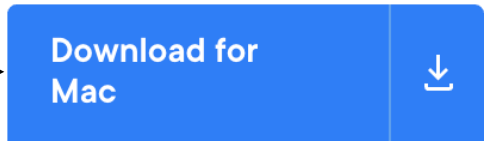
MacOS를 위한 도커 다운받기

도커를 다운받는 순서

Docker 사이트로 이동 → docker.com

get started 버튼 클릭 →

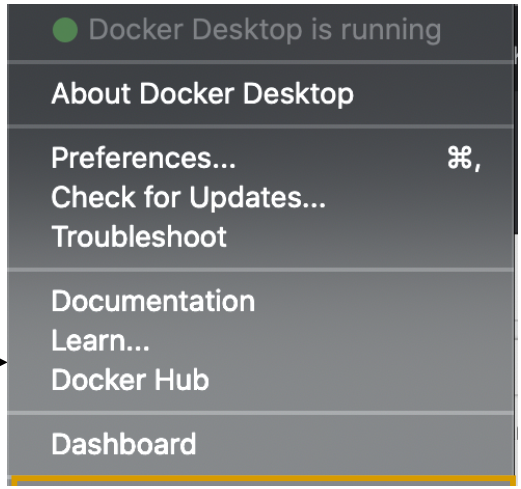
다운로드 도커 installer →

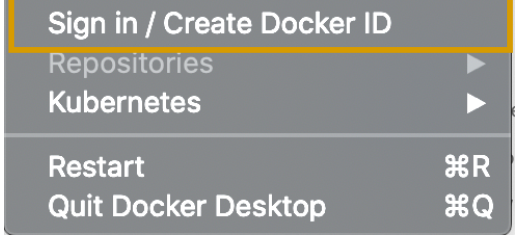


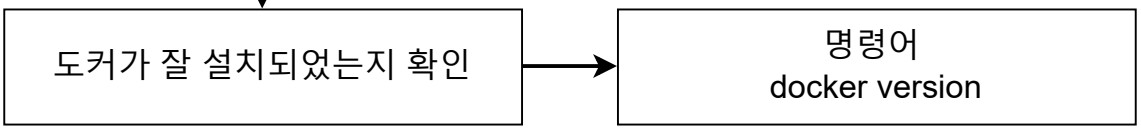
설치하기

Docker 사이트 회원 가입

Docker에 로그인 →







이렇게 해서 MAC OS를 위한 도커를 다운로드하였습니다.
이제는 Window OS를 위한 도커를 다운로드하여 보겠습니다.

Window를 위한 도커 다운받기

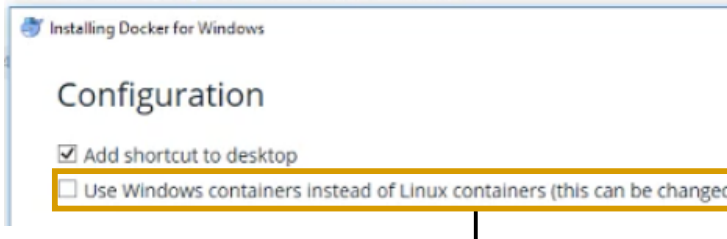
도커를 다운받는 순서

Docker 사이트로 이동 → docker.com

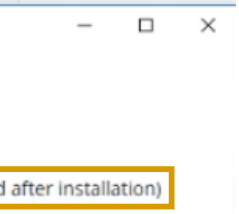
get started 버튼 클릭 →

다운로드 도커 installer →

설치하기



저희는 윈도우 컨테이너 대신에
리눅스 컨테이너를 사용하기에 클릭하지 말



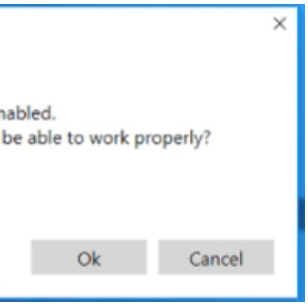
아주세요.

