

Network and Kubernetes

웹 어플리케이션 컨테이너 이미지 만들고 클라우드에 배포까지

Contents

- #1. Linux & Network basics
- #2. Python API Development
 - 개발 환경 조성 및 Fast API
- #3. Docker image to web server
 - Docker container image
 - 웹 서버, WSGI Interface
- #4. Compose, Orchestration, PaaS
 - Kubernetes 소개 및 구조 설명
 - Orchestration?
 - 내가 만든 웹 어플리케이션 구동하기

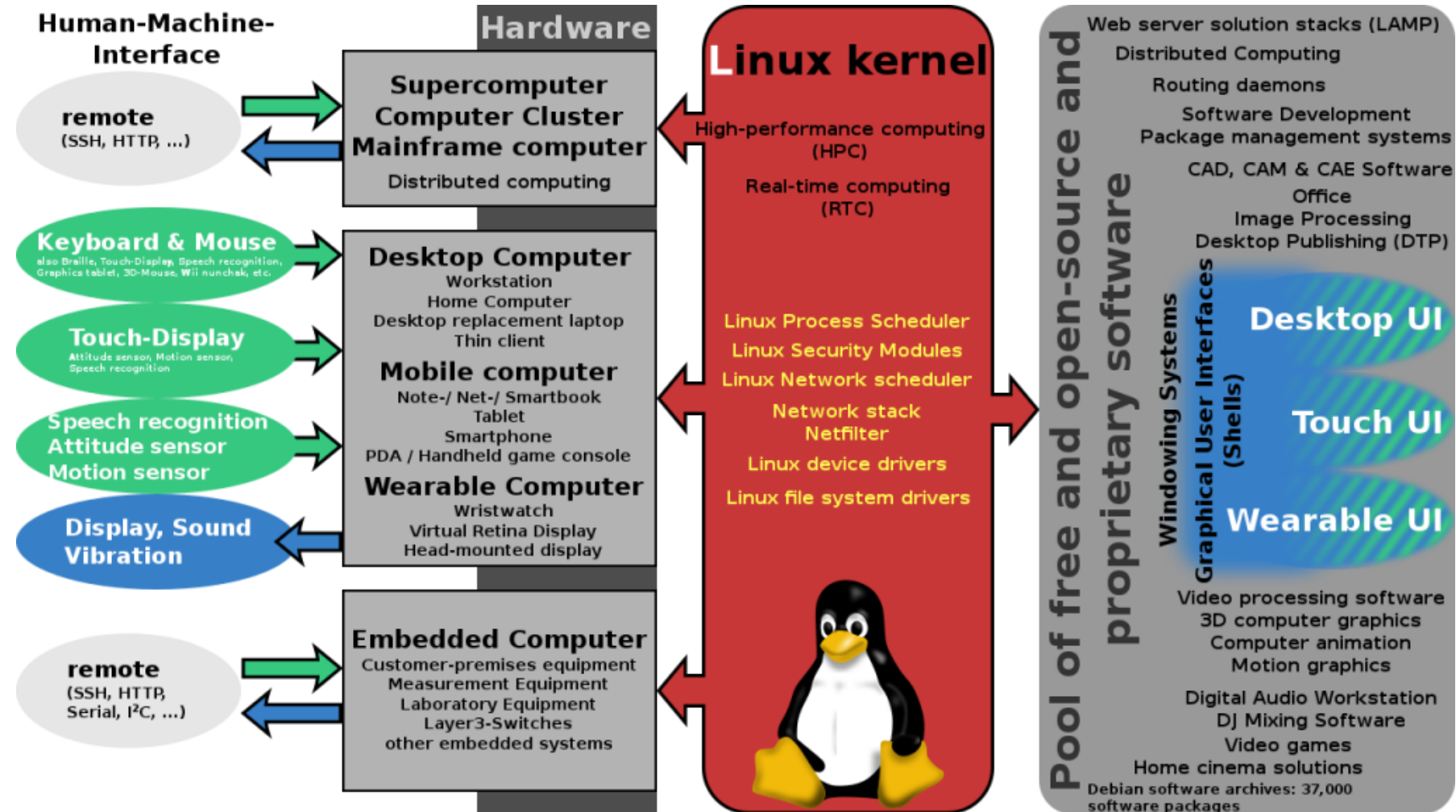
#1. 리눅스 기초

Linux / SSH Protocol / Firewall / Settings

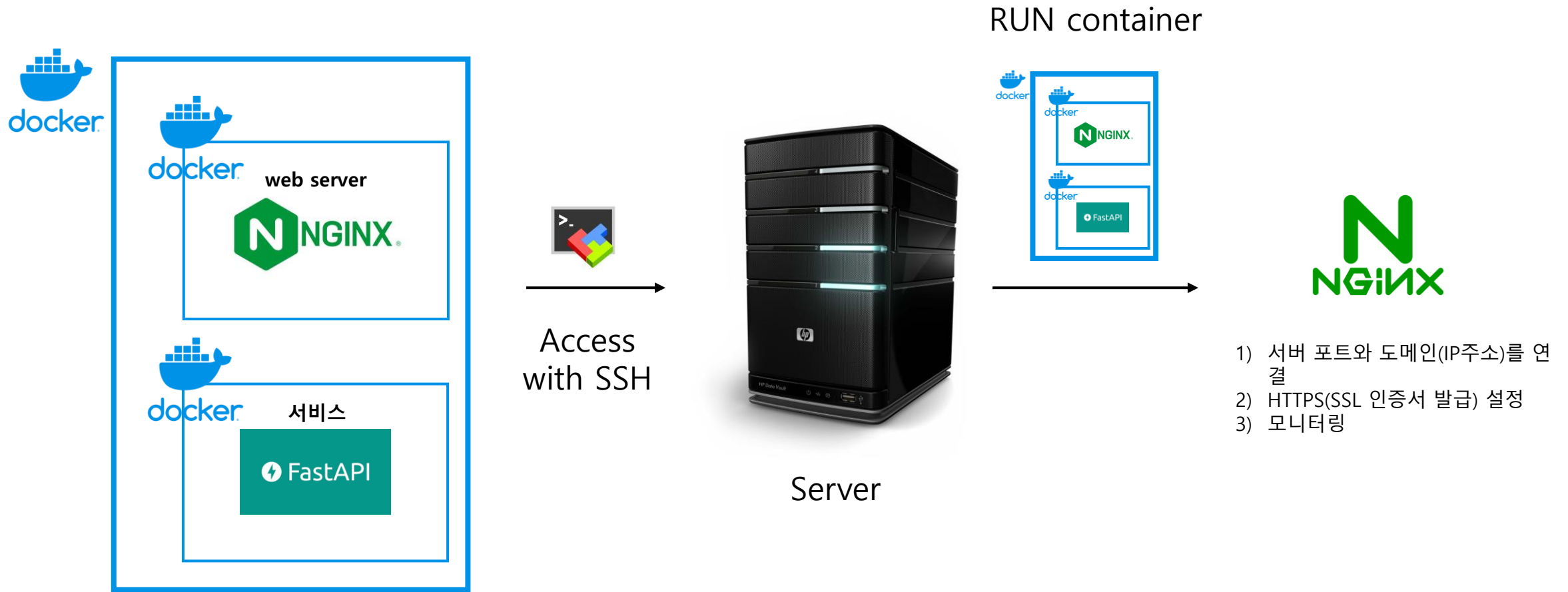
Topics

- 1) LINUX terminal 접속 및 간단한 명령어 실습
 - Linux
 - 실습 overview
 - Commands
- 2) Network Port, 방화벽, 클라우드 설정
 - SSH
 - 방화벽 개념

Linux?



Practice with a glance



- 로컬 환경에서 내가 만든 FastAPI 기반 코드를 docker container로 만들고, 이를 교내 Server PC에서 작동해본다.
- 배포 및 모니터링

Linux 명령어 실습

← ↻ 🔒 https://bellard.org/jslinux/ aぁ ☆ 📄 📁 ↺ ⬇

JSLinux

Run Linux or other Operating Systems in your browser!

The following emulated systems are available:

CPU	OS	User Interface	VFsync access	Startup Link	TEMU Config	Comment
x86	Alpine Linux 3.12.0	Console	Yes	click here	url	
x86	Alpine Linux 3.12.0	X Window	Yes	click here	url	Right mouse button for the menu.
x86	Windows 2000	Graphical	No	click here	url	Disclaimer.
x86	FreeDOS	VGA Text	No	click here	url	
riscv64	Buildroot (Linux)	Console	Yes	click here	url	
riscv64	Buildroot (Linux)	X Window	Yes	click here	url	Right mouse button for the menu.
riscv64	Fedora 33 (Linux)	Console	Yes	click here	url	Warning: longer boot time.
riscv64	Fedora 33 (Linux)	X Window	Yes	click here	url	Warning: longer boot time. Right mouse button for the menu.

© 2011-2021 Fabrice Bellard - [News](#) - [VM list](#) - [FAQ](#) - [Technical notes](#)

참고자료: [리눅스/Linux] 간단하게 리눅스 명령어 연습해 볼 수 있는 사이트 JS Linux — 🌟

리눅스 주요 명령어 소개

- pwd: Print Work Directory의 약자, 현재 작업 중인 디렉토리(내 위치)를 보여줌
- ls: List Segments의 약자로, pwd 기준 하위 디렉토리 및 파일 정보를 출력
- cd: Change Directory의 약자로, 현재 작업하고 있는 디렉토리의 위치를 이동하는 명령어
- mkdir: Make Directory, 새 폴더 만들기
- rmdir: Remove Directory, 폴더 지우기
- cp: copy의 약자로, 파일을 복사할 때 사용
- touch: 빈 파일 생성
- mv: Move의 약자로, 파일 위치 이동시 사용
- cat: Concatenate에서 유래, 파일 내용을 확인할 때 사용

리눅스 주요 명령어 소개

- ps: 현재 실행중인 프로세스를 확인
- chmod: Change Mode, 파일 또는 객체의 액세스 권한을 변경 -> 유저별로 파일 혹은 폴더에 대한 접근 차별화
- chown: 파일에 대한 소유자 / 그룹을 변경
- grep: global / regular expression / print 의 약어 + 합성어로, 파일 내 특정 텍스트를 검색 및 출력할 때 쓴다.
- kill: 잉여 프로세스 중지
- shutdown: 시스템 종료
- head: 파일 앞부분 미리보기
- tail: 파일 뒷부분 미리보기

리눅스 주요 명령어 소개

- zip, gzip, tar: 파일 및 디렉토리 압축
- unzip: 압축 해제
- ssh: SSH 프로토콜을 이용해 원격 호스트에 접속
- wget: 네트워크를 활용해 파일을 다운로드 할 때 사용
- curl: 네트워크를 활용해 파일을 내보낼 때 사용(업로드)
- apt-get: Debian 계열 패키지 및 소프트웨어 설치
- yum: RDHT(Red hat) 기반 시스템에서 소프트웨어 패키지를 관리
- vi/vim: 텍스트 에디터를 열어 파일을 직접 편집함
- clear: 인터페이스 청소기

