

Ch 2. Hello, Docker!

0. (Teaser) Docker



도커라는 이름을 가진 귀여운 고래마크가 여기저기서 등장하기 시작했습니다.

이 귀여운 고래가 하는 일에도 많은 사람들이 관심을 가지고 지켜보고 있습니다.

그래서, 도커가 뭔지 제대로 알아보기로 했습니다.

0.1 도커는 OO다

"도커는 컨테이너에서 애플리케이션을 실행하고 운영할 수 있도록 하는 플랫폼이다."

도커는 2019/2020 Stack Overflow에서 진행한 설문조사에서 'Most Wanted Platform' 부문에서 1위를 차지했습니다. (연관 기술인 Kubernetes의 경우 3위를 차지했네요.) 이는 모든 분야에 걸쳐 도커의 쓰임새가 널리 확장되고 있다는 의미와 함께, 경쟁력 있는 개발자/아키텍트로 성장하는 데에 있어 학습할만한 가치가 있다는 메세지를 던지고 있습니다.

0.2 마이그레이션

マイグレーション은 모든 시스템 담당자의 숙명이자 고통이고 보람입니다. 시간이 지나고 프로그램이 고도화되면 마이그레이션이 필요한 시점이 분명히 오게 되어있습니다. 기존 서비스에 영향을 미치지 않으면서 저비용으로 마이그레이션을 하는 것이 최선이지만 말처럼 쉽지 않습니다. 다음 두 가지의 옵션을 살펴봅시다.

IaaS : Infrastructure as a Service

PaaS : Platform as a Service

위 두 가지 모두 장단점이 뚜렷합니다.

IaaS를 활용할 경우, 여러 개의 가상 머신을 두고 각 가상 머신에 필요한 컴포넌트를 올립니다. 컴포넌트별 환경을 커스터마이징 할 수 있기 때문에 이식성이 좋지만 불필요한 자원 사용에 대한 비용이 증가하는 단점이 있습니다.

그럼 PaaS는 어떨까요? 클라우드 서비스에서 제공하는 플랫폼을 사용하는 것은 비용도 저렴하고 관리도 쉽습니다. 하지만 특정 클라우드 서비스에 종속된 애플리케이션을 타 서비스로 마이그레이션 하는 일은 만만한 작업이 아닙니다.

위 두 가지의 단점을 보완하기 위한 구원타자가 바로 도커입니다. 이식성이 좋으면서도 리소스 사용에 대한 비용을 줄일 수 있기 때문이죠. 가령 AWS에서 운영하던 프로그램을 MS Azure로 마이그레이션 하는 경우, 소스코드를 고치지 않고도 안정적으로 시스템 운영이 가능합니다.

0.3 "Hello, Docker!"

프로그래밍 언어를 처음 배울 때 콘솔 창에 처음 `Hello, World!` 를 띄웠던 기억이 생생합니다.

그 감격을 도커를 통해 느껴보려고 합니다.

일단 도커를 활용하기 위해서는 당연히 환경을 세팅해야 하는데, 필자는 Mac OS/Window에서 VMware를 실행하여 가상머신을 생성 후, Ubuntu Server를 설치하여 진행하였습니다.

0.3.1 Mac + VMware Fusion Player + Ubuntu Server

VMware 의 공식 사이트에 접속하면 비상업적 목적의 개인 사용자를 위한 VMware Fusion Player 을 무료로 제공하고 있습니다. Ubuntu 리눅스의 경우, 카카오 미러 페이지에서 다운받을 수 있으며 Server 배포판을 다운로드 받으면 됩니다.

▼ **VMware Fusion Player 다운로드 링크**

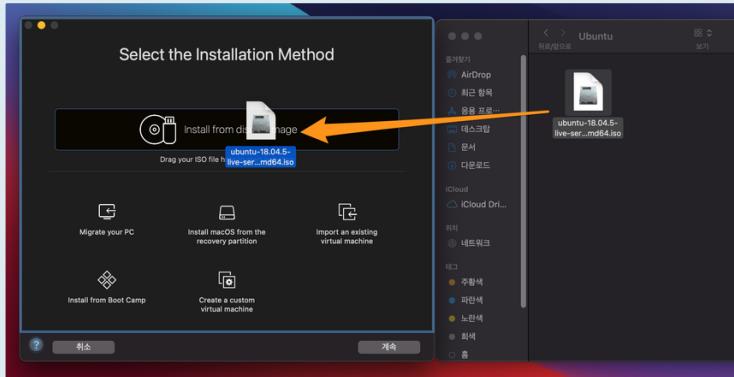
<https://my.vmware.com/web/vmware/evalcenter?p=fusion-player-personal>

▼ **Ubuntu 18.04.5 LTS (Bionic Beaver) 설치 이미지 다운로드 링크**

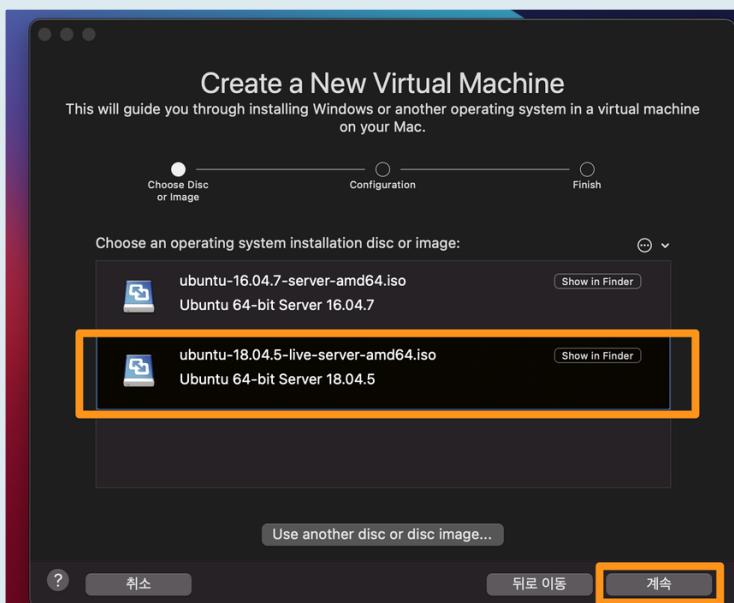
<https://mirror.kakao.com/ubuntu-releases/bionic/>