Docker 사이트 회원 가입

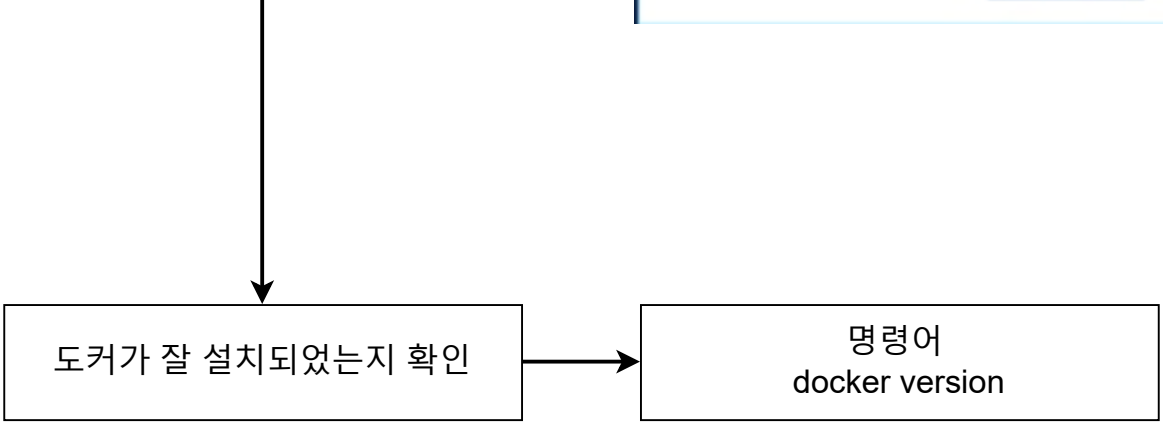
설치 후 나타나는 아이콘 클릭

컴퓨터 재시작

Docker에 로그인

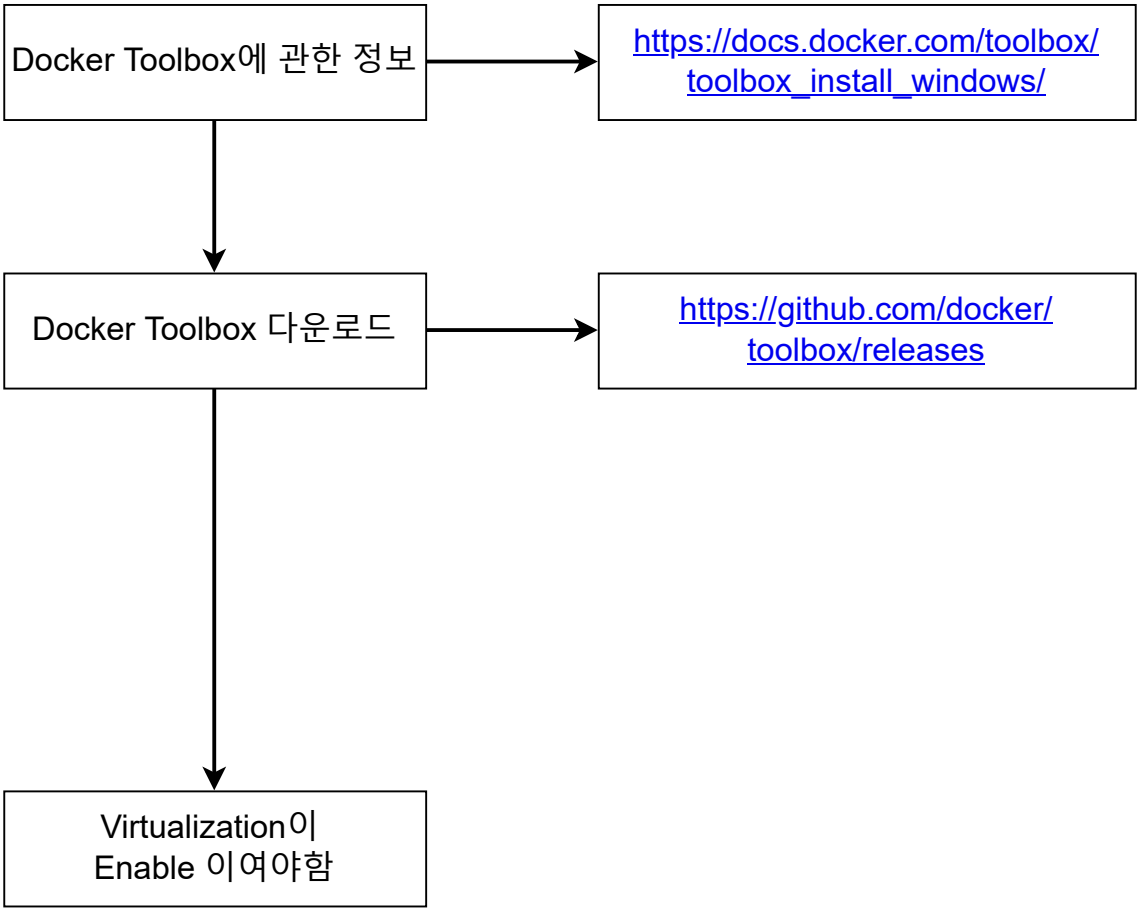

nabled.
be able to work properly?
Ok
Cancel


ud
one


도커가 잘 설치되었는지 확인
명령어
docker version

Window를 Home 유저를 위한 도커 다운받기

Docker Community Edition for Windows를 사용하려면
Hyper-V라는 것이 필요한데
Home 버전에서는 Hyper-V 라는 것을 지원하지 않기에
일반 윈도우 버전 도커를 사용할 수 없습니다.
그러기에 Docker Toolbox를 설치해서 사용해야 합니다.

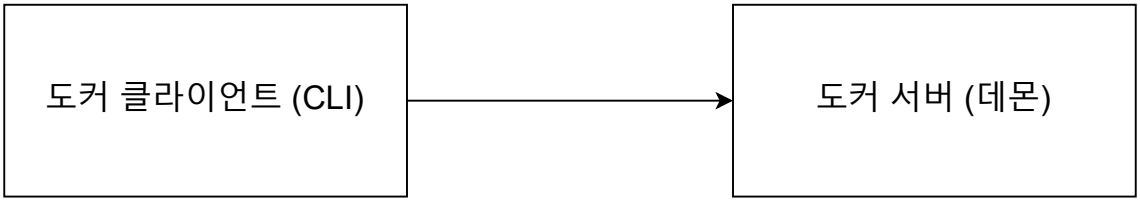


Docker Toolbox에서는 로컬 호스트에 접근할 때
Localhost 대신에 192.168.99.100을 사용해야 합니다.

Docker를 사용할 때의 흐름 감잡기

항상 도커를 사용할 때는...

1. 먼저 도커 CLI에 커맨드를 입력한다.
2. 그러면 도커 서버 (도커 Daemon)이 그 커맨드를 받아서 그것에 따라 이미지를 생성하든 컨테이너를 실행하든 모든 작업을 하게 된다.