리눅스 주요 명령어 소개

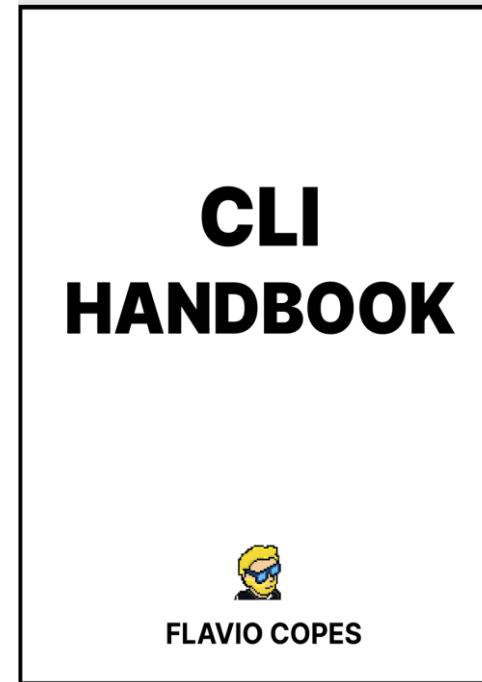
- sudo: Superuser do 라는 의미로, 관리자 권한으로 명령 실행 (온갖 위험한 거 할 때 사용)
- echo: 셸에서 문자열 출력할 때 사용 (ex. echo \$PATH)
- man: Manual, 명령어의 매뉴얼을 볼 때 사용 (ex. man ls == ls --help)
- history: 명령어 사용 기록 열람
- chroot: 루트 디렉토리(/) 를 변경
- whoami: 현재 사용자(나)의 username을 출력
- hostname: 호스트의 username을 출력
- ping: 타깃 호스트에 네트워크로 접근 가능한지, 가능하다면 얼마나 걸리는지 등을 테스트

Linux 명령어 parameter 관련 참고자료

리눅스 명령어 목록

1. `ls` - 디렉토리 내용을 나열합니다.
예시: `ls -l` (자세한 리스트 출력)
2. `cd` - 디렉토리를 변경합니다.
예시: `cd /home` (home 디렉토리로 이동)
3. `pwd` - 현재 작업 중인 디렉토리의 경로를 출력합니다.
예시: `pwd`
4. `touch` - 새 파일을 생성하거나 파일의 타임스탬프를 변경합니다.
예시: `touch newfile.txt`
5. `cp` - 파일이나 디렉토리를 복사합니다.
예시: `cp source.txt destination.txt`
6. `mv` - 파일이나 디렉토리를 이동하거나 이름을 변경합니다.
예시: `mv oldname.txt newname.txt`

⋮



The Linux Handbook

- 1. Introduction to Linux
- 2. man
- 3. ls
- 4. cd
- 5. pwd
- 6. mkdir
- 7. rmdir
- 8. mv
- 9. cp
- 10. open
- 11. touch
- 12. find
- 13. ln
- 14. gzip
- 15. gunzip
- 16. tar
- 17. alias
- 18. cat
- 19. less
- 20. tail
- 21. wc
- 22. grep
- 23. sort
- 24. uniq
- 25. diff
- 26. echo
- 27. chown
- 28. chmod
- 29. umask
- 30. du
- 31. df

3

참고자료: [리눅스에서 자주사용되거나 꼭 알아야 하는 명령어 모음 TOP 100 • 테크플레이](#)

참고자료: [The Linux Commands Handbook](#)

SSH 연결

SSH - 원격 접속 프로토콜

22 (표준 포트)
서버 포트번호 20010(비표준 포트)



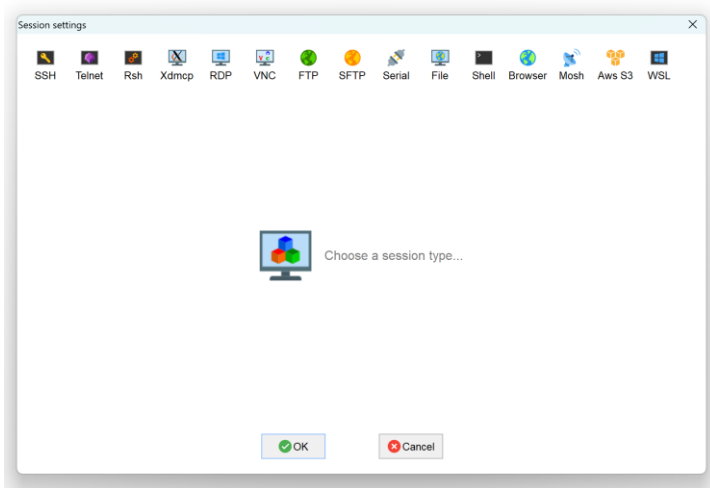
프로토콜? -> 통신 방식에 대한 규칙, 합의

HTTP request는 HTTP로 response,
HTTPS request는 HTTPS로 response

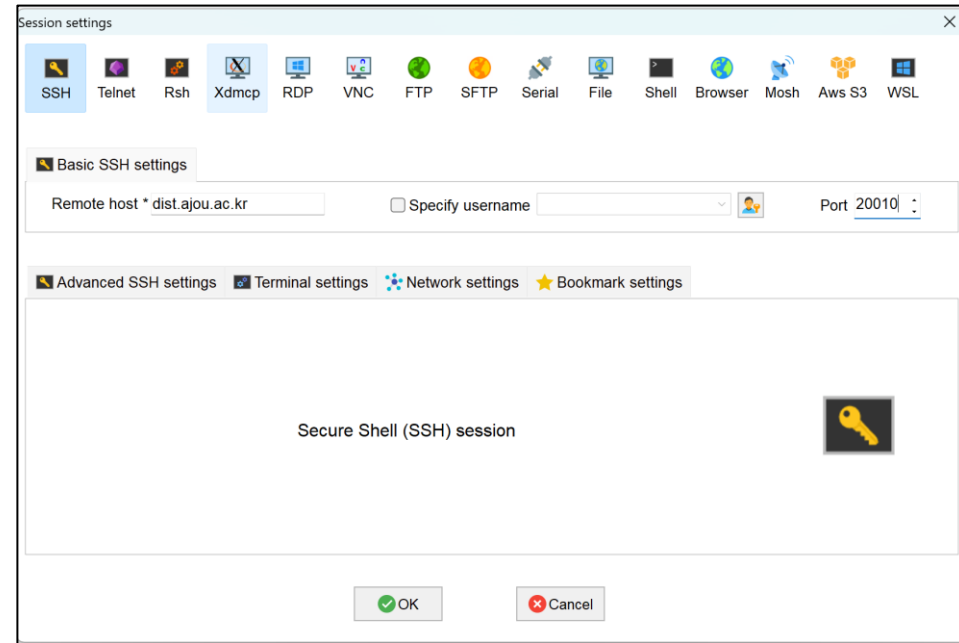


마찬가지로 SSH(Secure SHell) 또한 원격 접속을 위한 프로토콜이다.
이름에서도 알 수 있듯이, 높은 보안성을 가진 데이터 전송(암호화) 방식 때문에 원격 접속의 표준이 되는 프로토콜이다.

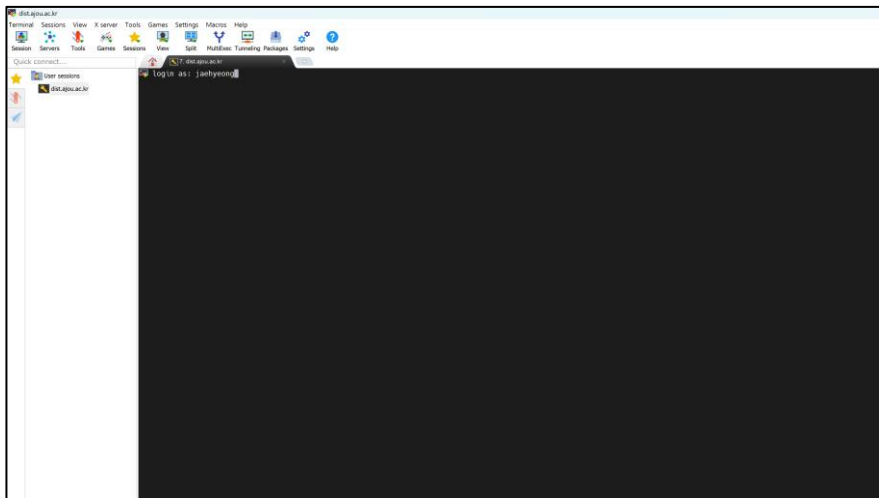
1



2

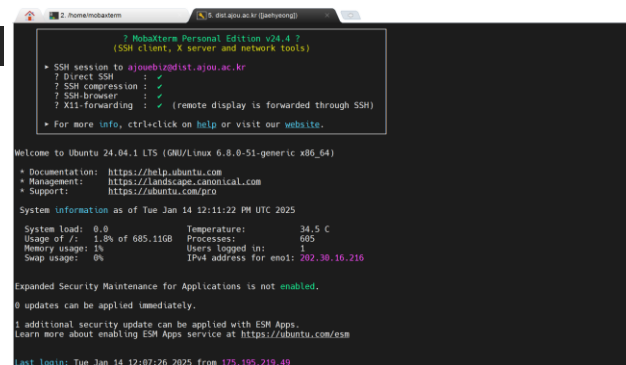


3



미리 등록된 username / pw 활용해서
접속

4



접속 성공

- 내가 만든 APP를 학교 서버에 업로드 및 구동하기 위해 MobaXterm으로 서버PC와 연결 (MobaXterm: 파일 관리 측면에서 Windows Powershell보다 유리한 Terminal)

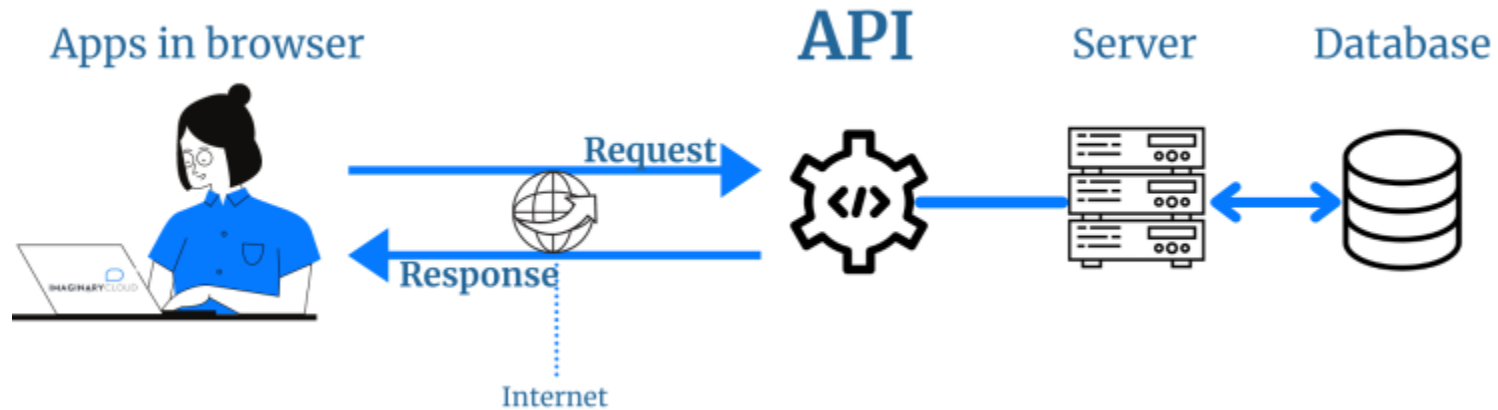
#2. API development

Python FastAPI, Streamlit

Topics

- 1) 웹 어플리케이션? 배포?
- 2) Scikit-learn 머신러닝 추론 모델 배포
- 3) Pytorch, Transformer, LLM 모델 배포

1) 웹 어플리케이션? 배포?



