

도커란

도커란 무엇인가?

- 도커는 2013년에 등장한 컨테이너 기반 가상화 도구입니다.
- 파이콘 US 2013^{Pycon US 2013}에서 솔로몬 하이크^{Solomon Hykes}는 [리눅스 컨테이너의 미래](#)^{The Future of Linux Container}라는 라이트닝 토크에서 도커를 처음 소개했습니다.
- 현재 도커는 GO 언어로 개발되고 있으며, 2014년 [도커콘 2014](#)^{DockerCon 2014}에서 1.0 버전을 발표하였습니다.

도커란 무엇인가?

- 도커는 리눅스 상에서 컨테이너 방식으로 프로세스를 격리해서 실행하고 관리할 수 있도록 도와주며, 계층화된 파일 시스템에 기반해 효율적으로 이미지(프로세스 실행 환경)을 구축할 수 있도록 해줍니다.
- 도커를 사용하면 이 이미지를 기반으로 컨테이너를 실행할 수 있으며, 다시 특정 컨테이너의 상태를 변경해 이미지로 만들 수 있습니다. 이렇게 만들어진 이미지는 파일로 보관하거나 원격 저장소를 사용해 쉽게 공유할 수 있으며, 도커만 설치되어 있다면 필요할 때 언제 어디서나 컨테이너로 실행하는 것이 가능합니다.

도커 컨테이너

- 도커 컨테이너는 일종의 소프트웨어를 소프트웨어의 실행에 필요한 모든 것을 포함하는 완전한 파일 시스템 안에 감싼다. 여기에는 코드, 런타임, 시스템 도구, 시스템 라이브러리 등 서버에 설치되는 무엇이든 아우릅니다.
- 이는 실행 중인 환경에 관계 없이 언제나 동일하게 실행될 것을 보증한다는 의미입니다.(위키백과)

왜 도커(Docker)를 써야하나요?

- 컨테이너는 애플리케이션을 환경에 구매 받지 않고 실행하는 기술입니다.
- 우리가 호스팅시 사용하는 시스템(운영체제)마다 각기 다른 명령어를 사용해야 합니다.
- 사용하는 os가 어떤 종류인지에 따라 설치방법이가 실행명령어가 달라져야 하는데 컨테이너 도구인 도커가 설치되어 있다면 어느 환경이든 상관 없이 동일한 명령어를 이용하여 관리 배포할 수 있습니다.

운영하면서 만들어지는 눈송이 서버들(Snowflake Servers)

- 각 서버마다 운영기록이 다르다
- 똑같은 일을 하는 두 서버가 있다 해도, A 서버는 한 달 전에 구성했고 B 서버는 이제 막 구성했다면, 운영체제부터 컴파일러, 설치된 패키지까지 두 서버가 완벽하게 일치하기는 어렵습니다.
- 두 서버간 차이점들이 각종 장애를 일으킵니다.
- 이렇게 서로 모양이 다른 서버들이 존재하는 상황을 **눈송이 서버**라고 합니다.
- 모든 눈송이의 모양이 다르듯, 서버들도 서로 다른 모습이라는 의미입니다.

운영하면서 만들어지는 눈송이 서버들(Snowflake Servers)

- A 서버와 B 서버에 이미지매직ImageMagick이라는 도구를 설치한다고 가정해봅시다. A 서버는 지난 달에 서버를 구성하면서 이미지매직을 설치했고, B 서버는 이제 막 구성한 상황이니 새로 이미지매직을 설치했습니다. 그리고 웹 서비스를 업데이트하여 각 서버에 배포합니다. A 서버에서 장애가 발생했습니다. 이 상황에서 장애 발생 원인은 대략 다음과 같이 추측할 수 있습니다.
 - 웹 서비스에서 이미지매직 최신 버전에 새로 추가된 기능을 사용했다.
 - 웹 서비스의 업데이트 부분 코드에 버그가 있다
 - 이미지매직의 버전이 다르다
 - 이미지매직이 의존하는 라이브러리의 버전이 다르다