실제로 CLI에서 커맨드를 입력해보기

1. docker run hello-world

```
jaewon@Jaewonui-MacBookPro ~ % docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:6a65f928fb91fcfbc963f7aa6d57c8eeb426ad9a20c7ee045538ef34847f44f1
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

설명

1. 도커 클라이언트에 다가 커맨드를 입력하니 클라이언트에서 도커 서버로 요청을 보냄

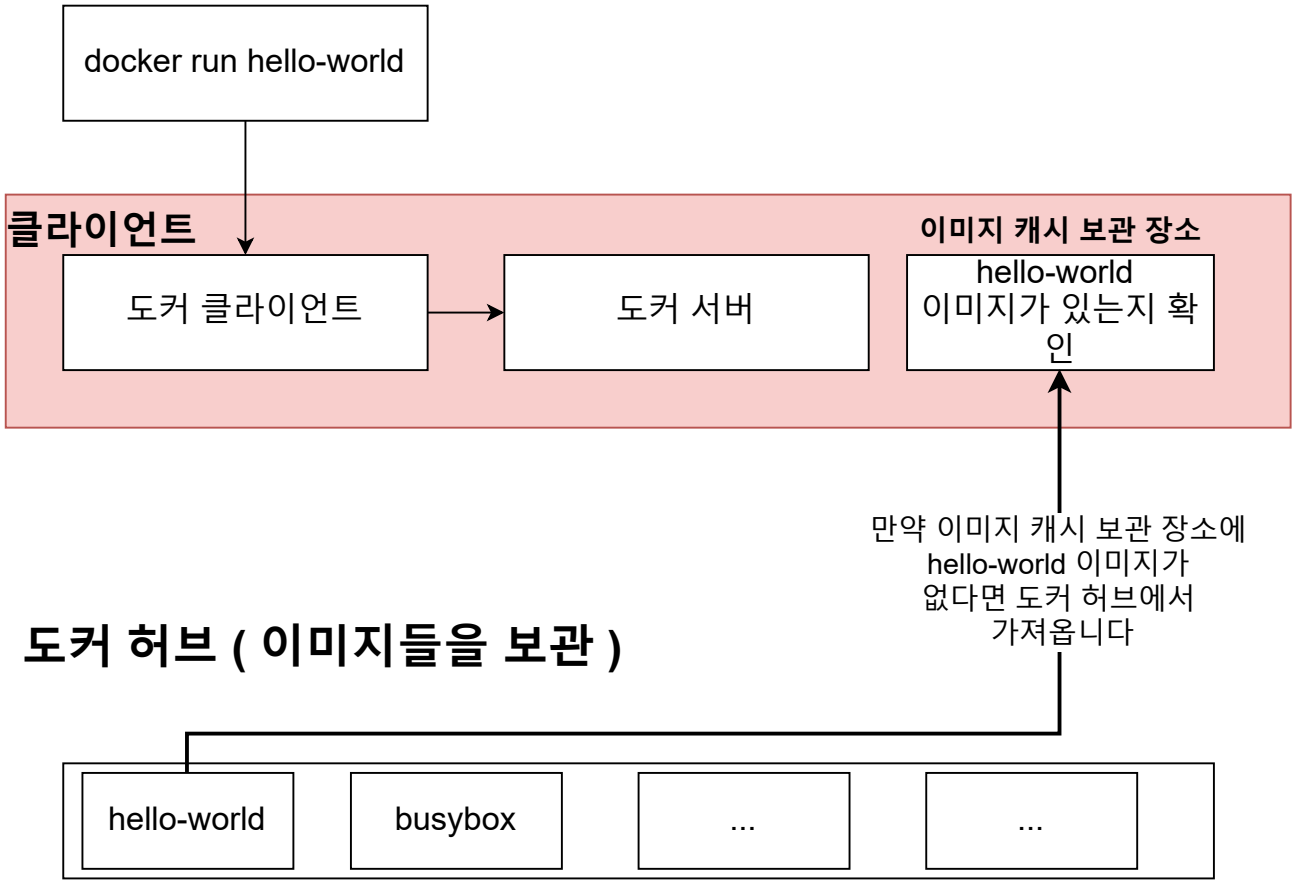
2. 서버에서 hello-world라는 이미지가 이미 로컬에 cache가 되어 있는지

확인

3. 현재는 없기에 Unable to find image ~라는 문구가 2번째 줄에 표시

4. 그러면 Docker Hub이라는 이미지가 저장되어 있는 곳에 가서 그 이미지를
   가져오고 로컬에 Cache로 보관한다.

5. 그 후 이제는 이미지가 있으니 그 이미지를 이용해서 컨테이너를 생성한다.



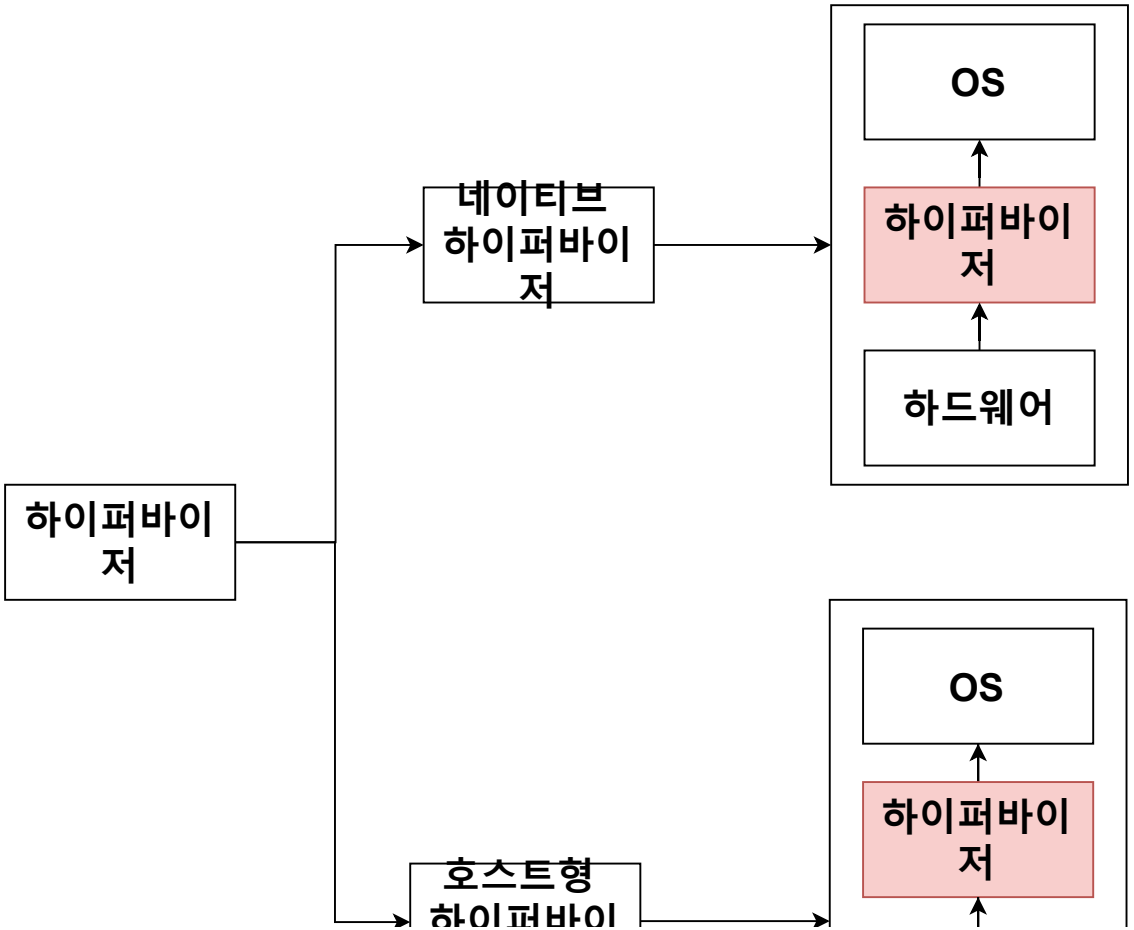
이제 hello-world 이미지가 캐쉬가 되어있으니 한번 더 docker run hello-world 하면 어떻게 될까?

1. Unable to find image ~ 라는 문구가 없이 프로그램이 실행됨.

도커와 기존의 가상화 기술과의 차이를 통한 컨테이너 이해