내가 만든 Python 함수를 인터넷 브라우저(HTTP, HTTPS 프로토콜)을 이용해 실행하려면

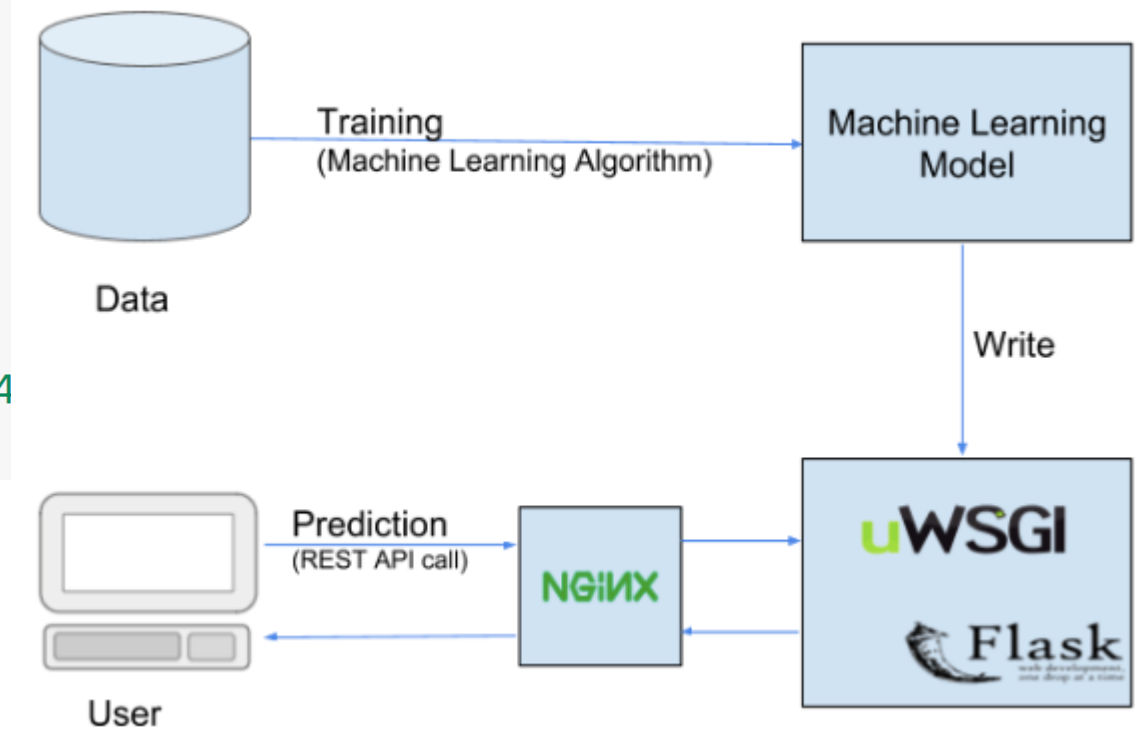
HTTP Request를 수신하고 HTTP Response를 응답할 수 있는 프레임워크가 필요

=> FastAPI, Flask, Node 등..

2) 머신 러닝 모델의 Deploy – pickle, joblib

```
# Creating Random Forest Model
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=20)
classifier.fit(X_train, y_train)

# Creating a pickle file for the classifier
import joblib
joblib.dump(classifier, 'ml_model_diabetes')
model = joblib.load('ml_model_diabetes')
modeltest = model.predict([[4,103,60,33,192,24
modeltest
```



#3. Docker

Docker / Docker Image / Docker compose / Server

Topics

- 1) App to Docker image / compose
 - Docker
 - Deployment
 - Docker compose
- 2) Web server / ASGI / WSGI
 - Web Server & (Web) Application Server
 - ASGI / WSGI
 - Settings

Docker

can you help me to launch this project?

you will need an npm installed first

which version do I need?

not sure, maybe John will know

he helped me yesterday

but no luck

...

X without Docker

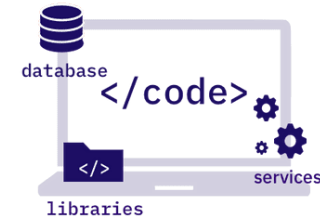
can you help me to launch this project?

sure, just run this command:

docker-compose up

thanks!

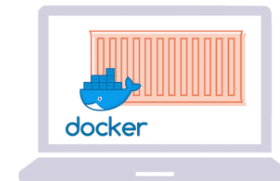
✓ with Docker



local status: **OK**



server status: **ERROR**
E: library XYZ missing



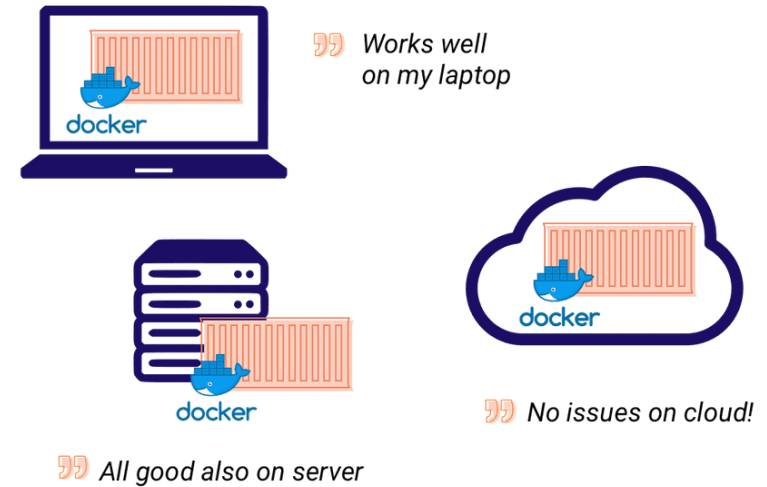
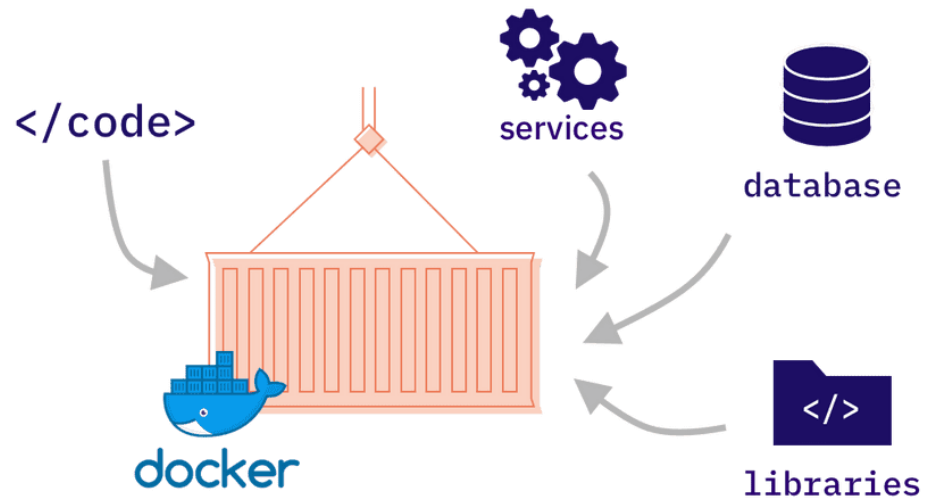
local status: **OK**



server status: **OK**

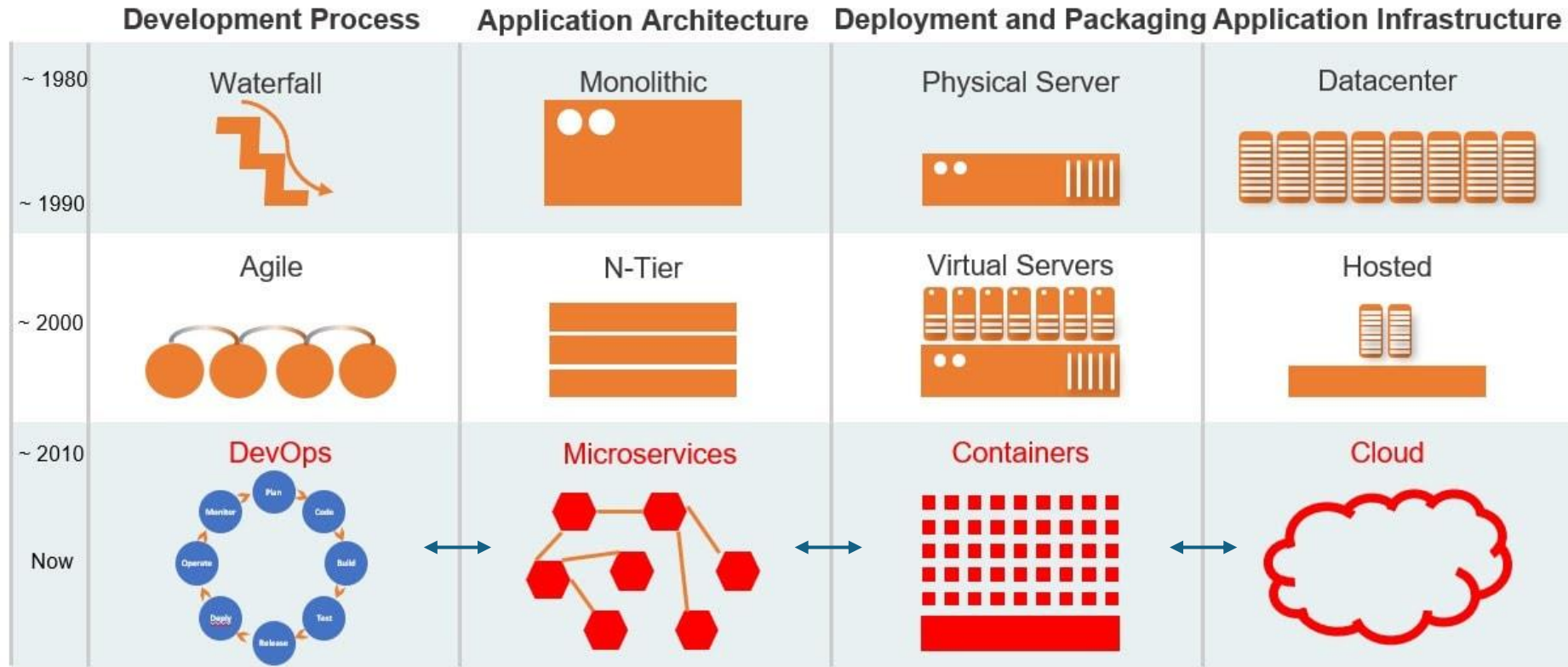
- 컨테이너화 하면 어떤 환경에서든 똑같이 동작한다 = 기존 서비스의 가상화에 유리하다 = **이식성**이 뛰어나다.