운영하면서 만들어지는 눈송이 서버들(Snowflake Servers)

- 장애 원인을 빨리 찾으려면 A 서버를 잘 아는 사람이 필요할텐데 마침 A 서버를 구성했던 사람이 팀을 옮겼거나 퇴사해서 옆에 없습니다. B 서버를 구성한 작업자는 A 서버가 구성되고 운영된 모든 과정을 파악하려고 애를 쓸 겁니다. 그래야 B 서버와의 차이점을 알아낼 수 있으니까요.
- 여기서는 두 서버 간 구성 시점이 한 달 밖에 차이나지 않지만, 막상 현업에서는 몇 년씩 차이나는 경우도 있습니다. 처음엔 각 서버 사이의 차이점이 눈송이 하나 만큼 작을지 모르지만, 서버를 오래 운영하다보면 점점 커져서 나중엔 집을 삼키는 거대한 눈덩이가 되어 있을 겁니다.

왜 도커(Docker)를 써야하나요?

- 도커란 **서버 운영 기록을 코드화**한것입니다.
- 도커 파일로 도커 이미지를 만들 수 있습니다.
- 도커 파일이 **서버 운영 기록**이라면, 도커 이미지는 운영 기록을 **실행할 시점**이라고 할 수 있습니다.

도커 파일 = 서버 운영 기록 코드화

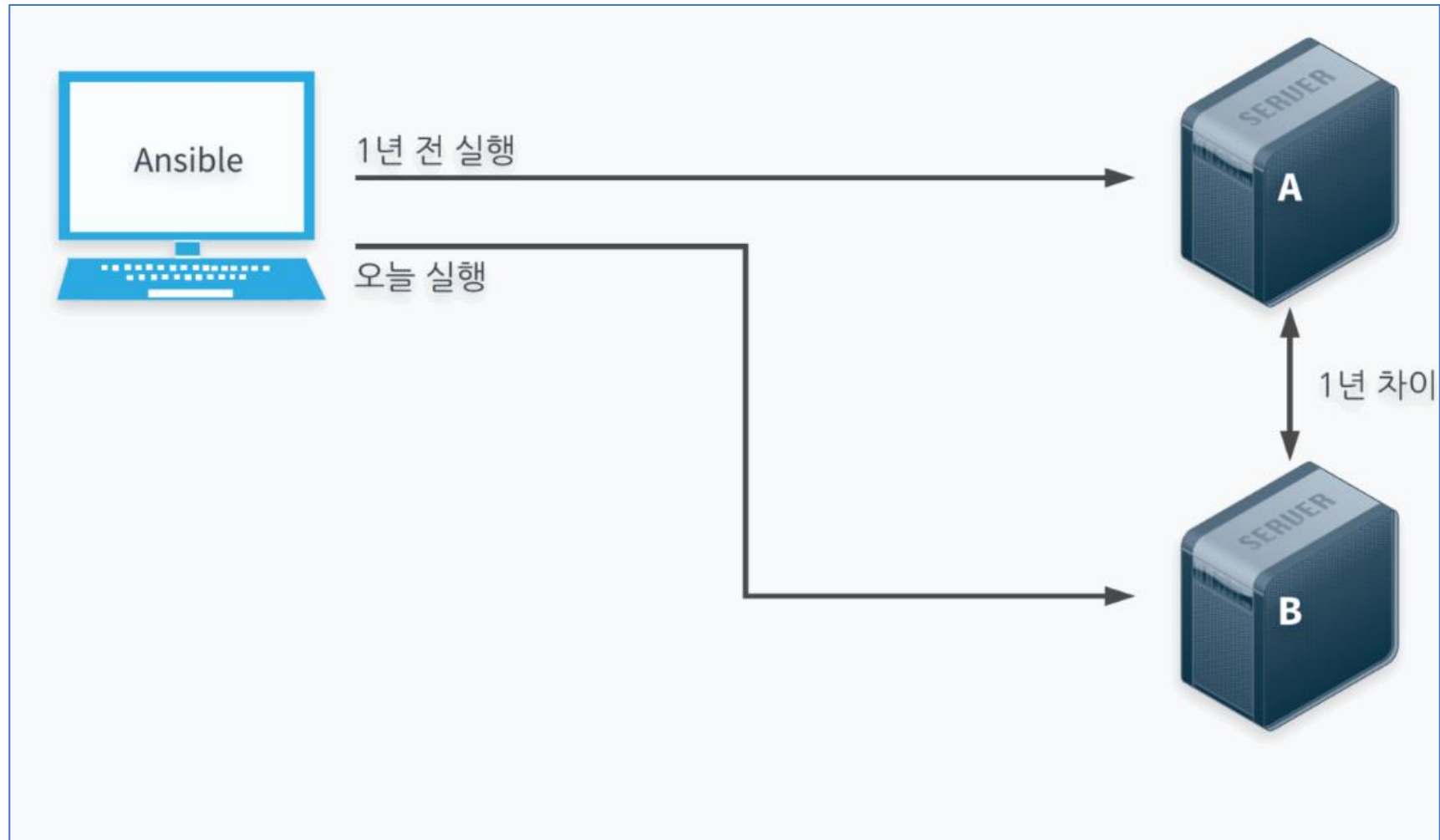
도커 이미지 = 도커 파일 + 실행 시점

도커 컨테이너 = 도커 이미지 + 환경 변수

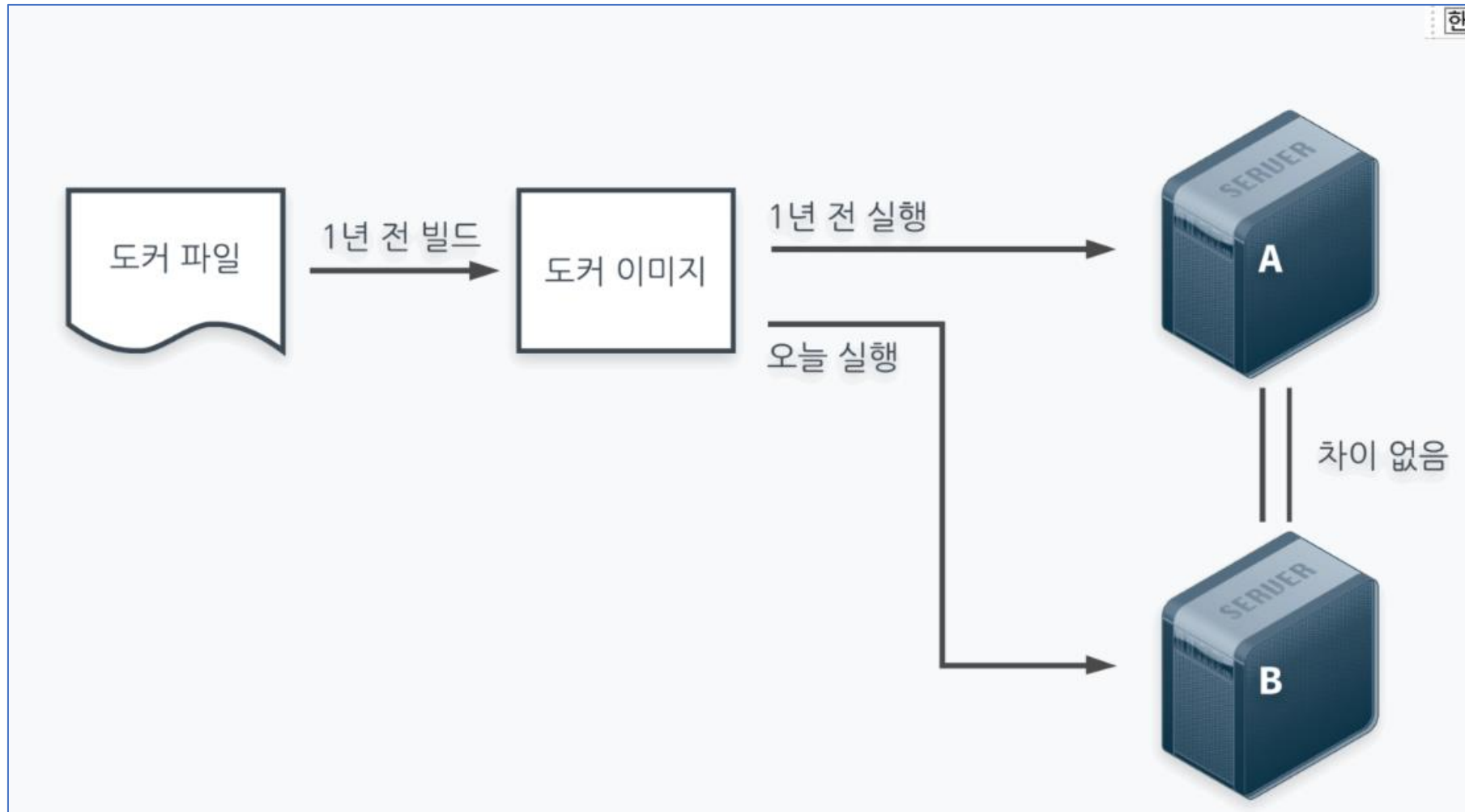
왜 도커(Docker)를 써야하나요?