▼ Ubuntu 18.04.5 LTS (Bionic Beaver) 설치 가이드

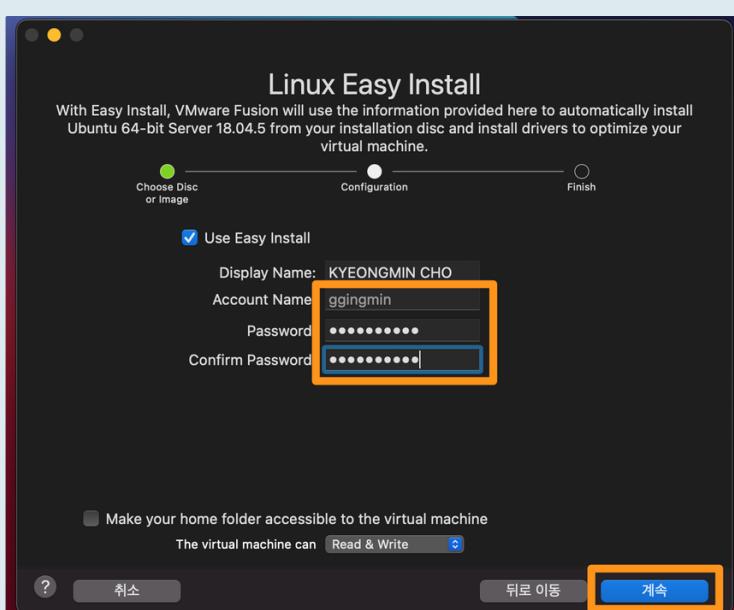
1. VMware 를 구동한 후, 다운받은 Ubuntu 설치 이미지(*.iso)를 드래그하여 추가한다.



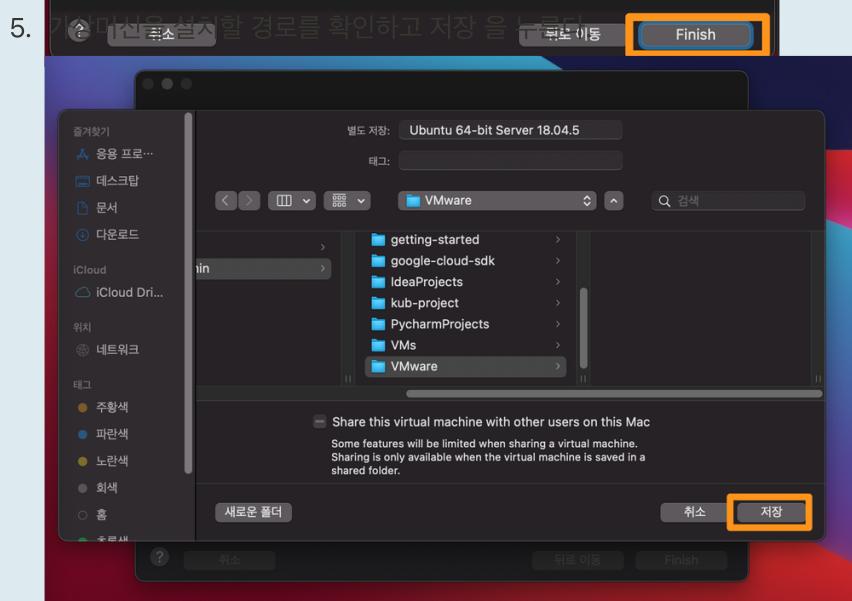
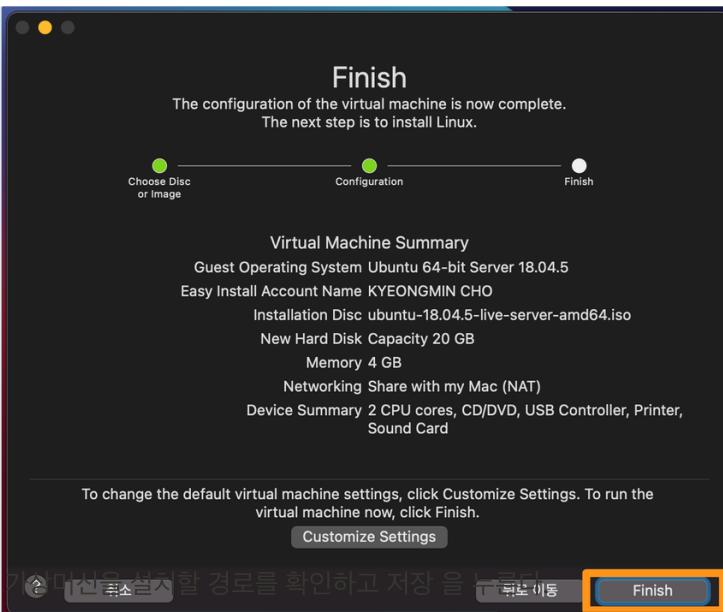
2. 추가한 이미지를 선택하고 다음 을 누른다.



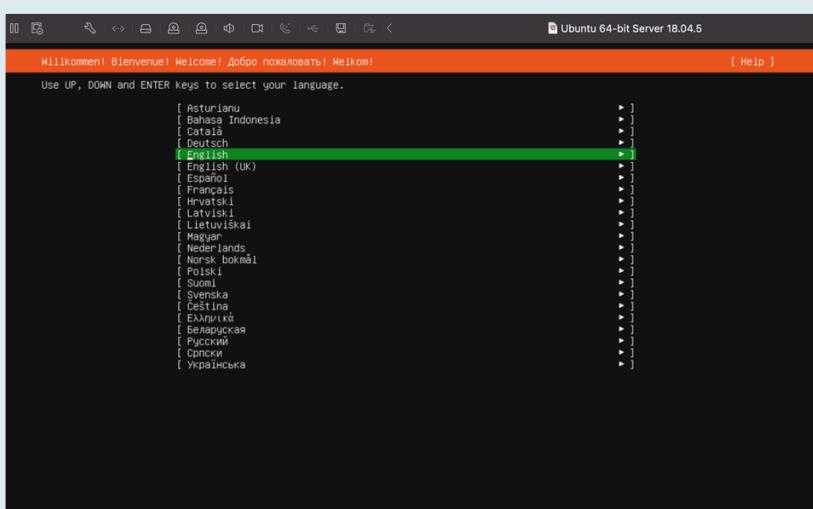
3. 계정명과 암호를 입력한 후 계속 을 누른다.