Docker



- 큰 서비스를 옮기는 것이 아니라, 마이크로서비스 단위로 컨테이너에 쌓고 빠르게 테스트할 수 있다. = 분산 어플리케이션화 = 확장성이 늘어난다.

Docker



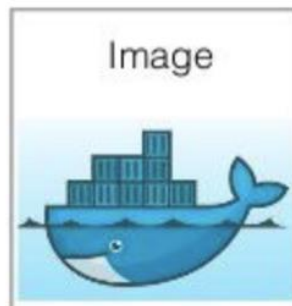
- 컨테이너 기술의 발전은 마이크로서비스 아키텍처를 더욱 효율적으로 구현할 수 있게 해줌
- 다양한 독립적인 서비스들이 서로 간섭 없이 운영될 수 있도록 지원, 클라우드 환경에서 유연하게 확장할 수 있는 기반을 마련

File, image, container

```
FROM ubuntu:16.04
MAINTAINER John Doe <john.doe@example.com>
WORKDIR /app
COPY . /app
RUN apt-get update && apt-get install -y python3
RUN python3 --help
```

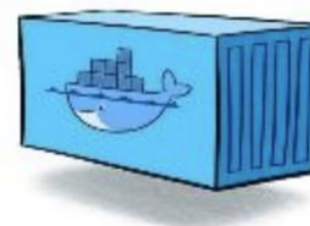
Dockerfile

build



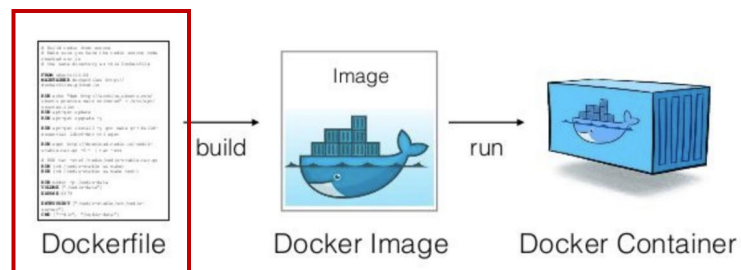
Docker Image

run



Docker Container

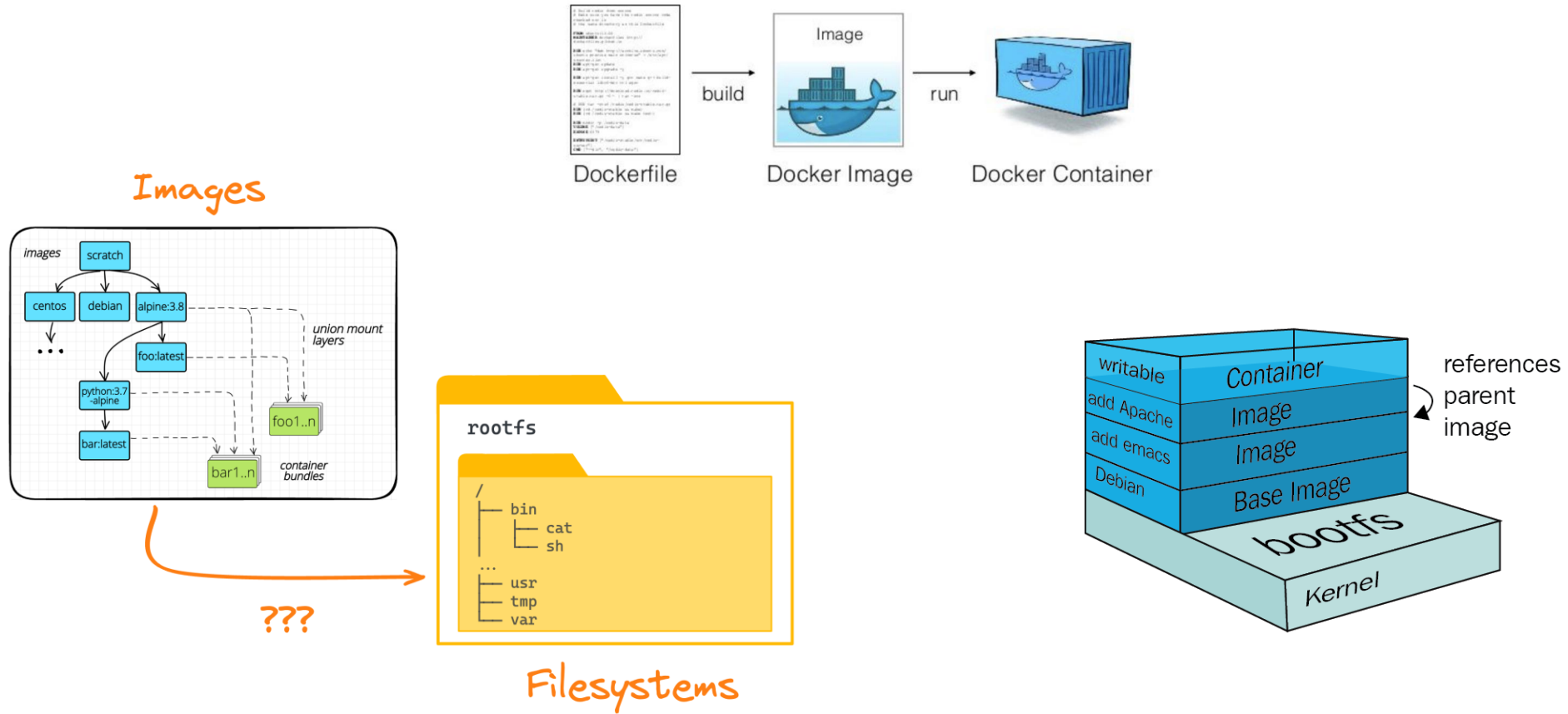
File, image, container



```
main.py • Dockerfile x requirements.txt index.html result.html
temp > Dockerfile
1 # 공식 Python 이미지를 기반으로 사용
2 FROM python:3.10-slim
3
4 # 작업 디렉터리 설정
5 WORKDIR /app
6
7 # 의존성 파일 복사
8 COPY app/requirements.txt .
9
10 # 의존성 설치
11 RUN pip install --no-cache-dir -r requirements.txt
12
13 # 애플리케이션 파일 복사
14 COPY app /app
15
16 # 애플리케이션 실행
17 CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

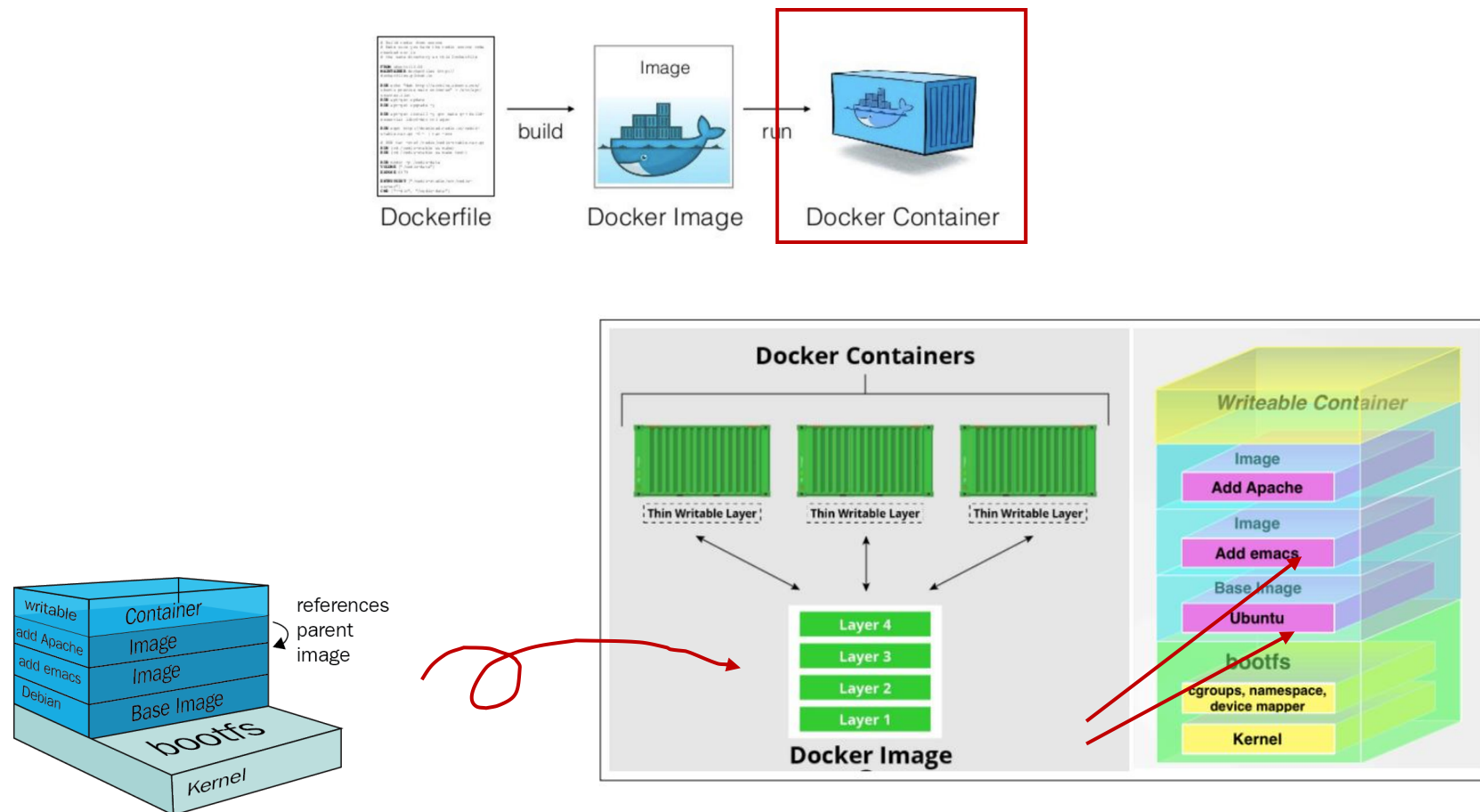
Dockerfile은
도커 이미지가 도커 컨테이너로 작동하기 위한 데
이터,
즉 실행되기 위한 조건들에 대한 metadata

File, image, container



Docker image ≠ image file이라는 것을 보여주기 위한 시각자료 (File system임)

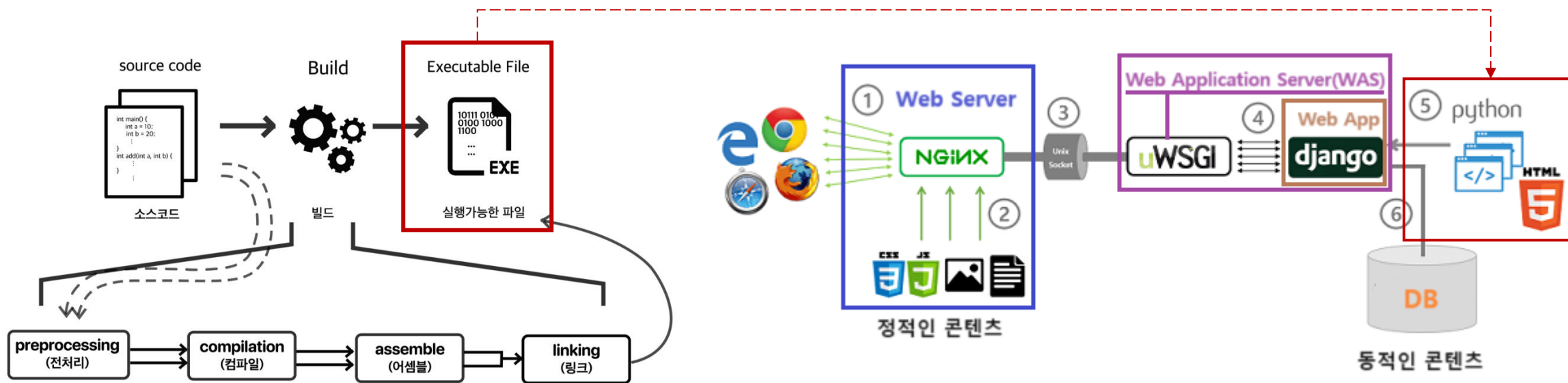
File, image, container



도커 이미지가 어떤 방식으로 컨테이너에 포함되는가?

Deployment?

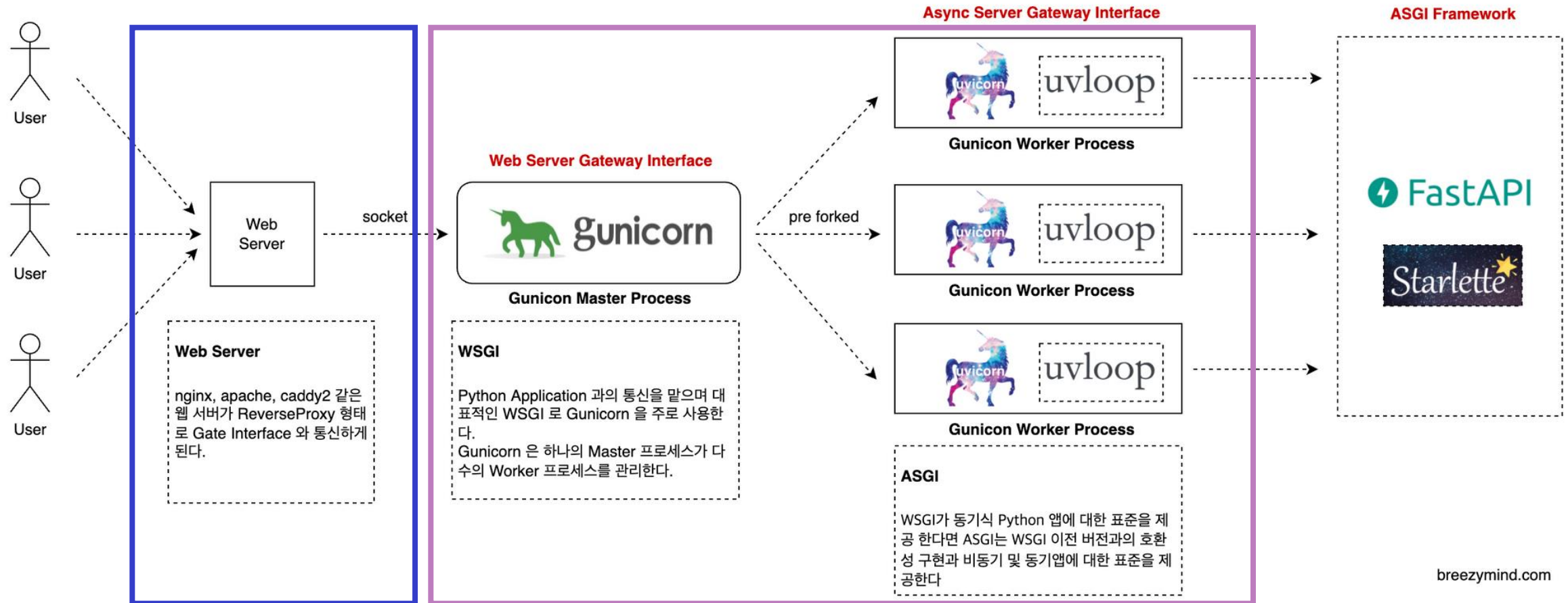
- 배포란 애플리케이션을 사용자가 사용할 수 있도록 하는 데 필요한 단계를 수행하는 것
- 웹 API의 경우, 일반적으로 사용자가 중단이나 오류 없이 애플리케이션에 효율적으로 접근할 수 있도록 좋은 성능, 안정성 등을 제공하는 서버 프로그램과 함께 원격 시스템에 이를 설치하는 작업을 의미한다.
- 이는 지속적으로 코드를 변경하고, 지우고, 수정하고, 개발 서버를 중지했다가 다시 시작하는 등의 개발 단계와 대조된다.



<개발 단계>

<배포 단계>

WSGI



- **WSGI, ASGI**는 웹 서버(static)와 웹 어플리케이션 서버(dynamic)가 통신하기 위한 인터페이스
- Gunicorn(WSGI)의 안정성과 Uvicorn(ASGI)의 비동기 처리 성능을 함께 활용하기 위한 구조

Guidelines in Deployment

FastAPI Document에서의 Deployment Guidelines

- When deploying a **FastAPI** application, or actually, any type of web API, there are several concepts that you probably care about, and using them you can find the **most appropriate** way to **deploy your application**.
- 배포 과정에서 중요하게 고려해야 할 요소
 - Security - HTTPS
 - Running on startup
 - Restarts
 - Replication (the number of processes running)
 - Memory
 - Previous steps before starting