가상화 기술 나오기 전 → **하이퍼 바이저 기반의 가상화 출현**

가상화 기술 나오기 전	하이퍼 바이저 기반의 가상화 출현
한대의 서버를 하나의 용도로만 사용 남는 서버 공간 그대로 방치... 하나의 서버에 하나의 운영체제, 하나의 프로그램만을 운영... 안정적 But 비효율적	논리적으로 공간을 분할하여 VM이라는 독립적인 가상 환경의 서버 이용 가능 하이퍼 바이저는 호스트 시스템에서 다수의 게스트 OS를 구동할 수 있게 하는 소프트웨어, 그리고 하드웨어를 가상화하면서 하드웨어와 각각의 VM을 모니터링하는 중간 관리자이다.

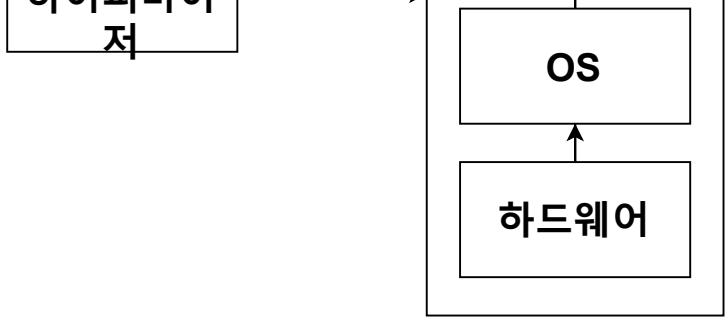


하이퍼 바이저
자원 효율적으로 사용
하지만 여러 하드웨어 드

일반적인 소프트웨
하드웨어 자
에뮬레이트 하

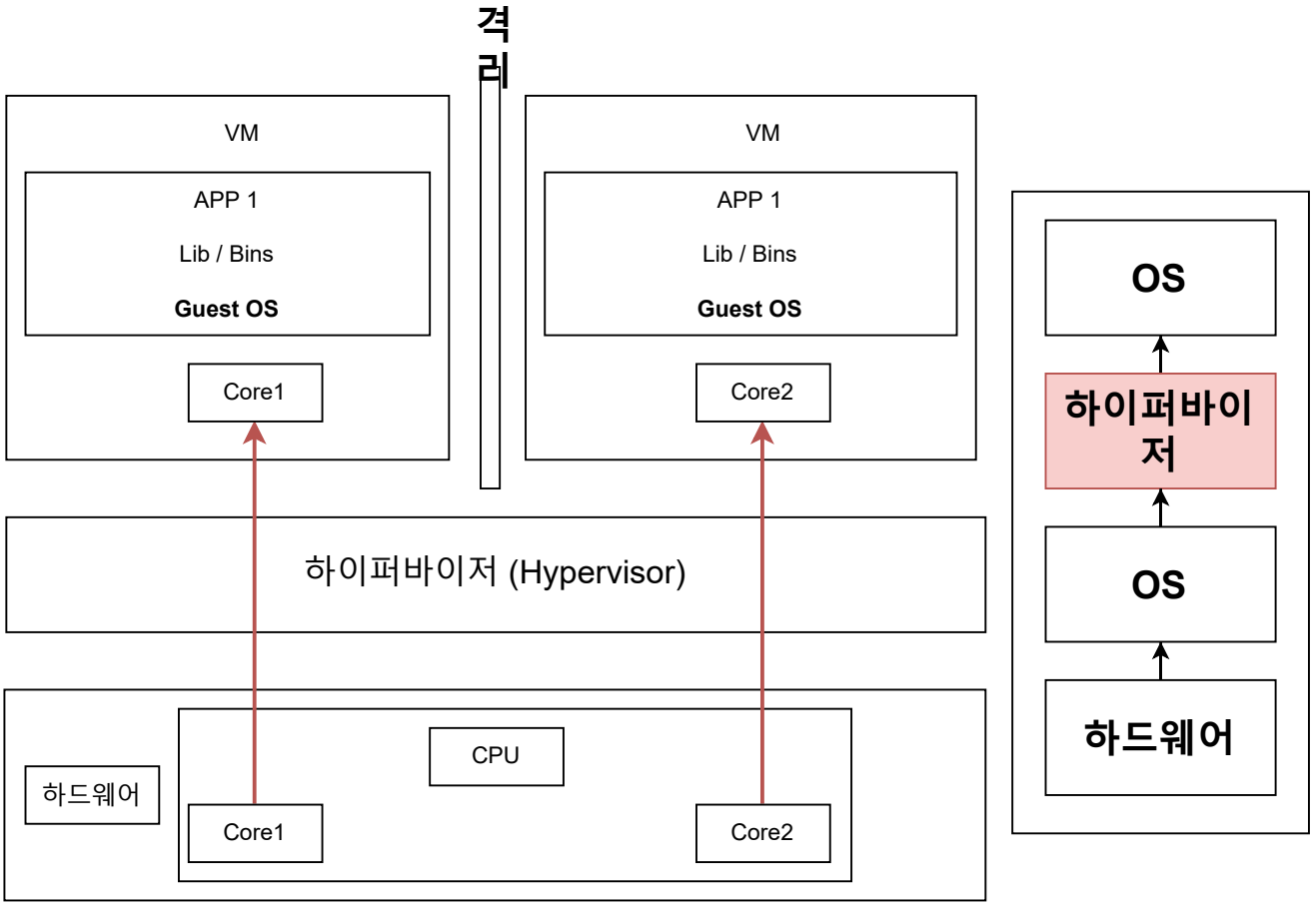
가 하드웨어를 직접 제어하기에
가능하며, 별도의 호스트 OS가 없으므로
오버헤드가 적다.
드라이버를 세팅해야 하므로 설치 어렵다.

어처럼 호스트 OS 위에서 실행되며,
원을 VM 내부의 게스트 OS에
는 방식으로 오버헤드가 크다.



하지만 게스트 OS 종류
일반적으

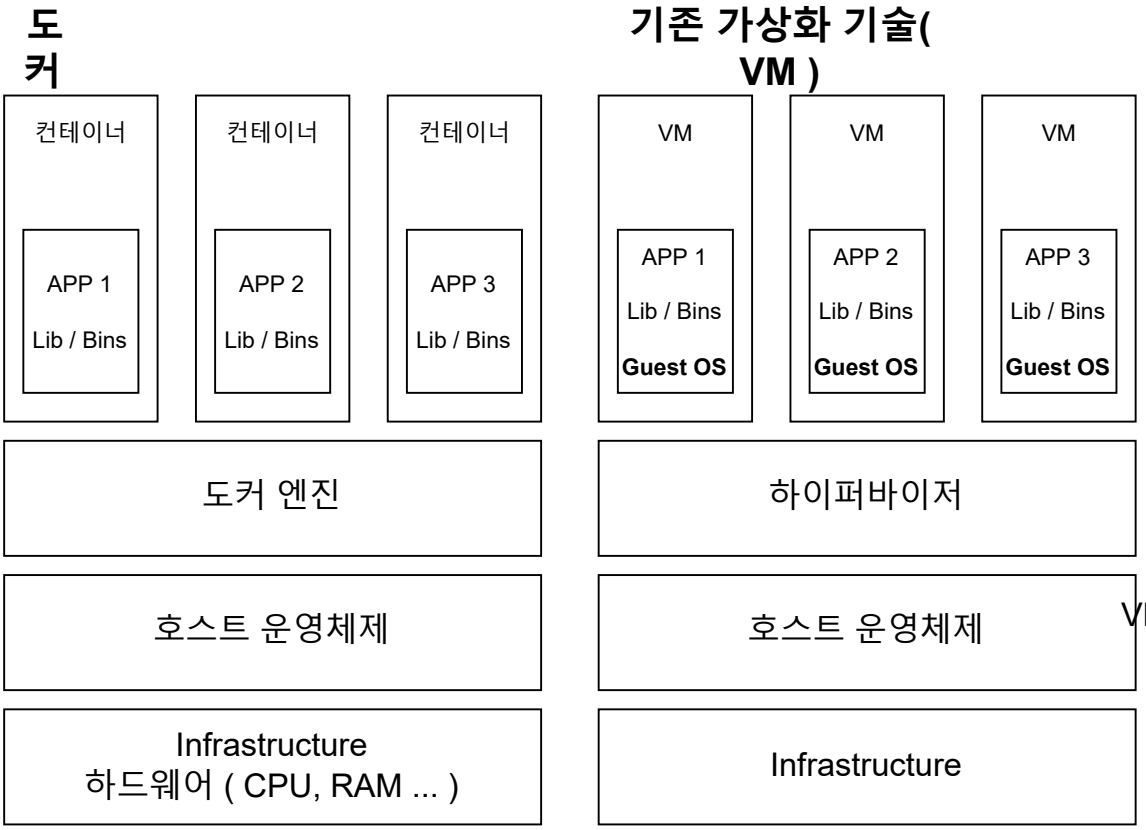
하이퍼 바이저 기반의 VM 구조



하이퍼바이저에 의해 구동되는 VM은 각 VM마다 독립된 가상 하드웨어 자원을 할당받습니다.
논리적으로 분리 되어 있어서 한 VM에 오류가 발생해도 다른 VM으로 퍼지지 않는다는 장점이 있습니다.

에 대한 제약이 없고 구현이 다소 쉽다.