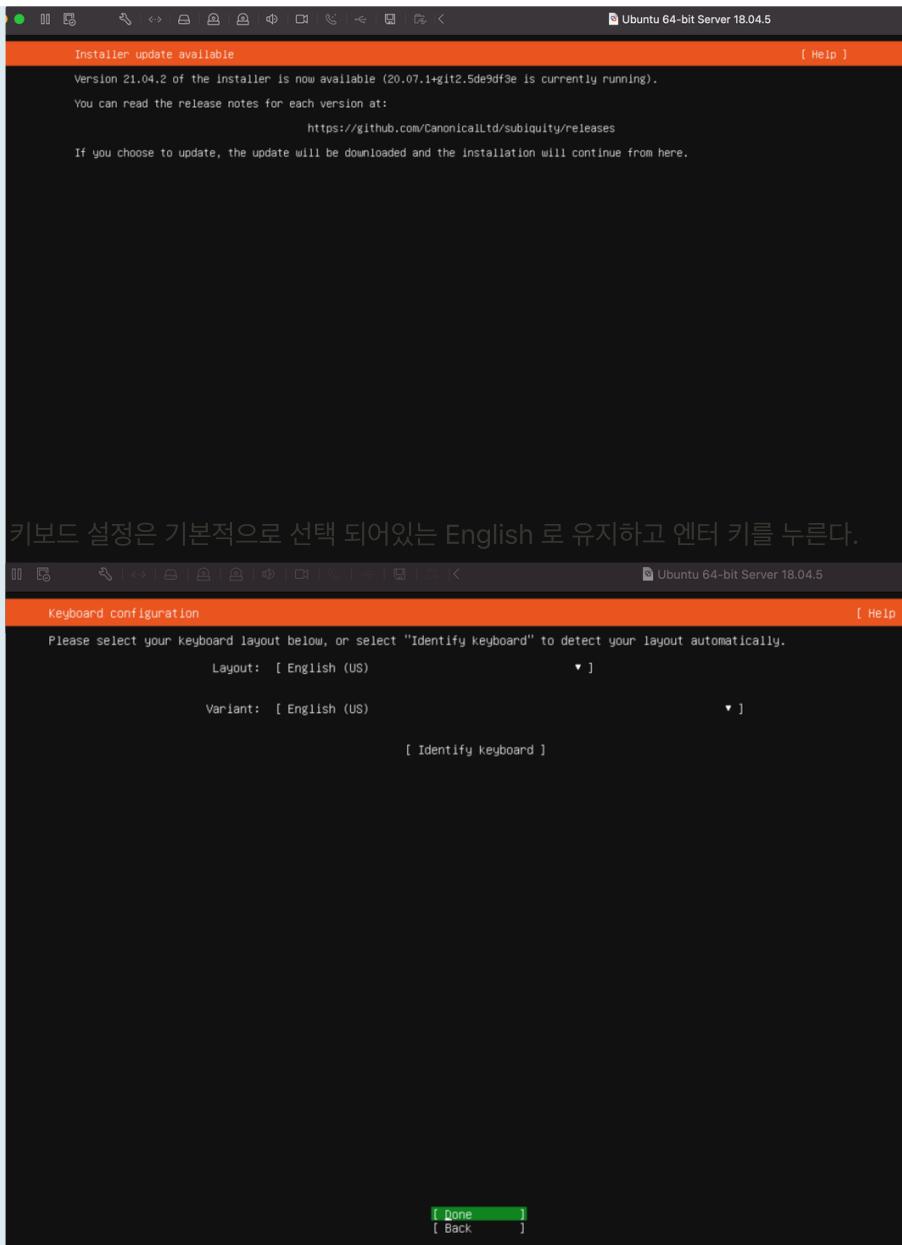
4. 설치가 진행될 가상머신의 사양을 확인한 후 Finish 를 누른다.



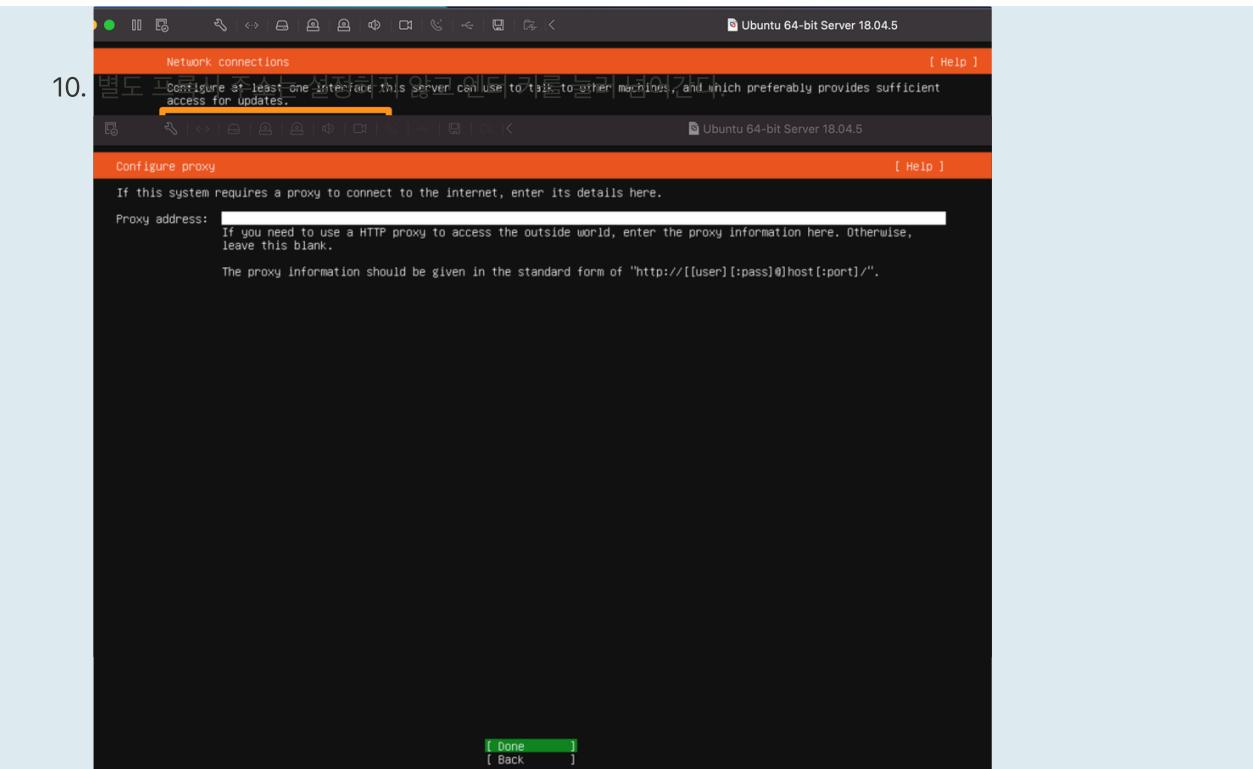
6. 저장을 누르면 가상머신이 구동되며 자동으로 설치가 진행된다. 언어를 영어로 선택하고 엔터 키를 누른다.