- 궁극적인 목표는 안전한 방식으로 API 클라이언트에 서비스를 제공하고, 중단을 방지하고, 컴퓨팅 리소스(예: 원격 서버/가상 머신)를 가능한 한 효율적으로 사용할 수 있도록 하는 것
- 크게 3가지 요소(보안, 자원 관리, 가용성) 측면에서 가이드라인을 살펴보고, deploy 예제 수행

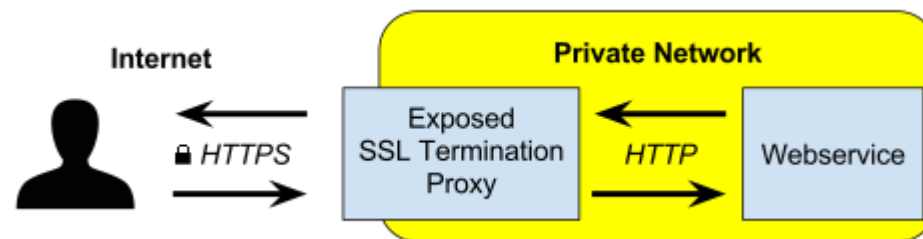
Guidelines in Deployment

보안 측면

https 인증서 얻기

TLS Termination Proxy

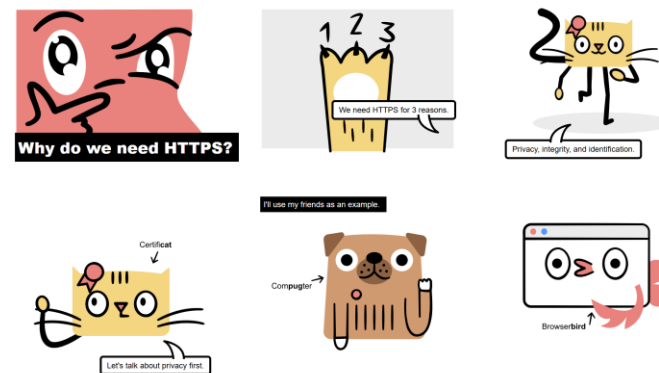
- ➔ 프록시(Load Balancer, LB)가 인터넷과 보안된 정보 교환
- ➔ 웹서비스(서비스/앱의 Back-end side)와 프록시(LB)는 HTTP 로 통신



TLS Termination 구조를 취함으로써 얻을 수 있는 장점

- ➔ 결과적으로 보안확인(Authenticate) 절차가 줄어들면서 CPU 사용량 감소
- ➔ 암호화(Encryption) 및 복호화(Decryption)에 많은 연산 자원이 소모되므로 Back-end side에서 연산 횟수를 줄이는 것이 관건임
- ➔ 뒤에 나오는 절차들(메모리 관리, 로드 밸런싱 등) 또한 리소스 관리 측면에서 고려되는 항목들

컷만화로 보는 HTTPS



Guidelines in Deployment

Resource managing 측면

- 다음은 1) 성능 개선 (Performance Improvement), 2) 확장성 (Scalability), 3) 가용성(High Availability) 확보를 위한 replicating \approx (scaling)전략에 대한 제안임
- **Uvicorn** with --workers
 - 하나의 Uvicorn 프로세스 관리자가 해당 IP와 포트에서 수신 대기하고 여러 개의 Uvicorn 워커 프로세스를 시작
- Kubernetes and other distributed container systems
 - Kubernetes 계층의 무언가가 IP와 포트에서 수신 대기, 복제는 각각 하나의 Uvicorn 프로세스가 실행되는 여러 개의 컨테이너를 갖는 방식으로 이루어짐
- Cloud services that handle this for you
 - 클라우드 서비스에서 복제를 처리할 수 있다. 실행할 프로세스나 사용할 컨테이너 이미지를 정의할 수 있지만, 어떤 경우든 단일 Uvicorn 프로세스가 될 가능성이 높으며, 클라우드 서비스가 이를 복제하는 역할을 담당한다.

Guidelines in Deployment

Continuous Operations / Load Balancing 측면

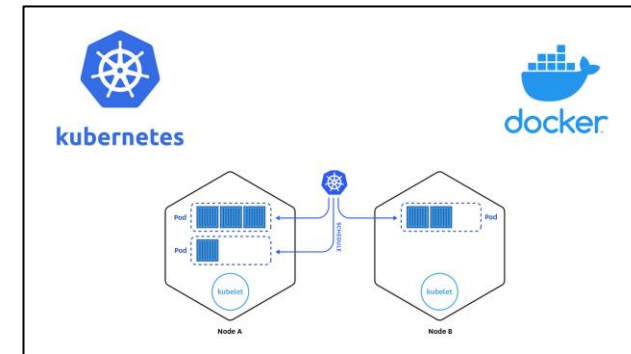
- 해당 측면에서 좀 더 가용성(High Availability) 확보의 목적성을 갖는다.
- Example Tools to Run at Startup
 - Some examples of the tools that can do this job are:
 - Docker
 - Kubernetes
 - Docker Compose
 - Docker in Swarm Mode
 - Systemd
 - Supervisor
 - Handled internally by a cloud provider as part of their services
 - Others...

1



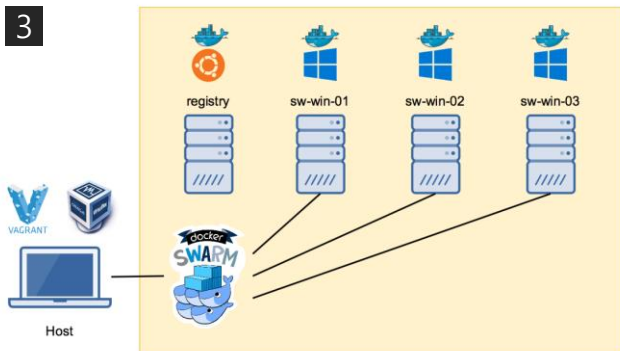
단일 서버에서 도커 컴포즈로 배포하기

2



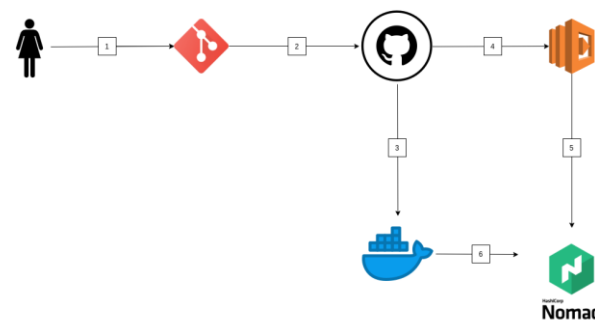
쿠버네티스 클러스터로 배포하기

3



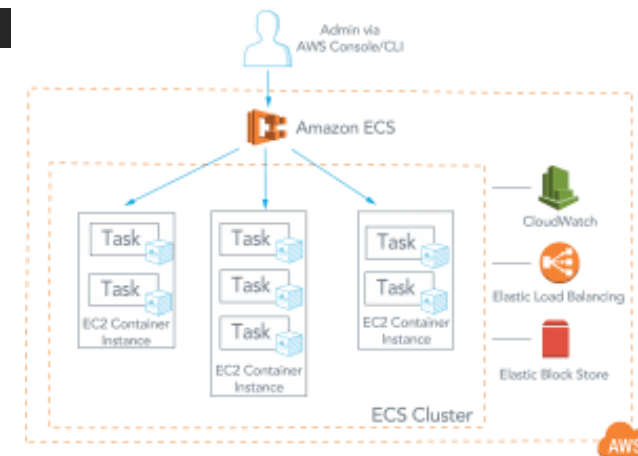
Docker swarm mode로 배포하기

4



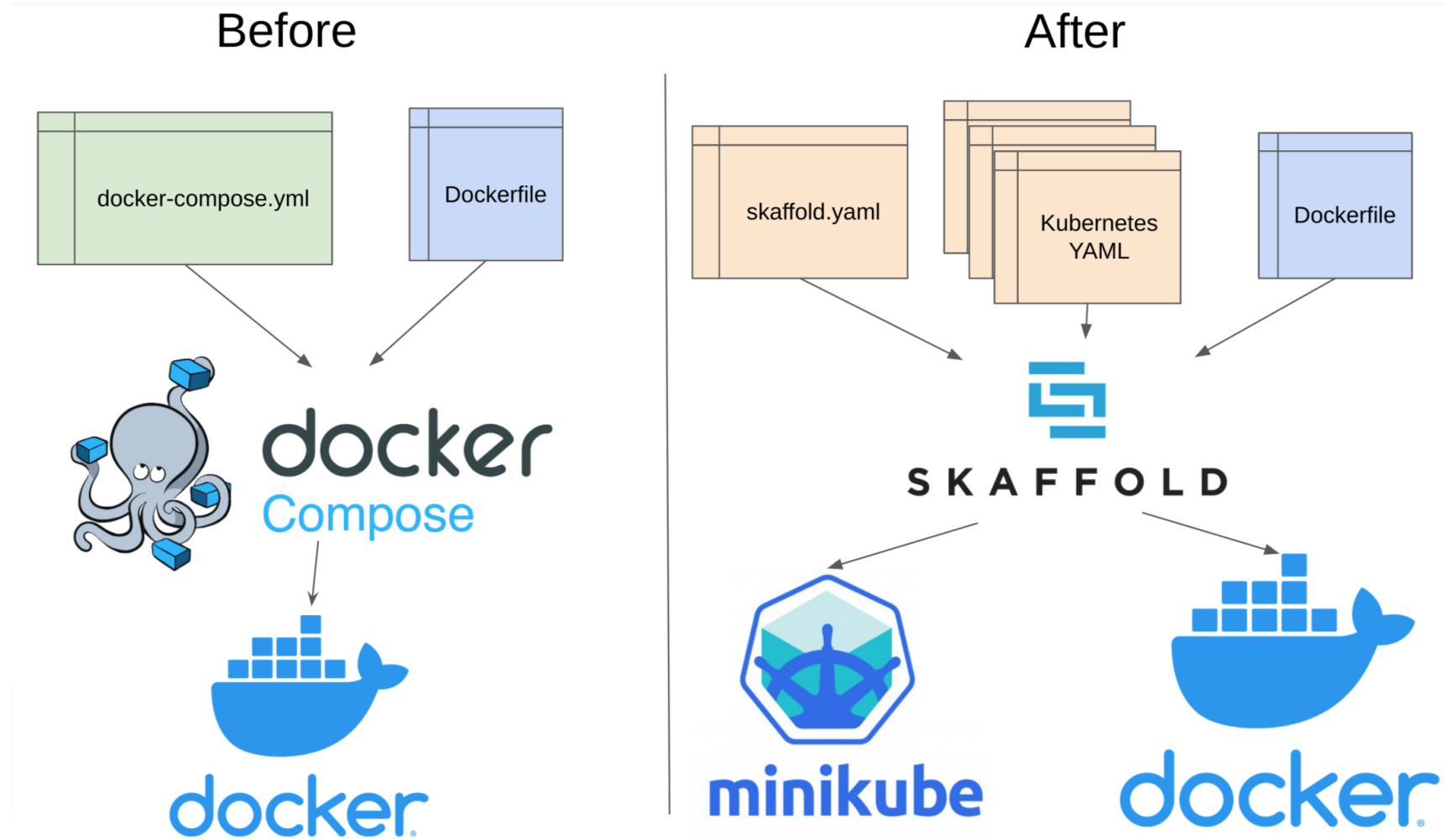
Nomad + github으로 배포 및 관리

5



내 컨테이너 이미지를 배포해줄 수 있는 서비스 찾기 (ex. AWS, Azure)

Docker compose



- 컨테이너는 개발자들이, 쉽게 소프트웨어가 어디에 배포되든, 잘 동작할 것이라는 것만 알면 되도록 만들어주었다. 컨테이너는 또, 종종 "microservices"라고 불리는 것을 가능하게 만들었다. 하나의 거대한 monolithic 애플리케이션을 가지는 대신에, microservice는 application을 여러 개의 작은 파로 쪼개고 각 파트들이 서로 대화가 가능하게 만들었다. 이 의미는 서로 다른 팀들이 그들이 그 애플리케이션들이 서로 상호작용하는 방법을 바꾸지 않는 한, 더 쉽게 각기 다른 파트의 애플리케이션에 대해서 일할 수 있도록 했다는 것이고, 그들이 서로에게서 독립적으로 일할 수 있게 했다는 의미이다. 이는 소프트웨어 개발을 더 빠르게 만들었고, 발생 가능한 에러를 테스트하는 것을 쉽게 만들었다. ("WTF is container?", Techcrunch)