- 작업자가 1년 전에 이 **서버 운영 기록을 바탕으로** 서버 A를 구성했고, 오늘 서버 B를 구성한다면, 두 서버에 대해 이미지매직이 설치된 시점은 1년의 차이가 발생합니다.
- 하지만 도커에서는 서버 A를 구성한 시점에서 도커 파일로 이미지를 만들어 두면, 서버가 구성되는 시점이 이미지를 만든 시점으로 고정됩니다. 이 이미지를 사용해 1년 전에 A 서버에 컨테이너를 배포하고, 오늘 B 서버에 컨테이너를 배포한다고 해도, 두 컨테이너 모두 이미지매직이 설치된 시점은 같습니다.

도커를 사용하지 않을 경우



도커를 사용할 경우



도커란

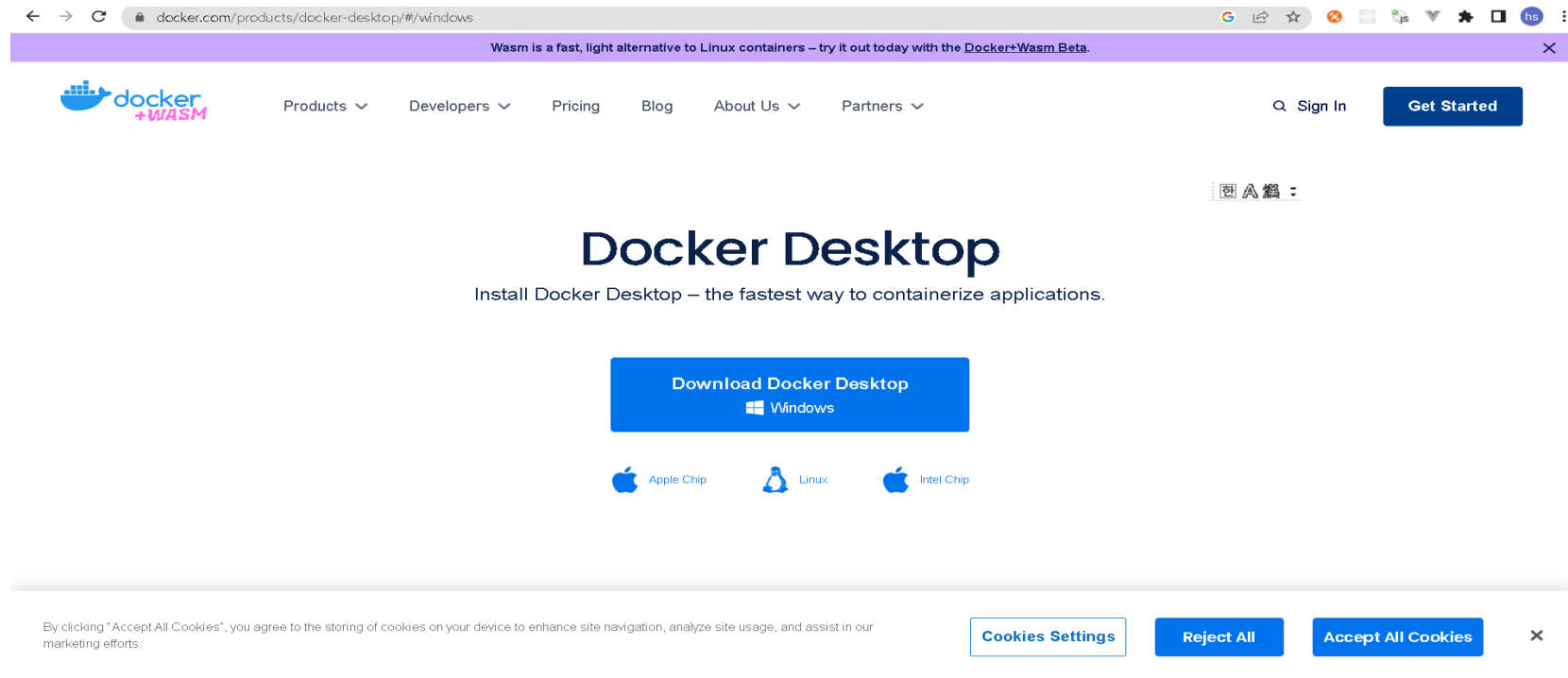
- 컨테이너는 하드웨어를 소프트웨어로 재구현하는 가상화(= 가상머신)와는 달리 프로세스의 실행 환경을 격리합니다. 컨테이너가 실행되고 있는 호스트 입장에서 컨테이너는 단순히 프로세스에 불과합니다. 사용자나 컨테이너 입장에서는 호스트와는 무관하게 동작하는 가상 머신처럼 보입니다. 그래서 컨테이너형 가상화라고 부르기도 합니다.
- 도커는 이러한 컨테이너 형 가상화를 지원하는 도구 중 하나입니다.
- 도커는 가상머신의 역할을 넘어서 어느 플랫폼에서나 특정한 상태를 그대로 재현가능한 애플리케이션 컨테이너를 관리하는 도구를 목표로 합니다.
- LXC(리눅스 컨테이너)로부터 파생된 도커 컨테이너는 가상머신과는 근본적으로 다른 접근 방법입니다.

도커란

- 도커는 가상머신과 같이 하드웨어를 가상화하는 것이 아니라, 리눅스 운영체제에서 지원하는 다양한 기능을 사용해 컨테이너(하나의 프로세스)를 실행하기 위한 별도의 환경(파일 시스템)을 준비하고, 리눅스 네임스페이스와 다양한 커널 기능을 조합해 프로세스를 특별하게 실행시켜줍니다

도커(Docker) 설치하고 기본적인 설정하기

- <https://www.docker.com/products/docker-desktop/#/windows>



우분트에 도커 설치하기 1

- <https://docs.docker.com/engine/install/ubuntu/>
- 필요한 라이브러리
 - apt-transport-https
 - ca-certificates
 - curl
 - software-properties-common

