Ch 3. 도커 엔진

3. 도커 엔진

> 💡 도커 엔진은 이미지 및 컨테이너를 생성하고 실행하는 코어 기능을 수행하는 컴포넌트입니다. 흔히 우리가 도커라고 부르는 것은 '도커 엔진'을 가리키는 말이죠.
> 이미지와 컨테이너가 만들어내는 개발 환경의 신세계로 들어가 봅시다.

도커를 설치하는 일은 아주 간단합니다. Windows/macOS/Linux 등 운영체제를 모두 지원하며, Windows와 macOS에서는 클라이언트를 설치하여 GUI로 손쉽게 사용이 가능합니다. 본 강의에서는 개발환경 통일 및 CLI 사용법을 익히기 위해 Linux 배포판인 Ubuntu Server 에서 실습을 진행하겠습니다.

> 💡 **MacOS의 경우 CPU의 종류에 따라 실습하는 방법에 다소 차이가 있습니다.**
> **실습 전 확인 부탁드립니다** 🙂

▼ **MacOS(Intel)**

1. VMware Fusion 을 다운로드 받아 설치합니다.
   - 다운로드 링크
2. Ubuntu Server 이미지를 다운로드 받습니다.
   - 다운로드 링크
3. VMware Fusion 에서 VM 생성 후, Ubuntu Server 를 설치합니다.
4. VM의 Ubuntu Server 에서 Docker Engine 을 설치합니다.
   - 3, 4의 과정은 윈도우에 Docker Engine 을 구축하는 것과 동일하므로 `강의 3-2` 를 참고해주시면 됩니다.
5. Mac에서 생성된 VM에 원격 접속한 후 `sudo docker version` 을 실행하여 Docker Engine 이 정상적으로 설치 됐는지 확인합니다.

▼ **MacOS(Apple Chip - M1)**

1. Docker Desktop (Apple Chip) 용을 다운로드 받아 설치합니다.
   - 다운로드 링크
2. Docker Desktop을 실행합니다.
3. 실행 완료 후 터미널에 `sudo docker version` 명령어를 작성하여 정상적으로 설치가 되었는지 확인합니다.
4. 강의에서 진행되는 각종 명령어는 별도 원격 접속 없이 터미널에 바로 입력하여 진행합니다.

3.1 도커 기본 명령

목차 0.3의 가이드대로 도커가 설치된 VM에 원격 접속을 해주시고 다음의 명령어를 실행해 보면서 도커와 친해지는 시간을 가져보겠습니다.

리눅스 환경에서 도커는 과거 TCP port 방식에서 UNIX Socket에 바인딩되는 방식으로 변경된 이후 root 권한을 필요로 합니다. 즉, 모든 docker 명령어 앞에 `sudo` 를 반드시 붙여야 한다는 뜻이죠.

3.1.1 도커 버전 확인하기

```
$ sudo docker version
```

```
ggingmin@ubuntu_server:~$ sudo docker version
[sudo] password for ggingmin:
Client: Docker Engine - Community
 Version:           20.10.7
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        f0df350
 Built:             Wed Jun  2 11:56:40 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.7
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       b0f5bc3
  Built:            Wed Jun  2 11:54:48 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.6
  GitCommit:        d71fcd7d8303cbf684402823e425e9dd2e99285d
 runc:
  Version:          1.0.0-rc95
  GitCommit:        b9ee9c6314599f1b4a7f497e1f1f856fe433d3b7
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

현재 제가 실습하고 있는 도커의 버전은 20.10.7 커뮤니티 버전이고 아래에 기타 정보들이 나오는 걸 확인할 수 있습니다.

2.1.2 도커 시스템 정보 확인하기

```
$ sudo docker system info
```

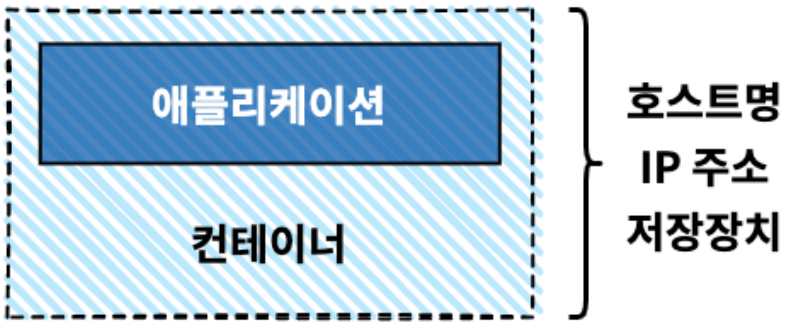
```
ggingmin@ubuntu_server:~$ sudo docker system info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Build with BuildKit (Docker Inc., v0.5.1-docker)
  scan: Docker Scan (Docker Inc., v0.8.0)