- 도커 이미지는 그냥 compressed layered file system이고, 컨테이너에서 재생될 수 있게 메타 데이터를 얹은 구조다.
- 도커 컨테이너는 resource isolation process이다.
- Kubernetes pod은 컨테이너가 아니다. at least one, or multiple container(s)로 구성된 별개의 객체
- 도커 컨테이너의 근간을 이루는 기술은 리눅스처럼 네트워크, 디스크, CPU 리소스를 쉼어하는, 즉 namespaces and cgroups이라는 kernel feature로 구현될 수 있는 resource isolated process다. 차별점은 가상화랑 엮어서 어디서든 작업환경을 repeat할 수 있다는 점, 컨테이너가 OS 커널을 공유함에 따라 기존 VM환경에서 할 수 없었던 범위의 마이크로서비스를 테스트할 수 있음.
- 왜 이름이 컨테이너냐? 컨테이너는 규격화 되어있고, 따라서 (선박에)많이 실을 수 있음. 여기에 초점을 둔 작명으로 보

#4. Kubernetes

Orchestration / Services

Topics

- 용어 및 주요 기능 소개
- Orchestration
 - 언제?
 - 어떻게?
- 서비스 구조 분석

Glossary

•노드(node)

- 노드는 쿠버네티스의 작업 장비(worker machine)이다.
- 작업 노드는 클러스터에 따라 VM이거나 물리 머신일 것이다. [파드](#) 실행에 필요한 로컬 데몬과 서비스를 가지고 있으며, 콘트롤 플레인에 의해서 관리된다. 노드에 있는 데몬은 [kubelet](#), [kube-proxy](#)와 [도커\(Docker\)](#) 같이 컨테이너 런타임을 구현한 [CRI](#)를 포함한다.
- 여담으로 초기 쿠버네티스 버전에서는 노드를 "미니언(Minions)"으로 불렀었다.

•Kubectl

- 쿠버네티스 API를 사용하여 쿠버네티스 클러스터의 [컨트롤 플레인](#)과 통신하기 위한 커맨드라인 툴
- 사용자는 쿠버네티스 오브젝트를 생성, 점검, 업데이트, 삭제하기 위해 kubectl을 사용할 수 있다.

•Kubelet

- 클러스터의 각 [노드](#)에서 실행되는 에이전트. Kubelet은 [파드](#)에서 [컨테이너](#)가 확실하게 동작하도록 관리한다.

•Namespace

- 쿠버네티스에서 하나의 [클러스터](#) 내에서 리소스 그룹의 격리를 지원하기 위해 사용하는 추상적 개념으로, 클러스터의 오브젝트를 체계화하고 클러스터의 리소스를 분리하는 방법을 제공한다.

•도커(Docker)

- 도커(구체적으로, 도커 엔진)는 운영 시스템 수준의 가상화를 제공하는 소프트웨어 기술이며, [containers](#) 로도 알려져 있다.
- Docker는 리눅스 커널의 리소스 격리 기능을 사용하며, 그 격리 기능의 예는 cgroups, 커널 네임스페이스, OverlayFS와 같은 조합 가능한 파일 시스템, 컨테이너가 단일 리눅스 인스턴스에서 독립적으로 실행되게 하여 가상 머신(VM)을 시작하고 관리하는 오버헤드를 피할 수 있도록 하는 기타 기능 등이 있다.

•컨트롤 플레인(Control Plane)

- 컨테이너의 라이프사이클을 정의, 배포, 관리하기 위한 API와 인터페이스들을 노출하는 컨테이너 오케스트레이션 레이어.

•파드(Pod)

- 컨테이너의 라이프사이클을 정의, 배포, 관리하기 위한 API와 인터페이스들을 노출하는 컨테이너 오케스트레이션 레이어.
- 파드는 일반적으로 하나의 기본 컨테이너를 실행하기 위해서 구성된다. 또한 파드는 로깅과 같이 보완적인 기능을 추가하기 위한 사이드카 컨테이너를 선택적으로 실행할 수 있다. 파드는 보통 [디플로이먼트](#)에 의해서 관리된다.

개요

쿠버네티스는 다음과 같은 기능을 제공합니다:

- Service discovery and load balancing**: 쿠버네티스는 DNS 이름을 사용하거나 자체 IP 주소를 사용하여 컨테이너를 노출할 수 있습니다. 컨테이너에 대한 트래픽이 많은 경우, Kubernetes는 네트워크 트래픽을 로드 밸런싱하고 분산하여 배포가 안정적으로 이루어지도록 할 수 있습니다.