7. 별도의 Ubuntu 버전 업데이트는 불필요하기 때문에 Continue without updating 을 선택하고 엔터 키를 누른다.

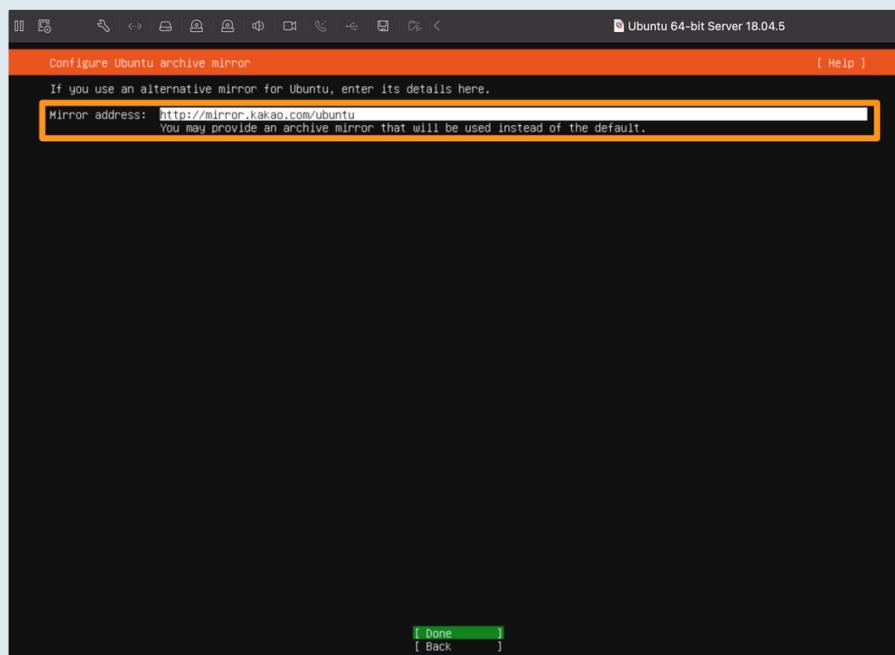


8. 키보드 설정은 기본적으로 선택 되어있는 English 로 유지하고 엔터 키를 누른다.
9. 네트워크 연결 설정에서는 별도 커스텀 없이 기본으로 할당되는 IP를 사용할 예정이므로 수정없이 엔터 키를 누른다. 표시된 IP는 Host PC에서 가상머신으로 접속하는 인터페이스 역할을 한다.

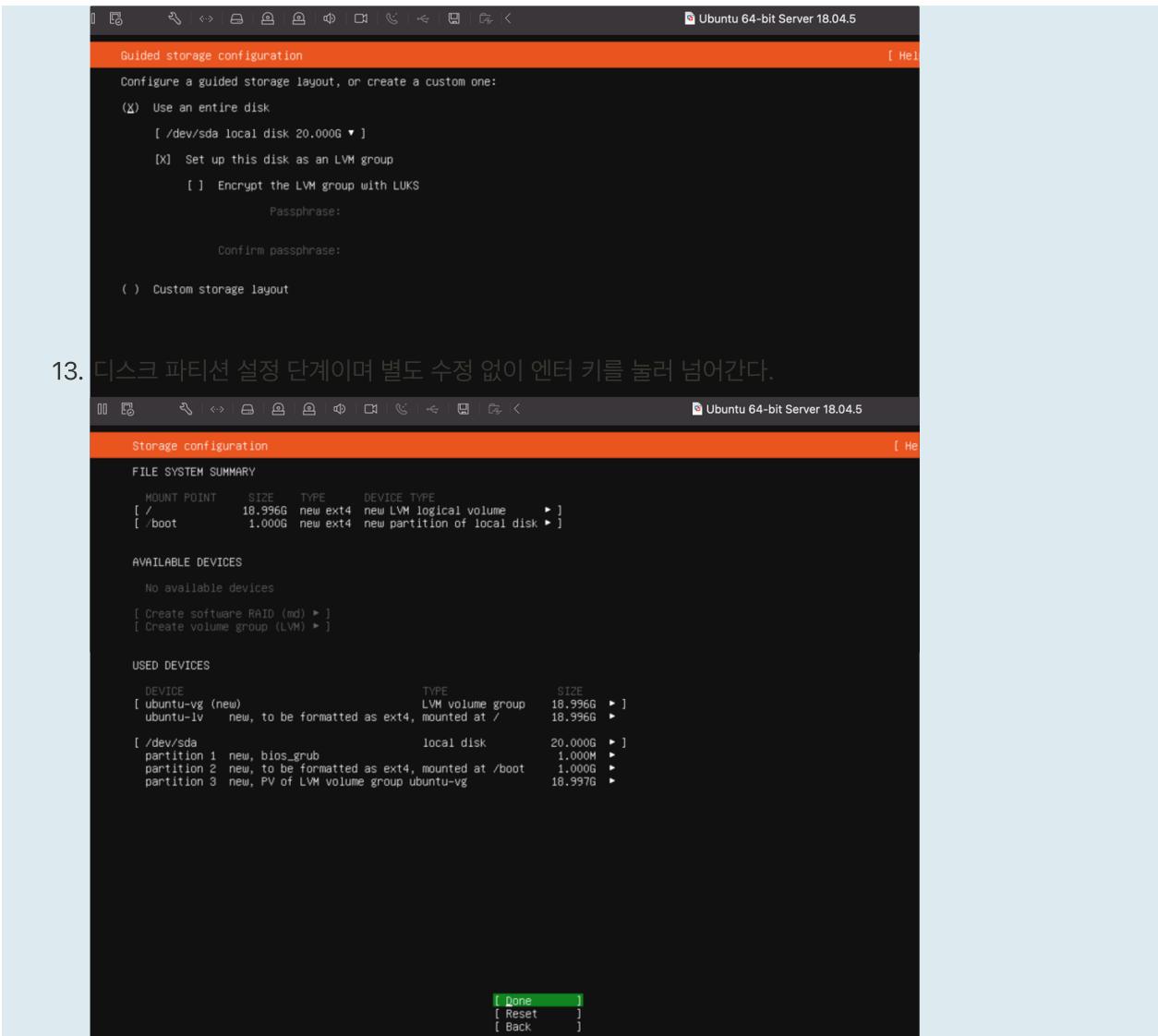


10. 별도 설정하지 않고 인터넷에 접근할 수 있는 환경을 갖다.