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
 Server Version: 20.10.7
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 1
```

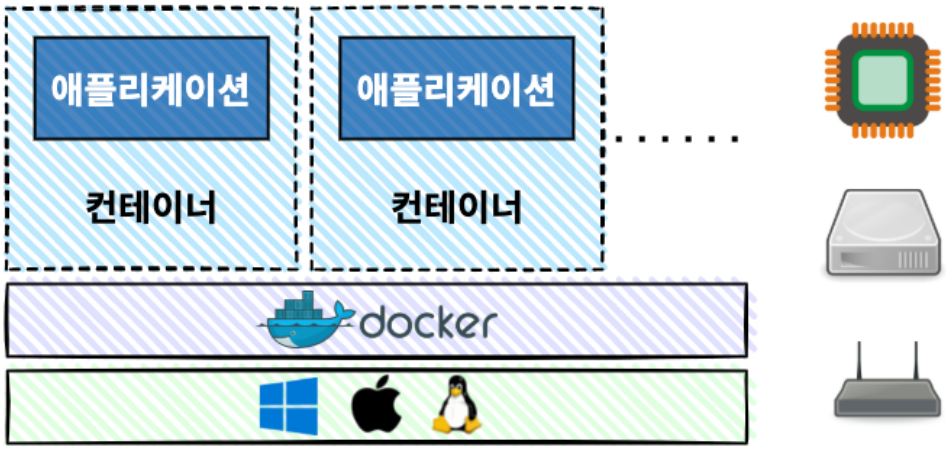
현재 제가 구동 중인 도커에는 컨테이너나 이미지가 없기 때문에 모두 0이라고 나옵니다. 추후 실습을 진행하면서 어떤 변화가 있는지 살펴보도록 하겠습니다.

3.2 도커 컨테이너

컨테이너를 아는 것이 곧 도커를 아는 것이라 할 정도로 컨테이너는 매우 중요한 개념입니다. 하지만 중요하다고 해서 어렵게 받아들일 필요는 없습니다. 그 구조를 보면 오히려 우리가 사용하고 있는 PC보다 간단하기 때문입니다. 아래 다이어그램을 한번 보겠습니다.



컨테이너는 도커가 설정한 네트워크 설정 및 저장장치에 따라 구성됩니다. 애플리케이션은 바로 이 가상의 공간에서 실행되죠. 컨테이너는 완벽하게 애플리케이션 실행 환경에 맞게 구성됩니다. 좀 더 멀리서 컨테이너를 바라보겠습니다.



이 사각형 전체를 컴퓨터라고 생각해봅시다. CPU, 저장장치, LAN 카드, 그래픽 카드 등의 하드웨어가 컴퓨터를 물리적으로 구성하고 있습니다. 여기에 OS를 기반으로 하여 도커 엔진이 구동되고, 컨테이너는 바로 이 도커 엔진의 통제에 의해 움직입니다.

언뜻 보면 VM을 구동하는 것과 비슷해보이지만 컨테이너에는 기본적으로 OS가 없습니다. 애플리케이션을 구동하기 위해 필요한 라이브러리만을 개별적으로 가지고 있을 뿐입니다. OS를 실행할 필요가 없기 때문에 부하가 적고 구동되는 속도 역시 빠릅니다. 또한, 애플리케이션은 각자 독립된 영역을 보장받기 때문에 런타임에서 발생하는 충돌을 피할 수 있습니다.

2.2.1 컨테이너 실행하기

기다리던 순간이죠. 직접 도커 엔진에 컨테이너를 실행해보도록 하겠습니다.

1) 대화형 컨테이너 실행

첫번째로 컨테이너에 올릴 시스템은 리눅스 베포판 중 하나인 CentOS 입니다.