- Storage orchestration**: Kubernetes를 사용하면 로컬 스토리지, 퍼블릭 클라우드 제공자 등 원하는 스토리지 시스템을 자동으로 마운트할 수 있습니다.
- Automated rollouts and rollbacks**: Kubernetes를 사용하여 배포된 컨테이너에 대해 원하는 상태를 설명할 수 있으며, 제어된 속도로 실제 상태를 원하는 상태로 변경할 수 있습니다. 예를 들어, 배포를 위한 새 컨테이너를 생성하고, 기존 컨테이너를 제거하고, 모든 리소스를 새 컨테이너에 적용하도록 Kubernetes를 자동화할 수 있습니다.
- Automatic bin packing**: 컨테이너화된 작업을 실행하는 데 사용할 수 있는 노드 클러스터를 Kubernetes에 제공합니다. 각 컨테이너에 필요한 CPU와 메모리(RAM)의 양을 Kubernetes에 알려줍니다.
- Self-healing**: Kubernetes는 장애가 발생한 컨테이너를 다시 시작하고, 컨테이너를 교체하고, 사용자 정의 상태 확인에 응답하지 않는 컨테이너를 죽이고, 서비스할 준비가 될 때까지 클라이언트에게 알리지 않습니다.
- Secret and configuration management**: Kubernetes를 사용하면 비밀번호, OAuth 토큰, SSH 키와 같은 민감한 정보를 저장하고 관리할 수 있습니다. 컨테이너 이미지를 다시 빌드하거나 스택 구성에 시크릿을 노출하지 않고도 시크릿과 애플리케이션 구성을 배포하고 업데이트할 수 있습니다.
- Batch execution**: 서비스 외에도, Kubernetes는 배치 및 CI 워크로드를 관리하여 원하는 경우 실패한 컨테이너를 대체할 수 있습니다.
- Horizontal scaling(수평적 크기 확산)**: 간단한 명령으로, UI를 통해, 또는 CPU 사용량에 따라 자동으로 애플리케이션을 확장 및 축소할 수 있습니다.
- IPv4/IPv6 이중 스택**: 파드 및 서비스에 IPv4 및 IPv6 주소 할당
- 확장성을 위한 설계**: 업스트림 소스 코드를 변경하지 않고도 Kubernetes 클러스터에 기능을 추가할 수 있습니다.

Orchestration

오케스트레이션이 나오게 된 원인 분석

컨테이너들이 점점 많아지면서... (수십, 수백개) -> 도커가 좋긴 한데, (관리 측면에서) 여전히 손이 많이 간다.
개발 -> 빌드 -> 복사 -> 실행 -> (?)에 대한 Needs 발생

구체적으로는, 도커-컨테이너 기반 마이크로서비스 생태계 구축 및 운영 과정에서 다음과 같은 요구사항들이 생겨나게 됨.

- 1) 배포 과정에서의 Needs
- 2) 서버 모니터링 측면에서의 Needs (배포 관리)
- 3) 컨테이너 버전 관리 측면 (버전 관리)
- 4) 서비스 조회 -> (마이크로서비스 <-> 로드밸런서 작업 세팅) 과정에서 은근 work 부하가 많음
- 5) 상태 관리 (컨테이너에 장애가 생길 때 관리의 자동화)

∴ Container Orchestration -> 복잡한 컨테이너 환경을 효과적으로 관리하기 위한 도구 (= 서버 관리자가 해야 할 일을 자동화 한 결과물)

Orchestration

클러스터 (Cluster)

중앙 제어 (master-node)

- 클러스터 관리를 위한 마스터 노드를 두고 마스터 노드가 클러스터에 명령을 내리는 방식
- 관리자는 마스터 노드를 관리하는 것만으로 클러스터를 관리할 수 있어야 함

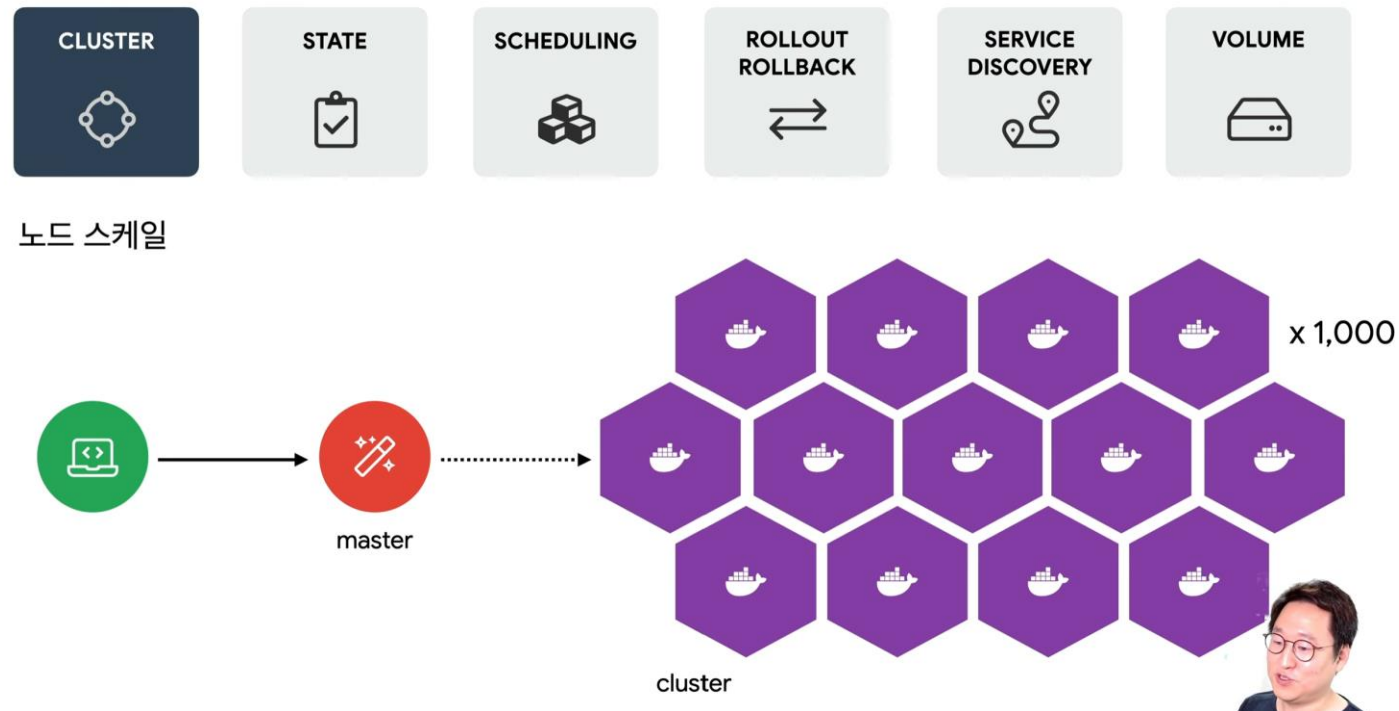
네트워킹

- 클러스터 내에서는 가상 네트워킹 등을 통해 노드 간에 네트워킹이 원활하게 이뤄져야 함

노드 스케일

- 클러스터 내에 컨테이너가 무한히 증가 하더라도 관리가 가능해야 함

컨테이너 오케스트레이션



자료 출처: 44BITS / [초보를 위한 쿠버네티스 안내서] 컨테이너 오케스트레이션이란?

Orchestration

컨테이너 오케스트레이션



상태 관리



상태 (State) 관리

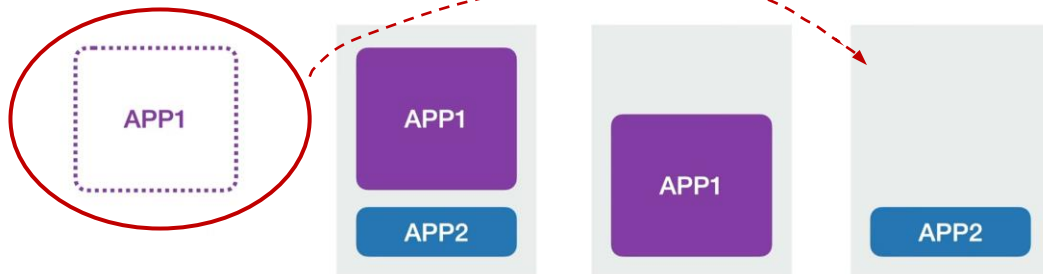
- 원하는 상태(예: 컨테이너 수량)를 설정하는 것 만으로 별도의 명령 없이 해당 상태로 변경될 수 있어야 함
- 장애로 인해 상태가 변경 된 경우 설정 된 상태를 유지하기 위한 명령이 자동 실행되어야 함

Orchestration

컨테이너 오케스트레이션



배포 관리



배포 관리 (Scheduling)

- 유휴 리소스를 관리하여 새로운 컨테이너 생성 시 알맞은 서버에 배치 시킬 수 있어야 함
- 추가 리소스가 필요 한 경우 리소스를 할당하여 컨테이너를 배치 시킬 수 있어야 함

Orchestration

컨테이너 오케스트레이션



배포 버전관리