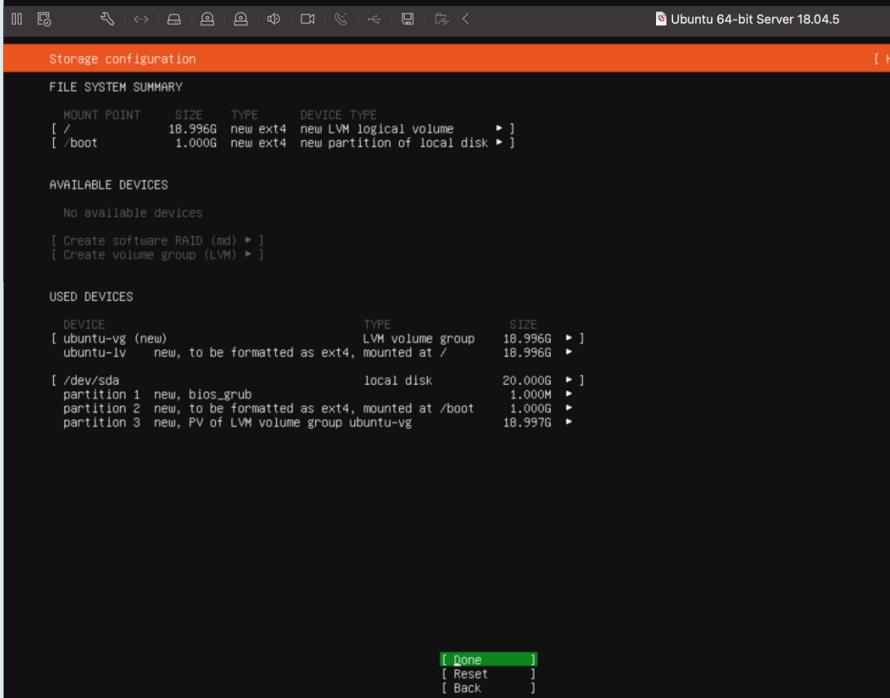
<http://mirror.kakao.com/ubuntu>



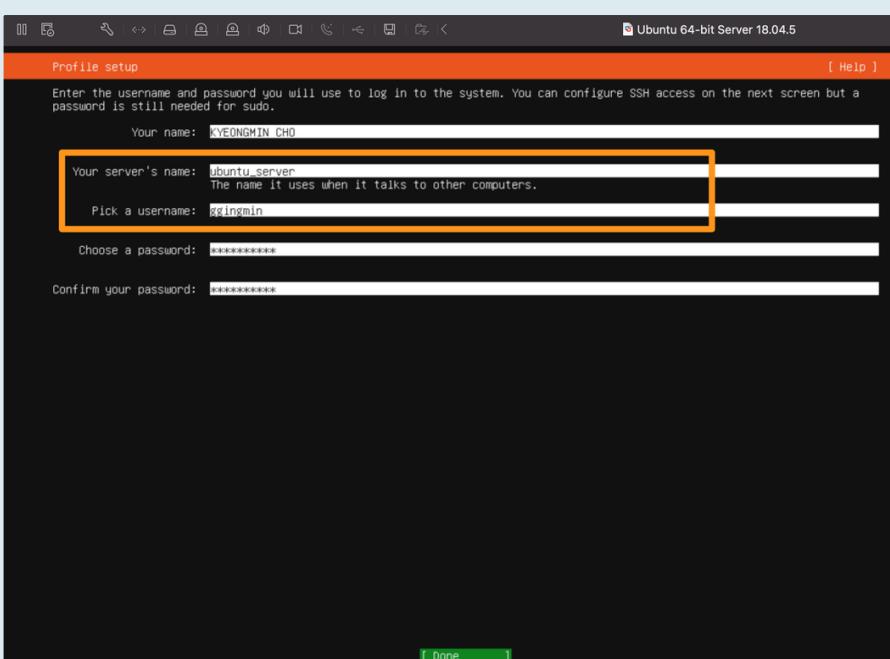
12. 사용할 디스크 용량을 설정하는 단계이며 별도 수정 없이 엔터 키를 눌러 넘어간다.



13. 디스크 파티션 설정 단계이며 별도 수정 없이 엔터 키를 눌러 넘어간다.

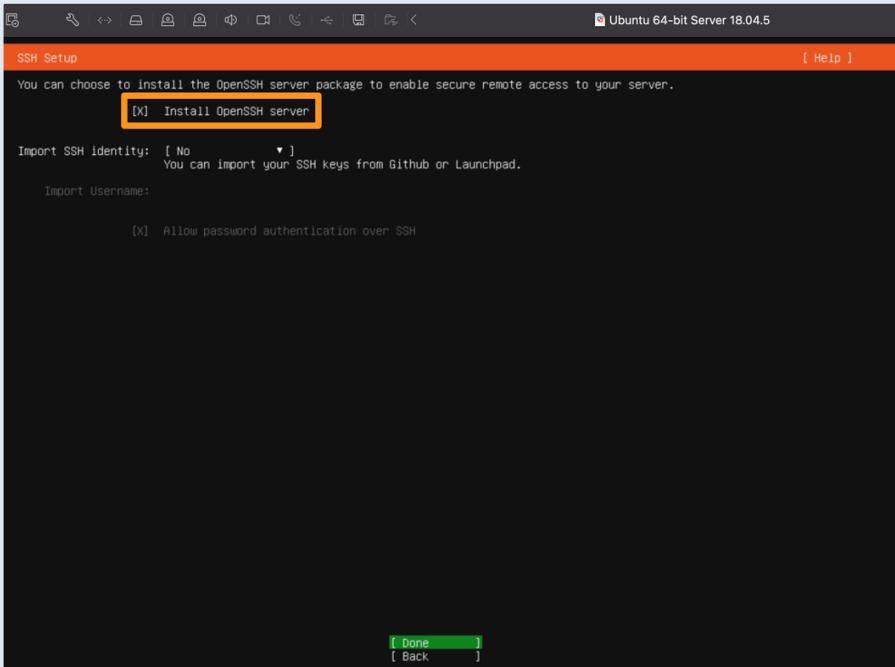


14. 서버의 이름과 계정명을 알맞게 입력하고 엔터 키를 누른다. VMware에서 지원하는 Easy Install에서 미리 정 보를 입력했지만 다시 입력을 하게끔 창이 뜨는데 좀 의아한 부분이다.