으로 많이 이용하는 방법!

이러한 가상화 기술에서 나온 컨테이너 가상화 기술



VM과 비교했을 때
게스트 OS가 필요

애플리케
호스트 OS의

VM은 애플리케이션
게스트 OS
훨씬

공통점

도커 컨테이너와 가상 머신은 기본 하드웨어에서 격리된 환경 내에 애플리케이션을 배치하는 방법입니다.

차이점

가장 큰 차이점은 격리된 환경을 얼마나 격리를 시키는지의 차이

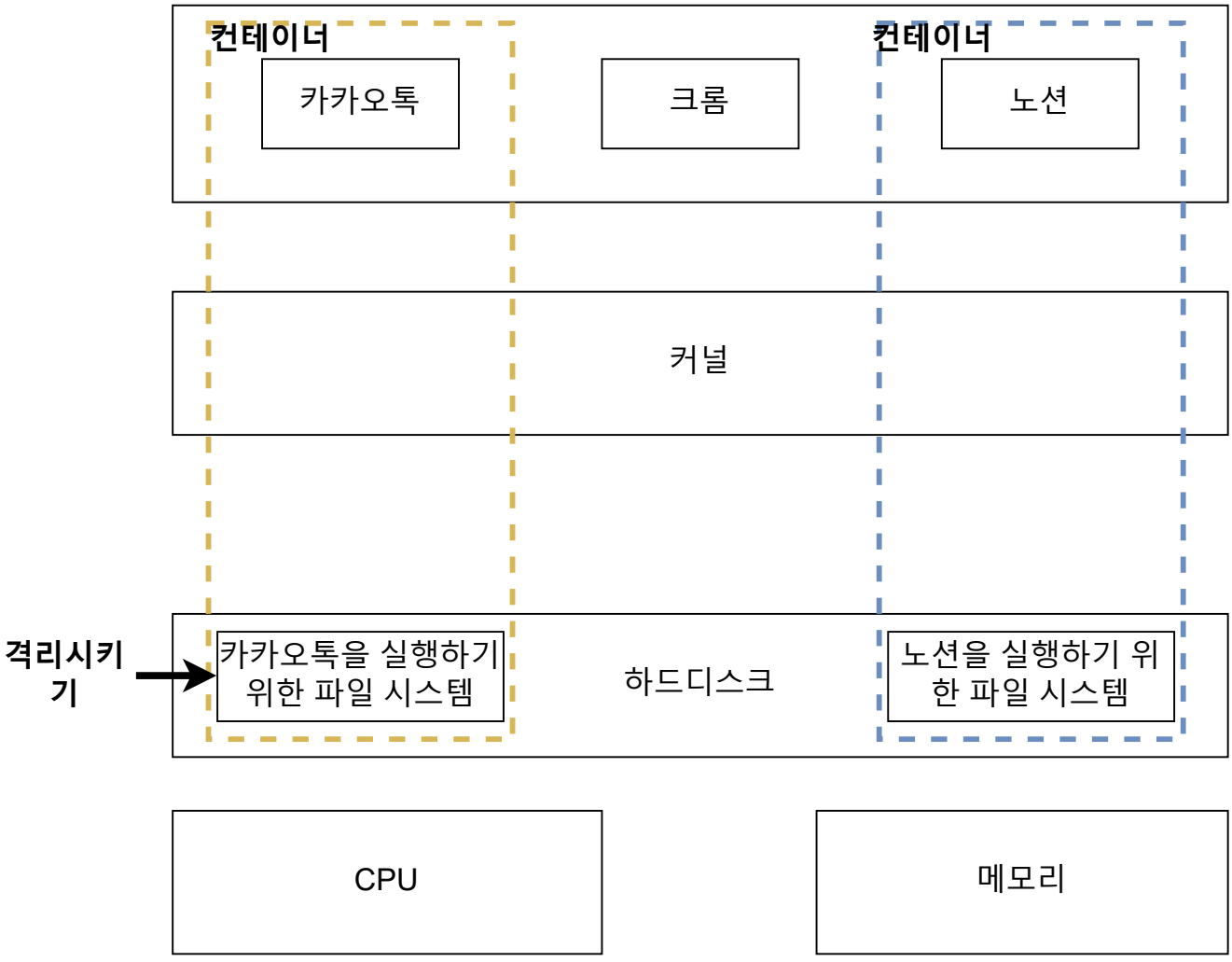
도커 컨테이너에서 돌아가는 애플리케이션은 컨테이너가 제공하는 격리 기능 내부에 샌드박스가 있지만, 여전히 같은 호스트의 다른 컨테이너와 동일한 커널을 공유한다. 결과적으로, 컨테이너 내부에서 실행되는 프로세스는 호스트 시스템(모든 프로세스를 나열할 수 있는 충분한 권한있음)에서 볼 수 있다.
예를 들어, 도커와 함께 몽고DB 컨테이너를 시작하면 호스트(도커가 아님)의 일반 쉘에 ps-e grep 몽고를 실행하면 프로세스가 표시됩니다.
또한, 컨테이너가 전체 OS를 내장할 필요가 없는 결과, 그것들은 매우 가볍고, 일반적으로 약 5-100 MB이다.

때 컨테이너는 하이퍼바이저와
요하지 않으므로 더 가볍습니다.

케이션을 실행할 때는 컨테이너 방식에서는
위에 애플리케이션의 실행 패키지인 이미지를
배포하기만 하면 되는데
을 실행하기 위해서 VM을 띄우고 자원을 할당한 다음,
S를 부팅하여 애플리케이션을 실행해야 해서
인 복잡하고 무겁게 실행을 해야 합니다.