우분트에 도커 설치하기 2

- 라이브러리 설치

```
sudo apt-get update
```

```
sudo apt-get install \
```

```
    ca-certificates \
```

```
    curl \
```

```
    gnupg \
```

```
    lsb-release
```

우분트에 도커 설치하기 3

- 도커의 공식 GPG 키 설정

```
sudo mkdir -m 0755 -p /etc/apt/keyrings
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --  
dearmor -o /etc/apt/keyrings/docker.gpg
```

우분트에 도커 설치하기 4

- 레포지토리 추가

```
echo \
```

```
"deb [arch=$(dpkg --print-architecture) signed-  
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu  
\
```

```
$(lsb_release -cs) stable" | sudo tee  
/etc/apt/sources.list.d/docker.list > /dev/null
```

우분트에 도커 설치하기 5

- 도커 설치하기 (CE -무료 , EE-상용)
- `sudo apt-get update`

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-  
plugin docker-compose-plugin
```

우분트에 도커 설치하기 6

- GPG 오류 발생시

```
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

```
sudo apt-get update
```

우분트에 도커 설치하기 7

- 도커 설치 테스트

- `sudo docker run hello-world`

```
conan@conan-virtual-machine:~$ sudo service docker start
conan@conan-virtual-machine:~$ sudo docker run hello-world

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/
```

도커기본예제(image 불러오기)

- <https://hub.docker.com/>
- `sudo docker search centos`
- 이미지 당겨오기
- `docker pull centos:latest`
- `docker run -i -t centos /bin/bash`
- `cat /etc/os-release`
- `ctrl-shift-d` 종료
- `ctrl-shift-P` `ctrl-shift-Q`

도커기본예제(image 불러오기) 2

```
docker image pull gihyodocker/echo:latest
```

```
sudo docker container run -t -p 9000:8080 gihyodocker/echo:latest
```

터미널을 하나 더 켜서 아래 명령어를 테스트한다

```
curl http://localhost:9000
```

```
docker container ls
```

```
sudo docker container 컨테이너아이디
```


도커 컨테이너 목록 확인

- `docker container ls -a`
- `docker rm 컨테이너아이디1 아이디2` 컨테이너 삭제
- `docker rmi 이미지아이디` 이미지 삭제

간단한 이미지 만들기

도커예제1

mkdir exam

cd exam

gedit helloworld

#!/bin/sh

echo "Hello, World!"

도커예제1

gedit Dockerfile

FROM ubuntu:16.04

COPY helloworld /usr/local/bin

RUN chmod +x /usr/local/bin/helloworld

CMD ["helloworld"]

Dockerfile

- FROM : 도커 이미지의 바탕이 될 이미지를 지정한다
 - 도커허브 레지스트리에 공개된 이미지이다
- RUN: 도커 이미지를 실행할때 컨테이너 안에서 실행할 명령어를 지정한다. 이미지 빌드시
- COPY : 도커가 동작중인 호스트 머신의 파일이나 디렉토리를 도커 컨테이너 안으로 복사한다
- CMD : 도커가 컨테이너 안에서 실행할 프로세스를 지정한다. 컨테이너 실행시

이미지 빌드(-t 옵션 필수:이미지에 이름부여)