```
sudo docker container run --interative --tty --name centos centos:latest
```

```
sudo docker container run -i -t --name centos centos:latest
```

```
sudo docker container run -it --name centos centos:latest
```

⇒ 위 세 가지 명령은 모두 같은 의미이며, 축약형 옵션의 사용여부에만 차이가 있습니다.

위 명령어를 보면 대충 도커 컨테이너를 실행하는 것 같은데 수상한 옵션이 붙어있습니다. 이 두 가지 옵션은 컨테이너에 설치된 애플리케이션에 콘솔로 명령을 하기 위함이며, 자세한 의미는 다음과 같습니다.

> `--interactive, -i` 표준 입력창을 엽니다.
> `--tty, -t` 장치에 tty를 할당합니다.
> `--name` 컨테이너의 이름을 세팅합니다. (이 옵션을 사용하지 않을 경우 영어 단어를 임의로 조합하여 랜덤하게 세팅됩니다.)

위 두 옵션을 붙이지 않을 경우 터미널에서 명령어를 통한 컨테이너 제어가 불가능하므로 유의해야 합니다.

```
ggingmin@ubuntu_server:~$ sudo docker container run -it --name centos centos:latest
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
7a0437f04f83: Pull complete
Digest: sha256:5528e8b1b1719d34604c87e11dcd1c0a20bedf46e83b5632cdeac91b8c04efc1
Status: Downloaded newer image for centos:latest
[root@9dc608d332c5 /]#
```

명령어를 통해 생성된 컨테이너에 centos 가 설치되었고, 이어 root 권한으로 원격 접속이 된 것을 확인할 수 있습니다.
원격 접속을 종료하려면 `exit` 명령어를 입력하면 되며 해당 컨테이너도 종료 상태로 바뀌게 됩니니다.

2) 백그라운드 컨테이너 실행

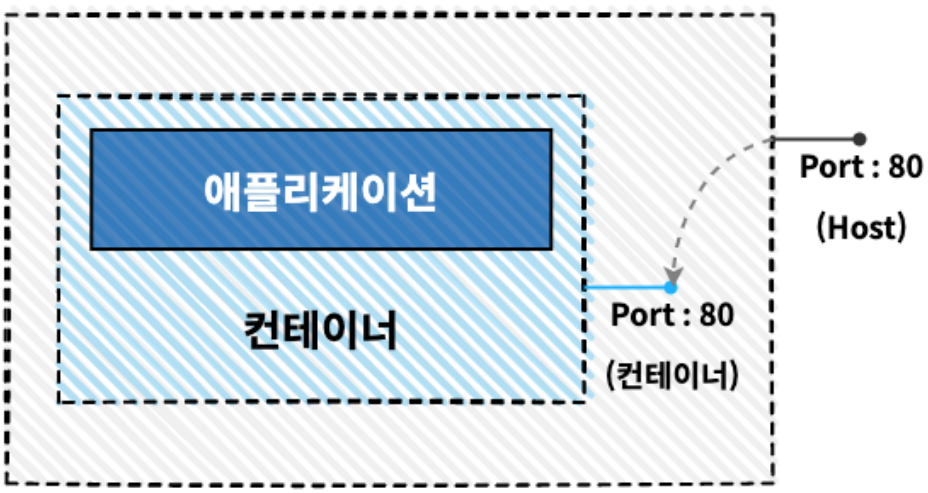
두번째로 컨테이너에 올릴 대상은 웹서버인 httpd입니다.