가상 머신과 함께 VM 내부에서 실행되는 모든 것은 호스트 운영 체제 또는 하이퍼바이저와 독립되어 있다. 가상 머신 플랫폼은 특정 VM에 대한 가상화 프로세스를 관리하기 위해 프로세스를 시작하고, 호스트 시스템은 그것의 하드웨어 자원의 일부를 VM에 할당한다. 그러나 VM과 근본적으로 다른 것은 시작 시간에 이 VM 환경을 위해 새롭고 이 특정 VM만을 위한 커널을 부팅하고 (흔히 다소 큰) 운영 체제 프로세스 세트를 시작한다는 것이다. 이것은 응용 프로그램만 포함하는 일반적인 컨테이너보다 VM의 크기를 훨씬 크게 만든다.
OS까지 가상화... 맥에서 윈도우를 깐다든지 리눅스에서 윈도우를 돌린다든지...
이러한 방법은 비교적 사용법이 간단할 수 있지만 굉장히 느리다.

컴퓨터에 실행되고 있는 프로세스들



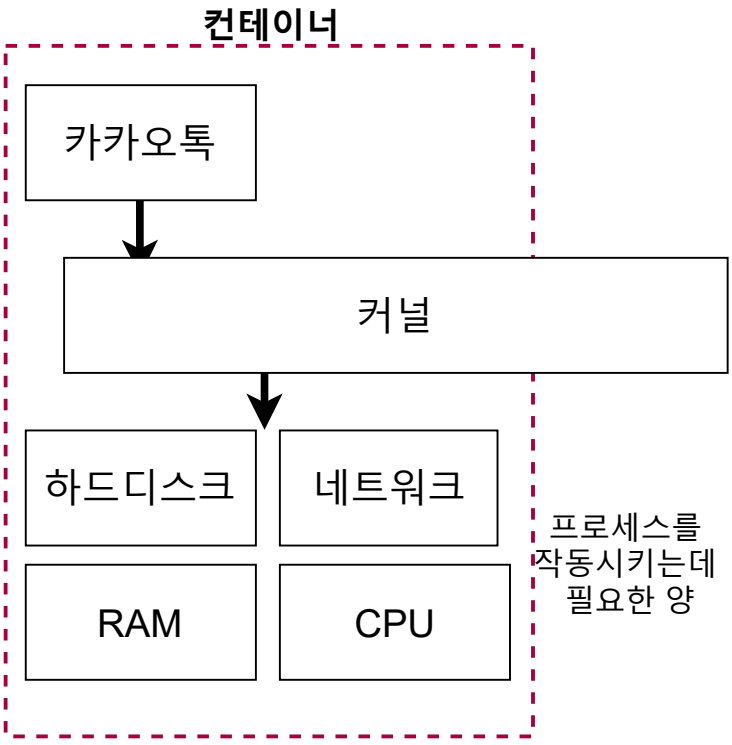
위에서 보면 컨테이너들을 격리시키는데 어떻게 해서 도커 컨테이너를 격리를 시킬까요 ?

먼저 리눅스에서 쓰이는 Cgroup(control groups)과 네임스페이스(namespaces)에 대해서 알아야 합니다.

이것들은 컨테이너와 호스트에서 실행되는 다른 프로세스 사이에 벽을 만드는 리눅스 커널 기능들입니다.

C Group	네임스페이스
CPU, 메모리, Network Bandwith, HD i/o 등 프로세스 그룹의 시스템 리소스 사용량을 관리	하나의 시스템에서 프로세스를 격리시킬 수 있는 가상화 기술

=> 이런 이슈이 사용량이 너무 많다면 그 어플리케이션 같은 것을 C group에 집어넣어서 CPU와 메모리 사용 제한 가능	몇개의 독립된 공간을 사용하는 것처럼 격리된 환경을 제공하는 경량 프로세스 가상화 기술