- `sudo docker image build -t example/helloworld .` (경로안의 모든파일)

```
[+] Building 16.4s (8/8) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 146B
=> [internal] load metadata for docker.io/library/ubuntu:16.04
=> [internal] load build context
=> => transferring context: 69B
=> [1/3] FROM docker.io/library/ubuntu:16.04@sha256:1f1a2d56de1d604801a9671f3011 10.6
=> => resolve docker.io/library/ubuntu:16.04@sha256:1f1a2d56de1d604801a9671f30119 0.0
=> => sha256:da8ef40b9ecabc2679fe2419957220c0272a965c5cf7e0269fa1aeeb 528B / 528B 0.0
=> => sha256:1f1a2d56de1d604801a9671f301190704c25d604a416f59e03c0 1.42kB / 1.42kB 0.0
=> => sha256:a3785f78ab8547ae2710c89e627783cfa7ee7824d3468cae6835 1.15kB / 1.15kB 0.0
=> => sha256:b6f50765242581c887ff1acc2511fa2d885c52d8fb3ac8c4bba1 3.36kB / 3.36kB 0.0
=> => sha256:58690f9b18fca6469a14da4e212c96849469f9b1be6661d234 46.50MB / 46.50MB 5.2
=> => sha256:b51569e7c50720acf6860327847fe342a1afbe148d24c529fb81df10 857B / 857B 1.7
=> => sha256:fb15d46c38dcd1ea0b1990006c3366ecd10c79d374f341687eb2cb23 170B / 170B 1.8
=> => extracting sha256:58690f9b18fca6469a14da4e212c96849469f9b1be6661d2342a4bf01 4.7
=> => extracting sha256:b51569e7c50720acf6860327847fe342a1afbe148d24c529fb81df105 0.0
=> => extracting sha256:da8ef40b9ecabc2679fe2419957220c0272a965c5cf7e0269fa1aeeb8 0.0
=> => extracting sha256:fb15d46c38dcd1ea0b1990006c3366ecd10c79d374f341687eb2cb23a 0.0
=> [2/3] COPY helloworld /usr/local/bin
=> [3/3] RUN chmod +x /usr/local/bin/helloworld
=> exporting to image
=> => exporting layers
=> => writing image sha256:385953ed7ab91b06ec497dd0eb68c5b27ec76f7d7f88b8bd47b45a 0.0
=> => naming to docker.io/example/helloworld 0.0
```

permission 에러시

- 사용자를 도커그룹에 추가한다
- \$USER : 현재 로그인한 사용자의 아이디를 가지고 있는 환경변수이다
- `sudo usermod -a -G docker $USER`

목록 확인

- sudo docker image ls

```
conan@conan-virtual-machine:~/다운로드/docker-kubernetes-master/ch01/ch01_1_3$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
example/helloworld  latest             385953ed7ab9       2 minutes ago      135MB
hello-world         latest            feb5d9fea6a5       17 months ago      13.3kB
```


컨테이너에서 이미지 실행하기

- `docker container run example/helloworld`

이미지 관리 명령어

- `docker image --help`

이미지 push

- <https://hub.docker.com/>에 회원가입
- 이름
- 이메일
- 패스워드
- 새로운 레포지토리를 만든다


레포지토리 만들기

← → ↺


hub.docker.com/repository/create?namespace=littleconan

🔖 ☆ 🛠️ 🧠 📄 📁 📌 📱 hs

Wasm is a fast, light alternative to Linux containers – try it out today with the [Docker+Wasm Beta](#). ✕

 Search Docker Hub

Explore Repositories Organizations Help ▾

Upgrade  littleconan ▾

Repositories > Create >

Using 1 of 1 private repositories. [Get more](#)

Create repository

littleconan ▾

Name

Description

Visibility

Using 1 of 1 private repositories. [Get more](#)

☐ Public 🌐
Appears in Docker Hub search results

☒ Private 🔒
Only visible to you

Cancel

Create

Pro tip

You can push a new image to this repository using the CLI

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to change *tagname* with your desired image repository tag.

docker image push

- docker login
- littleconan@hanmail.net/[REDACTED]
- docker image tag exam/helloworld littleconan/test1:0.1.0
- docker push littleconan/test:0.1.0

이미지 당기기

- `docker image ls | grep test1`
- `dorker rmi -f 이미지아이디` (동일한 해쉬를 갖는 이미지가 많아 한번에 지워야한다)
- `docker pull littleconan/test1:0.1.0`
- `docker container run -t littleconan/test1:0.1.0`

참고사이트

- <https://www.44bits.io/ko/post/easy-deploy-with-docker>
- <https://www.44bits.io/ko/post/why-should-i-use-docker-container>