Docker 101_2

도커 구조

저번 시간에 우리는 `docker` 라는 명령어를 이용하여 도커를 사용해왔다.

도커 명령어 위치를 알아보자.

```
$ which docker
/usr/bin/docker
```

실행 중인 도커 프로세스를 확인해보자.

```
$ ps aux | grep docker
/usr/bin/dockerd --raw-logs
```

컨테이너와 이미지를 다루는 명령어는 `/usr/bin/docker` 에서 실행.
도커 엔진의 프로세스는 `/usr/bin/dockerd` 파일로 실행되고 있음.

=> docker 명령어가 실제 도커 엔진이 아닌 클라이언트로서의 도커.

정리하자면, 도커는 크게 2가지로 나뉜다.

1. 서버로서의 도커
2. 클라이언트로서의 도커

서버로서의 도커

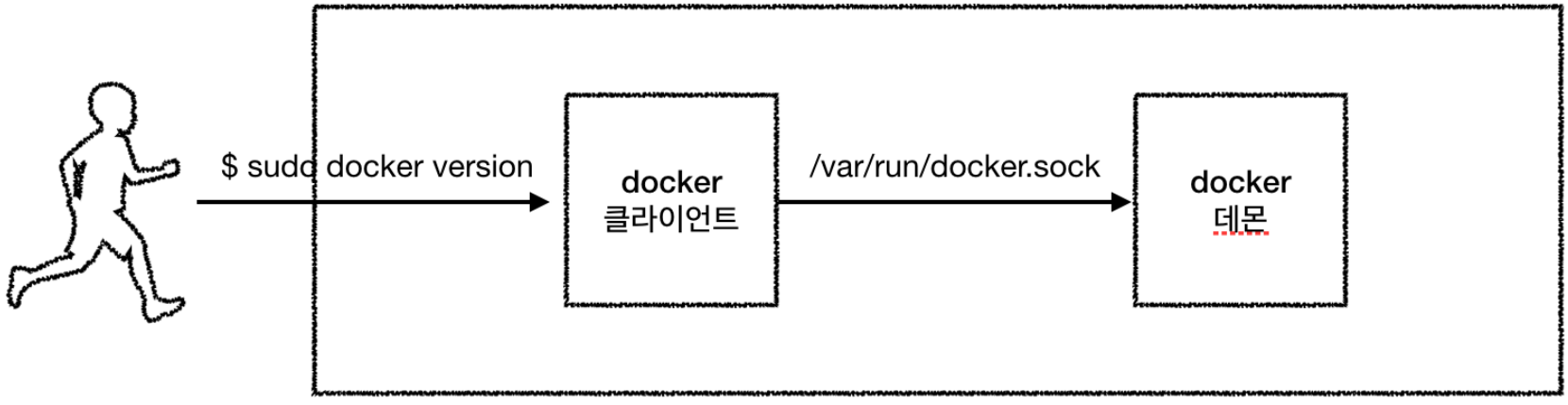
이미지를 관리해주고 컨테이너를 생성하고 실행하는 주체는 **도커 서버**.
이는 dockerd 프로세스로 동작.

도커 엔진은 외부에서 API 입력을 받아 도커 엔진의 기능을 수행하는데, 도커 프로세스가 실행되어 서버로서 입력을 받을 준비가 된 상태를 **도커 데몬** 이라고 한다.

클라이언트로서의 도커

도커 데몬은 API 입력을 받아 도커 엔진의 기능을 수행.
이 API 를 사용할 수 있도록 CLI 를 제공하는 것이 도커 클라이언트

도커 클라이언트와 도커 데몬



1. 사용자가 `docker` 명령어를 입력 = 도커 클라이언트 사용
2. 도커 클라이언트는 `/var/run/docker.sock` 에 위치한 소켓을 통해 도커 데몬의 API 를 호출
3. 도커 데몬은 이 명령어를 파싱하고 이에 해당하는 작업을 수행.
4. 수행 결과를 도커 클라이언테 반환하고 사용자에게 결과를 출력

일반적인 도커 데몬을 제어하는 순서를 살펴보았다.

각종 옵션을 추가해 실행한다면
위 순서에 별도의 과정이 포함될 수도 있다.

도커 클라이언트가 사용하는 소켓을 이용하여 도커 데몬에게 명령을 전달하였는데 이 뿐만아니라 tcp 원격으로 도커 데몬을 제어하는 방법도 있다.

도커 이미지 구조 이해

저번 시간에 이미지와 컨테이너를 설명하였다.

모든 컨테이너는 이미지를 기반으로 생성된다.
이미지를 다루는 방법은 도커 관리에서 중요한 요소라고 할 수 있다.

따라서 이에 대해 좀 더 자세히 알아보자.