이미지로 컨테이너를 만들기

전 강의들에서 이미지를 이용해서 컨테이너를 생성한다고 배웠습니다.
하지만 어떻게 해서 이미지를 이용해 컨테이너를 생성하는지는
다루지 않았기 때문에 이번 시간에
어떻게 컨테이너가 생성이 되게 되는지 자세히 알아보겠습니다.

시작하기 전에 기억해야 할 것

이미지는 응용 프로그램을 실행하는데 **필요한 모든 것**을 포함하고 있습니다.

그러면 그 필요한 것이 무엇일까요?

1. 컨테이너가 시작될 때 실행되는 명령어 ex) run kakaotalk

2. 파일 스냅숏 ex) 컨테이너에서 카카오톡을 실행하고 싶다면
카카오톡 파일 (카카오톡을 실행하는데 필요한 파일) 스냅숏

* 파일 스냅숏은 디렉터리나 파일을 카피한 것

이미지

1. 시작 시 실행할 명령어

2. 파일 스냅숏

카카오톡 파일

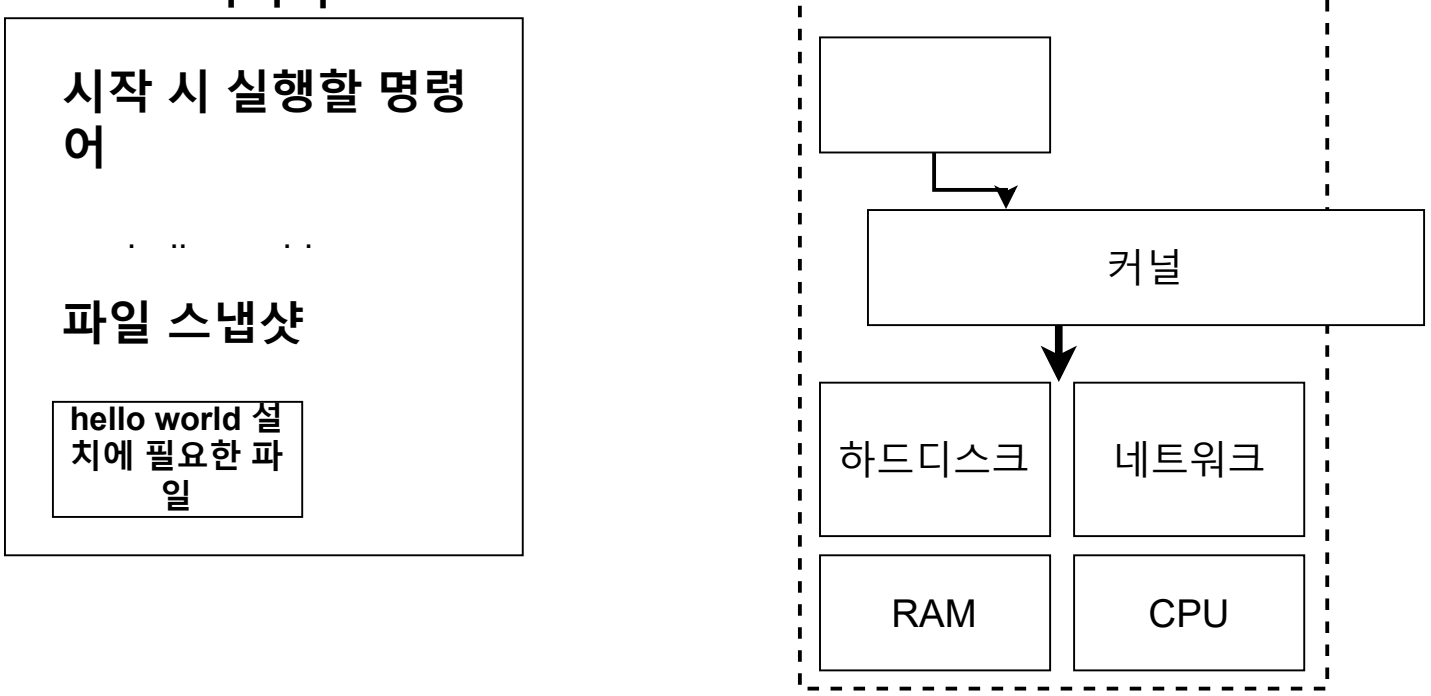
이미지로 컨테이너 만드는 순서

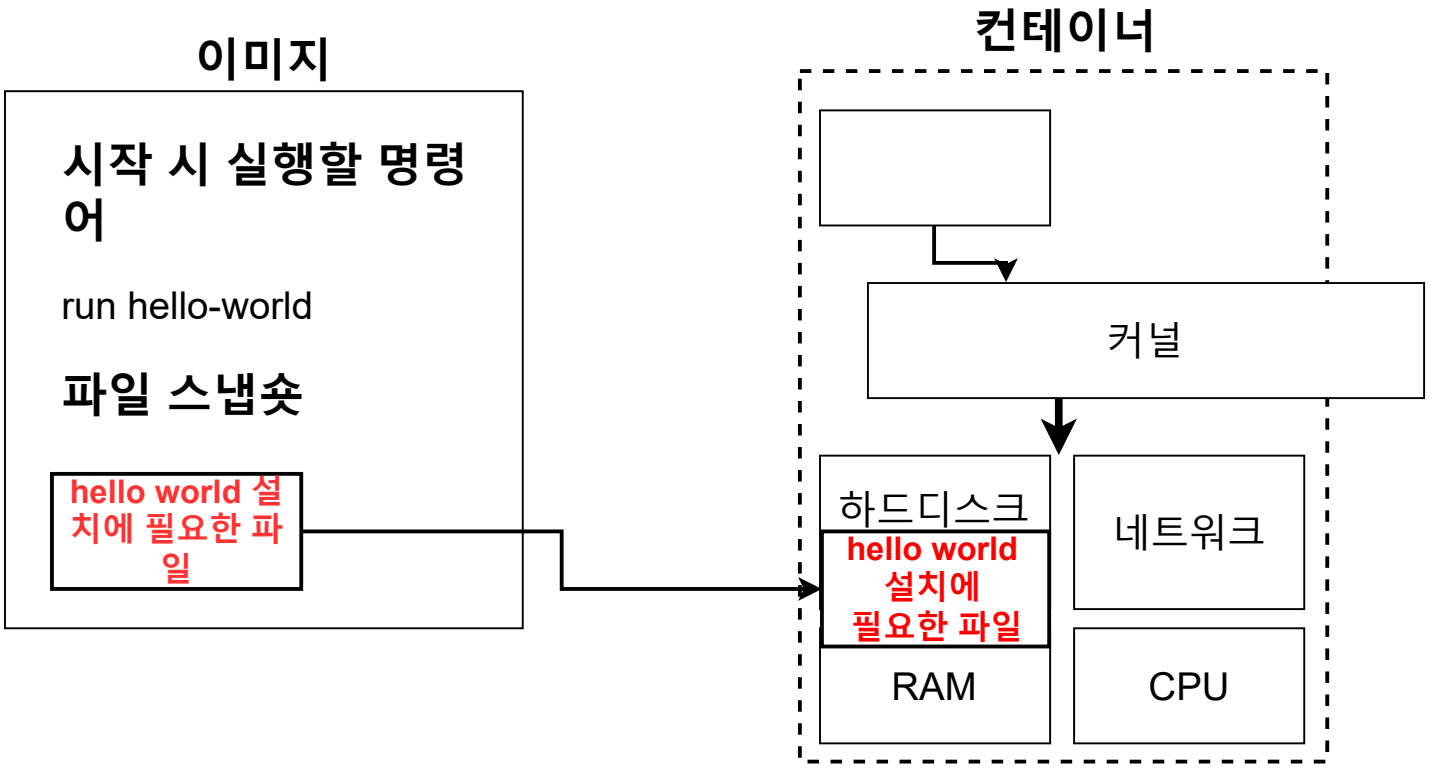
1. Docker 클라이언트에 docker run <이미지> 입력해줍니다.

2. 도커 이미지에 있는 파일 스냅샷을 컨테이너 하드 디스크에 옮겨 줍니다.

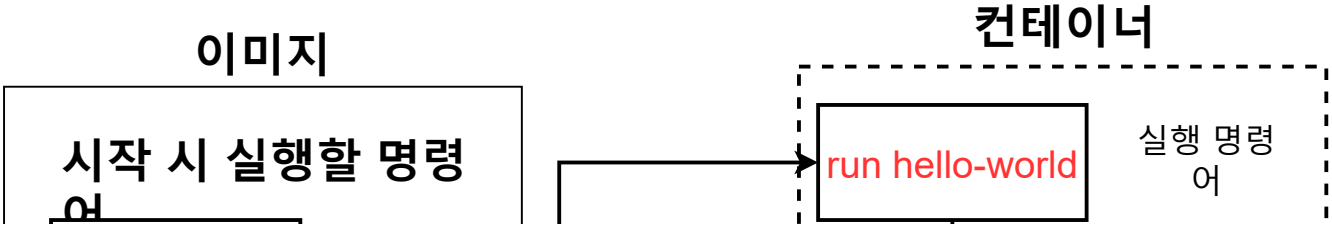
이미지

컨테이너





3. 이미지에서 가지고 있는 명령어 (컨테이너가 실행될때 사용될 명령어를)를 이용해서 카카오톡을 실행시켜줍니다.



run hello-world

파일 스냅숏

hello world 파일

커널

하드디스크
hello world