배포 버전관리 (Rollout / Rollback)

- 배포 시 개별 배포가 아닌 중앙에서 자동 배포가 진행될 수 있어야 함
- 롤백의 경우에도 중앙에서 관리하며 일괄적으로 롤백이 진행될 수 있어야 함

Orchestration

컨테이너 오케스트레이션



서비스 등록 및 조회



서비스 등록 및 조회 (Service Discovery)

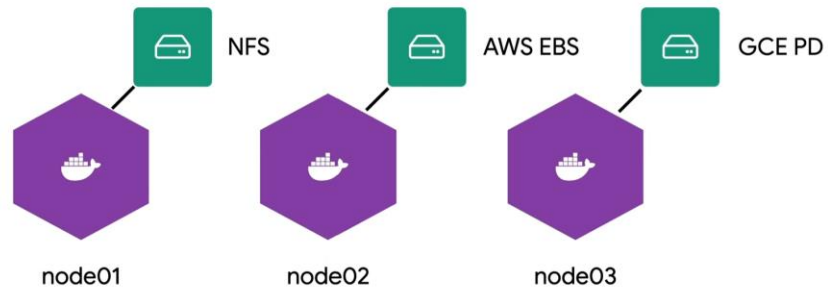
- 새로운 서비스가 추가 된 경우 자동으로 해당 서비스에 대한 설정이 추가 되어야 함
- 프록시 서버는 설정 저장소를 바라보며 설정이 변경 된 경우 프로세스 재시작을 통해 새 설정을 반영해야 됨

Orchestration

컨테이너 오케스트레이션

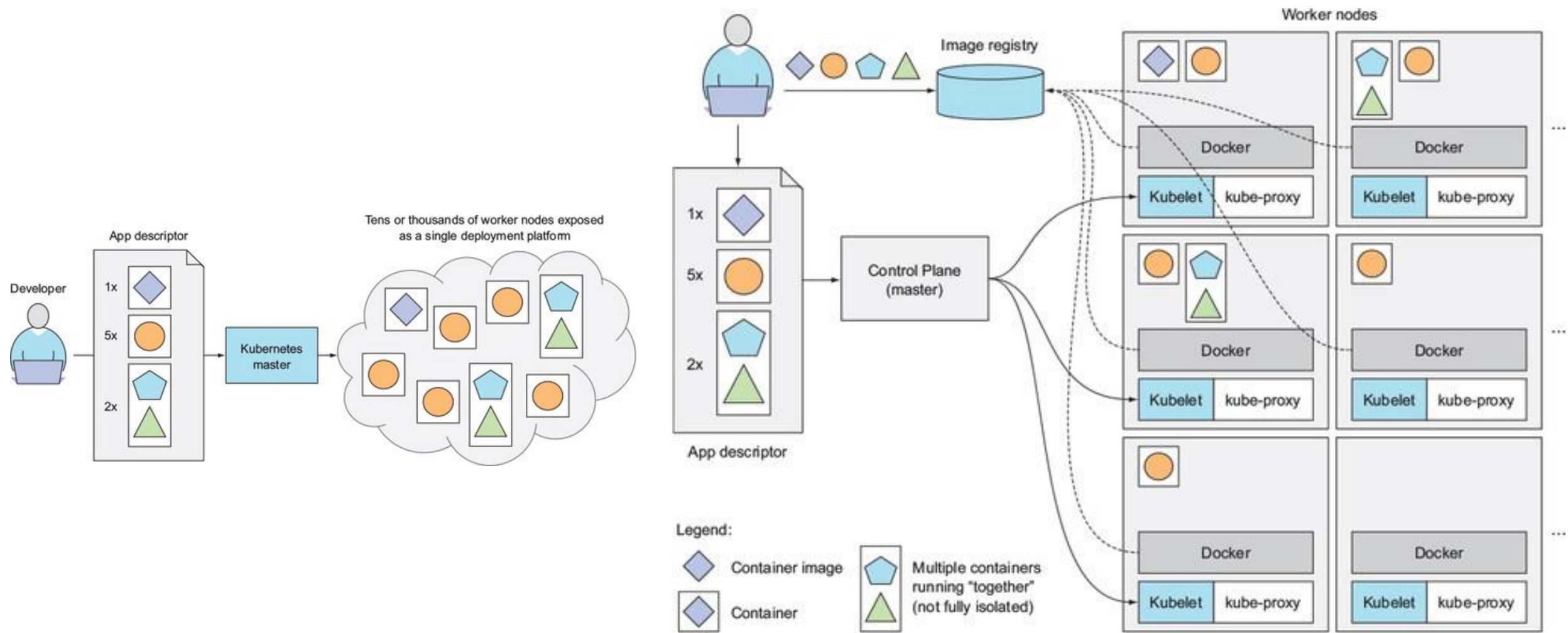


볼륨 스토리지



볼륨 스토리지 (Volume)

- 각 컨테이너 별 볼륨 관리를 추상적인 레벨에서 손쉽게 관리할 수 있어야 함



k8s 서비스 기획 고려사항

1. 컨설팅

- 요구 사항 수집 및 아키텍처 제안
- 목표 아키텍처 설계 / 리뷰
- 사용 및 오픈소스 솔루션 제안
(필요시 데모 진행)



2. 인프라 구축/설정

- 코드형 인프라 (Infrastructure as Code)
기반 인프라 구축
- 쿠버네티스 배포파일 작성
- Service Mesh 환경 구축



3. CI/CD 구축

- 빌드 및 테스트 파이프라인 구축
- GitOps기반 배포 파이프라인 구축
- 배포 전략 수립
(Canary 배포, Blue/Green 배포)

4. QA

- 성능 테스트
- 장애 테스트
- 요청/필요 시 Istio 기반 Circuit
Breaker 구성



5. 모니터링

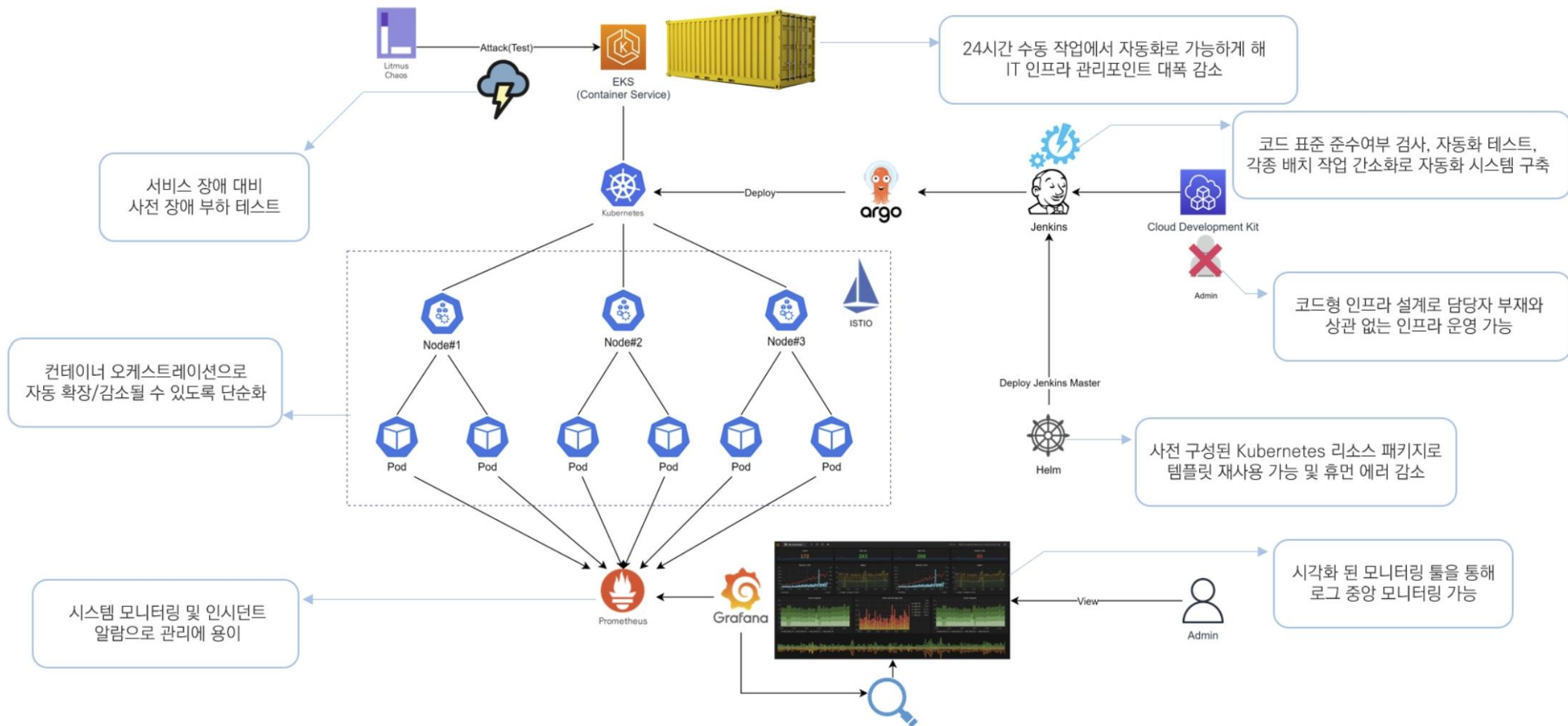
- 오픈소스 기반 구성 시 : Prometheus,
Grafana, Jaeger, Kiali
- AWS 기반 구성 시 : CloudWatch
Container Insight
- 기타 상용솔루션의 경우 배포 지원
(솔루션 비용은 별도)



6. 로깅

- AWS 기반 구축 시 : Fluentbit +
CloudWatch logs 연동
- 요청/필요 시 EFK(Elasticsearch
Fluentbit, Kibana) 구축

k8s 서비스 기획 고려사항



References

chap 1.

[참고자료: 리눅스에서 자주사용되거나 꼭 알아야 하는 명령어 모음 TOP 100 • 테크플](#)

[레이](#)
[참고자료: The Linux Commands Handbook](#)

[참고자료: \[리눅스/Linux\] 간단하게 리눅스 명령어 연습해 볼 수 있는 사이트 JS Linux — 🌟](#)

[41. Linux chroot 사용법](#)

[SSH란 무엇인가요?. SSH 란 무엇일까? SSH 를 왜 사용할까? SSH 의 장단점은? | by Seung Hyun \(James\) | Medium](#)

chap 2.

chap 3.

[Github Actions 및 AWS Lambda를 사용하여 Nomad에 배포 | 에 의해 iúnigoTechNews | 보통](#)

[배포 개념 - FastAPI](#)

[Docker를 이용한 Flask + Nginx + UWSGI 환경 구축하기.](#)

[ASGI 웹 프레임워크 FastAPI 를 시작하며](#)

[\[fastapi\] uvicorn, fastapi 비동기 메커니즘 이해 : 네이버 블로그](#)

[\[kubernetes\] TLS Termination](#)

[TLS termination proxy - Wikipedia](#)

[도커\(Docker\) / 깃랩\(GitLab\) / 허가받은 IP만 접속 가능\(IP whitelist\) 셋팅하기 / 공식문서 기준 / 보안 강화 / 아과노트](#)

[Docker Image VS Container: Learn How They Differ](#)

[What is Docker and why to use it? Explained for executives. | Accesto Blog](#)

chap 4.

[Glossary | Kubernetes](#)

[Overview | Kubernetes](#)

[\[kubernetes-in-action\] 1. 쿠버네티스 소개 | Sungsu's Tech Blog](#)

[쿠버네티스와 컨테이너 오케스트레이션, 그리고 핵심 설계 사상](#)

[\[K8s\] 쿠버네티스란 무엇인가?](#)

[\[초보를 위한 쿠버네티스 안내서\] 컨테이너 오케스트레이션이란? - YouTube](#)