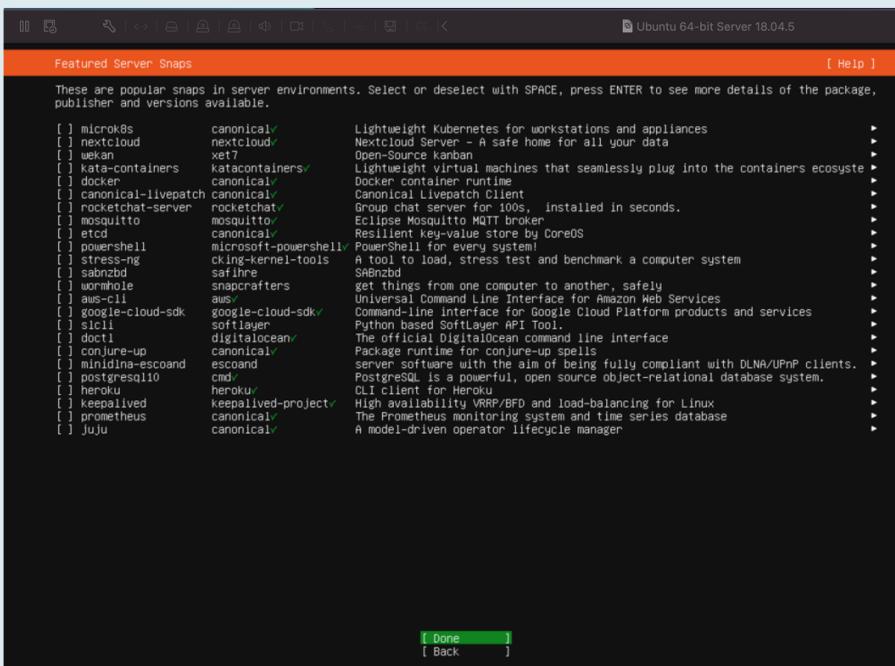


15. Host PC에서 ssh 방식으로 접속할 수 있도록 OpenSSH를 설치하는 체크박스에 체크를 한 후 엔터 키를 누

른다.



16. 서버 세팅을 위한 다양한 패키지가 표시된다. 추후 직접 터미널 명령어로 설치를 진행 예정이므로 지금은 엔터 키를 눌러 넘어간다.



17. 드디어 설치의 진행을 알리는 화면이 뜬다. 긴 여정이었다. 모든 설치가 완료되면 Reboot 버튼이 생성되며 엔터키를 눌러 재부팅을 한다.

18. 계정명과 암호를 입력한 후 시스템에 접속한다.

```
Installation complete! [ Help ]
Finished Install!
configuring lvm-volgroup: lvm_volvgroup-0
configuring lvm-partition: lvm_partition-0
configuring format: format-1
configuring mount: mount-1
configuring mount: mount-0
writing install sources to disk
running 'curtin extract'
curtin command extract
curtin command extracting image from cp:///media/filesystem
configuring installed system
running '/snap/bin/subiquity.subiquity-configure-run'
running '/snap/bin/subiquity.subiquity-configure-apt /snap/subiquity/1966/usr/bin/python3 true'
curtin command apt-config
curtin command in-target
running 'curtin-cuthooks'
curtin command cuthooks
curtin command apt-get update
installing microd.nautilus
ubuntu_server login: ggingmin
Password:
Last login: Tue Jun 1 01:40:32 UTC 2021 from 172.16.173.1 on pts/0
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-143-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

System information as of Tue Jun 1 06:31:58 UTC 2021

System load: 0.09 Processes: 195
Usage of /: 35.5% of 18.57GB Users logged in: 0
Memory usage: 7% IP address for ens33: 172.16.173.4
Swap usage: 0% IP address for docker0: 172.17.0.1

* Super-optimized for small spaces - read how we shrank the memory
  footprint of MicroK8s to make it the smallest full K8s around.

https://ubuntu.com/blog/microk8s-memory-optimisation

53 packages can be updated,
1 update is a security update.

New release '20.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
```

19. `ifconfig` 명령어를 입력한 후 다음의 IP를 확인한다.

```
ggingmin@ubuntu_server:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
        ether 02:42:6d:59:ff:94 txqueuelen 0 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.173.4 netmask 255.255.255.0 broadcast 172.16.173.255
        inet6 fe80::20c:29ff:fed5:e18 prefixlen 64 scopeid 0x20<link>
        ether 00:0c:29:d5:0e:18 txqueuelen 1000 (Ethernet)
        RX packets 12 bytes 1574 (1.5 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 24 bytes 2392 (2.3 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 84 bytes 6324 (6.3 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 84 bytes 6324 (6.3 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

20. Host OS(Mac)에서 터미널을 구동한 후 다음의 명령어를 실행한다.

```
$ sudo ssh (계정명)@(Guest PC의 이더넷 IP)
```

21. Guest PC의 비밀번호를 입력하고 엔터키를 누른 후, 인증키 생성 관련 안내가 나오면 yes 를 입력한다.

The screenshot shows a terminal window with the following text:

```
sudo ssh ggingmin@172.16.173.4
The authenticity of host '172.16.173.4 (172.16.173.4)' can't be established.
ECDSA key fingerprint is SHA256:zejwGX4WPndhFzjYiMSzjjfyG8ggg1N9KeDckq3QpY.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

Below this, the terminal displays the standard Ubuntu 18.04 LTS boot message and system information. It includes details about documentation, management, support, and system load. It also mentions a memory optimization note and package updates.

```
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-143-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

System information as of Tue Jun 1 06:44:21 UTC 2021

System load: 0.0          Processes:      167
Usage of /: 35.5% of 18.57GB  Users logged in:   1
Memory usage: 7%
Swap usage: 0%
IP address for ens3: 172.16.173.4
IP address for docker0: 172.17.0.1

* Super-optimized for small spaces - read how we shrank the memory
  footprint of MicroK8s to make it the smallest full K8s around.

https://ubuntu.com/blog/microk8s-memory-optimisation

53 packages can be updated.
1 update is a security update.

New release '20.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Tue Jun 1 06:31:57 2021
ggingmin@ubuntu_server:~$
```

설치가 완료되면 다음의 가이드에 따라 SSH 방식으로 Host OS에서 Guest OS(Ubuntu Server)에 접속합니다.

▼ Mac에서 SSH로 원격 접속하기

- VM에서 구동된 Ubuntu에서 `ifconfig` 명령어를 입력한 후 IP 주소를 확인합니다.

```
ggingmin@ubuntu_server:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
      ether 02:42:54:f3:10:7f txqueuelen 0  (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 172.16.173.6 netmask 255.255.255.0 broadcast 172.16.173.255
        inet6 fe80::20c:29ff:fed0:2310 prefixlen 64 scopeid 0x20<link>
          ether 00:0c:29:d0:23:10 txqueuelen 1000  (Ethernet)
            RX packets 8692 bytes 12804006 (12.8 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 889 bytes 76026 (76.0 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
          loop txqueuelen 1000 (Local Loopback)
            RX packets 132 bytes 11348 (11.3 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 132 bytes 11348 (11.3 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

- Mac Terminal을 키고 다음의 명령어를 입력합니다.

아래와 같이 나오면 정상적으로 접속이 완료된 것입니다.