```
sudo docker container run -d -p 80:80 --name apache httpd:latest
```

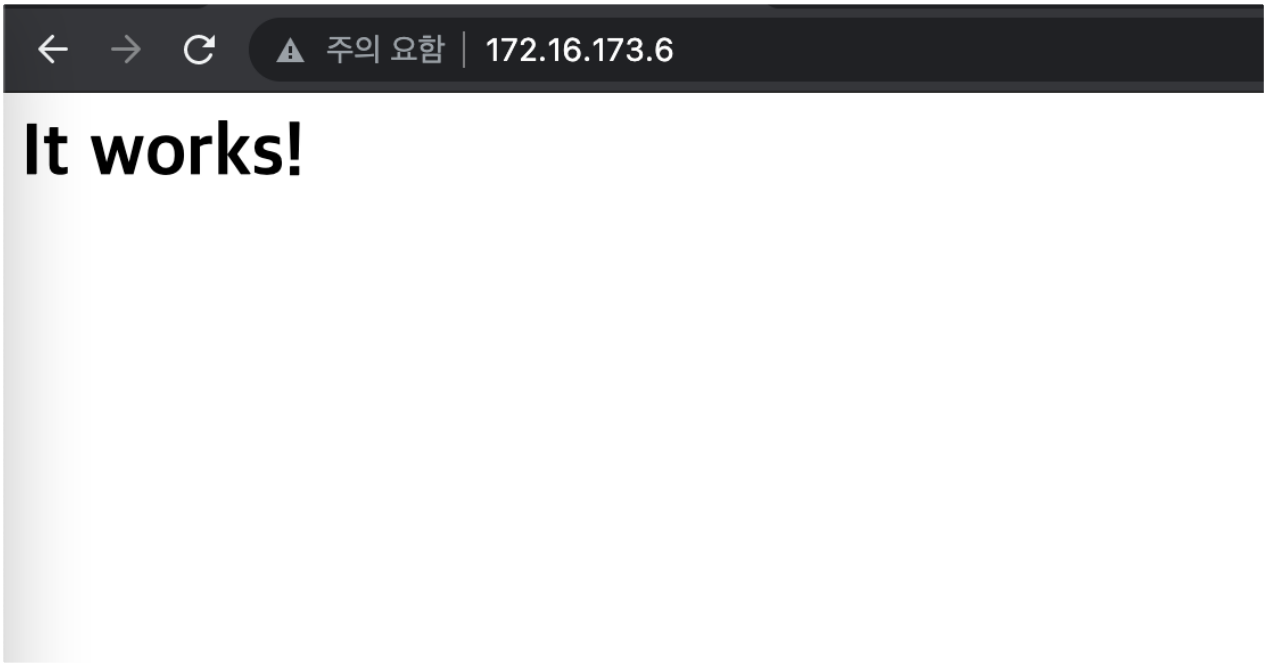
앞에서 봤던 것과 다른 옵션이 등장했습니다. 먼저 각각의 기능부터 살펴보겠습니다.

> `--detach, -d` 백그라운드에서 컨테이너를 실행합니다.
> `--publish, -p` 호스트/컨테이너 포트포워딩을 세팅합니다.

명령어를 해석해보면,
컨테이너를 "백그라운드"에서 실행하고 "호스트의 80번 포트를 컨테이너의 80번 포트로 포워딩" 한다.
실행 대상 이미지는 "httpd"의 "최신 버전"이고 컨테이너의 이름은 "apache"로 붙인다.
라고 할 수 있겠군요.



브라우저를 열고 `http://(컨테이너 IP):80` 를 입력하면 Apache 서버가 설치된 컨테이너의 80번 포트로 포워딩 되어 다음과 같이 정상적으로 접속이 됨을 확인할 수 있습니다.



컨테이너 내부에서 어떤 일이 일어나는지는 다음과 같이 로그를 조회하는 명령어로 확인이 가능합니다.

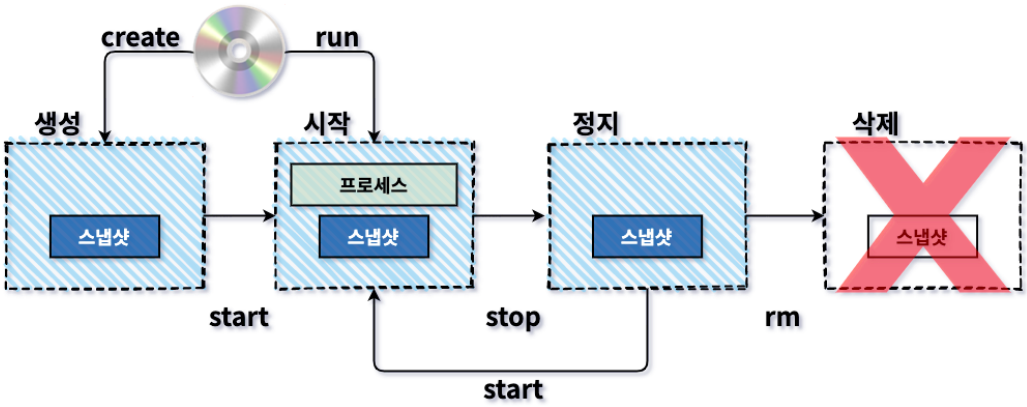
```
sudo docker container logs apache
```

`logs` 뒤에 로그를 조회하고자 하는 컨테이너 명을 입력하면 됩니다. 로그는 다음과 표시됩니다.