Git 설치

우선 우분투 컨테이너(이름은 git으로) 하나 실행하고 git 을 설치해보자.

```
$ sudo docker run -it --name git ubuntu:14.04 /bin/bash

root@a2fb46c1e158:/# git
bash: git: command not found

root@a2fb46c1e158:/# apt-get update
root@a2fb46c1e158:/# apt-get install -y git
root@a2fb46c1e158:/# git --version
git version 1.9.1
```

Git 설치 후 상태 변화 확인

diff 명령어로 Base Image와 컨테이너의 차이를 파악

```
$ sudo docker diff git | grep git | head -n 10
A /etc/bash_completion.d/git-prompt
A /var/lib/dpkg/info/git.conffiles
A /var/lib/dpkg/info/git.prerm
A /var/lib/dpkg/info/git.md5sums
A /var/lib/dpkg/info/git.list
A /var/lib/dpkg/info/git.preinst
A /var/lib/dpkg/info/git-man.md5sums
A /var/lib/dpkg/info/git.postinst
A /var/lib/dpkg/info/git-man.list
A /var/lib/dpkg/info/git.postrm
```

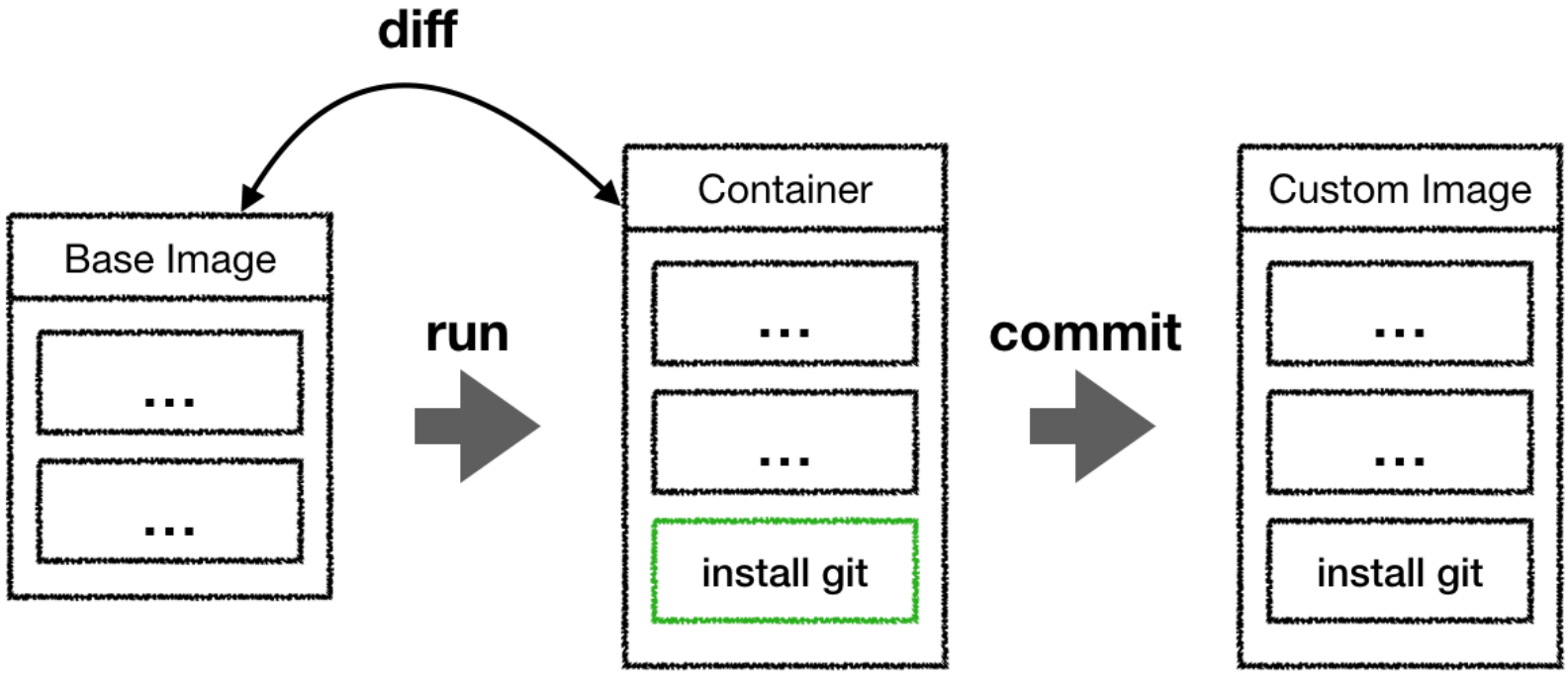
commit 명령어로 새로운 이미지 생성

```
$ sudo docker images | grep ubuntu

$ sudo docker commit git ubuntu:git
sha256:e77e4fa2335ca6a820631800780 7bd04e32ca927df7d85bdb4

$ sudo docker images | grep ubuntu
```

`images` 명령어를 통해 `commit` 전과 후를 파악.
새로운 이미지가 생성된 것을 확인 할 수 있다.


diff
Base Image
...
...
run
Container
...
...
install git
commit
Custom Image
...
...
install git

`inspect` 명령어로 좀 더 자세한 정보를 확인해보자.

```
$ sudo docker inspect ubuntu:14.04
$ sudo docker inspect ubuntu:git
```

우리가 볼 항목은 Layers 이다.