```
$ ssh (Ubuntu username)@(Ubuntu VM IP)
```

```
→ ~ ssh ggingmin@172.16.173.6
ggingmin@172.16.173.6's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-144-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information as of Wed Jun 16 06:25:41 UTC 2021

 System load:  0.04           Processes:           168
 Usage of /:   33.4% of 18.57GB  Users logged in:    1
 Memory usage: 8%             IP address for ens33: 172.16.173.6
 Swap usage:   0%
```

0.3.2 Window + VMware Workstation Player + Ubuntu Server

윈도우의 경우에도 큰 맥락에서 설치하는 방법은 비슷하며 VMware의 사용법 역시 크게 다르지 않습니다. 다음의 다운로드 링크를 참고하여 설치를 진행합니다.

추가로, 윈도우에서 Ubuntu Server에 SSH 접속을 하기 위한 XShell도 설치해주세요.

▼ VMware Workstation Player 다운로드 링크

<https://my.vmware.com/en/web/vmware/downloads/details?downloadGroup=WKST-PLAYER-1612&productId=1039&rPId=66621>

▼ Ubuntu 18.04.5 LTS (Bionic Beaver) 설치 이미지 다운로드 링크

<https://mirror.kakao.com/ubuntu-releases/bionic/>

▼ XShell 다운로드 링크

<https://www.netsarang.com/ko/free-for-home-school/>

0.3.3 Ubuntu에 Docker 설치하기

Ubuntu에 Docker를 설치하는 방법은 다음의 공식 문서를 참고하여 진행하면 됩니다.

▼ Docker Docs - Install Docker Engine

<https://docs.docker.com/engine/install/ubuntu/>

▼ Docker Docs - Install Docker Compose

<https://docs.docker.com/compose/install/>

0.3.4 Docker에서 hello-world 컨테이너 실행하기