```
ggingmin@ubuntu_server:~$ sudo docker container logs apache
AH00558: httpd: Could not reliably determine the server's fully qualified doma
in name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress
 this message
AH00558: httpd: Could not reliably determine the server's fully qualified doma
in name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress
 this message
[Wed Jun 16 11:40:36.047351 2021] [mpm_event:notice] [pid 1:tid 14022490737574
4] AH00489: Apache/2.4.48 (Unix) configured -- resuming normal operations
[Wed Jun 16 11:40:36.047483 2021] [core:notice] [pid 1:tid 140224907375744] AH
00094: Command line: 'httpd -D FOREGROUND'
172.16.173.1 - - [16/Jun/2021:11:42:46 +0000] "GET / HTTP/1.1" 304 -
172.16.173.1 - - [16/Jun/2021:11:45:18 +0000] "GET / HTTP/1.1" 304 -
```

3.2.3 컨테이너 생명 주기

'생명 주기'라는 용어는 프로그래밍 학습에서 자주 등장합니다. 가장 친숙한 예를 들어보자면 Java 쓰레드의 생명 주기가 있겠네요. 도커에서 제어하는 컨테이너도 이러한 생명 주기를 갖습니다. 다음의 그림은 컨테이너의 생명 주기를 나타냅니다.



우리가 앞에서 살펴본 명령어인 `run` 이 보입니다. 컨테이너를 생성함과 동시에 실행을 수행했었죠. `create` 는 컨테이너를 생성만 하고 시작은 하지 않습니다. 시작을 위해서는 별도로 `start` 명령을 내려야합니다. `rm` 은 컨테이너를 삭제하는 명령어로서, 정지된 컨테이너에 한해 삭제가 가능합니다. 만약 실행 중인 컨테이너를 삭제하고자 한다면 강제 삭제를 위한 옵션을 추가로 입력해야 합니다.

아래 이어서 컨테이너를 다루는 명령어를 살펴보겠습니다.

3.2.4 컨테이너 명령

컨테이너 명령어를 실습하기 전에 각각의 옵션 명세를 먼저 전달드립니다. 대표적으로 많이 쓰이는 것들 위주로 정리하였으며, 전체 옵션에 대해서는 도커 공식 문서 링크를 남겨드립니다.

▼ 도커 공식 문서

https://docs.docker.com/engine/reference/run/

1) `run` - 컨테이너 생성 & 실행

```
sudo docker container run -d -p 80:80 --name apache httpd:latest
```

2) `stop` - 컨테이너 중지

```
sudo docker container stop apache
```

3) `start` - 컨테이너 시작

▼ 옵션

start

Aa 명령어	이름	기능
	`--detach` , `-d`	컨테이너 생성 후 백그라운드에서 실행한다.
	`--interactive` , `-i`	표준 입력창을 엽니다.

개수 2

```
sudo docker container start apache
```

4) `restart` - 컨테이너 재시작

```
sudo docker container restart apache
```

5) `stats` - 컨테이너 구동 확인

▼ 옵션

stats

Aa 명령어	이름	기능
	`--all`, `-a`	실행/중지 된 모든 컨테이너의 상태를 확인합니다.
	`--format`	출력 포맷을 설정합니다.

개수 2

```
sudo docker container stats apache
```

```
sudo docker container stats apache --format "table
{{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}"
```

6) `ls` - 컨테이너 목록 조회

▼ 옵션

stats

Aa 명령어	이름	기능
	`--all`, `-a`	실행/중지 된 모든 컨테이너의 상태를 확인합니다.
	`--format`	출력 포맷을 설정합니다.

개수 2

```
sudo docker container ls
```

7) `attach` - 구동 중인 컨테이너 연결

```
sudo docker container attach apache
```

8) `exec` - 구동 중인 컨테이너에서 프로세스 실행

▼ 옵션

exec

Aa 명령어	이름	기능
	`--detach` , `-d`	컨테이너 생성 후 백그라운드에서 실행한다.
	`--interactive` , `-i`	표준 입력창을 엽니다.
	`--tty` , `-t`	장치에 tty를 할당합니다.
	`--user` , `-u`	컨테이너 실행 시, 사용자명이나 지정합니다.

개수 4

```
sudo docker container exec -it apache /bin/echo "Hello, Docker!"
```

```
sudo docker container exec -it apache bash
```

9) `top` - 컨테이너 내부에서 구동 중인 프로세스 확인

```
sudo docker container top apache
```

10) `rename` - 컨테이너 이름 변경

```
sudo docker rename apache apache_server
```

11) `cp` - 컨테이너 내부 파일 복사

```
sudo docker container cp apache:/usr/local/apache2/htdocs/index.html /tmp/index.html
```

```
sudo docker container cp /tmp/index.html apache:/usr/local/apache2/htdocs/index.html
```

12) `diff` - 컨테이너 변경 사항 확인

```
sudo docker diff apache
```