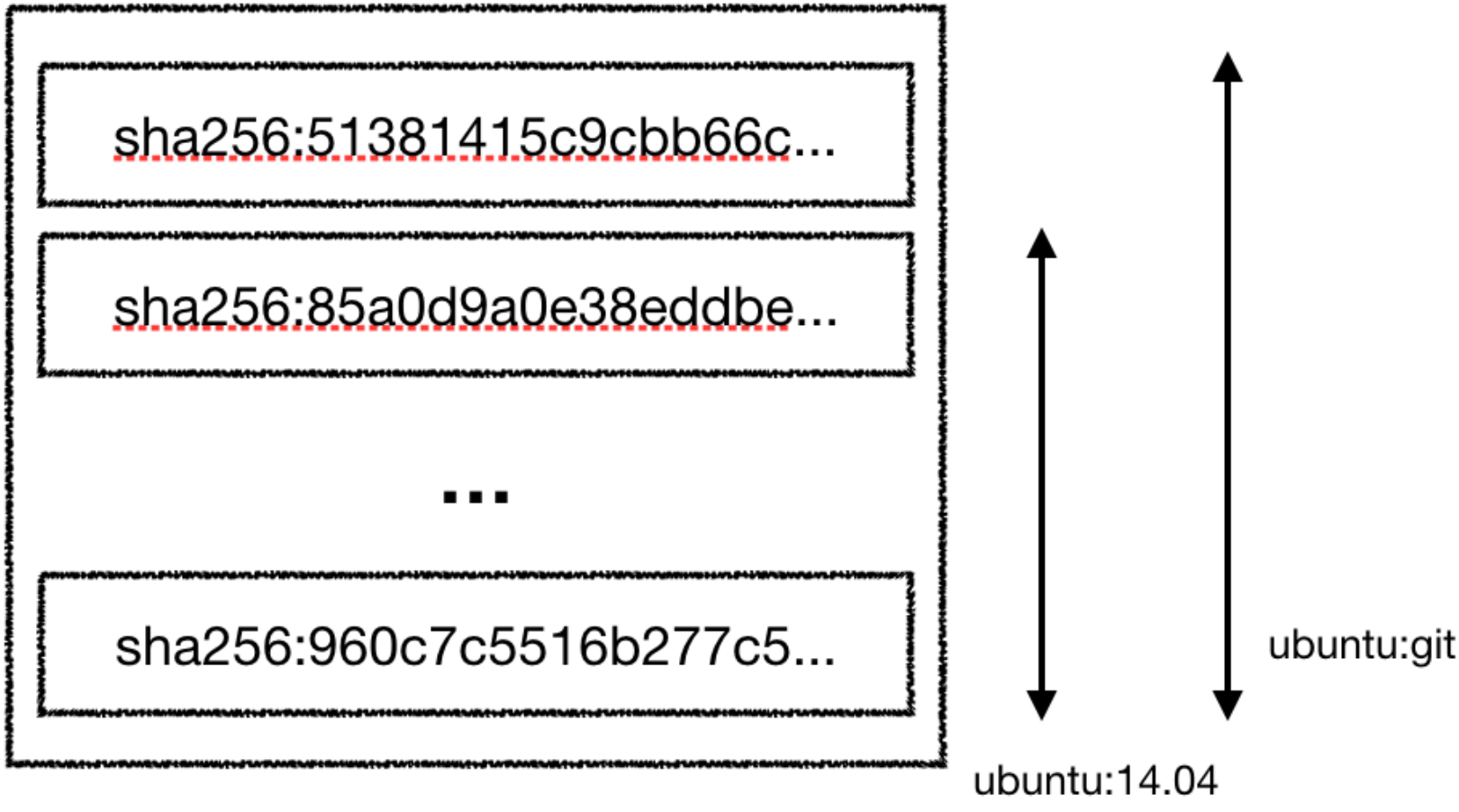
ubuntu:14.04

```
"Layers": [
  "sha256:960c7c5516b277c5...",
  "sha256:ea213108ab366bfe...",
  "sha256:fe72fc94fa1a8eba...",
  "sha256:85a0d9a0e38eddbe..."
]
```

ubnutu:git

```
"Layers": [
  "sha256:960c7c5516b277c5...",
  "sha256:ea213108ab366bfe...",
  "sha256:fe72fc94fa1a8eba...",
  "sha256:85a0d9a0e38eddbe...",
  "sha256:51381415c9cbb66c..."
  ]
```

이를 좀더 보기 쉽게 그림으로 표현하면 다음과 같다.



이미지를 `commit` 할 때 컨테이너에서 변경된 사항만 새로운 layer 로 저장하고 그 layer 를 포함해 새로운 이미지가 생성되는 것이다.