실행 파일

네트워크

RAM

CPU

프로세스를 작동시키는데 필요한 양

이미지

시작 시 실행할 명령어

run hello-world

파일 스냅숏

hello world 파일

컨테이너

Hello-World

실행 중인 프로세스

커널

하드디스크
hello world
실행 파일

네트워크

RAM

CPU

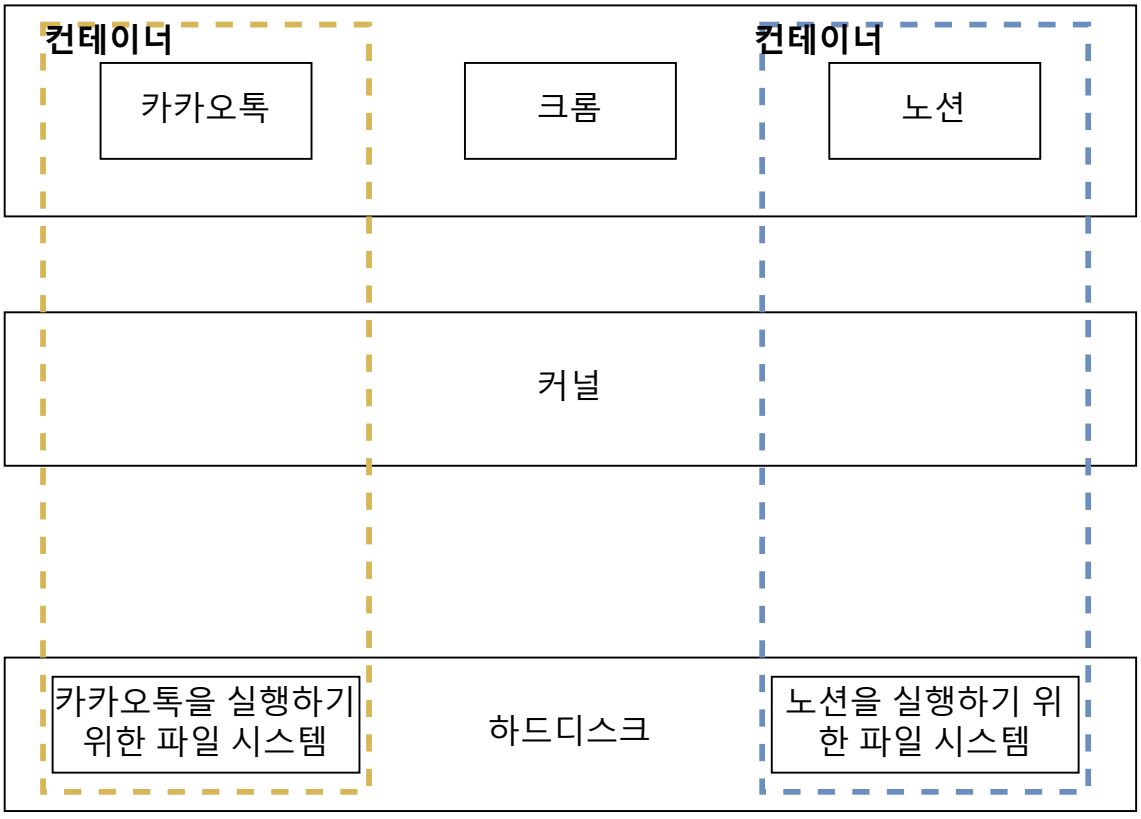
프로세스를 작동시키는데 필요한 양

Cgroup, 네임스페이스를 쓸수 있는 이유

컨테이너를 격리시킬 수 있는 이유는 Cgroup과 네임스페이스를 이용해서이다..

하지만 생각해보면 Cgroup과 네임스페이스는 리눅스 환경에서 사용되는 것인

컴퓨터에 실행되고 있는 프로세스들



하지만 Cgroup과 네이스페이스는 Linux에서만 가능

하지만 나는 지금 윈도우나 맥OS를 사용하고 있다.....

어떻게 된 것 인가 ??????

터미널을 킨 후

docker version 명령어를 입력한 후 서버 정보를 보면

```
Server: Docker Engine - Community
 Engine:
  Version:          19.03.8
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.17
```

```
Git commit:        afacb8b
Built:             Wed Mar 11 01:29:16 2020
OS/Arch:           linux/amd64
Experimental:      false
```