```
~$ sudo docker container run hello-world
```

위 명령어를 실행하면 반갑게도 **Hello from Docker!**라는 메세지가 터미널에 뜹니다.

설레는 순간이네요. 프로그래밍 언어를 처음 배우던 순간의 **Hello World!**처럼.

이제 본격적으로 도커에 대해 알아봅시다.

1. 컨테이너와 도커

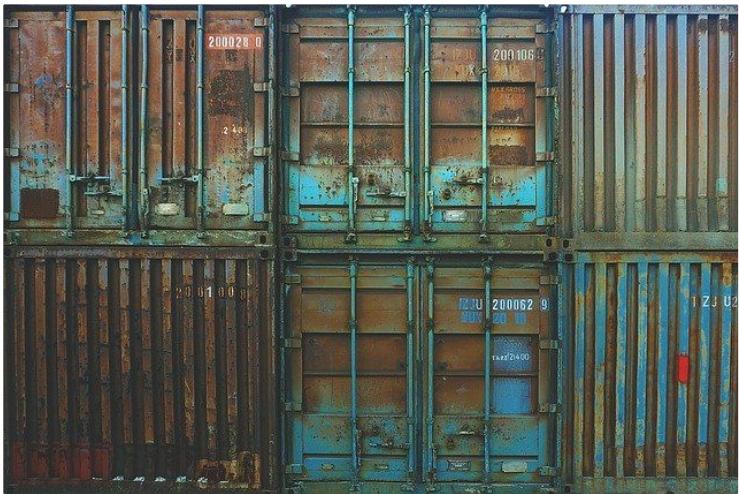


수학이든 과학이든 사회든, 결국 개념을 이해하는게 중요합니다.

어렴풋이 머리를 맴돌던 컨테이너의 개념을 확실히 잡고 넘어보도록 하겠습니다.

1.1 컨테이너와 MSA

지금 당장 컨테이너라는 단어를 들으면 다음의 이미지가 떠오를 것입니다.



항만에 가득 쌓인 컨테이너와 우리가 운용할 컨테이너는 어떤 관계가 있을까요? 기본적으로 이 둘의 역할은 같습니다. 무언가를 분리해서 담아놓고 그것들을 쭉 쌓아올리는 것이죠.

컴퓨터는 CPU, SSD, RAM, 그래픽 카드 등 각종 하드웨어로 구성되어 있습니다. 그리고 윈도우, macOS, 리눅스와 같은 운영체제 위에서 저희가 사용하는 어플리케이션이 실행되구요. 만약에 도입하려는 어플리케이션이나 플랫폼이 서로 다른 요구사항을 가진다면 이 문제를 어떻게 해결할 수 있을까요? 더불어, 성능에도 부정적인 영향을 끼치지 않아야 한다면 어떻게 해야 할까요?

컨테이너가 바로 그 해결책을 제시하고 있습니다. Host OS에 어플리케이션을 실행할 공간을 각각 구성하고 IP와 디렉토리는 분리시켜 놓은 것을 컨테이너라고 합니다. VMWare나 VirtualBox에서 VM을 운용해 보신 분들은 유사한 개념이라고 볼 수도 있지만 컨테이너는 Guest OS를 두지 않고 어플리케이션과 미들웨어를 실행합니다.

각 컨테이너에는 어플리케이션 별로 요구하는 환경을 배타적으로 세팅할 수 있으며, 구성된 컨테이너를 통합해 하나의 큰 시스템을 구축할 수 있습니다. 이를 **MSA(Microservice Architecture)**라고 합니다. MSA는 어플리케이션을 기능과 용도에 따라 모듈화하여 개발하고 이를 독립적으로 배포할 수 있는 환경을 지향합니다.

1.2 도커의 역할

컨테이너의 정의와 사용목적에 대해서 이해가 될 무렵 도커는 도대체 무슨 일을 하는지 궁금해집니다. 자세한 내용을 이야기하기 전에 도커의 심볼을 한 번 들여다 보겠습니다. 큼지막한 고래의 등에 네모난 상자들이 가득합니다. 상자들이 가리키는 것이 바로 이전에 살펴보았던 컨테이너입니다.

도커는 어플리케이션을 실행하는 데에 필요한 각종 요소(OS, 미들웨어, 네트워크 설정 등)들을 한데 묶어 하나의 이미지로 생성합니다. 이 과정은 마치 한창 CD를 쓰던 시절 "CD를 굽는" 작업에 비유할 수 있습니다. 이렇게 "구워진" 이미지는 컨테이너를 실행하는 데에 이용됩니다.

생성된 이미지는 Host OS 상에서만 이용되는 것이 아니라 [Docker Hub](#)이라는 레지스트리에 공유가 가능합니다. 이 레지스트리에는 제조사에서 제공하는 리눅스 배포판이나 웹서버, DBMS 등의 공식 이미지가 공개되어 있습니다. 개인이 본인 소유의 저장소에 이미지를 저장하는것 역시 가능합니다. 만약 사내 시스템과 관련한 민감 정보가 이미지에 포함된 경우라면 AWS, GCP 등에서 제공하는 사설 레지스트리를 활용할 수도 있으며, 개인 서버에 레지스트리를 구축하는 것도 가능합니다.

내용을 종합해보면 도커는 아래의 정의와 같습니다.



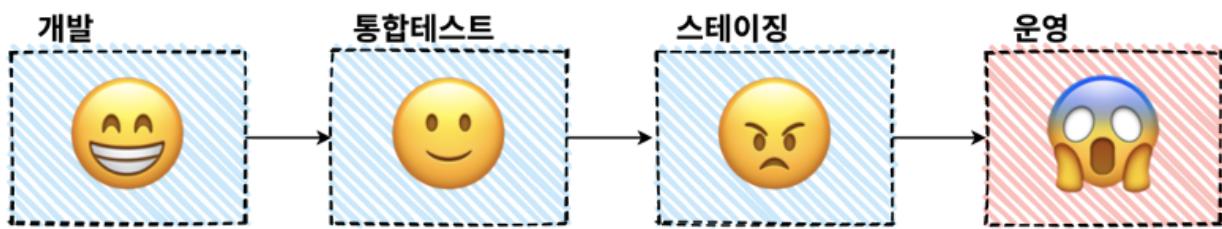
도커는 어플리케이션 및 실행 환경을 정의한 이미지를 생성/공유함과 동시에, 이를 이용하여 컨테이너를 작동 할 수 있도록 하는 플랫폼이다.

1.3 개발자 + 도커 = 😊

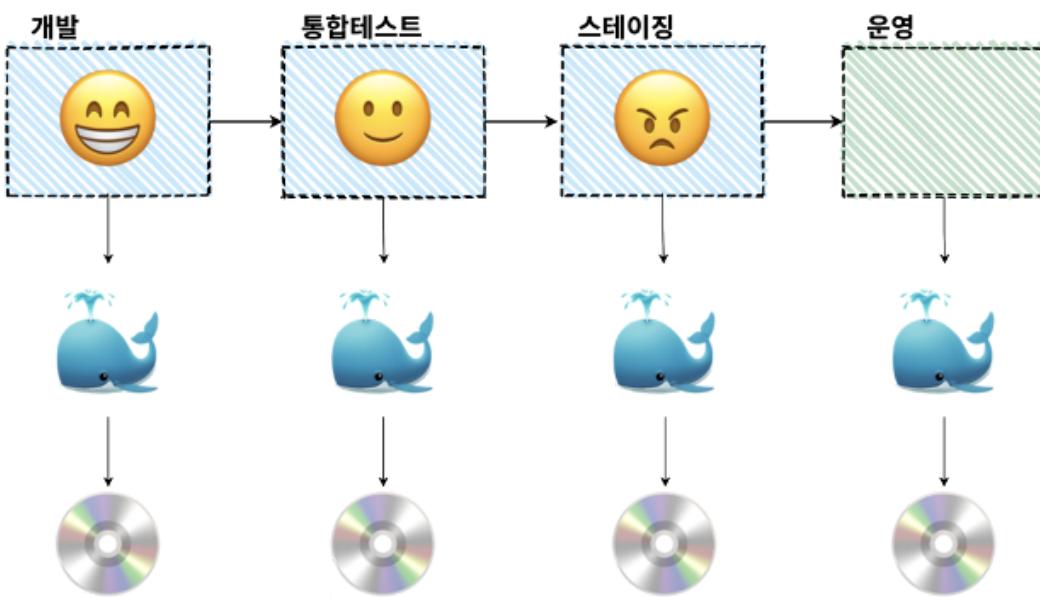
저는 유통 분야 기업에서 시스템 운영업무를 했었습니다. 현업의 요구사항에 따라 시스템을 개발/수정하고 주 단위로 반영을 했었죠. "개발 - 통합테스트 - 스테이징(가상운영) - 운영", 총 4단계 인프라 환경을 거쳐 프로덕션으로 배포가 되었습니다.

여느 때처럼 수정한 소스 코드가 개발과 통합테스트, 스테이징 환경에서 이상이 없는 것을 확인하고 운영 환경에 배포를 했는데... 배포한 프로그램에서 에러 로그가 폭주하기 시작했습니다. 눈앞이 깜깜해지고 머리가 하얘지는 와중에 룰백을 하고 원인을 찾기 시작했습니다. 소스는 아무리봐도 이상이 없는데... 발을 구르다가 원인이 환경변수 등을 다루는 설정 파일에 있음을 알게 됐습니다.

가령 스프링 프로젝트가 정상적으로 빌드되었다 하더라도 애플리케이션 런타임이나 기타 설정이 맞지 않으면 에러가 발생하게 되고 심한 경우 중대한 장애로 이어집니다. 도커는 이런 상황을 어떻게 해결할까요? 아래 그림을 한 번 보겠습니다.



각 환경이 동일하다는 보장이 없기 때문에 최종 배포까지 했던 모든 노력이 무색하고 중대 장애를 야기할 수 있습니다.



도커 기반의 개발환경에서는, 빌드 정보를 담고 있는 *Dockerfile*과 소스 코드를 함께 Git 저장소에 push 합니다. 이를 기반으로 빌드된 이미지는 Docker Registry에 저장되고 개발자는 이미지를 활용하여 각 환경에 알맞게 컨테이너를 생성하면 됩니다. 이미지에는 애플리케이션 런타임에 관련된 모든 미들웨어, 라이브러리 등 의존성이 전부 포함되어 있으므로 "개발 환경에서는 잘 됐는데 운영 환경에서 갑자기 안돼요"라는 억울한 상황을 미연에 방지할 수 있습니다.

도커의 등장은 '개발'과 '관리'로 이원화 된 역할을 하나로 묶어주는 중요한 역할을 합니다. 기존에는 두 영역의 업무를 수행에 필요한 지식이 상이했다면, 도커 시스템 상에서는 어떻게 *Dockerfile*을 작성해서 컨테이너를 구성할 것인가를 고민하면 됩니다. 애플리케이션 개발자가 실제 배포까지 수행 할 수 있는 환경을 구축했기 때문에 배포는 '관리'에서 할 때까지 마냥 기다리며 의존할 필요가 없어집니다.